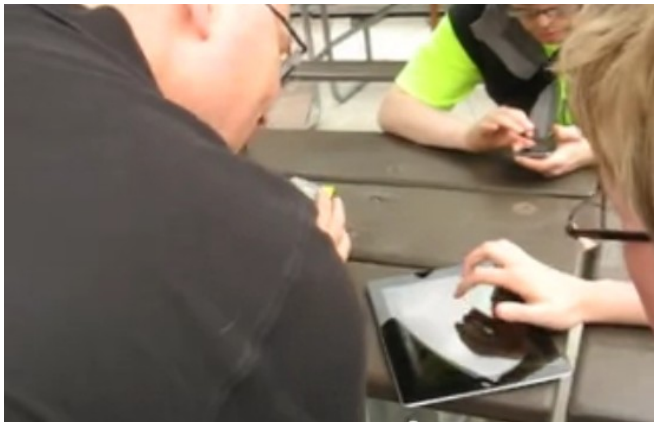
Addressing Mobile Users with SVG, Canvas and JS

Abram Hindle
abram.hindle@softwareprocess.es
Department of Computing Science
University of Alberta

Presented to ExchangeJS
The Edmonton Javascript Users Group
http://www.exchangejs.com/

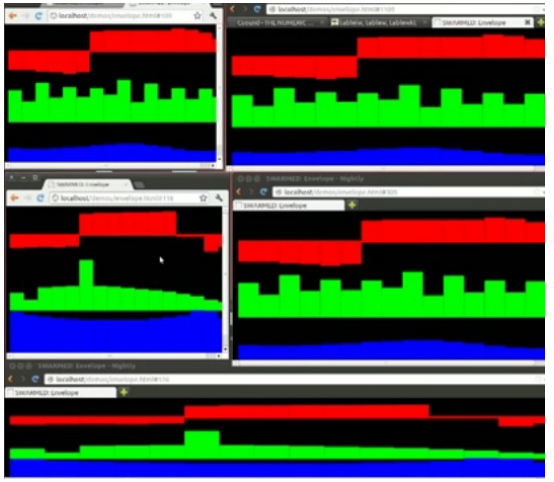
The Bane of Mobile

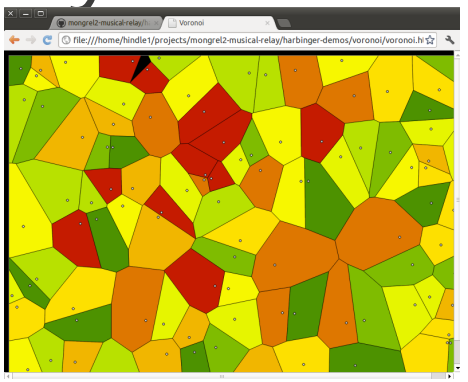
- Multiple Kinds of Browsers
- Multiple sets of supported features (SVG, Canvas, XML Http Request, Long Polling, Web Sockets etc.)
- Slightly different look and feel
- Multiple kinds of input
  - Touch Input is handled differently than mouse input
- Multiple Screen Sizes



Focus of the Talk

- The focus of this talk is interactive, graphics heavy UIs.
- UIs of games
- UIs of datavisualizers
- UIs of musical instruments
- Much of what is covered here will apply to plain-jane UIs as well (especially if you have drag operations).



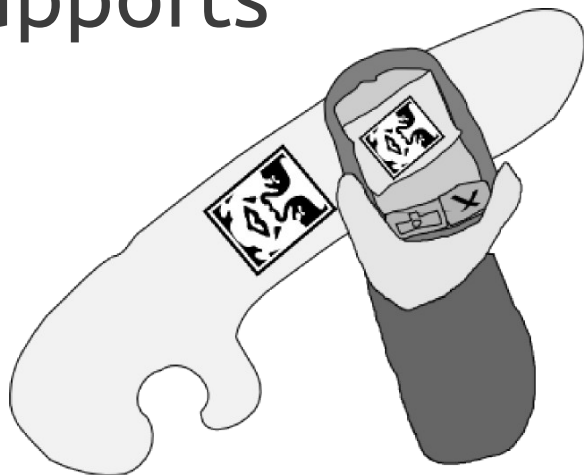


Coverage

- Using JS & HTML5
  - Full Screening an App
  - Figuring out where people are clicking in Canvas
  - Adding appropriate Touch Events to Canvas and SVG that allow for touch dragging
- Not covered:
  - Drawing on a canvas

Mobile Stereotypes

- IOS – can handle SVG and Canvas. Usually problem free except can act funny with captive networks
- Android 1 and 2: No SVG, usually Canvas is supported. Touch often works.
- Android 4: SVG, Touch, works pretty well.
- BlackBerry: A little different but supports SVG and Canvas

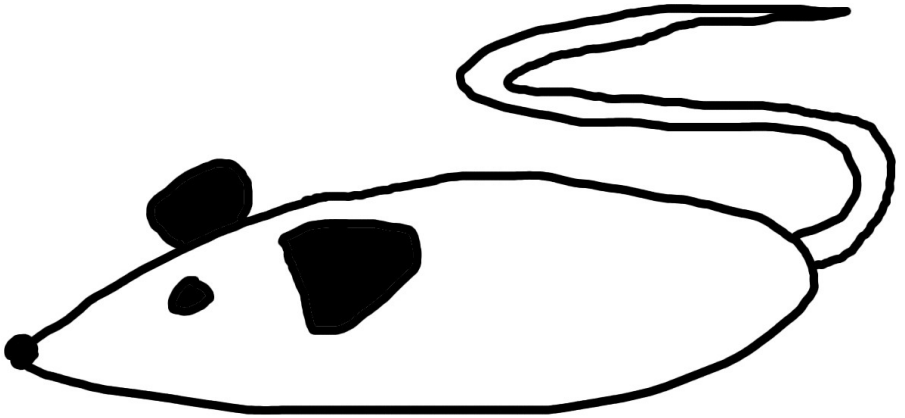


Javascript on Mobile

- Given the wide variety of browser and versions
  - Choose a good framework you are comfortable with that ensures portability.
  - JQuery Mobile seems to work ok.
- Some JS frameworks do not handle touch events by default.

Remember the Mouse?

- Your phone pretends to have mice.
- Mouse Down
- Mouse Up
- Mouse Move
- Mouse Click
- All irritating handlers to deal with.
- Also handlers about exiting and entering context (useful in dragging).
- Unfortunately the canvas doesn't rreturn a great location for you.



Get The Position of the Mouse for Mobile

```
// This code allows you to get the x,y coords of a touch from the first touch
// Or just fall back on the supposed mouse position
// Note that touches are stored as a list so in the multi-touch context you can address this
function getPosition(e) {
    if ( e.targetTouches && e.targetTouches.length > 0) {
        var touch = e.targetTouches[0];
        var x = touch.pageX  - canvas.offsetLeft;
        var y = touch.pageY  - canvas.offsetTop;
        return [x,y];
    } else {
        var rect = e.target.getBoundingClientRect();
        var x = e.pageX  - canvas.offsetLeft;
        var y = e.pageY  - canvas.offsetTop;
        return [x,y];
   }
}
```

Mouse Down Handler

```
mousedown: function(e) {
           e.preventDefault(); // Browser no don't do it!
           var pos = getPosition(e);
           var x = pos[0];
           var y = pos[1];
           self.clicked = 1;
           // This is a bad smell (x,y) should be
           // a position object, please excuse this.
           self.handleClicked(x,y);
},
```

MouseUp Handler

```
mouseup: function(e) {
        e.preventDefault();
        // we're not dragging no more
        self.clicked = 0;
},
```

MouseMove Handler

```
mousemove: function(e) {
            e.preventDefault();
            var pos = getPosition(e);
            var x = pos[0];
            var y = pos[1];
            // If we are dragging
            if (self.clicked != 0) {
                self.handleClicked(x,y);
            }
        },
```

Touch?

- Touch Events have different event types and contain different information than a simple mouse move, mouse up, mouse down, mouse click event.
- Touch Events and Mouse events are not compatible and do not really have compatible listeners.
- Touch is awkward because there is no mouse motion.

Touch?

- Touch Events have different event types and contain different information than a simple mouse move, mouse up, mouse down, mouse click event.
- Touch Events and Mouse events are not compatible and do not really have compatible listeners.
- Touch is awkward because there is no mouse motion.
- There are often multiple touch positions

TouchStart and TouchEnd Handler

- Because we used the getPosition function in our mouse handler, our handlers are now compatible, we just need to delegate to the right handler!
- This handler deals with the start of a touch event. And the End of it

```
touchstart: function(e) {
    return self.mousedown(e);
},
touchend: function(e) {
    return self.mouseup(e);
},
```

touchMove

- Touch Move handler deals with mouse motion but by abstracting location in getPosition we can just call our mousemove handler.
- Note we can rely on touchstart to occur first
- And we can rely on touchend to occur after

```
touchmove: function(e) {
        return self.mousemove(e);
},
```

Install The Handlers

- canvas.addEventListener("mousedown", self.mousedown, **false**);
  canvas.addEventListener("mousemove", self.mousemove, **false**);
  canvas.addEventListener("mouseup", self.mouseup, **false**);
  canvas.addEventListener("touchstart", self.touchstart, **false**);
  canvas.addEventListener("touchmove", self.touchmove, **false**);
  canvas.addEventListener("touchend", self.touchend, **false**);

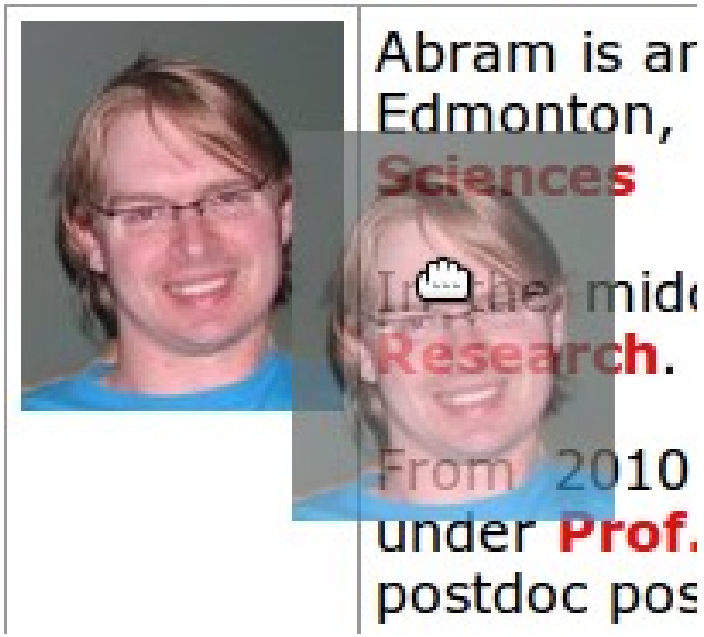
Note: these are added to the canvas.

Mobile Canvas Review

- For canvas you need to subtract offsets to find pixel click location
- For touch events your method for determining location are different than mouse events
- You need to install handlers
- Abstraction is your friend

Dragging Still Sucks

- Often when you drag on a canvas it will pull the canvas out of context like you would drag an image.
- We need to disable as much automatic dragging stuff in the browser as possible.



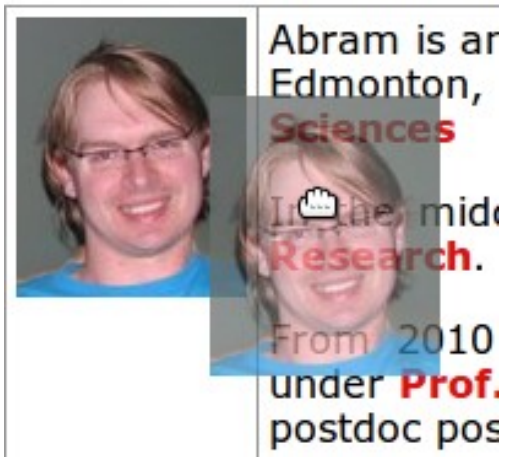
When I Drag It still drags funny

- We want to disable default dragging behaviour by calling preventDefault on the event.

```
function preventDefault(e) {
   e.preventDefault();
}
document.addEventListener("touchstart", preventDefault, false);
document.addEventListener("touchmove", preventDefault, false);
document.addEventListener("touchend", preventDefault, false);
document.addEventListener("click", preventDefault, false);
canvas.addEventListener("click", preventDefault, false);
```

When I drag it drags the Canvas!

- For webkitish browsers add to the style of your canvas: `-webkit-tap-highlight-color: transparent;`
- `<canvas id="c" width="100%" height="100%" style="-webkit-tap-highlight-color: transparent;"></canvas>`

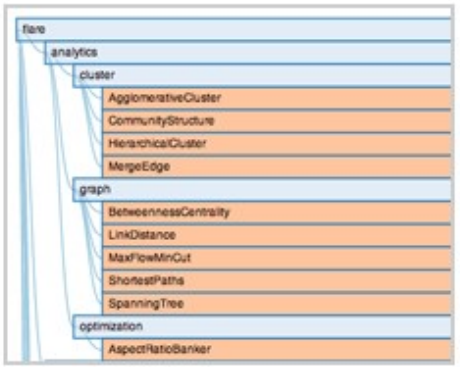


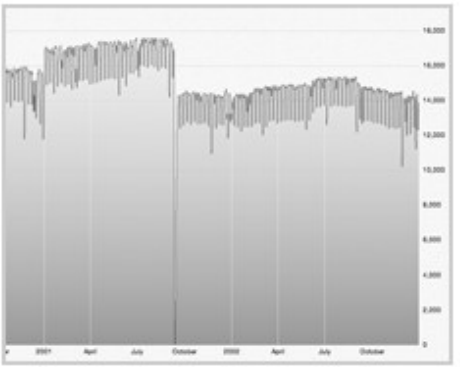
Full Screen Browser UI

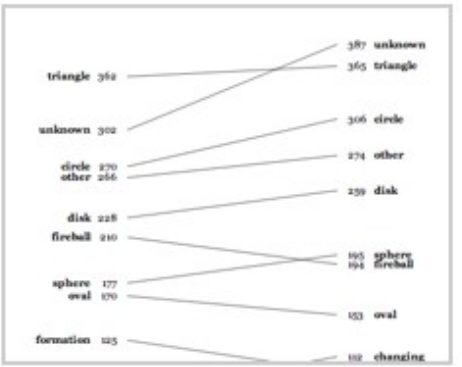
- Add meta information about the size of the page.
  - And in your JS, assuming a canvas, grab the canvas and resize it to the window size minus some fudge factor for scrollbars and decorations.
  -

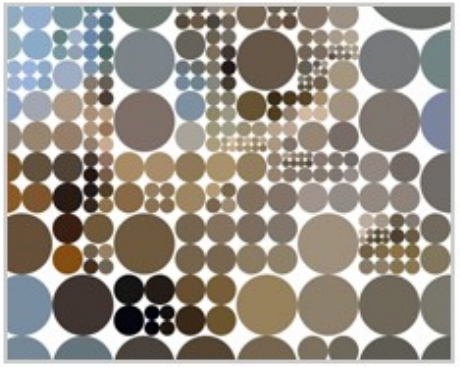
```
<meta content='width=device-width; height=device-
height; initial-scale=1.0; maximum-scale=1.0;
user-scalable=0;' name='viewport' />
<meta name="viewport" content="width=device-width"
/>
...
var canvas = document.getElementById("c");
var W = canvas.width  = window.innerWidth-fudgeX;
var H = canvas.height = window.innerHeight-fudgeY;
```

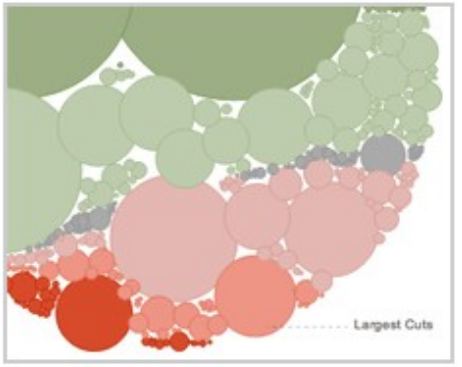
Data-Driven Documents

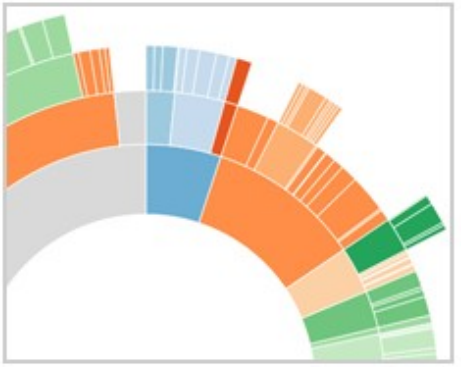












D3.js is a JavaScript library for manipulating documents based on data. **D3** helps you bring data to life using HTML, SVG and CSS. D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

See more examples.

Download the latest version here:

- d3.v3.zip

Or, to link directly to the latest release, copy this snippet:

```
<script src="http://d3js.org/d3.v3.min.js"></script>
```

D3.js

- Excellent Framework for Data Vizualization in SVG
- Beautiful and fluid animation
- Unfortunately SVG isn't too mobile-friendly compatible as Android 2 usually cannot support it.
- D3.js is great if it already supports the visualization you want. It is somewhat painful otherwise.
- D3.js provides touch support but it doesn't make it mouse compatible.
- SVG tends to work on BB, iOS, Android 4, and Firefox browsers.

D3 on Mobile Needs Touch Support

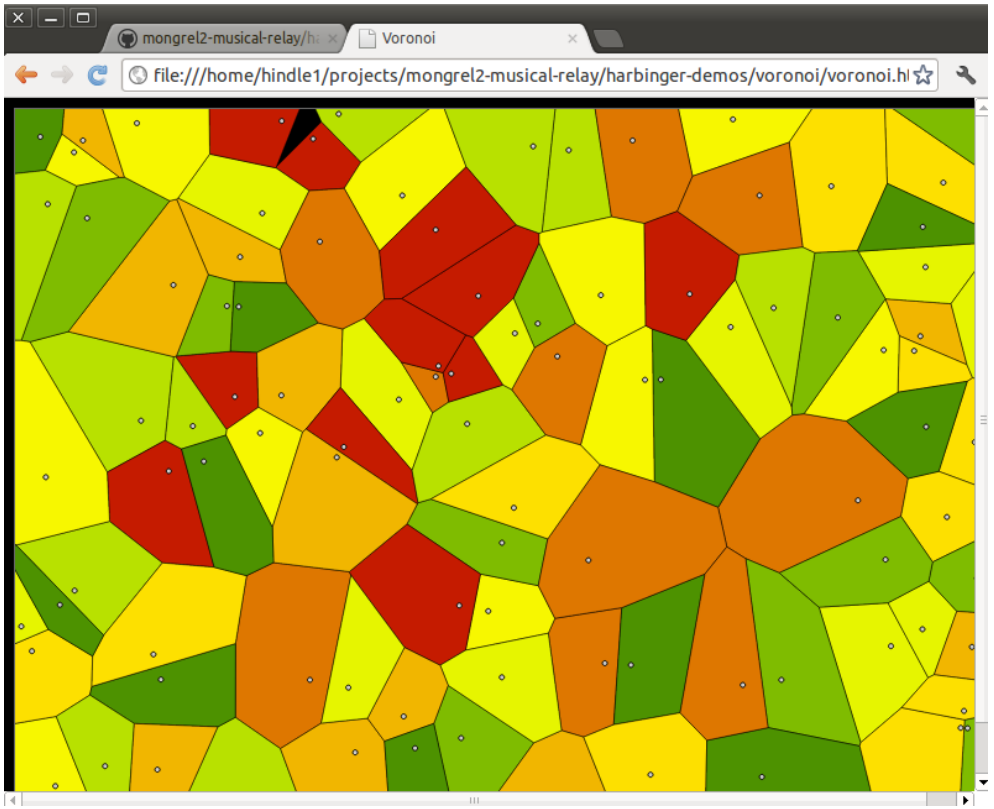
```
// D3 is full of hairy long chains like these:
var svg = d3.select("#chart")
  .append("svg")
    .attr("width", width)
    .attr("height", height)
    .attr("class", "PiYG")
    .on("mousedown", update)
    .on("mousemove", update);
```

D3 on Mobile Needs Touch Support

```
// D3 is full of hairy long chains like these, but
// You need to explicitly insert mouse and touch handlers
var svg = d3.select("#chart")
  .append("svg")
    .attr("width", width)
    .attr("height", height)
    .attr("class", "PiYG")
    .on("touchmove", touchUpdate)
    .on("touchstart", touchUpdate)
    .on("mousedown", update)
    .on("mousemove", update);
```

More D3 Handlers

```
function disableDragging() {
  if(d3.event.preventDefault)
    d3.event.preventDefault(); // note it has its own method of dealing with this
}
function update() {
  disableDragging();
  vertices[0] = d3.mouse(this); //mouse
  dealWithVertices();
};
function touchUpdate() {
    disableDragging();
    var touches = d3.touches(this); // touches
    if (touches.length > 0) { // more than 0 touches
        vertices[0] = touches[0];
        dealWithVertices();
    }
};
```



Conclusions

- Touch events are not mouse events
- Both canvas and svg need touch specific handlers or else touch screens become really awkward
- Abstracting touch events to mouse events often simplifies your problem but you still need to handle the touch events.

Resources

- Source code to the swarmed instrument
  - https://github.com/abramhindle/mongrel2-musical-relay
  - http://ur1.ca/chkz5
  - Web Resources of the instrument: http://ur1.ca/chkzh
  - http://skruntskrunt.ca/blog/2012/06/23/swarmed/
  - http://softwareprocess.es/blog/swarmed/
- D3.js http://d3js.org/
- Examples from JS1k
  - http://js1k.com/2010-first/