

D3.js

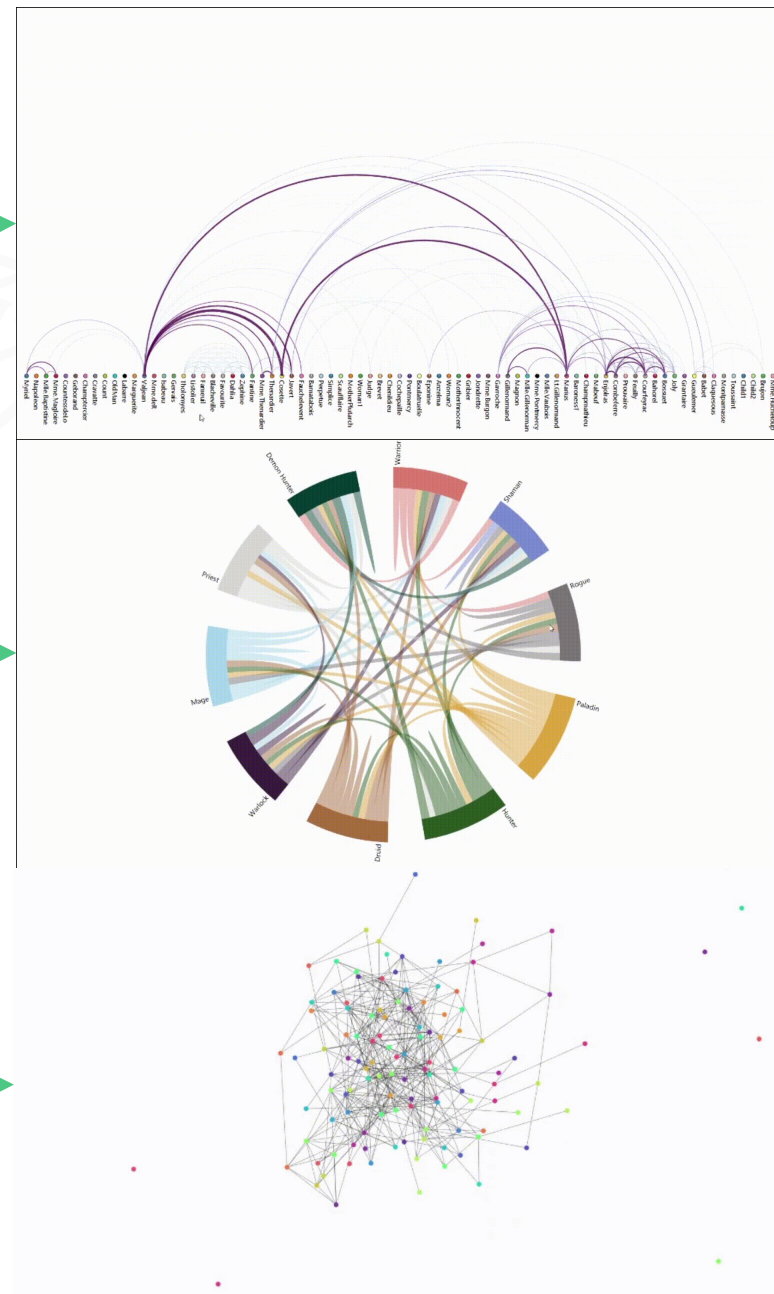
网络数据可视化

张松海 张少魁

清华大学 可视媒体研究中心

概览

- 网络数据的数据结构?
- 基于<path>实现弧长连接图:
 - <path>的弧线命令。
- 基于‘d3.chord’与‘d3.ribbon’实现弦图:
 - 使用d3.chord预‘分配’圆周。
 - 使用d3.ribbon生成弦的‘d’属性。
- D3.js力模拟基础:
 - d3-force的总体配置。
 - 不同力的作用与‘Tic-Toc’。
- 基于‘d3-force’实现力导图。



网络数据的数据结构?

【节点数、边数
与基于ID的连接】

```
{
  "#nodesorigin": 769,
  "#edgesorigin": 16656,
  "links": [
    {
      "source": 28,
      "target": 0
    },
    {
      "source": 31,
      "target": 0
    },
    {
      "source": 46,
      "target": 0
    }
  ],
}
```

- 网络数据包括**节点**的集合与**边**的集合:
 - **节点与边**通常分布在不同的文件中, 通过节点的ID索引。
- D3.js也没有统一的网络数据结构规范:
 - 只要能整理成D3.js对应接口接受的格式即可。

【节点列表】 + 【边列表】

- 常见的数据形式:
 - 【节点列表】 + 【连接矩阵】。
 - 【节点数、边数与基于ID的连接】。
 - 【节点列表】 + 【边列表】。
 -

```
source_id,target_id,value
Napoleon,Myriel,1
Mlle.Baptistine,Myriel,8
Mme.Magloire,Myriel,10
Mme.Magloire,Mlle.Baptistine,6
CountessdeLo,Myriel,1
Geborand,Myriel,1
Champtercier,Myriel,1
Cravatte,Myriel,1
Count,Myriel,2
OldMan,Myriel,1
```

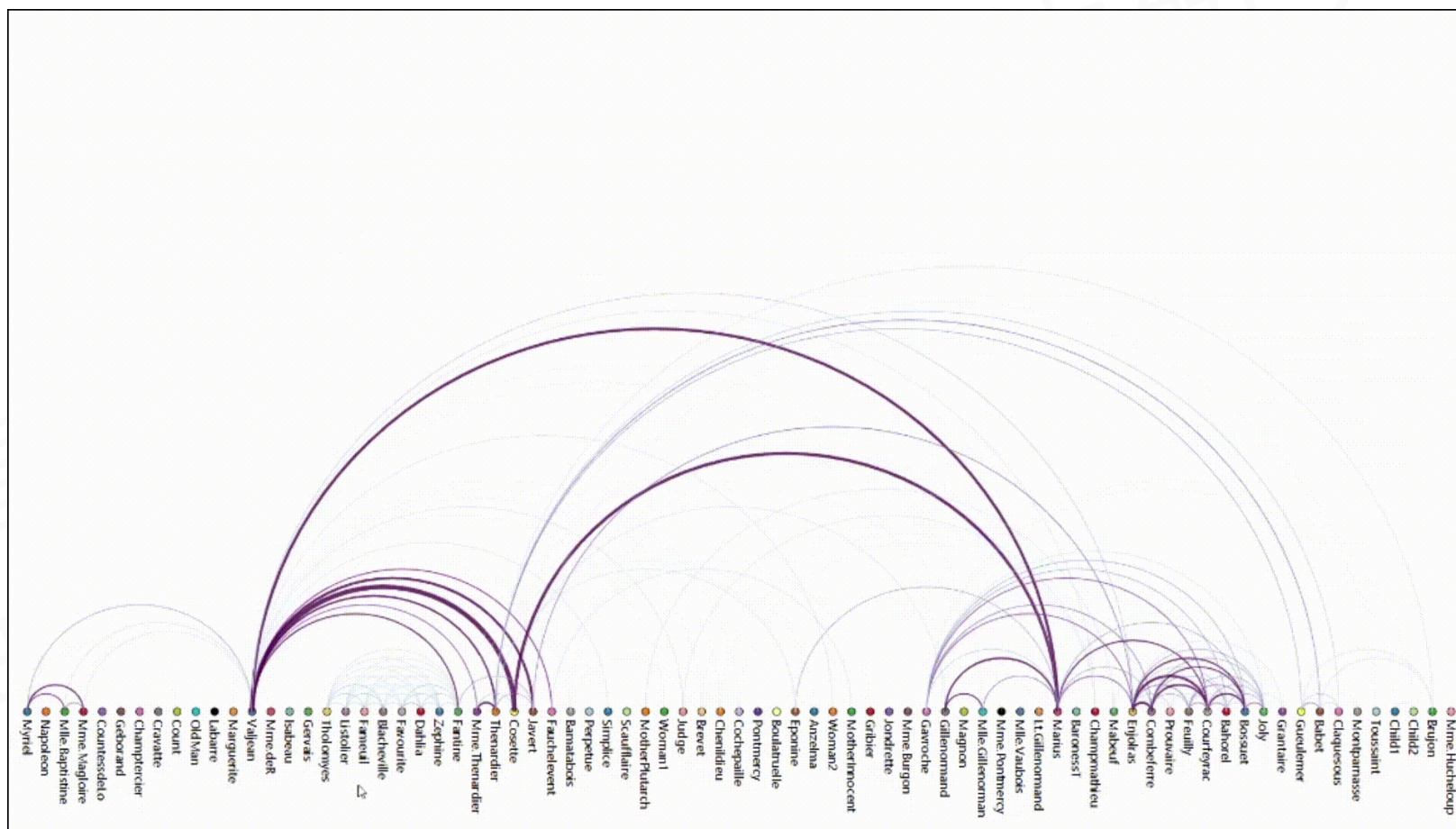
```
id,name,group
Myriel,Myriel,G1
Napoleon,Napoleon,G1
Mlle.Baptistine,Mlle.Baptistine,G1
Mme.Magloire,Mme.Magloire,G1
CountessdeLo,CountessdeLo,G1
Geborand,Geborand,G1
Champtercier,Champtercier,G1
Cravatte,Cravatte,G1
Count,Count,G1
OldMan,OldMan,G1
```

```
%%MatrixMarket matrix coordinate integer symmetric
77 77 254
2 1 1
3 1 8
4 1 10
```

【连接矩阵】

基于<path>实现弧长连接图

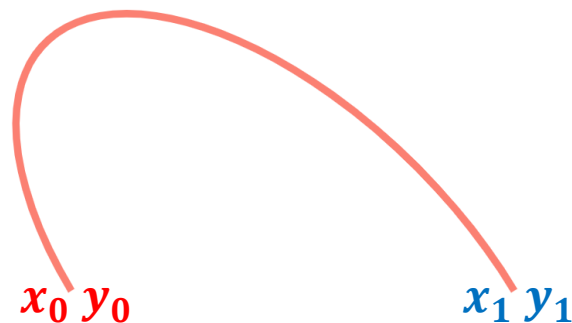
- 数据来源: <https://visdatasets.github.io/>。



《悲惨世界》各人物关系可视化

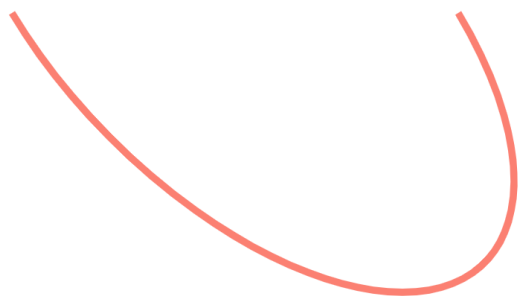
<path>的弧线命令

- SVG的<path>图元基于‘d’属性（笔顺）可绘制任意弧线：
 - 弧线是圆或椭圆。
- M x_0 y_0 A x_r y_r θ ? ? x_1 y_1 :
 - M x_0 y_0 : ‘d’属性的落笔，即**起点**。
 - A: 弧线命令，后续内容为弧线命令的各个参数，缺一不可。
 - x_r y_r θ : 弧线所在椭圆的横轴 & 纵轴 & 旋转。
 - x_1 y_1 : 弧线的**终点**。
 - ?: （不做要求）弧线命令的走向，均取0/1，暂可都填入0。
- e.g., M 1206 940 A 1 0.5 45 0 0 633 940



<path>的弧线命令

- $x_r y_r \theta$:
 - 弧线所在椭圆的横轴 & 纵轴 & 旋转。
 - $(x_0 y_0)$ 与 $(x_1 y_1)$ 距离可‘卡住’椭圆时，会在椭圆上寻找一段弧作为结果。
 - $(x_0 y_0)$ 与 $(x_1 y_1)$ 距离过大时，椭圆会先被放大。



M 100 940 A **1 0.5 45** 0 0 700 940



M 100 940 A **2000 1500 0** 0 0 700 940



M 100 940 A **500 800 0** 0 0 700 940

基于<path>实现弧长连接图

- 对于网络数据可视化，通常需要建立**节点与边**的索引：

- 通常边数据并不记录两个连接节点的属性、位置等。
- 边可以通过索引找到节点的位置，进而连接两点。

- 编程实例：

```
for(let link of links){  
  link.source = nodes.find(node => node.id === link.source_id);  
  link.target = nodes.find(node => node.id === link.target_id);  
}
```

.source_id是字符串；

.source是JavaScript对象。

- 基于模板字符串与<path>弧线指令，构造弧线：

- 编程实例：

```
const arc = d => {  
  return `M ${cx(d.source)} ${cy(d.source)} A ${arcRadii} ${arcAngle} 0 0 ${cx(d.target)} ${cy(d.target)}`;  
}
```

```
const cx = d => 2 * RADIUS + d.ord * width / NODENUM;  
const cy = d => height-100;
```

基于<path>实现弧长连接图

- 节点与边的Data-Join, 编程实例:

```
svg.selectAll('.myCircle').data(nodes).join('circle').attr('class', 'myCircle')
.attr('cy', cy).attr('cx', cx).attr('r', RADIUS)
.attr('stroke', 'black').attr('fill', d => colorNode(d.id));

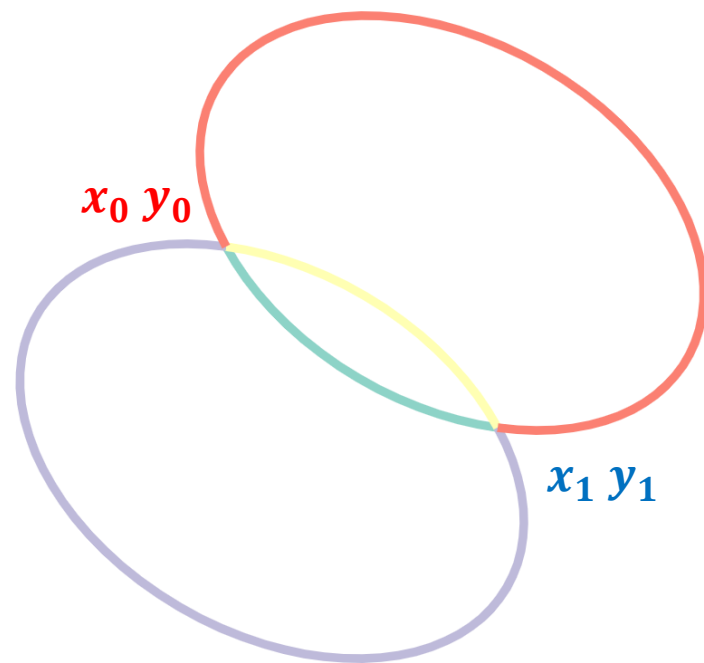
svg.selectAll('.myPath').data(links).join('path').attr('class', 'myPath')
.attr('d', arc).attr('fill', 'none')
.attr('stroke-width', d => d.value * 0.2).attr('stroke', d => colorLink(d.value));

svg.selectAll('.myText').data(nodes).join('text').attr('class', 'text')
.attr('y', cy).attr('x', cx).attr('dy', 10)
.attr('text-anchor', 'start').attr('writing-mode', 'vertical-rl')
.text(d => d.name);
```


Tip: LargeArc-Flag & Sweep-Flag

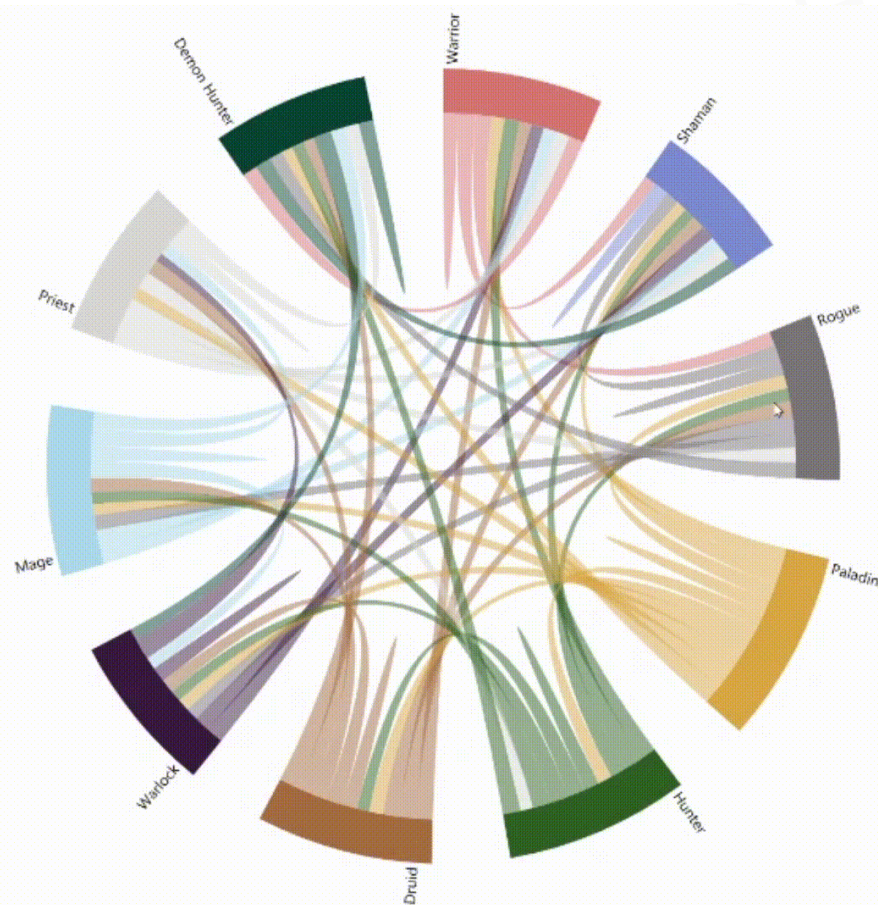
- (不做要求) 两点间的不同弧线, 在椭圆足够大时, 有四个。
- M $x_0 y_0$ A $x_r y_r \theta$ **LargeArc-Flag** **Sweep-Flag** $x_1 y_1$:
 - LargeArc-Flag: 走较大的弧线还是较小的弧线?
 - Sweep-Flag: 逆时针还是顺时针?
- 编程实例:

```
const path1 = svg.append('path')
  .attr('d', `M 100 100 A ${radii} ${angle} 0 0 ${endPoint}`)
  .attr('stroke', d3.schemeSet3[0])
const path2 = svg.append('path')
  .attr('d', `M 100 100 A ${radii} ${angle} 0 1 ${endPoint}`)
  .attr('stroke', d3.schemeSet3[1])
const path3 = svg.append('path')
  .attr('d', `M 100 100 A ${radii} ${angle} 1 0 ${endPoint}`)
  .attr('stroke', d3.schemeSet3[2])
const path4 = svg.append('path')
  .attr('d', `M 100 100 A ${radii} ${angle} 1 1 ${endPoint}`)
  .attr('stroke', d3.schemeSet3[3])
```



基于‘d3.chord’与‘d3.ribbon’实现弦图

- 数据来源: <https://lushi.163.com/bigdata/ladder2/20210524-20210530/>。



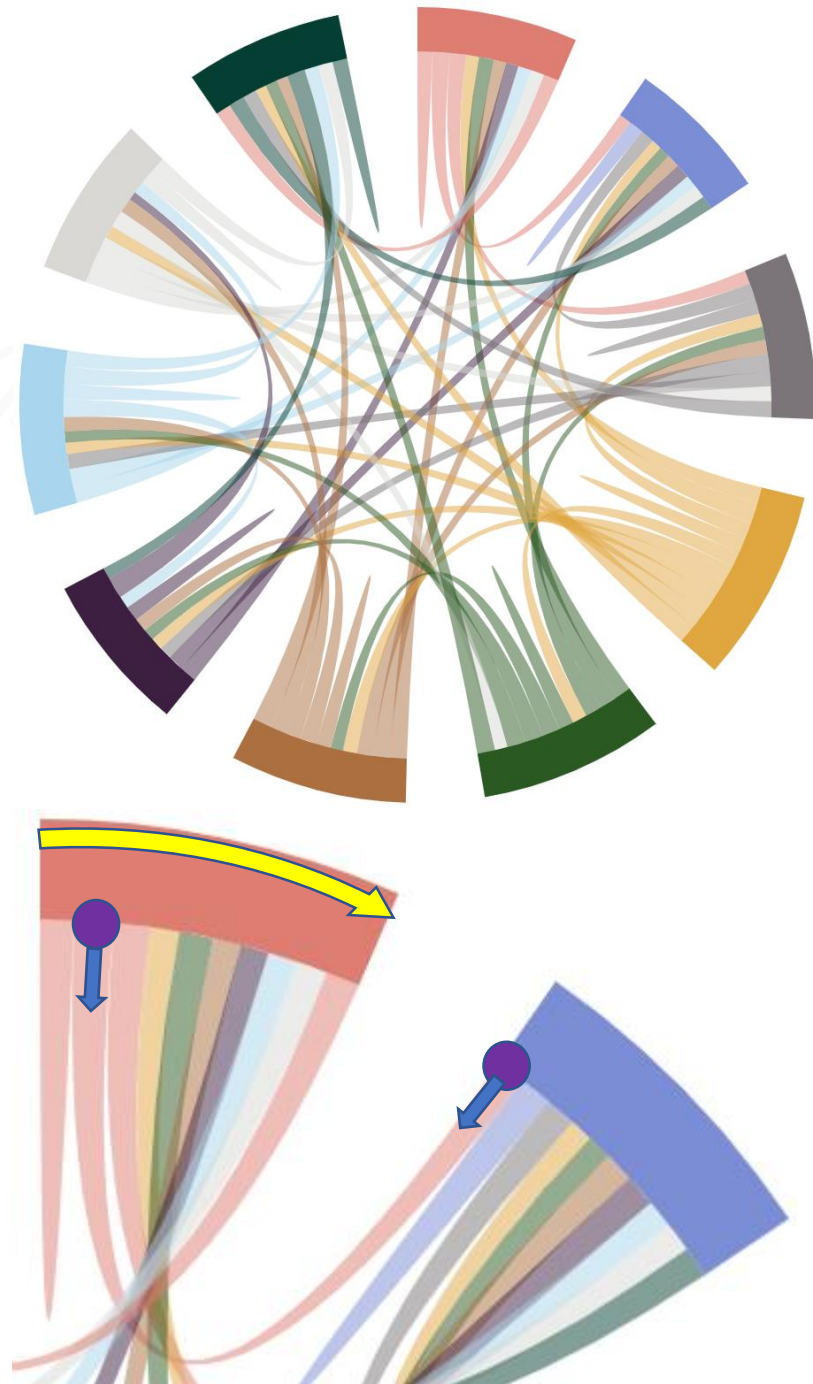
《炉石传说》各职业对阵胜率可视化

d3.chord

- let `chord = d3.chord().padAngle(angle);`
 - 定义一个函数，将节点与边划分到弦图的各个角度。
 - 节点会基于出度的总和 Σ 占有圆周的度数。
 - 边的两端分别是【节点A的所占角度】与【节点B的所占角度】。
 - .padAngle用于设置节点之间的间隙。
 - e.g., let chord = d3.chord().padAngle(0.2);
- 函数chord接受的输入为矩阵：
- 行、列分别表示‘第几个节点’。
- 如：

```
var matrix = [  
  [11975, 5871, 8916, 2868],  
  [ 1951, 10048, 2060, 6171],  
  [ 8010, 16145, 8090, 8045],  
  [ 1013, 990, 940, 6907]  
];
```

有向图的边权值



d3.chord

```
d3.json('hs_matrix.json').then(matrix => {  
  // adding ribbons between nodes.  
  let chord = d3.chord().padAngle(0.2);  
  let ribbons = chord(matrix);
```

- 编程实例与转换前后对比：
- 返回的结果为数组，每个元素代表一条边：
 - 每个元素包含'source'与'target'，即起点与终点。
 - source.startAngle: 起点的起始角度，source.endAngle: 起点的终止角度。
 - 需要进一步使用d3.ribbon来绘制<path>。

▼ Array(55) ⓘ

▶ 0: {source: {...}, target: {...}}

▶ 1: {source: {...}, target: {...}}

▶ 2: {source: {...}, target: {...}}

▼ 3:

▶ source: {index: 3, startAngle: 1.8061107170193138, endAngle: 1.8549218967799324, value: 56.98}

▶ target: {index: 0, startAngle: 0.13827835445698577, endAngle: 0.17513088083995892, value: 43.02}

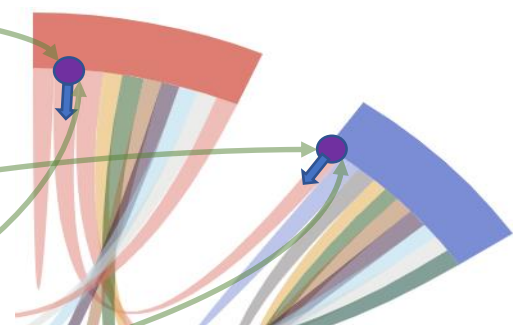
▶ __proto__: Object

▶ 4: {source: {...}, target: {...}}

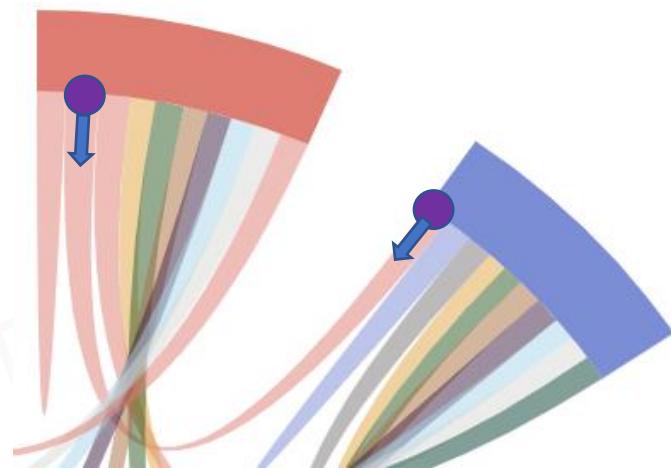
▶ 5: {source: {...}, target: {...}}

▼ 6:

▶ source: {index: 6, startAngle: 3.822525832621146, endAngle: 3.8725705697502324, value: 58.42}



d3.ribbon



- `d3.ribbon().radius(...)`
 - 返回一个函数，输入为两个节点，输出为<path>的'd'属性。
 - e.g., let `drawRibbon` = `d3.ribbon()`;
 - 输入数据，类似`d3.linkVertical`，必须包含'source'属性作为起点，'target'属性作为终点。
 - e.g., const `data1` = {
 'source': {startAngle: 0.1524114, endAngle: 0.1912972},
 'target': {startAngle: 1.2617078, endAngle: 1.5842927} };
 - 起点与终点均需包含各自的起始角度与终止角度。
 - 通过`.radius(...)`来指定Ribbon所在圆的半径。
 - e.g., let `drawRibbon` = `d3.ribbon().radius(400)`;
 - e.g., `drawRibbon(data1); //`

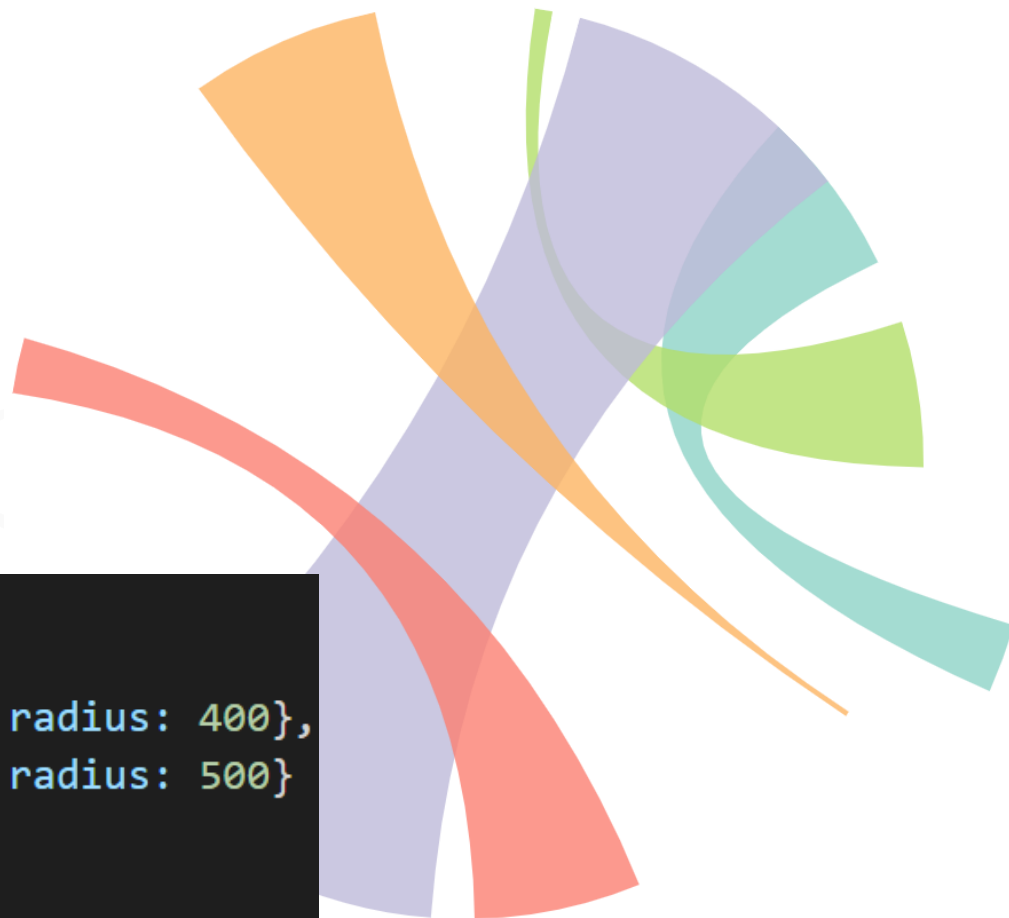
```
d: path("M 60.7288  
-395.363 A 400 400 0  
0 1 76.053 -392.703 Q  
0 0 381.044 -121.676  
A 400 400 0 0 1  
399.964 5.39839 Q 0 0  
60.7288 -395.363 Z");
```

d3.ribbon

- 编程实例:

```
const path1 = svg.append('path')
.attr('d', ribbon({
  source: {startAngle: 0.7524114, endAngle: 1.1212972, radius: 400},
  target: {startAngle: 1.8617078, endAngle: 1.9842927, radius: 500}
}))
.attr('fill', d3.schemeSet3[0])

const path2 = svg.append('path')
.attr('d', ribbon({
  source: {startAngle: 0.1524114, endAngle: 0.1912972, radius: 400},
  target: {startAngle: 1.2617078, endAngle: 1.5842927, radius: 400}
}))
.attr('fill', d3.schemeSet3[6])
```



d3.ribbon

- d3.chord返回的数组中，每个元素就是d3.ribbon接受的‘source’与‘target’格式。

```
▼ 3:  
  ▶ source: {index: 3, startAngle: 1.8061107170193138, endAngle: 1.8549218967799324, value: 56.98}  
  ▶ target: {index: 0, startAngle: 0.13827835445698577, endAngle: 0.17513088083995892, value: 43.02}  
  ▶ __proto__: Object  
▶ 4: {source: {...}, target: {...}}
```

- 编程实例：

```
let chord = d3.chord().padAngle(0.2);  
let ribbons = chord(matrix);  
let drawRibbon = d3.ribbon().radius(RADIUS);  
  
g.selectAll('.myRibbon').data(ribbons).join('path').attr('class', 'myRibbon')  
  .attr('d', drawRibbon).attr('fill', hero)  
  .attr('opacity', 0.5);
```

d3.chord

- 返回结果的‘groups’属性为数组，每个元素代表一个节点：
 - startAngle: 节点的起始角度，endAngle: 节点的终止角度；
 - index: 第几个节点，‘value’: 节点的出度总和。
 - 直接对应d3.arc的接口。

► 53: {source: {...}, target: {...}}

► 54: {source: {...}, target: {...}}

▼ groups: Array(10)

► 0: {index: 0, startAngle: 0, endAngle: 0.40925835610100947, value: 477.75000000000006}

► 1: {index: 1, startAngle: 0.6092583561010094, endAngle: 0.9807390177926949, value: 433.65}

► 2: {index: 2, startAngle: 1.1807390177926949, endAngle: 1.6061107170193138, value: 496.56}

► 3: {index: 3, startAngle: 1.8061107170193138, endAngle: 2.2997992218954475, value: 576.31}

► 4: {index: 4, startAngle: 2.4997992218954477, endAngle: 2.971309393251005, value: 550.42000000000001}

► 5: {index: 5, startAngle: 3.171309393251005, endAngle: 3.622525832621146, value: 526.7299999999999}

► 6: {index: 6, startAngle: 3.822525832621146, endAngle: 4.231981215246286, value: 477.97999999999996}

► 7: {index: 7, startAngle: 4.431981215246286, endAngle: 4.86750406365092, value: 508.41}

► 8: {index: 8, startAngle: 5.0675040636509205, endAngle: 5.480600153787163, value: 482.22999999999996}

► 9: {index: 9, startAngle: 5.6806001537871635, endAngle: 6.083185307179587, value: 469.96}

基于‘d3.chord’与‘d3.ribbon’实现弦图

- 编程实例:

```
g.selectAll('.myRibbon').data(ribbons).join('path').attr('class', 'myRibbon')
.attr('d', drawRibbon).attr('fill', hero)
.attr('opacity', 0.5);

// adding arcs representing nodes.
let arc = d3.arc().innerRadius(RADIUS).outerRadius(RADIUS+50);
g.selectAll('.myArc').data(ribbons.groups).join('path').attr('class', 'myArc')
.attr('d', arc).attr('fill', d => nodes[d.index].color);

// adding texts representing nodes.
g.selectAll('.myText').data(ribbons.groups).join('text').attr('class', 'myText')
.attr('transform', d => `
rotate(${d.startAngle * 180 / Math.PI - 90 + 2})
translate(${RADIUS + 50}, 0)
rotate(${d.startAngle * 180 / Math.PI < 180 ? 0 : 180})`)
.attr('text-anchor', d => d.startAngle * 180 / Math.PI < 180 ? 'start' : 'end')
.attr('dx', d => d.startAngle * 180 / Math.PI < 180 ? 5 : -5)
.text(d => nodes[d.index].name);
```

Tip: 数据处理接口与<path>绘制接口

- D3.js的使用需要协调好各个接口的输入、输出。
- D3.js提供一批接口用于<path>的‘d’属性生成。
- D3.js提供一批接口用于数据的处理、转换与扩充。
- 明确、不混淆各个接口的输入与输出有助于阅读、修改、创作D3.js的代码。
- 完整版表格见下一页。

接口名称	接口类型	常见用途
d3.line	<path>‘d’属性生成	折线图
d3.arc	<path>‘d’属性生成	饼图、玫瑰图、日晕图、弦图等
d3.pie	数据处理、转换与扩充	饼图
d3-projection	地理坐标映射	地图与地形等
d3.geoPath	<path>‘d’属性生成	地图与地形等
d3.contours	数据处理、转换与扩充	地形
d3.stack	数据处理、转换与扩充	堆叠柱状图与主题河流等
d3.area	<path>‘d’属性生成	主题河流等
d3.hierarchy	数据处理、转换与扩充	树状图、冰锥图、日晕图等
d3.tree	数据处理、转换与扩充	树状图与径向树状图等
d3.partition	数据处理、转换与扩充	冰锥图与日晕图等
d3.linkVertical & d3.linkHorizontal	<path>‘d’属性生成	树状图等
d3.chord	数据处理、转换与扩充	弦图等
d3.ribbon	<path>‘d’属性生成	弦图等

本门课程用到的重点接口一览，按照课程介绍顺序。

接口名称	接口类型	常见用途
d3.line	<path>‘d’属性生成	折线图等
d3.arc	<path>‘d’属性生成	饼图、玫瑰图、日晕图、弦图等
d3.pie	数据处理、转换与扩充	饼图等
d3-projection	地理坐标映射	地图与地形等
d3.geoPath	<path>‘d’属性生成	地图与地形等
d3.contours	数据处理、转换与扩充	地形
d3.stack	数据处理、转换与扩充	堆叠柱状图与主题河流等
d3.area	<path>‘d’属性生成	主题河流等
d3.hierarchy	数据处理、转换与扩充	树状图、冰锥图、日晕图等
d3.tree	数据处理、转换与扩充	树状图与径向树状图等
d3.partition	数据处理、转换与扩充	冰锥图与日晕图等
d3.linkVertical & d3.linkHorizontal	<path>‘d’属性生成	树状图、径向树状图、聚类树状图等
d3.chord	数据处理、转换与扩充	弦图等
d3.ribbon	<path>‘d’属性生成	弦图等

D3力模拟基础

- D3的力模拟与‘Transition’是完全不同的两个体系。
- `let nodes = [{}, {}, {}, {}, {}, {}];`
- `let simulation = d3.forceSimulation(nodes)` 定义后会发生…
 - **补全**nodes中每个节点的数据结构：
 - 包括index, x, y, vx, vy, 后两者为速度。
 - 开始**模拟**粒子运动：
 - 粒子质量为1。
 - 不断地通过内部timer触发‘tick’事件。
 - 根据一系列的‘力’来计算每个例子的加速度、速度、位置…
 - ‘力’都是哪来的呢？

```
▶ 0: {index: 0, x: 738.3806527975547, y: 343.0306620165279, vy: -0.029376627447530432, vx: -0.1457788915268187}
▶ 1: {index: 1, x: 825.4272261336539, y: 646.6813124206806, vy: -0.06945551447339872, vx: 0.021719091096546093}
▶ 2: {index: 2, x: 690.0380469592366, y: 677.8974649507733, vy: -0.06087482519618888, vx: -0.04212888347764266}
▶ 3: {index: 3, x: 420.41522887639513, y: 602.1439659029284, vy: -0.027755733036445917, vx: -0.12314103808246948}
▶ 4: {index: 4, x: 798.0128857957698, y: 315.80315062641995, vy: -0.009943149939567702, vx: -0.14329057737755713}
▶ 5: {index: 5, x: 2862.020067133912, y: -48.60341039601587, vy: -0.053645355558169884, vx: 0.22791145689693854}
▶ 6: {index: 6, x: 629.5063624938902, y: 752.78717521969, vy: -0.024194706402747724, vx: -0.0770175502627483}
```

不同力的作用

- `d3.forceManyBody().strength(value)`:
 - 粒子之间两两的作用力，类似于‘万有引力’。
 - ‘`.strength(value)`’用来设置力的大小，`value`为正互相吸引，为负则互相排斥。
- `d3.forceCenter(w, h).strength(value)`:
 - 指向某一个中心的力，会尽可能让粒子向中心靠近。
 - `.strength(...)`的用法同上。
 - ‘`d3.forceCenter(w, h)`’中的‘`w`’与‘`h`’为中心的位置，通常为画布的中心。
- `d3.forceLink(links).strength(strength).distance(distance)`:
 - 部分粒子之间的两两作用力，不同于‘`d3.forceManyBody`’。
 - ‘`d3.forceLink`’中，每个节点仅仅会与一部分节点有力的作用。
 - 有链接的节点间，受力的作用，**保持在特定的距离，即靠近互斥、远离吸引。**
 - 是否有链接需要通过图的边集合给出。
 - ‘**`.strength(vs)`**’与 ‘**`.distance(vd)`**’分别设置**力的大小**与**预期的距离**。

不同力的作用

- Link要通过一个数据格式给出，即link的source与target。
- 格式非常类似于‘d3.hierarchy’给出的root.links()。

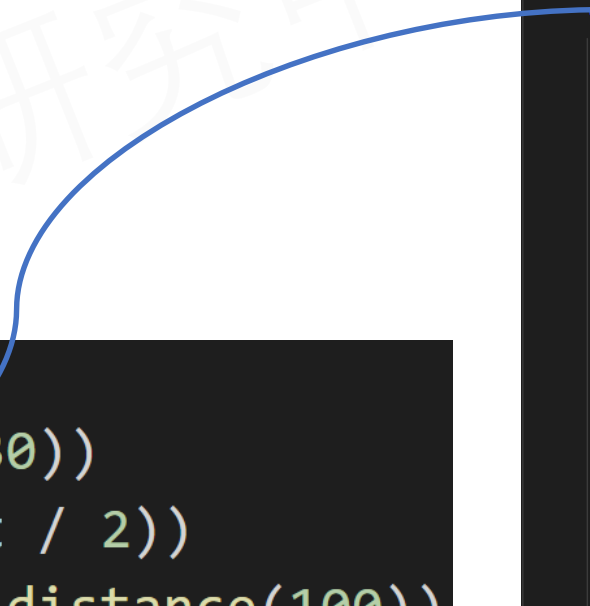
```
▼ [0 ... 99]
  ▶ 0: {source: {...}, target: {...}, index: 0}
  ▶ 1: {source: {...}, target: {...}, index: 1}
  ▶ 2: {source: {...}, target: {...}, index: 2}
  ▼ 3:
    ▶ source: {index: 55, x: 1013.1089966584524, y: 398.58031002386207, vy: -0.00047989706956436275, vx: -0.0019270898010742773}
    ▶ target: {index: 0, x: 896.3549810575581, y: 374.663789434326, vy: -0.001639673985733313, vx: -0.0018180969751919926}
    index: 3
    ▶ __proto__: Object
  ▶ 4: {source: {...}, target: {...}, index: 4}
  ▶ 5: {source: {...}, target: {...}, index: 5}
  ▶ 6: {source: {...}, target: {...}, index: 6}
  ▶ 7: {source: {...}, target: {...}, index: 7}
  ▶ 8: {source: {...}, target: {...}, index: 8}
  ▶ 9: {source: {...}, target: {...}, index: 9}
  ▶ 10: {source: {...}, target: {...}, index: 10}
  ▶ 11: {source: {...}, target: {...}, index: 11}
  ▶ 12: {source: {...}, target: {...}, index: 12}
  ▶ 13: {source: {...}, target: {...}, index: 13}
  ▶ 14: {source: {...}, target: {...}, index: 14}
```

不同力的作用

- 编程实例:

```
simulation = d3.forceSimulation(nodes)
  .force('manyBody', d3.forceManyBody().strength(-30))
  .force('center', d3.forceCenter(width / 2, height / 2))
  .force("link", d3.forceLink(links).strength(0.1).distance(100))
```

```
{
  "#nodesorigin": 769,
  "#edgesorigin": 16656,
  "links": [
    {
      "source": 28,
      "target": 0
    },
    {
      "source": 31,
      "target": 0
    },
    {
      "source": 46,
      "target": 0
    },
    {
      "source": 55,
      "target": 0
    },
    {
      "source": 85,
      "target": 0
    },
    {
      "source": 92,
      "target": 0
    },
  ],
}
```



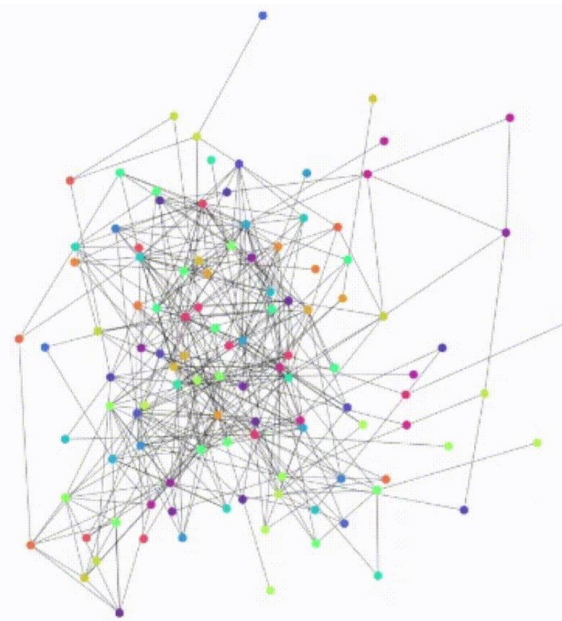
‘Tic-Toc’

- forceSimulation会通过每次‘tick’来更新当前节点的状态：
 - 状态包括位置、速度、加速度等。
- 更新后的状态仅仅为‘状态’：
 - 不会反映到任何图元，仅修改数据。
 - 需要添加修改图元属性的回调函数。
- 人为设置每次tick要如何更新图元
 - simulation.on('tick', ticked);
- 在初始化每个图元后，只要为simulation配置了‘tick’的回调，simulation会自动开始模拟。

```
function ticked() {  
  lines  
    .attr('x1', d => d.source.x)  
    .attr('y1', d => d.source.y)  
    .attr('x2', d => d.target.x)  
    .attr('y2', d => d.target.y);  
  circles  
    .attr('cx', d => d.x)  
    .attr('cy', d => d.y)  
}
```


基于'd3-force'实现力导图

- 数据来源: <http://networkrepository.com/socfb-Caltech36.php>
 - Rossi R, Ahmed N. The network data repository with interactive graph analytics and visualization[C]//Twenty-Ninth AAAI Conference on Artificial Intelligence. 2015.



基于'd3-force'实现力导图

- 编程实例:

```
// the data is from http://networkrepository.com/socfb-Caltech36.php;  
d3.json('socfb-Caltech36.json').then(data => {  
  links = data.links;  
  nodes = []  
  for(let i = 0; i <= data['#nodes']; i++){  
    nodes.push({"index":i});  
  }  
  
  color = d3.scaleSequential(d3.interpolateRainbow)  
    .domain([0, nodes.length-1])  
  
  render_init();  
  
  simulation = d3.forceSimulation(nodes)  
    .force('manyBody', d3.forceManyBody().strength(-30))  
    .force('center', d3.forceCenter(width / 2, height / 2))  
    .force("link", d3.forceLink(links).strength(0.1).distance(100))  
    .on('tick', ticked);  
})
```

```
const render_init = function(){  
  lines = svg.selectAll('line').data(links).join('line')  
    .attr('stroke', 'black')  
    .attr('opacity', 0.8)  
    .attr('stroke-width', .5);  
  circles = svg.selectAll('circle').data(nodes).join('circle')  
    .attr('r', 5)  
    .attr('fill', d => color(d.index))  
}  
  
function ticked() {  
  lines  
    .attr('x1', d => d.source.x)  
    .attr('y1', d => d.source.y)  
    .attr('x2', d => d.target.x)  
    .attr('y2', d => d.target.y);  
  circles  
    .attr('cx', d => d.x)  
    .attr('cy', d => d.y)  
}
```

Tip: 带权重的图?

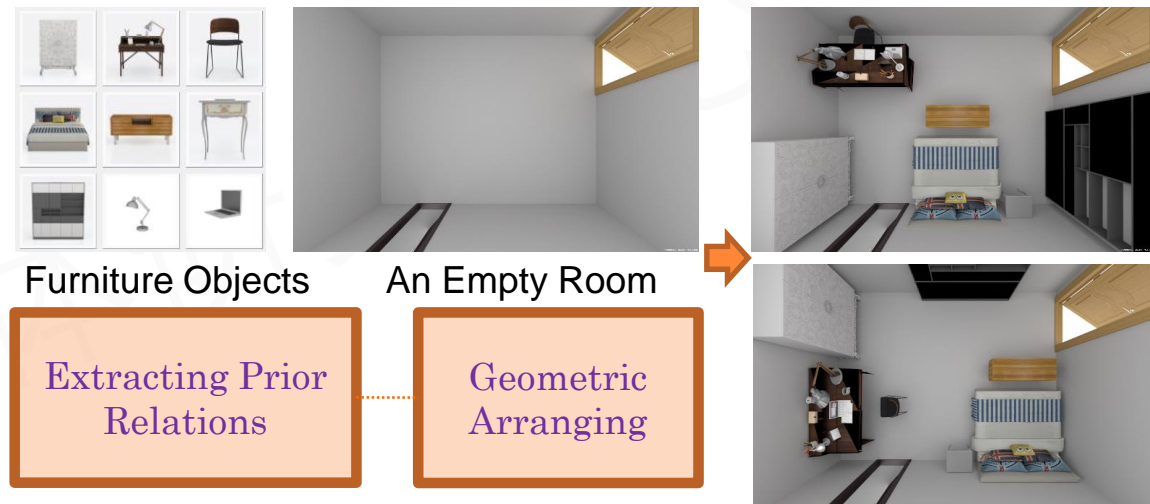
- `d3.forceLink(links).strength(...).distance(...)`:
 - 本质上根据link的权重设置forceLink的strength与distance。
 - 分别输入回调函数，基于每一个‘link’元素来设置各自的力与距离。

- 编程实例:

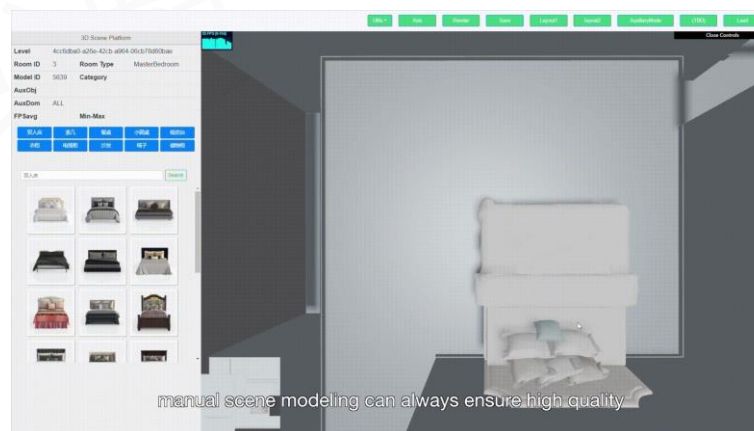
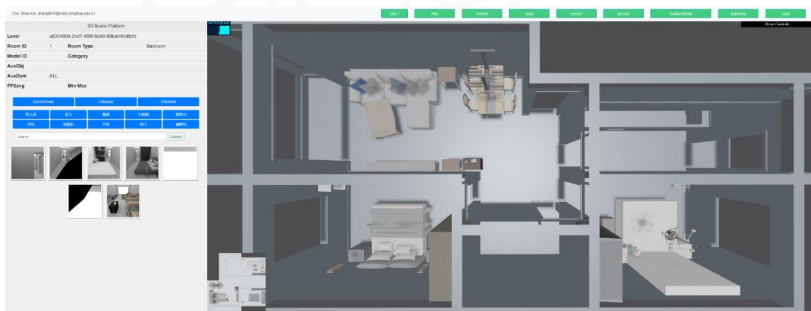
```
function strength(link) {  
    return link.weight * 0.01;  
}  
  
function distance(link) {  
    return 200 * (1 / link.weight);  
}  
  
var simulation = d3.forceSimulation(nodes)  
    .force('charge', d3.forceManyBody())  
    .force('center', d3.forceCenter(width / 2, height / 2))  
    .force("link", d3.forceLink(links).strength(strength).distance(distance))  
    .alphaTarget(0.1)  
    .on('tick', ticked);
```

This is the end of D3.js...but the start for more communications.

- 所有课程中的D3.js资源在Github上长期维护:
 - <https://github.com/Shao-Kui/D3.js-Demos>
 - zhangsk18@mails.tsinghua.edu.cn
 - Welcome to issue me on this project. 🐱
 - D3.js课程较2020年有较大变化, 补充并修改了大量内容, 删减了不必要的内容。
- The Researches of Shao-Kui:
 - Scene Synthesis & Layout Generation.
 - Front-End Rendering & Interaction.
- 特别感谢: 梁缘学长、周文洋。



Song-Hai Zhang, **Shao-Kui Zhang** et al. TVCG 2021 (CCF A)
& **Shao-Kui Zhang**, Wei-Yu Xie et al. GM 2021 (CCF B)



Interactive Demo

Shao-Kui Zhang, Yi-Xiao Li et al. (Anonymous Submission) 2021.
<https://toscode.gitee.com/THU-GGC/scenepatform>

Thank You!~
D3.js

张松海 张少魁 周文洋
清华大学 可视媒体研究中心