# How to measure browser graphics performance

**By** Ilmari Heikkinen

Ilmari is a contributor to Web**Fundamentals**

Benchmarking browser graphics in a nutshell: draw as much as you can while maintaining a smooth frame rate. Once your framerate drops, you know how much you can draw per frame. End of post. ...No? Ok, I'll explain some more.

Example time! Here's a small code snippet with a benchmarking **`tick`** function. The **`tick`** function calls a **`draw`** function with an increasing draw load until the draw takes consistently longer than 33 ms.

```
var t, previousTime;
var drawLoad = 1;
var slowCount = 0;
var maxSlow = 10;
// Note, you might need to polyfill performance.now and requestAnimationFrame
t = previousTime = performance.now();
var tick = function() {
    var maximumFrameTime = 1000/30; // 30 FPS
    t = performance.now();
    var elapsed = t - previousTime;
    previousTime = t;
    if (elapsed < maximumFrameTime || slowCount < maxSlow) {
        if (elapsed < maximumFrameTime) {
            drawLoad+=10;
        } else {
            slowCount++;
        }
        draw(drawLoad);
        requestAnimationFrame(tick);
    } else {
      // found maximum sustainable load at 30 FPS
      document.getElementById('res').innerHTML = ("could draw "+(drawLoad)+" in "
          maximumFrameTime + " ms");
    }
};
requestAnimationFrame(tick);
```

See the live example at jsFiddle

You can see how the benchmark keeps drawing more and more until it hits the point where it slows down. This is a nice and simple way to figure out how much you can draw at a smooth frame rate. You can also plug in your own draw function to the example and do some custom benchmarking, yay!

## Common caveats and pitfalls when benchmarking browser graphics

So, if the above example is the nice way to do it, what are the not so nice ways? The ways that lead to you benchmarking unrelated things or that give you weird performance metrics that don't seem to have anything to do with how fast your app is running. Glad you asked, here are the two most common ones I've seen around the web.

Measuring max FPS: draw a little bit every frame and measure the FPS. It doesn't work well for measuring graphics performance on Chrome because the underlying graphics implementation is synchronized to the screen refresh rate (so you get max 60 screen updates per second). Measuring draw call speed isn't going to be very helpful either as Chrome's drawing system puts your drawing commands into a command buffer that gets executed on the next screen refresh.

Using setTimeout for measuring graphics performance is another bad idea. The setTimeout interval is capped to 4 ms in browsers, so the most you can get out of it is 250 FPS. Historically, browsers had different minimum intervals, so you might have had a very broken trivial draw benchmark that showed browser A running at 250 FPS (4 ms min interval) and browser B running at 100 FPS (10 ms min interval). Clearly A is faster! Not! It could well be that B ran the draw code faster than A, say A took 3 ms and B took 1 ms. Doesn't affect the FPS, as the draw time is less than the minimum setTimeout interval. And if the browser renders asynchronously, all bets are off. Don't use setTimeout unless you know what you're doing.

## How to do it then

A better way to benchmark is to use a realistic drawing load and multiply it until the frame rate starts to chug. For example, if you're writing a top-down game with a tilemap terrain, try drawing the tilemap every frame and seeing if it runs at 60 FPS. If yes, increase load (draw tilemap twice every frame, with a clear in between). Continue increasing until the FPS drops to a new stable level. Now you know how many layers of tilemap you can draw per frame.

Different graphics applications have different needs, so you should write your benchmarks with that in mind. Measure the graphics features that you are using in your app. When you find a slow scenario, try to reduce it down to the smallest piece of code that reproduces it (and file a bug report at "new.crbug.com":http://new.crbug.com if it should be faster.)

To see how to write high-performance web graphics code, check out th Google I/O 2012 talk by Nat Duca and Tom Wiltzius from the Chrome GPU team.

---



*Last updated July 2, 2018.*