Trees

COMP8503

Visualization & Visual Analytics

Trees

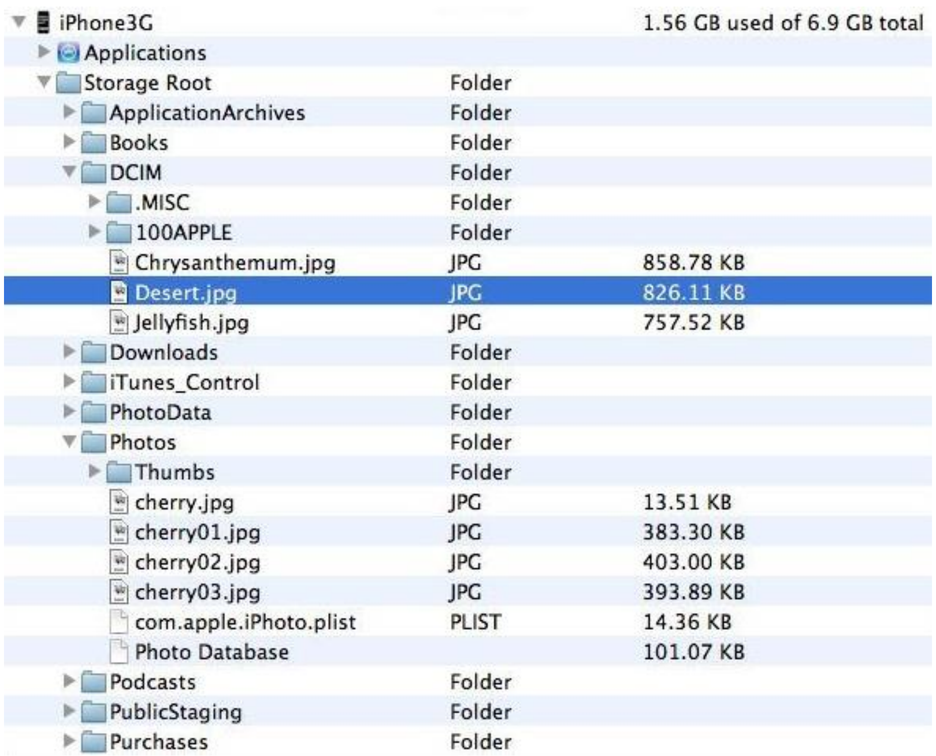
- A **tree** is a directed acyclic graph:
  - Exactly one unique vertex called the root with no parents
  - Every vertex except the root has a parent
  - There is a path from the root to each vertex
- Trees are good for representing:
  - Hierarchies (file systems, web sites, organization charts)
  - Branching Processes (family lineage, evolution)
  - Decision processes (search trees, game trees, decision trees)

Tree Layouts

- Indentation
  - Tree depth is encoded by indentation
- Node-Link Diagrams
  - Nodes connected by lines/curves
- Layered Diagrams
  - Hierarchical structure represented by layering, adjacency or alignment
- Enclosure/Containment Diagrams
  - Hierarchical structure represented by enclosure

- Tree layout can be done efficiently in O(n) or O(n log n) time

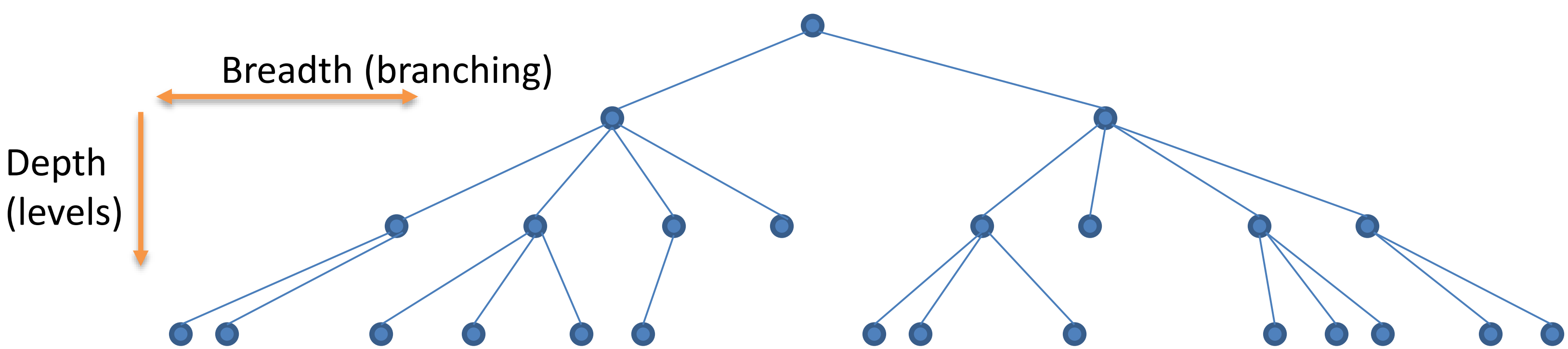
Indentation



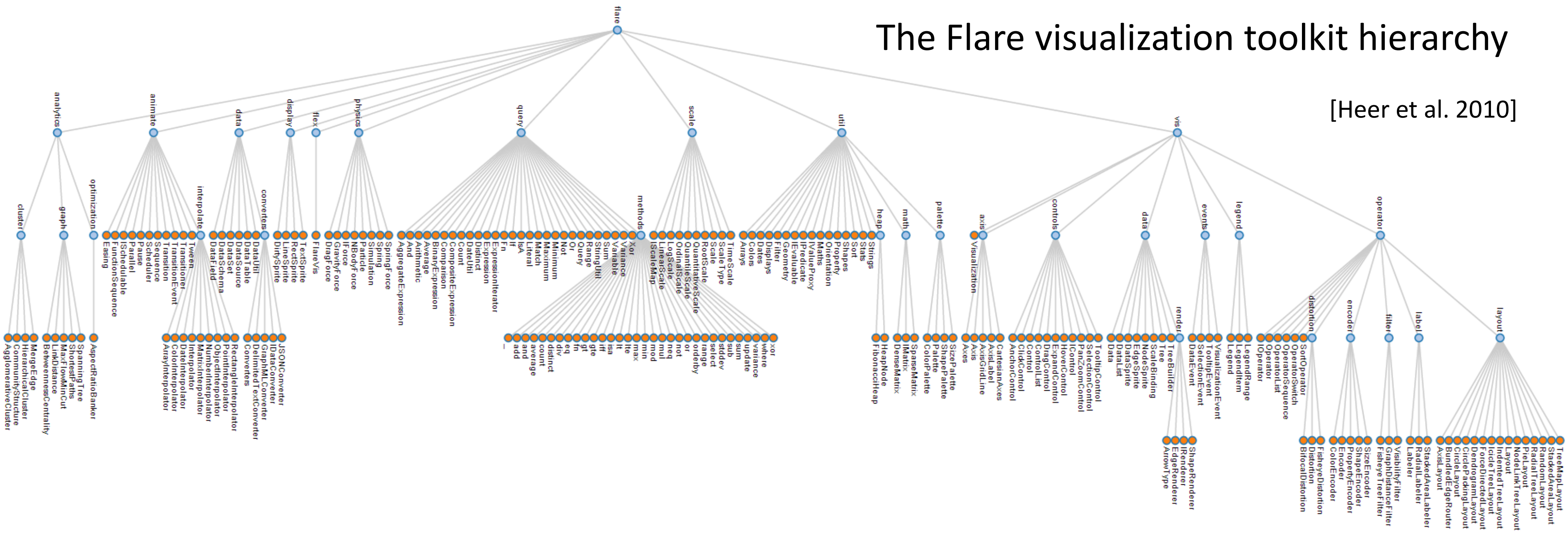
- All items are placed in rows
- Indentation is used to show parent/child relationship
- Multivariate data can be displayed
- Takes up excessive vertical space

Node-Link Diagrams

- Nodes are distributed in space, with straight or curved lines connecting them
- Breadth and depth of a tree are layout in different directions/dimensions in 2D
- Problem: breadth may grow exponentially when branching factor is large



Reingold-Tilford Algorithm

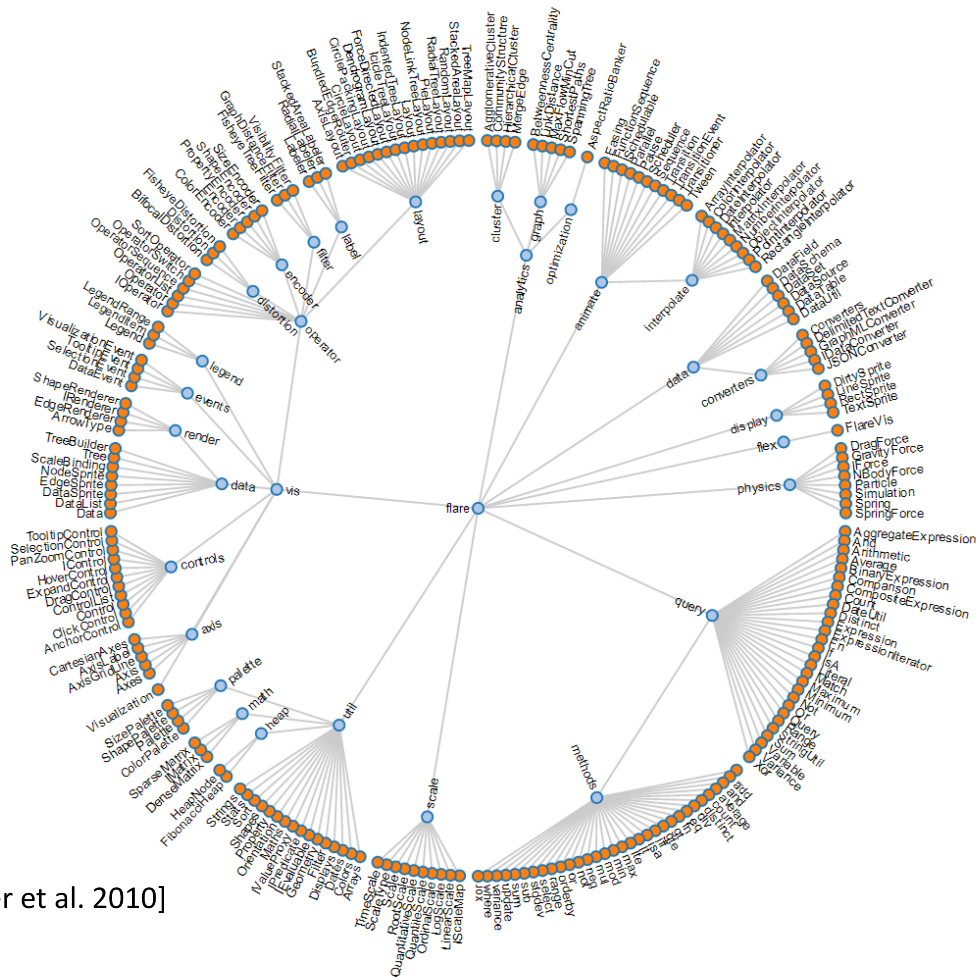


The Flare visualization toolkit hierarchy

[Heer et al. 2010]

- A classical layout in which children nodes are positioned under their common parent
- Same drawing for isometric subtrees and no edge crossing
- Leave nodes are at different levels

Radial Layout

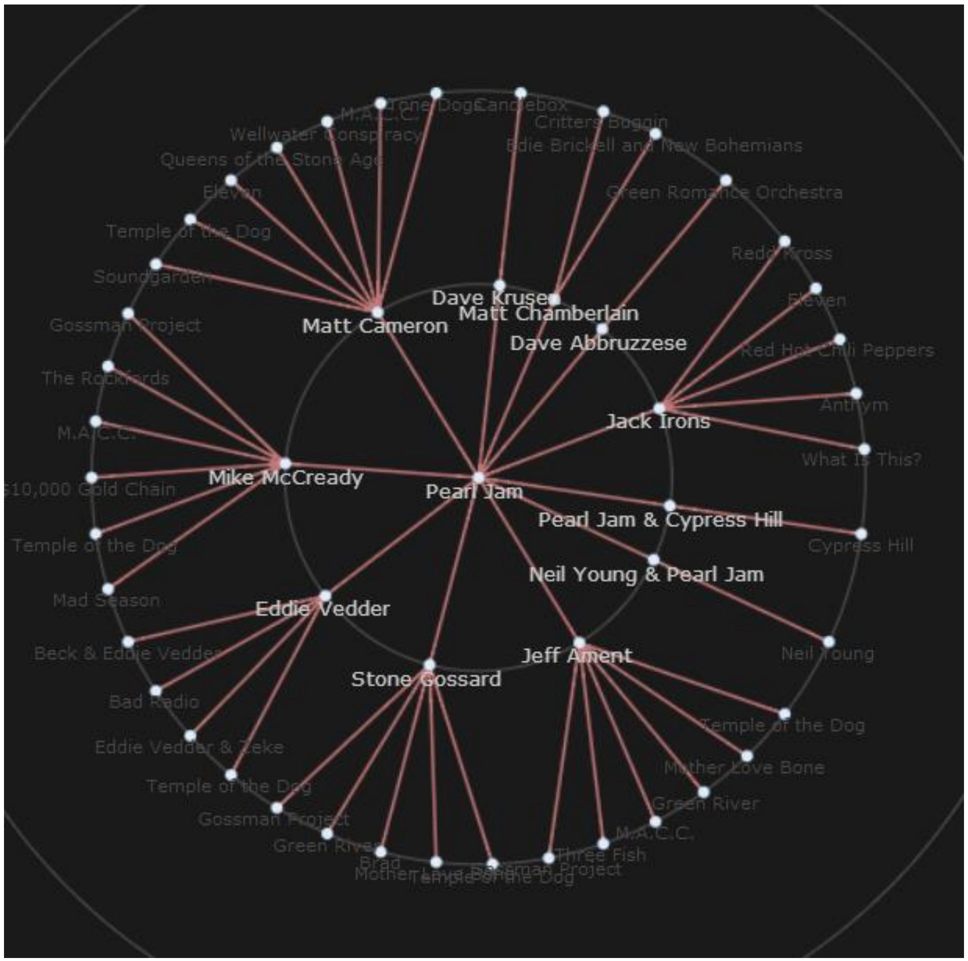


[Heer et al. 2010]

The Flare visualization toolkit hierarchy

- Node-link diagram but in polar co-ordinates
- Leaf nodes are all at the circumference (dendrogram)
- Root node at the center
- Visually more pleasing and spatially more efficient

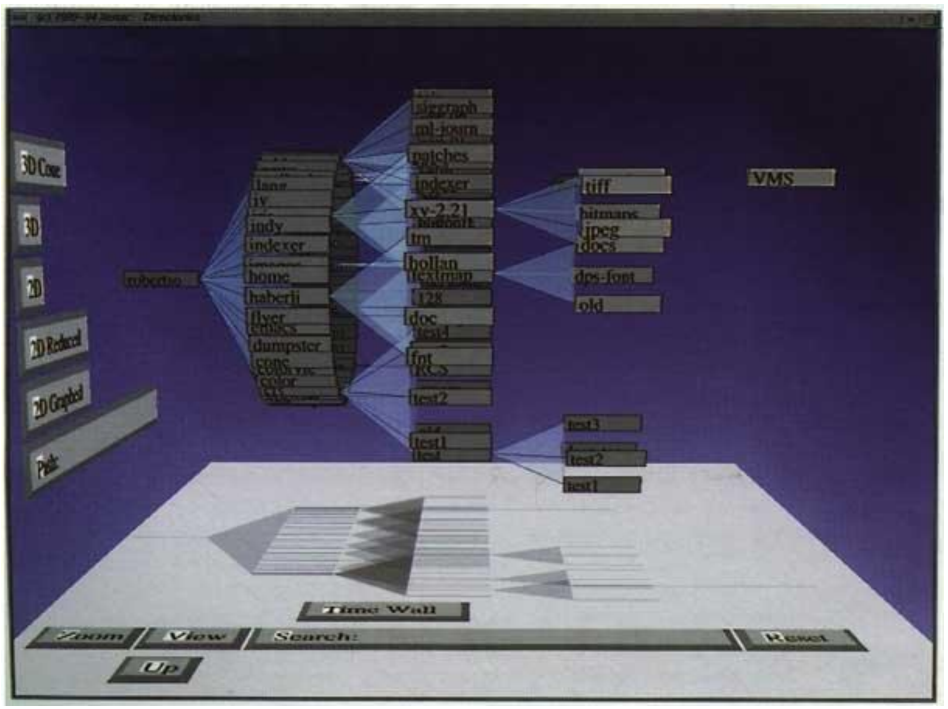
Radial Layout



An animated radial tree layout with root change

[http://philogb.github.io/jit/static/v20/Jit/Examples/RGraph/example1.html]

Cone Trees

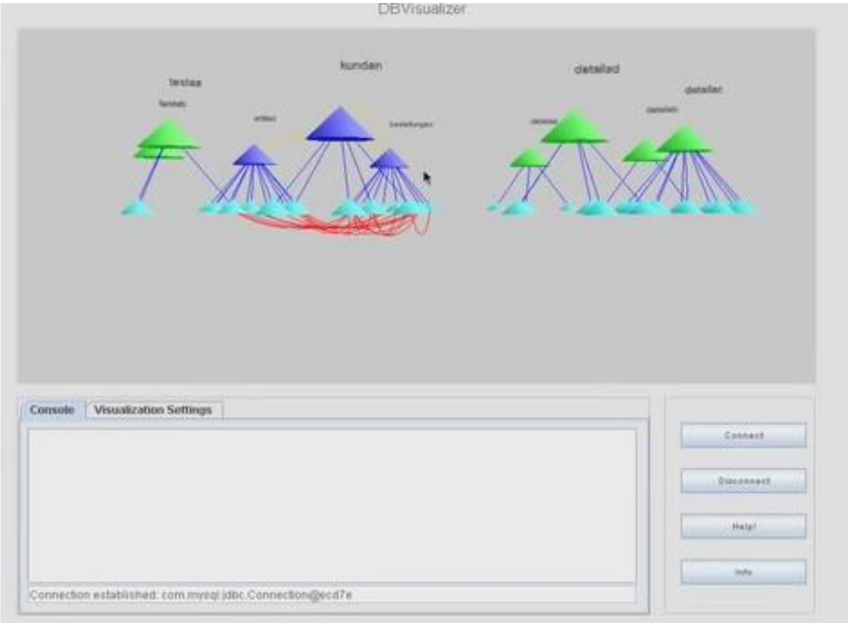


Robertson et al., "*Cone Trees: Animated 3D Visualizations of Hierarchical Information*", CHI'91.

- Make use of more space with 3D, i.e., an additional dimension
- Nodes at apex of a cone
- Children evenly spaced along the cone base

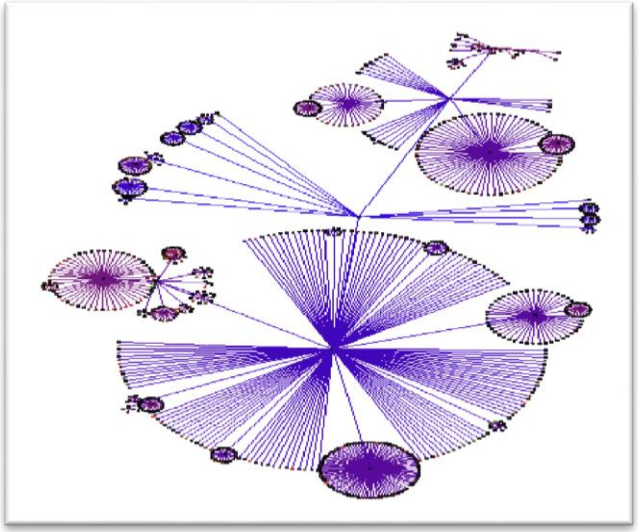
Cone Trees

- An example



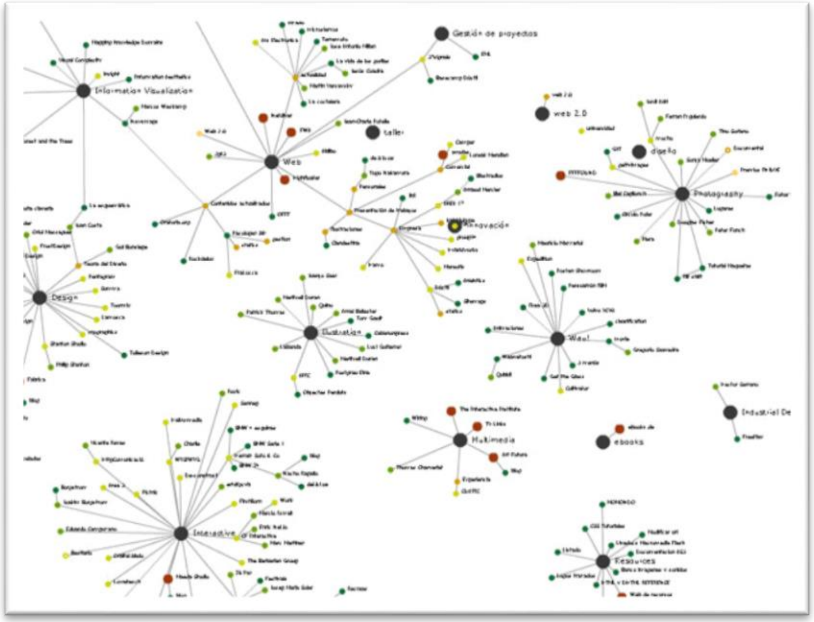
[https://www.youtube.com/watch?v=1eO1pgTVu-g]

Balloon Trees



Carriere and Kazman, "Research Report: Interacting with Huge Hierarchies: Beyond Cone Trees", *Proc. of Information Visualization '95.*

- Children are positioned around their parent
- Like flattening a cone tree, but without overlapping among circles.



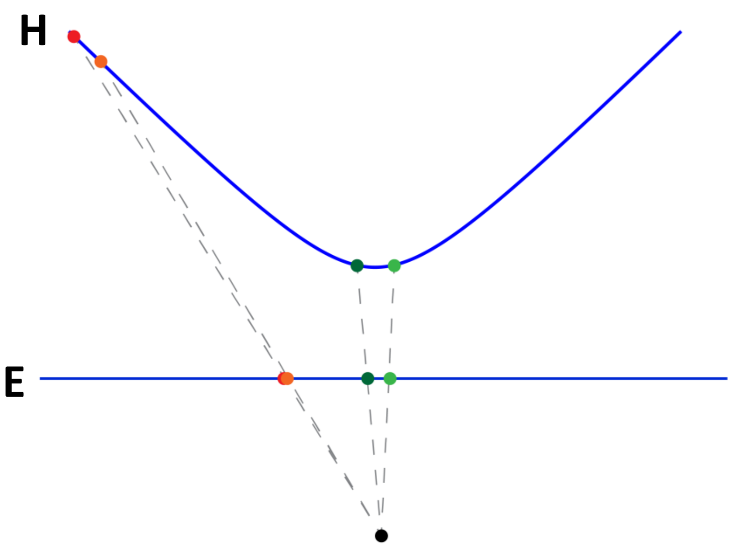
[http://www.webdesignerdepot.com/2009/06/50-great-examples-of-data-visualization/]

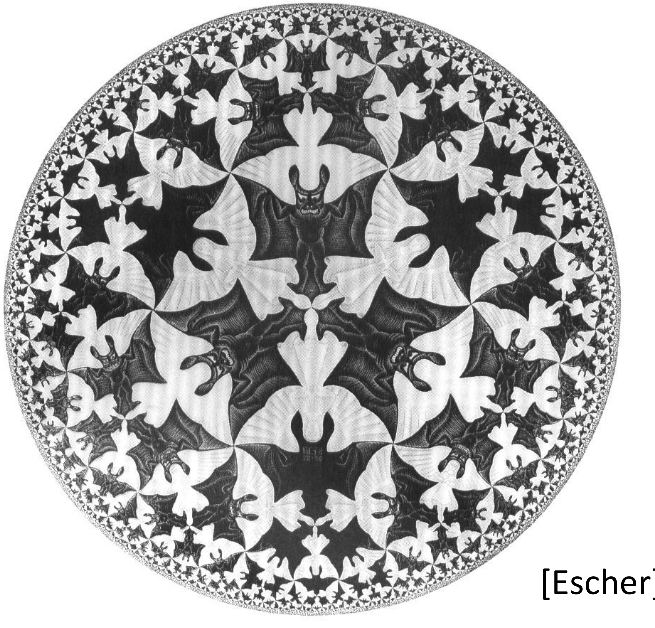
Problems of Node-Link Diagrams

- Tree breadth may grow exponentially
- Easily run out of space, even with space efficient layout
- Solution:
  - Filtering
  - Clustering / Aggregation
  - Interactions, e.g., focus + context, scroll & pan, zoom
  - Distortion (use different aspect ratios at different regions)

Hyperbolic Layout

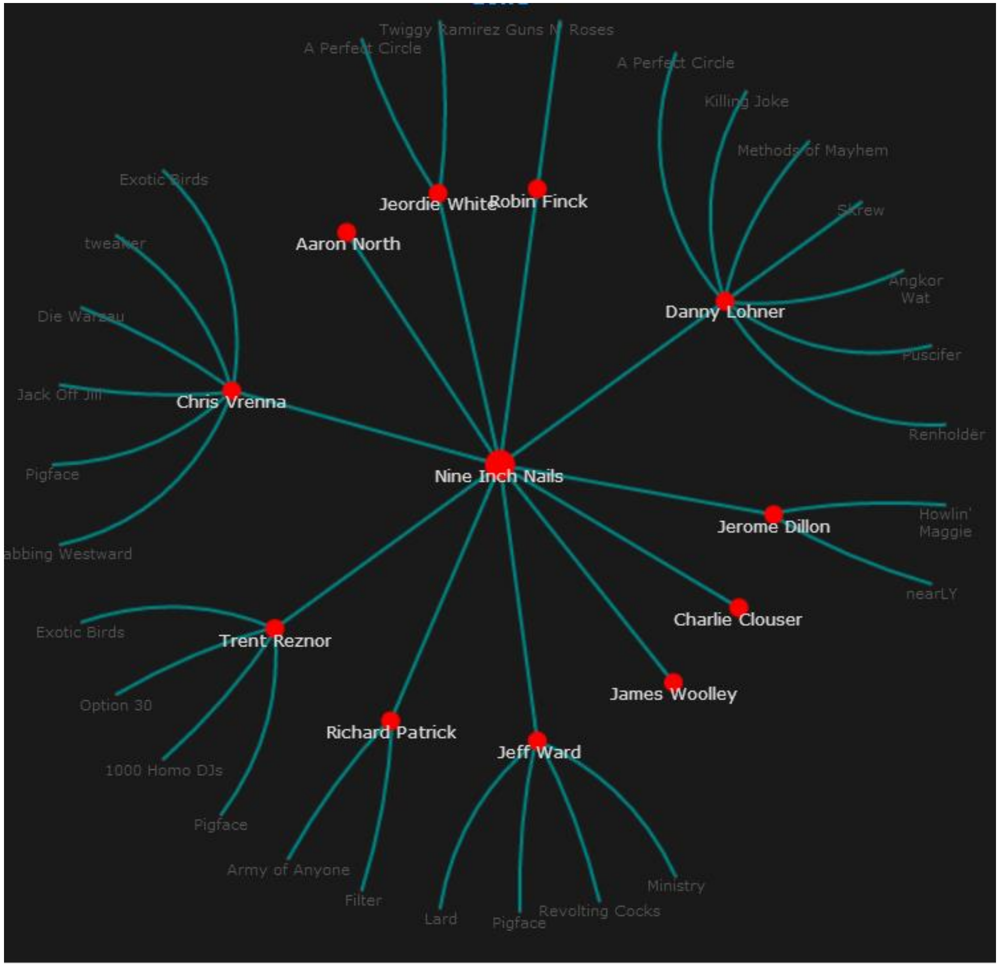
- A distorted view of a tree
- Perform a layout in the hyperbolic plane, then display the results in the Euclidean plane





[Escher]

Hyperbolic Layout



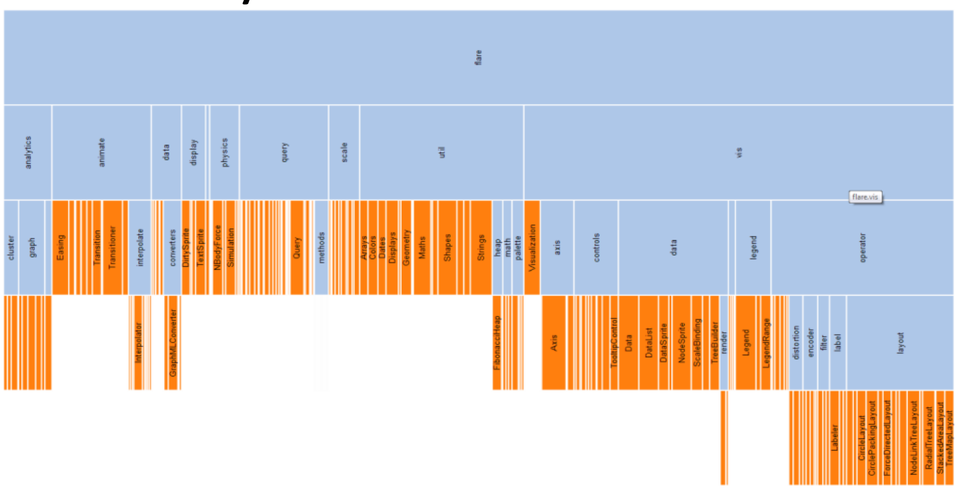
[http://philogb.github.io/jit/static/v20/Jit/Examples/Hypertree/example1.html]

Layered Diagrams

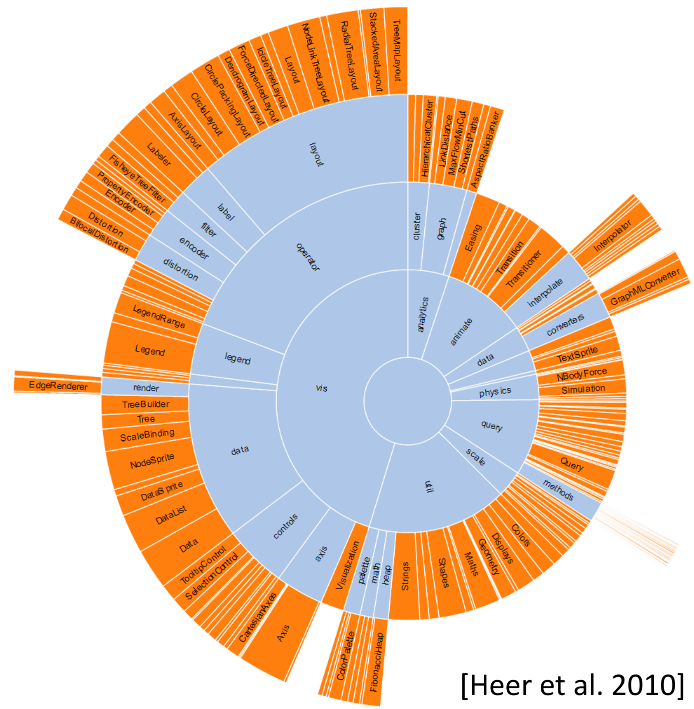
- Hierarchical structure represented by
  - Layering
  - Adjacency
  - Alignment
- Space-filling variant of the node-link diagrams
- Nodes are drawn as solid areas with lengths encoding an additional dimension
- Relative position of nodes implies node relationships
- Examples: Icicle layout, Sunburst layout

Layered Diagrams

Icicle Layout



Sunburst Layout

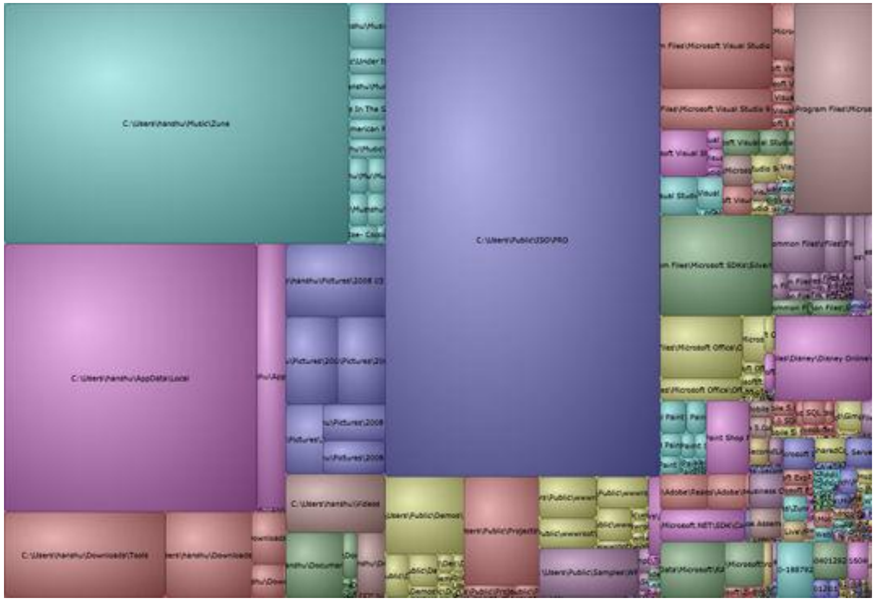


The Flare visualization toolkit hierarchy

[Heer et al. 2010]

Enclosure/Containment Diagrams

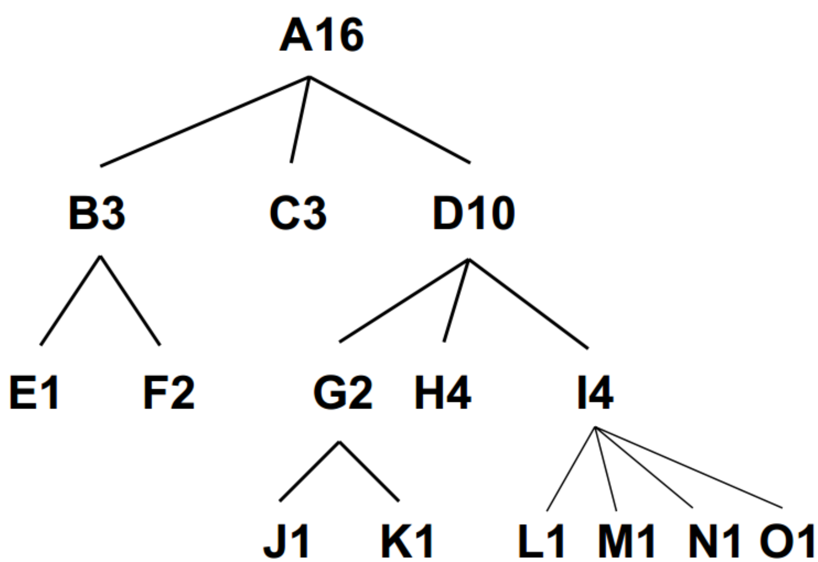
- Space filling and makes maximal use of the display space
- Relations are represented by enclosure or containment of shapes, e.g., rectangles / circles
- Size of shapes can be used to show attributes, e.g., file size
- Colors are used to show attributes or hierarchical relationships
- Examples:
  - TreeMaps and its variants

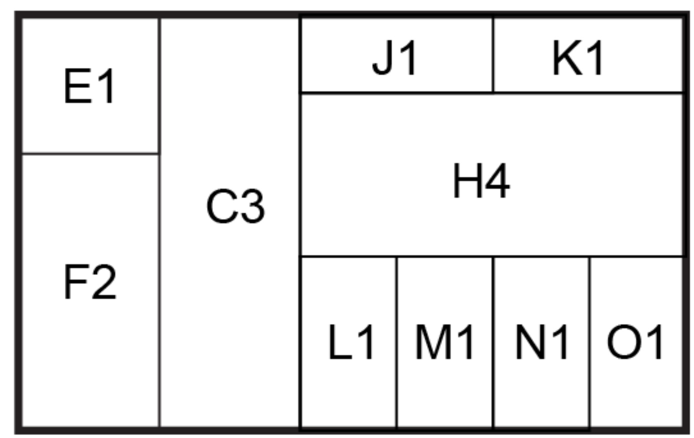


Treemaps

- A compact representation by subdividing an area into rectangles recursively
- The direction of subdivision alternates per level
- Advantages:
  - Gives a single overview of a tree
  - Large or small nodes are easily identified
- Problems:
  - Difficult to perceive hierarchical structure

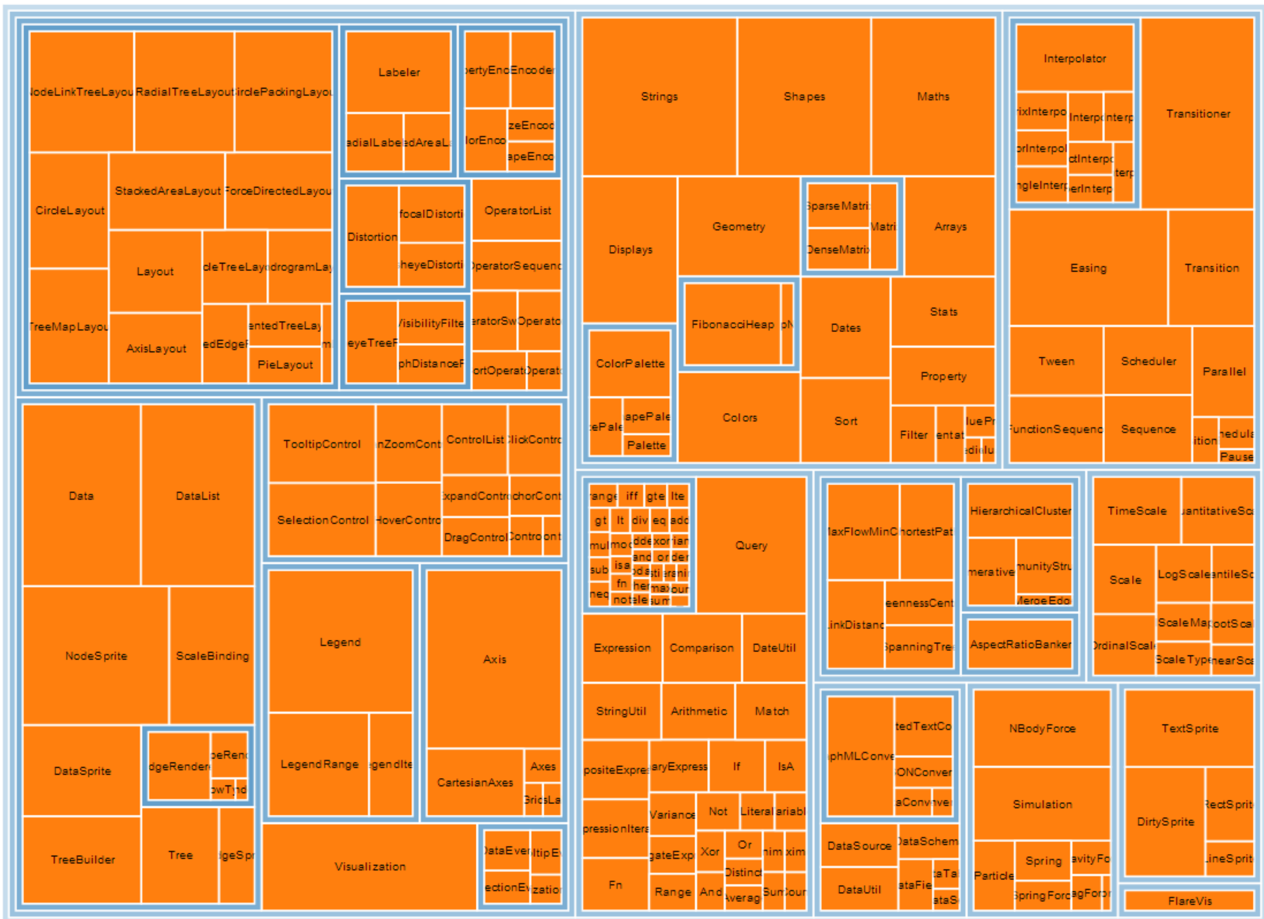
Treemaps





Johnson and Shneiderman, “Tree-Maps: a space-filling approach to the visualization of hierarchical information structures”, VIS '91.

Treemaps

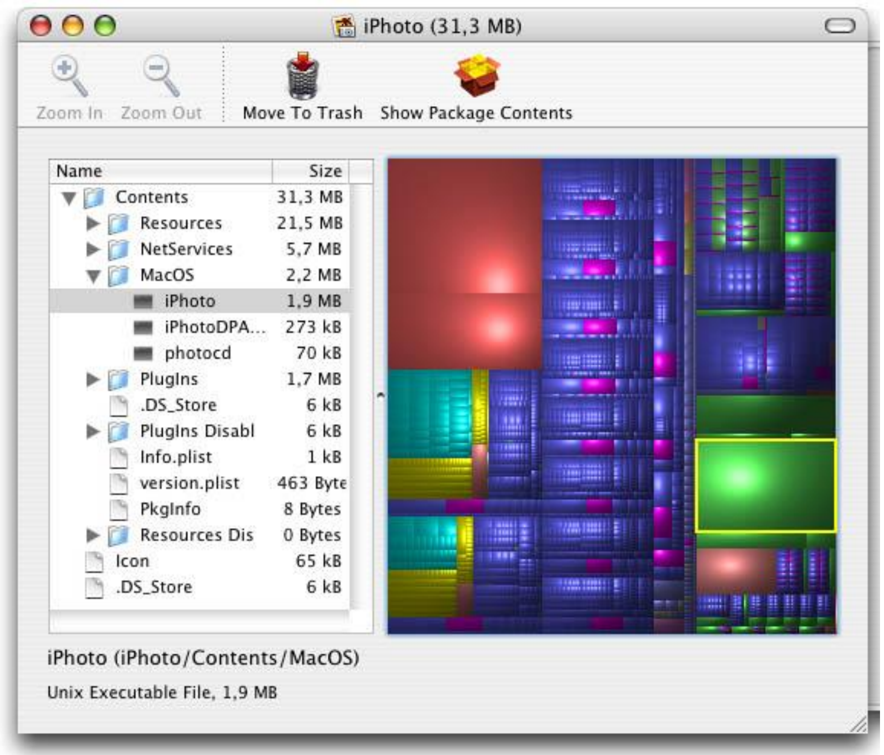


Use padding to emphasize enclosure

[Heer et al. 2010]

Cushion Treemaps

- Use shading to emphasize the hierarchical structure
- Resulting visualization is a surface consisting of recursive cushions



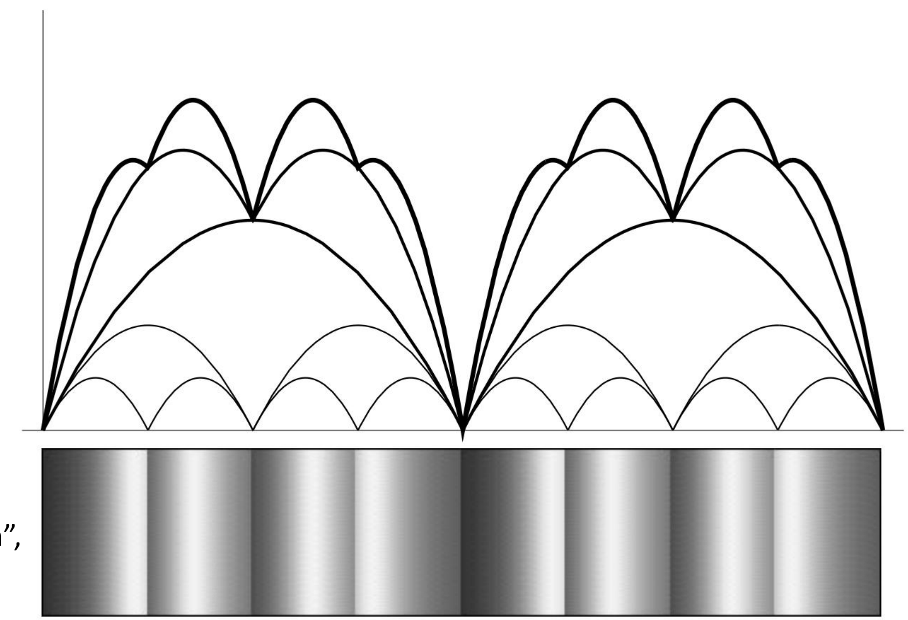
- Advantages:
  Easy to perceive structure
- Problems: Emergence of elongated rectangles

Disk Inventory X (http://www.derlien.com)

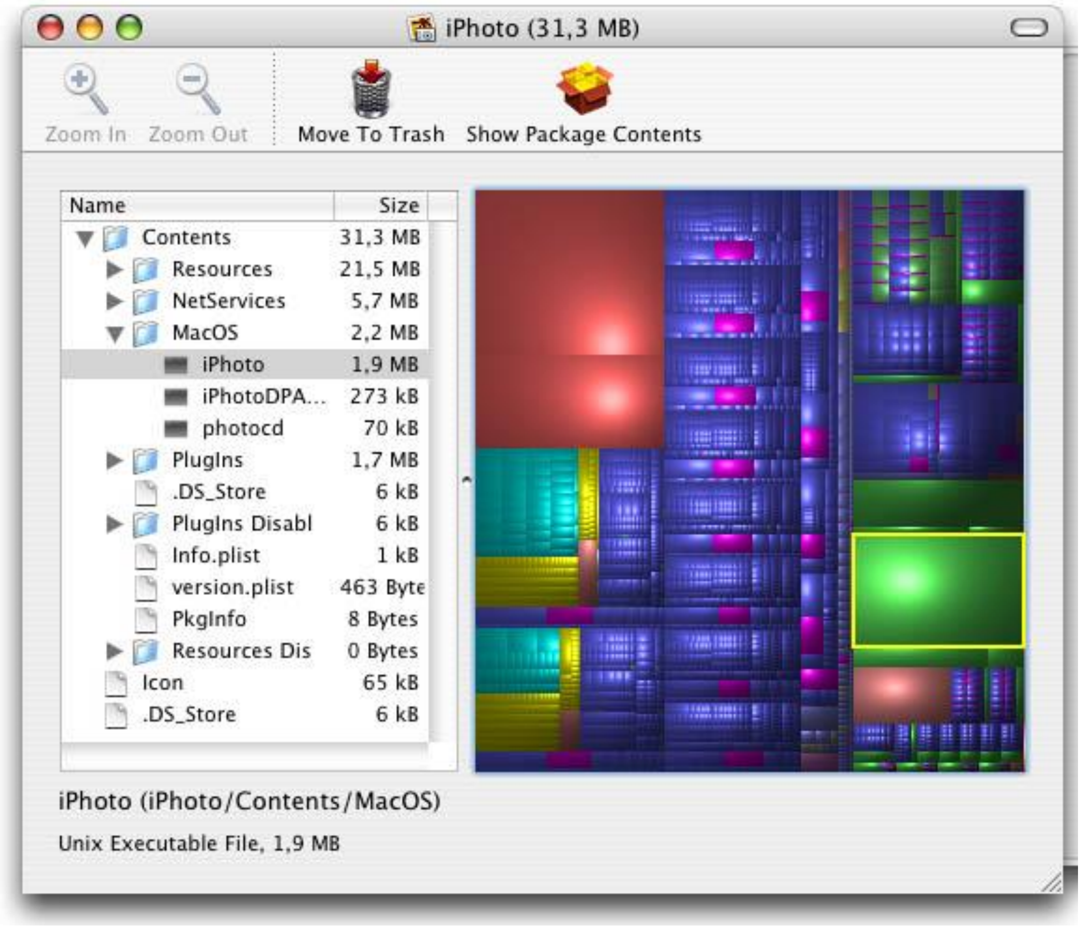
Cushion Treemaps

- Subdivide the interval and add a bump to each of the two subintervals. Do this recursively.
- Parabolic functions are used to make the ridges look like cushions.

Van Wijk and van de Wetering,
"Cushion Treemaps: Visualization of Hierarchical Information",
INFOVIS '99.



Cushion Treemaps



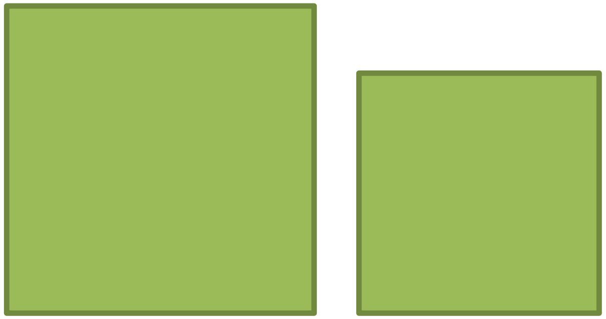
- Advantages: Easy to perceive structure
- Problems: Emergence of elongated rectangles

Disk Inventory X (http://www.derlien.com)

Rectangles vs. Squares?

- Which is easier to compare, in terms of size?

Which one is bigger?

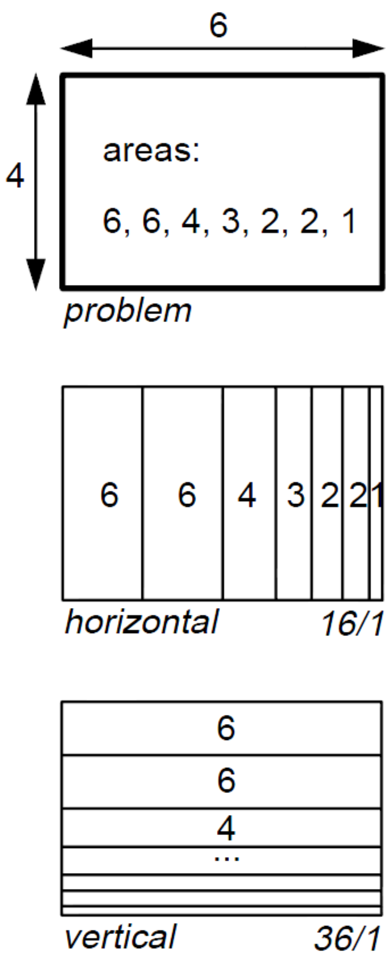


Which one is bigger?

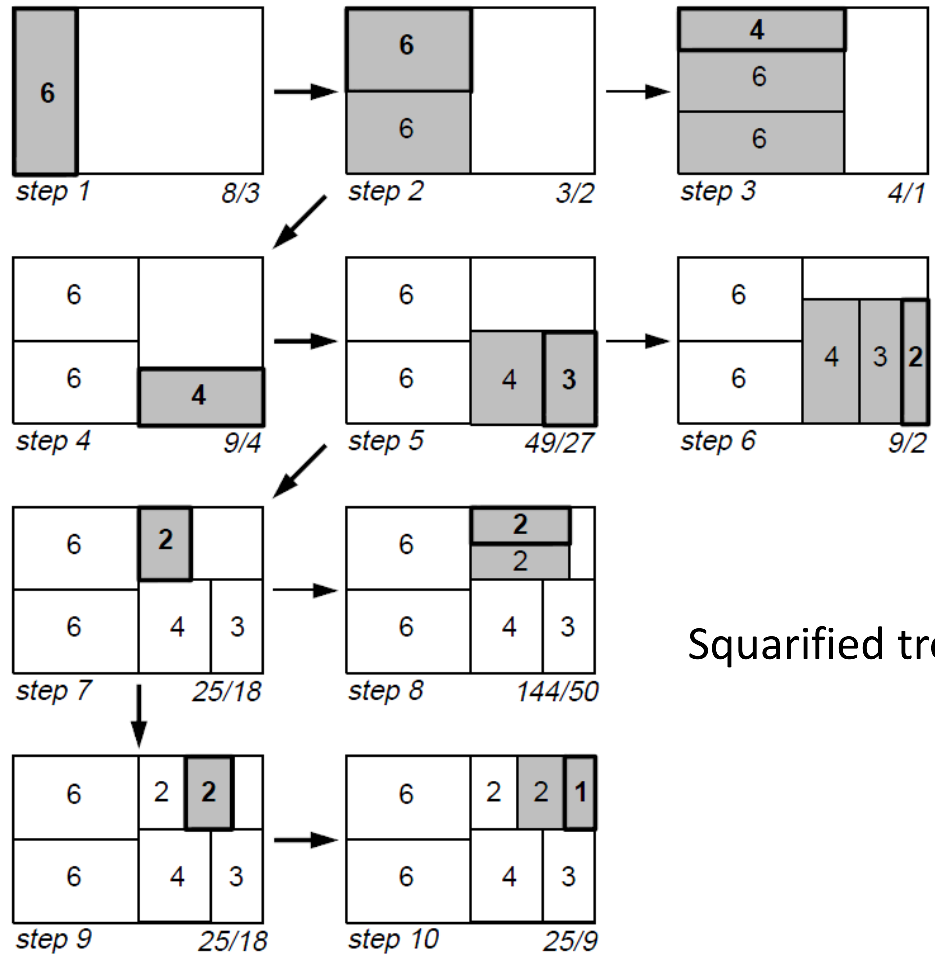
Squarified Treemaps

- We can compare better the size of rectangles when their aspect ratios are similar.
- Instead of subdividing area in a single dimension at each level, can choose both horizontal and vertical splits. Decision depends on whether adding a rectangle to a row will improve the layout (aspect ratio) or not.
- Take a greedy approach to process large rectangles first, in order to achieve better aspect ratios.

Squarified Treemaps



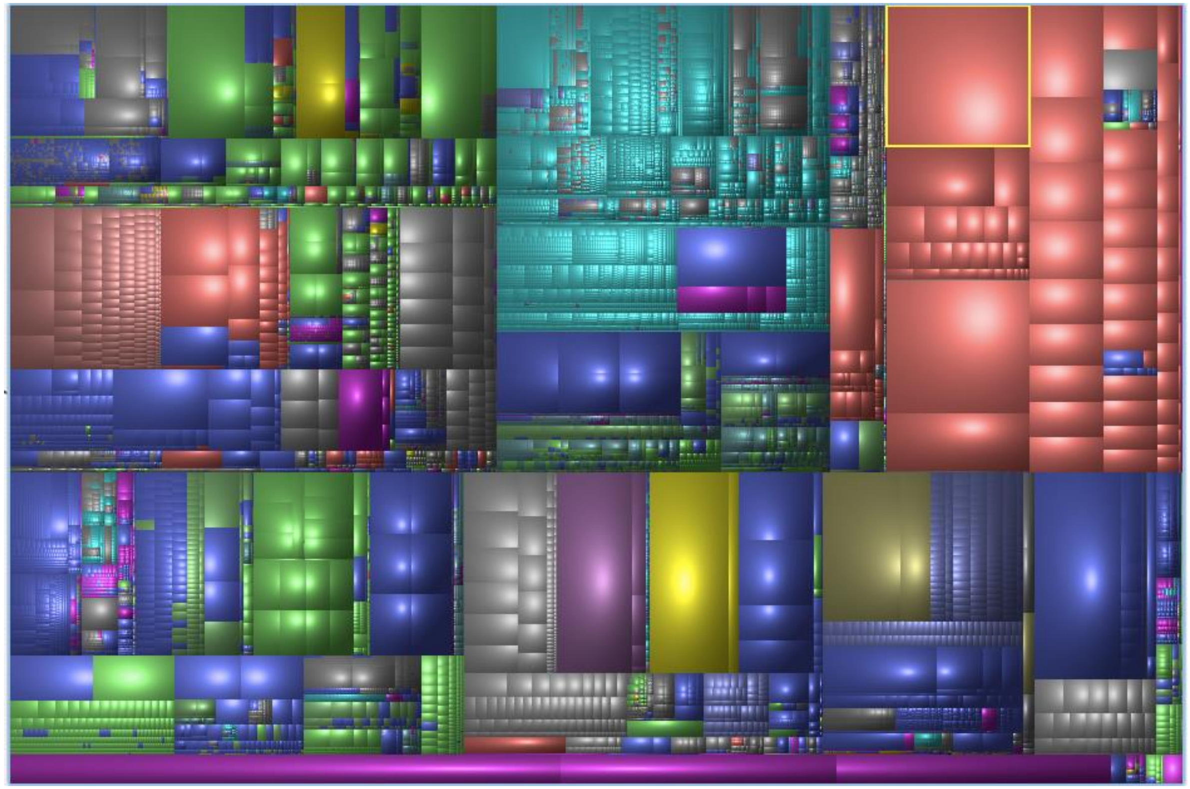
Original treemap



Squarified treemap

[Bruls et al., " Squarified Treemaps", TCVG 2000]

Squarified Treemaps

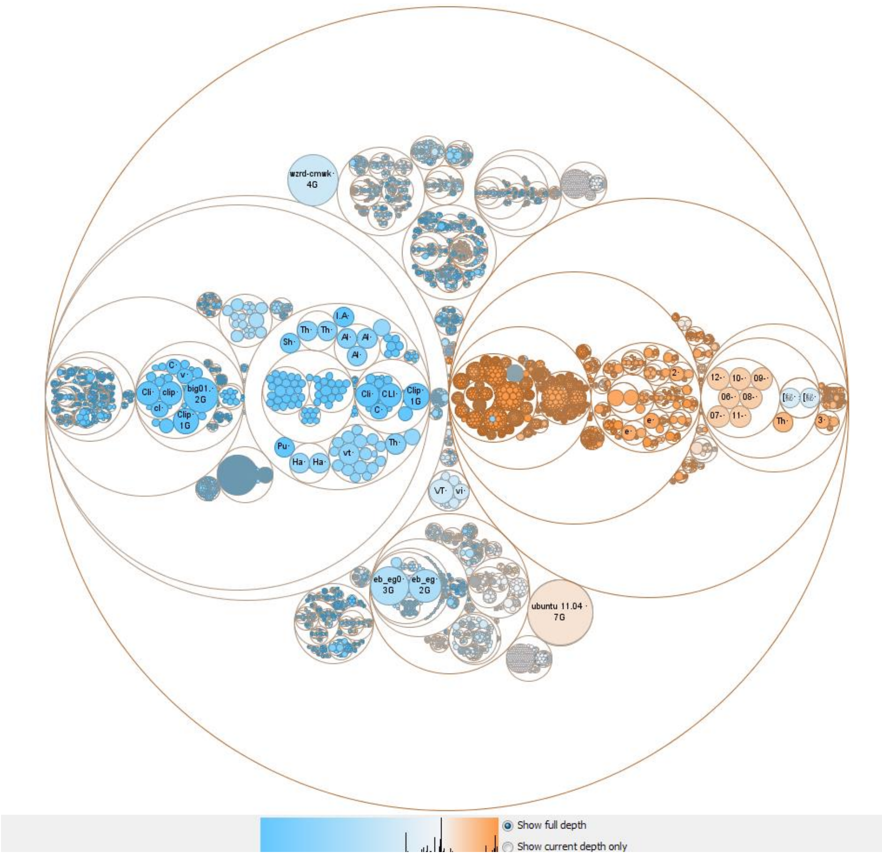


- Problems:
  - Change in dataset causes dramatic discontinuous change
  - Orders not preserved (Solution: Ordered Treemaps)

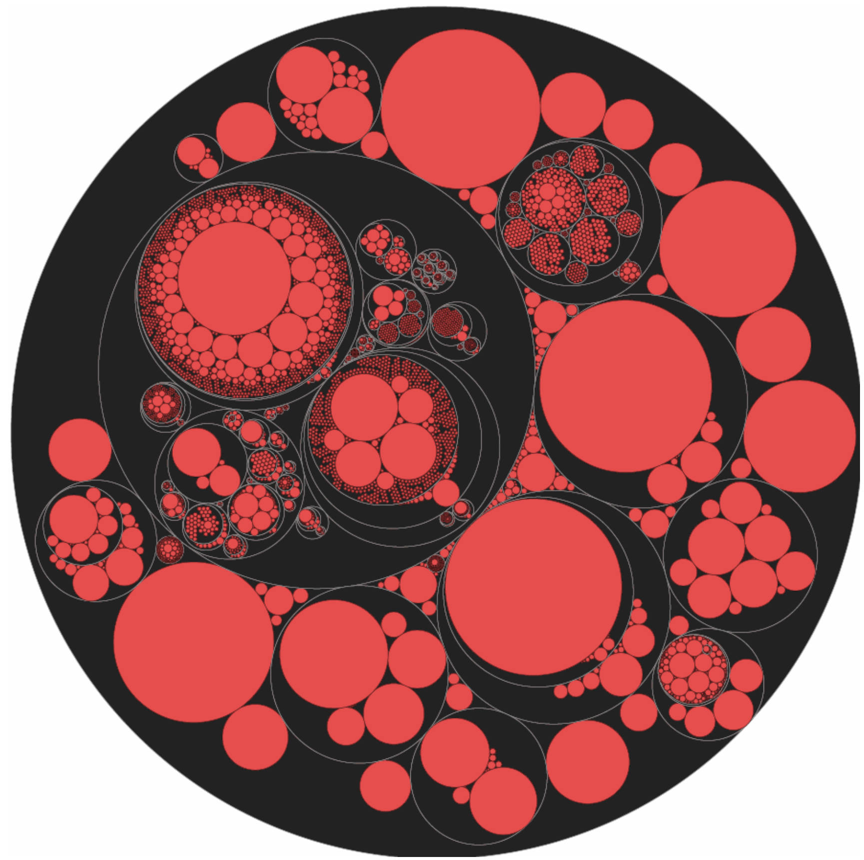
Circular Treemaps

- Use circles instead of rectangles
- Advantages:
  - Easy to compare sizes
  - Hierarchical structure is clear
- Problems:
  - Not as space efficient

Circular Treemaps



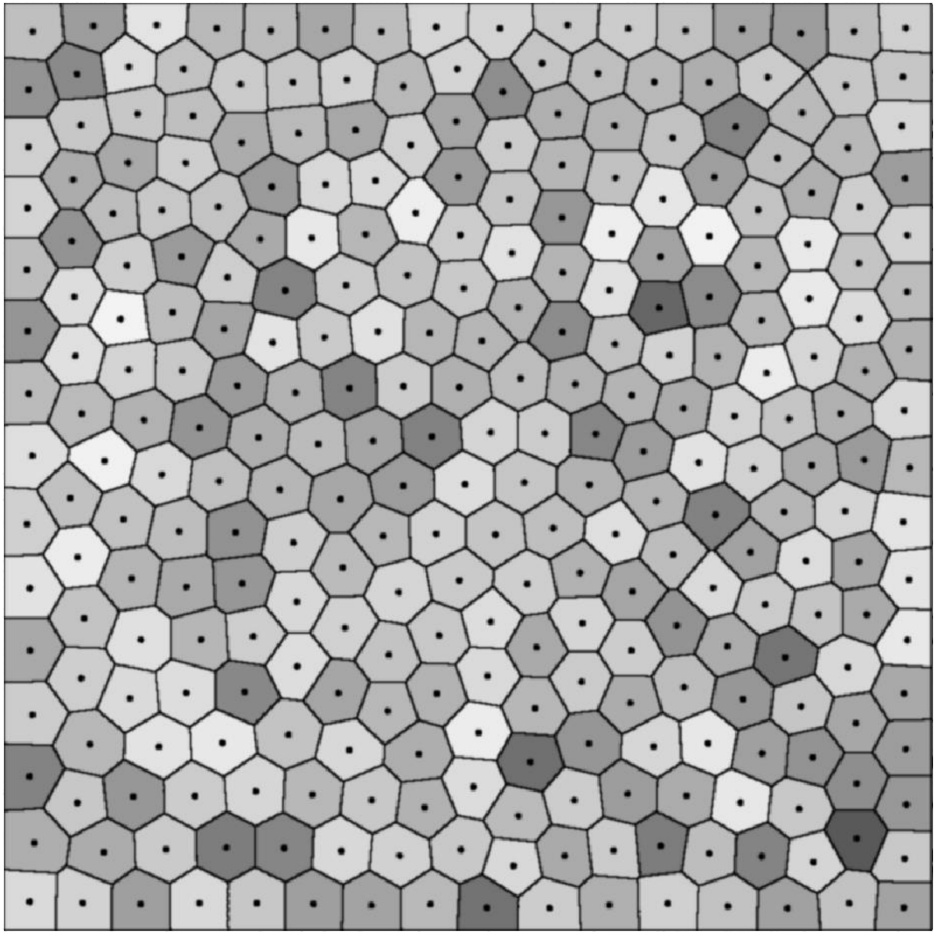
[Tree Visualizer]



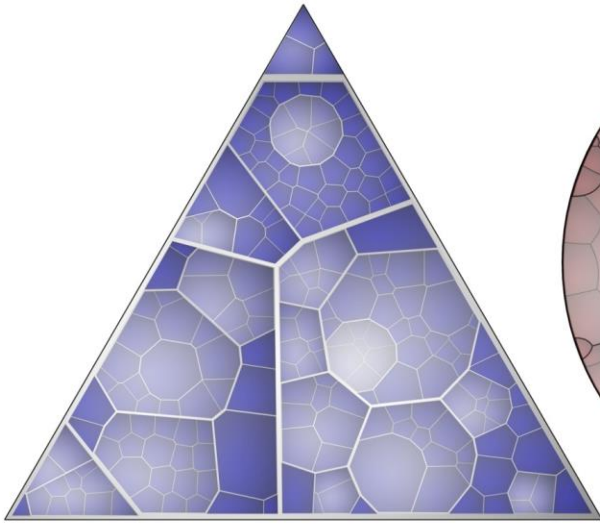
[Pebbles by Wetzel]

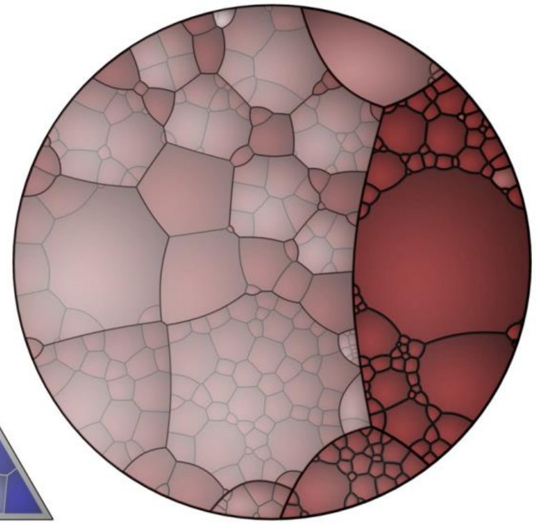
Voronoi Treemaps

- Make use of Centroidal Voronoi Tessellation

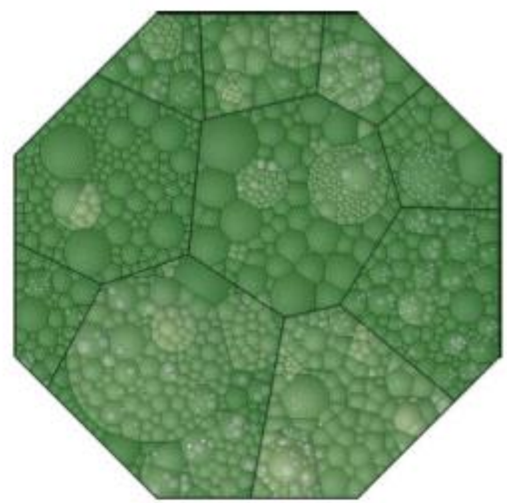


Voronoi Treemaps





[Balzer and Deussen, “Voronoi treemaps”, InfoVis 2005.]

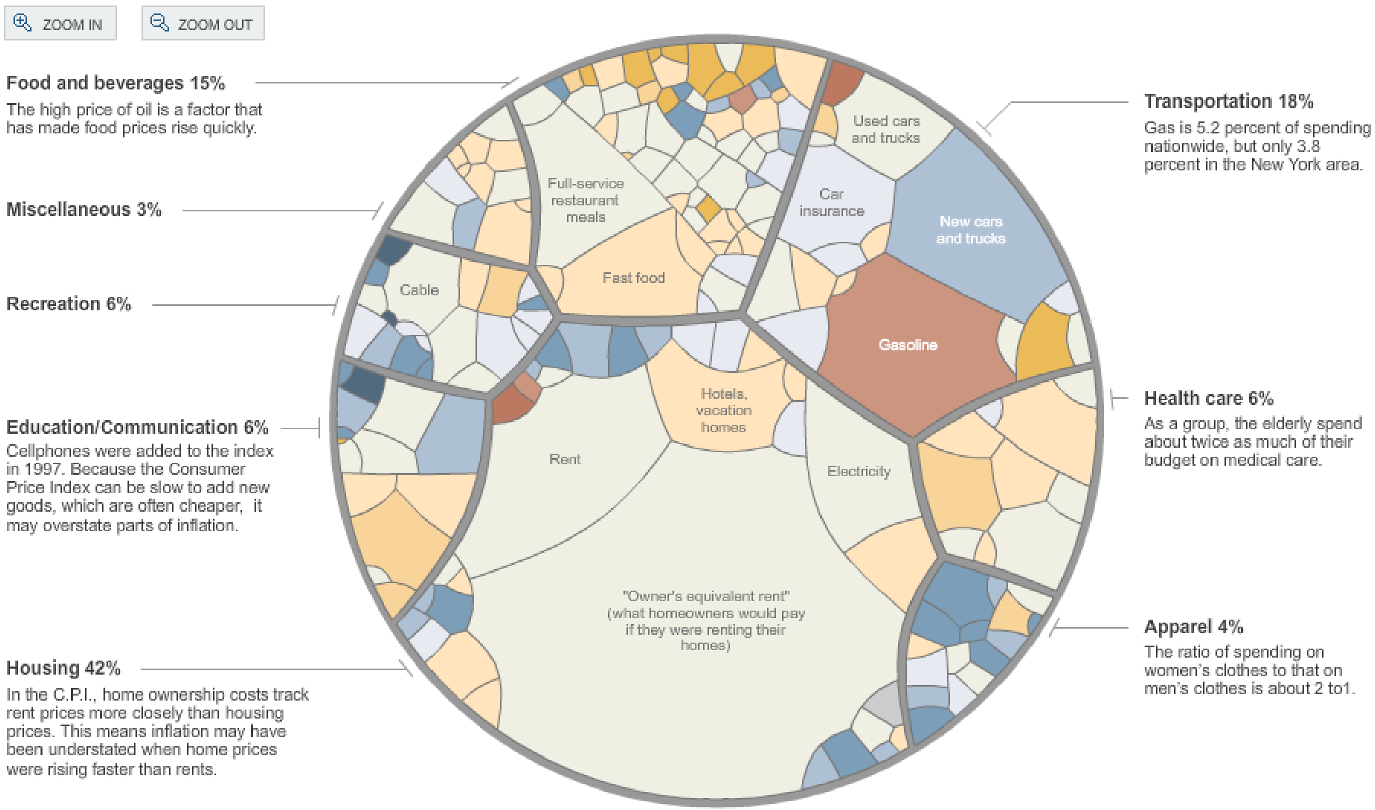


[Paul Murrel, University of Auckland]

Voronoi Treemaps

- Use arbitrary polygons to fill up an arbitrary space
- Boundaries not just formed by vertical and horizontal lines, therefore easier to see hierarchical structure
- Problems:
  - Computation involves an iterative process, which can be inefficient

Voronoi Treemaps



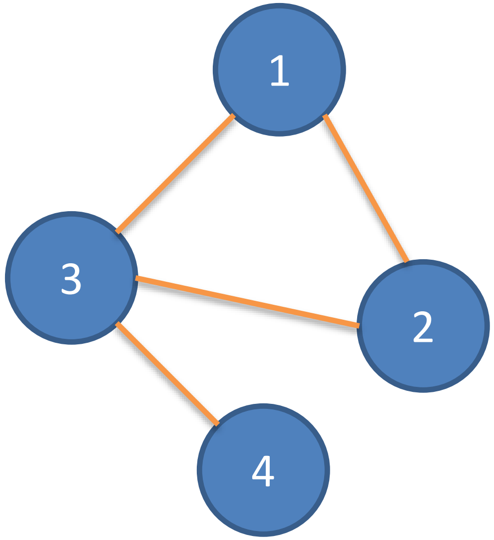
[http://www.nytimes.com/interactive/2008/05/03/business/20080403_SPENDING_GRAPHIC.html?_r=0]

Graphs

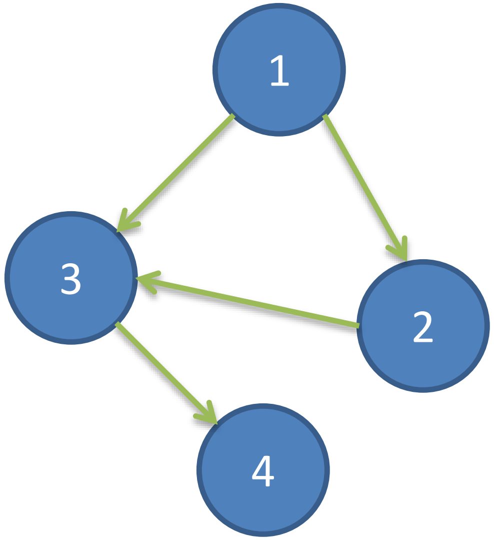
- **Graphs** are best for representing relational structures
  - Relational database
  - Object-oriented systems (class browsers)
  - Data structure (compiler data structure)
  - Real-time systems (state-transition diagrams)
  - Virtual reality (scene graphs)
  - Evolutionary trees, phylogenetic trees, genetic maps
  - Social networks, citation networks
  - Computer networks
  - World wide web
  - …

Graphs

- A graph **G=(V, E)** comprises a finite set **V** of vertices (or nodes) and a set **E** of edges (or lines), with each edge being an unordered pair (u, v) of vertices, u, v ∈ V.
- A **directed** graph consists of ordered pairs of vertices.



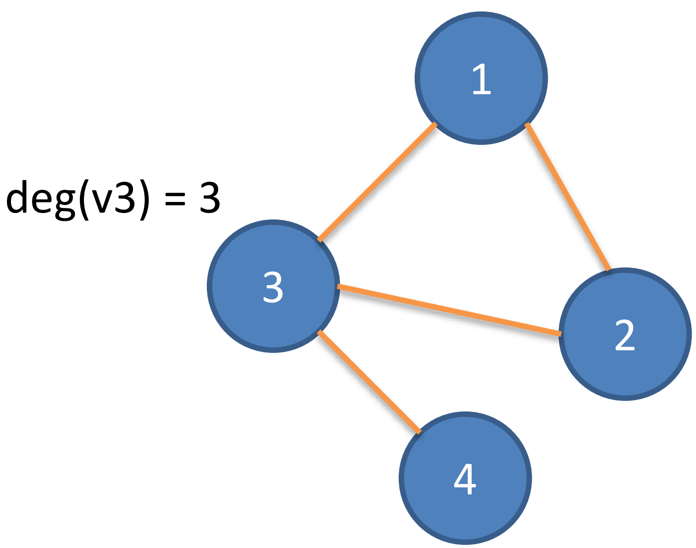
undirected graph



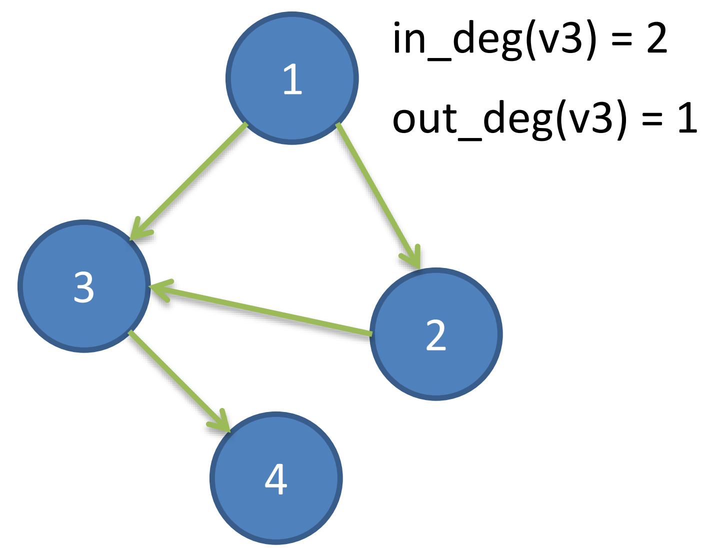
directed graph

Graphs

- The **degree** of a vertex
  = the number of edges incident to it
  = the number of neighboring vertices.



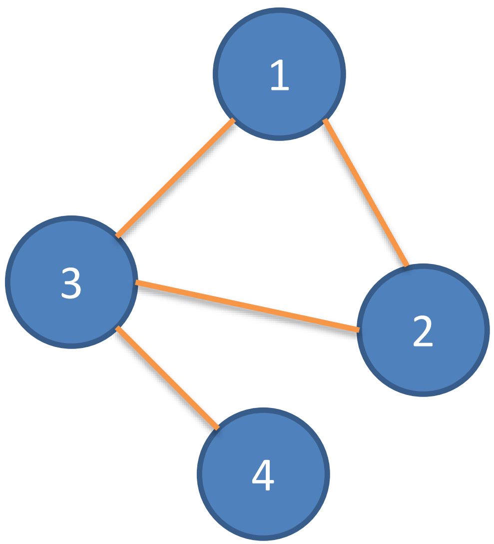
undirected graph



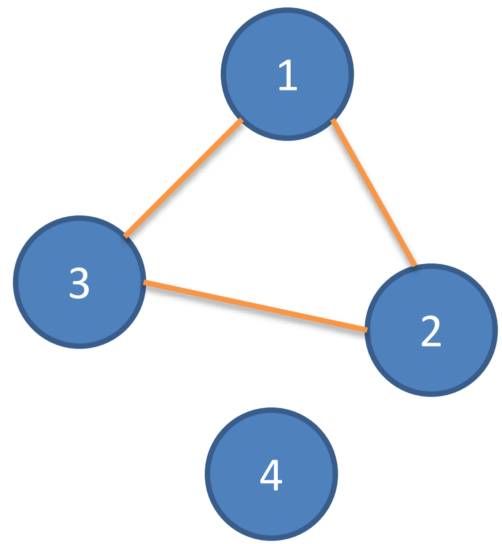
directed graph

Graphs

- A **path** is a sequence of edges connecting a sequence of vertices
- A graph is **connected** if there is a path between any two vertices.

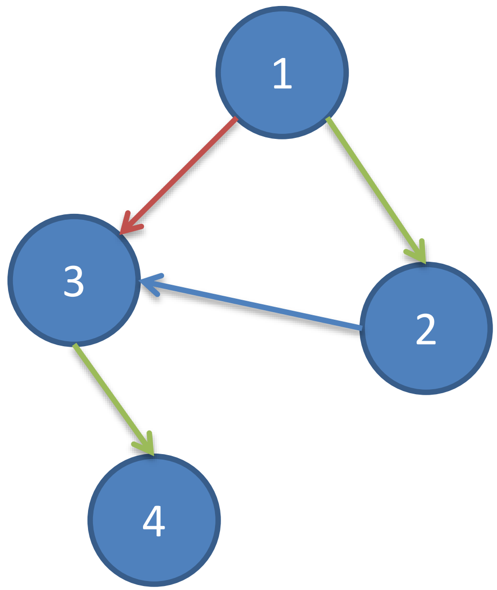


connected graph



disconnected graph

Graph Representation



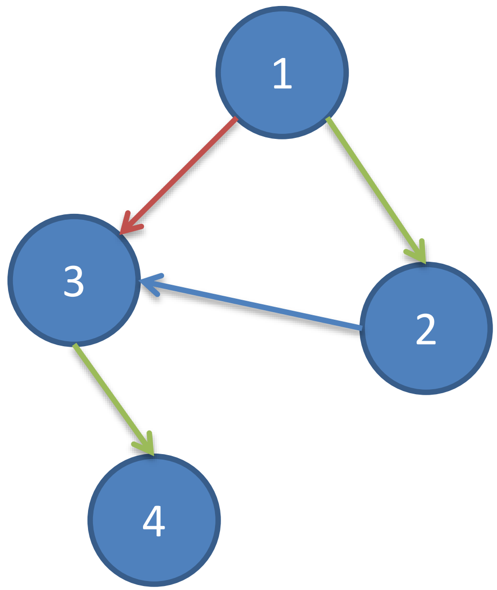
Adjacency Matrix

Target

		1	2	3	4
	1	0	2	1	0
	2	0	0	3	0
Source	3	0	0	0	2
	4	0	0	0	0

- $O(n^2)$ space
- Not a compact storage, especially when the matrix is a sparse matrix (i.e., a graph with only a few edges)

Graph Representation



Adjacency List

1 -> 2, 3
2 -> 3
3 -> 4

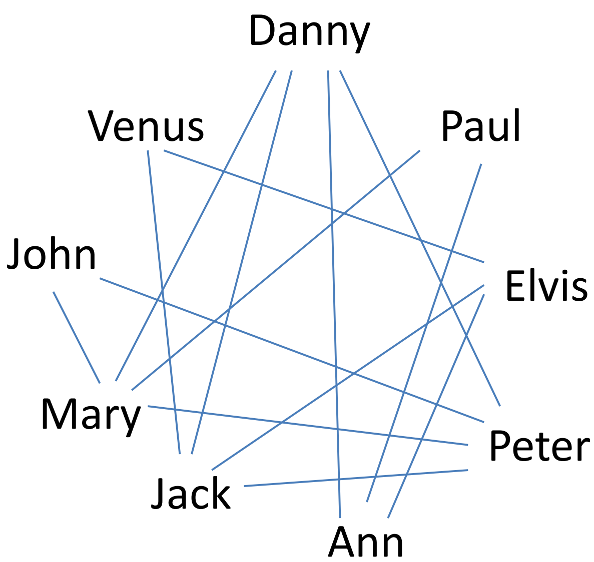
Edge List

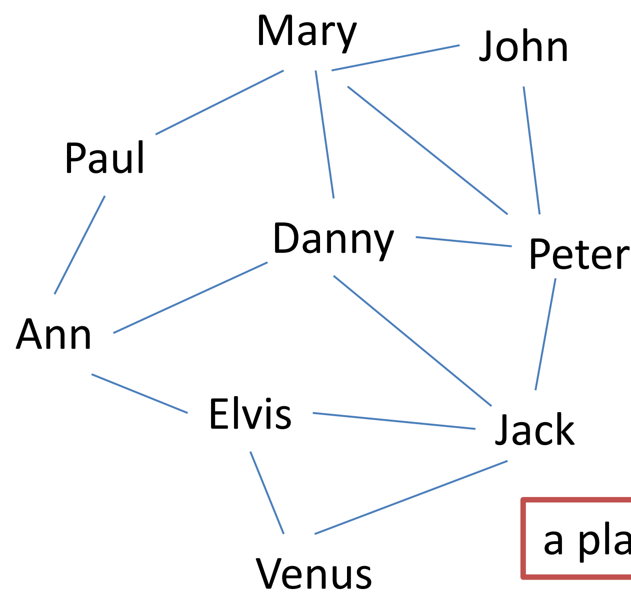
1, 2, **2**
1, 3, **1**
2, 3, **3**
3, 4, **2**

- Storage proportional to number of edges
- However, only support sequential access to edges

Graph Drawing

- There are many different ways of drawing a graph.





a planar graph

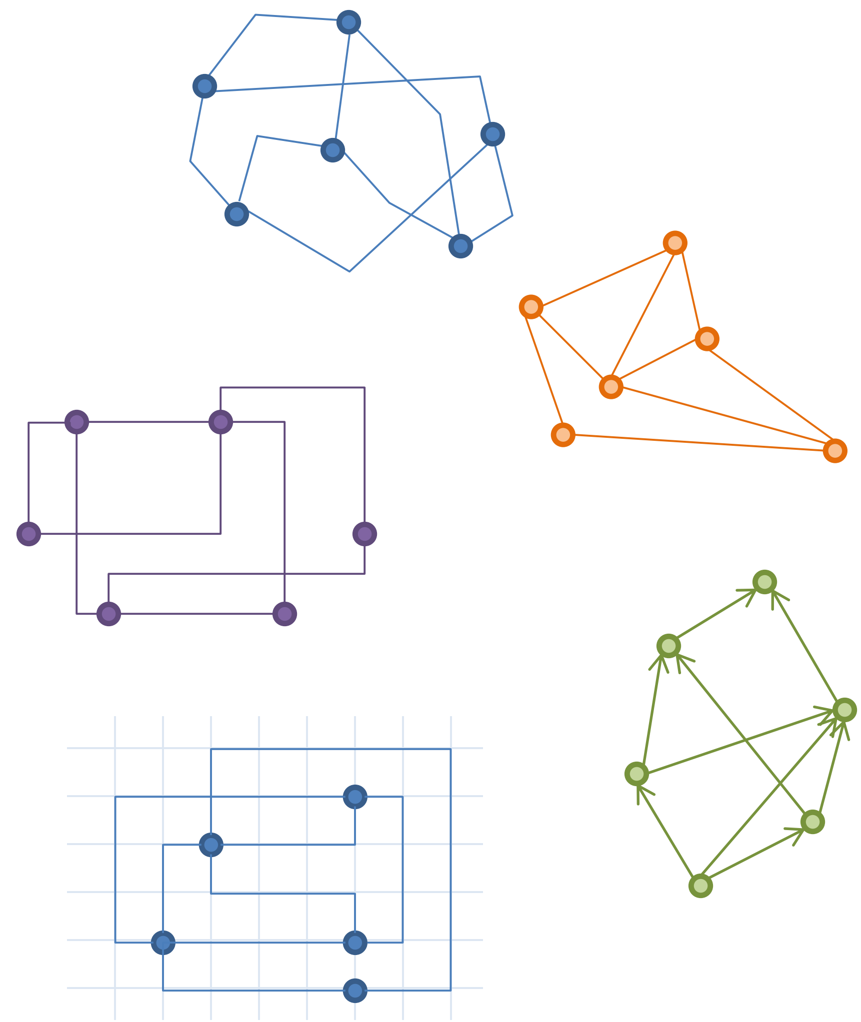
- Whether a drawing is good or not is subjective and depends on perception.

Graph Drawing

- Requirements
  - Drawing conventions
  - Aesthetics
  - Constraints
- Key issues
  - Graph size
    - need filtering, clustering?
  - Predictability
    - similar drawing for the same graph every time?
  - Time complexity
    - Is real-time interaction possible?

Drawing Conventions

- Straight-line drawing
- Polyline drawing
- Orthogonal drawing
- Planar drawing
- Grid drawing
- Upward drawing

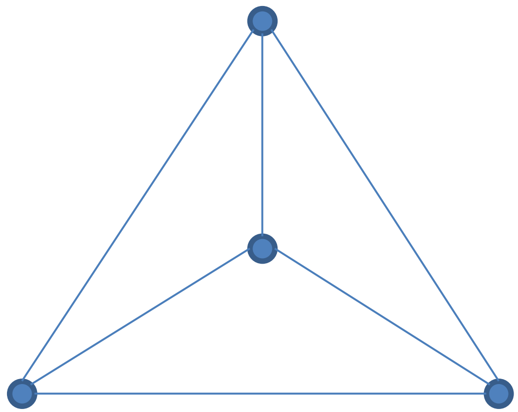


Aesthetics

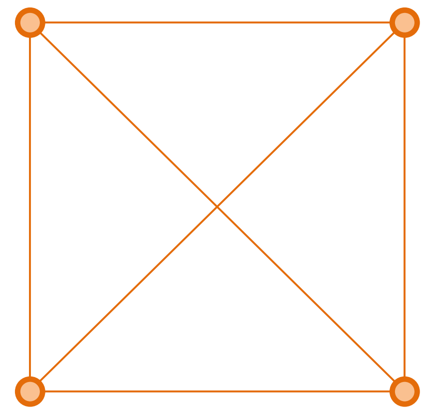
- Aims to characterize **readability** with general optimization goals:
  - Minimize
    - # of crossings
    - Total area
    - Total edge length
    - Maximum edge length
    - Total number of bends (in orthogonal drawings)
  - Maximize
    - Angular resolution (smallest angles between two edges)
    - Maximum display of symmetry
  - Maintain aspect ratio of the entire drawing

Aesthetics

- In general, aesthetic criteria are in conflicts.



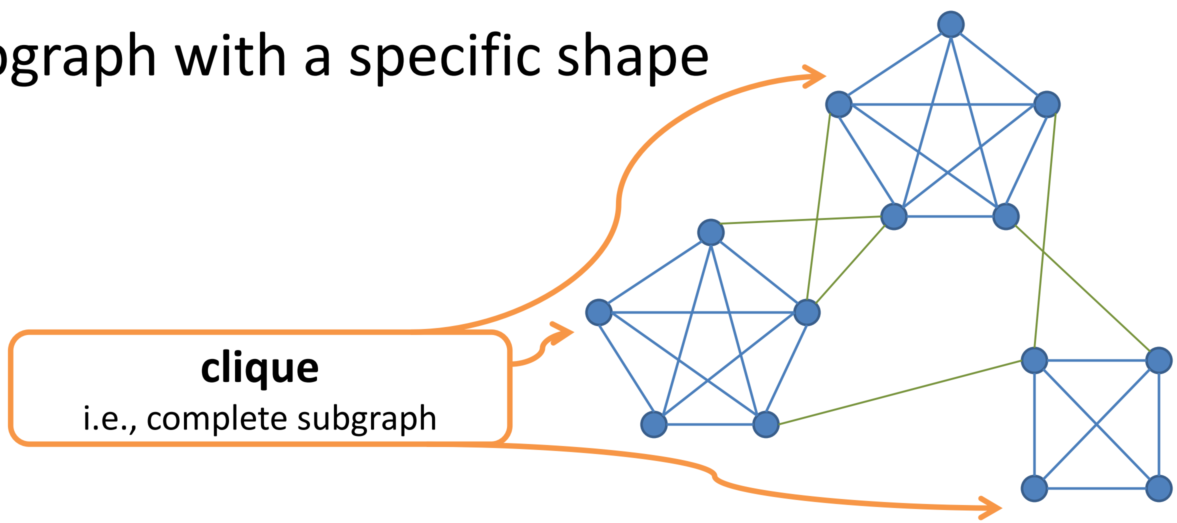
No edge crossing,
but symmetric



Symmetric,
but with edge crossing

Constraints

- Enhance readability by adding knowledge about the **semantics** of the graph
  - Place a vertex close to the middle
  - Place a vertex on the external boundary
  - Place a subset of vertices close together
  - Draw a subgraph with a specific shape



Reference

- Jeffrey Heer, Michael Bostock, and Vadim Ogievetsky. 2010. A tour through the visualization zoo. *Commun. ACM* 53, 6 (June 2010), 59-67. (http://hci.stanford.edu/jheer/files/zoo/)
- Matthew Ward, Georges Grinstein and Daniel Keim, "*Interactive Data Visualization: Foundations, Techniques, and Applications*", 2010 [Chapter 8]
- Isabel F. Cruz and Roberto Tamassia, “Graph Drawing Tutorial” (http://cs.brown.edu/~rt/papers/gd-tutorial/gd-constraints.pdf)