Low Latency Canvas on ChromeOS

*Status: active -- **go/lowlatency-canvas-on-chromeos** (**tinyurl.com/lowlatency-canvas-on-chromeos**)*
Authors: dcastagna@, mcasas@
Last Updated: 2019/03/22

Objective

This document describes ways of debugging the use of lowLatency **HTML Canvas**. For non-Web low-latency rendering, i.e. Android Apps or NaCl/Pepper plugins, check out **go/low-latency-rendering**. More insights on Canvas developing with Stylus can be found at **tinyurl.com/stylus-web-devex**.

Background

HTML 2D/3D Canvas Contexts can be created with the option ~~`lowLatency: true`~~[1] `desynchronized: true`:

```
var context_2d = canvas_a.getContext("2d", {~~lowLatency: true, ~~desynchronized: true});
// or
var context_3d = canvas_b.getContext("webgl", {~~lowLatency: true, ~~desynchronized: true});
```

This activates a different code path for compositing, where every JS loop activation (i.e. at 60Hz), causes a blit of the contents of the Context to a GpuMemoryBuffer that is then sent to the Display Compositor directly, skipping the latency introduced by the Renderer Compositor queue.

Under certain circumstances, this GpuMemoryBuffer can be cherry picked out of the Display Compositor queue and be **promoted** to a Hardware Overlay, that is then sent directly to the (Hardware) Display Controller (see tinyurl.com/display-debugging-on-cros); this reduces the latency and power consumption even further. When promoted to a Hardware Overlay, the GpuMemoryBuffer is held on to, effectively becoming the front-buffer, that is, what the Display Controller uses for scanout on screen. Guaranteeing this promotion is important to low latency.

Things to check for

✎ ~~While the feature is being developed, make sure that the **flag**:~~
~~`chrome://flags/#enable-experimental-web-platform-features`~~
~~is enabled in **chrome:flags** tab.~~ Not needed in Chrome >= M75.

✎ Canvas contexts must be **opaque** for the overlay promotion to work properly. This is specified via the creation attribute **`alpha: false`**, i.e.:

```
var context_2d = canvas_2d.getContext("2d",
                                      {lowLatency: true, desynchronized: true, alpha: false});
// or
var context_3d = canvas_3d.getContext("webgl",
                                      {lowLatency: true, desynchronized: true, alpha: false});
```

✎ Hardware Overlays does not work by default in all **orientations** due to hardware limitations. On Intel platforms, 0 and 180 degrees are fine, but 90 and 270 degrees are not; on ARM platforms, only the no rotation can be promoted to overlay. (See the Section Further debugging for methods to check this out). This can be circumvented though: In order to promote a canvas to overlay in a rotated orientation, you must apply the inverse rotation to the canvas with CSS, which may then

[1] After Chrome 75.0.x.0, see https://crbug.com/944199.

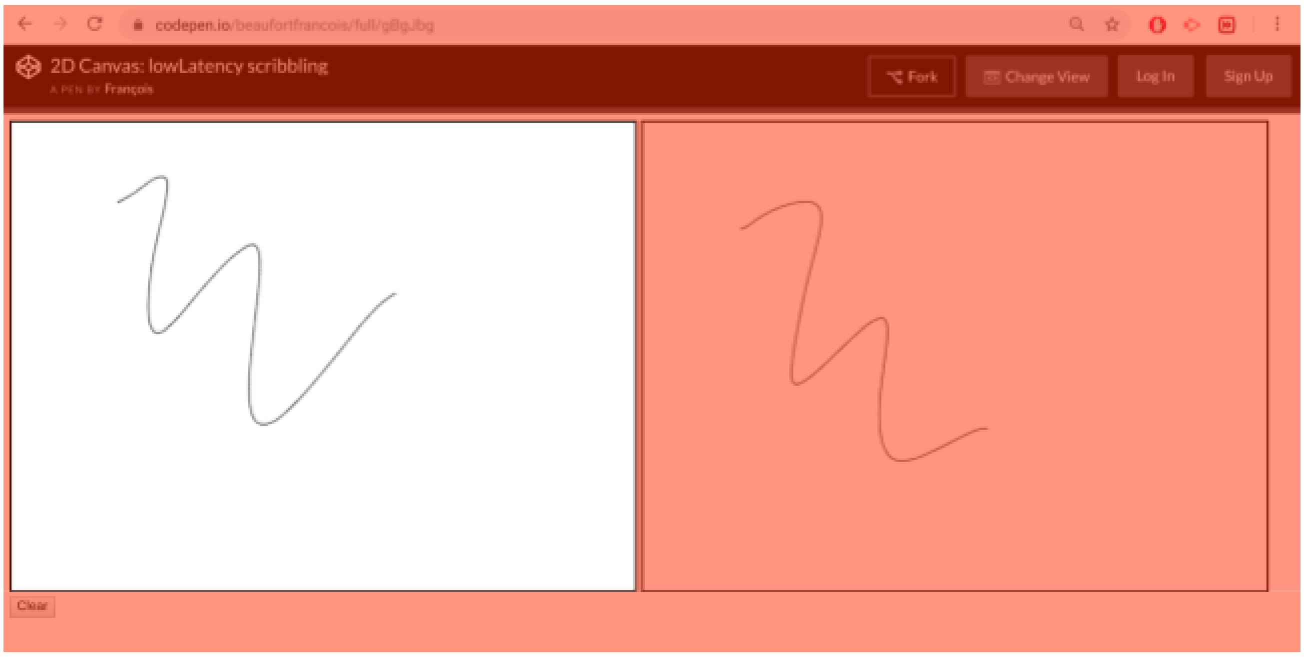
require your drawing code to reverse that rotation for serialization. (See doc on single-buffered rendering for details. Note that the flipY mentioned in that doc is not necessary for WebGL, but the rotation is.) Vertically or horizontally flipped elements are not promoted.

✎ Only one drawing "layer" can be promoted to Hardware Overlay; sometimes an unexpected one gets promoted, check the Section Further debugging for help with identifying which element, if any, gets promoted.

✎ A Canvas should be completely inside the screen to be correctly promoted, and on an integer pixel boundary. Not all scaling values can be promoted, e.g. on Intel the scaling should be kept 50%<= scaling <=200%.

Further debugging

Enabling the flag **`chrome:flags#tint-gl-composited-content`** causes a red shade to be applied to all composited contents; since overlays are not composited, they will be scanout unmodified. This can be used for verification/confirmation of the appropriate promotion, see e.g. the next demo (courtesy of **fbeaufort@**), where a lowLatency canvas is drawn next to a plain vanilla one -- the lowLatency one is correctly promoted to a hardware overlay.


codepen.io/beaufortfrancois/full/gBgJbg
2D Canvas: lowLatency scribbling
A PEN BY François
Fork
Change View
Log In
Sign Up
Clear

To measure latency, the typical method is to use a camera with 240 FPS video (e.g. Pixel 2 or a nice point-and-shoot) to record video of drawing a line with a stylus. You can then use any video playing app that supports frame-by-frame advance (e.g. VLC, mpv, mplayer) and then pick a point where the stylus is, touch it on your screen, and then count frames until the ink reaches that point. 1/240 * frame count is your latency in seconds. There’s some variance in when you decide the color has reached, partly based on the gray-to-gray response time of your desktop monitor, but so long as you’re consistent and take a number of measurements and average them, you can get a sense of the relative latency. An Optofidelity robot can automate this process, but these aren’t available for general testing use.

Resources

- 2D canvas scribbling demo (link)
- WebGL canvas scribbling demo (link)
- WebGL more sophisticated scribbling demo (link)
- WhatWG Canvas Specification (link)
- WebGL Specification (link)