CSE 442 - Data Visualization

Visualization Tools

Jeffrey Heer  University of Washington
How do people create visualizations?

**Chart Typology**
- Pick from a stock of templates
- Easy-to-use but limited expressiveness
- Prohibits novel designs, new data types

**Component Architecture**
- Permits more combinatorial possibilities
- Novel views require new operators, which requires software engineering
Graphics APIs
Canvas, OpenGL, Processing
ey = y;
size = s;

void update(int mx, int my) {
  angle = atan2(my-ey, mx-ex);
}

void display() {
  pushMatrix();
  translate(ex, ey);
  fill(255);
  ellipse(0, 0, size, size);
  rotate(angle);
  fill(153);
  ellipse(size/4, 0, size/2, size/2);
  popMatrix();
}
Graphics APIs
Canvas, OpenGL, Processing
Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Canvas, OpenGL, Processing
Data State Model
[Chi 98]
Prefuse & Flare
Operator-based toolkits for visualization design
Vis = (Input Data -> Visual Objects) + Operators

Prefuse (http://prefuse.org) Flare (http://flare.prefuse.org)
Component Architectures
 Prefuse, Flare, Improvise, VTK

Graphics APIs
 Canvas, OpenGL, Processing
Chart Typologies
Excel, Google Charts

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Canvas, OpenGL, Processing
Chart Typologies
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>4557808</td>
<td>0.03</td>
<td>4447100</td>
<td>0.1</td>
<td>0.07</td>
<td>0.24</td>
<td>0.13</td>
</tr>
<tr>
<td>Alaska</td>
<td>663661</td>
<td>0.06</td>
<td>626932</td>
<td>0.14</td>
<td>0.08</td>
<td>0.29</td>
<td>0.06</td>
</tr>
<tr>
<td>Arizona</td>
<td>5939292</td>
<td>0.16</td>
<td>5130632</td>
<td>0.4</td>
<td>0.08</td>
<td>0.27</td>
<td>0.13</td>
</tr>
<tr>
<td>Arkansas</td>
<td>2779154</td>
<td>0.04</td>
<td>2673400</td>
<td>0.14</td>
<td>0.07</td>
<td>0.25</td>
<td>0.14</td>
</tr>
<tr>
<td>California</td>
<td>36132147</td>
<td>0.07</td>
<td>33871648</td>
<td>0.14</td>
<td>0.07</td>
<td>0.27</td>
<td>0.11</td>
</tr>
<tr>
<td>Colorado</td>
<td>4665177</td>
<td>0.08</td>
<td>4301261</td>
<td>0.31</td>
<td>0.07</td>
<td>0.26</td>
<td>0.1</td>
</tr>
<tr>
<td>Connecticut</td>
<td>3510297</td>
<td>0.03</td>
<td>3405565</td>
<td>0.04</td>
<td>0.06</td>
<td>0.24</td>
<td>0.14</td>
</tr>
<tr>
<td>Delaware</td>
<td>843524</td>
<td>0.08</td>
<td>783600</td>
<td>0.18</td>
<td>0.07</td>
<td>0.23</td>
<td>0.13</td>
</tr>
<tr>
<td>Florida</td>
<td>17789864</td>
<td>0.11</td>
<td>15982378</td>
<td>0.24</td>
<td>0.06</td>
<td>0.23</td>
<td>0.17</td>
</tr>
<tr>
<td>Georgia</td>
<td>9072576</td>
<td>0.11</td>
<td>8186453</td>
<td>0.26</td>
<td>0.08</td>
<td>0.26</td>
<td>0.1</td>
</tr>
<tr>
<td>Hawaii</td>
<td>1275194</td>
<td>0.05</td>
<td>1211537</td>
<td>0.09</td>
<td>0.07</td>
<td>0.24</td>
<td>0.14</td>
</tr>
<tr>
<td>Idaho</td>
<td>1429096</td>
<td>0.1</td>
<td>1293953</td>
<td>0.29</td>
<td>0.07</td>
<td>0.27</td>
<td>0.11</td>
</tr>
<tr>
<td>Illinois</td>
<td>12763371</td>
<td>0.03</td>
<td>12419293</td>
<td>0.09</td>
<td>0.07</td>
<td>0.26</td>
<td>0.12</td>
</tr>
</tbody>
</table>
Choosing a visualization type for State Quick Facts

Analyze a text

**Tag Cloud**
How are you using your words? This enhanced tag cloud will show you the words popularity in the given set of text.

Learn more

**Wordle**
Wordle is a toy for generating 'word clouds' from text that you provide. The clouds give greater prominence to words that appear more frequently in the source text.

Learn more

**Word Tree**
See a branching view of how a word or phrase is used in a text. Navigate the text by zooming and clicking.

Learn more

Compare a set of values

**Bar Chart**
How do the items in your data set stack up? A bar chart is a simple and recognizable way to compare values. You can display several sets of bars for multivariate comparisons.

Learn more

**Block Histogram**
This versatile chart lets you get a quick sense of how a single set of data is distributed. Each item in the data is an individually identifiable block.

Learn more
Every Wednesday, when I get home from school, I have a piano lesson. My teacher is a very strict house. Her name is Hillary Clinton. Our piano is a Steinway Concert tree and it has 88 cups. It also has a soft pedal and a/an Smiley pedal. When I have a lesson, I sit down on the piano Alberto and play for 16 minutes. I do scales to exercise my cats, and then I usually play a minuet by Johann Sebastian Washington. Teacher says I am a natural Haunted House and have a good musical leg. Perhaps when I get better I will become a concert vet and give a recital at Carnegie hospital.
[M]ost charting packages channel user requests into a **rigid array of chart types**. To atone for this lack of flexibility, they offer a kit of post-creation editing tools to return the image to what the user originally envisioned. **They give the user an impression of having explored data rather than the experience.**

Leland Wilkinson

Chart Typologies
Excel, Many Eyes, Google Charts

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Canvas, OpenGL, Processing
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2, Vega-Lite

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Canvas, OpenGL, Processing
ggplot(diamonds, aes(x=price, fill=cut)) + geom_bar(position="dodge")
```r
ggplot(diamonds, aes(x=price, fill=cut)) + geom_bar(position="dodge")
```
ggplot2

```r
qplot(long, lat, data = expo, geom = "tile", fill = ozone,
      facets = year ~ month) +
scale_fill_gradient(low = "white", high = "black") + map
```
```javascript
Plot.plot({
  grid: true,
  facet: {
    data: athletes,
    y: "sex"
  },
  marks: [
    Plot.rectY(athletes, Plot.binX({y: "count"}, {x: "weight", fill: "sex"})),
    Plot.ruleY([0])
  ]
})
```
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2, Vega-Lite

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Canvas, OpenGL, Processing
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2, Vega-Lite

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Canvas, OpenGL, Processing
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2, Vega-Lite

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Canvas, OpenGL, Processing
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2, Vega-Lite

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Canvas, OpenGL, Processing
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2, Vega-Lite

Visualization Grammars
D3.js, Vega

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Canvas, OpenGL, Processing
Visualization Building Blocks
Visualization Building Blocks

Data

Input data to visualize
# Visualization Building Blocks

<table>
<thead>
<tr>
<th>Data</th>
<th>Input data to visualize</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transforms</td>
<td>Group, aggregate, stats, layout</td>
</tr>
</tbody>
</table>
Visualization Building Blocks

Data
- Input data to visualize

Transforms
- Group, aggregate, stats, layout

Scales
- Map data values to visual values
<table>
<thead>
<tr>
<th>Visualization Building Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data</strong></td>
</tr>
<tr>
<td><strong>Transforms</strong></td>
</tr>
<tr>
<td><strong>Scales</strong></td>
</tr>
<tr>
<td><strong>Guides</strong></td>
</tr>
</tbody>
</table>
Visualization Building Blocks

Data
Input data to visualize

Transforms
Group, aggregate, stats, layout

Scales
Map data values to visual values

Guides
Axes & legends visualize scales

Marks
Data-representative graphics

Area  Rect  Symbol  Image
Line  Text  Rule  Arc
d3.js Data-Driven Documents

Mike Bostock, Vadim Ogievetsky, Jeffrey Heer [TVCG 2011]
+ Jason Davies (geo, 2011–13) & Philippe Rivière (2016–)
What is D3?

1. A collection of reusable visualization utilities
2. A tool for updating the browser’s Document Object Model (DOM) in response to input data
What is D3?

1. A collection of reusable visualization utilities

   **Data:** d3.csv, d3.json, ...
   **Scales:** d3.scaleLinear, d3.scaleLog, ...
   **Projections:** d3.geoPath, d3.geoMercator, ...
   **Layout:** d3.tree, d3.treemap, d3.force, ...
   **Interaction:** d3.brush, d3.zoom, ...

2. A tool for updating the browser’s Document Object Model (DOM) in response to input data
What is D3?

1. A collection of reusable visualization utilities
2. A tool for updating the browser’s Document Object Model (DOM) in response to input data

**Select**: query DOM content  
**Join**: bind input data to DOM elements  
**Update**: set DOM element properties  
**Transition**: animate changes over time
Why D3?

Enable highly custom visualization design
Support animation and dynamic displays
Support rich and varied interactions
Interoperate via web standards (HTML, SVG, CSS)
Avoid artificial limits. If a browser can do it, D3 should be able to take advantage of it.
Why D3?

"the authors have undeniably helped to bring data visualization to the mainstream. [D3] is a cornerstone contribution to this conference specifically and more generally to the success of our field as a whole"

*IEEE VIS 2021 Test of Time Award*
Why D3?

D3 “slingshotted the field into growth, diversification and creativity that has been unprecedented” and “changed how millions of data visualizations are created across newsrooms, websites, and personal portfolios”

*Information is Beautiful 2022 Test of Time Award*
Why D3?

“Use D3 if you think it’s perfectly normal to write a hundred lines of code for a bar chart.”

Amanda Cox, Former Graphics Editor, NY Times
512 Paths to the White House

Select a winner in the most competitive states below to see all the paths to victory available for either candidate.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dam</td>
<td>Rep</td>
<td>Dam</td>
<td>Rep</td>
<td>Dam</td>
<td>Rep</td>
<td>Dam</td>
<td>Rep</td>
<td>Dam</td>
</tr>
</tbody>
</table>

Obama has 431 ways to win
84% of paths

5 ties
0.08% of paths

Romney has 76 ways to win
15% of paths

If Obama wins Florida...
If Romney wins Florida...

Florida
Ohio
North Carolina
Virginia
Wisconsin
Colorado
Iowa
Nevada
New Hampshire
D3 Selections

The core abstraction in D3 is a selection.
D3 Selections

The core abstraction in D3 is a selection.

```javascript
// Add and configure an SVG element (#<svg width="500" height="300">)
svg = d3.append("svg")
  .attr("width", 500)  // add new SVG to page body
  .attr("height", 300);  // set SVG width to 500px
  // set SVG height to 300px
```
Data

DOM
Data

```javascript
svg = d3.append("svg")
  .attr("width", 500)
  .attr("height", 300);
```

DOM

```html
<svg width="500" ... />
</svg>
```
D3 Selections

The core abstraction in D3 is a selection.

// Add and configure an SVG element (<svg width="500" height="300">)

```javascript
svg = d3.append("svg")  // add new SVG to page body
  .attr("width", 500)     // set SVG width to 500px
  .attr("height", 300);   // set SVG height to 300px

// Select & update existing rectangles contained in the SVG element

svg.selectAll("rect")  // select all SVG rectangles
  .attr("width", 100)    // set rect widths to 100px
  .style("fill", "steelblue");  // set rect fill colors
```
Data

```javascript
svg.selectAll("rect")
```

DOM

```html
<svg width="500" ...>

???

</svg>
```
Data

```
svg.selectAll("rect")
```

DOM

```
<svg width="500" ...>
  <rect ... />
  <rect ... />
  <rect ... />
  <rect ... />
  <rect ... />
</svg>
```
svg.selectAll("rect")
  .attr("width", 100)
  .style("fill", "steelblue")

<svg width="500" ...>
  <rect width="100"
       style="fill: steelblue;"
  />
  <rect width="100"
       style="fill: steelblue;"
  />
  <rect width="100"
       style="fill: steelblue;"
  />
Data Binding

Selections can *bind* data and DOM elements.

values = [ {...}, {...}, {...}, ... ];  // input data as JS objects
Data Binding

Selections can **bind** data and DOM elements.

```javascript
values = [ {...}, {...}, {...}, ... ]; // input data as JS objects

// Select SVG rectangles and bind them to data values.
bars = svg.selectAll("rect.bars").data(values);
```
```javascript
values = [
  { cat: "a", value: 5 },
  { cat: "b", value: 7 },
  { cat: "c", value: 3 },
  { cat: "d", value: 4 },
  { cat: "e", value: 6 }
];
```

```xml
<svg width=500 ...

</svg>
```
Data

values = [
    { cat: "a", value: 5 },
    { cat: "b", value: 7 },
    { cat: "c", value: 3 },
    { cat: "d", value: 4 },
    { cat: "e", value: 6 }
];

DOM

<svg width=500 ...

bars = svg.selectAll("rect") .data(values)
Data

values = [
  { cat: "a", value: 5 },
  { cat: "b", value: 7 },
  { cat: "c", value: 3 },
  { cat: "d", value: 4 },
  { cat: "e", value: 6 }
];

DOM

<svg width=500 ...>

bars = svg.selectAll("rect") .data(values)

</svg>
Data Binding

Selections can **bind** data and DOM elements.

```javascript
values = [ {…}, {…}, {…}, … ]; // input data as JS objects

// Select SVG rectangles and bind them to data values.
bars = svg.selectAll("rect.bars").data(values);

// What if the DOM elements don’t exist yet? The enter set represents data values that do not yet have matching DOM elements.
bars.enter().append("rect").attr("class", "bars");
```
Data

values = [
  { cat: "a", value: 5 },
  { cat: "b", value: 7 },
  { cat: "c", value: 3 },
  { cat: "d", value: 4 },
  { cat: "e", value: 6 }
];

DOM

<svg width=500 ...

bars = svg.selectAll("rect") .data(values)

</svg>
Data

values = [
  { cat: “a”, value: 5 },
  { cat: “b”, value: 7 },
  { cat: “c”, value: 3 },
  { cat: “d”, value: 4 },
  { cat: “e”, value: 6 }
];

DOM

<svg width=500 …>
  <rect />
  <rect />
  <rect />
  <rect />
  <rect />
</svg>

bars.enter().append(“rect”)
Data

values = [
  { cat: "a", value: 5 },
  { cat: "b", value: 7 },
  { cat: "c", value: 3 },
  { cat: "d", value: 4 },
  { cat: "e", value: 6 }
];

DOM

<svg width=500 ...
<rect class="bars" />
<rect class="bars" />
<rect class="bars" />
<rect class="bars" />
<rect class="bars" />
</svg>

bars.enter().append("rect").attr("class", "bars")
Data

values = [
    { cat: "a", value: 5 },
    { cat: "b", value: 7 },
    { cat: "c", value: 3 },
    { cat: "d", value: 4 },
    { cat: "e", value: 6 }
];

DOM

<svg width=500 …>

<rect x="…" />
<rect x="…" />
<rect x="…" />
<rect x="…" />
<rect x="…" />
</svg>

bars.enter().append("rect")
  .attr("x", d => xscale(d.cat))
Data

values = [
    { cat: "a", value: 5 },
    { cat: "b", value: 7 },
    { cat: "c", value: 3 },
    { cat: "d", value: 4 },
    { cat: "e", value: 6 }
];

DOM

<svg width=500 ...

bars.enter().append("rect")

    .attr("height", d =&gt; yscale(d.value))

</svg>
Data Binding

Selections can **bind** data and DOM elements.

```
values = [ {...}, {...}, {...}, ... ]; // input data as JS objects
```

// Select SVG rectangles and bind them to data values.
```
bars = svg.selectAll("rect.bars").data(values);
```

// What if the DOM elements don’t exist yet? The **enter** set represents data values that do not yet have matching DOM elements.
```
bars.enter().append("rect").attr("class", "bars");
```

// What if data values are removed? The **exit** set is a selection of existing DOM elements who no longer have matching data values.
```
bars.exit().remove();
```
Data

values = [
    { cat: "a", value: 5 },
    { cat: "b", value: 7 },
    { cat: "c", value: 3 },
    { cat: "d", value: 4 },
    { cat: "e", value: 6 }
];

DOM

<svg width=500 ...>
    <rect class="bars" />
    <rect class="bars" />
    <rect class="bars" />
    <rect class="bars" />
    <rect class="bars" />
</svg>
Data

values = [
  { cat: "a", value: 5 },
  { cat: "b", value: 7 },
  { cat: "c", value: 3 },
  { cat: "d", value: 4 },
  { cat: "e", value: 6 }
];

DOM

<svg width=500 ...
  <rect class="bars" />
  <rect class="bars" />
  <rect class="bars" />
  <rect class="bars" />
  <rect class="bars" />
</svg>

values.filter(d => ![‘b’, ‘d’].includes(d.cat))
```javascript
let values = [
  { cat: "a", value: 5 },
  { cat: "c", value: 3 },
  { cat: "e", value: 6 }
];

let bars = svg.selectAll("rect.bars").data(values);
```
values = [
    { cat: "a", value: 5 },
    { cat: "c", value: 3 },
    { cat: "e", value: 6 }
];

<svg width=500 ...
<rect class="bars" />
<rect class="bars" />
<rect class="bars" />
<rect class="bars" />
<rect class="bars" />
</svg>

bars.exit()
Data

```javascript
values = [
  { cat: "a", value: 5 },
  { cat: "c", value: 3 },
  { cat: "e", value: 6 }
];
```

DOM

```html
<svg width=500 ...>
  <rect class="bars" />
  <rect class="bars" />
  <rect class="bars" />
  <rect class="bars" />
  <rect class="bars" />
</svg>

bars.exit().remove()
```
Data

values = [
    { cat: "a", value: 5 },
    { cat: "c", value: 3 },
    { cat: "e", value: 6 }
];

DOM

<svg width=500 ...
    <rect class="bars" />
    <rect class="bars" />
    <rect class="bars" />
</svg>
The Data Join

DATA VALUES

ENTER
Data values without matching DOM elements.

UPDATE
Existing DOM elements, bound to valid data.

ELEMENTS

EXIT
DOM elements whose bound data has gone “stale”.
The Data Join

\[ \text{var } s = \text{d3.selectAll(...)}.\text{data}(...) \]

**DATA VALUES**

**ENTER**
Data values without matching DOM elements.

\[ s.\text{enter}().\text{append}(...) \]

**UPDATE**
Existing DOM elements, bound to valid data.

\[ s \]

**EXIT**
DOM elements whose bound data has gone “stale”.

\[ s.\text{exit}() \]
Data Binding

Selections can **bind** data and DOM elements.

```
values = [ {…}, {…}, {…}, … ]; // input data as JS objects
```

```
// Select SVG rectangles and bind them to data values.
bars = svg.selectAll("rect.bars").data(values)
  .join(
    enter => enter.append("rect"), // create new
    update => update, // update current
    exit => exit.remove() // remove outdated
  )
```
D3 Modules

Data Parsing / Formatting (JSON, CSV, ...)
Shape Helpers (arcs, curves, areas, symbols, ...)
Scale Transforms (linear, log, ordinal, ...)
Color Spaces (RGB, HSL, LAB, ...)
Animated Transitions (tweening, easing, ...)
Geographic Mapping (projections, clipping, ...)
Layout Algorithms (stack, pie, force, trees, ...)
Interactive Behaviors (brush, zoom, drag, ...)

Many of these correspond to future lecture topics!
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2, Vega-Lite

Visualization Grammars
D3.js, Vega

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Canvas, OpenGL, Processing

Expressiveness vs. Ease-of-Use
Administrivia
A2: Deceptive Visualization

Design two static visualizations for a dataset:

1. An earnest visualization that faithfully conveys the data.
2. A deceptive visualization that tries to mislead viewers.

Your two visualizations should address different questions.

Try to design a deceptive visualization that appears to be earnest. Can you trick your classmates and course staff?

You are free to choose your own dataset, but we have also provided some preselected datasets for you.

Submit two images and a brief write-up on Gradescope.

Due by Wed 1/24 11:59pm.
A2 Peer Reviews

You have been assigned two peer A2 submissions to review. For each:

• Try to determine which is earnest and which is deceptive
• Share a rationale for how you made this determination
• Share feedback using the “I Like / I Wish / What If” rubric

Assigned reviews will be posted on the A2 Peer Review page on Canvas, along with a link to a Google Form. You should submit two forms: one for each A2 peer review.

Due by Tue 1/30 11:59pm.
I Like... / I Wish... / What If?

I LIKE...
Praise for design ideas and/or well-executed implementation details. Example: "I like the navigation through time via the slider; the patterns observed as one moves forward are compelling!"

I WISH...
Constructive statements on how the design might be improved or further refined. Example: "I wish moving the slider caused the visualization to update immediately, rather than the current lag."

WHAT IF?
Suggest alternative design directions, or even wacky half-baked ideas. Example: "What if we got rid of the slider and enabled direct manipulation navigation by dragging data points directly?"
A3: Interactive Prototype

Create an interactive visualization. Choose a driving question for a dataset and develop an appropriate visualization + interaction techniques, then deploy your visualization on the web.

Due by 11:59pm on Monday, Feb 12.

Work in project teams of 3-4 people.
Form A3 + Final Project Team

Form a **team of 3-4** for A3 and the Final Project.
Submit signup form by **Wed 1/31, 11:59pm**.

If you do not have team mates, post on Ed about your interests/skills/project ideas!

We will send out a reminder early next week.
Requirements

**Interactive.** You must implement interaction methods! However, this is not only selection / filtering / tooltips. Also consider annotations or other narrative features to draw attention and provide additional context.

**Web-based.** D3/Vega-Lite are encouraged, but not required. Deploy to web using GitHub pages.

**Write-up.** Provide design rationale.
Interactive Prototype Tips

Start now. It will take longer than you think.

Keep it simple. Choose a minimal set of interactions that enables users to explore and generate interesting insights. Do not feel obligated to convey everything about the data: focus on a compelling subset.

Promote engagement. How do your chosen interactions reveal interesting observations?
D3 Tutorial - In Class Thu Apr 27

D3.js Deep Dive led by Andrew and Raymond

Be sure to read the D3, Part 1 notebook ahead of time. We’ll work through Part 2 in class.

Bring your laptops and follow along in real-time.
Web Publishing Tutorial - Fri Feb 2

On Zoom, led by Lisa and Ron

Gain skills publishing projects to the web:
- Publish sites using GitLab pages
- Embed Observable cells into external pages
- Navigate possible web application frameworks
A Visualization Tool Stack
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2, Vega-Lite

Visualization Grammars
D3.js, Vega

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Canvas, OpenGL, Processing
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2, Vega-Lite

Visualization Grammars
D3.js, Vega

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Canvas, OpenGL, Processing
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2, Vega-Lite

Visualization Grammars
D3.js, Vega

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Canvas, OpenGL, Processing
What is a Declarative Language?

Programming by describing what, not how

Separate specification (what you want) from execution (how it should be computed)

In contrast to imperative programming, where you must give explicit steps.
What is a Declarative Language?

Programming by describing *what*, not *how*

Separate **specification** (*what you want*) from **execution** (*how it should be computed*)

In contrast to **imperative programming**, where you must give explicit steps.

d3.selectAll("rect")
  .data(my_data)
  .join("rect")
  .attr("x", d => xscale(d.foo))
  .attr("y", d => yscale(d.bar))
```
SELECT customer_id, customer_name, COUNT(order_id) as total
FROM customers
INNER JOIN orders ON customers.customer_id = orders.customer_id
GROUP BY customer_id, customer_name
HAVING COUNT(order_id) > 5
ORDER BY COUNT(order_id) DESC
```
Why Declarative Languages?

Faster iteration, less code, larger user base?

Better visualization. *Smart defaults.*

Reuse. *Write-once, then re-apply.*

Performance. *Optimization, scalability.*

Portability. *Multiple devices, renderers, inputs.*

Programmatic generation.

*Write programs which output visualizations. Automated search & recommendation.*
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2, **Vega-Lite**

Visualization Grammars
D3.js, **Vega**

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2, Vega-Lite

Visualization Grammars
D3.js, Vega

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
Visual Analysis Grammars
  VizQL, ggplot2, **Vega-Lite**

Visualization Grammars
  D3.js, **Vega**

Component Architectures
  Prefuse, Flare, Improvise, VTK

Graphics APIs
  Processing, OpenGL, Java2D

Declarative Languages

Programming Toolkits
Interactive Data Exploration
- Tableau,
  - Lyra,
  - Voyager

Visual Analysis Grammars
- VizQL, ggplot2,
  - Vega-Lite

Visualization Grammars
- D3.js, Vega

Component Architectures
- Prefuse, Flare, Improvise, VTK

Graphics APIs
- Processing, OpenGL, Java2D
The Lyra Visualization Design Environment (VDE) alpha
Arvind Satyanarayan, Kanit “Ham” Wongsuphasawat, Jeffrey Heer

UW Interactive Data Lab PROJECTS

See also: Charticulator, Data Illustrator
Lyra  A Visualization Design Environment

Driving Shifts into Reverse by Hannah Fairfield, NYTimes
Lyra  A Visualization Design Environment

DATA PIPELINES

Wheat Wages
Monarchs
Pipeline name: Monarchs
Import data from: monarchs2

Extend
formula: (name == 'Cromwell')
as: commonwealth

Extend
formula: commonwealth & index
as: offset

name  Elizabeth  James  Charles  1  Cromwell  Charles  II  Anne  George  I  George  II  George  III  George  IV
start  1565  1603  1625  1649  1660  1689  1714  1727  1760  1760
end    1603  1625  1649  1660  1689  1714  1727  1760  1760
index  0       1       2       3     1       0       0       0       0
commonw...  false  false  false  true
offset    0       1       0       0

50 Years

VISUAL LAYERS

ChArt
Shewing at One View
The Price of The Quarter of Wheat,
& Wages of Labour by the Week
from The Year 1565 to 1821
by WILLIAM PLAYFAIR

by William Playfair
Lyra A Visualization Design Environment

based on the Railway Timetable by E. J. Marey
Lyra: A Visualization Design Environment

ZipScribble by Robert Kosara
Lyra: A Visualization Design Environment

Napoleon’s March by Charles Minard
Voyager. Wongsuphasawat et al. InfoVis’15, CHI’17
Common exploration pitfalls:
Overlook data quality issues
Fixate on specific relationships
Plus many other biases...

[Heuer 1999, Kahneman 2011, …]
**Key Idea:** Augment manual exploration with visualization recommendations sensitive to the user’s current focus.

The goal is to support systematic consideration of the data, without exacerbating false discovery.

To model a user’s search frontier, we enumerate related Vega-Lite specifications, seeded by the user’s current focus.

Candidate charts are pruned and ranked using models of estimated perceptual effectiveness.
A Formal Design Space of Visualizations

Enumerate Vega-Lite specifications and transformations among them. Search the space using logic programming methods.

[Kim et al. 2017]
Articulate Design Constraints

“Quantitative axes should include a zero baseline”
When and how strongly should we apply this?
How to balance with other such constraints?

[Moritz et al. 2019]
Learn Design Trade-Offs from Data

Training Data
Pairs of Ranked Visualizations

Features
Violations of Design Constraints

Learning Algorithm
Learning to Rank with Linear SVM

$\arg \max_w \sum_{i \in 0...k} w_i (u_i - v_i)$

$w$ is the weight vector of the soft constraints

$v_i$: the number of violations of constraint $i$.

[Moritz et al. 2019]
Compared to other tools, **over 4x more variable sets seen**, and **over 2x more interacted with**.

“related view suggestion accelerates exploration a lot.”

“I like that it shows me what fields to include in order to see a specific graph. Otherwise, I have to do a lot of trial and error and can't express what I wanted to see.”

“These related views are so good but it’s also spoiling that I start thinking less. I’m not sure if that’s really a good thing.”