previous Table of Contents next

CHAPTER 8

Mobile Devices and Performance Optimization

Now, let's talk about mobile devices and optimization. There is no mobile version of canvas, there's just canvas. It's the same API on desktop and mobile devices. Mobile devices are sometimes missing features, however, and are usually slower; but the same could be true on older desktops and browsers. So whenever you are making a canvas app it's important to consider performance and different ways to optimize your code.

Draw Less

The general mantra for performance is *draw less*.

don't draw hidden things. If you have four screens of information but only one is visible at a time, then don't draw the others.

use images instead of shapes. If you have some graphic that won't ever change or be scaled, then consider drawing it into an image at compile time using something like photoshop. In general images can be drawn much faster to the screen than vector artwork. This is especially true if you have some graphic that will be repainted over and over again like a sprite in a game.

cache using offscreen canvases. You can create new instance of the canvas object at runtime that aren't visible on screen. You can use these offscreen canvasas as a cache. When your app starts draw graphics into the offscreen canvas then just copy that over and over again to draw it. This gives you the same speed as using images over shapes, but you are generating these images at runtime and could potentially change them if needed.

image stretching. Since we are using images for lots of things already, consider stretching them for effects. Most canvas implementations have highly optimized code for

scaling and cropping images so it should be quite fast. There are also several versions of drawImage that let you draw subsections of an image. With these apis you can do clever things like caching a bunch of sprites into a single image, or wildly stretching images for funky effects. [screenshots]

only redraw the part of the screen you need. Depending on your app it may be possible to just redraw part of the screen. For example, if I have a ball bouncing around I don't need to erase and redraw the entire background. Instead I just need to redraw where the ball is and where it was on the previous frame. For some apps this could be a huge speedup.

Draw fewer frames Now that you are drawing as little per frame as possible try to draw fewer frames. To get smooth animation you might want to draw 100fps, but most computers max out at a 60fps screen refresh rate. There's no point in drawing more frames because the user will never see them. So how do you sync up with the screen refresh? Mozilla and WebKit have experimental apis to request that the browser call your code on the next screen refresh. This will replace your call to setInterval or setTimeout. Now the browser is in charge of giving you a consistent framerate, and it will ensure you don't go over 60fps. It can also do smart things like lowering the framerate if the user switches to a different tab. Mobile browsers are starting to implement this as well so background apps will be throttled back, saving on battery life.

The best way to draw less is to not draw it at all. If you have a static background then move it out of canvas and draw it with just an image in the browser. You can make the background of a canvas transparent so that a background image will show through. If you have large images to move around you may find they move faster and smoother by using CSS transitions rather than doing it with javascript in the canvas. In general CSS transitions will be faster because they are implemented in C rather than JS, but your mileage may vary, so test test test. Speaking of which: Chrome and Mozilla have great tools to help you debug and test your JavaScript. [names? examples?]

pixel aligned images. One final tip, in some implementations images and shapes will draw faster if they are draw on pixel boundaries. Some tests show a 2 to 3x speedup [verify] on the ipad canvas impl if you pixel align your sprites.

previous Table of Contents next