: tbosch@google.com

[Objective](#heading=h.7rzuxm8b5yqs)

[Background](#heading=h.xx0cggglnzv)

[Prior Art](#heading=h.xc6z08niy1ka)

[Detailed Design](#heading=h.xozd9evtwxw0)

[Definitions](#heading=h.km9fh2zfi92m)

[Use cases and features](#heading=h.83ddxi2a4m38)

[Platforms/browsers to support](#heading=h.d55ck38iqav)

[Implementation proposals](#heading=h.v0g40pnuture)

[Scrolling with scroll events](#heading=h.jsyvse16q9bl)

[Desktop](#heading=h.xct7x8243fxc)

[Snap points](#heading=h.4ckfepbxs2vw)

[Bouncing headers/footers](#heading=h.n5zgl4n3asbv)

[Sticky headers](#heading=h.3axj5nfnfhda)

[API](#heading=h.hn36mm0p5zq)

[Notes for the Implementation](#heading=h.7ssnwwxcifk3)

[Future work: Integration with Angular](#heading=h.hd2gnw63lzim)

# Objective

Provide basic features to implement common use cases for touch devices. The features

* can be implemented using the native features of browsers, at least for some browsers
* fallback to listening to touch events and change element position using transforms on devices that are fast enough
* are performant: as fast as possible (goal is 60+ FPS)

# Background

Users are accustomed to certain touch aware usage patterns. E.g. scroll through a list using their finger, circle through pictures of a carousel, remove entries out of a list by swiping them away. However:

* current implementations of carousel, infinite scrolling, … don’t share a common core and by this have a lot of redundancy and different approaches.
* current implementations mostly don't provide an option to use native scroll events, as older browsers and also some current browsers don't support them well. However, by this they prevent the optimal performance on new devices.

# Prior Art

* TODO: integrate beforescroll event of Chrome:
  + <https://docs.google.com/a/google.com/document/d/16Z6Lun15DoWNrE2imk7N-2WiRAaqc954LOfU2-2JSoI/edit#>
* Separate widgets that implement one use case
  + Carousel for Angula<https://github.com/revolunet/angular-carousel>
  + r: And many others...
* iScroll: Fake scrolling
* Scrolling through time approach
  + <https://docs.google.com/a/google.com/document/d/1vRUo_g1il-evZs975eNzGPOuJS7H5UBxs-iZmXHux48/edit>
  + Implemented using web-animations: <https://github.com/tbosch/touch-animation>
    - not used as web animations are hard to update when the layout in the DOM changes
    - not used as too slow (e.g. on Nexus 5)

# Detailed Design

## Definitions

* fake scrolling: implementation of scrolling that listens for touch events and moves the content by changing element.style.transform.

## 

## Use cases and features

* scvirtual scrollingroll with inertia
  + only keep the rows that are visible in the DOM and update them while scrolling. Needed to scroll through large data sets
  + infinite scrolling is a special case where data is pulled in the from the backend on demand
* pull to refresh
* carousel
* swipe to delete (see Gmail)
* sticky header
* parallax scrolling (e.g. )

Related use cases for project silk:

* http://src.chromium.org/chrome/trunk/src/tools/perf/page\_sets/key\_silk\_cases.json
* <https://docs.google.com/a/google.com/spreadsheet/ccc?key=0AmUAouCtyY6-dGdhWGE2RW54T0VBOWlkc2tLNXdVbUE&usp=drive_web>

The use cases above can be implemented given the following basic features:

* Scrolling with scroll events
  + during touch *and* inertia phase
  + for parallax scrolling, virtual scrolling, sticky header
* Snap points
  + for carousel, swipe to delete
* Bouncing headers/footers that can be stopped
  + for pull to refresh
* Sticky header
  + needs scrolling with scroll events

## Platforms/browsers to support

* Desktop: Chrome, Safari, IE10+, Firefox(<http://developer.android.com/about/dashboards/index.html?utm_source=ausdroid.net>)
  + ?? Android Browser for Android 2.3 ?? (20%)
  + Android Browser for Android 4.0-4.3 (76.8%)
  + Chrome for Android (1.4%)
  + Firefox for Android
* iOS (<https://developer.apple.com/support/appstore/>)
  + Mobile Safari for iOS6 (15%)
  + Mobile Safari for iOS7 (82%)
* Windows Phone:
  + Only start with Windows Phone 8 as this is the first OS with IE10+ and the distribution of Windows Phone 7 is not so big.

## Implementation proposals

### Scrolling with scroll events

# Desktop

* + fires scroll events as expected
  + inertia depends on the input device (e.g. Mac Touchpad gives inertia scrolling, logitech mouse wheel does not)
* iOS6+
  + does not fire scroll events during inertia
  + workaround: deactivate native inertia (-webkit-overflow-scrolling:none), implement inertia manually using css transitions
    - fire scroll events during inertia using polling for transition position inside of a requestAnimationFrame loop.
* Android
  + Android browser 2.3+
    - does not support (overflow) scrolling at all
    - use fake scrolling
  + Android browser 4+: TBD
  + Chrome for Android:
    - fires scroll events as expected
  + Firefox for Android:
    - fires scroll events as expected
* IE10 mobile
  + fires scroll events as expected

### Snap points

Strategies:

* Strategy 1 (preferred if applicable)
  + modify the inertia so that the scroll always ends at a snap point
  + only possible when inertia is controllable
* Strategy 2
  + wait until end of scroll and then do a css transition to the new position

Browsers:

* Desktop: Strategy 2
  + E.g. Mac Touchpad creates inertia scrolling by the OS, can't be stopped. Using container.style.overflow='hidden' to stop scrolling does work, but switching the property back again to 'scroll' to be ready for the next user scroll could continue the old scroll
* iOS6+: Strategy 1
  + scrolling (see above) uses fake inertia, can be controlled
* Android:
  + Android browser 2.3+: Strategy 1
    - use fake scrolling
  + Android browser 4+: TBD
  + Chrome for Android: Strategy 2
    - can't stop the scrolling and use a css transition, as scrollTop is already old when passed to the scroll handler, so stopping scroll using container.style.overflow='hidden' results in jumps
  + Firefox for Android: TBD
* IE10+: Strategy 1
  + use native css property -ms-scroll-snap-points

### Bouncing headers/footers

Strategies:

* Strategy 1: directly integrate into scrolling
  + right now only with fake scrolling
  + iOS7 allows to do this natively with -webkit-overflow-scrolling:touch, but this is in contradiction with our scrolling with scroll events for iOS.
* Strategy 2: use nested scrolling areas
  + set style.overflow='hidden' of child during bounce
  + use a css transform for the bounce, update scrollTop only once when the css transform is done.
  + note: does not move the scrolling slower when in overflow

Note: We don't want to mix fake scrolling with non fake scrolling

* e.g. if we have native scrolling, it would also react to differently to the mouse wheel / maybe not support it at all.

Browsers:

* Desktop: Strategy 2
* iOS6+: Strategy 2
  + There is a native way to do this, but this requires webkit-overflow-scrolling: touch, which can't be used as we want to get scroll events during scroll: <http://damien.antipa.at/2012/10/16/ios-pull-to-refresh-in-mobile-safari-with-native-scrolling/>
* Android
  + Android browser (all): Strategy 1
    - all use fake scrolling, so also use it for header/footer
  + Chrome for Android: Strategy 2
  + Firefox for Android: Strategy 2
* IE10+: Strategy 2
  + can be implemented using nested scrolling and snap points, see <http://dwcares.com/pull-to-refresh-2/>

### Sticky headers

* Same for all browsers
* switch between position:static and position:fixed based on the position of the element during scrolling
* position:fixed is supported on all mentioned browsers (<http://caniuse.com/css-fixed>)

## API

General API:

* use css classes to mark elements
* use event delegation on the document element for events to implement the fixes

Scrolling:

* Note: The scrollable content needs to be wrapped into a div so that it can be used more easily with fake scrolling (fake scrolling needs to have an element to assign the css transform to).

|  |
| --- |
| <div class="scroll">  <div class="content">  This content here will be scrolled  </div>  </div> |

Snap points:

* use a class snap that means "snap this element to the container border if there is no other element with the snap class that is closer to the container border"
* only moves elements so that they will be visible at the end of the snap

|  |
| --- |
| <div class="scroll">  <div class="content">  ...  <img class="snap">  ...  </div>  </div> |

Bouncy header/footer:

* use a container div with child header/footer/content divs

|  |
| --- |
| <div class="scroll">  <div class="header">...</div>  <div class="content">  This content here will be scrolled  </div>  <div class="footer">...</div>  </div> |

Sticky headers:

* use css classes to mark the fixed headers

|  |
| --- |
| <div class="scroll">  <div class="content">  <div>  <div class="sticky-header">first header</div>  Content for paragraph 1  </div>  <div>  <div class="sticky-header">another header</div>  Content for paragraph 2  </div>  …  </div>  </div> |

## Notes for the Implementation

* implement performance tests while implementing this so we know that it is jank free
* The touchmove polling rate is intolerably slow on many mobile devices, this causes sluggish scrolling when users move their finger slowly, however once they have released the inertia is very fluid
* Look into predicting finger location in situations where touchmove events aren’t frequent enough to provide a good experience

# Future work: Integration with Angular

z

After implementing a general framework for touch animations we will create special directives that integrate TouchAnimations with Angular

Examples

* Integrate with ngView so that pages can be swiped through
* Virtual scrolling directive that is as easy to use as ngRepeat
* Panel directive for mobile apps, for showing menus