

# D3.js – ToBeContinued

张松海、张少魁、周文洋、蔡韵

数据可视化 – D3.js

清华大学 可视媒体研究中心

# This is the 'Final' Chapter for D3.js

- Color-Gradient
- 添加图片
- 自定义SVGPattern (纹理)
- 主题河流
- Piecewise Scale
- 本次包含内容略琐碎…非常细节的地方时间原因不会全部涉及，但细节涉及到的问题均会在讲义中给出并尽可能伴随已解决的源代码。
- Formatter – D3中的格式化
- FileAttachMent
- d3.drag
- JS的异步机制
- 灵活使用Transition
- \* Transition.Tween (参考时间)
- D3的局限性
- D3.js讲解后的支持
- ... ..

# Color-Gradient

- 使用颜色的‘梯度’设置图元的填充
  - ‘fill’属性不是单纯可以使用‘颜色’进行填充

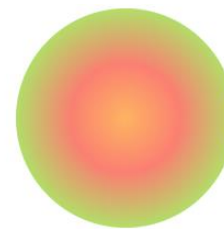
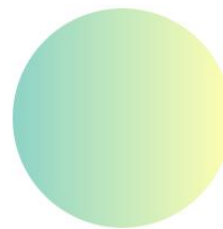
- Linear -> 线性‘梯度’

- Radial -> 从中心向外扩散的‘梯度’

- 设置图元的梯度填充？

- 1, 对填充进行定义并赋予索引
  - 2, **使用索引设置图元的fill属性**

- Code: <https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/html-tutorial/hello-colorgrad.html>



```
<defs>
<linearGradient id="linearGrad">
  <stop offset="0%" stop-color="#8dd3c7" stop-opacity="1"></stop>
  <stop offset="100%" stop-color="#ffffb3" stop-opacity="1"></stop>
</linearGradient>
</defs>
```

<defs>表示定义，即会被重复利用或者索引到的东西。

```
<svg width="200" height="200">
  <circle cx="100" cy="100" r="50" fill="url(#linearGrad)"/>
</svg>
```

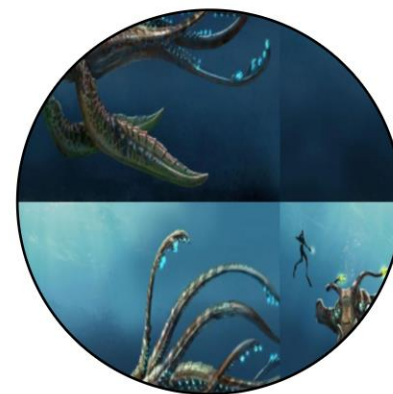
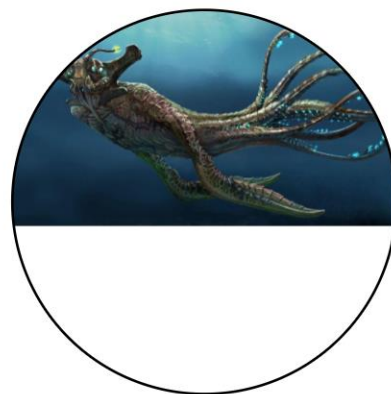
# Color-Gradient

- 使用D3设置Color-Gradient?
  - 本质上就是对纹理设置的‘增加’与‘修改’
  - D3对图元的操作不仅仅适用于SVG的‘图元’
  - Code: <https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/color-gradient.html>
- LinearGradient如何设置梯度的偏移与方向?
- RadialGradient如何设置‘圆心’?
- ‘梯度’是否可以有多个?
  - 多种颜色的交替渐变?
- (答案均在上页代码中)

```
var defs = svg.append("defs");
var gradient = defs.append("linearGradient")
    .attr("id", "linearGradient")
    // .attr("x1", "0%")
    // .attr("x2", "100%")
    // .attr("y1", "0%")
    // .attr("y2", "100%");
gradient.append("stop")
    // .attr('class', 'start')
    .attr("offset", "0%")
    .attr("stop-color", "#226B00")
    .attr("stop-opacity", 1);
gradient.append("stop")
    // .attr('class', 'end')
    .attr("offset", "100%")
    .attr("stop-color", "#DD6B66")
    .attr("stop-opacity", 1);
```

# 添加一幅图片

- Code: <https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/html-tutorial/hello-img.html>



# 添加一幅图片

- 添加一个纯粹的图片HTML元素:

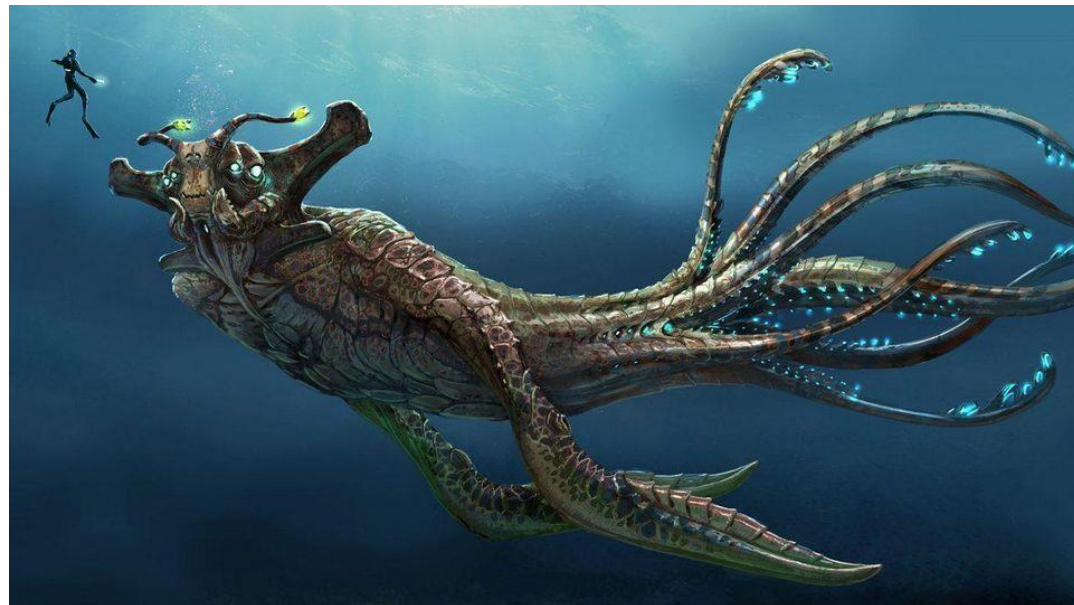
```
svg.append('image').attr('id', 'myimg')  
.attr("x", 600).attr("y", 350)  
.attr('width', 300).attr('height', 300)  
.attr('preserveAspectRatio', 'none')  
.attr("href", 'ff7.jpg');
```

- 一张图片自身以link的形式给出;
- ‘图片’(image)本身也是一个HTML元素;
- href表示图片的路径;
- preserveAspectRatio表示图片的‘对齐’与‘缩放’;



# preserveAspectRatio

- 此属性表示图片的‘对齐’与‘缩放’;
- 图片下方表示这个属性的可能取值
- 取值不限于下方的示例
- （不做要求）meet和slice本质上是  
对viewbox这个属性进行修改



原图



none



xMidYMid slice



xMidYMid meet



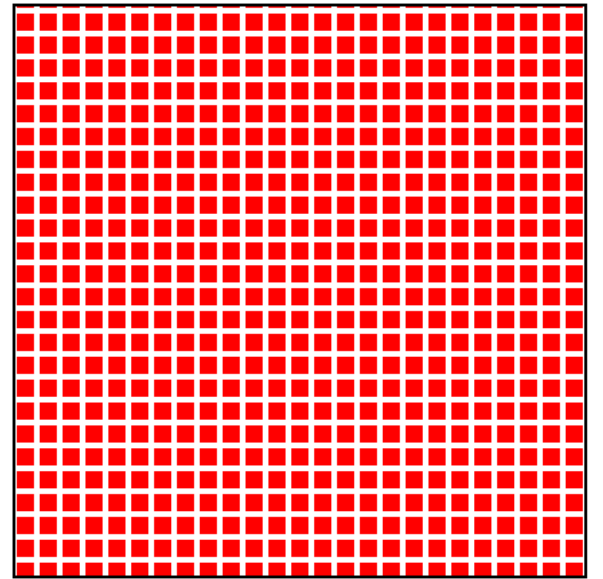
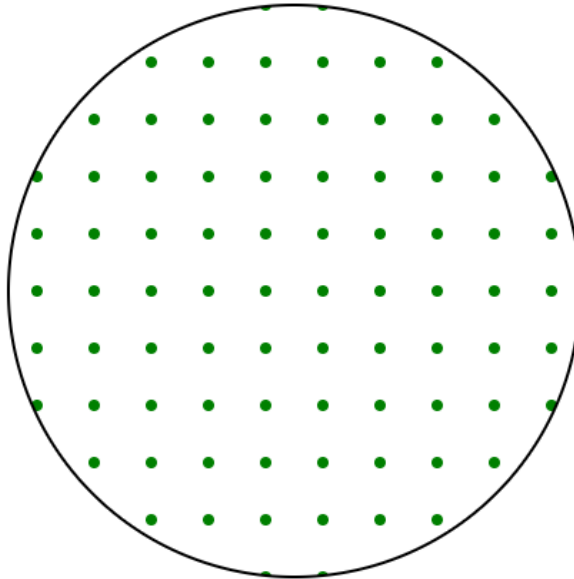
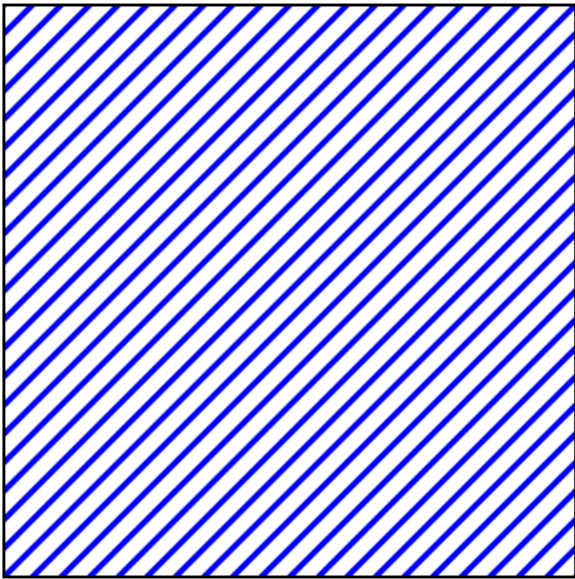
xMinYMin slice



xMaxYMax meet

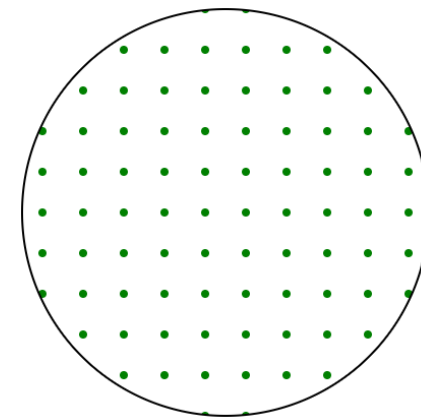
# SVG Patterns

- Code: <https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/html-tutorial/hello-texture.html>





# Pattern



- `<defs>`表示定义，即会被重复利用或者索引到的东西。
- `<patterns>`使用预定义的图形、图像等对一个图元进行填充，包括属性：
  - width, height
  - x, y
- width, height: 在**重复**下一个图案之前应该跨过多远
- x, y: Pattern的偏移
- **patternUnit: Pattern'使用'的坐标系（不做要求）**
- 预定义的图形、图像通过子节点的方式（如d3.append）给出

▼ `<defs>`

▼ `<pattern id="eofxu" patternUnits="userSpaceOnUse" width="20" height="20">`

`<circle cx="10" cy="10" r="2" fill="green" stroke="green" stroke-width="0"></circle>`

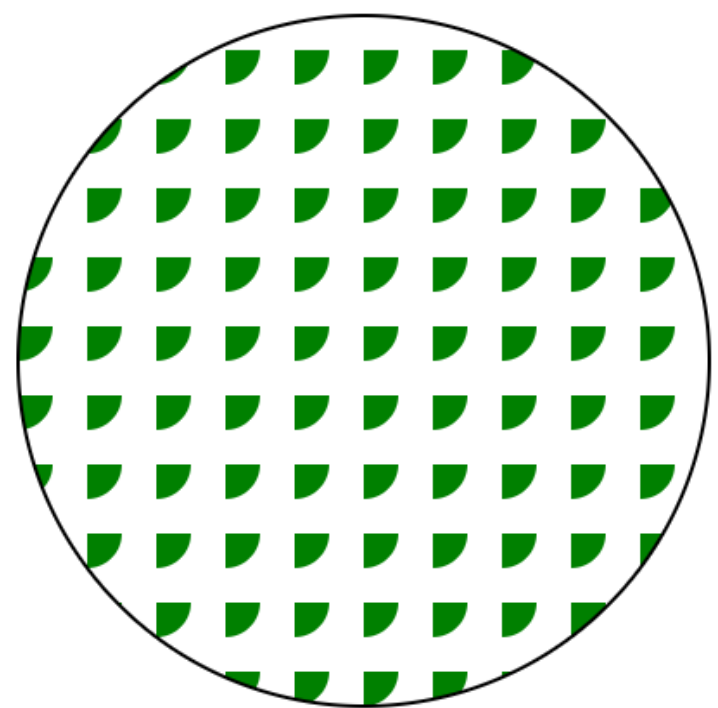
`</pattern>`

`</defs>`

`<circle class="mark" cx="400" cy="50" r="100" style="stroke: black; fill: url("#eofxu");"></circle>`

# SVG Patterns

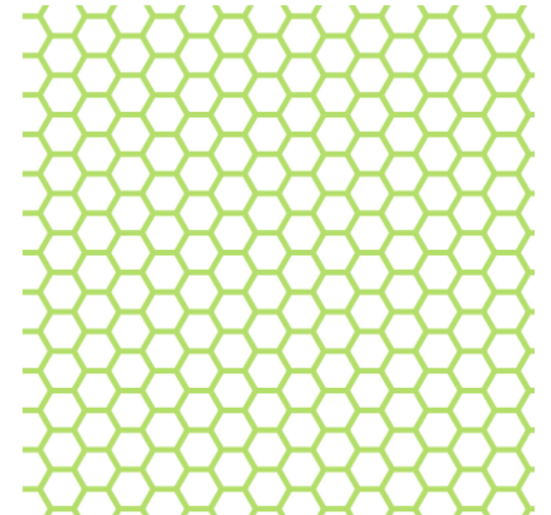
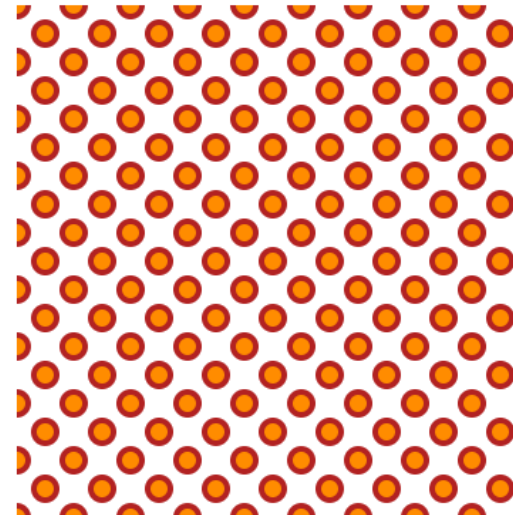
- 课后思考:
- 为什么按照如下设置会出现如图所示的情况?
- 可以把图片设置成图元的填充么? -> 图片纹理
  - Code也在hello-img.html中



```
<pattern id="exzwu" patternUnits="userSpaceOnUse" width="20" height="20">
<circle cx="0" cy="0" r="10" fill="green" stroke="green" stroke-width="0">
</circle>
</pattern>
```

# SVG Patterns – texture.js

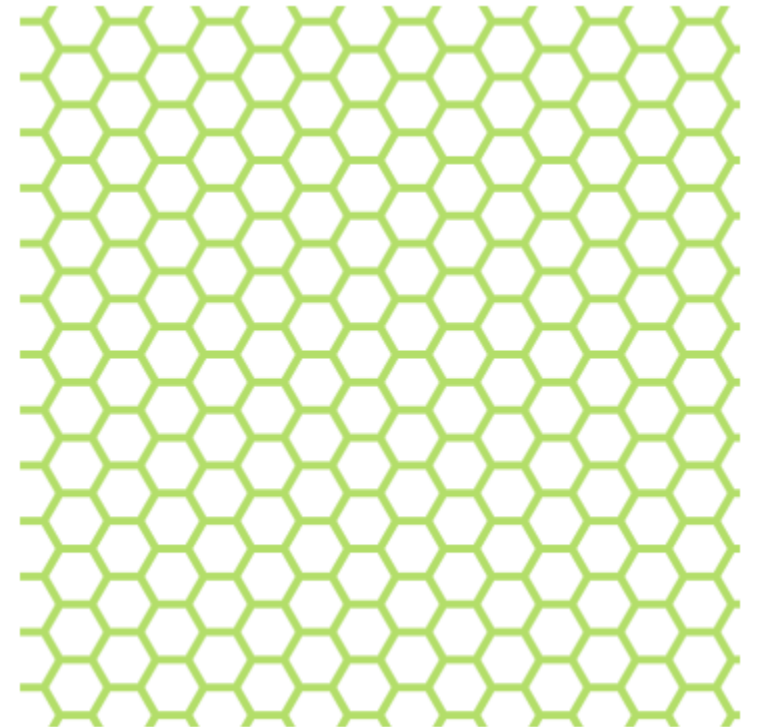
- 直接使用HTML定义纹理…emmm，回忆Path的'd'属性设置…
- texture.js可以帮助我们自定义SVGPattern! 😊
- Texture.js: <https://riccardoscalco.it/textures/>
- Code: <https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/d3-tutorial/hello-texture.html>



# SVG Patterns – texture.js

- 沿用了D3的语法
- 代码改编自官方示例: <https://riccardoscalco.it/textures/>

```
let t4 = textures.paths()  
.d("hexagons")  
.size(8)  
.strokeWidth(2)  
.stroke(d3.schemeSet3[6]);  
svg.call(t4);  
d3.select('#c4').attr('fill', t4.url());
```

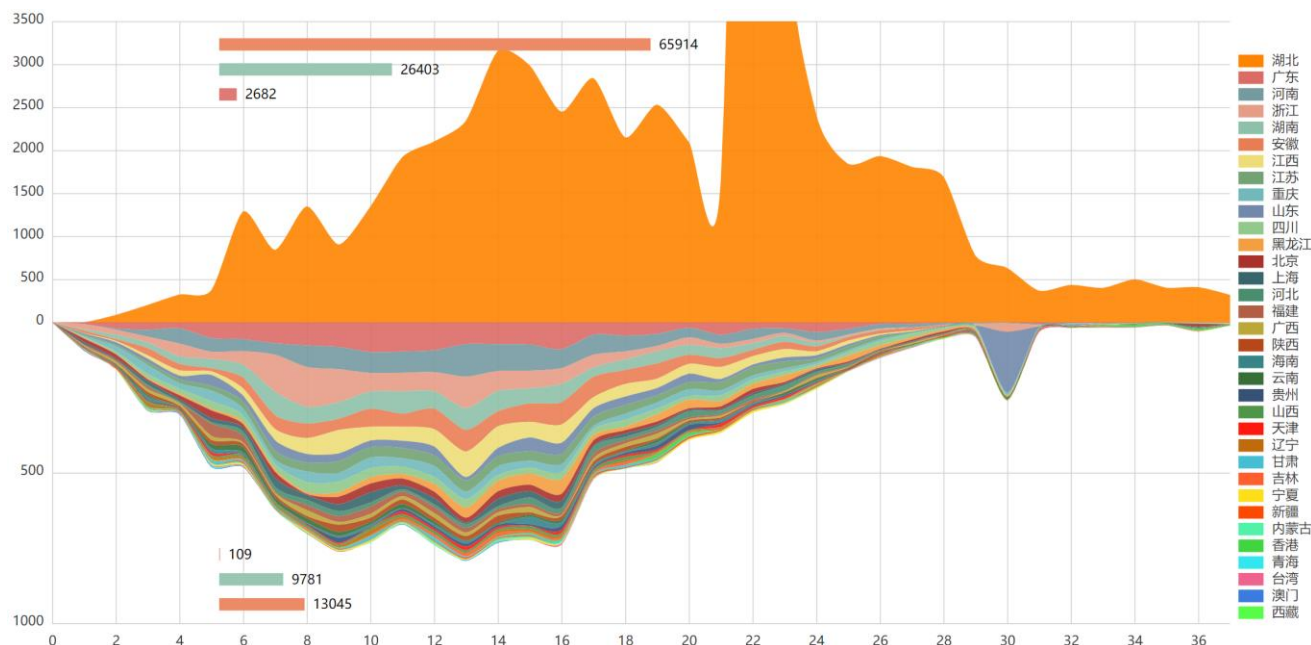


# Stack + Path (可以) = 主题河流 😊

- 主题河流
- 关键的接口：
  - d3.stack().offset(d3.stackOffsetWiggle)
  - d3.area(): 设置path的d属性
  - clipPath: 动效

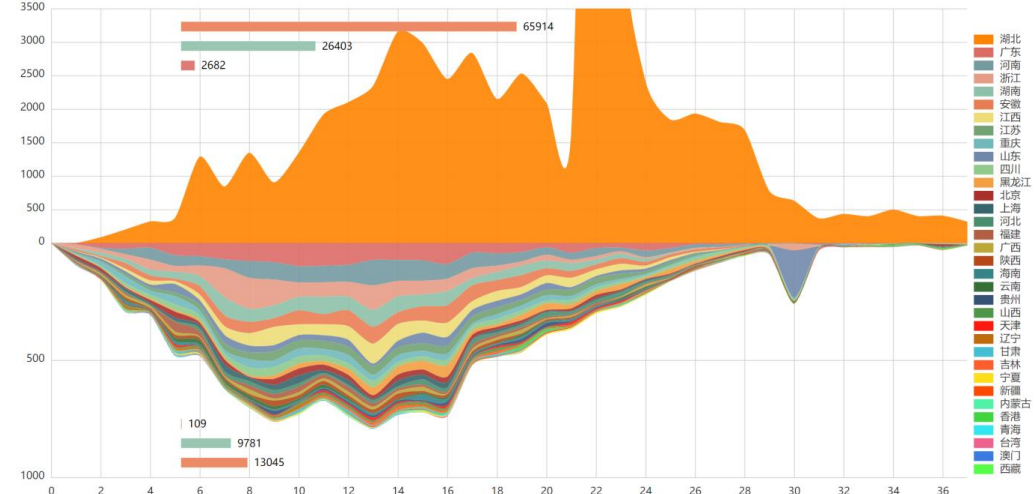
- Code: <https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/themeriver-lr.html>

```
const area = d3.area()  
.curve(d3.curveCardinal.tension(0.3))  
.x(d => xScale(xValue(d.data)))  
.y0(d => yScale(d[0]))  
.y1(d => yScale(d[1]));
```



# 同一坐标轴的不同尺度？

- 同一坐标轴的不同尺度？
- 同一比例尺的不同尺度反映在坐标轴上。

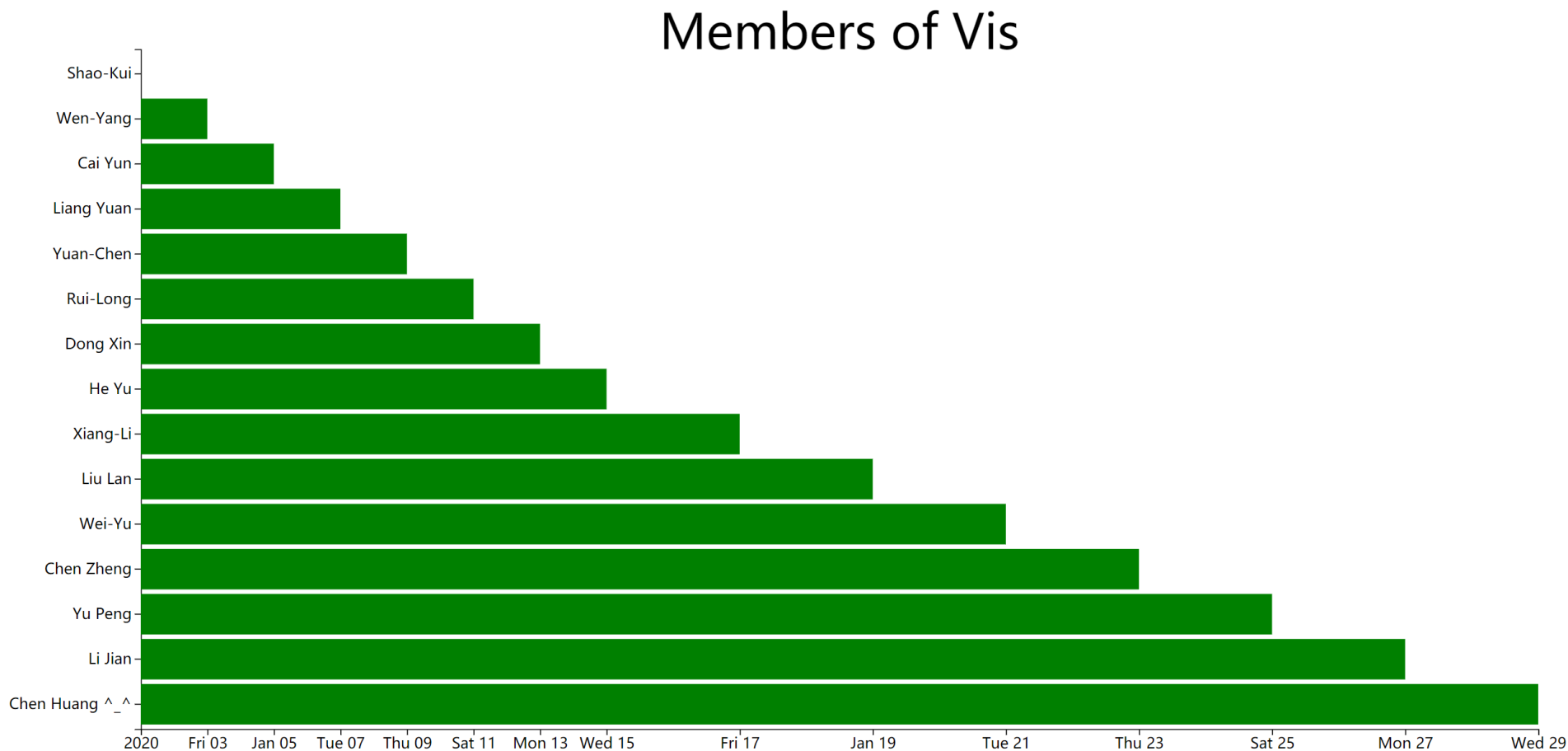


```
// Introducing y-Scale;  
yScale = d3.scaleLinear()  
  .domain([up_max, 0, -low_max])  
  .range([innerHeight, innerHeight / 2, 0])  
  .nice();
```



# 同一坐标轴的不同尺度? Cont.

- Code: <https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/d3-tutorial/timeScaleDiff.html>



# D3中的数字格式化与坐标轴的格式化

- `d3.format(specifier)`, `specifier`遵循下述规则
  - **`[[fill]align][sign][symbol][0][width][,][.precision][~][type]`**
- 典型的formater, ‘.’后的数字表示精度:
  - `d3.format('.2f')(666.666)` // 小数点后保留两位, 666.67
  - `d3.format('.2r')(2467)` // 只保留两位有效数字 2500
  - `d3.format('.3s')(2366.666)` // 只保留三位有效数字且加以后缀 2.37k
- `.tickFormat(d3.format(...))`: 根据formater来设置坐标轴上的数字格式
- 时间的格式化?
  - `d3.timeFormat( ... )` 如 `d3.timeFormat('%b-%d')`

# FileAttachment

- FileAttachment不是D3.js的接口，而是Observable提供的接口

## Using files

To use a file, first pass the file's name to the built-in FileAttachment function from the [Observable standard library](#).

```
attachment = ▼ FileAttachment {  
  name: "example.json"  
  <prototype>: ► FileAttachment {}  
}
```

```
attachment = FileAttachment("example.json")
```

# D3.js就没提供什么辅助交互的接口么? 😊

- Code: <https://github.com/Shao-Kui/D3.js-Demos/blob/master/static/graph.html>

- 拖拽:

```
function dragstarted(d) {  
    d3.select(this).raise().attr("stroke", "black");  
    simulation.stop();  
}  
function dragged(d) {  
    d3.select(this).attr("cx", d.x = d3.event.x).attr("cy", d.y = d3.event.y);  
    ticked();  
}  
function dragended(d) {  
    d3.select(this).attr("stroke", null);  
    simulation.restart();  
}  
const drag = d3.drag()  
    .on("start", dragstarted)  
    .on("drag", dragged)  
    .on("end", dragended);
```

# JS中的异步机制与其关于Transition的使用

- (本页与下一页内容不做要求)
- `async`: 将函数转换成异步函数, 即把返回值包装成一个Promise对象
- `await`: 强制等待异步函数执行结束, 比如:
- `let newdata = await d3.csv('static/data/2019worldpopulation.csv');`
- 注意: `await`关键字只能在异步函数中使用!
- 使用异步机制等待transition的结束
  - Why? -> 多个transition对于同一个图元的转换不会自动同步;
  - `transition.end()`可以返回一个Promise对象;
  - 使用`await`关键字可以对这个Promise进行等待。

# JS中的异步机制与其关于Transition的使用

```
let f = async () => {  
  //let counter = 0;  
  //while(counter < 5){  
  while(true){  
    //counter++;  
    await rects.transition().duration(1000)  
      .attr('width', () => xScale(Math.random() * 20))  
      .attr('fill', () => d3.interpolateRainbow(Math.random()))  
      .end();  
  }  
};  
f();
```



# transition是可以共享的…

- 使用D3定义Transition对象，并让其与多个图元共享
- `await transition.end()`则会同时等待所有相关图元


```
let transition = d3.transition().ease(d3.easeLinear)
  .duration(1000);
rects.transition(transition)
  .attr('width', d => xScale(d.value))
  .attr('fill', () => d3.interpolateRainbow(Math.random()));
texts.transition(transition)
  .attr('x', d => xScale(d.value))
await transition.end();
```

# transition.tween


- transition.attr v.s. selection.attr
  - 思考: **selection.transition().duration(3000).attr('fill', 'green');**
- selection.attr - 设置/获取图元的属性
- transition.attr - 使用默认的插值器在两个给定的属性之间进行过渡 (tween)
  - 例如从红色渐变到蓝色、从左边移动到右边……
  - tween: 介词, 之间、补间的含义
- transition.tween - 使用自定义的插值器进行过渡
- ‘自定义的插值器’: 返回函数的函数
  - 插值器本身的参数: 类似于.attr中的数据‘d’与索引‘i’
  - 返回函数的参数: 从0到1的、根据时间变化的数字t

# transition.tween


- transition.tween – 使用**自定义的插值器**进行过度
- ‘**自定义的插值器**’: **返回函数**的**函数**
  - **插值器**本身的参数: 类似于.attr中的数据‘d’与索引‘i’
  - **返回函数**的参数: 从0到1的、根据时间变化的数字t
  - ‘t’很自然表示某一时刻, 图元会在这一时刻关于某一属性有一个非常‘短暂’的属性值。
- **自定义的插值器**的作用? ...



t = 0.4



t = 0.6



t = 1

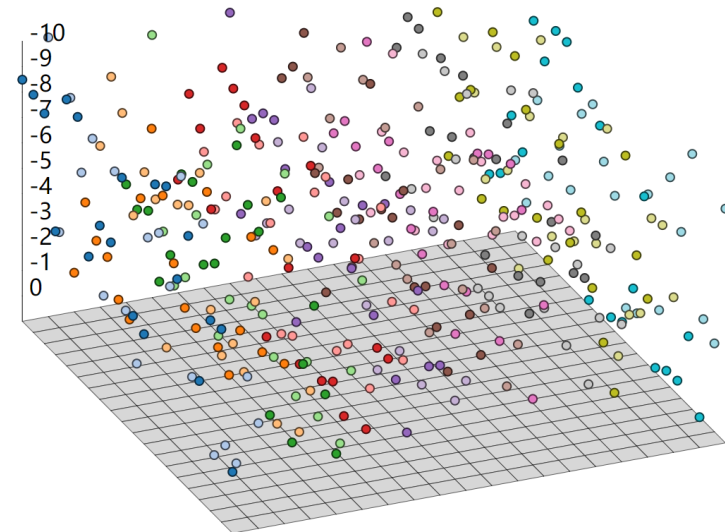
# transition.tween

- 自定义的插值器的作用? ...
- 返回的函数中, 参数只有表示某一时刻的't'...
  - 要如何根据某一时刻来设置图元属性?
  - 要如何确定图元属性渐变过程中的初始与最终状态?
- 插值器本身的参数: 类似于.attr中的数据'd'与索引'i'

```
texts.transition(transition)
  .tween("text", function(d) {
    var i = d3.interpolate(this.textContent, d.value);
    return function(t) {
      this.textContent = formatPercent(i(t));
    };
  });
```

# D3.js的局限性...

- D3尽管作为目前Web端最强大的可视化框架之一...
- 三维数据的可视化?
- 确实存在第三方的库可以解决...
  - <https://github.com/nieked3-3d>
  - <https://blocks.org/Nieked3-3d/1c15016ae5b5f11508f92852057136b5>
- 游戏引擎(unity, unreal...) or three.js
  - 使用three.js可视化三维场景并自由漫游
  - <https://github.com/Shao-Kui/SUNCG-Visualization-Using-Three.js>



# Thank You!

张松海、张少魁、周文洋、蔡韵

数据可视化 - D3.js

清华大学 可视媒体研究中心