# 【D3协作翻译任务及认领 2014-04-28】



## 前言：

没有难度，没有负担，快乐只因为你喜欢！我们有你而更精彩！

## 原则：

认领优先，协同翻译，事后校验，整合成稿。可以多人认领同个任务，讨论式翻译。认领可以直接在新任务列表后面加上你的昵称，重新上传即可（认领满了的任务也可以多个人共同参与，参与相关讨论）。

## 任务认领说明：

|  |  |
| --- | --- |
| **D3协作任务及认领** | |
| 任务描述 | 需认领新任务部分：  d3.layout部分、d3.geo部分、d3.geom部分、d3.behavior部分； |
| 时间 | 2014-04-28 至2014-05-03。  （2014-05-03）之前提交最终修改版。提交直接上传群文件即可。 |
| 网址 | <https://github.com/mbostock/d3/wiki/API-Reference> |
| 做法 | 【新任务认领说明】：  大家愿意翻译的同学，选这几部分中的感兴趣的几个。找到你认领的部分直接打开。  然后复制下api内容到word里面，规整一下格式（不要把网页上的格式也保留在word中，最好可以用格式刷刷一下格式），把相关的解释部分用中文替换，其中不懂的部分用红色标注；原来有的超级链接保留；你认为需要补充的知识，全部备注到文档最后，能补充的补充，不能补充的备注一下。  ③内容可以意译，尽量直译，出现的内容都要翻译。翻译中内容都以文本形式展示，不要出现图片等非文本形式。  ④具体示例如：群文件【core.math】api详细.docx所示，尽量按照里边的样式约束你的文档。  ⑤文件名称：“Task-”+任务编号。  ⑥做完后直接上传群文件。 |

## 【协作任务5】2014-04-28：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **协作任务周表** | | | | |
| **任务** | | **昵称** | **任务量** | **任务编号** |
| D3.layout.bundle | 概要 [#](https://github.com/mbostock/d3/wiki/Bundle-Layout#wiki-bundle) d3.layout.bundle()[#](https://github.com/mbostock/d3/wiki/Bundle-Layout#wiki-_bundle) bundle(links) |  | 361 | D3.bundle1 |
| D3.layout.chord | 概要 [#](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-chord) d3.layout.chord()[#](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-matrix) chord.matrix([matrix]) |  | 361 | D3.chord1 |
| [#](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-padding) chord.padding([padding])  [#](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-sortGroups) chord.sortGroups([comparator])  [#](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-sortSubgroups) chord.sortSubgroups([comparator])  [#](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-sortChords) chord.sortChords([comparator]) |  | 264 | D3.chord2 |
| [#](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-chords) chord.chords()  [#](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-groups) chord.groups() |  | 280 | D3.chord3 |
| D3.layout.cluster | 概要 [#](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-cluster) d3.layout.cluster()[#](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-_cluster) cluster(root)  [#](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-nodes) cluster.nodes(root) |  | 278 | D3.cluster1 |
| [#](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-links) cluster.links(nodes)  [#](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-children) cluster.children([children]) |  | 247 | D3.cluster2 |
| [#](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-sort) cluster.sort([comparator])  [#](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-separation) cluster.separation([separation]) |  | 220 | D3.cluster3 |
| [#](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-size) cluster.size([size])  [#](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-nodeSize) cluster.nodeSize([nodeSize])  [#](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-value) cluster.value([value]) |  | 218 | D3.cluster4 |
| D3.layout.hierachy | 概要 [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-hierarchy) d3.layout.hierarchy()[#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-_hierarchy) hierarchy(root) |  | 260 | D3.hierachy1 |
| [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-links) hierarchy.links(nodes)  [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-children) hierarchy.children([accessor]) |  | 249 | D3.hierachy2 |
| [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-sort) hierarchy.sort([comparator])  [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-value) hierarchy.value([value])  [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-revalue) hierarchy.revalue(root) |  | 241 | D3.hierachy3 |
| D3.layout.histogram | 概要 [#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-histogram) d3.layout.histogram()[#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-_histogram) histogram(values[, index])[#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-value) histogram.value([accessor]) |  | 282 | D3.histogram1 |
| [#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-range) histogram.range([range])  [#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-bins) histogram.bins()  [#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-bins) histogram.bins(count)  [#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-bins) histogram.bins(thresholds)  [#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-bins) histogram.bins(function) [#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-frequency) histogram.frequency([frequency]) |  | 316 | D3.histogram2 |
| D3.layout.partition | 概要 [#](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-partition) d3.layout.partition()[#](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-_partition) partition(root)  [#](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-nodes) partition.nodes(root) |  | 399 | D3.partition1 |
| [#](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-links) partition.links(nodes)  [#](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-children) partition.children([children]) |  | 246 | D3.partition2 |
| [#](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-sort) partition.sort([comparator])  [#](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-value) partition.value([value])  [#](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-size) partition.size([size]) |  | 207 | D3.partition3 |
| D3.layout.pie | 概要 [#](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-pie) d3.layout.pie()[#](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-_pie) pie(values[, index]) |  | 293 | D3.pie1 |
| [#](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-value) pie.value([accessor])  [#](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-sort) pie.sort([comparator])  [#](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-startAngle) pie.startAngle([angle])  [#](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-endAngle) pie.endAngle([angle]) |  | 303 | D3.pie2 |
| D3.layout.stack | 概要 [#](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-stack) d3.layout.stack()[#](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-_stack) stack(layers[, index]) |  | 295 | D3.stack1 |
| [#](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-values) stack.values([accessor])  [#](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-offset) stack.offset([offset]) |  | 332 | D3.stack2 |
| [#](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-order) stack.order([order])  [#](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-x) stack.x([accessor]) |  | 282 | D3.stack3 |
| [#](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-y) stack.y([accessor]) [#](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-out) stack.out([setter]) |  | 221 | D3.stack4 |
| D3.layout.tree | 概要 [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-tree) d3.layout.tree()[#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-_tree) tree(root)  [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-nodes) tree.nodes(root) |  | 313 | D3.tree1 |
| [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-links) tree.links(nodes)  [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-children) tree.children([children]) |  | 247 | D3.tree2 |
| [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-separation) tree.separation([separation])  [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-size) tree.size([size]) |  | 220 | D3.tree3 |
| [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-nodeSize) tree.nodeSize([nodeSize])  [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-sort) tree.sort([comparator])  [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-value) tree.value([value]) |  | 283 | D3.tree4 |
| D3.layout.treemap | 概要 [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-treemap) d3.layout.treemap()[#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-_treemap) treemap(root)  [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-nodes) treemap.nodes(root) |  | 365 | D3.treemap1 |
| [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-links) treemap.links(nodes)  [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-children) treemap.children([children]) |  | 247 | D3.treemap2 |
| [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-sort) treemap.sort([comparator])  [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-value) treemap.value([value])  [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-size) treemap.size([size]) |  | 207 | D3.treemap3 |
| [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-padding) treemap.padding([padding])  [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-round) treemap.round([round]) |  | 202 | D3.treemap4 |
| [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-sticky) treemap.sticky([sticky])  [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-mode) treemap.mode([mode]) |  | 223 | D3.treemap5 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 新任务原文

## [d3.layout (Layouts)](https://github.com/mbostock/d3/wiki/Layouts)

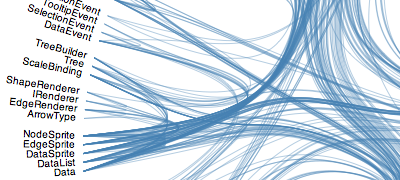
### [Bundle](https://github.com/mbostock/d3/wiki/Bundle-Layout)

* [d3.layout.bundle](https://github.com/mbostock/d3/wiki/Bundle-Layout#wiki-bundle) - construct a new default bundle layout.
* [bundle](https://github.com/mbostock/d3/wiki/Bundle-Layout#wiki-_bundle) - apply Holten's hierarchical bundling algorithm to edges.

**Bundle Layout**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Layouts](https://github.com/mbostock/d3/wiki/Layouts) ▸ **Bundle Layout**

Implements Danny Holten's [hierarchical edge bundling](http://www.win.tue.nl/%7Edholten/papers/bundles_infovis.pdf) algorithm. For each input link, a path is computed that travels through the tree, up the parent hierarchy to the least common ancestor, and then back down to the destination node. This sequence of nodes can then be used in conjunction with other [hierarchical layouts](https://github.com/mbostock/d3/wiki/Hierarchy-Layout), such as [cluster](https://github.com/mbostock/d3/wiki/Cluster-Layout) to generate bundled splines between nodes:



For example, consider this visualization of [software dependencies](http://bl.ocks.org/mbostock/1044242).

#### [#](https://github.com/mbostock/d3/wiki/Bundle-Layout#wiki-bundle) d3.layout.bundle()

Constructs a new default bundle layout. Currently, the bundle layout is stateless and thus only has a default configuration. The returned layout object is both an object and a function. That is: you can call the layout like any other function, and the layout has additional methods that change its behavior. Like other classes in D3, layouts follow the method chaining pattern where setter methods return the layout itself, allowing multiple setters to be invoked in a concise statement.

#### [#](https://github.com/mbostock/d3/wiki/Bundle-Layout" \l "wiki-_bundle) bundle(links)

Evaluates the bundle layout on the specified array of links, returning the computed path from the source to the target, through the [least common ancestor](http://en.wikipedia.org/wiki/Lowest_common_ancestor). Each input link must have two attributes:

* source - the source node.
* target - the target node.

Furthermore, each node must have one attribute:

* parent - the parent node.

This is a subset of the fields generated by the [hierarchy layouts](https://github.com/mbostock/d3/wiki/Hierarchy-Layout). The return value of the layout is an array of paths, where each path is represented as an array of nodes. Thus, the bundle layout does not compute the basis splines directly; instead, it returns an array of nodes which implicitly represent the control points of the spline. You can use this array in conjunction with [d3.svg.line](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-line) or [d3.svg.line.radial](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-line_radial) to generate the splines themselves. For example, if you were to use a [cluster](https://github.com/mbostock/d3/wiki/Cluster-Layout):

var cluster = d3.layout.cluster()

.size([2 \* Math.PI, 500]);

A suitable line generator for hierarchical edge bundling might be:

var line = d3.svg.line.radial()

.interpolate("bundle")

.tension(.85)

.radius(function(d) { return d.y; })

.angle(function(d) { return d.x; });

The bundle layout is designed to work in conjunction with the line generator's "bundle" interpolation mode, though technically speaking you can use any interpolator or shape generator. Holten's bundle strength parameter is exposed as the line's [tension](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-line_tension).

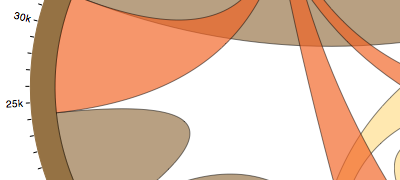
### [Chord](https://github.com/mbostock/d3/wiki/Chord-Layout)

* [d3.layout.chord](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-chord) - produce a chord diagram from a matrix of relationships.
* [chord.matrix](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-matrix) - get or set the matrix data backing the layout.
* [chord.padding](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-padding) - get or set the angular padding between chord segments.
* [chord.sortGroups](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-sortGroups) - get or set the comparator function for groups.
* [chord.sortSubgroups](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-sortSubgroups) - get or set the comparator function for subgroups.
* [chord.sortChords](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-sortChords) - get or set the comparator function for chords (z-order).
* [chord.chords](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-chords) - retrieve the computed chord angles.
* [chord.groups](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-groups) - retrieve the computed group angles.

**Chord Layout**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Layouts](https://github.com/mbostock/d3/wiki/Layouts) ▸ **Chord Layout**

**Chord diagrams** show relationships among a group of entities. For example, consider a hypothetical population of people with different hair colors: black, blonde, brown and red. Each person in this population has a preferred hair color for a dating partner; of the 29,630 (hypothetical) people with black hair, 40% (11,975) prefer partners with the same hair color. This preference is asymmetric: for example, only 10% of people with blonde hair prefer black hair, while 20% of people with black hair prefer blonde hair.



A chord diagram visualizes these relationships by drawing quadratic Bézier curves between arcs. The source and target arcs represents two mirrored subsets of the total population, such as the number of people with black hair that prefer blonde hair, and the number of people with blonde hair that prefer black hair. As another example, consider this chord diagram of [software dependencies](http://bl.ocks.org/mbostock/1046712).

The chord layout is designed to work in conjunction with the [chord shape](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-chord) and the [arc shape](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-arc). The layout is used to generate data objects which describe the chords, serving as input to the chord shape. The layout also generates descriptions for the groups, which can be used as input to the arc shape.

#### [#](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-chord) d3.layout.chord()

Constructs a new chord layout. By default, the input data is not sorted, and there is no padding between groups. Unlike some of the other layouts, the chord layout is not a function to be applied to data; instead, data is specified by setting the associated [matrix](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-matrix), and retrieved using the [chords](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-chords) and [groups](https://github.com/mbostock/d3/wiki/Chord-Layout#wiki-groups) accessors.

#### [#](https://github.com/mbostock/d3/wiki/Chord-Layout" \l "wiki-matrix) chord.matrix([matrix])

If matrix is specified, sets the input data matrix used by this layout. If matrix is not specified, returns the current data matrix, which defaults to undefined. The input matrix must be a [square matrix]([http://en.wikipedia.org/wiki/Matrix\_(mathematics\)#Square\_matrices](http://en.wikipedia.org/wiki/Matrix_%28mathematics%5C%29" \l "Square_matrices)) of numbers, such as:

[[11975, 5871, 8916, 2868],

[ 1951, 10048, 2060, 6171],

[ 8010, 16145, 8090, 8045],

[ 1013, 990, 940, 6907]]

Each row in the matrix corresponds to a distinct group, such as a hair color in the above example. Each column i in the matrix corresponds to the same group as row i; the cell ij corresponds to the relationship from group i to group j.

#### [#](https://github.com/mbostock/d3/wiki/Chord-Layout" \l "wiki-padding) chord.padding([padding])

If padding is specified, sets the angular padding between groups to the specified value in [radians](http://en.wikipedia.org/wiki/Radian). If padding is not specified, returns the current padding, which defaults to zero. You may wish to compute the padding as a function of the number of groups (the number of rows or columns in the associated matrix).

#### [#](https://github.com/mbostock/d3/wiki/Chord-Layout" \l "wiki-sortGroups) chord.sortGroups([comparator])

If comparator is specified, sets the sort order of groups (rows) for the layout using the specified comparator function. The comparator function is invoked for pairs of rows, being passed the sum of row i and row j. Typically, the comparator should be specified as either [d3.ascending](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_ascending) or [d3.descending](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_descending). If comparator is not specified, returns the current group sort order, which defaults to null for no sorting.

#### [#](https://github.com/mbostock/d3/wiki/Chord-Layout" \l "wiki-sortSubgroups) chord.sortSubgroups([comparator])

If comparator is specified, sets the sort order of subgroups (columns within rows) for the layout using the specified comparator function. The comparator function is invoked for pairs of cells, being passed the value of each cell. Typically, the comparator should be specified as either ascending or descending. If comparator is not specified, returns the current subgroup sort order, which defaults to null for no sorting.

#### [#](https://github.com/mbostock/d3/wiki/Chord-Layout" \l "wiki-sortChords) chord.sortChords([comparator])

If comparator is specified, sets the sort order of chords (z-order) for the layout using the specified comparator function. The comparator function is invoked for pairs of chords, being passed the minimum value of the associated source and target cells. Typically, the comparator should be specified as either ascending or descending. If comparator is not specified, returns the current chord sort order, which defaults to null for no sorting.

#### [#](https://github.com/mbostock/d3/wiki/Chord-Layout" \l "wiki-chords) chord.chords()

Returns the computed chord objects, given the layout's current configuration and associated matrix. If the chord objects were previously-computed, this method returns the cached value. Changing any attribute of the layout implicitly clears the previously-computed chords, if any, such that the next call to this method will recompute the layout. The returned objects have the following properties:

* source - an object describing the source.
* target - an object describing the target.

These objects, in turn, describe the underlying entity:

* index - the row index, i.
* subindex - the column index, j.
* startAngle - the start angle of the arc, in radians.
* endAngle - the end angle of the arc, in radians.
* value - the value of the associated cell ij, a number.

Note that these objects conveniently match the default accessors for the [chord](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-chord) generator; however, you can still override the accessors to tweak the layout, or simply manipulate the returned objects.

#### [#](https://github.com/mbostock/d3/wiki/Chord-Layout" \l "wiki-groups) chord.groups()

Returns the computed group objects, given the layout's current configuration and associated matrix. If the group objects were previously-computed, this method returns the cached value. Changing any attribute of the layout implicitly clears the previously-computed groups, if any, such that the next call to this method will recompute the layout. The returned objects have the following properties:

* index - the row index, i.
* startAngle - the start angle of the arc, in radians.
* endAngle - the end angle of the arc, in radians.
* value - the sum of the associated row i, a number.

Note that these objects conveniently match the default accessors for the [arc](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-arc) generator; however, you can still override the accessors to tweak the layout, or simply manipulate the returned objects.

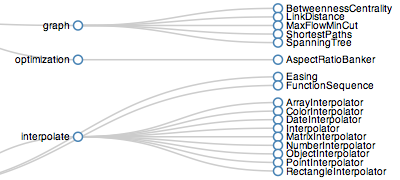
### [Cluster](https://github.com/mbostock/d3/wiki/Cluster-Layout)

* [d3.layout.cluster](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-cluster) - cluster entities into a dendrogram.
* [cluster](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-_cluster) - alias for cluster.nodes.
* [cluster.nodes](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-nodes) - compute the cluster layout and return the array of nodes.
* [cluster.links](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-links) - compute the parent-child links between tree nodes.
* [cluster.children](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-children) - get or set the accessor function for child nodes.
* [cluster.sort](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-sort) - get or set the comparator function for sibling nodes.
* [cluster.separation](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-separation) - get or set the spacing function between neighboring nodes.
* [cluster.size](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-size) - get or set the layout size in x and y.
* [cluster.nodeSize](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-nodeSize) - specify a fixed size for each node.

**Cluster Layout**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Layouts](https://github.com/mbostock/d3/wiki/Layouts) ▸ [Hierarchy](https://github.com/mbostock/d3/wiki/Hierarchy-Layout) ▸ **Cluster Layout**

The **cluster layout** produces [dendrograms](http://en.wikipedia.org/wiki/Dendrogram): node-link diagrams that place leaf nodes of the tree at the same depth. For example, a cluster layout can be used to organize software classes in a package hierarchy:



Like other classes in D3, layouts follow the method chaining pattern where setter methods return the layout itself, allowing multiple setters to be invoked in a concise statement.

#### [#](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-cluster) d3.layout.cluster()

Creates a new cluster layout with the default settings: the default sort order is null; the default children accessor assumes each input data is an object with a children array; the default separation function uses one node width for siblings, and two node widths for non-siblings; the default size is 1×1.

#### [#](https://github.com/mbostock/d3/wiki/Cluster-Layout" \l "wiki-_cluster) cluster(root) [#](https://github.com/mbostock/d3/wiki/Cluster-Layout" \l "wiki-nodes) cluster.nodes(root)

Runs the cluster layout, returning the array of nodes associated with the specified root node. The cluster layout is part of D3's family of [hierarchical](https://github.com/mbostock/d3/wiki/Hierarchy-Layout) layouts. These layouts follow the same basic structure: the input argument to the layout is the root node of the hierarchy, and the output return value is an array representing the computed positions of all nodes. Several attributes are populated on each node:

* parent - the parent node, or null for the root.
* children - the array of child nodes, or null for leaf nodes.
* depth - the depth of the node, starting at 0 for the root.
* x - the computed x-coordinate of the node position.
* y - the computed y-coordinate of the node position.

Although the layout has a size in x and y, this represents an arbitrary coordinate system; for example, you can treat x as a radius and y as an angle to produce a radial rather than Cartesian layout.

#### [#](https://github.com/mbostock/d3/wiki/Cluster-Layout" \l "wiki-links) cluster.links(nodes)

Given the specified array of nodes, such as those returned by [nodes](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-nodes), returns an array of objects representing the links from parent to child for each node. Leaf nodes will not have any links. Each link is an object with two attributes:

* source - the parent node (as described above).
* target - the child node.

This method is useful for retrieving a set of link descriptions suitable for display, often in conjunction with the [diagonal](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-diagonal) shape generator. For example:

svg.selectAll("path")

.data(cluster.links(nodes))

.enter().append("path")

.attr("d", d3.svg.diagonal());

#### [#](https://github.com/mbostock/d3/wiki/Cluster-Layout" \l "wiki-children) cluster.children([children])

If children is specified, sets the specified children accessor function. If children is not specified, returns the current children accessor function, which by default assumes that the input data is an object with a children array:

function children(d) {

return d.children;

}

Often, it is convenient to load the node hierarchy using [d3.json](https://github.com/mbostock/d3/wiki/Requests#wiki-d3_json), and represent the input hierarchy as a nested [JSON](http://json.org) object. For example:

{

"name": "flare",

"children": [

{

"name": "analytics",

"children": [

{

"name": "cluster",

"children": [

{"name": "AgglomerativeCluster", "size": 3938},

{"name": "CommunityStructure", "size": 3812},

{"name": "MergeEdge", "size": 743}

]

},

{

"name": "graph",

"children": [

{"name": "BetweennessCentrality", "size": 3534},

{"name": "LinkDistance", "size": 5731}

]

}

]

}

]

}

The children accessor is first invoked for root node in the hierarchy. If the accessor returns null, then the node is assumed to be a leaf node at the layout traversal terminates. Otherwise, the accessor should return an array of data elements representing the child nodes.

#### [#](https://github.com/mbostock/d3/wiki/Cluster-Layout" \l "wiki-sort) cluster.sort([comparator])

If comparator is specified, sets the sort order of sibling nodes for the layout using the specified comparator function. If comparator is not specified, returns the current group sort order, which defaults to null for no sorting. The comparator function is invoked for pairs of nodes, being passed the input data for each node. The default comparator is null, which disables sorting and uses tree traversal order. For example, to sort sibling nodes in descending order by the associated input data's string name attribute, say:

function comparator(a, b) {

return d3.ascending(a.name, b.name);

}

See [d3.ascending](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_ascending) or [d3.descending](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_descending) for details.

#### [#](https://github.com/mbostock/d3/wiki/Cluster-Layout" \l "wiki-separation) cluster.separation([separation])

If separation is specified, uses the specified function to compute separation between neighboring nodes. If separation is not specified, returns the current separation function, which defaults to:

function separation(a, b) {

return a.parent == b.parent ? 1 : 2;

}

A variation that is more appropriate for radial layouts reduces the separation gap proportionally to the radius:

function separation(a, b) {

return (a.parent == b.parent ? 1 : 2) / a.depth;

}

The separation function is passed two neighboring nodes a and b, and must return the desired separation between nodes. The nodes are typically siblings, though the nodes may also be cousins (or even more distant relations) if the layout decides to place such nodes adjacent.

#### [#](https://github.com/mbostock/d3/wiki/Cluster-Layout" \l "wiki-size) cluster.size([size])

If size is specified, sets the available layout size to the specified two-element array of numbers representing x and y. If size is not specified, returns the current size, which defaults to 1×1, or null if a [nodeSize](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-nodeSize) is in use. Although the layout has a size in x and y, this represents an arbitrary coordinate system. For example, to produce a radial layout where the tree breadth (x) in measured in degrees, and the tree depth (y) is a radius r in pixels, say [360, r].

#### [#](https://github.com/mbostock/d3/wiki/Cluster-Layout" \l "wiki-nodeSize) cluster.nodeSize([nodeSize])

If nodeSize is specified, sets a fixed size for each node as a two-element array of numbers representing x and y. If nodeSize is not specified, returns the current node size, which defaults to null, meaning that the layout has an overall fixed size, which can be retrieved using [size](https://github.com/mbostock/d3/wiki/Cluster-Layout#wiki-size).

#### [#](https://github.com/mbostock/d3/wiki/Cluster-Layout" \l "wiki-value) cluster.value([value])

If value is specified, sets the value accessor to the specified function. If value is not specified, returns the current value accessor which defaults to null, meaning that the value attribute is not computed. If specified, the value accessor is invoked for each input data element, and must return a number representing the numeric value of the node. This value has no effect on the cluster layout, but is generic functionality provided by hierarchy layouts.

### [Hierarchy](https://github.com/mbostock/d3/wiki/Hierarchy-Layout)

* [d3.layout.hierarchy](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-hierarchy) - derive a custom hierarchical layout implementation.
* [hierarchy](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-_hierarchy) - alias for hierarchy.nodes.
* [hierarchy.nodes](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-nodes) - compute the layout and return the array of nodes.
* [hierarchy.links](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-links) - compute the parent-child links between tree nodes.
* [hierarchy.children](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-children) - get or set the accessor function for child nodes.
* [hierarchy.sort](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-sort) - get or set the comparator function for sibling nodes.
* [hierarchy.value](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-value) - get or set the value accessor function.
* [hierarchy.revalue](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-revalue) - recompute the hierarchy values.

**Hierarchy Layout**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Layouts](https://github.com/mbostock/d3/wiki/Layouts) ▸ **Hierarchy Layout**

The hierarchy layout is an abstract layout that is not used directly, but instead allows code sharing between multiple hierarchical layouts. For an implementation, see one of:

* [Cluster](https://github.com/mbostock/d3/wiki/Cluster-Layout) - cluster entities into a dendrogram.
* [Pack](https://github.com/mbostock/d3/wiki/Pack-Layout) - produce a hierarchical layout using recursive circle-packing.
* [Partition](https://github.com/mbostock/d3/wiki/Partition-Layout) - recursively partition a node tree into a sunburst or icicle.
* [Tree](https://github.com/mbostock/d3/wiki/Tree-Layout) - position a tree of nodes tidily.
* [Treemap](https://github.com/mbostock/d3/wiki/Treemap-Layout) - use recursive spatial subdivision to display a tree of nodes.

Although not a hierarchy layout, the [bundle layout](https://github.com/mbostock/d3/wiki/Bundle-Layout) is also designed to work in conjunction with hierarchies.

#### [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-hierarchy) d3.layout.hierarchy()

Creates a new hierarchy layout with the default settings: the default sort order is by descending value; the default value accessor assumes each input data is an object with a numeric value attribute; and the default children accessor assumes each input data is an object with a children array.

#### [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout" \l "wiki-_hierarchy) hierarchy(root)

Runs the hierarchy layout, returning the array of nodes associated with the specified root node. The input argument to the layout is the root node of the hierarchy, and the output return value is an array representing the computed positions of all nodes. Several attributes are populated on each node:

* parent - the parent node, or null for the root.
* children - the array of child nodes, or null for leaf nodes.
* value - the node value, as returned by the value accessor.
* depth - the depth of the node, starting at 0 for the root.

In addition, most hierarchy layouts also compute x and y positions for nodes; see the implementing class for details.

#### [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-links) hierarchy.links(nodes)

Given the specified array of nodes returns an array of objects representing the links from parent to child for each node. Leaf nodes will not have any links. Each link is an object with two attributes:

* source - the parent node (as described above).
* target - the child node.

This method is useful for retrieving a set of link descriptions suitable for display, often in conjunction with the [diagonal](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-diagonal) shape generator. For example:

svg.selectAll("path")

.data(partition.links(nodes))

.enter().append("path")

.attr("d", d3.svg.diagonal());

#### [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-children) hierarchy.children([accessor])

If accessor is specified, sets the specified children accessor function. If accessor is not specified, returns the current children accessor function, which by default assumes that the input data is an object with a children array:

function children(d) {

return d.children;

}

Often, it is convenient to load the node hierarchy using [d3.json](https://github.com/mbostock/d3/wiki/Requests#wiki-d3_json), and represent the input hierarchy as a nested [JSON](http://json.org) object. For example:

{

"name": "flare",

"children": [

{

"name": "analytics",

"children": [

{

"name": "cluster",

"children": [

{"name": "AgglomerativeCluster", "size": 3938},

{"name": "CommunityStructure", "size": 3812},

{"name": "MergeEdge", "size": 743}

]

},

{

"name": "graph",

"children": [

{"name": "BetweennessCentrality", "size": 3534},

{"name": "LinkDistance", "size": 5731}

]

}

]

}

]

}

The children accessor is first invoked for root node in the hierarchy. If the accessor returns null, then the node is assumed to be a leaf node and the layout traversal terminates. Otherwise, the accessor should return an array of data elements representing the child nodes. The accessor is called with arguments node and depth.

#### [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-sort) hierarchy.sort([comparator])

If comparator is specified, sets the sort order of sibling nodes for the layout using the specified comparator function. If comparator is not specified, returns the current group sort order, which defaults to descending order by the associated input data's numeric value attribute:

function comparator(a, b) {

return b.value - a.value;

}

The comparator function is invoked for pairs of nodes, being passed the input data for each node. A null comparator disables sorting and uses tree traversal order. Comparator functions may also be implemented using [d3.ascending](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_ascending) or [d3.descending](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_descending).

#### [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout#wiki-value) hierarchy.value([value])

If value is specified, sets the value accessor to the specified function. If value is not specified, returns the current value accessor. The default accessor assumes that the input data is an object with a numeric value attribute:

function value(d) {

return d.value;

}

The value accessor is invoked for each input data element, and must return a number representing the numeric value of the node. For area-proportional layouts such as treemaps, this value is used to set the area of each node proportionally to the value; for other hierarchical layouts, the value has no effect on the layout.

#### [#](https://github.com/mbostock/d3/wiki/Hierarchy-Layout" \l "wiki-revalue) hierarchy.revalue(root)

Re-evaluates the values of each node in the specified tree starting at root, without re-sorting or recomputing the child nodes. This method can be used to recompute the values of each node without making any structural changes to the hierarchy. Primarily, it exists to support [sticky treemaps](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-sticky).

### [Histogram](https://github.com/mbostock/d3/wiki/Histogram-Layout)

* [d3.layout.histogram](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-histogram) - construct a new default histogram layout.
* [histogram](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-_histogram) - compute the distribution of data using quantized bins.
* [histogram.value](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-value) - get or set the value accessor function.
* [histogram.range](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-range) - get or set the considered value range.
* [histogram.bins](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-bins) - specify how values are organized into bins.
* [histogram.frequency](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-frequency) - compute the distribution as counts or probabilities.

**Histogram Layout**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Layouts](https://github.com/mbostock/d3/wiki/Layouts) ▸ **Histogram Layout**

A **histogram layout** shows the distribution of data by grouping discrete data points into bins. See [bl.ock 3048450](http://bl.ocks.org/mbostock/3048450) for example usage.

#### [#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-histogram) d3.layout.histogram()

Constructs a new histogram function with the default value accessor, range function, and bin function. By default, the histogram function returns frequencies. The returned layout object is both an object and a function. That is: you can call the layout like any other function, and the layout has additional methods that change its behavior. Like other classes in D3, layouts follow the method chaining pattern where setter methods return the layout itself, allowing multiple setters to be invoked in a concise statement.

#### [#](https://github.com/mbostock/d3/wiki/Histogram-Layout" \l "wiki-_histogram) histogram(values[, index])

Evaluates the histogram function on the specified array of values. An optional index may be specified, which is passed along to the range and bin function. The return value is an array of arrays: each element in the outer array represents a bin, and each bin contains the associated elements from the input values. In addition, each bin has three attributes:

* x - the lower bound of the bin (inclusive).
* dx - the width of the bin; x + dx is the upper bound (exclusive).
* y - the count (if [frequency](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-frequency) is true), or the probability (if frequency is false).

Note that the y attribute is the same as the length attribute, in frequency mode.

#### [#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-value) histogram.value([accessor])

Specifies how to extract a value from the associated data; accessor is a function which is invoked on each input value passed to [histogram](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-_histogram), equivalent to calling values.map(accessor) before computing the histogram. The default value function is the built-in [Number](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Number), which is similar to the identity function. If accessor is not specified, returns the current value accessor.

#### [#](https://github.com/mbostock/d3/wiki/Histogram-Layout" \l "wiki-range) histogram.range([range])

Specifies the range of the histogram. Values outside the specified range will be ignored. The range may be specified either as a two-element array representing the minimum and maximum value of the range, or as a function that returns the range given the array of values and the current index passed to [histogram](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-_histogram). The default range is the extent ([minimum](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_min) and [maximum](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_max)) of the values. If range is not specified, returns the current range function.

[#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-bins) histogram.**bins**()   
[#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-bins) histogram.**bins**(*count*)   
[#](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-bins) histogram.**bins**(*thresholds*)   
[#](https://github.com/mbostock/d3/wiki/Histogram-Layout" \l "wiki-bins) histogram.**bins**(*function*)

Specifies how to bin values in the histogram. If no argument is specified, the current binning function is returned, which defaults to an implementation of [Sturges' formula](http://en.wikipedia.org/wiki/Histogram) that divides values into bins using uniformly-spaced values. If a count is specified, the value [range](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-range) is divided evenly into the specified number of bins.

If an array of thresholds is specified, it defines the value thresholds used to bin, starting with the leftmost (lowest) value and ending with rightmost (highest) value. The n + 1 thresholds specify n bins. Any values less than *thresholds[1]* will be placed in the first bin; likewise any values greater than or equal to *thresholds[thresholds.length - 2]* will be placed in the last bin. Thus, although the first and last threshold are not used to assign values to bins, they are still necessary to define the x property of the first bin and the dx property of the last bin, respectively.

Lastly, if a binning function is specified, it is invoked when the layout is passed data, being passed the current [range](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-range), the array of values and the current index passed to [histogram](https://github.com/mbostock/d3/wiki/Histogram-Layout#wiki-_histogram). This function must then return an array of thresholds as described in the previous paragraph.

#### [#](https://github.com/mbostock/d3/wiki/Histogram-Layout" \l "wiki-frequency) histogram.frequency([frequency])

Specifies whether the histogram's y value is a count (frequency) or a probability (density); the default is frequency. If frequency is not specified, returns the current frequency boolean.

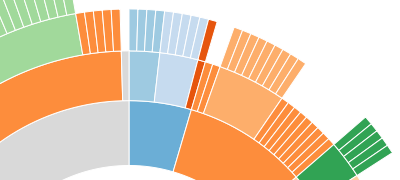
### [Partition](https://github.com/mbostock/d3/wiki/Partition-Layout)

* [d3.layout.partition](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-partition) - recursively partition a node tree into a sunburst or icicle.
* [partition](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-_partition) - alias for partition.nodes.
* [partition.nodes](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-nodes) - compute the partition layout and return the array of nodes.
* [partition.links](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-links) - compute the parent-child links between tree nodes.
* [partition.children](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-children) - get or set the children accessor function.
* [partition.sort](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-sort) - control the order in which sibling nodes are traversed.
* [partition.value](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-value) - get or set the value accessor used to size circles.
* [partition.size](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-size) - specify the layout size in x and y.

**Partition Layout**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Layouts](https://github.com/mbostock/d3/wiki/Layouts) ▸ [Hierarchy](https://github.com/mbostock/d3/wiki/Hierarchy-Layout) ▸ **Partition Layout**

The **partition layout** produces adjacency diagrams: a space-filling variant of a node-link tree diagram. Rather than drawing a link between parent and child in the hierarchy, nodes are drawn as solid areas (either arcs or rectangles), and their placement relative to other nodes reveals their position in the hierarchy. The size of the nodes encodes a quantitative dimension that would be difficult to show in a node-link diagram.



Like other classes in D3, layouts follow the method chaining pattern where setter methods return the layout itself, allowing multiple setters to be invoked in a concise statement.

#### [#](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-partition) d3.layout.partition()

Creates a new partition layout with the default settings: the default sort order is by descending value; the default value accessor assumes each input data is an object with a numeric value attribute; the default children accessor assumes each input data is an object with a children array; the default size is 1×1.

#### [#](https://github.com/mbostock/d3/wiki/Partition-Layout" \l "wiki-_partition) partition(root) [#](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-nodes) partition.nodes(root)

Runs the partition layout, returning the array of nodes associated with the specified root node. The partition layout is part of D3's family of [hierarchical layouts](https://github.com/mbostock/d3/wiki/Hierarchy-Layout). These layouts follow the same basic structure: the input argument to the layout is the root node of the hierarchy, and the output return value is an array representing the computed positions of all nodes. Several attributes are populated on each node:

* parent - the parent node, or null for the root.
* children - the array of child nodes, or null for leaf nodes.
* value - the node value, as returned by the value accessor.
* depth - the depth of the node, starting at 0 for the root.
* x - the minimum x-coordinate of the node position.
* y - the minimum y-coordinate of the node position.
* dx - the x-extent of the node position.
* dy - the y-extent of the node position.

Although the layout has a size in x and y, this represents an arbitrary coordinate system; for example, you can treat x as a radius and y as an angle to produce a radial rather than Cartesian layout. In Cartesian orientation, x, y, dx and dy correspond to the "x", "y", "width" and "height" attributes of the SVG [rect](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-svg_rect) element. In radial orientation, they can be used to compute the innerRadius, startAngle, outerRadius and endAngle of an [arc](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-arc) generator. The Cartesian orientation may be called an **icicle tree**, while the radial orientation is called a **sunburst**.

#### [#](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-links) partition.links(nodes)

Given the specified array of nodes, such as those returned [nodes](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-nodes), returns an array of objects representing the links from parent to child for each node. Leaf nodes will not have any links. Each link is an object with two attributes:

* source - the parent node (as described above).
* target - the child node.

This method is useful for retrieving a set of link descriptions suitable for display, often in conjunction with the [diagonal](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-diagonal) shape generator. For example:

svg.selectAll("path")

.data(partition.links(nodes))

.enter().append("path")

.attr("d", d3.svg.diagonal());

#### [#](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-children) partition.children([children])

If children is specified, sets the specified children accessor function. If children is not specified, returns the current children accessor function, which by default assumes that the input data is an object with a children array:

function children(d) {

return d.children;

}

Often, it is convenient to load the node hierarchy using [d3.json](https://github.com/mbostock/d3/wiki/Requests#wiki-d3_json), and represent the input hierarchy as a nested [JSON](http://json.org) object. For example:

{

"name": "flare",

"children": [

{

"name": "analytics",

"children": [

{

"name": "cluster",

"children": [

{"name": "AgglomerativeCluster", "size": 3938},

{"name": "CommunityStructure", "size": 3812},

{"name": "MergeEdge", "size": 743}

]

},

{

"name": "graph",

"children": [

{"name": "BetweennessCentrality", "size": 3534},

{"name": "LinkDistance", "size": 5731}

]

}

]

}

]

}

The children accessor is first invoked for root node in the hierarchy. If the accessor returns null, then the node is assumed to be a leaf node, and the layout traversal terminates. Otherwise, the accessor should return an array of data elements representing the child nodes.

#### [#](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-sort) partition.sort([comparator])

If comparator is specified, sets the sort order of sibling nodes for the layout using the specified comparator function. If comparator is not specified, returns the current group sort order, which defaults to descending order by the associated input data's numeric value attribute:

function comparator(a, b) {

return b.value - a.value;

}

The comparator function is invoked for pairs of nodes, being passed the input data for each node. A null comparator disables sorting and uses tree traversal order. Comparator functions may also be implemented using [d3.ascending](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_ascending) or [d3.descending](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_descending).

#### [#](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-value) partition.value([value])

If value is specified, sets the value accessor to the specified function. If value is not specified, returns the current value accessor, which assumes that the input data is an object with a numeric value attribute:

function value(d) {

return d.value;

}

The value accessor is invoked for each input data element, and must return a number representing the numeric value of the node. This value is used to set the area of each node proportionally to the value.

#### [#](https://github.com/mbostock/d3/wiki/Partition-Layout#wiki-size) partition.size([size])

If size is specified, sets the available layout size to the specified two-element array of numbers representing x and y. If size is not specified, returns the current size, which defaults to 1×1.

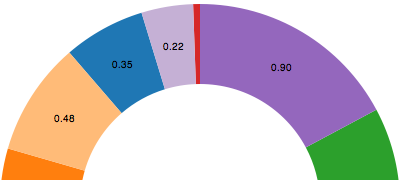
### [Pie](https://github.com/mbostock/d3/wiki/Pie-Layout)

* [d3.layout.pie](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-pie) - construct a new default pie layout.
* [pie](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-_pie) - compute the start and end angles for arcs in a pie or donut chart.
* [pie.value](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-value) - get or set the value accessor function.
* [pie.sort](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-sort) - control the clockwise order of pie slices.
* [pie.startAngle](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-startAngle) - get or set the overall start angle of the pie.
* [pie.endAngle](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-endAngle) - get or set the overall end angle of the pie.

**Pie Layout**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Layouts](https://github.com/mbostock/d3/wiki/Layouts) ▸ **Pie Layout**

The pie layout is a convenience for computing the start and end angles of arcs that comprise a pie or donut chart:



You don't need to use the pie layout to create a pie chart; you can just use the [arc shape](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-arc) directly if you prefer. The pie layout simply makes it easier to convert an array of data into an array of objects with startAngle and endAngle attributes that range from 0 to 2π, which you can then pass to the arc shape generator.

#### [#](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-pie) d3.layout.pie()

Constructs a new pie function with the default value accessor (number), sort comparator (descending value), start angle (0) and end angle (2π). The returned layout object is both an object and a function. That is: you can call the layout like any other function, and the layout has additional methods that change its behavior. Like other classes in D3, layouts follow the method chaining pattern where setter methods return the layout itself, allowing multiple setters to be invoked in a concise statement.

#### [#](https://github.com/mbostock/d3/wiki/Pie-Layout" \l "wiki-_pie) pie(values[, index])

Evaluates the pie function on the specified array of values. An optional index may be specified, which is passed along to the start and end angle functions. The return value is an array of arc descriptors:

* value - the data value, returned by the value accessor.
* startAngle - the start angle of the arc in radians.
* endAngle - the end angle of the arc in radians.
* data - the original datum for this arc.

The elements are returned in the original order, matching values, even if a [sort](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-sort) order is applied; this preserves the original index of each element in the values array, which is nice if you are using the index to generate a categorical color or other display property.

#### [#](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-value) pie.value([accessor])

Specifies how to extract a value from the associated data (e.g. sets the accessor function for the pie layout to use); accessor is a function which is invoked on each input value passed to [pie](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-_pie), equivalent to calling values.map(accessor) before computing the pie layout. The function is passed two arguments: the current datum and the current index. The default value function is the built-in [Number](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Number), which is similar to the identity function. If accessor is not specified, returns the current value accessor.

#### [#](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-sort) pie.sort([comparator])

If comparator is specified, sets the sort order of data for the layout using the specified comparator function. Pass null to disable sorting. If comparator is not specified, returns the current sort order. The sort order defaults to descending value. Sorting preserves the index (and z-index) of input values, affecting only the computed angles. The comparator function is invoked for pairs of data elements from the values array passed to [pie](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-_pie). Comparator functions may also be implemented using [d3.ascending](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_ascending) or [d3.descending](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_descending).

#### [#](https://github.com/mbostock/d3/wiki/Pie-Layout" \l "wiki-startAngle) pie.startAngle([angle])

If angle is specified, sets the overall start angle of the pie layout to the specified value in radians. If angle is not specified, returns the current value, which defaults to 0. The start angle can be specified either as a constant or as a function; if a function, it is evaluated once when the [pie](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-_pie) function is called, being passed the current data and index.

#### [#](https://github.com/mbostock/d3/wiki/Pie-Layout" \l "wiki-endAngle) pie.endAngle([angle])

If angle is specified, sets the overall end angle of the pie layout to the specified value in radians. If angle is not specified, returns the current value, which defaults to 2π. The end angle can be specified either as a constant or as a function; if a function, it is evaluated once when the [pie](https://github.com/mbostock/d3/wiki/Pie-Layout#wiki-_pie) function is called, being passed the current data and index.

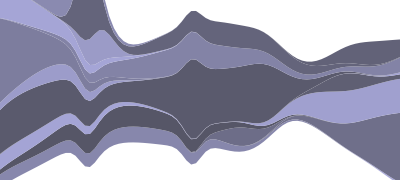
### [Stack](https://github.com/mbostock/d3/wiki/Stack-Layout)

* [d3.layout.stack](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-stack) - construct a new default stack layout.
* [stack](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-_stack) - compute the baseline for each series in a stacked bar or area chart.
* [stack.values](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-values) - get or set the values accessor function per series.
* [stack.order](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-order) - control the order in which series are stacked.
* [stack.offset](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-offset) - specify the overall baseline algorithm.
* [stack.x](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-x) - get or set the x-dimension accessor function.
* [stack.y](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-y) - get or set the y-dimension accessor function.
* [stack.out](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-out) - get or set the output function for storing the baseline.

**Stack Layout**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Layouts](https://github.com/mbostock/d3/wiki/Layouts) ▸ **Stack Layout**

The stack layout takes a two-dimensional array of data and computes a baseline; the baseline is then propagated to the above layers, so as to produce a stacked graph. Several baseline algorithms are supported, along with sorting heuristics to improve perception, as described in [“Stacked Graphs—Geometry & Aesthetics”](http://www.leebyron.com/else/streamgraph/download.php?file=stackedgraphs_byron_wattenberg.pdf) by Byron & Wattenberg.



The stack layout operates in an arbitrary two-dimensional x and y coordinate space, similar to D3's other layouts, including [tree](https://github.com/mbostock/d3/wiki/Tree-Layout). Thus, layers can be stacked vertically, horizontally, or even [radially](http://hint.fm/projects/flickr/). While the "zero" offset is the default, a streamgraph can be generated using the "wiggle" offset, which attempts to minimize change in slope weighted by layer thickness.

#### [#](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-stack) d3.layout.stack()

Constructs a new stack layout with the default offset (zero) and order (null). The returned layout object is both an object and a function. That is: you can call the layout like any other function, and the layout has additional methods that change its behavior. Like other classes in D3, layouts follow the method chaining pattern where setter methods return the layout itself, allowing multiple setters to be invoked in a concise statement.

#### [#](https://github.com/mbostock/d3/wiki/Stack-Layout" \l "wiki-_stack) stack(layers[, index])

Computes the y-coordinate baseline for each series (layer) in layers, and then propagate that baseline to the other layers. In the simplest case, layers is a two-dimensional array of values. All of the 2nd-dimensional arrays must be the same length. The [y](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-y) and [x](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-x) accessors are used to define the y-thickness of each layer at the given x-position, respectively. Thus, by default the following attributes are required on each value:

* x - the x-position of the value.
* y - the y-thickness of the value.
* y0 - the minimum y-position of the value (baseline).

These attributes can be customized by overriding the accessors and the [out](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-out) function.

#### [#](https://github.com/mbostock/d3/wiki/Stack-Layout" \l "wiki-values) stack.values([accessor])

Specifies how to extract values from the associated element in layers; accessor is a function which is invoked on each input layer passed to [stack](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-_stack), equivalent to calling layers.map(accessor) before computing the stack layout. The default values function is the built-in [Object](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object), which is similar to the identity function. If accessor is not specified, returns the current values accessor.

The values accessor can be used to associate additional data per-layer, rather than per-point. For example, say your data were structured as follows:

var layers = [

{

"name": "apples",

"values": [

{ "x": 0, "y": 91},

{ "x": 1, "y": 290}

]

},

{

"name": "oranges",

"values": [

{ "x": 0, "y": 9},

{ "x": 1, "y": 49}

]

}

];

Specify a values accessor that retrieves the points for each layer:

var stack = d3.layout.stack()

.offset("wiggle")

.values(function(d) { return d.values; });

Then, if you wanted to add a tooltip for each layer, you might say:

svg.selectAll("path")

.data(stack(layers))

.enter().append("path")

.attr("d", function(d) { return area(d.values); })

.append("title")

.text(function(d) { return d.name; });

#### [#](https://github.com/mbostock/d3/wiki/Stack-Layout" \l "wiki-offset) stack.offset([offset])

If offset is specified, sets the stack offset algorithm to the specified value. If offset is not specified, returns the current offset algorithm. The following string values are supported:

* silhouette - center the stream, as in [ThemeRiver](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.2977&rep=rep1&type=pdf).
* wiggle - minimize weighted change in slope.
* expand - normalize layers to fill the range [0,1].
* zero - use a zero baseline, i.e., the y-axis.

In addition to a string, offset may be specified as a function. The input to the offset function is the layer data, converted to a standardized representation: a two-dimensional array of values, where each value is represented as a two-element array [x, y]. The return value of the offset function must be an array of values which represents the y-coordinates of the baseline. For example, the default "zero" offset is implemented as:

function offset(data) {

var j = -1,

m = data[0].length,

y0 = [];

while (++j < m) y0[j] = 0;

return y0;

}

#### [#](https://github.com/mbostock/d3/wiki/Stack-Layout" \l "wiki-order) stack.order([order])

If order is specified, sets the stack order to the specified value. If order is not specified, returns the current order. The following string values are supported:

* inside-out - sort by index of maximum value, then use balanced weighting.
* reverse - reverse the input layer order.
* default - use the input layer order.

In addition to a string, order may be specified as a function. The input to the order function is the layer data, converted to the standardized representation: a two-dimensional array of values, where each value is represented as a two-element array [x, y]. The return value of the order function must be an array of indexes which represents the layer order. For example, the default order is implemented as:

function order(data) {

return d3.range(data.length);

}

See also [d3.range](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_range).

#### [#](https://github.com/mbostock/d3/wiki/Stack-Layout" \l "wiki-x) stack.x([accessor])

Specifies how to access the x-coordinate of each value's position. If accessor is specified, sets the accessor to the specified function. If accessor is not specified, returns the current function, which by default assumes that each input value has an x attribute:

function x(d) {

return d.x;

}

The x-accessor is invoked for each input value, for each input layer, being passed the current data (d) and index (i). The return value of the accessor must be a number. Although the x-accessor is invoked for all layers (not just the bottommost layer), the stack layout assumes that the x-coordinates of all layers are consistent. In other words, the stack layout currently requires that the layers be homogenous: each must contain the same number of values at the same x-coordinates. If your data is not so regular, you will need to reinterpolate the data before computing the stack.

#### [#](https://github.com/mbostock/d3/wiki/Stack-Layout" \l "wiki-y) stack.y([accessor])

Specifies how to access the y-coordinate of each value's thickness. If accessor is specified, sets the accessor to the specified function. If accessor is not specified, returns the current function, which by default assumes that each input value has a y attribute:

function y(d) {

return d.y;

}

The y-accessor is invoked for each input value, for each input layer, being passed the current data (d) and index (i). The return value of the accessor must be a number. With the exception of the "expand" offset, the stack layout does not perform any automatic scaling of data. To simplify scaling, use this layout in conjunction with a [linear scale](https://github.com/mbostock/d3/wiki/Quantitative-Scales#wiki-linear) or similar.

#### [#](https://github.com/mbostock/d3/wiki/Stack-Layout" \l "wiki-out) stack.out([setter])

Specifies how to propagate the computed baseline to above layers. If setter is specified, it is used as the output function. If setter is not specified, returns the current output function, which by default assumes that each input value has y and y0 attributes:

function out(d, y0, y) {

d.y0 = y0;

d.y = y;

}

The setter is invoked for each input value, for each input layer, being passed the current data (d), the computed y0 value, and the computed y-thickness. In all cases except the "expand" offset, the y-thickness is the same as the input value returned by [y](https://github.com/mbostock/d3/wiki/Stack-Layout#wiki-y), and thus may be ignored.

### [Tree](https://github.com/mbostock/d3/wiki/Tree-Layout)

* [d3.layout.tree](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-tree) - position a tree of nodes tidily.
* [tree](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-_tree) - alias for tree.nodes.
* [tree.nodes](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-nodes) - compute the tree layout and return the array of nodes.
* [tree.links](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-links) - compute the parent-child links between tree nodes.
* [tree.children](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-children) - get or set the children accessor function.
* [tree.sort](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-sort) - control the order in which sibling nodes are traversed.
* [tree.separation](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-separation) - get or set the spacing function between neighboring nodes.
* [tree.size](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-size) - specify the layout size in x and y.
* [tree.nodeSize](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-nodeSize) - specify a fixed size for each node.

**Tree Layout**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Layouts](https://github.com/mbostock/d3/wiki/Layouts) ▸ [Hierarchy](https://github.com/mbostock/d3/wiki/Hierarchy-Layout) ▸ **Tree Layout**

The **tree** layout produces tidy node-link diagrams of trees using the [Reingold–Tilford “tidy” algorithm](http://emr.cs.iit.edu/%7Ereingold/tidier-drawings.pdf). For example, a tree layout can be used to organize software classes in a package hierarchy:



Like most other layouts, the object returned by d3.layout.tree is both an object and a function. That is: you can call the layout like any other function, and the layout has additional methods that change its behavior. Like other classes in D3, layouts follow the method chaining pattern where setter methods return the layout itself, allowing multiple setters to be invoked in a concise statement.

#### [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-tree) d3.layout.tree()

Creates a new tree layout with the default settings: the default sort order is null; the default children accessor assumes each input data is an object with a children array; the default separation function uses one node width for siblings, and two node widths for non-siblings; the default size is 1×1.

#### [#](https://github.com/mbostock/d3/wiki/Tree-Layout" \l "wiki-_tree) tree(root) [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-nodes) tree.nodes(root)

Runs the tree layout, returning the array of nodes associated with the specified root node. The tree layout is part of D3's family of [hierarchical](https://github.com/mbostock/d3/wiki/Hierarchy-Layout) layouts. These layouts follow the same basic structure: the input argument to the layout is the root node of the hierarchy, and the output return value is an array representing the computed positions of all nodes. Several attributes are populated on each node:

* parent - the parent node, or null for the root.
* children - the array of child nodes, or null for leaf nodes.
* depth - the depth of the node, starting at 0 for the root.
* x - the computed x-coordinate of the node position.
* y - the computed y-coordinate of the node position.

Although the layout has a size in x and y, this represents an arbitrary coordinate system; for example, you can treat x as a radius and y as an angle to produce a radial rather than Cartesian layout.

#### [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-links) tree.links(nodes)

Given the specified array of nodes, such as those returned by [nodes](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-nodes), returns an array of objects representing the links from parent to child for each node. Leaf nodes will not have any links. Each link is an object with two attributes:

* source - the parent node (as described above).
* target - the child node.

This method is useful for retrieving a set of link descriptions suitable for display, often in conjunction with the [diagonal](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-diagonal) shape generator. For example:

svg.selectAll("path")

.data(tree.links(nodes))

.enter().append("path")

.attr("d", d3.svg.diagonal());

#### [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-children) tree.children([children])

If children is specified, sets the specified children accessor function. If children is not specified, returns the current children accessor function, which by default assumes that the input data is an object with a children array:

function children(d) {

return d.children;

}

Often, it is convenient to load the node hierarchy using [d3.json](https://github.com/mbostock/d3/wiki/Requests#wiki-d3_json), and represent the input hierarchy as a nested [JSON](http://json.org) object. For example:

{

"name": "flare",

"children": [

{

"name": "analytics",

"children": [

{

"name": "cluster",

"children": [

{"name": "AgglomerativeCluster", "size": 3938},

{"name": "CommunityStructure", "size": 3812},

{"name": "MergeEdge", "size": 743}

]

},

{

"name": "graph",

"children": [

{"name": "BetweennessCentrality", "size": 3534},

{"name": "LinkDistance", "size": 5731}

]

}

]

}

]

}

The children accessor is first invoked for root node in the hierarchy. If the accessor returns null, then the node is assumed to be a leaf node at the layout traversal terminates. Otherwise, the accessor should return an array of data elements representing the child nodes.

#### [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-separation) tree.separation([separation])

If separation is specified, uses the specified function to compute separation between neighboring nodes. If separation is not specified, returns the current separation function, which defaults to:

function separation(a, b) {

return a.parent == b.parent ? 1 : 2;

}

A variation that is more appropriate for radial layouts reduces the separation gap proportionally to the radius:

function separation(a, b) {

return (a.parent == b.parent ? 1 : 2) / a.depth;

}

The separation function is passed two neighboring nodes a and b, and must return the desired separation between nodes. The nodes are typically siblings, though the nodes may also be cousins (or even more distant relations) if the layout decides to place such nodes adjacent.

#### [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-size) tree.size([size])

If size is specified, sets the available layout size to the specified two-element array of numbers representing x and y. If size is not specified, returns the current size, which defaults to 1×1. The layout size is specified in x and y, but this is not limited screen coordinates and may represent an arbitrary coordinate system. For example, to produce a radial layout where the tree breadth (x) is measured in degrees, and the tree depth (y) is a radius r in pixels, say [360, r].

The size property is exclusive with [tree.nodeSize](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-nodeSize); setting tree.size sets tree.nodeSize to null.

#### [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-nodeSize) tree.nodeSize([nodeSize])

If nodeSize is specified, sets a fixed size for each node as a two-element array of numbers representing x and y. If nodeSize is not specified, returns the current node size, which defaults to null indicating that the layout is determined using the overall [tree.size](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-size) property instead of using a fixed node size. The layout size is specified in x and y, but this is not limited screen coordinates and may represent an arbitrary coordinate system.

The nodeSize property is exclusive with [tree.size](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-size); setting tree.nodeSize sets tree.size to null.

#### [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-sort) tree.sort([comparator])

If comparator is specified, sets the sort order of sibling nodes for the layout using the specified comparator function. If comparator is not specified, returns the current group sort order, which defaults to null for no sorting. The comparator function is invoked for pairs of nodes, being passed the input data for each node. The default comparator is null, which disables sorting and uses tree traversal order. For example, to sort sibling nodes in descending order by the associated input data's numeric value attribute, say:

function comparator(a, b) {

return b.value - a.value;

}

Sorting by the node's name or key is also common. This can be done easily using [d3.ascending](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_ascending) or [d3.descending](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_descending).

#### [#](https://github.com/mbostock/d3/wiki/Tree-Layout#wiki-value) tree.value([value])

If value is specified, sets the value accessor to the specified function. If value is not specified, returns the current value accessor which defaults to null, meaning that the value attribute is not computed. If specified, the value accessor is invoked for each input data element, and must return a number representing the numeric value of the node. This value has no effect on the tree layout, but is generic functionality provided by hierarchy layouts.

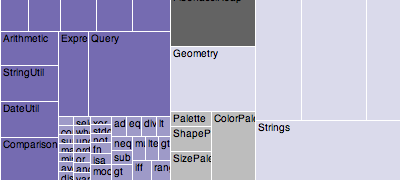
### [Treemap](https://github.com/mbostock/d3/wiki/Treemap-Layout)

* [d3.layout.treemap](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-treemap) - use recursive spatial subdivision to display a tree of nodes.
* [treemap](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-_treemap) - alias for treemap.nodes.
* [treemap.nodes](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-nodes) - compute the treemap layout and return the array of nodes.
* [treemap.links](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-links) - compute the parent-child links between tree nodes.
* [treemap.children](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-children) - get or set the children accessor function.
* [treemap.sort](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-sort) - control the order in which sibling nodes are traversed.
* [treemap.value](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-value) - get or set the value accessor used to size treemap cells.
* [treemap.size](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-size) - specify the layout size in x and y.
* [treemap.padding](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-padding) - specify the padding between a parent and its children.
* [treemap.round](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-round) - enable or disable rounding to exact pixels.
* [treemap.sticky](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-sticky) - make the layout sticky for stable updates.
* [treemap.mode](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-mode) - change the treemap layout algorithm.

**Treemap Layout**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Layouts](https://github.com/mbostock/d3/wiki/Layouts) ▸ [Hierarchy](https://github.com/mbostock/d3/wiki/Hierarchy-Layout) ▸ **Treemap Layout**

Introduced by [Ben Shneiderman](http://www.cs.umd.edu/hcil/treemap-history/) in 1991, a **treemap** recursively subdivides area into rectangles. As with [adjacency diagrams](https://github.com/mbostock/d3/wiki/Partition-Layout), the size of any node in the tree is quickly revealed. “Squarified” treemaps use approximately-square rectangles, which offer better readability and size estimation than naïve “slice-and-dice” subdivision. Fancier algorithms such as [Voronoi](http://portal.acm.org/citation.cfm?id=1056018.1056041) and [jigsaw](http://hint.fm/papers/158-wattenberg-final3.pdf) treemaps also exist but are less common.



Like other classes in D3, layouts follow the method chaining pattern where setter methods return the layout itself, allowing multiple setters to be invoked in a concise statement.

#### [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-treemap) d3.layout.treemap()

Creates a new treemap layout with the default settings: the default sort order is by descending value; the default value accessor assumes each input data is an object with a numeric value attribute; the default children accessor assumes each input data is an object with a children array; the default size is 1×1.

#### [#](https://github.com/mbostock/d3/wiki/Treemap-Layout" \l "wiki-_treemap) treemap(root) [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-nodes) treemap.nodes(root)

Runs the treemap layout, returning the array of nodes associated with the specified root node. The treemap layout is part of D3's family of [hierarchical layouts](https://github.com/mbostock/d3/wiki/Hierarchy-Layout). These layouts follow the same basic structure: the input argument to the layout is the root node of the hierarchy, and the output return value is an array representing the computed positions of all nodes. Several attributes are populated on each node:

* parent - the parent node, or null for the root.
* children - the array of child nodes, or null for leaf nodes.
* value - the node value, as returned by the value accessor.
* depth - the depth of the node, starting at 0 for the root.
* x - the minimum x-coordinate of the node position.
* y - the minimum y-coordinate of the node position.
* dx - the x-extent of the node position.
* dy - the y-extent of the node position.

Note that this will modify the nodes that you pass in!

Although the layout has a size in x and y, this represents an arbitrary coordinate system; for example, you can treat x as a radius and y as an angle to produce a radial rather than Cartesian layout. In Cartesian orientation, x, y, dx and dy correspond to the "x", "y", "width" and "height" attributes of the SVG [rect](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-svg_rect) element.

#### [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-links) treemap.links(nodes)

Given the specified array of nodes, such as those returned by [nodes](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-nodes), returns an array of objects representing the links from parent to child for each node. Leaf nodes will not have any links. Each link is an object with two attributes:

* source - the parent node (as described above).
* target - the child node.

This method is useful for retrieving a set of link descriptions suitable for display, often in conjunction with the [diagonal](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-diagonal) shape generator. For example:

svg.selectAll("path")

.data(partition.links(nodes))

.enter().append("path")

.attr("d", d3.svg.diagonal());

#### [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-children) treemap.children([children])

If children is specified, sets the specified children accessor function. If children is not specified, returns the current children accessor function, which by default assumes that the input data is an object with a children array:

function children(d) {

return d.children;

}

Often, it is convenient to load the node hierarchy using [d3.json](https://github.com/mbostock/d3/wiki/Requests#wiki-d3_json), and represent the input hierarchy as a nested [JSON](http://json.org) object. For example:

{

"name": "flare",

"children": [

{

"name": "analytics",

"children": [

{

"name": "cluster",

"children": [

{"name": "AgglomerativeCluster", "size": 3938},

{"name": "CommunityStructure", "size": 3812},

{"name": "MergeEdge", "size": 743}

]

},

{

"name": "graph",

"children": [

{"name": "BetweennessCentrality", "size": 3534},

{"name": "LinkDistance", "size": 5731}

]

}

]

}

]

}

The children accessor is first invoked for root node in the hierarchy. If the accessor returns null, then the node is assumed to be a leaf node at the layout traversal terminates. Otherwise, the accessor should return an array of data elements representing the child nodes.

#### [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-sort) treemap.sort([comparator])

If comparator is specified, sets the sort order of sibling nodes for the layout using the specified comparator function. If comparator is not specified, returns the current group sort order, which defaults to descending order by the associated input data's numeric value attribute:

function comparator(a, b) {

return b.value - a.value;

}

The comparator function is invoked for pairs of nodes, being passed the input data for each node. A null comparator disables sorting and uses tree traversal order. Comparator functions may also be implemented using [d3.ascending](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_ascending) or [d3.descending](https://github.com/mbostock/d3/wiki/Arrays#wiki-d3_descending).

#### [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-value) treemap.value([value])

If value is specified, sets the value accessor to the specified function. If value is not specified, returns the current value accessor, which assumes that the input data is an object with a numeric value attribute:

function value(d) {

return d.value;

}

The value accessor is invoked for each input data element, and must return a number representing the numeric value of the node. This value is used to set the area of each node proportionally to the value.

#### [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-size) treemap.size([size])

If size is specified, sets the available layout size to the specified two-element array of numbers representing x and y. If size is not specified, returns the current size, which defaults to 1×1.

#### [#](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-padding) treemap.padding([padding])

Get or set the padding for each treemap cell, in pixels. The padding determines the amount of extra space to reserve between the parent and its children; this space can be used to indicate the hierarchy through enclosure, or to reserve space for parent labels. If no padding is used, then the leaves of the tree will completely fill the layout's size.

If padding is specified, sets the new padding and returns the treemap layout; if padding is not specified, returns the current padding. The padding may be specified several ways:

* A null value disables padding; null is equivalent to zero.
* A number indicates uniform padding, in pixels, on all four sides.
* An array of numbers indicates the top, right, bottom and left padding values.

The padding may also be specified as a function which returns one of the three above values. This function is evaluated for each internal (non-leaf) node, and can be used to compute padding dynamically.

#### [#](https://github.com/mbostock/d3/wiki/Treemap-Layout" \l "wiki-round) treemap.round([round])

If round is specified, sets whether or not the treemap layout will round to exact pixel boundaries. This can be nice to avoid antialiasing artifacts in SVG. If round is not specified, returns whether the treemap will be rounded.

#### [#](https://github.com/mbostock/d3/wiki/Treemap-Layout" \l "wiki-sticky) treemap.sticky([sticky])

If sticky is specified, sets whether or not the treemap layout is "sticky": a sticky treemap layout will preserve the relative arrangement of nodes across transitions. The allocation of nodes into squarified horizontal and vertical rows is persisted across updates by storing a z attribute on the last element in each row; this allows nodes to be resized smoothly, without shuffling or occlusion that would impede perception of changing values. Note, however, that this results in a suboptimal layout for one of the two states. If sticky is not specified, returns whether the treemap layout is sticky.

Implementation note: sticky treemaps cache the array of nodes internally; therefore, it is not possible to reuse the same layout instance on multiple datasets. To reset the cached state when switching datasets with a sticky layout, call sticky(true) again. Since version [1.25.0](https://github.com/mbostock/d3/tree/v1.25.0), hierarchy layouts no longer copy the input data by default on each invocation, so it may be possible to eliminate caching and make the layout fully stateless.

#### [#](https://github.com/mbostock/d3/wiki/Treemap-Layout" \l "wiki-mode) treemap.mode([mode])

If mode is specified, sets the layout algorithm. If mode is not specified, returns the current layout algorithm, which defaults to "squarify". The following modes are supported:

* squarify - rectangular subdivision; squareness controlled via the target [ratio](https://github.com/mbostock/d3/wiki/Treemap-Layout#wiki-ratio).
* slice - horizontal subdivision.
* dice - vertical subdivision.
* slice-dice - alternating between horizontal and vertical subdivision.

## [d3.geo (Geography)](https://github.com/mbostock/d3/wiki/Geo)

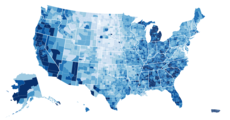
### [Paths](https://github.com/mbostock/d3/wiki/Geo-Paths)

* [d3.geo.path](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-path) - create a new geographic path generator.
* [path](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-_path) - project the specified feature and render it to the context.
* [path.projection](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-path_projection) - get or set the geographic projection.
* [path.context](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-path_context) - get or set the render context.
* [path.pointRadius](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-path_pointRadius) - get or set the radius to display point features.
* [path.area](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-path_area) - compute the projected area of a given feature.
* [path.centroid](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-path_centroid) - compute the projected centroid of a given feature.
* [path.bounds](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-path_bounds) - compute the projected bounds of a given feature.
* [d3.geo.graticule](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-graticule) - create a graticule generator.
* [graticule](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-_graticule) - generate a MultiLineString of meridians and parallels.
* [graticule.lines](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-graticule_lines) - generate an array of LineStrings of meridians and parallels.
* [graticule.outline](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-graticule_outline) - generate a Polygon of the graticule’s extent.
* [graticule.extent](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-graticule_extent) - get or set the major & minor extents.
* [graticule.majorExtent](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-graticule_majorExtent) - get or set the major extent.
* [graticule.minorExtent](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-graticule_minorExtent) - get or set the minor extent.
* [graticule.step](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-graticule_step) - get or set the major & minor step intervals.
* [graticule.majorStep](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-graticule_majorStep) - get or set the major step intervals.
* [graticule.minorStep](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-graticule_minorStep) - get or set the minor step intervals.
* [graticule.precision](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-graticule_precision) - get or set the latitudinal precision.
* [d3.geo.circle](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-circle) - create a circle generator.
* [circle](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-_circle) - generate a piecewise circle as a Polygon.
* [circle.origin](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-circle_origin) - specify the origin in latitude and longitude.
* [circle.angle](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-circle_angle) - specify the angular radius in degrees.
* [circle.precision](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-circle_precision) - specify the precision of the piecewise circle.
* [d3.geo.area](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-area) - compute the spherical area of a given feature.
* [d3.geo.bounds](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-bounds) - compute the latitude-longitude bounding box for a given feature.
* [d3.geo.centroid](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-centroid) - compute the spherical centroid of a given feature.
* [d3.geo.distance](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-distance) - compute the great-arc distance between two points.
* [d3.geo.interpolate](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-interpolate) - interpolate between two points along a great arc.
* [d3.geo.length](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-length) - compute the length of a line string or the perimeter of a polygon.
* [d3.geo.rotation](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-rotation) - create a rotation function for the specified angles [λ, φ, γ].
* [rotation](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-_rotation) - rotate the given location around the sphere.
* [rotation.invert](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-rotation_invert) - inverse-rotate the given location around the sphere.

**Geo Paths**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Geo](https://github.com/mbostock/d3/wiki/Geo) ▸ **Geo Paths**

For cartographic visualizations, D3 supports a handful of components for displaying and manipulating **geographic data**. These components use the [GeoJSON format](http://geojson.org/geojson-spec.html)—a standard way of representing geographic features in JavaScript. (See also the [TopoJSON format](https://github.com/mbostock/topojson), an extension of GeoJSON that is significantly more compact.) To convert shapefiles to GeoJSON, use ogr2ogr, part of the [GDAL package](http://www.gdal.org/).



Some other tools you may be interested in:

* [TopoJSON](https://github.com/mbostock/topojson) - shapefile simplification, topology construction and GeoJSON compression.
* [Shapely](http://trac.gispython.org/lab/wiki/Shapely) - manipulation of planar geometry objects.
* [ColorBrewer](http://colorbrewer2.org) - color scales for maps.
* [PostGIS](http://postgis.refractions.net/) - a geospatial database.

The primary mechanism for displaying geographic data is [d3.geo.path](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-path). This class is similar to [d3.svg.line](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-line) and the other SVG shape generators: given a geometry or feature object, it generates the path data string suitable for the "d" attribute of an SVG path element. The d3.geo.path class can [render directly to Canvas](http://bl.ocks.org/mbostock/3783604), which may offer better performance when animating the projection.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-path) d3.geo.path()

Creates a new geographic path generator with the default settings: the [albersUsa](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-albersUsa) projection and a point radius of 4.5 pixels.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-_path) path(feature[, index])

Returns the path data string for the given feature, which may be any GeoJSON feature or geometry object:

* Point - a single position.
* MultiPoint - an array of positions.
* LineString - an array of positions forming a continuous line.
* MultiLineString - an array of arrays of positions forming several lines.
* Polygon - an array of arrays of positions forming a polygon (possibly with holes).
* MultiPolygon - a multidimensional array of positions forming multiple polygons.
* GeometryCollection - an array of geometry objects.
* Feature - a feature containing one of the above geometry objects.
* FeatureCollection - an array of feature objects.

The type "Sphere" is also supported, which is useful for rendering the outline of the globe. A sphere has no coordinates. An optional index may be specified, which is passed along to the [pointRadius](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-pointRadius) accessor; the index is passed automatically when the path generator is invoked by [selection.attr](https://github.com/mbostock/d3/wiki/Selections#wiki-attr).

**Important:** the inside of a polygon is all points that the polygon winds around in a clockwise order. If your GeoJSON input has polygons in the wrong winding order, you must reverse them, say via [ST\_ForceRHR](http://www.postgis.org/docs/ST_ForceRHR.html); you can also convert your GeoJSON to [TopoJSON](https://github.com/mbostock/topojson), and this will happen automatically.

To display multiple features, you can place them in a single feature collection and a single path element:

svg.append("path")

.datum({type: "FeatureCollection", features: features})

.attr("d", d3.geo.path());

Alternatively, you can create multiple distinct path elements:

svg.selectAll("path")

.data(features)

.enter().append("path")

.attr("d", d3.geo.path());

Using distinct path elements is typically slower than a single path element for a collection. However, distinct path elements are preferred if you want interact with features separately (e.g., using CSS :hover or click events).

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-path_projection) path.projection([projection])

If projection is specified, sets the projection used by the path generator to the specified projection function. If projection is not specified, returns the current projection, which defaults to [albersUsa](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-albersUsa). The projection is typically one of D3's built-in [geographic projections](https://github.com/mbostock/d3/wiki/Geo-Projections); however, any function can be used. A projection function takes a two-element array of numbers representing the coordinates of a location, [*longitude*, *latitude*], and returns a similar two-element array of numbers representing the projected pixel position [*x*, *y*]. For example, a rudimentary spherical Mercator projection:

function mercator(coordinates) {

return [

coordinates[0] / 360,

(-180 / Math.PI \* Math.log(Math.tan(Math.PI / 4 + coordinates[1] \* Math.PI / 360))) / 360

];

}

Internally, this point projection function is wrapped with a fallback [stream transformation](https://github.com/mbostock/d3/wiki/Geo-Streams) that performs [adaptive resampling](http://bl.ocks.org/mbostock/3795544). However, the fallback stream does not perform any clipping or cutting.

For more control over the stream transformation, the projection may be specified as an object that implements the stream method. ([See example.](http://bl.ocks.org/mbostock/5663666)) The stream method takes an output stream as input, and returns a wrapped stream that projects the input geometry; in other words, it implements [projection.stream](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-stream).

If projection is null, the path uses the identity transformation, where the input geometry is not projected and is instead rendered directly in raw coordinates. This can be useful for fast rendering of already-projected geometry, or for fast rendering of the equirectangular projection.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-path_context) path.context([context])

If context is specified, sets the render context and returns the path generator. If the context is null, then the path generator will return an SVG path string when [invoked](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-_path) on a given feature. If the context is non-null, the path generator will instead call methods on the specified context to render geometry. The context must implement the following methods:

* beginPath()
* moveTo(x, y)
* lineTo(x, y)
* arc(x, y, radius, startAngle, endAngle)
* closePath()

Note that this is a subset of the canvas element’s [2D rendering context](http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#2dcontext), and thus a canvas context can be passed to the path generator, in which case geometry will be rendered [directly to the canvas](http://bl.ocks.org/mbostock/3783604). If context is not specified, returns the current render context, which defaults to null.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-path_area) path.area(feature)

Computes the projected area (in square pixels) for the specified feature. Point, MultiPoint, LineString and MultiLineString features have zero area. For Polygon and MultiPolygon features, this method first computes the area of the exterior ring, and then subtracts the area of any interior holes. This method observes any clipping and resampling performed by the projection stream.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-path_centroid) path.centroid(feature)

Computes the projected centroid (in pixels) for the specified feature. This is handy for, say, labeling state or county boundaries, or displaying a symbol map. The [noncontiguous cartogram](http://mbostock.github.com/d3/ex/cartogram.html) example scales each state around its centroid. This method observes any clipping and resampling performed by the projection stream.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-path_bounds) path.bounds(feature)

Computes the projected bounding box (in pixels) for the specified feature. This is handy for, say, zooming in to a particular feature. This method observes any clipping and resampling performed by the projection stream.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-path_pointRadius) path.pointRadius([radius])

If radius is specified, sets the radius used to display Point and MultiPoint features to the specified number. If radius is not specified, returns the current radius. While the radius is commonly specified as a number constant, it may also be specified as a function which is computed per feature, being passed the feature and index arguments from the [path](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-_path) function. For example, if your GeoJSON data has additional properties, you might access those properties inside the radius function to vary the point size; alternatively, you could [d3.svg.symbol](https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-symbol) and a [projection](https://github.com/mbostock/d3/wiki/Geo-Projections) for more control over the display.

**Shape Generators**

Note: to generate a great arc in D3, simply pass a LineString-type geometry object to d3.geo.path. D3’s projections use great-arc interpolation for intermediate points (with [adaptive resampling](http://bl.ocks.org/mbostock/3795544)), so there’s no need to use a shape generator to create great arcs.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-graticule) d3.geo.graticule

Constructs a feature generator for creating graticules.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-_graticule) graticule()

Returns a MultiLineString geometry object representing all meridians and parallels for this graticule.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-graticule_lines) graticule.lines()

Returns an array of LineString geometry objects, one for each meridian or parallel for this graticule.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-graticule_outline) graticule.outline()

Returns a Polygon geometry object representing the outline of this graticule, i.e. along the meridians and parallels defining its extent.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-graticule_extent) graticule.extent(extent)

If extent is specified, sets the major and minor extents of this graticule. If extent is not specified, returns the current minor extent, which defaults to ⟨⟨-180°, -80° - ε⟩, ⟨180°, 80° + ε⟩⟩.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-graticule_majorExtent) graticule.majorExtent(extent)

If extent is specified, sets the major extent of this graticule. If extent is not specified, returns the current major extent, which defaults to ⟨⟨-180°, -90° + ε⟩, ⟨180°, 90° - ε⟩⟩.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-graticule_minorExtent) graticule.minorExtent(extent)

If extent is specified, sets the minor extent of this graticule. If extent is not specified, returns the current minor extent, which defaults to ⟨⟨-180°, -80° + ε⟩, ⟨180°, 80° - ε⟩⟩.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-graticule_step) graticule.step(step)

If step is specified, sets the major and minor step for this graticule. If step is not specified, returns the current minor step, which defaults to ⟨10°, 10°⟩.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-graticule_majorStep) graticule.majorStep(step)

If step is specified, sets the major step for this graticule. If step is not specified, returns the current major step, which defaults to ⟨90°, 360°⟩.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-graticule_minorStep) graticule.minorStep(step)

If step is specified, sets the minor step for this graticule. If step is not specified, returns the current minor step, which defaults to ⟨10°, 10°⟩.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-graticule_precision) graticule.precision(precision)

If precision is specified, sets the precision for this graticule, in degrees. If precision is not specified, returns the current precision, which defaults to 2.5°.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-circle) d3.geo.circle

Constructs a feature generator for creating circles centered at a given geographic location with a given radius in degrees.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-_circle) circle(arguments…)

Returns a GeoJSON Polygon approximating a circle. The origin accessor specifies how to determine the origin for the given arguments; the default accessor uses the constant ⟨0°,0°⟩.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-circle_origin) circle.origin([origin])

If origin is specified, sets the circle origin. A two-element coordinate array should be specified, or an accessor function. If origin is not specified, returns the current origin, which defaults to ⟨0°,0°⟩.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-circle_angle) circle.angle([angle])

If angle is specified, sets the angular radius of the circle in degrees. If angle is not specified, returns the current radius, which defaults to 90°.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-circle_precision) circle.precision([precision])

If precision is specified, sets the precision of the interpolated circle segments in degrees. These interpolated segments are inserted when a feature is clipped by the circle. If precision is not specified, returns the current precision, which defaults to 6°.

**Spherical Math**

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-area) d3.geo.area(feature)

Returns the spherical area of the specified feature in [steradians](http://mathworld.wolfram.com/Steradian.html). See also [path.area](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-path_area), which computes the projected area on the Cartesian plane.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-centroid) d3.geo.centroid(feature)

Returns the spherical centroid of the specified feature. See also [path.centroid](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-path_centroid), which computes the projected centroid on the Cartesian plane.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-bounds) d3.geo.bounds(feature)

Returns the spherical bounding box for the specified feature. The bounding box is represented by a two-dimensional array: [​[*left*, *bottom*], [*right*, *top*]​], where left is the minimum longitude, bottom is the minimum latitude, right is maximum longitude, and top is the maximum latitude. See also [path.bounds](https://github.com/mbostock/d3/wiki/Geo-Paths#wiki-path_bounds), which computes the projected bounding box on the Cartesian plane.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-distance) d3.geo.distance(a, b)

Returns the great-arc distance in radians between the two points *a* and *b*. Each point is specified as an array [*longitude*, *latitude*], with coordinates expressed in degrees.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-length) d3.geo.length(feature)

Returns the great-arc length of the specified feature in [radians](http://mathworld.wolfram.com/Radian.html). For polygons, returns the perimeter of the exterior ring plus that of any interior rings.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-interpolate) d3.geo.interpolate(a, b)

Returns an interpolator given the two locations a and b. Each location must be represented as a two-element array of [*longitude*, *latitude*]. The returned interpolator is a function which takes a single parameter t as input, where t ranges from 0 to 1. A value of 0 returns the location a, while a value of 1 returns the location b. Intermediate values interpolate from a to b along the spanning great arc.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-rotation) d3.geo.rotation(rotate)

Specifies a rotation in the form of an array, [λ, φ, γ]. The elements of the array are angles in degrees, and specify a rotation in the following order: longitudinal, latitudinal and about the origin. If the last element of the array, γ, is omitted, this defaults to 0. Returns a function, which rotates a given location as described below.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-_rotation) rotation(location)

Rotates a given location according to the angles specified for this rotation, in the order described above. A location is specified as an array [*longitude*, *latitude*], with coordinates expressed in degrees. Returns a new array representing the rotated location.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Paths" \l "wiki-rotation_invert) rotation.invert(location)

Rotates a given location according to the angles specified for this rotation, but with the order described above reversed. A location is specified as an array [*longitude*, *latitude*], with coordinates expressed in degrees. Returns a new array representing the rotated location.

### [Projections](https://github.com/mbostock/d3/wiki/Geo-Projections)

* [d3.geo.projection](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-projection) - create a standard projection from a raw projection.
* [projection](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-_projection) - project the specified location.
* [projection.invert](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-invert) - invert the projection for the specified point.
* [projection.rotate](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-rotate) - get or set the projection’s three-axis rotation.
* [projection.center](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-center) - get or set the projection’s center location.
* [projection.translate](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-translate) - get or set the projection’s translation position.
* [projection.scale](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-scale) - get or set the projection’s scale factor.
* [projection.clipAngle](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-clipAngle) - get or set the radius of the projection’s clip circle.
* [projection.clipExtent](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-clipExtent) - get or set the projection’s viewport clip extent, in pixels.
* [projection.precision](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-precision) - get or set the precision threshold for adaptive resampling.
* [projection.stream](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-stream) - wrap the specified stream listener, projecting input geometry.
* [d3.geo.projectionMutator](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-projectionMutator) - create a standard projection from a mutable raw projection.
* [d3.geo.albers](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-albers) - the Albers equal-area conic projection.
* [albers.parallels](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-albers_parallels) - get or set the projection's two standard parallels.
* [d3.geo.albersUsa](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-albersUsa) - a composite Albers projection for the United States.
* [d3.geo.azimuthalEqualArea](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-azimuthalEqualArea) - the azimuthal equal-area projection.
* [d3.geo.azimuthalEquidistant](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-azimuthalEquidistant) - the azimuthal equidistant projection.
* [d3.geo.conicConformal](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-conicConformal) - the conic conformal projection.
* [d3.geo.conicEquidistant](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-conicEquidistant) - the conic equidistant projection.
* [d3.geo.conicEqualArea](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-conicEqualArea) the conic equal-area (a.k.a. Albers) projection.
* [d3.geo.equirectangular](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-equirectangular) - the equirectangular (plate carreé) projection.
* [d3.geo.gnomonic](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-gnomonic) - the gnomonic projection.
* [d3.geo.mercator](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-mercator) - the spherical Mercator projection.
* [d3.geo.orthographic](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-orthographic) - the azimuthal orthographic projection.
* [d3.geo.stereographic](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-stereographic) - the azimuthal stereographic projection.
* [d3.geo.azimuthalEqualArea.raw](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-azimuthalEqualArea_raw) - the raw azimuthal equal-area projection.
* [d3.geo.azimuthalEquidistant.raw](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-azimuthalEquidistant_raw) - the azimuthal equidistant projection.
* [d3.geo.conicConformal.raw](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-conicConformal_raw) - the raw conic conformal projection.
* [d3.geo.conicEquidistant.raw](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-conicEquidistant_raw) - the raw conic equidistant projection.
* [d3.geo.conicEqualArea.raw](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-conicEqualArea_raw) the raw conic equal-area (a.k.a. Albers) projection.
* [d3.geo.equirectangular.raw](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-equirectangular_raw) - the raw equirectangular (plate carrée) projection.
* [d3.geo.gnomonic.raw](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-gnomonic_raw) - the raw gnomonic projection.
* [d3.geo.mercator.raw](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-mercator_raw) - the raw Mercator projection.
* [d3.geo.orthographic.raw](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-orthographic_raw) - the raw azimuthal orthographic projection.
* [d3.geo.stereographic.raw](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-stereographic_raw) - the raw azimuthal stereographic projection.
* [d3.geo.transverseMercator.raw](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-transverseMercator_raw) - the raw transverse Mercator projection.

**Geo Projections**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Geo](https://github.com/mbostock/d3/wiki/Geo) ▸ **Geo Projections**

Several common projections are included with default build of D3; these are shown below. Numerous less-common projections are available in the [extended geographic projections plugin](https://github.com/d3/d3-geo-projection/) and the [polyhedral projection plugin](https://github.com/d3/d3-plugins/tree/master/geo/polyhedron).

|  |  |  |  |
| --- | --- | --- | --- |
| d3.geo.albersUsa | d3.geo.azimuthalEqualArea | d3.geo.azimuthalEquidistant | d3.geo.conicEqualArea |
| d3.geo.conicConformal | d3.geo.conicEquidistant | d3.geo.equirectangular | d3.geo.gnomonic |
| d3.geo.mercator | d3.geo.orthographic | d3.geo.stereographic | d3.geo.transverseMercator |

**Standard Abstract Projection**

Most projections provided by D3 are created via d3.geo.projection and are configurable: you can rotate the globe, scale or transform the canvas, etc. Unless you’re implementing a new raw projection, you probably won’t use the d3.geo.projection constructor, but you are likely to use the configuration methods.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-projection) d3.geo.projection(raw)

Constructs a new projection from the specified raw point projection function. For example, a Mercator projection can be implemented as:

var mercator = d3.geo.projection(function(λ, φ) {

return [

λ,

Math.log(Math.tan(π / 4 + φ / 2))

];

});

(See [src/geo/mercator.js](https://github.com/mbostock/d3/blob/master/src/geo/mercator.js) for the full implementation.) If the raw function supports an invert method, then the returned projection will expose a corresponding [invert](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-invert) method.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-_projection) projection(location)

Projects forward from spherical coordinates (in degrees) to Cartesian coordinates (in pixels). Returns an array [*x*, *y*] given the input array [*longitude*, *latitude*]. May return null if the specified location has no defined projected position, such as when the location is outside the clipping bounds of the projection.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-invert) projection.invert(point)

Projects backward from Cartesian coordinates (in pixels) to spherical coordinates (in degrees). Returns an array [*longitude*, *latitude*] given the input array [*x*, *y*]. Not all projections implement invert; for noninvertible projections, this method is undefined.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-rotate) projection.rotate([rotation])

If rotation is specified, sets the projection’s [three-axis rotation](http://bl.ocks.org/mbostock/4282586) to the specified angles λ, φ and γ ([yaw, pitch and roll](http://en.wikipedia.org/wiki/Aircraft_principal_axes), or equivalently [longitude, latitude](http://en.wikipedia.org/wiki/Geographic_coordinate_system) and roll) in degrees and returns the projection. If rotation is not specified, returns the current rotation which defaults [0, 0, 0]. If the specified rotation has only two values, rather than three, the roll is assumed to be 0°.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-center) projection.center([location])

If center is specified, sets the projection’s center to the specified location, a two-element array of longitude and latitude in degrees and returns the projection. If center is not specified, returns the current center which defaults to ⟨0°,0°⟩.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-translate) projection.translate([point])

If point is specified, sets the projection’s translation offset to the specified two-element array [*x*, *y*] and returns the projection. If point is not specified, returns the current translation offset which defaults to [480, 250]. The translation offset determines the pixel coordinates of the projection’s [center](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-center). The default translation offset places ⟨0°,0°⟩ at the center of a 960×500 area.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-scale) projection.scale([scale])

If scale is specified, sets the projection’s scale factor to the specified value and returns the projection. If scale is not specified, returns the current scale factor which defaults to 150. The scale factor corresponds linearly to the distance between projected points. However, scale factors are not consistent across projections.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-clipAngle) projection.clipAngle(angle)

If angle is specified, sets the projection’s clipping circle radius to the specified angle in degrees and returns the projection. If angle is null, switches to [antimeridian cutting](http://bl.ocks.org/mbostock/3788999) rather than small-circle clipping. If angle is not specified, returns the current clip angle which defaults to null. Small-circle clipping is independent of viewport clipping via [clipExtent](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-clipExtent).

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-clipExtent) projection.clipExtent(extent)

If extent is specified, sets the projection’s viewport clip extent to the specified bounds in pixels and returns the projection. The extent bounds are specified as an array [​[*x0*, *y0*], [*x1*, *y1*]​], where *x0* is the left-side of the viewport, *y0* is the top, *x1* is the right and *y1* is the bottom. If extent is null, no viewport clipping is performed. If extent is not specified, returns the current viewport clip extent which defaults to null. Viewport clipping is independent of small-circle clipping via [clipAngle](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-clipAngle).

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-precision) projection.precision(precision)

If precision is specified, sets the threshold for the projection’s [adaptive resampling](http://bl.ocks.org/mbostock/3795544) to the specified value in pixels and returns the projection. This value corresponds to the [Douglas–Peucker](http://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm) distance. If precision is not specified, returns the projection’s current resampling precision which defaults to Math.SQRT(1/2).

A precision of 0 disables adaptive resampling.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-stream) projection.stream(listener)

Returns a projecting [stream](https://github.com/mbostock/d3/wiki/Geo-Streams) wrapper for the specified listener. Any geometry streamed to the wrapper is projected before being streamed to the wrapped listener. A typical projection involves several stream transformations: the input geometry is first converted to radians, rotated on three axes, clipped to the small circle or cut along the antimeridian, and lastly projected to the Cartesian plane with adaptive resampling, scale and translation.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-projectionMutator) d3.geo.projectionMutator(rawFactory)

Constructs a new projection from the specified raw point projection function factory. This function does not return the projection directly, but instead returns a mutate method that you can call whenever the raw projection function changes. For example, say you’re implementing the Albers equal-area conic projection, which requires configuring the projections two parallels. Using closures, you can implement the raw projection as follows:

// φ0 and φ1 are the two parallels

function albersRaw(φ0, φ1) {

return function(λ, φ) {

return [

/\* compute x here \*/,

/\* compute y here \*/

];

};

}

Using d3.geo.projectionMutator, you can implement a standard projection that allows the parallels to be changed, reassigning the raw projection used internally by d3.geo.projection:

function albers() {

var φ0 = 29.5,

φ1 = 45.5,

mutate = d3.geo.projectionMutator(albersRaw),

projection = mutate(φ0, φ1);

projection.parallels = function(\_) {

if (!arguments.length) return [φ0, φ1];

return mutate(φ0 = +\_[0], φ1 = +\_[1]);

};

return projection;

}

Thus, when creating a mutable projection, the mutate function is never exposed, but can be used to recreate the underlying raw projection easily. For the full implementation, see [src/geo/albers.js](https://github.com/mbostock/d3/blob/master/src/geo/albers.js).

**Standard Projections**

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-albers) d3.geo.albers()

An alias for [d3.geo.conicEqualArea](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-conicEqualArea), with USA-centric defaults: scale 1000, translate [480, 250], rotation [96°, 0°], center ⟨-0.6°, 38.7°⟩ and parallels [29.5°, 45.5°], making it suitable for displaying the United States, centered around [Hutchinson, Kansas](https://maps.google.com/maps?q=Hutchinson,+Kansas&z=5) in a 960×500 area. The central meridian and parallels are specified by the [USGS](http://www.usgs.gov/) in the 1970 [National Atlas](http://www.nationalatlas.gov/).

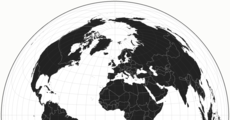
#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-albersUsa) d3.geo.albersUsa()



The Albers USA projection is a composite projection of four Albers projections designed to display the forty-eight lower United States alongside Alaska and Hawaii. Although intended for choropleths, it scales the area of Alaska by a factor of 0.35x (a *lie factor* of 3); Hawaii is shown at the same scale as the lower forty-eight.

The Albers USA projection does not support rotation or centering.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-azimuthalEqualArea) d3.geo.azimuthalEqualArea()



The azimuthal equal-area projection is also suitable for choropleths. A [polar aspect](http://bl.ocks.org/mbostock/4364903) of this projection is used for the United Nations logo.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-azimuthalEquidistant) d3.geo.azimuthalEquidistant()



The azimuthal equidistant projection preserves distances from the projection’s center: the distance from any projected point to the projection’s center is proportional to the great arc distance. Thus, circles around the projection’s center are projected to circles on the Cartesian plane. This can be useful for visualizing distances relative to a point of reference, such as commute distances.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-conicConformal) d3.geo.conicConformal()



Lambert’s conformal conic projection projects the globe conformally onto a cone.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-conicConformal_parallels) conicConformal.parallels([parallels])

If parallels is specified, sets the projection’s standard parallels to the specified two-element array of latitudes (in degrees) and returns the projection. If parallels is not specified, returns the current parallels.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-conicEqualArea) d3.geo.conicEqualArea()

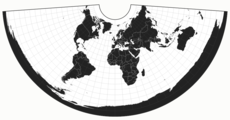


The Albers projection, as an [equal-area](http://en.wikipedia.org/wiki/Map_projection#Equal-area) projection, is recommended for [choropleths](http://mbostock.github.com/d3/ex/choropleth.html) as it preserves the relative areas of geographic features.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-conicEqualArea_parallels) conicEqualArea.parallels([parallels])

If parallels is specified, sets the Albers projection’s standard parallels to the specified two-element array of latitudes (in degrees) and returns the projection. If parallels is not specified, returns the current parallels. To minimize distortion, the parallels should be chosen to surround the projection’s [center](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-center).

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-conicEquidistant) d3.geo.conicEquidistant()



#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-conicEquidistant_parallels) conicEquidistant.parallels([parallels])

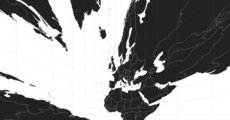
If parallels is specified, sets the projection’s standard parallels to the specified two-element array of latitudes (in degrees) and returns the projection. If parallels is not specified, returns the current parallels.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-equirectangular) d3.geo.equirectangular()



The equirectangular, or plate carrée projection, is the simplest possible geographic projection: the identity function. It is neither equal-area nor conformal, but is sometimes used for raster data. See [raster reprojection](http://bl.ocks.org/mbostock/4329423) for an example; the source image uses the equirectangular projection.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-gnomonic) d3.geo.gnomonic()



The gnomonic projection is an azimuthal projection that projects great circles as straight lines. See the [interactive gnomonic](http://bl.ocks.org/mbostock/3795048) for an example.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-mercator) d3.geo.mercator()



The spherical Mercator projection is commonly used by tiled mapping libraries (such as [OpenLayers](http://openlayers.org/) and [Leaflet](http://leaflet.cloudmade.com/)). For an example displaying raster tiles with the Mercator projection, see the [d3.geo.tile plugin](http://bl.ocks.org/mbostock/4150951). It is [conformal](http://en.wikipedia.org/wiki/Map_projection#Conformal); however, it introduces severe area distortion at world scale and thus is not recommended for choropleths.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-orthographic) d3.geo.orthographic()



The orthographic projection is an azimuthal projection suitable for displaying a single hemisphere; the point of perspective is at infinity. See the [animated world tour](http://bl.ocks.org/mbostock/4183330) and [interactive orthographic](http://bl.ocks.org/mbostock/3795040) for examples. For a general perspective projection, see the [satellite projection](http://bl.ocks.org/mbostock/3790444).

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-stereographic) d3.geo.stereographic()



The stereographic projection is another perspective (azimuthal) projection. The point of perspective is on the surface of the sphere, looking in; it is thus commonly used for celestial charts. See the [interactive stereographic](http://bl.ocks.org/mbostock/3763057) for an example.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-transverseMercator) d3.geo.transverseMercator()



The transverse Mercator projection. Note: this transverse Mercator projection does not currently support antimeridian cutting and is only intended for small areas, such as state-level maps; for whole-Earth maps using the transverse Mercator projection, use a [rotated Mercator projection](http://bl.ocks.org/mbostock/4695821) instead.

**Raw Projections**

D3 exposes several raw projections, designed for reuse when implementing a composite projection (such as [Sinu–Mollweide](http://bl.ocks.org/mbostock/4319903), which combines the raw [sinusoidal](http://bl.ocks.org/mbostock/3712399) and [Mollweide](http://bl.ocks.org/mbostock/3734336) projections). Raw projections are typically wrapped using [d3.geo.projection](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-projection) before use. These are point functions that take spherical coordinates λ and φ (in radians) as input and return a two-element array (also in radians) as output. Many raw projections also implement an inverse projection for mapping from planar to spherical coordinates.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-albers_raw) d3.geo.albers.raw(φ0, φ1)

An alias for [d3.geo.conicEqualArea.raw](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-conicEqualArea_raw).

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-azimuthalEqualArea_raw) d3.geo.azimuthalEqualArea.raw

The raw [azimuthal equal-area projection](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-azimuthalEqualArea).

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-azimuthalEquidistant_raw) d3.geo.azimuthalEquidistant.raw

The raw [azimuthal equidistant projection](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-azimuthalEquidistant).

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-conicConformal_raw) d3.geo.conicConformal.raw(φ0, φ1)

Returns a raw [conformal conic projection](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-conicConformal) with the specified parallels in radians.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-conicEqualArea_raw) d3.geo.conicEqualArea.raw(φ0, φ1)

Returns a raw [Albers projection](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-conicEqualArea) with the specified parallels in radians.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-conicEquidistant_raw) d3.geo.conicEquidistant.raw(φ0, φ1)

Returns a raw [equidistant conic projection](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-conicEquidistant) with the specified parallels in radians.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-equirectangular_raw) d3.geo.equirectangular.raw

The raw [equirectangular projection](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-equirectangular).

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-gnomonic_raw) d3.geo.gnomonic.raw

The raw [gnomonic projection](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-gnomonic).

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-mercator_raw) d3.geo.mercator.raw

The raw [Mercator projection](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-mercator).

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-orthographic_raw) d3.geo.orthographic.raw

The raw [orthographic projection](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-orthographic).

#### [#](https://github.com/mbostock/d3/wiki/Geo-Projections" \l "wiki-stereographic_raw) d3.geo.stereographic.raw

The raw [stereographic projection](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-stereographic).

### [Streams](https://github.com/mbostock/d3/wiki/Geo-Streams)

* [d3.geo.stream](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-stream) - convert a GeoJSON object to a geometry stream.
* [stream.point](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-stream_point) - indicate an x, y (and optionally z) coordinate.
* [stream.lineStart](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-stream_lineStart) - indicate the start of a line or ring.
* [stream.lineEnd](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-stream_lineEnd) - indicate the end of a line or ring.
* [stream.polygonStart](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-stream_polygonStart) - indicate the start of a polygon.
* [stream.polygonEnd](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-stream_polygonEnd) - indicate the end of a polygon.
* [stream.sphere](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-stream_sphere) - indicate a sphere.
* [d3.geo.transform](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-transform) - transform streaming geometries.
* [transform.stream](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-transform_stream) - wraps a given stream.
* [d3.geo.clipExtent](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-clipExtent) - a stream transform that clips geometries to a given axis-aligned rectangle.
* [clipExtent.extent](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-clipExtent_extent) - sets the clip extent.

**Geo Streams**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Geo](https://github.com/mbostock/d3/wiki/Geo) ▸ **Geo Streams**

For fast transformations of geometry without temporary copies of geometry objects, D3 uses **geometry streams**. The main [d3.geo.stream](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-d3_geo_stream) method converts a GeoJSON input object to a stream: a series of method calls on a stream listener. In addition, D3 provides several stream transformations that wrap listeners and transform the geometry. For example, the [projection.stream](https://github.com/mbostock/d3/wiki/Geo-Projections#stream) interface transforms spherical coordinates to Cartesian coordinates, and [d3.geo.path](https://github.com/mbostock/d3/wiki/Geo-Paths) serializes geometry to either SVG or Canvas. Internally, clipping and rotating are also implemented as stream transformations.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-stream) d3.geo.stream(object, listener)

Streams the specified [GeoJSON](http://geojson.org) object to the specified stream listener. (Despite the name “stream”, these method calls are currently synchronous.) While both features and geometry objects are supported as input, the stream interface only describes the geometry, and thus additional feature properties are not visible to listeners.

**Stream Listeners**

Stream listeners must implement several methods to traverse geometry. Listeners are inherently stateful; the meaning of a [point](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-point) depends on whether the point is inside of a [line](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-lineStart), and likewise a line is distinguished from a ring by a [polygon](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-polygonStart).

#### [#](https://github.com/mbostock/d3/wiki/Geo-Streams" \l "wiki-stream_point) listener.point(x, y[, z])

Indicates a point with the specified coordinates x and y (and optionally z). The coordinate system is unspecified and implementation-dependent; for example, [projection streams](https://github.com/mbostock/d3/wiki/Geo-Projections#wiki-stream) require spherical coordinates in degrees as input. Outside the context of a polygon or line, a point indicates a point geometry object ([Point](http://www.geojson.org/geojson-spec.html#point) or [MultiPoint](http://www.geojson.org/geojson-spec.html#multipoint)). Within a line or polygon ring, the point indicates a control point.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Streams" \l "wiki-stream_lineStart) listener.lineStart()

Indicates the start of a line or ring. Within a polygon, indicates the start of a ring. The first ring of a polygon is the exterior ring, and is typically clockwise. Any subsequent rings indicate holes in the polygon, and are typically counterclockwise.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Streams" \l "wiki-stream_lineEnd) listener.lineEnd()

Indicates the end of a line or ring. Within a polygon, indicates the end of a ring. Unlike GeoJSON, the redundant closing coordinate of a ring is not indicated via [point](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-point), and instead is implied via lineEnd within a polygon. Thus, the given polygon input:

{

"type": "Polygon",

"coordinates": [

[[0, 0], [1, 0], [1, 1], [0, 1], [0, 0]]

]

}

Will produce the following series of method calls on the listener:

listener.polygonStart();

listener.lineStart();

listener.point(0, 0);

listener.point(1, 0);

listener.point(1, 1);

listener.point(0, 1);

listener.lineEnd();

listener.polygonEnd();

#### [#](https://github.com/mbostock/d3/wiki/Geo-Streams" \l "wiki-stream_polygonStart) listener.polygonStart()

Indicates the start of a polygon. The first line of a polygon indicates the exterior ring, and any subsequent lines indicate interior holes.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Streams" \l "wiki-stream_polygonEnd) listener.polygonEnd()

Indicates the end of a polygon.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Streams" \l "wiki-stream_sphere) listener.sphere()

Indicates the sphere (the globe; the unit sphere centered at ⟨0,0,0⟩).

**Stream Transforms**

A stream transform wraps a stream listener, transforming the geometry before passing it along to the wrapped listener. A [geographic projection](https://github.com/mbostock/d3/wiki/Geo-Projections) is one example of a stream transform. The [d3.geo.transform](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-transform) class provides an easy way of implementing a custom stream transform.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Streams" \l "wiki-transform) d3.geo.transform(methods)

Creates a new stream transform using the specified hash of methods. The hash may contain implementations of any of the standard stream listener methods: [sphere](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-stream_sphere), [point](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-stream_point), [lineStart](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-stream_lineStart), [lineEnd](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-stream_lineEnd), [polygonStart](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-stream_polygonStart) and [polygonEnd](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-stream_polygonEnd). Any method that is not present in the specified hash will be implemented a pass-through directly to the wrapped stream. To access the wrapped stream within a method, use this.stream. For example, to implement a simple [2D matrix transform](http://bl.ocks.org/mbostock/5663666):

function matrix(a, b, c, d, tx, ty) {

return d3.geo.transform({

point: function(x, y) { this.stream.point(a \* x + b \* y + tx, c \* x + d \* y + ty); },

});

}

This transform can then be used in conjunction with [d3.geo.path](https://github.com/mbostock/d3/wiki/Geo-Paths). For example, to implement a 2D affine transform that flips the *y*-axis:

var path = d3.geo.path()

.projection(matrix(1, 0, 0, -1, 0, height));

#### [#](https://github.com/mbostock/d3/wiki/Geo-Streams" \l "wiki-transform_stream) transform.stream(listener)

Given the specified stream listener, returns a wrapped stream listener that applies this transform to any input geometry before streaming it to the wrapped listener.

#### [#](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-clipExtent) d3.geo.clipExtent()

Create a new stream [transform](https://github.com/mbostock/d3/wiki/Geo-Streams#wiki-transform) that implements axis-aligned rectangle clipping. This is typically used to clip geometry to the viewport after [projecting](https://github.com/mbostock/d3/wiki/Geo-Projections).

#### [#](https://github.com/mbostock/d3/wiki/Geo-Streams" \l "wiki-clipExtent_extent) clipExtent.extent([extent])

If extent is specified, sets the clip extent to the specified rectangle [​[*x0*, *y0*], [*x1*, *y1*]​] and returns this transform. If extent is not specified, returns the current clip extent, which defaults to [​[0, 0], [960, 500]​].

## [d3.geom (Geometry)](https://github.com/mbostock/d3/wiki/Geometry)

### [Voronoi](https://github.com/mbostock/d3/wiki/Voronoi-Geom)

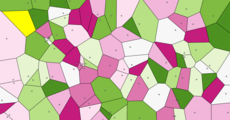
* [d3.geom.voronoi](https://github.com/mbostock/d3/wiki/Voronoi-Geom#wiki-voronoi) - create a Voronoi layout with default accessors.
* [voronoi](https://github.com/mbostock/d3/wiki/Voronoi-Geom#wiki-_voronoi) - compute the Voronoi tessellation for the specified points.
* [voronoi.x](https://github.com/mbostock/d3/wiki/Voronoi-Geom#wiki-x) - get or set the x-coordinate accessor for each point.
* [voronoi.y](https://github.com/mbostock/d3/wiki/Voronoi-Geom#wiki-y) - get or set the y-coordinate accessor for each point.
* [voronoi.clipExent](https://github.com/mbostock/d3/wiki/Voronoi-Geom#wiki-clipExtent) - get or set the clip extent for the tesselation.
* [voronoi.links](https://github.com/mbostock/d3/wiki/Voronoi-Geom#wiki-links) - compute the Delaunay mesh as a network of links.
* [voronoi.triangles](https://github.com/mbostock/d3/wiki/Voronoi-Geom#wiki-triangles) - compute the Delaunay mesh as a triangular tessellation.

**Voronoi Geom**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Geometry](https://github.com/mbostock/d3/wiki/Geometry) ▸ **Voronoi Geom**

Voronoi layouts are particularly useful for invisible interactive regions, as demonstrated in Nate Vack’s [Voronoi picking](http://bl.ocks.org/njvack/1405439) example. See Tovi Grossman’s paper on [bubble cursors](http://www.tovigrossman.com/BubbleCursor) for a related concept.

#### [#](https://github.com/mbostock/d3/wiki/Voronoi-Geom#wiki-voronoi) d3.geom.voronoi()



Creates a Voronoi layout with default accessors.

#### [#](https://github.com/mbostock/d3/wiki/Voronoi-Geom" \l "wiki-_voronoi) voronoi(data)

Returns an array of polygons, one for each input vertex in the specified data array. If any vertices are coincident or have NaN positions, the behavior of this method is undefined: most likely, invalid polygons will be returned! You should filter invalid vertices, and consolidate coincident vertices, before computing the tessellation.

#### [#](https://github.com/mbostock/d3/wiki/Voronoi-Geom#wiki-x) voronoi.x([x])

If x is specified, sets the x-coordinate accessor. If x is not specified, returns the current x-coordinate accessor, which defaults to:

function(d) { return d[0]; }

#### [#](https://github.com/mbostock/d3/wiki/Voronoi-Geom#wiki-y) voronoi.y([y])

If y is specified, sets the y-coordinate accessor. If y is not specified, returns the current y-coordinate accessor, which defaults to:

function(d) { return d[1]; }

#### [#](https://github.com/mbostock/d3/wiki/Voronoi-Geom#wiki-clipExtent) voronoi.clipExtent([extent])

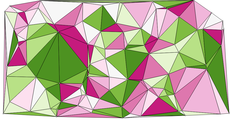
Gets or sets the clip extent of the Voronoi layout; see [this example](http://bl.ocks.org/mbostock/4237768). This implementation does not clip the returned polygons by default, but will clip them to a given rectangle if specified; this is strongly recommended, as unclipped polygons may have large coordinates which do not display correctly.

Alternatively, you can also employ custom clipping without specifying a size, either in SVG or by post-processing with [polygon.clip](https://github.com/mbostock/d3/wiki/Polygon-Geom#wiki-clip).

#### [#](https://github.com/mbostock/d3/wiki/Voronoi-Geom#wiki-links) voronoi.links(data)

…

#### [#](https://github.com/mbostock/d3/wiki/Voronoi-Geom" \l "wiki-triangles) voronoi.triangles(data)



Returns an array of triangles.

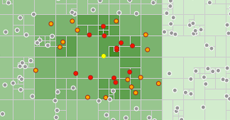
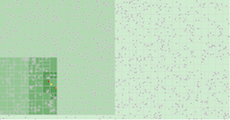
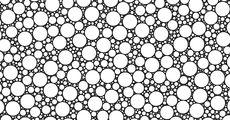
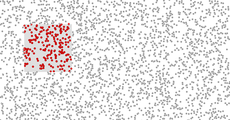
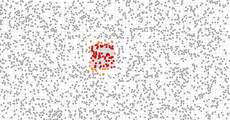
### [Quadtree](https://github.com/mbostock/d3/wiki/Quadtree-Geom)

* [d3.geom.quadtree](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-quadtree) - constructs a quadtree for an array of points.
* [quadtree.add](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-add) - add a point to the quadtree.
* [quadtree.visit](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-visit) - recursively visit nodes in the quadtree.

**Quadtree Geom**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [Geometry](https://github.com/mbostock/d3/wiki/Geometry) ▸ **Quadtree Geom**

A **quadtree** is a two-dimensional recursive spatial subdivision. This implementation uses square partitions, dividing each square into four equally-sized squares. Each point exists in a unique node; if multiple points are in the same position, some points may be stored on internal nodes rather than leaf nodes. Quadtrees can be used to accelerate various spatial operations, such as the Barnes-Hut approximation for computing n-body forces, or collision detection.



#### [#](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-quadtree) d3.geom.quadtree()

Creates a new quadtree factory with the default [x-accessor](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-x), [y-accessor](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-y) and [extent](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-extent). The [returned function](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-_quadtree) can be used to create any number of quadtrees from data with the factory’s configuration.

#### [#](https://github.com/mbostock/d3/wiki/Quadtree-Geom" \l "wiki-_quadtree) quadtree(points)

Constructs a new quadtree for the specified array of data points, returning the root node of a new quadtree. The x- and y-coordinates of each point are determined using the current [x-](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-x) and [y-](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-y) accessor functions. To build a quadtree by adding points incrementally, the specified points array can be empty, and then points can be later [added](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-add) to the returned root node; in this case, you must also specify the [extent](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-extent) of the quadtree.

Each node in the quadtree has several properties:

* nodes - a sparse array of the four child nodes in order: top-left, top-right, bottom-left, bottom-right
* leaf - a boolean indicating whether this is a internal or leaf node
* point - the point associated with this node, if any (may apply to either internal or leaf nodes)
* x - the x-coordinate of the associated point, if any
* y - the y-coordinate of the associated point, if any

The returned root node also defines [add](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-add) and [visit](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-visit) methods.

#### [#](https://github.com/mbostock/d3/wiki/Quadtree-Geom" \l "wiki-add) root.add(point)

Adds the specified new point to the quadtree.

#### [#](https://github.com/mbostock/d3/wiki/Quadtree-Geom" \l "wiki-visit) root.visit(callback)

Visits each node in the quadtree, invoking the specified callback with arguments {*node*, x1, y1, x2, y2} for each node. Nodes are traversed in pre-order. If the callback returns true for a the given node, then the children of that node are not visited; otherwise, all child nodes are visited.

#### [#](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-x) quadtree.x([x])

If x is specified, sets the x-coordinate accessor and returns this quadtree factory. If x is not specified, returns the current x-coordinate accessor, which defaults to:

function(d) { return d[0]; }

For each point added to the quadtree, either during [initial construction](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-_quadtree) or lazily [added](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-add), the x-accessor is invoked with the arguments {*d*, *i*}, where d is the current point and i is its index in the array of all points. The x-accessor must then return a numeric value indicating the x-coordinate of the given point. The x-accessor may also be defined as a constant number rather than a function, if desired.

#### [#](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-y) quadtree.y([y])

If y is specified, sets the y-coordinate accessor and returns this quadtree factory. If y is not specified, returns the current y-coordinate accessor, which defaults to:

function(d) { return d[1]; }

For each point added to the quadtree, either during [initial construction](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-_quadtree) or lazily [added](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-add), the y-accessor is invoked with the arguments {*d*, *i*}, where d is the current point and i is its index in the array of all points. The y-accessor must then return a numeric value indicating the y-coordinate of the given point. The y-accessor may also be defined as a constant number rather than a function, if desired.

#### [#](https://github.com/mbostock/d3/wiki/Quadtree-Geom" \l "wiki-extent) quadtree.extent([extent])

If extent is specified, sets the current extent and returns this quadtree factory. If extent is not specified, returns the current extent, which defaults to null. When the extent is null, an extent will be computed automatically by scanning the array of input points passed to the [quadtree constructor](https://github.com/mbostock/d3/wiki/Quadtree-Geom#wiki-_quadtree). Otherwise, the extent must be specified as a two-dimensional array [​[*x0*, *y0*], [​*x1*, *y1*]​], where x0 and y0 are the lower bounds of the extent, and x1 and y1 are the upper bounds of the extent. Setting an extent is required when constructing a quadtree lazily from an initially-empty set of nodes.

### [Polygon](https://github.com/mbostock/d3/wiki/Polygon-Geom)

* [d3.geom.polygon](https://github.com/mbostock/d3/wiki/Polygon-Geom#wiki-polygon) - create a polygon from the specified array of points.
* [polygon.area](https://github.com/mbostock/d3/wiki/Polygon-Geom#wiki-area) - compute the counterclockwise area of this polygon.
* [polygon.centroid](https://github.com/mbostock/d3/wiki/Polygon-Geom#wiki-centroid) - compute the area centroid of this polygon.
* [polygon.clip](https://github.com/mbostock/d3/wiki/Polygon-Geom#wiki-clip) - clip the specified polygon to this polygon.

**Polygon Geom**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Geometry](https://github.com/mbostock/d3/wiki/Geometry) ▸ **Polygon Geom**

#### [#](https://github.com/mbostock/d3/wiki/Polygon-Geom#wiki-polygon) d3.geom.polygon(vertices)

Returns the input array of vertices with additional methods attached, as described below.

#### [#](https://github.com/mbostock/d3/wiki/Polygon-Geom#wiki-area) polygon.area()

Returns the signed area of this polygon. If the vertices are in counterclockwise order, the area is positive, otherwise it is negative.

#### [#](https://github.com/mbostock/d3/wiki/Polygon-Geom#wiki-centroid) polygon.centroid()

Returns a two-element array representing the centroid of this polygon.

#### [#](https://github.com/mbostock/d3/wiki/Polygon-Geom" \l "wiki-clip) polygon.clip(subject)

Clips the subject polygon against this polygon. In other words, returns a polygon representing the intersection of this polygon and the subject polygon. Assumes the clip polygon is counterclockwise and convex.

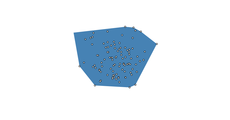
### [Hull](https://github.com/mbostock/d3/wiki/Hull-Geom)

* [d3.geom.hull](https://github.com/mbostock/d3/wiki/Hull-Geom#wiki-hull) - create a convex hull layout with default accessors.
* [hull](https://github.com/mbostock/d3/wiki/Hull-Geom#wiki-_hull) - compute the convex hull for the given array of points.
* [hull.x](https://github.com/mbostock/d3/wiki/Hull-Geom#wiki-x) - get or set the x-coordinate accessor.
* [hull.y](https://github.com/mbostock/d3/wiki/Hull-Geom#wiki-y) - get or set the y-coordinate accessor.

**Hull Geom**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Geometry](https://github.com/mbostock/d3/wiki/Geometry) ▸ **Hull Geom**

#### [#](https://github.com/mbostock/d3/wiki/Hull-Geom#wiki-hull) d3.geom.hull()



Create a new hull layout with the default x- and y-accessors.

#### [#](https://github.com/mbostock/d3/wiki/Hull-Geom" \l "wiki-_hull) hull(vertices)

Returns the convex hull for the specified vertices array, using the current x- and y-coordinate accessors. The returned convex hull is represented as an array containing a subset of the input vertices, arranged in counterclockwise order (for consistency with [polygon.clip](https://github.com/mbostock/d3/wiki/Polygon-Geom#wiki-clip)).

#### [#](https://github.com/mbostock/d3/wiki/Hull-Geom#wiki-x) hull.x([x])

If x is specified, sets the x-coordinate accessor. If x is not specified, returns the current x-coordinate accessor, which defaults to:

function(d) { return d[0]; }

#### [#](https://github.com/mbostock/d3/wiki/Hull-Geom#wiki-y) hull.y([y])

If y is specified, sets the y-coordinate accessor. If y is not specified, returns the current y-coordinate accessor, which defaults to:

function(d) { return d[1]; }

## [d3.behavior (Behaviors)](https://github.com/mbostock/d3/wiki/Behaviors)

### [Drag](https://github.com/mbostock/d3/wiki/Drag-Behavior)

* [d3.behavior.drag](https://github.com/mbostock/d3/wiki/Drag-Behavior#wiki-drag)
* [drag.origin](https://github.com/mbostock/d3/wiki/Drag-Behavior#wiki-origin)
* [drag.on](https://github.com/mbostock/d3/wiki/Drag-Behavior#wiki-on)

**Drag Behavior**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Behaviors](https://github.com/mbostock/d3/wiki/Behaviors) ▸ **Drag Behavior**

This behavior automatically creates event listeners to handle drag gestures on an element. Both mouse events and touch events are supported.

#### [#](https://github.com/mbostock/d3/wiki/Drag-Behavior#wiki-drag) d3.behavior.drag()

Constructs a new drag behavior.

#### [#](https://github.com/mbostock/d3/wiki/Drag-Behavior" \l "wiki-on) drag.on(type[, listener])

Registers the specified listener to receive events of the specified type from the drag behavior. If no listener is specified, returns the currently-registered listener for the specified event type. (The type may include a namespace; see [dispatch.on](https://github.com/mbostock/d3/wiki/Internals#wiki-dispatch_on) for additional details.) The following events are supported:

* dragstart - when a drag gesture starts.
* drag - when the drag gesture moves.
* dragend - when the drag gesture finishes.

Drag events (but not dragstart and dragend events) expose "x" and "y" properties representing the current position of the drag gesture in local coordinates. By default, this position is simply the [mouse](https://github.com/mbostock/d3/wiki/Selections#wiki-d3_mouse) (or [touch](https://github.com/mbostock/d3/wiki/Selections#wiki-d3_touches)) position; however, the position can be modified by specifying an [origin](https://github.com/mbostock/d3/wiki/Drag-Behavior#wiki-origin). The drag event also exposes "dx" and "dy" properties representing the element’s coordinates relative to its position at the beginning of the gesture, which is occasionally more convenient than specifying an explicit origin.

During a drag gesture, some browser default behaviors (such as text selection) are prevented. In addition, the default behavior for a click event immediately a non-empty drag gesture is prevented, so as to allow the dragging of links. When registering your own click listener on draggable elements, you can check whether the click event was suppressed as follows:

selection.on("click", function() {

if (d3.event.defaultPrevented) return; // click suppressed

console.log("clicked!");

});

When combining drag behaviors with other event listeners for interaction events, you may also consider stopping propagation on the source event to prevent multiple actions:

drag.on("dragstart", function() {

d3.event.sourceEvent.stopPropagation(); // silence other listeners

});

#### [#](https://github.com/mbostock/d3/wiki/Drag-Behavior" \l "wiki-origin) drag.origin([origin])

If origin is specified, sets the origin accessor to the specified function. If origin is not specified, returns the current origin accessor which defaults to null.

The origin accessor function is used to determine the starting position (the “origin”) of the element being dragged; this allows the drag behavior to preserve the offset between the mouse position and the starting element position during drag. If the origin accessor is null, then the element position is set to the mouse position on drag; this can cause a noticeable jump on large elements. If an origin accessor is specified, the function is called on mousedown. The function is invoked in the same manner as other operator functions, being passed the current datum d and index i, with the this context as the clicked-on DOM element. To access the current event, use the global [d3.event](https://github.com/mbostock/d3/wiki/Selections#wiki-d3_event). The origin accessor must return an object with x and y properties representing the starting coordinates of the element being dragged.

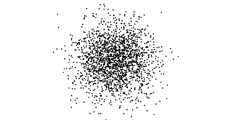
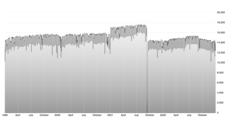
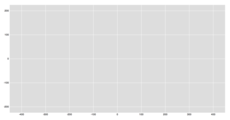
Frequently the origin accessor is specified as the identity function: function(d) { return d; }. This is suitable when the datum bound to the dragged element is already an object with x and y attributes representing its current position. For example: <http://bl.ocks.org/1557377>

### [Zoom](https://github.com/mbostock/d3/wiki/Zoom-Behavior)

* [d3.behavior.zoom](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-zoom) - create a zoom behavior.
* [zoom](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-_zoom) - apply the zoom behavior to the selected elements.
* [zoom.scale](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-scale) - the current scale factor.
* [zoom.translate](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-translate) - the current translate offset.
* [zoom.scaleExtent](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-scaleExtent) - optional limits on the scale factor.
* [zoom.center](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-center) - an optional focal point for mousewheel zooming.
* [zoom.size](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-size) - the dimensions of the viewport.
* [zoom.x](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-x) - an optional scale whose domain is bound to the x extent of the viewport.
* [zoom.y](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-y) - an optional scale whose domain is bound to the y extent of the viewport.
* [zoom.on](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-on) - listeners for when the scale or translate changes.
* [zoom.event](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-event) - dispatch zoom events after setting the scale or translate.

**Zoom Behavior**

[Wiki](https://github.com/mbostock/d3/wiki/Home) ▸ [API Reference](https://github.com/mbostock/d3/wiki/API-Reference) ▸ [Behaviors](https://github.com/mbostock/d3/wiki/Behaviors) ▸ **Zoom Behavior**



This behavior automatically creates event listeners to handle zooming and panning gestures on a container element. Both mouse and touch events are supported.

#### [#](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-zoom) d3.behavior.zoom()

Constructs a new zoom behavior.

#### [#](https://github.com/mbostock/d3/wiki/Zoom-Behavior" \l "wiki-_zoom) zoom(selection)

Applies the zoom behavior to the specified selection, registering the necessary event listeners to support panning and zooming.

#### [#](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-translate) zoom.translate([translate])

Specifies the current zoom translation vector. If not specified, returns the current translation vector, which defaults to [0, 0].

#### [#](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-scale) zoom.scale([scale])

Specifies the current zoom scale. If not specified, returns the current zoom scale, which defaults to 1.

#### [#](https://github.com/mbostock/d3/wiki/Zoom-Behavior" \l "wiki-scaleExtent) zoom.scaleExtent([extent])

Specifies the zoom scale's allowed range as a two-element array, [*minimum*, *maximum*]. If not specified, returns the current scale extent, which defaults to [0, Infinity].

#### [#](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-center) zoom.center([center])

If center is specified, sets the [focal point](http://bl.ocks.org/mbostock/6226534) [*x*, *y*] for mousewheel zooming and returns this zoom behavior. If center is not specified, returns the current focal point, which defaults to null. A null center indicates that mousewheel zooming should zoom in and out around the current mouse location.

#### [#](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-size) zoom.size([size])

If size is specified, sets the viewport size to the specified dimensions [*width*, *height*] and returns this zoom behavior. If size is not specified, returns the current viewport size which defaults to [960, 500]. A size is needed to support [smooth zooming](https://github.com/mbostock/d3/wiki/Transitions#wiki-d3_interpolateZoom) during transitions.

#### [#](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-x) zoom.x([x])

Specifies an x-scale whose domain should be automatically adjusted when zooming. If not specified, returns the current x-scale, which defaults to null. If the scale's domain or range is modified programmatically, this function should be called again. Setting the x-scale also resets the scale to 1 and the translate to [0, 0].

#### [#](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-y) zoom.y([y])

Specifies an y-scale whose domain should be automatically adjusted when zooming. If not specified, returns the current y-scale, which defaults to null. If the scale's domain or range is modified programmatically, this function should be called again. Setting the y-scale also resets the scale to 1 and the translate to [0, 0].

#### [#](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-on) zoom.on(type, listener)

Registers the specified listener to receive events of the specified type from the zoom behavior. The following types are supported:

* zoomstart - at the start of a zoom gesture (e.g., touchstart).
* zoom - when the view changes (e.g., touchmove).
* zoomend - at the end of the current zoom gesture (e.g., touchend).

If an event listener was already registered for the same type, the existing listener is removed before the new listener is added. To register multiple listeners for the same event type, the type may be followed by an optional namespace, such as "zoom.foo" and "zoom.bar". To remove a listener, pass null as the listener.

For mousewheel events, which happen discretely with no explicit start and end reported by the browser, events that occur within 50 milliseconds of each other are grouped into a single zoom gesture. If you want more robust interpretation of these gestures, please petition your browser vendor of choice for better touch event support.

When fired, the d3.event object will contain the following properties:

* scale - a number; the current scale.
* translate - a two-element array representing the current translation vector.

#### [#](https://github.com/mbostock/d3/wiki/Zoom-Behavior" \l "wiki-event) zoom.event(selection)

If selection is a selection, immediately dispatches a zoom gesture to registered listeners, as the three event sequence zoomstart, zoom and zoomend. This can be useful in triggering listeners after setting the [translate](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-translate) or [scale](https://github.com/mbostock/d3/wiki/Zoom-Behavior#wiki-scale) programatically. If selection is a transition, registers the appropriate tweens so that the zoom behavior dispatches events over the course of the transition: a zoomstart event when the transition starts from the previously-set view, zoom events for each tick of the transition, and finally a zoomend event when the transition ends. Note that the transition will be [interrupted](https://github.com/mbostock/d3/wiki/Selections#wiki-interrupt) if the user starts zooming before the transition ends.

Harry，2014年4月28日,周一