Data visualization

COSC 480B

Reyan Ahmed

rahmed1@colgate.edu

Lecture 14

Writing layouts and components

Overview

- Writing a custom legend component
- Writing a custom grid layout
- Adding functionality to make layout and component settings customizable
- Adding interactivity to components

Creating a layout

d3.gridLayout.js

```
d3.gridLayout = () => {
   function processGrid(data) {
   console.log(data)
   }
   return processGrid
}
var grid = d3.gridLayout()
grid([1,2,3,4,5])                    1
```

1 Prints [1,2,3,4,5] to the console

Creating a layout

Here’s a simple spec:

- We want to have a default arrangement of that grid—say, equal numbers of rows and columns.
- We also want to let the user define the number of rows or columns.
- We want the grid to be laid out over a certain size.
- We also need to allow the user to define the size of the grid.

Creating a layout

Updated processGrid function

```
function processGrid(data) {
   var rows = Math.ceil(Math.sqrt(data.length));                    1
   var columns = rows;
   var cell = 0;                                                    2
   for (var rowNumber = 0; rowNumber < rows; rowNumber++) {
      for (var cellNumber = 0; cellNumber < columns; cellNumber ++) {   3
         if (data[cell]) { cellNumber                               4
            data[cell].y = rowNumber                                  5
            cell++                                                6
         }
         else {
           break
         }
      }
   }
   return data
}
```

1 Calculates the number of rows/columns
2 Initializes a variable to walk through the dataset
3 Loops through the rows and columns
4 This assumes the data consists of an array of objects
5 Sets the current datapoint to corresponding row and column
6 Increments the datapoint variable

Creating a layout

Using our grid layout

```
d3.csv("nodelist.csv", makeAGrid)
function makeAGrid(data) {
   var scale =
   d3.scaleLinear().domain([0,5]).range([100,400]);       1
    var salaryScale = d3.scaleLinear().domain([0,300000])
       .range([1,30]).clamp(true)
   var grid = d3.gridLayout();
   var griddedData = grid(data);
   d3.select("svg").selectAll("circle")
      .data(griddedData)
      .enter()
      .append("circle")
      .attr("cx", d => scale(d.x))                    2
      .attr("cy", d => scale(d.y))
          .attr("r", d => salaryScale(d.salary))
      .style("fill", "#93C464");
}
```

1 A scale to fit our grid onto our SVG canvas
2 Sets circles to a scaled position based on the layout's calculated x and y values

Creating a layout

The results of our makeAGrid function that uses our new d3.gridLayout to arrange the data in a grid. In this case, our data consists of employees that are each represented as a green circle laid out on a grid and size by salary.

Creating a layout

Update the grid with more elements

```
var newEmployees = d3.range(14).map(d => {
   var newPerson = {id: "New Person " + d, salary: d * 20000}      1
   return newPerson
})
var doubledArray = data.concat(newEmployees);                      2
var newGriddedData = grid(doubledArray);
d3.select("svg").selectAll("circle")
   .data(newGriddedData)
   .enter()
   .append("circle")                                                3
   .attr("cx", 0)
   .attr("cy", 0)
      .attr("r", d => salaryScale(d.salary))
   .style("fill", "#41A368");
d3.select("svg").selectAll("circle")
   .transition()                                                    4
   .duration(1000)
   .attr("cx", d => scale(d.x))
   .attr("cy", d => scale(d.y))
```

1 Creates 14 new employees with increasing salaries
2 Combines the original dataset with our new dataset
3 Adds any new employees at 0,0
4 Moves all employees (old and new) to their newly computed positions

Creating a layout

The grid layout has automatically adjusted to the size of our new dataset. Notice that our new elements are above the old elements, but our layout has changed in size from a 4 x 4 grid to a 5 x 5 grid, causing the old elements to move to their newly calculated position.

Creating a layout

d3.gridLayout with size functionality

```
d3.gridLayout = function() {
var gridSize = [0,10];                                  1
   var gridXScale = d3.scaleLinear();                        2
var gridYScale = d3.scaleLinear();
function processGrid(data) {
        var rows = Math.ceil(Math.sqrt(data.length));
           var columns = rows;
        gridXScale.domain([1,columns]).range([0,gridSize[0]])   3
           gridYScale.domain([1,rows]).range([0,gridSize[1]])
           var cell = 0
        for (var rowNum = 1; rowNum <= rows; rowNum++) {
          for (var cellNum = 1; cellNum <= columns; cellNum++) {
            if (data[cell]) {
              data[cell].x = gridXScale(cellNum)              4
              data[cell].y = gridYScale(rowNum)
              cell++
            }
```

1 Initializes the variable with a default value
2 Creates two scales but doesn't define their range or domain
3 Defines the range and domain each time the layout is called
4 Applies the scaled values as x and y

Creating a layout

```
          else {
            break
          }
        }
      }
    return data;
  }
  processGrid.size = (newSize) => {                    5
        if (!arguments.length) return gridSize
      gridSize = newSize
      return this
  }
  return processGrid
}
```

5 Getter/setter function for layout size

Creating a layout

Calling the new grid layout

```
var grid = d3.gridLayout();
grid.size([400,400]);                                1
var griddedData = grid(data);
d3.select("svg")
   .append("g")
   .attr("transform", "translate(50,50)")
   .selectAll("circle").data(griddedData)
   .enter()
   .append("circle")
   .attr("cx", d => d.x)                             2
   .attr("cy", d => d.y)
   .attr("r", d => salaryScale(d.salary))
var newEmployees = [];
for (var x = 0;x < 14;x++) {
     var newPerson = {id: "New Person " + x, salary: x * 20000};
     newEmployees.push(newPerson);
}
```

1 Sets layout size
2 Position circles with their x/y values

Creating a layout

```
var doubledArray = data.concat(newEmployees)
    var newGriddedData = grid(doubledArray)
    d3.select("g").selectAll("circle").data(newGriddedData)
       .enter()
       .append("circle")
       .attr("cx", 0)
       .attr("cy", 0)
       .attr("r", d => salaryScale(d.salary))
       .style("fill", "#41A368")
    d3.select("g").selectAll("circle")
       .transition()
       .duration(1000)
       .attr("cx", d => d.x)
       .attr("cy", d => d.y)
       .on("end", resizeGrid1)                    3
```

3 At the end of the transition, calls resizeGrid1

Creating a layout

The resizeGrid1() func

```
function resizeGrid1() {
  grid.size([200,400]);                1
  grid(doubledArray);
  d3.select("g").selectAll("circle")
    .transition()
    .duration(1000)
    .attr("cx", d => d.x)
    .attr("cy", d => d.y)
    .on("end", resizeGrid2)
};
```

1 Changes the size, reapplies the layout, and updates the display

Creating a layout

```
function resizeGrid2() {
  grid.size([400,200])                2
  grid(doubledArray)
  d3.select("g").selectAll("circle")
    .transition()
    .duration(1000)
    .attr("cx", d => d.x)
    .attr("cy", d => d.y)
}
```

2 Again, with a different size

Creating a layout

The grid layout run with a 400 x 400 size setting

Creating a layout

The grid layout run in a 200 x 200 size (left) and a 400 x 200 size (center), and a 200 x 400 size (right)

Creating a layout

Layout code for calculating height and width of grid cells

```
var gridCellWidth = gridSize[0] / columns;
var gridCellHeight = gridSize[1] / rows;
//other code
  for (var i = 1; i <= rows; i++) {
    for (var j = 1; j <= columns; j++) {
      if (data[cell]) {
        data[cell].x = gridXScale(j);
        data[cell].y = gridYScale(i);
        data[cell].height = gridCellHeight;       1
        data[cell].width = gridCellWidth;         1
        cell++;
      }
      else {
        break;
      }
    }
  }
```

1 New code to set the height and width of the grid cells so we can use rectangles instead of circles

Creating a layout

Appending rectangles with our layout

```
d3.select("svg")
  .append("g")
  .attr("transform", "translate(50,50)")
  .selectAll("circle").data(griddedData)
  .enter()
  .append("rect")
    .attr("x", d => d.x - (d.width / 2))                1
    .attr("y", d => d.y - (d.height / 2))               1
    .attr("width", d => d.width)                        1
    .attr("height", d => d.height)                      1
    .style("fill", "#93C464")

...
```

1 The updated grid layout calculates the space each grid cell takes up

Creating a layout

```
d3.select("g").selectAll("rect").data(newGriddedData)
    .enter()
    .append("rect")
    .style("fill", "#41A368")

d3.select("g").selectAll("rect")
      .transition()
      .duration(1000)
      .attr("x", d => d.x - (d.width / 2))              2
      .attr("y", d => d.y - (d.height / 2))              2
      .attr("width", d => d.width)                      2
      .attr("height", d => d.height)                     2
      .on("end", resizeGrid1);                          3
```

2 Height and width are used to set the rectangle size and position with an animated transition
3 At the end of the animation, trigger another animation to show how the updated settings of the grid size can be used to dynamically update display of the grid

Creating a layout

```
function resizeGrid1() {                                    4
      grid.size([200,400]);
      grid(doubledArray);
      d3.select("g").selectAll("rect")
        .transition()
        .duration(1000)
        .attr("x", d => d.x - (d.width / 2))
        .attr("y", d => d.y - (d.height / 2))
        .attr("width", d => d.width)
        .attr("height", d => d.height)
        .on("end", resizeGrid2);
};
```

4 Each of these gives new rectangle sizes based on the new grid.size settings

Creating a layout

```
function resizeGrid2() {                                4
  grid.size([400,200]);
  grid(doubledArray);
  d3.select("g").selectAll("rect")
    .transition()
    .duration(1000)
    .attr("x", d => d.x - (d.width / 2))
    .attr("y", d => d.y - (d.height / 2))
    .attr("width", d => d.width)
    .attr("height", d => d.height)
};
```

4 Each of these gives new rectangle sizes based on the new grid.size settings

Creating a layout

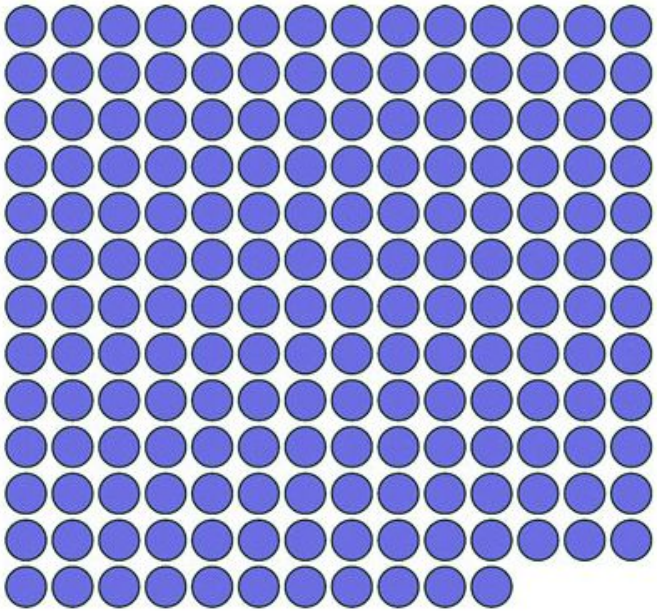
The three states of the grid layout using rectangles for the grid cells

Writing your own components

- We have seen some components
  - Scale
  - Brush
  - Etc.
- We are going to build a legend component
- We will stick with our grid layout
- But we will use world.geojson file as the data source

Loading sample data

The countries of the world as a grid



Loading sample data

Loading the countries of the world into a grid

```
d3.json("world.geojson ", data => {
    makeAGrid(data);
  })
function makeAGrid(data) {
  var grid = d3.gridLayout();
  grid.size([300,300]);
  var griddedData = grid(data.features);
  griddedData.forEach(country => {
    country.size = d3.geoArea(country);          1
  });
```

1 Calculates the area of each country and appends that to the datapoint

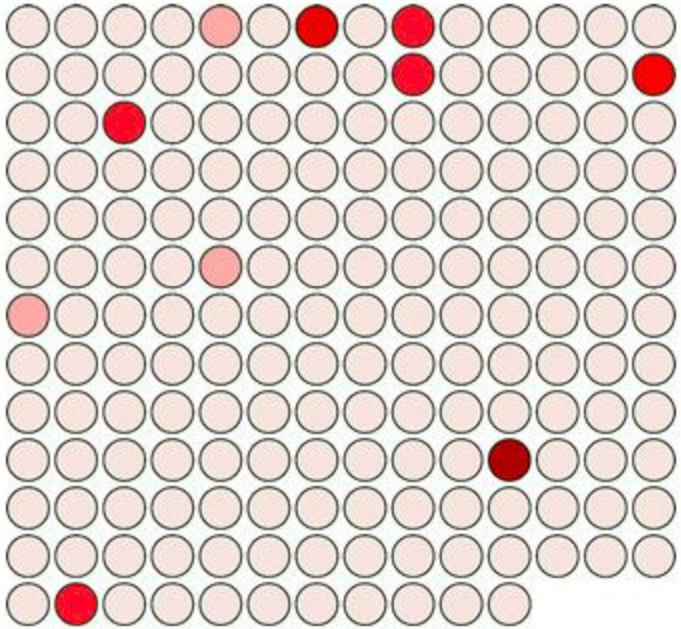
Loading sample data

```
d3.select("svg")
      .append("g")
      .attr("transform", "translate(50,50)")
      .selectAll("circle")
      .data(griddedData)
      .enter()
      .append("circle")                          2
      .attr("cx", d => d.x)
      .attr("cy", d => d.y)
      .attr("r", 10)
      .style("fill", "#75739F")
      .style("stroke", "#4F442B")
      .style("stroke-width", "1px");
  };
```

2 Appends a circle for each country

Loading sample data

Circles representing countries colored by area



Loading sample data

Changing the color of our grid

```
var griddedData = d3.selectAll("circle").data();                1
var sizeExtent = d3.extent(griddedData, d => d.size)
var countryColor = d3.scaleQuantize()
            .domain(sizeExtent).range(colorbrewer.Reds[7])
d3.selectAll("circle").style("fill", d => countryColor(d.size))
```

1 Gets the data array bound to our circles

Loading sample data

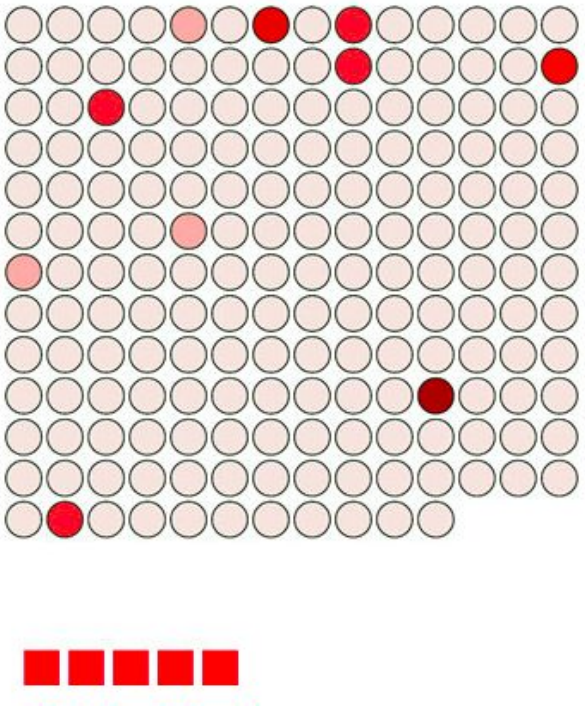
A simple component

```
d3.simpleLegend = () => {
    function legend(gSelection) {          1
      var testData = [1,2,3,4,5];
      gSelection.selectAll("rect")         2
         .data(testData)
      .enter()
      .append("rect")
      .attr("height", 20)
      .attr("width", 20)
      .attr("x", (d,i) => i *25)
      .style("fill", "red")
      return this;
    }
    return legend;
  };
```

1 A component is sent a selection with .call()
2 Appends to that selection a set of rectangles

Loading sample data

The new legend component, when called by a <g> element placed below our grid, creates five red rectangles.



Loading sample data

```
var newLegend = d3.simpleLegend();
d3.select("svg").append("g")
   .attr("id","legend")
   .attr("transform", "translate(50,400)")
   .call(newLegend);
```

Linking components to scales

```
countryColor.range()                    1
```

1 [“#fee5d9”, “#fcbba1”, “#fc9272”, “#fb6a4a”, “#ef3b2c”, “#cb181d”, “#99000d”]

Linking components to scales

```
countryColor.invertExtent("#fee5d9")        1
```

1 [0.000006746501002759535, 0.05946855349777645]

Linking components to scales

Updated legend function

```
d3.simpleLegend = function() {
  var data = [];
  var size = [300,20];                                    1
  var xScale = d3.scaleLinear();                            2
  var scale;                                              3
  function legend(gSelection) {
  createLegendData(scale);                                  4
  var xMin = d3.min(data, d => d.domain[0])                     5
  var xMax = d3.max(data, d => d.domain[1])
  xScale.domain([xMin,xMax]).range([0,size[0]])                   6
  gSelection.selectAll("rect")
  .data(data)
  .enter()
  .append("rect")
  .attr("height", size[1])                                  7
  .attr("width", d => xScale(d.domain[1]) -  xScale(d.domain[0]))
  .attr("x", d => xScale(d.domain[0]))
  .style("fill", d => d.color);
  return this;
  };
```

1 Sets a default size
2 Initializes an x-axis scale but doesn't set domain or range
3 The scale that will be sent to the component
4 Calls the function to process the scale into a data array
5 Calculates the min/max of the scale data
6 Sets the x-axis scale
7 Draws rectangles based on component settings and scale data

Linking components to scales

```
  function createLegendData(incScale) {                    8
    var rangeArray = incScale.range();
    data = [];
    for (var x in rangeArray) {
      var colorValue = rangeArray[x];
      var domainValues = incScale.invertExtent(colorValue);
      data.push({color: colorValue, domain: domainValues})
    }
  };
  legend.scale = function(newScale) {                      9
    if (!newScale) return scale;
    scale = newScale;
    return this;
  };
  return legend;
};
```

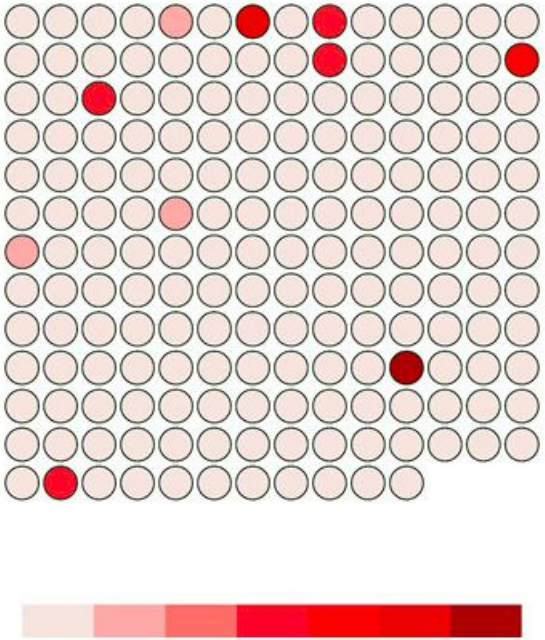
8 Processes the scale into a data array
9 Setter/getter to set the legend's scale

Linking components to scales

```
var newLegend =
d3.simpleLegend().scale(countryColor);
d3.select("svg").append("g")
   .attr("transform","translate(50,400)")
   .attr("id", "legend").call(newLegend);
```

Linking components to scales

The updated legend component is automatically created, with a <rect> element for each band in the quantize scale that’s colored according to that band’s color.



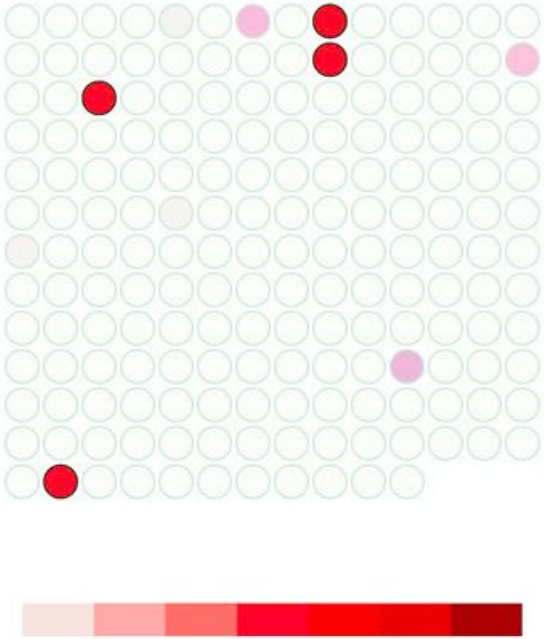
Linking components to scales

Legend interactivity

```
d3.select("#legend").selectAll("rect").on("mouseover",
legendOver);
function legendOver(d) {
    d3.selectAll("circle")
        .style("opacity", p => {
          if (p.size >= d.domain[0] && p.size <= d.domain[1])
{
                return 1;
          } else {
                return .25;
          }
        });
};
```

Linking components to scales

The legendOver behavior highlights circles falling in a particular band and deemphasizes the circles not in that band by making them transparent.



Linking components to scales

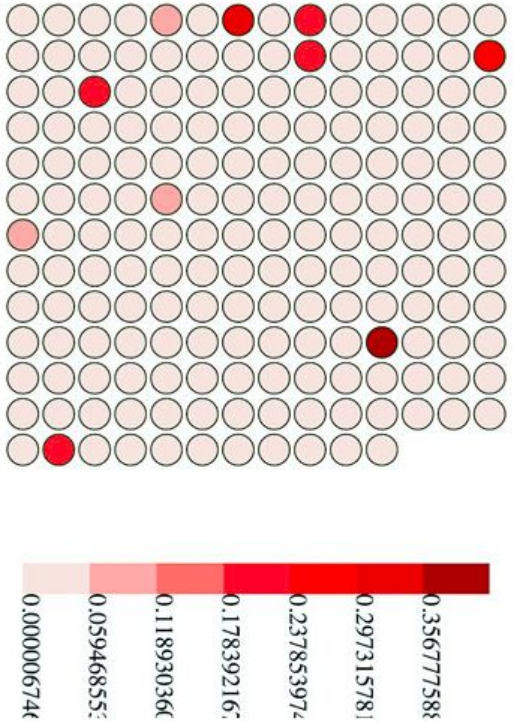
Text labels for legend

```
gSelection.selectAll("text")
   .data(data)
   .enter()
   .append("g")                                          1
   .attr("transform", d => "translate(" +
xScale(d.domain[0]) + ","
     + size[1] + ")")
   .append("text")
      .attr("transform", "rotate(90)")
      .text(d => d.domain[0]);
```

1 The text element needs to be placed in a g so that it can be translated and then rotated; otherwise, it'll be rotated and then translated, which would place it at the translation relative to its new rotation (taking the text off the page)

Linking components to scales

Our legend with rudimentary labels



Adding component labels

Title and unit attributes of a legend

```
var title = "Legend";
var numberFormat = d3.format(".4n");
var units = "Units";                          1
//other code
legend.title = function(newTitle) {
  if (!arguments.length) return title;
  title = newTitle;
  return this;
};                                          2
legend.unitLabel = function(newUnits) {
  if (!arguments.length) return units;
  units = newUnits;
  return this;
};
legend.formatter = function(newFormatter) {
  if (!arguments.length) return numberFormat;
  numberFormat = newFormatter;
  return this;
};
```

1 These are added right after var scale inside the d3.simpleLegend function
2 All these functions are added right after legend.scale

Adding component labels

Updated legend drawing code

```
gSelection.selectAll("line")                                    1
   .data(data)
   .enter()
   .append("line")
   .attr("x1", d => xScale(d.domain[0]))                              2
   .attr("x2", d => xScale(d.domain[0]))
   .attr("y1", 0)
   .attr("y2", size[1] + 5)
   .style("stroke", "black")
   .style("stroke-width", "2px");
```

1 This follows your existing code to draw the legend <rect> elements, and updates the text
2 Each line is drawn at the breakpoint and drawn a little lower to “point” at the breakpoint value

Adding component labels

```
gSelection.selectAll("text")
   .data(data)
   .enter()
   .append("g")
   .attr("transform",
    d => `translate(${(xScale(d.domain[0]))},${(size[1] +
20)})`)
   .append("text")
   .style("text-anchor", "middle")
   .text(d => numberFormat(d.domain[0]));                    3
gSelection.append("text")
   .attr("transform",
    d => `translate(${(xScale(xMin))},${(size[1] - 30)})`)
   .text(title);                                             4
gSelection.append("text")
   .attr("transform",
    d => `translate(${(xScale(xMax))},${(size[1] + 20)})`)
   .text(units);                                             5
```

3 Anchors your unrotated labels at the midpoint and formats the value according to the set formatter
4 Adds a fixed, user-defined title above the legend rectangles and at the minimum value position
5 Adds a fixed, user-defined unit label on the same line as the labels but at the maximum value position

Adding component labels

Calling the legend with title and unit setting

```
var newLegend = d3.simpleLegend()
   .scale(countryColor)
   .title("Country Size")
   .formatter(d3.format(".2f"))
   .unitLabel("Steradians");                                    1
d3.select("svg").append("g").attr("transform",
"translate(50,400)")
      .attr("id", "legend")
      .call(newLegend);                                         2
```

1 Sets the legend title and unit labels and formats to reflect the data being visualized
2 This part is unchanged

Adding component labels

Our legend with title, unit labels, appropriate number formatting, and additional graphical elements to highlight the breakpoints