Network Layout

Maneesh Agrawala

CS 448B: Visualization
Fall 2020

1

Last Time: Animation

2

Implementing Animation

3

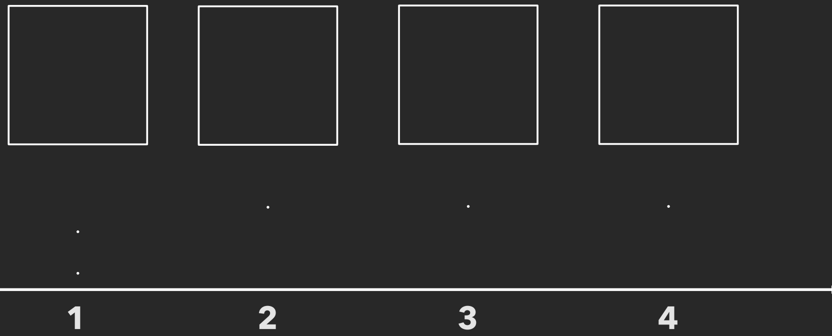
Animation Approaches

Frame-based Animation

Redraw scene at regular interval (e.g., 16ms)
Developer defines the redraw function

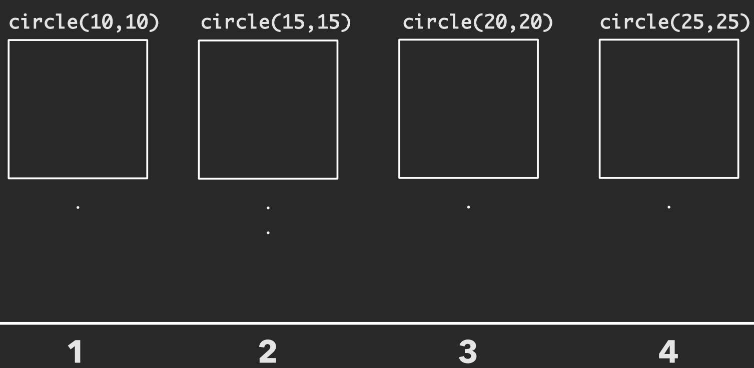
4

Frame-based Animation



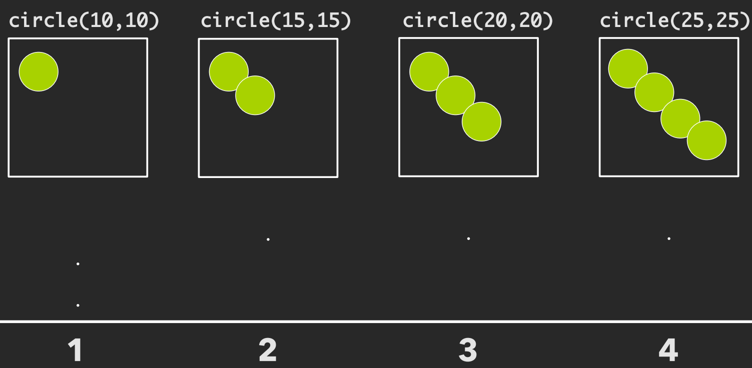
5

Frame-based Animation



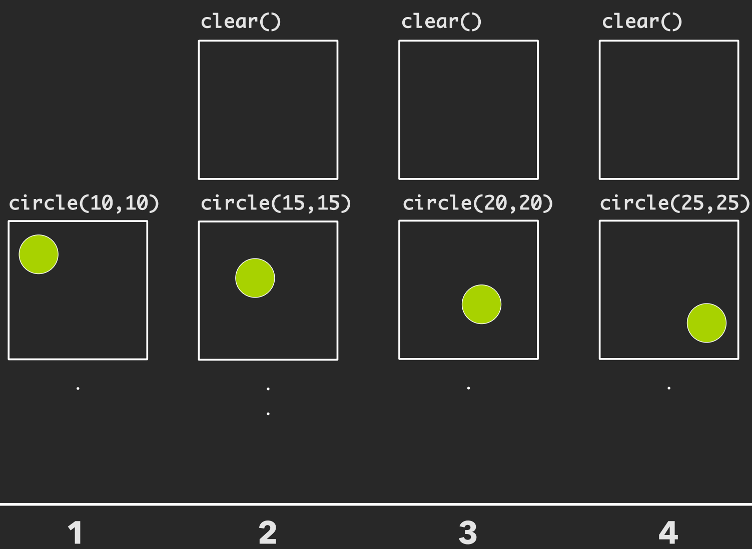
6

Frame-based Animation



7

Frame-based Animation



8

Animation Approaches

Frame-based Animation

Redraw scene at regular interval (e.g., 16ms)
Developer defines the redraw function

Animation Approaches

Frame-based Animation

Redraw scene at regular interval (e.g., 16ms)
Developer defines the redraw function

Transition-based Animation (Hudson & Stasko '93)

Specify property value, duration & easing (tweening)
Typically computed via interpolation

step(*fraction*) { $x_{now} = x_{start} + fraction * (x_{end} - x_{start})$; }

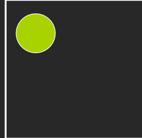
Timing & redraw managed by UI toolkit

Transition-based Animation

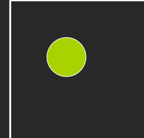
from: **(10,10)** to: **(25,25)** duration: **3sec**

dx=25-10

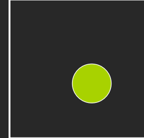
x=10+(t/3)*dx



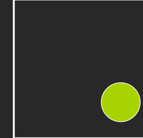
x=10+(t/3)*dx



x=10+(t/3)*dx



x=10+(t/3)*dx



0s **1s** **2s** **3s**

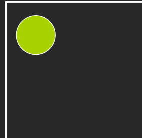
11

Transition-based Animation

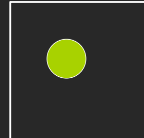
from: **(10,10)** to: **(25,25)** duration: **3sec**
Toolkit handles frame-by-frame updates

dx=25-10

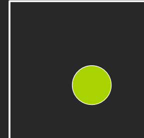
x=10+(t/3)*dx



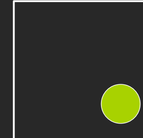
x=10+(t/3)*dx



x=10+(t/3)*dx



x=10+(t/3)*dx



0s **1s** **2s** **3s**

12

D3 Transitions

Any d3 *selection* can be used to drive animation.

13

D3 Transitions

Any d3 *selection* can be used to drive animation.

```
// Select SVG rectangles and bind them to data values.
var bars = svg.selectAll("rect.bars").data(values);
```

14

D3 Transitions

Any d3 *selection* can be used to drive animation.

```
// Select SVG rectangles and bind them to data values.
var bars = svg.selectAll("rect.bars").data(values);
// Static transition: update position and color of bars.
bars
  .attr("x", (d) => xScale(d.foo))
  .attr("y", (d) => yScale(d.bar))
  .style("fill", (d) => colorScale(d.baz));
```

15

D3 Transitions

Any d3 *selection* can be used to drive animation.

```
// Select SVG rectangles and bind them to data values.
var bars = svg.selectAll("rect.bars").data(values);
// Animated transition: interpolate to target values using default timing
bars.transition()
  .attr("x", (d) => xScale(d.foo))
  .attr("y", (d) => yScale(d.bar))
  .style("fill", (d) => colorScale(d.baz));
```

16

D3 Transitions

Any d3 *selection* can be used to drive animation.

```
// Select SVG rectangles and bind them to data values.

var bars = svg.selectAll("rect.bars").data(values);
// Animated transition: interpolate to target values using default timing
bars.transition()
   .attr("x", (d) => xScale(d.foo))
   .attr("y", (d) => yScale(d.bar))
   .style("fill", (d) => colorScale(d.baz));

   // Animation is implicitly queued to run!
```

17

D3 Transitions, Continued

```
bars.transition()
   .duration(500)              // animation duration in ms
   .delay(0)                   // onset delay in ms
   .ease(d3.easeBounce)  // set easing (or "pacing") style
   .attr("x", (d) => xScale(d.foo))
   …
```

18

D3 Transitions, Continued

```
bars.transition()
   .duration(500)              // animation duration in ms
   .delay(0)                   // onset delay in ms
   .ease(d3.easeBounce)        // set easing (or "pacing") style
   .attr("x", (d) => xScale(d.foo))
   ...

bars.exit().transition()  // animate elements leaving display
  .style("opacity", 0)    // fade out to fully transparent
  .remove();              // remove from DOM upon completion
```

19

Easing Functions

Goals: **stylize animation, improve perception.**

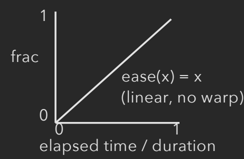
Basic idea is to warp time: as *duration* goes from start (0%) to end (100%), dynamically adjust the *interpolation fraction* using an easing function**.**

20

Easing Functions

Goals: **stylize animation, improve perception.**

Basic idea is to warp time: as *duration* goes from start (0%) to end (100%), dynamically adjust the *interpolation fraction* using an easing function**.**

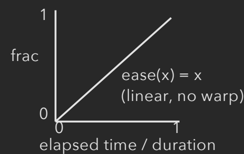


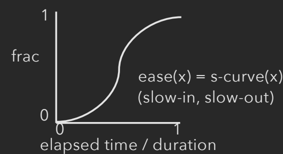
21

Easing Functions

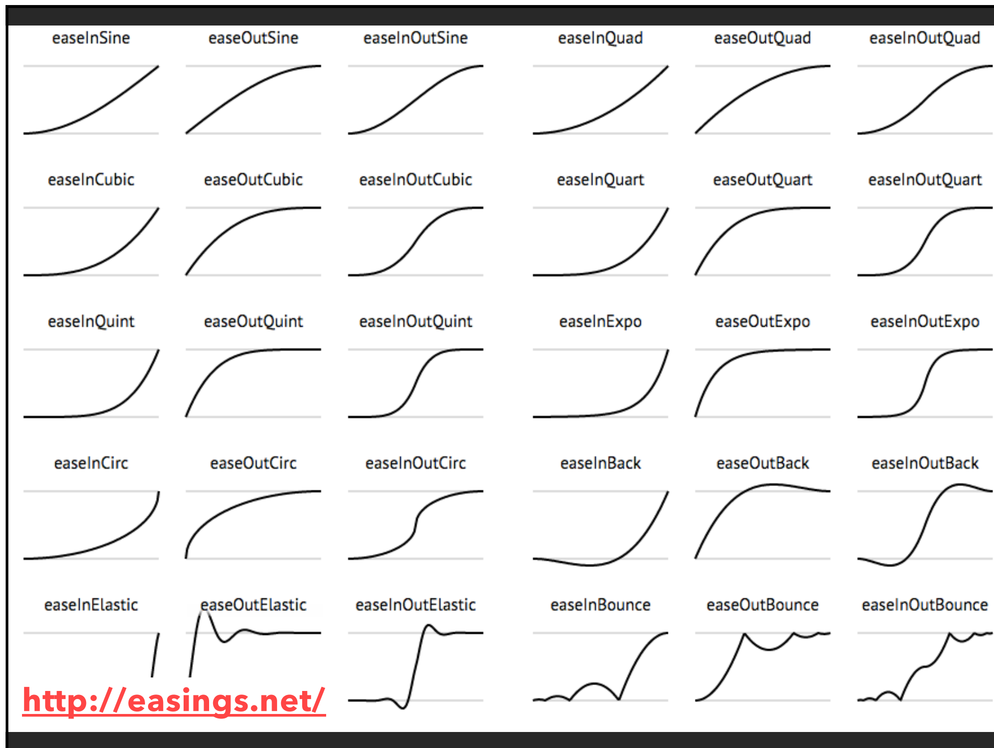
Goals: **stylize animation, improve perception.**

Basic idea is to warp time: as *duration* goes from start (0%) to end (100%), dynamically adjust the *interpolation fraction* using an easing function**.**





22



23

Summary

Animation is a salient visual phenomenon
Attention, object constancy, causality, timing

Design with care: congruence & apprehension

For processes, static images may be preferable
For transitions, animation has some benefits, but consider task and timing

27

Announcements

28

Final project

Data analysis/explainer or conduct research
- **Data analysis**: Analyze dataset in depth & make a visual explainer
- **Research**: Pose problem, Implement creative solution

Deliverables
- **Data analysis/explainer**: Article with multiple interactive visualizations
- **Research**: Implementation of solution and web-based demo if possible
- **Short video (2 min)** demoing and explaining the project

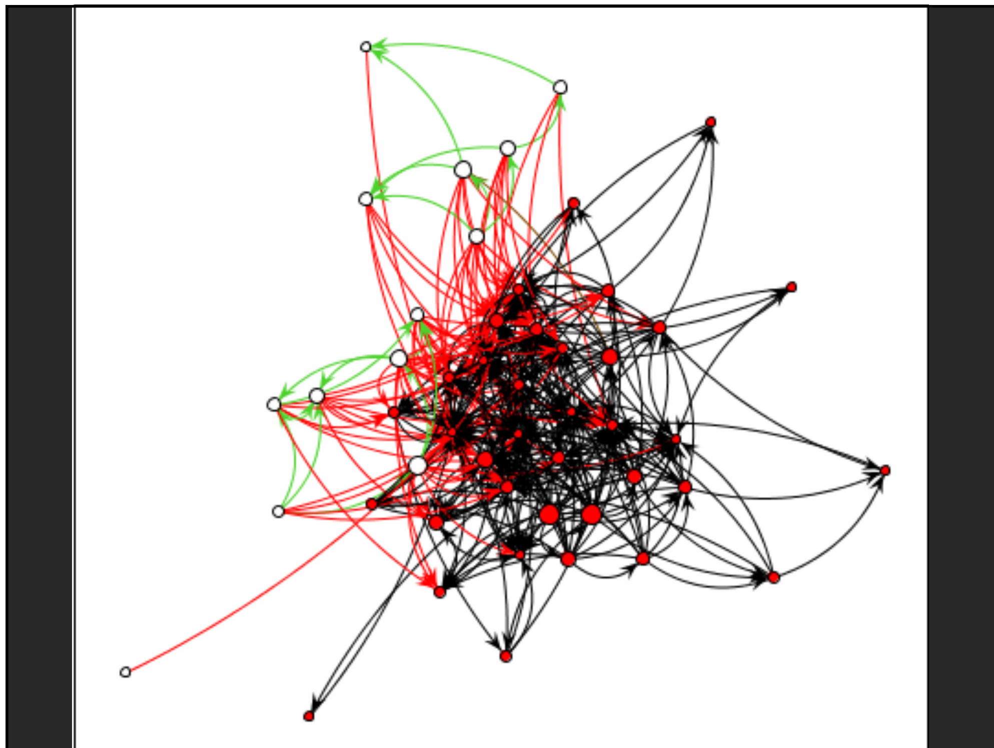
Schedule
- Project proposal: Thu 10/29
- Design Review and Feedback: Tue 11/17 & Thu 11/19
- Final code and video: Sat 11/21 11:59pm

Grading
- Groups of up to 3 people, graded individually
- Clearly report responsibilities of each member

29

Network Layout

30



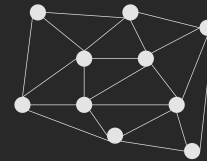
31

Graphs and Trees

Graphs

Model relations among data

Nodes and edges

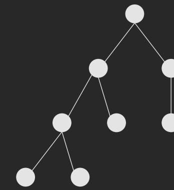


Trees

Graphs with hierarchical structure

Connected graph with N-1 edges

Nodes as *parents* and *children*



32

Spatial Layout

Primary concern – layout of nodes and edges

Often (but not always) goal is to depict structure

- Connectivity, path-following
- Network distance
- Clustering
- Ordering (e.g., hierarchy level)

33

Topics

Tree Layout
Node-Link Graph Layout
- Sugiyama-Style Layout
- Force-Directed Layout

Alternatives to Node-Link Graph Layout
- Matrix Diagrams
- Attribute-Drive Layout

35

Tree Layout

36

Tree Visualization

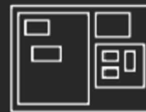
Indentation

- Linear list, indentation encodes depth

Node-Link diagrams

- Nodes connected by lines/curves

Enclosure diagrams



- Represent hierarchy by enclosure

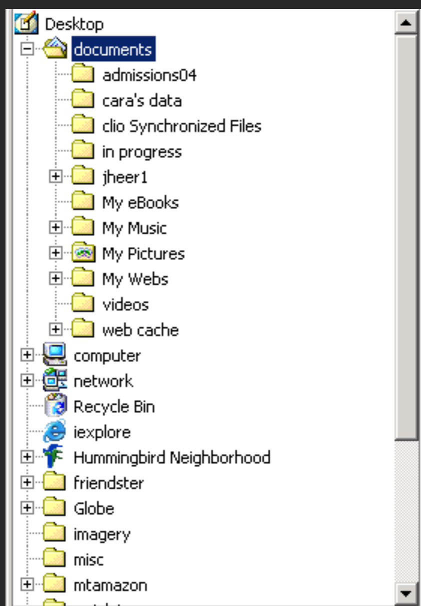
Layering

- Layering and alignment

Tree layout is fast: O(n) or O(n log n), enabling real-time layout for interaction

37

Indentation



Items along vertically spaced rows

Indentation shows parent/child relationships

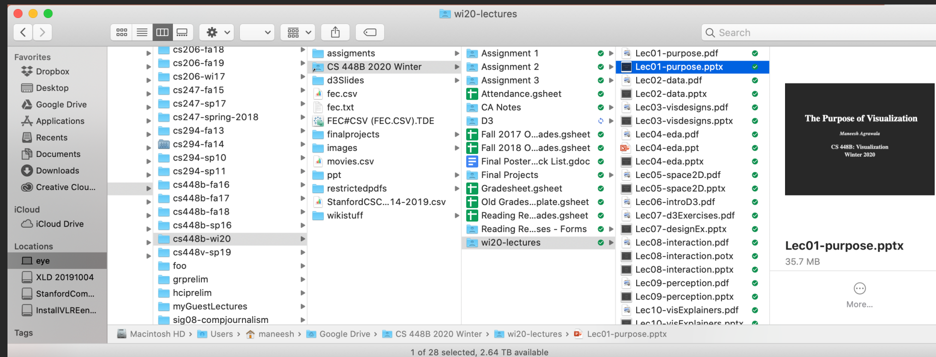
Often used in interfaces

Breadth/depth contend for space

Often requires scrolling

38

Single-Focus (Accordion) List



Separate breadth & depth in 2D
Focus on single path at a time

39

Node-Link Diagrams

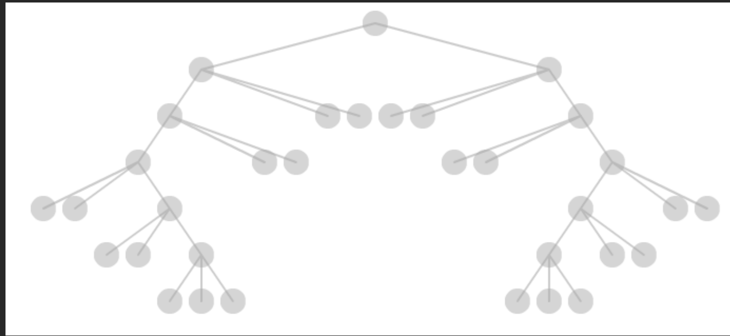
Nodes distributed in space, connected by lines
Use 2D space to break apart breadth and depth
Space used to communicate hierarchical orientation
- Typically towards authority or generality

40

Basic Recursive Approach

Repeatedly divide space for subtrees by leaf count

- Breadth of tree along one dimension
- Depth along the other dimension

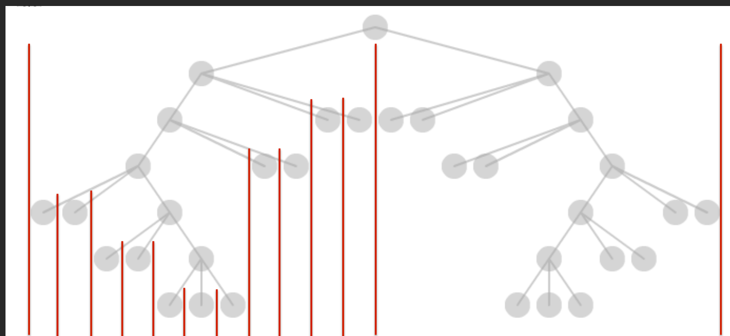


42

Basic Recursive Approach

Repeatedly divide space for subtrees by leaf count

- Breadth of tree along one dimension
- Depth along the other dimension



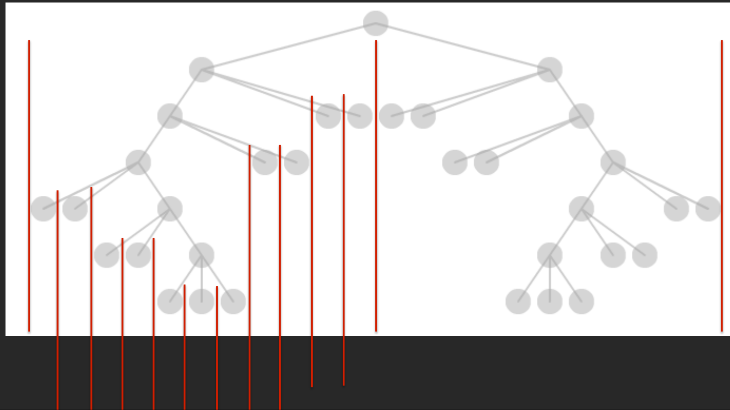
43

Basic Recursive Approach

Repeatedly divide space for subtrees by leaf count

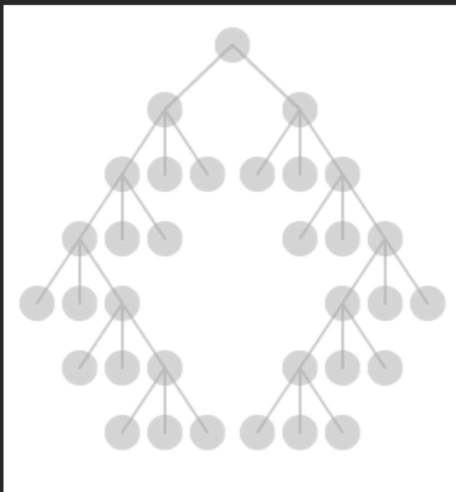
- Breadth of tree along one dimension
- Depth along the other dimension

Problem: Exponential growth of breadth



44

Reingold & Tilford' s Tidier Layout



Goal: maximize density and symmetry.

Originally for binary trees, extended by Walker to cover general case.

This extension was corrected by Buchheim et al. to achieve a linear time algorithm

45

Reingold-Tilford Layout

Design concerns

- Clearly encode depth level
- No edge crossings
- Isomorphic subtrees drawn identically
- Ordering and symmetry preserved
- *Compact layout (don't waste space)*

Reingold-Tilford Algorithm

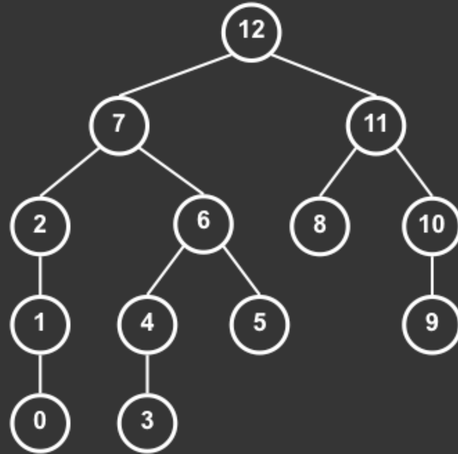
Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees

- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions

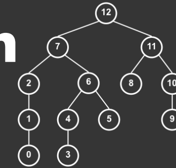
- Sum of initial layout and aggregated shifts

Reingold-Tilford Algorithm



48

Reingold-Tilford Algorithm

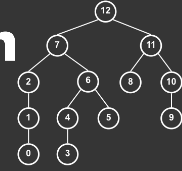


Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts

49

Reingold-Tilford Algorithm

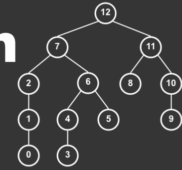


Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts

50

Reingold-Tilford Algorithm

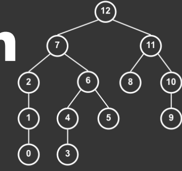


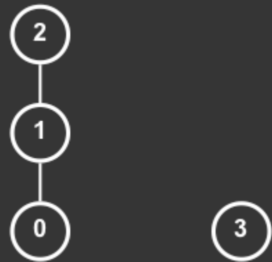
Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts

51

Reingold-Tilford Algorithm





Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees

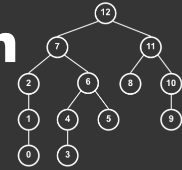
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

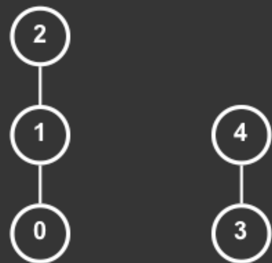
Top-down (preorder) pass for assignment of final positions

- Sum of initial layout and aggregated shifts

52

Reingold-Tilford Algorithm





Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees

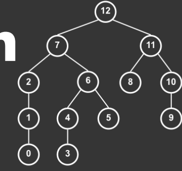
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions

- Sum of initial layout and aggregated shifts

53

Reingold-Tilford Algorithm

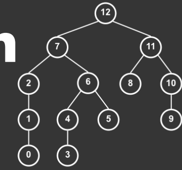


Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts

54

Reingold-Tilford Algorithm

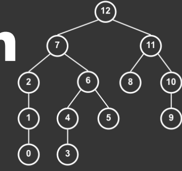


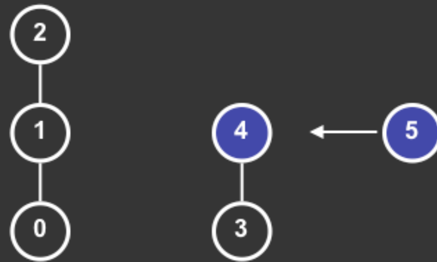
Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts

55

Reingold-Tilford Algorithm



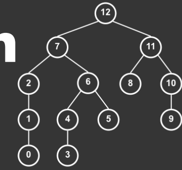


Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts

56

Reingold-Tilford Algorithm

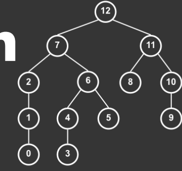


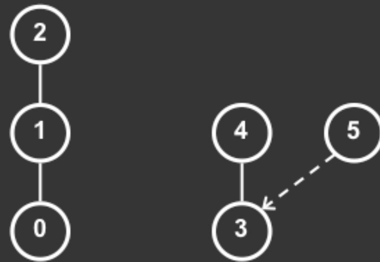
Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts

57

Reingold-Tilford Algorithm



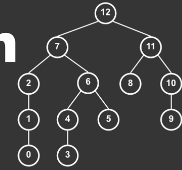


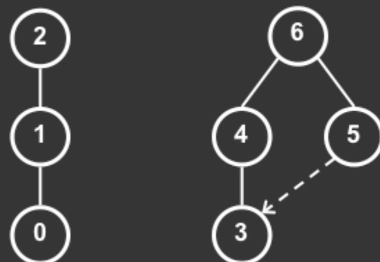
Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts

58

Reingold-Tilford Algorithm



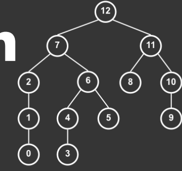


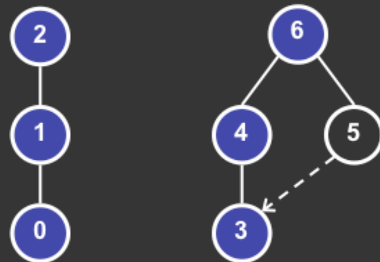
Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts

59

Reingold-Tilford Algorithm

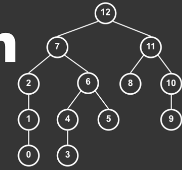


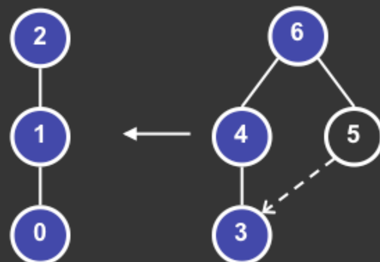


- Linear algorithm – starts with bottom-up (postorder) pass
- Set Y-coord by depth, arbitrary starting X-coord
- Merge left and right subtrees
  - Shift right as close as possible to left
    - Computed efficiently by maintaining subtree contours
  - "Shifts" in position saved for each node as visited
  - Parent nodes are centered above their children
- Top-down (preorder) pass for assignment of final positions
  - Sum of initial layout and aggregated shifts

60

Reingold-Tilford Algorithm

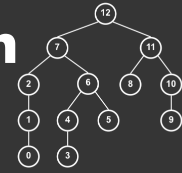


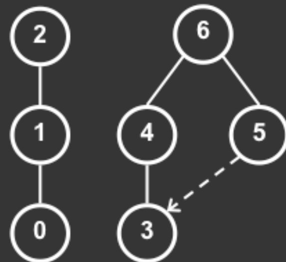


- Linear algorithm – starts with bottom-up (postorder) pass
- Set Y-coord by depth, arbitrary starting X-coord
- Merge left and right subtrees
  - Shift right as close as possible to left
    - Computed efficiently by maintaining subtree contours
  - "Shifts" in position saved for each node as visited
  - Parent nodes are centered above their children
- Top-down (preorder) pass for assignment of final positions
  - Sum of initial layout and aggregated shifts

61

Reingold-Tilford Algorithm

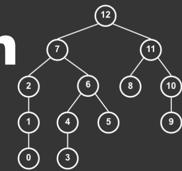


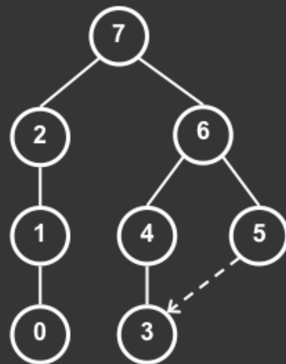


- Linear algorithm – starts with bottom-up (postorder) pass
- Set Y-coord by depth, arbitrary starting X-coord
- Merge left and right subtrees
  - Shift right as close as possible to left
    - Computed efficiently by maintaining subtree contours
  - "Shifts" in position saved for each node as visited
  - Parent nodes are centered above their children
- Top-down (preorder) pass for assignment of final positions
  - Sum of initial layout and aggregated shifts

62

Reingold-Tilford Algorithm

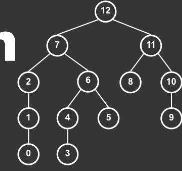


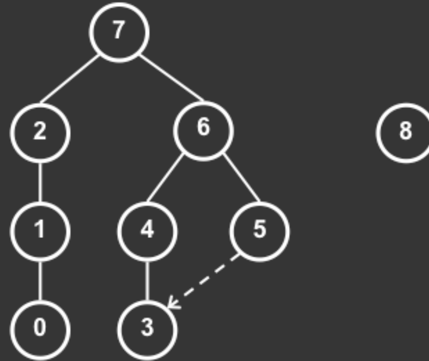


- Linear algorithm – starts with bottom-up (postorder) pass
- Set Y-coord by depth, arbitrary starting X-coord
- Merge left and right subtrees
  - Shift right as close as possible to left
    - Computed efficiently by maintaining subtree contours
  - "Shifts" in position saved for each node as visited
  - Parent nodes are centered above their children
- Top-down (preorder) pass for assignment of final positions
  - Sum of initial layout and aggregated shifts

63

Reingold-Tilford Algorithm





Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees

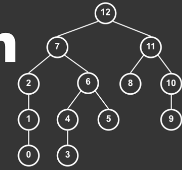
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

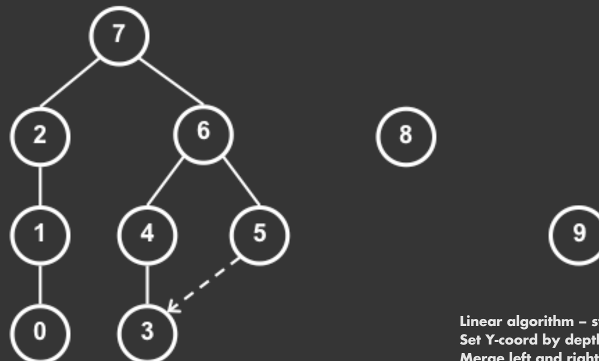
Top-down (preorder) pass for assignment of final positions

- Sum of initial layout and aggregated shifts

64

Reingold-Tilford Algorithm





Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees

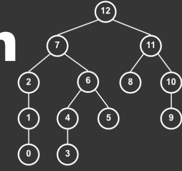
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

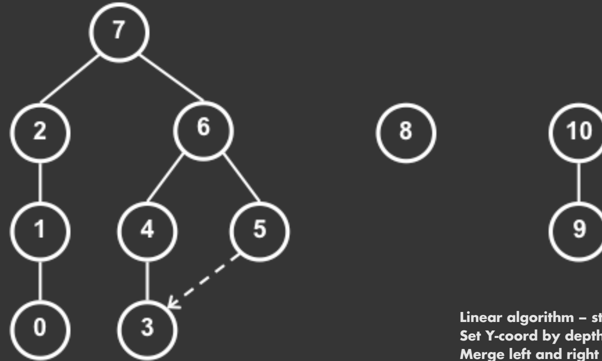
Top-down (preorder) pass for assignment of final positions

- Sum of initial layout and aggregated shifts

65

Reingold-Tilford Algorithm

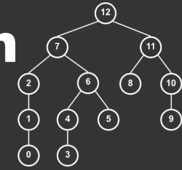


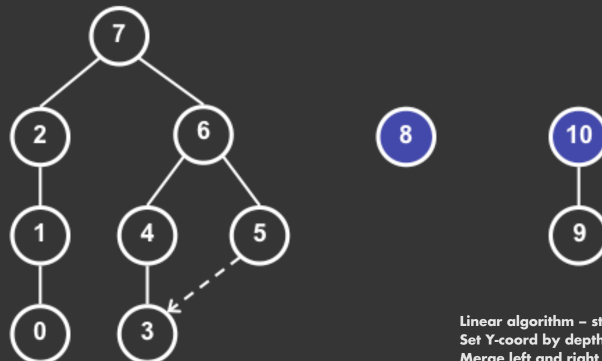


- Linear algorithm – starts with bottom-up (postorder) pass
- Set Y-coord by depth, arbitrary starting X-coord
- Merge left and right subtrees
  - Shift right as close as possible to left
    - Computed efficiently by maintaining subtree contours
  - "Shifts" in position saved for each node as visited
  - Parent nodes are centered above their children
- Top-down (preorder) pass for assignment of final positions
  - Sum of initial layout and aggregated shifts

66

Reingold-Tilford Algorithm

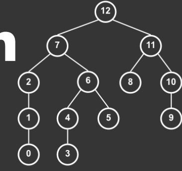


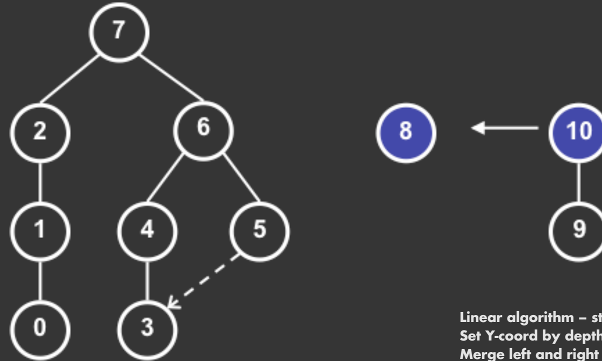


- Linear algorithm – starts with bottom-up (postorder) pass
- Set Y-coord by depth, arbitrary starting X-coord
- Merge left and right subtrees
  - Shift right as close as possible to left
    - Computed efficiently by maintaining subtree contours
  - "Shifts" in position saved for each node as visited
  - Parent nodes are centered above their children
- Top-down (preorder) pass for assignment of final positions
  - Sum of initial layout and aggregated shifts

67

Reingold-Tilford Algorithm



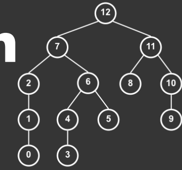


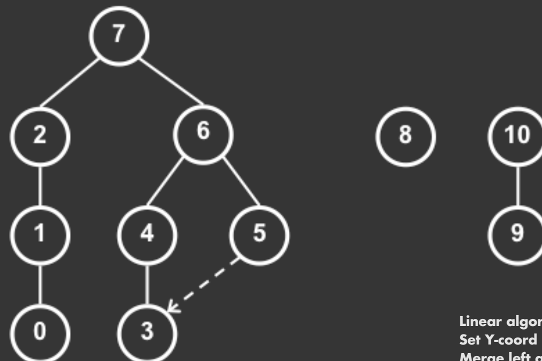
Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts

68

Reingold-Tilford Algorithm



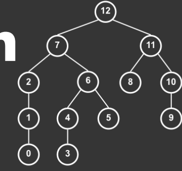


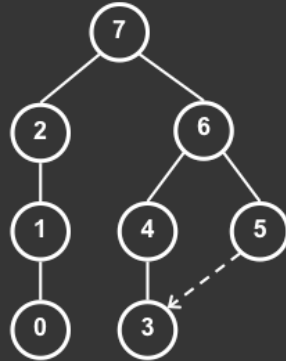
Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

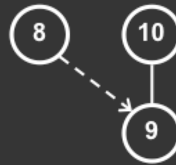
Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts

69

Reingold-Tilford Algorithm





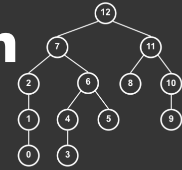


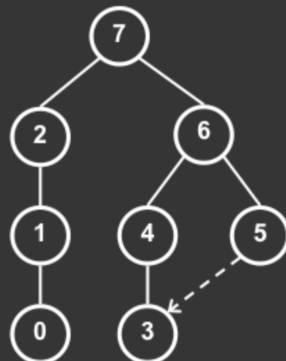
Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

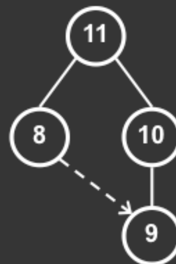
Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts

70

Reingold-Tilford Algorithm





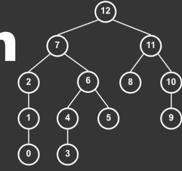


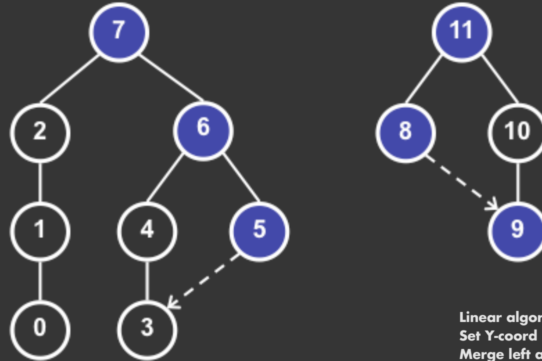
Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts

71

Reingold-Tilford Algorithm





Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees

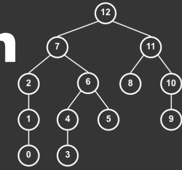
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

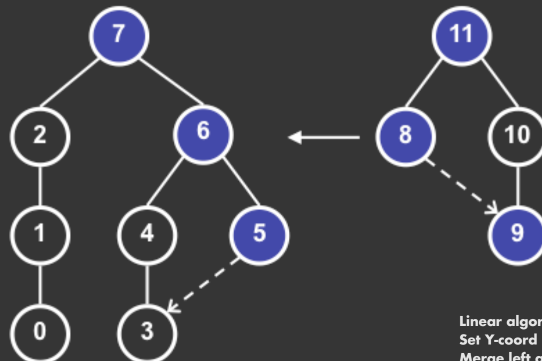
Top-down (preorder) pass for assignment of final positions

- Sum of initial layout and aggregated shifts

72

Reingold-Tilford Algorithm





Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees

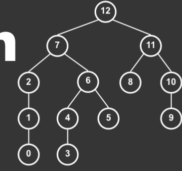
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

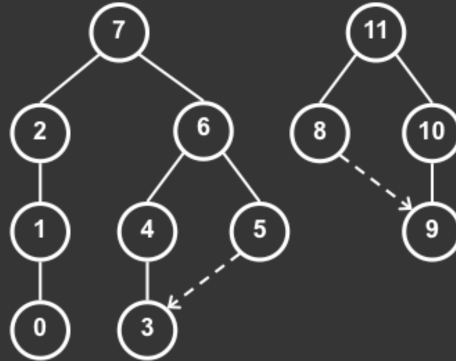
Top-down (preorder) pass for assignment of final positions

- Sum of initial layout and aggregated shifts

73

Reingold-Tilford Algorithm



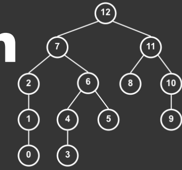


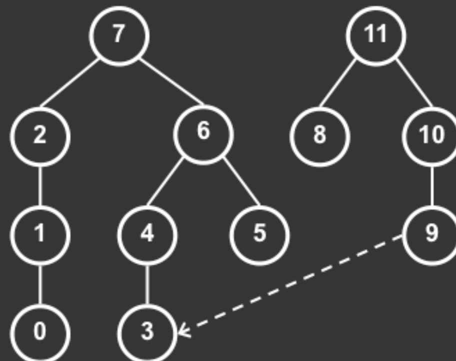
Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts

74

Reingold-Tilford Algorithm



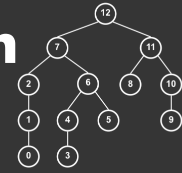


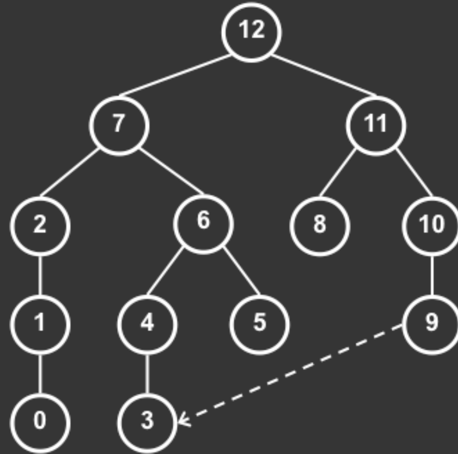
Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts

75

Reingold-Tilford Algorithm





Linear algorithm – starts with bottom-up (postorder) pass
Set Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees

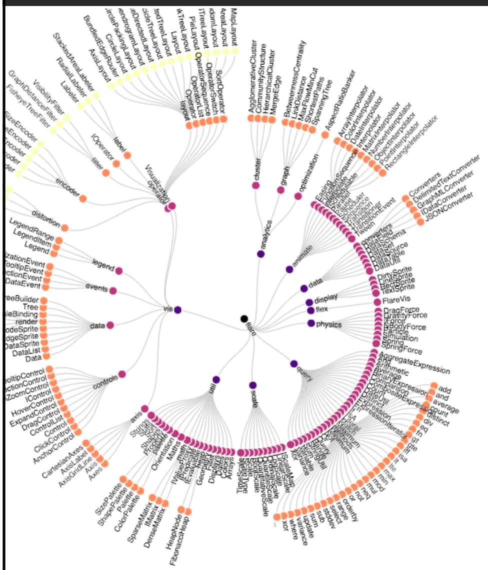
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- "Shifts" in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions

- Sum of initial layout and aggregated shifts

76

Radial Layout



Node-link diagram in polar coords

Radius encodes depth root at center

Angular sectors assigned to subtrees (recursive approach)

Reingold-Tilford approach can also be applied here

79

Problems with Node-Link Diagrams

Scale

Tree breadth often grows exponentially

Even with tidier layout, quickly run out of space

Possible solutions

Filtering

Focus+Context

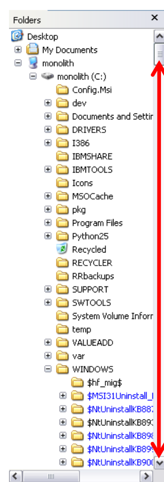
Scrolling or Panning

Zooming

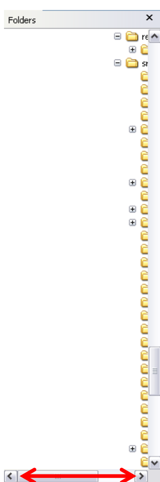
Aggregation

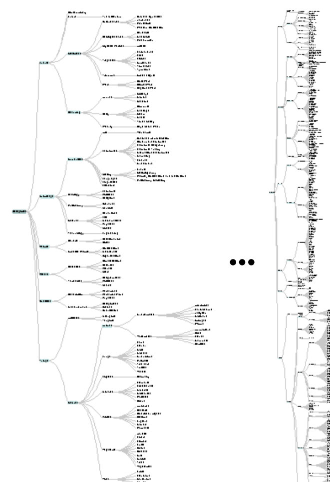
82

Visualizing Large Hierarchies



…





… …

Indented Layout

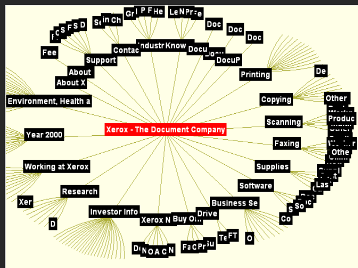
Reingold-Tilford Layout

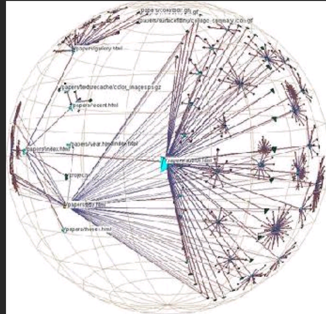
83

MC Escher, *Circle Limit IV*

84

Hyperbolic Layout





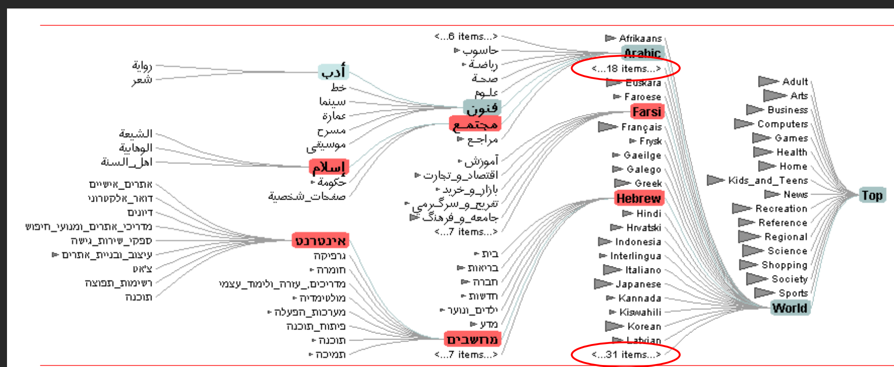
Layout in hyperbolic space, then project on to Euclidean plane

Why? Like tree breadth, the hyperbolic plane expands exponentially

Also computable in 3D, projected into a sphere

85

Degree-of-Interest Trees [AVI 04]

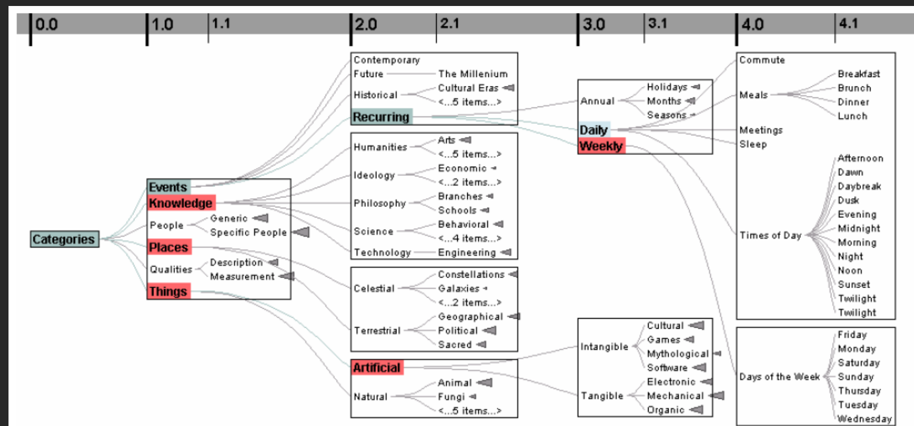


Space-constrained, multi-focal tree layout

https://www.youtube.com/watch?v=RTQ0N4QY0vc
https://observablehq.com/@d3/collapsible-tree

86

Degree-of-Interest Trees



Cull “un-interesting” nodes on a per block basis until all blocks on a level fit within bounds

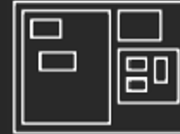
Center child blocks under parents

https://www.youtube.com/watch?v=RTQ0N4QY0vc
https://observablehq.com/@d3/collapsible-tree

87

Enclosure Diagrams

Encode structure using spatial enclosure
Popularly known as TreeMaps



Benefits

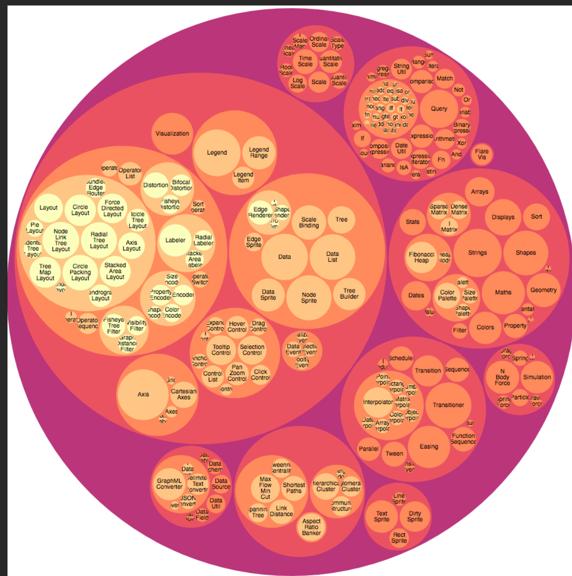
- Provides a single view of an entire tree
- Easier to spot large/small nodes

Problems

- Difficult to accurately read depth

88

Circle Packing Layout



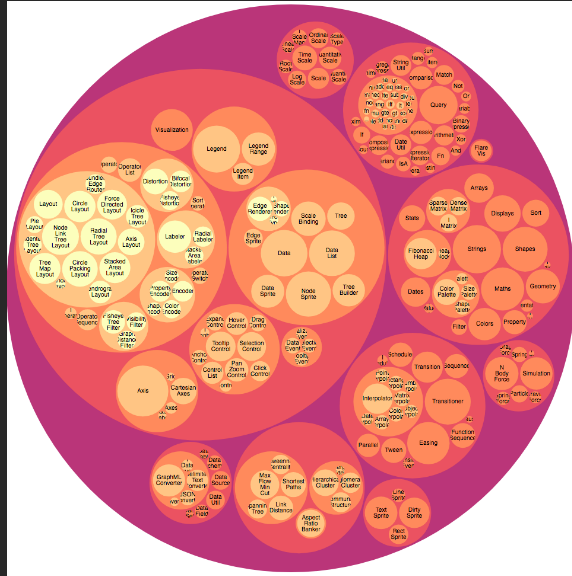
Nodes represented as sized circles

Nesting to show parent-child relationships

Problems:

89

Circle Packing Layout



Nodes represented as sized circles

Nesting to show parent-child relationships

Problems:
- Inefficient use of space
- Parent size misleading

90

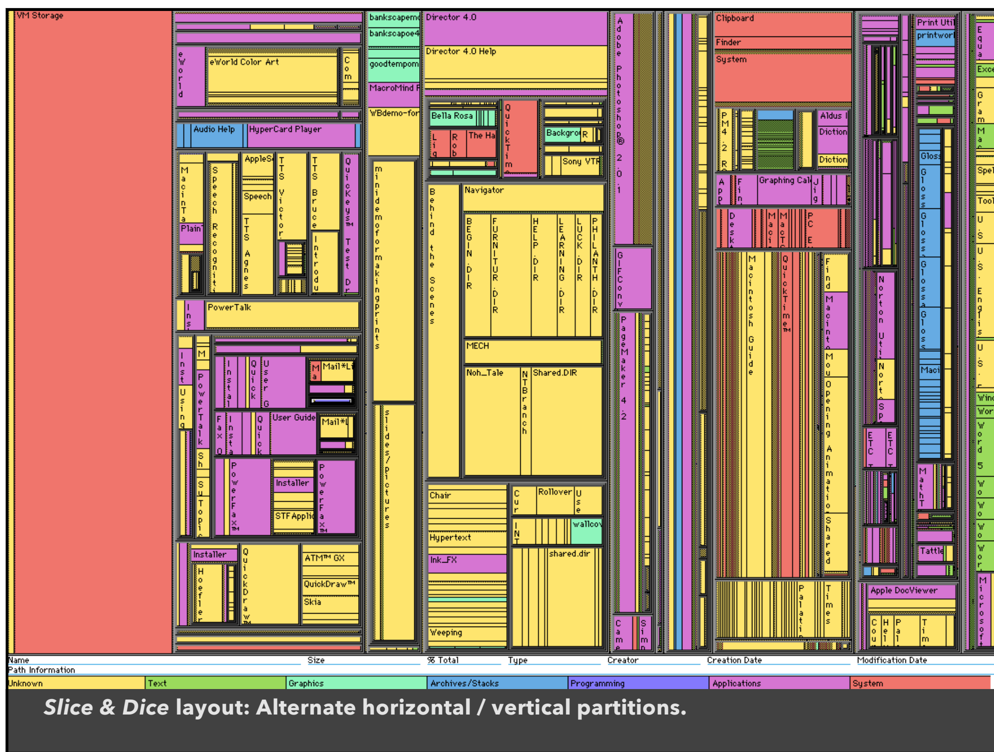
Treemaps

Hierarchy visualization that emphasizes values of nodes via area encoding

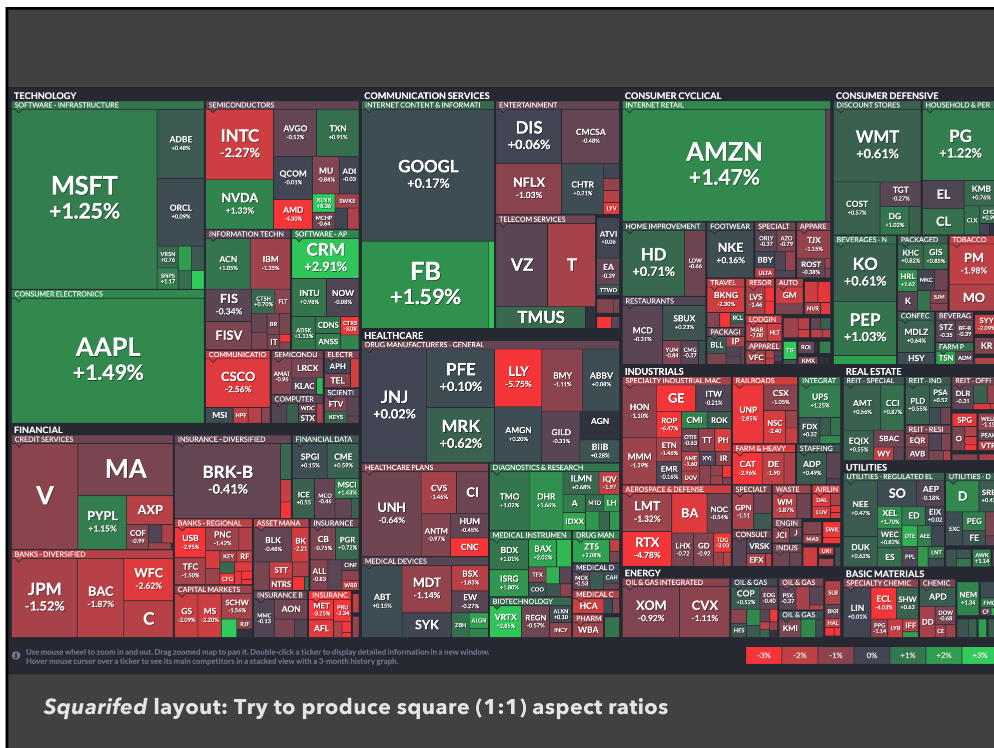
Partition 2D space such that leaf nodes have sizes proportional to data values

First layout algorithms proposed by Shneiderman et al. in 1990, with focus on showing file sizes on a hard drive

92



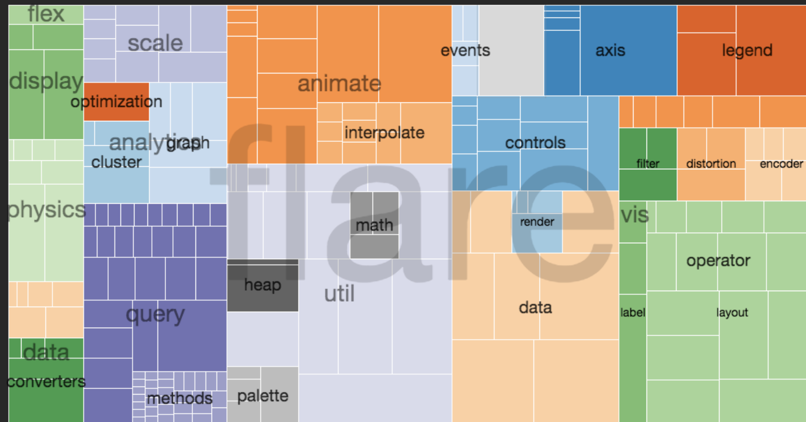
93



94

Squarified Treemaps [Bruls 00]

Greedy optimization for objective of square rectangles
Slice/dice within siblings; alternate whenever ratio worsens



https://vega.github.io/vega/examples/treemap/

95

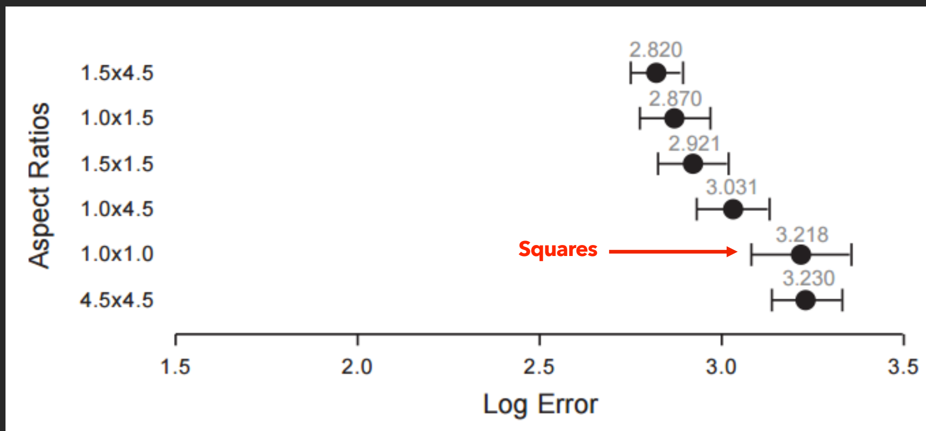
Why Squares

Posited Benefits of 1:1 Aspect Ratios

1. Minimize perimeter, reducing border ink.
2. Easier to select with a mouse cursor.
   Validated by empirical research & Fitt's Law!
3. Similar aspect ratios are easier to compare.
   Seems intuitive, but is this true?

96

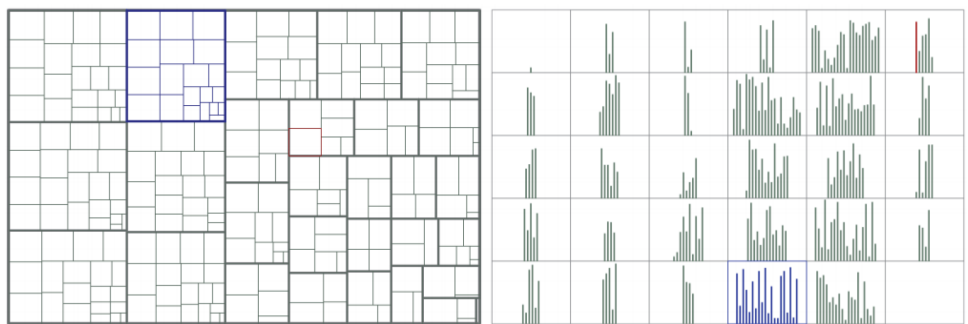
Error vs. Aspect Ratio [Kong 10]



1. Comparison of squares has higher error!
2. Squarify works because it fails to meet its objective?

97

Treemaps vs. Bar Charts [Kong 10]

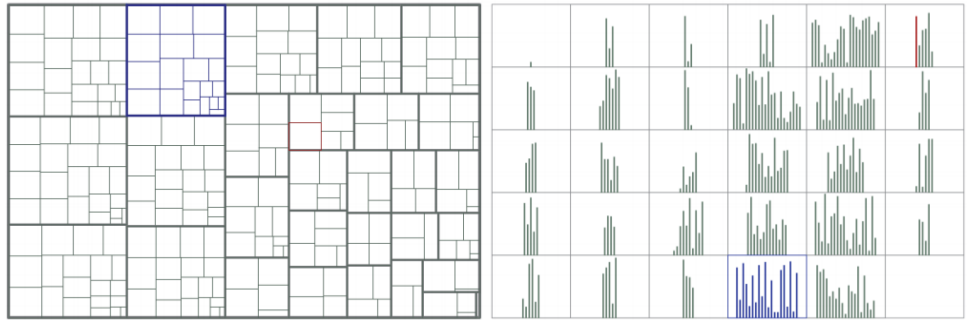


Height more perceptually effective than area

What if element count is high?
What about comparing groups of elements such as leaf values to internal node values?

99

Treemaps vs. Bar Charts [Kong 10]



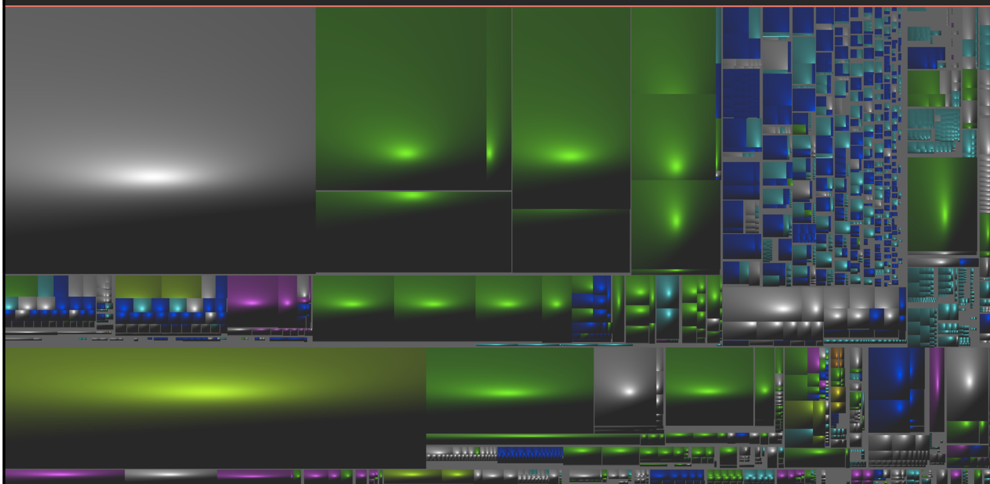
At low densities (< 4k elements), bar charts more accurate than treemaps for leaf-node comparisons.

At higher density, treemaps led to faster judgments.

Treemaps better for group-level comparisons.

100

Cushion Treemaps [van Wijk 99]



Use shading to emphasize hierarchical structure

101

Cascaded Treemaps [Lü 08]

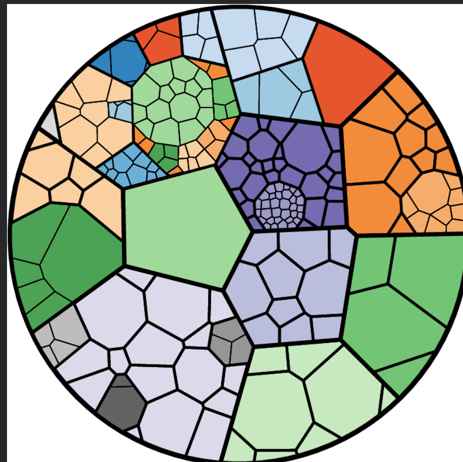
Use 2.5D effect emphasize hierarchical structure

102

Voronoi Treemaps [Balzer 05]

Treemaps with arbitrary polygonal shape and boundary

Uses iterative, weighted Voronoi tessellations to achieve cells with value-proportional areas



103

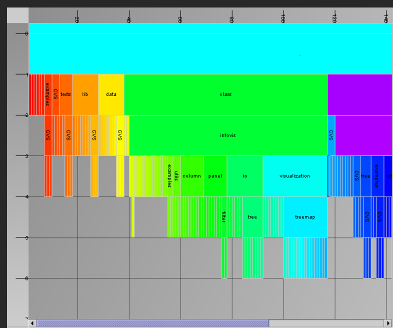
Layered Diagrams

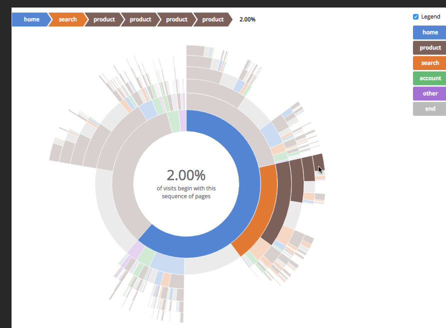
Signify tree structure using
Layering
Adjacency
Alignment

Involves recursive sub-division of space
Can apply the same set of approaches as in node-link layout

105

Icicle and Sunburst Trees

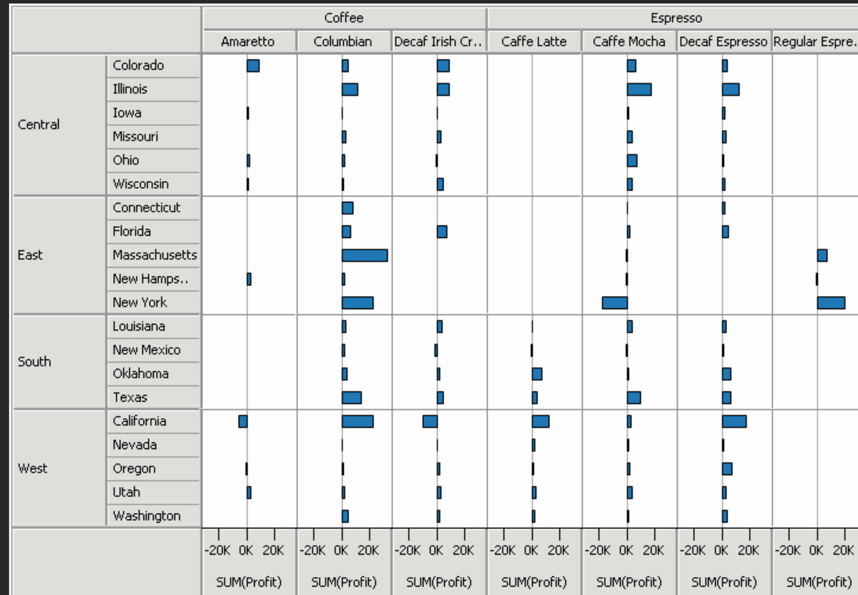




Higher-level nodes get a larger layer area, whether that is horizontal or angular extent
Child levels are layered, constrained to parent's extent

106

Layered Tree Drawing



107

Node-Link Graph Layout

109

Spanning Tree Layout

Many graphs are tree-like or have useful spanning trees

Websites, Social Networks

Use tree layout on spanning tree of graph

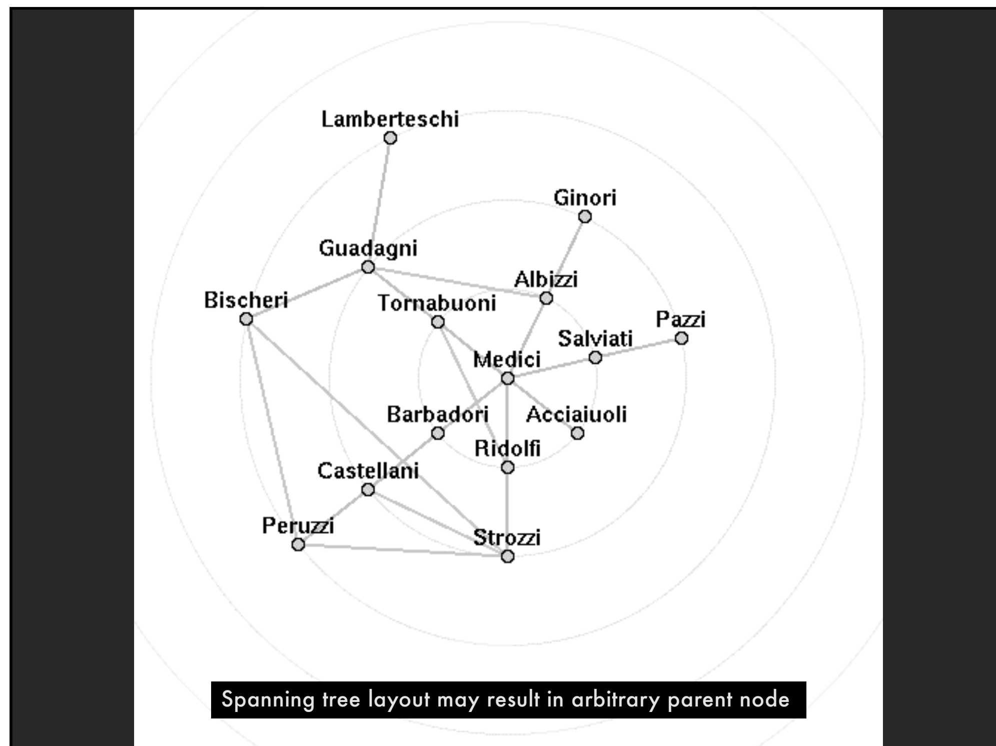
Trees created by BFS / DFS

Min/max spanning trees

Fast tree layouts allow graph layouts to be recalculated at interactive rates

Heuristics may further improve layout

111



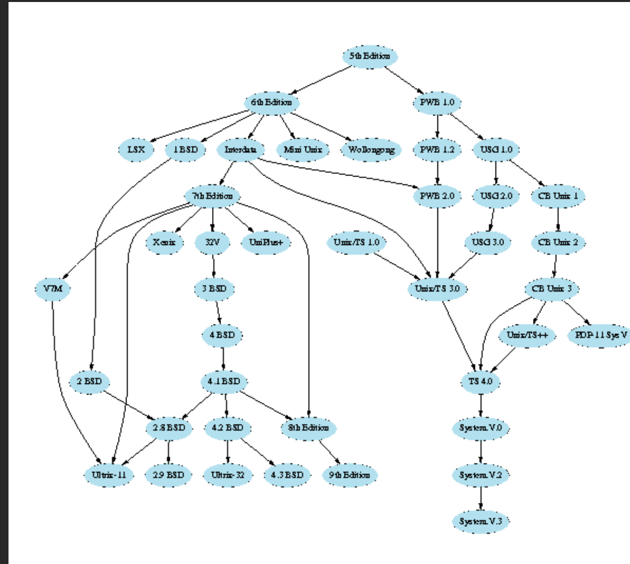
Spanning tree layout may result in arbitrary parent node

112

Sugiyama-style graph layout

Evolution of the UNIX operating system

Hierarchical layering based on descent



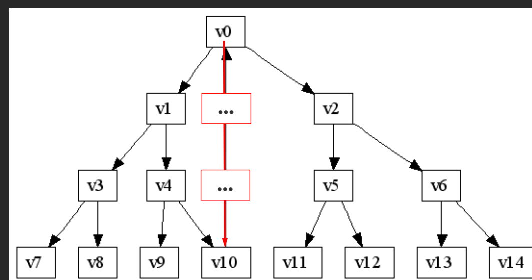
Sugiyama-style graph layout

Layer 1

Layer 2

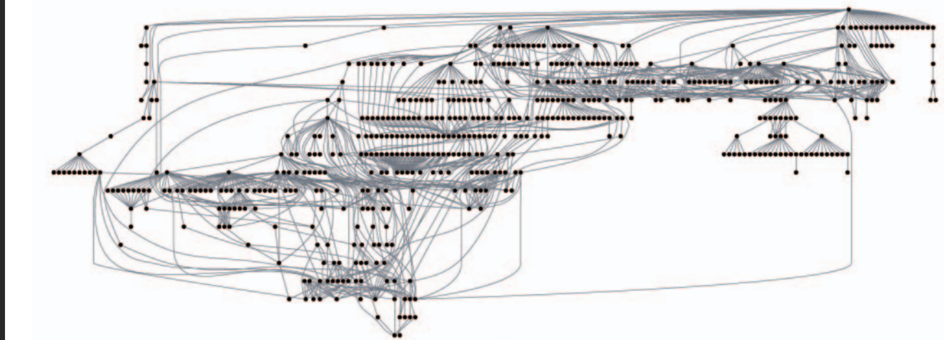
Layer 3

Layer 4



Reverse some edges to remove cycles
Assign nodes to hierarchy layers → Longest path layering
    Create dummy nodes to “fill in” missing layers
Arrange nodes within layer, minimize edge crossings
    Route edges – layout splines if needed

Produces hierarchical layout



Sugiyama-style layout emphasizes hierarchy

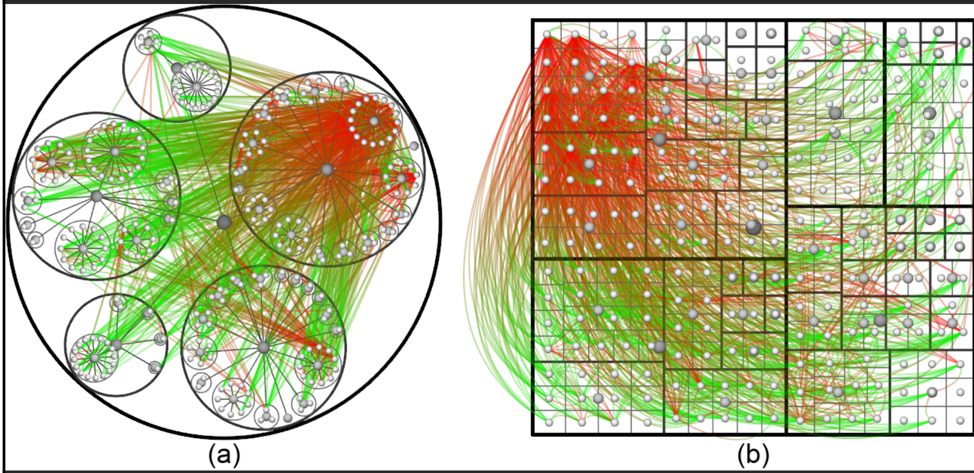
However, cycles in the graph may mislead.
Long edges can impede perception of proximity.

115

Hierarchical Edge Bundles

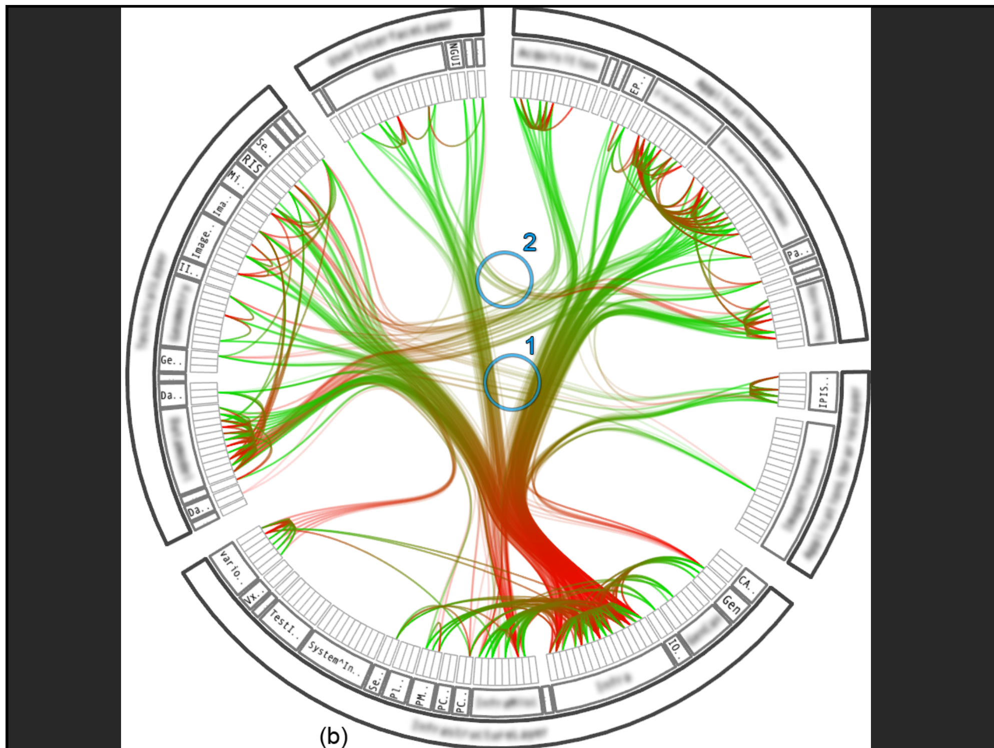
117

Trees with Adjacency Relations



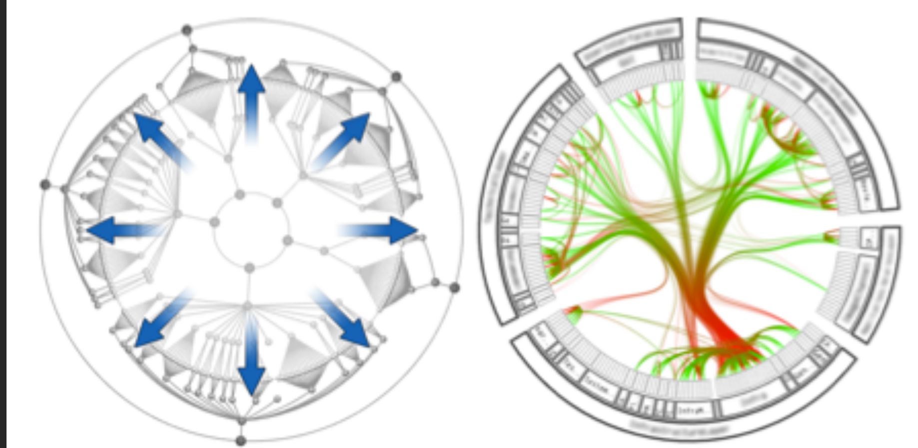
(a) (b)

118



(b)

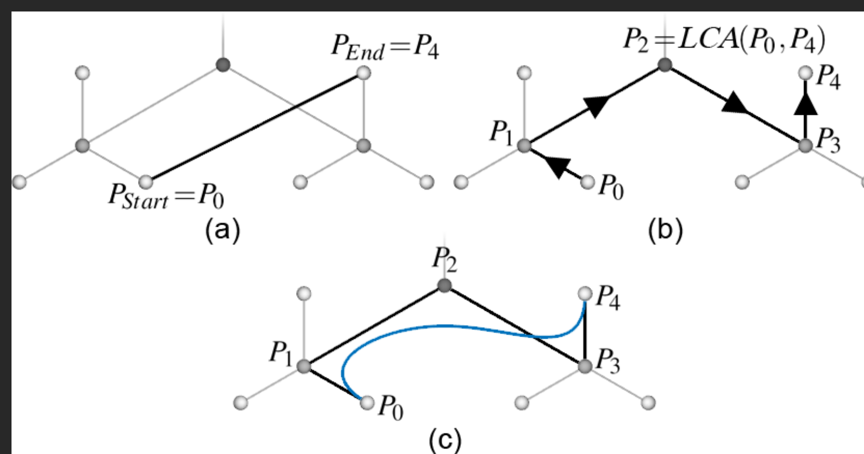
119



Use radial tree layout for inner circle
Mirror to outside
Replace inner tree with hierarchical edge bundles

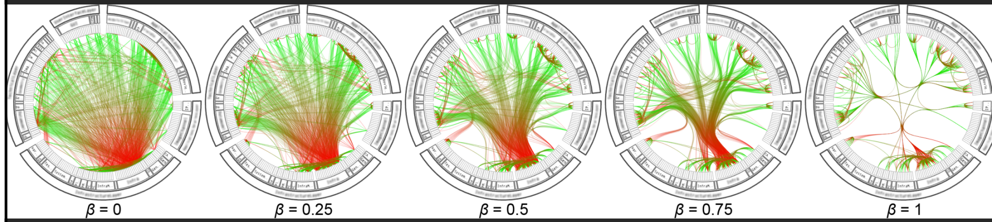
120

Bundle Edges along Hierarchy



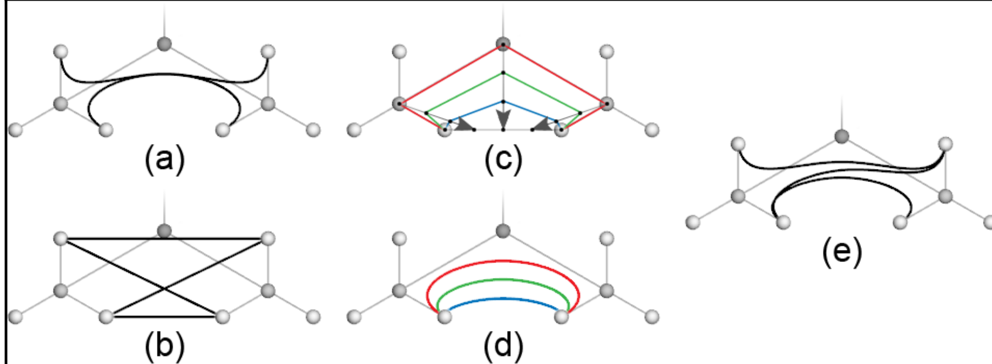
121

Increasing Edge Tension

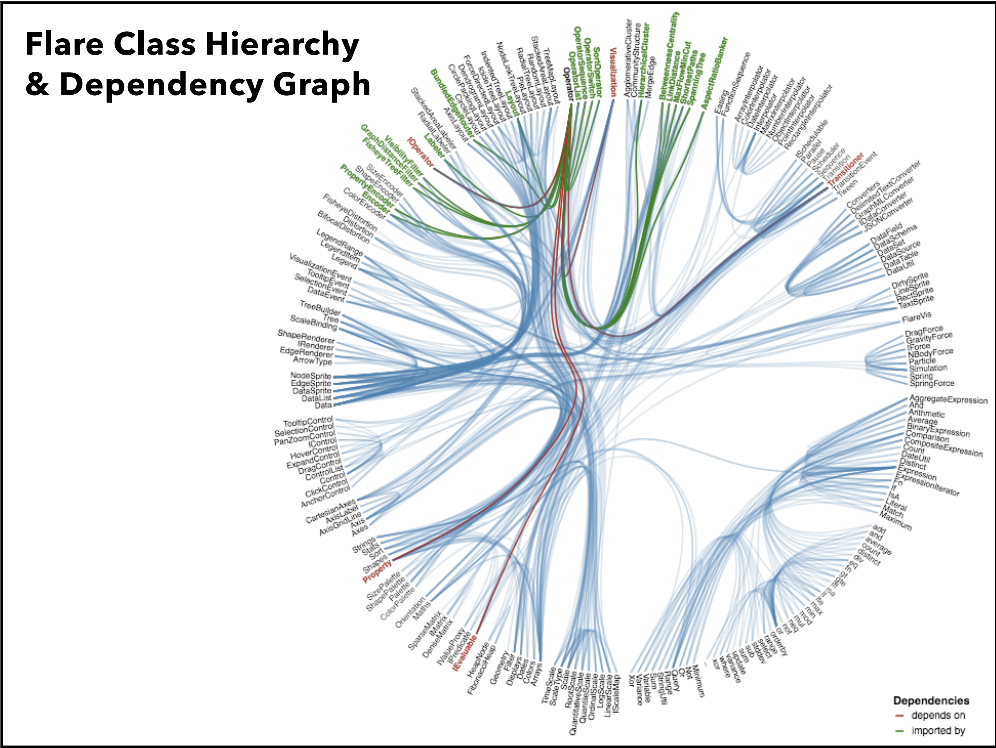


122

Configuring Edge Tension

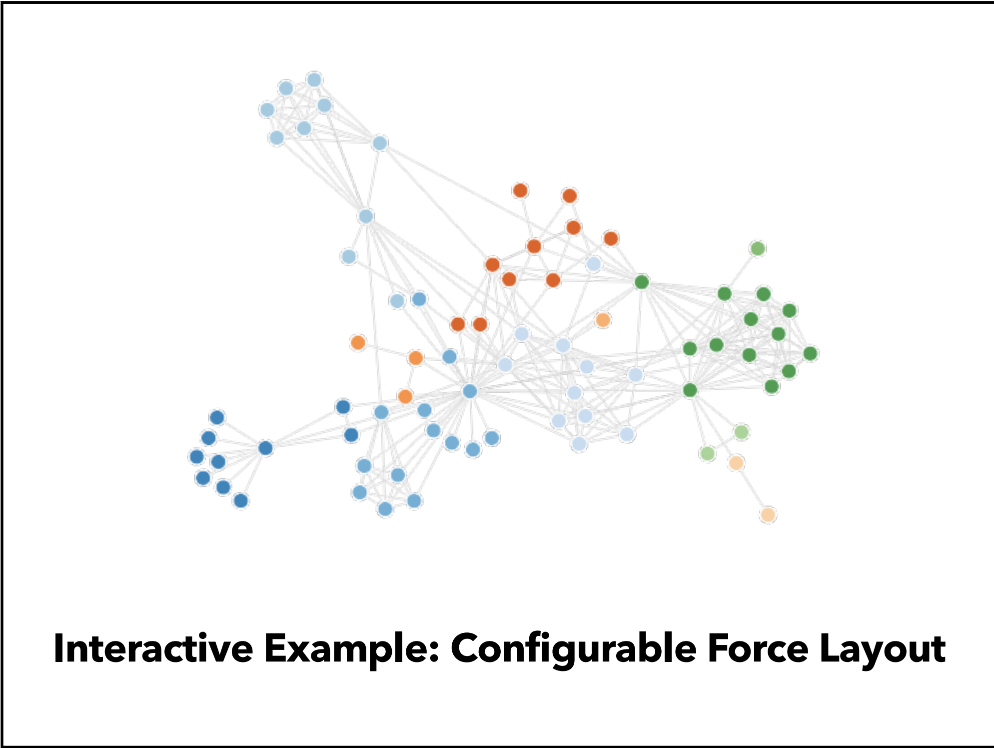


123

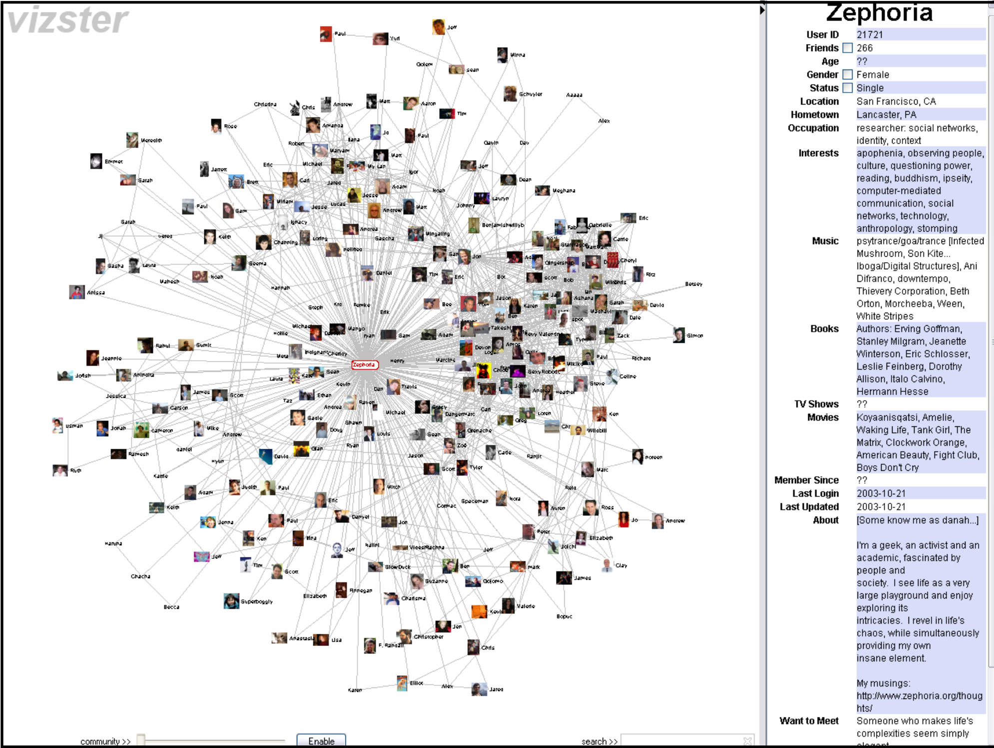


124

125



126

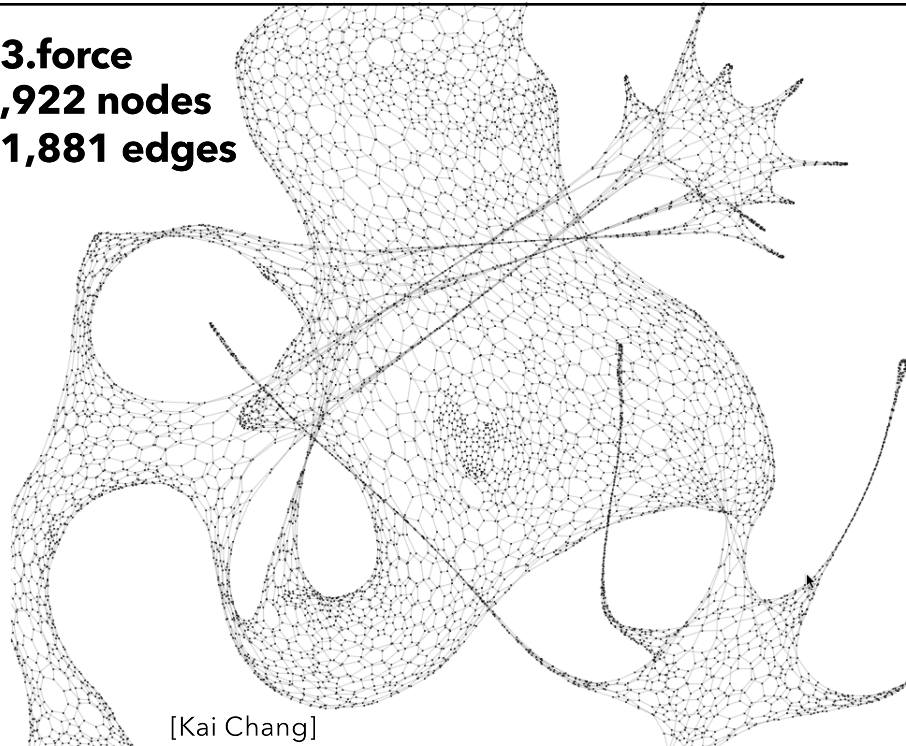


127

Use the Force!

http://mbostock.github.io/d3/talk/20110921/

128



129

Force-Directed Layout

Nodes = charged particles $F = q_i * q_j / d_{ij}^2$
with air resistance $F = -b * v_i$
Edges = springs $F = k * (L - d_{ij})$

D3's force layout uses velocity Verlet integration

Assume uniform mass *m* and timestep *Δt*:

F = ma → F = a → F = Δv / Δt → F = Δv

Forces simplify to velocity offsets!

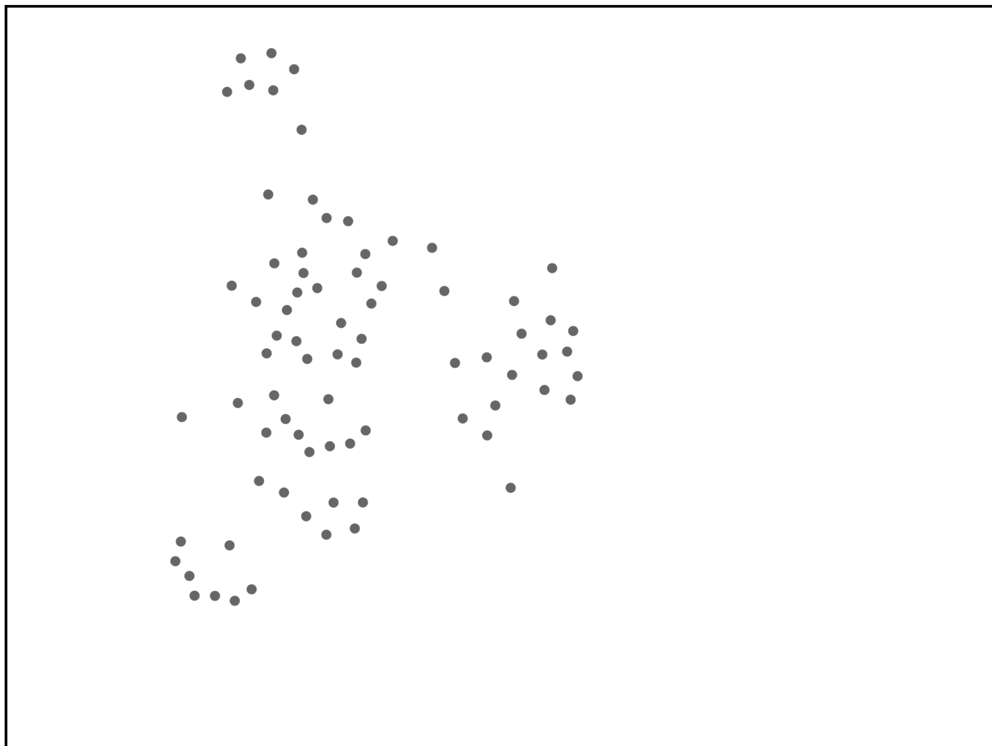
Repeatedly calculate forces, update node positions

Naïve approach $O(N^2)$

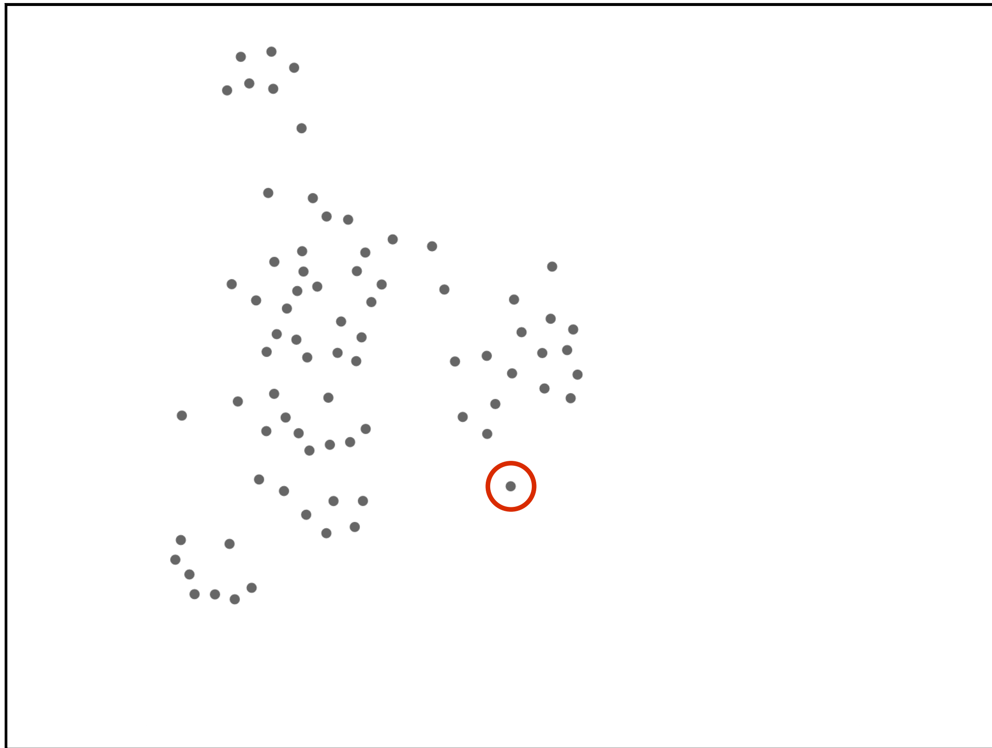
Speed up to O(N log N) using quadtree or k-d tree

Numerical integration of forces at each time step

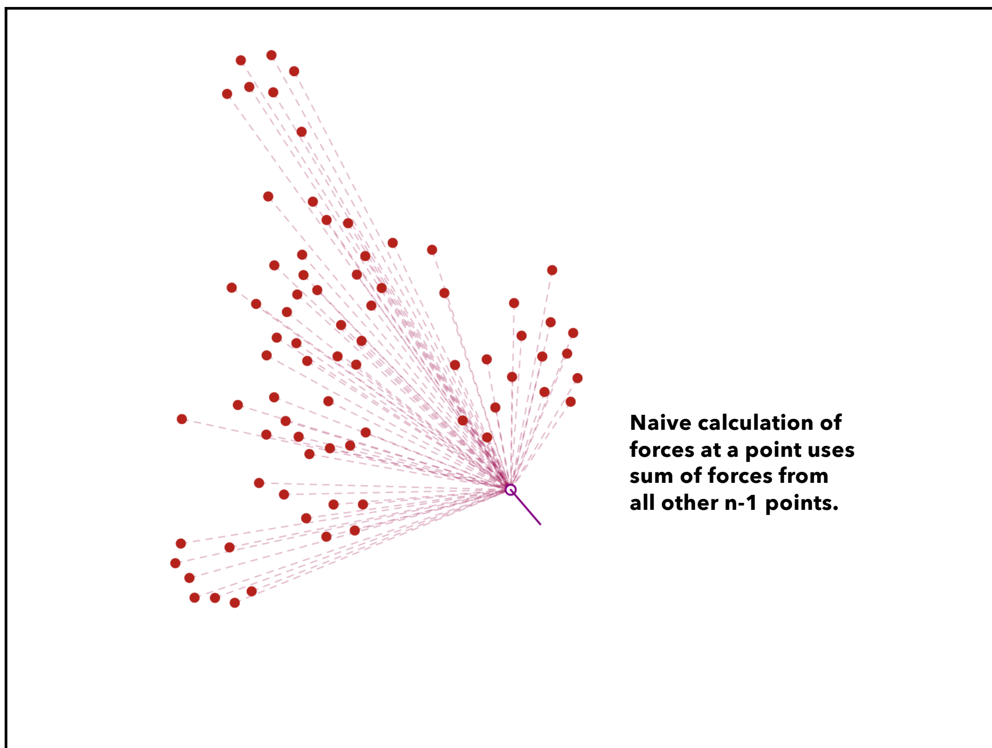
130



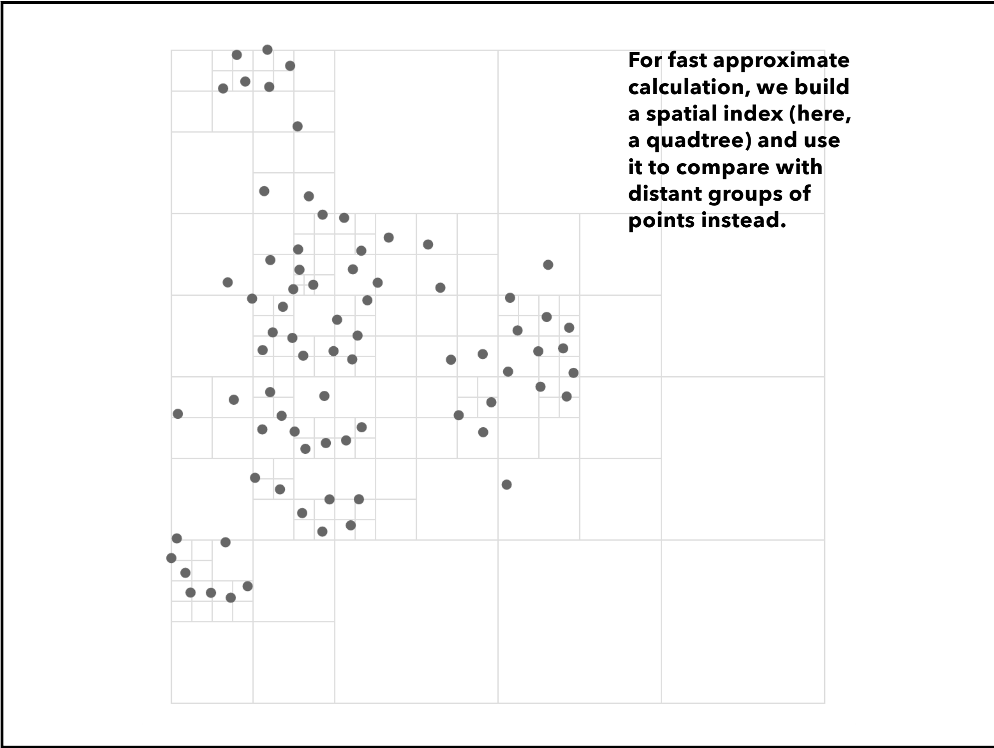
131



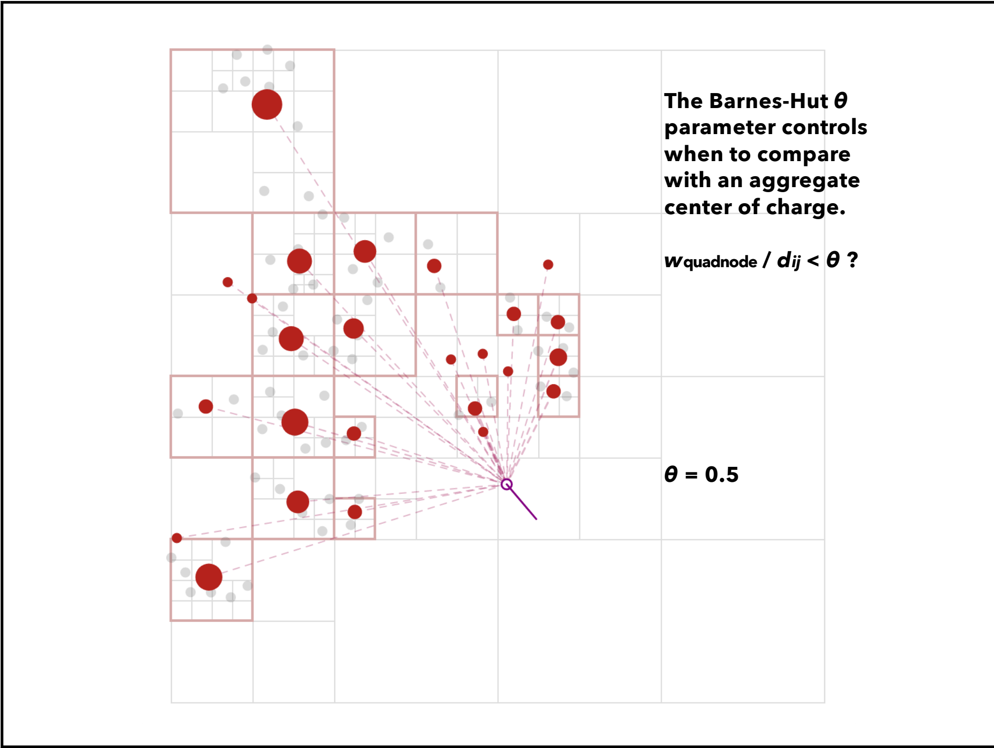
132



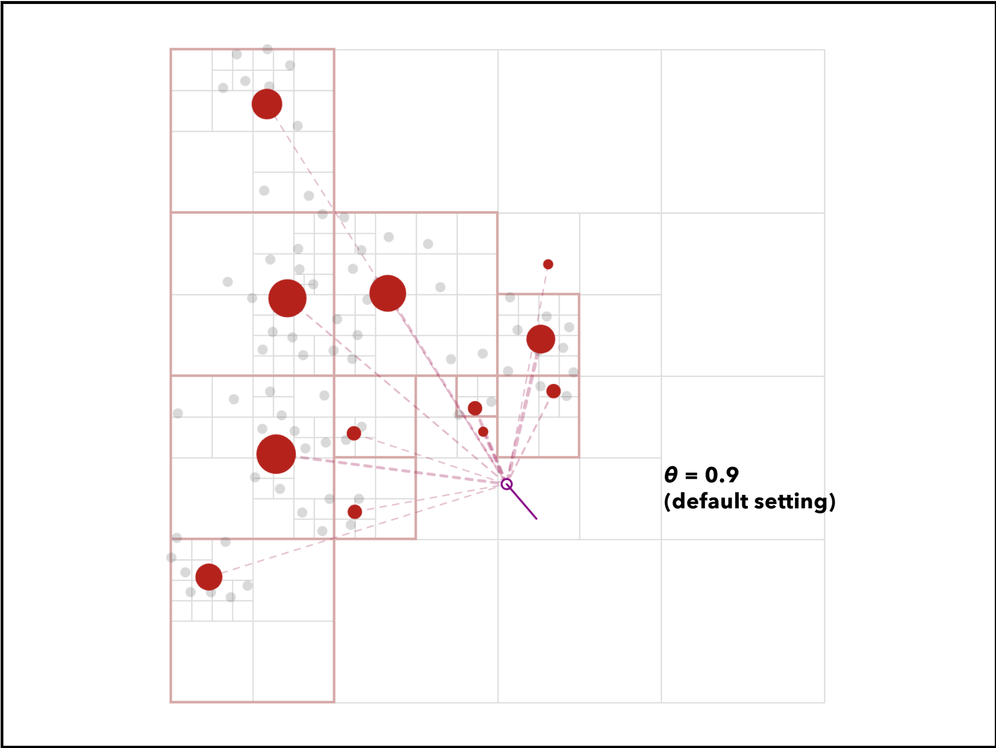
133



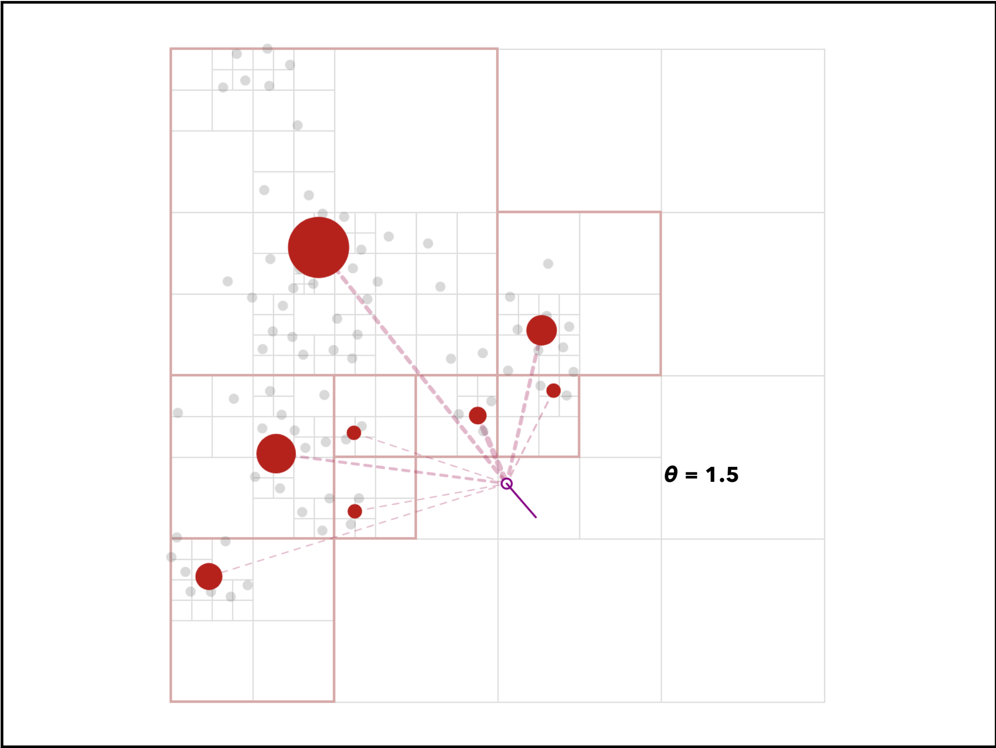
134



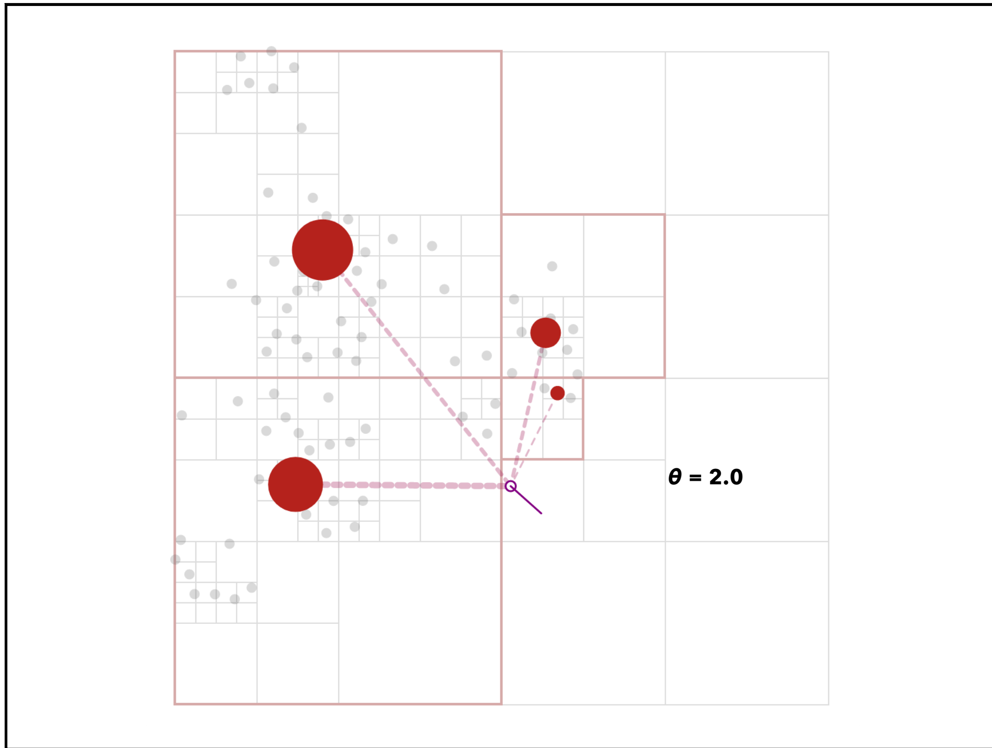
135



136



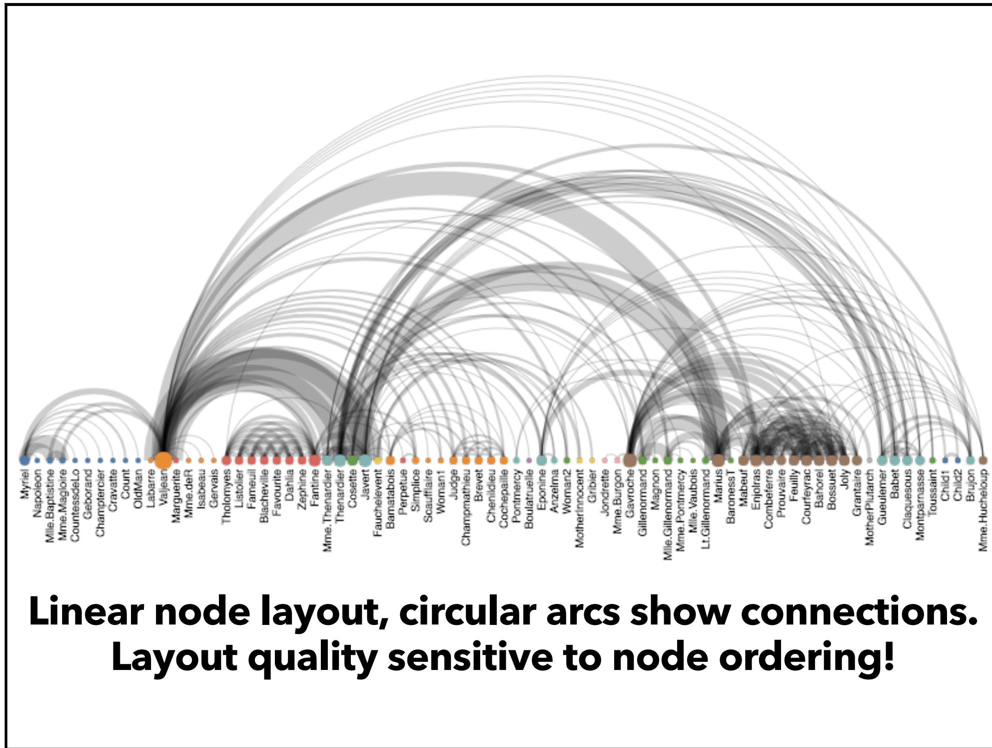
137



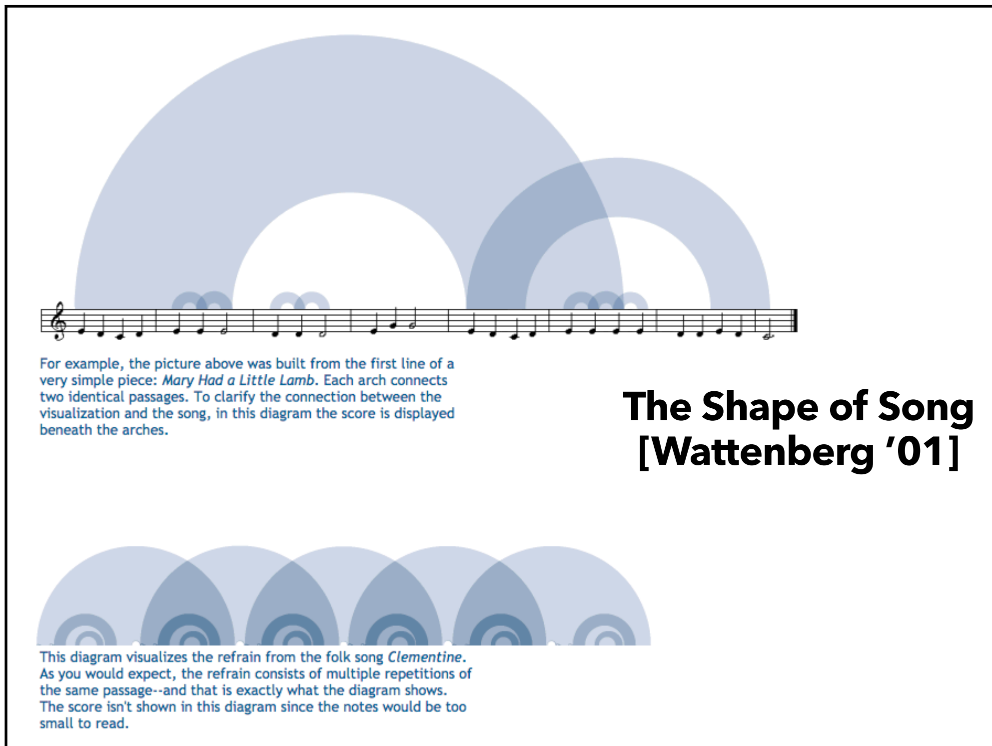
138

Alternative Layouts

140

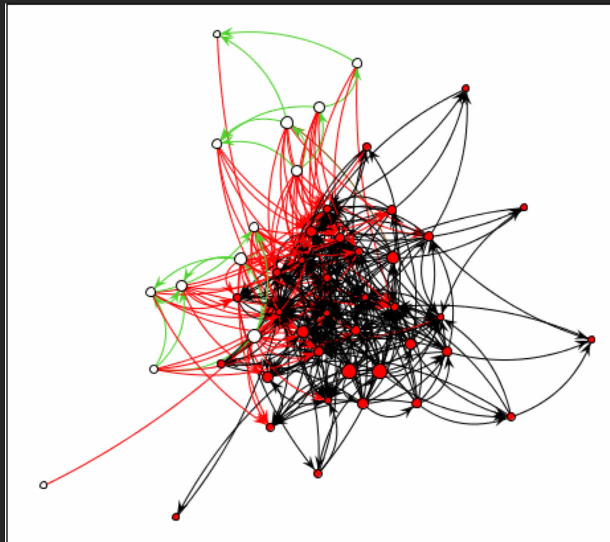


141



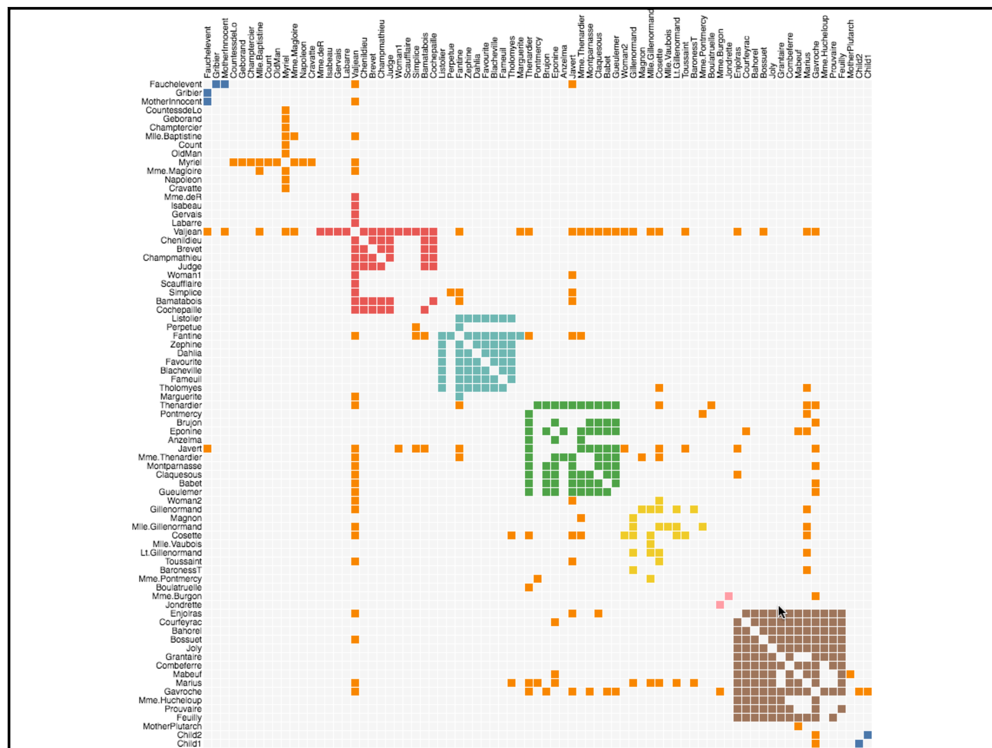
142

Limitations of Node-Link Layout



Edge-crossings and occlusion

143



146

Attribute-Driven Layout

Large node-link diagrams get messy!
Is there additional structure we can exploit?

Idea: Use data attributes to perform layout

- e.g., scatter plot based on node values

Dynamic queries and/or brushing can be used to explore connectivity

155

Attribute-Driven Layout

The "Skitter" Layout

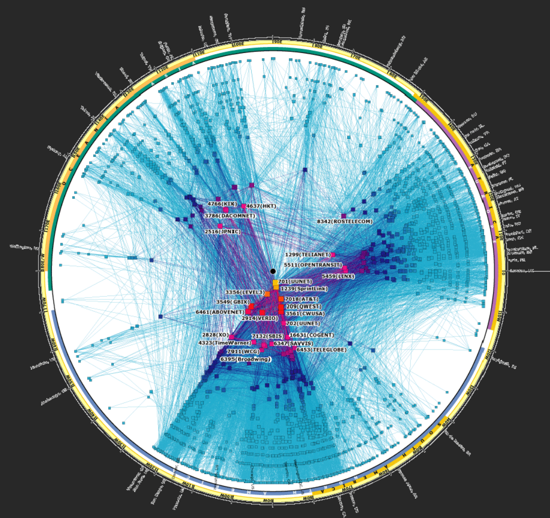
- Internet Connectivity
- Radial Scatterplot

Angle = Longitude

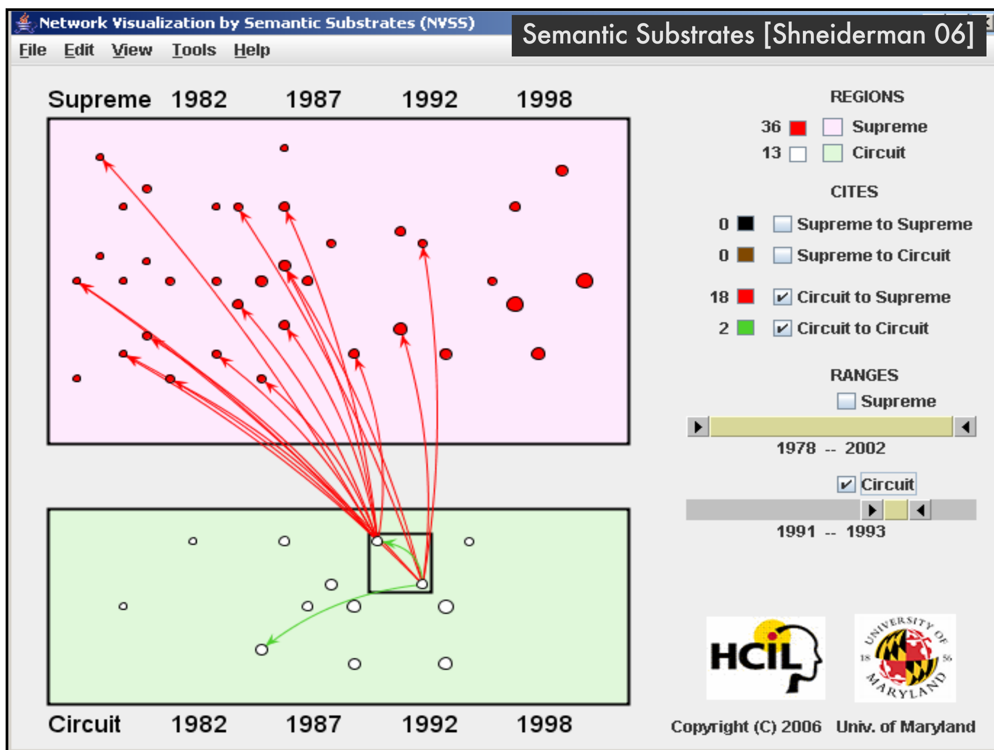
- Geography

Radius = Degree

- # of connections
- (a statistic of the nodes)



156



157

Summary

Tree Layout

Indented / Node-Link / Enclosure / Layers

How to address issues of scale?

- Filtering and Focus + Context techniques

Graph Layout

Tree layout over spanning tree

Hierarchical "Sugiyama" Layout

Optimization (Force-Directed Layout)

Attribute-Driven Layout

164