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.

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

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

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

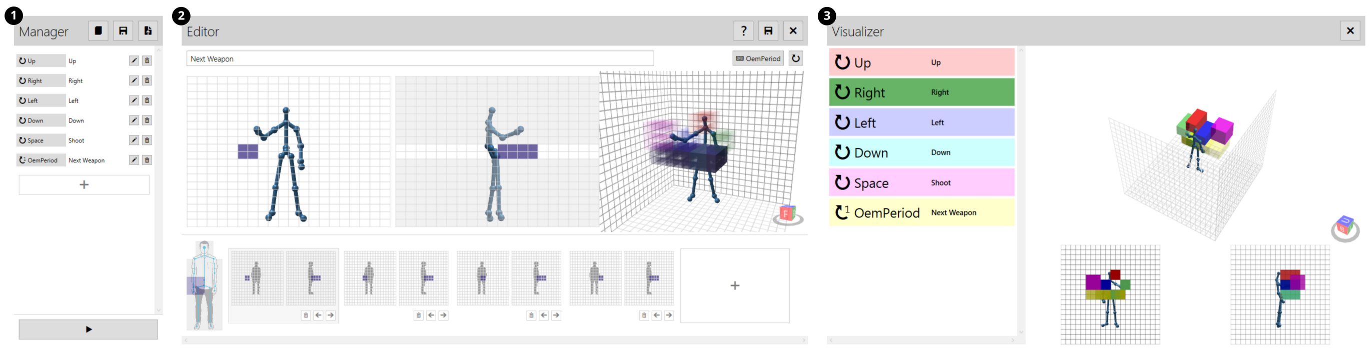


Figure 1. Hotspotizer consists of three modules: (1) The Manager lists all of the gestures in the current collection and allows creating, saving and loading collections as well as adding, removing and editing individual gestures. (2) The Editor is the main workspace where gestures are authored. (3) The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

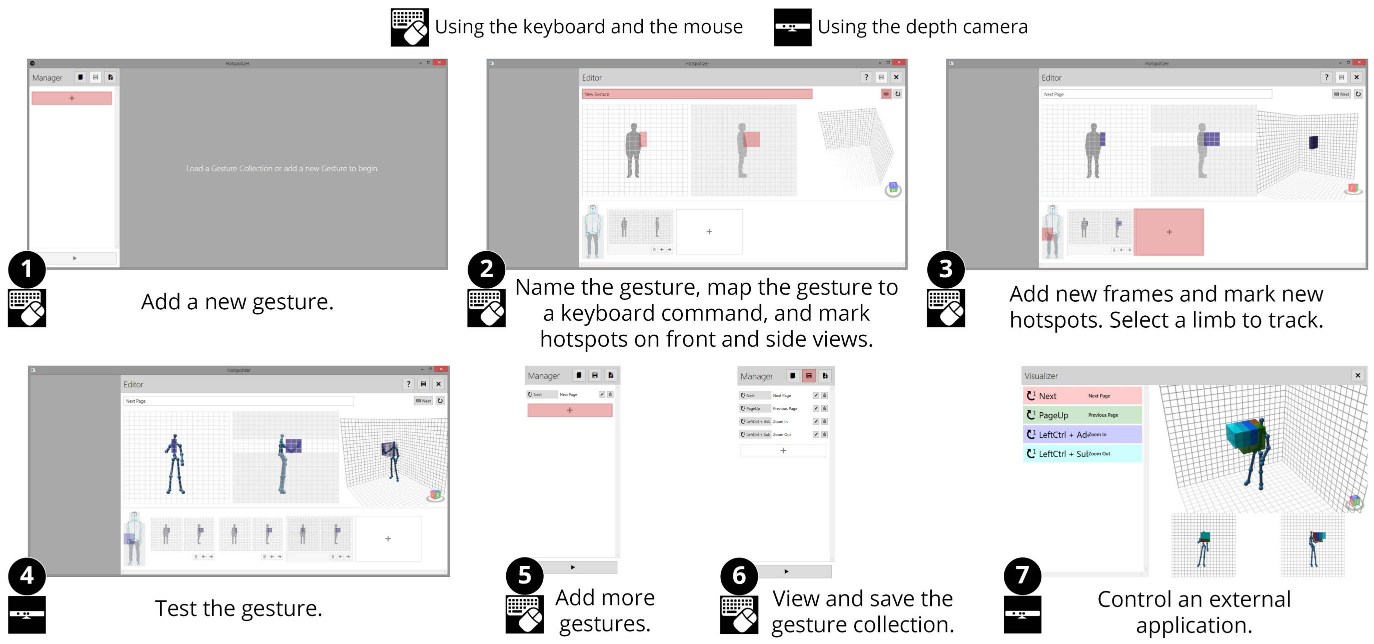

Using the keyboard and the mouse
Using the depth camera
Load a Gesture Collection or add a new Gesture to begin.
1
Add a new gesture.
2
Name the gesture, map the gesture to a keyboard command, and mark hotspots on front and side views.
3
Add new frames and mark new hotspots. Select a limb to track.
4
Test the gesture.
5
Add more gestures.
6
View and save the gesture collection.
Next
PageUp
LeftCtrl + Add
LeftCtrl + Sub
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).

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

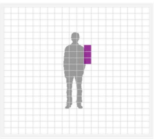
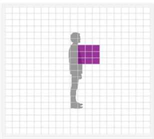
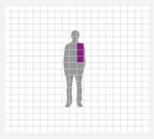
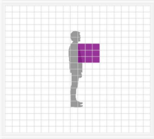
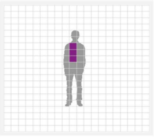
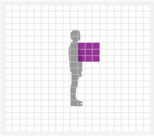
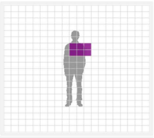
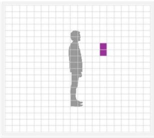
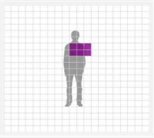
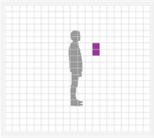
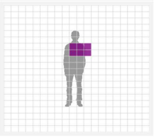
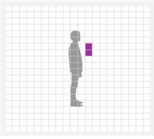
Function	Hotspots			Limb	Command
	Frame	Front View	Side View		
Next Page	1			Left Hand	*Page Down* (once)
	2				
	3				
Zoom In	1			Right Hand	*Ctrl + Plus* (once)
	2				
	3				

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.

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

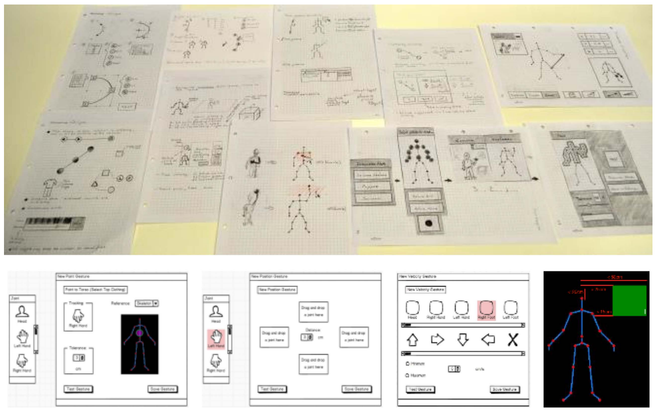


Figure 4. Rough sketches, paper prototypes and mockups were used to gather feedback which directed design and development.

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.

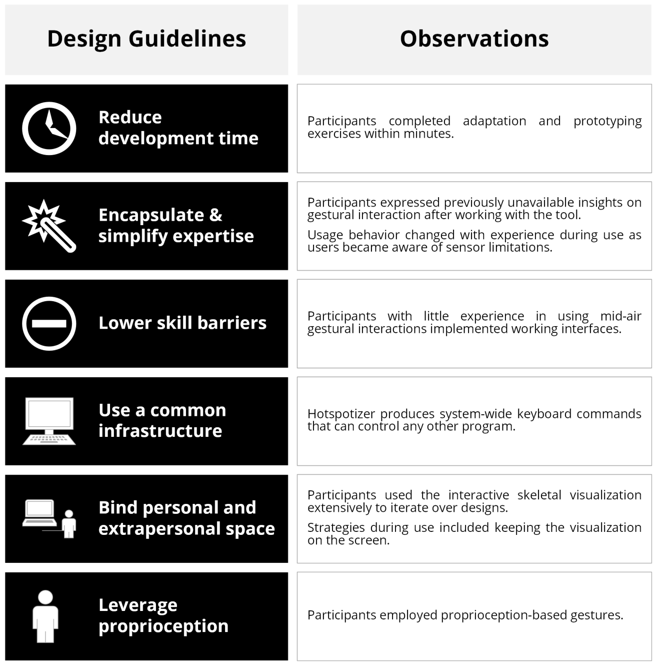
To evaluate Hotspotizer in use, we conducted two studies. The first was a study with 5 users to assess if Hotspotizer conforms to its design rationale. The second was a class workshop with 6 students working in pairs to build interactive prototypes of gestural interfaces. Qualitative results from these 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

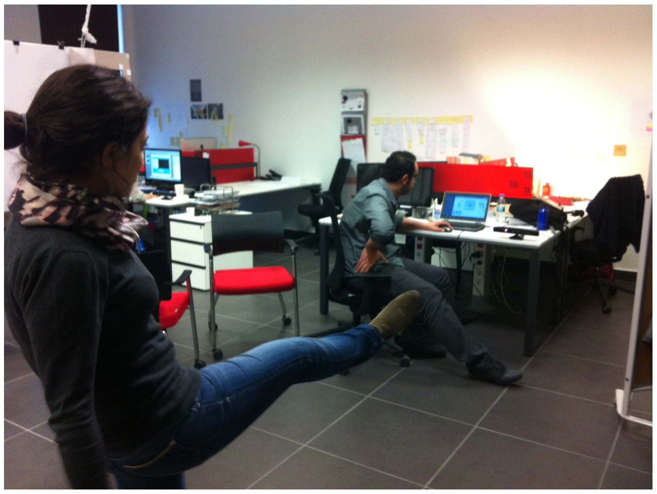


Figure 6. User strategies included working in pairs. One user performs gestures in front of the sensor while the other marks hotspots that correspond to limb positions.

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).

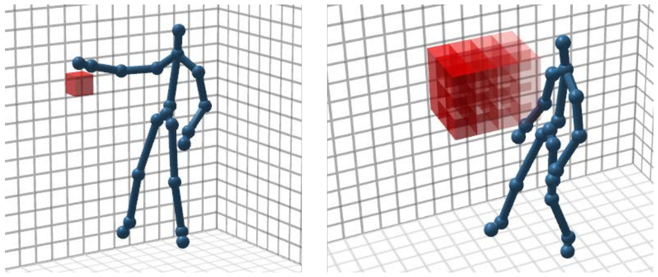


Figure 7. Initially, users preferred gesture designs that involved small hotspots and unspecified motion. Frames were added to constrain motion, and hotspots were enlarge to allow for variations during gesturing. Here, both panes depict hotspot configurations that may be used for a "punch" gesture. The 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.