Viewport Issue Smorgasbord

bokan@chromium.org

This document is meant to describe and track the issues and challenges in web platform features related to the viewport. The potential solutions are mostly underpinned by the virtual-viewport work that's enabled on all Blink based platforms starting in M41. See the HTML5Rocks article to learn more about that.

Pinch-Zoom

Pinch-zoom on Chrome now uses a two-viewport model known as "virtual viewport". As of M41 this is the pinch-zoom model on all Blink platforms.

	Chrome	***Firefox***	***IE***	***Safari***
Pinch Model	virtual-viewport	single-viewport	virtual-viewport	mixed-viewport
Pinch causes OnResize	No	No	Yes (on gesture end)	No

Safari has a complicated model where they use a single viewport until some zoom threshold is reached. Once the user zooms in further, they stop shrinking the "layout" viewport and zoom in using the "visual" viewport. When in the "visual" mode, scrolling causes both viewports to scroll but at different rates.

Viewport Scroll Order

With multiple viewports, the question to ask is "which should we scroll first" (demo)? That is, when we use the scroll wheel or a touch gesture, which viewport does it target? Scrolling the layout viewport first will prevent moving position:fixed elements as long as possible. Scrolling the visual viewport first allows the user to scroll position:fixed elements easier.

Chrome	***Firefox***	***IE***	***Safari***
Layout (outer)	N/A	Visual (inner)	Simultaneous

Chrome is currently investigating whether to switch to IE's visual-first model. Tracked in 443724

An advantage of scrolling the layout viewport first is that position:fixed elements scroll with the user for as long as possible. Things like navbars and menus remain usable.

An advantage of scrolling the visual viewport first is that the user can easily scroll between position:fixed elements that aren't visible in the visual viewport. For example, if a position:fixed element is obscured by the keyboard in the ChromeOS model (more below), the user doesn't have to scroll all the way to the bottom of the page to see it.

On Screen Keyboard

The on-screen keyboard (OSK) is used for text input in the absence of a physical keyboard. It typically appears when an input text field is focused and disappears when focus is lost. When the browser is fullscreen (as is always the case on mobile platforms, or maximized on desktop platforms) the OSK will obscure the browser window.

Historically, Chrome would simply resize the browser widget. This allows the bottom part of the page to be scrolled into view and keeps position:fixed bottom elements visible above the keyboard. However, it also triggers potentially expensive resize handlers layouts. It can also result in a poor UI if the page wasn't designed to fit into such a small viewport. For example, bottom position:fixed elements may obscure the entire viewport (including the input field).

ChromeOS introduced a new model for OSK resizes. When the OSK appears it resizes the visual viewport but leaves the layout viewport unchanged. This means the page can still be fully scrolled to the bottom in the viewable area but it doesn't trigger resize handlers or cause layout. It also leaves position: fixed elements under the keyboard (they can be scrolled into view when the visual viewport scrolls) so they won't obscure the smaller viewport.

	ChromeOS	***Chrome Other***	***Firefox***	***IE***	***Safari***
OSK Resizes	Visual Viewport	Window/Widget	Window/Widget	Window/Widget	Visual Viewport?*

***There's no resize event but position fixed bottom elements can still be scrolled into view**

There are plans to experiment with using the ChromeOS model on Android; however, there may be compat impact in doing so. Tracked in 404315

Top Controls

Similar to the OSK issue. The top controls (i.e. the URL bar) are a floating bar in Chrome Android that can be shown or hidden by scrolling the body of a web page. Today, showing/hiding the top controls will resize the browser widget. For performance reasons, showing the top

controls doesn't cause the resize to happen until after the user lifts their finger (the scroll ends). When that happens, the widget is resized and Blink (and thus, the page) hears about it. Until the resize occurs, the top controls are treated as if they were simply overlaying the page.

This is problematic as scrolling can cause the page to change size. e.g. If an author uses viewport units (vw/vh) to size their page text, text will change size when the user scrolls down and hides the controls or scrolls up and shows them. Tracked in 428132

One idea to solve this is to have the top controls resize the visual viewport. Since that doesn't cause a layout or fire resize handlers, it can be done dynamically during the scroll. The major sticking point is that top controls would now obscure top position:fixed elements. If we implement position:device-fixed authors could use that instead but we still have the long tail of existing content that this would break. We'd need some kind of mitigation.

@viewport

The @viewport rule is designed to replace the <meta name="viewport"> tag. The major advantage @viewport has is that it is designed to apply on all devices. The viewport meta is applied only on phone and tablet browsers. Authors took advantage of this fact and littered it over desktop pages. A common pattern was to put a fixed width meta like <meta name="viewport" content="width=1024">; expecting that small screened phones get sent to an m. address tailored for phones, desktops would ignore it, and tablets would use the meta to layout the page in a reasonable size. Enabling the viewport meta on desktops is a non-starter because of this.

However, with the lines between desktop and mobile becoming increasingly blurry and many desktop devices shipping with touchscreens, it has become important to be able to control the viewport on devices other than mobiles. The canonical example here is Google Maps. It handles touch events to have a customized pinch-zoom experience. It does this by prevent-defaulting the touch events so that they don't cause pinch zoom, watching for pinch gestures, and scaling the background maps as appropriate. However, the touch events can only be prevent-defaulted after touch handlers have been registered. If the user pinch-zooms before that happens, the Maps UI, which doesn't expect to be scaled, gets zoomed in; the user gets stuck in a broken app since pinch-zooms are now handled by the app.

	Chrome	***Firefox***	***IE***	***Safari***
@viewport support	implemented (shipping tracked in 235457)	none	shipped (prefixed)	none

position:fixed and position:device-fixed

position:fixed elements now stick to the layout viewport, meaning that zooming in doesn't cause them to fill the visual viewport. In most legacy cases, this is an improved experience as it looks just like we magnified the page. However, what about cases where the author does actually want elements to stay in the visual viewport under zoom?

IE introduced a prefixed position:device-fixed that attaches elements to the visual viewport and doesn't scale them under pinch-zoom. This allows authors to provide an "always visible" toolbar/navbar experience that works rationally. Furthermore, it would allow authors to keep elements above the keyboard in the ChromeOS OSK model. If Chrome goes the way of moving UI to mutate the visual viewport then something like device-fixed will become necessary to allow authors to layout around the UI widgets.

Another issue in this space is what size to make the layout viewport? Note that the layout viewport and the layout size can be different. Today on Chrome, the layout viewport is sized to the visual viewport at minimum scale. This means that if a linebox breaks out of the initial containing block, the layout viewport will be larger than expected. In a pathological case, animations can cause the content size to grow (thereby making the min-scale smaller) and so the layout viewport would grow as well. Tracked in 437303

	Chrome	***Firefox***	***IE***	***Safari***
position:fixed viewport	layout	visual	layout	mixed-viewport
Supports device-fixed	No	No	Yes	No
Layout Viewport Size	Minimum-scale	N/A	Minimum-scale*	N/A
Minimum Scale	Content size or 0.25	Content size	Layout/ICB Size	Layout/ICB Size

***Though it seems like IE sizes to the minimum-scale, there's no way to control minimum scale in IE (at least on a Surface, I don't have a WinPhone to test)**