CS 519: Scientific Visualization

Graph Visualization

Eric Shaffer

Some slides adapted Alexandru Telea, *Data Visualization Principles and Practice*
• consider all pairs of columns \( c_i, c_j \)
  • construct a scatterplot \( P_{ij} \) for all pairs \( c_i, c_j \) over all rows
  • arrange \( P_{ij} \) in a matrix
• What question does it seek to answer?
• Scalability is an issues
Trees

- typically model aggregation or containment
- central to most InfoVis applications
  - data is hierarchical
  - data can be organized hierarchically (clustering)

Layout

- mapping operation for tree visualization
- associates positions (optionally: sizes) to tree nodes (optionally: edges)
- given a tree $T = (N, E)$ with nodes $N = \{n_i\}$ and edges $E = \{e_i\}$
- node layout

$$L : N \to \mathbb{R}^m \leftrightarrow \mathbb{R}^m$$

- layout is usually decoupled from
  - choice of shape of a node or edge and its
  - Node/edge visual attributes (color, shading, texture)
Node-Link Layouts

• most widespread: very intuitive

Examples
• icon size: folder size
• icon shape+color: level in tree

rooted tree

radial tree

cone tree

bubble tree

ffmpeg video code C library: 785 files, 42 folders

(Layouts generated with the Tulip visualization system [www.labri.fr/tulip])
• very familiar to virtually everybody
• size (# children) and depth of sub-trees easy to perceive
• smooth edge shading → emphasize colors of small nodes
• unbalanced aspect ratios can occur → limited scalability
Radial Layout

- like rooted tree, but arc-length used instead of $x$ axis
- size (# children) and depth of sub-trees easy to perceive
- good aspect ratio guaranteed
- nodes close to root get less space
• a subtree gets a full circle instead of a circle sector
• better spreading of the nodes for large trees
• variable edge lengths
• hard to distinguish node depth in the tree
• a subtree gets a full cone instead of a sector / circle / line
• 3D effectively shows the tree depth
• combines bubble tree (seen from above) with rooted tree (seen from profile)
• 3D is tricky: occlusions, perspective shortening, navigation
Treemaps

Basic ideas:
- 1 node = 1 rectangle
- child node rectangles: nested in the parent node rectangle (recursive subdivision)
- leaf rectangle size and color show data attributes
- edges: not drawn explicitly!
- very compact: tens of thousands of nodes on one screen
- hierarchy depth unclear

ffmpeg C library
Treemap “Slice and Dice” Layout Algorithm

- Alternate the axis you divide each level
- Divide the axis into lengths based on a metric.
  - Examples: uniform, number of nodes in subtree, application specific importance
- Recurse until all the nodes have been processed
- Only leaves explicitly visible
Squaremaps

Two extensions:
• enforce near-square aspect ratios during rectangle subdivision [Bruls et al '00]
• use shading: cushion profiles to convey hierarchical structure [Van Wijk et al '99]

- for each rectangle $r_i$ (except root)
  - define 2D parabola $h_i(x,y)$ with height $H_i = f^d$ ($d = \text{depth of } r_i \text{ in tree}$)
- compute global height $h = \Sigma_i h_i$
- for each image pixel $I(x,y)$
  - $I(x,y) = \text{shading} (h(x,y))$
  - use an oblique light source for better results
Treemap “Squarified” Layout Algorithm

- Start with list of child nodes
- Sort by area, largest first
- Split the longest axis
- Start filling one “half” in alternate direction of the split (step 2)
  - Keep filling until aspect ratios get worse (step 3)
- Repeat process on the unfilled portion of the original
Squarified Cushion Treemaps

- borders can be *implicit* in the shading discontinuities
- discontinuity strength conveys tree depth levels

Image generated with the SequoiaView tool [(www.win.tue.nl/sequoiaview)](http://www.win.tue.nl/sequoiaview)
Squarified Cushion Treemaps

Comparison of methods

hard disk, ~30K files

- slice and dice layout
- unbalanced sizes
- structure is not very clear

- squarified layout
- balanced sizes
- structure is more visible
Squarified Treemaps

Map of the Market (http://www.smartmoney.com/map-of-the-market)
- visualize stock exchange data online
  - hierarchy: nesting of companies within sectors
  - rectangle size: market capitalization
  - color: gain (green) ... loss (red)
**Squarified Cushion Treemaps**

Treemaps from tables:
- recall the multiple-sorting idea for tables?
  - implicitly creates a hierarchy from a table (with just a few clicks)
  - visualize hierarchy with a treemap

- stock exchange table
- three levels:
  - market sector
  - companies
  - prices / day over 1 month
- colors
  - red: daily loss
  - green: daily gain
  - light blue: unavailable data

Discovered a small emerging company with steady growth!
US Presidential Election 2012
Directed Acyclic Graphs

- class hierarchies (multiple inheritance)
- organization structure (multiple bosses)
- also created from general graphs by removing cycles

Hierarchical layout [Sugiyama et al '81]

Algorithm:

- swap edges to eliminate cycles and get a directed (rooted) graph
- for every level, starting from root:
  - assign y coordinate as function of level
  - permute nodes on level to minimize edge lengths/crossings
- edges drawn as curves (splines) to minimize crossings
Directed Acyclic Graphs

Hierarchical layout scalability
• OK for < 1000 nodes or edges
• too many crossings and/or bad aspect ratios for large graphs
• we shall see later how to handle large graphs

Bison library call graph (3.214 nodes, 14.382 edges)
General Graphs

- not necessarily directed; can have cycles

**Force-directed layouts**

- design an energy function $E : \mathbb{R}^m \to \mathbb{R}_+$ which is low when layout is “good”
  - connected nodes should be close $\rightarrow$ layout distance should reflect graph-theoretic
  - nodes should not overlap
  - aspect ratio should be balanced

[Fruchterman and Reingold '91]

- Define $E$ in terms of forces
  - Repulsive force $F_r$ between all pairs of vertices
  - Attractive force $F_a$ between neighboring pairs of vertices

- find node positions $p_i$ by minimizing $E$:
  - move vertices iteratively $F_a + F_r$
  - Damp the movement with each iteration
    (e.g. decrease max distance $t$ that a vertex can move)
  - Stop when movements are small

- Optimization
  - compute $F_r$ only w.r.t. close nodes (to save time)
  - use spatial search structures (e.g. octrees) to find close nodes
Force-Directed Layouts

Problems:

- the energy function is not monotonic!
- minimizers such as previous gradient descent work locally $\rightarrow$ can get stuck in local minima
- solve this by more advanced minimizers (see e.g. [Di Battista et al '94])
- drawings not intuitive; no clear ordering $\rightarrow$ where to start reading the drawing??

energy 2D height plot

we start here...

...minimizer may get stuck here (local minimum)

...instead of arriving here (global minimum)
Force-Directed Layouts

Examples

call graph

inheritance tree

call graph

inheritance forest
Graph Splatting

• node-link layouts get easily cluttered; remove problem by not drawing edges
• key idea: transform discrete dataset into a continuous one!
• do a force-directed layout
• convolve nodes (and optionally edges) with a Gaussian filter (radius $r =$ simplification level)

$$f(x, y) = \sum_{i=1}^{N} k_i e^{-\left(\frac{|x-x_i|}{r}\right)^2}$$

• render the resulting density signal using SciVis methods

Java call graph (5000 nodes, 2000 relations), courtesy Nokia Research
Graph Splatting

- use factor $k_i$ to specify the importance of $i^{th}$ node
- example: $k_i = \text{number of requirements / provisions of a component (out/ingoing edges)}$
- high densities denote strongly connected clusters / important nodes

components using String class

package requirements
Graph Splatting

- the splat field can be visualized also as a height plot
- other SciVis techniques possible too (isolines, clipping...)
- however: the result quality strongly depends on the layout quality!
Matrix Plots

- A graph can be represented as an adjacency matrix.

Layout Comparison

- consider a software call graph
  - nodes: functions; edges: call relations ($f$ calls $g$ – directed graph)
  - functions are contained in modules ($f \in A$, $g \in B$, …)
  - some calls are allowed, some now (depending on system architecture)
- which layout is best for spotting call structure problems?

![Matrix plot](image1)
![Force directed](image2)
![Hierarchical DAG](image3)

- no clutter, clean image
- show violations by colors (red=not allowed)
- tracing call paths is hard
- detecting “connected components” is hard

- some connected components visible
- good for path tracing
- image is cluttered...
- …even if this graph is really small
- least preferred option

- some structure is apparent...
- …but is it a layout artefact?
- top-down order meaningful
Matrix Plots

Interaction
• zoom in and out – display the matrix at different levels of detail
• smoothly interpolate the cell sizes (using alpha blending) during zooming

Matrix Plots and Node-Link Layouts

- matrix plots are compact and clean, but don’t clearly show structure
- node-link layouts show structure, but get cluttered

Idea
- aggregate graph (find clusters of strongly-connected components)
- show each cluster with a matrix plot
- show cluster-graph with a node-link layout

Largest component of InfoVis co-authorship network

Aggregation

• recall the discussion on semantic resampling?
• graph aggregation
  • produces a simpler/smaller “cluster graph” from a large one
  • nodes: partitioned between disjoint clusters
  • edges:
    • intra-cluster: relate nodes in same cluster
    • inter-cluster: relate clusters
  • many clustering methods (strongly-connected components, data based, …)
• visualize
  • the cluster graph
  • cluster internals using a cluster icon (e.g. a matrix plot)

We will meet this idea several times further on!
Other Techniques

directionally constrained layouts
Voronoi treemaps
radial icicle plots
star plots

theme rivers
beam trees
arc trees