Übung Open Data:

Datenaktualisierung und Transitionen

Termin 9, 28. April 2016

Dr. Matthias Stürmer und Prof. Dr. Thomas Myrach

Forschungsstelle Digitale Nachhaltigkeit
Institut für Wirtschaftsinformatik
Universität Bern

Terminübersicht Übungen

- 25.02.2016: Informationen zu den Übungen, App-Demos & Einführung in Tools
- 03.03.2016: Einführung Web-Programmierung
- 10.03.2016: Open Data Speed Dating
- 17.03.2016: Einführung D3.js & Daten einbinden in D3.js
- 24.03.2016: Anpassen von bestehenden Apps & Bibliotheken die D3.js verwenden
- 31.03.2016: Osterferien
- 07.04.2016: Daten visualisieren & Layouts
- 14.04.2016: Skalen und Achsen & Responsive Design
- 21.04.2016: User Experience, Usability Patterns
- **28.04.2016: Zwischenpräsentation & Datenaktualisierung und Transitionen**
- 05.05.2016: Auffahrt
- 12.05.2016: Interactivity & Geomapping
- 19.05.2016: 3D Web-Programmierung mit Three.js & Programming Coaching
- 26.05.2016: Abschlusspräsentationen
- 02.06.2016: frei

Zwischenpräsentation

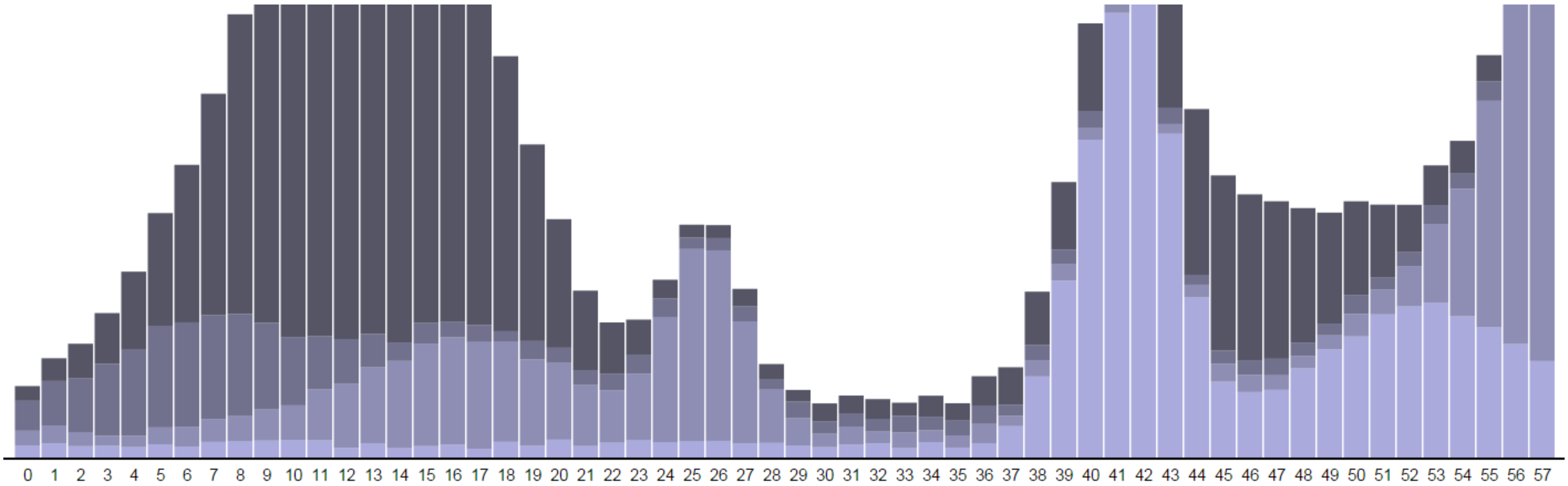
Reihenfolge:

1. Daniela Schmutz und Eva-Maria Künzi
2. Dominik von Fischer und Christoph Gauch
3. Barbara Stutz und Nathalie Bratschi
4. Lukas Günther und Roland Pfister
5. Nicola Lüthi
6. Thomasz Kolonko
7. Stéphanie Würth

> **Jede Gruppe hat 5 Minuten Slot:**
2 Minuten Präsentation und 3 Minuten Feedback

Agenda

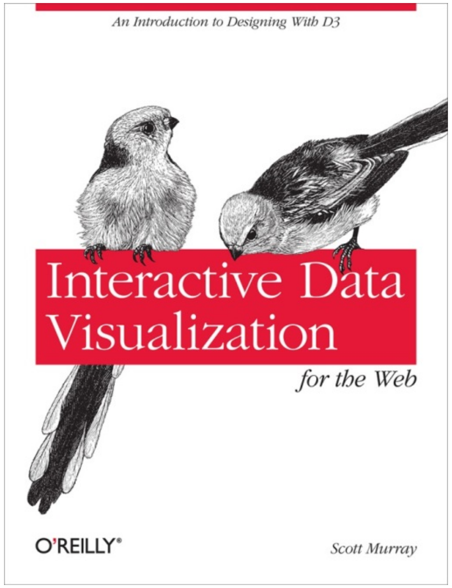
1. **Updating Data**
2. Transitions
3. Updating Axes



Interactive Data Visualization for the Web

- **Chapter 9.**
  Updates, Transitions, and Motion:

- http://chimera.labs.oreilly.com/books/1230000000345/ch09.html



Updating Data

- The simplest kind of update is when **all data values are updated** at the same time *and* the **number of values stays** the same.
  1. Modify the values in your dataset.
  2. Rebind the new values to the existing elements (thereby overwriting the original values).
  3. Set new attribute values as needed to update the visual display.
- Before any of those steps can happen, though, some **event** needs to kick things off.
- We will need a “trigger,” something that happens *after* page load to apply the updates. How about a **mouse click?**

Interaction via Event Listeners

The listener listens for a **click event** occurring on our selection p. When that happens, the listener function is executed:

```
d3.select("p")
    .on("click", function() {
        //Do something mundane and annoying on click
        alert("Hey, don't click that!");
    });
```

Changing the Data

Update **`dataset`** by overwriting its original values:

```
//On click, update with new data
d3.select("p")
    .on("click", function() {

        //New values for dataset
        dataset = [ 11, 12, 15, 20, 18, 17, 16, 18, 23, 25,
                    5, 10, 13, 19, 21, 25, 22, 18, 15, 13 ];

        //Update all rects
        svg.selectAll("rect")
           .data(dataset)
           .attr("y", function(d) {
                return h - yScale(d);
           })
           .attr("height", function(d) {
                return yScale(d);
           });
    });
```

The **`rects`** can maintain their horizontal positions and widths; all we really need to update are their **`heights`** and **`y`** positions. See *03_updates_all_data.html*

Fixing Labels and Colors

We forgot to update the bar **colors.** Fix it by copy-paste from above:

```
.attr("fill", function(d) {
     return "rgb(0, 0, " + (d * 10) + ")";
});
```

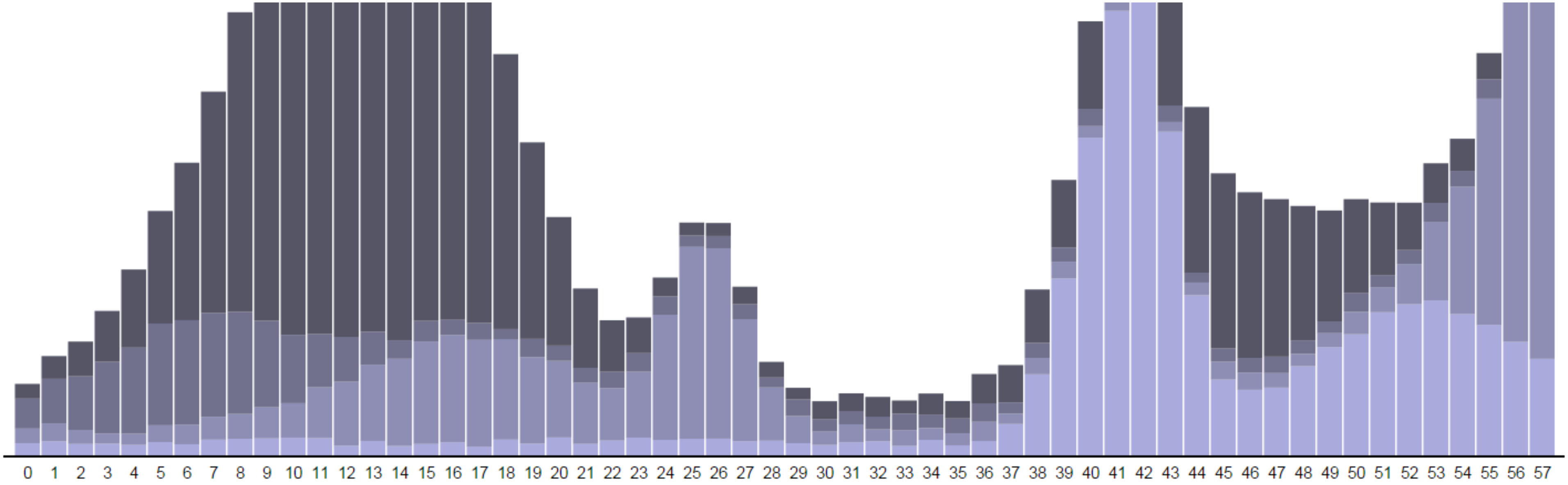
And we forgot to update the **labels.** Fix it by copy-paste from above:

```
svg.selectAll("text")
   .data(dataset)
   .text(function(d) {
        return d;
   })
   .attr("x", function(d, i) {
        return xScale(i) + xScale.rangeBand() / 2;
   })
   .attr("y", function(d) {
        return h - yScale(d) + 14;
   });
```

See *04_updates_all_data_fixed.html*

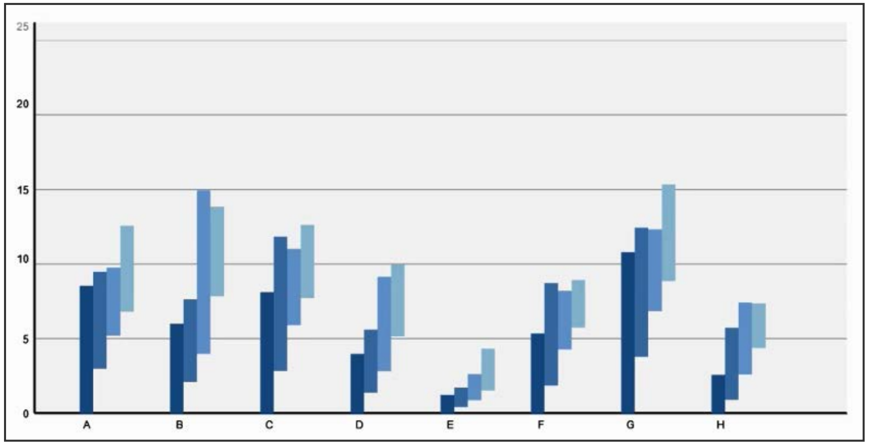
Agenda

1. Updating Data
2. **Transitions**
3. Updating Axes



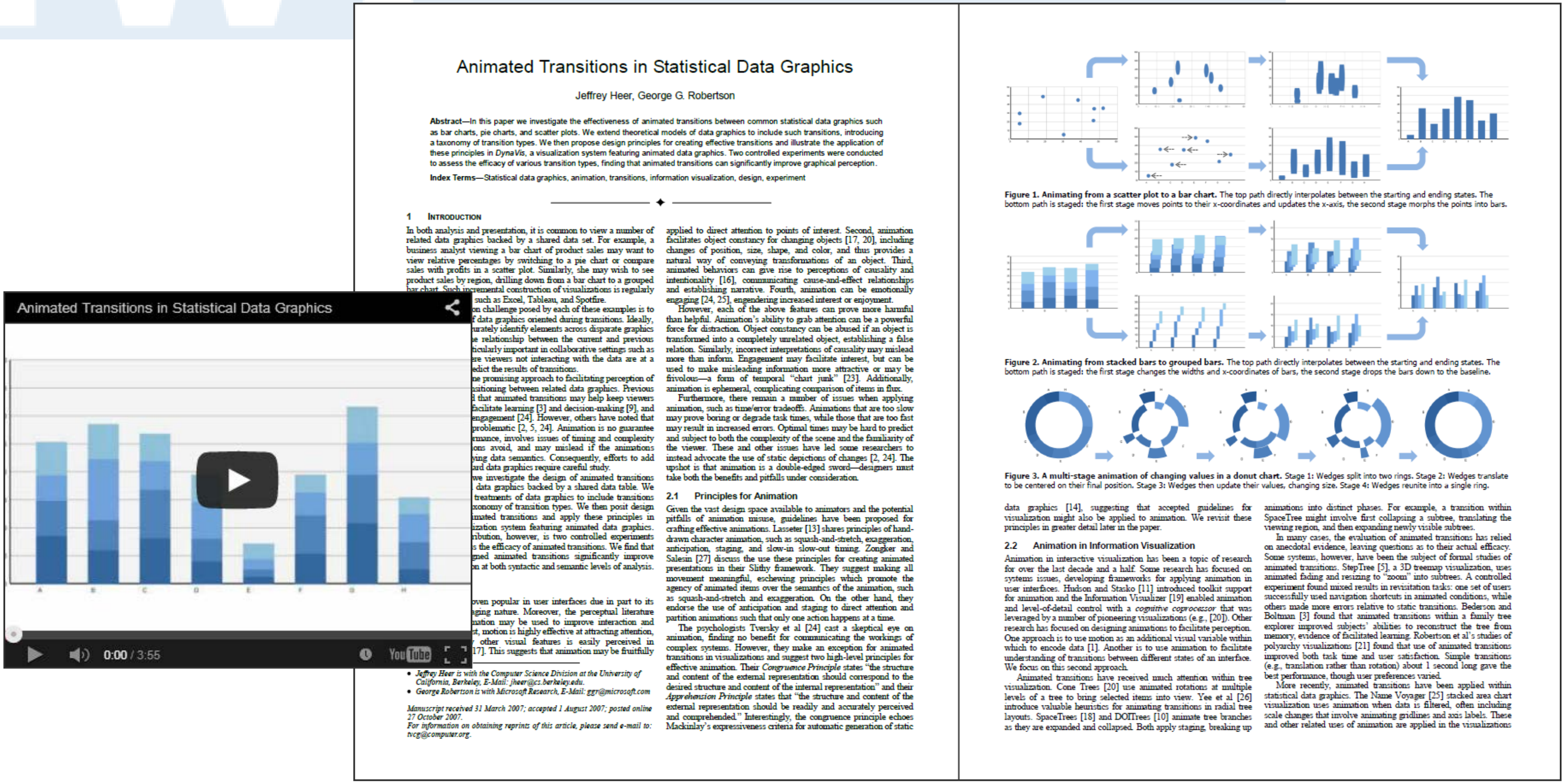
Transitions

- A **transition** is a special type of **selection** where the operators apply smoothly over time rather than instantaneously.
- Transitions may have per-element **delays and durations,** computed using functions of data similar to other operators.
- Why do transitions? **To better explain your data!**
- For example, you can **sort** elements and then **stagger** the transition for better perception of element reordering during the transition: **Heer and Robertson, 2007**



Source: https://github.com/mbostock/d3/wiki/Transitions

Heer and Robertson, 2007



Animated Transitions in Statistical Data Graphics

Jeffrey Heer, George G. Robertson

Abstract—In this paper we investigate the effectiveness of animated transitions between common statistical data graphics such as bar charts, pie charts, and scatter plots. We extend theoretical models of data graphics to include such transitions, introducing a taxonomy of transition types. We then propose design principles for creating effective transitions and illustrate the application of these principles in *DynaVis*, a visualization system featuring animated data graphics. Two controlled experiments were conducted to assess the efficacy of various transition types, finding that animated transitions can significantly improve graphical perception.

Index Terms—Statistical data graphics, animation, transitions, information visualization, design, experiment

✦

1 INTRODUCTION

In both analysis and presentation, it is common to view a number of related data graphics backed by a shared data set. For example, a business analyst viewing a bar chart of product sales may want to view relative percentages by switching to a pie chart or compare sales with profits in a scatter plot. Similarly, she may wish to see product sales by region, drilling down from a bar chart to a grouped bar chart. Such incremental construction of visualizations is regularly such as Excel, Tableau, and Spotfire.

on challenge posed by each of these examples is to f data graphics oriented during transitions. Ideally, urately identify elements across disparate graphics e relationship between the current and previous ticularly important in collaborative settings such as ere viewers not interacting with the data are at a edict the results of transitions.

ne promising approach to facilitating perception of nsitioning between related data graphics. Previous d that animated transitions may help keep viewers facilitate learning [3] and decision-making [9], and engagement [24]. However, others have noted that problematic [2, 5, 24]. Animation is no guarantee rmance, involves issues of timing and complexity ions avoid, and may mislead if the animations ying data semantics. Consequently, efforts to add ard data graphics require careful study.

we investigate the design of animated transitions data graphics backed by a shared data table. We treatments of data graphics to include transitions xonomy of transition types. We then posit design imated transitions and apply these principles in ization system featuring animated data graphics. ribution, however, is two controlled experiments s the efficacy of animated transitions. We find that gned animated transitions significantly improve on at both syntactic and semantic levels of analysis.

oven popular in user interfaces due in part to its aging nature. Moreover, the perceptual literature nation may be used to improve interaction and st, motion is highly effective at attracting attention, other visual features is easily perceived in 17]. This suggests that animation may be fruitfully applied to direct attention to points of interest. Second, animation facilitates object constancy for changing objects [17, 20], including changes of position, size, shape, and color, and thus provides a natural way of conveying transformations of an object. Third, animated behaviors can give rise to perceptions of causality and intentionality [16], communicating cause-and-effect relationships and establishing narrative. Fourth, animation can be emotionally engaging [24, 25], engendering increased interest or enjoyment.

However, each of the above features can prove more harmful than helpful. Animation's ability to grab attention can be a powerful force for distraction. Object constancy can be abused if an object is transformed into a completely unrelated object, establishing a false relation. Similarly, incorrect interpretations of causality may mislead more than inform. Engagement may facilitate interest, but can be used to make misleading information more attractive or may be frivolous—a form of temporal "chart junk" [23]. Additionally, animation is ephemeral, complicating comparison of items in flux.

Furthermore, there remain a number of issues when applying animation, such as time/error tradeoffs. Animations that are too slow may prove boring or degrade task times, while those that are too fast may result in increased errors. Optimal times may be hard to predict and subject to both the complexity of the scene and the familiarity of the viewer. These and other issues have led some researchers to instead advocate the use of static depictions of changes [2, 24]. The upshot is that animation is a double-edged sword—designers must take both the benefits and pitfalls under consideration.

2.1 Principles for Animation

Given the vast design space available to animators and the potential pitfalls of animation misuse, guidelines have been proposed for crafting effective animations. Lasseter [13] shares principles of hand-drawn character animation, such as squash-and-stretch, exaggeration, anticipation, staging, and slow-in slow-out timing. Zongker and Salesin [27] discuss the use these principles for creating animated presentations in their Slithy framework. They suggest making all movement meaningful, eschewing principles which promote the agency of animated items over the semantics of the animation, such as squash-and-stretch and exaggeration. On the other hand, they endorse the use of anticipation and staging to direct attention and partition animations such that only one action happens at a time.

The psychologists Tversky et al [24] cast a skeptical eye on animation, finding no benefit for communicating the workings of complex systems. However, they make an exception for animated transitions in visualizations and suggest two high-level principles for effective animation. Their *Congruence Principle* states "the structure and content of the external representation should correspond to the desired structure and content of the internal representation" and their *Apprehension Principle* states that "the structure and content of the external representation should be readily and accurately perceived and comprehended." Interestingly, the congruence principle echoes Mackinlay's expressiveness criteria for automatic generation of static data graphics [14], suggesting that accepted guidelines for visualization might also be applied to animation. We revisit these principles in greater detail later in the paper.

2.2 Animation in Information Visualization

Animation in interactive visualization has been a topic of research for over the last decade and a half. Some research has focused on systems issues, developing frameworks for applying animation in user interfaces. Hudson and Stasko [11] introduced toolkit support for animation and the Information Visualizer [19] enabled animation and level-of-detail control with a *cognitive coprocessor* that was leveraged by a number of pioneering visualizations (e.g., [20]). Other research has focused on designing animations to facilitate perception. One approach is to use motion as an additional visual variable within which to encode data [1]. Another is to use animation to facilitate understanding of transitions between different states of an interface. We focus on this second approach.

Animated transitions have received much attention within tree visualization. Cone Trees [20] use animated rotations at multiple levels of a tree to bring selected items into view. Yee et al [26] introduce valuable heuristics for animating transitions in radial tree layouts. SpaceTrees [18] and DOITrees [10] animate tree branches as they are expanded and collapsed. Both apply staging, breaking up animations into distinct phases. For example, a transition within SpaceTree might involve first collapsing a subtree, translating the viewing region, and then expanding newly visible subtrees.

In many cases, the evaluation of animated transitions has relied on anecdotal evidence, leaving questions as to their actual efficacy. Some systems, however, have been the subject of formal studies of animated transitions. StepTree [5], a 3D treemap visualization, uses animated fading and resizing to "zoom" into subtrees. A controlled experiment found mixed results in revisitation tasks: one set of users successfully used navigation shortcuts in animated conditions, while others made more errors relative to static transitions. Bederson and Boltman [3] found that animated transitions within a family tree explorer improved subjects' abilities to reconstruct the tree from memory, evidence of facilitated learning. Robertson et al's studies of polyarchy visualizations [21] found that use of animated transitions improved both task time and user satisfaction. Simple transitions (e.g., translation rather than rotation) about 1 second long gave the best performance, though user preferences varied.

More recently, animated transitions have been applied within statistical data graphics. The Name Voyager [25] stacked area chart visualization uses animation when data is filtered, often including scale changes that involve animating gridlines and axis labels. These and other related uses of animation are applied in the visualizations

Figure 1. Animating from a scatter plot to a bar chart. The top path directly interpolates between the starting and ending states. The bottom path is staged: the first stage moves points to their x-coordinates and updates the x-axis, the second stage morphs the points into bars.

Figure 2. Animating from stacked bars to grouped bars. The top path directly interpolates between the starting and ending states. The bottom path is staged: the first stage changes the widths and x-coordinates of bars, the second stage drops the bars down to the baseline.

Figure 3. A multi-stage animation of changing values in a donut chart. Stage 1: Wedges split into two rings. Stage 2: Wedges translate to be centered on their final position. Stage 3: Wedges then update their values, changing size. Stage 4: Wedges reunite into a single ring.

- *Jeffrey Heer is with the Computer Science Division at the University of California, Berkeley, E-Mail: jheer@cs.berkeley.edu.*
- *George Robertson is with Microsoft Research, E-Mail: ggr@microsoft.com*

Manuscript received 31 March 2007; accepted 1 August 2007; posted online 27 October 2007.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

Source: http://vis.berkeley.edu/papers/animated_transitions/

Use transition()

Insert **`transition()`** below where your selection is made, and above where any attribute changes are applied:

```
//Update all rects
svg.selectAll("rect")
   .data(dataset)
   .transition()   // <-- This is new!
   .attr("y", function(d) {
        return h - yScale(d);
   })
   .attr("height", function(d) {
        return yScale(d);
   })
   .attr("fill", function(d) {
        return "rgb(0, 0, " + (d * 10) + ")";
   });
```

See *05_transition.html*

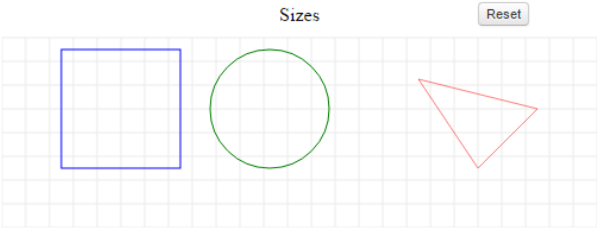
About transition()

- Without **`transition()`**, D3 evaluates every **`attr()`** statement immediately, so the changes in height and fill happen right away.
- When you add **`transition()`**, D3 introduces the element of time.
- Rather than applying new values all at once, D3 **interpolates** between the old values and the new values, meaning it normalizes the beginning and ending values, and **calculates all their in-between states.**
- D3 is also smart enough to recognize and interpolate between **different attribute value formats.**
- For example, if you specified a height of **`200px`** to start but transition to just **`100`** (without the px). Or if a **`blue`** fill turns **`rgb(0,255,0)`**.

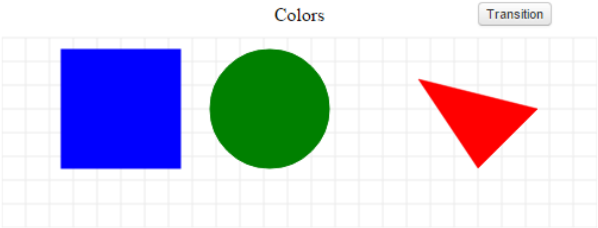
How to fine-tune transitions

- **Duration of a transition: `.duration(1000)`** (in milliseconds)
- **Type of motion: `.ease("VALUE")`**
  - `cubic-in-out` (default): produces gradual acceleration and deceleration
  - `linear`: there is no gradual acceleration and deceleration—the elements simply begin moving at an even pace, and then they stop abruptly.
  - `circle`: Gradual ease in and acceleration until elements snap into place.
  - `elastic`: The best way to describe this one is “sproingy.” [elastisch]
  - `bounce`: Like a ball bouncing, then coming to rest.
- **Short break: `.delay(1000)`** (in milliseconds)

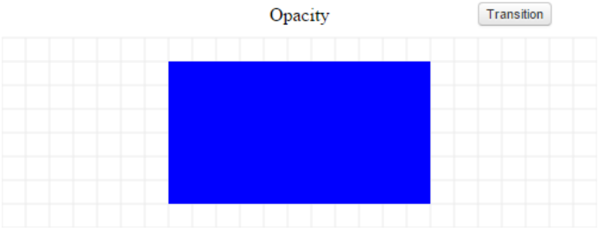
Examples of transitions



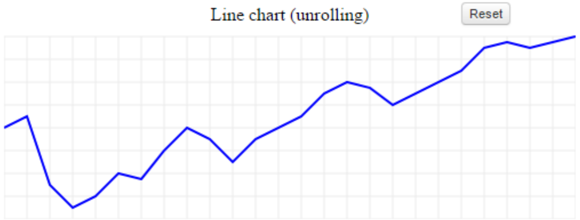
Likewise, when you change size, your object grows (or shrinks). You can use width and height for shapes like rectangles, or r for circles.

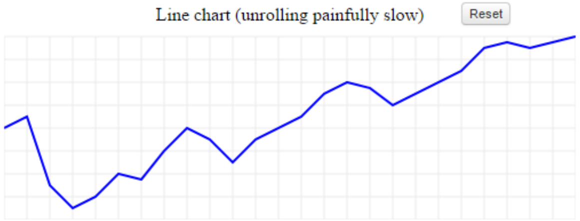


Color is really a numerical attribute, too, and it is indeed possible (and very useful) to transition from one color to another. In svg, color is a style attribute that is defined by fill or stroke.

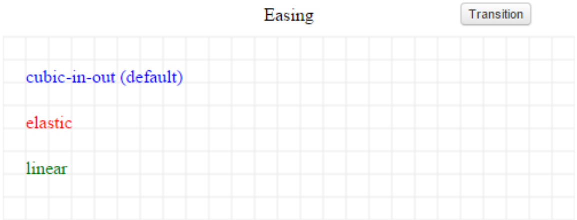


Consider the following two examples (which you'll have to start with the button)





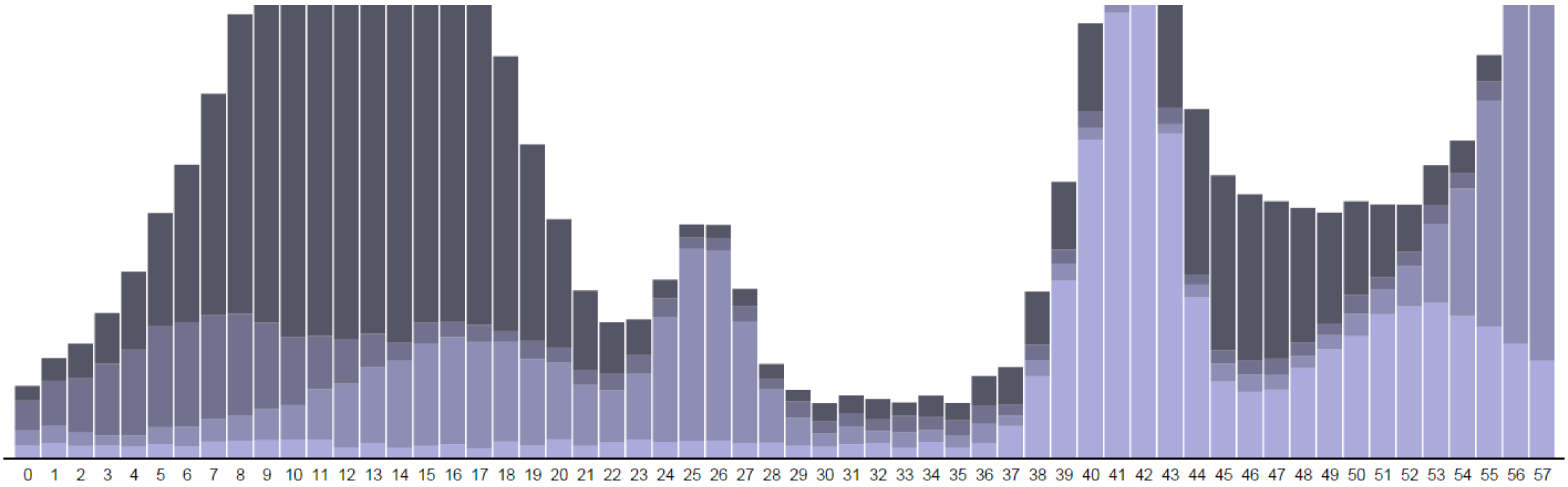
Isn't the second one simply atrocious? You may find it hard to believe that it only wasted 25 seconds of your time.



Source: http://blog.visual.ly/creating-animations-and-transitions-with-d3-js/

Agenda

1. Updating Data
2. Transitions
3. **Updating Axes**



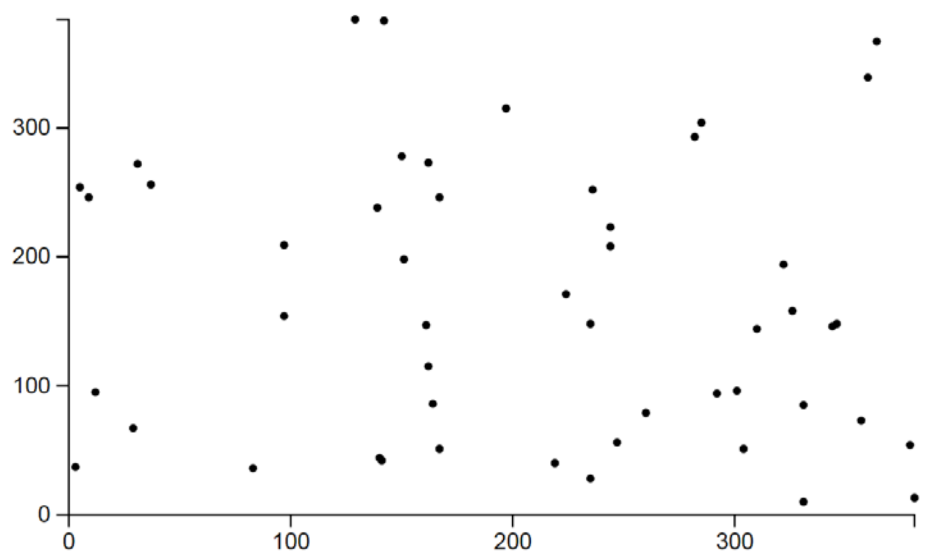
Updating Axes

```
//On click, update with new data
d3.select("p")
  .on("click", function() {

    //New values for dataset
    var numValues = dataset.length;         //Count original length of dataset
    var maxRange = Math.random() * 1000;    //Max range of new values
    dataset = [];                           //Initialize empty array
    for (var i = 0; i < numValues; i++) {   //Loop numValues times
      var newNumber1 = Math.floor(Math.random() * maxRange);
      var newNumber2 = Math.floor(Math.random() * maxRange);
      dataset.push([newNumber1, newNumber2]); //Add new number to array
    }

    //Update scale domains
    xScale.domain([0, d3.max(dataset, function(d) { return d[0]; })]);
    yScale.domain([0, d3.max(dataset, function(d) { return d[1]; })]);
```

See *19_axes_static.html*

Updating Axes

```
   //Update all circles
   svg.selectAll("circle")
      .data(dataset)
      .transition()
      .duration(1000)
      .attr("cx", function(d) {
         return xScale(d[0]);
      })
      .attr("cy", function(d) {
         return yScale(d[1]);
      });
});
```



What's not happening yet is that the axes aren't updating. Fortunately, that is simple to do.

See *19_axes_static.html*

Updating Axes

Add the **class names x and y** to our x- and y-axes, respectively. This will help us select those axes later:

```
//Create x-axis
svg.append("g")
    .attr("class", "x axis")    // <-- Note x added here
    .attr("transform", "translate(0," + (h - padding) + ")")
    .call(xAxis);

//Create y-axis
svg.append("g")
    .attr("class", "y axis")    // <-- Note y added here
    .attr("transform", "translate(" + padding + ",0)")
    .call(yAxis);
```

See *20_axes_dynamic.html*

Updating Axes

Down in the `click` function simply add:

See *20_axes_dynamic.html*

```
//Update x-axis
svg.select(".x.axis")        //Select the x axis
    .transition()            //Initiate a transition
    .duration(1000)          //Set the transition’s duration
    .call(xAxis);            //Call the appropriate axis generator

//Update y-axis
svg.select(".y.axis")
    .transition()
    .duration(1000)
    .call(yAxis);
```

`transition()` handles all the interpolation magic for you — watch those ticks fade in and out!

Each axis generator is already referencing a scale (either **`xScale`** or **`yScale`**). Because those scales are being updated, the axis generators can calculate what the new tick marks should be.