jQMultiTouch: Lightweight Toolkit and Development Framework for Multi-touch/Multi-device Web Interfaces

Michael Nebeling and Moira C. Norrie
Institute of Information Systems, ETH Zurich
CH-8092 Zurich, Switzerland
{nebeling,norrie}@inf.ethz.ch

ABSTRACT

Application developers currently have to deal with the increased proliferation of new touch devices and the diversity in terms of both the native platform support for common gesture-based interactions and touch input sensing and processing techniques, in particular, for custom multi-touch behaviours. This paper presents *jQMultiTouch*—a lightweight web toolkit and development framework for multi-touch interfaces that can run on many different devices and platforms. jQMultiTouch is inspired from the popular jQuery toolkit for implementing interfaces in a device-independent way based on client-side web technologies. Similar to jQuery, the framework resolves cross-browser compatibility issues and implementation differences between device platforms by providing a uniform method for the specification of multi-touch interface elements and associated behaviours that seamlessly translate to browser-specific code. At the core of jQMultiTouch is a novel input stream query language for filtering and processing touch event data based on an extensible set of *match predicates* and *aggregate functions*. We demonstrate design simplicity for developers along several example applications and discuss performance, scalability and portability of the framework.

Author Keywords

Multi-device interface toolkit; multi-touch framework

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: User Interfaces—*Input devices and strategies, Interaction styles*

General Terms

Design, Human Factors

INTRODUCTION

Application developers face the increased proliferation of new touch devices, nowadays ranging from smartphones, tablet PCs and touch notebooks to all-in-one touchscreen solutions, tabletop systems and interactive walls. Unless an application is to be designed for a specific device only, it is becoming increasingly difficult for developers to create interfaces that cater for the wide range of possible settings. In particular, there are three major technical and design challenges for current multi-touch frameworks: **1) Many technological differences between touch devices**: Multi-touch devices vary in terms of the amount of touch points they can track simultaneously as well as the tracking speed and the support for other advanced sensing techniques based on accelerometers or tilt sensors. Some devices such as the Microsoft Surface[1] tabletop system provide additional tracking support for tangible objects in the form of physical tokens, while others are limited to finger tracking. Available frameworks are therefore often designed for a specific technological setup, e.g. DiamondSpin [21] or DTFlash [7] for the DiamondTouch system [5], and tend to focus their support on either finger or tangible object tracking [13]; **2) Different software architectures and implementation methods**: Developers require a wide range of skill sets and experience with different programming languages and software development kits (iPhone, Android, etc.) in order to build applications for the latest generation of smartphones and tablet computers [3]. Some frameworks such as PyMT [9] and MT4j [16] therefore instead build on cross-platform programming languages such as Python or Java and protocols such as TUIO [12] to achieve a higher degree of interoperability between different platforms and devices, but this comes at the cost of additional abstraction layers and requires device-specific drivers that implement the protocols [6]; **3) Limited support for extensibility**: Existing frameworks are often designed for a single class of applications only. For example, most of the aforementioned frameworks are specifically designed for either mobile platforms or tabletop systems and are therefore not easily extended towards other settings. In addition, the implementations typically provide a fixed set of basic gestures and are generally difficult to extend with support for custom and application-specific multi-touch behaviours. Recent solutions such as Midas [20] or Proton [14] introduce domain-specific languages with the aim of supporting developers in the design and implementation of new gestures, but do not share our specific goal of supporting multi-touch interface development for many different devices.

We propose *jQMultiTouch*—a lightweight web toolkit for creating multi-touch interfaces that can run on multiple devices.

[1] http://www.surface.com

EICS'12, June 25–26, 2012, Copenhagen, Denmark.

Our framework directly builds on top of modern browser engines, such as WebKit[2] or Mozilla's Gecko[3], and therefore carries the potential of providing a lightweight solution for multi-touch application development based on established and widespread web technologies. jQMultiTouch is similar in implementation to two recently developed web frameworks, jQTouch[4] and Sencha Touch[5], but more general in terms of the concepts and features it supports since both these frameworks are primarily designed for mobile application development. More importantly, they suffer from rather limited support for multi-touch in that interactions can involve only one screen object at a time. While this may be sufficient on small-screen devices usually operated by a single user, the potential benefits of larger interactive surfaces, where multi-finger, multi-hand or even multi-user input can play an important role [11, 24], are not leveraged. jQMultiTouch therefore aims to be a more general framework that addresses the core requirements of multi-touch and gesture-based interactions within interfaces, but at the same time, does not limit itself to a specific platform or type of device.

jQMultiTouch is inspired from jQuery[6], one of the most popular web toolkits that has arguably changed the way developers nowadays implement web interfaces. jQuery is designed to simplify web scripting tasks such as the selection and manipulation of DOM elements and handling of events through the help of callbacks, as well as creating advanced interfaces and interactions with animations and visual effects that typically involve sliding and fading of web page elements. jQuery provides powerful abstractions from low-level implementation details and resolves cross-browser compatibility issues, which contributes to the ease of use and design simplicity for developers. jQMultiTouch is not only similar to jQuery in terms of the idea of providing a lightweight and general framework, but also because of the fact that we have adopted ideas from jQuery and applied them to some of the core concepts. The main technical contributions of our work include (1) a device-independent method for the processing and handling of multi-touch events within web interfaces, (2) a novel concept of a touch history that functions as the central source for event handling which is particularly useful given the more complex event flows with multi-touch and gesture-based interactions, as well as (3) a toolkit and multi-touch framework based on only native web technologies that do not require external browser plug-ins.

Applications based on our framework have been successfully deployed on many new touch devices, including Apple's iPhone and iPad or other Android-based smartphones and tablets, the TouchSmart[7] all-in-one PC and tabletop systems such as Microsoft's Surface, without the need for switching between special software development kits. Our decision to build on web technologies also has other advantages. In particular, active support for touch input or gesture-based modality *within* web interfaces is still in its infancy in that multi-touch interaction in a web context is generally limited to gestures for scrolling and zooming of content as interpreted by web browsers. The fact that most modern browsers have recently started to integrate support for processing touch input is promising as it means that multi-touch support will no longer be limited to specific browsers or require additional plug-ins such as Flash or Silverlight. However, the problem remains that native browser support still varies considerably in terms of touch event models and default browser behaviour due to the lack of standards. For example, Firefox 4 introduced custom `MozTouchDown`, `MozTouchMove` and `MozTouchUp` events[8] with minimal information in terms of touch coordinates relative to the viewport of the browser and a unique identifier to track continuous touches of the same input source. On the other hand, WebKit-based browsers, such as Apple's Safari which is used on the iPhone and iPad, only trigger handlers associated with `touchstart`, `touchmove` and `touchend` which provide additional information that is absent in Firefox, such as scale factors and degree of rotation if the commonly associated gestures are currently being performed on the target element. Moreover, the event callback mechanism differs considerably and touch event data is separated into all active touches on the screen, touches only related to the current target element and changed touches since the last time an event was handled. Again, in Firefox this important information is missing. For developers, this means that applications are presented with different input data depending on the browser and therefore considerable effort is required to eliminate cross-browser compatibility issues. We argue that solving these problems can enable a new generation of web interfaces that will start to include carefully designed multi-touch features and therefore bring real benefits to users when working with applications on a touch device.

We begin by presenting the concepts of jQMultiTouch and its main features as well as the implementation. This is followed by an evaluation of the framework in two parts. First, we present two applications that we developed based on jQMultiTouch and discuss them in more detail. We then sketch the range of possible applications and the framework support for rapid prototyping by showing more examples created by students as part of an assignment. We close with a discussion of the performance and extensibility of our framework.

JQMULTITOUCH

In the spirit of jQuery, jQMultiTouch provides abstractions from low-level multi-touch event handling details as well as cross-browser support for custom and default gesture-based interactions with interface elements. Figure 1 shows the four-layered architecture for applications based on our framework as well as the main components responsible for multi-touch support. The framework builds on basic browser support for multi-touch events available in modern browsers and extends it with customisable event handlers and attachable behaviours.

[2] http://www.webkit.org
[3] http://developer.mozilla.org/en/Gecko
[4] http://www.jqtouch.com
[5] http://www.sencha.com/touch
[6] http://jquery.com
[7] http://www.hp.com/touchsmart

[8] Note that, since Firefox 6, the touch event API has aligned with the W3C proposal. However, W3C's Touch Events specification is under active development and browser support not always consistent.

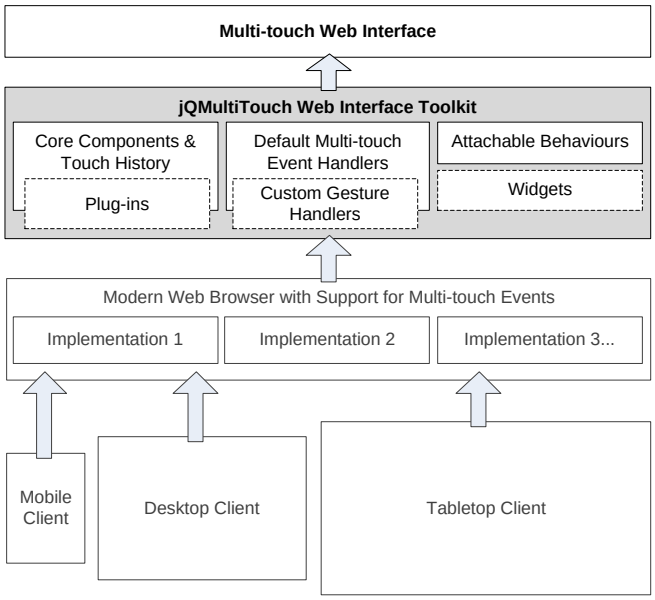


Figure 1: Four-layered architecture for multi-touch web applications based on our framework with support for different mobile, desktop and tabletop platforms as well as customisable multi-touch event handlers and attachable behaviours.

Applications developed with our framework can run on a number of different mobile, desktop and tabletop platforms. jQMultiTouch therefore functions as both a run-time environment and an extensible platform for developers to meet application-specific requirements.

To illustrate how jQMultiTouch can be used for multi-touch interface development, we refer to the simple picture viewing application shown in Figure 2. The code uses a simple jQuery statement `$('img')` to select all images loaded into the web interface and attaches the `touchable` behaviour provided by jQMultiTouch (Figure 3). Doing this first of all transforms all images into touch areas and then associates corresponding handlers by setting options for the default behaviour. In the example, we allow users to tap and hold images in order to perform unconstrained drag-n-drop operations by setting `draggable` to `true`. This means that pictures can be moved around by translating their x and y coordinates relative to the viewport. We further enable scaling using pinch/spread gestures so that users can enlarge or shrink the pictures within a certain range for the scale factor. This is achieved by setting the `scalable` option accordingly. Finally, we set the `rotatable` option to allow rotation of images by steps of 15 degrees. All features are enabled based on the attachable `touchable` behaviour provided by jQMultiTouch.

Core Features

Our framework consists of four components for touch event handling, touch event tracking and coordination, touch event capture and delegation and touch device detection (Figure 3). The core features of jQMultiTouch are built around a unified method for touch event handling independent of the specific platform and browser used to execute the interface. For this, we introduce browser-neutral `touchDown`, `touchMove` and

```
$('img').touchable({
  draggable: true, scalable: { min: 0.4,
    max: 2.0 }, rotatable: { step: '15deg' } });
```

Figure 2: Simple picture viewing web application supporting multi-touch based on our framework, enabling controlled drag-n-drop, scale and rotate actions for images by assigning the new `touchable` behaviour.

`touchUp` events that are then mapped to the specific events used in browsers. Moreover, the touch data associated with touch events is normalised so that rotation and scale properties are provided independent of native browser support. This is achieved with the help of additional methods for measuring the angle and distance between touch points in order to determine the scale factor and degree of rotation. The event callback handlers provided by jQMultiTouch will be fed with separate information about all active touches and only target-related ones. This data can then be used to determine whether users interact with other components and to define interactions between them. In addition to the standard callback mechanism that may require to create and pass on custom data between touch events, e.g. for data initialisation at `touchDown`, manipulation during `touchMove` and clean-up after `touchUp` events, our framework also provides a separate gesture callback handler. This handler can then be used to manage custom data in a central place. Also, methods are provided to control default browser behaviours and prevent conflicting actions, e.g. for scaling only specific content, such as images in the previous example, rather than the entire page. Finally, the framework defines a method for detecting if the particular device in use supports touch input. If it does, this will then trigger attached behaviours for touch elements. Also included is a method for determining the orientation of the device and a callback mechanism for handling changes.

Touch History Concept

One of the core concepts of the framework addresses the need for touch event tracking and coordination which is required in the case of multiple consecutive or simultaneous touches on one or more interface elements, e.g. for dragging several images at the same time similar to the previous example. To track the position of active touches and the order in which these have appeared and changed over time, jQMultiTouch introduces the concept of a central touch event history (Figure 5). While most existing frameworks internally also work

Component	Features
Touch event handling	**Unified touch event listeners** that work across different browser implementations
	Common touch event properties and multi-leveled information about active touches
	Configurable **default handlers** for common dragging, scaling and rotation operations
Touch event tracking and coordination	**Touch history** for keeping track of touch events and changes in touch data
	Separate **gesture callback handler**
	Mechanisms to **control default browser behaviour** and prevent conflicting actions
Touch event capture and delegation	**Event capture for simultaneous touches** on one or more target elements
	Event delegation to transfer capture to other elements
	Automatic capture release after timeout if no new touch data available for an active touch
Touch device detection	Method to **detect whether device is touch-enabled**
	Method to **determine orientation of device** as well as **callback handler for changes**

(a) Feature Overview

Feature	Examples	Description
$.fn.touchable	$(element).touchable({ draggable: true, scalable: true, rotatable: true });	Marks the element as touchable and registers default dragging, scaling and rotation behaviours
$.fn.touches	if ($(element).touches().length > 1) { … }	Returns true if more than one touch was registered on the DOM element
$.touchHistory	$.touchHistory.each(function() { … })	Keeps a record of the entire touch history and can be used for global analysis
$.touchPrevent Default	$.touchPreventDefault = false;	Enables default browser behaviour (default is to prevent default behaviour)
$.touchEnabled	if ($.touchEnabled()) { … }	Checks whether touch input is available on the device
$.touchReady	$.touchReady = function() { … }	Registers a callback to be executed on startup if device supports touch
$.orientation Changed	$.orientationChanged = function(orientation) { … }	Registers a callback to handle changes of the device's orientation

(b) Features

Figure 3: jQMultiTouch consists of four different components for touch input processing on touch devices. These components are implemented as a set of jQuery extensions specifically for multi-touch interface development.

Predicate	Examples	Description
type	history.filter({ type: 'touchMove' }) history. filter({ type: ['touchDown', 'touchUp'] })	Constrains history to a single type or set of down/move/up events
target	history.filter({ target: element }) history.filter({ target: [element1, element2] })	Constrains history to one or several DOM elements
touch	history.filter({ touch: $.touches[0] })	Filters history by the first touch only
finger	history.filter({ finger: '0..2' })	Filters history by the first three fingers
time	history.filter({ time: '1..100' }) history.filter({ time: '<100' })	Constrains history to a 100ms time window

(a) Filter predicates

Predicate	Examples	Description
clientX/ clientY	history.match({ clientX: '>550' }) history.match({ clientX: '500..550' })	Returns true if every touch in the history is within the position constraints
deltaX/ deltaY	history.match({ deltaX: '>100' }) history.match({ deltaX: '>100', deltaY: '+-10' })	Returns true if the difference between the positions of the first and last touch event in the history is within constraints
netX/ netY	history.match({ netX: '>100' })	Returns true if the finger has moved at least 100 pixels to the right over the series of touch events in the history
All filter predicates	history.filter({ finger: 0, time: '1..100' }).match({ deltaX: '<-100' }) history.match({ finger: 0, deltaX: '<-100', time: '1..100' })	Supports also filter predicates as a shorthand to first constrain and then match the history

(b) Match predicates

Figure 4: One of the core components is jQMultiTouch's touch history. The touch query language provides an extensible set of filter and match predicates as well as aggregate functions for online gesture recognition based on the history of touch events.

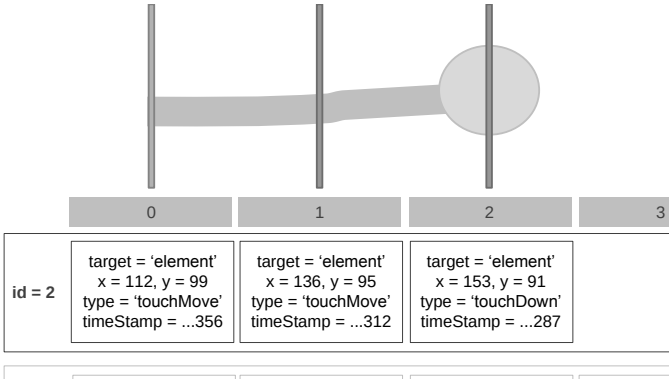
with some kind of touch event buffer, we formalise the concept and provide several new ways of making use of it for uniform touch event handling across different devices.

History Keeping

Figure 5a shows an example of a touch history for the start of a simple swipe-left gesture. For every new source indicated by a different touch id, a separate stream will be created in the history and updated with consecutive `touchDown`, `touchMove` and `touchUp` events. Each of the entries in the history stores touch data such as the position together with the target touch area and the time when the touch occurred. The target is by default locked in consecutive events which enables a simple form of multi-capture for all active touches.

History Evaluation

To make working with the touch history easier for developers, jQMultiTouch provides methods for querying and filtering the entries according to a combination of criteria. Figure 4 shows several example queries. In particular, the framework provides an extensible set of *match predicates* (e.g. `type`, `target`, `touch`, `finger`, `time`, `clientX/Y`) and *aggregate functions* (e.g. `deltaX/Y`, `netX/Y`, `angle`, `area`) for evaluating the current touch history. With the help of these, the history can be filtered by touch id, input source and target elements as well as limited to only return touch data of previous events within a certain window of time. For example, it is possible to compute the delta for a series of touch points and from that to see how each touch has changed within a given time frame. Figure 5b shows how the touch query mechanisms can be used to detect simple swipe left/right gestures. The code implements an example gesture callback handler that is provided with the current touch event `e` that triggered the handler and a touch history related to the target element that binds it. The basic mechanism, in the example used to process the history and look for swipe left/right gestures, is inspired by the way jQuery allows developers to



(a) Excerpt of touch history

```
function gestureHandler(e, history) {
  if (history.match({ finger: 0, deltaX: '<
      -100', time: '1..100' })) {
    // TODO swipe left handler
  } else if (history.match({ finger: 0,
      deltaX: '> 100', time: '1..100' })) {
    // TODO swipe right handler
  }
}
```

(b) Simple gesture handler

Figure 5: Illustration of touch history for a swipe left gesture on `element` and a possible gesture handler.

script animations.[9] In jQMultiTouch, we apply this concept to define simple gesture templates programmatically using relative values for variables to be tracked and compared. The `match` method of the touch history therefore takes a gesture template as an argument and returns `true` if it matches against the touch history. In the example, we define two simple templates that look for touch changes of the horizontal position. The evaluation then first computes the delta between the first and the last `touchMove` events within the last 100 milliseconds (indicated by the 1..100 range expression). This delta is then compared to the provided value, i.e. swipe left is then recognised if the x position in the touch data has decreased by at least 100 pixels, and swipe right if the delta for x is greater than 100.

Note that the gesture callback handler can be used for registering additional gestures, potentially giving priorities to certain gestures as well as resolving conflicts between them, and to execute corresponding actions. For more complex gestures, the `match` method of the touch history can also take an array of gesture templates and will then return true only if they all match the given criteria. Distinct or partly overlapping periods defined for the `time` variable in templates can then be used to define composed or sequential gestures and to increase sensitivity of gesture recognition. In addition, the templates used for gestures can also be shared between different callback handlers by using global variables instead. Rather than relying on comprehensive gesture recognition frameworks, such as iGesture [22] or the 1$ gesture recogniser [23] for more complex stroke-based gestures, this provides a simple way of detecting basic online gestures as they are executed on one or multiple touch elements, such as pinch-to-zoom, panning and tilting which are typical for multi-touch interaction [11, 24].

History Manipulation

Finally, the touch history provides several methods for manipulating the touch history. For example, overriding the target of a touch event provides a basic mechanism for delegating touch events so that the capture can be transferred to elements other than the original target. This can, for instance, be useful to support dragging of a dynamically created intermediate representation of the dragged element rather than the original target on which the `touchDown` event occurred. The same mechanism can also be used for controlling multi-capture and automatically releasing event capture after a certain timeout, i.e. if no further updates on touch data have been received. This is necessary for cases where an active touch leaves the browser window and hence does not fire the expected `touchUp` event.

Attachable Behaviours

As mentioned earlier, our framework provides default implementations for basic interactions with touch-enabled web interface elements, such as drag-n-drop, zoom and rotate actions. The underlying gestures, i.e. moving two fingers apart/toward each other for enlarging/shrinking elements, and using one finger to pivot around another or moving two fingers in opposing directions for rotation, are thereby detected and interpreted either based on native support, as in the case of Safari, or using features of the touch history component to compare the distance and angle between consecutive touch events, e.g. in Firefox. The default handlers provided by jQMultiTouch therefore enable the basic set of interactions supported in many multi-touch applications [14, 20]. However, to provide developers with a richer set of features, their behaviour is also customisable through a number of parameters as illustrated in Figure 2. For example, it is possible to limit the scope for drag-n-drop operations to only certain areas or special components of the web interface, as well as the scale factors for zoomable elements and the degrees and steps by which elements can be rotated. Finally, jQMultiTouch also allows for intercepting the chain of events through various before/during/after callback handlers, or even completely replacing the default behaviour with custom implementations.

One of the advantages of having attachable behaviours similar to jQuery is that they can be associated with any interface component. This allows developers to reuse custom behaviours, bundle them with new, maybe application-specific components and share them between different applications.

Legacy Support and Extension Mechanisms

Touch input shares some commonalities with mouse input since both trigger a series of down/move/up events with point

[9] http://api.jquery.com/animate

coordinates of where the input occurred. In all modern web browsers, single touch input is per default mapped to mouse events with the benefit that traditional implementations remain operational also on touch devices. On the other hand, simultaneous touches are usually not translated to mouse events and can therefore not be processed by traditional event handlers. This raises two major problems for application developers. First, the fact that single touches also fire mouse events is not always convenient, especially when mouse input should be treated differently from touch. Second, even the most advanced implementations for interacting with web interface elements offered by the jQuery UI framework[10] will not work properly with multiple objects at the same time even if the respective event handling methods are linked to touch events. The reason for this is that current implementations typically rely on the fact that there is normally only one variable to track for the mouse, i.e. the position of the mouse cursor. Hence, often a single global variable is used to store the current position, which would then be overridden with every other touch event being processed. To prevent such conflicts, it is important that touch-related data is cleanly associated with the target it concerns, but this requires fundamental changes in the code of most existing solutions.

jQMultiTouch essentially provides two solutions to this problem. First, the attachable behaviour mechanism can be used to override existing implementations. For example, jQMultiTouch's `draggable` and `scalable` behaviours could be used to override similar `draggable` and `resizable` behaviours of the standard jQuery UI framework. This provides a simple way of automatically extending existing applications with support for multi-touch and could also provide the basis for turning single-user web interfaces into multi-user applications. Second, building on jQMultiTouch's `touchDown`, `touchMove` and `touchUp` events in addition to traditional mouse handlers provides a way of supporting advanced multi-touch features as well as maintaining legacy support. jQMultiTouch's ability to control default browser behaviour can then be used to disable default browser behaviour so that touch events will not automatically fire mouse events.

Finally, the basic support for gesture recognition based on the touch history could be easily extended by registering new match predicates and aggregate functions. It is also possible to combine jQMultiTouch with existing gesture recognition libraries. To this end, jQMultiTouch provides a method for converting the data stored in the touch history to a format supported by the recogniser and vice-versa. We have used this technique to integrate jQMultiTouch with the lightweight JavaScript implementation of the popular 1$ unistroke recogniser [23][11].

IMPLEMENTATION

jQMultiTouch is implemented as an extension of jQuery. The implementation consists of three main components: a class `touchHistory` for history keeping, evaluation and manipulation, the `touchable` behaviour for elements to be associated with basic multi-touch interactions, as well as a default `gesture` callback handler. jQMultiTouch has been tested and is compatible with WebKit-based browsers such as Safari on iPhone/iPad, the Android browser and Firefox on Windows 7 touch PCs.

The touch history relies on basic JavaScript array operations for maintaining a history of events. The touch history prototype class provides two methods, `start` and `stop`, for segmenting the touch history using match predicates. Each segment can then be further constrained and evaluated using the `filter` and `match` methods shown in Figure 4. Because each of these functions returns a new touch history object similar to the way it is done in jQuery, it is possible to specify multiple different processing steps in sequence.

As already mentioned, elements marked as `touchable` can be configured with a number of options for touch event handling, such as custom callback handlers as well as default dragging, scaling and rotation behaviours. Each element will be associated with a CSS marker class `ui-state-touchable`, which can also be used for formatting and styling, and bind to the default gesture event handler with cross-browser compatible implementations of the standard behaviours.

The default gesture handler processes a browser-specific touch event `e` and tries to map the type of the event to the uniform touch events `touchDown`, `touchMove` and `touchUp`, or exits if the event cannot be matched by our implementation. For every changed touch, it updates the data or creates a new touch object in the case of a `touchDown` event. In the next step, the touch event will be cached and associated with a `timeStamp`. The uniform `touchEvent` object created in this way will then be passed on to associated touch event handlers of the target elements together with a history of the active touch. The handler also triggers an event for custom gestures, which is instead given a touch history related to the current target rather than only the touch that triggered the event. This excerpt of the history can therefore be used to recognise gesture-based interactions that involved multiple active touches. Each new touch event is appended to the touch history, the size of which is constrained by a configurable maximum size. The touch will remain active until a `touchUp` event is received.

APPLICATIONS

We have used jQMultiTouch for the development of a number of applications as part of our research as well as in teaching and student projects. In this section, we present selected applications based on our framework. The first is FBTouch, an extension of the Facebook picture tagging interface with adaptations for touch and multi-touch. The second is TFlickr, an adaptation of Flickr's picture editing application with more advanced multi-touch handlers compared to FBTouch. The first application was created by the first author and lead developer of the framework to evaluate the feature support, while the second was created in a two-months internship project of a Bachelor student to test the ease-of-use for new developers. Our preliminary evaluation therefore aims to demonstrate the flexibility and potential of the framework as well as providing first insights concerning usability.

[10] http://www.jqueryui.com

[11] http://dev.globis.ethz.ch/jqmultitouch/dollar.html

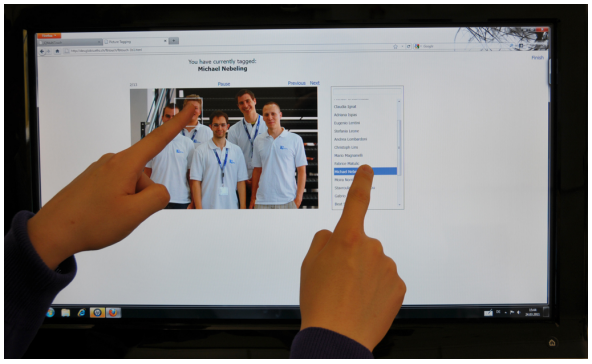
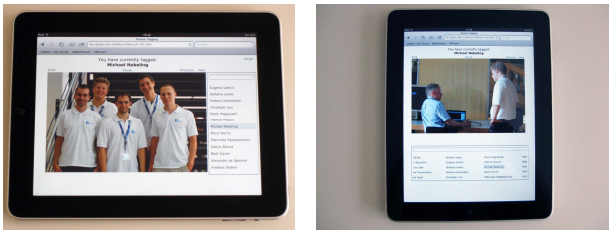


Figure 6: One of the multi-touch versions we have designed and evaluated for a simple picture tagging application similar to Facebook, here using a two-point tagging interaction.



(a) Landscape mode using the two-point tagging interaction

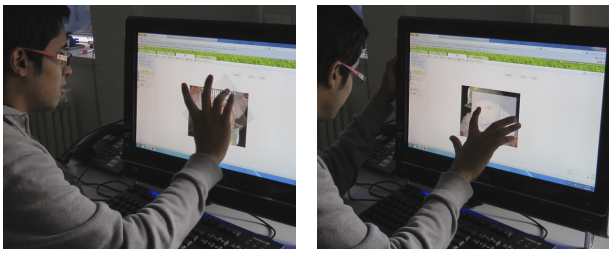
(b) Portrait mode using drag-n-drop interactions instead

Figure 7: Another FBTouch prototype for the iPad using different multi-touch interfaces according to device orientation.

FBTouch

As shown in Figure 6, FBTouch provides a multi-touch web interface for tagging people in pictures. The two main components of the FBTouch application are the picture viewing control and the list of selectable tags. The design is based on the picture tagging application known from Facebook, but has been extended to experiment with two new multi-touch interfaces. Both interfaces enable multi-touch gestures not available in the original Facebook interface, i.e. swipe right or left to navigate to the previous or next picture, spread to overlay a larger version of the picture and pinch to hide the overlay again. The first interface shown in Figure 6 uses a two-point tagging interaction that requires two hands with one finger touching the picture and the other a name in the list. The second version of the interface uses a drag-n-drop interaction that requires dragging a name from the list and dropping it on a person shown in the picture.

Interestingly, Windows 7 on the TouchSmart with which the interfaces were developed and tested, did not allow for simultaneously touching the picture and interacting with Windows standard controls such as the list control used for the name tags. We therefore enhanced the scrolling mechanism in the list of names to support scrolling when users touch the picture at the same time and to prevent accidental tagging/untagging when scrolling occurred prior to the interaction. Not to remove names from the list via drag-n-drop in the second interface, we built on the touch delegation features of jQMultiTouch to drag a thumbnail of the person's photo as an intermediate representation of the original touch target. We also exploited multiple touch event captures so that simultaneous dragging of two or more photo tags is generally possible using multi-finger/multi-hand interaction. To support this, we switched to a horizontal layout to instead place the list below the picture. Also here the default scrolling mechanism was adapted for horizontal scrolling not to interfere with active dragging operations. In another version we developed for the iPad, we make use of both layouts as we switch between the interfaces when the device is rotated (Figure 7).



(a) Rotate interaction

(b) Crop interaction

Figure 8: The Flickr interface recreated using features of jQMultiTouch for common picture editing tasks such as rotate and crop using multi-touch interactions.

TFlickr

Like FBTouch for Facebook, TFlickr is a multi-touch version of Flickr's interface for common picture editing tasks. As mentioned before, the project was carried out in an internship which consisted of three parts. First, the student was asked to explore various adaptations and new multi-touch interactions as possible extensions of the original application. Second, since this project built on an earlier version of jQMultiTouch, the task was to overcome current limitations by making small adjustments to the implementation in order to meet the requirements of the new application. Third, the implementation of several of the new prototype interfaces was simplified by building on the advanced framework support. The final TFlickr application was then composed of the most promising prototypes.

In the first phase, the student created multiple versions of the interface, e.g. for rotate and crop picture editing tasks as shown in Figure 8. The framework support was already considered fairly comprehensive at this stage. However, the project still identified the need for more callbacks, e.g. to provide entry and exit points for extending the rotate function with step-wise behaviour and allowing for more precise selection of the crop area. In addition, a mechanism for temporarily overriding default behaviours and to disable/enable them as required was considered necessary as well as additional parameters for configuring the new features and required thresholds. These requirements led to the latest version of jQMultiTouch reported in this paper with the support for customisable default behaviours mentioned earlier.

RAPID PROTOTYPING WITH JQMULTITOUCH

To further evaluate the framework and its support for multiple different devices, we created an assignment as part of an HCI class designed for Bachelor computer science students.

Figure 9: CNN example application implementing several gestures for navigating between screens.

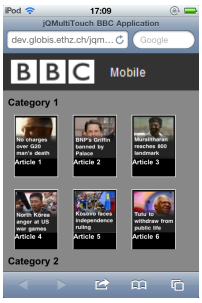
The assignment was divided into two parts and ran over three weeks. First, students were asked to think of an application that could potentially benefit from multi-touch interaction and to first create story boards and paper prototypes before starting with the implementation. In the second part, students were given an introduction to jQMultiTouch and its main features using code examples and were shown how they could use their own devices for development. Since we wanted to minimise the coding effort and given that not all students had a lot of experience with jQuery, they were encouraged to build on the following simple example application as a starting point for their own solutions.

The CNN application shown in Figure 9 consists of a set of five screens with simple gesture-based interactions for navigating between screens (using swipe left/right), setting the news site edition (swipe down on the homescreen) and going to the front page from all other screens (via two-finger swipe). To demonstrate some of the other features of jQMultiTouch, the application automatically adapts to landscape mode when the device is rotated and adjusts the content to fit different screen dimensions and resolutions. Most importantly, the implementation is based on a very lightweight skeleton that makes heavy use of images rather than complex HTML and CSS. We found that this would require less programming skill and, while still using many features of jQMultiTouch, would focus the students' attention on the rapid prototyping of interactions and multi-touch behaviours rather than other implementation details.

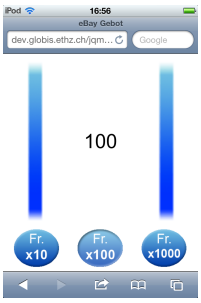
Finally, to further guide the design process, we encourage students to use the following method which we found useful for creating the FBTouch and TFlickr prototypes.

1. Basic touch enhancements
2. Extension of interaction model towards multi-touch
3. Alternative designs to meet user preferences and skills
4. Optional adaptations to meet special device characteristics

While the assignment was not mandatory and required to complete the course, it still attracted the interest of 8 groups with a total of 24 of around 50 students registered in the course. The most popular devices included the iPhone and Android-based phones HTC Desire, Sony XPERIA and Samsung Galaxy SII.



(a) BBC



(b) eBay

(c) Craigslist

(d) VIS Gallery

Figure 10: Student solutions based on the CNN application that range from simple modifications for the BBC web site, over experimental interfaces for eBay to more complex adaptations of existing web sites for multi-touch.

Students reported no major issues and most were able to test and build their solutions using their own touch devices. We show a selection of the submitted assignments in Figure 10. The first shown in Figure 10a is a variant of the example application which was extended for the BBC web site using simple gestures for flipping through different articles and browsing categories within the same screen rather than navigating between different pages. The second application shown in Figure 10b was created from scratch and not based on the example code we provided. It is not a complete implementation of the anticipated interface, but the general idea was to provide a multi-touch interface for bidding on auction platforms such as eBay by using multiple fingers to select and sliders to adjust the price in steps of 10, 100 or 1000 Francs. The third application is a more complete adaptation of the Craigslist web site for mobile touch devices (Figure 10c). Users are provided with a number of gestures to ease navigation between different categories and narrowing down the search results. The last application is in implementation more similar to the example we provided, but creates a whole new experience when translating the concepts to a photo gallery with multi-touch support as an adaptation of an existing student union web site (Figure 10d). In addition to flick left/right gestures for browsing through the pictures, users can also swipe down on a picture to download it to their device. The application also makes use of the layout orientation features provided by jQMultiTouch as the photo gallery shows more or less pictures in horizontal direction according to device orientation.

In general, the assignment was well received by students and led to a number of simple, yet interesting, solutions. The informal feedback concerning jQMultiTouch was positive and gives reason to believe that the framework is both of practical and research value. One student explained: "I found the

framework fairly easy to work with, but our group did not apply it in very much depth. Conceptually though, I found the framework easy to understand, and it seems capable of supporting projects of all different complexities."

DISCUSSION AND RELATED WORK

We have demonstrated that jQMultiTouch can cater for a wide range of applications and enable the rapid prototyping of multi-device/multi-touch interfaces. We have promoted a web-based approach to designing multi-touch applications that can run on different types of devices. Many of our examples, however, relate to mobile application development and therefore add to the ongoing debate on web *vs.* native implementations [3]. Proponents of the first argue for reduced implementation and maintenance cost, while advocates of the latter see benefits in terms of performance and interface design. One of the main benefits of jQMultiTouch is that developers can build on the web programming stack that they may already be familiar with and therefore only have to learn one method of specifying multi-touch and gesture-based interactions that is compatible with many different devices. While our specific focus with jQMultiTouch leaves out the widget support for emulating the look-and-feel of native mobile applications, it could still provide a complete development environment when integrated with other existing jQuery-based frameworks such as jQTouch. Moreover, while several works have contributed the design of a general multi-touch architecture, e.g. [6, 13], our solution seems more lightweight and direct since we leverage native browser support as much as possible. The current lack of support for tangible widgets in our framework could be mitigated by other techniques similar to CapWidgets [15].

From a more general perspective and given the examples in the paper, jQMultiTouch also provides new ways of adapting existing interfaces for touch and multi-touch. The adaptation of web sites to different devices is a popular topic in web engineering, but research has often aimed at fully automatic methods, e.g. for retargeting existing web interfaces to mobile phones [4, 10]. Other research has mainly looked at different models of user interface abstraction, e.g. CAMELEON [1], and model-driven approaches for generating interfaces adapted to different user, platform and environment contexts [2, 8, 18]. In particular, the authoring of adaptive and multi-modal user interfaces has been the subject of extensive research. However, the focus has tended to be on logical descriptions of user interfaces and the design of domain-specific languages rather than leveraging existing solutions [17]. Of the various existing approaches only MARIA [19] has included support for the new generation of touch devices, but this is limited to a mapping of concepts at the concrete user interface level. A critical goal of our work has therefore been to find more lightweight solutions that, in particular, build on only native web technologies, i.e. HTML, CSS and JavaScript, and integrate well with existing web scripting toolkits such as jQuery.

Our discussion addresses three remaining important topics with respect to the proposed framework: development effort, performance of applications and extension mechanisms.

Design Simplicity

It is difficult to carry out direct comparisons between implementations based on jQMultiTouch and other existing multi-touch frameworks due to fundamental differences. However, especially when compared to browser-specific code, the abstractions provided by jQMultiTouch lead to cleaner implementations and therefore add to the design simplicity for developers. In particular, the history concept with its query and evaluation mechanisms as well as the lookup table for active touches require less helper variables in event handlers because custom state can be attached to the touches or the history object and therefore be tracked more easily between callbacks. While this may not be so obvious from the simple code examples given in the paper, this has been recognised as a major issue [14] and can become particularly complex in larger applications.

Execution Performance

As already mentioned, jQMultiTouch has been used on many different mobile devices including iPhone 3G/4G, iPod 1G/2G, iPad 1G/2G, EeePad transformer tablet/notebook, as well as the TouchSmart all-in-one desktop computer. While it is difficult to cross-test all possible configurations and provide reliable data due to considerable differences between many of these devices and available browser implementations, we did not see major performance issues in terms of the multi-touch interaction on any of the devices. The simple picture viewing application from the first example executed on a modern smartphone performs almost as well as on the full-blown desktop PC showing very high refresh rates, but starts to drop in frequency the larger the images are scaled. However, this is a limitation of current browser support for CSS3 2D Transforms and not an issue related to our framework.

Scalability and Portability

While we can therefore argue that the current implementation of jQMultiTouch has the potential to scale across many different devices ranging from mobile phones to large interactive surfaces, we have to critically note that this may change if more browser vendors start to build on proprietary methods for touch event handling rather than aiming at standards. We therefore welcome recent efforts to create a W3C recommendation for common touch event models[12]. In the meantime, however, jQMultiTouch can provide a viable alternative and allow developers to build applications for a range of devices, as well as contributing an advanced framework for handling touch and other input data in a consistent way.

jQMultiTouch is available for download from the project web site[13]. We hope this encourages interested developers to experiment with existing framework support in their own applications and contribute refinements or new extensions building on the different mechanisms we have built into the framework for exactly this purpose.

CONCLUSION

In this paper, we presented jQMultiTouch, a lightweight framework for the rapid prototyping and development of multi-touch web interfaces that can run on many different devices.

[12] http://www.w3.org/TR/touch-events

[13] http://dev.globis.ethz.ch/jqmultitouch

We have shown how this framework was used to improve the interaction of existing applications on touch devices as well as for providing application-specific support for gesture-based modalities and multi-touch interaction techniques that go beyond basic zooming and panning actions.

In our ongoing research, we are building on the multi-touch framework in several projects to develop new methods for web interface adaptation as well as exploring novel touch and gesture-based interaction techniques especially useful in a web context. In addition, we believe that the extensible query-based input processing techniques presented in this paper can cater for other kinds of continuous input data, e.g. for handling 3D skeletal tracking data of Microsoft Kinect.

ACKNOWLEDGMENTS

We would like to thank Sai Swaminathan and Max Speicher for their help with the implementation of jQMultiTouch and some of the example applications. Special thanks go to the HCI class 2011 at ETH Zurich who also contributed several example applications. This work was supported by the SNF under research grant 200021_121847.

REFERENCES

1. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi- Target User Interfaces. *IWC 15* (2003).
2. Ceri, S., Daniel, F., Matera, M., and Facca, F. M. Model-driven Development of Context-Aware Web Applications. *TOIT 7*, 1 (2007).
3. Charland, A., and LeRoux, B. Mobile Application Development: Web vs. Native. *CACM 54*, 5 (2011).
4. Chen, Y., Ma, W., and Zhang, H. Detecting Web Page Structure for Adaptive Viewing on Small Form Factor Devices. In *Proc. WWW* (2003).
5. Dietz, P. H., and Leigh, D. DiamondTouch: A Multi-User Touch Technology. In *Proc. UIST* (2001).
6. Echtler, F., and Klinker, G. A Multitouch Software Architecture. In *Proc. NordiCHI* (2008).
7. Esenther, A., and Wittenburg, K. Multi-User Multi-Touch Games on DiamondTouch with the DTFlash Toolkit. In *Proc. INTETAIN* (2005).
8. Frăsincar, F., Houben, G.-J., and Barna, P. Hypermedia presentation generation in Hera. *IS 35*, 1 (2010).
9. Hansen, T. E., Hourcade, J. P., Virbel, M., Patali, S., and Serra, T. PyMT: A Post-WIMP Multi-Touch User Interface Toolkit. In *Proc. ITS* (2009).
10. Hattori, G., Hoashi, K., Matsumoto, K., and Sugaya, F. Robust Web Page Segmentation for Mobile Terminal Using Content-Distances and Page Layout Information. In *Proc. WWW* (2007).
11. Hinrichs, U., and Carpendale, S. Gestures in the Wild: Studying Multi-Touch Gesture Sequences on Interactive Tabletop Exhibits. In *Proc. CHI* (2011).
12. Kaltenbrunner, M., Bovermann, T., Bencina, R., and Costanza, E. TUIO: A Protocol for Table-Top Tangible User Interfaces. In *Proc. GW* (2005).
13. Kammer, D., Keck, M., Freitag, G., and Wacker, M. Taxonomy and Overview of Multi-touch Frameworks: Architecture, Scope and Features. In *Proc. EICS, Workshop on Engineering Patterns for Multi-Touch Interfaces* (2010).
14. Kin, K., Hartmann, B., DeRose, T., and Agrawala, M. Proton: Multitouch Gestures as Regular Expressions. In *Proc. CHI* (to appear).
15. Kratz, S. G., Westermann, T., Rohs, M., and Essl, G. CapWidgets: Tangible Widgets versus Multi-Touch Controls on Mobile Devices. In *Proc. CHI Extended Abstracts* (2011).
16. Laufs, U., Ruff, C., and Zibuschka, J. MT4j - A Cross-platform Multi-touch Development Framework. In *Proc. EICS, Workshop on Engineering Patterns for Multi-Touch Interfaces* (2010).
17. Nebeling, M., Grossniklaus, M., Leone, S., and Norrie, M. C. XCML: Providing Context-Aware Language Extensions for the Specification of Multi-Channel Web Applications. *WWW 15*, 4 (2012).
18. Niederhausen, M., van der Sluijs, K., Hidders, J., Leonardi, E., Houben, G.-J., and Meißner, K. Harnessing the Power of Semantics-Based, Aspect-Oriented Adaptation for AMACONT. In *Proc. ICWE* (2009).
19. Paternò, F., Santoro, C., and Spano, L. MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments. *TOCHI 16*, 4 (2009).
20. Scholliers, C., Hoste, L., Signer, B., and Meuter, W. D. Midas: A Declarative Multi-Touch Interaction Framework. In *Proc. TEI* (2011).
21. Shen, C., Vernier, F., Forlines, C., and Ringel, M. DiamondSpin: an extensible toolkit for around-the-table interaction. In *Proc. CHI* (2004).
22. Signer, B., Kurmann, U., and Norrie, M. C. iGesture: A General Gesture Recognition Framework. In *Proc. ICDAR* (2007).
23. Wobbrock, J. O., Wilson, A. D., and Li, Y. Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *Proc. UIST* (2007).
24. Wu, M., and Balakrishnan, R. Multi-Finger and Whole Hand Gestural Interaction Techniques for Multi-User Tabletop Displays. In *Proc. UIST* (2003).