Rendering in Canvas

Agenda of Today's Workshop

- Understand rendering in `<canvas>`
  - Implement brush interaction
  - Implement additional customization options
- How to draw in `<canvas>`
  - Basic drawing commands
  - Animation
  - The case of a dynamic, time-based visualization
- Think about use cases

SVG vs. HTML Canvas

SVG	Canvas
`<svg>` elements are DOM elements	`<canvas>` manipulates pixels, not DOM elements
`<svg>` elements can be manipulated with JavaScript and CSS	To manipulate pixels, `<canvas>` needs to redraw
`<svg>` elements can listen to events	Much harder in `<canvas>`
	`<canvas>` is essentially an image with programmatic drawing capabilities (a drawing API)

SVG vs. HTML Canvas in Data Visualization

SVG

1. Data is bound to `<svg>` DOM elements

2. Use visual encoding of the DOM elements to express data.

Canvas

SVG vs. HTML Canvas in Data Visualization

SVG

1. Data is bound to `<svg>` DOM elements

2. Use visual encoding of the DOM elements to express data.

Canvas

Data can't be bound to pixels!

We can still use the layout, scales, and other data manipulation modules of d3, but we must rethink the "data -- DOM" pattern.

Think about how we might draw a bar chart

Canvas API

Setting up a canvas

```
var canvas = d3.select('#plot')
    .append('canvas')
    .attr('width',w)
    .attr('height',h)
    .node(),
    ctx = canvas.getContext('2d');
```

Canvas API

Setting up a canvas

```
var canvas = d3.select('#plot')
    .append('canvas')
    .attr('width',w)
    .attr('height',h)
    .node(),
    ctx = canvas.getContext('2d');
```

1. First, we create a `<canvas>` DOM node
2. `<canvas>` exposes drawing contexts (2d or 3d), which allows us to then create and manipulate content.

Canvas API

Basic drawing commands

There are only two types of primitives in canvas: path or rect

```
context2d.fillRect(x,y,w,h);
context2d.clearRect(x,y,w,h);
context2d.strokeRect(x,y,w,h);
```

Canvas API

Basic drawing commands

For paths, you must begin a new path, apply a series of drawing commands, [optionally] close the path, and either stroke or fill the path.

```
ctx.beginPath();
...
//drawing commands
...
ctx.stroke();
ctx.fill();
```

```
//drawing commands
ctx.moveTo(x,y);
ctx.lineTo(x,y);
ctx.bezierCurveTo();
ctx.quadraticCurveTo();
ctx.arc();
ctx.rect();
ctx.arcTo();
```

Canvas API

Basic drawing commands

How are stroke, fill, and style determined?

```
ctx.fillStyle = ‘red’;
ctx.strokeStyle = ‘blue’;
ctx.lineWidth = 2;
ctx.globalAlpha = .5;
```

Canvas API

Exercise 1

Practice these basic drawing commands.

Canvas API

Animation in canvas

```
function draw(){

  //draw one frame of <canvas>

}
```

Canvas API

Animation in canvas

```
function draw(){

  //draw one frame of <canvas>

  window.requestAnimationFrame(draw);
}

window.requestAnimationFrame(draw);
```

Canvas API

Animation in canvas

```
function draw(){

  //draw one frame of <canvas>

  window.requestAnimationFrame(draw);
}

window.requestAnimationFrame(draw);
```

2. draw function will create one frame of animation

3. at the end of each frame, call draw function again, which will draw the next frame

1. call draw function

Canvas API

Animation in canvas

```
function draw(){
  //clear previous content
  ctx.clearRect(0,0,w,h);

  //draw one frame of <canvas>
  //...

  window.requestAnimationFrame(draw);
}

window.requestAnimationFrame(draw);
```

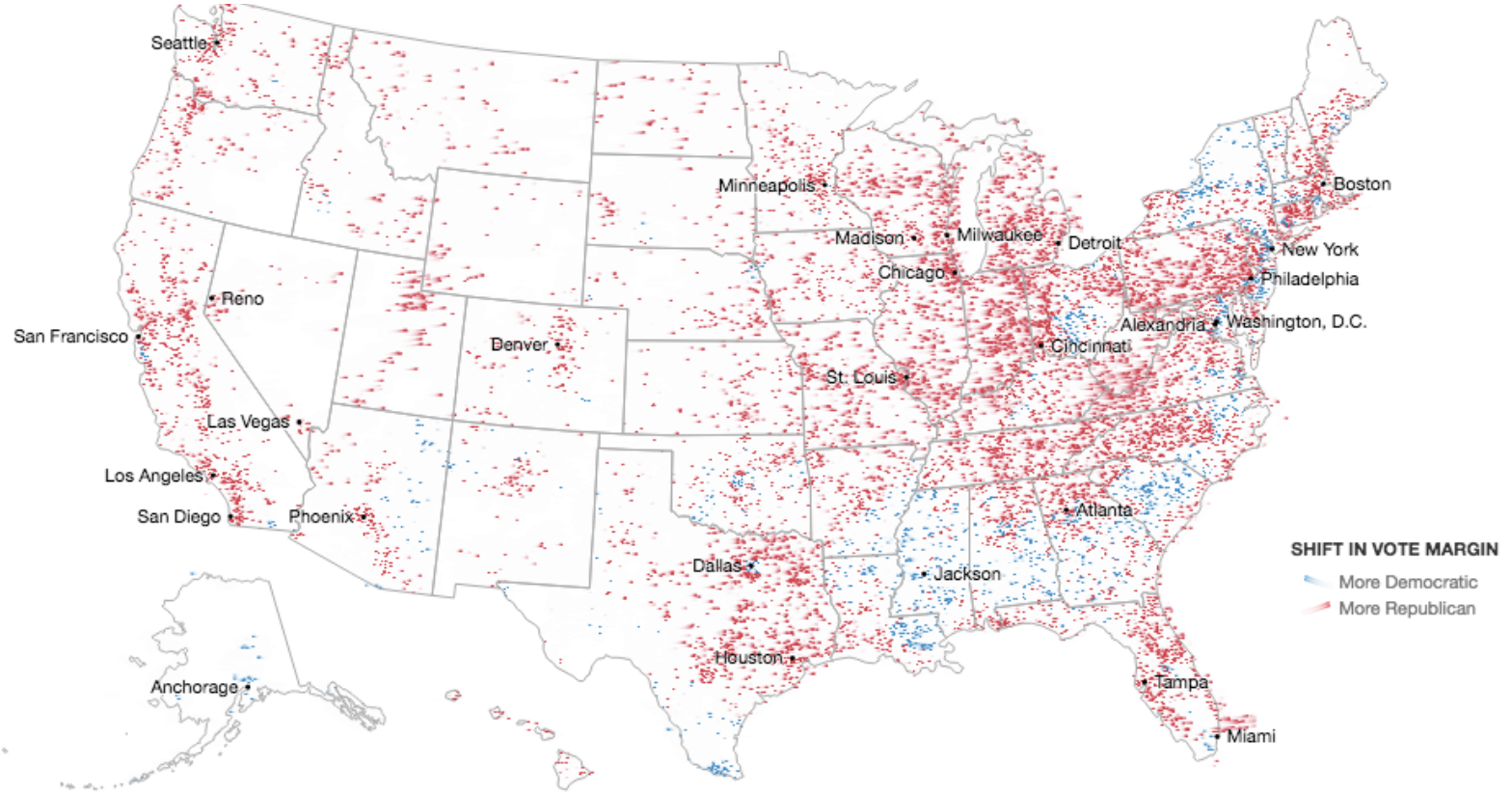
Canvas API

Exercise 2

Practice basic animation techniques.

Think about effective deployment of canvas rendering

- Performance
- Visual expressiveness
- Use in time-dynamic visualizations



http://www.nytimes.com/interactive/2012/11/11/sunday-review/counties-moving.html?_r=0