Point, Click, Tap, Touch

Building Multi-Device Web Interfaces

Boris Smus
Rick Byers
Most mobile devices have touch.

Mobile use projected to overtake desktop use by 2014.

~25% of Windows 8 laptop sales have a touchscreen.


Source: Display Search [http://goo.gl/5OhKS](http://goo.gl/5OhKS)
The web has a powerful touch event API

touchstart, touchmove, touchend, touchcancel
(eg. scrolling on android).

Touches on the screen have consistent identifiers.

Works for mobile as well as touch on desktop.

Good support across modern mobile browsers (iOS, Android, etc).

Most touch devices support 10+ simultaneous points.

DEMO
Design suggestions
Tip 1: Touch interfaces require bigger targets

1. Physical size ≠ device pixels (pixels per inch)
2. Device pixels ≠ CSS pixels (device pixel ratio)

<table>
<thead>
<tr>
<th>Device</th>
<th>Calculation</th>
<th>Size in px</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nexus 4</td>
<td>( \frac{1}{(\text{device pixel ratio}) , (\text{dpi}) , (\text{inch per mm}) , (\text{phys size})} = \left(\frac{1}{2 , \text{cps/dpx}}\right) \left(320 , \text{dpx/in}\right) \left(\frac{1}{25.4 , \text{in/mm}}\right) \left(9 , \text{mm}\right) )</td>
<td>57 px</td>
</tr>
<tr>
<td>Nexus 7</td>
<td>( \left(\frac{1}{1.325}\right) , 216 , \left(\frac{1}{25.4}\right) , 9 )</td>
<td>58 px</td>
</tr>
<tr>
<td>Nexus 10</td>
<td>( \left(\frac{1}{2}\right) , 300 , \left(\frac{1}{25.4}\right) , 9 )</td>
<td>53 px</td>
</tr>
<tr>
<td>Chromebook Pixel</td>
<td>( \left(\frac{1}{2}\right) , 239 , \left(\frac{1}{25.4}\right) , 9 )</td>
<td>42 px</td>
</tr>
</tbody>
</table>

Optimal touch target size range is 42 to 58 px.
Tip 2: Don't rely on hover

CSS hover (:hover), JS hover (mouseover/mouseout)

Touch has no true hover state, but browsers fake it on tap.

Avoid hover, especially with underlying links.

DEMO
Touch laptops enable new interactions

Responsive input

Maps

Object transforms

Source: smus.com
Avoiding Common Problems
Problem 1: Assuming touch support implies no mouse

```javascript
if ('ontouchstart' in window)
    element.addEventListener('touchstart', activate);
else
    element.addEventListener('mousedown', activate);
```

Breaks mouse input on touchscreen laptops!

Example
Solution: Listen to both mouse and touch events

Call `preventDefault` in the touch handler to avoid redundant mouse events.

```javascript
element.addEventListener('touchstart', activate);
element.addEventListener('mousedown', activate);
function activate(event) {
    ...
    event.preventDefault();
}
```
Solution: Enable touch event support to test

You can test that you haven't broken mouse support by enabling the flag:

```
chrome://flags/#touch-events
```

**Enable touch events** Mac, Windows, Linux, Chrome OS
Force touchscreen support to always be enabled or disabled, or to be enabled when a touchscreen is detected on startup (Automatic, the default).  

We hope to turn this on by default when more sites fix this bug.
Problem 2: Touch event targeting isn't the same as mouse

**MouseEvent** targets the element under the cursor

1.mousedown
2.mousemove*
3.mouseout
4.mouseover
5.mousemove*
6.mouseup

**TouchEvent** targets the node where the touch started

1.touchstart
2.touchmove*
3.touchend

Even when it's moved or removed!
Solution: Put handlers directly on the touched element

Necessary only when it could be removed or moved in the DOM

element.addEventListener('touchstart', function(event) {
  ...
  event.target.addEventListener('touchmove', onMove);
  event.target.addEventListener('touchend', onEnd);
  event.target.addEventListener('touchcancel', onEnd);
})

Demo element removal
Problem 3: Making it harder to hit small targets

Touch center point too imprecise for targeting

Chrome provides "touch adjustment"
On gestures (tap, long-press, etc.)
  ● Score all touchable elements under the finger
  ● Adjust position to the most likely target

Touch events themselves are never modified

Demo
Solution: put click handlers on each tappable element

For touch adjustment to work properly:

1. Activation must rely on 'click'
   (or 'contextmenu', 'mousedown', 'mouseup', or :active)

2. Each activatable element must have its own event handler
   (or other signal indicating it's tappable)

Extra effort is required for #2 if you rely on event delegation
Demo: Gmail star
Problem 4: Gesture APIs are browser specific

How to detect pinch, rotate, etc?

GestureEvent

MSGesture

Not another proprietary gesture API!
Solution: Rely on libraries for gesture detection

Several libraries doing this well cross-browser today, eg: **Hammer.js**

```javascript
Hammer(element).on('transformstart', function(event) { ... })
Hammer(element).on('transform', function(event) {
    update(event.gesture.scale, event.gesture.rotation);
})
Hammer(element).on('transformend', function(event) { ... })
```

**DEMO**

Many others, eg:

- **TouchSwipe** jQuery plugin
- **Touchy** jQuery plugin
- **QUO JS**
- **Deeptissue JS**
Performance
Problem 5: the click event is delayed on mobile devices

Implemented for double-tap-to-zoom gesture.

Approx. 300ms delay of click event on most touch-enabled browsers.

Causes pages to feel slow or unresponsive.
Solution: use a good fastclick library

Fastclick libraries listen for `touchend` instead.

In Chrome desktop, no click delays at all.

In Chrome for Android, no delays for fixed viewports (`user-scalable=no`).

Make sure your fastclick library knows that!
(eg. [https://github.com/ftlabs/fastclick](https://github.com/ftlabs/fastclick))
Problem 6: touchmove can fire very quickly

Often much faster than 60 Hz (render speed)

Movements of many fingers may get coalesced into one touchmove event, but very platform-dependent.

Impact varies depending on browser. DEMO
Solution: avoid expensive operations in event handlers

Do not re-render `event.touches` array on touchmove.

Store `event.touches` and use `requestAnimationFrame`.

```javascript
function updateTouches(event) { touches = event.touches; }
document.addEventListener('touchmove', updateTouches);
window.requestAnimationFrame(renderEverything);
function renderEverything() {
  // TODO: Render code goes here.
  window.requestAnimationFrame(renderEverything);
}
```
Problem 7: Touch handlers can cause scroll jank

Jank-free smooth touch scrolling is critical to engagement!

Chrome tries to scroll on the GPU thread, but event handling on main thread.

If there is a touch handler, scrolling must wait to see if `event.preventDefault` is called.

```js
function reallyFast(event) {}  
document.addEventListener('touchstart', reallyFast);  
document.addEventListener('touchmove', reallyFast);  
```
Solution: Avoid unnecessary use of touch event handlers

Ask yourself: do you really need that touch handler?

By default, touch input generates common DOM events. 

click, scroll, contextmenu all fire.

Touch also sets CSS pseudo classes like :active

You don't always need to implement touch-specific event handlers.
Solution: Keep touch event regions small

Ask yourself: does your touch area need to be so large?

Chrome keeps track of which parts of the page have touch event handlers.

For each part of your page, decide between:
- smooth scrolling OR
- touch event handling

Don't add touch handlers to the document or body!
Future of touch on the web
Goals for future directions with touch on the web

Reduce the need to use touch events directly
- eg: Touch support for HTML5 drag-and-drop APIs

Give developers more control over browser default behavior
- eg. customizing scrolling behavior so you don't have to reimplement scrolling yourself in JavaScript

Better cross-browser support
- Working with Microsoft to standardize some of the touch features from IE10
Standardizing input model from IE10. Key design points:

- abstraction and extensibility
- touch extends the MouseEvent model
- touch behavior specified declaratively

Discussion: public-pointer-events@w3.org

Microsoft built a prototype for chromium
Beginning experimental support in Blink

Try one of the early pointer events polyfills:

- hand.js - handjs.codeplex.com/
- Polymer PointerEvents - github.com/polymer-project/PointerEvents
- Points.js - https://github.com/Rich-Harris/Points
Some benefits of pointer events

Scroll jank impossible - specify desired behavior declaratively:

- **touch-action: none**
  - touch drag doesn't scroll
  - get all events

- **touch-action: auto**
  - touch drag scrolls
  - still get 'down' event
  - on scroll get 'cancel' event

Code sharing demo

- [Mouse+Touch](#) (131 lines, 56% shared)
- [Mouse+Pointer](#) (89 lines, 97% shared)
Conclusion
Make your site a joy to use with touch!

Tell us about the problems you have and what we can improve!

Resources
Come talk to us in the Chrome 'Questions' bar after the talk
Touch Events: www.w3.org/TR/touch-events/
Touch event discussion: public-webevents@w3.org
Pointer Events: www.w3.org/TR/pointerevents/, Learn more
H5R Article: www.html5rocks.com/en/mobile/touchandmouse/
Rick's G+ stream for touch issues and questions: www.rbyers.net/plus
Dump events test page: www.rbyers.net/eventTest.html
Thank You!

Please submit feedback: http://goo.gl/wuvkR

rbyers@google.com  http://smus.com
http://rbyers.net/plus  google.com/+BorisSmus
@RickByers  @borismus
Solution: Emulate mouse targeting with `elementFromPoint`

```javascript
element.addEventListener('touchmove', function(event) {
  ...
  var touch = event.targetTouches[0];
  var over = document.elementFromPoint(touch.clientX, touch.clientY);
  var last = lastover[touch.identifier];
  if (over != last) {
    last.dispatchEvent(makeEvent('my-touchout'), ...);
    over.dispatchEvent(makeEvent('my-touchover'), ...);
    lastover[touch.identifier] = over;
  }
});
```