Network Layout

Maneesh Agrawala

CS 448B: Visualization
Fall 2021

Reading Response Questions/Thoughts

For the final project, do you have a recommendation of a place to go to view other data visualization research papers that conducted user studies?

As animations contain more and more data, is it possible that we can overload or overstimulate the user? Can animations be harmful by being too distracting? If so, how can we safeguard our designs to make sure they don’t cause this overstimulation?

Is there a more formal or mathematical rule set governing which colors to use to highlight information, and which to contrast? Or is it mostly a combination of multiple factors that you need to see to know? In a similar vein, do colors need to be different in shade as well as color for black and white printing? How do we know to vary transparency with color or just color?

How seriously should we take self-reported stated preferences when evaluating the strength of a visualization? How much should we weight user’s expressed preference relative to usability, learning, and recall data when evaluating the efficacy of a visualization?
Last Time: Animation
Understanding Motion

How does it work?

Two-cylinder Stirling engine
http://www.keveney.com/Vstirling.html
Problems [Tversky 02]

Difficulties in understanding animation

- Difficult to estimate paths and trajectories
- Motion is fleeting and transient
- Cannot simultaneously attend to multiple motions
- Trying to parse motion into events, actions and behaviors
- Misunderstanding and wrongly inferring causality
- Anthropomorphizing physical motion may cause confusion or lead to incorrect conclusions

Solution I: Break into static steps

1. Expansion. At this point, most of the gas in the cylinder has just been shifted to the hot cylinder. The gas heats and expands, driving both pistons inward.

2. Transfer. At this point, the gas has expanded (about 3 times in this example). Most of the gas (about 2/3) is still located in the hot cylinder. Physical momentum causes the center piston to push all the gas to the cool cylinder.

3. Contraction. Now the majority of the expanded gas has been shifted to the cool cylinder. It cools and contracts, drawing both pistons inward.

4. Transfer. The new contracted gas is still located in the cool cylinder. Physical momentum causes the cool piston, moving 90 degrees, to transfer the gas to the hot cylinder to complete the cycle.

Two-cylinder Stirling engine
http://www.keveney.com/Vstirling.html
Challenges

Choosing the set of steps

- How to segment process into steps?
- Note: Steps often shown sequentially for clarity, rather than showing everything simultaneously

Tversky suggests

- Coarse level – segment based on objects
- Finer level – segment based on actions
  - Static depictions often do not show finer level segmentation

Animated Transitions in Statistical Graphics
Log Transform
Sorting
Filtering
Change Encodings
Change Data Dimensions
Change Data + Encodings

Change Encodings + Axis Scales
Data Graphics & Transitions

Visual Encoding

Change selected data dimensions or encodings

Animation to communicate changes?

Animated Transitions in Statistical Data Graphics

Jeffrey Heer
George G. Robertson

Microsoft Research
Study Conclusions

Appropriate animation improves graphical perception
Use simple staged transitions, but doing one thing at a time not always best

Axis re-scaling hampers perception
Avoid if possible (use common scale)
Maintain landmarks better (delay fade out of gridlines)

Subjects preferred animated transitions

Implementing Animation
Animation Approaches

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

circle(10,10)  circle(15,15)  circle(20,20)  circle(25,25)

1  2  3  4

clear()  clear()  clear()  clear()
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

\[
\text{step}(\text{fraction}) \{ \text{x}_{\text{now}} = \text{x}_{\text{start}} + \text{fraction} \times (\text{x}_{\text{end}} - \text{x}_{\text{start}}); \}
\]

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

Toolkit handles frame-by-frame updates
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);
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));

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));
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!

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))
...

D3 Transitions, Continued
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

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

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

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 different interactive visualizations
- Research: Implementation of solution and web-based demo if possible
- Short video (2 min) demoing and explaining the project

Schedule
- Project proposal: Wed 11/3
- Design Review and Feedback: 10th week of quarter
- Final code and video: Fri 12/10 11:59pm

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

Tree Layout
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

---

Indentation

Items along vertically spaced rows
- Indentation shows parent/child relationships
- Often used in interfaces
- Breadth/depth contend for space
- Often requires scrolling
Visualizing Large Hierarchies

Indented Layout

Single-Focus (Accordion) List

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

Basic Recursive Approach

Repeatedly divide space for subtrees by leaf count
  - Breadth of tree along one dimension
  - Depth along the other dimension
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
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.

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

**Initial bottom-up (postorder) tree traversal**
- Set y-coordinate based on depth
- Initialize x-coordinate to zero

**At each parent node, merge left and right subtrees**
- Shift right subtree as close as possible to left
  - Computed efficiently by maintaining subtree contours
- Center parent nodes above children
- Record “Shift” in position offset for right subtree

**Final top-down (preorder) traversal to set x-coordinates**
- Sum aggregated shift
Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize x-coordinate to zero
- At each parent node, merge left and right subtrees
  - Shift right subtree as close as possible to left
  - Computed efficiently by maintaining subtree contours
  - Center parent nodes above children
  - Record “Shift” in position offset for right subtree
Final top-down (preorder) traversal to set x-coords
- Sum aggregated shift
Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize x-coordinate to zero
- At each parent node, merge left and right subtrees
  - Shift right subtree as close as possible to left
    - Computed efficiently by maintaining subtree contours
  - Center parent nodes above children
  - Record “Shift” in position offset for right subtree
Final top-down (preorder) traversal to set x-coords
- Sum aggregated shift
Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize x-coordinate to zero
At each parent node, merge left and right subtrees
- Shift right subtree as close as possible to left
- Computed efficiently by maintaining subtree contours
- Center parent nodes above children
- Record “Shift” in position offset for right subtree
Final top-down (preorder) traversal to set x-coords
- Sum aggregated shift
Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize x-coordinate to zero
- At each parent node, merge left and right subtrees
  - Shift right subtree as close as possible to left
  - Computed efficiently by maintaining subtree contours
  - Center parent nodes above children
  - Record "Shift" in position offset for right subtree

Final top-down (preorder) traversal to set x-coords
- Sum aggregated shift
Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize x-coordinate to zero
At each parent node, merge left and right subtrees
- Shift right subtree as close as possible to left
- Computed efficiently by maintaining subtree contours
- Center parent nodes above children
- Record "Shift" in position offset for right subtree
Final top-down (preorder) traversal to set x-coords
- Sum aggregated shift

Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize x-coordinate to zero
At each parent node, merge left and right subtrees
- Shift right subtree as close as possible to left
- Computed efficiently by maintaining subtree contours
- Center parent nodes above children
- Record "Shift" in position offset for right subtree
Final top-down (preorder) traversal to set x-coords
- Sum aggregated shift
Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize x-coordinate to zero

At each parent node, merge left and right subtrees
- Shift right subtree as close as possible to left
  - Computed efficiently by maintaining subtree contours
- Center parent nodes above children
- Record "Shift" in position offset for right subtree

Final top-down (preorder) traversal to set x-coords
- Sum aggregated shift
Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize x-coordinate to zero
At each parent node, merge left and right subtrees
- Shift right subtree as close as possible to left
  - Computed efficiently by maintaining subtree contours
  - Center parent nodes above children
- Record "Shift" in position offset for right subtree
Final top-down (preorder) traversal to set x-coords
- Sum aggregated shift
Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize x-coordinate to zero
At each parent node, merge left and right subtrees
- Shift right subtree as close as possible to left
- Computed efficiently by maintaining subtree contours
- Center parent nodes above children
- Record "Shift" in position offset for right subtree
Final top-down (preorder) traversal to set x-coords
- Sum aggregated shift
Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize x-coordinate to zero
- At each parent node, merge left and right subtrees
  - Shift right subtree as close as possible to left
  - Computed efficiently by maintaining subtree contours
  - Center parent nodes above children
  - Record “Shift” in position offset for right subtree

Final top-down (preorder) traversal to set x-coords
- Sum aggregated shift
Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize x-coordinate to zero
- At each parent node, merge left and right subtrees
  - Shift right subtree as close as possible to left
  - Computed efficiently by maintaining subtree contours
  - Center parent nodes above children
  - Record "Shift" in position offset for right subtree

Final top-down (preorder) traversal to set x-coords
- Sum aggregated shift
Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize coordinates to zero
- At each parent node, merge left and right subtrees
  - Shift right subtree as close as possible to left
  - Computed efficiently by maintaining subtree contours
  - Center parent nodes above children
  - Record “Shift” in position offset for right subtree
Final top-down (preorder) traversal to set x-coords
- Sum aggregated shift

108

109
Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize x-coordinate to zero
- At each parent node, merge left and right subtrees
  - Shift right subtree as close as possible to left
  - Center parent nodes above children
  - Computed efficiently by maintaining subtree contours
- Record "Shift" in position offset for right subtree
- Final top-down (preorder) traversal to set x-coords
  - Sum aggregated shift
Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize x-coordinate to zero
- At each parent node, merge left and right subtrees
  - Shift right subtree as close as possible to left
    - Computed efficiently by maintaining subtree contours
  - Center parent nodes above children
  - Record "Shift" in position offset for right subtree
Final top-down (preorder) traversal to set x-coords
- Sum aggregated shift

Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize x-coordinate to zero
- At each parent node, merge left and right subtrees
  - Shift right subtree as close as possible to left
    - Computed efficiently by maintaining subtree contours
  - Center parent nodes above children
  - Record "Shift" in position offset for right subtree
Final top-down (preorder) traversal to set x-coords
- Sum aggregated shift
Reingold-Tilford Algorithm

Initial bottom-up (postorder) tree traversal
- Set y-coordinate based on depth
- Initialize x-coordinate to zero
- At each parent node, merge left and right subtrees
  - Shift right subtree as close as possible to left
    - Computed efficiently by maintaining subtree contours
  - Center parent nodes above children
  - Record “Shift” in position offset for right subtree

Final top-down (preorder) traversal to set x-coords
- Sum aggregated shift
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

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
Visualizing Large Hierarchies

Indented Layout

Reingold-Tilford Layout

MC Escher, Circle Limit IV
**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

**Degree-of-Interest Trees [AVI 04]**

- Space-constrained, multi-focal tree layout

[Links provided for videos and interactive trees]
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

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
Circle Packing Layout

Nodes represented as sized circles
Nesting to show parent-child relationships

Problems:

- Inefficient use of space
- Parent size misleading
**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.
Squarified layout: Try to produce square (1:1) aspect ratios

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/
**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?*

---

**Error vs. Aspect Ratio** [Kong 10]

1. Comparison of squares has higher error!
2. Squarify works because it fails to meet its objective?
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?*
   *Extreme ratios & squares-only more inaccurate.*
   *Balanced ratios better? Target golden ratio?*