Using <canvas> for a great user experience

Ken Sundermeyer
Sr. Director of Engineering
iControl Networks

What we'll discuss today

1. The cross-platform challenge for eng & design
2. When to bring canvas into HTML5
3. A component model for canvas
    1. Do's and Don'ts - learn from my mistakes
    2. Let's build a simple canvas component
4. Q&A and sample code

The cross platform challenge

Targeting many platforms

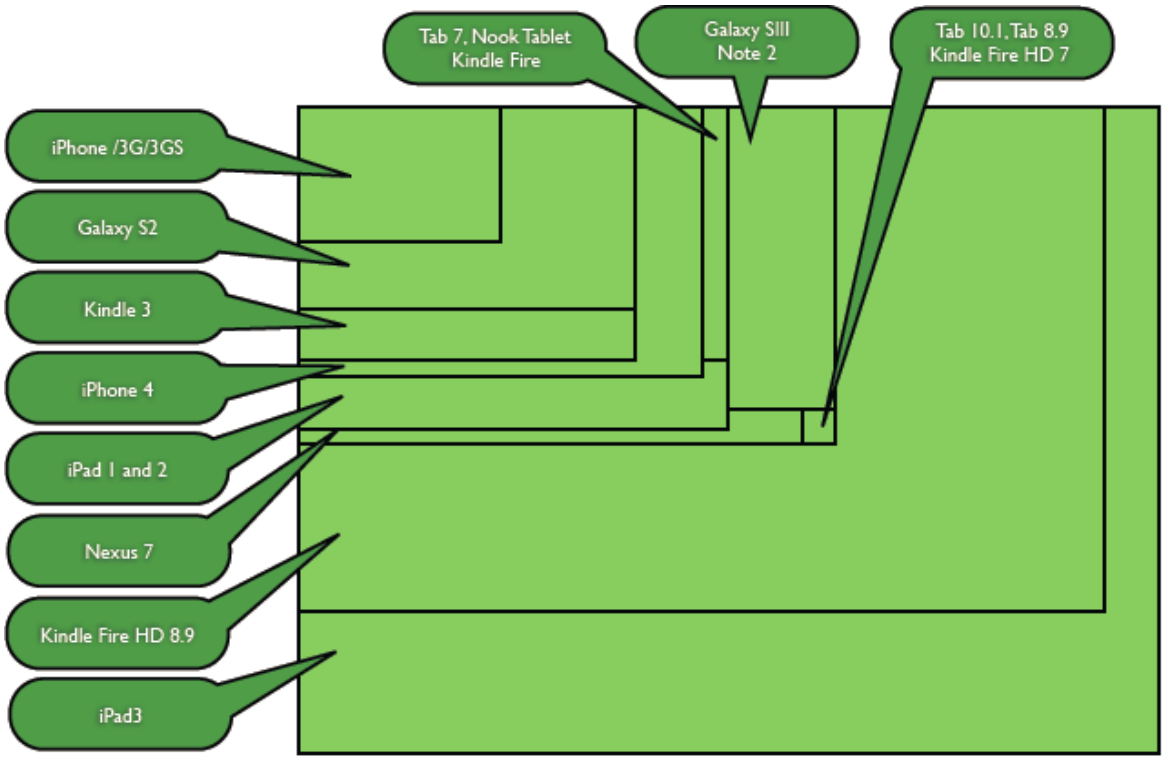
iControl makes the apps for ADT, Comcast, Time Warner, and many others. Our partners expect:

- iPhone + iPad
- Android phones, tabs, Kindle
- Web app across multiple browsers
- Touch screens
- Consistency across platforms
- Rapid new feature development
- All on a limited budget

To meet these demands, we use HTML5, and hybrid apps for mobile.

Demos: ADT hybrid web CL TG

Scare slide: screens, OS's, browsers



X

Android OS:

2.3
3.1
4.0
4.1
4.2
4.3

Apple HW on iOS 6.x + 7.x:

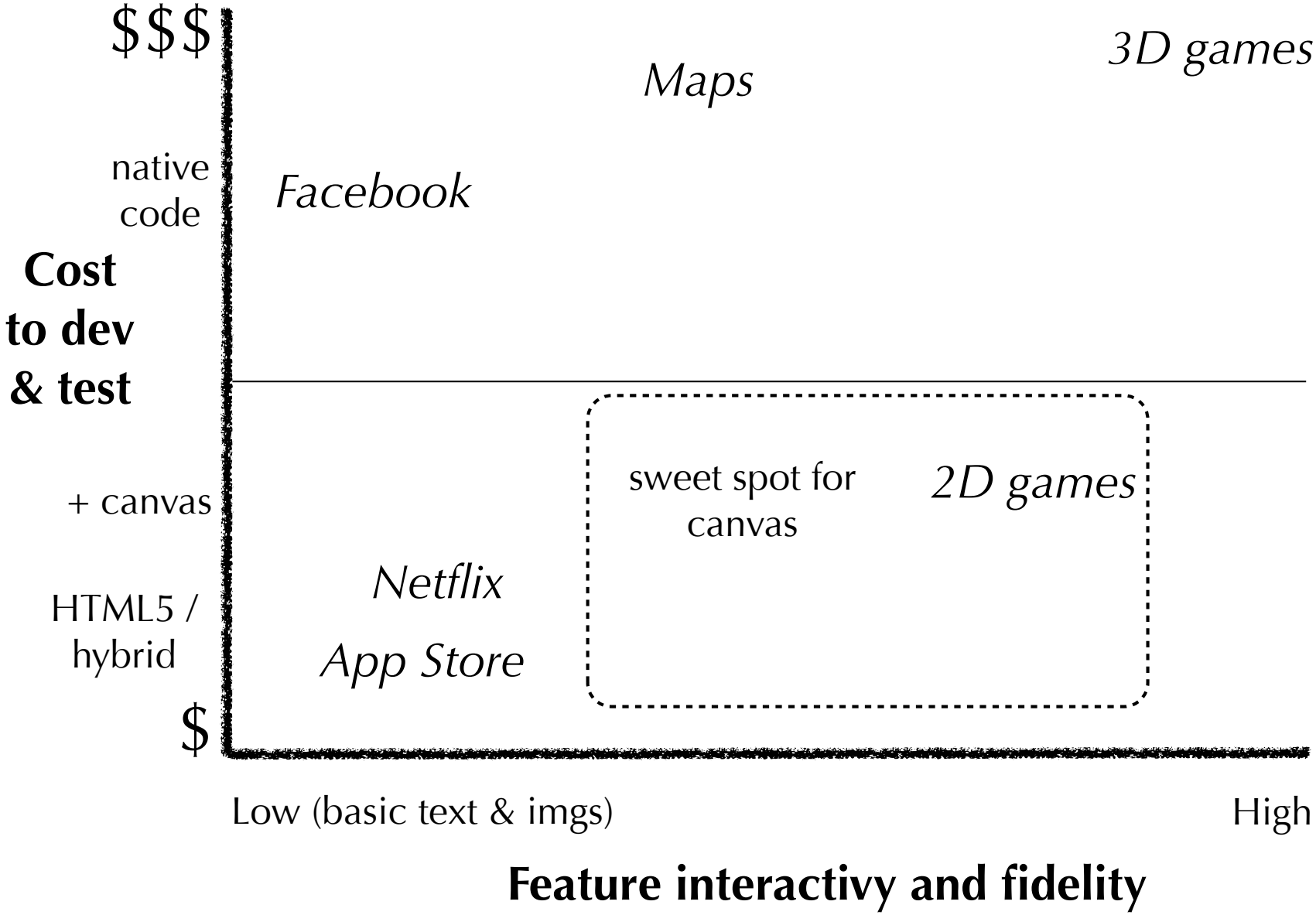
1) Apple iPhone 3GS
2) Apple iPhone 4
3) Apple iPhone 4S
4) Apple iPhone 5
5) Apple iPod Touch (3rd gen)
6) Apple iPod Touch (4th gen)
7) Apple iPad (original)
8) Apple iPad 2
9) Apple iPad (3rd gen)
10) Apple iPad Mini

Supported Operating Systems and Browsers

Windows XP SP3	Windows Vista	Windows 7	Windows 8 / Windows RT	Mac OS X (10.5 and higher)
Chrome 20+ **Firefox** 13+ **Internet Explorer** 7+	**Chrome** 20+ **Firefox** 13+ **Internet Explorer** 7+	**Chrome** 20+ **Firefox** 13+ **Internet Explorer** 7+	**Chrome** 20+ (Windows 8 only) **Firefox** 13+ (Windows 8 only) **Internet Explorer** 10	**Chrome** 20+ **Firefox** 13+ **Safari** 5.0.3+

When to use canvas

Cross-platform development choices



Demo of some components

Don't need canvas for:

- simple btns, gradients, photos. Keep using DIVs, CSS, IMG

May want canvas for:

- faster download (no more bitmaps!)
- want high-fidelity graphics at every size
- UI components beyond buttons/tabs/scrolling
- interactive features that feel native

Examples of how we use canvas:

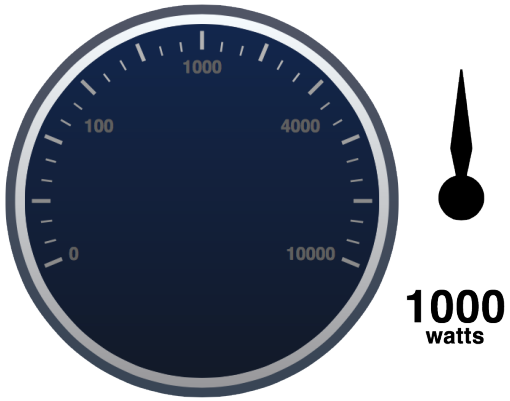
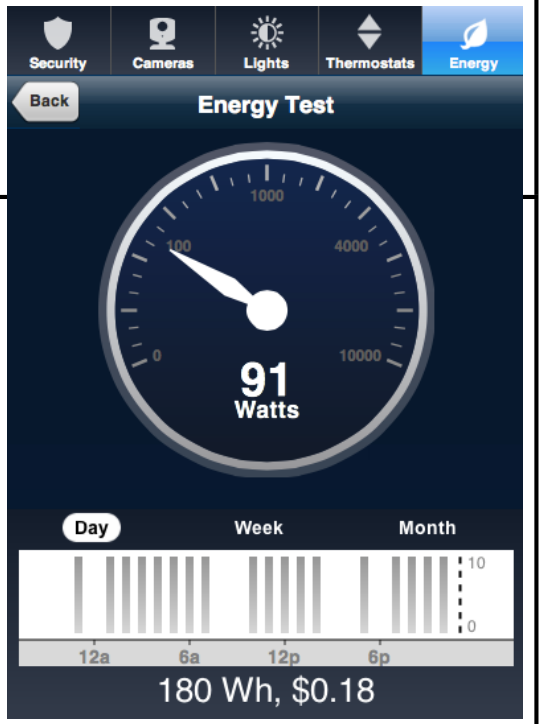
1. simple graphics: icons and symbols
2. non-interactive but changeable: diagram, meter
3. simple interaction: slider, stepper, button
4. animation: orb
5. full encapsulation: tab
6. high interaction: grapher, home view editor

These are all cross-platfom and designed separately from app development.

Components

A component model for canvas

Integrated design process using canvas

1. Component Developer	2. Designer	3. App developer
• handles touch + events • implements animations • takes care of caching and performance • builds automated tests	• draws art in Illustrator • exports canvas javascript • iterates with test page on tablets and phones	• places component • binds to data
		

Export canvas from Illustrator: Ai2Canvas

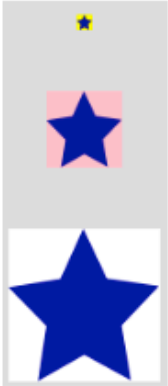
Raw export from Adobe Illustrator

```
<html lang="en">
<head> <meta charset="UTF-8" /> <title>star</title>
<script>
  function init() {
    var canvas = document.getElementById("canvas");
    var ctx = canvas.getContext("2d"); draw(ctx);
  }
  function draw(ctx) {// layer1/Path
    ctx.save();
    ctx.beginPath(); ctx.moveTo(161.3, 198.0);
    ctx.lineTo(99.8, 164.6); ctx.lineTo(38.8,
199.0);
    ctx.lineTo(49.9, 127.1); ctx.lineTo(0.0, 76.6);
    ctx.lineTo(68.4, 65.6); ctx.lineTo(98.5, 0.0);
    ctx.lineTo(129.6, 65.1); ctx.lineTo(198.2,
75.0);
    ctx.lineTo(149.0, 126.3); ctx.lineTo(161.3,
198.0);
    ctx.closePath();
    ctx.fillStyle = "rgb(0, 0, 158)"; ctx.fill();
    ctx.restore();
  }
</script>
</head>
<body onload="init()" style="">
  <canvas id="canvas" width="200" height="200">
</canvas>
</body>
</html>
```

Draw function embedded in "ic_draw_star.js"

```
ic_myControl.prototype.draw_star = function(ctx, greekPct,
clientType) {
  ctx.save();
  ctx.beginPath(); ctx.moveTo(161.3, 198.0);
  ctx.lineTo(99.8, 164.6); ctx.lineTo(38.8, 199.0);
  ctx.lineTo(49.9, 127.1); ctx.lineTo(0.0, 76.6);
  ctx.lineTo(68.4, 65.6); ctx.lineTo(98.5, 0.0);
  ctx.lineTo(129.6, 65.1); ctx.lineTo(198.2, 75.0);
  ctx.lineTo(149.0, 126.3); ctx.lineTo(161.3, 198.0);
  ctx.closePath();
  ctx.fillStyle = "rgb(0, 0, 158)"; ctx.fill();
  ctx.restore();
}
```

scaled canvases



A few canvas do's and don'ts

1. **canvas tags *require* old-school width / height attributes**
2. **canvas is perfect gentleman on iOS, good on IE9+, Android reqs some tweaks**
3. **clearing canvases: clearRect() fails on Android 4.1, width=width fails Safari**
4. **contexts are *global,* so always use save / restore**
5. **don't use fillRect() with gradients (fails on some Android 4.0.4 phones)**

```
gradient = ctx.createLinearGradient(x, y-5, x, h+15);
gradient.addColorStop(0.00, "rgb(101, 116, 140)");
gradient.addColorStop(0.2, "rgb(62, 73, 96)");
gradient.addColorStop(1.00, "rgb(32, 44, 67)");
ctx.fillStyle = gradient;
//BEFORE (don't do this!)
//  ctx.fillRect(x, y, w, h);  //BAD!! causes vanishing canvas bug in Android 4.0.
//AFTER:
ctx.beginPath();
ctx.moveTo(x,  y); ctx.lineTo(x+w,y); ctx.lineTo(x+w,y+h); ctx.lineTo(x,  y+h); ctx.lineTo(x,  y);
ctx.closePath();
ctx.fill();
```

6. **for IE9+, avoid sequence of multipe arcTo without any lineTo (draws wrong!)**

```
// BAD
   pCtx.beginPath();
   pCtx.moveTo(x, y+r);
   pCtx.arcTo(x, y, x+r, y, r);
   pCtx.arcTo(w, y, w, y+r, r);
// GOOD
   pCtx.beginPath();
   pCtx.moveTo(x, y+r);
   pCtx.arcTo(x, y, x+r, y, r);
   pCtx.lineTo(w-r, y);
   pCtx.arcTo(w, y, w, y+r, r);
   pCtx.lineTo(w, h-r);
```

Let's build a meter component

1. create our class constructor + update method
2. draw placeholder images
3. draw and rotate the meter needle
4. animate the needle using setTimeout, add Illustrator drawing
5. make scaleable and support high DPI devices
6. performance: measure, and improve thru caching
7. using angular directives with components
8. handling mouse events / touch (different example)

Supporting touch input

All UI components must support both mouse input (for desktop) and touch input (for mobile). This is accomplished by mapping touch input to mouse handling. Here's an example javascript class that captures those events properly (in this case, assumes the class was passed a canvas object, which is set to this.canvasObj). To detect the correct offset x,y position on a canvas is complex for different browsers and phones, so we use the getCanvasMousePosition function in ic_helper library:

```
<script type="text/javascript" src="ic_helper.js"></script>
<script type="text/javascript">
  function ic_myClass(canvasObj) {
    this.canvasObj = canvasObj;

    //first, add basic detection for touch devices
    this.isTouchDevice = false;
    try { // figure out if this is touch device
      if (('ontouchstart' in window) || ("ontouchstart" in document.documentElement)) this.isTouchDevice =
true;
    } catch (e) {}

    //next, map canvas mouse or touch events in your class to your event handlers
    var that = this; //need this so that an instance of your class gets events correctly
    this.canvasObj[this.isTouchDevice?"ontouchstart":"onmousedown"] = function (e) { that._mousedown(e) };
    this.canvasObj[this.isTouchDevice?"ontouchmove" :"onmousemove"] = function (e) { that._mousemove(e) };
    this.canvasObj[this.isTouchDevice?"ontouchend"  :"onmouseup"  ] = function (e) { that._mouseup(e) };
  }

  //next, handle each event
  ic_myClass.prototype._mousedown = function (e) {
    //get the browser-compatible mouse/touch position relative to canvas
    var mousePos = this.getCanvasMousePosition(this.pCanvasObj, e);
    // handle event here: mousePos.x and mousePos.y
  }

  ic_myClass.prototype._mousemove = function (e) { }
  ic_myClass.prototype._mouseup = function (e) { }
</script>
```

Q&A

sample code and slides:

www.sundermeyer.com/canvas