Information Visualization

Information Visualization

Motivation

Information Visualization attempts to visualize a wider spectrum of data types than Scientific Visualization. The latter mainly focuses on so-called physical data. In many cases, the nature of the data implies a certain type of visualization. However, Information Visualization attempts to help users form a mental image about data that has no physical placement. Information has no innate shape and color, so its visualization has a purely abstract character. Information Visualization covers areas, such as visual reasoning, visual data modeling, visual programming, visual information retrieval and browsing, and spatial reasoning.

Information Visualization

Data types

File systems, internet connections, network of streets, and communications systems are examples of connected structures which can be modeled using a graph of different specializations. The most common types of graphs are:

- Undirected graph: this is a tuple $G=\{V,E\}$ with a set of nodes $V=\{v_1,\ldots,v_n\}$ and a set of edges $E=\{(v_i,v_j)\}\subset\{(v_i,v_j)|I,j=1,\ldots,n;\ i\neq j\}$. Often times, there are more data elements attached to the edges.
- Directed graph: this is again a tuple $G=\{V,E\}$ with $V=\{v_1,\ldots,v_n\}$ and $E=V\times V$. Again, additional data can be attached to the edges.
- Trees: a tree is a graph without any cycles.
- Multigraph: both directed and undirected graphs allow the presence of multiple edges connecting the same nodes. These graphs are called multigraphs.

Information Visualization

These data types occur frequently. A simple example is the directory hierarchy on a hard drive. Typical questions are:

- Where am I?
- Where is the file I am looking for?

Other typical applications are:

- Organization chart at a hospital
- Taxonomy of biological species
- Evolutionary trees
- Molecular and genetic charts
- Phylogenic trees
- Biochemical reaction paths

Information Visualization

Besides the type of graph (tree, acyclic directed graph, arbitrary directed graph, undirected graph, multigraph) the size of the graph (number of nodes and edges) is a key point in the visualization.

A user is usually not able to comprehend large graphs from a single image. The visualization then only shows the complexity of the graph.

From a certain size on, there are no suitable layout algorithms anymore simply because there is not enough space on the screen. Hence, we need to reduce the complexity first.

Information Visualization

Graph drawing

The basic problem of graph drawing can be described easily:

Problem: Let a set of nodes and a set of edges be given. Compute the positions for the nodes and the curves representing the edges.

It is not easily explained what a good layout is. Battista at al. [IEEE TVCG 6(1): 24~ , G. Di battista, P. Eades, R. Tamassia, and I. G. Tollis, „Algorithms for Drawing Graphs: An Annotated Bibliograph", Geometry: Theory and Applications, vol. 4, no. 5, pp. 235-282, 1994] list hundreds of papers devoted to this issue. The difficulty is to define properties of the graph and classify the layouts. A typical property of a graph is being planar. Hence, we need a fast algorithm which can check that property. For an undirected graph a complexity of O(n) can be achieved [J. Hopcroft and R. E. Tarjan, „ Efficient Planarity Testing", J. ACM vol. 21, no. 4, pp. 549-568, 1974]. A possible layout algorithm could be based on a regular grid and use only integer coordinates for the nodes.

Information Visualization

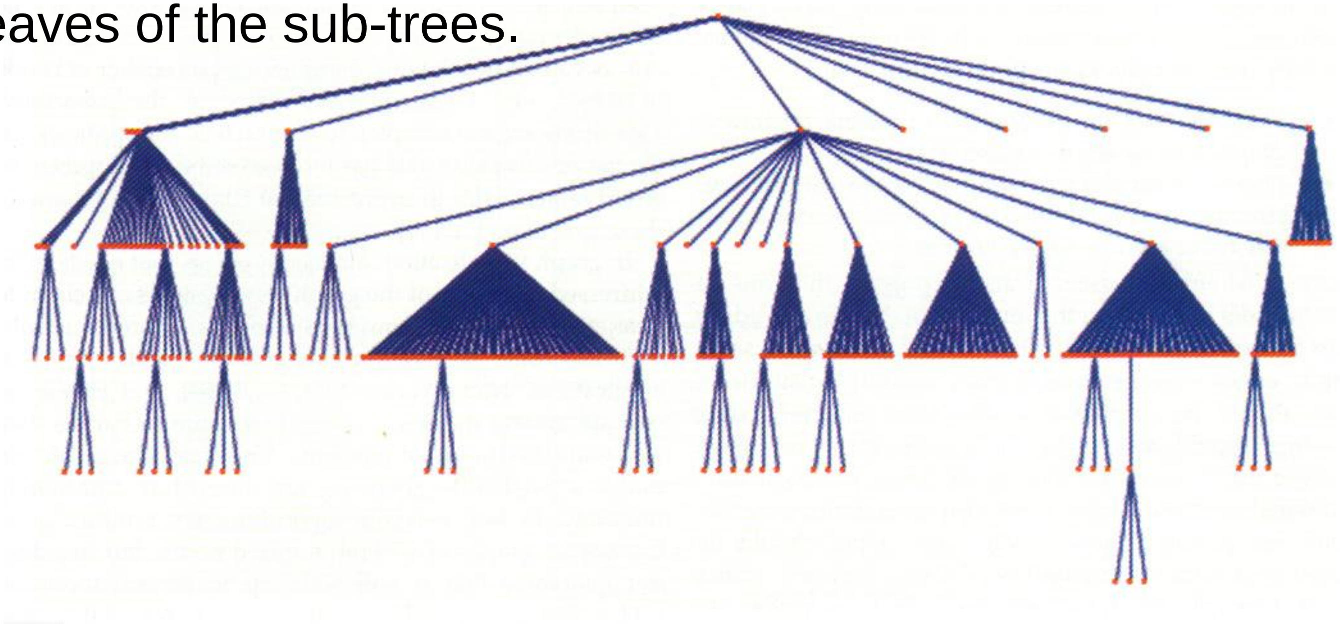
For the visualization, however, the property of being planar is not of that much importance. But the minimization of intersections between the curves representing the edges is a major goal.

In addition, the layout influences the perception of the graph.

Information Visualization

Tree drawing

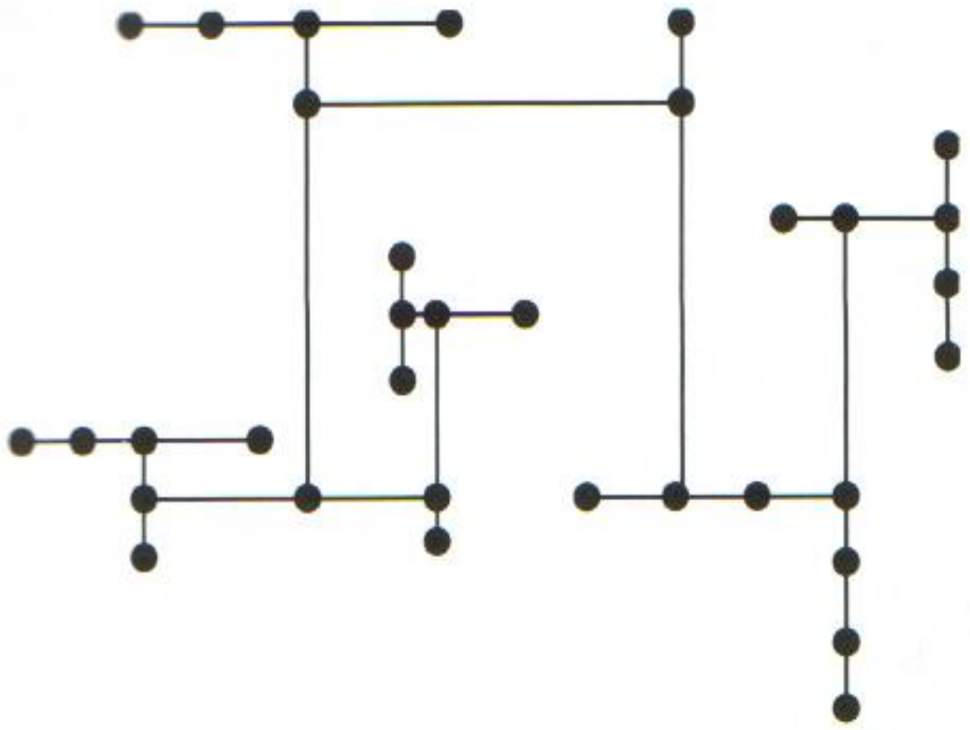
The algorithm of Reingold and Tilford [E. M. Reingold and J. S. Tilford, „Tidier Drawing of Trees“, IEEE Trans. Software Eng., vol7, no2, pp. 223-228, 1981] generates a classic tree representation. All nodes of the same level are located at the same height. The horizontal space is split up according to the number of leaves of the sub-trees.



A tree layout for a moderately large graph.

Information Visualization

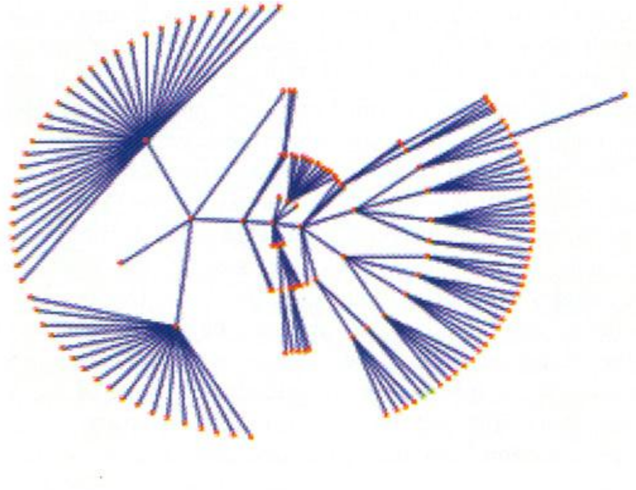
It is also possible to use an H-like pattern: [P. Eades, „Drawing Free Trees“, Bulletin of the Inst. For the Combinatorics and Its Applications, pp. 10-36, 1992]



H-tree layout.

Information Visualization

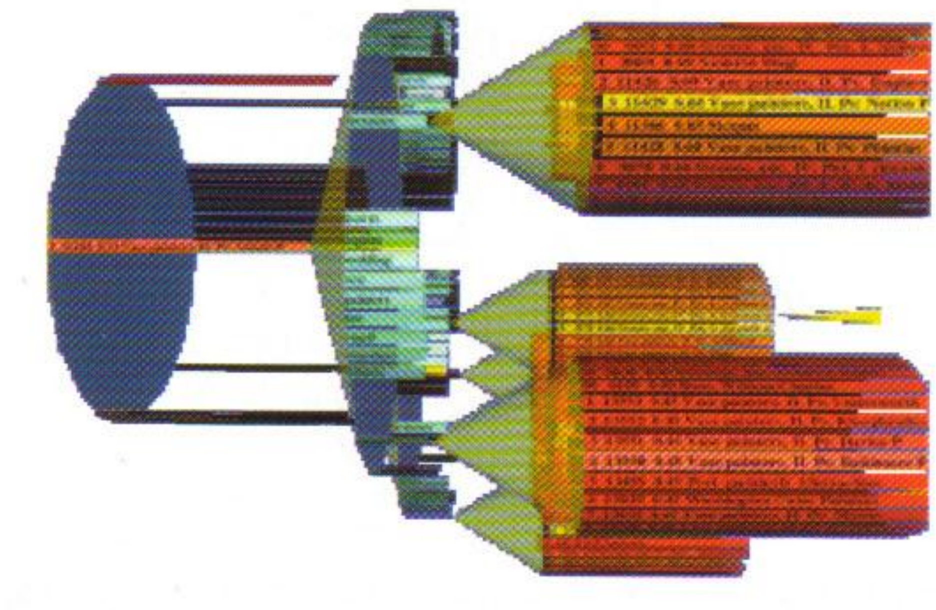
Another variant uses a radial pattern. The root of the tree is located at the center of the image while the nodes are placed on concentric circles. In addition, the algorithm avoids intersections by choosing fixed sections for the sub-trees. It is possible to weaken the last condition to achieve better results [I.Herman, G. Melancon, M. M. De Ruiter, and M. Delest, „Latour-A Tree Visualization System“, Proc. Symp. Graph Drawing GD'99, pp. 392-399, 1999. A more detailed version in: Reports of the Centre for Math. And Computer Sciens, Report number INS-R9904, available at: http://www.cwi.nl/InfoVisu/papers/LatourOverview.pdf, 1999]



Radial view.

Information Visualization

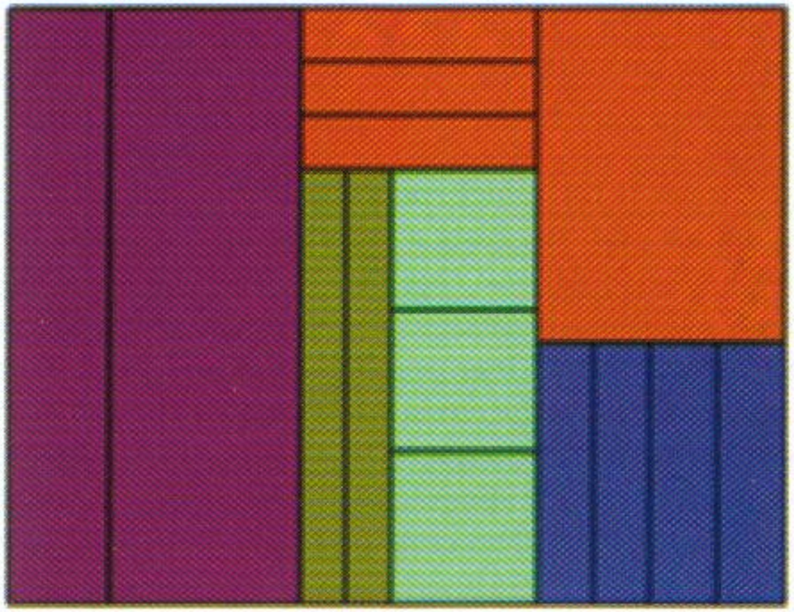
In a cone tree [J. Carriere and R. Kazman, „Research Report: Interacting with Huge Hierarchies: Bryond Cone Trees“, Proc. IEEE Conf. Information Visualization '95, pp. 74-81, 1995], siblings are located along a circle which, including the parent node, results in a cone.



A cone tree. (Courtesy of M. Hemmje, GMD, Germany [59].)

Information Visualization

A tree map [B. Johnson and B. Schneiderman, „Tree-Maps: A Space-Filling Approach to the Visualzation of Hierarchical Information Structures“, Proc. IEEE Visualization '91, pp. 275-282, 1991] assigns a rectangle to each sub-tree which is then sub-divided further. Using the size of the rectangles, additional information can be conveyed.



Tree-map: rectangles with color belong to the same level of the (tree) hierarchy. (Adapted from Johnson and Schneiderman [72]).

Information Visualization

The previously discussed methods are predictable, i.e. they give the exact same results for the same input which is an important property in visualization.

This is no longer true when using simulated annealing or spring models.

Information Visualization

Layout in the hyperbolic plane

Even though a radial layout of the tree utilizes the space more efficiently compared to classic approaches, it still does not provide sufficient results. The number of nodes grows exponentially per level, while the circle grows linearly within the Euclidian plane. In the hyperbolic plane however, the growth is exponential!

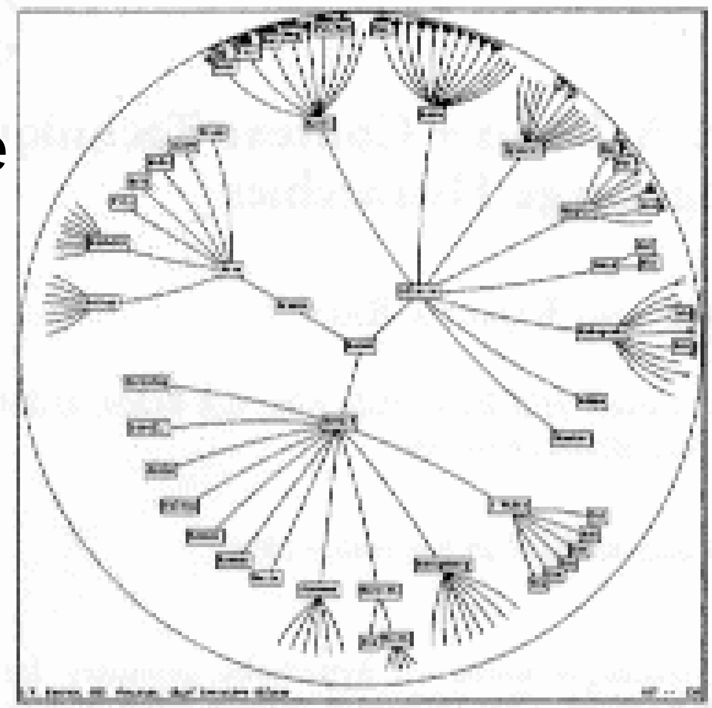


Figure 1. A partial organization chart of Xerox (ca. 1988)

Information Visualization

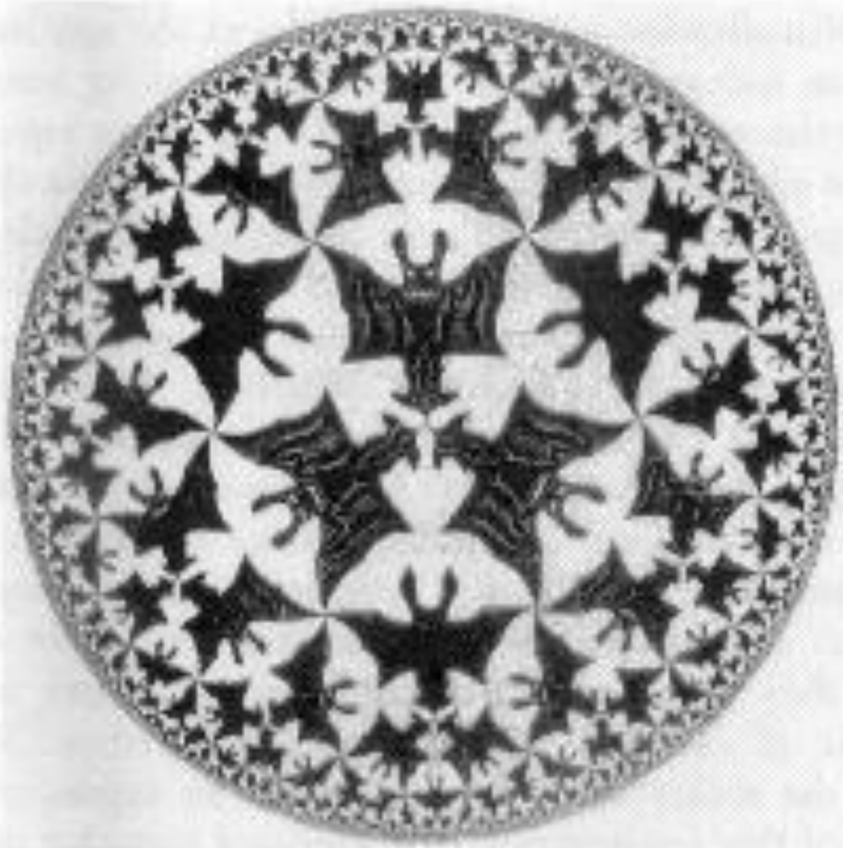


Figure 2. Original inspiration for the hyperbolic browser. Circle Limit IV (Heaven and Hell), 1960, ©1994 M.C. Esher/Cordon Art—Baarn—Holland. All rights reserved. Printed with permission

Information Visualization

Basically, the hyperbolic layout is a radial layout where the size gets smaller rapidly when approaching the boundary of the circle. Here, the circular section is determined based on the number of children without considering their grand-children in order to be able to lay out even large trees.

Information Visualization

It is possible to navigate through such a layout by moving the center of projection.

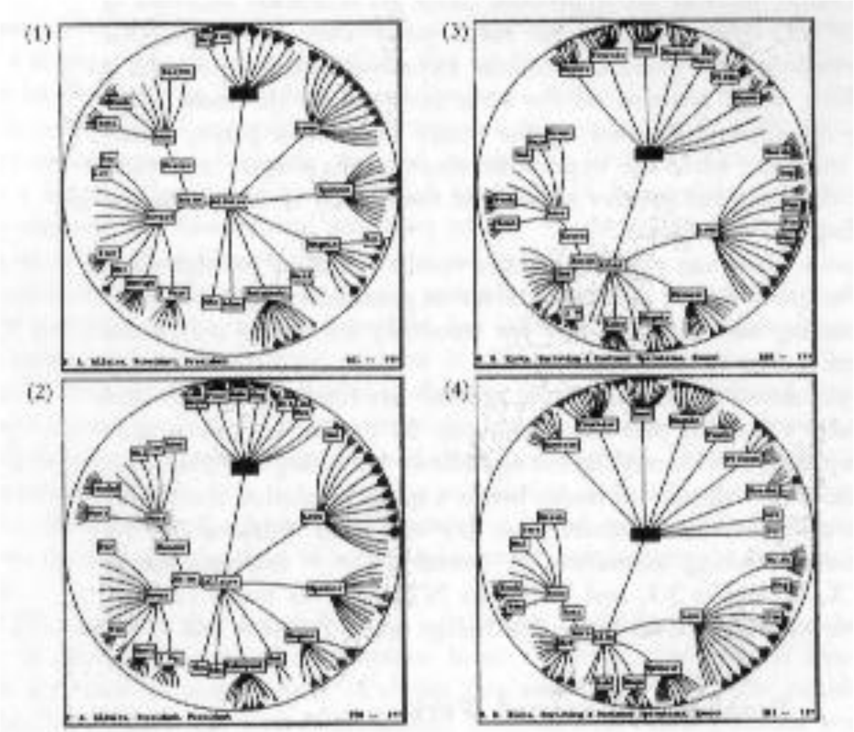


Figure 3. Clicking on the blackened node brings it into focus at the center

We will see later that we need to avoid a rotation of the origin after several movements of the center.

Information Visualization

The layout is arranged in the hyperbolic plane and then mapped onto a circle within the Euclidian plane. This maps circles onto circles so that it is possible to leave some space for annotations. An ellipse formed by the siblings, parent, and middle child can also be used for annotations.

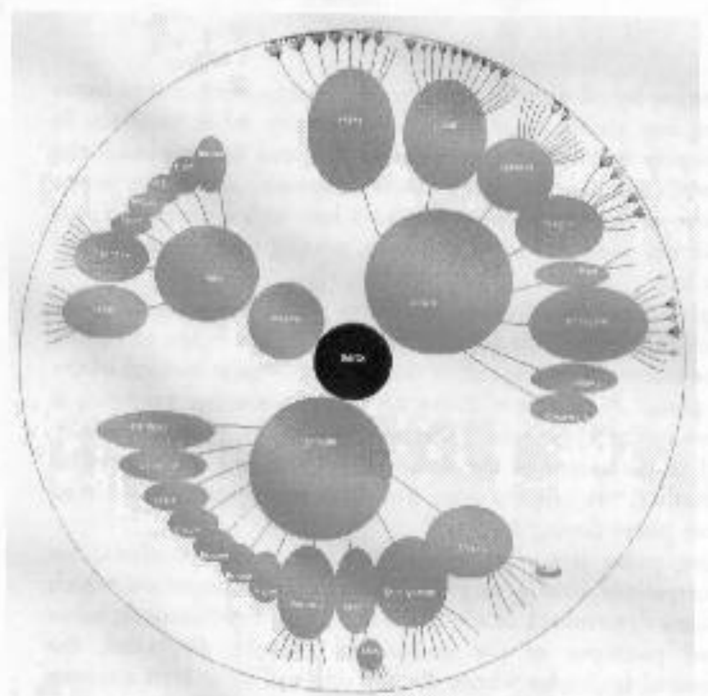


Figure 8. The regions available to nodes for displaying information

Information Visualization

Hyperbolic geometry

Euclid postulated five axioms for his geometry:

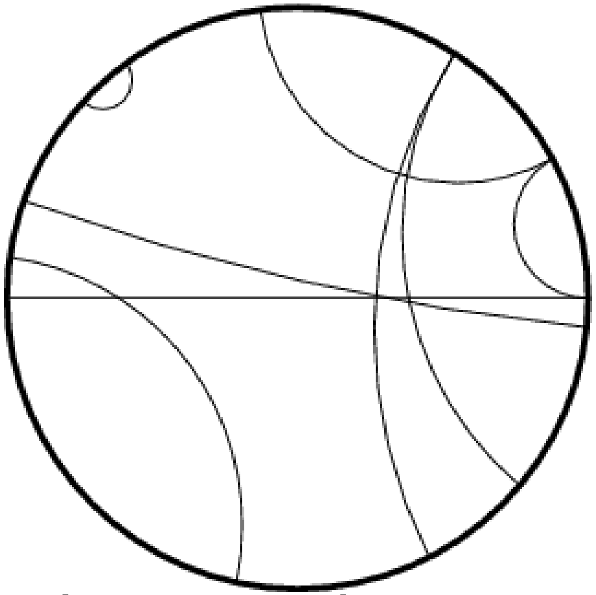
- There is one line segment that connects two points.
- Every line segments can be extended to a infinite straight line.
- For every line segment, a circle can be constructed that has the segment as its radius and one end point as its center.
- All orthogonal angles are congruent.
- For every straight line and a given point that is located on the line, there is exactly one infinite straight line that does not intersect the existing one.

By replacing the last axiom with one allowing that there are several parallel lines we get the hyperbolic geometry.

Information Visualization

Poincaré model

The model of a hyperbolic plane according to Poincaré consists of a unit disc C (complex)



$$P = \{ z \in C \mid \|z\| < 1\}$$ “points”

with the locally dependent metric

$$ds^2 = \frac{(dx^2 + dy^2)}{(1 - x^2 - y^2)}$$ “distance”

The “straight lines” are circular arcs orthogonal to the boundary.

Since it is a conform model the angles between the circular arcs determine the angles of straight lines in the hyperbolic plane.

Information Visualization

For the navigation, we need a transformation of the hyperbolic plane. These can be described by

$$T_{P,\theta} : C \to C$$

$$z \to \frac{\theta z + P}{1 + \overline{P}\,\theta z}$$

with $P, \theta \in C$, $|P| < 1$, $|\theta| = 1$ and $\overline{p}$ as its conjugated point.

This is a rotation of angle θ and a subsequent translation of the origin to P.

When combining two of these transformations we get

$$P' = \frac{\theta_2 P_1 + P_2}{\theta_2 P_1 \overline{P_2} + 1} \qquad \theta' = \frac{\theta_1 \theta_2 + \theta_1 \overline{P_1} P_2}{\theta_2 P_1 \overline{P_2} + 1}$$

Information Visualization

Layout

The layout is realized as a recursive procedure with a circular section as its parameter. The circular section is described by a vertex, the end point of the line segment cutting the section in half, and the half the angle.

A simple layout uses the vertex as the location for the parent node und sub-divides the section according to the number of children. Better results are usually achieved when using a logarithmic scale when sub-dividing. The distance to the child is determined by

$$d = \sqrt{\frac{(1-s)^2 \sin(a)^2}{2s} + 1 - \frac{(1-s^2)\sin(a)}{2s}}$$

where a is half of the angle of the child’s circular section and s is the desired distance between the child and the edge of the circular section. Good results can be achieved using $s=0.12$.

Information Visualization

Visualization

The visualization of the graph is achieved by drawing the nodes (if desired with annotations) and the edges as circles which correspond to hyperbolic line segments.

The computation of the center of a circle for complex numbers $a,b \in P$ uses the following formula:

$$d = \mathrm{Re}(a)\,\mathrm{Im}(b) - \mathrm{Re}(b)\,\mathrm{Im}(a)$$

$$c = \frac{i}{2} \cdot \frac{(a(1 + \left\|b^2\right\|) - b(1 + \left\|a^2\right\|)}{d}$$

Information Visualization

Navigation

The layout of the graph in the hyperbolic plane is not changed. Only the mapping of the complex unit disc is manipulated using the previous transformation.

We start with $T_{0,1}$. If the user moves the mouse from s to e and the point p is not supposed to be rotated then we get:

$$a = T_{-p,1}(s)$$

$$b = \frac{\operatorname{Re}((e - a)(1 - \overline{a}\,\overline{e}))i}{1 - (ae)^2}$$

$$T = T_{-p,1} \circ T_{b,1}$$

For a smooth transition it is possible to introduce intermediate steps.

Information Visualization

Literature

[Lamping, Rao, “The Hyperbolic Browser : A Focus + Context Technique for Visualizing Large Hierarchies”, Journal of Visual Languages and Computing 7, 33-55, 1996]

Additional hyperbolic approaches:

[Munzner, “HB: Laying Out Large Directed Graphs in 3D Hyperbolic Space”, IEEE Visualization '97 Proceedings, IEEE CS, 1997, 2-10]

Information Visualization

Layout of general directed graphs

For these types of graphs, a suitable layering is identified first which assigns an integer number to every node. Most methods are based on extraction of an acyclic sub-graph which contains all nodes. This way, all nodes receive a number and are ordered in rows from top to bottom so that all edges of the acyclic graph point downward. This arrangement is used for minimizing the steps, often only up to the next level. This problem, however, is NP hard.

A heuristic [W. Tutte, „How to Draw a Graph“, Proc. London Math. Soc., vol. 3, no. 13, pp. 743-768, 1963] defines an order for the first and last level and requires that every node is located at the center of gravity of its neighbor (w.r.t. the graph). This results in a system of linear equations. A comparison of different heuristics is given by Laguna and Marti [M. Laguna and R. Marti, „Heuristics and Meta-Heuristics for 2-Layer Straight Line Crossing Minimization“, URL: http://www-bus.colorado.edu/Faculty/Laguna/, 1999].

Information Visualization

Spring-based methods

These methods model nodes and edges as physical entities connected by springs. This results in an optimization problem which can only be solved locally (hence not predictable). In addition, these methods tend to be slow with a complexity of $O(n^3)$ [A. Frick, A. Ludwig, and H. Mehldau, „A Fast Adaptive Layout Algorithm for Undirected Graphs“, Proc. Symp. Graph Drawing GD '93, pp. 389-403, 1994].

Information Visualization

Layout of undirected graphs

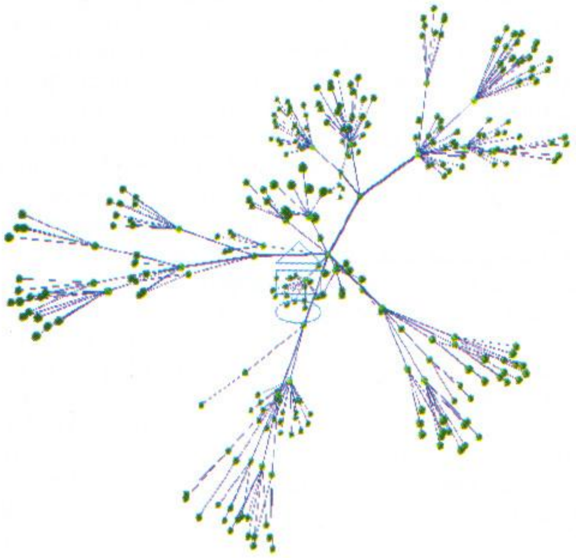
For undirected graphs, we usually start with a spanning tree. This tree is then laid out. The edges can be assigned weights before computing the spanning tree. Good algorithms have complexities $O(N \log N)$ or $O(E \log N)$ (E: number of edges, N: number of nodes).

Information Visualization

3-D Layout

It is also possible to use 3-D-based layouts. Besides the cone tree representation, two approaches are shown here. However, current input devices are not very suitable for navigation.

Information Cube. (Courtesy of J. Rekimoto, Sony Computer Science Laboratory, Inc., Japan [104].)

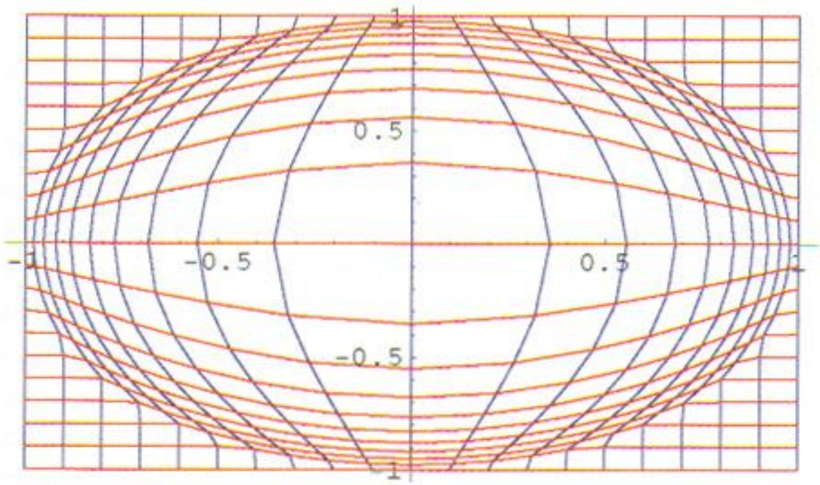


3D version of a radial algorithm. (Courtesy of S. Benford, University of Nottingham, U.K.)

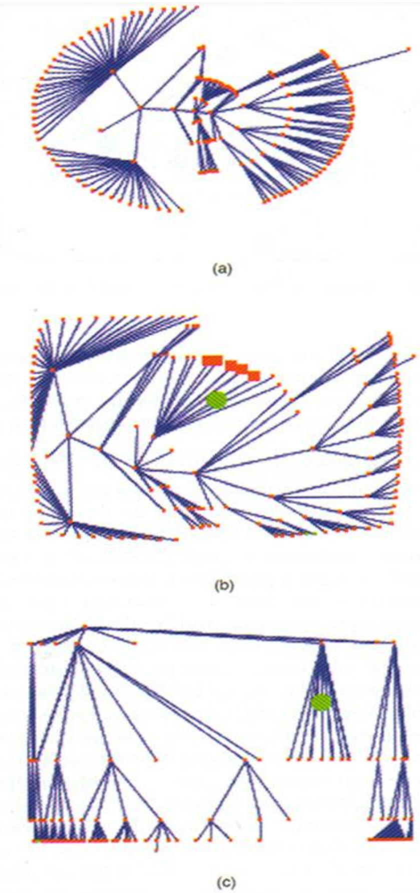
Information Visualization

Exploration

For large trees, a fish-eye view can help explore the data.



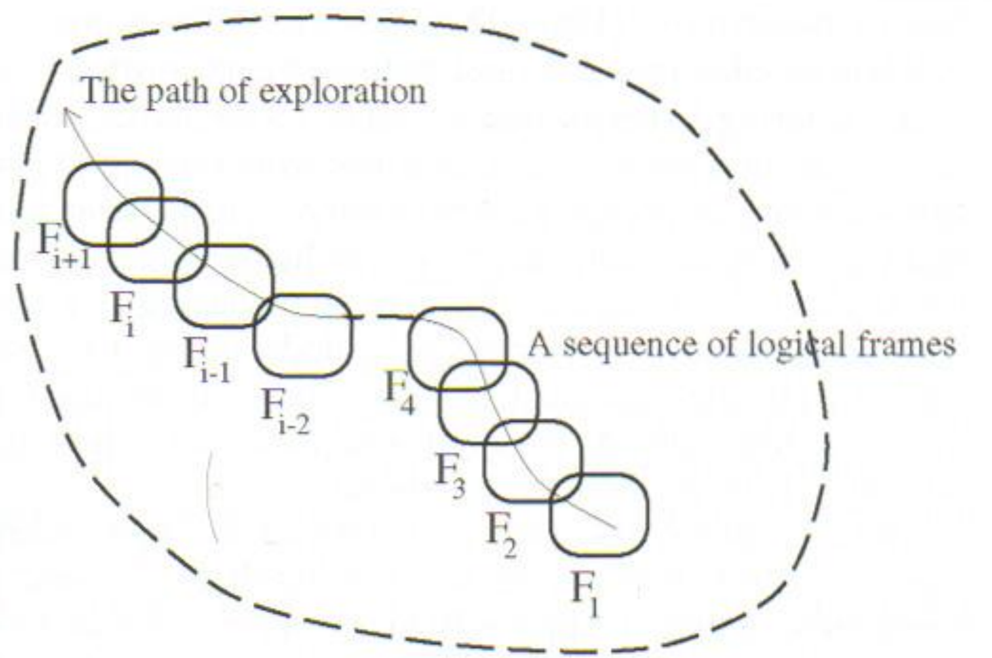
Fisheye distortion of a regular grid of the plane. The distortion factor is 4.



Fisheye distortion. (a) Represents the graph without the fisheye. (b) Uses polar fisheye, whereas (c) uses Cartesian fisheye with a different layout of the same graph. The green dots on (b) and (c) denote the focal points of the fisheye distortion. Note the extra edge-crossing on (b).

Information Visualization

In addition, a windowing technique can be used for exploring large graphs.



Exploration of a huge graph. (Adapted from Huang et al. [69].)

Information Visualization

Clustering

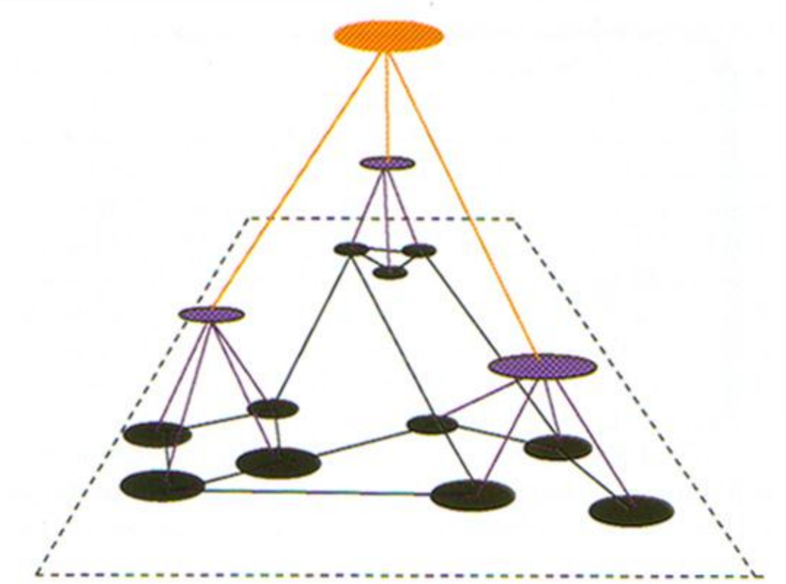
If the graph is too large to display it can be simplified. Usually, this is done by clustering nodes. Man differentiates different approaches:

- Structural clustering: combine nodes based on the structure
- Content-based clustering: combine semantically similar nodes

Almost all techniques are based on structural clustering, since it is easier to implement and the method can be applied to any graph independently of the application.

Information Visualization

The algorithm generates disjoint clusters. The clusters are then form a new graph by using each cluster as a node and drawing an edge if there is an edge between elements of two different clusters.



A structure induced by hierarchical clustering. (Adapted from Eades and Feng [37].)

Information Visualization

For the clustering, a metric for the nodes is required. This metric can be structural or content-based. Clusters are then formed based on the “distance” between nodes using a pre-defined threshold.

Information Visualization

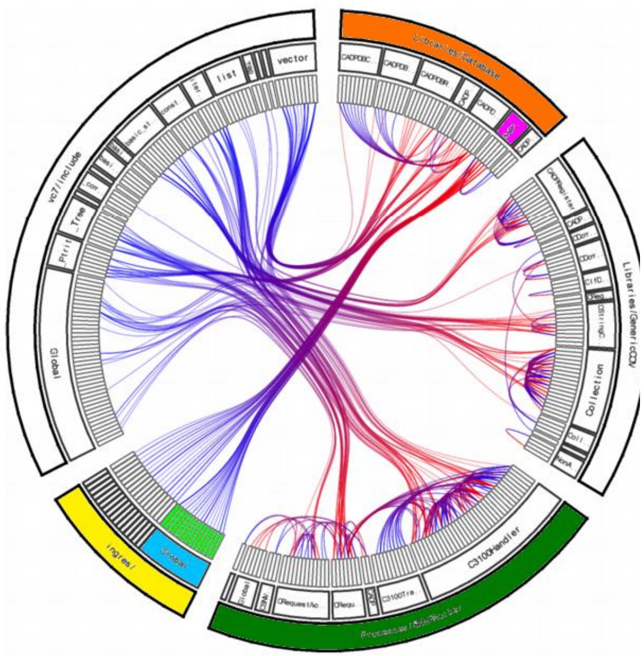
It is also possible to assign a value of relevance to the nodes and edges. There are three different approaches:

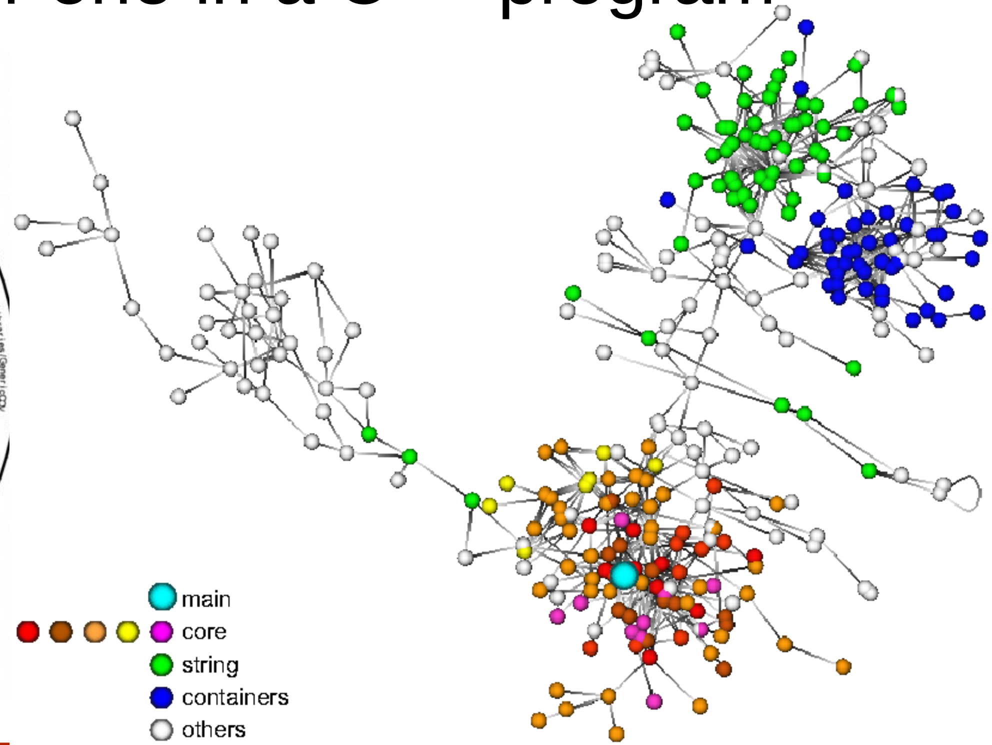
- Ghosting: less relevant nodes and edges are shifted towards the background
- Hiding: less relevant elements are omitted
- Grouping: less relevant elements are combined

Information Visualization

Applications

Call graph: graph connections represent that a method calls another one in a C++ program





Images courtesy of Alexandru Telea

Information Visualization

Multidimensional visualization

The treatment of multidimensional data sets is an important data visualization issue. Each point in a data set is described by an n-dimensional coordinate, where $n>3$. Here we assume that each coordinate is an independent variable, and that we wish to visualize a single dependent variable. An application of multi-dimensional data is financial visualization, where we might want to visualize return on investment as a function of interest rate, initial investment, investment period, and income, to name just a few possibilities.

Information Visualization

There are two fundamental problems that we must address when applying multidimensional visualization. These are the problems of *projection* and *understanding*.

The problem of projection is that in using computer graphics we have two dimensions in which to present our data. Using 3-D graphics we can give the illusion of three dimensions.

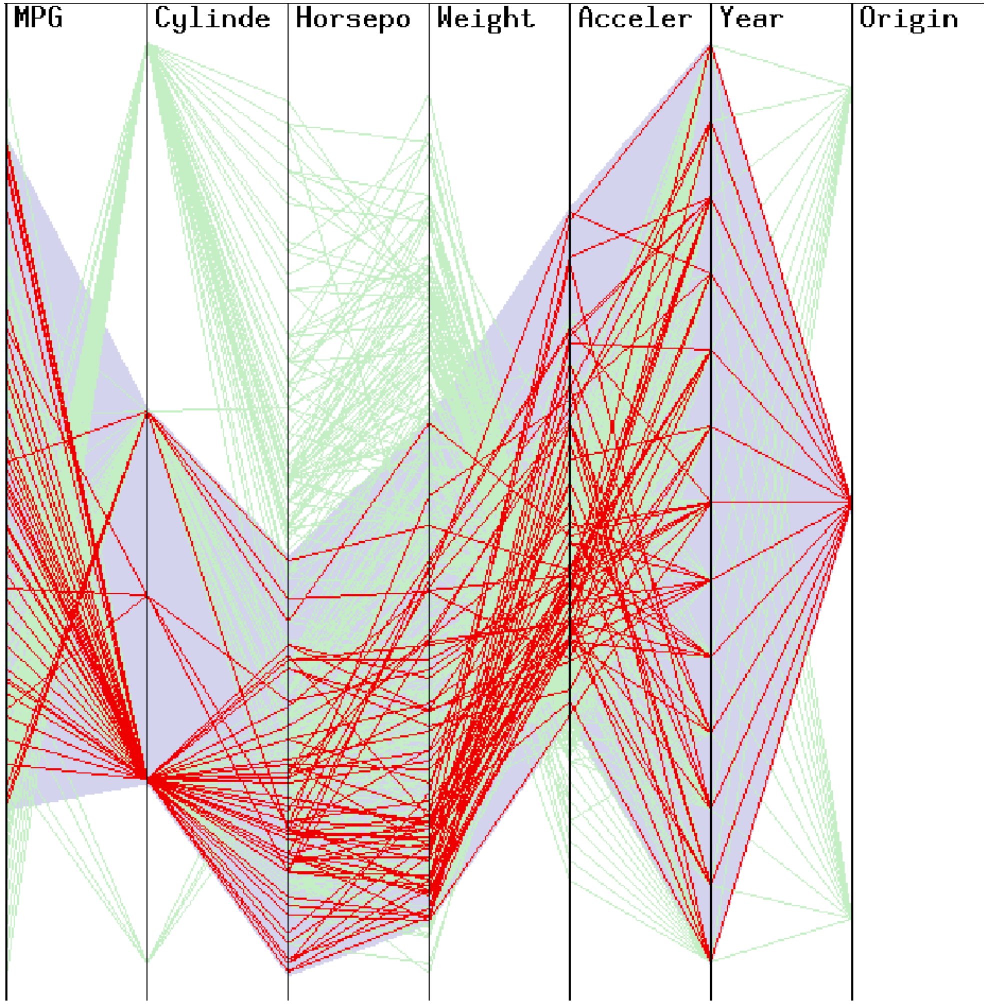
The problem of understanding is that humans do not easily comprehend more than three dimensions, or possibly three dimensions plus time.

Information Visualization

Parallel coordinates

One approach to multidimensional visualization is the use of *parallel coordinates*. Instead of plotting points on orthogonal axes, the i-th dimensional coordinate of each point is plotted along separate, parallel axes. When using parallel coordinates, points appear as lines. As a result, plots of n-dimensional points appear as sequences of line segments that may intersect or group to form complex fan patterns. Unfortunately, if the number of points becomes large, and the data is not strongly correlated, the resulting plots can become a solid mass of black, and any data trends are drowned in the visual display.

Information Visualization



Information Visualization

Another multivariable technique is using glyphs. This technique associates a portion of the glyph with each variable. Although glyphs cannot generally be designed for arbitrary *n*-dimensional data, in may applications we can create glyphs to convey the information we are interested in.

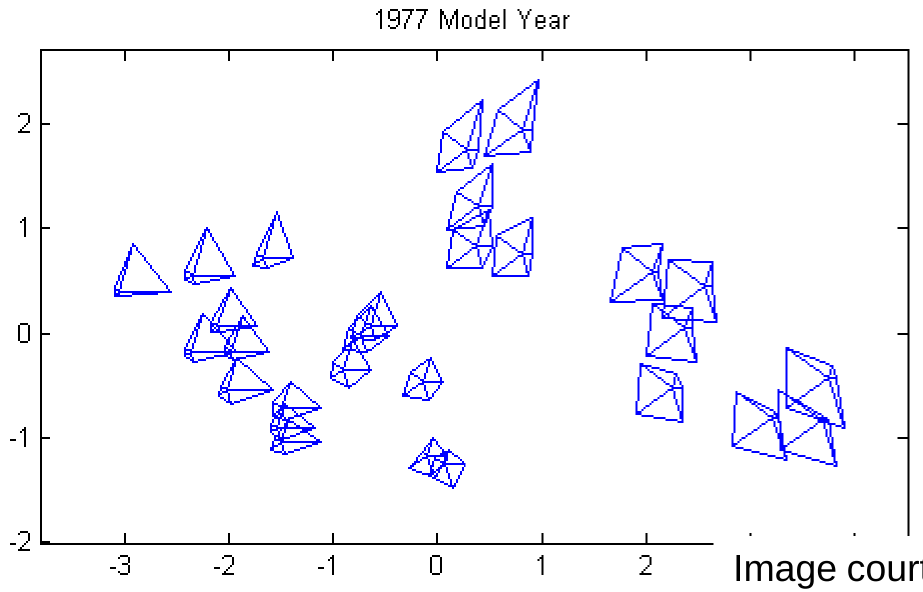


Image courtesy of mathworks

Information Visualization

Text Visualization

Text is an important attribute in Information Visualization data sets. The information contained in this data can be structured into three categories: **content**, **structure**, and **metadata**. The content describes the information contained in the text itself. The structure characterizes how the text is organized. Finally, metadata describes all types of information related to the text that are not contained in the text itself.

Information Visualization

Text Visualization (continued)

A different dimension of text visualization concerns the origin of the data. All types of text-related information can be already present in the document to visualize, or can be computed using various text-analysis methods in order to support a certain task. This process falls within the data-enrichment step of the visualization pipeline. Text analysis is an extremely wide topic, including techniques that range from neural networks and statistical analysis to lexical, syntactic, and semantic analysis and natural language processing.

Information Visualization

Visualizing Program Code

Given the high prominence and complexity of source code in the software industry, it is natural to consider how visualization can aid its comprehension. Source code has several particular properties, including the following:

- Exact: strictly defined grammars with non-ambiguous semantics
- Large-scale: can have millions of lines of code
- Relational: e.g. dependencies of packages, interfaces of modules
- Hierarchical: e.g. hierarchies of data structures or classes
- Heavily attributed: many attributes that express their semantics

Information Visualization

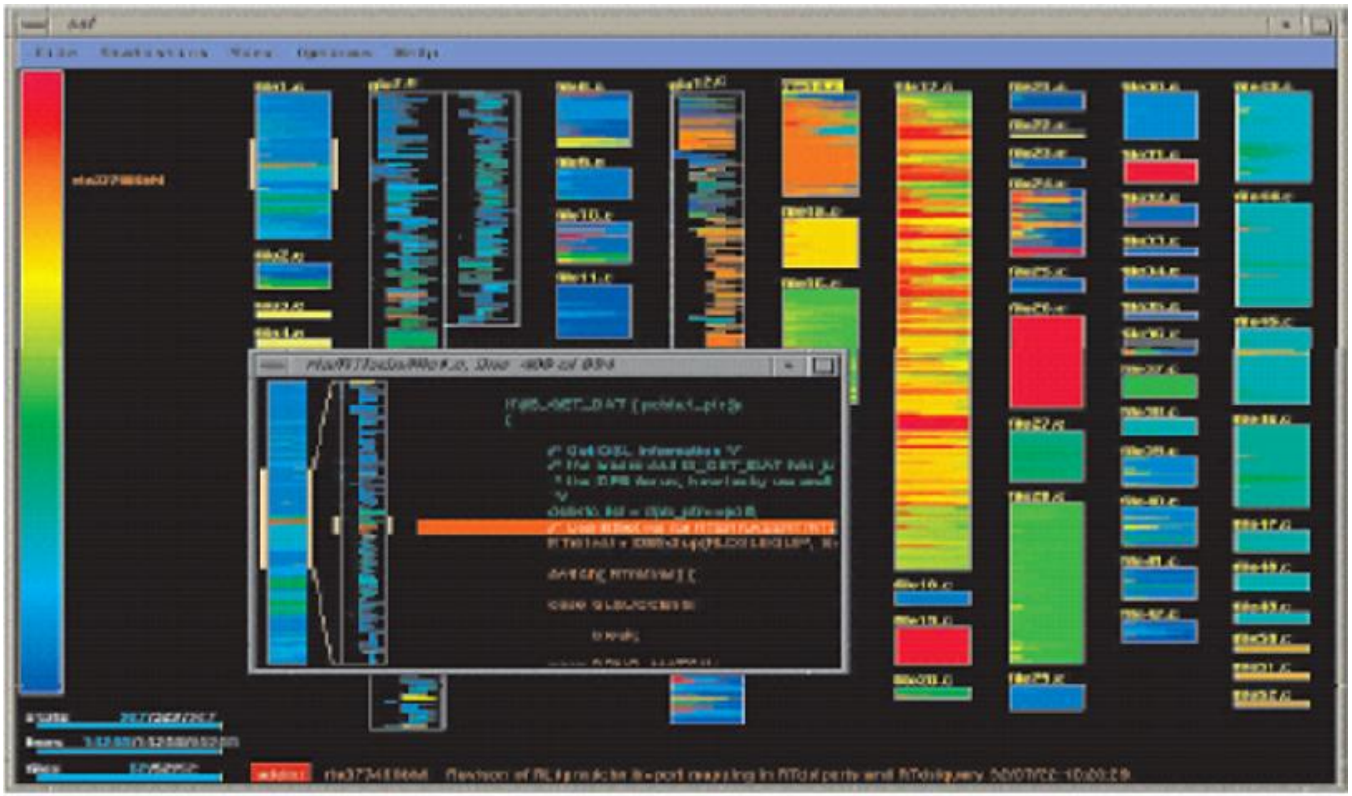


Image courtesy of Alexandru Telea

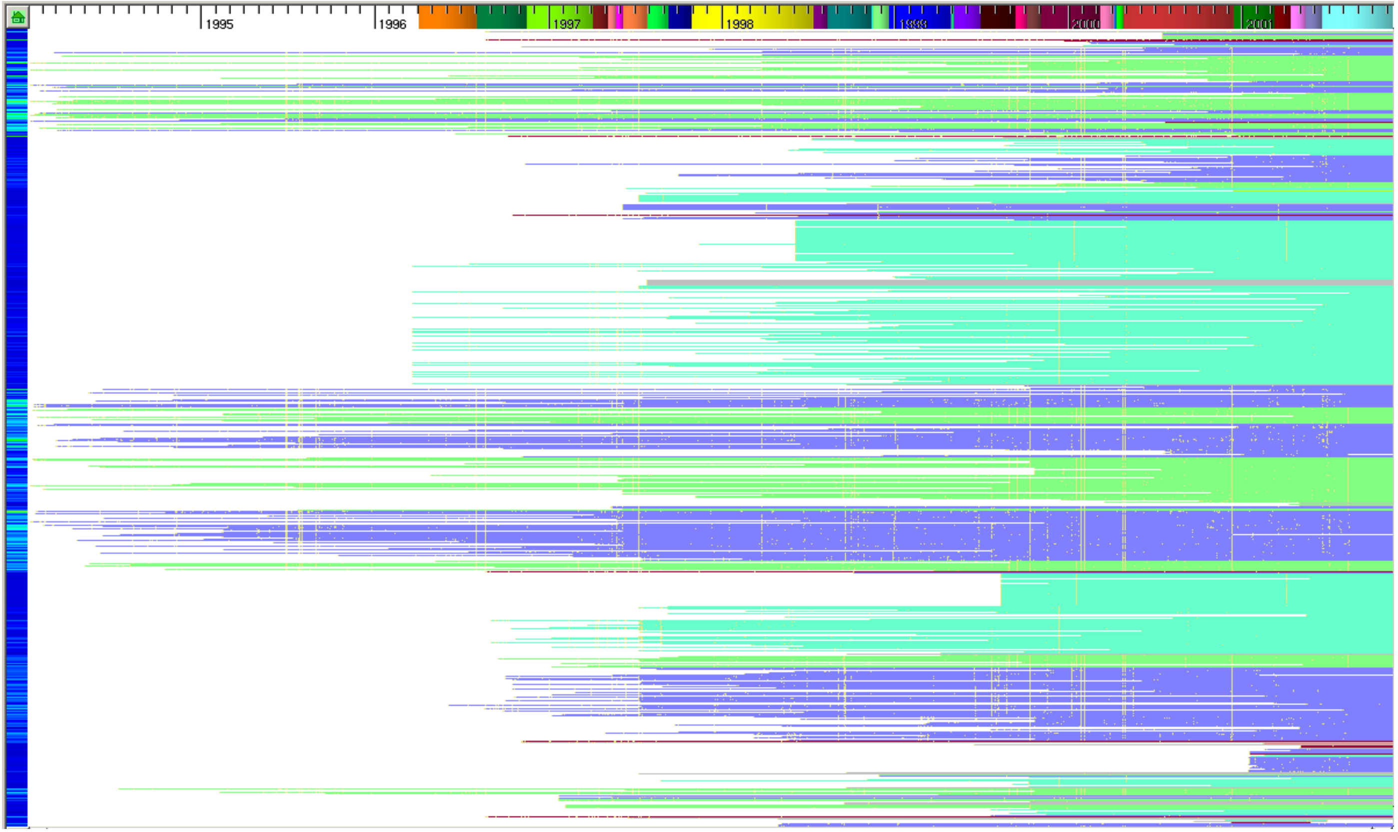
Visualization of C source code using the SeeSoft tool. Color shows the code age (red=new, blue=old). The smaller window shows details for a region in focus in the form of actual source code text.

Information Visualization

Visualizing Software Evolution

Software evolution can be plotted over time. A two-dimensional layout is used, where the x-axis represents the time during which the code has changed, and the y-axis the files in the project. Every file in the code base is mapped to a horizontal pixel line, partitioned in several segments. These pixel strips can be stacked in several orders along the y-axis. In the following image, the order follows a depth-first traversal of the code base. Hence, pixel strips close along the y-axis correspond to files situated at a small distance in terms of their directory path.

Information Visualization



Visualization of the evolution of the VTK software project. Yellow dots indicate the file modification events.

Image courtesy of Alexandru Telea

Information Visualization

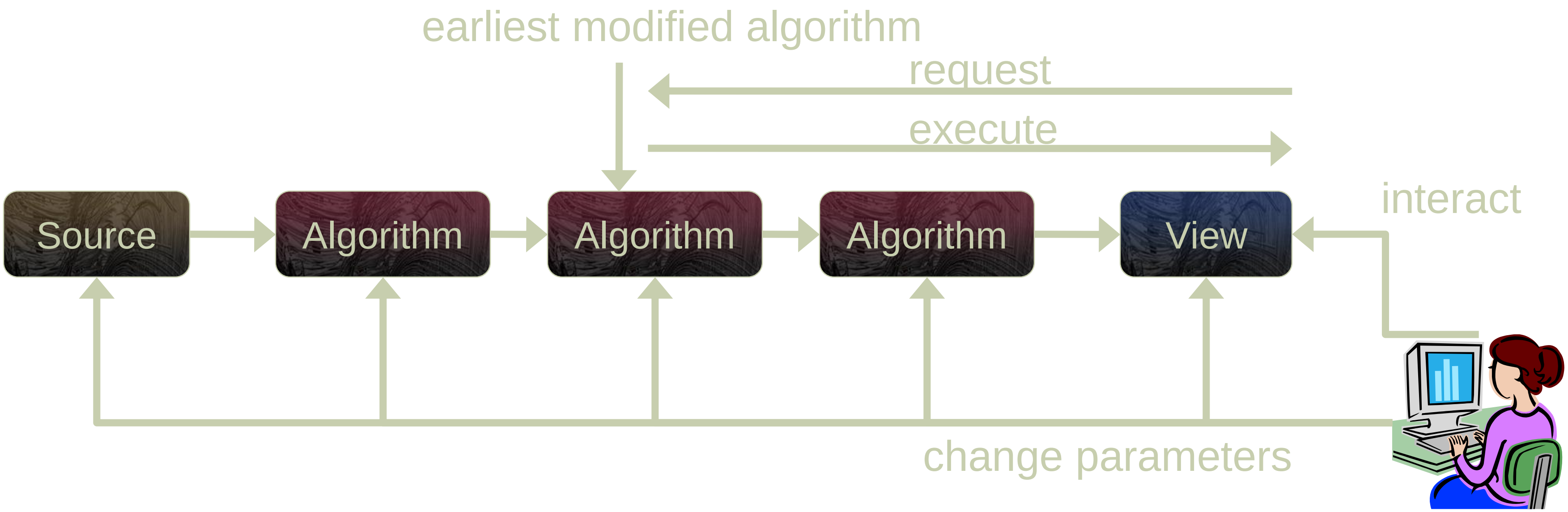
Similarly authorship can be visualized. In the following image, files have been sorted along the y-axis in decreasing order of activity. Each file version is colored to show the ID of the author who committed that version to the repository, i.e., who was responsible for the respective file changes. This color mapping lets us quickly discover who were the most active authors.

Information Visualization

Visualization of author contributions in the VTK software project.

Image courtesy of Alexandru Telea

VTK Pipeline (Sidebar)



Demand-driven

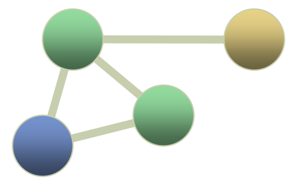
Extensible, component design

Shallow copy of data

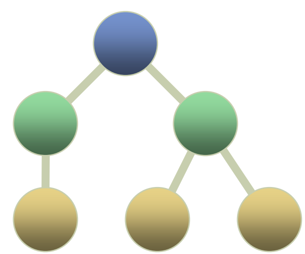
Data Structures

vtkTable

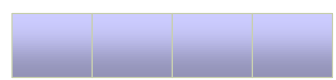
vtkGraph (network)



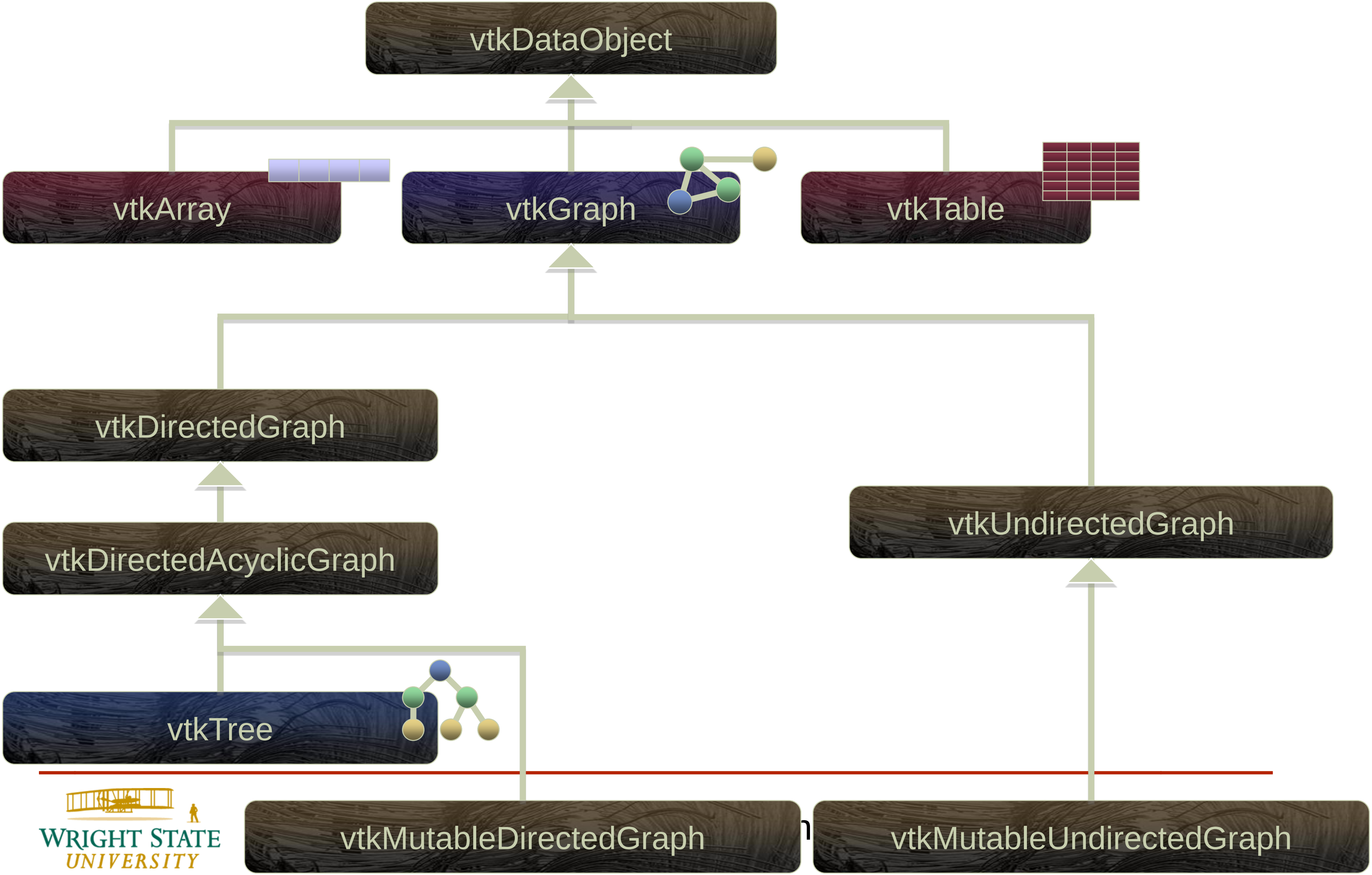
vtkTree (hierarchy)



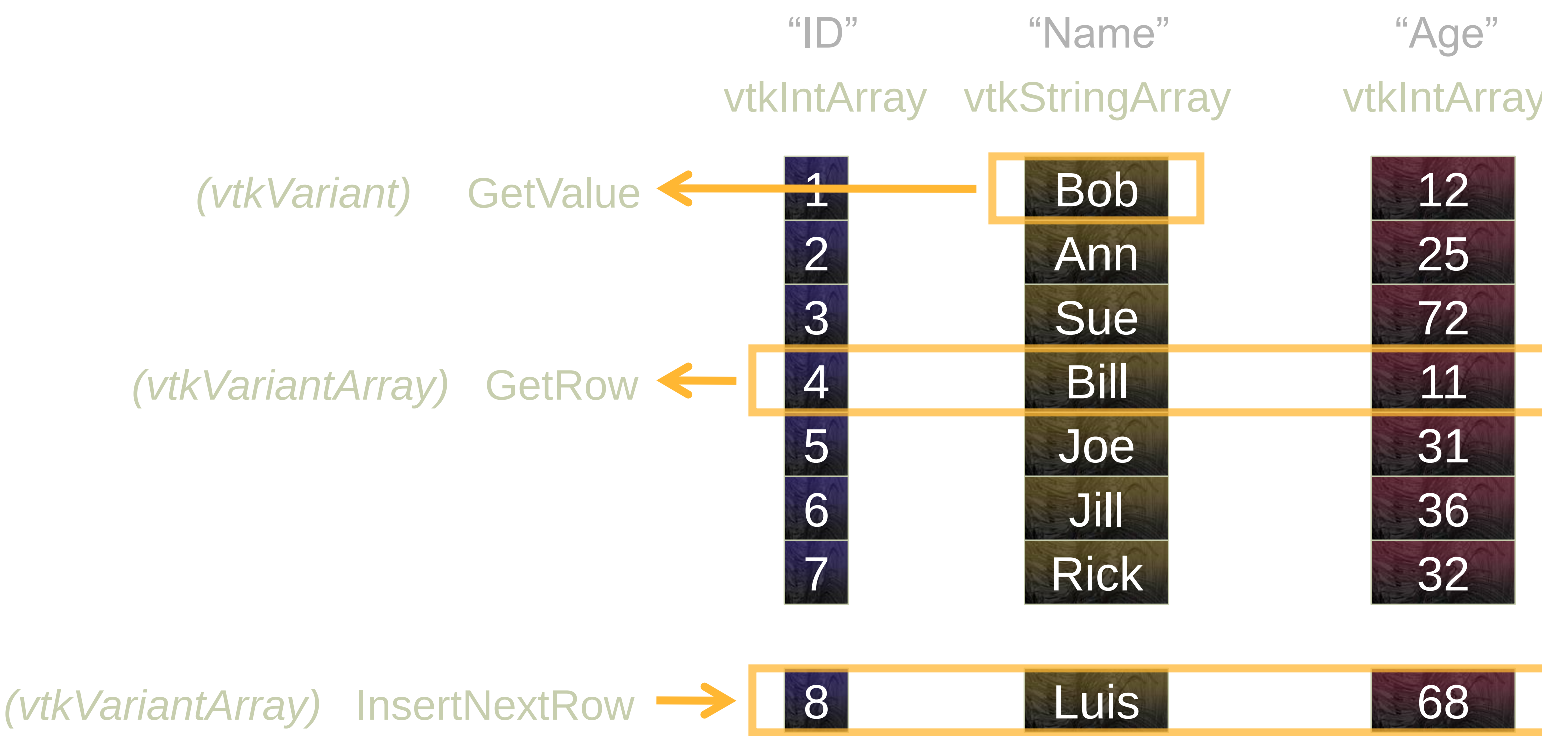
vtkArray



Data Structures



vtkTable

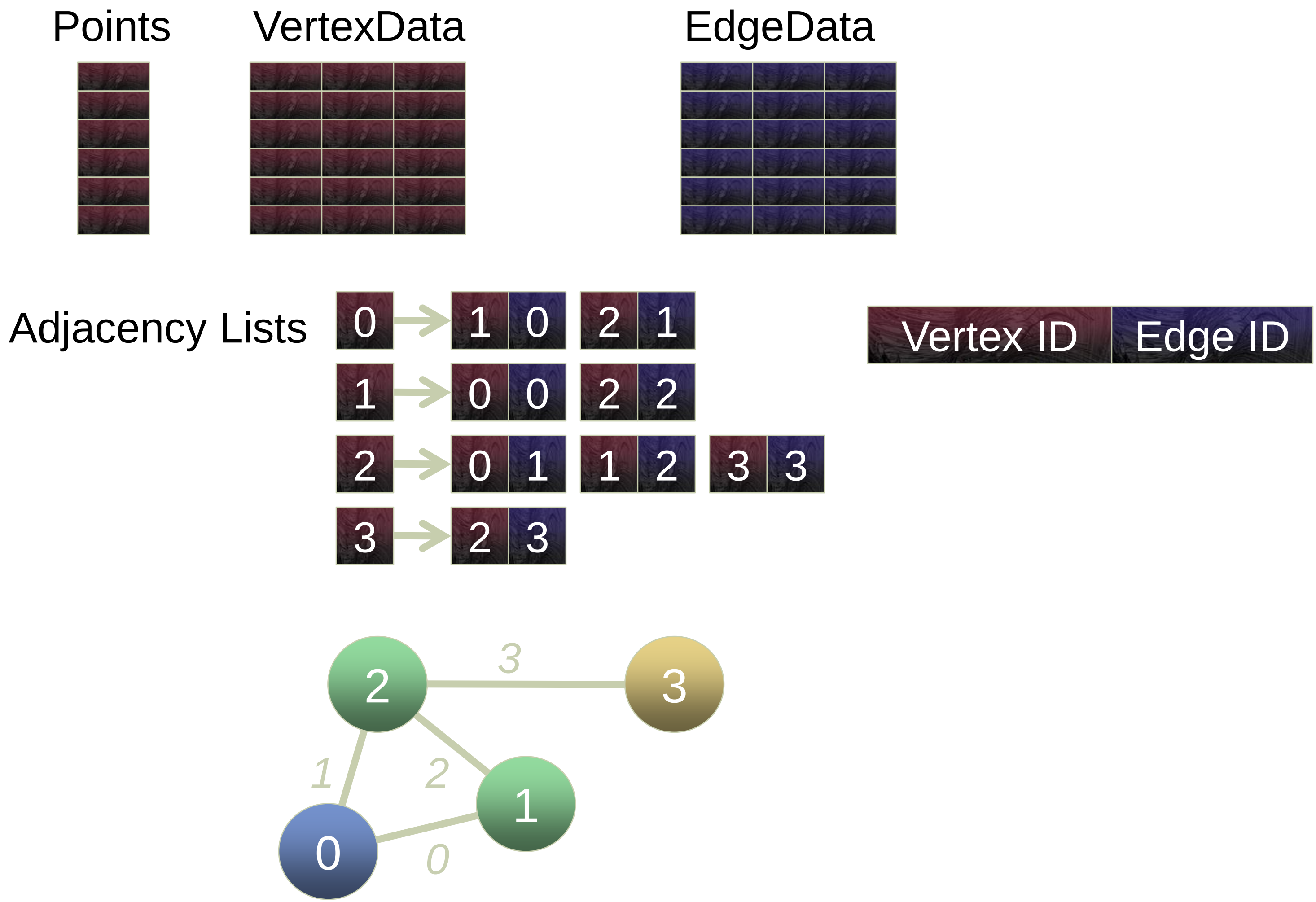

"ID"
vtkIntArray
"Name"
vtkStringArray
"Age"
vtkIntArray
(vtkVariant) GetValue
(vtkVariantArray) GetRow
(vtkVariantArray) InsertNextRow
1 Bob 12
2 Ann 25
3 Sue 72
4 Bill 11
5 Joe 31
6 Jill 36
7 Rick 32
8 Luis 68

Creating a vtkTable

```
vtkTable* t = vtkTable::New();
vtkIntArray* col1 = vtkIntArray::New();
col1->SetName("ID");
col1->InsertNextValue(0);
col1->InsertNextValue(1);
t->AddColumn(col1);

vtkStringArray* col2 = vtkStringArray::New();
col2->SetName("Name");
col2->InsertNextValue("a");
col2->InsertNextValue("b");
t->AddColumn(col2);
```

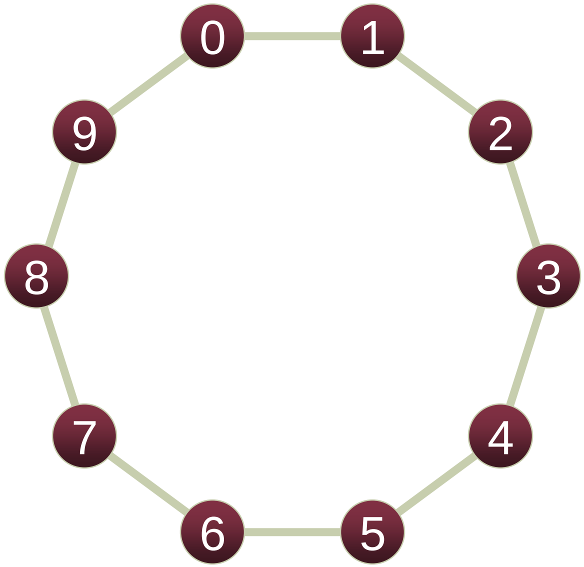
ID	Name
0	a
1	b

vtkGraph and Subclasses



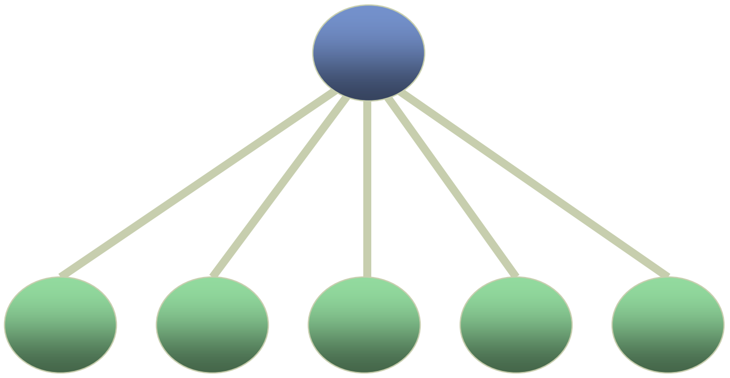
Creating a Graph

```
vtkMutableDirectedGraph* g = vtkMutableDirectedGraph::New();

vtkIntArray* vertId = vtkIntArray::New();
vertId->SetName("id");
g->GetVertexData()->AddArray(vertId);
for (vtkIdType v = 0; v < 10; ++v)
  {
  g->AddVertex();
  vertId->InsertNextValue(v);
  }

for (vtkIdType e = 0; e < 10; ++e)
  {
  g->AddEdge(e, (e+1)%10);
  }
```



Creating a Tree

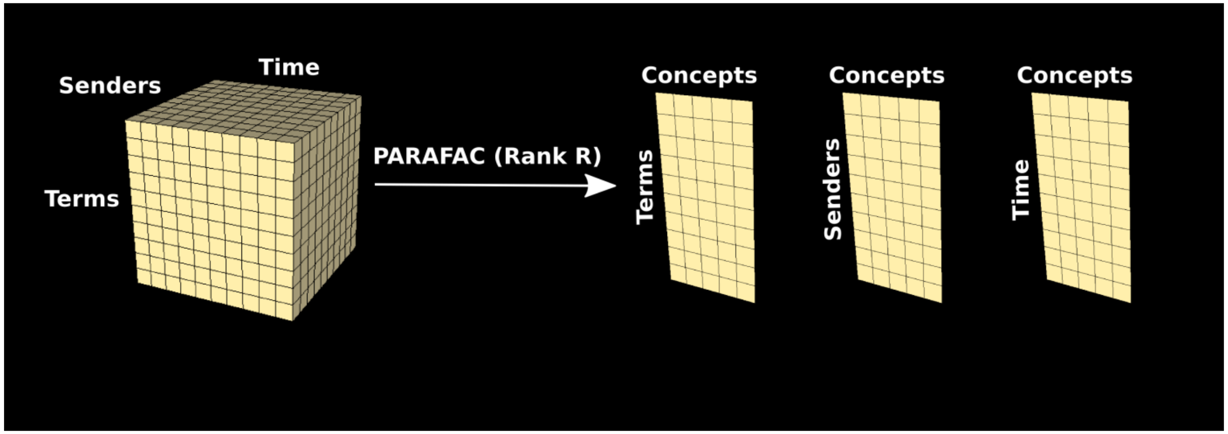
```
vtkMutableDirectedGraph* g = vtkMutableDirectedGraph::New();

vtkIdType root = g->AddVertex();
for (vtkIdType v = 0; v < 5; ++v)
  {
  g->AddChild(root);
  }

vtkTree* t = vtkTree::New();
t->ShallowCopy(g);
g->Delete();
```



vtkArray and Subclasses

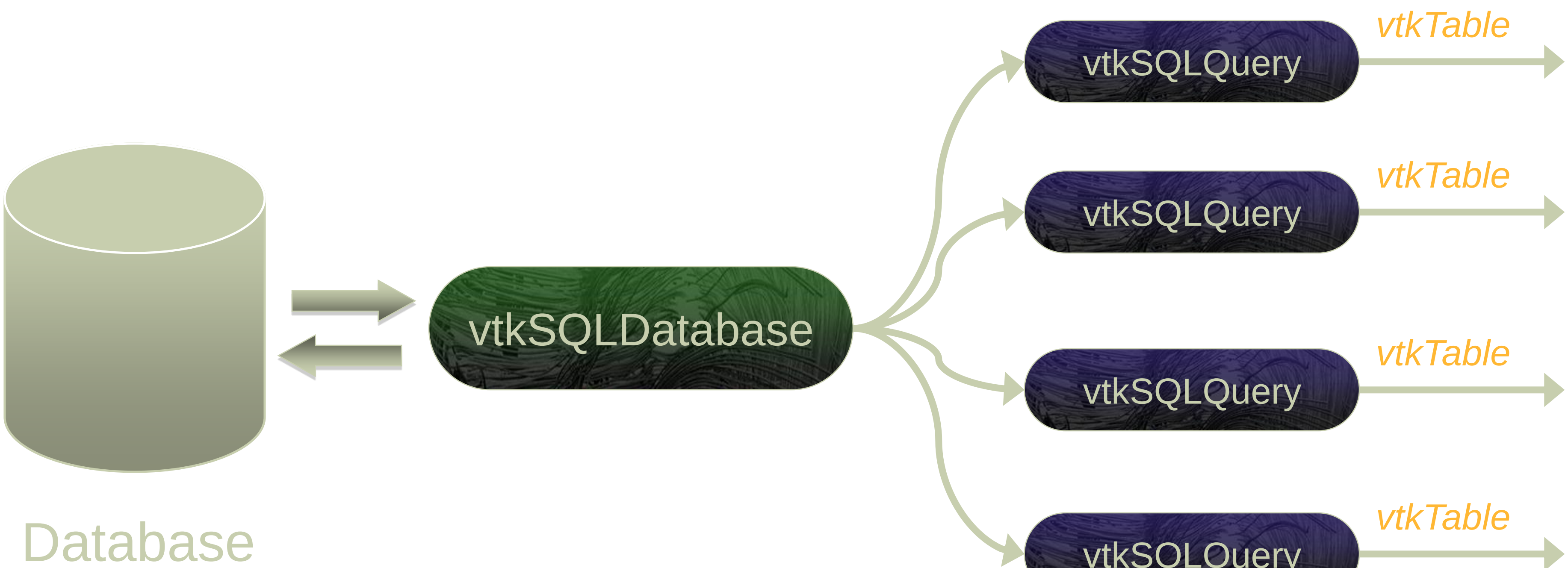
Provides a flexible hierarchy of arbitrary-dimension arrays, including sparse and dense storage, efficient access, and support for custom array types.

Using vtkArray with tensor decomposition methods such as PARAFAC:



PARAFAC = PARAllel FACtor analysis for multi-way arrays

Database Access In VTK



Creating a Database Connection

```
#include <vtkSQLDatabase.h>
vtkSQLDatabase *db = vtkSQLDatabase::CreateFromURL(
    “mysql://username@dbserver.domain.com/databasename”);
bool openStatus = db->Open(“mypassword”);
```

OR

```
#include <vtkMySQLDatabase.h>
vtkMySQLDatabase *db = vtkMySQLDatabase::New();
    db->SetUserName(“username”);
    db->SetHostName(“dbserver.domain.com”);
    db->SetDataBaseName(“databasename”);
    db->Open(“password”);
```

Querying a Database

```
vtkSQLQuery *query = db->GetQueryInstance();
query->SetQuery("SELECT name FROM employees WHERE salary >
    100000");
Bool status = query->Execute();


while (query->NextRow())
{
    cout << query->DataValue(0).ToString() << " is making too much money,
    hire a new PhD."
}


query->Delete();
```

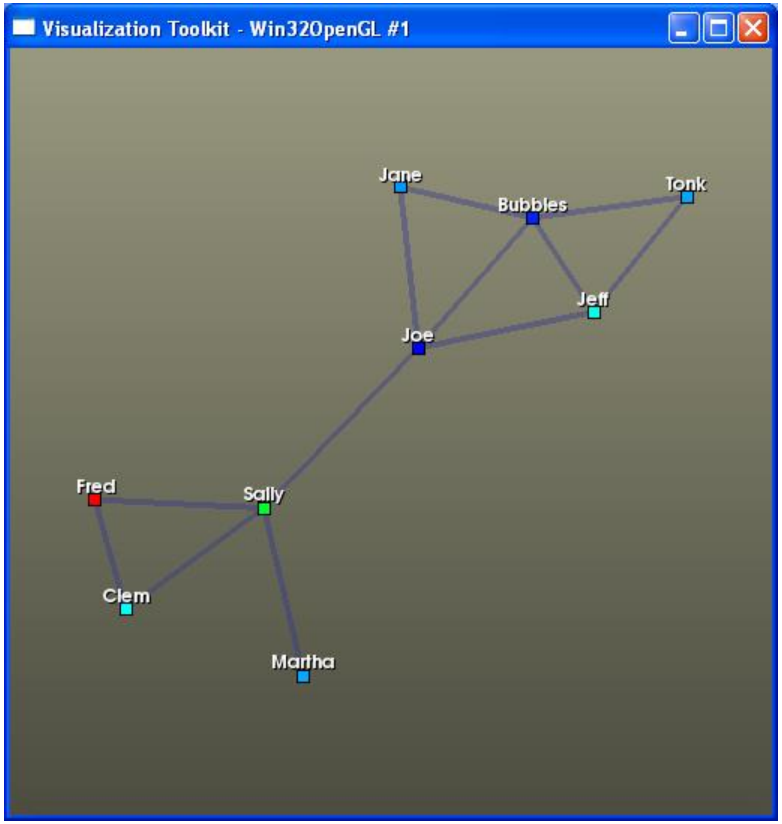
Reading Results the Easy Way

```
vtkRowQueryToTable *tableReader =
    vtkRowQueryToTable::New();
vtkSQLQuery *query = db->GetQueryInstance();

query->SetQuery("SELECT name, salary, age FROM employees");
tableReader->SetQuery(query);

tableReader->Update(); // will execute query and read the results
                       // into a vtkTable

tableReader->Delete();
query->Delete();
```

Available Database Drivers

- SQLite
  - Public domain - ships with VTK
- PostgreSQL
  - Depends on libpq (part of the Postgres distribution)
- MySQL
  - Depends on libmysqlclient (part of the MySQL distribution)
- ODBC
  - Depends on having ODBC libraries installed
    - Unix (incl. Mac): iODBC, unixodbc
    - Windows: MS ODBC
    - Also requires vendor-specific driver for your particular database
- Add your own! It's simple.
  - Subclass vtkSQLDatabase, vtkSQLQuery and implement abstract methods
  - Add optional support to CreateFromURL()

Python Database Example



Using vtkDatabase and vtkRowQueryToTable to hit a database, pull data, and create graphs.

VTK/Examples/Infovis/Python/database.py

Table and Tree Readers

vtkDelimitedTextReader

- Creates a vtkTable from delimited text files, including CSV.

vtkISIReader

- Creates a vtkTable from files in the ISI bibliographic citation format.
- Reference: http://isibasic.com/help/helpprn.html#dialog_export_format

vtkRISReader

- Creates a vtkTable from files in the RIS bibliographic citation format.
- Reference: http://en.wikipedia.org/wiki/RIS_(file_format)

vtkFixedWidthTextReader

- Creates a vtkTable from text files with fixed-width fields.

vtkXMLTreeReader

- Creates a vtkTree using the structure of any XML file.
- XML elements, text, and attributes are mapped to vertex attributes in the output graph.

Graph Readers and Sources

vtkRandomGraphSource

Creates a graph containing a random collection of vertices and edges.

vtkSQLGraphReader

Creates a vtkGraph from a pair of SQL vertex and edge queries.

vtkDIMACSGraphReader

Creates a vtkGraph from files in DIMACS format.

Reference: http://www.dis.uniroma1.it/~challenge9/format.shtml

vtkChacoGraphReader

Creates a vtkGraph from files in Chaco format.

Reference: http://www.sandia.gov/~bahendr/chaco.html

vtkTulipReader

Creates a vtkGraph from files in Tulip format.

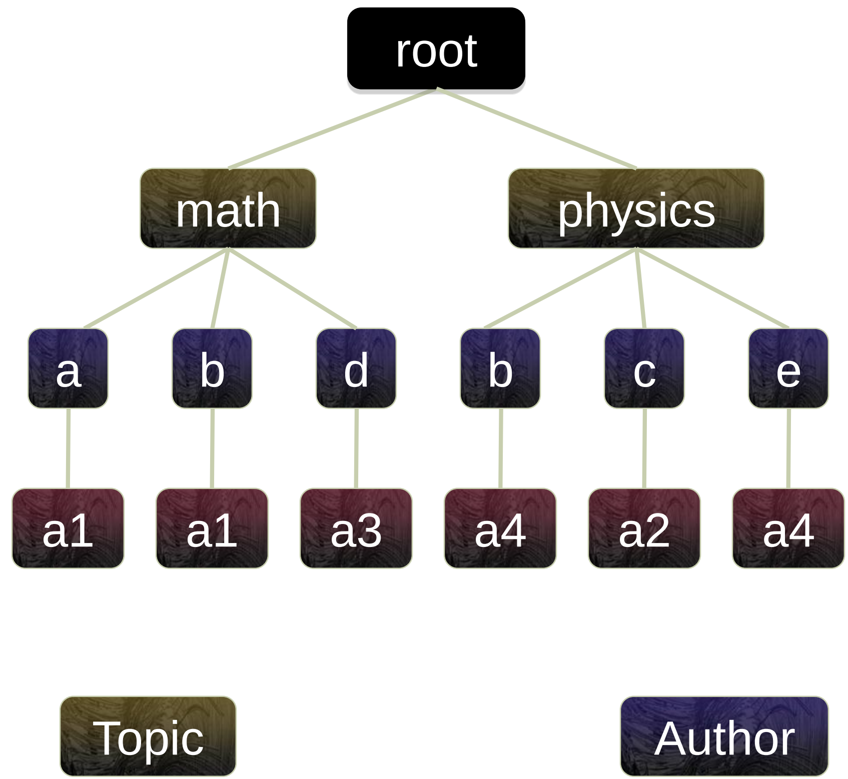
Reference: http://tulip.labri.fr/tlpformat.php

vtkGeoRandomGraphSource

Creates a graph containing a random collection of geo-located vertices and edges.

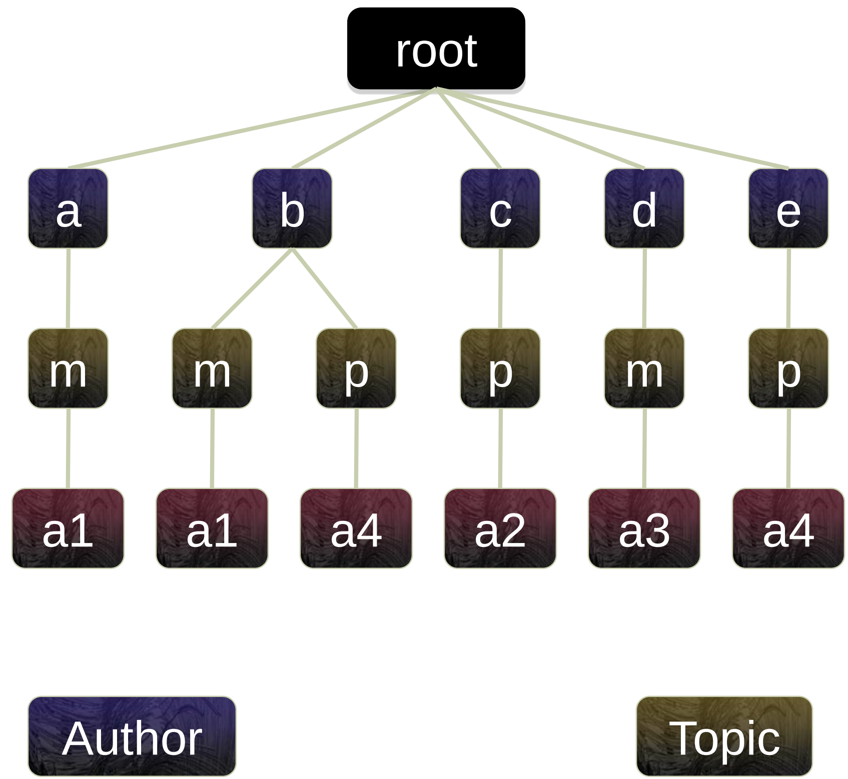
vtkTableToTree – Part I

Author	Article	Topic
a	a1	math
b	a1	math
c	a2	physics
d	a3	math
e	a4	physics
b	a4	physics



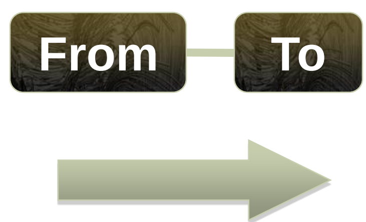
vtkTableToTree – Part II

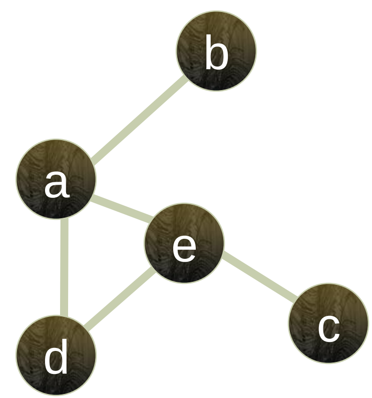
Author	Article	Topic
a	a1	math
b	a1	math
c	a2	physics
d	a3	math
e	a4	physics
b	a4	physics



vtkTableToGraph – Part I

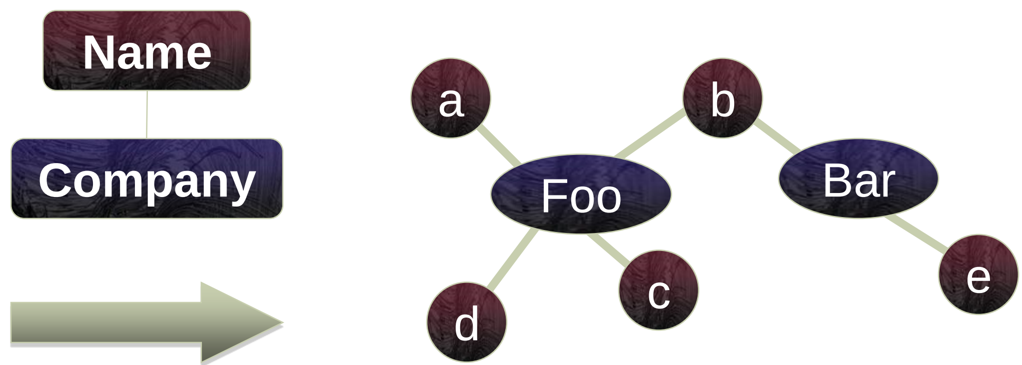
From	To
a	d
a	e
b	a
e	c
e	d





vtkTableToGraph – Part II

Name	Company
a	Foo
b	Bar
c	Foo
d	Bar
e	Bar
b	Foo



vtkTableToGraph – Part III

Author	Article	Topic
a	a1	math
b	a1	math
c	a2	physics
d	a3	math
e	a4	physics
b	a4	physics

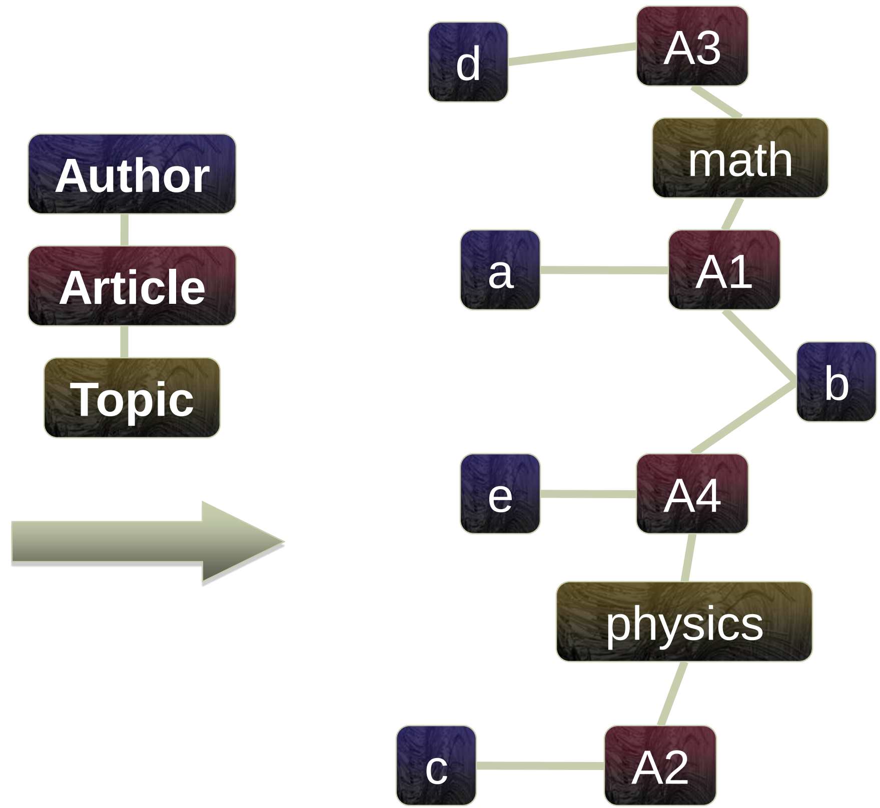
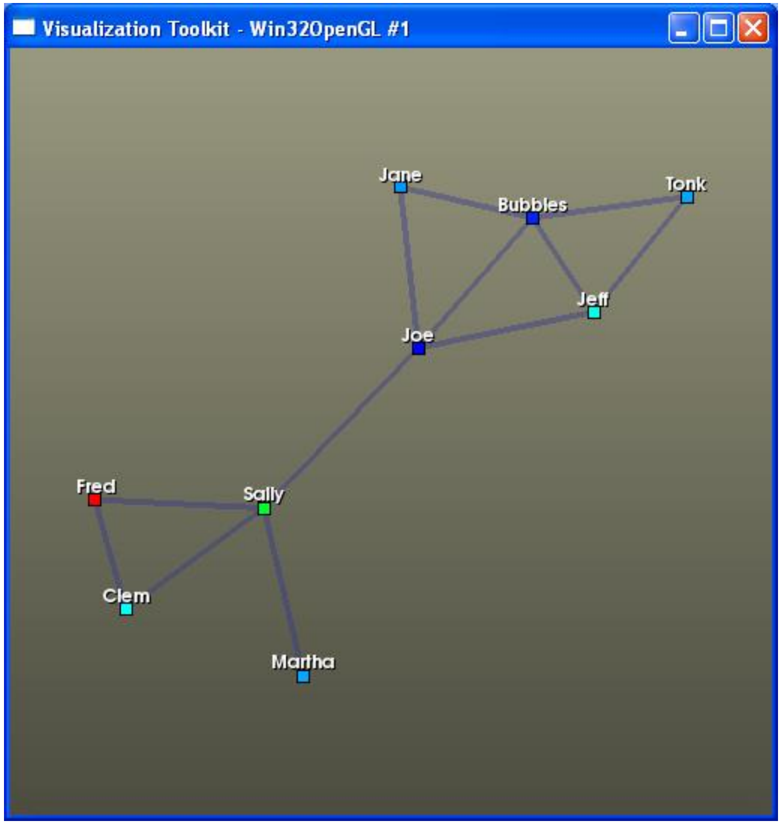


Table to Graph Example



Example that demonstrates the use of vtkTableToGraph filter.

VTK/Examples/Infovis/Python/database.py

Graph/Tree Layout Strategies

vtkDatabase

vtkQuery

vtkTable

vtkTableToGraph

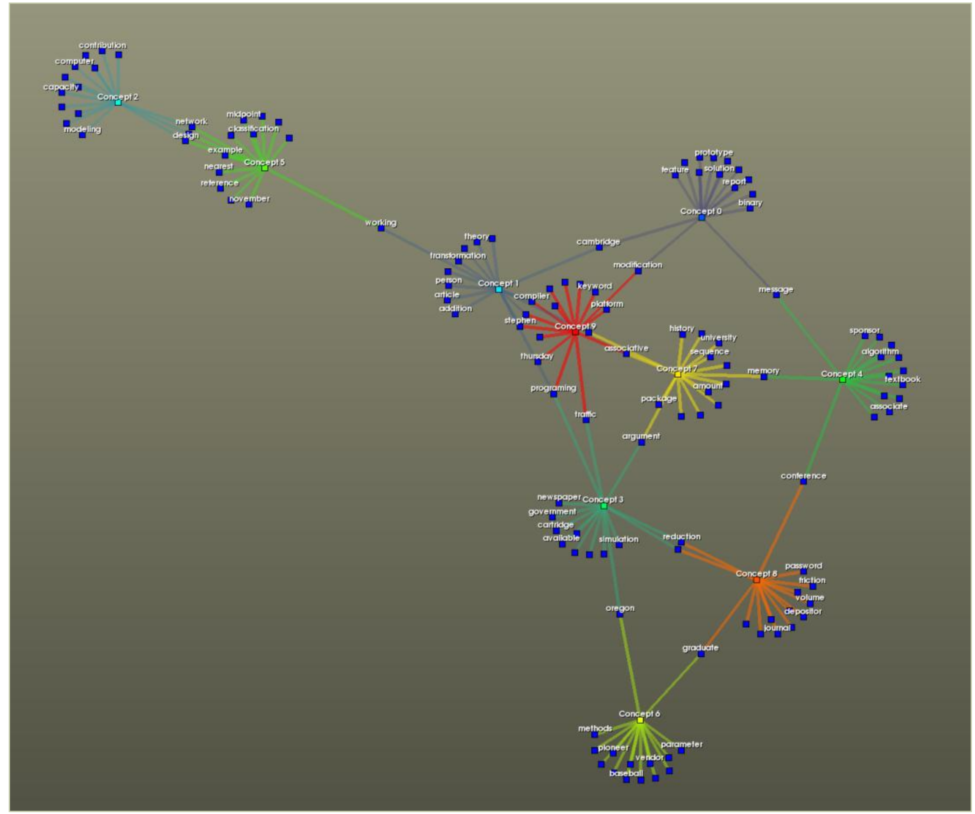
vtkGraph

vtkGraphLayout

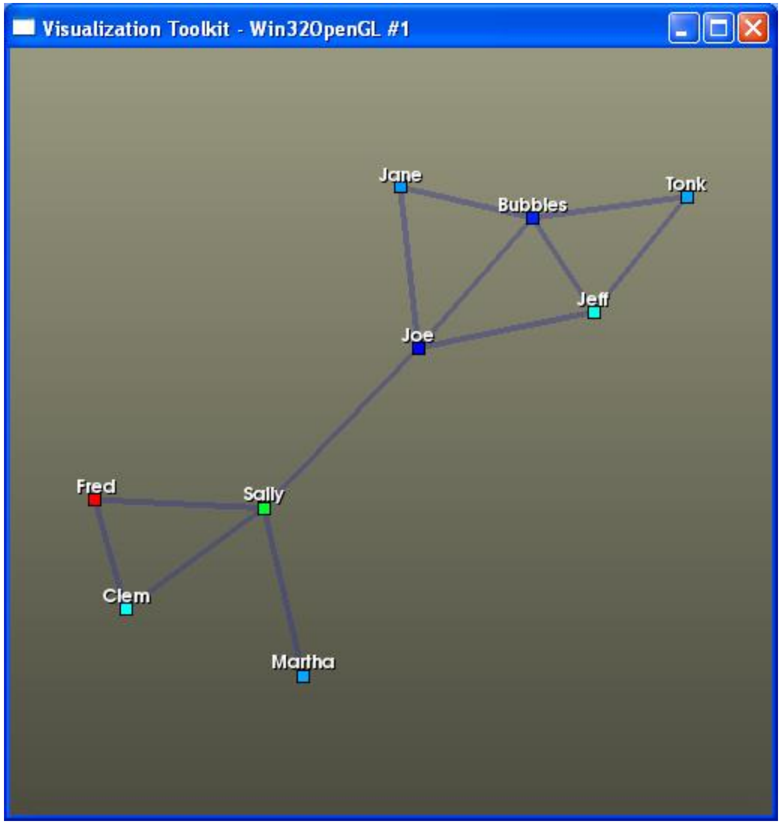
Layout Strategy

vtkGraph + Coordinates

Rest of Pipeline



Graph Layout Strategies Example

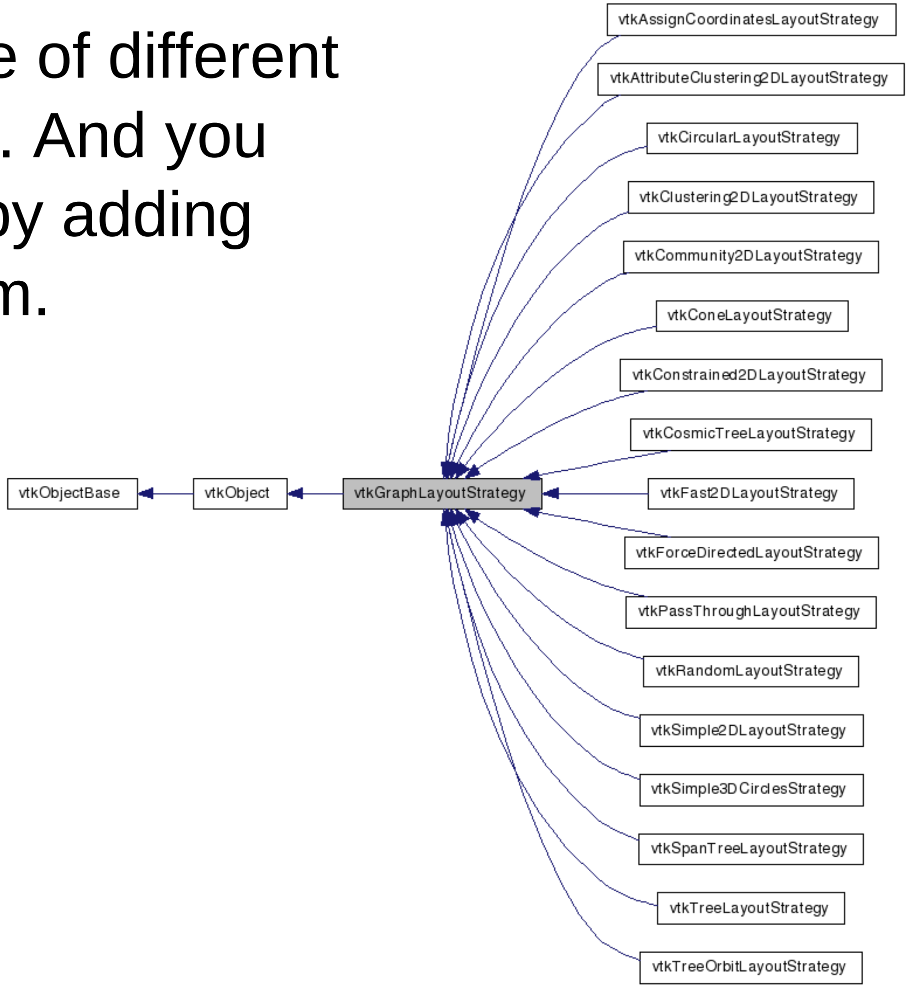


Adding new graph layouts to Titan is a snap!

VTK/Examples/Infovis/Python/database.py

Layout Strategies in VTK

VTK provides a multitude of different layout strategies already. And you can still expand this list by adding your own layout algorithm.



Qt Adapters

vtkQtAbstractModelAdapter

- Inherits from QAbstractItemModel, Qt's generic item model for views
- Provides common infrastructure for converting QModelIndex to VTK ids.

vtkQtTableModelAdapter

- Inherits from vtkQtAbstractModelAdapter
- Adapts underlying vtkTable instance to a Qt model

vtkQtTreeModelAdapter

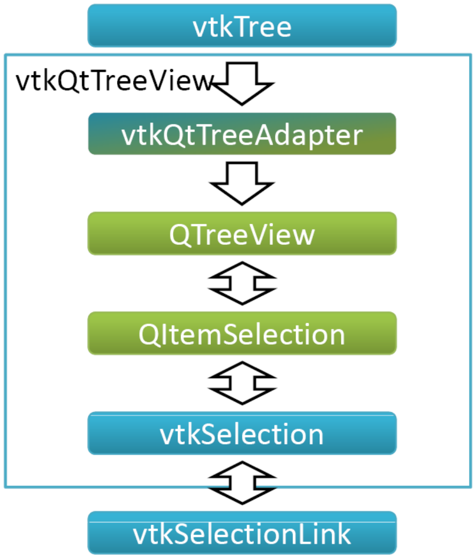
- Inherits from vtkQtAbstractModelAdapter
- Adapts underlying vtkTree instance to a Qt model

QTableView, QTreeView

- Display a QAbstractItemModel

vtkQtTable/TreeView, vtkQtTable/TreeRepresentation

- Puts QTableView, QTreeView into VTK view/representation framework using the model adapter classes
- Supports selection linking with other VTK views.

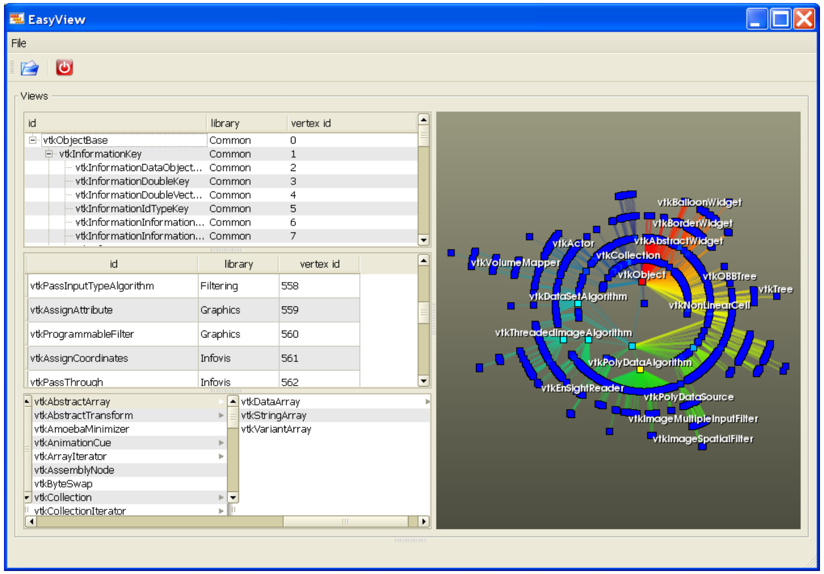


Qt Adapters C++ Example

Qt a reasonable model/view architecture for tables and trees *(specifically shown are QTableView, QTreeView, QColumnView).*

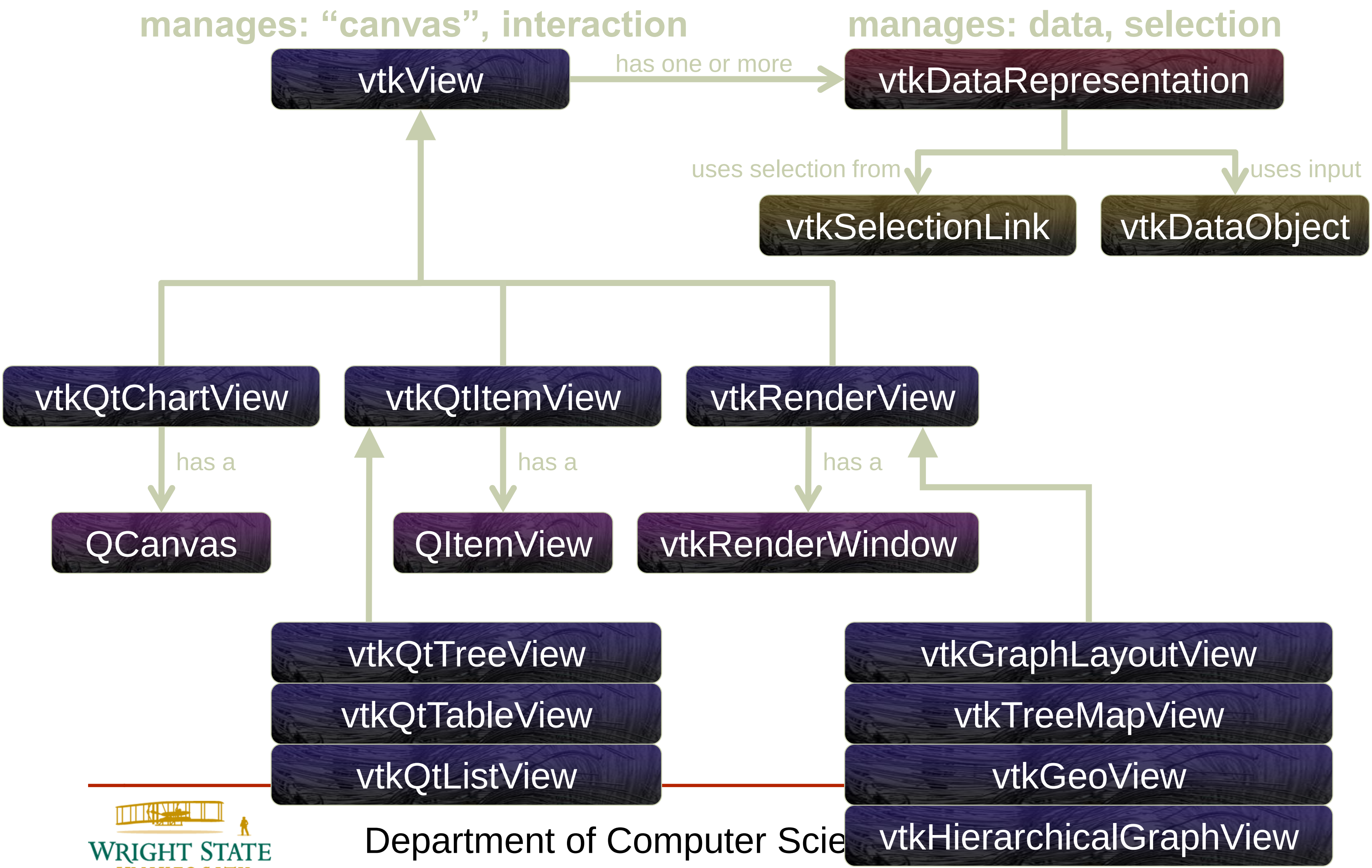
Code “clips” from VTK/Examples/Infovis/Cxx/EasyView

```
…
this->XMLReader    = vtkSmartPointer<vtkXMLTreeReader>::New();
this->TreeView     = vtkSmartPointer<vtkQtTreeView>::New();

// Set widget for the tree view
this->TreeView->SetItemView(this->ui->treeView);
…

// Create xml reader
this->XMLReader->SetFileName( fileName.toAscii() );
…

// Now hand off tree to the tree view
this->TreeView->SetRepresentationFromInputConnection(
    this->XMLReader->GetOutputPort());
…
```



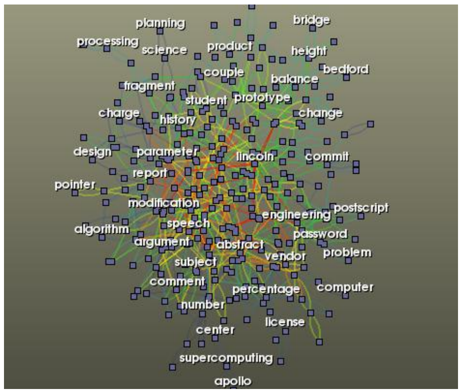
VTK/Examples/Infovis/Cxx/EasyView

Views


manages: "canvas", interaction
manages: data, selection
vtkView
has one or more
vtkDataRepresentation
uses selection from
uses input
vtkSelectionLink
vtkDataObject
vtkQtChartView
vtkQtItemView
vtkRenderView
has a
has a
has a
QCanvas
QItemView
vtkRenderWindow
vtkQtTreeView
vtkQtTableView
vtkQtListView
vtkGraphLayoutView
vtkTreeMapView
vtkGeoView
vtkHierarchicalGraphView

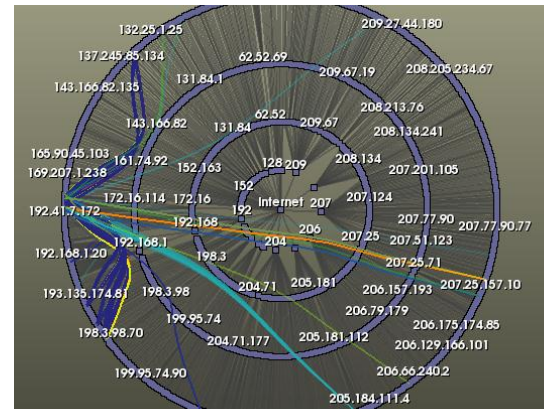
Views Tour

vtkGraphLayoutView

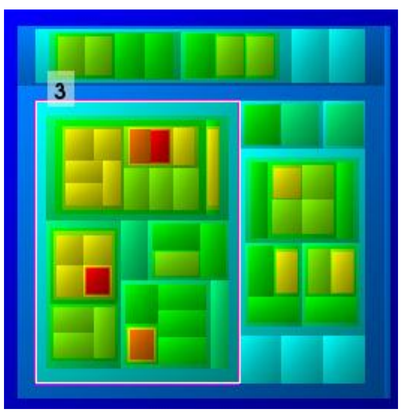


vtkGeoView

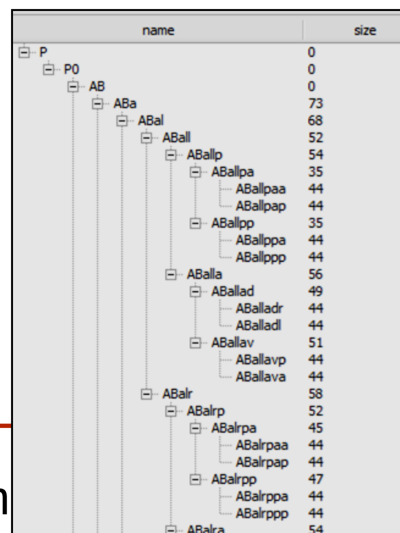
vtkHierarchicalGraphView



vtkTreeMapView

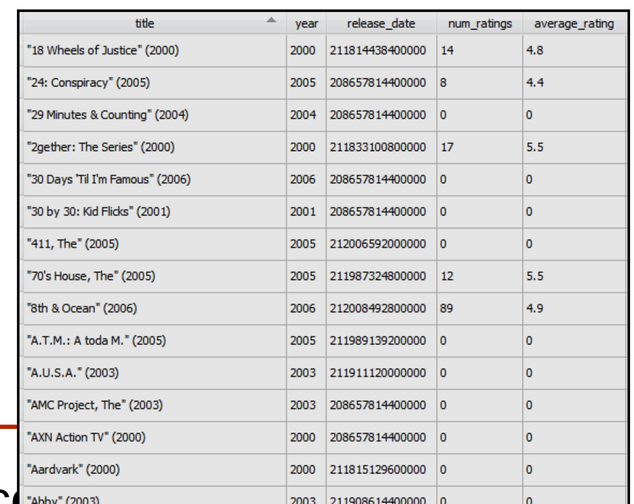


vtkQtTreeView



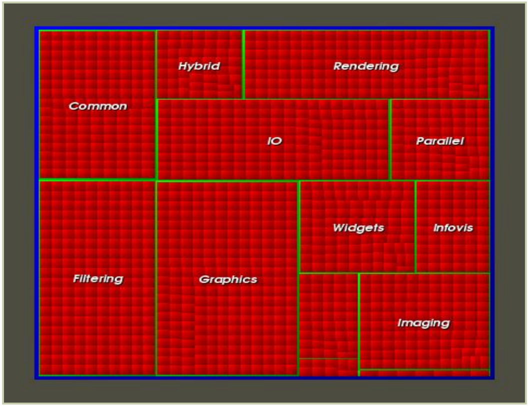
name	size
P	0
P0	0
AB	0
ABa	73
ABal	68
ABall	52
ABallp	54
ABallpa	35
ABallpaa	44
ABallpap	44
ABallpp	35
ABallppa	44
ABallppp	44
ABalla	56
ABallad	49
ABalladr	44
ABalladl	44
ABallav	51
ABallavp	44
ABallava	44
ABalr	58
ABalrp	52
ABalrpa	45
ABalrpaa	44
ABalrpap	44
ABalrpp	47
ABalrppa	44
ABalrppp	44
ABalra	54

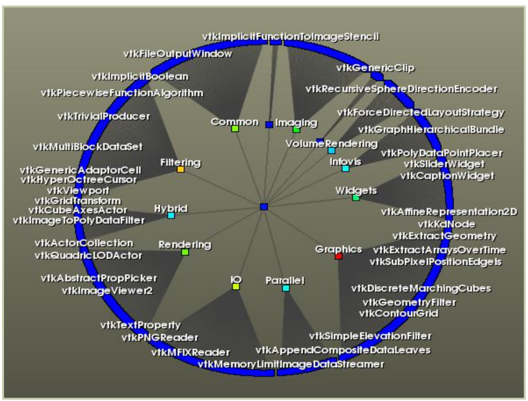
vtkQtTableView

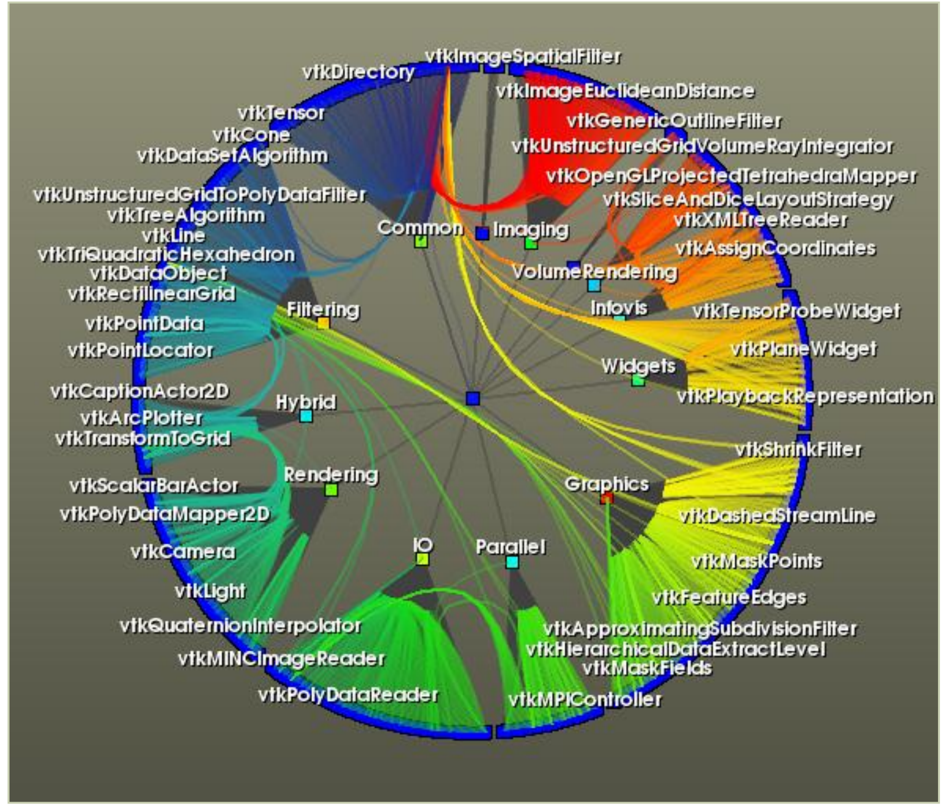


title	year	release_date	num_ratings	average_rating
"18 Wheels of Justice" (2000)	2000	211814438400000	14	4.8
"24: Conspiracy" (2005)	2005	208657814400000	8	4.4
"29 Minutes & Counting" (2004)	2004	208657814400000	0	0
"2gether: The Series" (2000)	2000	211833100800000	17	5.5
"30 Days 'Til I'm Famous" (2006)	2006	208657814400000	0	0
"30 by 30: Kid Flicks" (2001)	2001	208657814400000	0	0
"411, The" (2005)	2005	212006592000000	0	0
"70's House, The" (2005)	2005	211987324800000	12	5.5
"8th & Ocean" (2006)	2006	212008492800000	89	4.9
"A.T.M.: A toda M." (2005)	2005	211989139200000	0	0
"A.U.S.A." (2003)	2003	211911120000000	0	0
"AMC Project, The" (2003)	2003	208657814400000	0	0
"AXN Action TV" (2000)	2000	208657814400000	0	0
"Aardvark" (2000)	2000	211815129600000	0	0
"Abby" (2003)	2003	211908614400000	0	0

Views Python Example



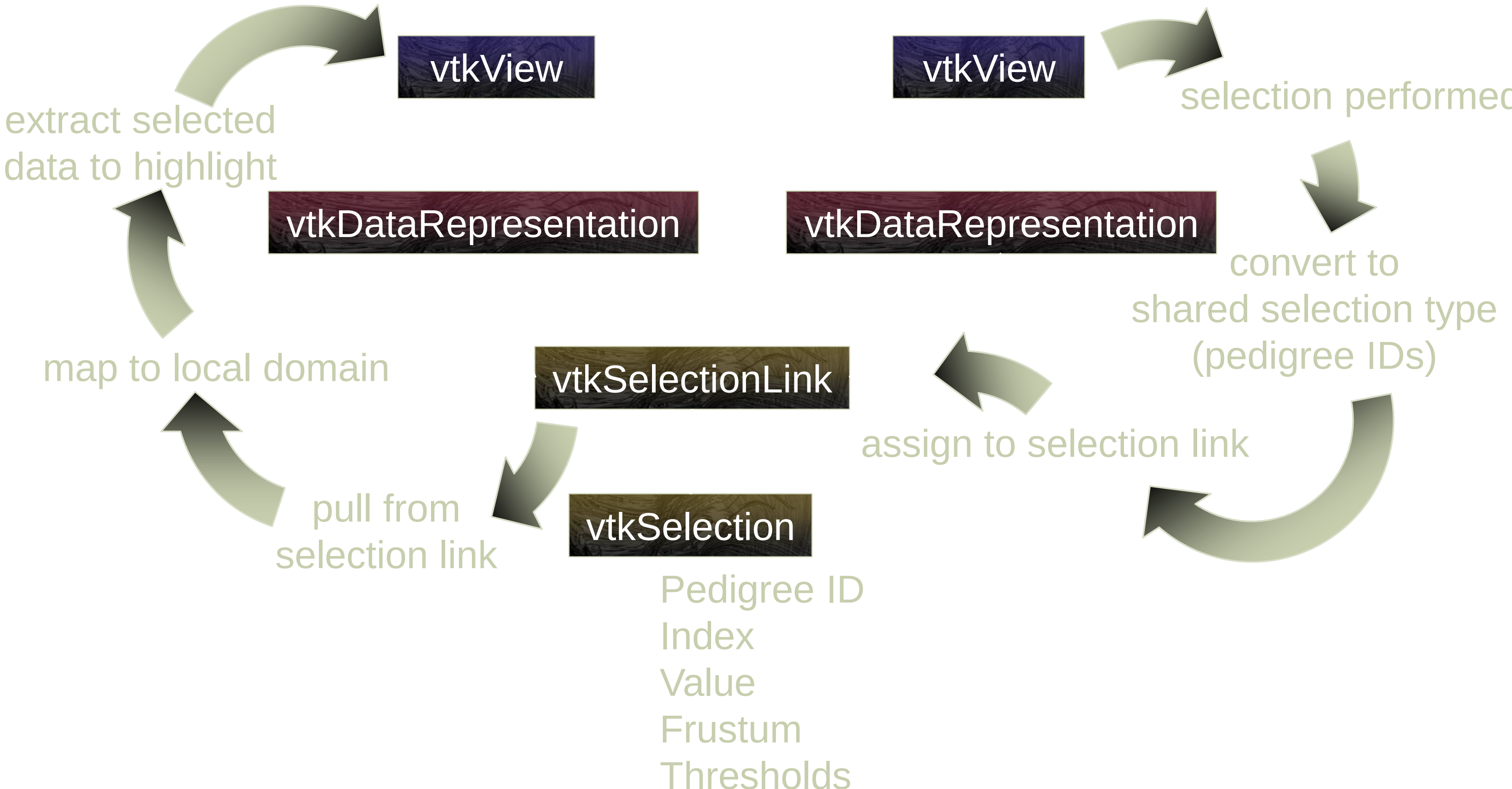




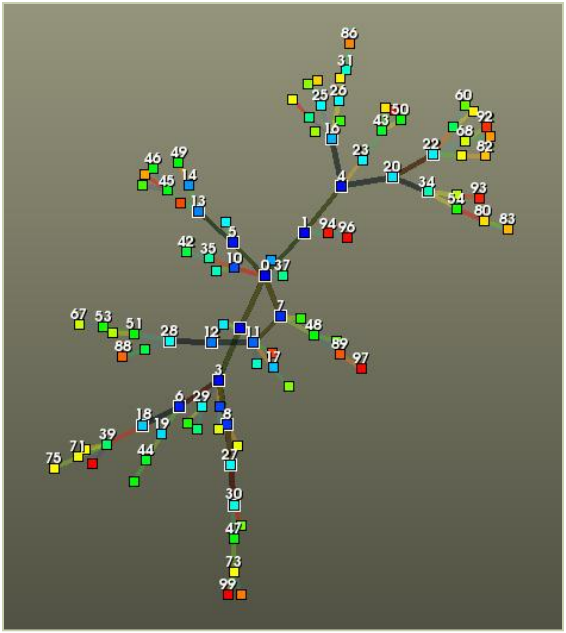
The VTK class hierarchy shown in a vtkTreeMapView, a vtkGraphLayoutView (with tree layout) and a vtkHierarchicalGraphView. The last view also pulls in a graph to show class inheritance relationships.

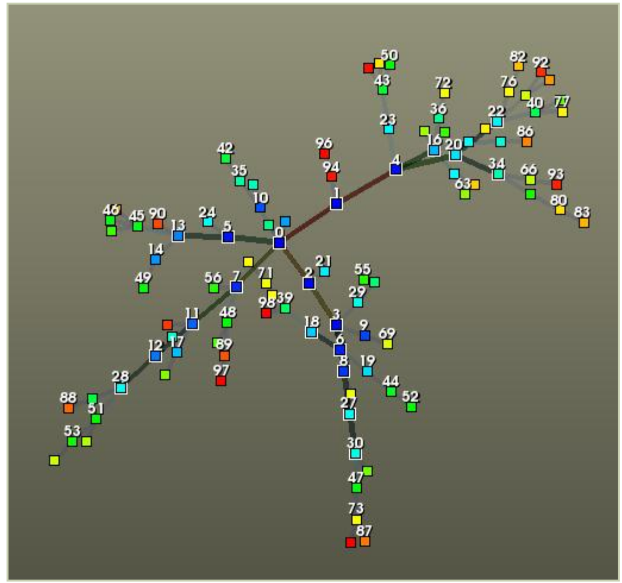
VTK/Examples/Infovis/Python/views.py

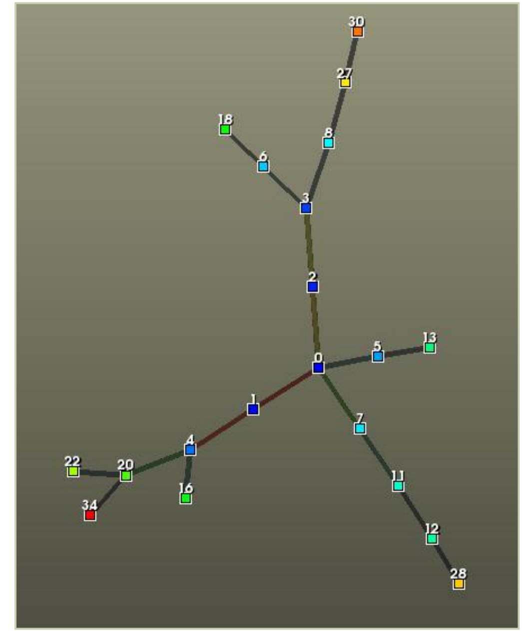
Linked Selection



Selection Python Example





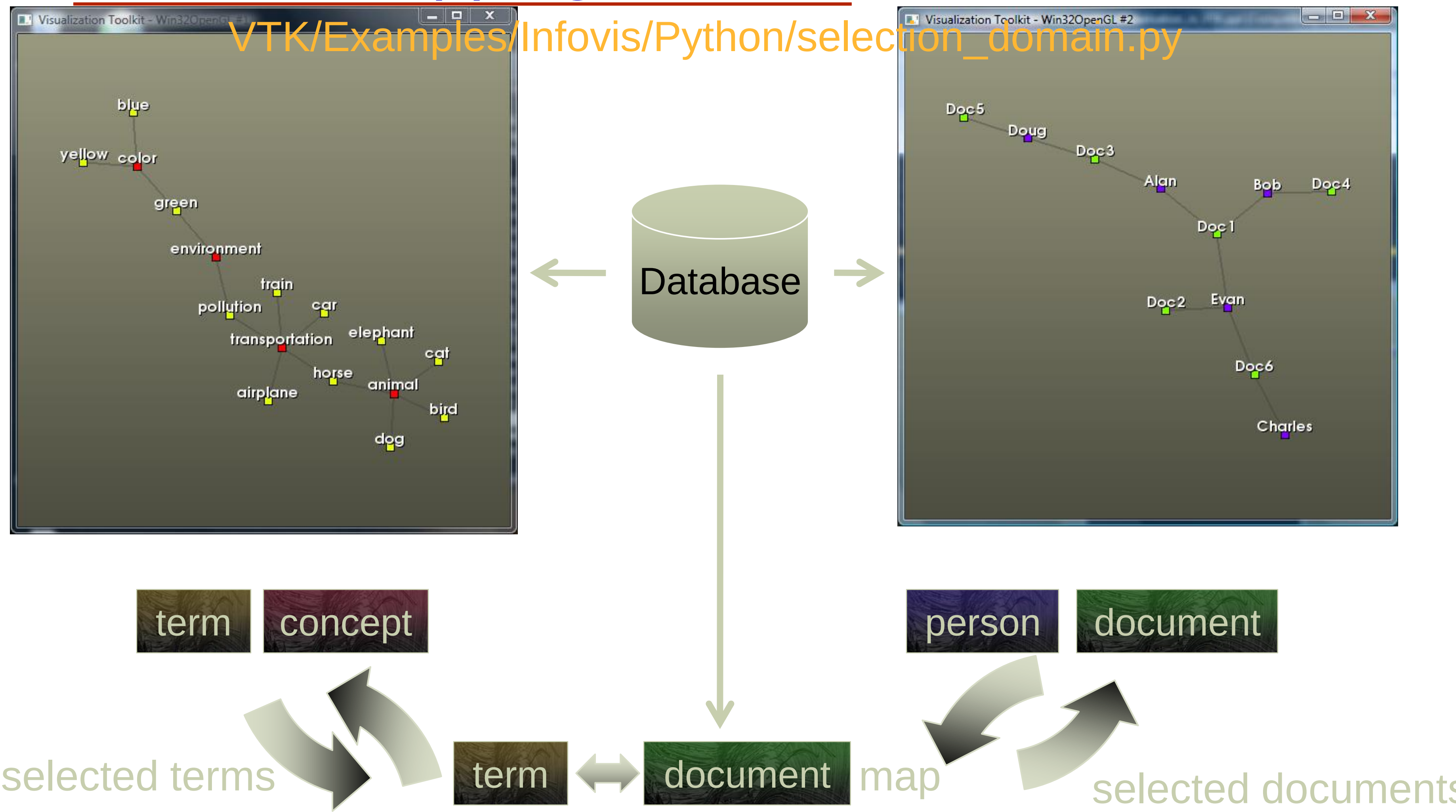


This example demonstrates the use of vtkSelectionLink and vtkSelectionSource. Any vtk view can link it's selection with any other view. vtkSelections are quite flexible and can be used in a variety of ways, here we select edges with high centrality.

VTK/Examples/Infovis/Python/selection.py

Domain Mapping

VTK/Examples/Infovis/Python/selection_domain.py

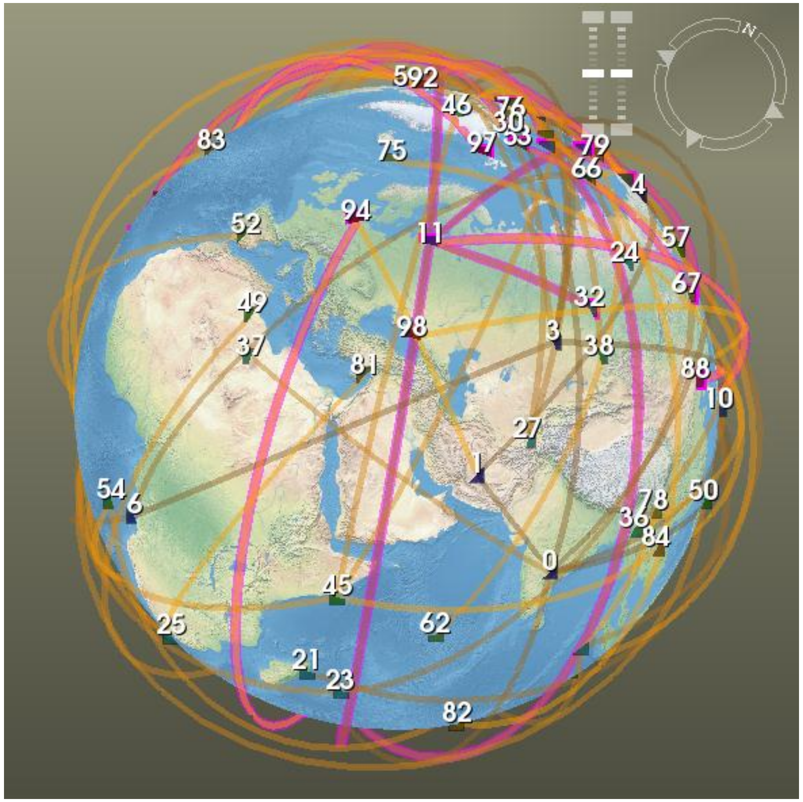


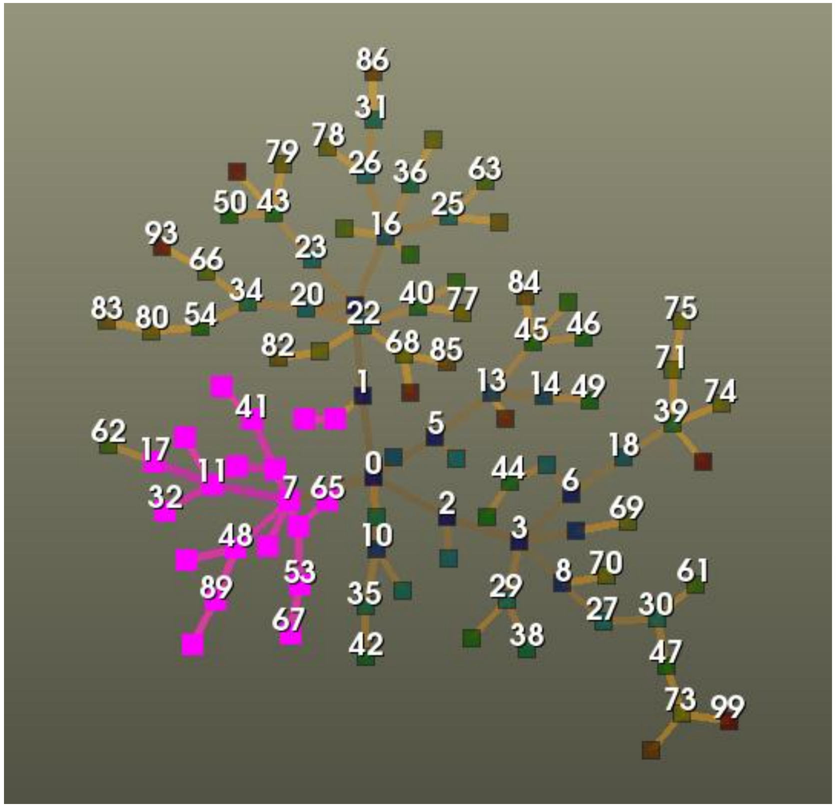
Geographic visualization

- Current features (in VTK now)
  - 3D vtkGeoView
  - Multi-resolution texture and geometry
  - Display placemarks with relationships (i.e. a geolocated graph)
  - Deep integration with other VTK views
    - Takes vtkDataObject input
    - Linked selection with other views
    - Easily embedded into larger applications
- Developing features
  - vtkGeoView2D
  - Multi-texturing overlay images with blending
  - More input sources

3D GeoView Python Example

Uses vtkGeoView and vtkGeoRandomGraphSource, linked with the same graph in a vtkGraphLayoutView.

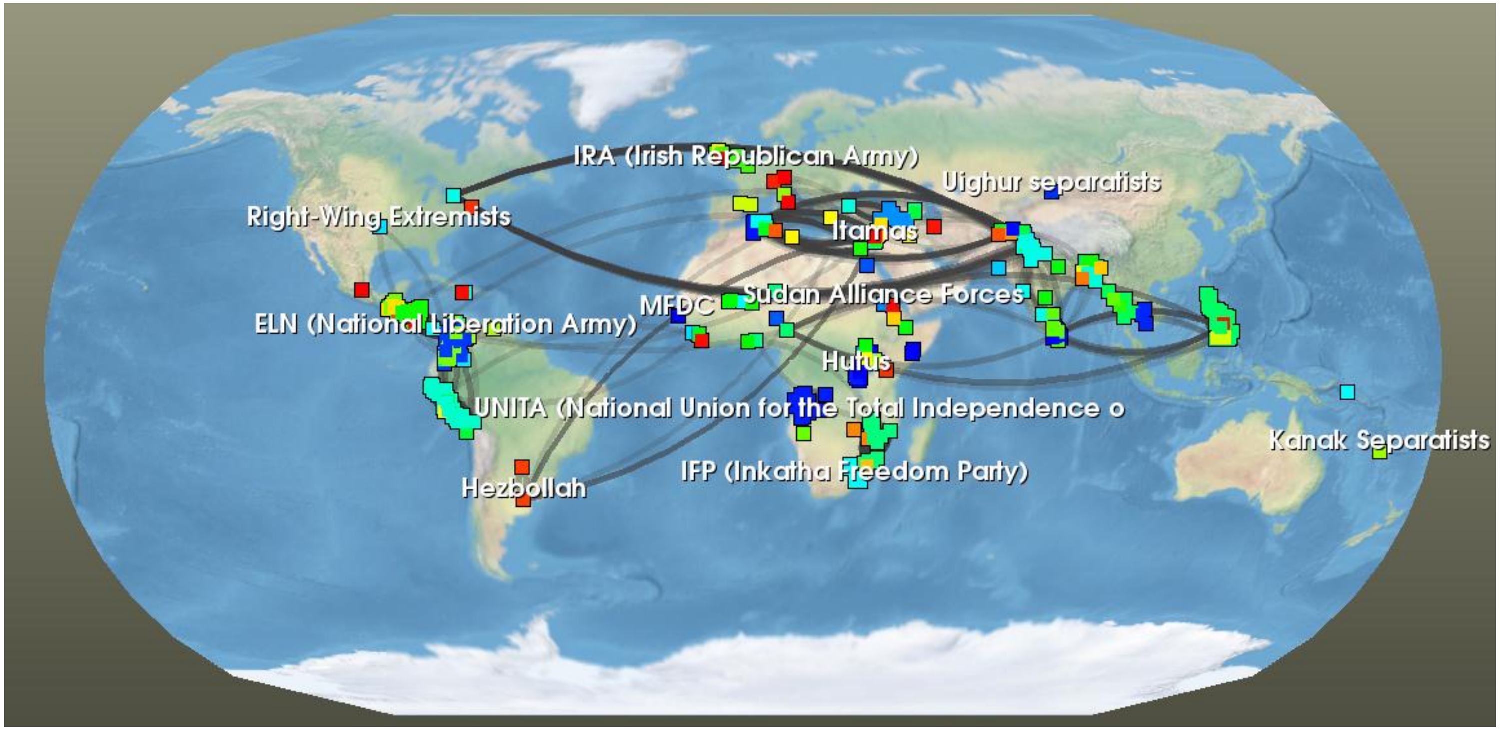




VTK/Examples/Infovis/Python/geovis.py

GeoView Python Examples

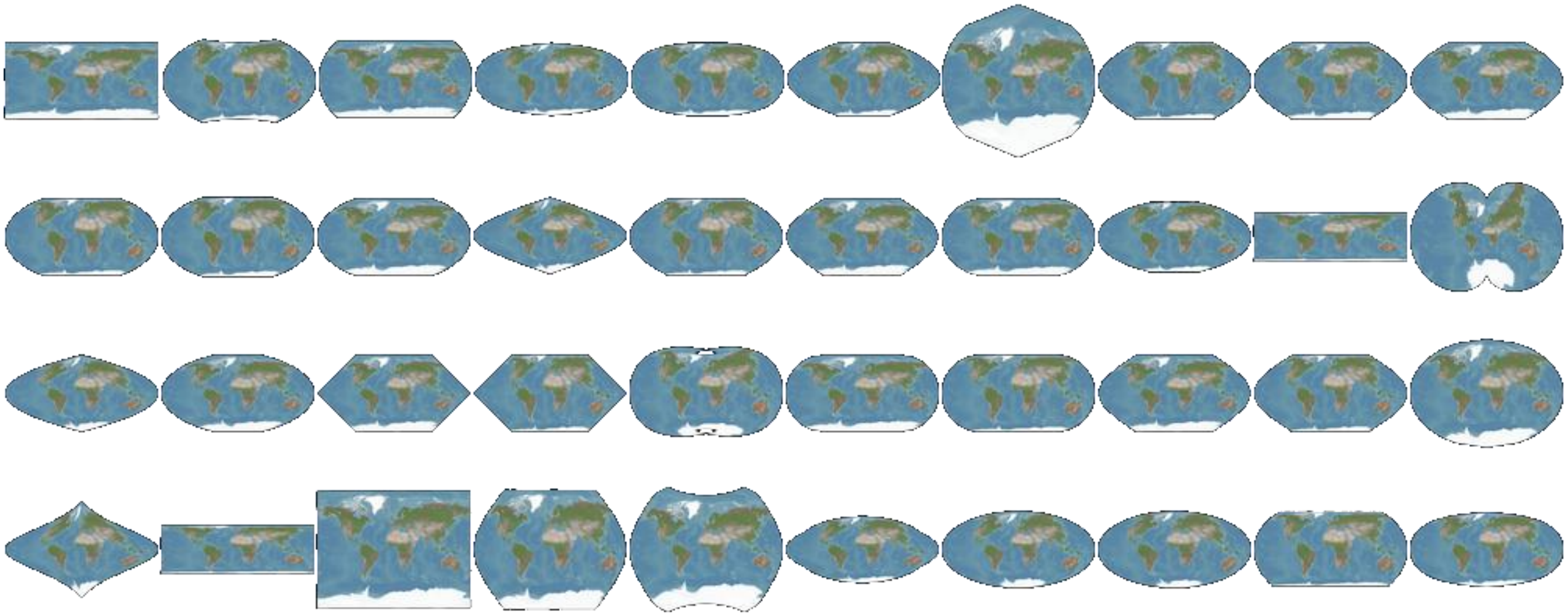
Pulls data from the publicly available GTD (Global Terrorism Database) and uses vtkGeoView2D.



vtkSNL/Examples/Python/Infovis/gtd_geovis_2d.py

Projections

All projections from the open-source Proj.4 projection library are available to vtkGeoView2D.



Break Time!

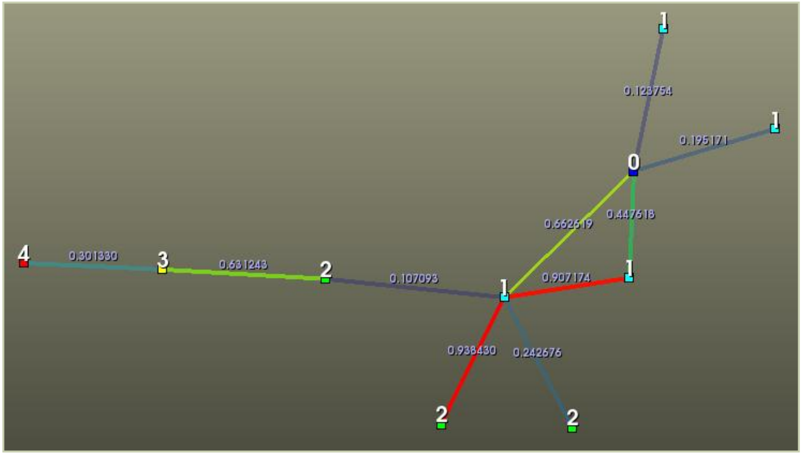
Graph Algorithms, Statistics, and Algebraic Methods are next…

Boost Graph Library (BGL) Adapter

vtkBoostGraphAdapter.h implements the BGL graph concepts for vtkGraph.

vtkBoostBreadthFirstSearch
vtkBoostBreadthFirstSearchTree
vtkBoostBiconnectedComponents
vtkBoostBrandesCentrality
vtkBoostConnectedComponents
vtkBoostKruskalMinimumSpanningTree
vtkBoostPrimMininumSpanningTree

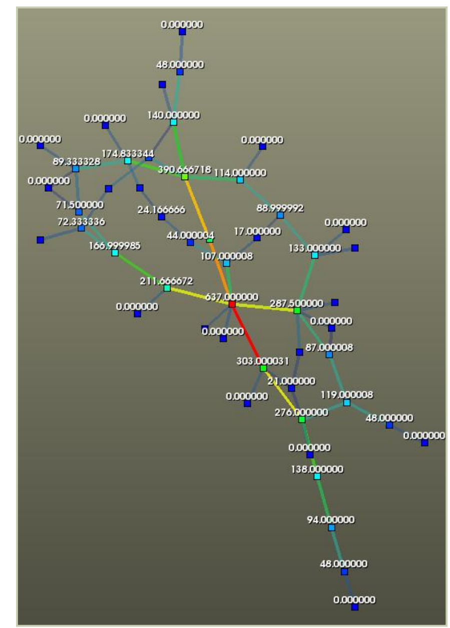
BGL Python Examples



Running vtkBoostBreadthFirstSearch and coloring/labeling the vertices based on the distance from the seed point.

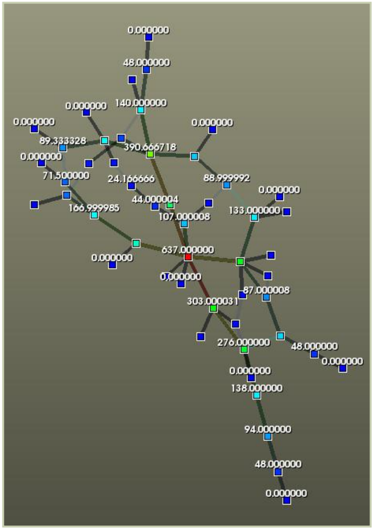
VTK/Examples/Infovis/Python/boost_bfs.py

Running vtkBoostBrandesCentrality and coloring/labeling the edges and vertices based on centrality.



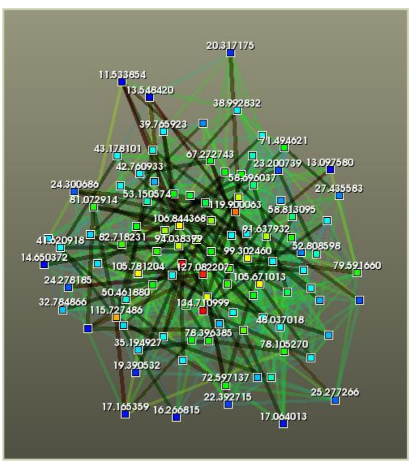
VTK/Examples/Infovis/Python/boost_centrality.py

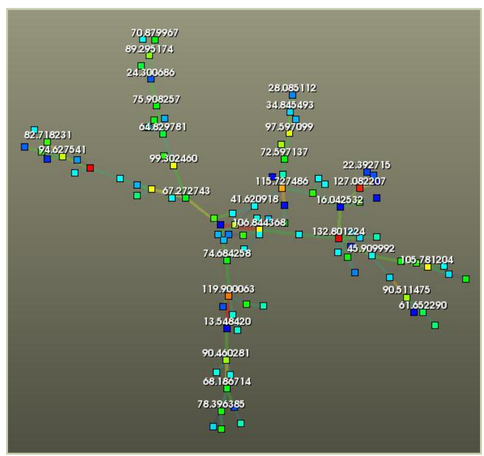
BGL Python Examples



Running vtkBoostBrandesCentrality and then vtkBoostKruskalMinimumSpanningTree to compute a ‘maximal’ spanning tree on high centrality edges.

VTK/Examples/Infovis/Python/boost_mst.py

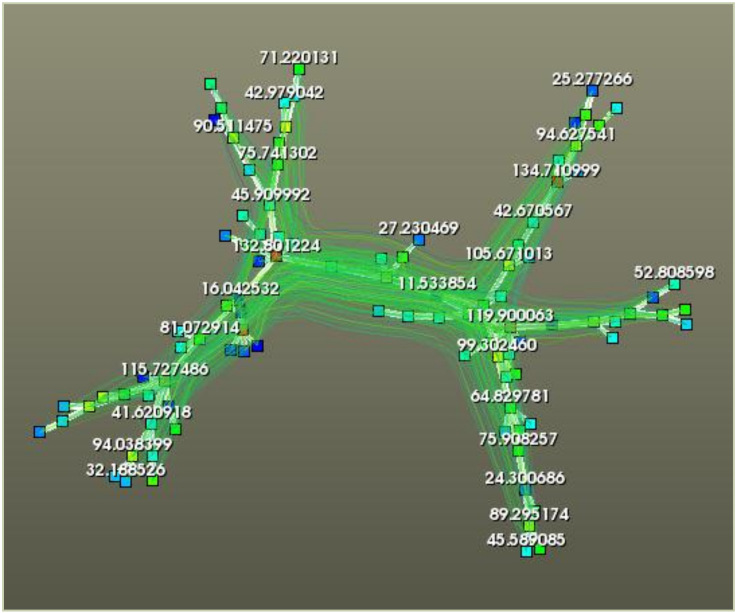


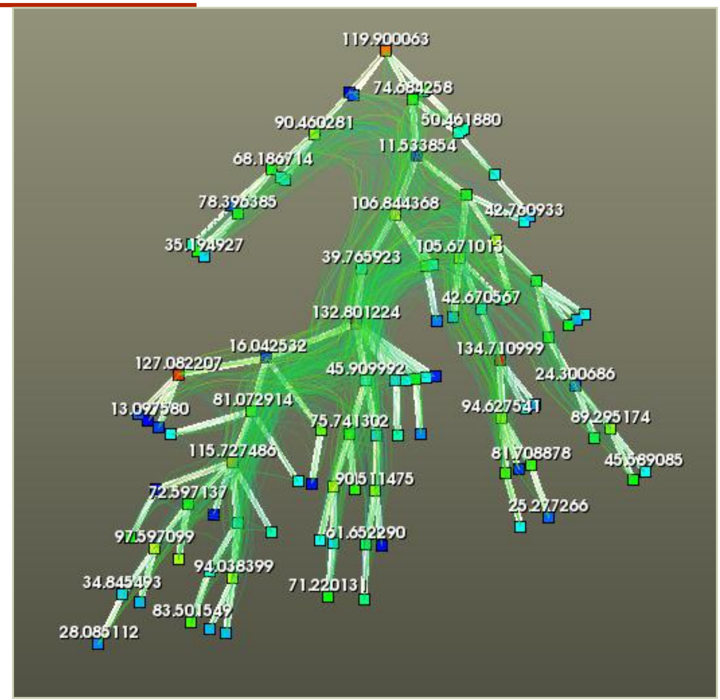


Running the same boost algorithms as above on a more complicated graph and then using vtkExtractSelectedGraph to send the extracted MST to another view.

VTK/Examples/Infovis/Python/boost_mst_extract_graph.py

BGL Python Examples



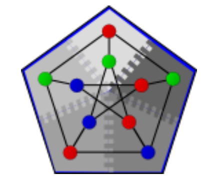


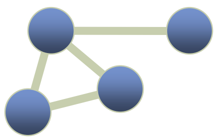
Now showing how the original graph and its computed 'Maximal' spanning tree can both be sent to vtkHierarchicalGraphView. The MST is used to drive the hierarchy and layout, the original graph edges are 'bundled' by using the hierarchy as control points.

VTK/Examples/Infovis/Python/boost_mst_with_hgv.py

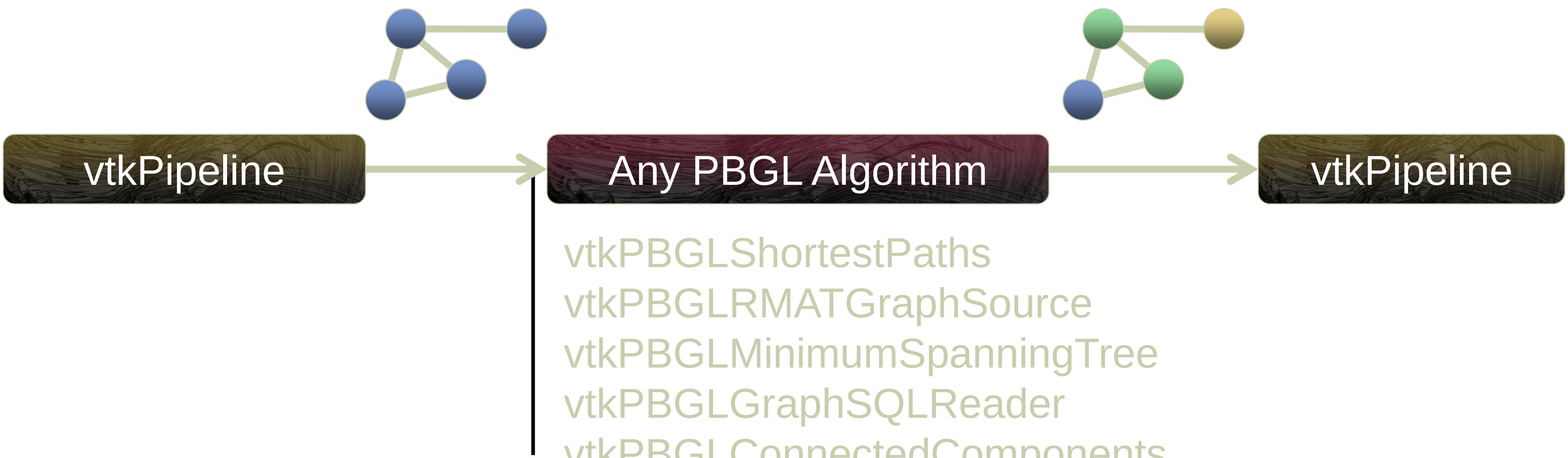
Parallel Boost Graph Library (PBGL) Adapter

vtkPBGLGraphAdapter.h implements the PBGL graph concepts for a vtkGraph (with associated vtkPBGLDistributedGraphHelper).





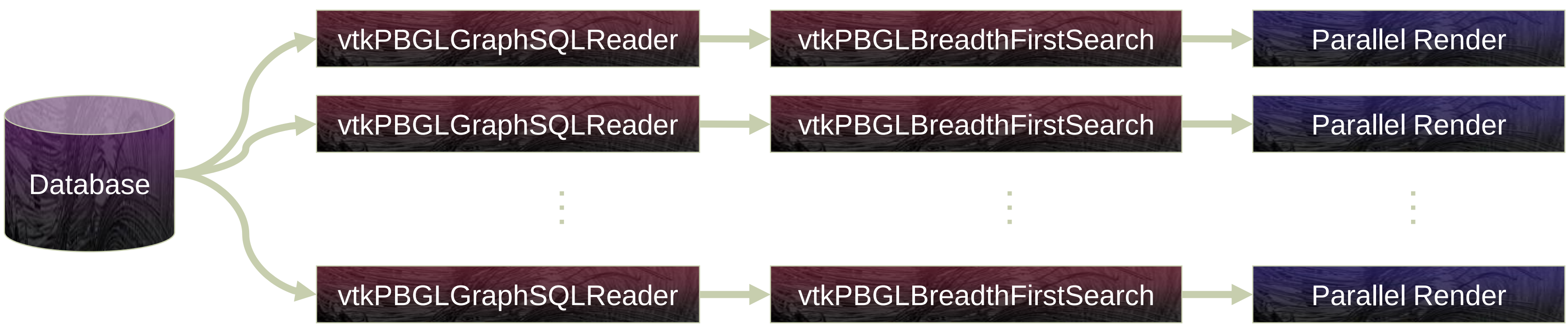
= vtkGraph.SetDistributedHelper(PBGL);



vtkPBGLShortestPaths
vtkPBGLRMATGraphSource
vtkPBGLMinimumSpanningTree
vtkPBGLGraphSQLReader
vtkPBGLConnectedComponents
vtkPBGLVertexColoring
vtkPBGLBreadthFirstSearch
vtkPBGLRandomGraphSource

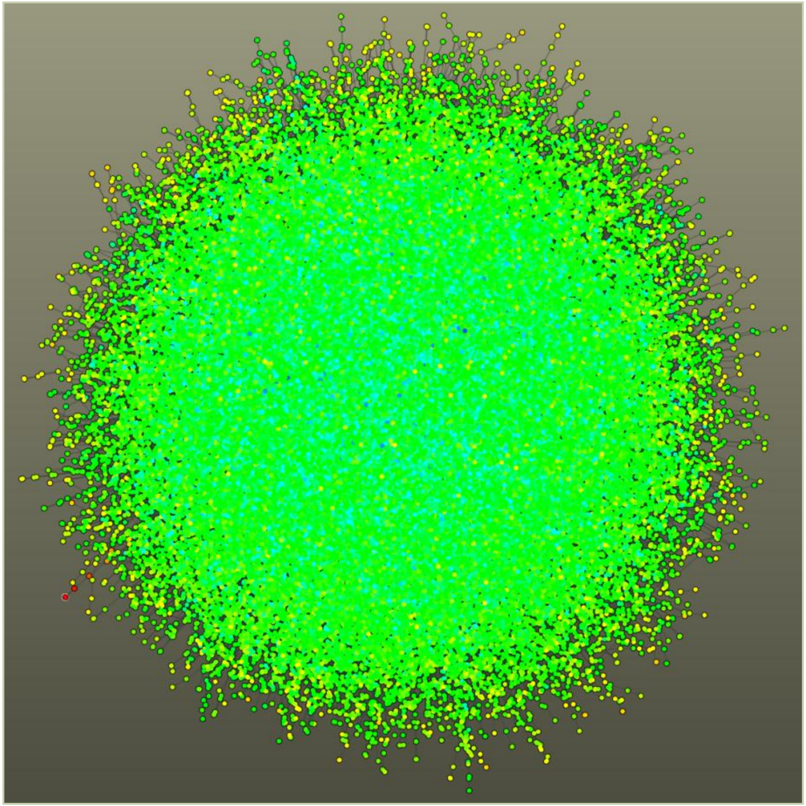
Parallel Graph Analysis – PBGL *(work in progress)*

PBGL: Parallel Boost Graph Library – http://www.osl.iu.edu/research/pbgl
Andrew Lumsdaine, Douglas Gregor (Indiana University)



Currently in the “Hello World” stage:
Running a BFS on a random graph containing 50M vertices and 500M edges on 80 nodes.

PBGL Examples

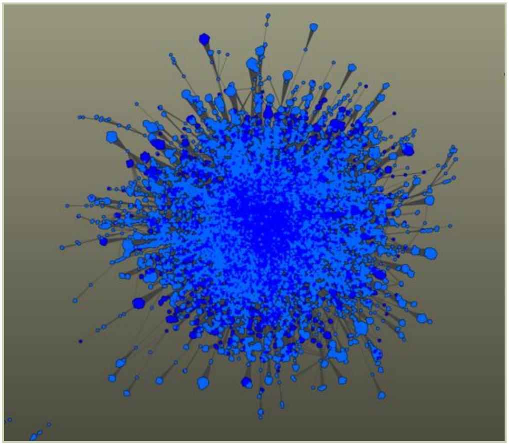


Performing BFS on a random graph with 100K vertices and 100K edges in parallel, collecting the graph and viewing it in graph layout view.

VTK/Examples/Infovis/Cxx/ParallelBFS.cxx

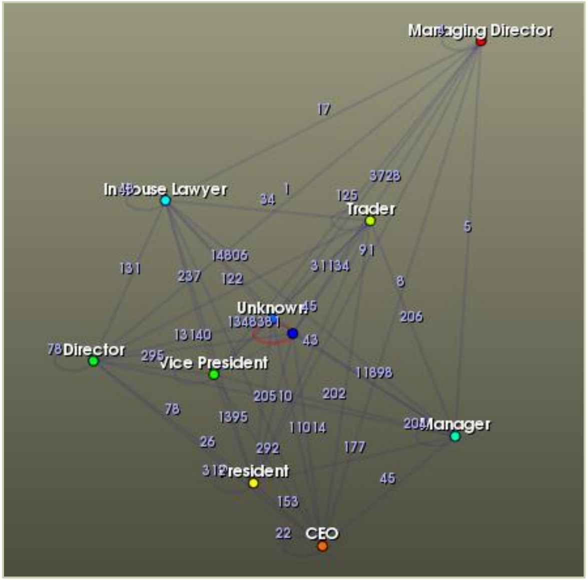
PBGL Examples

VTK/Parallel/Testing/Cxx/TestPBGLGraphSQLReader.cxx



The Enron email corpus graph, containing 75K email accounts and 2M email communications.

Using a parallel pipeline to extract summary information of how people with different job titles interact.



Multi-Threaded Graph Library (MTGL) Adapter

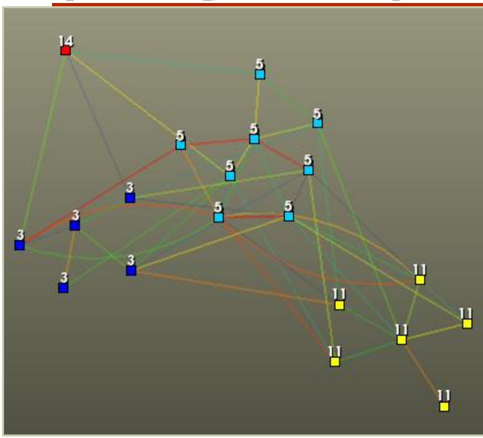
vtkMTGLGraphAdapter.h implements the MTGL graph concepts for vtkGraph.

vtkMTGLCommunityFinder
vtkMTGLHierarchicalCommunityFinder
vtkMTGLSearchEdgeTime
vtkMTGLSearchSSSPDeltastepping
vtkMTGLSelectionFilterCSG
vtkMTGLSelectionFilterST

… list is growing…

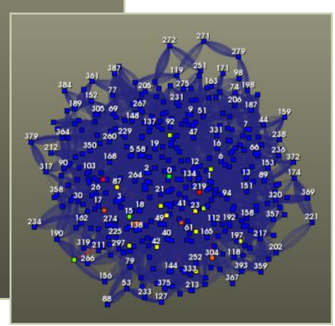
Cray XMT: Massively multithreaded platform, great for graph algorithms. ☺

MTGL Python Examples *(work in progress)*



Running vtkMTGLCommunityFinding and coloring/labeling the vertices based on the community.

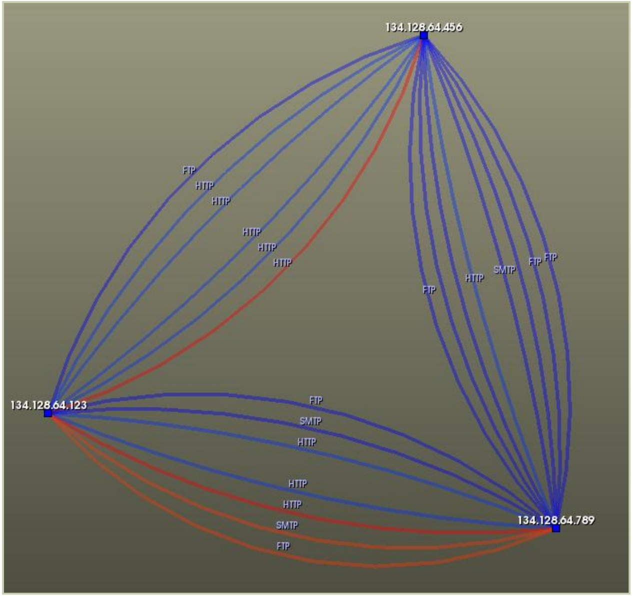
vtkSNL/Examples/Python/Infovis/mtgl_community.py



Running vtkBoostTemporalSearchFwd and coloring/labeling the edges and vertices based on earliest ‘reachability’.

vtkSNL/Examples/Python/Infovis/temporal_search_test.py

Contingency Statistics Example



Running contingency statistics on network transfers illuminates protocols going over non-standard network ports.

VTK/Examples/Infovis/Python/contingency_port_protocol.py

Demonstrates a conditional probability calculation p(port | protocol).