Rich Web Graphics: CSS3, SVG, HTML5 Canvas?

(Or … WebGL?)

Danny Brian

✉ danny@brians.org

My goals for this session.

- That you gain with a solid understanding of the options for browser-based animation.
- That you understand the pros and cons of the various options.
- That you can select the right option for your next big project.
- That you understand the performance implications of different approaches, *even for “non-rich” web content.* (That you care about jank.)

The Web: What’s so “rich” about it?

(Your answer will depend a lot on how you define the technologies.)

What is the web?

A. Port 80. Done.

B. A hierarchical document programming environment (DOM).

C. Openness and accessibility of information.

D. A slow moving body of committee standards.

What is the web browser?

A. A portable application runtime.

B. A hierarchical document programming environment (DOM).

C. A collection of constantly evolving features.

D. Disparate implementations of a slow moving body of committee standards.

E. The most used piece of free software, ever.

What are the goals of creating a rich(er) web experience?

- Creating a more visually appealing application.
- Making games for the browser.
- Demonstrating awesome development abilities.

Imperative versus declarative animations.

- *“Tell me what to do”* versus *“tell me what you want to happen”.*

- JavaScript animations are imperative and constrained by the browser’s main thread and processing power. But, you can do whatever you want.

- CSS3/SMIL animations are declarative and able to be hardware accelerated. You can’t do whatever you want.

The options under consideration today.

- **HTML with JavaScript animation.**
- **HTML with CSS3 animation.**
- **SVG.**
- **HTML5 Canvas.**
- **WebGL.**

(A sidebar on browser developer tools.)

1. HTML with JavaScript animations.

- Yes we’re talking about “old-school” jQuery animations. You’ve got a loop going on that’s using **setInterval()** or **setTimeout()** under the hood, like frames.
- But we’re also talking about low-level CSS3 style manipulation.

CSS*x* manipulation with JavaScript.

```
el = document.querySelector('#something');
el.style.transform = 'transform(30px, 50px)';
el.style.webkitTransform = 'transform(30px, 50px)';
```

2. HTML with CSS3 animations.

- Talking about CSS3 transitions of any animatable property.
- Differ (a bit) between browsers.
- Usually includes many CSS3 additions, as well as older properties.
- Also includes many SVG properties.
- Hardware acceleration depends on the property!

CSS3 Transitions.

```
div {
    width: 100px;
}

div:hover {
    width: 300px;
}

div {
    transition: width 2s;
}
```

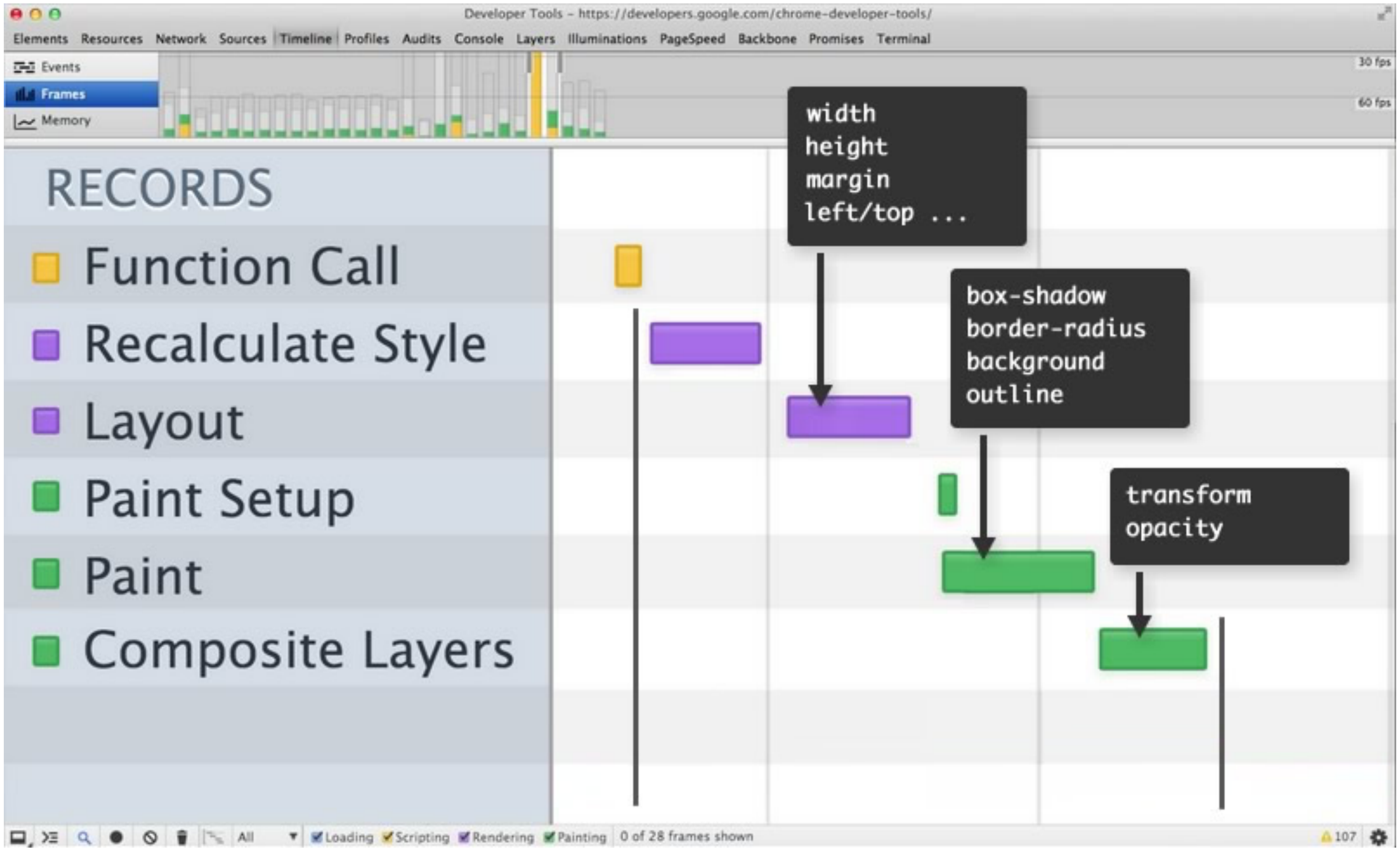
Dissecting some examples of CSS3 animations.

http://cssdeck.com/labs/google-doodle-in-css-follow-up

http://codepen.io/juliangarnier/pen/idhuG

http://threejs.org/examples/css3d_periodictable.html

From DOM to the screen.



See http://www.html5rocks.com/en/tutorials/speed/high-performance-animations/

Keep CSS3 transitions simple.

- **transform: translate(*n*px, *n*px);**
- **transform: scale(*n*)**
- **rotate(*n*deg)**
- **opacity:*n*;**

Pros and cons of CSS3 transition animations.

- Provides hardware acceleration for CSS3 transforms.
- Easy to implement.
- Difficult to compose from code.
- Transitions are “start-and-watch” canned animations. Interactivity with the animations is limited.
- Sequencing via CSS code is a pain, and animation events are fairly inconsistent between browsers.

A look at some tools and libraries to work with CSS3 animations.

- Adobe Edge Animate.
- famo.us.
- Animate.CSS.
- Polymer’s Web Animation polyfill.

3. SVG.

- Vector graphics with XML/DOM model familiar to web developers.
- Has its own animation language — SMIL (Synchronized Multimedia Integration Language).
- Can be styled and animated to varying degrees with CSS3 transitions, depending on the browser.
- You can both include SVG files as <img> or use <svg>.

SVG elements.

```
<svg height="210" width="400">
  <path d="M150 0 L75 200 L225 200 Z" />
  <rect x="50" y="20" rx="20" ry="20" width="150" height="150"
style="fill:red;stroke:black;stroke-width:5;opacity:0.5" />
</svg>
```

SVG animation with JavaScript.

```
<circle id="circle1" cx="20" cy="20" r="10"
            style="stroke: none; fill: #ff0000;"/>
```

```
var circle = document.getElementById(“circle1");
circle.setAttribute(“cx”,
     parseInt(circle.getAttribute("cx")) + 1);
```

SVG animation with CSS3?

SVG animation with SMIL.

```
<svg width="300px" height="100px">
    <rect x="0" y="0" width="300" height="100" stroke="black"
        stroke-width="1" />
    <circle cx="0" cy="50" r="15" fill="blue" stroke="black"
        stroke-width="1">
      <animate attributeName="cx" from="0" to="100" dur="5s"
         repeatCount="indefinite" />
    </circle>
</svg>
```

Dissecting some examples of SVG animation.

http://codepen.io/noahblon/pen/kIAGq

http://bl.ocks.org/mbostock/4062045

http://codepen.io/cjgammon/pen/vIEge

Some pros and cons of SVG.

- Infinite scalability of vectors, compress-ability.
- Familiar programming model.
- Great browser and tools support.
- Not all the libraries you love will “just work” with SVG.
- SMIL adoption seems slow, but SVG also has improving CSS3 support and Web Animations to look forward to.

A look at some tools and libraries to work with SVG.

- Adobe Edge Animate CC.
- Illustrator.
- Inkscape.
- Snap.svg.
- d3.js.
- Raphaël.
- Velocity.js.

4. HTML5 Canvas.

- Basically an empty **<img>** with a programmatic API — bitmap.
- The API is low-level, with shape primitives such as rectangles, lines, and text. It also provides pixel-level manipulation, which is where the more interesting use cases lie.
- The primitives are drawing functions, not objects. The canvas has no state beyond what it displays — no “scene graph”.
- In other words, to animate canvas content, you clear and redraw it.

Some pros and cons of Canvas.

- You can pretty much do anything you want.
- Decent browser support.
- Hardware acceleration with **drawImage()**.
- You don’t get the familiar DOM model — you’re treating the browser as a box of features.
- You most definitely want to use a 3rd-party library, which is a dependency.

A look at some tools and libraries to work with Canvas.

- Paper.js.
- EaselJS.
- KineticJS.
- Fabric.js.

And what about WebGL?

So what *should* we use?

The clear choices.

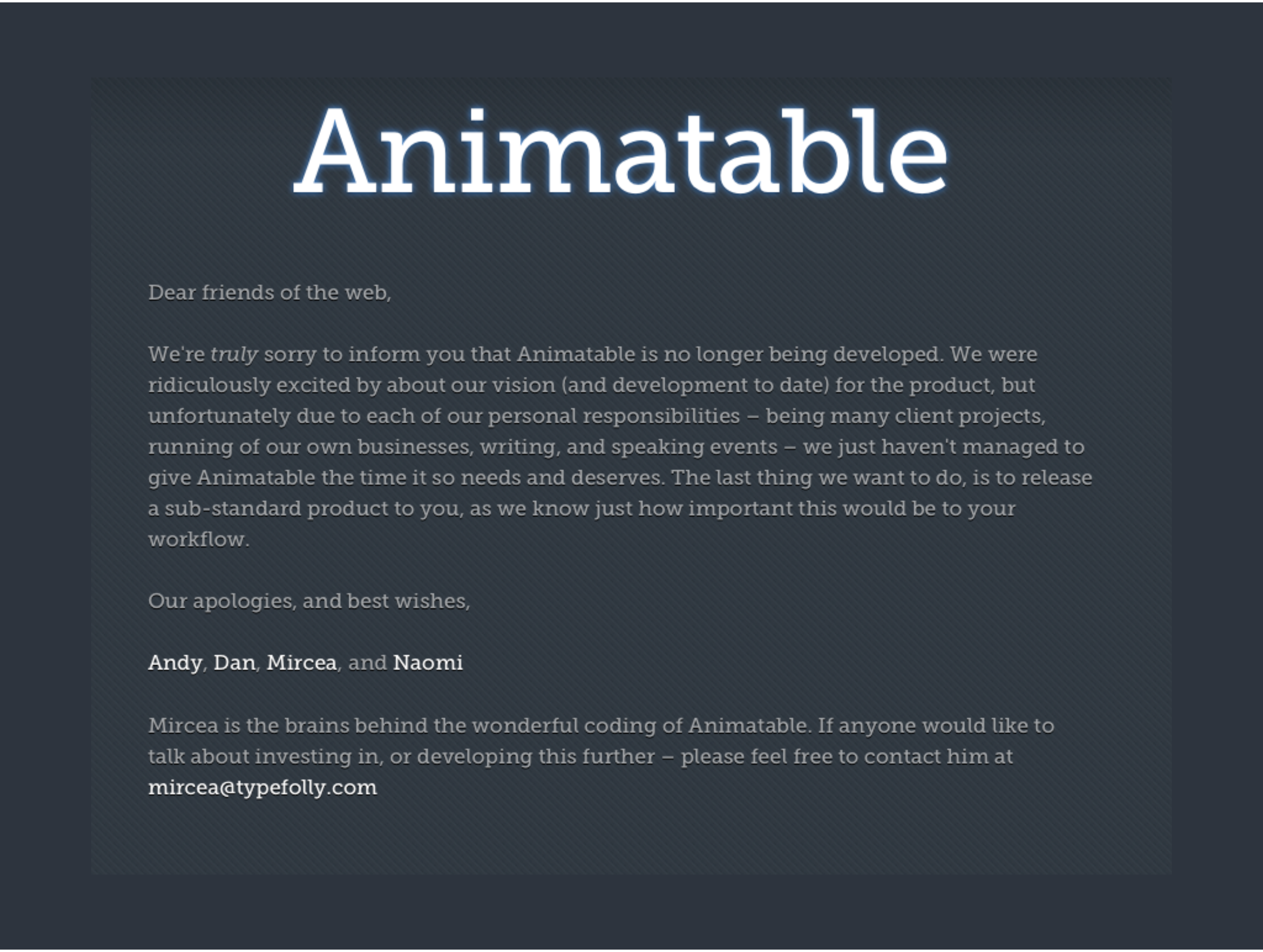
- Support for older browsers: **PO JavaScript.**
- Content-oriented apps requiring smooth panning elements, fades: **CSS3.**
- Rich data visualization: **SVG or Canvas.**
- Browser-based games: **SVG or Canvas.**
- Cutting edge browser-based video games: **WebGL**.

So what *should* we use?
The less clear choices.

- Support for older browsers: **PO JavaScript.**
- Content-oriented apps requiring smooth panning elements, fades: **CSS3.**
- Rich data visualization: **SVG or Canvas.**
- Browser-based games: **SVG or Canvas.**
- Cutting edge browser-based video games: **WebGL**.

Remember that you *can* target the browser with other development tools.

- Unity 5 and Unreal Engine 4 can publish WebGL JavaScript.
- Emscripten can compile C/C++ to JavaScript.
- Adobe Flash can publish HTML5 Canvas.
- Lots of managed application frameworks like Vaadin generate the web code.



Animatable

Dear friends of the web,

We're *truly* sorry to inform you that Animatable is no longer being developed. We were ridiculously excited by about our vision (and development to date) for the product, but unfortunately due to each of our personal responsibilities – being many client projects, running of our own businesses, writing, and speaking events – we just haven't managed to give Animatable the time it so needs and deserves. The last thing we want to do, is to release a sub-standard product to you, as we know just how important this would be to your workflow.

Our apologies, and best wishes,

Andy, Dan, Mircea, and Naomi

Mircea is the brains behind the wonderful coding of Animatable. If anyone would like to talk about investing in, or developing this further – please feel free to contact him at mircea@typefolly.com