AnyGesture: A Flexible Web-based Modality Management System
Technical Report HMT-11-04

Sean T. Hayes

Abstract

Touch interaction is the default method of input on most of today’s smartphones and tablets. The general public is familiar and relents with basic interactions utilized by touch technology to interact with web content while on the go. However, there are many limitations with this direct input method. Gestures combined with other modalities can be utilized to improve accuracy and speed especially while the user is mobile. However, developer tools to develop novel interactions for mobile web applications are limited. AnyGesture is a lightweight gesture management system to manage the low-level input and high-level gestural output utilized by custom gesture recognizers. A preliminary proof-of-concept has demonstrated the flexibility and ease of development for utilized custom recognizers of discrete and continuous gestures from mouse and touch input.

Index Terms

gestures, multimodal, interaction, mobile, web, HTML5, CSS3, application.

I. Introduction

Touch interaction is common on most portable consumer devices. Current touch interaction is generally basic and inefficient (e.g., tapping large buttons, swipes to avoid unintentional input, etc.). These interactions are direct, simple, and intuitive, but have many limitations. Targets must be large enough that it is not completely covered by the contact area. Often a small number of controls dominate the limited screen real-estate.

The unique attributes of touch input can be used to reduce errors and provide flexible intuitive manipulation for a variety of controls and data types. Touch input permits leveraging more degrees of freedom than are typically utilized. Each contact represents an area rather than a single point. This area has a shape by which the orientation of the finger may be inferred and some devices measure the contact pressure as well. Multiple contact points may be present simultaneously. These contacts can move somewhat independently of each other allowing a large number of movement combinations. While these attributes have been incorporated in various techniques, effective touch interaction for mobile devices has not been achieved.

Web technologies are growing to support more complex applications and provide features specifically for mobile devices. Developers now have the ability to implement applications exclusively from standards-based web technologies (e.g., HTML, CSS, JavaScript, etc.) that run in the browser. Because the technology is standards-based, cross-platform development is natural. Platform specific features can extend the shared code as desired. Compared to a native application, hardware access is limited. As shown in Figure 1, a web application only has the API’s available within the browser, while a native app can access all the high-level and low-level API’s provided by the operating system.
While touch support is present in the latest draft specifications, no standard for high-level gestures is being developed. Some mobile browsers and web libraries extend touch input to support gestures. However, these gesture types are usually predefined and sometimes limited to a specific control type. The tools necessary to easily develop new gesture types and custom responses to such interactions are important for the advancement and adaption of new input techniques. Mobile devices are also supporting more sensor types (e.g., accelerometer, compass, camera, etc.). Providing a library to effectively access and fuse other sensors and modalities will be useful to developers and researchers working on novel and traditional interaction techniques.

*AnyGesture* is a new gesture management system that supports flexible high-level interaction recognition from lower-level web-browser events. This paper describes the design, implementation challenges of the system. Insights on how to utilize the system are presented. Finally, the current limitations and future development plans are presented.

## II. Background

The number of mobile operating systems and support libraries available continues to increase. This section provides an overview of current support by highlighted abilities of many of the most popular platforms.

### A. Standards

There is no standard for gesture input. However, the draft standards for HTML5 by the World Wide Web Consortium (W3C) include a new low-level touch-event API [2]. Touch events, created by the browser, occur when there is a change in the current touch-point(s) in contact with the surface. Touch contacts may be a stylus or the fingers, but events are herein described in terms of finger contacts. There are six types of touch events in the specification.

- **Start**: A finger has been placed on the touch surface.
- **End**: A finger has been removed from the touch surface.
- **Move**: One or more touch points have been moved.
- **Enter**: A touch point has moved within an object within the page.
- **Leave**: A touch point has moved out of an object within the page.

Fig. 1: An generalized comparison of the hardware accessibility for web applications versus native applications [1].
• **Cancel**: A touch point has been “disrupted in a implementation-specific manner.”

These touch events are handled by implementing a listener function for each event type of interest and adding it to the Document Object Model (DOM) element within which the events will occur. Touch events are utilized by the other libraries including AnyGesture to create high-level gesture events that can be the result of multiple touch-events received over time.

Echtler has been working with the Applications Area Working Group to develop a standard specification language for defining gestures [3]. Gestural Interface Specification Language, GISpL, is designed to enable unambiguous descriptions of gestures. The language describes both the motions to trigger a gesture and information about the handling of the gesture by the system. The idea is that new gestures can be easily added to a system and promote shared interactions across applications and platforms. However, I am unaware of any systems that currently utilize the draft standard.

### B. System and Browser Support

iOS (version 3.2+), the operating system found on Apple’s mobile devices, supports gesture recognizers for native applications [4]. However, custom recognizers only receive touch events and cannot be implemented using web technologies. Apple does implement a non-standard gesture event that is accessible through the browser [5]. While gestures defined by the OS are available, custom gesture-recognizer support is not provided. [6].

The Android OS supports low-level multitouch events and single touch gesture events for native applications [7]. However, like iOS, gesture events are not available within the browser. Windows Mobile supports five kinds of multitouch gestures natively. Custom gesture support is not available within or without the browser.

None of these platforms support gesture recognition using modalities other than touch. Such recognition is possible, but it has to be managed by the application developer with only low-level support from the operating system and browser.

### C. Web-based Touch Libraries

There are many frameworks available to aid in the development of mobile web applications. However, few provide the flexibility to effectively support a large set of custom gesture types: custom input events, state maintenance, and automated gesture sorting and event firing.

Another limitation of currently available frameworks is that custom gestures are received as multiple events even if a single event triggered the recognition of multiple gesture types. By separating gestures into multiple events, the handler cannot take into consideration the other interactions being simultaneously performed on the target. For example, a two fingered touch event might trigger rotate, zoom, and pan events simultaneously (see Figure 2). The handler may wish to only perform the most significant gesture and properly detect accidental gestures. If all the recognized gestures should be handled, improved efficiency can be obtained by performing all calculations together fol-
lowed by a single update the view. These tasks are not possible if handlers receive the gestures in separate events without greatly increasing the complexity of the handling and introducing delays (e.g., maintain a state buffer that is checked on a timer).

jQuery Mobile is a framework that extends jQuery to unified user interfaces similar to native phone and tablet applications [8]. jQueryMobile recognizes tap, tap and hold, swipe, and scroll gestures [9]. Custom gesture support is not provided, but example plugins are available that add new event types [10]. All aspects of the gesture recognition must be implemented by the author of the plugin.

Sencha Touch is similar to jQuery Mobile, but specifically tries to match the native look and feel of iPhone, Android and BlackBerry devices [11]. Sencha Touch seeks to unify low-level input events from mouse, pen, and touch devices. These events are all mapped to a single set of events, which eliminates writing separate code for each modality [12]. While this adds a little abstraction if you are interested only in those event types, Sencha Touch does not provide any gesture support beyond the built-in gesture types related to their data visualization widgets (i.e., pan, rotate, zoom, highlight, reset, combine). An API for custom events is available, but management of the recognition, states, and triggers are left to the developer.

The Dojo toolkit provides tools for developing both desktop and mobile applications [13] and to the author’s knowledge provides the best gesture support for mouse and touch input. The Dojo toolkit, like Sencha Touch, unifies the mouse and touch events into a standard set of events. It currently provides only a few built in gesture recognizers, but will also manage custom gestures. Custom gestures recognizers are extensions of Dojo’s base gesture object and automatically receive press, move, release, and cancel events along with the states previously set by the recognizer [14]. However, each custom gesture is responsible for firing the gesture events. Therefore, gesture are received individually, even if triggered by a shared input event.

### III. Method

#### A. Features of AnyGesture

AnyGesture was designed to be flexible. The developer has the freedom to define the type of events (e.g., touch, mouse, user interface, audio, etc.) interpreted; the source of the events; the identification, attributes and states of the gesture; and the action to be taken when a gesture is trigger. Some examples of inputs types can be seen on the right size of Figure 1. Green sensors indicate complete access from the web browser and shades closer to brown indicate extremely limited access from the browser.

The gesture recognizer supports discrete and continuous gestures types. That is, the gesture could result in a single gesture event triggered (e.g., a double tap, or sketched shape as in Figure 4) or multiple gesture events that pass the updated states as the gesture is being performed (e.g., rotate, zoom, and pan in Figure 2).

Gesture events are created and fired automatically by AnyGesture. Gesture Events contain all the gestures that are triggered by the same input event and received by the same target. The inclusion of all gestures in a single event provides a full context to the event handler to make better decisions compared to other libraries as described in Section II C.

#### B. Design

There are four components involved in utilizing AnyGesture: the manager module, gesture recognizer objects, gesture objects, and gesture events. The manager performs all the event handling, maintains the statues of the current gestures, and invokes the recognizers. The manager is scoped
and should not require extension or modification by the application developer. Figure 3 provides an overview the data flow through the manager. The manager listens for any input events requested by the the registered recognizers; the recognizers do not listen for an event directly. The manager first filters the events by source and type, matches any currently active pattern states, and sends the event to all the appropriate recognizers. In addition to the input event, the recognizer receives a reference to the source element or object, and the a gesture object. If the recognizer is receiving an event for a destination target for the first time since a previous gesture was complete or canceled, a new gesture object is automatically as defined by the recognizer and passed to the recognizer. Otherwise, the passed gesture object will contain any custom states set by the recognizer in previous events. State tracking is essential for recognizers gesture that occur as a combination of multiple inputs or over time. This method provides flexible per gesture state tracking that does not need to be managed by the application developer. Of course, a recognizer could track more prolonged or global states not directly associated with a single gesture manually if desired. After interpreting the input events in context of the source and targets, the recognizer updates the gesture object and returns a recognition status to the manager.

The manager then organizes the results first by recognizers status. If the status indicates the the gesture has been triggered, finished, or canceled, the gesture will be included in a gesture event. If the the status was maybe or ignore, the gesture objects will be processed accordingly without triggering a gesture event. Internal references to ended gestures are removed and the gesture object is recycled. Gestures to be sent are then added to each target that has registered a gesture listener for that gesture type. A single gesture will be included in multiple gesture events if multiple registered targets exist. Also, gesture events may contain multiple gestures.

Gesture events contain a list of all the gestures for the given target as well as a list of the active gestures and a list of the canceled gestures. The original event source is also provided by default. Any information that the recognizer wishes to include with the event is contained within the specific gesture object. The gesture object also contains some read-only attributes which include a unique id, the overall type of gesture that also identifies the recognizer, and the state of the gesture (i.e., started, updated, finished, canceled).
C. Implementation

AnyGesture is written in pure JavaScript and can be utilized with other client side frameworks. Features of the latest ECMAScript Language Specification (version 5.1) are utilized. Therefore modern browsers is required. Some standard functions (e.g., Array.prototype.indexOf) are added at runtime if the browser currently does not support them. A JavaScript hash table implementation, JSHashTable [15], by Tim Down was also utilized within AnyGesture. Hash tables are highly effective for organizing the various data throughout the system. Javascript objects inherently act as maps (e.g., \( x[y] = z \), stores the value \( z \) into object \( x \) with a key of \( y \)). However, the key must be able to be uniquely represented as a string. JSHashTable allows any data type to be used as a key within the hash.

The AnyGesture source supports full (advanced) optimization with Google’s Closure Compiler, which is a tool to make Javascript download and run faster [16]. The compiled code is only 33.51% of the original size.

D. Testing and Future Work

AnyGesture is still very early in the testing phase. Current testing has included two gesture types: strokes and icons. Both gestures support finger and mouse input. The iconic gestures are recognized using the algorithms implemented from the Dollar Recognizer [17] and includes the gestures described in Figure 4. Four browsers were utilized for testing within Windows 7: Arora 0.11 with WebKit version 534.34, Microsoft Internet Explorer 9, Mozilla Firefox 8, and Google Chrome 16. All browsers behaved consistently and successfully recognized and handled the gestures. The recognition process has been successfully tested using a single input source, multiple input events, and a single destination source. Further testing is necessary to validate the implementation. Also, more gesture types need to be utilized to validate the system design. Furthermore, it is theoretically possible that the system will function with a combination of modalities beyond touch and mouse.
interaction. Further proof of concepts are required to verify this possibility.

IV. Conclusion

The AnyGesture design will be a lightweight and highly flexible pattern recognition system. Although it is early in the testing phase, the system has been shown be effective with some mouse- and touch-based event types. Evaluation of development of AnyGesture will continue to create a robust management system and libraries of gesture recognizer plugins. This tool will prove useful to researchers and developers of mobile web applications seeking to create robust and flexible recognition systems. Due to the simplicity of the design, AnyGesture can also be ported to native platforms.

REFERENCES
