Engineers from MSIE, FF, Safari, and Chrome met on the Microsoft campus on June 23rd, 2014 to brainstorm touch on the web and discuss platform direction in general.

Takeaways: main opportunities for further collaboration

- Lots depends on speccing viewports
    - AI: Tab, Matt
    - Device adaptation: AI Sam to let us know if they can comment on it
- Exposing visual viewport
    - device-fixed, IE team to describe their behavior
    - AI: bokan to share proposal to www-style
- Future of touch events / pointer events
    - need rational API for all input devices that we can iterate in standards organization
    - Google proposes exploring extending touch events with all the good properties of pointer events - no consensus on this (IE and Firefox teams feel pointer events are still the better path)
    - Ted: like the look of touch-action
    - Sam: don't agree with the use case having one model for all events
- Hover/active rationalization
    - AI: Jacob to try to define what is done today, Zeeshan to add Chrome's
- Scroll response effects
    - Chrome continue to experiment to see how far they can get without new APIs
    - T(Apple) expressed interest in discussing input-linked animations in Fx taskforce
    - Infinite scrolling
- Fractional scroll offsets
    - IE to try turning it on by default to see what's breaking
    - Safari have fractional layout on in trunk but breaking
    - Chrome trying to enable fractional scroll offsets

Attendees

- Microsoft: Jacob Rossi, Matt Rakow, Justin Rogers, Matt Esquivel, Alex Woolf, Rick James, Matt Kotsensas, Andrew Bunn, Li-Hsin Huang, Keshav Puttaswamy
- Google: Rick Byers, Max Heinritz, Dave Bokan, Gary Kacmarcik, Tim Dresser, Tab Atkins, Alex Elias, Zeeshan Qureshi, Yufeng Shen
- Apple: Sam Weinig, Ted O'Connor
- Mozilla: Matt Brubeck

Schedule

Time	**Topic**
9:30 – 10:00	Introductions, Agenda Bashing

10:00—10:30 Pointer / Touch Events Sync
10:30—11:00 CSS :active, :hover, :focus with touch
11:00 – 11:30 Viewports: review the status quo
Get everyone on the same page as to what each browser currently does
11:30—12:30 Viewports++: where do we want to go?
Open discussion on the good parts of each implementation, problems we still need to solve, top priorities for the coming year
12:30-1:30 Lunch
1:30 – 2:00 On-Screen Keyboards (moved from 5:00)
2:00 - 3:00 Zoom Related APIs
Device-Fixed, zoomTo(), Coordinates (conversion APIs, floating point precision, etc), sub zoomers
3:00-4:00 Scroll Customization APIs
Snap Points, Scroll Blocking behavior, Customizing Physics,
4:00—4:30 Break
4:30 – 5:30 Scroll Response APIs
Events during/after scroll, scroll event triggers, scroll synchronized effects, fractional offsets, input driven animations
5:00 - 5:30 Takeaways
5:30 - 6:30 Travel
6:30 – 8:30 Social Event (Lucky Strike)

Introduction

Rick's overview of Blink priorities

- Why?
  - Interoperability is the web's killer feature
  - We are driving developers to proprietary platforms one incompatibility at a time.
  - Can we accelerate the pace and quality of improvement of the platform (more general) by focusing on input?
- Proposal for today
  - Explain priorities
  - Look for overlap
  - Follow up afterwards
    - Don't expect to agree on designs today
  - Primary goal: Look for common ground
- Blink input principles
  - Mobile first
    - Plan for world where most internet access is from phones and tablets
    - We've won desktop, need to compete on mobile
  - Performance

        - Goal: smooth, predictable, low latency, ...
        - A few years ago we were focus on the thread
        - Now we're focused on dejanking the main thread… seeing the other thread as a crutch
    - Richness
        - Close the power gap with native mobile platforms
            - Make it less painful
        - Focus on use cases to drive what we need to do (e.g. pull-to-refresh)
    - Rationality
        - Must be extensible, layered, understandable, incremental
        - We want the obvious things to tend to be correct
        - Primitives first:
            - Ie: implement the kernel before you implement the library

Discussion

- Jacob's comments:
    - Overall, this resonates with the IE team
    - MS's biggest goal is to get all users on current, up-to-date IE
- Rick(Google): Previously we were focused on bringing touch to all devices including laptops, now we're focused primarily on phones/tablets
- Jacob: MS still cares about touch on laptops
- Rick(Google): We care about resolving interoperability and having a rational platform, and if that requires changes on desktop we'll still want to do that. Mobile-first is generally subservient to rationality
    - Don't want there to be a mobile platform and a desktop platform
- Sam(Apple) How do we define mobile? Rick(Google): phones and tablet
- Sam(Apple): we aren't convinced that the future is so clear, we might eg have convergence between mobile and desktop
- Rick(Google): We don't have to agree on this, but we just want to explain where we're coming from
- Alex(Google): One thing that's notably different about mobile: performance expectations are much higher on mobile devices, we input, physics-driven effects
    - Optimizing for latency - could apply to any platform, desktop, TV-browser, phone
- Matt(Moz): Need to drop assumptions to account for mobile
    - Not everyone is going to have a 2G machine
    - Sam(Apple): iPhone is just as powerful as older laptops
    - Not all phones will be like this
- Sam(Apple): Apple is **not** assuming that phone will be low powered
    - The web should bridge the divide between desktop and mobile. It shouldn't widen the divide.
- Sam(Apple): We're interested in enabling scroll effects, enabling people to make those effects fancy scroll effect easily, snap points as an example. We want to enable people to make scroll effects without breaking the threaded model

Pointer vs Touch Events

- What's the problem?
    - Devs want same code for input handling desktop and mobile
- Need to tweak behavior for stylus
- Android benefited from a single input API - MotionEvent
    - Apps written assuming touch mostly worked fine when stylus support was incrementally added. Would have been nuts to require all apps to be rewritten to handle stylus.
- Handling mouse and touch input requires more lines of code
    - writing mouse + touch is duplicating much of the core code
    - writing mouse + pointer can share more code
- Pointer Events in Blink?
    - Two years ago it was very attractive
    - Threaded scrolling everywhere (never blocked by JS)
    - But it doesn't fit with current priorities:
        - Mobile first
            - Touch events are here to stay on mobile
        - Fast by default
            - Pointer events have hit testing and double event dispatch burden.
            - Hit testing takes 2.5% of frame time. We're try to eliminate these types of per-frame costs on the main thread.
        - As rich as native mobile platforms
            - Mutual exclusion of touch and event handling
    - Growing consensus at Google is that Blink shouldn't invest in Pointer events at this time. Not ruled out, but no plans to implement.
    - We feel must invest in incrementally evolving Touch Events
        - This is controversial… web devs want Pointer Events
- T(Apple): The motivating scenario given earlier could be addressed by improving drag events
- R(Google): We see draggable interface as sugar on top of some other underlying touch input primitives
    - Off-thread input event handling: not our focus for the next 12 months
- S(Apple): If you design for direct manipulation (ie touch input) you should write different code than if your were designing for indirect manipulation (ie mouse input)
    - Apple had touch events generate mouse events, but this was for compatibility since there was not a lot of touch-enabled content at the time.
    - For the same site: some components designed for direct manipulation and some designed for indirect manipulation
- Problem: many websites query for touch support
    - if (touch-available) then support-touch-events else support-mouse-events
    - They should listen for both types of events.

    - But they turn off mouse event support because they are not sure if they are real mouse events or synthetic.
    - This is a problem for touch-based laptops
    - How many sites are broken in this way?
        - Don't have good data, but anecdotally the situation is getting better.
        - MS tested: 10 / 100 top sites in last month had this issue
    - Feature detection is hard and currently imprecise / not well-defined
    - MS: we want the web to work no matter what input device you have (we only want people to register for one set of events)
        - But the input type may change over time within one session
        - Regardless of whether we're all going to support pointer events, we need to offer a way to determine which set of events to listen to
- Which engines support pointer events
    - Metro FF has implementation behind a flag
    - Trident ships it
    - Google has a polyfill as part of Polymer, with native support for touch-action in Chrome 36+
- Blink: discussion around Pointer / Touch Events is in the context of “this is good for the web” and “we want to be good web stewards while preserving good relationship other people invested in pointer events”
- Brainstorming
    - Should be easy to polyfill the differences of different event models
        - Question: Cost of polyfill vs. double dispatch
            - Polyfill should be close to native
                - Current issue (recursive hittest through Shadow DOM elements) in Blink is a bug
                - This is controversial… amount of encapsulation provided by Shadow DOM is not agreed upon
    - Touch-action is a huge first step
        - Now shipping in Chrome 36
        - Use cases
            - Control the 300ms click delay (touch-action:manipulation)
            - Side swipe (touch:pan-y)
            - Polyfills
            - Declarative signal about intent to engine
    - Fix problems hit by Polymer
    - Set/release capture for touch/mouse?
    - Allow sending Touch Events for mouse?
        - Mozilla has a dev flag to enable this so that devs can test their code on non-touch enabled devices.
        - Chrome dev tools does this
- The kernel’s job is to expose the power, the framework’s job is to be the "pit of success"
- A tell for the native vs web: checkboarding vs jank

    - Developer should have control
- S(Apple): native apps have a lot of people testing on their users' platform
    - Web developers have a wide range of skills: amateurs to professionals
        - Tendency to test only on their main platform; little to no testing on other web platforms
        - Web platform tries to prevent devs from making problems
    - The kernel will expose footguns. We don't agree that's good.
    - As engine developers we don't let people shoot themselves in the foot
- People create rich experiences on the web, to get it to scroll, we need to get the page out of the way
- Change style and manipulate the DOM => (N^2 algorithm). developers
- Perhaps CSS should be explained as a set of primitives
    - T(Apple): Interesting science experiment
- Max: Imagine what we're doing as an onion
    - Outer layer = framework API. we're unlikely to find agreement
    - Inner layer that enables the frameworks, perhaps we can agree here
- Every other successful platform...
    - Rick(G)...ships a framework with the primitives
    - Sam(A) ...but they also have a unifying vision
- Jacob(MS)
    - When you have a higher-level API, you can bring the API to the new platforms easier
- A(Google)
    - Why we have gone from thinking about adding sugar to adding primitives: we've found the sugar doesn't cover all the use cases we thought it needed to
    - "position: sticky" doesn't cover all the use cases we thought it would
        - S(A) different problem: Fixed and Sticky work terribly with Zoom
    - There can be a specced model, but we don't have to expose the model through an API
    - Pull-to-refresh: we cannot capture the entire set of things people want to do with some simple sugar: the space is too complex
- Need to be able to specify where on the page you want events

CSS :active, :hover with touch (11:15)

- When exactly to fire :active or :hover?
- Do we want to spec it?
- IE does multitouch active and hover single focus
    - Only primary finger can set focus, all other fingers can set active and hover
- When you lift, does it become hover only (for hover menus)?
    - Webkit/chrome yes, IE no
- Hit testing isn't really specced. CSS defers it to HTML, HTML doesn't really define it

- Motivating use case: when you move off of button, it should be not :hover
    - http://ie.microsoft.com/testdrive/Browser/TouchFirstControls/
- Next step: create a survey of what everyone is doing with :active and :hover and then find the intersection and take discussion to CSS WG
    - This will also reference the non-existing hit-testing spec
- Sam to ask Ben to share browser-engineer-to-engineer what the behavior of the Touch Control stuff is

Viewports (11:25)

- Matt(IE): Viewport defined in CSS 2.1 spec to be both the visual and layout viewport - doesn't take pinch-zoom into account
    - Two viewports: visual and layout (different terminologies across teams)
    - Determine which one is being referred to at each reference
    - Focused on keeping buttons fixed on the right side
    - Great demo: http://lostworldsfairs.com/atlantis/
        - Chrome is currently broken, except ChromeOS M36+ (visual viewport) is mostly well behaved
    - The model is the fix content is being scaled, but attachment is to top left of the layout viewport
    - A useful mental model is a magnifying glass
    - Scrolling moves the visual viewport first, then the layout viewport when the boundary of the layout viewport is reached
- Dave(Google): Slide deck
    - Currently: fixed position elements are tracked as you zoom in
    - Philosophy: not exposed to the platform in any way
    - Browser zoom changes device pixel ratio
        - window.innerWidth/Height does not change
        - Browser feature - mobile sites shouldn't need to zoom
        - Pinch does not change device pixel ratio
    - If you want a more featureful pinch experience, build it yourself
    - Longer-term we should expose both scale and X and Y offset
        - MS currently fires resize events and page x / y offset
    - We may need to expose something to the platform to ship on Android
    - For on-screen keyboard
        - By default, keyboard pops up and changes the visual viewport
            - Same behavior as MSIE (and same for other page overlays)
        - Going forward, we'd like to give the app more control over the on-screen keyboard
    - S(Apple): WebKit forces fixed elements out of normal constraints to make them more accessible

        - Using 2 viewports addresses this problem since the visual viewport can scroll into view. (so static elements at the bottom of the pages aren't stuck behind the virtual keyboard)
    - WebKit only has once conceptual viewport
        - But fixes just one side
    - MS introduce “device-fixed” element fixed to visual viewport for UI bars attached to bottom
        - They reference the visual viewport
        - They do not scale when pinch
        - Needed to rationalize coordinate space
        - Jacob to share MSDN on device fixed
        - Example: toolbar above the virtual keyboard
            - toolbar doesn't scale (just like the keyboard doesn't)
            - content in view does scale.
        - The API allows you to animate the element with the keyboard when it pops up
        - Keyboard could cause relayout
            - e.g., for 100% height items.
        - MS: We exposed all the APIs needed to do this in JavaScript
        - The scrolling is happening on one thread and the UI element is overlayed and drawn on another thread
    - Dave Bokan to share his written proposal for viewport behavior
    - Sam: Problem with exposing primitives here: high-frequency events is the wrong way to do animations, want to tell compositor about synchronized animation curves
    - Ideally not expose pinch to any current properties and expose new properties for visual viewport
    - We could provide an onVisualResize event
        - No event-by-event update
        - Apple: you want to tell the compositor in advance
    - Coordinate spaces we’re working in and how to convert between them
    - Alex(Google): we wanted to converge on CSS pixels in floating point, avoid physical pixels
    - Google is shipping fractional touch event coordinates in existing API - see existing sites get smoother for slow-motion drags
    - -ms-resize event fired before the portrait-landscape orientation to give developer and opportunity to change things before relayout fired by the orientation change happens
    - Do we do device-fixed or expose some underlying primitives? The latter comes with footguns.
        - AI: rbyers to start thread on blink-dev
    - Off-thread input handling

        - Having a separate thread that can handle input is helpful for filtering events from lots of different streams
        - What's the name for this? "shader" small bit of logic that generates some output
    - Mobile web gets page offset, innerHeight/innerWidth, etc
        - We expose different coordinate systems
        - We'll break sets that rely on these things changing
        - Ideally, we'd expose a different zoom model
- Device adaptation spec
    - MS wants this
    - Rick(Google) agrees we should do this to rationalize a key platform property
    - Should we be specifying what metaview port gives
        - Sam: yes
        - AI: Sam to check if Apple can talk about device adaptation
            - Agree device adaptation should apply to all devices
- How does MS show fixed position elements? Listen to resize, check visual viewport size

Lunch (12:15)

- Blink: we want to make the main thread faster so developers can implement anything they want
- But we should keep the fast threaded model for complex pages with layout
- Web apps are not the not normal, web pages are the norm
- We will pursue both at once
- Apple's concern is that we are driving API design based on this future vision that we don't really know is possible
- Sam: De-janking the main thread is the standard work that we're doing, and after 5 years it's still not good enough for 60fps
- Do you start from use cases? Or start from exposing the metal?
- Platonic primitives sound right
- How do you measure touch latency in a cross-browser way
    - Sam: We should build a cross-browser benchmark test suite - if Chrome is really doing much better, shame the other browsers into catching up
        - This is the game we play and it's a good game
    - Moz uses HDMI capture card
    - Google just posted details of their end-to-end camera testing, but rely mainly on chrome instrumentation and test-only APIs for automated latency testing
    - Apple tried camera tests but found them not super actionable

On-Screen Keyboards (1:20)

- ZeeshanQ's slide deck
- Use case
    - Polymer page search box: you get the blinking cursor but no keyboard
    - On tap on magnifying glass, the input field focuses programmatically, and can't show the OSK because the focus callback isn't a user gesture
    - Lots of complexity here
    - MS has some things… if the last input was touch and on an editable field, then you can programmatically focus the input box
- Don't want to show keyboard if there's autofocus on load
    - Metro on desktop or tablet is the best implementation from MS
- jQuery has some interesting behavior where it tries really hard to focus
    - Set up a thread with jQuery folks
- Behavior should not be specific to the convertible vs non device
- Detect virtual vs. physical keyboard
    - Keyboard object in Window.Navigator?
    - Keyboard.isVirtual()
- Developers want to kick off animations and then show OSK
    - Perhaps Keyboard.show() Keyboard.hide()
    - show/hide too easy to abuse (be spammy)
- Use case: knowing that something is obscuring your content
- Two separate reasons page may focus today, no way to differentiate:
    - Because there's nothing else for the user to do (e.g. search / login page onload)
    - Because you want to draw the user's attention to where key strokes will go even if it's not the primary use age (eg. shopping site)
- Focus when there's a physical keyboard, but not for virtual keyboard
- MS's model
    - focus shows keyboard anytime after the user touches page (ie last input was a touch… needs to be a tap at the system level)
    - Don't want to be spammy with the keyboard
- What's the use case that's most important?
    - focusing an input field after an animation
    - User event cohesion… defining user gesture more precisely
- People want the show and hide apis for the wrong reason
    - Oftentimes this breaks accessibility
    - There are a few more complex resource
    - How do you keep selection when you press a button
- What about the search onfocus on page load
    - e.g. login screen
    - keyboard.show would be abused
    - Win8 focus with the off-screen search box is actually offscreen

- Things that are easy to do and spammy, people do. Things that hard to do and spammy, people don't do.
- Win8 has two different apps open at the same time, split window
- What's the difference between native apps and the web in this case?
    - The web is untrusted, iOS apps have a review process
- Allow customization of action button according to customization
    - Just an attribute, doesn't require programmatic api
- some apps want to resize when the keyboard pops up
    - There'd be a preventDefault, you could emulate that with a scrollable div iframe, the keyboard will still show
    - You'd go through the whole list
    - height media query doesn't fire up on
    - Sam: super into knowing when something is obscuring a lot of your content
        - having willShowKeyboard event makes sense
        - could attach animation curve
- Eastern european field pop up their own keyboard instead of using system virtual keyboard
- Takeaway: Chrome to consider changing heuristics to match MSIE, propagating user gesture tokens across animation end, consider speccing it

Zoom Related APIs (2:00)

- Matt Rakow
    - mscontentzoomfactor … lets you declare any piece of content zoomable… you can zoom in on it and the surrounding content doesn't zoom… called "zoomers"
    - usecase: Bing maps, though they didn't end up using it
    - subzoomer can go beyond 100% zoom
    - Bing maps want rotation and multizoom
    - Store product images want to be zoomable
- overflow scroll is similar, but MSIE doesn't let you zoom on non-scrollable elements
- what happens after you lift your fingers? Does it snapback?
- Semantic zoom? Zooming in lets you see more detail
    - Additional viewports that are within a page
- zoomTo
    - Offer authors the ability to programmatically zoom in a particular part of the content
- Google hasn't heard too much about this use case from web developers
- Seems like very useful for the MSIE Metro UI
- Pinching out feels weird compared to Keynote or Powerpoint
- Sub-scrollable, just changing the transform on the element itself, not the viewport
- "zooming" or "stretching"
- The region-level primitives: zoomable and scrollable

- We could plumb it through the double tap zoom code for full-page zoom
    - sub-zoomers would be more work
- ZoomIntoView API - agree it could make sense
    - can be used to explain zoom to text field
- **Most interesting question: direct use cases**

Break (2:30)

Scroll Customization APIs (2:40)

- Sam: Sub-scrolling isn't very common, often instead doc scrolling with fixed header
    - Alex: but that breaks scrollbars
    - Sam: on Cocoa (Mac/iOS) we rely on content inset rects for that
        - eg. header may be slightly transparent
        - also obscured rects to define where events go
    - The scrollbar on news.google.com
    - subscrollable region may not be sufficient
    - You may want to have a transparent fixed region at the top of the scrollable region… the scrollbar shouldn't appear on this region, so web devs should be able to define a scrollbar inset
        - It's a Mac concept, not on the web
- See scrolling split between controlling the inputs to scroll offset and responding to the outputs
- Things that are customizing the scroll offset
- Objectives
    - All scroll-driven effects possibles in native apps on the web.
    - Preserve threaded-scrolling in the common cases
    - Ease of development as a third priority, lower than the other two.
- Some examples
    - Pull to refresh
        - eg: Twitter, Facebook app: pull/swipe down at top of screen to get new content
        - There's a friction that affects the scroll behavior
        - There's a lot of possible ways to do this
    - Hidey bars
        - Bar that scrolls along with content
        - May stay visible or auto-hide depending on how much of the bar has scrolled off the screen
- Pull-to-refresh requirements
    - Drive an arbitrary main thread effect on overscroll based on the overscroll distance

        - Pulling down a header
        - Expanding a refresh indicator horizontally
    - Transition between threaded scrolling and main thread effect needs to be smooth
        - Considering changing the disposition of the first touchmove
        - Need to be able to alternate freely
        - No missed frames
        - Every frame must know the precise overscroll offset
        - We can never guarantee not missing a frame
    - Fling in and out of the main thread effect
        - Requires knowledge of scroll velocity at the transition point
        - Need the ability to initiate a fling following the platform defined physics.
- Hidey bars requirements
    - The bottom of a top-anchored hidey bar must stick to the content
    - Demo with red and green box
        - http://tdresser.github.io/sync-scroll-offset-test/no-raf.html
        - When there's a scroll event handler, expand the frame budget a little bit
        - Scroll event comes in right after rAF
- Touch Prediction
    - IE Predicts where the finger is going to be to reduce latency, for both scrolling and raw pointer events
    - In Windows Metro apps there is a pointer.point interface… MSIE api for getting the unpredicted coordinates
    - Causes some rubberbanding
    - Painting apps haven't complained to MS
- This entire approach depends on the main thread being not super janky
- Windows Mail app is HTML
    - Lots of requests coming in and janking and blocking on the main thread
    - No amount of tuning is going to fix this jank on the main thread
    - We're burdened by the legacy of the web, but for these new effects we're focused on new apps
    - MSIE uses this to motivate the need for another thread
- Google: We've been using this to drive some of our perf improvements
- Triggering a fling on the page during a pull to refresh, how to tie the main thread behavior to the off thread
    - On iPhone this would be hard… e.g. many inputs: finger size, how you're holding the phone, ...
    - There are a lot of benefits to not hitting the main thread… could we just allow web devs to declare this behaviour upfront and offload it entirely to the off thread?
    - Tim tried with CSS animations and it didn't work
- Sam: interested in a way for pages to disable scroll rubber banding
- Linking Web Animations to scroll effects?
    - Apple: not clear if the whole Web Animations API is needed

    - Tie certain inputs to scroll position
    - A threshold when you get an event?
- scrollLimits is intended to constrain scrolling… might be useful
    - scrollOffsets need to give negative offsets
    - Can't control the friction, but you can imagine an API that lets you control a curve
    - 99% use case: stretchy or bouncy or no-effect or trampolining
    - Snap Point API
- What do web developers want? They want…
    - (a) their page to feel like the platform
    - (b) to be able to define their scroll effects
- We want to unlock innovation on the web
    - Pull to refresh was invented on native platforms because the platform allowed for this innovation
        - They were able to subclass the UI scroll event
    - "position: sticky" was the opposite…
    - Ted(Google): css property auto with timing function could help
        - Rick(Google): this is a composable lower-level abstraction, we agree
- Twitterific was a targeted at a specific device and a specific OS, tested again that
- Seems possible on the web platform today: turning off effects, then get notifications on the main thread, you could do transforms
    - Hard part is when the page has been scrolling for a
    - Send touchmove (cancellable = false), only after the overscroll, touchmoves are synchronous. With the blue glow the hand off is not noticeable
- Scrollbar customization
    - App wants scrollbars to be consistent
- Sam: people want a first order API to do... something related to position: sticky

Scroll Response APIs (2:55)

- TDresser's slide deck
- Things that happen in response to the scroll offset
- At scroll position X, something should happen
- element visibility, scroll visibility pseudoclass event
- media queries aren't sufficient: might want different behavior when scrolling up vs scrolling down
- Attaching an "inview" thing to the element?
- will-change = "go fast" just like "ztransform"
- Ripe for experimentation by vendors
- To help with infiniteScroll, engine might have some idea about which elements will reach the bottom of the page: will-reveal-bottom, will-reveal-top
- scroll event listener verse an event that fires a specific point during the scroll
    - The Verge has 400 scroll listeners…. this is shitty

        - Scroll listener to load resource when user reaches scroll point
    - You might be able to get closer to the point
    - In a tech mashup world, having a single handler is not possible to make this possible… this is why the “fire at a specific point” API
- Async dispatching the scroll event isn’t that bad from Blink’s perspective
    - Post every 250ms or something? Or every frame?
    - There are some sites that update stuff too aggressively.
- Could we add velocity/prediction information, current delta, etc to the touch events
    - If you hit a limit (e.g. overscroll), how far did you after that limit
- MS added an API for inertia, to get the target of the scroll
- At a certain point you will reach unpainted content
- scroll event that you’re listening to, maybe we can create a more specific event type or behavior… maybe something to reveal when something is in view
- The rAF callback is called every frame during scroll, but scroll doesn’t block on rAF
- Use cases: fade in images, scroll things into view
- Share proposals for infinite scrollers
- Could we introduce a notion of interrupting events to the web platform?
- What you want as a developer is the ability to say “fill in unpainted pixels with this content” … on iOS native this is done with higher-level constructs like UITableView
- Carouseling scrollers
    - No way to build this today.
    - msn.com does this.
    - Problem comes when there’s a success set of quick flicks
    - Reused and not repopulated
- Sam(Apple): the web is totally lacking in these two abstractions: UITableView and Carousels
- The primitive route: let web devs implement something like UITableView themselves, or in frameworks
- The nonprimitive route: give devs some ability to pick the filler content
- Sam: intentionally don't fire scroll events every frame - lots of unnecessary battery churn, still have lagging
- Sam: adding information for the scroll (velocity, phase, etc.) would be very interesting
    - IE has API on manipulationStateChanged event to get target of scroll
- Sam: would like to have standardized list of DOM properties guaranteed not to affect layout
    - then only these could be connected to input-coupled-animations
- Parallax: background attachment
    - Original design was not sufficient
    - CSS transforms on z origin
    - cases with zooming were not covered by this case
    - Two ways to treat offset of parallaxed background
    - As you zoom in the image will move
    - Doesn’t allow round tripping between zooms, but it’s what you want

    - Parallaxed element didn't need to be round tripped, didn't need to be zoomed. (rountripped = zoom out happens in the same way)
    - Several types of parallax behavior were request
    - Set of values that could be used in their css based on scroll position and scale factor
    - Can you animate background-scale, background-size? to counter the zooming in
    - CSS 3d parallax demo: http://codepen.io/scottkellum/full/bHEcA
- A list of CSS properties that don't affect layout
    - Tying scroll position and zoom position to input off thread?
    - Proposal is to start with just the CSS properties that don't affect layouts

Mozilla

While we're talking about zooming and scrolling…

- pdf.js… easier ways to target device pixels, while using browser's built-in scrolling and zooming
- What's the problem?
- After the zoom, want pdf to crisp.
- MSIE has four different types of zoom
- dpi*optical zoom can get at device widths, but there are floating point conversion issues
- Fire a resize width and height while zooming?
- We should not figure out a name for the api