Interaction with d3

Visweek d3 workshop

Two broad types of interaction with d3.

- Forms

And it doesn't have to be a full-fledged form: controls like drop-down menus, tick boxes, sliders etc.

- Interaction on elements of the chart proper

SVG or HTML elements: clicking, moving the cursor in or out, etc.

Good news!

While they may look different, they both really work the same.

And it's not super complicated.

Here's an example

```
var w=960,h=500,flag=0;
var svg=d3.select("#chart").append("svg").attr("width",w).attr("height",h);
var myRect=svg
    .append( "rect").attr({x:100,y:100,width:100,height:100})
    .style("fill","steelblue");

myRect.on("click",function() {
    flag=1-flag;
    myRect.style("fill", flag?"darkorange":"steelblue");
})
```

Here's an example

var w=960,h=500,**flag=0**;

Note this flag variable initially set to 0.

```
var w=960,h=500,flag=0;
var svg=d3.select("#chart").append("svg").attr("width",w).attr("height",h);
var myRect=svg
    .append( "rect").attr({x:100,y:100,width:100,height:100})
    .style("fill","stee
```

(Here we used a shorthand notation to avoid typing 4 .attr methods. Nothing to do with interaction but hey)

```
var w=960,h=500,flag=0;
var svg=d3.select("#chart").append("svg").attr("width",w).attr("height",h);
var myRect=svg
    .append( "rect").attr({x:100,y:100,width:100,height:100})
    .style("fill","steelblue");

myRect.on("click",function() {

})
```

That's where the action is.
on method, an event, a function.

```
var w=960,h=500,flag=0;
var svg=d3.select("#chart").append("svg").attr("width",w).attr("height",h);
var myRect=svg
    .append( "rect").attr({x:100,y:100,width:100,height:100})
    .style("fill","steelblue");

myRect.on("click",function() {
    flag=1-flag;
    myRect.style("fill", flag?"darkorange":"steelblue");
})
```

And now for the win:
we just toggle the value of flag (0 becomes 1 and vice versa)
then we style our rectangle according to that value : orange if flag is 1, else blue.

on + event + function

That's the general gist of it.

There are few events you should know.

"click" is the workhorse of events. Click anything (a rectangle, text, a shape) and things happen.

"**mouseover**", "**mouseout**" are other good ones.

Great for highlighting stuff and all.

"**change**" is great for forms (the value of the form changed).

Going further: events on groups

By setting an event listener on a "g" element, it will be triggered by any interaction on any element of that group.

Going further: events and data

Can the function access the underlying data of the element? Of course!

If the element has data tied to it, you can do

```
myElement.on("click",function(d) {
   // … operations on data …
})
```

Going further: playing with forms

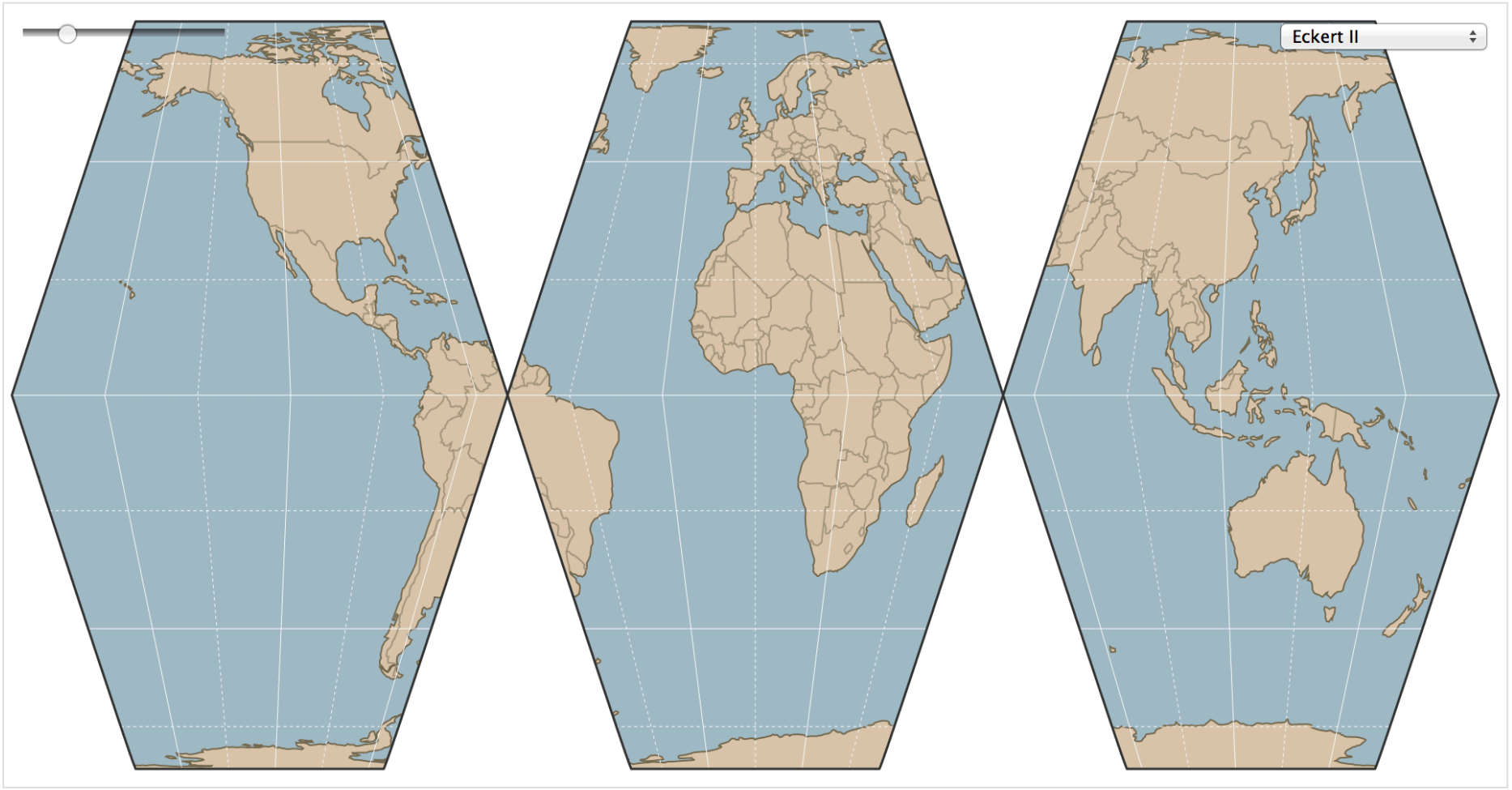
Usually the point of form controls is to access the value selected by the user. So you'll often have something like:

```
myControl.on("change",function() {
    doStuff(this.value);
})
```

this.value will store the value of the control.

Interrupted Projection Transitions

September 28, 2012



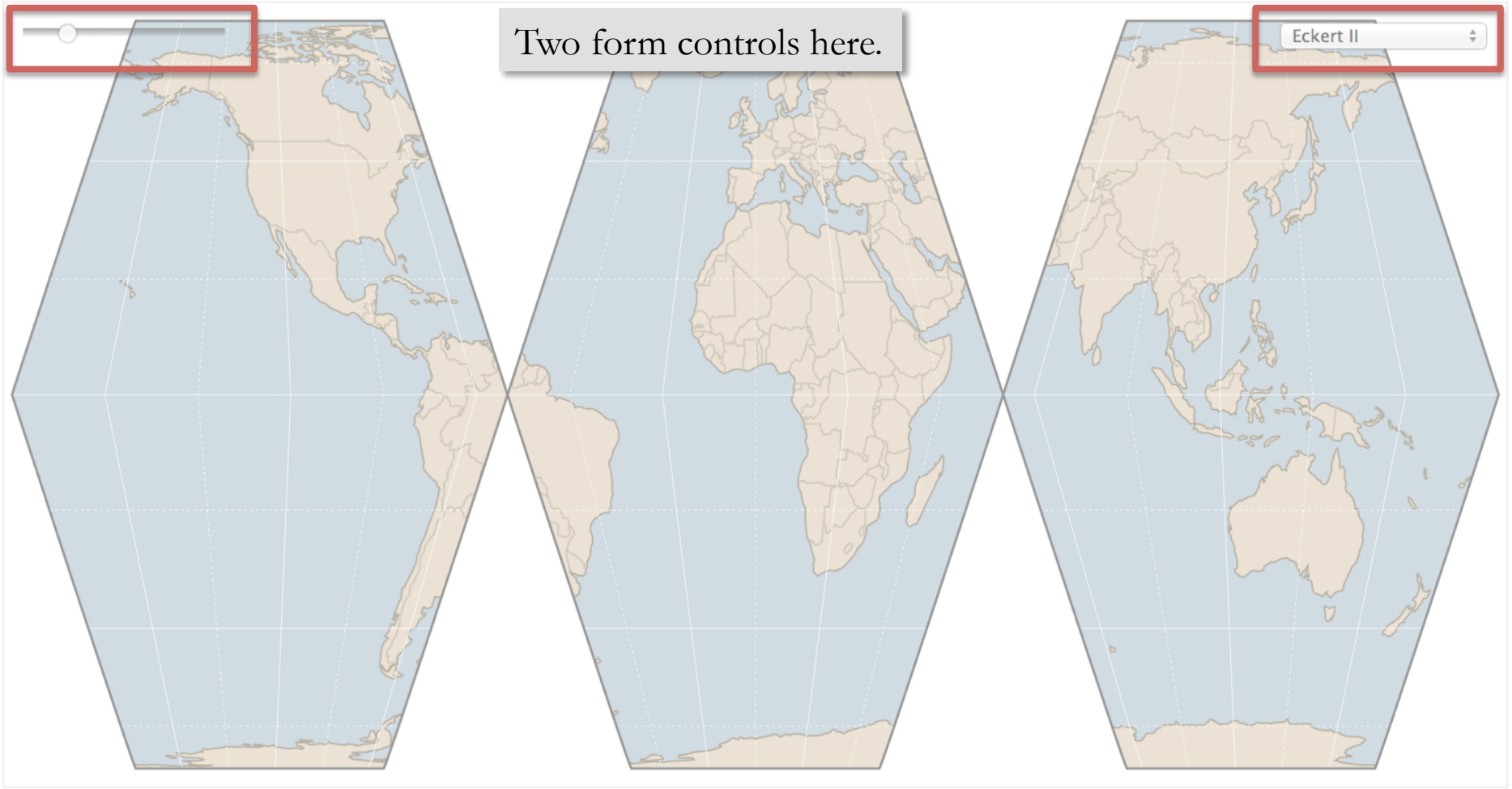
Interrupted projections using the geo.projection D3 plugin.

Open in a new window.

http://bl.ocks.org/3739100

Interrupted Projection Transitions

September 28, 2012



Interrupted projections using the geo.projection D3 plugin.

Open in a new window.

http://bl.ocks.org/3739100

Let's see how it looks like in the code

```
<body>
<select id="projection-menu"></select>
<input type="range" id="gores" step="1" min="1" max="12" value="3">
```

The two controls are created there:
- drop-down (select)
- slider (input with type "range")

Let's see how it looks like in the code

```
d3.select("#gores").on("change", function() {
    gores();
    …
});
…
function gores() {
  var n = +d3.select("#gores").property("value");
  …
}
```

When the slider value changes, the "change" event is triggered and so the gores() function is called.

Then, this function does stuff depending on the position of the slider, which is obtained thusly: by looking up the property "value" of the slider.

Let's see how it looks like in the code

```
menu.selectAll("option")
    .data(options)
  .enter().append("option")
    .text(function(d) { return d.name; });

var menu = d3.select("#projection-menu")
    .on("change", change);

function change() {
  …
  update(options[this.selectedIndex]);
}
```

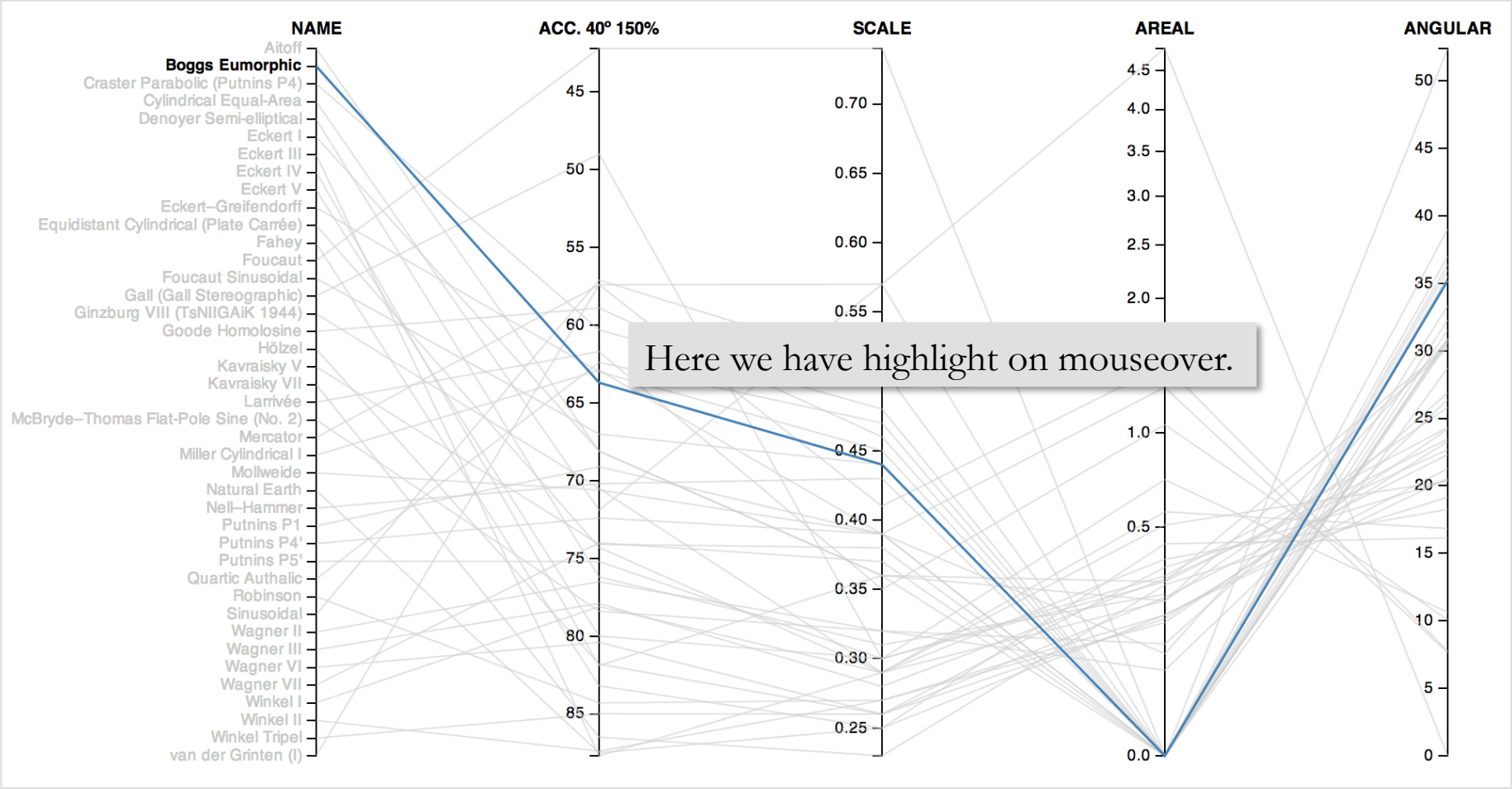
Here, the menu is going to be populated with data.

Then, just as before, a change event will trigger a function : change.

This function will call another one based on which item is selected in the drop down menu.

Map Projection Distortions

September 12, 2012



A comparison of 41 map projections by four different types of distortion. Lower is better. Data transcribed from the Natural Earth Projection.

Open in a new window.

http://bl.ocks.org/3709000

Let's see how it looks like in the code

```
var projection = svg.selectAll(".axis text,.background path,.foreground path")
    .on("mouseover", mouseover)
    .on("mouseout", mouseout);

function mouseover(d) {
  svg.classed("active", true);
  projection.classed("inactive", function(p) { return p !== d; });
  projection.filter(function(p) { return p === d; }).each(moveToFront);
}

function mouseout(d) {
  svg.classed("active", false);
  projection.classed("inactive", false);
}

function moveToFront() {
  this.parentNode.appendChild(this);
}
```

This is our on - event – function

This function is called upon mouseover.
It uses the underlying value of the line (d).

The formatting is done by assigning a CSS class.

And this function is called upon mouseout, it essentially resets the other one.