End-User Authoring of Mid-Air Gestural Interactions

by

Mehmet Aydın Baytaş

A Thesis Submitted to the

Graduate School of Social Sciences and Humanities

in Partial Fulfillment of the Requirements for the Degree of

Master of Arts

in

Design, Technology & Society

KOÇ UNIVERSITY

September 2014

KOÇ UNIVERSITY

Graduate School of Social Sciences and Humanities

This is to certify that I have examined this copy of a master's thesis by

Mehmet Aydın Baytaş

and have found that it is complete and satisfactory in all respects, and that any and all revisions reqired by the final examining committee have been made.

Committee Members:

Oğuzhan Özcan
Professor of Design
(Thesis Advisor)

Yücel Yemez
Associate Professor of Computer Engineering
(Thesis Advisor)

Kerem Rızvanoğlu
Associate Professor of Informatics

Evren Yantaç
Associate Professor of Design

Tilbe Göksun
Assistant Professor of Psychology

Date: 11 September 2014

Abstract

Devices that sense the alignment and motion of human limbs via computer vision have recently become a commodity; enabling a variety of novel user interfaces that use human gesture as the main input modality. The design and development of these interfaces requires programming tools that support the representation, creation and manipulation of information on human body gestures. Following concerns such as usability and physical differences among individuals, these tools should ideally target end-users and designers as well as professional software developers.

This thesis documents the design, development, deployment and evaluation of a software application to support gesture authoring by end-users for skeletal tracking vision-based input devices. The software enables end-users without programming experience to introduce gesture control to computing applications that serve their own goals; and provides developers and designers of gestural interfaces with a rapid prototyping tool that can be used to experientially evaluate designs.

Keywords

Hotspotizer; gestural interaction; gesture authoring; visual programming; end-user development; interface prototyping; mid-air gestures; perceptual interaction; Kinect.

Dedication

This thesis is dedicated to my friend Hasan Sinan Bank, who has shown me how even seemingly impossible things can be done quite quickly and easily, once you actually put yourself to work.

Acknowledgements

The work presented in this paper has been conducted as part of research supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK), project number 112E056.

Much of the research described in this thesis has been conducted as part of efforts initiated and directed by my advisors, Oğuzhan Özcan and Yücel Yemez. For two years they have continuously dedicated their time and energy to nourishing my growth as a researcher. Being their student has been a pleasure for me.

Ayça Ünlüer and Tilbe Göksun have provided invaluable input in terms of creative vision and scholarly insights. I honestly don't know where my research would have ended up without their advice.

Evren Yantaç, Ahmet Börütecene, Oğuz Turan Buruk, Damla Çay, Özge Genç, Ahmet Güzererler, Barış Serim, İlker Temuzkuşu, and all of the others with whom I have had the pleasure of occupying the same workspace have been a continuous source of inspiration and motivation.

Çağatay Başdoğan, Hakan Ürey, Metin Sezgin, Kaan Akşit, Hasan Sinan Bank, and Selim Ölçer have personally shown me on numerous occasions that engineering is not merely a vocation, but a supremely empowering state of mind.

I thank the anonymous participants in my studies, who have given their time and feedback to support my research.

Above all, I thank my mother and my father, for making me into who I am.

Notice of Prior Publication

Parts of this thesis have been adapted from the following publications:

> Mehmet Aydın Baytaş, Yücel Yemez, and Oğuzhan Özcan. 2014 (forthcoming). **Hotspotizer: End-User Authoring of Mid-Air Gestural Interactions.** In *Proceedings of the 8th Nordic Conference on Human-Computer Interaction (NordiCHI '14).*

> Mehmet Aydın Baytaş, Yücel Yemez, and Oğuzhan Özcan. 2014. **User Interface Paradigms for Visually Authoring Mid-air Gestures: A Survey and a Provocation.** In *Proceedings of the Workshop on Engineering Gestures for Multimodal Interfaces (EGMI 2014).*

Additionally, during the course of the research described in this thesis, the author has contributed to the following publication:

> Oğuzhan Özcan, Ayça Ünlüer, Mehmet Aydın Baytaş, and Barış Serim. 2012. **Re-thinking Spherical Media Surfaces by Re-reading Ancient Greek Vases.** Paper presented at the workshop *"Beyond Flat Displays: Towards Shaped and Deformable Interactive Surfaces,"* co-located with the *ACM International Conference on Interactive Tabletops and Surfaces (ITS '12).*

See Appendices B, C, and D for reproductions of these publications.

Contents

Chapter 1

Introduction

1.1 Motivation

Historically, using the alignment and motion of human limbs as an input modality for human-computer interfaces has been accomplished through intrusive methods — by placing markers or sensors on the body. Up until recently, non-intrusive sensing of human limb positions has been limited to research efforts (see Moeslund, Hilton, and Krüger (2006); Porta (2002); Moeslund and Granum (2001); and Gavrila (1999) for surveys of these works). In recent years, vision-based skeletal tracking sensors have become commercially available from a variety of established vendors such as Microsoft[1] (Figure 1.1) and Asus[2]. Non-intrusive — or *perceptual* (Crowley, Coutaz, and Bérard, 2000; Turk and Robertson, 2000) — sensing of body movements has thus become widely accessible for both commercial and non-commercial applications (Francese, Passero, and Tortora, 2012).

Figure 1.1 – The Microsoft Kinect sensor is equipped with a depth camera that can "see" the positions and motion of human limbs.

There are a variety of computing applications where the non-intrusive detection of human limb positions can be desirable as an input modality. Gaming is an obvious one (see Figure 1.2), where using movements with "prior mappings" to real-world happenings increases immersion (Cairns et al., 2014). Another one is user interfaces on public interactive systems: An input modality that does not require physical contact is often cheaper to deploy and maintain, and more hygienic to use. Of course, there are numerous other contexts where hygiene considerations can make a touch-less interface desirable: Cooking, gardening, working on a dirty mechanism, and performing surgery (Wen et al., 2013) come to mind. Other applications for perceptual interfaces include convenient control of smart homes (Tang and Igarashi, 2013), interactive art and musical instruments[3], interfaces for manipulating 3D images (Gallo, 2013), and spatial medicine (Huang, 2011; Lozano-Quilis et al., 2013; Simmons et al., 2013).

The design and development of perceptual interfaces requires that *gestures* — limb positions and movements that constitute inputs to the interface — be programmed (Lü and Li, 2012) —

[1] microsoft.com/en-us/kinectforwindows
[2] www.asus.com/Multimedia/Motion_Sensor_Products
[3] vimeo.com/45417241

Figure 1.2 – Gaming with the Microsoft Kinect. The sensor detects the motion of large human limbs without requiring any markers or devices to be worn or wielded.

or *authored* (Hartmann, Abdulla, et al., 2007; Kim and Nam, 2013) — in a machine-readable manner and mapped to events within the interactive system. This can be done in a textual programming environment using tools supplied by vendors of gesture-sensing hardware[4][5] or third parties[6]. Using textual programming to author gestures, however, has drawbacks — both for adept software developers and for comparatively non-technical users such as designers, artists, hobbyists or researchers in fields other than computing. (Borrowing the definition from Ko, Abraham, et al. (2011); I will henceforth refer to these users who use or produce software not as an end, but as a means towards goals in their own domain, as *end-users*). These drawbacks can be expressed in terms of Norman's (1986, 2002) concepts of the *gulf of execution* and the *gulf of evaluation*. Specifically; for end-users, textual programming embodies a significant *gulf of execution* — a chasm between the user's goals and the actions taken within a system to achieve those goals – since it introduces additional tasks like setting up the programming environment and getting used to the development ecosystem. For both end-users and software developers, textual programming embodies a significant *gulf of evaluation* — a gap between a system's output and the users' expectations and intentions — since it does not allow for rapid testing of whether an authored gesture specification conforms to the design that the user has in mind. (See Figure 1.3 for a visualization of the *gulf of execution* and the *gulf of evaluation*.) From a software engineering perspective, Hoste and Signer (2014) suggest that imperative textual programming "cannot cope" with mid-air gestures "due to the inversion of control where the execution flow is defined by input events rather than by the program, the high programming effort for maintaining an event history and the difficulty of expressing complex patterns." Thus, textual programming does not fully support the embodied (Dourish, 2004), reflective (Schön, 1984), and experiential (Lindell, 2014) practices inherent in the design, construction and evaluation (Hartmann, Klemmer, et al., 2006) of these highly interactive artifacts (Lim, Stolterman, and Tenenberg, 2008; Myers, Hudson, and Pausch, 2000).

An appropriately designed user interface that matches the user's needs "in a form that can be readily interpreted and manipulated" (Norman, 1986) helps bridge the gulfs of evaluation and execution.

This thesis presents my attempt at producing an appropriate user interface to support the design and development of perceptual gesture-based interfaces by end-users.

[4] microsoft.com/en-us/kinectforwindowsdev
[5] softkinetic.com
[6] kinecttoolbox.codeplex.com

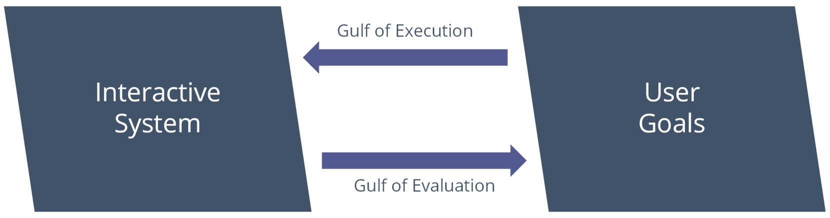


Figure 1.3 – The gulfs of execution and evaluation pertain to unidirectional aspects of interaction: The *gulf of execution* lies between the user's goals and the system; the *gulf of evaluation* divorces the system response from the users' expectations.

1.2 Aim

The aim of this thesis is to document the design, development, deployment and evaluation of a *software application to support end-users' authoring gross mid-air gestures* for *skeletal tracking perceptual input devices*.

The term *end-user* refers to those who utilize or produce software as a means towards goals in their own domain, rather than producing computing applications as an end (Ko, Abraham, et al., 2011). A mid-air gesture authoring tool may target diverse populations of end users including designers, artists, hobbyists, gamers and educators.

The methods employed in the design and evaluation of the application are selected to be appropriate for these purposes. The practices employed for the construction and deployment of the application also reflect its end-user focus: The application must perform well and reliably on users' computers, be easy to obtain and set up, and be maintainable to facilitate rapid adaptation to evolving technologies and user needs (Brooks, 1995; McConnell, 2009).

1.2.1 Research Questions

The main research question pursued in this thesis is as follows:

- *How can end-users' authoring of gross mid-air gestures for skeletal tracking interfaces be supported with a software tool?*

The main research question engenders the following secondary questions:

- What are the desiderata and design considerations that would pertain to mid-air gesture authoring software for end-users?
- What methods are appropriate to evaluate the application?

1.2.2 Hypothesis and Expected Contributions

I hypothesize that a suitably designed gesture authoring tool will accomplish the following:

- It will enable *end-users* with no experience in textual programming and/or gestural interfaces to introduce gesture control to computing applications that serve their own goals.
- It will provide *developers* and *designers* of gestural interfaces with a rapid prototyping tool that can be used to experientially evaluate designs.

I expect the following to be the *contributions* of this work:

1. *A software application* for authoring mid-air gestures that will accomplish the goals above and constitute an authentic contribution as an artifact of research through design.

2. Insights derived from the design, development, deployment and evaluation of the gesture authoring software that may inform future interaction design research and practice.

The software application constitutes an artifact of *research through design* (Frayling, 1993). Thus, it is expected to fulfill the following criteria proposed by Zimmerman, Forlizzi, and Evenson (2007) for the evaluation of such artifacts:

- *Process.* The methods employed must be selected rationally and documented rigorously.

- *Invention.* Various topics must be integrated in a novel fashion to create the artifact.

- *Relevance.* The artifact must be situated within a real, current context; while supporting a shift towards a justifiably preferable state.

- *Extensibility.* The work must enable the future exploitation of the knowledge derived from it.

1.3 Scope

This section defines the scope of the research presented in this thesis. Table 1.1 presents a summary of the scope, while the text serves to elaborate on details and clarify the terminology used.

In human-computer interaction (HCI) and interaction design (IxD) literature, the usage of the word *gesture* is ambiguous: Depending on the context, it may denote finger strokes on a touchscreen (Lü and Li, 2013), deformations inflicted on a tangible input device (Warren et al., 2013), full-body poses (Walter, Bailly, and Müller, 2013), even finger movements on a keyboard (Zhang and Li, 2014). For the purposes of this thesis; the following definition, adapted from Kurtenbach and Hulteen (1990), will be used:

A gesture is a movement or position of a human body part that conveys information.

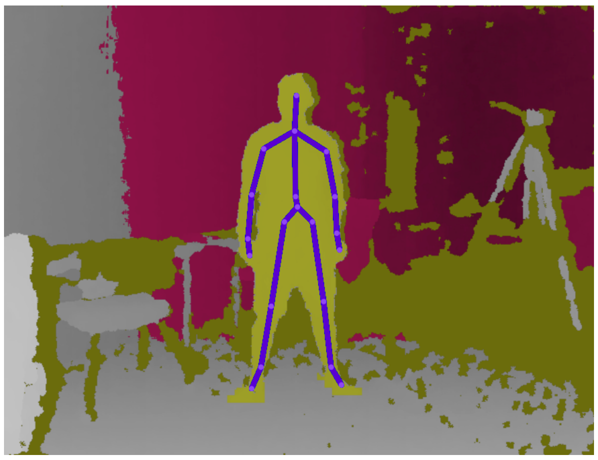


Figure 1.4 – The Microsoft Kinect sensor employs an infrared projector-camera pair to capture 3D depth images, and fits a skeletal model onto what resembles a human body in the image.

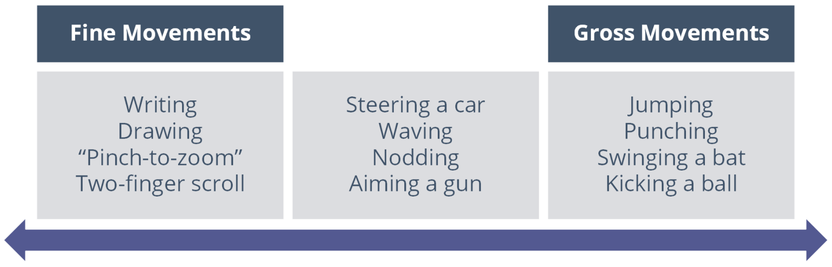


Figure 1.5 – The continuum of *fine* vs. *gross* movements.

By design, in order to accommodate existing works, this definition is broad. It allows for the use of any body part in gesturing as well as the use of sensing devices such as mouse, styli and gloves. It does not require an explicit intention to justify gesturing, thus accommodating non-command user interfaces (Nielsen, 1993a) such as those that respond to affective (Kapur et al., 2005) and habitual (Liu et al., 2009) gestures.

More specifically, I use the term *mid-air gestures* to denote gestures that are performed in a volume where limbs can move freely in 3 dimensions; e.g. free space. This excludes gestures that are constrained to affect a tangible surface or a controller device that mechanically changes form; e.g. a keyboard, a touch-sensitive surface, or a shape display (Follmer et al., 2013).

The gesture sensing input device used during the course of this work was a Microsoft *Kinect for Xbox 360*; chosen from among alternatives due to its availability. The device employs an infrared projector-camera pair to capture 3D *depth images*. If what resembles a typical human body is present in the depth image, the positions (relative to the sensor) of its large limbs are detected using machine learning (Girshick et al., 2011; Shotton, Fitzgibbon, et al., 2011; Shotton, 2012; Shotton, Girshick, et al., 2013). A *skeletal model* of the user is produced in this fashion (Figure 1.4). The alignment and motion of the skeletal model is then used to to control interactive applications. This type of gesture-sensing hardware is said to detect the movements and/or location of human body parts *perceptually* — without requiring physical contact (Crowley, Coutaz, and Bérard, 2000; Turk and Robertson, 2000). This excludes, from the scope of this thesis, systems that sense gestures using devices that must be worn, wielded, or touched — e.g. a mouse, a stylus, a ring[7][8], or an accelerometer (Ashbrook and Starner, 2010; Kela et al., 2006).

The Microsoft *Kinect for Windows Software Development Kit (SDK)* version 1.8 was used to implement gesture sensing. The capabilities of the Kinect sensor and the SDK are not limited to skeletal tracking; they also include the detection of hand gestures, speech recognition, background removal from videos, facilitating proxemic interaction (Ballendat, Marquardt, and Greenberg, 2010), fusing color and 3D images, and fusing data from multiple sensors. These topics, however, lie outside the scope of this work.

In kinesiology, human movements are classified according to movement precision (Haibach, Reid, and Collier, 2011): *Gross motor skills* denote large and comparatively imprecise movements produced by large muscles; e.g. jumping, or lifting weights. *Fine motor skills* involve smaller movements with higher accuracy and precision; e.g. typing, or writing. Gestures do not always belong strictly to one of two discrete classes. Rather, the distinction between fine and gross gestures forms a continuum characterized by the size of the engaged musculature and the trade-off between force and precision (Edwards, 2010) (Figure 1.5). One limitation of the skeletal tracking technology used for this work is that it can only detect gross gestures[9]. Thus, this work deals specifically with issues related to the use of *gross movements* of the human limbs as an interaction modality in computing.

[7]wearfin.com
[8]hellonod.com
[9]Currently, the Kinect SDK does have support for hand gestures. However, this feature was not available while the design work described in this thesis was done.

Aspect	Coverage	Excluded Topics
Spatial Qualities of Gestures	Mid-Air Gestures	Surface Gestures Tangibles Pointing Devices
Gesture Sensing Input Devices	Perceptual Input Devices (specifically, Microsoft Kinect)	Touch Sensors Inertial Sensors Wearables
Sensor Capabilities	Skeletal Tracking	Hand and Finger Gestures Speech Recognition Image Processing Proxemics Sensor Fusion
Gestural Bulk[10]	Gross Movements	Fine Movements

Table 1.1 – Summary of the topics covered within and excluded from the scope of the research presented in this thesis.

In sum, the scope of this work covers the design and development of a *software tool for authoring gross mid-air gestures* for interactive computing systems that employ *skeletal tracking perceptual input devices*.

1.4 Method

The method that I adopted to guide the design and development of the gesture authoring tool can be summarized as follows:

1. *Prior work* that may inform the design of a gesture authoring tool for skeletal tracking interfaces is surveyed to situate the work within the context of pertinent current research, reveal design guidelines, and determine appropriate strategies for design and evaluation. Chapter 2 documents this effort.
2. *Formative studies* are conducted with a focus group comprising 10 participants that form a representative sample from the target user populations, using prototypes with varying levels of fidelity. The process and the results of these studies, filtered through the lens of the aforementioned design guidelines, determine the nature and the core features of the gesture authoring tool — *what* it is going to be. Section 4.1 describes these studies in detail.
3. The resulting design for the gesture authoring tool is implemented as a working application. Various aspects of the implementation are described in Chapter 3.
4. *Summative studies* are conducted to assess if the implementation fulfills the previously stated aims for this work. A user study with 5 participants is conducted to assess the conformance of the artifact with its design rationale. A classroom workshop with 6 participants reveals the tool's potential in supporting rapid prototyping of user interface designs. Section 4.3 describes these studies and their results.

[10]See Section 2.1.

Chapter 2

Background and Related Work

This chapter presents a survey of previously published related works that inform the design of a gesture authoring tool for skeletal tracking interfaces; in order to situate the work within the context of pertinent current research, reveal insights that may inform the design of the tool, and determine appropriate strategies for design and evaluation.

2.1 Gestural Interaction

This section discusses dimensions that characterize gesture-based interactive systems and propose terms that clearly label pertinent concepts. Thus, the scope of and the design space for the research described in this thesis is partly situated in relation to previous works on gestural interaction.

Studies related to human gesture in HCI and IxD draw from a variety of domains, including but not limited to industrial design, psychology, anthropology, linguistics and computing. Here, also drawing from a multitude of disciplines, I argue that there are three important dimensions that characterize the design space for systems that utilize gesture as the main means of interaction (see Figure 2.1):

1. The *capture medium* is a high-level description of the hardware used to recognize gestures.
2. The *gestural bulk* is a description of the body parts involved in gesturing.
3. The gestural *engagement domain* is a description the kinds of gesture that the system utilizes.

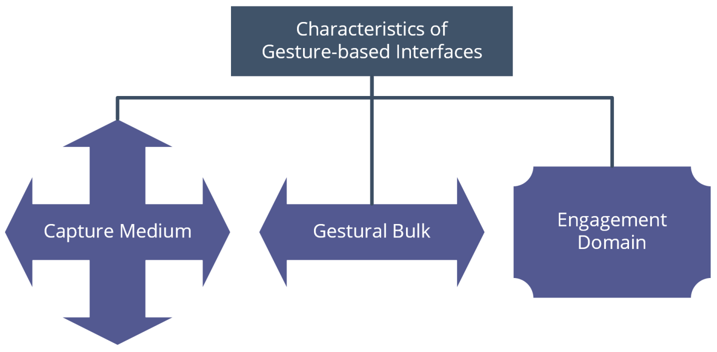


Figure 2.1 – Visualizing the three dimensions that characterize gesture-based interactive systems.

The first of these dimensions, the *capture medium*, describes the hardware technology used for sensing gestures. The hardware in this sense may comprise a 2-dimensional space that allows gesturing with a tangible pointing device such as a mouse or pen or a surface that detects touch events without utilizing a pointing device; tangible sensors that detect gestures in 3D space; perceptual input devices that "see" human movement from a distance; or myriad other current and emerging technologies (Figure 2.2).

Capture media differ in accordance to the degrees of freedom of the space in which the gestures are performed. It should be noted that while the degrees of freedom of the performance space is related to the degrees of freedom that the hardware can sense, the two are not the same: For instance, when gesture sensing is performed using cameras with algorithms based on edge detection instead of depth sensing (Moeslund and Granum, 2001; Moeslund, Hilton, and Krüger, 2006); computers are often only able to recognize activity in the horizontal and vertical, while changes in depth are ignored. However, from the user's perspective, the gestures are performed on a 3-dimensional medium, regardless of the level of detail that the computer can sense.

Adopting the user's perspective, I distinguish between *free-form* and *constrained* capture media. I propose term *constrained* to identify capture media where gestures are performed on a 2-dimensional surface, while the term *free-form* identifies capture media where gestures are performed in a 3-dimensional volume. Examples to constrained capture media are computer mice, trackpads and touchscreens; while camera- and accelerometer-based gesture sensing systems constitute examples to free-form capture media.

Capture media also differ according to whether or not they require special input devices to be worn or wielded by the user. An input device in this sense denotes any electronic device or object that is directly coupled to the movement or position of the body part(s) that make up the gesture. In most cases where such input devices are used, the system senses only the movements or the position of the input device. Quek (1996) has previously coined the term *unencumbered* to describe a class of capture media that do not employ such devices. The term has been used previously to refer only to *free-form* interactions. I wish to extend its definition to also accommodate *constrained* capture media that do not require pointing devices be used on the gesture-sensing surface, e.g. touchscreens that can be manipulated by human fingers alone. Conversely, when the capture medium — whether *constrained* or *free-form* —- relies on input devices such as mice, styli, gloves or accelerometers; I propose the term *equipped* to describe it.

A device that allows gestural interaction does not need to afford only one capture medium. Indeed, modern smartphones offer touchscreens that allow for both equipped and unencumbered input, while they also function as equipped free-form capture media since they harbor accelerometers and gyroscopes.

The second dimension, the *gestural bulk* describes body parts involved in interacting with the system. Following conventional terminology in kinesiology; I classify gestures for HCI as those relying on *fine motor movements* and those that rely on *gross motor movements*. *Fine movements* are precise and involve small musculature; e.g. typing on a keyboard or writing with a pen. *Gross movements* require the use of larger muscles and emphasize muscular force over precision; e.g. jumping or weight lifting. Section 1.3 describes this dimension in greater detail, in order to clarify the scope of my research.

The last of the dimensions I propose to classify gesture-based interactive computing systems is the gestural *engagement domain*, which describes what kinds of gestures a user interface utilizes. Psychology (McNeill, 1992, 2008) often provides the basis for classifications and analyses of human gesture in HCI and IxD literature[1] (Eisenstein and Davis, 2006; Kettebekov, 2004; Wexelblat, 1998). The perspective on the classification of human gestures proposed by Quek et al. (2002)

[1]In computing, the term *classification* also refers to the outcomes from a machine learning algorithm. Here, I use the term in pertinence to the characterization of gestures.

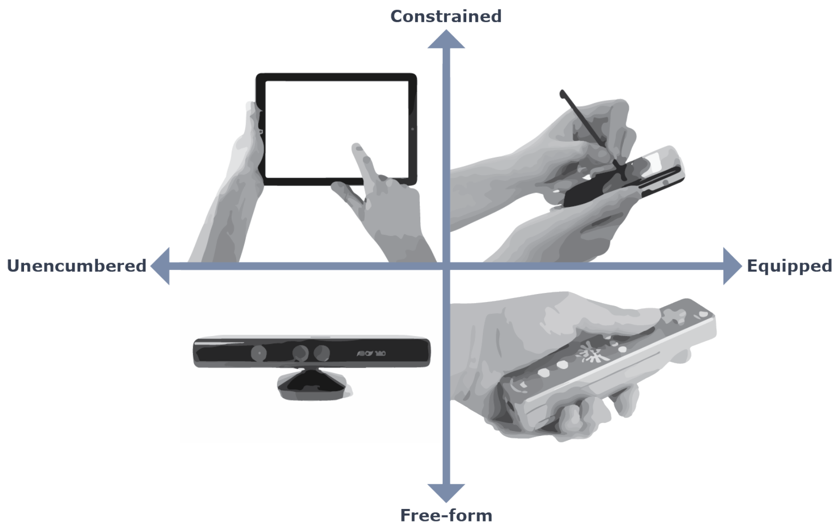


Figure 2.2 – Classifying capture media. A capture medium can be *constrained* or *free-form* depending on the space where gestures are performed, and *unencumbered* or *equipped* depending on the use or non-use of an input device that proxies human movement.

and extended by Karam and schraefel (2005) draws from these studies and forms an appropriate basis for my classification, albeit with modifications. I distinguish human gestures as belonging to one of four classes:

1. *Deictic gestures* that involve pointing to convey the identity and/or position of an entity.
2. *Manipulative gestures* that involve a direct connection between movements and the properties of an entity in the system in use.
3. *Semaphoric gestures* that function as symbols attached to a clear meaning.
4. *Gesticulations* that accompany speech.

Deictic gestures involve pointing (often with hands and/or fingers) to communicate the identity and/or the position of an entity. The canonical example for the use of deictic gestures in computing applications was implemented in MIT's "Media Room," where free-form pointing at items on a computer screen while uttering verbs and pronouns was exploited as an interface modality (Bolt, 1980). Somewhat unconventionally, since I adopt a broad definition for what *gesture* denotes, what I label as deictic gestures also includes everyday interactions such as pointing and clicking with a mouse or pressing a button on a touchscreen.

Manipulative gestures correspond to situations where "a tight relationship between the actual movements of the gesturing hand/arm with the entity being manipulated" is established (Quek et al., 2002). Using such gestures, often on *constrained capture media*, for moving, resizing and otherwise transforming objects on a display are common in contemporary desktop and mobile computing scenarios. Examples for manipulative gestures on constrained capture media include actions such as "drag-and-drop," drawing a box with the mouse cursor to select multiple items on the screen, and touch-scrolling documents on a touchscreen tablet. On free-form capture media; the primary interactions in sports games such as bowling or tennis on the Nintendo Wii constitute examples for manipulative gestures: The movements of the handheld controller are tightly related to the movements of a ball, a racquet, a sword etc.

Semaphoric gestures — also referred to as "emblems" McNeill (2008) — are static poses or dynamic movements that function as symbols attached to a clear meaning. Sign language gestures — which some researchers such as McNeill (2008); and textciteKaram:2005 consider to be distinct from semaphores — may be considered semaphoric to the extent that they relate to my analysis. Examples for semaphoric gestures commonly used in computing include the mouse-controlled "navigation gestures" implemented in version 11 of the Opera web browser and the "wave to engage" gesture that proposed in Microsoft's 2013 *Kinect for Windows Human Interface Guidelines*. Although demonstrated in many works (Cao and Balakrishnan, 2003; Lenman, Bretzner, and Thuresson, 2002; Wilson and Shafer, 2003); Wexelblat (1995) disputes the usefulness of strictly semaphoric gestures in HCI, with the argument that "the one-to-one mapping of input to command reduces gesture to only the expressive power of a function-key pad." However, recent sensing technologies and implementation aides allow for very rapid development of interactions that employ semaphoric gestures. This leads to a proliferation of content that exploits the limited expressive power of semaphores for meaningful use. Today, in many cases, the use of semaphoric gestures over a key pad makes sense when aspects such as the system's cost and context, pedagogical considerations, and/or the overall user experience (Fogtmann, Fritsch, and Kortbek, 2008) are taken into account.

The final class of gestures that is relevant for HCI and IxD are *gesticulations*, which comprise what McNeill (2008) calls "motion that embodies a meaning relatable to the accompanying speech" — i.e. body movements produced along with speech to clarify or augment its content. The recognition of gesticulations is an important technical challenge in computing. The most common applications for gesticulations in HCI are affect recognition and multimodal interfaces where they accompany a speech recognition system to remove ambiguity and extend the interactive capacity (Kopp, Tepper, and Cassell, 2004; Krum et al., 2002; Silva and Arriaga, 2003).

There are, of course, gestures that do not fall strictly into one of the categories above. A swipe towards the left on a trackpad or tablet that is commonly utilized to invoke a "go back" command, for example, may be classified as a manipulative as well as a semaphoric gesture depending on the context and the system's feedback. However, I find this classification to be relevant and useful for examining gesture-based interfaces for what sort of gestural triggers they implement; hence specifying which domain of gestures that such interfaces can engage.

Some of the terms and concepts I propose are, to my knowledge, novel; and I believe they will foster future research and discussion: The distinction of *constrained* vs. *free-form* and *equipped* vs. *unencumbered capture media* will come in handy for classifying works appropriately. Yet, gesture-based HCI is a rapidly advancing field, with new technologies and concepts being introduced continuously. Thus, I am not putting forward an exhaustive and conclusive treatment of the topic. I covered what I believe are the most salient characteristics of gesture-based interactive systems and compiled a set of terms and concepts to support discussion and situation of my work.

In terms of the concepts introduced in this section, the scope of this thesis pertains to the design of a gesture authoring tool for use with *perceptual* (*unencumbered, free-form*) input devices that sense *gross* gestures. In line with the desiderata uncovered through formative studies (see Section 4.1), the gestural *engagement domain* that influences the expressive power of the resulting authoring tool is limited to *semaphoric gestures*, although support for *deictic* and *manipulative* gestures can be "hacked" together using various strategies (see Chapter 3).

2.2 End-User Programming

"Programming" can be defined as "the process of transforming a mental plan of desired actions for a computer into a representation that can be understood by the computer" (Myers, Ko, and Burnett, 2006). The study of various aspects of programming has been a long-established topic of human-computer interaction research. Traditionally, the focus of this field has been on the activities of professional programmers and novices who are aiming to become professionals. A relatively recent topic of interest is the study of end-user programming as a topic distinct from the activities of professional and novice software developers (Myers, Ko, and Burnett, 2006). As such, interaction with a diverse array of devices — e.g. TVs, telephones, alarm clocks... — and a

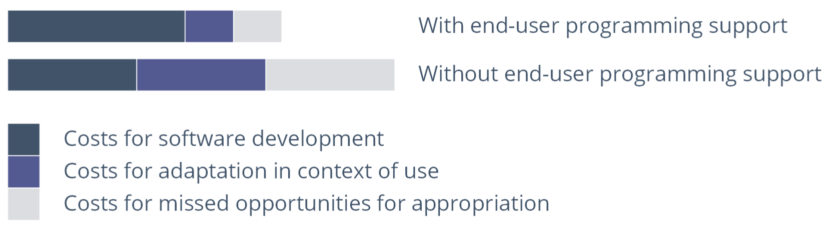


Figure 2.3 – Comparing the cost structure for software development, adaptation and appropriation with and without end-user programming support. Adapted from Wulf and Jarke (2004).

diverse assortment of tasks comprise topics of interest for programming research.

What differentiates an end-user from a professional programmer is their *goals*: Professionals create and maintain software as an end, while end-users produce and customize software artifacts to support their own goals in some other domain (Ko, Abraham, et al., 2011). In many domains, the case for end-user programming — rather than entrusting all development to professionals — is mainly economical: Wulf and Jarke (2004) argue that support for end-user programming makes software investments more efficient since empowering end-users to customize software mitigates the need for expensive development teams and processes (see Figure 2.3). The effect is significant, since end-users outnumber professional programmers by orders of magnitude (Scaffidi, Shaw, and Myers, 2005). From a user-focused perspective, research on end-user programming is motivated by usability and engagement concerns (Germonprez, Hovorka, and Collopy, 2007); as well as the simple fact that some systems such as smart homes (Blackwell, 2004) and health-related applications (Lange et al., 2011; Rizzo et al., 2011) must be customizable to suit individual needs (Holloway and Julien, 2010).

This thesis covers the design and development of a gesture authoring tool for designers' prototyping novel user interfaces and end-users' extending existing interfaces with the ability to recognize and respond to custom mid-air gestures. Various works on end-user programming that inform this effort are discussed below. A complete survey of this field is beyond the scope of this work; I recommend surveys by Paternò (2013); and Myers, Ko, and Burnett (2006) as a starting point for the interested reader.

One strand of research on end-user programming considers issues beyond the construction and customization of software; e.g. design, testing, debugging, integration, reuse, and security. This strand, called *end-user software engineering*, aims improve the *quality* of software artifacts produced by end-users by leveraging knowledge derived from professional software engineering. A comprehensive review of this research is beyond the scope of this thesis. While end-users' design, specification, testing, debugging, and reuse of software artifacts are indeed relevant for a gesture authoring tool; this thesis (as Chapter 4 describes) approaches such issues from a design — rather than software engineering — perspective. For the reader interested in end-user software engineering, I recommend surveys by Burnett, Cook, and Rothermel (2004), and Ko, Abraham, et al. (2011).

A number of end-user programming researchers focus on psychological and cognitive issues that relate to end-user programmers. One objective for such research, as Blackwell (2006) declares in his survey of the field, is "to increase our understanding of human cognition by studying a rather extreme domain of reasoning." Another objective is to tackle quality issues by informing the design of end-user programming tools and methods. Topics of interest include the difficulties of learning (Ko, Myers, and Aung, 2004; Pea and Kurland, 1987) and performing (Lewis and Olson, 1987) programming-related tasks, and how people envision programming concepts (Pane, Myers, and Ratanamahatana, 2001).

From among research on the psychology of programming, particularly relevant for a gesture authoring tool is work by Ko, Myers, and Aung (2004) where the authors identify six "learning barriers" that obstruct end-user programmers across a variety of contexts:

1. *Design barriers* are difficulties that are inherent in a problem, independent from how the solution is represented. They represent the inability of a learner to construct a solution to a given problem, which must be accomplished before the solution is implemented as software.
2. *Selection barriers* impede learners from discovering what components are afforded by the programming environment, and which of those can be used to implement their design for a software solution.
3. *Coordination barriers* hinder learners' understanding of how various components offered by the programming environment can be combined to achieve desired behaviors.
4. *Use barriers* obscure the intent, usage and effects of programming components. To illustrate with an example: a learner may have determined that they need to use a "list" structure to implement an algorithm, but they may not know how to declare and initialize one within the programming environment.
5. *Understanding barriers* arise when learners are not able to compare the external behavior, i.e. the results, of the software with their expectations. This is usually a result of an absence or inadequacy of feedback as to what the software does or does not do.
6. *Information barriers* disrupt learners' understanding of the internal workings of the software. They manifest as learners' inability to test hypotheses they might have about how the software does what it does.

The authors relate these learning barriers to Norman's (1986, 2002) concepts of the *gulf of execution* and the *gulf of evaluation* (which I described in section 1.1 to motivate the development of a gesture authoring tool). Specifically, they explicate that *design*, *coordination*, and *use* barriers spawn gulfs of *execution*; *understanding* barriers pose gulfs of *evaluation*; while *selection* and *information* barriers constitute gulfs of *execution* and *evaluation*. The authors recommend adapting Norman's (1986, 2002) recommendations for bridging the gulfs and overcoming the learning barriers.

2.3 Design and Evaluation of User Interface Authoring Tools

Olsen (2007) argues that user interface design tools, particularly those that deal with unconventional interaction techniques (e.g. mid-air gesture sensing), do not lend themselves to conventional software evaluation methods. One reason for this is that such tools require domain-specific expertise, which — by the nature of novel tools — no user population possesses. Another reason is that these tools support complex tasks with high inter-user variability in terms of the users' mental models of the tasks. "Meaningful comparisons between two tools for a realistically complex problem are confounded in so many ways as to make statistical comparisons more fantasy than fact." (Olsen, 2007) From the framework proposed by Olsen for the evaluation of user interface toolkits, I derived the following four guidelines to direct the design of my mid-air gesture authoring tool:

- *Reduce development time.* A good authoring tool should allow for the rapid implementation of design changes. This can be encouraged by reducing the number of choices that have to be made to express a design. (Granted, there may exist a tradeoff between this concern and the expressive power of the authoring tool.)
- *Encapsulate and simplify expertise.* Considerable technical know-how is required to design and develop applications for emerging technologies. A good design tool liberates the designer from the need for prior knowledge, yet communicates the capabilities and limitations of the technology to nudge the designer towards feasible designs.

- *Lower skill barriers.* Empowering new populations of users to envision and implement designs "expands the set of people who can effectively create new applications." (Olsen, 2007)
- *Make use of a common infrastructure.* It is difficult to get users to adopt a new standard. As much as possible, authoring tools should hook up to existing and widely adopted tools and practices, and complement existing workflows; upgrading rather than negating the common denominator.

Employing a user interface paradigm for expressing design choices that reflects the problem being solved and embodies the constraints of the design space (Norman, 1993) serves all four the guidelines above.

In addition, Shoemaker et al. (2010) propose design guidelines for body-centric interaction with large displays. From among the guidelines they propose, two generalize to influence the design of an authoring tool for mid-air gestures:

- Interaction using mid-air gestures at a distance should be *"mediated through a representation that binds personal and extrapersonal space."* A means for communicating the constraints and opportunities of the interaction space to the user is recommended for mid-air gestural interfaces. This holds for design tools that target these interactions.
- It is recommended that *users' sense of proprioception be leveraged* by allowing some operations to be performed in the user's personal space, without requiring visual feedback. In terms of authoring interactions, this guideline calls for encouraging gesture designs that capitalize on proprioception through the nature of the authoring paradigm.

In sum, six guidelines derived from previous work form the basis of my design rationale for the gesture authoring interface (Figure 2.4). The first four, derived from Olsen's (2007) work, identify and address concerns that pertain to user interface design tools. The last two, derived from the work of Shoemaker et al. (2010), attend to concerns related to perceptual interactions in general. Whether or not the final design for the authoring tool conforms to these guidelines is evaluated through user studies.

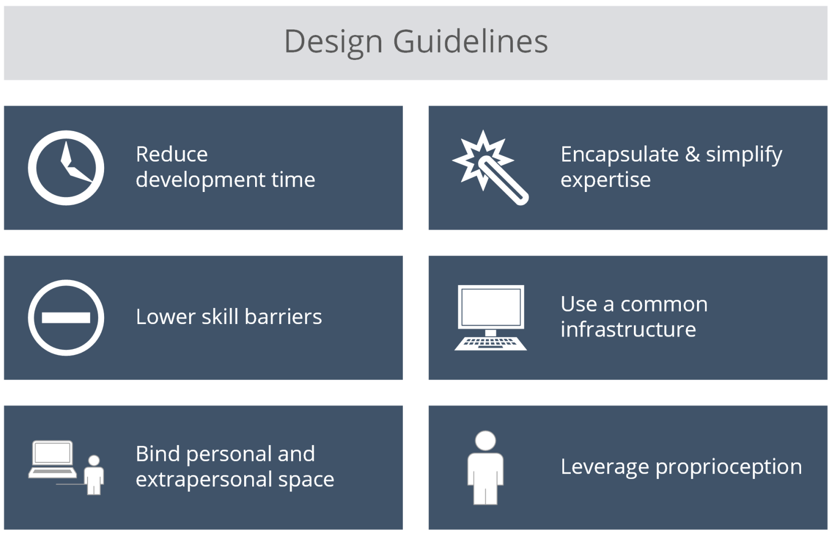

Design Guidelines
Reduce development time
Encapsulate & simplify expertise
Lower skill barriers
Use a common infrastructure
Bind personal and extrapersonal space
Leverage proprioception

Figure 2.4 – Guidelines derived from the literature formed the basis of the design rationale for a gesture authoring tool.

Additionally, from a programming perspective (see Section 2.2), Myers, Hudson, and Pausch (2000) identify five themes that influence the success of user interface tools:

- User interface tools should strive to achieve a low *threshold* — i.e. be easy to learn — and a high *ceiling* — i.e. significant expressive power.
- Successful tools lead users to making the right choices and avoiding wrong designs by offering a well-designed *path of least resistance*
- A tool should embody *predictability* and avoid unpredictable automatic operations.
- Developers of user interface tools should stay on top of developments, since user interface technologies are *moving targets* that can change significantly or become obsolete at a rapid pace.
- A tool should only *address the parts of the user interface that are needed*.

The first of these themes is specifically captured in part by Olsen's recommendations that a tool should lower skill barriers and empower new users. A high level of expressive power is desirable, but the weight of this concern will be governed by user needs (see Section 4.1). The encapsulation and simplification of expertise, along with the use of user-centered design tools and methods, integrate all of these themes.

2.4 Authoring Mid-Air Gestures

This section presents an overview of prior research on gesture authoring tools which has influenced my design. While development tools provided by vendors of gesture-sensing hardware focus on supporting textual programming, ongoing research suggests a set of diverse approaches to the problem of how to represent and manipulate three-dimensional gesture data. Existing works approach the issue in three ways that constitute distinct paradigms for visually authoring mid-air gestures. These are:

1. using 2-dimensional graphs of the data from the sensors that detect movement;
2. using a visual markup language; and,
3. representing movement information using a timeline of frames.

In addition, there are approaches to manipulating gesture information that rely predominantly on textual representations.

These paradigms for visualizing and manipulating gesture data often interact with two approaches to authoring gesture information:

- Authoring gestures by *declaration* involves the use of a high-level syntax to describe gesture information without specifying a computational control flow.
- Authoring gestures by *demonstration* is done by recording one or more examples and employing machine learning techniques to train a recognizer.

In addition, gestures can be defined through *imperative* programming, in terms of a sequence of states or actions. This is the standard approach for authoring gestures in general-purpose textual programming environments. From a design perspective, this approach embodies significant gulfs of execution and evaluation, while erecting learning barriers for end-users (see Sections 1.1 and 2.2).

Imperative authoring of gestures is also suboptimal from a software engineering perspective (see Section 1.1; as well as Hoste and Signer (2014)). Thus, imperative textual programming has not influenced my design significantly.

The three visual authoring paradigms enumerated above do not have to be used exclusively, and nor do demonstration and declarative programming. Aspects of different paradigms may find their place within the same user interface. A popular approach, for example, is to introduce gestures by demonstration, convert gesture data into a visual representation, and then declaratively modify it.

Below, I use examples from the literature to elaborate on the approaches enumerated above. I comment on their strengths and weaknesses based on previously published evaluations conducted with software that implement them.

Some of the work discussed below pertains to gesture-sensing systems which employ intrusive methods (e.g. markers or inertial sensors) rather than perceptual input devices. Even though the scope of this thesis does not fully encompass the intrusive sensing of mid-air gestures; the user interfaces of gesture authoring applications for intrusive sensors have aspects that inform the design of an authoring tool for perceptual interfaces. Thus, tools that target intrusive sensing applications and tools for perceptual interfaces are both considered.

2.4.1 Using Graphs of Movement Data

Visualizing and manipulating movement data using 2-dimensional graphs that represent low-level kinematic information is a popular approach for authoring mid-air gestures. This approach is often preferred when gesture detection is performed using inertial sensors such as accelerometers and gyroscopes. It also accommodates other sensors that read continuously variable data such as bending, light and pressure. Commonly the horizontal axis of the graph represents time while the vertical axis corresponds to the reading from the sensor. Often a "multi-waveform" occupies the graph, in order to represent data coming in from multiple axes of the sensor. Below, we study three software tools that implement graphs for representing gesture data: *Exemplar*, *MAGIC* and *GIDE*.

Exemplar

Exemplar (Hartmann, Abdulla, et al., 2007) relies on *demonstration* to acquire gesture data and from a variety of sensors - accelerometers, switches, light sensors, bend sensors, pressure sensors and joysticks. Once a signal is acquired via demonstration, on the resulting graph, the developer marks the area of interest that corresponds to the desired gesture. The developer may interactively apply filters on the signal for offset, scaling, smoothing and first-order differentiation. *Exemplar* offers two methods for recognition: One is pattern matching, where the developer introduces many examples of a gesture using the aforementioned method and new input is compared to the examples. The other is thresholding, where the developer manually introduces thresholds on the raw or filtered graph and gestures are recognized when motion data falls between the thresholds. This type of thresholding also supports hysteresis, where the developer introduces multiple thresholds that must be crossed for a gesture to be registered.

Exemplar's user studies suggest that this implementation of the paradigm is successful in increasing developer engagement with the workings and limitations of the sensors used. Possible areas of improvement include a technique to visualize multiple sensor visualizations and events and finer control over timing for pattern matching.

MAGIC

Ashbrook and Starner's (2010) *System for Multiple Action Gesture Interface Creation (MAGIC)* is another tool that implements the 2-dimensional graphing paradigm. The focus of *MAGIC* is programming by *demonstration*. It supports the creation of training sets with multiple examples of the same gesture. It allows the developer to that keep track of the internal consistency of the provided training set; and check against conflicts with other gestures in the vocabulary and an "Everyday Gesture Library" of unintentional, automatic gestures that users perform during daily activities. *MAGIC* uses the graph paradigm only to visualize gesture data and does not support manipulation on the graph.

One important feature in *MAGIC* is that the motion data graph may be augmented by a video of the gesture example being performed. Results from user studies indicate that this feature has been highly favored by users, during both gesture recording and retrospection. Interestingly, it is reported that the "least-used visualization [in *MAGIC*] was the recorded accelerometer graph;" with most users being "unable to connect the shape of the three lines [that correspond to the 3 axes of the accelerometer reading] to the arm and wrist movements that produced them." Features preferred by developers turned out to be the videos, "goodness" scores assigned to each gesture according to how they match gestures in and not in their own class, and a sorted list depicting the "distance" of a selected example to every other example.

GIDE

Gesture Interaction Designer (GIDE) by Zamborlin et al. (2014) features an implementation of the graph paradigm for authoring accelerometer-based mid-air gestures. *GIDE* leverages a "modified" hidden Markov model approach to learn from a single example for each gesture in the vocabulary. The user interface implements two distinct features: (1) Each gesture in the vocabulary is housed in a "gesture editor" component which contains the sensor waveform, a video of the gesture being performed, an audio waveform recorded during the performance, and other information related to the gesture. (2) A "follow" mode allows the developer to perform gestures and get real- time feedback on the system's estimate of which gesture is being performed (via transparency and color) and where they are within that gesture. This feedback on the temporal position within a gesture is multimodal: The sensor multi-waveform, the video and the audio waveform from the video are aligned and follow the gestural input. *GIDE* also supports "batch testing" by recording a continuous performance of multiple gestures and running it against the whole vocabulary to check if the correct gestures are recognized at the correct times.

User studies on *GIDE* reveal that the combination of multi- waveform, video and audio was useful in making sense of gesture data. Video was favored particularly since it allows developers to still remember the gestures they recorded after an extended period of not working on the gesture vocabulary. Another finding from the user studies was the suggestion that the "batch testing" feature where the developer records a continuous flow of many gestures to test against could be leveraged as a design strategy — gestures could be extracted from a recorded performance of continuous movement.

Discussion

Graphs that display acceleration data seem to be the standard paradigm for representing mid-air gestures tracked using acceleration sensors. This paradigm supports direct manipulation for segmenting and filtering gesture data, but manipulating acceleration data directly to modify gestures is unwieldy. User studies show that graphs depicting accelerometer (multi-)waveforms are not effective as the sole representation of gesture information, but work well as a component within a multimodal representation along with video.

2.4.2 Visual Markup Languages

Using a visual markup language for authoring gestures can allow for rich expression and may accommodate a wide variety of gesture-tracking devices, e.g. accelerometers and skeletal tracking, at the same time. The syntax of these visual markup languages can be of varying degrees of complexity, but depending on the sensor(s) used for gesture detection, making use of the capabilities of the hardware may not require a very detailed syntax. Below, I examine a software tool, *EventHurdle*, that implements a visual markup language for gesture authoring; and I discuss a gesture spotting approach based on control points which does not feature a concrete implementation, but provides valuable insight.

EventHurdle

Kim and Nam (2013) describe a declarative hurdle-driven visual gesture markup language implemented in the *EventHurdle* authoring tool. The *EventHurdle* syntax supports gesture input from single-camera-based, physical sensor-based and touch-based gesture input. In lieu of a timeline or graph, *EventHurdle* projects gesture trajectory onto a 2-dimensional workspace. The developer may perform the gestures, visualize the resulting trajectory on the workspace, and declaratively author gestures on the workspace by placing "hurdles" that intersect the gesture trajectory. Hurdles may be placed in ways that result in serial, parallel and/or recursive compositions. "False hurdles" are available for specifying unwanted trajectories. While an intuitive way to visualize movement data from pointing devices, touch gestures and blob detection; this approach does not support the full range of expression inherent in 3-dimensional mid-air gesturing.

Gestures defined in *EventHurdle* are configurable to be location-sensitive or location-invariant. By design, orientation- and scale-invariance are not implemented in order to avoid unnecessary technical options that may distract from "design thinking."

User studies on *EventHurdle* comment that the concept of hurdles and paths is "easily understood" and it "supports advanced programming of gesture recognition." Other than this, supporting features, rather than the strengths and weaknesses of the paradigm or comparison with other paradigms, have been the focus of user studies.

Worth noting is that *EventHurdle* is implemented as a plug-in for Adobe Flash[2], which may pose as a barrier for users who have not invested in the software.

Control Points

Hoste, Rooms, and Signer's (2013) versatile and promising approach uses spatiotemporal constraints around control points to describe gesture trajectories. While the focus of the approach is on gesture spotting (i.e. the segmentation of a continuous trajectory into discrete gestures) and not gesture authoring, they do propose a human-readable and manipulable external representation. This external representation has significant expressive power and support for programming constructs such as negation (for declaring unwanted trajectories) and user-defined temporal constraints. While the authors' approach is to infer control points for a desired gesture from an example, the representation they propose also enables the manual placement of control points.

The authors do not describe an authoring implementation that has been subjected to user studies. However, they discuss a number of concepts that add to the expressive power of using control points as a visual markup language to represent and manipulate gesture information. The first is that it is possible to add temporal constraints to the markup; i.e. a floor or ceiling value can be specified for the time taken by the tracked limb or device to travel between control points. This is demonstrated not on the graphical markup (which can be done easily), but on textual

[2]adobe.com/products/flash

code generated to describe a gesture – another valuable feature. The second such concept is that the control points are surrounded by boundaries whose size can be adjusted to introduce spatial flexibility and accommodate "noisy" gestures. Third, boundaries can be set for negation when the variation in the gesture trajectory is too much. The authors discuss linear or planar negation boundaries only, but introducing negative control points into the syntax could also be explored. Finally, a "coupled recognition process" is introduced, where a trained classifier can be called to distinguish between potentially conflicting gestures; e.g. a circle and a rectangle that share the same control points.

One limitation of this approach is the lack of support for scale invariance. One way of introducing scale invariance may be to automatically scale boundary sizes and temporal constraints with the distance between control points. However, it is likely that the relationship between optimal values for these variables is nonlinear, which could make automatic scaling infeasible.

Discussion

The expressive power and usability of a visual markup language may vary drastically depending on the specifics of the language and the implementation. The general advantage of this paradigm is that it is suitable for describing and manipulating location-based gesture information (rather than acceleration-based information commonly depicted using graphs). This makes using a visual markup language suitable for mid-air gestures detected by depth-sensing cameras, where the interaction space is anchored to the sensor and the users' body parts move in relation to each other and the sensor. Either the motion sensing device or part of the skeletal model could be used to define a reference frame and gesture trajectories could be authored in a location-based manner using a visual markup language.

2.4.3 Timelines

Timelines of keyframes are commonly used in video editing applications. They often consist of a series of ordered thumbnails and/or markers that represent the content of the moving picture and any editing done on it, such as adding transitions. A collection of commercial[3,4] and research (Tang and Igarashi, 2013) efforts implement timelines along with demonstration for authoring skeletal tracking gestures. Introducing gestures via demonstration requires the temporal segmentation of intended gestures from intermediate movements to be done manually - this is accomplished through manual editing on a timeline of keyframes.

Gesture Studio

One application that implements a timeline to visualize gesture information is the commercial *Gesture Studio*[5]. The application works only with sensors that detect gestures through skeletal tracking using an infrared depth camera. Users introduce gestures in *Gesture Studio* by demonstration, through performing and recording examples. The timeline is used to display thumbnails for each frame of the skeleton information coming from the depth sensor. The timeline is updated after the user finishes recording a gesture; while during recording, a rendering of the skeletal model tracked by the depth sensor provides feedback. After recording, the user may remove unwanted frames from the timeline to trim gesture data for segmentation. Reordering frames is not supported since gestures are captured at a high frame rate (depending on the sensor, usually around 30 frames per second), which would make manual frame-by-frame editing inconvenient. The process through

[3]gesturepak.com
[4]gesturestudio.ca
[5]gesturestudio.ca

which these features have been selected is opaque, since there are no published studies that present the design process or evaluate *Gesture Studio* in use.

Discussion

In gesture authoring interfaces, timelines make sense when gesture tracking encompasses many limbs and dynamic movements that span more than a few seconds. Spatial and temporal concerns for gestures in two dimensions, such as those performed on surfaces, can be represented on the same workspace. The representation of mid-air gestures requires an additional component such as a timeline to show the change over time.

Timelines of keyframes are used often in conjunction with programming by *demonstration*, for the manual segmentation of gesture data from intermediate bodily movements. For end-users without familiarity with machine learning concepts, the task of composing good training samples is not trivial. Moreover, this method cannot be used if the depth sensing device is not available during development (e.g. due to malfunction or devices being shared between users).

2.4.4 Textual Approaches

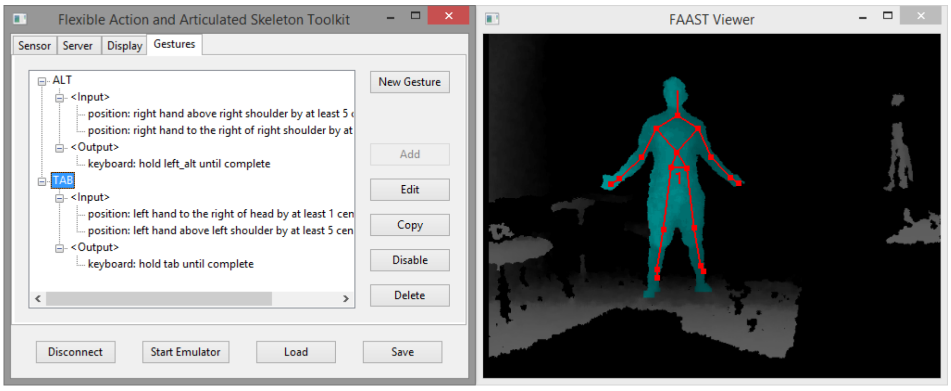
Domain-specific declarative gesture specification languages that augment general purpose programming tools are a very common approach to gesture authoring for both constrained and free-form capture media (see Section2.1). Scholliers et al.'s (2011) *Midas*; *GeforMT* by Kammer et al. (2010); Echtler and Butz' (2012) *GISpL*; *GestureAgents* by Julià, Earnshaw, and Jordà (2013); Spano et al.'s (2013) *GestIT* and work by Khandkar and Maurer (2010) are examples of such efforts. Most of the works within this body of research concentrate on the formal specification (Lamsweerde, 2000; Sommerville, 2010) of gestures — i.e. providing a rigorous, complete description of gesture information. As such, these tools are not designed to be utilized by end-users. The exception is the *Flexible Action and Articulated Skeleton Toolkit (FAAST)* by Suma et al. (2013), which provides a graphical user interface, along with an understandable grammar and vocabulary for the authoring of mid-air gestures.

FAAST

For declaratively authoring mid-air gestures for skeletal tracking, the *Flexible Action and Articulated Skeleton Toolkit (FAAST)* (Suma et al., 2013) provides atomic *action primitives* that can be used to compose rules in plain English such as "right hand above right shoulder by at least 20 cm." (Figure 2.5) These constraints specify the position of, the speed of, or the angle between limbs, as well as general body orientation. *FAAST* controls other applications on the computer via mapping gestures to keyboard and mouse events. While describing gestures using atomic rules affords significant expressive power, this representation does not embody a visualization of the constraints embedded in the design space and thus may not serve to bridge the gulf of execution that obstructs end-users.

FAAST has been adopted widely among hobbyists and numerous applications that utilize the toolkit have been exhibited online[6]. The authors draw attention to the permissive license that accompanies the software as they explain its popularity. Also notable in this regard is that *FAAST* — uniquely among the gesture authoring tools considered in this section — embodies the design guideline of leveraging a common infrastructure and interfaces with arbitrary third-party applications.

[6] projects.ict.usc.edu/mxr/faast/faast-video-gallery/



```
ALT
  <Input>
    position: right hand above right shoulder by at least 5 c
    position: right hand to the right of right shoulder by at
  <Output>
    keyboard: hold left_alt until complete
TAB
  <Input>
    position: left hand to the right of head by at least 1 cen
    position: left hand above left shoulder by at least 5 cen
  <Output>
    keyboard: hold tab until complete
```

Figure 2.5 – FAAST (Suma et al., 2013) provides atomic primitives that can be used to compose rules in plain English that map to mid-air gestures.

2.4.5 Discussion

Above, tools that exemplify user interface paradigms for visually and textually authoring mid-air gestures have been presented (Table 2.1). For sensor-based gesturing, the standard paradigm used to represent gesture information appears to be projecting the sensor waveforms onto a graph. Graphs appear to work well as components that represent sensor- based gestures, allow experimentation with filters and gesture recognition methods, and support direct manipulation to some extent. User studies show that while the graphs alone may not allow developers to fully grasp the connection between movements and the waveform (Ashbrook and Starner, 2010), they have been deemed useful as part of a multimodal gesture representation (Zamborlin et al., 2014). Using hurdles as a visual markup language offers an intuitive and expressive medium for gesture authoring, but it is not able to depict fully 3-dimensional gestures. Using spherical control points may be more conducive to direct manipulation while still affording an expressive syntax, but no implementation of this paradigm exists for authoring mid-air gestures. Finally, timelines of frames may come in handy for visualizing dynamic gestures with many moving elements, such as in skeletal tracking; but, used in this fashion, they allow only visualization and not manipulation.

System	UI Paradigm	Programming Approach	Insights from Evaluation
Exemplar (Hartmann, Abdulla, et al., 2007)	Graphs	Demonstration	Increases engagement with sensor workings and limitations.
MAGIC (Ashbrook and Starner, 2010)	Graphs (multi-waveform)	Demonstration	Users unable to connect waveform to physical movements. Optional video is favored over graphs.
GIDE (Zamborlin et al., 2014)	Graphs (multi-waveform, with video)	Demonstration	Multimodal representation helps make sense of gesture data.
EventHurdle (Kim and Nam, 2013)	Visual markup language	Declaration	Easily understood. Supports "advanced" programming.
Control Points (Hoste, Rooms, and Signer, 2013)	Visual markup language	Declaration and Demonstration	Not implemented.
Gesture Studio[7]	Timeline	Demonstration	Not published.
FAAST (Suma et al., 2013)	Textual	Declaration	Widely adopted due to standalone, end-to-end implementation.

Table 2.1 – Summary of studies on systems that exemplify user interface paradigms for authoring mid-air gestures.

Chapter 3

Hotspotizer: Description

This chapter describes a novel user interface paradigm for authoring mid-air gestures based on *space discretization*, and its implementation as part of an end-to-end software tool designed to support end-users: *Hotspotizer*. The design of the space discretization paradigm and Hotspotizer's user interface has been informed by insights from previously published research and the analysis of related artifacts (described in Chapter 2), along with the use of methods from user-centered design (described in Chapter 4).

3.1 Space Discretization

This section introduces a user interface paradigm based on space discretization for visualizing and manipulating gesture information. The essence of the paradigm is the partitioning of the interaction space into discrete cells. These discrete cells within the workspace may be marked to become hotspots – or *hotspotized* – that register when a limb or input device passes through them. Multiple cells can be hotspotized to form large regions with relaxed spatial constraints; and temporal ordering between different regions can be specified to express movement. In essence, hotspots behave like virtual, invisible buttons positioned in the interaction space (Figure 3.1).

Figure 3.2 shows how this paradigm can be applied to describe a gesture trajectory that follows the form of the letter *Z* on a 2-dimensional surface. Four regions consisting of multiple hotspots have been defined and they must be traversed in a certain order — indicated by the numbers on the figure — for the gesture to register.

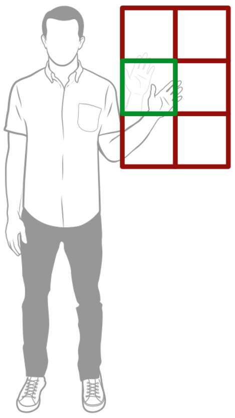


Figure 3.1 – Hotspots behave like virtual buttons set in the interaction space. A wide variety of gestures can be described in as sequences of hotspot configurations.

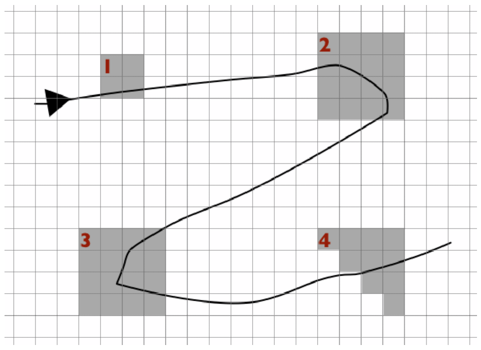


Figure 3.2 – A 2-dimensional surface "Z" gesture defined using ordered hotspots in discretized space.

The paradigm is versatile in that it can be used to author gestures for a variety of capture media (see Section 2.1) ranging from touch-sensitive surfaces to perceptual interaction spaces.

The partitioned workspace must be placed in relation to a certain frame of reference. The origin of this frame of reference may be, depending on the capture medium, a certain point on a touch-sensitive surface; a camera that consists a perceptual input device; a certain limb of the user; the initial point where a stylus makes contact with a touchscreen; etc. Barring some rare cases, hotspot configurations that make up a certain gesture will differ if the origin of the workspace changes.

In line with Shoemaker et al.'s (2010) recommendation that users' sense of proprioception be leveraged within body-centric perceptual interactions (see Section 2.3), the hotspot array's frame of reference can originate from the user's center of gravity for applications that use coarse movements. This way, the user's position in relation to the perceptual sensor does not affect gesture recognition, as long as the sensor can build the correct skeletal model.

Authoring dynamic movements relies on temporal constraints between hotspots. This can take the form of a simple inter-keyframe timeout; where the time that elapses between the traversal of subsequent hotspots must not exceed a given value. A variety of visualization styles can be explored for the authoring of the temporal constraints. Using an integrated visualization that shows spatial and temporal constraints together (as in Figure 3.2) is one option; as is splitting motion into discrete keyframes. With a focus on the latter option, design considerations for interacting with temporal constrains are discussed in Sections 4.2 and 3.2.

Gestures designed using space discretization are dependent on location, scale and orientation with respect to the workspace. Affixing the workspace to a physical part of the interactive system — including the user's limbs — can be exploited to introduce location and orientation invariance. For supporting scale invariance, the paradigm affords a degree of spatial flexibility; hotspotizing a larger volume of cells allows for relaxed gesture boundaries. (User studies described in Section 4.3 have shown that using large hotspots in lieu of spatially overconstrained gesture designs is indeed a desirable — although not inherently discoverable — strategy.)

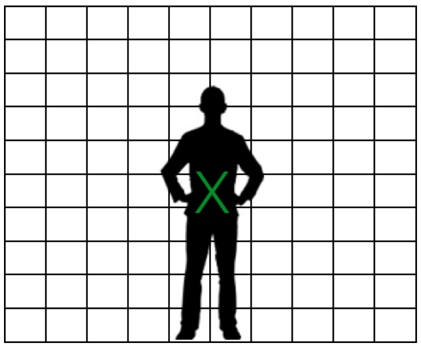


Figure 3.3 – For full-body perceptual interactions, the origin for the hotspot array's frame of reference can be affixed to the centroid of the user. This design leverages proprioception (see Section 2.3).

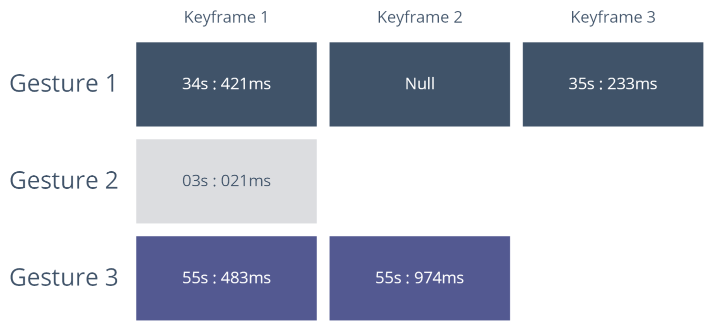


Figure 3.4 – A jagged array used to keep a memory of how tracked limbs traverse hotspots. The elements of the outer array are arrays that correspond to the gestures that the system can recognize. The elements of the inner arrays keep track of when the tracked limb last occupied the hotspots that belong to each keyframe of gesture.

In essence, aspects of this technique are based on Hoste, Rooms, and Signer's (2013) control points paradigm, modified to confine the locations of the control points to discrete pre-defined locations and standardize the shapes of control point boundaries. Multiple areas or volumes within the workspace can be hotspotized and added together to create custom shapes. Dynamic movements can be defined by splitting motion into keyframes related by temporal or merely ordinal constraints.

3.1.1 Gesture Spotting in Discretized Space

The nature of perceptual user interfaces is such that input signals are often in the form of continuous streams. The continuous stream often comprises meaningful information intermingled with portions of the signal that are irrelevant or even confounding — i.e. noise. Locating where meaningful information begins and ends within the continuous stream — i.e. distinguishing signal from noise — and interpreting the input signal is called *spotting* (Rose, 1992). *Gesture spotting* is the identification of meaningful gestures in a real-time continuous stream where gestures are performed alongside intermediate, unintentional movements (Alon et al., 2009; Elmezain et al., 2010; Kang, Lee, and Jung, 2004; Lee and Kim, 1999; Malgireddy et al., 2010). *Identification* in this sense denotes both the segmentation of meaningful portions of the stream from noise; and identifying what those meaningful portions actually mean — i.e. *classification* or *labeling* in machine learning terms (Alon et al., 2009; Malgireddy et al., 2010).

For gesture-based interfaces, the difficulty of gesture spotting depends on the choice of capture medium (see Section 2.1). On constrained capture media, gesture segmentation is often inherent because input devices or limbs do not make contact with the gesture-sensing medium during intermediate movements. On equipped capture media, segmentation can be triggered manually, e.g. via a button on the input device. With perceptual interfaces, gesture spotting must be performed

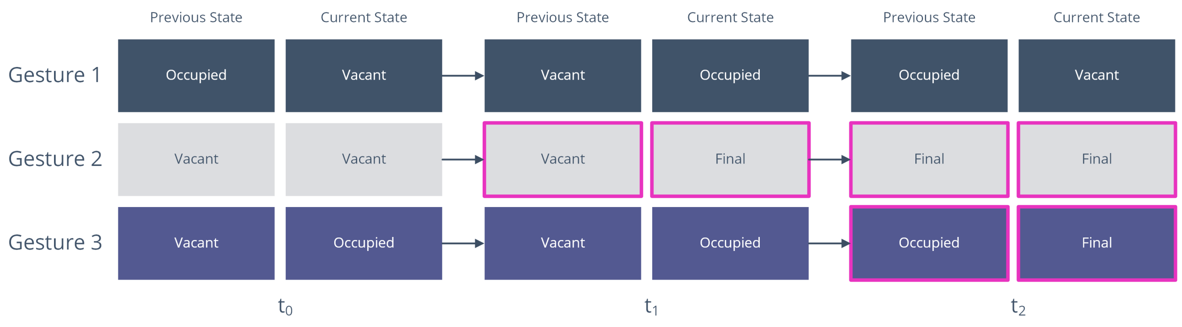


Figure 3.5 – A 2-dimensional array keeps track of what state each gesture is in. With each incoming frame from the input device, the present state expires and gets pushed back in the array to represent history. The highlights indicate candidacy for gesture recognition (Gesture 2, as in Figure 3.4, has one keyframe).

Listing 3.1 – Part 1 of pseudocode for gesture spotting in discretized space. This part of the algorithm records the traversal of hotspots.

```
# Traverse all gestures
foreach gesture in AllGestures:
  i = AllGestures.IndexOf(gesture)
  # Advance gesture state
  States[i][0] = States[i][1]
  # Traverse each keyframe of the current gesture
  foreach keyframe in gesture.Keyframes:
    j = gesture.Keyframes.IndexOf(keyframe)
    # See if the limb tracked by the current gesture
    # occupies the any hotspotized areas for this keyframe
    if keyframe.Bounds.Contain(gesture.TrackedLimb.Coordinates):
      # If it does, record *when*
      Times[i][j] = TimeOfNow
      # Also check if this is the last keyframe
      # and set gesture state
      if keyframe == gesture.Keyframes.Last:
        States[i][1] = Final
      else:
        States[i][1] = Occupied
    # If this keyframe does not register any activity...
    else:
      # Set the gesture state accordingly
      States[i][1] = Vacant
```

automatically. (Indeed, as Turk and Robertson (2000) imply, this one of the characteristics that define perceptual interaction.)

Defining gestures as sequences of hotspot sets in discretized space significantly alleviates the difficulties associated with perceptual gesture spotting. The interaction space consists of a finite volume, and hotspots are defined over a finite number of discrete compartments that span portions of the interaction space. Hotspots behave like virtual buttons that have two states: an *on* state for when the tracked limb or input device occupies the hotspot, and an *off* state for when the hotspot does not register occupation. These characteristics of operating in discretized space reduce the gesture spotting problem to one that can be solved by simple thresholding (Hartmann, Abdulla, et al., 2007; Hoste, Rooms, and Signer, 2013) — without requiring the application of machine learning concepts.

An algorithm for traversing hotspots within a workspace and performing gestures spotting can be expressed in two parts. This algorithm assumes that the interface between the input device and the application logic is event-driven, but it can adapt to a polling model as well[1]. When using the Microsoft Kinect sensor as the input device, this means that the positions of the tracked limbs that comprise the skeletal model are published to the application at regular intervals (by default, at 30 frames per second). The two parts of the algorithm are executed consecutively upon the arrival of each data frame.

A second assumption of the algorithm is that temporal constraints that separate keyframes are

[1]See 100experiencepoints.com/polling-vs-event-driven-models/ for a discussion on event-driven and polling interfaces between application components.

Listing 3.2 – Part 2 of pseudocode for gesture spotting in discretized space. This part of the algorithm determines whether the previously recorded activity should trigger any of the gestures we are spotting for.

```
# Traverse all gestures
foreach gesture in AllGestures:
  i = AllGestures.IndexOf(gesture)
  # Single- and multi-keyframe gestures are handled differently
  if gesture.Keyframes.Count == 1:
    # Check states
    if States[i][0] == Vacant and States[i][1] == Final:
      gesture.Engage()
    if States[i][0] == Final and States[i][1] == Vacant:
      gesture.Disengage()
  else:
    # Check states
    if not(States[i][0] == Final) and States[i][1] == Final:
      if InterKeyframeTimeoutIsNotExceeded(i):
        gesture.Engage()
    if States[i][0] == Final and not(States[i][1] == Final):
      gesture.Disengage()
```

defined in terms of a fixed inter-keyframe timeout value. For a gesture to register, hotspots that belong to its consecutive keyframes must be traversed before the timeout occurs.

The first part of the algorithm is a loop that traverses hotspots and checks if the hotspot is occupied by a limb that is being tracked. Listing 3.1 provides pseudocode for this portion of the algorithm. Notice that each gesture has one limb associated with it, and it does not care about where the other body parts are. The algorithm can be adapted to accommodate gesture designs that require tracking the movement of more than one limb, but this more complicated case lies outside the scope of the current implementation (see Section 3.2) and this thesis.

A 2-dimensional *jagged* (or *ragged*) *array* where arrays of different lengths comprise the elements of an encompassing array [2] keeps a record of the activity going on in the hotspots. The size of the outer dimension is equal to the number of distinct gestures (in machine learning terms, gesture *classes*) that can be recognized by the system. The inner arrays have one element for every keyframe of the gesture that they relate to. Each element of the inner array holds *time* information. In the C# programming language, used to realize the implementation discussed in this thesis, this is done by using the `DateTime` structure provided in the `System` namespace of the *.NET 4.5 Framework* class library[3]. Figure 3.4 shows a visualization of this array. Notice on Listing 3.1 that elements of this array are updated only when a keyframe registers a tracked limb. The absence of a limb in the hotspots of a keyframe does not delete previous information.

Another 2-dimensional array is needed to keep a memory of what state every gesture is in. Trivially, the limb being tracked by the gesture may be occupying a hotspot that belongs to one of the gesture's keyframes, in which case the gesture would be in an `Occupied` state; or none of the gesture's keyframes may be registering activity, in which case the gesture's state can be called `Vacant`. Additionally, it is important to know *which* keyframe is registering activity — is it just any of the keyframes, or is it the last one? If the gesture's ultimate keyframe has registered the most recent activity, this marks the end point of the gesture trajectory and segments the gesture

[2] stackoverflow.com/questions/18269123
[3] msdn.microsoft.com/en-us/library/system.datetime

from subsequent "noise." Thus, a third state — which we can label as `Final` — is used to keep track of this information. These states are held in a 2-dimensional array of size $n \times 2$; where n is the number of gesture types that can be recognized by the system, and two array elements for each gesture record the *previous* and *current* states of the gestures respectively. Notice, again on Listing 3.1, that the states are updated with each incoming event from the input device: The expiring *current* state expires gets pushed back and becomes the *previous* state, while the new *current* state is computed anew (see Figure 3.5 for a visualization of this activity).

Thus, we now know the times when every keyframe of every gesture has been last occupied by the limb that it is tracking. Now, the second part of the algorithm determines whether this activity corresponds to any of the gestures we are spotting for (Listing 3.2).

Notice that this is an overview of the basics of the gesture spotting algorithm. The implementation is subject to various additions in order to handle edge cases, allow further expressive power in terms of interaction designs (such as the *tap/hold* option for output events), and introduce interactive feedback (similar to Zamborlin et al.'s (2014) "follow" mode) to the user interface. I refer interested readers to the source code.

3.2 Hotspotizer

This section presents an overview of the features and workings of Hotspotizer: an end-to-end, standalone toolkit for end-users' authoring gross mid-air gestures and mapping them to keyboard events. In this manner, Hotspotizer can relay keyboard commands to arbitrary third-party desktop applications and adapt any user interface for gesture control. Hotspotizer currently only supports Microsoft's Kinect for Windows and Xbox Kinect sensors.

See Figure 3.6 for screenshots of the three modules that make up Hotspotizer: (1) The *Manager* module is for editing, saving and loading collections of gestures; and adding, deleting and editing individual gestures. (2) The *Editor* comprises the main workspace where gesture authoring occurs. Finally, (3) the *Visualizer* module presents visual feedback while the gesture recognizer is engaged and relaying system-wide keyboard commands.

3.2.1 Usage

To describe how mid-air gestures can be authored and mapped to keyboard events using Hotspotizer, lets consider the case of an end-user, Ali, who would like to adapt a document viewing application for gesture control. (Ali may require this functionality in contexts where touching a device to navigate a document is undesirable; e.g. when performing surgery on a patient or repairing an oily mechanism.) Figure 3.7 depicts his workflow, and the numbers in parentheses throughout this section relate to the numbered panes in the figure.

Ali has to be able to cycle up and down between the pages of a document, as well as zoom in and out, using mid-air gestures. These actions may correspond to different keyboard commands depending on the document viewing application; lets assume that, respectively, the *Page Up*, *Page Down* keys and the *Ctrl + Plus* and *Ctrl + Minus* key combinations are used. To cycle between pages, the left hand is swiped in air as if turning the pages of a real, albeit large book. To zoom in and out, the right hand performs beckoning and pushing motions. Figure 3.8 shows one way of describing these two gestures in terms of hotspots (for brevity, the *page up* and *zoom out* gestures are not shown).

Creating and Editing Gestures

Hotspotizer greets Ali with the Manager module containing empty gesture collection upon launch. Ali creates a new gesture in the collection, launching the Editor module **(1)**. Here, Ali

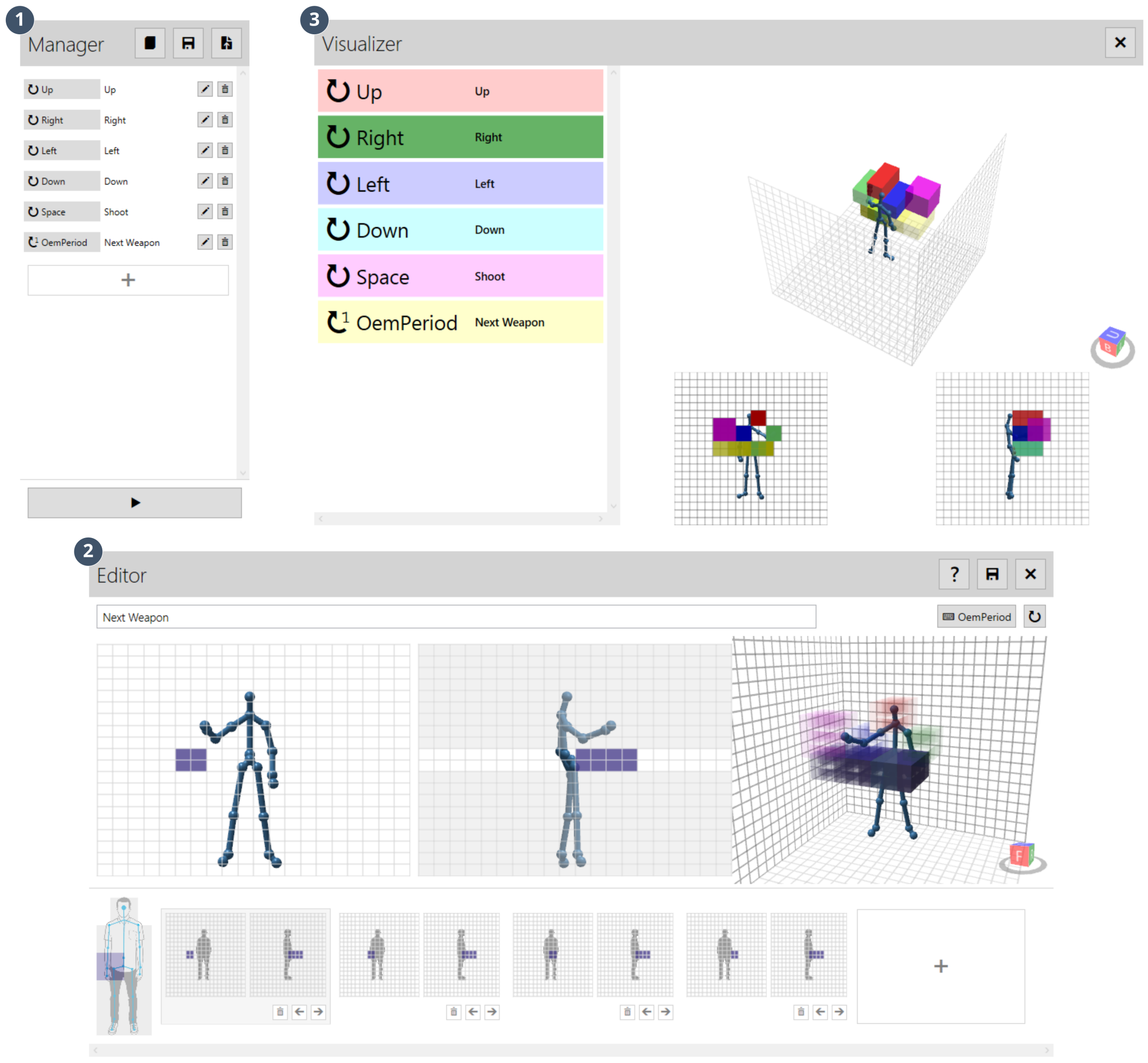


Figure 3.6 – Hotspotizer consists of three modules: (1) The Manager lists all of the gestures in the current collection and allows creating, saving and loading collections as well as adding, removing and editing individual gestures. (2) The Editor is the main workspace where gestures are authored. (3) The Visualizer provides interactive feedback on available and recognized gestures.

assigns a name for the gesture for easy recall, specifies the Page Down key to be triggered when the gesture is performed, and confirms that the *loop* toggle button is not checked – otherwise performing the gesture and holding the tracked limb over the hotspots in the last frame continues to hold down the keyboard key assigned to the gesture **(2)**.

Ali moves on to the main workspace where they use the front- and side-views over a representation of the tracked user to mark the positions of the hotspots for the first frame. Initially, all of the cells in the side view are disabled and grayed out. Marking cells on the front view enables the corresponding rows on the side view, whereupon Ali can mark the vertical and depth-wise position of their hotspots. Once hotspots are specified in all three dimensions by using these two grids, they appear on the 3D viewport on the right.

Once Ali completes marking the first frame's hotspots, they can proceed to add another frame using the button next to the timeline of keyframes, and then another **(3)**. Finally, after marking the second and third frames' hotspots, Ali selects the left hand as the limb that will be used in performing this gesture.

After saving the first gesture into the collection, Ali is taken back to the Manager module where they can add the remaining gestures and see the existing gestures to review, edit or delete them **(5)**. Once they are satisfied with the gesture collection they created, Ali can save the collection into a file for later use **(6)**.

Testing Designs and Controlling External Applications

At any time when using the Editor module, if they have a Kinect sensor connected to the computer, Ali may step in front of the sensor and see a rendering of the skeletal model of their body on the front, side and 3D viewports **(4)**. This feature can be used to rapidly test and tune hotspot locations at design time.

Testing over the whole gesture collection is available through the Visualizer module **(7)**. This module depicts a list of the gestures in the current collection and all of their hotspots on 3D, front and side viewports. Each gesture is shown in a different color. On the 3D viewport, transparency implies the order of hotspots. Hotspots glow when the tracked limb enters them in the correct order.

The Visualizer module also embeds the keyboard simulator. Launching the visualizer attaches a virtual keyboard to the system, which relays associated key events upon the successful performance of gestures. The visualization and the emulator continue to work when Hotspotizer is not in focus or is minimized.

3.2.2 Space Discretization Specifics

In the current implementation, the space around the skeletal model tracked by the Kinect sensor is partitioned into cubes that are 15cm on each side. The total workspace is a large cube that is 3m on each side. While this is much larger than both the horizontal and vertical reach of many people; this is by design, to accommodates unusually tall users. The centroid of the cube that comprises the workspace is affixed to the "hip center" joint returned by the Kinect sensor, which roughly corresponds to the centroid of the user's body. By specifying and tracking joint movements relative to the user's skeletal model rather than the sensor's position in real space, the user interfaces leverages the user's sense of proprioception (Shoemaker et al., 2010) in gesturing.

To describe gestures, the cubic cells within the workspace are marked to become hotspots – or *hotspotized* – that register when a specified limb passes through them. Hotspotizing is accomplished by using front and side views in the Editor workspace. The front view is used to specify the horizontal and vertical positions of the hotspots. The side view is used to confirm the vertical and specify depth-wise positions. The design of this interaction style was inspired by architectural and engineering drawings.

Limbs available for tracking are the hands, feet, elbows, knees and the head. Each gesture can track only one limb. The design of user interface for authoring gestures that utilize multiple limbs without embodying excessive complexity can be explored in future work (see Section 5.3).

In order to enable the authoring of dynamic movements along with static poses, movements are split into discrete keyframes. A timeline in the Editor module shows the keyframes and allows adding, removing, reordering and editing actions. Hotspots within subsequent frames do not need to be adjacent, but the frames need to be traversed in the correct order and within a certain time limit for a gesture to be recognized. The inter-frame timeout in Hotspotizer is pre-set to 500ms. If more than 500ms elapses between a tracked limb engaging hotspots of subsequent frames, the gesture is not recognized. In order to relieve the user interface from complexity, this parameter has not been made adjustable.

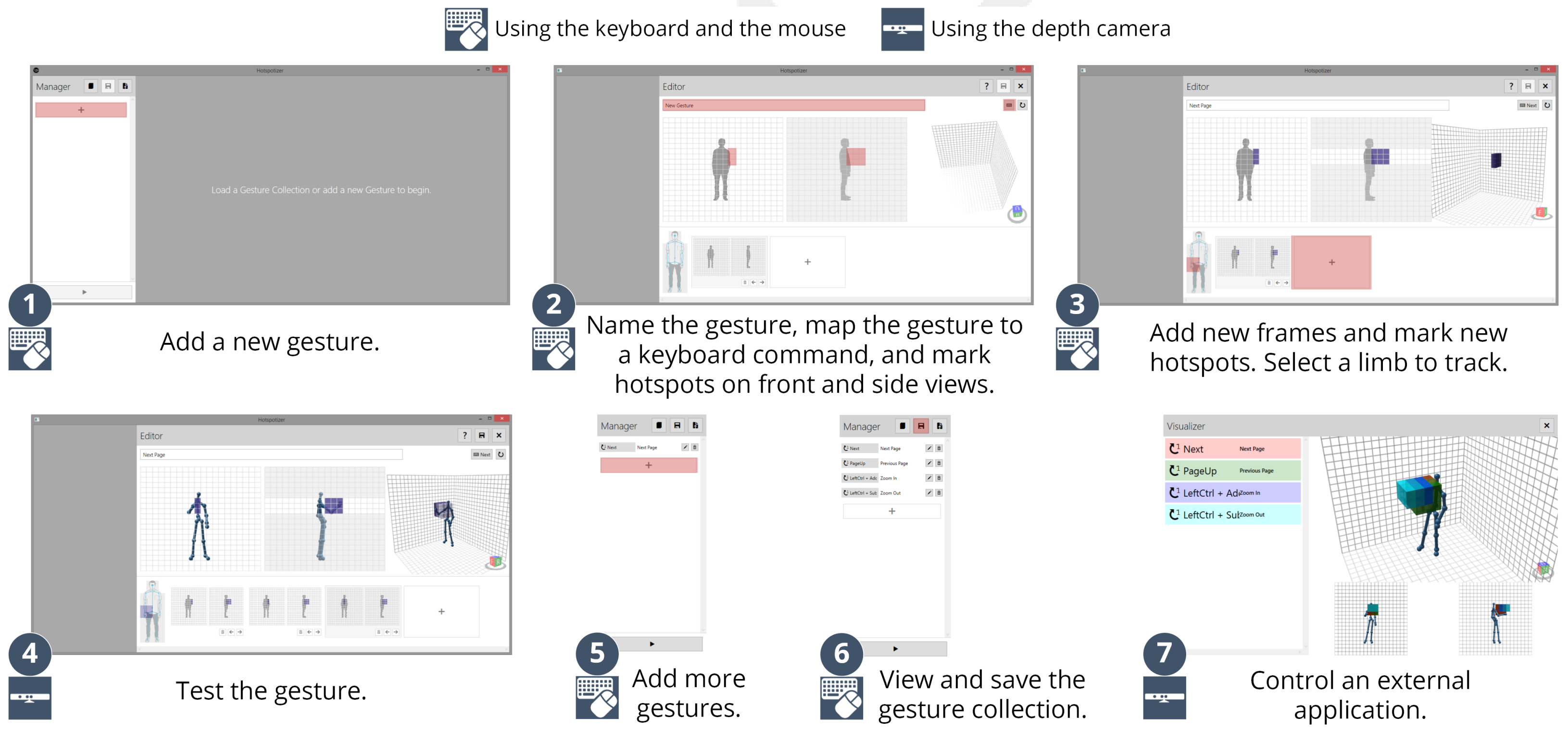

Using the keyboard and the mouse
Using the depth camera
1
Add a new gesture.
2
Name the gesture, map the gesture to a keyboard command, and mark hotspots on front and side views.
3
Add new frames and mark new hotspots. Select a limb to track.
4
Test the gesture.
5
Add more gestures.
6
View and save the gesture collection.
7
Control an external application.
Manager
Load a Gesture Collection or add a new Gesture to begin.
Editor
Visualizer
Next
PageUp
LeftCtrl + Ad
LeftCtrl + Sul
Next Page
Previous Page
Zoom In
Zoom Out

Figure 3.7 – The workflow of an end-user authoring gestures in Hotspotizer.

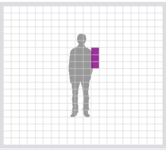
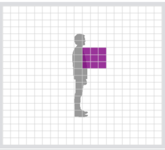
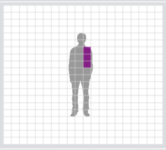
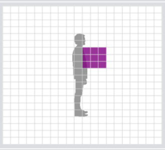
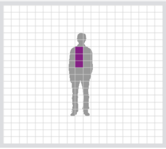
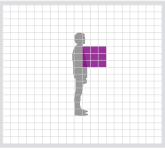
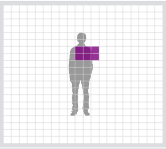
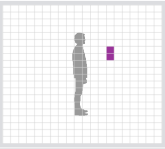
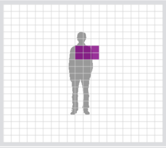
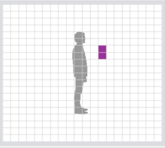
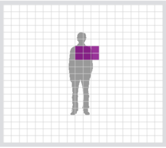
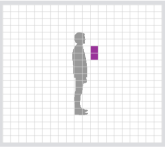
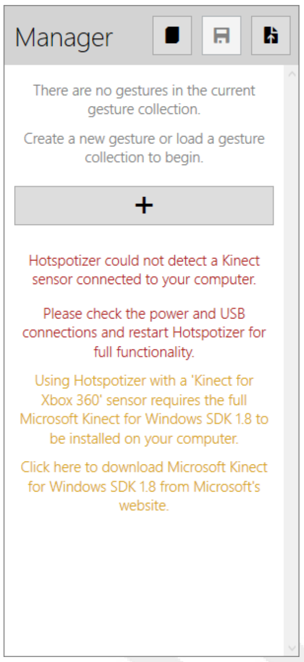
Function	Hotspots			Limb	Command
	Frame	Front View	Side View		
Next Page	1			Left Hand	*Page Down* (once)
	2				
	3				
Zoom In	1			Right Hand	*Ctrl + Plus* (once)
	2				
	3				

Figure 3.8 – *Next page* and *zoom in* gestures to control a document viewing application and the keyboard functionality that they map to. The diagrams show, respectively, hotspot arrangements for a swipe gesture and a beckoning motion.

The space discretization technique supports a versatile array of features. The size of hotspots could be made adjustable, even adaptive; to allow for fine gesturing close to the user's body and more relaxed gesture boundaries at a distance. The total workspace volume could be made adjustable. The workspace could be defined in reference to limbs other than the center or in reference to the environment; supporting whole-body movements, a larger interaction space and rich proprioceptive interactions. Temporal constraints could be made adjustable or adaptive to allow designs that exploit velocity and acceleration in gesturing. Hotspotizer does not implement these features. The design of the interface focuses on rapid development, simplification of expertise and lowering of skill barriers. Through pre-adjusted parameters for space discretization and timing, the complexity of the gesture authoring process has been reduced and the capabilities of the sensor are encapsulated within the interface. Future work may investigate empowering expert users with adjustability while maintaining the value added for non-experts.


Manager

There are no gestures in the current gesture collection.

Create a new gesture or load a gesture collection to begin.

+

Hotspotizer could not detect a Kinect sensor connected to your computer.

Please check the power and USB connections and restart Hotspotizer for full functionality.

Using Hotspotizer with a 'Kinect for Xbox 360' sensor requires the full Microsoft Kinect for Windows SDK 1.8 to be installed on your computer.

Click here to download Microsoft Kinect for Windows SDK 1.8 from Microsoft's website.

Figure 3.9 – If the Kinect sensor is not functioning properly or if pre-requisite software is not installed on the computer, Hotspotizer prompts the user with a warning message.

3.2.3 Implementation Details

Hotspotizer was written in the *C#* programming language[4], using the *Microsoft .NET Framework 4.5*[5] and the *Windows Presentation Foundation (WPF)*[6] subsystem therein to create the user interface.

The following open source packages were used:

- *Windows Input Simulator*[7] for keyboard emulation;
- *Json.NET*[8] for reading and writing gesture data to files; and,
- *Helix 3D Toolkit*[9] for 3D graphics.

Hotspotizer runs on Microsoft's *Windows 7* and *Windows 8* operating systems and requires the *Microsoft Kinect Runtime*[10], and, if used with an *Xbox Kinect* sensor, the *Kinect SDK*[11] to be installed on the user's computer.

Care has been taken to make the process of installing and running Hotspotizer as straightforward as possible, in order to accommodate diverse user populations. Hotspotizer is packaged as a Windows application that can, as convention dictates, be installed from a single executable installer file, launched from the *Start Menu*, and uninstalled from the operating system's *Control Panel*. Upon launch, Hotspotizer checks for its external requirements, the Kinect Runtime and SDK. If the requirements are unavailable, it prompts the user to install them, providing links to the web pages where they can be downloaded (Figure 3.9).

[4] msdn.microsoft.com/library/kx37x362
[5] microsoft.com/net
[6] msdn.microsoft.com/library/ms754130
[7] inputsimulator.codeplex.com
[8] json.codeplex.com
[9] helixtoolkit.codeplex.com
[10] microsoft.com/download/details.aspx?id=40277
[11] microsoft.com/download/details.aspx?id=40278

3.2.4 Discussion

The space discretization paradigm and its current implementation in Hotspotizer feature strengths and limitations that manifest as side effects of design choices.

One strength of the implementation is that gesture recognition is not influenced by the user's position and orientation within the sensor's field of view, provided that the depth image is not distorted and the sensor can build an accurate skeletal model of the user. Since the discretized workspace is affixed to the user's hip, hotspot locations are defined relative to the user's own body and the traversal of hotspots is detected properly as long as the skeletal model is built correctly. As a limitation of the depth sensor, skeletal modeling fails under certain conditions; e.g. the user turning their back to the sensor or engaging in contortions, the presence of objects that resemble a human form in the sensor's field of view, etc. Hotspotizer automatically hides the skeletal representation and halts gesture recognition when failures occur, and resumes operation when the sensor provides a skeletal model.

Certain limitations result from the design choice to prioritize leveraging a common infrastructure for end-users by mapping gestures to keyboard events. This obscures what Hartmann, Abdulla, et al. (2007) call "association semantics" — i.e. the relationship between commands relayed to applications from Hotspotizer and the resulting application behaviors — and limits the expressive power of the gesture authoring paradigm.

A further limitation is that Hotspotizer currently does not support authoring continuous — or *online* (Hoste and Signer, 2014) — gestures that affect some variable while they are being performed (as opposed to *offline* gestures that execute commands when the gesture is performed from the beginning to the end). This is not a limitation of the space discretization paradigm; since, theoretically, smaller portions of a gesture could be assigned to affect continuous variables (albeit in a quantized manner). Likewise, gestures that involve pointing at or manipulating dynamic user interface components in third party applications are not supported. This could be overcome by linking the discretized space model around the user with the virtual space of the user interface. However, these features require integration with a fully featured programming environment or language, which is beyond the design goals for this project. Exploring "tighter integration with application logic" (Hartmann, Abdulla, et al., 2007) to empower software developers is a goal for future work.

Chapter 4

Hotspotizer: Design and Evaluation

Hotspotizer has been developed through a process that utilizes user-centered design tools, in order to fulfill the needs of end-users. The choice of methods employed for the design work was influenced by methods used in previously published research on similar software (Ashbrook and Starner, 2010; Kin et al., 2012; Long, Landay, and Rowe, 1999; Lü and Li, 2012, 2013; Reis et al., 2012). This section describes the evolution of Hotspotizer; the evaluation of the final prototype; and insights gained throughout design, development, and deployment. Figure 4.1 depicts a summary of the design and development processes on a timeline.

4.1 Formative Studies

In the early stages of the design work for a tool to support authoring mid-air gestural interactions, the motivating question was *what* to build. The expected outcome from this stage — which Klemmer (2014) calls "needfinding" — is the identification of the desiderata and the core features for the gesture authoring interface. To this end, two focus group meetings were conducted with a representative sample of end-users. The knowledge derived from the focus groups, along with insights from related work and analysis of related artifacts (described in Section 2.4) informed subsequent design work. The later stages of development are described in Section 4.3.

4.1.1 Focus Groups

For user interface design, focus groups are a technique that can be used before higher fidelity design and development work to elicit user wants and needs (Nielsen, 1997). The nature of focus groups is informal and the results are generally qualitative. Focus groups have limited power,

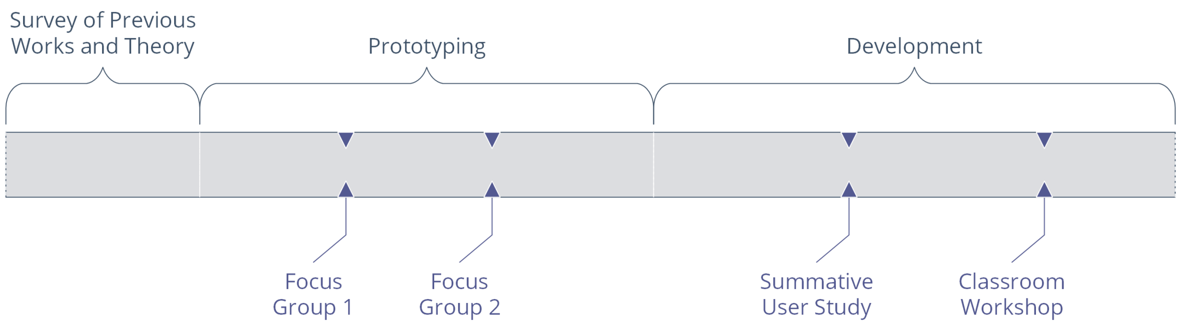


Figure 4.1 – A summary of the design and development processes.

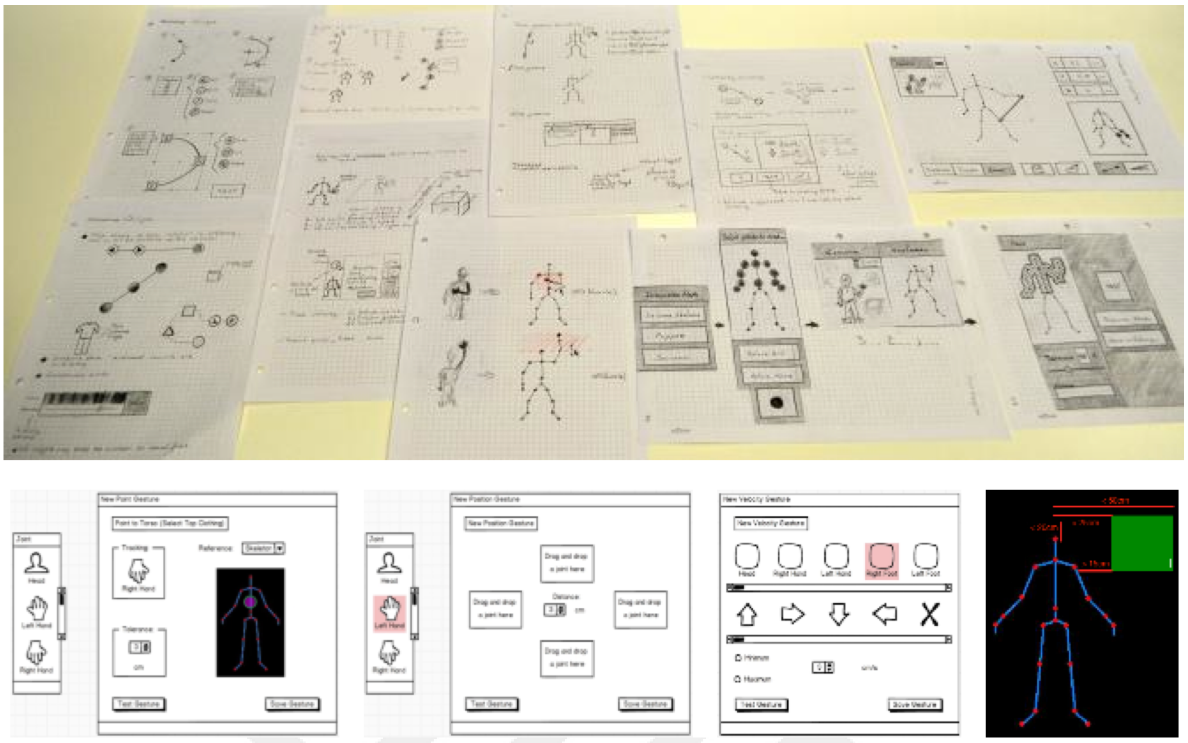


Figure 4.2 – Rough sketches, paper prototypes, digital wireframes and interactive prototypes were used to gather feedback which directed design and development.

precision and accuracy due to methodological issues (Franz, 2011; Smithson, 2000); but they are recommended as an initial needfinding strategy in the pursuit of open-ended questions (Kitzinger, 1995; Morgan, 1996; Nielsen, 1997).

Nielsen (1997) recommends "exposing users to the most concrete examples of the technology being discussed" to improve the accuracy of the data gathered during focus groups. Per his advice, discussions during the focus group meetings were facilitated using various prototypes of different levels of fidelity. Initially, concepts were produced in the form of rough sketches and paper prototypes. During a second focus group meeting, a second round of prototypes of varying fidelity were used, which included video sketches and interactive user interface prototypes built using the *Processing*[1] programming language. Figure 4.2 shows samples from these preliminary renderings.

Participants

Participants consisted of a group of 10 potential users, aged 22-31 (μ=26), from diverse backgrounds. While recruited from among students and staff of a single university and not representative of a wide demographic, they represented the target users of a gesture authoring tool well. Each had different skills and interests. Among them was an industrial designer, a semi-professional musician, an electronics engineer, a computer scientist, a museum studies student, an interaction designer, a psychologists and a legal consultant. All participants were regular users of desktop computing applications. They were all were self-reportedly familiar with mid-air gestural interaction in the context of gaming; though none had any familiarity with existing tools for authoring custom interfaces. Participants' profiles was appropriate for the aim of the study: to elicit desiderata for a tool that would enable "nonconsumers" (Christensen and Raynor, 2003; Fried, 2008) by lowering the threshold for building custom gesture interfaces.

[1] processing.org

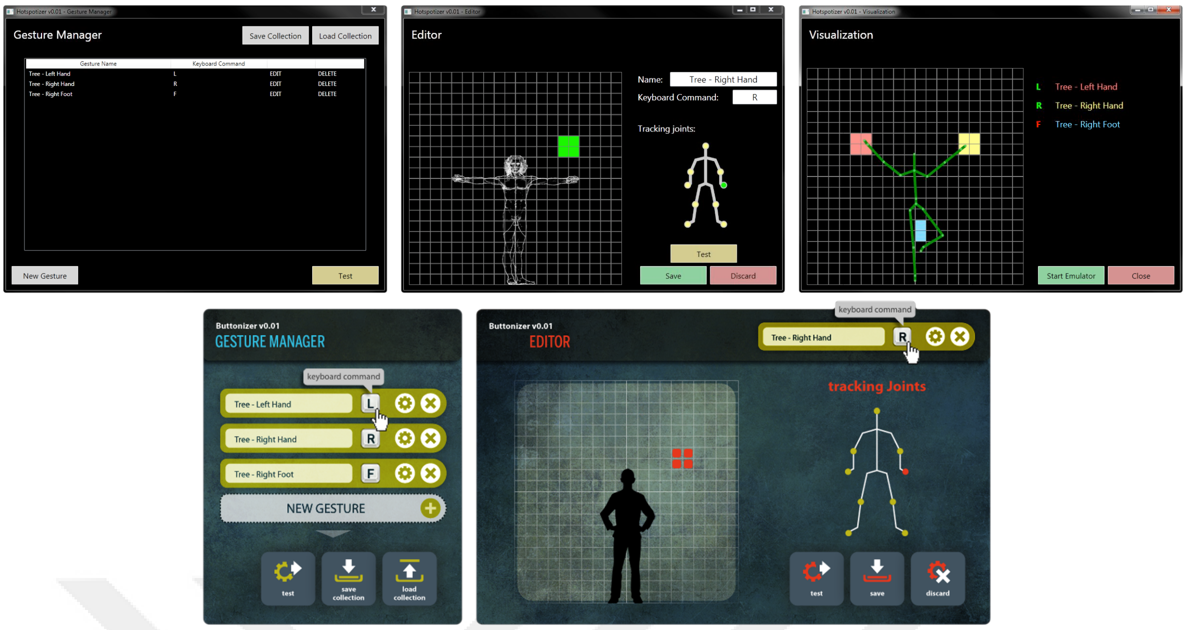


Figure 4.3 – Early designs implemented rudimentary functionality and visually differed from the current user interface. Screenshots on the top row depict a very early prototype. The bottom row depicts a version with custom graphic design elements, later abandoned in favor of components native to the operating system.

Procedure

Two meetings were conducted with the participants. Qualitative and semi-structured feedback formed the basis for the results obtained during these meetings.

During the first meeting, participants were given an introductory presentation on gestural interfaces, enabling technologies, and applications. Including this presentation, the meeting time slightly exceeded one hour. After the presentation, we discussed possible applications of gesture interfaces in participants' own domains of interest, with a focus on the use of mid-air gestures. Initial ideas for the design of a gesture authoring tool, in the form of rough sketches and paper prototypes, were presented to the participants. These initial prototypes were inspired by design considerations derived from the literature (see Section 2.3) and previous works on tools for authoring mid-air gestures (see Section 2.4). Feedback collected from participants regarding the initial prototypes formed the basis for a second round of concepts.

Another round of sketches and prototypes was prepared, some of them higher fidelity; e.g. as a mock screencast showing the use of various modules in a gesture authoring suite, video sketches depicting a gesture authoring process based on programming by demonstration (see Section 2.4), and interactive user interface prototypes realized with *Processing*. Knowledge derived from the comments and reactions of the participants to these concepts culminated in the construction of a prototype application with rudimentary features. This application implemented a simple version of the space discretization paradigm for authoring mid-air gestures, which was identified during the course of the second meeting.

Results

Early concepts that were presented to focus group participants included an end-to-end environment for creating gesture-controlled interactive movies that fused gesture authoring and content creation in one application; ready-made widgets that pre-implemented gesture control and plugged into existing development and design environments; and tools to overlay information (such as

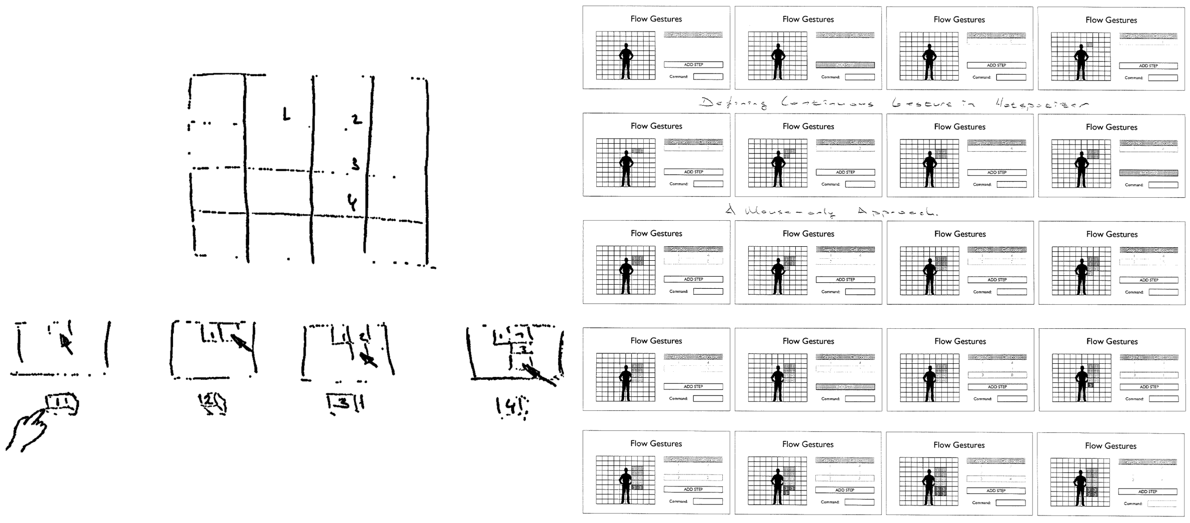


Figure 4.4 – Rough early design sketches and higher fidelity wireframes were used to reflect on the workflow and user interface components for visualizing and authoring dynamic movements of human limbs.

the distance between two specific joints) onto a visualization of a skeletal model, to complement textual programming. Discussions on possible applications for custom gesture control revealed that a modular approach that can interface with a diverse variety of applications is preferable to a full-blown content creation suite. Moreover, even among users engaged in design or programming activities, tools used for these purposes varied greatly. This illustrated the value of a standalone application rather than a tool that generates code in a specific programming language or plugs into a specific environment.

The idea of creating virtual buttons or hotspots in the space around the user and using them to define gestures was depicted in the sketches that were shown in the second meeting, as well as an interactive mockup developed in *Processing* (see Figure 4.2). Other ideas included an application that recognized static poses and a graphical language consisting of atomic primitives for composing gestures. Here, the concept of space discretization was proposed by a participant, an interaction designer. Upon interacting with the mockup of an interface where free-form areas in space can be made into gesture-tracking hotspots, she commented that she often makes use of squared paper when sketching. Instead of defining free-form regions in space, why not divide space into squares and constrain hotspots to these squares? Further discussion with participants revealed that this paradigm is grasped more easily than composing with atomic actions or constraints, or even demonstration. Moreover, using a visualization of the skeletal model and the space around it allows direct manipulation (Hutchins, Hollan, and Norman, 1985); encapsulates the limitations and prospects of the design space; capitalizes on proprioception; and can mediate interaction through a tight feedback loop (Wilson, 2012).

Hotspotizer was developed as an implementation of this *space discretization* paradigm yielded by these workshops. The design guidelines derived from Olsen (2007) and Shoemaker et al. (2010) informed initial prototypes. They were also used as filters that transformed the findings from the formative studies into a concrete user interface design. The decision to map gestures to key press events from an emulated keyboard was grounded in Olsen's (2007) principle for building on an infrastructure that is common across users and situations. The use of the user's centroid — rather than the Kinect sensor itself — as the origin for the grid of hotspots functions as a binding between personal and extrapersonal space and leverages the user's sense of proprioception. Adherence to the remaining three design considerations guided the use of focus groups and user studies as design tools.

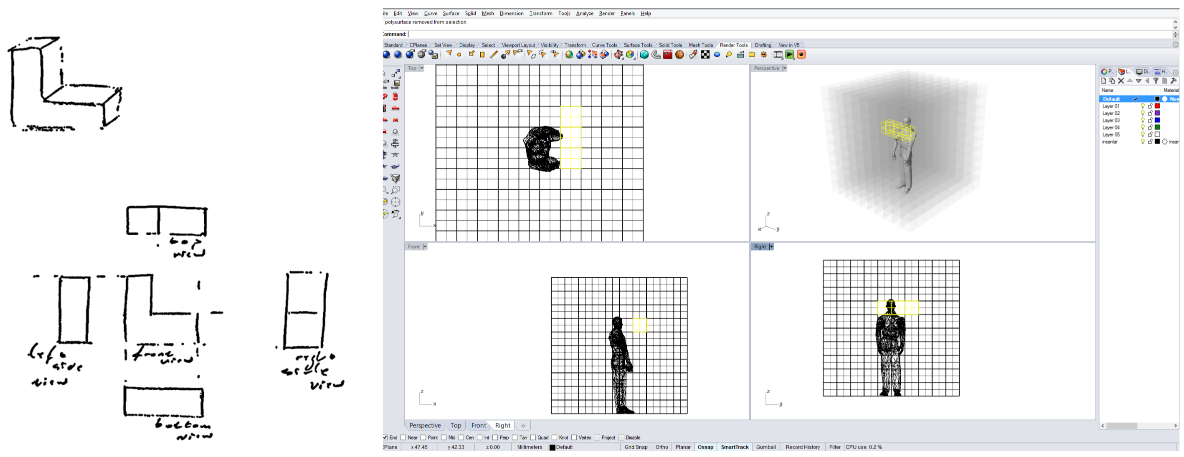


Figure 4.5 – User interface components for visualizing and manipulating 3-dimensional spatial constraints using front- and side-views was inspired 3D modeling software and technical drawings for expressing architectural and engineering designs.

4.2 User Interface Design

The knowledge on user needs, abilities, and preferences uncovered during the formative studies, along with insights derived from the analysis of gesture authoring software developed previously for both research and commercial purposes, has been utilized in the construction of the user interface for Hotspotizer. Design insights from previous works were collected from both previously published research findings, and from experience in using the artifacts. (Section 2.4 describes these efforts.)

The initial prototype featured only 2-dimensional gesture authoring capability, ignoring the depth component and tracking motion in the horizontal and vertical dimensions only. No key combinations were allowed, gestures could only be mapped to single key presses. There are also significant differences in the functionality and visual design of the user interface between this preliminary prototype and the final application. Two iterations on the interface design are shown in Figure 4.3.

Following the initial stripped-down prototype, new features such as manipulating 3-dimensional spatial constraints and defining movement by using a timeline of keyframes (Figure 4.4) were added to the application. Each feature underwent an iterative design and development process; beginning with rough sketches, evolving through prototypes of increasing fidelity, and finally culminating in the implementation (Figure 4.6).

Beside tools for authoring gestural interactions (see Section 2.4), a wide range of software and tangible design tools inspired the composition of the user interface. The interaction style for the main workspace for authoring 3-dimensional constraints using front- and side-views was inspired by 3D modeling software, along with architectural and engineering drawings (Figure 4.5).

The current version of the user interface is discussed in a detailed manner in Chapter 3.

4.3 Summative Studies

To evaluate Hotspotizer in use, two studies were conducted. The first was a study with 5 users to assess if Hotspotizer conforms to its design rationale. The second was a class workshop with 6 students working in pairs to build interactive prototypes of gestural interfaces. Qualitative results from these summative studies confirm that Hotspotizer conforms to its design rationale Figure 4.7 summarizes the observations that relate to the design guidelines, and the pertinent details are discussed below.

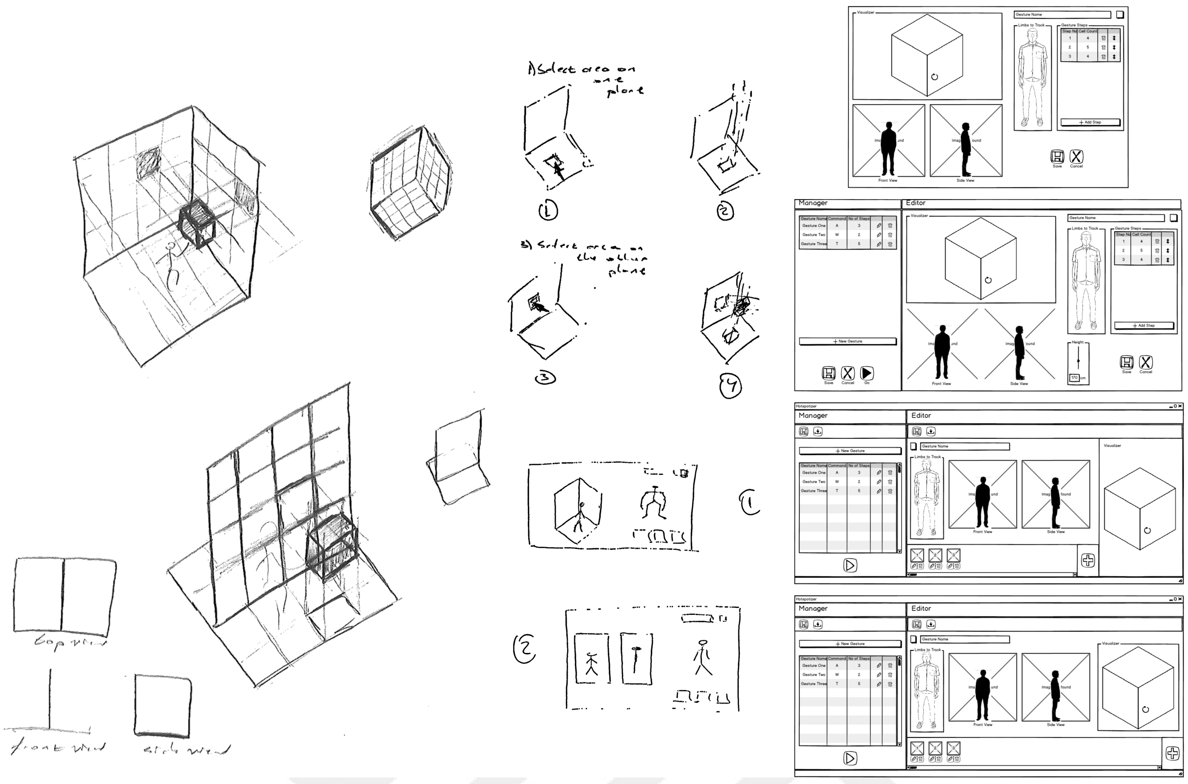


Figure 4.6 – Rough early design sketches and higher fidelity paper prototypes were used to reflect on the workflow and user interface components.

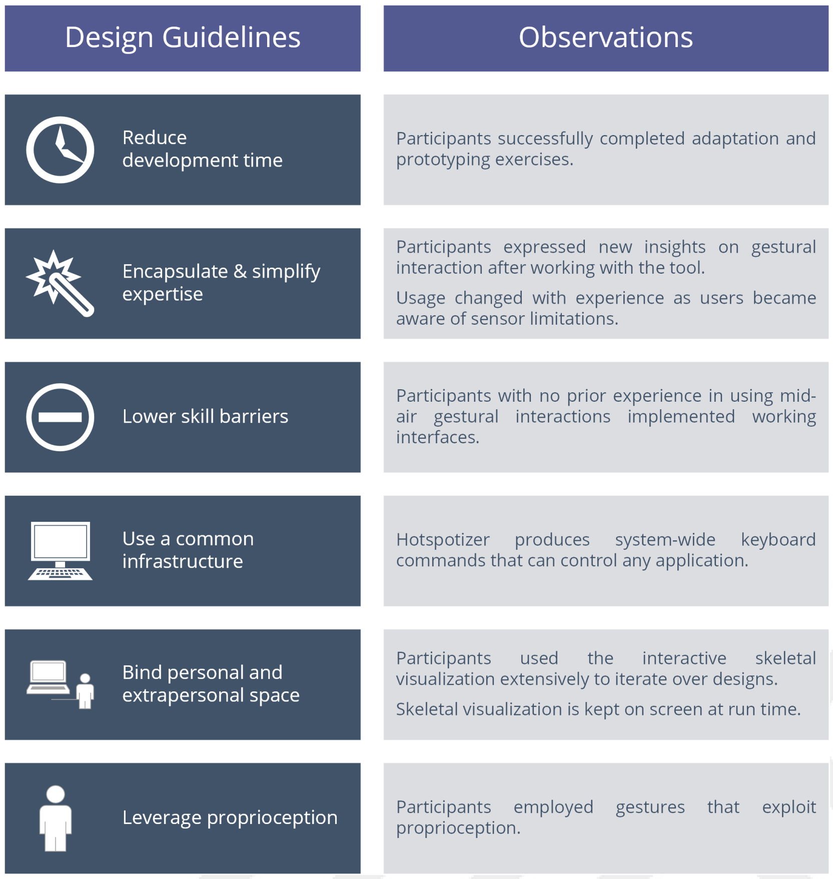
4.3.1 User Study

Participants

For the user study, five graduate students from a single university were recruited: an industrial designer, a museum studies student, a computer scientist, a psychologist and an interaction designer. These were not the same people who participated in the previous workshops. Participants were given a pre-study questionnaire where, on average, they self-reported a low level of experience with computer programming (μ=2.1 on a 5-point Likert scale) and a low-medium level of experience with using mid-air gesture-based interfaces (μ=2.4).

Procedure

Participants were given the task of adapting a non-gestural interface for a computer game to gesture control. They were provided a PC with a Kinect sensor. The game that was to be adapted for gesture control was a side-scrolling platformer — this style of game was selected since users were expected to be fully familiar with the game mechanics and not be distracted from the process of gesture authoring. The participants were not given specific gestures to implement, but the game required three commands to operate: *left* and *right* for movement, and a *jump* command. Participants were required to play through and complete the first level of the game using gestures at the end of the study. Participants first finished one level of the game using a keyboard; and they were gave a demonstration of Hotspotizer before they began authoring gestures to control the game. Participants were not explicitly instructed to think aloud (Boren and Ramey, 2000; Holzinger, 2005; Jaspers et al., 2004; Nielsen, 1993b), but they nevertheless made comments during the task, which were recorded along with observations on behavior.



Design Guidelines	Observations
Reduce development time	Participants successfully completed adaptation and prototyping exercises.
Encapsulate & simplify expertise	Participants expressed new insights on gestural interaction after working with the tool. Usage changed with experience as users became aware of sensor limitations.
Lower skill barriers	Participants with no prior experience in using mid-air gestural interactions implemented working interfaces.
Use a common infrastructure	Hotspotizer produces system-wide keyboard commands that can control any application.
Bind personal and extrapersonal space	Participants used the interactive skeletal visualization extensively to iterate over designs. Skeletal visualization is kept on screen at run time.
Leverage proprioception	Participants employed gestures that exploit proprioception.

Figure 4.7 – Qualitative findings from two studies affirm that Hotspotizer is in keeping with our design rationale.

Results

All five participants were able to complete the assignment successfully, within 5-14 minutes (μ=7.4min) after being given the demonstration and left alone with the interface. Participants commented that the interface was *"easy to use"* and *understandable*.

Participants iterated rapidly over gesture designs — for each gesture, they went through 2-6 (μ=3) cycles of hotspotizing cells on the Editor and moving into the sensor's range to test designs in person. Static hand positions were preferred for the *left* and *right* commands, while the *jump* command inspired diverse gestures including kicking and nodding. A common error across participants was that they marked areas outside the reach of the arms and the legs.

Semi-structured post-study interviews revealed that users had gained insights about the workings of skeletal tracking gestural interfaces. Support for full-body postures such as jumping, along with compositions that involve multiple limbs and grab detection were reported to be desirable as additional features. This is in line with my vision for future work (see Section 5.3).

4.3.2 Class Workshop

Participants

The workshop was conducted with 6 graduate students who were taking a course titled "Design Thinking for Interactivity." Participants worked in groups of two, with the three groups working the same time on different PCs.

All participants — per course requirements — were familiar with interaction design concepts and user interface prototyping processes. They self-reported low levels of experience with textual computer programming and using mid-air gestures to interact with computing applications outside

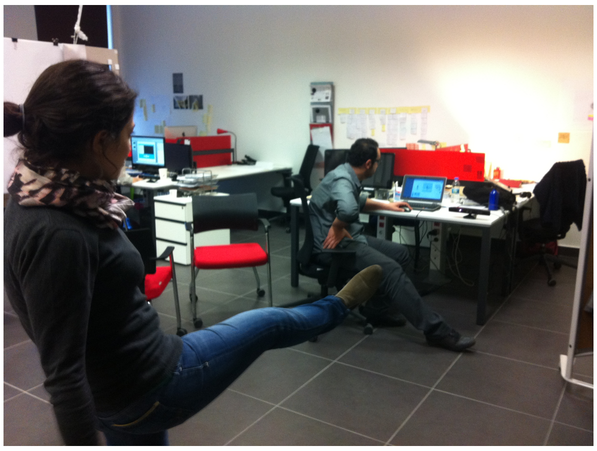


Figure 4.8 – User strategies included working in pairs. One user performs gestures in front of the sensor while the other marks hotspots that correspond to limb positions.

of gaming (μ=1.8 and μ=2 on a 5-point Likert scale, respectively). One exception was a participant who claimed some understanding of software development concepts due to his experience as a graphic designer working on computer games, but even he did not have any programming experience. Again, per course requirements and the curriculum, participants were familiar with all of the software used during the study, except for Hotspotizer.

Procedure

The study began with a 20-minute presentation on how the Hotspotizer interface works. Participants were then tasked with creating interactive prototypes for three different systems (one per group) by following a single given use case for each system. The three systems comprised interactive digital signage for a movie theater, a penalty kick game and a video jukebox for public use. Participants were to create the visual design for the system's screens in Microsoft PowerPoint[2], and assign gestures to shortcut keys in PowerPoint to add interactivity. Each group was provided a Kinect sensor, a PC with Hotspotizer and PowerPoint installed, and a cheat sheet that exposed keyboard commands available in PowerPoint. A diverse set of interactions is possible in this manner, including moving between screens, starting and stopping video, adjusting the volume of the system, displaying versatile animations and automatically triggering timed behavior.

Results

All of the three groups were able to complete their implementations of an interactive prototype, from scratch, within the 60 minutes allocated for the activity. On average, about one third of this time was spent ideating and sketching designs, one third on composing visuals in PowerPoint and one third on authoring gestures with Hotspotizer.

The digital signage prototype was controlled by six hand gestures that involved pointing, swiping, pushing and pulling. The penalty kick game employed four gestures: kicking a ball towards the left, the right and the center; and making a large circle with the hand to restart. The video jukebox prototype was controlled by five gestures that comprised swipes and touching various parts of the head and the torso.

Participants expressed enjoyment from the process of creating interactivity and working with new interface technology. *"A few days ago I did not even know that [mid-air gesture control] was possible. Now I just made my own working design,"* commented one participant.

Initially, users did struggle to understand the workings of the skeletal tracking. Two groups attempted to use gestures with minute differences that the Kinect sensor may not distinguish from

[2]office.microsoft.com/en-us/powerpoint

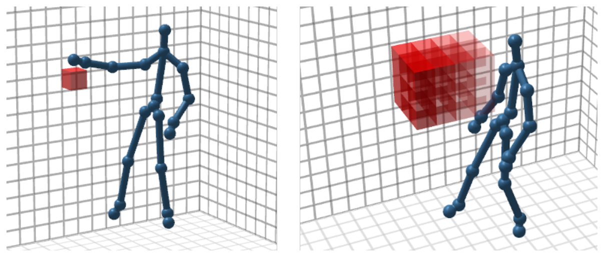


Figure 4.9 – Initially, users preferred gesture designs that involved small hotspots and unspecified motion. Frames were added to constrain motion, and hotspots were enlarge to allow for variations during gesturing. Here, both panes depict hotspot configurations that may be used for a “punch” gesture. The configuration on the right is more conducive to robust recognition because of its sequentially constrained and spatially relaxed nature, compared to the rather extremely simplistic design on the left.

each other, such as touching the eye with one finger versus touching the nose. Through trial and error, participants revised their gesture designs to match the capabilities of the sensor.

A limitation to the space discretization paradigm was expected to surface: Hotspots configured for one user could be inappropriate for another user due to differences in body size. After the three groups completed their projects, they tried out each other’s implementations to see if this was the case. The only time when gestures from a new user were not recognized was in the case of the football game, where large leg movements were involved. Differences in the length of the legs hindered gesture recognition across users. Tuning the gesture design to involve larger hotspot areas alleviated the problem. When using hand gestures, no issues were apparent.

When working in pairs rather than alone, users adopted a different strategy when editing gestures: A single user would mark hotspots using the static on-screen silhouette of a human body as a reference and then test using the interactive representation. Working in pairs, one of the users preferred to stand in front of the sensor and perform gestures, while the other watched the moving representation on the screen and used it as a reference when marking hotspots (Figure 4.8). To allow a single user to enjoy the advantages of using the interactive skeletal model for authoring, future work can implement the ability to infer hotspots from demonstration, along with voice control to interact with the program from a distance (see Section 5.3).

Participants were interviewed after the study, where they suggested that while editing, being able to see where hotspots belonging to previously authored gestures reside could be beneficial. This visualization was later added into the Editor module in a later version of Hotspotizer.

4.3.3 Generalizable Observations

During the summative studies, observations that are relevant for the design of mid-air gestural interfaces in general were encountered.

Users who self-reported little experience with mid-air gestural interfaces (a vast majority among participants) tended to be unaware of the limitations regarding the sensor’s field of view. This manifested as an initial tendency to stand too close to the sensor and perform gestures in areas outside the sensor’s field of view. Within minutes, users adjusted to become aware of the boundaries of the interaction area. To promote users’ awareness of the depth sensor’s field of view, the depth map provided by the sensor could be displayed on screen, as opposed to displaying the user’s skeleton alone.

As they tested and used their own gesture-controlled designs, users tended to keep the Hotspotizer interface open and utilize the on-screen representation of the human skeleton. This confirms that the requirements for including a tight feedback loop and a representation for reporting the user’s actions within space are justified. Based on this observation, I can recommend that interfaces based on mid-air gestures include a representation of the tracked skeleton(s).

In general, when designing gestures, users preferred to start with static poses or specify only the end point of a gesture trajectory, utilizing only one frame to implement their designs. In simple cases, such as in controlling the side-scrolling platformer, these designs did suffice. However, as the quantity and complexity of gestures in the interface increases, this approach results in a high number *false positives* in gesture recognition due to intermediate movements intersecting hotspots. Users, due to inexperience, did not anticipate this. Through trial and error, gesture designs were revised and conflicts were resolved, by adding frames and authoring *movement* further constrain designs. Often, gesture designs resulted in *false negatives* due to spatially overconstrained designs that involved small volumes, requiring precise and accurate performance of gestures. Participants, through trial and error, revised their designs by enlarging hotspotized *volumes* to allow for some degree of ambiguity when performing gestures. The general tendency among users was to initially design gestures that were temporally or *sequentially underconstrained* and *spatially overconstrained*. Designs that minimize conflicts by *introducing sequential constraints* (i.e. more frames) while allowing for some flexibility by *relaxing spatial constraints* (i.e. more hotspots) were observed to be more conducive to robust recognition (see Figure 4.9).

4.3.4 Discussion

Olsen (2007) reasons that conventional structured usability evaluation methods are not appropriate for novel user interfaces and design tools in particular; since they violate three important assumptions of usability testing:

- *“Walk up and use.”* The first assumption is that the system being evaluated must be operable with minimal training. This holds for tools intended for a very wide non-expert user base — e.g. home appliances — and for tools that target a population with “shared expertise” — e.g. doctors or teachers. User interface design, Olsen argues, requires “specialized expertise” which will vary highly among participants recruited for a usability test, even among those belonging to the same profession.

- *The standardized task assumption.* “A task that is suitable for a usability experiment must have low inherent variability so that any variance can be assigned to the differing techniques being tested, not to variations in approach to the task or user expertise.” (Olsen, 2007) In other words, software built to support complex tasks involving a high number of steps and many solutions cannot be evaluated through conventional usability approaches. The design of gestural interactions is one such task.

- *Scale.* Due to economic and psychological issues — e.g. participants’ attention spans — usability tests must be completed within 1-2 hours. The return on investment — in terms of statistical significance — for running lengthy and multiple sessions is low.

Olsen recommends design considerations and evaluation criteria for novel user interface tools, which were described in Section 2.3 and used extensively to inform the design process for Hotspotizer. The user studies described in this chapter, while inspired by conventional usability tests in terms of the procedures followed, were conducted with a focus on obtaining qualitative results. Also in line with the approach recommended by Nielsen (2000, 2011), and Nielsen and Landauer (1993); I used usability-inspired procedures to uncover bugs and implementation errors in the system, elicit user strategies, inspire new features, evaluate the use and misuse of existing features, and determine if users can make sense of the user interface and the task in general. These efforts have proven valuable in informing design and development. In short, while I agree with Olsen that conventional usability testing with a quantitative focus is not a productive approach for the for novel interfaces; but usability methods could be adapted in these cases towards comparatively informal procedures that deliver qualitative feedback from real users. This feedback is highly valuable for user interface design.

Chapter 5

Conclusion and Future Work

This thesis described efforts in developing a *software tool for authoring mid-air gestures* to support the activities of end-users. For this purpose, through guidelines derived from the literature and a user-centered design process; desiderata, design considerations, and evaluation strategies were uncovered. These led to the development of a user interface paradigm based on space discretization for visualizing and declaratively manipulating mid-air gesture information. This paradigm was implemented in Hotspotizer, a standalone Windows application that maps mid-air gestures to commands issued from an emulated keyboard. Hotspotizer was evaluated through a user study and class workshop.

Findings from the evaluation sessions verify that Hotspotizer observes its design rationale and supports gesture authoring for end-users. Using Hotspotizer, *gestural interactions were implemented by users who did not have the skills* to use textual programming tools. Usage strategies and users' choices for gesture designs implied that users understood the *domain expertise* embedded in the interface and leveraged their *sense of personal space and proprioception* in interacting with the system. Hotspotizer was used to control other programs on a PC, making use of a *common infrastructure*.

Parts of the research described in this thesis have culminated in two academic publications. One is a paper presented at the Workshop on Engineering Gestures for Multimodal Interfaces (EGMI 2014), part of *the sixth ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2014)* (Baytaş, Yemez, and Özcan, 2014b). The other is a forthcoming paper to be presented at and published in the proceedings of *the 8th Nordic Conference on Human-Computer Interaction (NordiCHI '14)* (Baytaş, Yemez, and Özcan, 2014a).

5.1 Revisiting the Research Questions

The research described in this thesis principally sought to answer *how end-users' authoring of gross mid-air gestures for skeletal tracking interfaces be supported with a software tool*. One way to enable end-users' authoring of gross mid-air gestures for skeletal tracking interfaces has been found to be with a software tool that implements a user interface paradigm for visually representing and manipulating gesture information by *splitting motion into discrete keyframes* and using *spatial constraints confined to a discrete compartments in a 3-dimensional array.* This method has been observed to be understandable and accessible for end-users designing gestures for current perceptual sensors. Of course, as user interface technologies change over time, the benefits and drawbacks associated with this approach and other methods of authoring gestures may have to be reassessed.

A secondary research question was *specifying the desiderata and design considerations that would pertain to mid-air gesture authoring software for end-users.* The necessary and desirable features for the gesture authoring tool were determined through *formative studies* that comprised

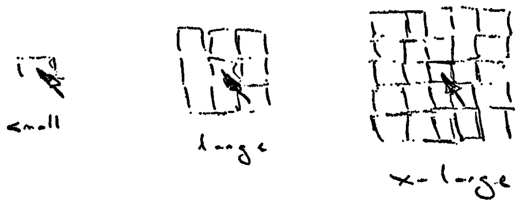


Figure 5.1 – An early stage design sketch for a new feature: adjustable brush size may support overcoming users' tendency to spatially overconstrain gesture designs.

workshops with *focus groups* where feedback was gathered using *prototypes* of varying levels of fidelity. The production of the initial prototypes, as well as the formulation of the findings from these studies were directed by *design guidelines derived from previous research*. Nonetheless, while the processes I employed seem to have produced good results; as Zimmerman, Forlizzi, and Evenson (2007) would concur, different desiderata and design considerations may also have led to a valid solution.

Finally, another secondary research question regarded the *evaluation* of the tool. Whether or not the final artifact observed its design rationale and fulfilled its purpose as an enabler for end-users and a rapid prototyping tool for designers was evaluated through *summative studies* comprising a *user study* with 5 participants and a *classroom workshop* with 6 design students. All participants successfully completed the given tasks, confirming that the design fulfills the aforementioned criteria. Qualitative findings from these studies inform subsequent iterations on the software and highlight opportunities for future work. However, due to issues set forth by Olsen (2007) and discussed in Section 2.3, structured usability tests that yield comparable quantitative results have not been conducted with Hotspotizer. This exposes an opportunity for future research on the evaluation of novel intelligent interfaces from a design perspective.

5.2 Revisiting the Hypothesis and Contributions

Revisiting my hypothesis for the accomplishments of a successfully designed tool for authoring skeletal tracking gestures; evaluations demonstrate that my design accomplishes the following:

- Hotspotizer enables *end-users* with no experience in textual programming and/or gestural interfaces to introduce gesture control to computing applications that serve their own goals. Results from a summative user study with 5 participants and a classroom workshop with 6 participants, described in Section 4.3, confirm this: Participants without prior experience in developing gesture-based interfaces have successfully completed gesture authoring tasks and demonstrated an understanding of related concepts in usage strategies and post-study interviews. These results are based on qualitative findings. As I stated previously, per Olsen's (2007) recommendation, Hotspotizer has not been subjected to usability tests that would lead to structured, quantitative, comparable results. This highlights an opportunity for future research on methods the structured evaluation of novel tools.

- The application provides *developers* and *designers* of gestural interfaces with a rapid prototyping tool that can be used to experientially evaluate designs. It has been used precisely for this purpose at a workshop with 6 participants in the context of a design-oriented classroom (Section 4.3). Design students have implemented prototypes of gesture-based interfaces using Hotspotizer as part of their workflow. As perceptual interactions become pervasive, future research can focus on getting feedback from professional designers and programmers with experience with these technologies.

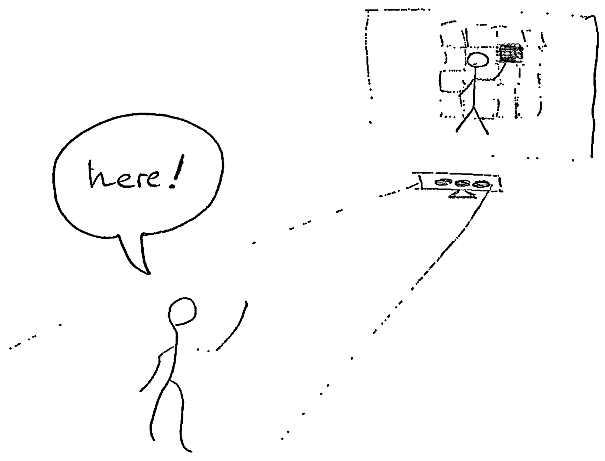


Figure 5.2 – An early stage design sketch for a new feature: inferring hotspots from demonstration while controlling segmentation via a speech interface.

As expected, Hotspotizer fulfills the criteria proposed by Zimmerman, Forlizzi, and Evenson (2007) for the evaluation of research-through-design artifacts (Frayling, 1993). The design and development *process* employs methods that have been selected rationally and documented in this manuscript. Various topics have been integrated in a novel fashion to create an artifact with the qualities of an *invention*. The resulting artifact, Hotspotizer, demonstrates *relevance*. It is situated within a real, current context; while supporting a shift towards a justifiably preferable state. Finally, the work is *extensible*, as it enables the future exploitation of the knowledge derived from it. Extensible insights gained during design, development and evaluation are documented in this thesis, and the software has been made freely available for use.

The contributions of this work, as expected, are as follows:

1. The *primary contribution* from this work is, *Hotspotizer*, a *software application* that encompasses an end-to-end solution to authoring mid-air gestures for skeletal tracking input devices. The application accomplishes its previously stated goals of enabling end-users and supporting the activities of designers, and constitutes an authentic contribution as an artifact of research through design. The latest stable release of Hotspotizer's can be obtained online as a free download, for use under the MIT license.

2. *Insights that may inform future interaction design research and practice* have been derived from the design, development, deployment and evaluation of the gesture authoring software. Principally, it has been found that the discretization of both spatial and temporal aspects of gestures contributes to an appropriate user interface paradigm for representing and manipulating gesture information.

5.3 Future Work

The research described in this thesis instigates a number of opportunities for future work. These opportunities can be examined in two broad categories. First, Hotspotizer can be expanded and improved by addressing a number of technical and design challenges that have surfaced during the research described in this thesis. Second, the space discretization paradigm for authoring gestures can be evaluated, refined and adapted for use with a wider variety of input devices.

5.3.1 Expanding Hotspotizer

One strand of future work may deal with expanding the expressive power of Hotspotizer by implementing new features in a user-friendly manner.

While it did not come up in the user studies, I find that the current visualization style may become convoluted as gesture collections grow in size. Exploring alternative ways of visualizing many gestures within one workspace is on our agenda for future versions of the software.

Currently, (as I discussed in Section 3.2) Hotspotizer does not directly support "online" (REF) — i.e. continuous — gesturing, since it adopts a traditional event-based model for detecting and responding to gestures. As such, support for *manipulative* and *deictic* gestures, which are common across gesture-based user interfaces, is severely limited. As Myers, Hudson, and Pausch (2000) recommend, ideally, the "continuous nature of the input [should] be preserved." This, however, requires "tighter integration with application logic" (Hartmann, Abdulla, et al., 2007) through interfacing with a textual programming language or third-party applications integrating support for continuous input streams. Unfortunately, the first option oversteps the scope of the Hotspotizer project (See Section 1.3). The second option can be explored for a limited set of third-party applications.

Among other features are negative hotspots that mark space that should not be engaged when gesturing (i.e. negation (Hoste and Signer, 2014)), a movable frame of reference for the workspace to enable gesturing around peripheral body parts, resizable hotspot boundaries, adjustable timeout, compositions that involve multiple limbs, and recognition of hand movements. As implied by user studies, the capability to infer hotspots from *demonstration*, and *speech recognition* to control the application from a distance are features that may further accelerate user workflows (Figure5.2).

Incorporating classifier-coupled gesture recognition (Hoste, Rooms, and Signer, 2013) could serve to alleviate recognizer errors (Myers, Hudson, and Pausch, 2000), and, when needed, to decouple overlapping gesture definitions.

Studies have revealed a tendency among users to initially overconstrain gesture designs in terms of spatial restrictions; i.e. users tend to begin with gesture designs that utilize too few hotspots (see Section 4.3). To steer away users from this tendency, the size of the cursor — or "brush" — for marking hotspots could be made adjustable; with the default size being set to mark a larger area and the finer brush made accessible as an option (Figure 5.1).

5.3.2 Space Discretization

A second strand of future work may focus on evaluating and refining the space discretization paradigm.

The usability and expressive power of the user interface paradigm is independent from its implementation. However, due to various factors that constrain its scope, this work does not evaluate the paradigm separately. In order to refine the user interface paradigm and support implementations in different contexts, user studies can be conducted to evaluate the difficulty of understanding and manipulating gestures visualized as hotspots in discretized space. Kin et al. (2012) have conducted a study that examines the speed and accuracy of users' understanding various representations for touch gestures. A similar study that examines representations of gross gestures may inform future work on gesture authoring and documentation.

The space discretization paradigm may also have value for authoring gestures enabled using technologies other than skeletal tracking. I encourage designers, developers, and researchers to adopt the paradigm for use in different contexts.

Bibliography

Alon, J. et al. (2009). "A Unified Framework for Gesture Recognition and Spatiotemporal Gesture Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.9 (cit. on p. 25).

Ashbrook, D. and T. Starner (2010). "MAGIC: A Motion Gesture Design Tool". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM (cit. on pp. 5, 15, 20, 21, 35).

Ballendat, T., N. Marquardt, and S. Greenberg (2010). "Proxemic Interaction: Designing for a Proximity and Orientation-aware Environment". In: *ACM International Conference on Interactive Tabletops and Surfaces*. ACM (cit. on p. 5).

Baytaş, M. A., Y. Yemez, and O. Özcan (2014a). "Hotspotizer: End-User Authoring of Mid-Air Gestural Interactions". In: *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational*. ACM (cit. on p. 45).

— (2014b). "User Interface Paradigms for Visually Authoring Mid-Air Gestures: A Survey and a Provocation ". In: *Proceedings of the Workshop on Engineering Gestures for Multimodal Interfaces*. Ed. by F. Echthler et al. Sun SITE Central Europe (CEUR) (cit. on p. 45).

Blackwell, A. F. (2004). "End-user Developers at Home". In: *Communications of the ACM* 47.9 (cit. on p. 11).

— (2006). "Psychological Issues in End-User Programming". In: *End User Development*. Ed. by H. Lieberman, F. Paternò, and V. Wulf. Vol. 9. Springer Netherlands, pp. 9–30 (cit. on p. 11).

Bolt, R. A. (1980). ""Put-that-there": Voice and Gesture at the Graphics Interface". In: *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques*. ACM (cit. on p. 9).

Boren, T. and J. Ramey (2000). "Thinking aloud: reconciling theory and practice". In: *IEEE Transactions on Professional Communication* 43.3 (cit. on p. 40).

Brooks, F. P. (1995). *The Mythical Man-Month: Essays On Software Engineering*. Pearson Education (cit. on p. 3).

Burnett, M., C. Cook, and G. Rothermel (2004). "End-user Software Engineering". In: *Communications of the ACM* 47.9 (cit. on p. 11).

Cairns, P. et al. (2014). "The Influence of Controllers on Immersion in Mobile Games". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM (cit. on p. 1).

Cao, X. and R. Balakrishnan (2003). "VisionWand: Interaction Techniques for Large Displays Using a Passive Wand Tracked in 3D". In: *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*. ACM (cit. on p. 10).

Christensen, C. and M. Raynor (2003). "Value Networks and the Impetus to Innovate". In: *The Innovator's Solution: Creating and Sustaining Successful Growth*. Harvard Business School Press. Chap. 3 (cit. on p. 36).

Crowley, J. L., J. Coutaz, and F. Bérard (2000). "Perceptual User Interfaces: Things That See". In: *Communications of the ACM* 43.3 (cit. on pp. 1, 5).

Dourish, P. (2004). *Where the Action Is: The Foundations of Embodied Interaction*. MIT press (cit. on p. 2).

Echtler, F. and A. Butz (2012). "GISpL: Gestures Made Easy". In: *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*. ACM (cit. on p. 19).

Edwards, W. (2010). *Motor Learning and Control: From Theory to Practice*. Cengage Learning (cit. on p. 5).

Eisenstein, J. and R. Davis (2006). "Visual and Linguistic Information in Gesture Classification". In: *ACM SIGGRAPH 2006 Courses*. ACM (cit. on p. 8).

Elmezain, M. et al. (2010). "Robust methods for hand gesture spotting and recognition using Hidden Markov Models and Conditional Random Fields". In: *Proceedings of the 2010 IEEE International Symposium on Signal Processing and Information Technology* (cit. on p. 25).

Fogtmann, M. H., J. Fritsch, and K. J. Kortbek (2008). "Kinesthetic Interaction: Revealing the Bodily Potential in Interaction Design". In: *Proceedings of the 20th Australasian Conference on Computer-Human Interaction: Designing for Habitus and Habitat*. ACM (cit. on p. 10).

Follmer, S. et al. (2013). "inFORM: Dynamic Physical Affordances and Constraints Through Shape and Object Actuation". In: *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*. ACM (cit. on p. 5).

Francese, R., I. Passero, and G. Tortora (2012). "Wiimote and Kinect: Gestural User Interfaces Add a Natural Third Dimension to HCI". In: *Proceedings of the International Working Conference on Advanced Visual Interfaces*. ACM (cit. on p. 1).

Franz, N. K. (2011). "The Unfocused Focus Group: Benefit or Bane?." In: *Qualitative Report* 16.5 (cit. on p. 36).

Frayling, C. (1993). "Research in Art and Design". In: *Royal College of Art Research Papers* 1.1 (cit. on pp. 4, 47).

Fried, J. (2008). *Why we disagree with Don Norman*. https://signalvnoise.com/posts/904-why-we-disagree-with-don-norman (cit. on p. 36).

Gallo, L. (2013). "A Study on the Degrees of Freedom in Touchless Interaction". In: *SIGGRAPH Asia 2013 Technical Briefs*. ACM (cit. on p. 1).

Gavrila, D. M. (1999). "The Visual Analysis of Human Movement: A Survey". In: *Computer Vision and Image Understanding* 73.1 (cit. on p. 1).

Germonprez, M., D. Hovorka, and F. Collopy (2007). "A Theory of Tailorable Technology Design". In: *Journal of the Association for Information Systems* 8.6 (cit. on p. 11).

Girshick, R. et al. (2011). "Efficient Regression of General-activity Human Poses from Depth Images". In: *Proceedings of the 2011 International Conference on Computer Vision*. IEEE Computer Society (cit. on p. 5).

Haibach, P. S., G. Reid, and D. H. Collier (2011). *Motor Learning and Development*. Human Kinetics (cit. on p. 5).

Hartmann, B., L. Abdulla, et al. (2007). "Authoring Sensor-based Interactions by Demonstration with Direct Manipulation and Pattern Recognition". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM (cit. on pp. 2, 15, 21, 26, 34, 48).

Hartmann, B., S. R. Klemmer, et al. (2006). "Reflective Physical Prototyping Through Integrated Design, Test, and Analysis". In: *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*. ACM (cit. on p. 2).

Holloway, S. and C. Julien (2010). "The Case for End-user Programming of Ubiquitous Computing Environments". In: *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*. ACM (cit. on p. 11).

Holzinger, A. (2005). "Usability Engineering Methods for Software Developers". In: *Communications of the ACM* 48.1 (cit. on p. 40).

Hoste, L., B. D. Rooms, and B. Signer (2013). “Declarative Gesture Spotting using Inferred and Refined Control Points”. In: *Proceedings of the 2nd International Conference on Pattern Recognition Applications and Methods* (cit. on pp. 17, 21, 25, 26, 48).

Hoste, L. and B. Signer (2014). “Criteria, Challenges and Opportunities for Gesture Programming Languages”. In: *Proceedings of the Workshop on Engineering Gestures for Multimodal Interfaces*. Sun SITE Central Europe (CEUR) (cit. on pp. 2, 15, 34, 48).

Huang, J.-D. (2011). “Kinerehab: A Kinect-based System for Physical Rehabilitation: A Pilot Study for Young Adults with Motor Disabilities”. In: *The Proceedings of the 13th International ACM SIGACCESS Conference on Computers and Accessibility*. ACM (cit. on p. 1).

Hutchins, E. L., J. D. Hollan, and D. A. Norman (1985). “Direct Manipulation Interfaces”. In: *Human Computer Interaction* 1.4 (cit. on p. 38).

Jaspers, M. W. et al. (2004). “The think aloud method: a guide to user interface design”. In: *International Journal of Medical Informatics* 73.11–12 (cit. on p. 40).

Julià, C. F., N. Earnshaw, and S. Jordà (2013). “GestureAgents: An Agent-based Framework for Concurrent Multi-task Multi-user Interaction”. In: *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*. ACM (cit. on p. 19).

Kammer, D. et al. (2010). “Towards a Formalization of Multi-touch Gestures”. In: *ACM International Conference on Interactive Tabletops and Surfaces*. ACM (cit. on p. 19).

Kang, H., C. W. Lee, and K. Jung (2004). “Recognition-based gesture spotting in video games”. In: *Pattern Recognition Letters* 25.15 (cit. on p. 25).

Kapur, A. et al. (2005). “Gesture-Based Affective Computing on Motion Capture Data”. In: *Proceedings of the First International Conference on Affective Computing and Intelligent Interaction*. Springer-Verlag (cit. on p. 5).

Karam, M. and m. c. schraefel (2005). *A Taxonomy of Gestures in Human Computer Interactions*. Technical Report. University of Southampton (cit. on p. 9).

Kela, J. et al. (2006). “Accelerometer-based Gesture Control for a Design Environment”. In: *Personal and Ubiquitous Computing* 10.5 (cit. on p. 5).

Kettebekov, S. (2004). “Exploiting Prosodic Structuring of Coverbal Gesticulation”. In: *Proceedings of the 6th International Conference on Multimodal Interfaces*. ACM (cit. on p. 8).

Khandkar, S. H. and F. Maurer (2010). “A Domain Specific Language to Define Gestures for Multi-touch Applications”. In: *Proceedings of the 10th Workshop on Domain-Specific Modeling*. ACM (cit. on p. 19).

Kim, J.-W. and T.-J. Nam (2013). “EventHurdle: Supporting Designers’ Exploratory Interaction Prototyping with Gesture-based Sensors”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM (cit. on pp. 2, 17, 21).

Kin, K. et al. (2012). “Proton++: A Customizable Declarative Multitouch Framework”. In: *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*. ACM (cit. on pp. 35, 48).

Kitzinger, J. (1995). “Qualitative research: introducing focus groups”. In: *British Medical Journal* 311.7000 (cit. on p. 36).

Klemmer, S. (2014). *Human-Computer Interaction*. Online course on Coursera. Retrieved 22 August 2014 from https://www.coursera.org/course/hciucsd (cit. on p. 35).

Ko, A. J., R. Abraham, et al. (2011). “The State of the Art in End-user Software Engineering”. In: *ACM Computing Surveys* 43.3 (cit. on pp. 2, 3, 11).

Ko, A. J., B. A. Myers, and H. H. Aung (2004). “Six Learning Barriers in End-User Programming Systems”. In: *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing*. IEEE Computer Society (cit. on p. 11).

Kopp, S., P. Tepper, and J. Cassell (2004). "Towards Integrated Microplanning of Language and Iconic Gesture for Multimodal Output". In: *Proceedings of the 6th International Conference on Multimodal Interfaces*. ACM (cit. on p. 10).

Krum, D. M. et al. (2002). "Speech and Gesture Multimodal Control of a Whole Earth 3D Visualization Environment". In: *Proceedings of the Symposium on Data Visualisation 2002*. Eurographics Association (cit. on p. 10).

Kurtenbach, G. and E. A. Hulteen (1990). "Gestures in Human-Computer Communication". In: *The Art of Human-Computer Interface Design*. Ed. by B. Laurel. Addison Wesley, pp. 309–317 (cit. on p. 4).

Lamsweerde, A. v. (2000). "Formal Specification: A Roadmap". In: *Proceedings of the Conference on The Future of Software Engineering*. ACM (cit. on p. 19).

Lange, B. et al. (2011). "Development and Evaluation of Low Cost Game-Based Balance Rehabilitation Tool Using the Microsoft Kinect Sensor". In: *Engineering in Medicine and Biology Society,EMBC, 2011 Annual International Conference of the IEEE* (cit. on p. 11).

Lee, H.-K. and J. Kim (1999). "An HMM-based threshold model approach for gesture recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21.10 (cit. on p. 25).

Lenman, S., L. Bretzner, and B. Thuresson (2002). "Using Marking Menus to Develop Command Sets for Computer Vision Based Hand Gesture Interfaces". In: *Proceedings of the Second Nordic Conference on Human-computer Interaction*. ACM (cit. on p. 10).

Lewis, C. and G. Olson (1987). "Empirical Studies of Programmers: Second Workshop". In: ed. by G. M. Olson, S. Sheppard, and E. Soloway. Ablex Publishing Corp. Chap. Can Principles of Cognition Lower the Barriers to Programming?, pp. 248–263 (cit. on p. 11).

Lim, Y.-K., E. Stolterman, and J. Tenenberg (2008). "The Anatomy of Prototypes: Prototypes As Filters, Prototypes As Manifestations of Design Ideas". In: *ACM Transactions on Computer-Human Interaction* 15.2 (cit. on p. 2).

Lindell, R. (2014). "Crafting Interaction: The Epistemology of Modern Programming". In: *Personal and Ubiquitous Computing* 18.3 (cit. on p. 2).

Liu, J. et al. (2009). "uWave: Accelerometer-based Personalized Gesture Recognition and Its Applications". In: *Pervasive and Mobile Computing* 5.6 (cit. on p. 5).

Long Jr., A. C., J. A. Landay, and L. A. Rowe (1999). "Implications for a Gesture Design Tool". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM (cit. on p. 35).

Lozano-Quilis, J. A. et al. (2013). "Virtual Reality System for Multiple Sclerosis Rehabilitation Using KINECT". In: *Proceedings of the 7th International Conference on Pervasive Computing Technologies for Healthcare*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (cit. on p. 1).

Lü, H. and Y. Li (2012). "Gesture Coder: A Tool for Programming Multi-touch Gestures by Demonstration". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM (cit. on pp. 1, 35).

— (2013). "Gesture Studio: Authoring Multi-touch Interactions Through Demonstration and Declaration". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM (cit. on pp. 4, 35).

Malgireddy, M. et al. (2010). "A Framework for Hand Gesture Recognition and Spotting Using Subgesture Modeling". In: *Proceedings of the 20th International Conference on Pattern Recognition* (cit. on p. 25).

McConnell, S. (2009). *Code Complete*. Microsoft Press (cit. on p. 3).

McNeill, D. (1992). *Hand and Mind: What Gestures Reveal about Thought*. University of Chicago Press (cit. on p. 8).

— (2008). *Gesture and Thought*. University of Chicago Press (cit. on pp. 8–10).

Moeslund, T. B. and E. Granum (2001). “A Survey of Computer Vision-Based Human Motion Capture”. In: *Computer Vision and Image Understanding* 81.3 (cit. on pp. 1, 8).

Moeslund, T. B., A. Hilton, and V. Krüger (2006). “A Survey of Advances in Vision-based Human Motion Capture and Analysis”. In: *Computer Vision and Image Understanding* 104.2 (cit. on pp. 1, 8).

Morgan, D. L. (1996). “Focus Groups”. In: *Annual Review of Sociology* 22.1996 (cit. on p. 36).

Myers, B. A., A. J. Ko, and M. M. Burnett (2006). “Invited Research Overview: End-user Programming”. In: *CHI '06 Extended Abstracts on Human Factors in Computing Systems*. ACM (cit. on pp. 10, 11).

Myers, B., S. E. Hudson, and R. Pausch (2000). “Past, Present, and Future of User Interface Software Tools”. In: *ACM Transactions on Computer-Human Interaction* 7.1 (cit. on pp. 2, 14, 48).

Nielsen, J. (1993a). “Noncommand User Interfaces”. In: *Communications of the ACM* 36.4 (cit. on p. 5).

— (1993b). *Usability Engineering*. Morgan Kaufmann (cit. on p. 40).

— (1997). *The Use and Misuse of Focus Groups*. http://www.nngroup.com/articles/focus-groups/ (cit. on pp. 35, 36).

— (2000). *Why You Only Need to Test with 5 Users*. http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/ (cit. on p. 44).

— (2011). *Parallel & Iterative Design Competitive Testing H̄igh Usability*. http://www.nngroup.com/articles/parallel-and-iterative-design/ (cit. on p. 44).

Nielsen, J. and T. K. Landauer (1993). “A Mathematical Model of the Finding of Usability Problems”. In: *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*. ACM (cit. on p. 44).

Norman, D. A. (1986). “Cognitive Engineering”. In: *User Centered System Design*. Ed. by D. A. Norman and S. W. Draper. CRC Press. Chap. 3, pp. 31–61 (cit. on pp. 2, 12).

— (1993). *Things that make us smart: Defending human attributes in the age of the machine*. Basic Books (cit. on p. 13).

— (2002). *The Design of Everyday Things*. Basic Books (cit. on pp. 2, 12).

Olsen Jr., D. R. (2007). “Evaluating User Interface Systems Research”. In: *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*. ACM (cit. on pp. 12, 13, 38, 44, 46).

Pane, J. F., B. A. Myers, and C. A. Ratanamahatana (2001). “Studying the Language and Structure in Non-programmers' Solutions to Programming Problems”. In: *International Journal of Human-Computer Studies* 54.2 (cit. on p. 11).

Paternò, F. (2013). “End User Development: Survey of an Emerging Field for Empowering People”. In: *ISRN Software Engineering* 2013 (cit. on p. 11).

Pea, R. D. and D. M. Kurland (1987). “Mirrors of Minds: Patterns of Experience in Educational Computing”. In: ed. by R. D. Pea and K. Sheingold. Ablex Publishing Corp. Chap. On the Cognitive Effects of Learning Computer Programming, pp. 147–177 (cit. on p. 11).

Porta, M. (2002). “Vision-based user interfaces: methods and applications”. In: *International Journal of Human-Computer Studies* 57.1 (cit. on p. 1).

Quek, F. K. H. (1996). “Unencumbered Gestural Interaction”. In: *IEEE MultiMedia* 3.4 (cit. on p. 8).

Quek, F. et al. (2002). “Multimodal Human Discourse: Gesture and Speech”. In: *ACM Transactions on Computer-Human Interaction* 9.3 (cit. on pp. 8, 9).

Reis, B. et al. (2012). “Increasing Kinect Application Development Productivity by an Enhanced Hardware Abstraction”. In: *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. ACM (cit. on p. 35).

Rizzo, A. S. et al. (2011). "Virtual Reality and Interactive Digital Game Technology: New Tools to Address Obesity and Diabetes". In: *Journal of Diabetes Science and Technology* 5.2 (cit. on p. 11).

Rose, R. (1992). "Discriminant wordspotting techniques for rejecting non-vocabulary utterances in unconstrained speech". In: *Proceedings of the 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 2 (cit. on p. 25).

Scaffidi, C., M. Shaw, and B. Myers (2005). "Estimating the Numbers of End Users and End User Programmers". In: *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE Computer Society (cit. on p. 11).

Scholliers, C. et al. (2011). "Midas: A Declarative Multi-touch Interaction Framework". In: *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*. ACM (cit. on p. 19).

Schön, D. A. (1984). *The Reflective Practitioner: How Professionals Think in Action*. Basic Books (cit. on p. 2).

Shoemaker, G. et al. (2010). "Whole Body Large Wall Display Interfaces". In: *CHI '10 Extended Abstracts on Human Factors in Computing Systems*. ACM (cit. on pp. 13, 24, 30, 38).

Shotton, J., A. Fitzgibbon, et al. (2011). "Real-time Human Pose Recognition in Parts from Single Depth Images". In: *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society (cit. on p. 5).

Shotton, J. (2012). "Conditional Regression Forests for Human Pose Estimation". In: *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society (cit. on p. 5).

Shotton, J., R. Girshick, et al. (2013). "Efficient Human Pose Estimation from Single Depth Images". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.12 (cit. on p. 5).

Silva, V. H. Z. and H. H. A. Arriaga (2003). "Evaluation of a Visual Interface for Gesticulation Recognition". In: *Proceedings of the Latin American Conference on Human-computer Interaction*. ACM (cit. on p. 10).

Simmons, S. et al. (2013). "Prescription Software for Recovery and Rehabilitation Using Microsoft Kinect". In: *Proceedings of the 7th International Conference on Pervasive Computing Technologies for Healthcare*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (cit. on p. 1).

Smithson, J. (2000). "Using and analysing focus groups: limitations and possibilities". In: *International Journal of Social Research Methodology* 3.2 (cit. on p. 36).

Sommerville, I. (2010). "Dependability and security specification". In: *Software Engineering (9th Edition)*. Addison Wesley, pp. 309–340 (cit. on p. 19).

Spano, L. D. et al. (2013). "GestIT: A Declarative and Compositional Framework for Multiplatform Gesture Definition". In: *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. ACM (cit. on p. 19).

Suma, E. A. et al. (2013). "Adapting User Interfaces for Gestural Interaction with the Flexible Action and Articulated Skeleton Toolkit". In: *Computers & Graphics* 37.3 (cit. on pp. 19–21).

Tang, J. K. T. and T. Igarashi (2013). "CUBOD: A Customized Body Gesture Design Tool for End Users". In: *Proceedings of the 27th International BCS Human Computer Interaction Conference*. British Computer Society (cit. on pp. 1, 18).

Turk, M. and G. Robertson (2000). "Perceptual User Interfaces". In: *Communications of the ACM* 43.3 (cit. on pp. 1, 5, 26).

Walter, R., G. Bailly, and J. Müller (2013). "StrikeAPose: Revealing Mid-air Gestures on Public Displays". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM (cit. on p. 4).

Warren, K. et al. (2013). "Bending the Rules: Bend Gesture Classification for Flexible Displays". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM (cit. on p. 4).

Wen, R. et al. (2013). "In Situ Spatial AR Surgical Planning Using Projector-Kinect System". In: *Proceedings of the Fourth Symposium on Information and Communication Technology*. ACM (cit. on p. 1).

Wexelblat, A. (1995). "An Approach to Natural Gesture in Virtual Environments". In: *ACM Transactions on Computer-Human Interaction* 2.3 (cit. on p. 10).

— (1998). "Research Challenges in Gesture: Open Issues and Unsolved Problems". In: *Proceedings of the International Gesture Workshop on Gesture and Sign Language in Human-Computer Interaction*. Springer-Verlag (cit. on p. 8).

Wilson, A. D. (2012). "Sensor- and Recognition-Based Input for Interaction". In: *The Human-Computer Interaction Handbook*. Ed. by J. A. Jacko. CRC Press. Chap. 7, pp. 133–156 (cit. on p. 38).

Wilson, A. and S. Shafer (2003). "XWand: UI for Intelligent Spaces". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM (cit. on p. 10).

Wulf, V. and M. Jarke (2004). "The Economics of End-user Development". In: *Communications of the ACM* 47.9 (cit. on p. 11).

Zamborlin, B. et al. (2014). "Fluid Gesture Interaction Design: Applications of Continuous Recognition for the Design of Modern Gestural Interfaces". In: *ACM Transactions on Interactive Intelligent Systems* 3.4 (cit. on pp. 16, 20, 21, 28).

Zhang, H. and Y. Li (2014). "GestKeyboard: Enabling Gesture-based Interaction on Ordinary Physical Keyboard". In: *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*. ACM (cit. on p. 4).

Zimmerman, J., J. Forlizzi, and S. Evenson (2007). "Research Through Design As a Method for Interaction Design Research in HCI". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM (cit. on pp. 4, 46, 47).

Appendix A

Attributions

- Figure 1.1 by James Pfaff / CC BY-SA 3.0
- Figure 1.2 by Sergey Galyonkin / CC BY-SA 2.0
- Figure 3.1 contains an image retrieved from Microsoft's Kinect for Windows Human Interface Guidelines v1.8.0
- Figure 4.3 contains graphic design work by A. Ayça Ünlüer

Appendix B

Hotspotizer: End-User Authoring of Mid-Air Gestural Interactions

This chapter reproduces the following publication:

Mehmet Aydın Baytaş, Yücel Yemez, and Oğuzhan Özcan. 2014. Hotspotizer: End-User Authoring of Mid-Air Gestural Interactions. In Proceedings of the 8th Nordic Conference on Human-Computer Interaction (NordiCHI '14).

Hotspotizer: End-User Authoring of Mid-Air Gestural Interactions

Mehmet Aydın Baytaş
Design Lab
Koç University, 34450 Istanbul
mbaytas@ku.edu.tr

Yücel Yemez
Department of Computer Engineering
Koç University, 34450 Istanbul
yyemez@ku.edu.tr

Oğuzhan Özcan
Design Lab
Koç University, 34450 Istanbul
oozcan@ku.edu.tr

ABSTRACT
Drawing from a user-centered design process and guidelines derived from the literature, we developed a paradigm based on space discretization for declaratively authoring mid-air gestures and implemented it in Hotspotizer, an end-to-end toolkit for mapping custom gestures to keyboard commands. Our implementation empowers diverse user populations – including end-users without domain expertise – to develop custom gestural interfaces within minutes, for use with arbitrary applications.

Author Keywords
Hotspotizer; gestural interaction; gesture authoring; visual programming; end-user development; interface prototyping.

ACM Classification Keywords
D.2.2 Software Engineering: Design Tools and Techniques; H.5.2 Information Interfaces and Presentation (e.g. HCI): User Interfaces

INTRODUCTION

Input devices that sense mid-air gestures through depth imaging and skeletal tracking have become widely available in the recent years. Diverse populations of users – software developers, interaction designers, artists [14], students [27], hobbyists [17], researchers and end-users at home [19] – are served by the possibilities offered by full-body mid-air gestural interaction. This diversity has led one such device, the Microsoft Kinect, to claim a world record in 2011 as the "fastest selling consumer electronics device" [29]. Despite diverse and numerous users and the popularity of the hardware, there are few commercial software products that leverage mid-air gestures outside of gaming, and sales of Kinect-based games have not caught up with the success of the device [6]. This trend has been linked to design and user experience issues with gesture-controlled software; issues which stem from difficulties in design and development [24]. Chief among these difficulties is that for both adept programmers and comparatively non-technical users, the domain-specific knowledge, time and effort required to implement custom gestural interactions is prohibitive.

NordiCHI '14, October 26 - 30 2014, Helsinki, Finland
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2542-4/14/10 $15.00.
http://dx.doi.org/10.1145/2639189.2639255

The diverse populations of users come with a variety of situations in which they may want to create custom gestural interfaces. These include rapid prototyping of gesture-based interactive applications [8], developing interactive art and performances [21], experiential learning [27], gaming [20] and adapting non-gestural interfaces for gesture control [25]. Although user populations and their aims in using gesture sensing may vary, implementing custom gestural interfaces boils down to two key tasks: (1) *visualizing, creating and manipulating gesture information* and (2) *mapping the recognition of the desired gestures to events in the computer*.

These two tasks constitute what we refer to as *authoring* gestures. Depending on the aims and skills of a particular user in a particular situation, there will naturally be other tasks involved in developing a custom gestural interface. However, gesture authoring is the key activity that is common across users developing gestural interfaces in a variety of situations. Supporting gesture authoring enables many user populations to use custom gestural interfaces for many purposes.

In the absence of authoring tools, gestural interface development is accomplished using textual programming tools provided by vendors of gesture-sensing hardware[1,2], and third parties[3]. Textual programming creates a significant gulf of execution – a chasm between user goals and actions taken within a system to achieve those goals [15] – for end users, since it introduces additional tasks such as setting up a programming environment. Moreover, for both end-users and professional programmers, textual programming embodies a significant gulf of evaluation - a gap between a system's output and the users's expectations and intentions [15] - during gesture authoring, since it does not allow for rapid testing for whether the authored gesture specification conforms to the design that the user has in mind.

This need identified above prompts the following research question: *How can we support authoring mid-air gestures for end-users with a software tool?* We began with this broad question and refined our vision for the software tool through

[1]microsoft.com/en-us/kinectforwindowsdev
[2]softkinetic.com
[3]kinecttoolbox.codeplex.com

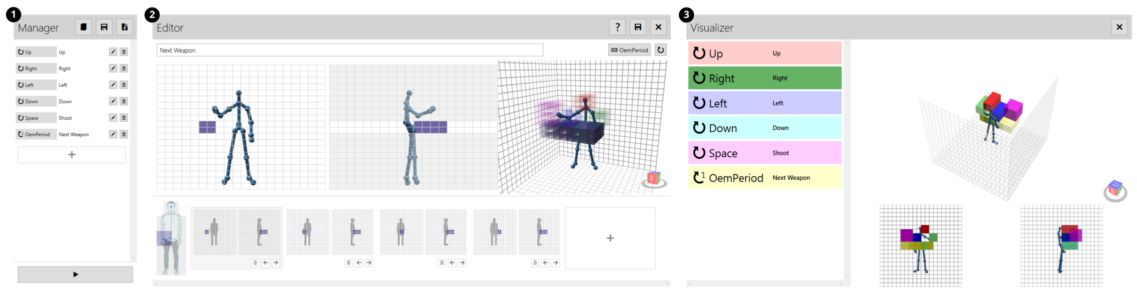


Figure 1. Hotspotizer consists of three modules: (1) The Manager lists all of the gestures in the current collection and allows creating, saving and loading collections as well as adding, removing and editing individual gestures. (2) The Editor is the main workspace where gestures are authored. (3) The Visualizer provides interactive feedback on available and recognized gestures.

reviewing previous work on user interface design tools and conducting formative studies with potential users. As a result, we developed *Hotspotizer*: an end-to-end, *plug-and-play* application for declaratively authoring custom gestures using a purely graphical interface and mapping them to commands issued from an emulated keyboard. (See Figure 1 for screenshots from the application.)

For visualizing and manipulating gesture information, we designed and implemented a user interface paradigm based on *space discretization*. We designed the interface to be as simple as possible to use while supporting a diverse set of gestures. The paradigm renders the space around the user's body as a 3-dimensional array of discrete cubic cells, which can be marked as hotspots that register human movement when a tracked limb passes through them (*hotspotized*). Using this paradigm, a wide variety of gestures can be represented as sets of hotspots around the user's body. Via a timeline of keyframes, dynamic movements can be authored in this manner. Our implementation leverages the user's sense of proprioception by defining the array of cells relative to the user's center of gravity rather than the gesture sensor.

Hotspotizer is available online as a free download[4], released under the MIT License[5].

DESIGN AND EVALUATION OF AUTHORING TOOLS

Olsen argues that user interface design tools, particularly those that deal with unconventional interaction techniques (such as mid-air gesture sensing), do not lend themselves to conventional software evaluation methods [18]. One reason for this is that such tools require domain-specific expertise, which – by the nature of novel tools – no user population possesses. Another reason is that these tools support complex tasks with high inter-user variability in terms of the users' mental models of the tasks. "Meaningful comparisons between two tools for a realistically complex problem are confounded in so many ways as to make statistical comparisons more fantasy than fact." [18] We derived the following four guidelines for the design of a gesture authoring tool from the framework proposed by Olsen for the evaluation of user interface toolkits:

[4]designlab.ku.edu.tr/design-thinking-research-group/hotspotizer
[5]opensource.org/licenses/MIT

(1) *Reduce development time.* A good authoring tool should be flexible to allow for the rapid implementation of design changes. This can be encouraged reducing the number of choices that have to be made to express a design.

(2) *Encapsulate and simplify expertise.* Considerable technical know-how is required to design and develop applications for emerging technologies. A good design tool liberates the designer from the need for prior knowledge, yet communicates the capabilities and limitations of the technology to nudge the designer towards feasible designs.

(3) *Lower skill barriers.* Empowering new populations of users to envision and implement designs "expands the set of people who can effectively create new applications." [18]

(4) *Make use of a common infrastructure.* It is difficult to get users to adopt a new standard. As much as possible, authoring tools should hook up to existing and widely adopted tools and practices, and complement existing workflows; upgrading rather than negating the common denominator.

Employing a paradigm for expressing design choices that reflects the problem being solved and embodies the constraints of the design space [16] serves all four the guidelines above.

Shoemaker, Tsukitani, Kitamura and Booth propose design guidelines for body-centric interaction with large displays [23]. Two of these generalize to influence the design of an authoring tool for mid-air gestures.

The first is that **(5)** gestural interaction at a distance should be *"mediated through a representation that binds personal and extrapersonal space."* A means for communicating the constraints and opportunities of the interaction space to the user is recommended for mid-air gestural interfaces. This holds for design tools that target these interactions.

Second, Shoemaker et al. recommend that **(6)** the *users' sense of proprioception be leveraged* by allowing some operations to be performed in the user's personal space, without requiring visual feedback. In terms of authoring interactions, this guideline calls for encouraging gesture designs that capitalize on proprioception through the nature of the authoring paradigm.

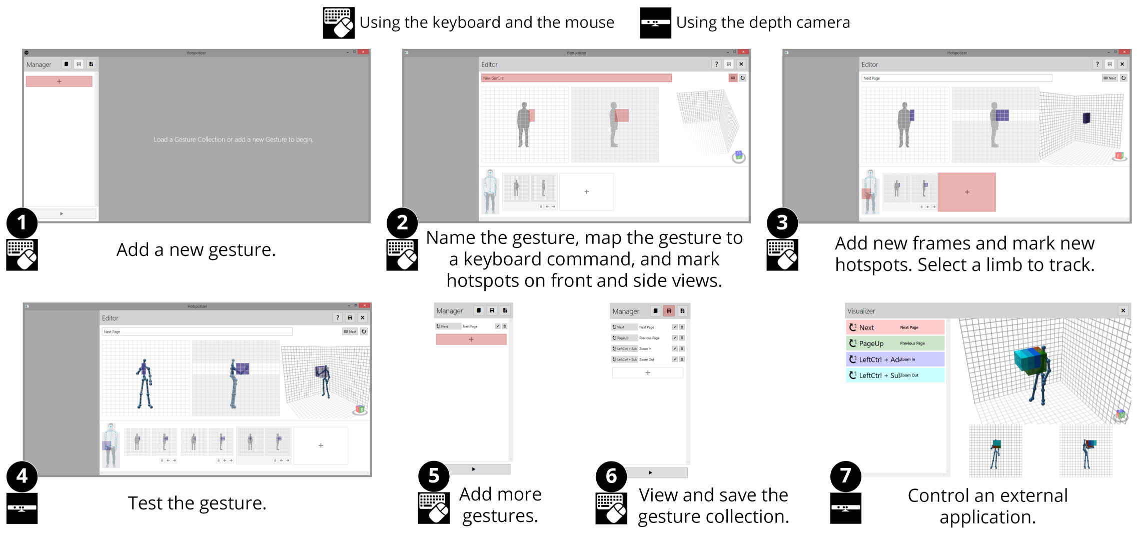

Using the keyboard and the mouse
Using the depth camera
1
Add a new gesture.
2
Name the gesture, map the gesture to a keyboard command, and mark hotspots on front and side views.
3
Add new frames and mark new hotspots. Select a limb to track.
4
Test the gesture.
5
Add more gestures.
6
View and save the gesture collection.
7
Control an external application.

Figure 2. The workflow of an end-user using Hotspotizer to adapt an application for gesture control.

These six guidelines, derived from previous work, form the basis of our design rationale for the Hotspotizer interface.

RELATED WORK

The design of Hotspotizer has been motivated by research on end-user development, aspiring to build "systems that are *easy to develop*" [11] for users who undertake software development not as an end in itself but as a means towards a goal in some other domain [10]. Hotspotizer's features have been influenced by prior research on gesture authoring tools. Historically, gesture authoring by end-users has been studied in mobile contexts, for applications targeting tablets and phones equipped with touchscreens [9, 12, 13, 22] and inertial sensors [1, 3]. More recently, researchers have explored a variety of methods for authoring mid-air gestures on devices based on inertial sensors and depth sensors. Broadly, these methods come in two flavors: Authoring gestures by *declaration* involves the use of a high-level syntax to describe gesture information. Authoring gestures by *demonstration* is done by recording one or more examples and employing machine learning techniques to train a recognizer. These two techniques interact with a variety of user interface paradigms to visualize and manipulate mid-air gesture information [2]. From this body of work, below, we discuss four approaches to gesture authoring that have influenced the features of Hotspotizer.

For declaratively authoring mid-air gestures for skeletal tracking, the *Flexible Action and Articulated Skeleton Toolkit (FAAST)* [25] provides atomic *action primitives* that can be used to compose rules in plain English such as "right hand above right shoulder by at least 20 cm." These constraints specify the position of, the speed of, or the angle between limbs, as well as general body orientation. FAAST controls other applications on the computer via mapping gestures to keyboard and mouse events. While describing gestures using atomic rules affords significant expressive power, this representation does not embody a visualization of the constraints embedded in the design space and thus may not serve to bridge the gulf of execution that obstructs end-users [16]. We go beyond FAAST in terms of visually representing the design space to accelerate authoring, encapsulate expertise and further enable non-expert users.

EventHurdle [8] leverages a visual markup language based on *hurdles* that define gesture trajectories for the declarative authoring of gestures for touchscreens, for inertial sensors and in mid-air. While the *hurdle* paradigm is versatile and effective as a representation of the design space of surface gestures, it has not been demonstrated to be particularly effective in describing fully 3-dimensional trajectories. The tool is implemented as a plug-in for Adobe Flash[6], which poses a barrier for users who have not invested in the software. Hotspotizer builds on the idea of using a purely visual syntax for authoring gestures.

A collection of commercial[7,8] and research [26] efforts implement *demonstration* for authoring skeletal tracking gestures. While demonstration seems to be straightforward solution, it requires the temporal segmentation of intended gestures from intermediate movements to be done manually, often by editing on a timeline of keyframes. For end-users without familiarity with machine learning concepts, the task of composing good training samples is not trivial. Moreover, this method cannot be used if the depth sensing device is not available during development (e.g. due to malfunction or

[6]adobe.com/products/flash
[7]gesturepak.com
[8]gesturestudio.ca

devices being shared between users). Thus, the current version of Hotspotizer does not implement demonstration as a method to author gestures. Yet, influenced by demonstration-based tools, Hotspotizer uses a timeline of keyframes to represent dynamic movements, extending the idea into the context of declarative authoring.

Finally, Hoste, De Rooms and Signer [4] describe an approach for representing gesture information using spatial boundaries around sequenced *control points*. The paradigm has been designed primarily for human readability and manipulability. The sizes and placement of these boundaries can be adjusted to vary the *strictness* of gesture paths, and programming constructs like negation or additional temporal constraints can be introduced for rich expressions. An implementation of this representation for gesture authoring is not available and its use for 3D gestures is not documented. The paradigm implemented in Hotspotizer is functionally similar to the idea of using control points to capture gesture trajectories. We develop the idea into a concrete user interface by adopting cubic boundaries for control points in discretized 3D space.

USING HOTSPOTIZER

To describe how mid-air gestures can be specified and mapped to keyboard events using Hotspotizer, we will consider the case of an end-user, Ali, who would like to adapt a document viewing application for gesture control. (Ali may require this functionality in settings where touching an input device to navigate inside a document is undesirable; e.g. when performing surgery on a patient or repairs on oily engine parts.) Figure 2 depicts this workflow, and the numbers in parentheses throughout this section relate to the numbered panes in the figure.

Ali wants to be able to cycle up and down between the pages of a document, as well as zoom in and out, using mid-air gestures. These actions may correspond to different keyboard commands depending on the document viewing application; we will assume that, respectively, the *Page Up*, *Page Down* keys and the *Ctrl + Plus* and *Ctrl + Minus* key combinations are used. To cycle between pages, the left hand will be swiped in air as if turning the pages of a real, albeit large book. To zoom in and out, the right hand will perform beckoning and pushing motions. Figure 3 shows one way of describing these two gestures in terms of hotspots (for brevity, the *page up* and *zoom out* gestures are not shown).

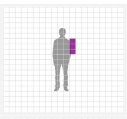
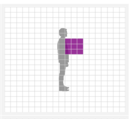
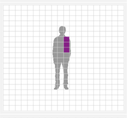
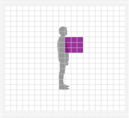
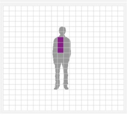
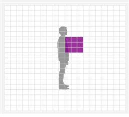
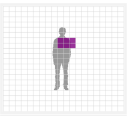
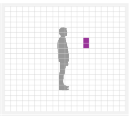
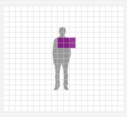
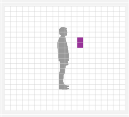
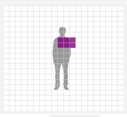
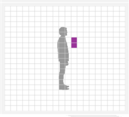
Function	Hotspots			Limb	Command
	Frame	Front View	Side View		
Next Page	1			Left Hand	*Page Down* (once)
	2				
	3				
Zoom In	1			Right Hand	*Ctrl + Plus* (once)
	2				
	3				

Figure 3. *Next page* and *zoom in* gestures to control a document viewing application and the keyboard functionality that they map to. The diagrams show, respectively, hotspot arrangements for a swipe gesture and a beckoning motion.

Creating and Editing Gestures

Hotspotizer greets Ali with the Manager module containing empty gesture collection upon launch. Ali creates a new gesture in the collection, launching the Editor module **(1)**. Here, Ali assigns a name for the gesture for easy recall, specifies the Page Down key to be triggered when the gesture is performed, and confirms that the *loop* toggle button is not checked – otherwise performing the gesture and holding the tracked limb over the hotspots in the last frame continues to hold down the keyboard key assigned to the gesture **(2)**.

Ali moves on to the main workspace where they use the front- and side-views over a representation of the tracked user to mark the positions of the hotspots for the first frame. Initially, all of the cells in the side view are disabled and grayed out. Marking cells on the front view enables the corresponding rows on the side view, whereupon Ali can mark the vertical and depth-wise position of their hotspots. Once hotspots are specified in all three dimensions through these two grids, they appear on the 3D viewport on the right.

Once Ali completes marking the first frame's hotspots, they proceed to add another frame using the button next to the timeline below, and then another **(3)**. Finally, after marking the second and third frames' hotspots, Ali selects the left hand as the limb that will be used in performing this gesture.

After saving the first gesture into the collection, Ali is taken back to the Manager module where they can add the remaining gestures and see the existing gestures to review, edit or delete them **(5)**. Once they are satisfied with the gesture collection they created, Ali can save the collection into a file for later use **(6)**.

Testing Designs and Controlling External Applications

At any time when using the Editor module, if they have a Kinect sensor connected to the computer, Ali may step in front of the sensor and see a rendering of the skeletal model of their body on the front, side and 3D viewports **(4)**. This feature can be used to rapidly test and tune hotspot locations at design time.

Testing over the whole gesture collection is available through the Visualizer module **(7)**. This module depicts a list of the gestures in the current collection and all of their hotspots on 3D, front and side viewports. Each gesture is shown in a different color. On the 3D viewport, transparency implies the order of hotspots. Hotspots glow when the tracked limb enters them in the correct order.

The Visualizer module also embeds the keyboard simulator. Launching the visualizer attaches a virtual keyboard to the system, which relays associated key events upon the successful performance of gestures. The visualization and the emulator continue to work when Hotspotizer is not in focus or is minimized.

SPACE DISCRETIZATION

Hotspotizer implements a paradigm based on space discretization for visualizing and manipulating gesture information. In the current implementation, we partition the space around the skeletal model tracked by the Kinect sensor into cubes that are 15cm on each side.

The total workspace is a large cube that is 3m on each side. While this is much larger than both the horizontal and vertical reach of many people; this is by design, to accommodates unusually tall users. The centroid of the cube that comprises the workspace is affixed to the "hip center" joint returned by the Kinect sensor. By specifying and tracking joint movements relative to the user's skeletal model rather than the sensor's position in real space, we aimed to leverage the user's sense of proprioception [23] in gesturing.

To describe gestures, the cubic cells within the workspace may be marked to become hotspots – or *hotspotized* – that register when a specified joint passes through them. Joints available for tracking are the hands, feet, elbows, knees and the head. Hotspotizing is accomplished by using front and side views in the Editor workspace. The front view is used to specify the horizontal and vertical positions of the hotspots. The side view is used to confirm the vertical and specify depth-wise positions. The design of this interaction style was inspired by architectural and engineering drawings.

In order to enable the authoring of dynamic movements along with static poses, we split movements into discrete keyframes. A timeline in the Editor module shows the keyframes and allows adding, removing, reordering and editing actions. Hotspots within subsequent frames do not need to be adjacent, but the frames need to be traversed in the correct order and within a certain time limit for a gesture to be recognized. The inter-frame timeout in Hotspotizer is 500ms. If more than 500ms elapses between a tracked limb engaging hotspots of subsequent frames, the gesture is not recognized.

Gestures designed using space discretization are dependent on location, scale and orientation with respect to the workspace, which is affixed to the user's hip or center of gravity. However, the paradigm affords a degree of spatial flexibility; hotspotizing a larger volume of cells allows for relaxed gesture boundaries.

This paradigm itself supports a versatile array of features. The size of hotspots could be made adjustable, even adaptive; to allow for fine gesturing close to the user's body and more relaxed gesture boundaries at a distance. The total workspace volume could be made adjustable. The workspace could be defined in reference to limbs other than the center or in reference to the environment; supporting whole-body movements, a larger interaction space and rich proprioceptive interactions. The inter-frame timeout could be made adjustable to allow designs that exploit velocity and acceleration in gesturing. Hotspotizer does not implement these features. The design of the interface focuses on rapid development, simplification of expertise and lowering of skill barriers. Through pre-adjusted parameters for space discretization and timing, we reduce the complexity of the gesture authoring process and encapsulate the capabilities of the sensor. Future work may investigate empowering expert users with adjustability while maintaining the value added for non-experts

IMPLEMENTATION DETAILS

Hotspotizer was written in the C# programming language[9], using the Microsoft .NET Framework 4.5[10] and the Windows Presentation Foundation (WPF)[11] subsystem therein to render the user interface. We used the open source packages Windows Input Simulator[12] for keyboard emulation, Json.NET[13] for reading and writing gesture data to files and Helix 3D Toolkit[14] for 3D graphics. To run, Hotspotizer requires the Microsoft Kinect Runtime[15], and, if used with an Xbox Kinect sensor, the Kinect Software Development Kit (SDK)[16].

We took care to make the process of installing and running Hotspotizer as straightforward as possible, in order to accommodate diverse user populations. We packaged it as a Windows application that can be conventionally installed, uninstalled and launched from the Start Menu. Upon launch, Hotspotizer checks for its external requirements, the Kinect Runtime and SDK. If the requirements are unavailable, it prompts the user to install them, providing links to the web pages where they can be downloaded.

DESIGN AND EVALUATION

Hotspotizer has been developed through a user-centered design process, in order to fulfill the needs of diverse user populations. This section describes the evolution of Hotspotizer, the evaluation of the final prototype, and insights gained throughout development and deployment.

[9]msdn.microsoft.com/library/kx37x362
[10]microsoft.com/net
[11]msdn.microsoft.com/library/ms754130
[12]inputsimulator.codeplex.com
[13]json.codeplex.com
[14]helixtoolkit.codeplex.com
[15]microsoft.com/download/details.aspx?id=40277
[16]microsoft.com/download/details.aspx?id=40278

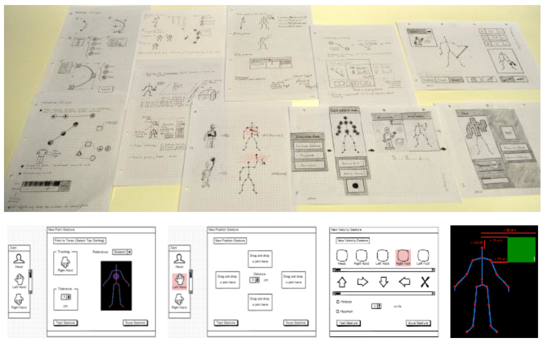


Figure 4. Rough sketches, paper prototypes and mockups were used to gather feedback which directed design and development.

Formative Studies

In the early stages of our attempt at designing a tool to support authoring mid-air gestural interactions, the motivating question was *what* to build. Design efforts were guided largely by qualitative and semi-structured feedback from users and inspiration from related work. We produced many concepts in the form of rough sketches and paper prototypes. Concepts at this early stage included an end-to-end environment for creating gesture-controlled interactive movies that fused gesture authoring and content creation in one application; ready-made widgets that pre-implemented gesture control, to be plugged into existing development and design environments; and tools to overlay information (such as the distance between two specific joints) onto a visualization of a skeletal model, to complement textual programming.

The rough sketches, paper prototypes and mockups have been presented at a workshop to a group of 10 potential users, aged 22-31 (μ=26), from diverse backgrounds. While recruited from among students and staff of a single university and not representative of a wide demographic, the participants represented the target users of Hotspotizer well. Each had different skills and interests. Among them was an industrial designer, a semi-professional musician, an electronics engineer, a computer scientist, a museum studies student, an interaction designer, a psychologists and a legal consultant. After a presentation on current design tools for mid-air gestural interfaces and our concepts, we collected feedback and made note of new ideas. Although all of the users were self-reportedly familiar with mid-air gestural interaction in the context of gaming, none had any familiarity with existing tools for authoring custom interfaces. Discussions on possible applications for custom gesture control revealed that a modular approach that can interface with other applications is preferable to a full-blown content creation suite. Moreover, even among users engaged in design or programming activities, tools used for these purposes varied greatly. This illustrated the value of a standalone application rather than a tool that generates code in a specific programming language or plugs into a specific environment.

We prepared another round of sketches and prototypes, some of them higher fidelity, such as a mock screencast showing the use of various modules in a gesture authoring suite. The idea of creating virtual buttons or hotspots in the space around the user and using them to define gestures was depicted in the sketches, as well as an interactive mockup developed in Processing[17] (see Figure 4). Other ideas included an application that recognized static poses and a graphical language consisting of atomic primitives for composing gestures. We presented these at a second workshop with the same 10 participants. Here, the concept of space discretization was proposed by a participant, an interaction designer. Upon interacting with the mockup of an interface where free-form areas in space can be made into gesture-tracking hotspots, she commented that she often makes use of squared paper when sketching. Instead of defining free-form regions in space, why not divide space into squares and constrain hotspots to these squares? Further discussion with participants revealed that this paradigm is grasped more easily than composing with atomic actions or constraints, or even demonstration. Moreover, using a visualization of the skeletal model and the space around it allows direct manipulation [7]; encapsulates the limitations and prospects of the design space; capitalizes on proprioception; and mediates interaction through a tight feedback loop [28].

We developed Hotspotizer as an implementation of the space discretization paradigm yielded by these workshops. We observed the design guidelines derived from Olsen [18] and Shoemaker et al. [23]. The decision to map gestures to key press events from an emulated keyboard was grounded in the principle of building on an infrastructure that is common across users and situations.

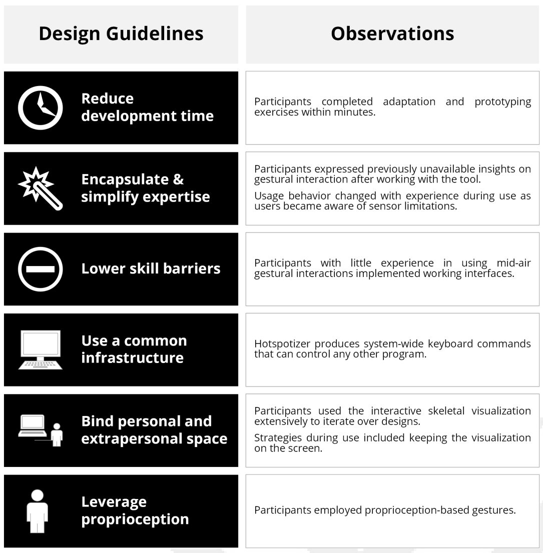
To evaluate Hotspotizer in use, we conducted two studies. The first was a study with 5 users to assess if Hotspotizer conforms to its design rationale. The second was a class workshop with 6 students working in pairs to build interactive prototypes of gestural interfaces. Qualitative results from these summative studies confirm that Hotspotizer conforms to our design rationale (see Figure 5 for a summary of the observations that relate to the design guidelines).

User Study

For the user study we recruited five graduate students: an industrial designer, a museum studies student, a computer scientist, a psychologist and an interaction designer. These were not the same people who participated in the previous workshops. Participants were given a pre-study questionnaire where, on average, they self-reported a low level of experience with computer programming (μ=2.1 on a 5-point Likert scale) and a low-medium level of experience with using mid-air gesture-based interfaces (μ=2.4).

Participants were given the task of adapting a non-gestural interface on computer game for gesture control. They were provided a PC with a Kinect sensor. The game was a side-scrolling platformer. We selected this style of game since we expected users to be fully familiar with the mechanics and not be distracted from the process of gesture authoring. We did not specify what gestures to use, but the game required

[17]processing.org



Design Guidelines	Observations
Reduce development time	Participants completed adaptation and prototyping exercises within minutes.
Encapsulate & simplify expertise	Participants expressed previously unavailable insights on gestural interaction after working with the tool. Usage behavior changed with experience during use as users became aware of sensor limitations.
Lower skill barriers	Participants with little experience in using mid-air gestural interactions implemented working interfaces.
Use a common infrastructure	Hotspotizer produces system-wide keyboard commands that can control any other program.
Bind personal and extrapersonal space	Participants used the interactive skeletal visualization extensively to iterate over designs. Strategies during use included keeping the visualization on the screen.
Leverage proprioception	Participants employed proprioception-based gestures.

Figure 5. Qualitative findings from two studies affirm that Hotspotizer is in keeping with our design rationale.

three commands to operate: *left* and *right* for movement, and a *jump* command. We required participants to play through and complete the first level of the game using gestures. We let participants finish the level using a keyboard and gave a demonstration of Hotspotizer before we had them design gestures.

All five participants were able to complete the assignment successfully, within 5-14 minutes (μ=7.4min) after being given the demonstration and left alone with the interface. Unanimously, the participants commented that the interface was *"easy to use"* and understandable. We observed that users iterated rapidly over gesture designs - for each gesture, participants went through 2-6 (μ=3) cycles of hotspotizing cells on the Editor and moving into the sensor's range to test designs. Static hand positions were preferred for the *left* and *right* commands, while the *jump* command inspired diverse gestures including kicking and nodding. A common error was that they marked areas outside the reach of the arms and the legs.

Semi-structured post-study interviews revealed that users had gained insights about the workings of skeletal tracking gestural interfaces. Support for full-body postures such as jumping, along with compositions that involve multiple limbs and grab detection were reported to be desirable as additional features. This is in line with our vision for future work.

Class Workshop

We conducted a workshop with 6 graduate students taking a course titled "Design Thinking for Interactivity." Participants worked in groups of two, at the same time. They were given a 20-minute presentation on how the interface works; and tasked with creating interactive prototypes for three different systems (one per group), following a single given use case for each system. The three systems comprised interactive digital signage for a movie theater, a penalty kick game and a video jukebox for public use. Participants were to create the visual design for the system's screens in PowerPoint[18], and assign gestures to shortcut keys in PowerPoint to add interactivity. Each group was provided a Kinect sensor, a PC with Hotspotizer and PowerPoint installed, and a cheat sheet that exposed keyboard commands available in PowerPoint. A diverse set of interactions is possible in this manner, including moving between screens, starting and stopping video, adjusting the volume of the system, displaying versatile animations and automatically triggering timed behavior.

All three groups were able to complete the implementation of an interactive prototype, from scratch, within the 60 minutes allocated for the activity. On average, about one third of this time was spent ideating and sketching designs, one third on composing visuals in PowerPoint and one third on authoring gestures with Hotspotizer. The penalty kick game employed four gestures: kicking a ball towards the left, the right and the center; and making a large circle with the hand to restart. The digital signage prototype was controlled by six hand gestures that involved pointing, swiping, pushing and pulling. The video jukebox prototype was controlled by five gestures that comprised swipes and touching various parts of the head and the torso.

Participants had self-reported low levels of experience with computer programming and using mid-air gestural interfaces (μ=1.8 and μ=2 on a 5-point Likert scale, respectively). They expressed enjoyment from the process of creating interactivity and working with new interface technology. *"A few days ago I did not even know that [mid-air gesture control] was possible. Now I just made my own working design,"* commented one participant.

Initially, users did struggle to understand the workings of the skeletal tracking. Two groups attempted to use gestures with fine differences that the Kinect sensor may not distinguish from each other, such as touching the eye with one finger versus touching the nose. Through trial and error, participants revised their gesture designs to match the capabilities of the sensor.

We expected a limitation to the space discretization paradigm to surface: Hotspots configured for one user could be inappropriate for another user due to differences in body size. After the three groups completed their projects, we had them try out each other's implementations to see if this was the case. We observed that the only time when gestures from a new user were not recognized was in the case of the football game, where large foot movements were involved. Differences in leg size hindered gesture recognition across users. Tuning the gesture design to involve larger hotspot areas alleviated the problem. When using hand gestures, no issues were apparent.

We observed that when working in pairs rather than alone, users adopted a different strategy when editing gestures: A

[18]office.microsoft.com/en-us/powerpoint

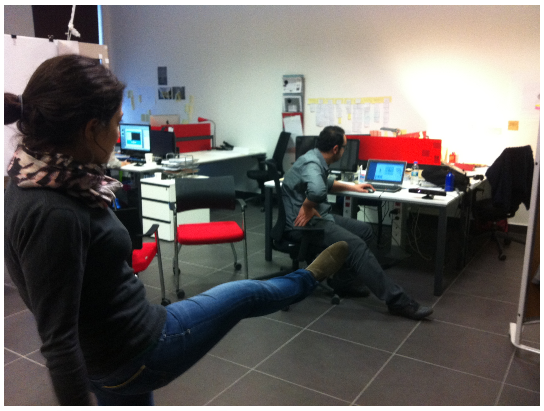


Figure 6. User strategies included working in pairs. One user performs gestures in front of the sensor while the other marks hotspots that correspond to limb positions.

single user would mark hotspots using the static on-screen silhouette of a human body as a reference and then test using the interactive representation. Working in pairs, one of the users preferred to stand in front of the sensor and perform gestures, while the other watched the interactive representation on the screen and used it as a reference when marking hotspots (Figure 6). To allow a single user to enjoy the advantages of using the interactive skeletal model for authoring, future work can implement the ability to infer hotspots from demonstration, along with voice control to interact with the program from a distance. We interviewed participants after the study, where they suggested that while editing, being able to see where hotspots belonging to previously authored gestures reside could be beneficial. This visualization was later added into the Editor module.

Generalizable Observations

During evaluation, we came across observations that are relevant for the design of mid-air gestural interfaces in general.

We noticed that users who self-reported little experience with mid-air gestural interfaces (a vast majority among participants) tended to be unaware of the limitations regarding the sensor's field of view. This manifested as an initial tendency to stand too close to the sensor and perform gestures in areas outside the sensor's field of view. Within minutes, users adjusted to become aware of the boundaries of the interaction area. To promote users' awareness of the depth sensor's field of view, the depth map provided by the sensor could be displayed on screen, as opposed to displaying the user's skeleton alone.

We observed that as they test and use their own gesture-controlled designs, users tend to keep Hotspotizer interface open and utilize the on-screen representation of the human skeleton. This confirms that our requirements for including a tight feedback loop and a representation for reporting the user's actions within space are justified. Based on this observation, we recommend interfaces based on mid-air gestures to include a representation of the tracked skeleton(s).

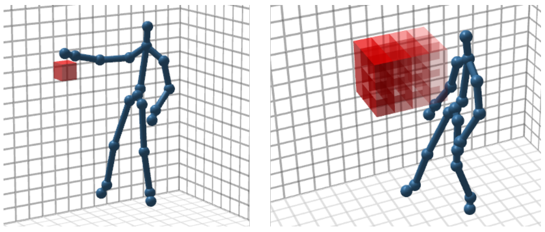


Figure 7. Initially, users preferred gesture designs that involved small hotspots and unspecified motion. Frames were added to constrain motion, and hotspots were enlarge to allow for variations during gesturing. Here, both panes depict hotspot configurations that may be used for a "punch" gesture. The configuration on the right is more conducive to robust recognition because of its sequentially constrained and spatially relaxed nature, compared to the rather extremely simplistic design on the left.

In general, when designing gestures, users preferred to start with static poses or specify only the end point of a gesture trajectory, utilizing only one frame to implement their designs. In simple cases, such as in controlling the side-scrolling platformer, these designs did suffice. However, as the quantity and complexity of gestures in the interface increases, this approach results in a high number *false positives* in gesture recognition due to intermediate movements intersecting hotspots. Users, due to inexperience, did not anticipate this. Through trial and error, gesture designs were revised and conflicts were resolved, by adding frames and authoring *movement* further constrain designs. Often, gesture designs resulted in *false negatives* due to spatially overconstrained designs that involved small volumes, requiring precise and accurate performance of gestures. Participants, through trial and error, revised their designs by enlarging hotspotized *volumes* to allow for some degree of ambiguity when performing gestures. The general tendency among users was to initially design gestures that were temporally or *sequentially underconstrained* and *spatially overconstrained*. Designs that minimize conflicts by *introducing sequential constraints* (i.e. more frames) while allowing for some flexibility by *relaxing spatial constraints* (i.e. more hotspots) were observed to be more conducive to robust recognition (see Figure 7).

DISCUSSION

The space discretization paradigm and its current implementation in Hotspotizer feature strengths and limitations that manifest as side effects of design choices.

One strength of the implementation is that gesture recognition is not influenced by the user's position and orientation within the sensor's field of view, provided that the depth image is not distorted and the sensor can build an accurate skeletal model of the user. Since the discretized workspace is affixed to the user's hip, hotspot locations are defined relative to the user's own body and the traversal of hotspots is detected properly as long as the skeletal model is built correctly. As a limitation of the depth sensor, skeletal modeling fails under certain conditions; e.g. the user turning their back to the sensor or engaging in contortions, the presence of objects that resemble a human form in the sensor's field of view, etc. Hotspotizer automatically hides the skeletal representation and halts gesture recognition when failures occur, and resumes operation when the sensor provides a skeletal model.

Certain limitations result from the design choice to prioritize leveraging a common infrastructure for end-users by mapping gestures to keyboard events. This obscures "association semantics" [3] (i.e. the same keyboard command may trigger different behaviors in different applications) and limits the expressive power of the gesture authoring paradigm. Hotspotizer currently does not support authoring continuous - or *online* [5] - gestures that affect some variable while they are being performed (as opposed to *offline* gestures that execute commands when the gesture is performed from the beginning to the end). This is not a limitation of the space discretization paradigm; since, theoretically, smaller portions of a gesture could be assigned to affect continuous variables (albeit in a quantized manner). Likewise, gestures involving pointing at or manipulating dynamic interface objects are not supported. This could be overcome by linking the discretized space model around the user with the virtual space of the user interface. However, these features require integration with a development environment, which is beyond the initial design goals. Exploring "tighter integration with application logic" [3] to empower software developers is a goal for future work.

CONCLUSION AND FUTURE WORK

We described our efforts in developing a *software tool for authoring mid-air gestures* to support the activities of diverse user populations. For this purpose, through guidelines derived from the literature and a user-centered design process, we developed a paradigm based on space discretization for visualizing and declaratively manipulating mid-air gesture information. We implemented this paradigm in Hotspotizer, a standalone Windows application that maps mid-air gestures to commands issued from an emulated keyboard. We evaluated Hotspotizer through a user study and class workshop.

Our findings from the evaluation sessions verify that Hotspotizer observes our design rationale and supports gesture authoring for end-users. We observed that *gestural interactions were implemented within minutes by users who did not have the skills* to use textual programming tools. User strategies and design choices implied that users understood the *domain expertise* embedded in the interface and leveraged their *sense of personal space and proprioception* in interacting with the system. Hotspotizer was used to control other programs on a PC, making use of a *common infrastructure*.

While it did not come up in the user studies, we find that the current visualization style may become convoluted as gesture collections grow in size. Exploring alternative ways of visualizing many gestures within one workspace is on our agenda for future versions of the software.

Future work may deal with implementing features that enhance the capacity for gestural expression. Among these are negative hotspots that mark space that should not be engaged when gesturing (i.e. negation [5]), a movable frame of reference for the workspace to enable gesturing around peripheral body parts, resizable hotspot boundaries, adjustable timeout, compositions that involve multiple limbs, and recognition of hand movements. Incorporating classifier-coupled gesture recognition [4] could serve, when needed, to decouple overlapping gesture definitions. As implied by user studies, the capability to infer hotspots from *demonstration*, and *speech recognition* to control the application from a distance are features that may further accelerate user workflows.

The space discretization paradigm may have value for authoring gestures enabled using technologies other than skeletal tracking. We encourage other researchers to adopt the paradigm for use in different contexts.

ACKNOWLEDGEMENTS

The work presented in this paper is part of research supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK), project number 112E056.

We thank the anonymous participants in our study and workshops for their time and feedback.

REFERENCES

1. Ashbrook, D., and Starner, T. MAGIC: A Motion Gesture Design Tool. In *Proc. CHI 2010* (2010), 2159–2168.
2. Baytaş, M. A., Yemez, Y., and Özcan, O. User Interface Paradigms for Visually Authoring Mid-Air Gestures: A Survey and a Provocation . In *Proc. EGMI 2014* (2014).
3. Hartmann, B., Abdulla, L., Mittal, M., and Klemmer, S. R. Authoring Sensor-based Interactions by Demonstration with Direct Manipulation and Pattern Recognition. In *Proc. CHI 2007* (2007), 145–154.
4. Hoste, L., De Rooms, B., and Signer, B. Declarative Gesture Spotting Using Inferred and Refined Control Points. In *Proc. ICPRAM 2013* (2013), 144–150.
5. Hoste, L., and Signer, B. Criteria, Challenges and Opportunities for Gesture Programming Languages. In *Proc. EGMI 2014* (2014).
6. Hughes, D. Microsoft Kinect shifts 10 million units, game sales remain poor. `http://www.huliq.com/10177/microsoft-kinect-shifts-10-million-units-game-sales-remain-poor`, 2011. Accessed: 2014-07-08.
7. Hutchins, E. L., Hollan, J. D., and Norman, D. A. Direct Manipulation Interfaces. *Human Computer Interaction 1*, 4 (1985), 311–338.
8. Kim, J.-W., and Nam, T.-J. EventHurdle: Supporting Designers' Exploratory Interaction Prototyping with Gesture-based Sensors. In *Proc. CHI 2013* (2013), 267–276.
9. Kin, K., Hartmann, B., DeRose, T., and Agrawala, M. Proton++: A Customizable Declarative Multitouch Framework. In *Proc. UIST 2012* (2012), 477–486.

10. Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M. B., Rothermel, G., Shaw, M., and Wiedenbeck, S. The State of the Art in End-user Software Engineering. *ACM Computing Surveys 43*, 3 (2011), 21:1–21:44.

11. Lieberman, H., Paternò, F., Klann, M., and Wulf, V. End-User Development: An Emerging Paradigm. In *End User Development*, H. Lieberman, F. Paternò, and V. Wulf, Eds., vol. 9. Springer Netherlands, 2006, 1–8.

12. Long, A. C., Landay, J. A., and Rowe, L. A. "Those Look Similar!" Issues in Automating Gesture Design Advice. In *Proc. PUI 2001* (2001), 1–5.

13. Lü, H., and Li, Y. Gesture Studio: Authoring Multi-touch Interactions Through Demonstration and Declaration. In *Proc. CHI 2013* (2013), 257–266.

14. Marquardt, Z., Beira, J. a., Em, N., Paiva, I., and Kox, S. Super Mirror: A Kinect Interface for Ballet Dancers. In *Proc. CHI 2012* (2012), 1619–1624.

15. Norman, D. A. Cognitive Engineering. In *User Centered System Design*, D. A. Norman and S. W. Draper, Eds. CRC Press, 1896, ch. 3, 31–61.

16. Norman, D. A. *Things that make us smart: Defending human attributes in the age of the machine*. Basic Books, 1993.

17. Oliver, A., Kang, S., Wünsche, B. C., and MacDonald, B. Using the Kinect As a Navigation Sensor for Mobile Robotics. In *Proc. IVCNZ 2012* (2012), 509–514.

18. Olsen, Jr., D. R. Evaluating User Interface Systems Research. In *Proc. UIST 2007* (2007), 251–258.

19. Panger, G. Kinect in the Kitchen: Testing Depth Camera Interactions in Practical Home Environments. In *CHI EA 2012* (2012), 1985–1990.

20. Raghuraman, S., Venkatraman, K., Wang, Z., Wu, J., Clements, J., Lotfian, R., Prabhakaran, B., Guo, X., Jafari, R., and Nahrstedt, K. Immersive Multiplayer Tennis with Microsoft Kinect and Body Sensor Networks. In *Proc. MM 2012* (2012), 1481–1484.

21. Rodrigues, D. G., Grenader, E., Nos, F. d. S., Dall'Agnol, M. d. S., Hansen, T. E., and Weibel, N. MotionDraw: A Tool for Enhancing Art and Performance Using Kinect. In *CHI EA 2013* (2013), 1197–1202.

22. Rubine, D. Specifying Gestures by Example. In *Proc. SIGGRAPH 1991* (1991), 329–337.

23. Shoemaker, G., Tsukitani, T., Kitamura, Y., and Booth, K. S. Body-Centric Interaction Techniques for Very Large Wall Displays. In *Proc. NordiCHI 2010* (2010), 463–472.

24. Stein, S. Kinect, 2011: Where art thou, motion? `http://www.cnet.com/news/kinect-2011-where-art-thou-motion/`, 2011. Accessed: 2014-07-08.

25. Suma, E. A., Krum, D. M., Lange, B., Koenig, S., Rizzo, A., and Bolas, M. Adapting user interfaces for gestural interaction with the flexible action and articulated skeleton toolkit. *Computers & Graphics 37*, 3 (2013), 193 – 201.

26. Tang, J. K. T., and Igarashi, T. CUBOD: A Customized Body Gesture Design Tool for End Users. In *Proc. BCS-HCI 2013* (2013), 5:1–5:10.

27. Villaroman, N., Rowe, D., and Swan, B. Teaching Natural User Interaction Using OpenNI and the Microsoft Kinect Sensor. In *Proc. SIGITE 2011* (2011), 227–232.

28. Wilson, A. D. Sensor- and Recognition-Based Input for Interaction. In *The Human-Computer Interaction Handbook*, J. A. Jacko, Ed. CRC Press, 2012, ch. 7, 133–156.

29. Yin, S. Microsoft Kinect Holds World Record for 'Fastest-Selling' Device. `http://www.pcmag.com/article2/0,2817,2381724,00.asp`, 2011. Accessed: 2014-07-08.

Appendix C

User Interface Paradigms for Visually Authoring Mid-Air Gestures: A Survey and a Provocation

This chapter reproduces the following publication:

Mehmet Aydın Baytaş, Yücel Yemez, and Oğuzhan Özcan. 2014. User Interface Paradigms for Visually Authoring Mid-air Gestures: A Survey and a Provocation. In Proceedings of the Workshop on Engineering Gestures for Multimodal Interfaces (EGMI 2014).

User Interface Paradigms for Visually Authoring Mid-Air Gestures: A Survey and a Provocation

Mehmet Aydın Baytaş[1], Yücel Yemez[2], Oğuzhan Özcan[1]
[1]Design Lab
Koç University, 34450 İstanbul
[2]Department of Computer Engineering
Koç University, 34450 İstanbul
{mbaytas, yyemez, oozcan}@ku.edu.tr

ABSTRACT
Gesture authoring tools enable the rapid and experiential prototyping of gesture-based interfaces. We survey visual authoring tools for mid-air gestures and identify three paradigms used for representing and manipulating gesture information: graphs, visual markup languages and timelines. We examine the strengths and limitations of these approaches and we propose a novel paradigm to authoring location-based mid-air gestures based on space discretization.

Author Keywords
Gestural interaction; gesture authoring; visual programming; interface prototyping.

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

INTRODUCTION

The recent proliferation of commercial input devices that can sense mid-air gestures, led by the introduction of the Nintendo Wii and the Microsoft Kinect, has enabled both professional developers and end-users to harness the power of full-body gestural interaction. However, despite the availability of the hardware, applications that leverage gestural interaction have not been thriving. A striking fact is that while the Kinect has broken records as the fastest-selling consumer electronics device in history, sales of games that utilize the Kinect have been poor [5]. This has been associated with design and user experience issues stemming from difficulties in designing and developing software [7]. Specifically, for both adept programmers and comparatively non-technical but creative users such as students, designers, artists and hobbyists, the amounts of time, effort and domain-specific knowledge required to implement custom gestural interactions is prohibitive.

Ongoing research aims to support gestural interaction design and development with gesture authoring tools. These tools aim at enabling rapid and experiential prototyping, which are essential practices for creating compelling designs [2]. However, few projects have gained widespread adoption. One issue that contributes to the low rate of adoption is the difficulty of balancing the trade-offs between complexity and expressive power of the paradigm used to represent and manipulate gesture information: Interfaces employed for gesture authoring may become convoluted and difficult to use in order to fully tap into the expressive power of human gesture; or they may omit useful features as they aim for usability and rapidity.

In this paper, we survey existing paradigms for visually authoring mid-air gestures and present a provocation, a novel gesture authoring paradigm, which we have implemented in the form of an end-to-end application for introducing gesture control to existing software and novel prototypes.

The rest of this paper is organized as follows: We first present three user interface paradigms – graphs, visual markup languages and timelines – used in current visual gesture authoring tools. Existing implementations of each paradigm are examined and discussed in terms of their capabilities and limitations. Results from evaluations with real users, if published, are emphasized. We then present a provocation in the form of a novel user interface paradigm for authoring mid-air gestures, based on space discretization and influenced by existing paradigms. We discuss future work and conclude by presenting a summary of our results.

PARADIGMS FOR AUTHORING MID-AIR GESTURES

Authoring tools for mid-air gestural interfaces are still in their infancy. Development tools provided by vendors of gesture-sensing input devices are focused on textual programming. Ongoing research suggests a set of diverse approaches to the problem of how to represent and manipulate three-dimensional gesture data. Existing works approach the issue in three ways that constitute distinct paradigms. These are:

1. using 2-dimensional *graphs* of the data from the sensors that detect movement;
2. using a *visual markup language*; and,
3. representing movement information using a *timeline* of frames.

These paradigms often interact with two programming approaches: Demonstration and declaration. Programming by demonstration enables developers to describe behavior by example. In the case of gestures, many examples of the

EGMI 2014, 1st International Workshop on Engineering Gestures for Multimodal Interfaces, June 17 2014, Rome, Italy.

same behavior are often provided in order to account for the differences in gesturing between users and over time. Declarative programming of gestures involves describing behavior using a high-level specification language. This specification language may be textual or graphical.

The paradigms we list above do not have to be used exclusively, and nor do demonstration and declarative programming. Aspects of different paradigms may find their place within the same authoring tool. A popular approach to authoring gestures is to introduce gestures by demonstration, convert gesture data into a visual representation, and then declaratively modify it

In this section, we describe the above approaches in detail, with examples from the literature. We comment on their strengths and weaknesses based on evaluations conducted with software that implement them.

Using Graphs of Movement Data

Visualizing and manipulating movement data using 2-dimensional graphs that represent low-level kinematic information is a popular approach for authoring mid-air gestures. This approach is often preferred when gesture detection is performed using inertial sensors such as accelerometers and gyroscopes. It also accommodates other sensors that read continuously variable data such as bending, light and pressure. Commonly the horizontal axis of the graph represents time while the vertical axis corresponds to the reading from the sensor. Often a "multi-waveform" occupies the graph, in order to represent data coming in from multiple axes of the sensor. Below, we study three software tools that implement graphs for representing gesture data: Exemplar, MAGIC and GIDE.

Exemplar

Exemplar [3] relies on demonstration to acquire gesture data and from a variety of sensors - accelerometers, switches, light sensors, bend sensors, pressure sensors and joysticks. Once a signal is acquired via demonstration, on the resulting graph, the developer marks the area of interest that corresponds to the desired gesture. The developer may interactively apply filters on the signal for offset, scaling, smoothing and first-order differentiation. (Figure 1) Exemplar offers two methods for recognition: One is pattern matching, where the developer introduces many examples of a gesture using the aforementioned method and new input is compared to the examples. The other is thresholding, where the developer manually introduces thresholds on the raw or filtered graph and gestures are recognized when motion data falls between the thresholds. This type of thresholding also supports hysteresis, where the developer introduces multiple thresholds that must be crossed for a gesture to be registered.

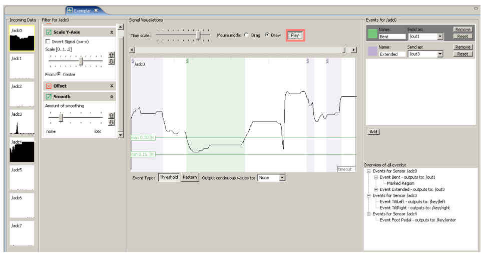


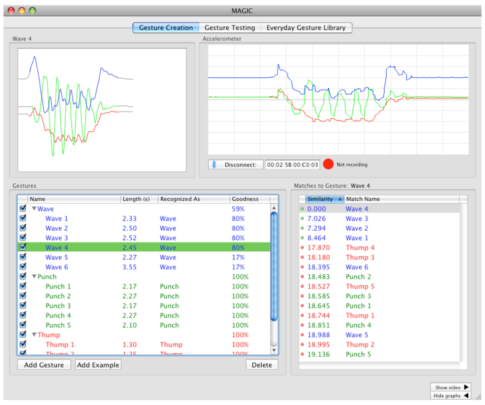
Figure 1: The Exemplar gesture authoring environment. [3] From left to right, the interface reflects the developer's workflow: Data from various sensors connected to the system is displayed as thumbnails and the sensor of interest is selected; filters are applied to the incoming signal; areas of interest are marked for pattern recognition or thresholds are set; and the resulting gesture is mapped to output events.

Exemplar's user studies suggest that this implementation of the paradigm is successful in increasing developer engagement with the workings and limitations of the sensors used. Possible areas of improvement include a technique to visualize multiple sensor visualizations and events and finer control over timing for pattern matching.

System for Multiple Action Gesture Interface Creation (MAGIC)

Ashbrook and Starner's MAGIC [1] is another tool that implements the 2-dimensional graphing paradigm. The focus of MAGIC is programming by demonstration. It supports the creation of training sets with multiple examples of the same gesture. It allows the developer to that keep track of the internal consistency of the provided training set; and check against conflicts with other gestures in the vocabulary and an "Everyday Gesture Library" of unintentional, automatic gestures that users perform during daily activities. MAGIC uses the graph paradigm only to visualize gesture data and does not support manipulation on the graph. (Figure 2)

One important feature in MAGIC is that the motion data graph may be augmented by a video of the gesture example being performed. Results from user studies indicate that this feature has been highly favored by users, during both gesture recording and retrospection. Interestingly, it is reported that the "least-used visualization" in MAGIC "was the recorded accelerometer graph;" with most users being "unable to connect the shape of the three lines" that correspond to the 3 axes of the accelerometer reading "to the arm and wrist movements that produced them." Features preferred by developers turned out to be the videos, "goodness" scores assigned to each gesture according to how they match gestures in and not in their own class, and a sorted list depicting the "distance" of a selected example to every other example.



	Name	Length (s)	Recognized As	Goodness
✓	Wave			59%
✓	Wave 1	2.33	Wave	80%
✓	Wave 2	2.50	Wave	80%
✓	Wave 3	2.52	Wave	80%
✓	Wave 4	2.45	Wave	80%
✓	Wave 5	2.27	Wave	17%
✓	Wave 6	3.55	Wave	17%
✓	Punch			100%
✓	Punch 1	2.17	Punch	100%
✓	Punch 2	2.27	Punch	100%
✓	Punch 3	2.17	Punch	100%
✓	Punch 4	2.27	Punch	100%
✓	Punch 5	2.10	Punch	100%
✓	Thump			100%
✓	Thump 1	1.30	Thump	100%
✓	Thump 2	1.25	Thump	100%

Matches to Gesture: Wave 4

Similarity	Match Name
0.000	Wave 4
7.026	Wave 3
7.294	Wave 2
8.464	Wave 1
17.870	Thump 4
18.180	Thump 3
18.395	Wave 6
18.483	Punch 2
18.527	Thump 5
18.585	Punch 3
18.645	Punch 1
18.744	Thump 1
18.851	Punch 4
18.988	Wave 5
18.995	Thump 2
19.136	Punch 5

Figure 2: MAGIC's gesture creation interface. [2]

Gesture Interaction Designer (GIDE)
More recently, GIDE [8] features an implementation of the graph paradigm for authoring accelerometer-based mid-air gestures. GIDE leverages a "modified" hidden Markov model approach to learn from a single example for each gesture in the vocabulary. The user interface implements two distinct features: (1) Each gesture in the vocabulary is housed in a "gesture editor" component which contains the sensor waveform, a video of the gesture being performed, an audio waveform recorded during the performance, and other information related to the gesture. (2) A "follow" mode allows the developer to perform gestures and get real-time feedback on the system's estimate of which gesture is being performed (via transparency and color) and *where* they are within that gesture. (Figure 3) This feedback on the temporal position within a gesture is multimodal: The sensor multi-waveform, the video and the audio waveform from the video are aligned and follow the gestural input. GIDE also supports "batch testing" by recording a continuous performance of multiple gestures and running it against the whole vocabulary to check if the correct gestures are recognized at the correct times.

User studies on GIDE reveal that the combination of multi-waveform, video and audio was useful in making sense of gesture data. Video was favored particularly since it allows developers to still remember the gestures they recorded after an extended period of not working on the gesture vocabulary. Another finding from the user studies was the suggestion that the "batch testing" feature where the developer records a continuous flow of many gestures to test against could be leveraged as a design strategy – gestures could be extracted from a recorded performance of continuous movement.

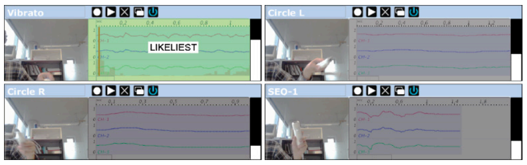


Figure 3: The "follow" mode in the GIDE interface. [8]

Discussion
Graphs that display acceleration data seem to be the standard paradigm for representing mid-air gestures tracked using acceleration sensors. This paradigm supports direct manipulation for segmenting and filtering gesture data, but manipulating acceleration data directly to modify gestures is unwieldy. User studies show that graphs depicting accelerometer (multi-)waveforms are not effective as the sole representation of a gesture, but work well as a component within a multimodal representation along with video.

Visual Markup Languages
Using a visual markup language for authoring gestures can allow for rich expression and may accommodate a wide variety of gesture-tracking devices, e.g. accelerometers and skeletal tracking, at the same time. The syntax of these visual markup languages can be of varying degrees of complexity, but depending on the sensor(s) used for gesture detection, making use of the capabilities of the hardware may not require a very detailed syntax. In this section we examine a software tool, EventHurdle, that implements a visual markup language for gesture authoring; and we discuss a gesture spotting approach based on control points which has not been implemented as a gesture authoring tool, but provides valuable insight.

EventHurdle
Kim and Nam describe a declarative hurdle-driven visual gesture markup language implemented in the EventHurdle authoring tool [6]. The EventHurdle syntax supports gesture input from single-camera-based, physical sensor-based and touch-based gesture input. In lieu of a timeline or graph, EventHurdle projects gesture trajectory onto a 2-dimensional workspace. The developer may perform the gestures, see the resulting trajectory on the workspace, and declaratively author gestures on the workspace by placing "hurdles" that intersect the gesture trajectory. Hurdles may be placed in ways that result in serial, parallel and/or recursive compositions. (Figure 4) "False hurdles" are available for specifying unwanted trajectories. While an intuitive way to visualize movement data from pointing devices, touch gestures and blob detection; this approach does not support the full range of expression inherent in 3-dimensional mid-air gesturing.

10

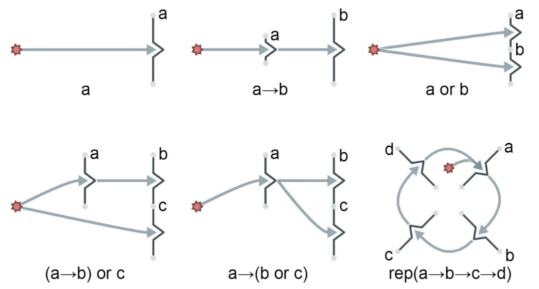


Figure 4: EventHurdle's visual markup language allows for a variety of compositions: (from top left) a simple gesture with one hurdle; serial and parallel compositions; combinations of serial and parallel compositions; recursive gesturing. [6]

Gestures defined in EventHurdle are configurable to be location-sensitive or location-invariant. By design, orientation- and scale-invariance are not implemented in order to avoid unnecessary technical options that may distract from "design thinking."

User studies on EventHurdle comment that the concept of hurdles and paths is "easily understood" and it "supports advanced programming of gesture recognition." Other than this, supporting features, rather than the strengths and weaknesses of the paradigm or comparison with other paradigms, have been the focus of user studies.

Control Points

Hoste, De Rooms and Signer describe a versatile and promising approach that uses spatiotemporal constraints around control points to describe gesture trajectories [4]. While the focus of the approach is on gesture spotting (i.e. segmentation of a continuous trajectory into discrete gestures) and not gesture authoring, they do propose a human-readable and manipulable external representation. (Figure 5) This external representation has significant expressive power and support for programming constructs such as negation (for declaring unwanted trajectories) and user-defined temporal constraints. While the authors' approach is to infer control points for a desired gesture from an example, the representation they propose also enables the manual placement of control points.

The authors do not describe an implementation that has been subjected to user studies. However, they discuss a number of concepts that add to the expressive power of using control points as a visual markup language to represent and manipulate gesture information. The first is that it is possible to add temporal constraints to the markup; i.e. a floor or ceiling value can be specified for the time taken by the tracked limb or device to travel between control points. This is demonstrated not on the graphical markup (which can be done easily), but on textual code generated to describe a gesture – another valuable feature. The second such concept is that the control points are surrounded by boundaries whose size can be adjusted to introduce spatial flexibility and accommodate "noisy" gestures. Third, boundaries can be set for negation when the variation in the gesture trajectory is too much. The authors discuss linear or planar negation boundaries only, but introducing negative control points into the syntax could also be explored. Finally, a "coupled recognition process" is introduced, where a trained classifier can be called to distinguish between potentially conflicting gestures; e.g. a circle and a rectangle that share the same control points.

One limitation of this approach is the lack of support for scale invariance. One way of introducing scale invariance may be to automatically scale boundary sizes and temporal constraints with the distance between control points. However, it is likely that the relationship between optimal values for these variables is nonlinear, which could make automatic scaling infeasible.

Discussion

The expressive power and usability of a visual markup language may vary drastically depending on the specifics of the language and the implementation. The general advantage of this paradigm is that it is suitable for describing and manipulating location-based gesture information (rather than acceleration-based information commonly depicted using graphs). This makes using a visual markup language suitable for mid-air gestures detected by depth-sensing cameras, where the interaction space is fixed and the limbs of the users move in relation to each other. Either the motion sensing device or certain parts of the skeletal model could be used to define a reference frame and gesture trajectories could be authored in a location-based manner using a visual markup language.

Timelines

Timelines of frames are commonly used in video editing applications. They often consist of a series of ordered thumbnails and/or markers that represent the content of the moving picture and any editing done on it, such as adding transitions.

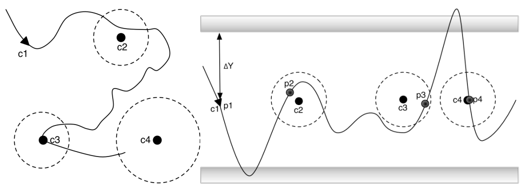


Figure 5: Using control points to represent gestures [4]. **(Left) A "noisy" gesture still gets picked up due to relaxed boundaries around control points. (Right) Negation is introduced via vertical boundaries so that large movements in the vertical axis are distinguished from the desired gesture.**

System	UI Paradigm	Programming Approach	Insights from user studies
Exemplar [3]	Graphs	Demonstration	Increases engagement with sensor workings and limitations.
MAGIC [1]	Graphs (multi-waveform)	Demonstration	Users unable to connect waveform to physical movements. Optional video is favored.
GIDE [8]	Graphs (multi-waveform with video)	Demonstration	Multimodal representation helps make sense of gesture data.
EventHurdle [6]	Visual markup language	Declaration	Easily understood. Supports "advanced" programming.
Control Points [4]	Visual markup language	Declaration / Demonstration	Not implemented.
Gesture Studio [1]	Timeline	Demonstration	Not published.

Table 1: Summary of studies on systems that exemplify three user interface paradigms for visually authoring mid-air gestures.

Gesture Studio

One application that implements a timeline to visualize gesture information is the commercial Gesture Studio.[1] The application works only with sensors that detect gestures through skeletal tracking using an infrared depth camera. Developers introduce gestures in Gesture Studio by demonstration, through performing and recording examples. The timeline is used to display thumbnails for each frame of the skeleton information coming from the depth sensor. The timeline is updated after the developer finishes recording a gesture, while during recording a rendering of the skeletal model tracked by the depth sensor provides feedback. After recording, the developer may remove unwanted frames from the timeline to trim gesture data for segmentation. Reordering frames is not supported since gestures are captured at a high frame rate (depending on the sensor, usually around 30 frames per second), which would make manual frame-by-frame editing inconvenient. The process through which these features have been selected is opaque, since there are no published studies that present the design process or evaluate Gesture Studio in use.

Discussion

In gesture authoring interfaces, timelines make sense when gesture tracking encompasses many limbs and dynamic movements that span more than a few seconds. Spatial and temporal concerns for gestures in 2 dimensions, such as those performed on surfaces, can be represented on the same workspace. The representation of mid-air gestures requires an additional component such as a timeline to show the change over time.

Discussion

We have presented a number of systems that exemplify three user interface paradigms for visually authoring mid-air gestures for computing applications (see Table 1 for a summary). For sensor-based gesturing, the standard paradigm used to represent gesture information appears to be projecting the sensor waveforms onto a graph. Graphs appear to work well as components that represent sensor-based gestures, allow experimentation with filters and gesture recognition methods, and support direct manipulation to some extent. User studies show that while the graphs alone may not allow developers to fully grasp the connection between movements and the waveform [1], they have been deemed useful as part of a multimodal gesture representation [8]. Using hurdles as a visual markup language offers an intuitive and expressive medium for gesture authoring, but it is not able to depict fully 3-dimensional gestures. Using spherical control points may be more conducive to direct manipulation while still affording an expressive syntax, but no implementation of this paradigm exists for authoring mid-air gestures. Finally, timelines of frames may come in handy for visualizing dynamic gestures with many moving elements, such as in skeletal tracking; but used in this fashion they allow only visualization and not manipulation.

There are paradigms that allow for the authoring of sensor-based gestures both declaratively and through demonstration. For skeletal tracking interfaces, tools based on demonstration exist, but we have not come across visual declarative programming tools for skeletal tracking interfaces. In the next section, we propose a user interface paradigm for declaratively authoring mid-air gestures for skeletal tracking interfaces.

PROVOCATION: SPACE DISCRETIZATION AS A NOVEL PARADIGM FOR AUTHORING MID-AIR GESTURES

The paradigms that we surveyed above each have their strengths and weaknesses. We wish to propose a novel paradigm for declaratively authoring mid-air gestures, which we will call *space discretization.* This paradigm conceptually supports both declaration and demonstration as ways to introduce gestures, and direct manipulation to edit them. The paradigm is adaptable for sensor-based interactions and touch gestures. We will present a rendition aimed at authoring gestures for skeletal tracking interfaces.

[1] http://gesturestudio.ca/

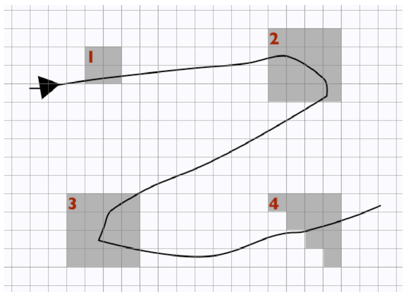


Figure 6: A 2-dimensional "Z" gesture defined using ordered hotspots in discretized space.

Overview and Implementation

We have implemented this paradigm as part of an application called *Hotspotizer*. The application has been developed as an end-to-end suite to facilitate rapid prototyping of gesture-based interactions and adapting arbitrary interface for gesture control. Collections of gestures can be created, saved, loaded, modified and mapped to a keyboard emulator within the application. The current version is configured to work with the Microsoft Kinect sensor and is available online as a free download.[2]

The paradigm we implemented works by partitioning the space around the tracked skeletal model into discrete spatial compartments. In a manner that is similar to the use of control points in Hoste, De Rooms and Signer's approach, these discrete compartments can be marked and activated to become "hotspots" that register movement when a tracked limb enters them. (Figure 6) Our approach may be likened to modifying the control points paradigm to use cubic instead of spherical boundaries and allow the placement of control points only at discrete locations in space. This is due to the difficulty of manipulating continuously moveable control points in 3 dimensions. Furthermore, using discrete hotspots instead of control points allows for the boundaries of the control points to be in custom shapes rather than spheres only. Considering the precision of current skeletal tracking devices, the difficulty of manipulating free-form regions rather than discrete compartments does not pay off.

In Hotspotizer, the compartments are cubes that measure 15 cm on each side and the workspace is a cube, 300 cm on each side, the centroid of which is fixed to the tracked skeleton's "hip center" joint returned by the Kinect sensor. (Figure 7) The workspace has been sized to accommodate larger users, and the compartments have been sized, through empirical observations, to reflect the sensor's precision. The alignment of the workspace to the user's body results in gestures being location-invariant with respect to the user's position relative to the depth camera.

[2] http://designlab.ku.edu.tr/design-thinking-research-group/hotspotizer/

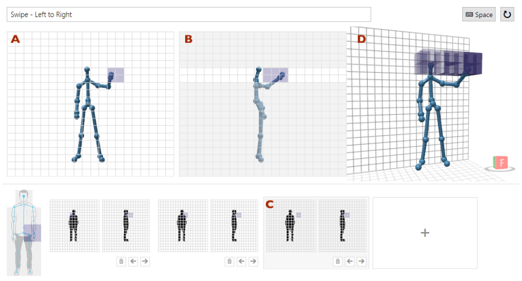


Figure 7: A 3-dimensional "swipe" gesture to be performed with the right hand, implemented in Hotspotizer. The front view (A) and the side view (B) depict the third frame, selected from the timeline (C). The 3D viewport (D) depicts all three frames, using transparency to imply the order.

However, gestures in Hotspotizer are always location-dependent with respect to the gesturing limb's position relative to the rest of the body. Scale- and orientation-invariance are not automatically supported, but it is possible to arrange hotspots in creative ways that allow the same gesture to be executed on different scales.

Splitting gesture data into frames, which are navigated using a timeline, supports authoring dynamic movements. The side view and front view grids only display hotspots that belong to one frame at a time, since placing all of the hotspots that belong to different frames of a gesture on the same grids results in a convoluted interface. During gesture tracking, if the tracked limb enters any one of the hotspots that belongs to a frame, the entire frame registers a "hit." For a gesture to be registered, its frames must be hit in the correct order and the time that elapses between subsequent frames registering a hit must not exceed a pre-defined timeout. Conceptually the timeout could be adjustable; in the current implementation, for the sake of a simple user interface, it is hard-coded to 500ms in Hotspotizer.

In essence, we propose a design for an expressive user interface paradigm for authoring mid-air gestures detected through skeletal tracking. Aspects of this design are based on the control points paradigm described in [4]. We modified the paradigm to confine the locations of the control points to discrete pre-defined locations and use cubic control point boundaries of fixed size, which can be added together to create custom shapes. We also introduce a timeline component so that spatial and temporal constraints can be manipulated unambiguously.

Future Work

Future work includes features to enrich the expressiveness of the paradigm and evaluating its performance in use.

The current implementation of the paradigm in Hotspotizer supports only declaration – manually specifying hotspots by selecting relevant areas on a grid. The interface may be extended to allow the introduction of gestures through demonstration, by inferring hotspots automatically from recorded gestures.

"Negative hotspots" to mark compartments that should not be crossed when gesturing are a possibility for future iterations on Hotspotizer. So is supporting gestures performed by multiple limbs; possibly by using a multi-track timeline and coupling keyframes where movements of the limbs should be synchronized.

In order to describe more complex gestures, it may make sense to introduce classifier-coupled gesture recognition. One shortage of the paradigm is that it does not accommodate the repeated usage of hotspots within different frames of a gesture well. If a gesture requires that a certain hotspot be hit twice, for example, the current implementation does not afford a way of detecting whether the first or the second hit is registered as a user performs the gesture.

Finally, as the precision of skeletal tracking devices increases and in order to accommodate devices that track smaller body parts such as the hands, adjustable workspace and compartment sizing may be introduced.

Formative evaluations have been conducted throughout the development Hotspotizer, focusing on prioritizing features and the visual design of the interface. Results of these, along with summative evaluations that compare the application to existing solutions and uncover user strategies for using the tool will be published in the future.

CONCLUSION

We reviewed existing paradigms for authoring mid-air gestures and discussed how graphs of sensor waveforms are suitable components that represent acceleration-based gesture data; how visual markup languages are better suited for location-based gesture data; and how timelines are used to communicate dynamic gesturing. We presented a novel gesture authoring paradigm for authoring mid-air gestures sensed by skeletal tracking: a visual markup language based on space discretization supported by a timeline to visualize temporal aspects of gesturing. Future work may build supporting features onto this paradigm and evaluate its performance in use by developers.

ACKNOWLEDGEMENT
The work presented in this paper is part of research supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK), project number 112E056.

REFERENCES

1. Ashbrook, D. and Starner, T. MAGIC: A Motion Gesture Design Tool. *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*, ACM Press (2010), 2159.
2. Buxton, B. *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann, Boston, 2007.
3. Hartmann, B., Abdulla, L., Mittal, M., and Klemmer, S.R. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07*, ACM Press (2007), 145.
4. Hoste, L., De Rooms, B., and Signer, B. Declarative Gesture Spotting Using Inferred and Refined Control Points. *Proceedings of the 2nd International Conference on Pattern Recognition Applications and Methods (ICPRAM 2013)*, (2013).
5. Hughes, D. Microsoft Kinect shifts 10 million units, game sales remain poor. *HULIQ*, 2012. http://www.huliq.com/10177/microsoft-kinect-shifts-10-million-units-game-sales-remain-poor.
6. Kim, J.-W. and Nam, T.-J. EventHurdle. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13*, ACM Press (2013), 267.
7. Stein, S. Kinect, 2011: Where art thou, motion? *CNET*, 2011. http://www.cnet.com/news/kinect-2011-where-art-thou-motion/.
8. Zamborlin, B., Bevilacqua, F., Gillies, M., and D'inverno, M. Fluid gesture interaction design. *ACM Transactions on Interactive Intelligent Systems 3*, 4 (2014), 1–30.

Appendix D

Rethinking Spherical Media Surfaces by Re-reading Ancient Greek Vases

This chapter reproduces the following publication:

Oğuzhan Özcan, Ayça Ünlüer, Mehmet Aydın Baytaş, and Barış Serim. 2012. Rethinking Spherical Media Surfaces by Re-reading Ancient Greek Vases. Paper presented at the workshop "Beyond Flat Displays: Towards Shaped and Deformable Interactive Surfaces," co-located with the ACM International Conference on Interactive Tabletops and Surfaces (ITS '12).

Rethinking Spherical Media Surfaces by Re-reading Ancient Greek Vases

Oğuzhan Özcan
Design Lab
Koç University
Rumelifeneri Road
34450, Istanbul, Turkey
oozcan@ku.edu.tr

Barış Serim
Design Lab
Koç University
Rumelifeneri Road
34450, Istanbul, Turkey
bserim@ku.edu.tr

Ayça Ünlüer
Dept. of Communication Design
Faculty of Art and Design
Yıldız Technical University
Istanbul, Turkey
ayca.unluer@gmail.com

Mehmet Aydın Baytaş
Design Lab
Koç University
Rumelifeneri Road
34450, Istanbul, Turkey
mbaytas@ku.edu.tr

Abstract

In this paper, we propose re-reading of past artifacts and traditions as a possible way to inspire the design of future media on non-flat displays. As an example, we illustrate how different narrative typologies found in ancient Greek vases, *circular story reading*, *bottom-up time reading*, *abstract and realistic contrast reading* and *reading in alignment,* can yield alternatives to interactive content specific to spherical media. We conclude by pointing out design considerations regarding the composition of graphic elements on spherical surfaces.

Author Keywords
Spherical displays; re-reading; story-telling.

ACM Classification Keywords
H.5.2 [Information Interfaces and Presentation]: User Interfaces.

Introduction

Research on non-flat displays has covered ample ground. Primitives such as the cylinder [2], the sphere [1] and bent surfaces [15]; and irrational forms [4] have been proposed as static media. Interactive displays that react to manipulations such as bending [8], rolling-out [6, 13] and folding [7, 9] have been considered. These works also discuss functionality and content for non-flat forms, but the discussions are often limited to the adaptations and eliminations of functions and content that already exists on conventional displays [1, 2]. We argue that this is not sufficient to fully explore the capabilities of non-flat display technology.

In order to discover new ways of using novel interactive media effectively, it is necessary to try unconventional approaches. One such approach is to attempt to re-read the solutions to similar problems that cultures in the past have come up with. Past cultures have been contemplating on variations of today's design problems for years. Today's media technology necessitates concepts and narratives be developed quickly, but past experiences, cultivated over the years, hold clues to enriching contemporary media experiences.

In our previous research, we have found evidence that supports our position: The traditions of shadow play [10], miniature painting [11] and calligraphy [14] and the artifacts of these traditions have offered clues as to what conventional displays are capable of. Similarly, by examining ancient cultural artifacts, we may anticipate what the mainstream applications of non-flat media will look like in the future and what forms of presentation they will afford.

We propose that re-reading the ways that imagery on ancient Greek vases have been composed may inspire content development for spherical interactive surfaces.

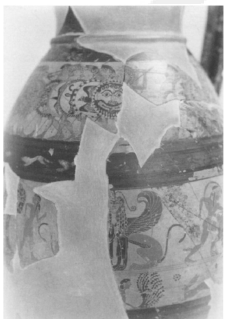


Figure 1: Detail from Chigi vase. [5]

Understanding "Re-reading the Past"
Being inspired by the past does not necessitate imitating it in new forms. Ideally, design solutions from the past should be re-evaluated from a contemporary perspective and they should inspire novel concepts, e.g. the Wayang Kulit shadow play where male and female viewers see the show from different sides of the screen may inspire the development of a two-sided display [10]. It is exciting that the user experience afforded by a two-sided display existed in the past, when the technology did not.

Similarly, re-reading miniature paintings allows us to discover concepts such as placing elements outside the picture's frame and representing concepts by juxtaposing its constituents [11], which lend us clues as to achieve narrative richness in current media.

Re-reading traditional Turkish calligraphy also yields surprises, in that the act of writing and drawing has been considered to be a multi-dimensional activity in the past. The traditional calligraphist completes his work, a continuous form, in one breath. He is surrounded by candlelight and music that helps him focus as he works. He desires to trace and imprint the forms that he imagines in air, but he may only confine himself to a two-dimensional medium [14]. Coupling the philosophy and rituals of traditional calligraphy with current technology would no doubt yield novel forms of calligraphy.

Ancient Greek Vases as Inspiration for Spherical Interactive Media
Four different types of narratives are observed in ancient Greek vases, where a multitude of themes are presented simultaneously:

Circular story reading (horizontal direction)
Sequential images, flowing horizontally on the vase, assume a linear storyboard formation. However, in

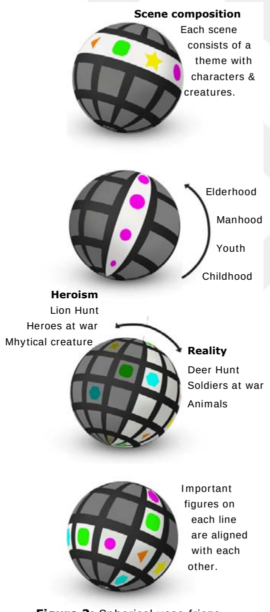


Figure 2: Spherical vase frieze reading regarding horizontal direction, vertical direction, hemispheres and alignment respectively.

each scene, certain characters are drawn facing left and others facing right. There is no clear and distinct direction and order to read in. The intent here is to motivate the reader to turn the vase as they view. Different stories emerge as the vase is revolves in the clockwise and counterclockwise directions. [5]

Bottom-up time reading (vertical direction)
Emotional and epic narratives ensue in vases of this character, summarizing on the life of a mythological character. The reading order is vertical, from the bottom up. The younger years of the character appear progresses upward, they are presented with later events in the character's life. Events are ordered vertically and they unfold in the horizontal direction.

Abstract and realistic contrast reading (hemispheric look)
One side of the vase depicts commoners and stories from daily life, and the other side comprises mythological characters and adventures.

Reading in alignment
Every vertical row deals with a different concept and presents the concept using mythological characters, creatures and objects. The elements that comprise the concept may look random, but they appear more realistic on one side and more heroic on the other. One row may depict a hunting scene, one side illustrating a deer hunt and the other side portraying a skirmish against a lion. As the reading progresses from bottom up, the subject matter of the rows are transformed in particular order. A vase may depict stages in the life of a man in this fashion: The bottommost row may embody children hunting, with an upper row illustrating youth entertaining themselves at horse races, while above there are scenes from a war where adults are fighting each other. Important characters, objects or creatures, the focal points of the rows, have been aligned as to maintain connections between rows.

Conclusion
Today's interactive media often presents nonlinear stories on a medium confined to a single flat display. For long this arrangement has allowed us to imagine interactive cinema and continues to be a functional and popular interface for interactive games [3]. Many works deal with the continuation of the user experience afforded by such media into non-flat media. However, the content and narratives specific to non-flat displays have not been covered extensively in the literature. One paper proposes an icosahedron shape to exploit the interactive potential of multifaceted displays and gestural interaction with the object to navigate the user interface. This, however, is an experiment in interaction styles and proposes no narrative form [12].

The typological analysis above suggests that there is no linear horizontal or vertical order to reading these ancient Greek vases. This intends to motivate the reader to rotate the vase in his hand.

It is possible to exploit the four typologies outlined above ("Circular Reading", "Bottom-up Time Reading", "Hemispheric Look" and "Reading in Alignment") to create interactive storytelling concepts specific to spherical media. Previous analyses on miniature painting indicate that it is possible to use mapping techniques for storytelling [11]. Beyond these planar mapping experiences, using these four typologies, it is possible to compose with stills and animated images on a spherical matrix. Care should be taken to maintain the viewer's attention on the medium and the following questions should be considered:

- will the matrix comprise stills or animated images only, or a combination of the two?
- if stills and animated images are to be combined, what should their ratio be?
- within the stills and the animated images, how will the pictures be composed in relation to the spherical matrix?

Utilizing re-reading methods, it may be possible to develop novel narrative forms and content that fully exploits contemporary technology, such as spherical displays. These new media forms may then become tools for novel gaming or learning experiences.

References

[1] Benko, H., Wilson, A. D., and Balakrishnan, R. Sphere: multi-touch interactions on a spherical display. In Proc. UIST 2008, ACM Press (2008), 77-86.

[2] Beyer, G., Alt, F., Müller, J., Schmidt, A., Isakovic, K., Klose, S., Schiewe, M., and Haulsen, I. Audience behavior around large interactive cylindrical screens. In Proc. CHI 2011. ACM Press (2011),1021–1030.

[3] Çavus, M., and Ozcan, O. To Watch from Distance: An interactive Film Model Based on Brechtian Film Theory. Digital Creativity 21, 2 (2010), 127-140.

[4] Dalsgaard, P. and Halskov, K. 3d projection on physical objects: design insights from five real life cases. In Proc. CHI 2011. ACM Press (2011), 1041-1050.

[5] Hurwit, J. M. Reading the Chigi Vase. Hesperia: The Journal of the American School of Classical Studies at Athens , 71, 1 (2002), 1-22.

[6] Khalilbeigi, M., Lissermann, R., Mühlhäuser, M., and Steimle, J. Xpaaand: Interaction techniques for rollable displays. In Proc. CHI 2011. ACM Press (2011), 2729-2732.

[7] Khalilbeigi, M., Lissermann, R., Kleine, W., and Steimle, J. FoldMe: Interacting with Dual-sided Foldable Displays. In Proc. TEI 2012. ACM Press (2012), 33-40.

[8] Lahey, B., Girouard, A., Burleson, W. and Vertegaal, R. PaperPhone: understanding the use of bend gestures in mobile devices with flexible electronic paper displays. In Proc. CHI 2011. ACM Press (2011), 1303-1312.

[9] Lee, J. C., Hudson, S. E. and Tse, E. Foldable interactive displays. In Proc. UIST 2008. ACM Press (2008), 287-290.

[10] Özcan, O. Cultures, the Traditional Shadow Play, and Interactive Media Design. Design Issues 18 , 3 (2002), 18-26.

[11] Özcan, O. Turkish - Ottoman miniature art within the context of electronic information design education. Journal of Technology and Design Education 15, 3 (2005), 237–252.

[12] Poupyrev, I., Newton-Dunn, H., Bau, O. D20: Interaction with Multifaceted Display Devices. In Proc. CHI 2006. ACM Press (2006), 1241-1246.

[13] Steimle, J. and Olberding, S. When Mobile Phones Expand Into Handheld Tabletops. In Proc. CHI 2012. ACM Press (2012), 271-280.

[14] Ünlüer, A and Özcan, O. Sound and Silence in the Line: Re-Reading Turkish Islamic Calligraphy for Interactive Media Design. Leonardo 43, 5 (2010), 450-456.

[15] Weiss, M., Voelker, S., Sutter,C., and Borchers, J. BendDesk: dragging across the curve. In Proc. ITS 2010. ACM Press (2010), 1-10