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.

http://ceur-ws.org/Vol-1190/.

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.

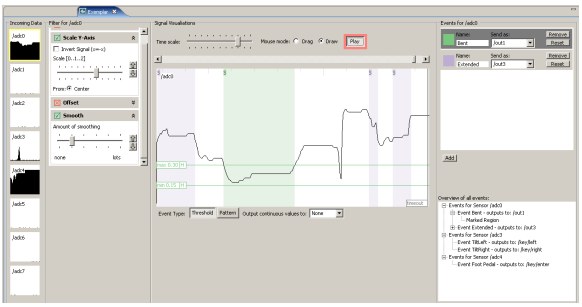

Figure 1: The Exemplar gesture authoring environment. [3] From left to right, the interface reflects the developer's workflow: Data from various sensors connected to the system is displayed as thumbnails and the sensor of interest is selected; filters are applied to the incoming signal; areas of interest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

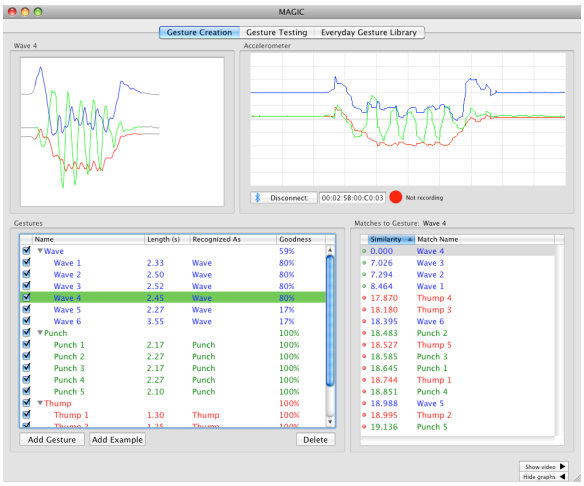

Figure 2: MAGIC's gesture creation interface. [2]

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.

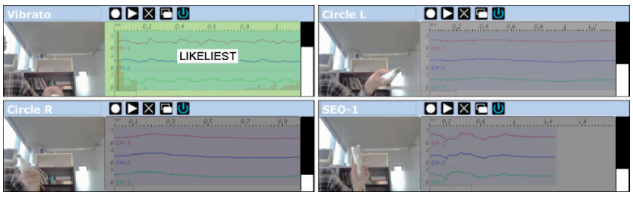

Figure 3: The "follow" mode in the GIDE interface. [8]

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.

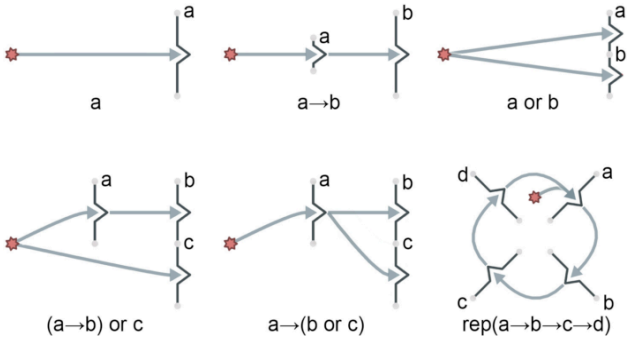

Figure 4: EventHurdle's visual markup language allows for a variety of compositions: (from top left) a simple gesture with one hurdle; serial and parallel compositions; combinations of serial and parallel compositions; recursive gesturing. [6]

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.

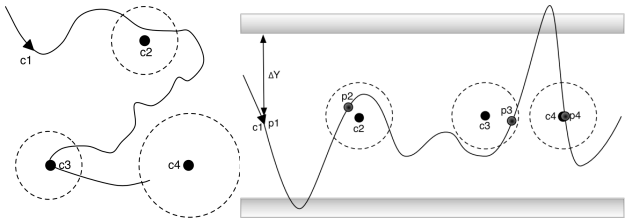

Figure 5: Using control points to represent gestures [4]**. (Left) A "noisy" gesture still gets picked up due to relaxed boundaries around control points. (Right) Negation is introduced via vertical boundaries so that large movements in the vertical axis are distinguished from the desired gesture.**

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/

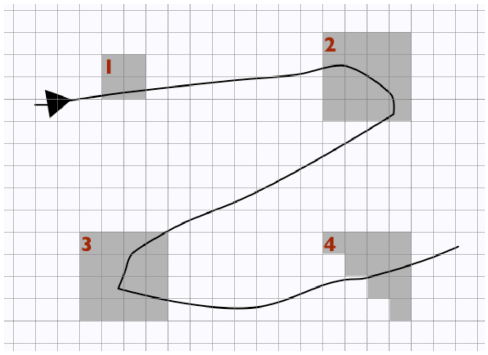

Figure 6: A 2-dimensional "Z" gesture defined using ordered hotspots in discretized space.

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/

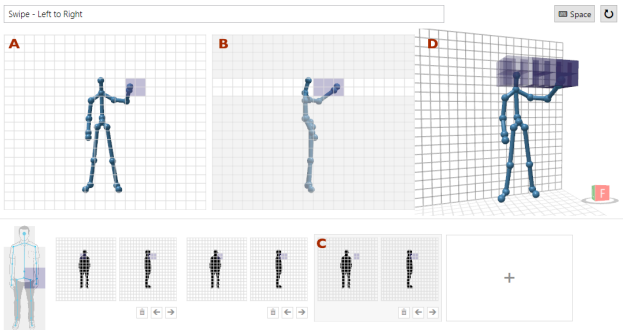

Figure 7: A 3-dimensional "swipe" gesture to be performed with the right hand, implemented in Hotspotizer. The front view (A) and the side view (B) depict the third frame, selected from the timeline (C). The 3D viewport (D) depicts all three frames, using transparency to imply the order.

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.