Authoring Visualizations

Jeffrey Heer

CS 294-10: Visualization
Fall 2007

Today

Software Architectures for Visualization
- Graphics and Interaction
- Visualization frameworks
- Characterizing software tools

Goals
- Practical concepts for building visualizations
- Appreciation of design trade-offs in tools
The Basics: Interactive Graphics

2D Graphics Model

- **Drawing Canvas with coordinate system**
  - Origin typically at top-left, increasing down and to the right
  - Units depend on the output medium
    - Screen $\rightarrow$ pixels, printer $\rightarrow$ cm / inches
- **Graphics Context**
  - Device-independent drawing abstraction
  - Potentially holds state for
    - Clipping region
    - Color
    - Typeface
    - Stroke model
    - Coordinate transforms
- **Rendering methods**
  - Draw, fill shapes
  - Draw text strings
  - Draw images
2D Graphics Implementations

- OpenGL (obviously, also includes 3D)
- Postscript / PDF
  - Very influential, inspired the following:
  - Java2D, GDI+ (Win32), Quartz (MacOS X)
    - Platform specific 2D graphics APIs
- Processing
  - Graphics API designed for ease-of-use
  - Based on the metaphor of an artist’s sketchbook
  - Basic interaction: raw mouse and key events
Travel Time Tube Map

User Interface Toolkits

- Low-level graphics APIs lack structure
  - No built-in notion of visual objects
  - No routing of input events to objects
  - No layout support

- User Interface toolkits
  - Spatially organized set of components
  - Event-driven interaction
Containment Hierarchy

Window
  Panel
    Label
    TextArea
  Panel
    Button
    Button

Event Dispatch

Event Queue
- Mouse moved \((t_0, x, y)\)
- Mouse pressed \((t_1, x, y, 1)\)
- Mouse dragged \((t_2, x, y, 1)\)
- Key typed \((t_3, 'F1')\)
- ...

(queues and dispatches incoming events in a dedicated thread)

/* callback for TextArea */
public void mouseMoved(e) {
    // process mouse moved event
}
An alternative structure

- Scenegraph
  - Commonly used in 3D toolkits, also applicable in 2D.
  - Models visual elements, properties and groupings in a semantic directed acyclic graph
  - Groups specified relative to their own coordinate systems
  - Can include object groupings, multiple cameras
  - Well suited for panning and zooming

![Scenegraph Diagram]

Scenegraph-based tools

- Adobe Flash
  - Hierarchy of “DisplayObject” types
  - Transform of parent affects all children
    - Alpha (transparency)
    - Position
    - Rotation
    - Scale

- Piccolo (Java and C#/.NET)
  - Java Toolkit for Zoomable User Interfaces
  - Functionality consolidated in top-level component
  - Extensibility achievable through compile-time inheritance
  - The successor of Pad, Pad++
Visualization Toolkits
How to support diverse visualizations?

Useful higher-level tools?

[In-Class Brainstorm]

- Data
  - Data Transforms (Aggregation)
  - Filters
- Visual Encoding
  - Layout Algorithms
- Animation
- Interaction
  - Brushing and Linking (Coordination)
  - Dynamic Queries
  - Selection
- Diagnostics and Meta-Data
- Automated Evaluation
**Needs of an InfoVis Framework?**

Most UI Toolkits provide unified structures for Graphics and Interaction.

InfoVis frameworks must also consider:
- Data modeling and manipulation
- Mappings from data to visuals
- Higher-level constructs also possible
  - Layout techniques
  - Visual transformations
  - Interaction techniques (dyn. queries...)

---

**Information Visualizer**

Perhaps the first integrated framework for visualization.

Built on early Silicon Graphics machines ($$$), using a LISP graphics language.

Provided a centralized ‘governor’ that oversaw animation, ensuring 100ms or better frame rates, decreasing rendering quality as necessary.

Video: Cone Trees, Perspective Wall
InfoVis Toolkit [Fekete 2004]

Extensible collection of infovis ‘widgets’
- scatterplot, treemaps, graph visualizations, etc
Table-based data model, similar to database
General interactive components
- dyn queries, distortion lenses, excentric labels

http://infovis.sourceforge.net
Structuring InfoVis Applications
Structuring InfoVis Applications

InfoVis apps often require flexibility

Small Multiples and multiple views
  Different visual encodings of the same data

Overview + Detail displays
  Different views of the same visualization

Flexible and varied user input sources
  Interaction techniques, dynamic queries

Improvise by Chris Weaver
InfoVis Reference Model [Card et al]

- **Data Transformations**
  - Mapping raw data into an organization fit for visualization
- **Visual Mappings**
  - Encoding abstract data into a visual representation
- **View Transformations**
  - Changing the view or perspective onto the visual representation
  - User interaction can feed back into any level

**Reference Model Examples**

- **Visual mappings**
  - Layout (assigning x,y position)
  - Size, Shape, Color, Font, etc...
- **View Transformations**
  - Navigation: Panning and Zooming
  - View Distortion (e.g., fisheye lens)
InfoVis Reference Model

- Extension of MVC pattern
- Tiered level of models
  - Data model and visualization model
  - Visualization model can have any number of view/controllers
  - Controllers can feedback to the view or either model

Apply the model: cone trees

Raw Data: File system directories
- Data Transformations: Traverse file system subtree

Data Tables: Parsed/extracted directory tree
- Visual Mappings: Assign 3D coordinates to tree elements (layout), assign colors, fonts. Set lighting.

Visual Structures: 3D model of tree
- View Transformations: Camera placement

View: Rendered, interactive visualization
Interaction: Selection of new focus node: changes visual mappings by forcing new layout calculation
prefuse

Visualization Component Models
Monolithic vs. Polylithic Design

monolithic: primarily use compile-time inheritance to extend functionality
Component or “widget” model for different visualization types (Advizor, InfoVis Toolkit)

polylithic: primarily use run-time composition to extend functionality
Fine-grained operators are composed to create desired behaviors (prefuse, VTK)

Systems are rarely purely one or the other

c.f. Bederson et al, “Toolkit Design for Interactive Structured Graphics”, TSE 30(8),
The Hierarchical Approach

widget hierarchies
extension by subclassing
typing is static
often can't decompose visualizations into compositions of basic techniques

monolithic toolkits
- those that primarily use compile-time inheritance to extend functionality
- [Bederson et al]
A Compositional Approach

Chain together desired components
Extend/replace techniques directly
Directly add new components (or lists of components) to customize visualizations
Enables dynamic changes in composition

polyolithic toolkits
- those that primarily use run-time composition to extend functionality [Bederson et al]
Operator Pattern

Decompose visual data processing into a series of composable operators, enabling flexible and reconfigurable visual mappings.

Operator Pattern

Class Hierarchy of Visualization Operators

Create Compound (Batch) Operators or Operator Conditionals
Comparing approaches

Monolithic (widget approach)

```java
vis = new ScatterPlot(data, xField, yField,
                      shapeField, sizeField);
```

Polyolithic (operator approach)

```java
vis = new Visualization(data);
vis.operators = [
    new AxisLayout(xField, yField),
    new ShapeEncoder(shapeField),
    new SizeEncoder(sizeField)
];
```

Visualization Operators in Prefuse

<table>
<thead>
<tr>
<th>Layout</th>
<th>Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>AxisLayout, GridLayout, …</td>
<td>VisibilityFilter</td>
</tr>
<tr>
<td>ForceDirectedLayout</td>
<td>GraphDistanceFilter</td>
</tr>
<tr>
<td>RadialTreeLayout</td>
<td>FisheyeTreeFilter</td>
</tr>
<tr>
<td>TreeMapLayout</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph/Tree Layout</th>
<th>Distortion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BifocalDistortion</td>
</tr>
<tr>
<td></td>
<td>FisheyeDistortion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Control Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColorAction, DataColorAction</td>
<td>ActionList (sequential grouping)</td>
</tr>
<tr>
<td>SizeAction, DataSizeAction</td>
<td>ActionSwitch (conditional eval.)</td>
</tr>
<tr>
<td>ShapeAction, DataShapeAction</td>
<td>RepaintAction</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Animation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VisibilityAnimator</td>
<td></td>
</tr>
<tr>
<td>LinearAnimator, PolarAnimator</td>
<td></td>
</tr>
<tr>
<td>ColorAnimator, FontAnimator</td>
<td></td>
</tr>
<tr>
<td>SizeAnimator</td>
<td></td>
</tr>
</tbody>
</table>

Actions can be run once or repeatedly over a time interval, controlled by an ActivityManager
**Trade-Offs**

Monolithic model cited as easier for programmers
- Fits existing programming models well
- Less code for common cases
- Works well when not much extensibility is needed

Polyolithic model provides more flexibility and dynamic behavior
- Easier to add, extend, and modify application behavior
- Supports creation of end-user (non-programmer) tools

**Animation**

Operator pattern provides mechanism for fine-grained composition of techniques

However, what about time-based processing?
- Animation (e.g., interpolation, iterated layout)
- Hysteresis (e.g., delayed reaction to input)

Two Approaches
- Frame-based (redraw scene for each frame)
- Time-based (update items over a duration)
Scheduler Pattern

Provide schedulable activities for implementing time-sensitive, potentially recurring operations.

![Activity Diagram](image)

Operator + Scheduler

Implement Operators within the Scheduler pattern: each Operator is also an Activity

Enables animated and timing-sensitive visualization operators.
Animation in prefuse

Animation API: `public void run(double frac)`
Single method allows easy extensibility
All Actions can be scheduled to run repeatedly over a time interval, enabling animation.

`frac` is a value between 0 and 1 indicating the progress through the time interval.

A pacing (or easing) function $p(x)$ can be used to modify the rate of change of the `frac` parameter.

Dynamic Queries
Dynamic Query Bindings

- **Range Queries**
  - Filter a range of values
  - Ordinal and quantitative data
  - Widgets: range slider, slider

- **List Queries**
  - Filter individual data values
  - Any data type, though small lists are better
  - Widgets: combo box, list box, radio buttons, checkboxes

- **Search Queries**
  - Filter items in response to a text search
  - Search engine types: prefix matching, regular expressions, full keyword search (inverted index)
  - Textual data
  - Widgets: text box

---

A Predicate is a boolean function over the tuples of a table. It can be used as input to various operators, including filtering operations.

Dynamic Query widgets are generated by the binding. Interaction changes the model, which notifies the binding, updating the predicate.
Summary

Why Tools Matter

- **Transform the cost structure of development**
- Allow developers to work faster or do more than they could do otherwise
  - Allow reuse of complex techniques
  - Shared structures can facilitate collaboration and communication, even across groups
  - Provides problem-solving framework – can reify and standardize successful approaches
- Influence the design, conventions, and variety of interfaces and visualizations we encounter
  - The oft-debated affects of PowerPoint or Excel charts on business and academia
  - The entrenchment of the WIMP UI paradigm
**Purpose of Toolkits**

The *threshold*: how difficult it is to learn and use the tools

The *ceiling*: how much can be accomplished using a given system

The goal of toolkit design is to *lower the threshold* and *raise the ceiling*

---

**Evaluation**

The basic question: how to gauge the threshold and ceiling of a toolkit?

*Threshold*: usability evaluation

An API is a user interface where programmers are the users.

*Ceiling*: design space analysis

Identify design dimensions, assess the coverage of these dimensions, often by building demonstrative applications.
Summary

Visualization implementation spans multiple levels of abstraction

InfoVis Reference Model
- Decouple data, visual model, and display

Operators, not widgets, provide more flexibility for design and implementation
- Filters, Layout, Encoding, Distortion…

Take note of both the threshold and ceiling of available tools