

Data Selection



Perception and Interaction

Basic Pointing Methods

Point Selection

Mouse Hover / Click

Touch / Tap

Select Nearby Element (e.g., Bubble Cursor)

Region Selection

Rubber-band or Lasso

Area Cursors (“Brushes”)

Basic Pointing Methods

Everyone here is familiar with very basic pointing methods... mapping from user interface devices like a mouse or touchpad, on to points or regions on-screen. You might then click on an object or point, and so select an object for highlighting and further operations.

Slightly more subtle than that is the bubble cursor, that tries to allow for inaccuracy in point selection. You can get ‘near enough’ to an object and then click... and then that object is selected. You get visual feedback on what’s happening, by (surprisingly enough) a circle or bubble around the cursor that extends out shows that is near enough for this action. For more details and examples, see here:
<https://www.tovigrossman.com/BubbleCursor/>

Regions can be selected in similar direct ways. You can draw round an object, with rubber-band lines or lassos... or brush (usually rectangular)

areas to select the subset of objects in that area. Usually, the program will simply store the set of the selected objects, and that set will then get highlighted in that view... and in any other linked views (this is *brushing and linking*, as mentioned in the previous lecture).

These are all well-established methods.

Brushing and Linking

Select (“brush”) a subset of data
See selected data in other views

The components must be *linked*
by *tuple* (matching data points), or
by *query* (matching range or values)

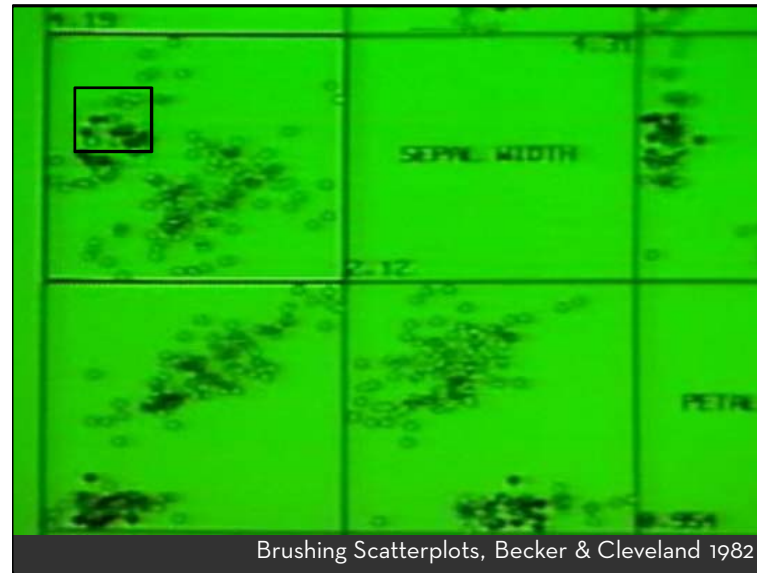
Brushing and Linking

This takes us on to a basic operation, used to link different visualisations—different views or GUI components. This was introduced to you in the previous lecture, on the visualization pipeline.

Originally, in statistical graphics in the 1970s and 1980s, this technique was called ‘brushing’... the metaphor being that you would paint some objects in one view with a colour, and the set of objects *as displayed in other linked views* would be brushed with the same colour... so as to highlight the same objects in all views.

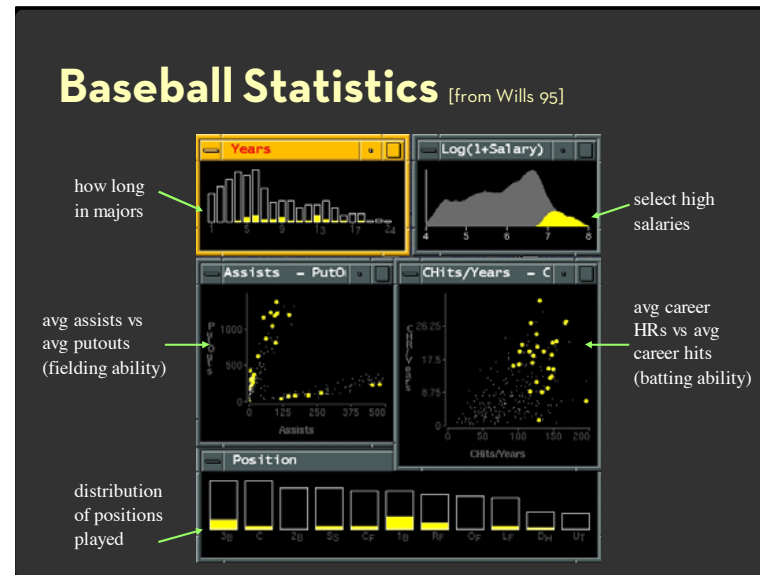
This is still the normal approach... *by tuple*. The brushed data points or data objects are treated individually. In each view, the system finds the selected set of individual data objects, and highlights each set member in the same way.

However, we can also link *by query*, and we will explain that more soon here. This is not yet common... but maybe it should be, as it is so powerful. We can use a query that is, effectively, run in each view. Here we are not linking the views via a *set* of objects, but via a *specification* of objects.



[image shows a very old screenshot of one of the first examples of brushing, from Becker and Cleveland, 1982]

It is slightly difficult to see, but here is some of the original 'brushing and linking' work from 1982. You may see that they originally tried it out on the iris data that we've seen already in these lectures.



[image shows a baseball statistics visualization involving five linked views, from Wills 1995: a bar chart top left shows how long players have been in the major leagues, a 2D line graph top right shows salaries, two scatterplots in the middle show data on fielding ability and batting ability, and the bottom histogram shows a distribution of the positions played.]

Here's a clearer example, showing very early work on linked views. There has been a selection of a high range of salaries for baseball players, to see the matching objects in these other linked views. However, the unselected objects are greyed out or put into the background in different ways—for example, the top of each bar in a bar chart stays the same height, even if it is only partially filled with the selection colour—so that the unselected objects stay visible as context.

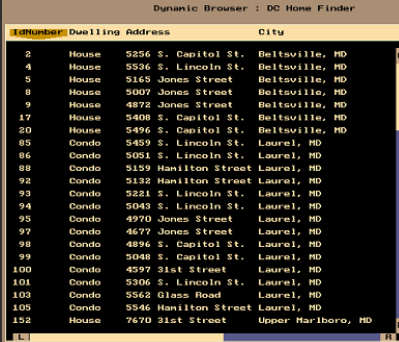
Dynamic Queries

Dynamic queries

Dynamic queries form another well-established but under-utilized technique.

Query and Results

```
SELECT house FROM palo_alto_homes  
WHERE price < 1,000,000 AND bedrooms > 2  
ORDER BY price
```



IdNumber	Dwelling Address	City
2	House 5256 S. Capitol St.	Beltsville, MD
4	House 5536 S. Lincoln St.	Beltsville, MD
5	House 5165 Jones Street	Beltsville, MD
8	House 5007 Jones Street	Beltsville, MD
9	House 4872 Jones Street	Beltsville, MD
17	House 5408 S. Capitol St.	Beltsville, MD
20	House 5496 S. Capitol St.	Beltsville, MD
85	Condo 5459 S. Lincoln St.	Laurel, MD
86	Condo 5051 S. Lincoln St.	Laurel, MD
88	Condo 5159 Hamilton Street	Laurel, MD
92	Condo 5132 Hamilton Street	Laurel, MD
93	Condo 5221 S. Lincoln St.	Laurel, MD
94	Condo 5043 S. Lincoln St.	Laurel, MD
95	Condo 4970 Jones Street	Laurel, MD
97	Condo 4677 Jones Street	Laurel, MD
98	Condo 4896 S. Capitol St.	Laurel, MD
99	Condo 5048 S. Capitol St.	Laurel, MD
100	Condo 4597 31st Street	Laurel, MD
101	Condo 5306 S. Lincoln St.	Laurel, MD
103	Condo 5562 Glass Road	Laurel, MD
105	Condo 5546 Hamilton Street	Laurel, MD
152	House 7670 31st Street	Upper Marlboro, MD

Query and results

[image shows a SELECT from an SQL system, with conditions for price, number of bedrooms, ordered by price... as well as a table showing the result of that query]

Originally this technique was really intended to replace database-style queries, such as this one selecting houses matching the given criteria or conditions.

Think about what it's like to just use a database-style query. An example is here in the slide.

Issues

1. For programmers
2. Rigid syntax
3. Only shows exact matches
4. Too few or too many hits
5. No hint on how to reformulate the query
6. Slow question-answer loop
7. Results returned as table

Issues with database queries

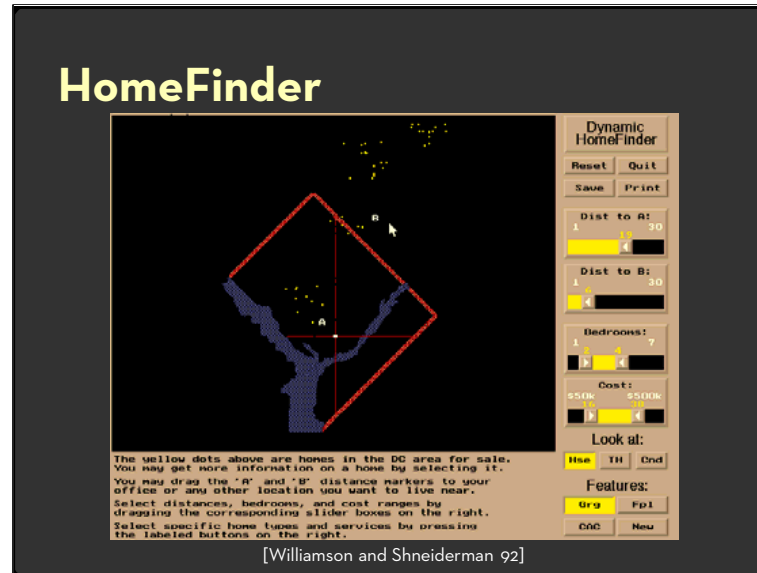
Most people can't write such SQL-style queries! Really, they are only for programmers. The syntax is too strict and rigid, and also this style only returns exact matches. We can't see the context of non-matching objects.

Often, it's hard to control how many objects such queries return. We can't see the space in which these queries operate, and so we often get too few or too many hits.

Once it's done, it's done. We can't see what is nearby in the space of information, so that we know how to adjust or reformulate the query.

Also, because one usually manually types these queries in (perhaps speeded up by cutting and pasting), it takes a long time to put together a question and get an answer back. The loop of questioning and answering is slow.

Finally, of course, the database style of result is the table... and that is limiting, as we've seen, in terms of comprehension and exploration.

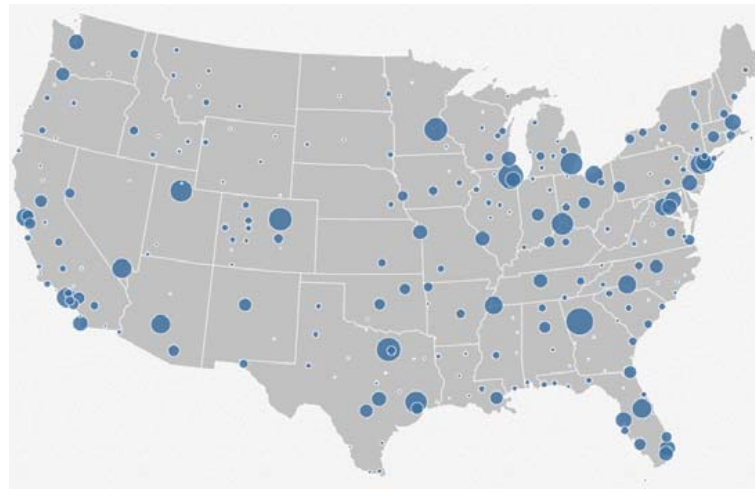


HomeFinder

[image shows a screenshot of the first system to use dynamic queries, HomeFinder, Williamson and Shneiderman, 1992. A map of houses for sale takes up the bulk of the screen, and then double-ended range sliders for each of the quantitative database attributes are on the right, along with some buttons for simple ordinal/nominal attributes.

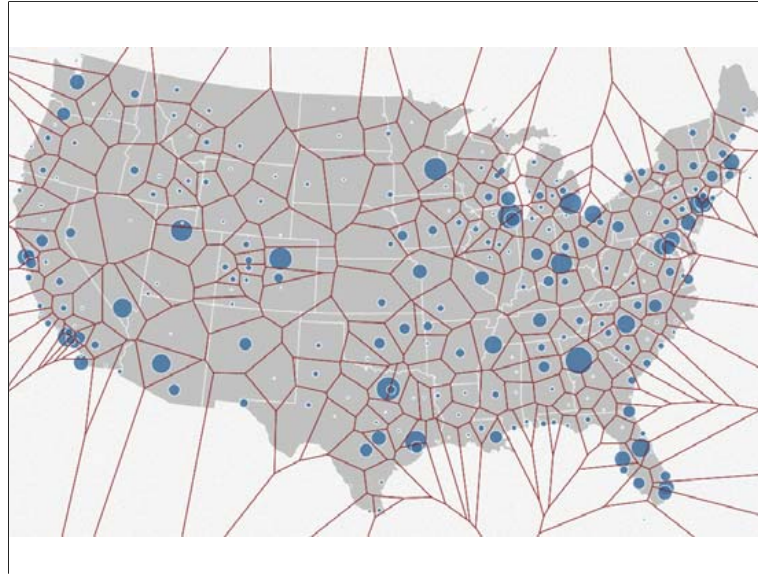
The HomeFinder system at the University of Maryland was one of the first to support the use of sliders to interactively adjust ranges and other query specifications. Chris Williamson was Ben Shneiderman's student, if I remember correctly, and they supported mouse-driven changes to the query... primarily via the sliders and buttons on the right... which supported real-time feedback and gave the user visual resources to guide reformulation of the query.

Note that it's impossible to make a database query that is syntactically incorrect here. The system will always create a valid SELECT, even though this means that it's constrained to a fairly simple set of ANDed expressions.



Consider this map of the US, with cities marked on it of various sizes, as blue circles. You can imagine the normal actions... for example, click on a blue circle and get more data on it.

Consider though that the cities define regions... and maybe we should be able to use that 'higher level' information about the cities.



[image shows same map as before, but overlaid with a Voronoi diagram. Each region defines the set of points around each city that are closer to that city than any other. At the edge of the map these regions are semi-infinite]

Here we have mapped out the minimal geometric regions around each city, without any concern for state lines, or shore lines, or anything else. We could use this structure though, so that if you click inside the region then you get the information on the city within it. This would make selection easier.

Also, though, this is using a *semantic structure* to drive the selection process.

https://en.wikipedia.org/wiki/Voronoi_diagram

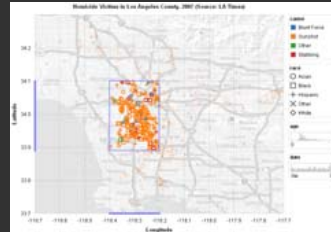
Generalized Selection

Generalised Selection

A number of researchers have advanced more general selection processes and techniques, based on these ideas. We'll look at one style now...

Visual Queries

Model selections as **declarative queries**.



$(-118.371 \leq \text{lon} \text{ AND } \text{lon} \leq -118.164) \text{ AND } (33.915 \leq \text{lat} \text{ AND } \text{lat} \leq 34.089)$

Visual Queries

[image shows a map of an urban area, with dots overlaid for homicide cases. A rectangular region has been drawn out, and all the dots outside it are greyed out. Top right is a legend that has the 4 colours used to encode 4 different causes, and 5 shapes to encode 5 races, and two double-ended sliders for the lat/long selection. Along the bottom is a Boolean expression: $(-118.371 \leq \text{lon} \text{ AND } \text{lon} \leq -118.164) \text{ AND } (33.915 \leq \text{lat} \text{ AND } \text{lat} \leq 34.089)$]

Consider a visual query, on a map or scatterplot. You might draw out a rectangle and so choose a set of symbols to be highlighted and selected.

We can define and record this selection in terms of just the set of selected objects – all the dots inside the rectangle we drew. However, we can also define it *declaratively*... in this case, as a query we can run again later, even if there is different data to be visualised.

An example is given here... the rectangle can be defined in terms of tree of Boolean operators applied to lat/long values: here, we are ANDing together two expressions,

each of which is also an AND expression. The leaves of the tree are conditions on either a latitude value or a longitude value, e.g. $\text{lat} \leq 34.089$

Visual Queries

Model selections as **declarative queries** over the domain of visualized data.

Applicable to **dynamic, time-varying data**

Retarget selection **across visual encodings**

Perform operations on **query structure**

We can therefore keep re-using this kind of declarative query. If the visualised data is dynamic, varying over time, then periodically (or when we detect a relevant change) we can re-run the query and update our selection.

We can also take that query, which is defined in terms of data properties (not visual encodings), and reuse it on a different visualisation... on a different encoding of the same data. This is very powerful when you have multiple linked views, all running at the same time, but one can also apply it to successive views displayed over time.

We can also generalise this kind of selection operation, in that the query is a structure that we can modify on the basis of user interactions or needs. We can adapt the query in the light of what the user is doing. We can manipulate that Boolean expression in all kinds of ways, to suit exploration and interaction.

Query Relaxation

Generalize an input query to create an expanded selection, according to:

1. A *semantic structure* describing the data
2. A *traversal policy* for that structure

Query relaxation

The strongest example of this is query relaxation...

This is relaxing... in other words, generalizing or expanding the selection... to include a semantically meaningful but wider set of objects. We use these two resources: a semantic structure that lets us make such a generalisation or relaxation, and a policy for moving over it, and so changing the selection expressed by it.

This is best described using, unsurprisingly, moving images. So, please now look at the video from folk who pioneered this technique at Berkeley University, ... It's on Moodle, as well as the next slide. I'll also put the original research paper up there.

Generalized Selection via Interactive Query Relaxation

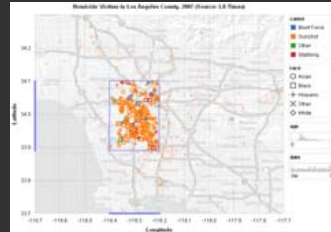
Jeffrey Heer | Maneesh Agrawala | Wesley Willett
University of California, Berkeley

[slide contains the video accompanying the research paper 'Generalized Selection via Interactive Query Relaxation' by Heer, Agrawala and Willett.

This video has a lot of really useful detail but it is not examinable material, as there is no accessible version of it, as far as I know.

Visual Queries

Model selections as **declarative queries**.



$(-118.371 \leq \text{lon} \text{ AND } \text{lon} \leq -118.164) \text{ AND } (33.915 \leq \text{lat} \text{ AND } \text{lat} \leq 34.089)$

Visual Queries

[same image and text as slide 69]

Let me summarise by going back to this image again from a few slides back...

We have a semantic structure describing homicide data – it's a tree of Boolean expressions, with leaves being conditions on the raw data attributes, like lat, long, cause of death, and race of victim. The example expression at the bottom of the slide shows the current state of that tree, using just two of those attributes at the moment... lat and long.. and values for them at the leaves.

Of course, we could change the lat/long values to make a slightly different rectangle, or to make it cover the entire map. We could also step up the tree, and add or remove Boolean expressions. For example, we could add in a new Boolean expression, other attributes of this homicide data. We could use another Boolean to add in an operator to select a particular cause of death, for example. The video offers examples such as removing all the lat/long constraints, and then filtering based on other attributes, like cause of death and race of victim.

The video also gives the example of taking the same tree of Booleans, and using it to create a different visualization, like a stacked bar chart. We can use this same declarative query, and then create a different view based on it... or you could have multiple linked views, like the map and the bar chart side-by-side, and then have the same declarative query used in both at the same time.

Data Selection

Selection can be about more than clicking on an object

- Selecting a region supports *brushing and linking*
- We can represent the selection by *tuple* or by *query*

Selection by query → powerful interactions, e.g.:

- Dynamic queries: using sliders, checkboxes, etc. to define the query, so it's easy to explore the data visually
- Generalised selection: using semantic structure of the data, to explore data at different levels of abstraction

So let me summarise...

Data selection can or should be about more than just clicking on individual objects or data items

The most common example of this is drawing out a region of the screen, so as to select many data items. This is the basic idea of brushing data, so that they are highlighted in linked views.

We can represent the selected items by tuple, i.e. as a set of individual data items, or by query, as a declarative specification of data items.

Selection by query lets us design and use some very powerful interaction methods. We've seen two here.

Dynamics queries rely on interaction devices, such as sliders and checkboxes, to define and adapt the query. The query is always syntactically valid, and we can get quick visual feedback on results.

Generalised selection uses the semantic structure of the data. It relies on what we know about the data at different levels of abstraction, so we can relax the query, to be 'higher level' or abstract, or constrain it to be more specific or 'low level'... all in ways that make sense with regard to the semantic model of the data.