Faculty of Science, Engineering and Technology

COS30045 Data Visualisation

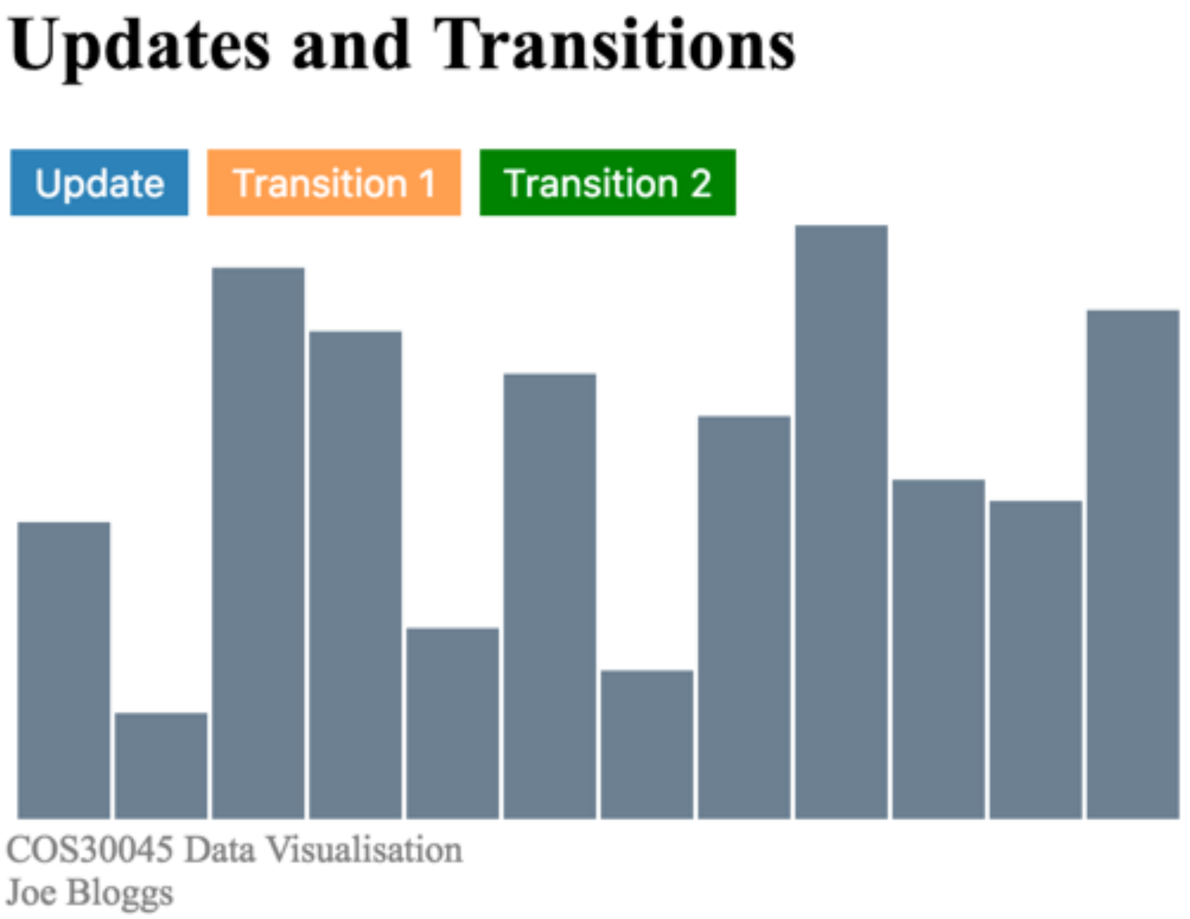
Task 5.2 D3 Transitions

ILO	Create web-based interactive visualisations using real-world data sets.
Aim:	Use D3 to generate smooth transitions between data updates.
Resources:	*Textbook:* Murray Ch 9 Murray on ProQuest Murray on Safari (Make sure you use v2 of Murray as per links above)
To be marked as Complete your submission must:	Submit working code that meets the requirements specified in document below. Demonstrate appropriate use of HTML, CSS and D3. Properly formatted code Well commented code with references to code sourced from web, stack overflow etc. where appropriate. Demonstrate and explain code to tutor in class.
Submission	Submit to Doubtfire • code showing a variety of transitions types Bring code to class to demonstrate to tutor

Note: The functions handling scale have changed between D3 v3 and D3 v4. This is something to be aware of if you are doing your on research into this topic. Make sure you use Murray Ed 2. Code examples from Ed 1 will not work.

Overview

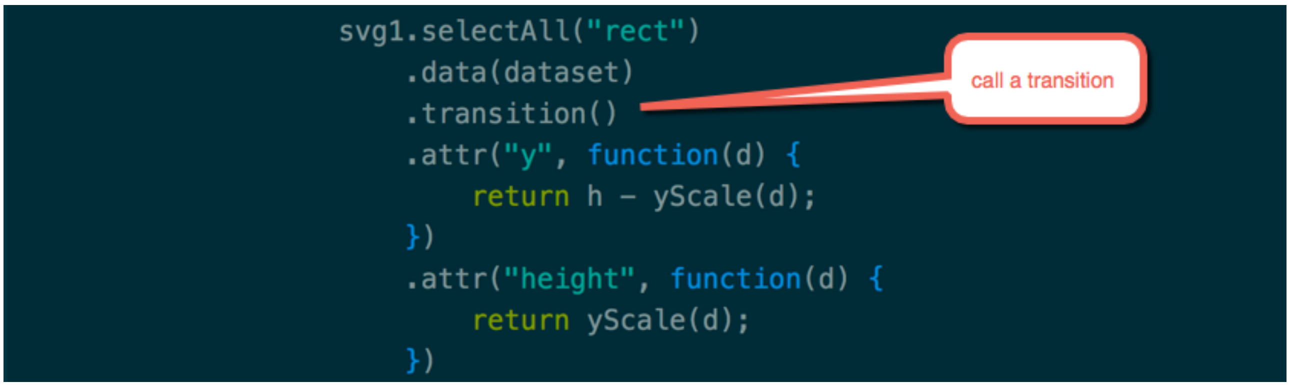
At the end of Task 5.2 we had a chart that updated with random values up to 25. However, the transition between updates was very sharp. Basically the old data set was removed and the new data put on top. In this tutorial we will be working on making the transition between data sets smoother and more engaging by experimenting with a variety of transition animations. It should look a bit like this:



Note: Transition demonstration optional

Step 1: Start with the code from Task 5.1

The first step in generating a smooth transition between data sets is as easy as one line of code:

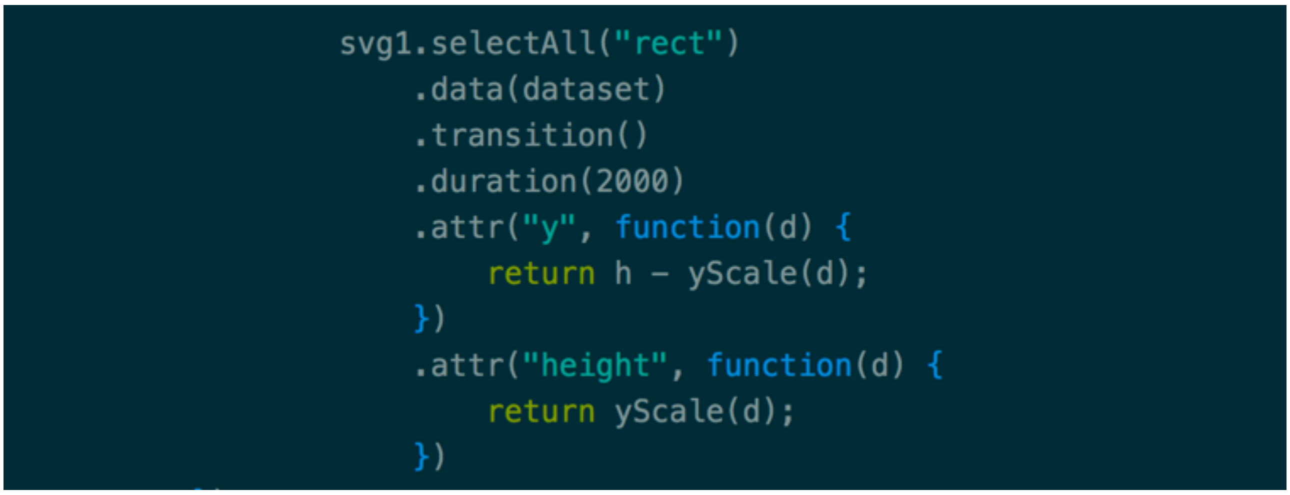


```
svg1.selectAll("rect")
    .data(dataset)
    .transition()
    .attr("y", function(d) {
        return h - yScale(d);
    })
    .attr("height", function(d) {
        return yScale(d);
    })
```

Add `transition()` to your code from 5.1. Save and watch what happens when you update. There is now a smoother transition between the data sets. D3 interpolates the difference between the start and end states and animates the values to change over time. The default time between the start state and end state is 250 ms.

Step 2: Change the duration of the transition

To control how long it takes for the transition to take place you can use `duration ()`. Extend the length of the transition to 2000 ms.



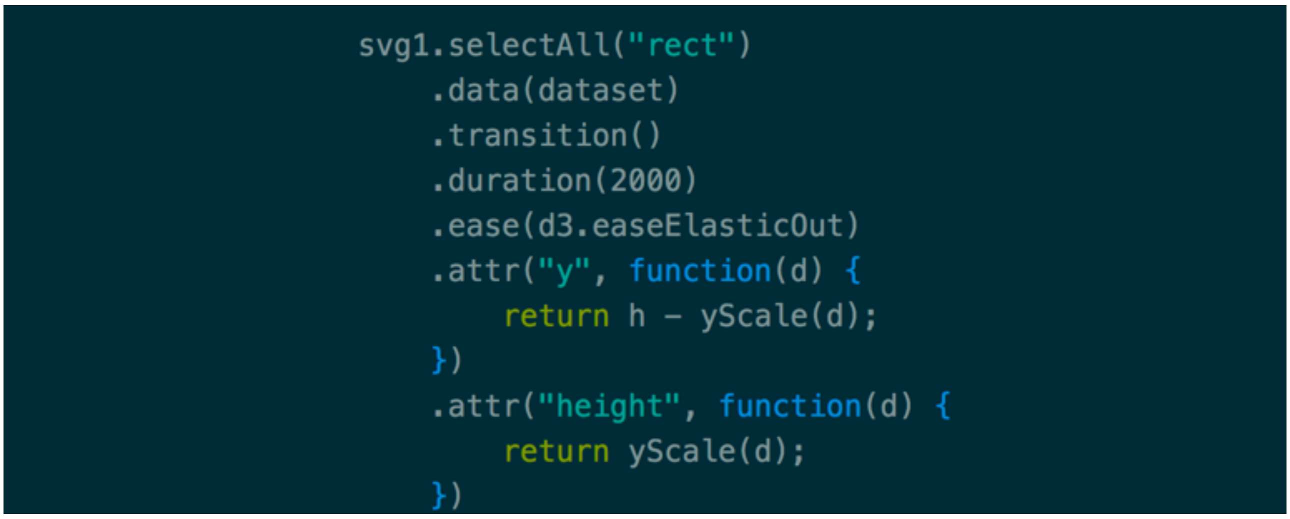
```
svg1.selectAll("rect")
    .data(dataset)
    .transition()
    .duration(2000)
    .attr("y", function(d) {
        return h - yScale(d);
    })
    .attr("height", function(d) {
        return yScale(d);
    })
```

Now you should see the transition in 'slow motion'. Change the duration to 5000 ms and you will be able to see how the animation changes speed during the transition. It starts slow, then speeds up and then finishes slow. This is called easing.

Step 3: Experiment with different types of easing

There are a number of different easing functions in D3. The default is `d3.easeCubicIn-Out`. Experiment with a number of different easing functions. For example;

```
d3.EaseCircleIn
d3.easeCircleOut
d3.easeElasticOut
```



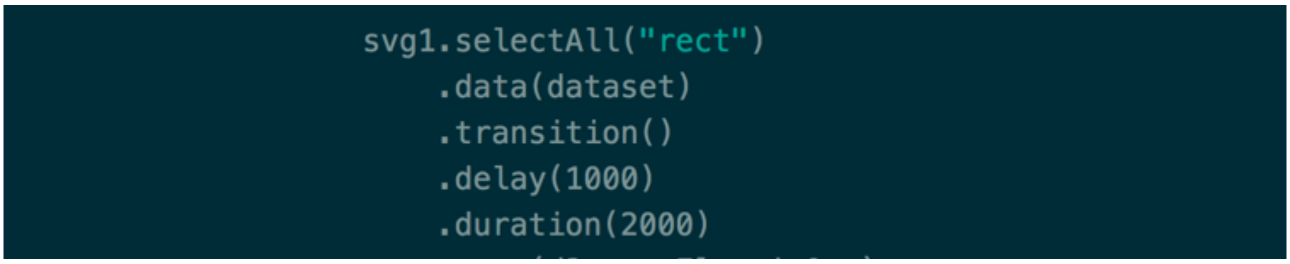
```
svg1.selectAll("rect")
    .data(dataset)
    .transition()
    .duration(2000)
    .ease(d3.easeElasticOut)
    .attr("y", function(d) {
        return h - yScale(d);
    })
    .attr("height", function(d) {
        return yScale(d);
    })
```

See [d3-ease] for a full list of ease transitions.

If you are feeling ambitions try making a demo web page that allows users to try out different transitions.

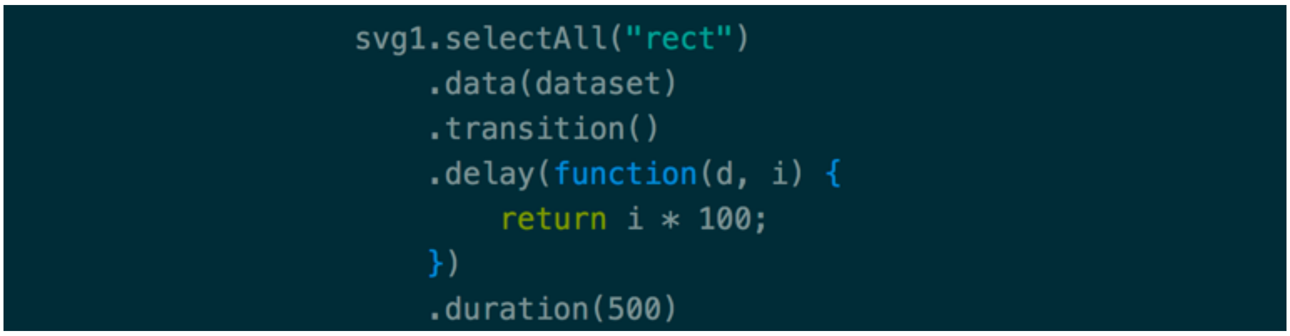
Step 3: Using delays

You can also delay when a transition occurs. We can add a static delay



```
svg1.selectAll("rect")
    .data(dataset)
    .transition()
    .delay(1000)
    .duration(2000)
```

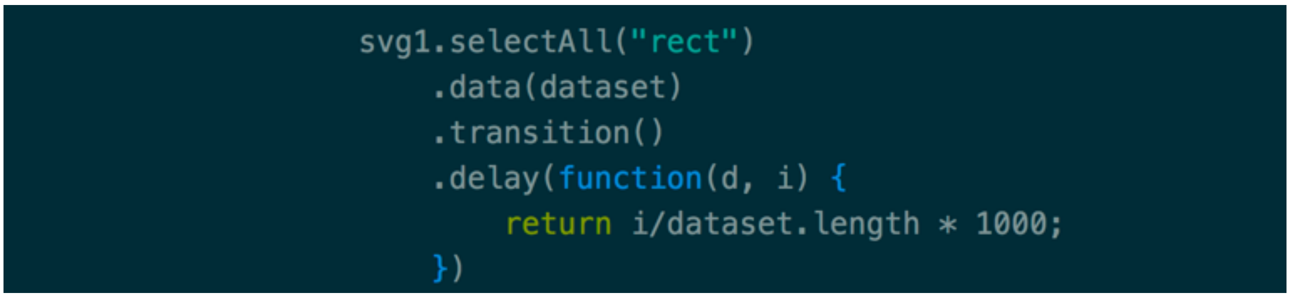
or we could use delays dynamically to stagger the application of a transition. This can make it easier for people to follow the transition.



```
svg1.selectAll("rect")
    .data(dataset)
    .transition()
    .delay(function(d, i) {
        return i * 100;
    })
    .duration(500)
```

This delay function takes the index of the data value and multiplies it by 100 ms so each element is delayed 100s more than the the previous one. This gives a nice staggered animation where the transition rolls a cross the bars.

As a result of this code if the number of data elements increases, then so does the length of the transition. The length of a transition can be scaled by by dividing i by the time you want the transition to take.



```
svg1.selectAll("rect")
    .data(dataset)
    .transition()
    .delay(function(d, i) {
        return i/dataset.length * 1000;
    })
```

Now the animation will always take 1000 ms no matter how many data points are in the data set… a feature which will come in handing in our next task where we will look at adding and removing data.

Be ready to demonstrate your code in class!