
DEPARTMENT OF
COMPUTER SCIENCE
計算機科學系

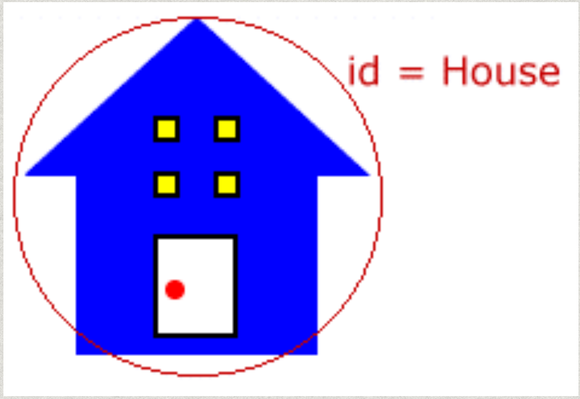
# SCALABLE VECTOR GRAPHICS AND D3.JS

## ITEC2016 - DATA-DRIVEN VISUALIZATION FOR THE WEB

CHAPTER 5 - 2018 - HKBU

DR. MARTIN CHOY

# Recap and more




- What is SVG?
  - SVG is a language for describing 2D vector and mixed vector/raster graphics in XML.
- SVG syntax, structure, coordinate systems and rendering order
- Basic shape and styles
- Today - Document Structure – Grouping and referencing objects

# Grouping Element(1)

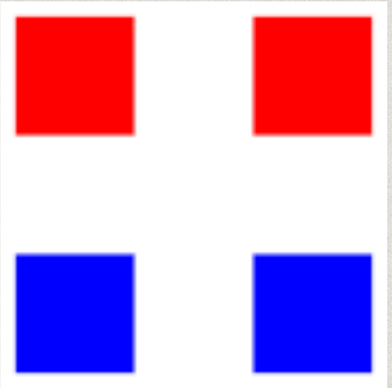
- The <g> element is the element for **grouping** and **naming** collections of drawing elements.
- Enables to **execute same operation on all items in the group**.
- Can be used in conjunction with <desc> and <title> elements to provide description and semantics about the group

# Grouping Element(2)

- Each group can be given the id **attribute** for purposes of animation and re-usability.
- Any <g> element can contain other <g> elements **nested** within it to an arbitrary depth.
- Drawing element that is not contained in a <g> element can be considered as a group of its own.

# Simple Grouping

```
<svg width="500" height="500">

     <!-- Red Square Group -->
    <g id="group1" style="fill:red">
        <rect x="100" y="100" width="100" height="100"/>
        <rect x="300" y="100" width="100" height="100"/>
    </g>

     <!-- Blue Square Group -->
    <g id="group2" style="fill:blue">
        <rect x="100" y="300" width="100" height="100"/>
        <rect x="300" y="300" width="100" height="100"/>
    </g>

</svg>
```

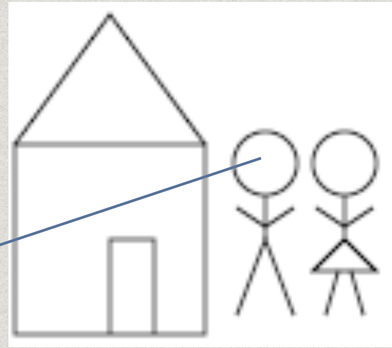


# Another Example (1/3)

```
<svg width="240px" height="240px">
    <g id="house" style="fill: none; stroke: black;">
          <desc>House with door</desc>
          <rect x="6" y="50" width="60" height="60"/>
          <polyline points="6 50, 36 9, 66 50"/>
          <polyline points="36 110, 36 80, 50 80, 50 110"/>
      </g>
    <g id="man" style="fill: none; stroke: black;">
        <desc>Male human</desc>
        <circle cx="85" cy="56" r="10"/>
        <line x1="85" y1="66" x2="85" y2="80"/>
        <polyline points="76 104, 85 80, 94 104"/>
        <polyline points="76 70, 85 76, 94 70"/>
    </g>
    <g id="woman" style="fill: none; stroke: black;">
        <desc>Female human</desc>
        <circle cx="110" cy="56" r="10"/>
        <polyline points="110 66, 110 80, 100 90, 120 90, 110 80"/>
        <line x1="104" y1="104" x2="108" y2="90"/>
        <line x1="112" y1="90" x2="116" y2="104"/>
        <polyline points="101 70, 110 76, 119 70"/>
    </g>
</svg>
```

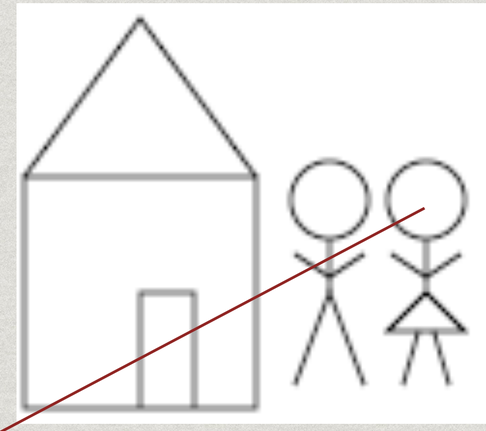
# Another Example (2/3)

```
<svg width="240px" height="240px" viewBox="0 0 240 240">
    <g id="house" style="fill: none; stroke: black;">
        <desc>House with door</desc>
        <rect x="6" y="50" width="60" height="60"/>
        <polyline points="6 50, 36 9, 66 50"/>
        <polyline points="36 110, 36 80, 50 80, 50 110"/>
    </g>
    <g id="man" style="fill: none; stroke: black;">
            <desc>Male human</desc>
            <circle cx="85" cy="56" r="10"/>
            <line x1="85" y1="66" x2="85" y2="80"/>
            <polyline points="76 104, 85 80, 94 104"/>
            <polyline points="76 70, 85 76, 94 70"/>
      </g>
    <g id="woman" style="fill: none; stroke: black;">
        <desc>Female human</desc>
        <circle cx="110" cy="56" r="10"/>
        <polyline points="110 66, 110 80, 100 90, 120 90, 110 80"/>
        <line x1="104" y1="104" x2="108" y2="90"/>
        <line x1="112" y1="90" x2="116" y2="104"/>
        <polyline points="101 70, 110 76, 119 70"/>
    </g>
</svg>
```



# Another Example (3/3)

```
<svg width="240px" height="240px" viewBox="0 0 240 240">
    <g id="house" style="fill: none; stroke: black;">
        <desc>House with door</desc>
        <rect x="6" y="50" width="60" height="60"/>
        <polyline points="6 50, 36 9, 66 50"/>
        <polyline points="36 110, 36 80, 50 80, 50 110"/>
    </g>
    <g id="man" style="fill: none; stroke: black;">
        <desc>Male human</desc>
        <circle cx="85" cy="56" r="10"/>
        <line x1="85" y1="66" x2="85" y2="80"/>
        <polyline points="76 104, 85 80, 94 104"/>
        <polyline points="76 70, 85 76, 94 70"/>
    </g>
    <g id="woman" style="fill: none; stroke: black;">
        <desc>Female human</desc>
        <circle cx="110" cy="56" r="10"/>
        <polyline points="110 66, 110 80, 100 90, 120 90, 110 80"/>
        <line x1="104" y1="104" x2="108" y2="90"/>
        <line x1="112" y1="90" x2="116" y2="104"/>
        <polyline points="101 70, 110 76, 119 70"/>
    </g>
</svg>
```



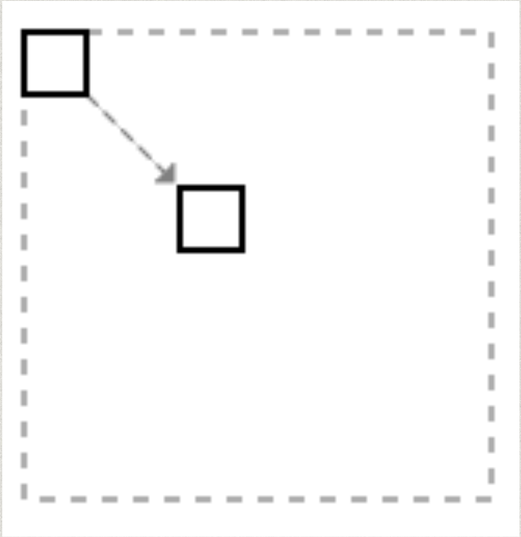
# Transformations(1)

- A new **user coordinate system can be established by specifying transformations** in the form of a transform attribute on a group of graphical elements.
- The transform attribute transforms all user space coordinates and lengths on the given element and all of its ancestors.
- Transformations maybe nested and so have cumulative effect.

# Transformations(2)

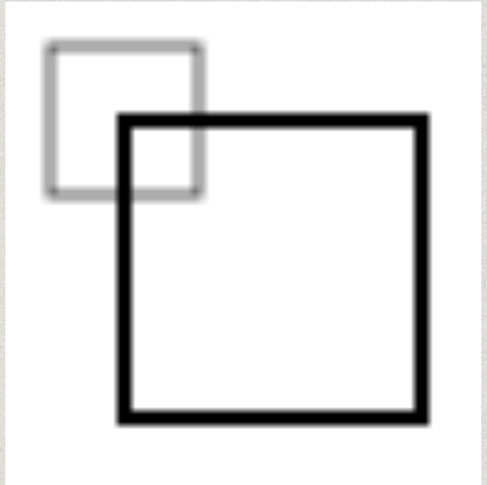
- Common transformations:
  - **translate(x, y)** - establish a new coordinate system whose origin is at (x, y).
  - **rotate(a)** – rotates the coordinate system by a degrees around the origin.
  - **scale(a, b)** – scales the coordinate system – x axis by a and y axis by b.

# Linear Transformations



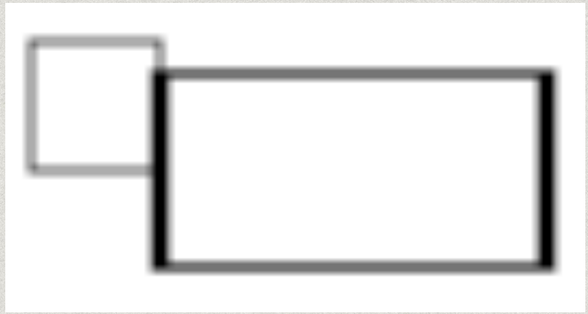
```
<svg width="200px" height="200px">

    <g id="square">
        <rect x="10" y="10" width="20" height="20"
            style="fill: none; stroke:black; stroke-width: 2;"/>
    </g>

    <use xlink:href="#square" transform="translate(50,50)"/>
</svg>
```

# Scale Transformations (1)



```
<svg width="300" height="300">

    <g id="square">
        <rect x="10" y="10" width="20" height="20"
                    style="fill: none; stroke: black;"/>
    </g>
    <use xlink:href="#square" transform="scale(2)"/>

</svg>
```

# Scale Transformations (2)

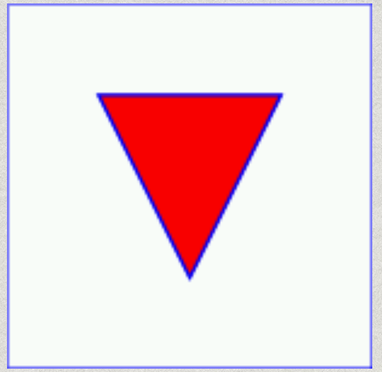


```
<svg width="200px" height="200px">

    <g id="square">
        <rect x="10" y="10" width="20" height="20"
                                style="fill: none; stroke: black;"/>
    </g>
    <use xlink:href="#square" transform="scale(3, 1.5)"/>

</svg>
```

# Paths(1)

- **Paths represent the outline of a shape** which can be filled, stroked, used as a clipping path, or any combination of the three.
- **Paths are described by a set of points**.
- The geometry of the path is defined in terms of **moveto (M m)**, **lineto (L l)**, and **closepath (Z)**.
- Path is represented in SVG by the <path> element.

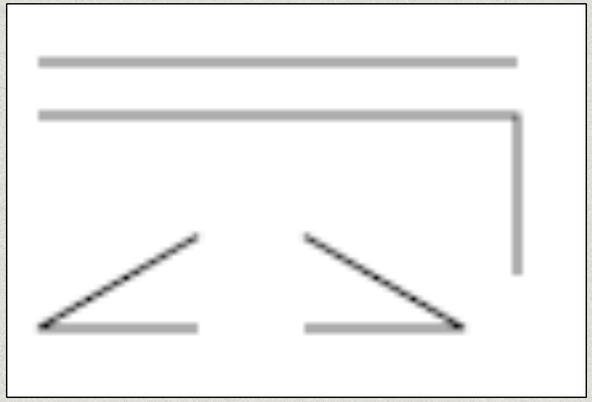
# Paths(2)



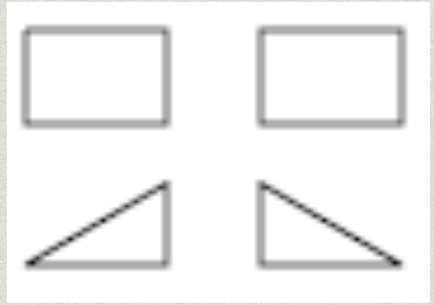
```
<svg width="400" height="400">

    <rect x="1" y="1" width="398" height="398"
                        style="fill:none; stroke:blue"/>
    <path d="M 100 100 L 300 100 L 200 300 z"
                       style="fill:red; stroke:blue; stroke-width:3"/>

</svg>
```

# Paths (move to and line to)

```
<svg width="200px" height="200px">

    <g transform="translate(20,20)">
        <g style="stroke: black; fill: none;">
            <path d="M 10 10 L 100 10"/>
            <path d="M 10, 20  L 100, 20  L  100,50"/>
            <path d="M 40 60, L 10 60, L 40 42.68,
                     M 60 60, L 90 60, L 60 42.68"/>
        </g>
    </g>

</svg>
```



# Paths (close path)

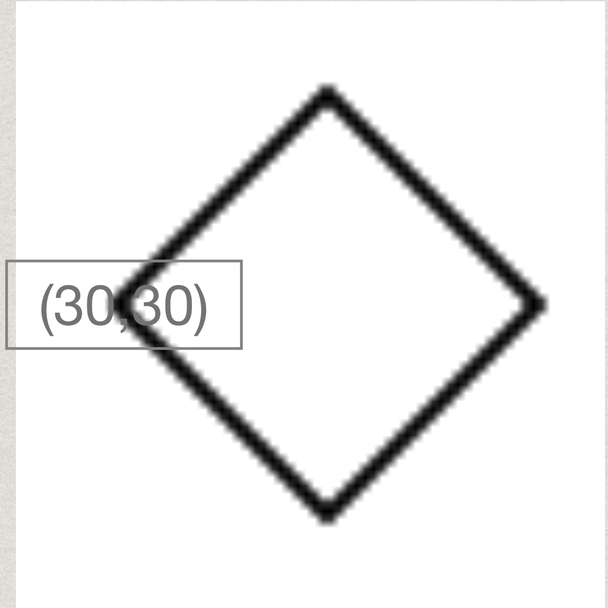


```
<svg width="200px" height="200px">

    <g transform="translate(20,20)">
        <g style="stroke: black; fill: none;">
            <path d="M 10 10, L 40 10, L 40 30, L 10 30, L 10 10"/>
            <path d="M 60 10, L 90 10, L 90 30, L 60 30, Z"/>
            <path d="M 40 60, L 10 60, L 40 42.68,
                     Z M 60 60, L 90 60, L 60 42.68, Z"/>
        </g>
    </g>

</svg>
```

# Paths shortcuts

- Horizontal line (H h)
- Vertical line (V v)
- Multiple sets of coordinate

```
<!-- 6 paths draw the same diamond -->
<path d="M 30 30 L 55 5 L 80 30 L 55 55 Z"/>
<path d="M 30 30 L 55 5 80 30 55 55 Z"/>
<path d="M 30 30 55 5 80 30 55 55 Z"/>
<path d="m 30 30 l 25 -25 l 25 25 l -25 25 z"/>
<path d="m 30 30 l 25 -25 25 25 -25 25 z"/>
<path d="m 30 30 25 -25 25 25 -25 25 z"/>
```

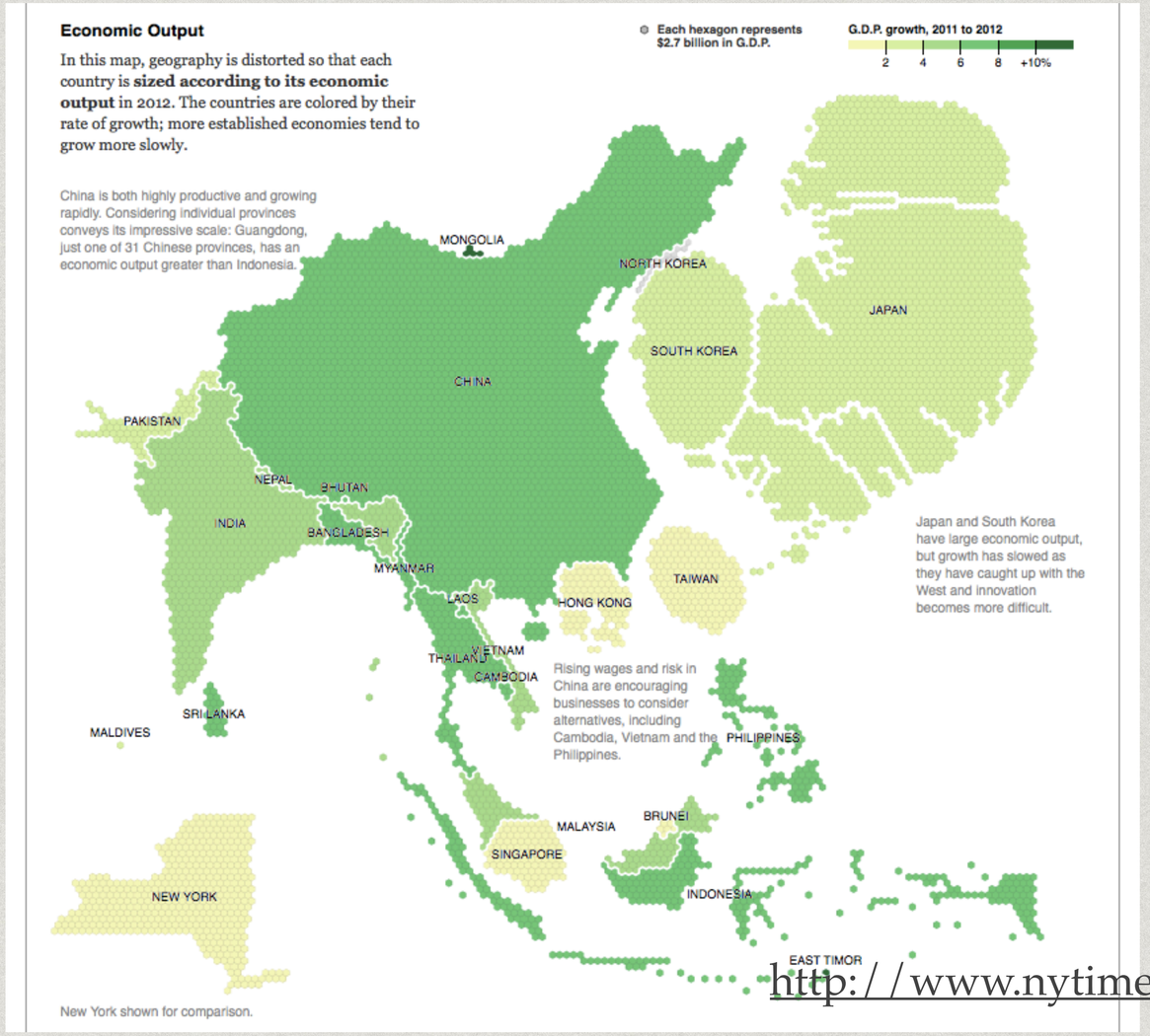


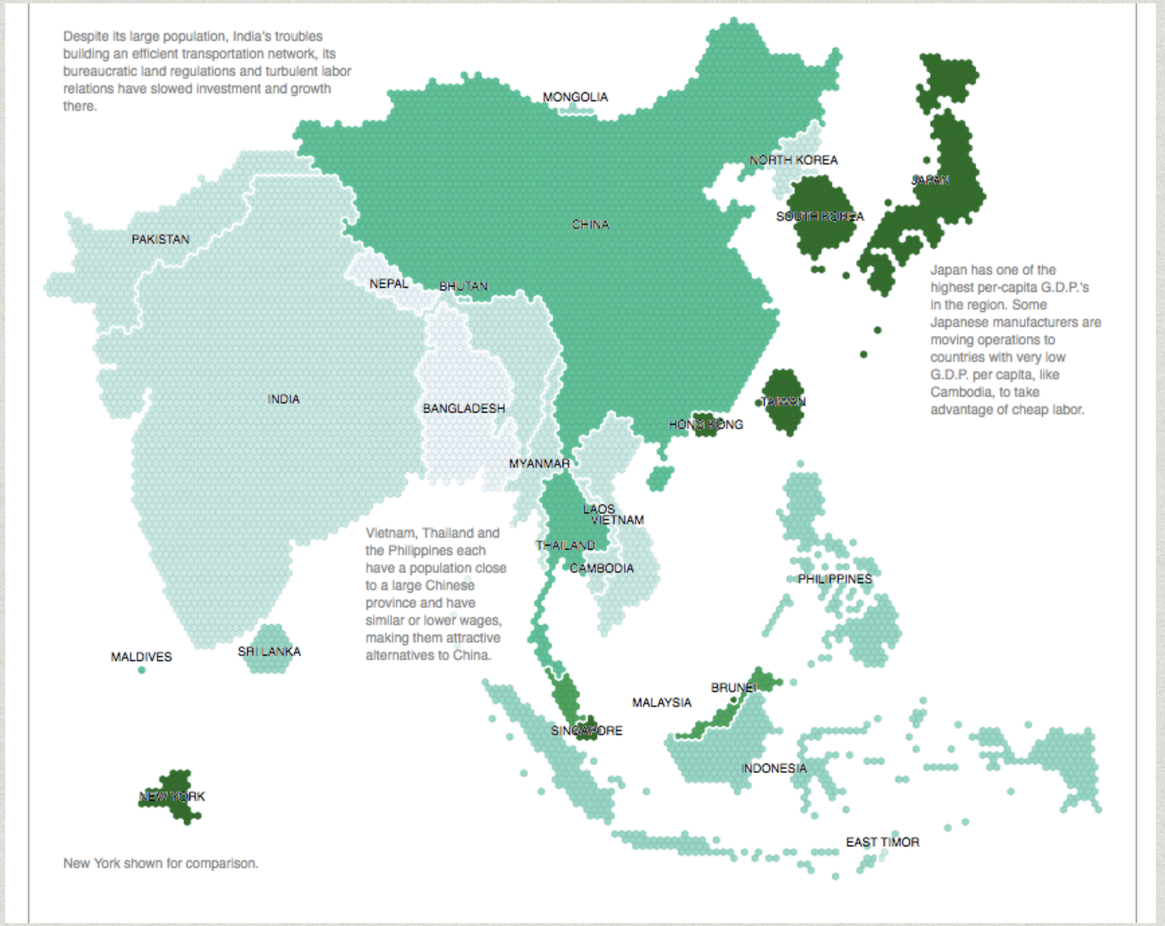
# The Text element

- <text> **is used to specify text that is rendered with other graphical elements.**
- This means that we can apply transformation, clipping, masking, and so to text.
- Fonts are as specified in CSS.

# GEOGRAPHIC DATA IN D3.JS

# Geographic Data




http://www.nytimes.com/interactive/2013/04/08/business/global/asia-map.html

# Geographic Data

- The first task for any map is **finding geometry**.
  - Geometry could be available in **GeoJSON** or **TopoJSON** formats.
    - They are first-class citizens in D3.js.
  - Other formats like **shapefile**, **csv** and **deb** are also useful as there’re converters such as
    - http://mapshaper.org

# Geographic Data

- **Hong Kong Districts** TopoJSON:

  https://github.com/ywng/d3-js-map-hong-kong

- TopoJSON Client Documentation:

  https://github.com/topojson/topojson-client/blob/master/README.md#mesh

- D3 provides helper function to **parse JSON file**.

```
d3.json("HKG_adm.json").then(function (topoData) {
```

# Geographic Data

- A geometry object could then be constructed as a **Path in SVG**.

```
svg.append("g")
    .attr("class", "districts")
    .selectAll("path")
    .data(topojson.feature(topoData, topoData.objects.HKG_adm1_1).features)
    .enter().append("path")
    .attr("class", "district")
    .attr("d", geoPath);
```

- Overlapped arc segments are treated as borders, also a Path in SVG.

```
svg.append("path")
        .attr("class", "district-borders")
        .attr("d", geoPath(topojson.mesh(topoData, topoData.objects.HKG_adm1_1,
                                         function (a, b) { return a !== b; })));
```

# D3.JS FORCE LAYOUT

# D3.js Force Layout

- D3's force layout uses a **physics based simulator** for **positioning visual elements**
  - all elements **repel** one another
  - elements are **attracted to center(s) of gravity**
  - linked elements (e.g. friendship) are a **fixed distance apart** (network visualization)
  - elements may not **overlap** (collision detection)

# D3.js Force Layout

- This creates 18 nodes for simulation.
- node is an array of 18 <g> elements with class named as “node”.
- enter() helps to create more <g> if the size of abuses grows
- Each <g> has a circle in it.

```
simulation.nodes(abuses);
```

```
var node = svg.selectAll(".node")
    .data(abuses)
    .enter().append("g")
    .attr("class", "node");
```

```
node.append('circle')
    .attr('r', 30)
    .attr('fill', "blue");
```

# D3.js Force Layout

```
simulation.on("tick", function () {
    node
        .attr("transform", function (d) {
            return "translate(" + d.x + "," + d.y + ")";
        });
});
```

- Browsers will repeatedly call tick and execute the function specified.
- This will move the objects according to the forces applied.