Spatial Layout

Maneesh Agrawala

CS 294-10: Visualization
Fall 2007

Assignment 3: Visualization Software

Create an interactive visualization application – you choose data domain and visualization technique.

1. Describe data and storyboard interface due Oct 3 (before class)
2. Implement interface and produce final writeup due Oct 15 (before class)
3. Submit the application and a final writeup on the wiki

Can work alone or in pairs
Final write up due before class on Oct 15, 2007
Final project

Design new visualization method
- Pose problem, Implement creative solution

Deliverables
- Implementation of solution
- 8-12 page paper in format of conference paper submission
- 2 design discussion presentations

Schedule
- Project proposal: 10/24
- Initial problem presentation: 10/24, 10/29 or 10/31
- Midpoint design discussion: 11/19, 11/21 or 11/26
- Final paper and presentation: To be determined

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

Spatial Layout
Example: Timeline label layout

Problem

**Input:** Set of graphic elements (scene description)

**Goal:** Select visual attributes for elements

- Position
- Orientation
- Size
- Color
- ...

![Diagram](image-url)
Topics

Direct rule-based methods
Constraint satisfaction
Optimization
Example-based methods

Direct Rule-Based Methods
Rule-based timeline labeling

- Alternate above/below line
- Center labels with respect to point on line

10 labels

Rule-based timeline labeling

- Alternate above/below line
- Center labels with respect to point on line

20 labels
Excentric labeling [Fekete & Plaisant 99]

http://www.cs.umd.edu/hcil/excentric/

Dynamic space management [Bell 00]

Manage free space on desktop to prevent window overlap

Video (0:46s)
Dynamic space management [Bell 00]

Goal: Place new elements to avoid overlap
- Elements are axis-aligned rectangles
- Keep track of largest empty space rectangles
Pros and cons

Pros
- Designed to run extremely quickly
- Simple layout algorithms are easy to code

Cons
- Complex layouts require large rule bases with lots of special cases

Linear Constraint Satisfaction
Network of layout constraints

Constraints as linear equations

Local propagation
- Set any variable
- Update other variables to maintain constraints

One-way
- Each constraint has 1 output variable
- Update output when any input changes

Multi-way
- Each constraint can be written so that any variable is output
- More complicated to maintain

C1: rect2.top = rect1.top + rect1.height + 10
C2: rect2.height = rect1.height
C3: rect2.bottom = rect2.top + rect2.height
One-way constraints form a directed acyclic graph (DAG). Given the value for any variable we propagate its value locally through the graph updating the other variable.

Page layout example [Weitzman and Wittenburg 94]
Adaptive document layout [Jacobs 03]

Users authors templates which use one-way constraints to adapt to changes in page size

ADL template authoring [Jacobs 03]

Video
Pros and cons

Pros
- Often run fast (at least one-way constraints)
- Constraint solving systems are available online
- Can be easier to specify relative layout constraints than to code direct layout algorithm

Cons
- Easy to over-constrain the problem
- Constraint solving systems can only solve some types of layout problems
- Difficult to encode desired layout in terms of mathematical constraints

Optimization
Layout as optimization

Scene description

- **Geometry**: polygons, bounding boxes, lines, points, etc.
- **Layout parameters**: position, orientation, scale, color, etc.

Large design space of possible layouts

To use optimization we will specify ...

- **Initialize/Perturb functions**: Form a layout
- **Penalty function**: Evaluate quality of layout
- .. and find layout that minimizes penalty
Optimization algorithms

There are lots of them:

- line search
- Newton’s method, A*
- tabu
- gradient descent
- conjugate gradient
- linear programming
- quadratic programming
- simulated annealing

Differences

- Speed
- Memory
- Properties of the solution
- Requirements

Simulated annealing

\[ \text{currL} \leftarrow \text{Initialize()} \]  \hspace{1cm} \text{Form initial layout}

\[ \text{while}(! \text{termination condition}) \]

\[ \text{newL} \leftarrow \text{Perturb(currL)} \]  \hspace{1cm} \text{Perturb to form new layout}

\[ \text{currE} \leftarrow \text{Penalty(currL)} \]  \hspace{1cm} \text{Evaluate quality of layouts}

\[ \text{newE} \leftarrow \text{Penalty(newL)} \]

\[ \text{if}((\text{newE} < \text{currE}) \text{ or } (\text{rand}(0,1) < e^{\Delta E/T})) \]

\[ \text{then currL} \leftarrow \text{newL} \]  \hspace{1cm} \text{Always accept lower penalty}

\[ \text{Decrease}(T) \]  \hspace{1cm} \text{Small probability of accepting higher penalty}

**Perturb**: Efficiently cover layout design space

**Penalty**: Describes desirable/undesirable layout features
Scene description

Geometry
- Pie slices
- Anchors for labels
- Labels
  - Bounding boxes

Layout parameters

- Position \((x, y)\)
- Leader line
- Word wrap
- Color
- Alignment
- Orientation
- Scale
Many dimensions $\rightarrow$ large space

- Position $(x, y)$
- Leader line
- Word wrap
- Color
- Alignment
- Orientation
- Scale

2D x 50 labels $\rightarrow$ 100D space

Penalties

Overlap & Distance
- Label – anchor slice
- Label – other slices
- Label – label

Leader lines
- Length
- Intersections

Word Wrap

Annealing minimizes sum of all penalties
Overlap: Label – Anchor Slice

Avoid partial overlap: No penalty if fully inside/outside

Overlap: Label – Anchor Slice

Penalize partial overlap by overlap amount
Distance: Label – Anchor Slice

Ensure label near center of edge of anchor slice

Minimize distance $d$
Penalties

Overlap & Distance
- Label – anchor slice
- Label – other slices
- Label – label

Leader lines
- Length
- Intersections

Word Wrap

Annealing minimizes sum of all penalties

Demo
Pros and cons

Pros
- Much more flexible than linear constraint solving systems

Cons
- Can be relatively slow to converge
- Need to set penalty function parameters (weights)
- Difficult to encode desired layout in terms of mathematical penalty functions

Design principles

Sometimes specified in design books
- Tufte, Few, photography manuals, cartography books …
- Often specified at a high level
- Challenge is to transform principles into constraints or penalties

Cartographer Eduard Imhof’s labeling heurists transformed into penalty functions for an optimization based point labeling system [Edmondson 97]
Example-Based Methods

Preference elicitation [Gajos and Weld 05]

Learn characteristics of good designs

- Generate designs based on a parameterized design space
- Ask designers if they are good or bad
- Learn good parameters values based on responses
Nonlinear Inverse Opt.  [Vollick et al. 07]

Learn label layout style from single example

Horizontal/Vertical

Nonlinear Inverse Opt.  [Vollick et al. 07]

Learn label layout style from single example

Parallel Leader Lines
Artistic Resizing

A Technique for Rich Scale-Sensitive Vector Graphics

Pierre Dragicevic
Stéphane Chatty
David Thevenin
Jean-Luc Vinot

The Resizing Problem

- Fixed size
- Naive scaling
- Artistic resizing
Expressing Artistic Resizing

- Commonly described using formulae

\[
\begin{align*}
  x_L &= (w-w_L)/2 \\
  y_L &= (h-h_L)/2 \\
  w_L &= 20 \\
  h_L &= 10 \\
  w_B &= 5 \\
  h_B &= 5 \\
  r &= 20
\end{align*}
\]

- These formulae are:
  - Translated into code by the programmer
  - Or used as an input to constraint-solving systems

Example-Based Approach

1. Designers produce variants using their authoring tool
2. System interprets the example set
Artistic Resizing
How does it work?

- Assumes the exclusive use of:
  - Copy & paste for adding new examples
  - Affine transformation tools (move, scale, rotate, shear)
- Based on local interpolation of transformations

Artistic Resizing
How does it work?

- Each variant of $T_1$ is associated with the example’s bounding box
Artistic Resizing
How does it work?

- Problem of multivariate interpolation

Pros and cons

**Pros**
- Often much easier to specify desired layout via example

**Cons**
- Usually requires underlying model
- Model will constrain types of layouts possible
- Large design spaces likely to require lots of examples to learn parameters well