Data Visualization (DSC 530/CIS 602-01)

Data

Dr. David Koop
Focus+Context Overview

mbed

- Elide Data
- Superimpose Layer
- Distort Geometry

Reduce

- Filter
- Aggregate
- Embed

[Munzner (ill. Maguire), 2014]
Elision: DOI Trees

[Heer and Card, 2004]
Superimposition with Interactive Lenses

(a) Alteration

(b) Suppression

[ChronoLenses and Sampling Lens in Tominski et al., 2014]
Superimposition with Interactive

(c) Enrichment

[Extended Lens in Tominski et al., 2014]
Distortion

It can be difficult to observe micro and macro features simultaneously with complex graphs. If you zoom in for detail, the graph is too big to view in its entirety. If you zoom out to see the overall structure, small details are lost. Focus + context techniques allow interactive exploration of an area.

[M. Bostock, http://bost.ocks.org/mike/fisheye/]
Cartesian Distortion

[M. Bostock, http://bost.ocks.org/mike/fisheye/]
Cartesian Distortion

[M. Bostock, http://bost.ocks.org/mike/fisheye/]
Focus+Context in Graph Exploration

(a) Bring (step 1) – Selecting a node fades out all graph elements but the node neighborhood.
(b) Bring (step 2) – Neighbor nodes are pulled close to the selected node.
(c) Go – After selecting a neighbor (the green node in Fig. 4(b)), a short animation brings the focus towards a new neighborhood.

Figure 4: Illustration of the Bring & Go interaction.

In order to be able to easily interact with the edge-bundled graphs, even for basic interactions like panning and zooming, we have to optimize the curves rendering by reducing the computational load on the CPU as much as possible. One solution could be to pre-compute all curve points and store them in memory; this obviously is not efficient in terms of memory usage, considering that we want to draw a large amount of fine-grained rendered curves. For example, drawing $10^{5}$ curves (edges) with 100 points per curves – one point being stored as 3 floats (4 bytes each), the total amount of memory use would be $\sim 10^8$ bytes (more than 110 Mbytes).

Another solution will be to use the built-in components of high level graphics API for rendering curves. For instance, in OpenGL, that task can be achieved by using a standard feature called evaluators. Evaluators can be used to construct curves and surfaces based on the Bernstein basis polynomials. This includes B´ezier curves and patches, and B-splines. An evaluator is set up from an array of control points and allows to compute curve points on the GPU by sending the parameter $t$ to the rendering pipeline. However, most of the OpenGL implementations have restrained the maximum authorized number of control points to eight. So to draw a B´ezier curve or a cubic B-spline with more than eight control points using evaluators, it has to be done piecewise by subdividing the curve to render into curves with fewer control points. Consequently, the performance to draw high order curves with this technique decreases as the number of control points grows. So even if evaluators work well to render curves with a small number of control points, they are not suitable to resolve our issue of drawing curves with several dozens of control points efficiently.

4.2 GPU-intensive spline rendering

Our solution delegates the computation of curve points to the GPU which is perfectly well designed to perform vectorial computation and floating points operations. By using the OpenGL Graphics API, we can encapsulate those tasks in a shader program. This type of program, written in a C-like language called GLSL (OpenGL Shading Language), allows to modify the default behavior of some processing units in the rendering pipeline – the vertex processing unit can be customized this way. The purpose of vertex processing stage is to transform each vertex's 3D position in virtual space to the 2D coordinates at which it appears on the screen. By designing a vertex shader we can manipulate properties such as node position or color, with all computations executed on the GPU. Shaders offer tangible benefits since they are well suited for parallel processing as most modern GPUs have multiple shader pipelines. The vertex shader we designed is activated each time we render a curve on screen. Before sending vertex coordinates to the GPU, the curve's control points are transferred to the shader and stored in an array. The maximum size of that array is hardware dependent and determined at runtime. On recent GPU, more than one thousand control points can be handled. Focus+Context in Graph Exploration

[Lambert et al., 2010]
Venn Diagram

- Famous Women
  - Hillary Clinton
  - Mother Teresa
- Nicole Kidman
- Jennifer Aniston
- Will Smith
- George Clooney
- Richard Dean Anderson
- Matt LeBlanc

Movie Stars
- James Earl Jones
- Harrison Ford

TV Stars

[http://askville.amazon.com/idea-Venn-diagram/AnswerViewer.do?requestId=8420613]
Scalability

• How to show the intersection of four sets? 8?
• Euler Diagrams: only show intersections/containments that exist
• Still run into scalability issues

[Diagram of the British Isles and its constituent parts, including Ireland, the United Kingdom, Great Britain, and various island territories.]

[Wikipedia]
4.4 Sets over Scatterplots

Scatterplots have clearly defined spatiality due to the numerical positioning of items. We add Bubble Sets to a reimplementation of the well-known GapMinder Trendalyzer. This scatterplot shows fertility rate against life expectancy and is animated over time. Data points represent countries, sized by population, and color and set membership is defined by the continent. The grouping of the Sub-Saharan Africa countries highlighted in Figure 6 reveals that while most of the countries in this set had high fertility rates and low life expectancies, there are two outliers: Mauritius and Reunion, which are islands in the Indian Ocean. As the data set includes data for many years, and since Bubble Sets are calculated at interactive rates, the temporal changes can be convincingly shown through animation.

5. DISCUSSION AND FUTURE WORK

We have presented Bubble Sets, a method for automatically drawing set membership groups over existing visualizations with different degrees of requirements for primary spatial rights. In contrast to other overlaid containment set visualizations, Bubble Sets maximize set membership inclusion and minimize inclusion of non-set members. In fact, Bubble Sets can guarantee that all set members will be within one container, as opposed to the more common multiple disjoint containers. While Bubble Sets cannot guarantee non-set member exclusion, the routing algorithm minimizes these occurrences.

Within our isocontour approach, we have implemented several heuristics to reduce surface calculation and rendering time, such as grouping pixels for potential calculations and restricting the regions in which items influence the potential field. The current implementation works without noticeable lag; items can be dragged, and the surface follows. For our examples, order of 655 nodes, 65–85 sets. For example, it takes on average 65 ms to calculate the virtual edge set, fill the energy field, find the contour, and render the Sub-Saharan Africa set in a window size 6855 pixels. That set has 0 items, and the entire scatter plot has 685 points. The majority of this time is spent creating the virtual edge set. An incremental approach, using A search as in [89], may provide improvements in speed and stability. As the number of items, the screen resolution, or the number of sets increases, so will the rendering time. Additional techniques, such as grouping close items into larger pseudo-nodes, and caching the energy field values between frames, may increase the capacity of the system.

ACKNOWLEDGMENTS

Thanks to the following organizations for supporting this research: iCORE, CFI, NSERC, and SMART Technologies.
KelpFusion

(a) Bubble Sets
(b) Kelp Diagrams
(c) LineSets
(d) KelpFusion (dense)
(e) KelpFusion (medium)

[Meulemans et al., 2013]
Projects

• Feedback

• Designs:
  - Show what is done so far and different ideas along the way
  - Sketch/mockup items that haven't yet been integrated
  - Note interactions/types of visualizations
  - Concentrate on the design of the visualization
  - Think about how your design impacts the ability to answer questions
Data Wrangling

- Problem 1: Visualizations need data
  - Solution: The Web!

- Problem 2: Data has extra information I don't need
  - Solution: Filter it

- Problem 3: Data is dirty
  - Solution: Clean it up

- Problem 4: Data isn't in the same place
  - Solution: Combine data from different sources

- Problem 5: Data isn't structured correctly
  - Solution: Reorder, map, and nest it
Hosting data

- github.com
- gist.github.com
- myjson.com
- Google Drive
- Other services
Why JavaScript?

- Python and R have great support for this sort of processing
- Data comes from the Web, want to put visualizations on the Web
- Sometimes unnecessary to download, process, and upload!
- More tools are helping JavaScript become a better language
Online JavaScript Resources

- [http://learnjsdata.com/](http://learnjsdata.com/)
- Good coverage of data wrangling using JavaScript
LeafletJS Example:
New York City Murder by Precinct