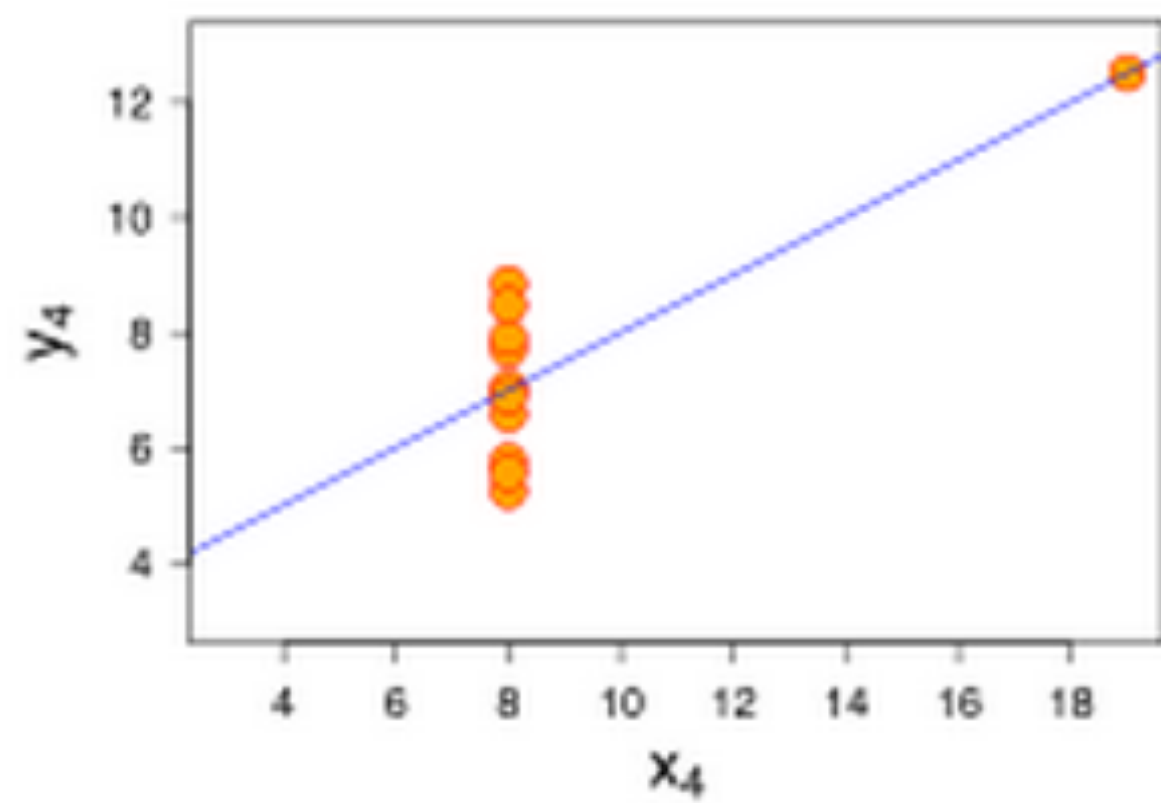
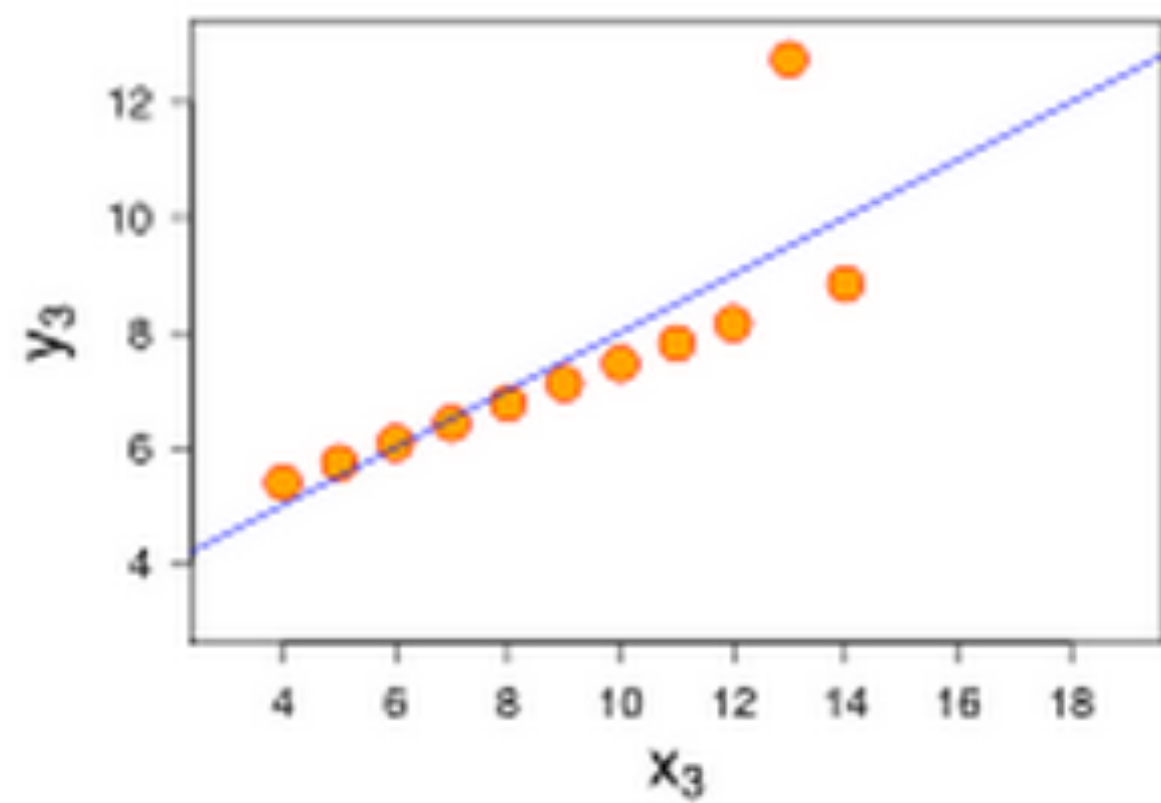
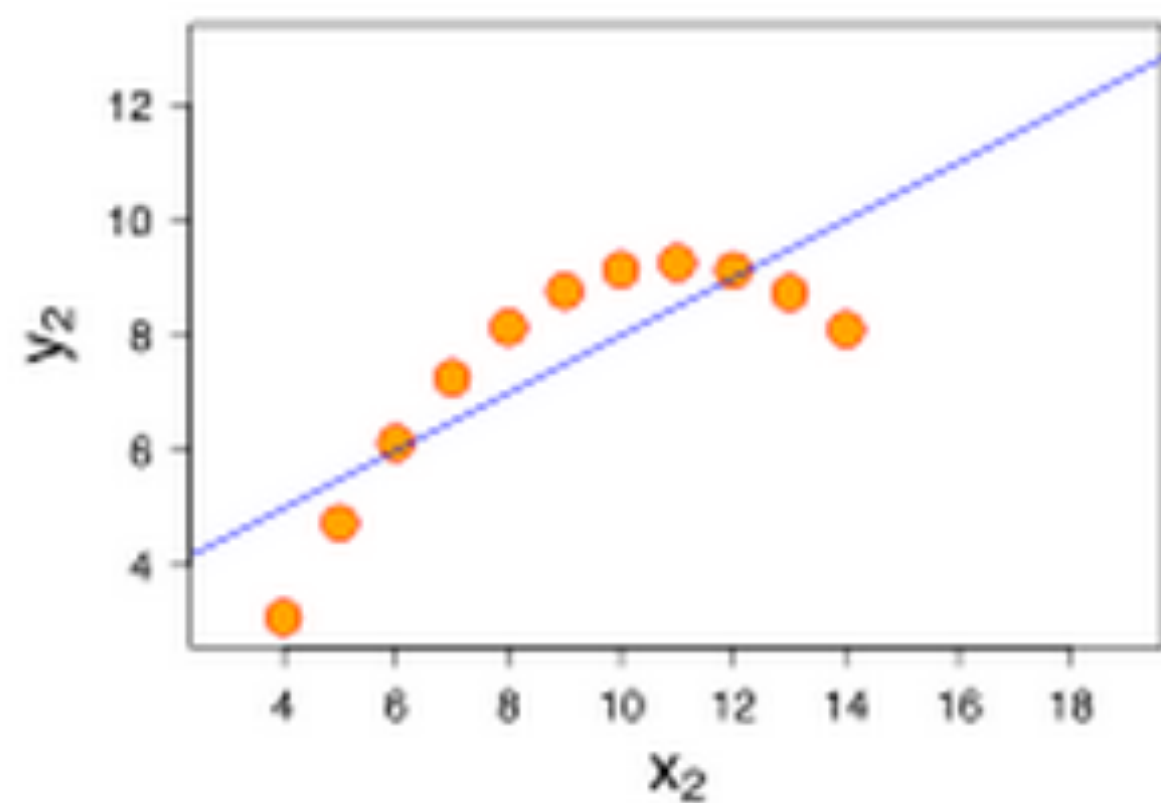
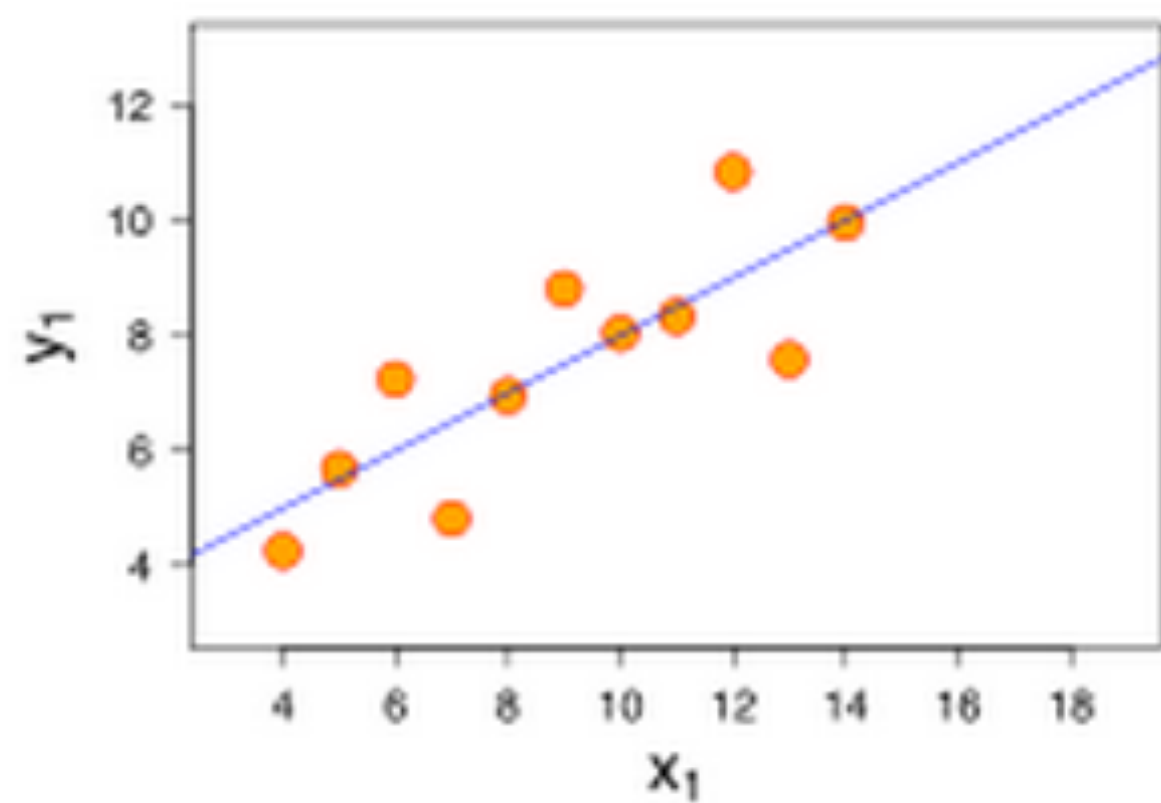


# Retrospective, Wrap-Up, What's Next

CSC444

Property	Value
Mean of $x$ in each case	9 (exact)
Sample variance of $x$ in each case	11 (exact)
Mean of $y$ in each case	7.50 (to 2 decimal places)
Sample variance of $y$ in each case	4.122 or 4.127 (to 3 decimal places)
Correlation between $x$ and $y$ in each case	0.816 (to 3 decimal places)
Linear regression line in each case	$y = 3.00 + 0.500x$ (to 2 and 3 decimal places, respectively)

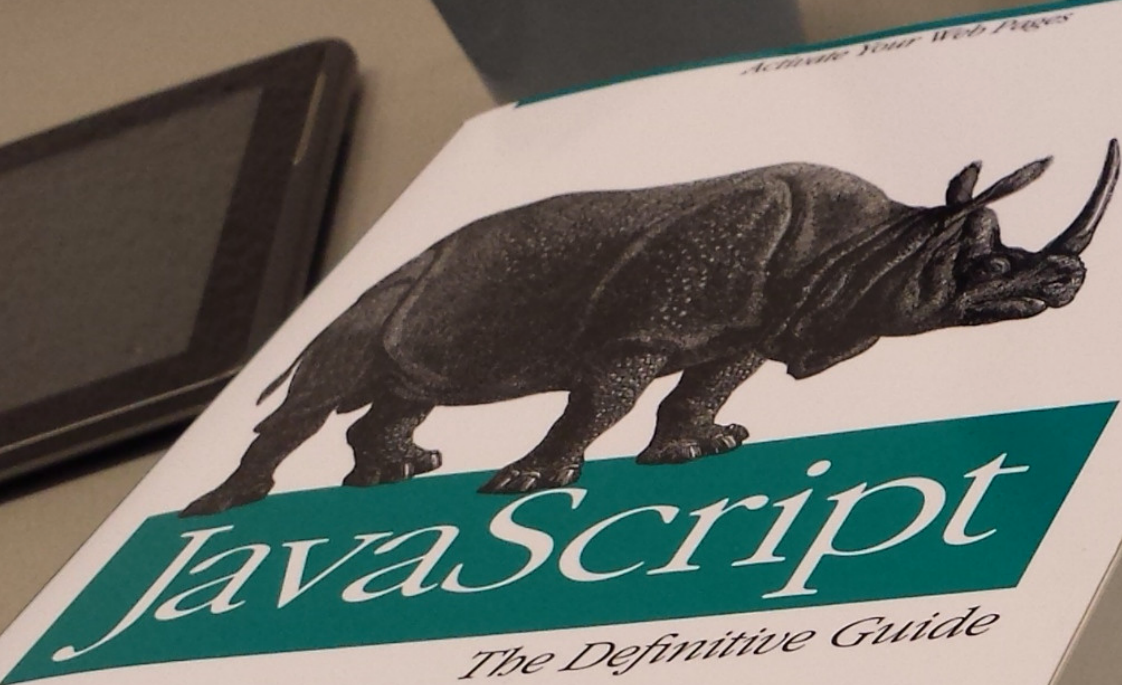
[http://en.wikipedia.org/wiki/Anscombe%27s\\_quartet](http://en.wikipedia.org/wiki/Anscombe%27s_quartet)



We do visualization not  
because it's pretty  
(although it can  
certainly be!), but  
because **it works  
better**



# Mechanics

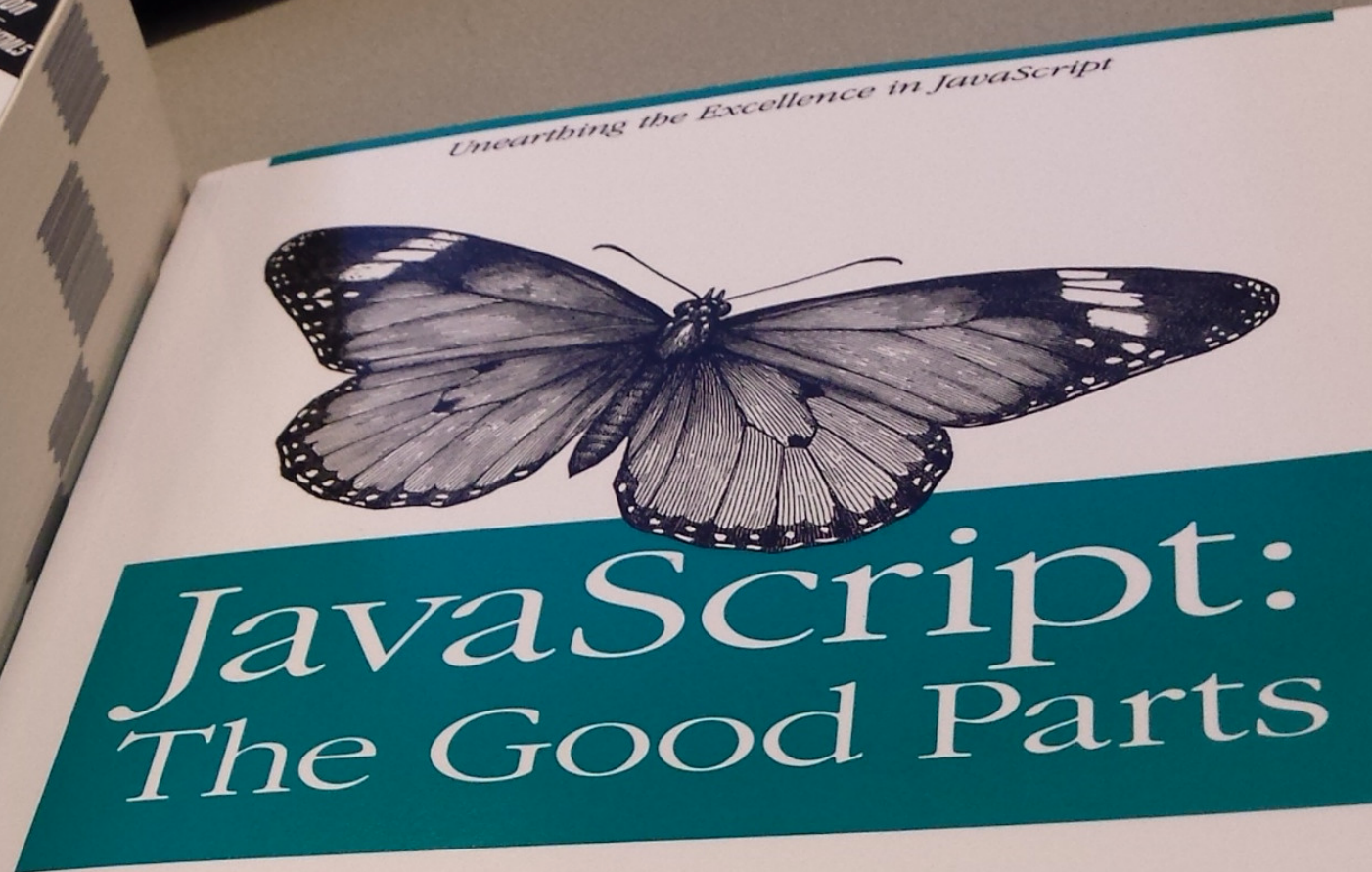


David Flanagan

O'REILLY®

YAHOO! PRESS

Douglas Crockford







**D3.js** is a JavaScript library for manipulating documents based on data. **D3** helps you bring data to life using HTML, SVG, and CSS. D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

See [more examples](#).

# Why did we bother?

- It's the state of the art
  - (I know, right?! If you care, come help me fix it!)
- It's what actually gets used in the real world

 d3 / d3

 Watch ▾

3,702

★ Unstar

75,141

 Fork

19,153

- What you learned in this class is exactly what the New York Times pros use

# What did we leave out?

- We learned how to use d3, and we learned how to write a part of it
- But we didn't go into a lot of detail of how d3 is implemented
- If we want to improve things, we must first understand them
- API design for visualization is important!

# What did we leave out?

- Web technologies for more complex graphics
  - Canvas, WebGL
- Non-web technologies
  - Raw OpenGL, for when all else fails



## SVG: ~1K points

```
svg.append("rect")
  .attr("class", "overlay")
  .attr("width", width)
  .attr("height", height);

var circle = svg.selectAll("circle")
  .data(data)
  .enter().append("circle")
  .attr("r", 2.5)
  .attr("transform", transform);

function zoom() {
  circle.attr("transform", transform);
}

function transform(d) {
  return "translate(" + x(d[0]) + "," + y(d[1])
}
```

<http://bl.ocks.org/mbostock/3680957>

## Canvas: ~50K points

```
function zoom() {
  canvas.clearRect(0, 0, width, height);
  draw();
}

function draw() {
  var i = -1, n = data.length, d, cx, cy;
  canvas.beginPath();
  while (++i < n) {
    d = data[i];
    cx = x(d[0]);
    cy = y(d[1]);
    canvas.moveTo(cx, cy);
    canvas.arc(cx, cy, 2.5, 0, 2 * Math.PI);
  }
  canvas.fill();
}
```

<http://bl.ocks.org/mbostock/3681006>

# WebGL: ~1M points

```
1 function WebGLCircleRenderer(glowContext, circleCount, colors, radii, alpha) {
2     this.context = glowContext;
3     this.count = circleCount;
4
5     var vertShader = [
6         "uniform mat4 u_matrix;",
7         "attribute float a_x;",
8         "attribute float a_y;",
9         "attribute float a_radius;",
10        "attribute vec3 a_color;",
11        "varying vec3 v_color;",
12
13        "void main() {",
14            "    gl_PointSize = a_radius;",
15            "    gl_Position = u_matrix * vec4(a_x, a_y, 1.0, 1.0);",
16            "    v_color = a_color;",
17        "}"
18    ].join("\n");
19
20    var fragShader = [
21        "precision mediump float;",
22        "uniform float u_alpha;",
23        "varying vec3 v_color;",
24
25        "void main() {",
26            "    float centerDist = length(gl_PointCoord - 0.5);",
27            "    float radius = 0.5;",
28            // works for overlapping circles if blending is enabled
29            "    gl_FragColor = vec4(v_color, u_alpha * step(centerDist, radius));",
30        "}"
31    ].join("\n");
32
33    var circleShaderInfo = {
34        vertexShader: vertShader,
35        fragmentShader: fragShader,
36
37        data: {
38            // uniforms
39            // Use a transformation matrix that makes 1 unit 1 pixel.
40            u_matrix: { value: new Float32Array([
41                2 / this.context.width, 0, 0, 0,
42                0, 2 / this.context.height, 0, 0,
43                0, 0, 1, 0,
44                -1, -1, 0, 1
45            ])}},
46            u_alpha: { value: new Float32Array([alpha]) },
47
48            // attributes
49            a_color: new Float32Array(colors),
50            a_radius: new Float32Array(radii),
51            a_x: new Float32Array(circleCount),
52            a_y: new Float32Array(circleCount)
53        },
54
55        primitives: this.context.GL.POINTS,
56
57        interleave: {
58            a_x: false,
59            a_y: false
60        },
61
62        usage: {
63            a_x: this.context.GL.DYNAMIC_DRAW,
64            a_y: this.context.GL.DYNAMIC_DRAW
65        }
66    };
67
68    this.shader = new GLOW.Shader(circleShaderInfo);
69 }
70
71 WebGLCircleRenderer.prototype.setPositions = function(xs, ys) {
72     this.shader.attributes.a_x.bufferSubData(xs);
73     this.shader.attributes.a_y.bufferSubData(ys);
74 };
75
76 WebGLCircleRenderer.prototype.draw = function() {
77     this.shader.draw();
78 };
79
80 WebGLCircleRenderer.prototype.dispose = function() {
81     delete this.context;
82     this.shader.dispose();
83     delete this.shader;
84 };
```

```
27 var context, stats, animationID, circleRenderer;
28 function initPage() {
29     var container = document.getElementById("container");
30
31     context = new GLOW.Context({
32         width: container.offsetWidth,
33         height: container.offsetHeight,
34         alpha: false
35     });
36     if (null === context.GL) {
37         alert("no WebGL");
38         return false;
39     }
40
41     container.appendChild(context.domElement);
42     context.setupClear( { red: 0, green: 0, blue: 0 } );
43     context.GL.enable(context.GL.BLEND);
44     context.GL.blendFunc(context.GL.SRC_ALPHA,
45                         context.GL.ONE_MINUS_SRC_ALPHA);
46
47     stats = new Stats();
48     stats.setMode(0);
49     stats.domElement.style.position = 'absolute';
50     stats.domElement.style.left = '0px';
51     stats.domElement.style.top = '0px';
52     document.body.appendChild( stats.domElement );
53
54     return true;
55 }
56
57 function initCircles() {
58     if (animationID !== undefined) {
59         cancelAnimationFrame(animationID);
60         circleRenderer.dispose();
61     }
62
63     var numPoints = parseInt(document.getElementById("numCircles").value);
64     var minRadius = 5;
65     var maxRadius = parseInt(document.getElementById("maxRadius").value);
66     var alpha = parseFloat(document.getElementById("alpha").value);
67     var maxVelocity = 1.5;
68     var bands = 3;
69     var bandWidth = 0.75;
70     var pointsPerBand = (numPoints / bands) | 0;
71
72     var colors = new Float32Array(numPoints * 3);
73     var xs = new Float32Array(numPoints);
74     var ys = new Float32Array(numPoints);
75     var radii = new Float32Array(numPoints);
76     var phase = new Float32Array(numPoints);
77
78     for (var band = 0; band < bands; band++) {
79         for (var i = 0; i < pointsPerBand; i++) {
80             var point = (band * pointsPerBand) + i;
81             colors[(point * 3) + ((band + 0) % 3)] = 0.8 * (1 / pointsPerBand);
82             colors[(point * 3) + ((band + 1) % 3)] = 1;
83             colors[(point * 3) + ((band + 2) % 3)] = 0.8 * (1 - (1 / pointsPerBand));
84
85             xs[point] = (1 / pointsPerBand) * context.width;
86             ys[point] = ((band / bands) * context.height) + (Math.random() * ((context.height * bandWidth) / bands));
87             radii[point] = minRadius + (Math.random() * (maxRadius - minRadius));
88             phase[point] = Math.random() * Math.PI * 2;
89         }
90     }
91 }
```

```
93 circleRenderer = new WebGLCircleRenderer(context, numPoints,
94                                           colors, radii, alpha);
95
96 var theta = 0;
97 var dTheta = 0.01;
98 var multiplier = 1.5;
99 function step() {
100     stats.begin();
101
102     theta = (theta + dTheta) % (Math.PI * 2);
103     for (var i = 0; i < numPoints; i++) {
104         ys[i] += Math.sin(theta + phase[i]) * multiplier;
105     }
106     circleRenderer.setPositions(xs, ys);
107
108     context.cache.clear();
109     context.clear();
110     circleRenderer.draw();
111     animationID = requestAnimationFrame(step);
112
113     stats.end();
114 }
115
116 animationID = requestAnimationFrame(step);
117 }
118
119 if (initPage()) {
120     var drawButton = document.getElementById("drawButton");
121     drawButton.onclick = initCircles;
122     initCircles();
123 }
124 }
```



# CUDA/OpenGL: 32M points

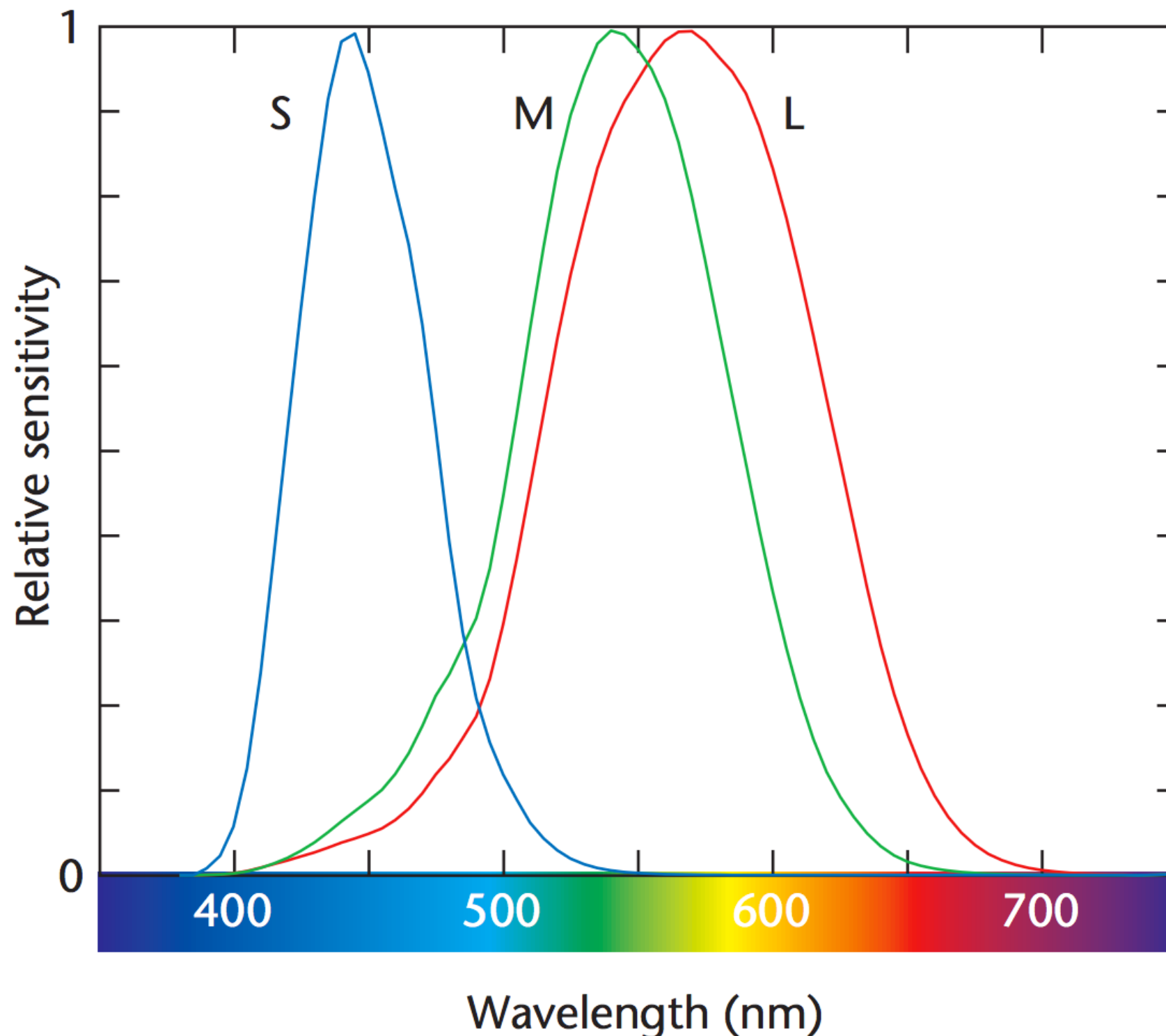


<https://www.youtube.com/watch?v=NDLPoJsqqoA>

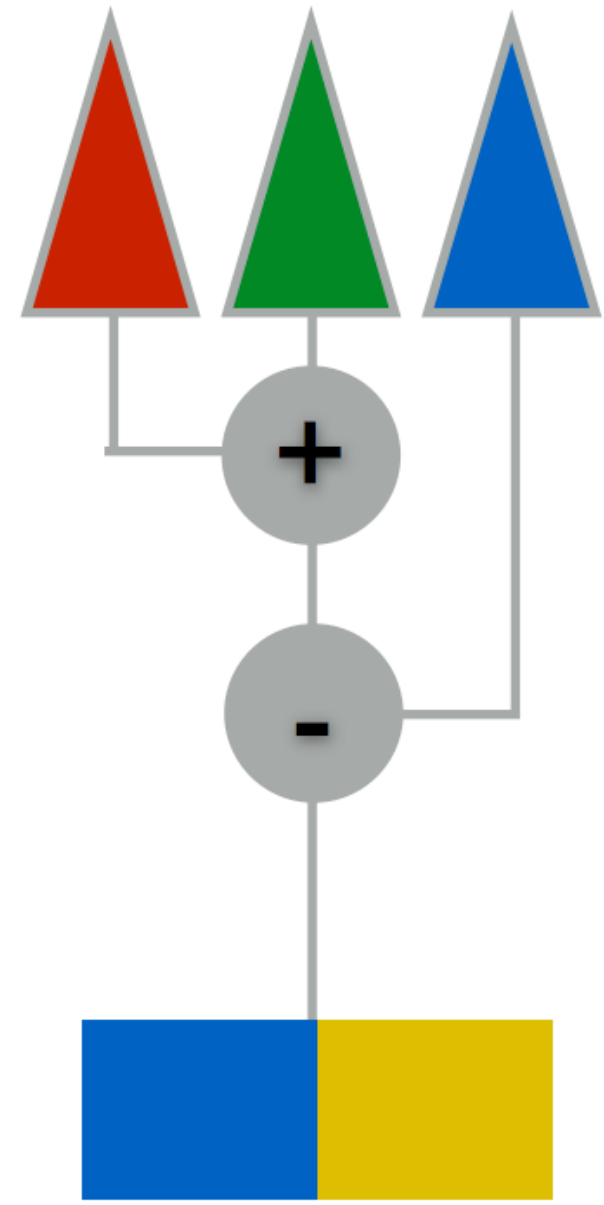
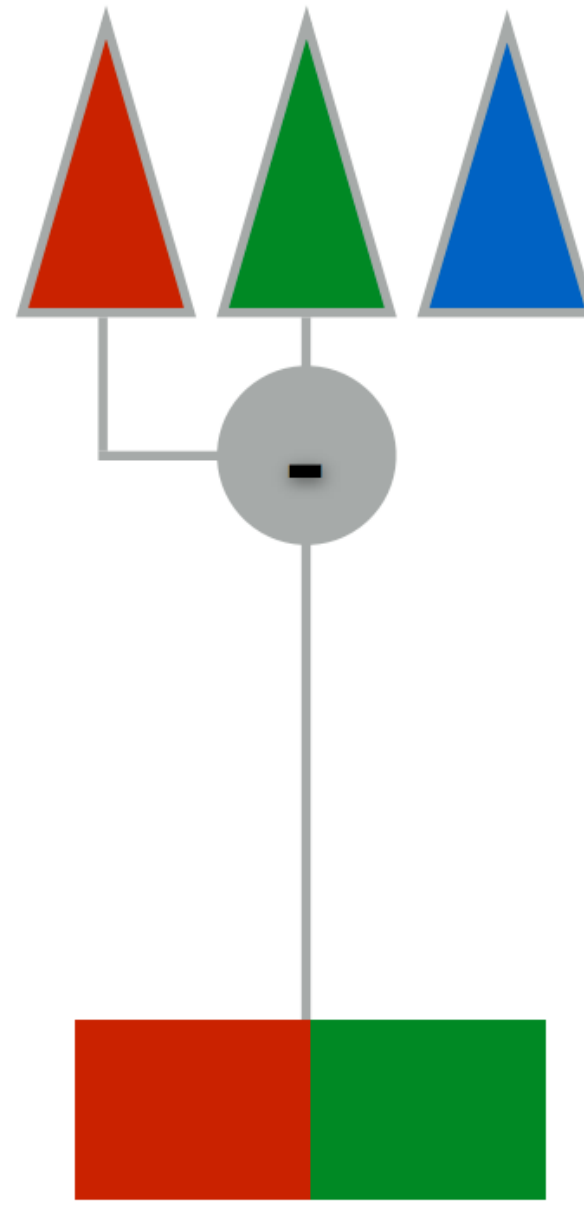
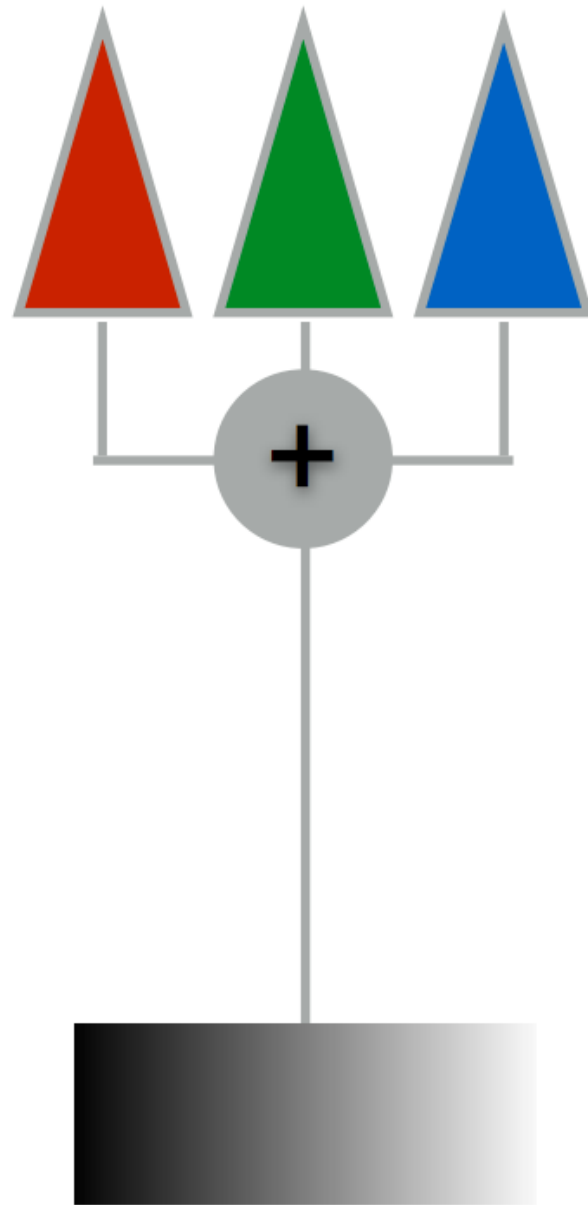
# Principles

# Color Vision

# How does your eye work?



# OPPONENT PROCESS MODEL



# Polar Lab (or HCL)

- “Perceptually uniform”, like Lab
- Transform ab to polar coordinates: radius is Chroma, Angle is Hue
- **Like HSV, but device-independent. All else being equal, think HCL first**

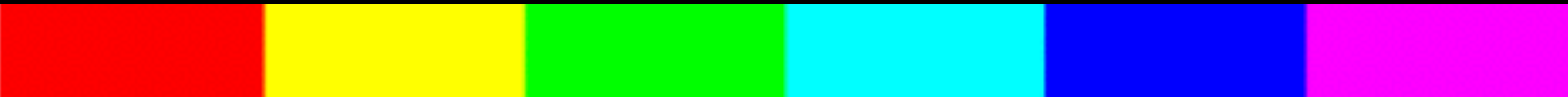
[http://cscheid.net/static/20120216/hcl\\_frame.html](http://cscheid.net/static/20120216/hcl_frame.html)

If you're going to use the rainbow colormap, use an **isoluminant** version, **quantize** it, or **both**



**Bad**







**Better**



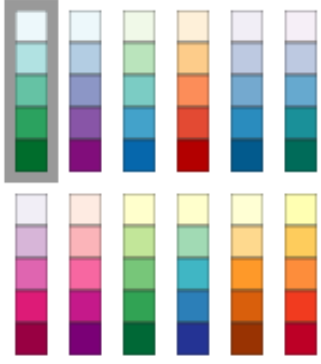



Number of data classes: 3   [how to use](#) [updates](#) [downloads](#) [credits](#)


**COLORBREWER 2.0**  
color advice for cartography

Nature of your data:  sequential  diverging  qualitative 


Pick a color scheme:


Multi-hue: 


Single hue: 




Only show: 


☐ colorblind safe  
☐ print friendly  
☐ photocopy safe





Context: 


☐ roads  
☐ cities  
☒ borders 

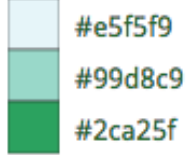
Background: 

 solid color  terrain 

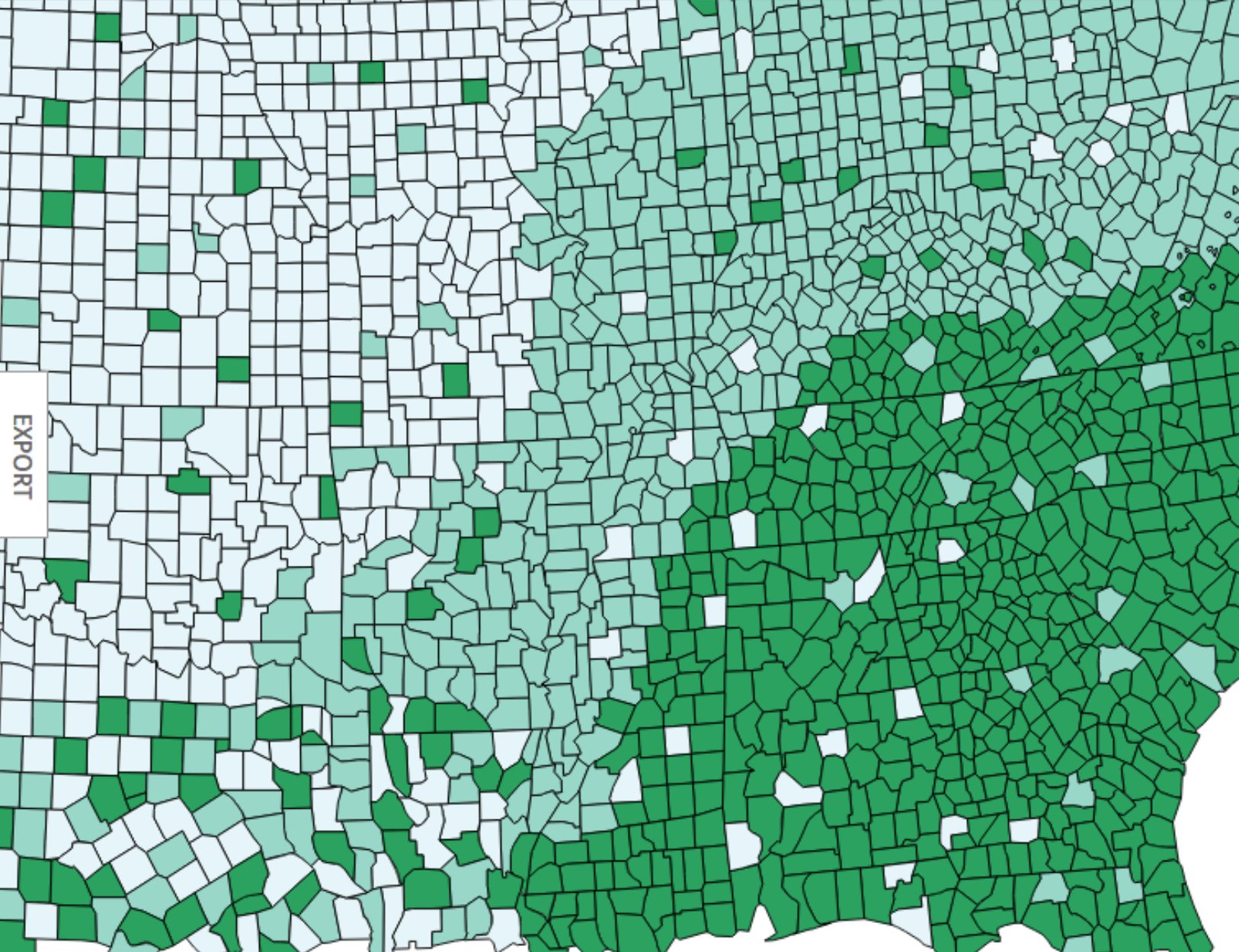
  
color transparency

3-class BuGn    


HEX 

  
#e5f5f9  
#99d8c9  
#2ca25f

EXPORT



© Cynthia Brewer, Mark Harrower and The Pennsylvania State University  
[Support](#)  
[Back to Flash version](#)  
[Back to ColorBrewer 1.0](#)



# COLORBREWER



Generate



Number of colors

1

Score importance

Perceptual Distance



Name Difference



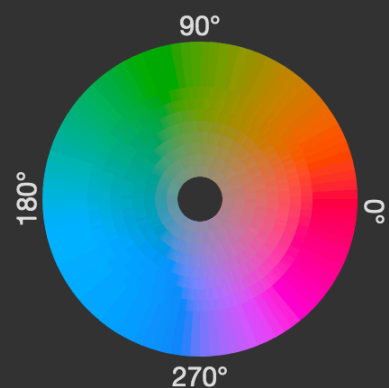
Pair Preference



Name Uniqueness



Select hue filters



Results: Color space Hex RGB Lab LCH

Array format " ' No quote

Charts

## Instructions

To generate a palette with  $n$  colors, just enter the number of colors you want and click *Generate*. Bigger palettes will take longer than smaller palettes to make. Results will automatically appear when ready.

For greater detail, please consult our [paper](#) or the [source code](#).

### Score Importance

#### Perceptual Distance

Increasing *Perceptual Distance* favors palette colors that are more easily discriminable to the human eye. To accurately model human color acuity, this is performed using [CIEDE2000](#) in [CIE Lab](#) color space.

#### Name Difference

Increasing *Name Difference* favors palette colors that share few common names. This is similar to perceptual distance, but can lead to different results in certain areas of color space. This happens when there are many different names for perceptually close colors (e.g., red and pink are perceptually close but named differently). Colorgorical calculates this using Heer and Stone's [Name Difference function](#), which is built on top of the [XKCD color-name survey](#).

#### Pair Preference

Increasing *Pair Preference* favors palette colors that are, on average, predicted to be more aesthetically preferable together. Typically these colors are similar in hue, have different lightness, and are cooler colors (blues and greens). Pair Preference is based off of [Schloss and Palmer's research on color preference](#).

#### Name Uniqueness

Increasing *Name Uniqueness* favors palette colors that are uniquely named. Some colors like **red**

## About

Colorgorical was built by Connor Gramazio.

## Documentation

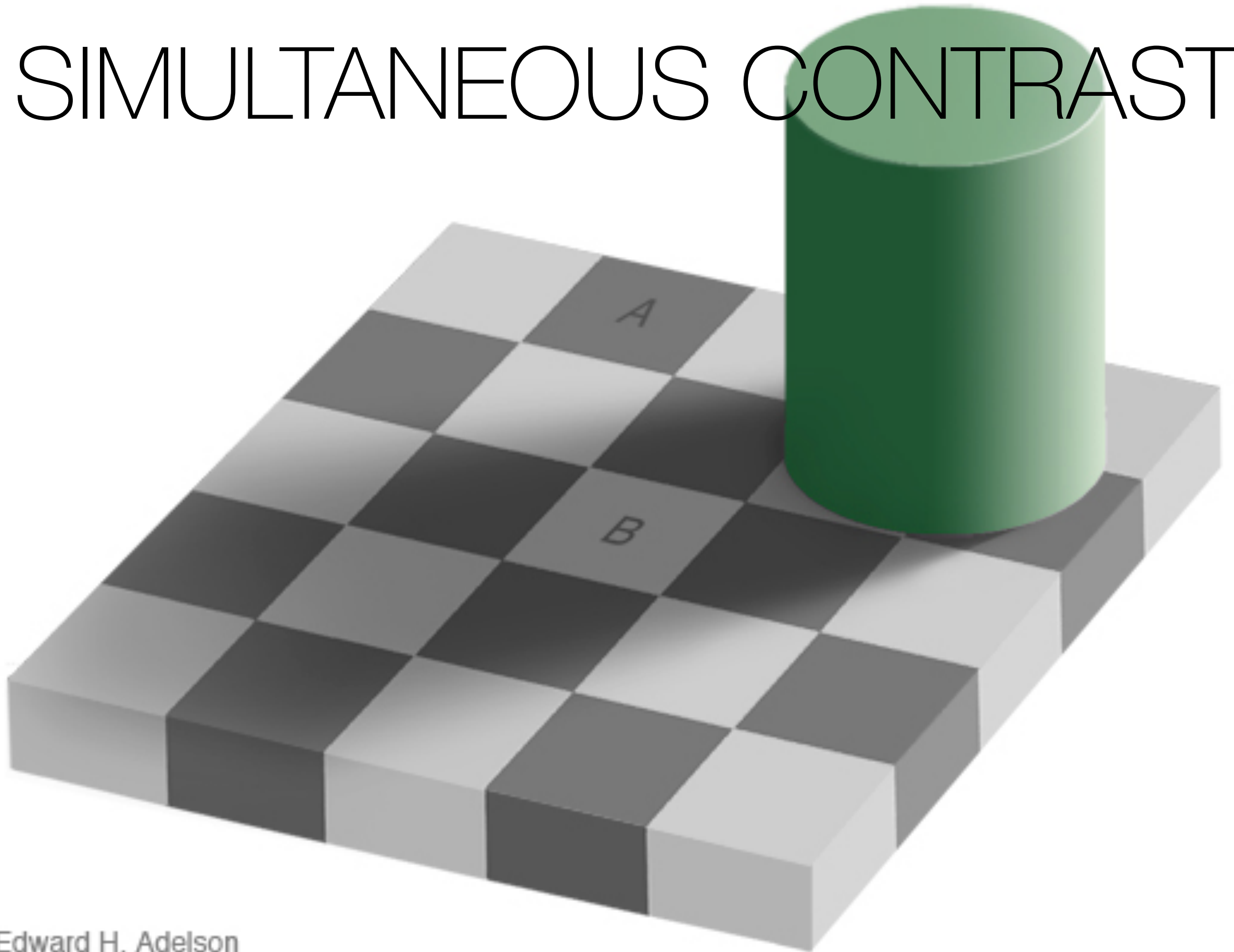
If you'd like to read more about how Colorgorical works, please see the [documentation](#).

If you use Colorgorical, please use the following citation:

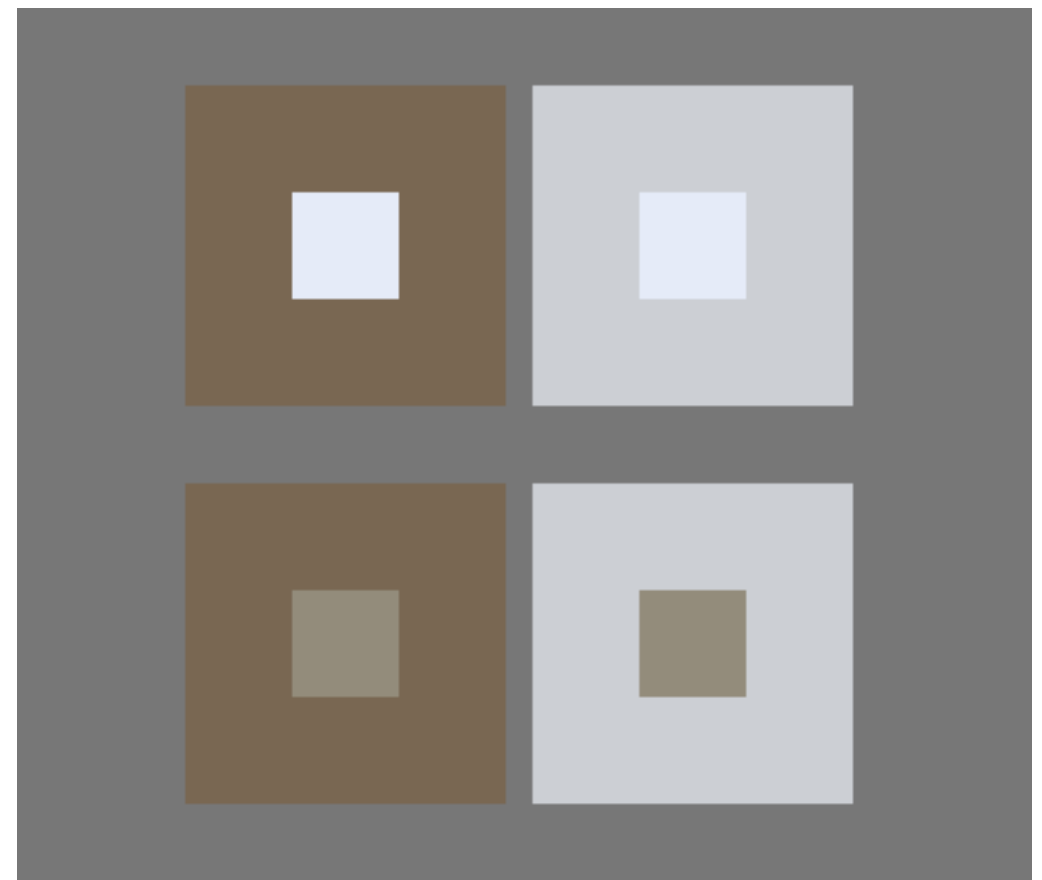
```
@article{gramazio-2017-ccd,
  author={Gramazio, Connor C. a},
  journal={IEEE Transactions on Visualization and Computer Graphics},
  title={Colorgorical: creating color palettes},
  year={2017}
```

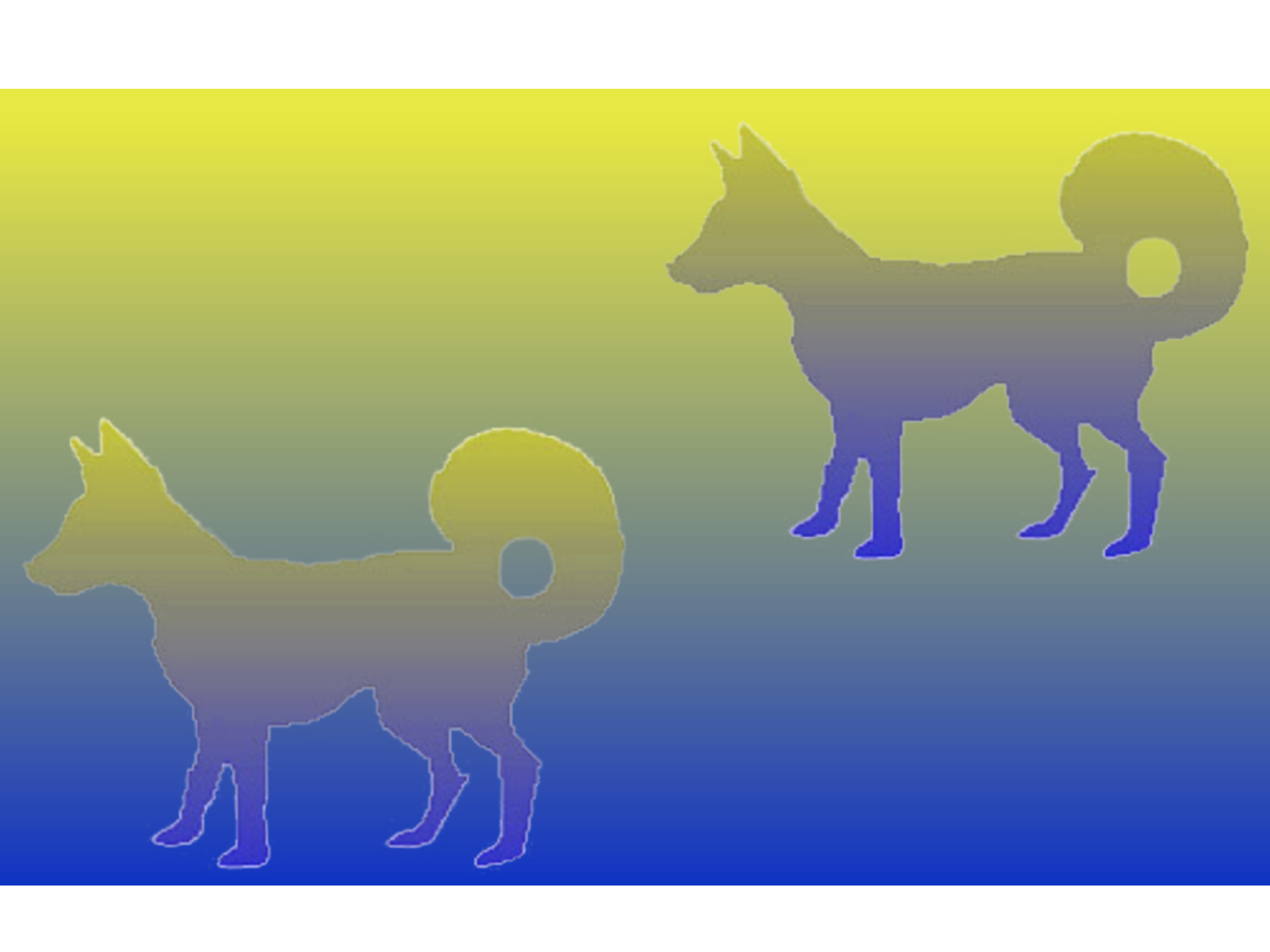
# COLORGORICAL

# SIMULTANEOUS CONTRAST



# SIMULTANEOUS CONTRAST

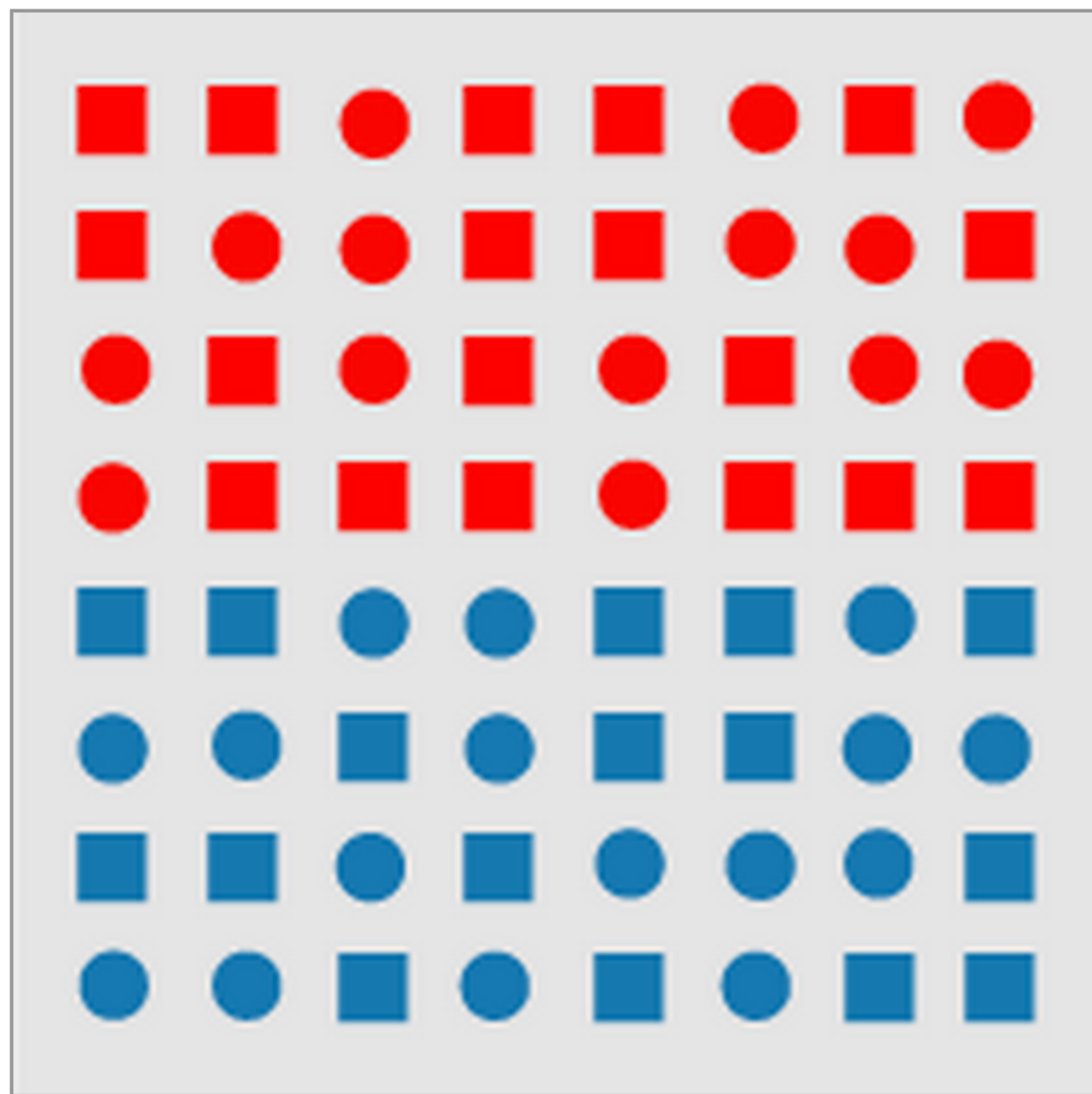




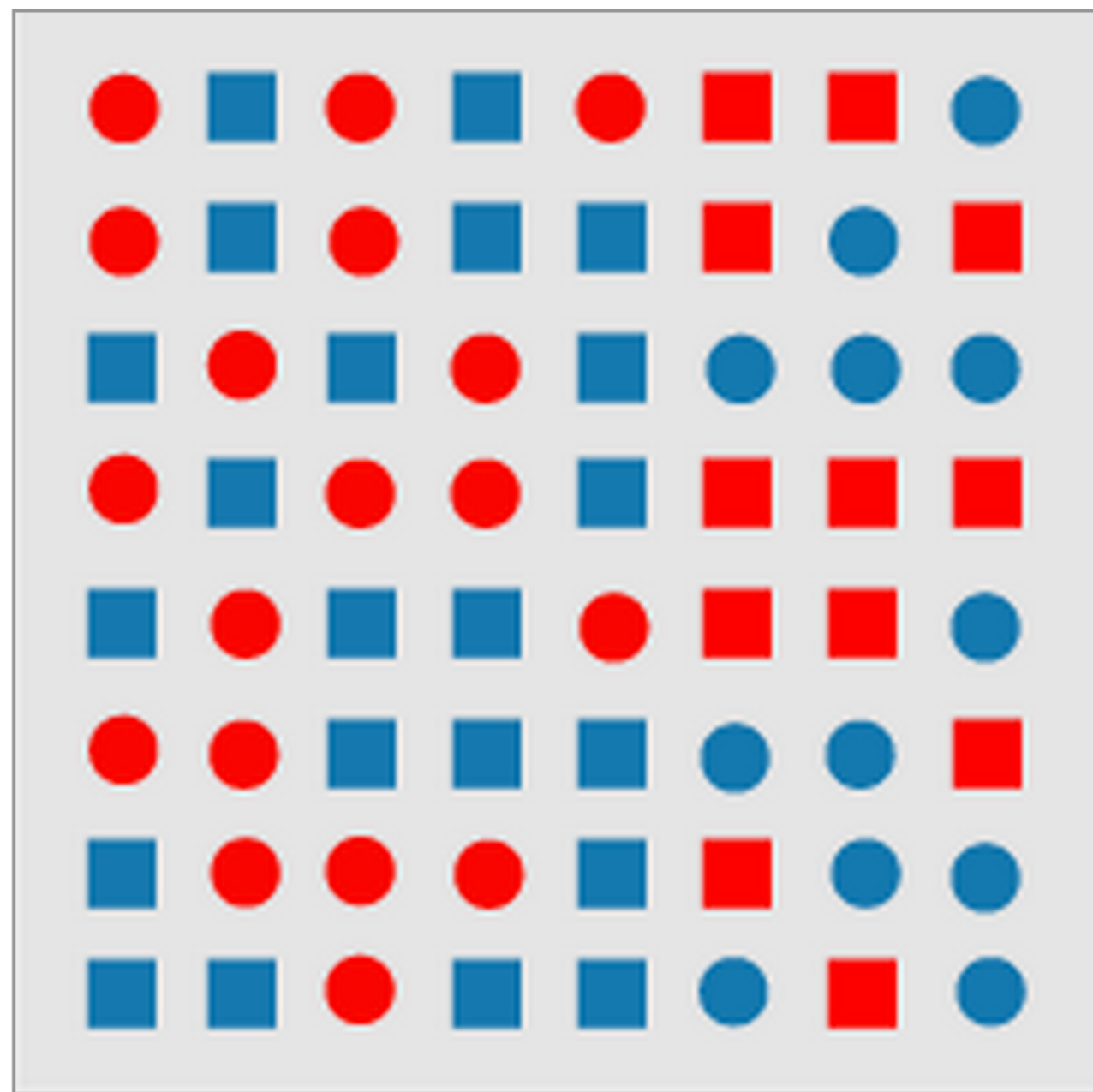


PREATTENTIVENESS,

OR “VISUAL POP-OUT”

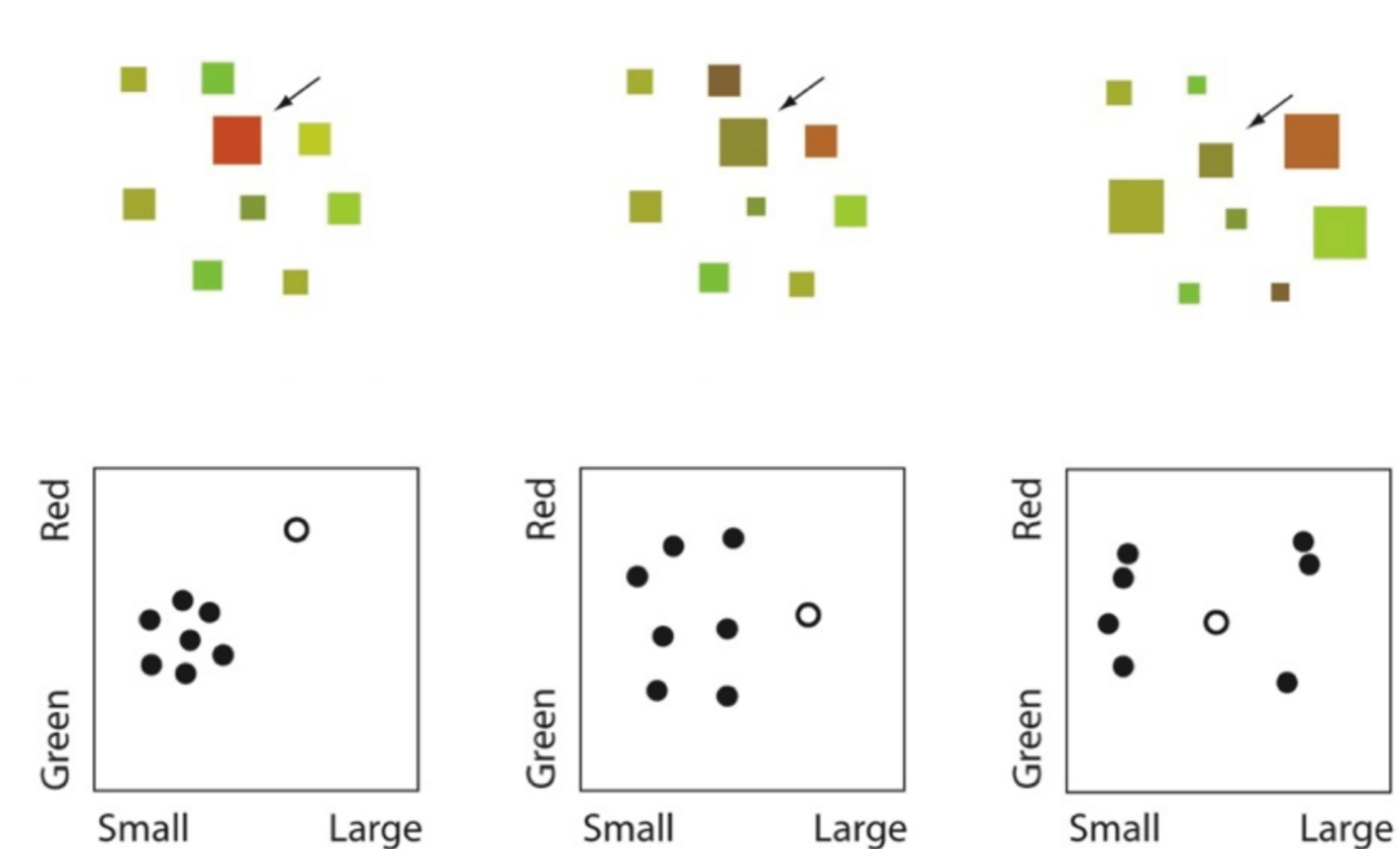


(a)



(b)

# Mixing is not always pre-attentive





Preattentiveness, **only**  
**one-channel-at-a-**  
**time.**

## ➔ Position

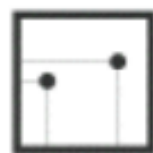
➔ Horizontal



➔ Vertical



➔ Both



## ➔ Color



## ➔ Shape



## ➔ Tilt



## ➔ Size

➔ Length



➔ Area



➔ Volume



# Cleveland/McGill perception papers

## Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods

WILLIAM S. CLEVELAND and ROBERT MCGILL\*

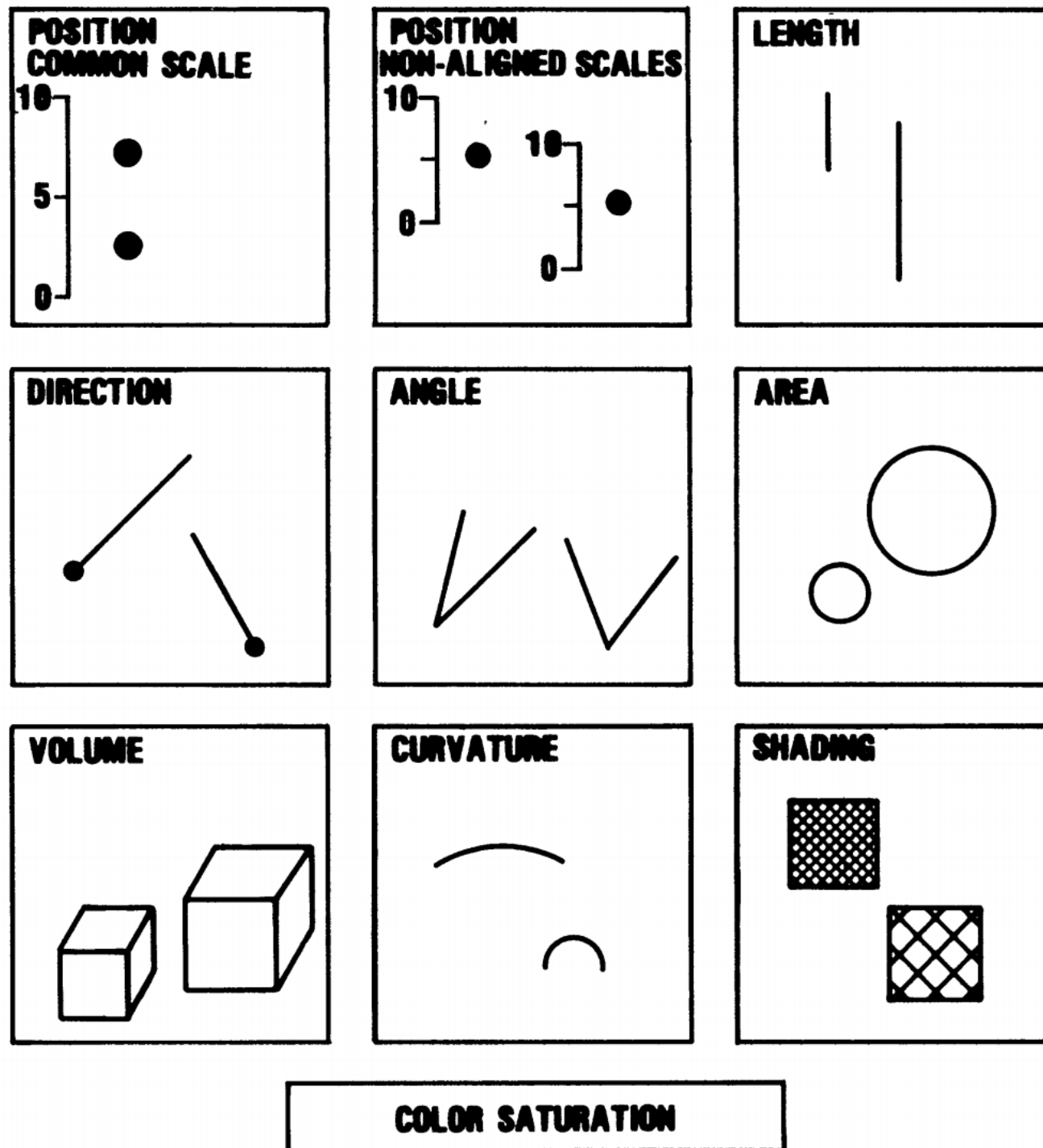
---

The subject of graphical methods for data analysis and for data presentation needs a scientific foundation. In this article we take a few steps in the direction of establishing such a foundation. Our approach is based on *graphical perception*—the visual decoding of information encoded on graphs—and it includes both theory and experimentation to test the theory. The theory deals with a small but important piece of the whole process of graphical perception. The first part is an identification of a set of *elementary perceptual tasks* that are carried out when people extract quantitative information from graphs. The second part is an ordering of the tasks on the basis of how accurately people perform them. Elements of the theory are tested by experimentation in which subjects record their judgments of the quantitative information on graphs. The experiments validate these elements but also suggest that the set of elementary tasks should be expanded. The theory provides a guideline for graph construction: Graphs should employ elementary tasks as high in the ordering as possible. This principle is applied to a variety of graphs, including bar charts, divided bar charts,

largely unscientific. This is why Cox (1978) argued, “There is a major need for a theory of graphical methods” (p. 5), and why Kruskal (1975) stated “in choosing, constructing, and comparing graphical methods we have little to go on but intuition, rule of thumb, and a kind of master-to-apprentice passing along of information. . . . there is neither theory nor systematic body of experiment as a guide” (p. 28–29).

There is, of course, much good common sense about how to make a graph. There are many treatises on graph construction (e.g., Schmid and Schmid 1979), bad practice has been uncovered (e.g., Tufte 1983), graphic designers certainly have shown us how to make a graph appealing to the eye (e.g., Marcus et al. 1980), statisticians have thought intensely about graphical methods for data analysis (e.g., Tukey 1977; Chambers et al. 1983), and cartographers have devoted great energy to the construction of statistical maps (Bertin 1973; Robinson, Sale, and Morrison 1978). The ANSI manual on time series charts (American National Standards Institute 1979) provides guidelines for making graphs, but the manual ad-

# Cleveland/McGill perception papers



## Better to worse:

1. Position along a common scale
2. Positions along nonaligned scales
3. Length, direction, angle
4. Area
5. Volume, curvature
6. Shading, color saturation

Figure 1. Elementary perceptual tasks.

Pie Chart Bad,  
Scatterplot Good

# Cleveland/McGill perception papers

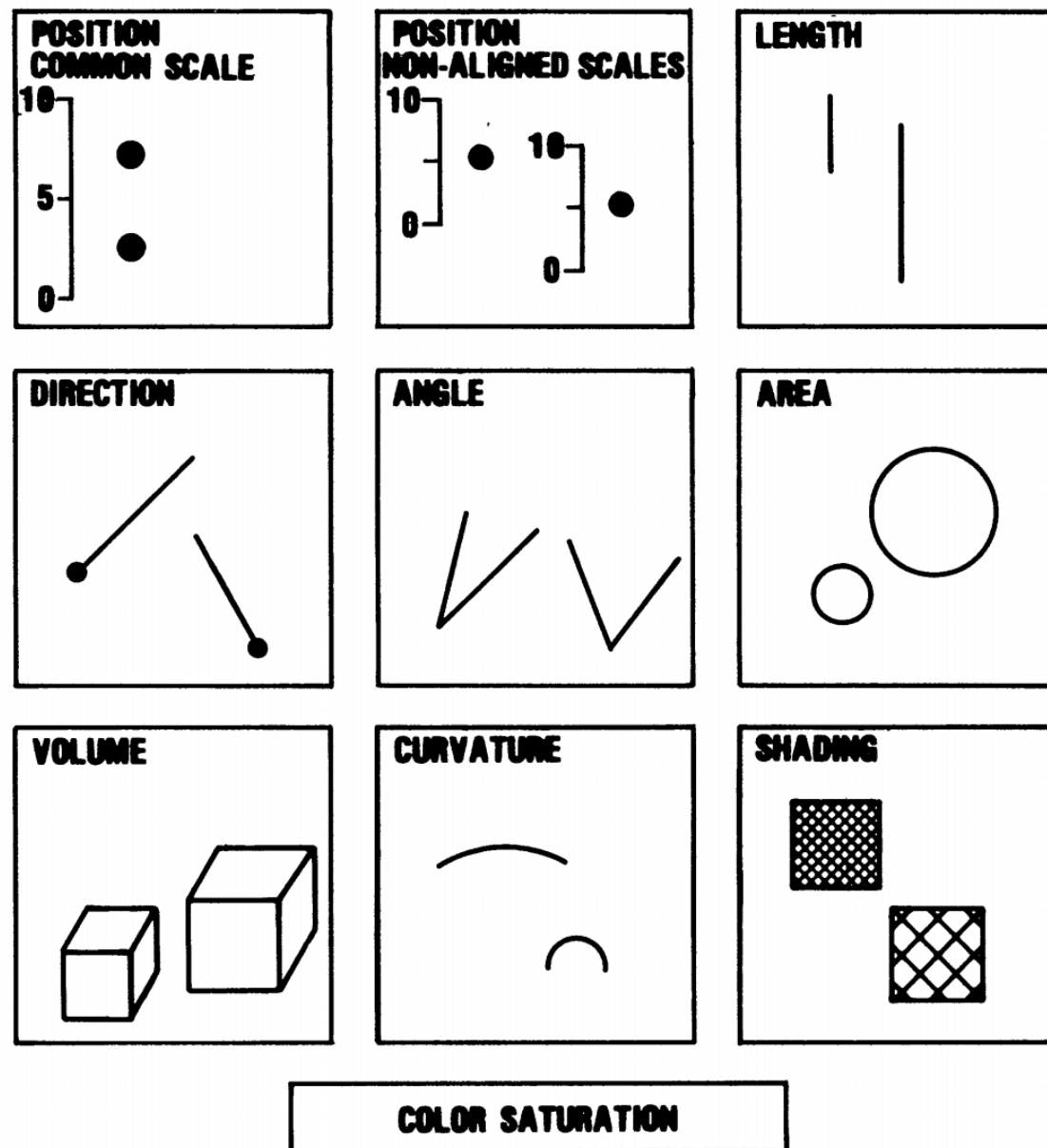


Figure 1. Elementary perceptual tasks.

- Notice the “elementary perceptual tasks”
- What about higher-level tasks?

# Integral vs. Separable Channels

Separable

Integral



color x location

color x motion

color x shape

size x orientation

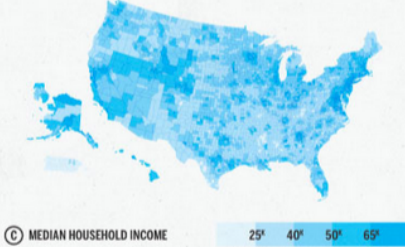
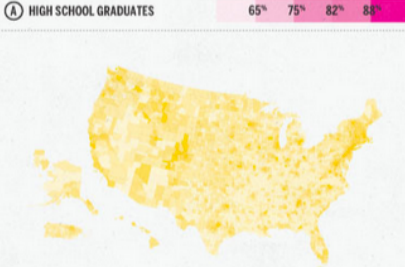
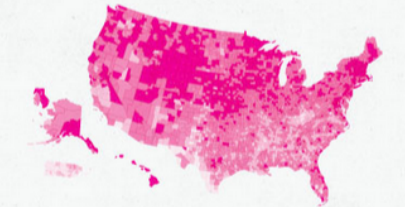
x-size x y-size

r-g x y-b



READING,  
WRITING,  
AND EARNING MONEY

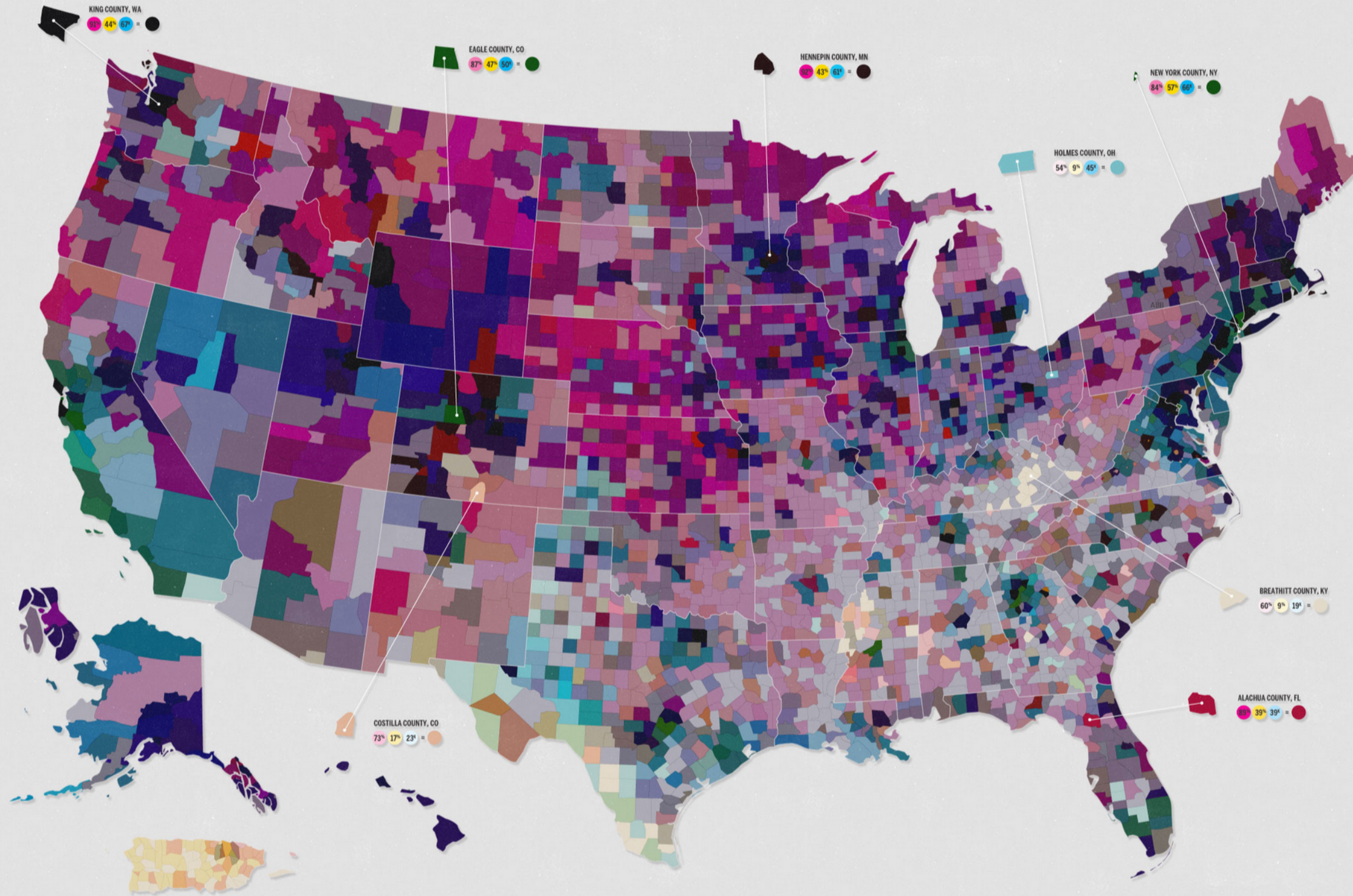
The latest data from the U.S. Census's American Community Survey paints a fascinating picture of the United States at the county level. We've looked at the educational achievement and the median income of the entire nation, to see where people are going to school, where they're earning money, and if there is any correlation.



The map at right is a product of overlaying the three sets of data. The variation in hue and value has been produced from the data shown above. In general, darker counties represent a more educated, better paid population while lighter areas represent communities with fewer graduates and lower incomes.



A collaboration between GOOD and Gregory Hubacek  
SOURCE: US Census



Trivariate (!) Color Map (terrible, terrible idea)

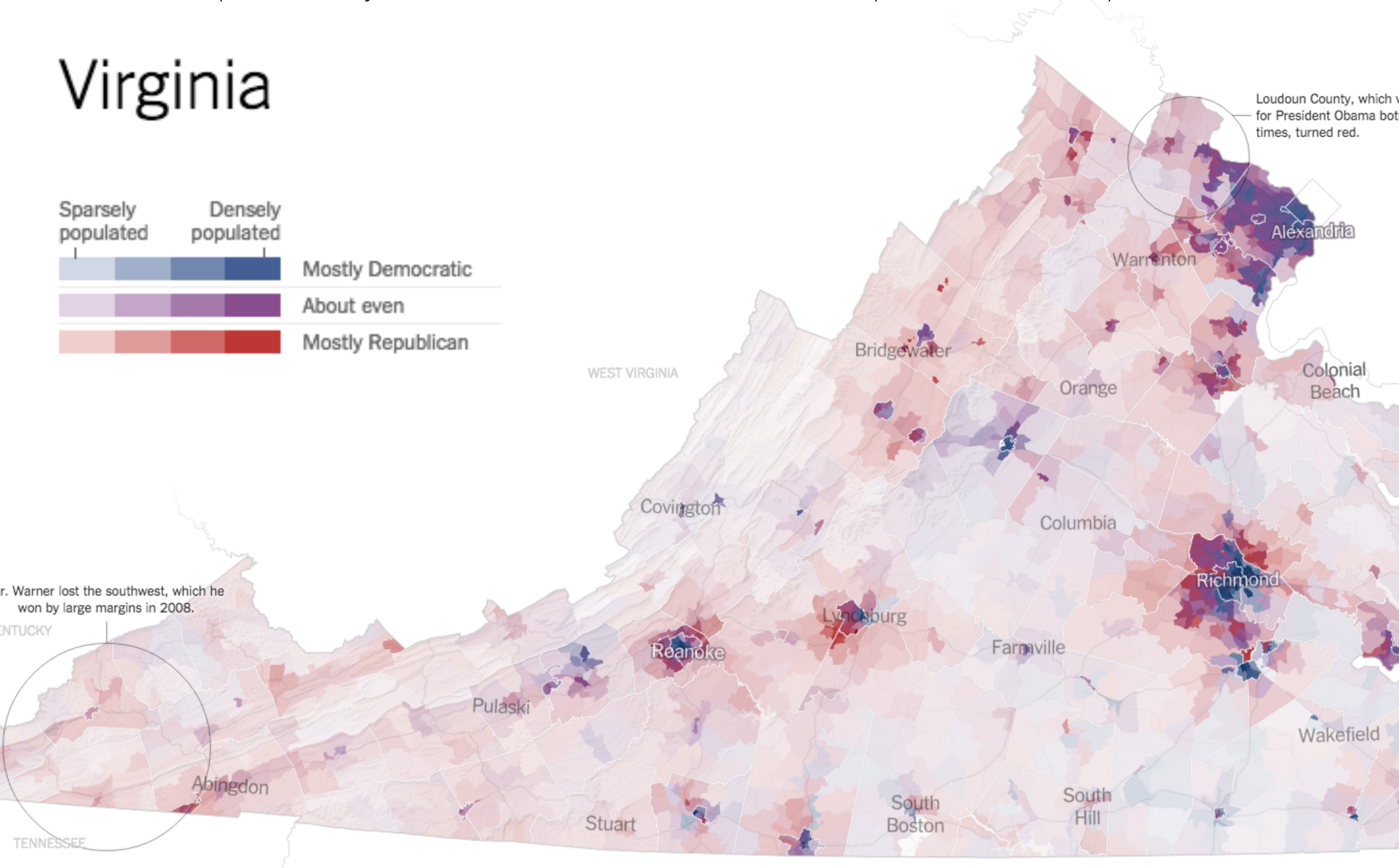
<http://magazine.good.is/infographics/america-s-richest-counties-and-best-educated-counties#open>



# The best bivariate colormap I know

<http://www.nytimes.com/interactive/2014/11/04/upshot/senate-maps.html>

## Virginia

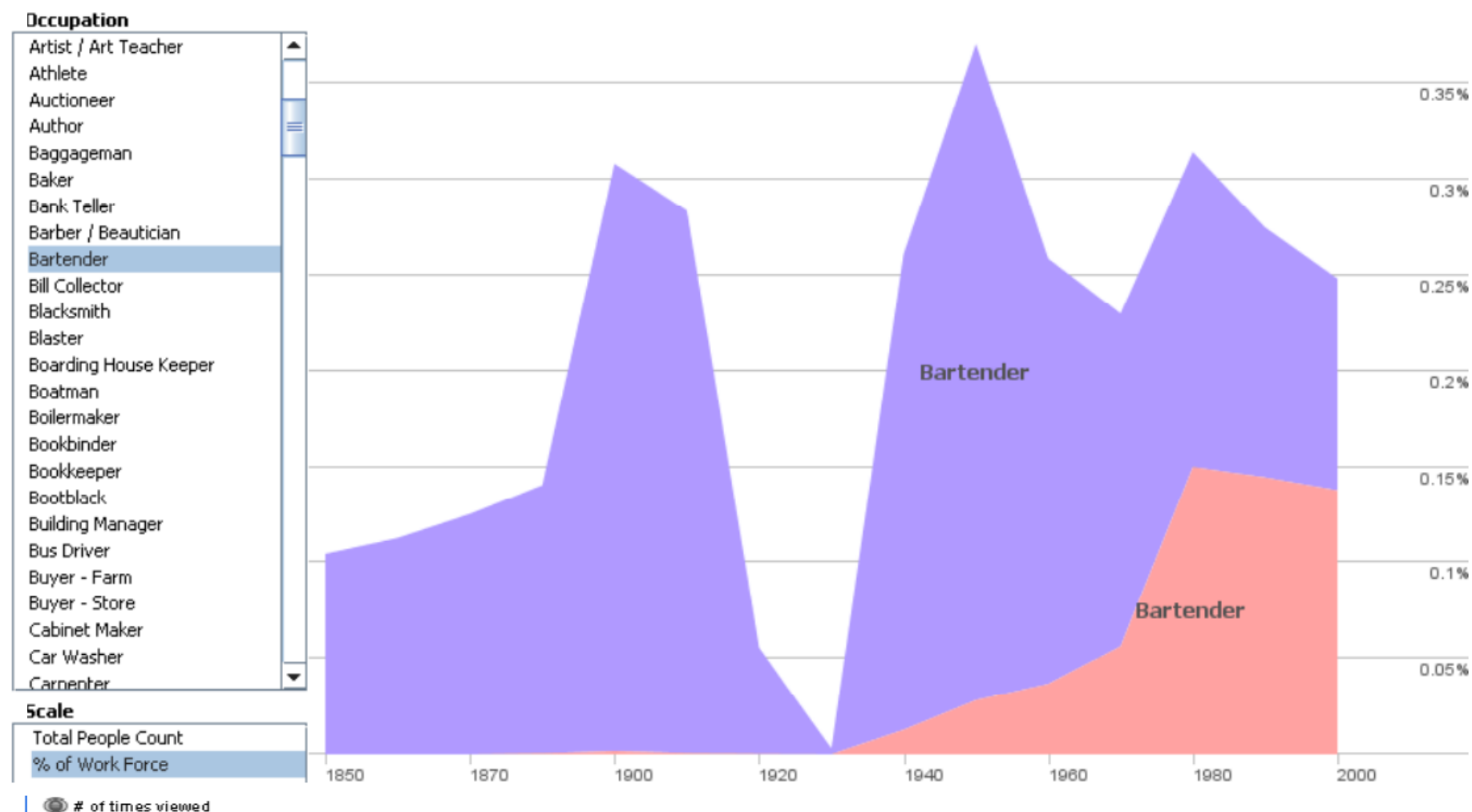


# Bivariate Color Maps are Possible, but Hard

pay attention to the **behavior of the variables** you're mapping from, and the **behavior of the channels** you're mapping to.

# Interaction

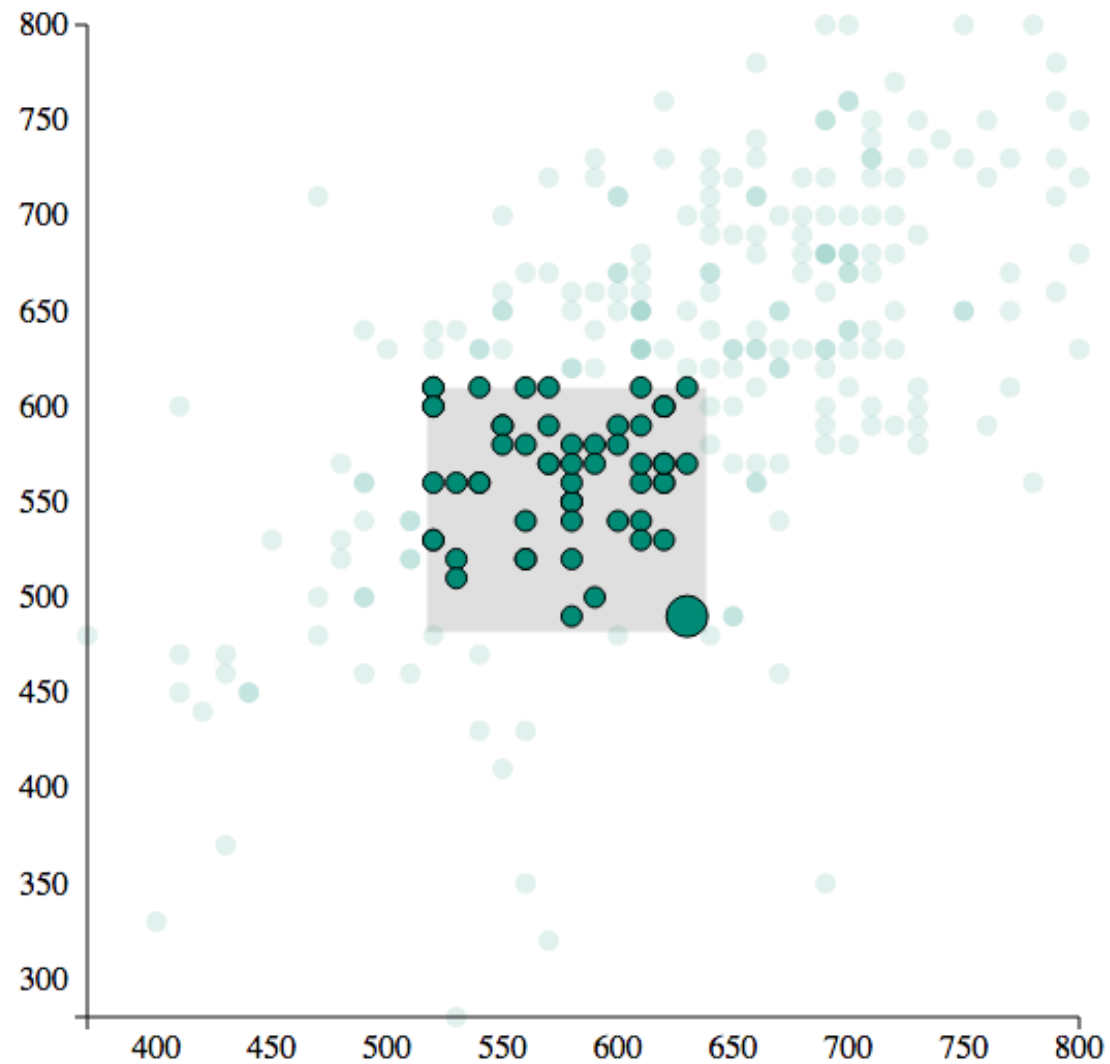
- Interpret the state of elements in the UI as a **clause** in a **query**. As UI changes, update data



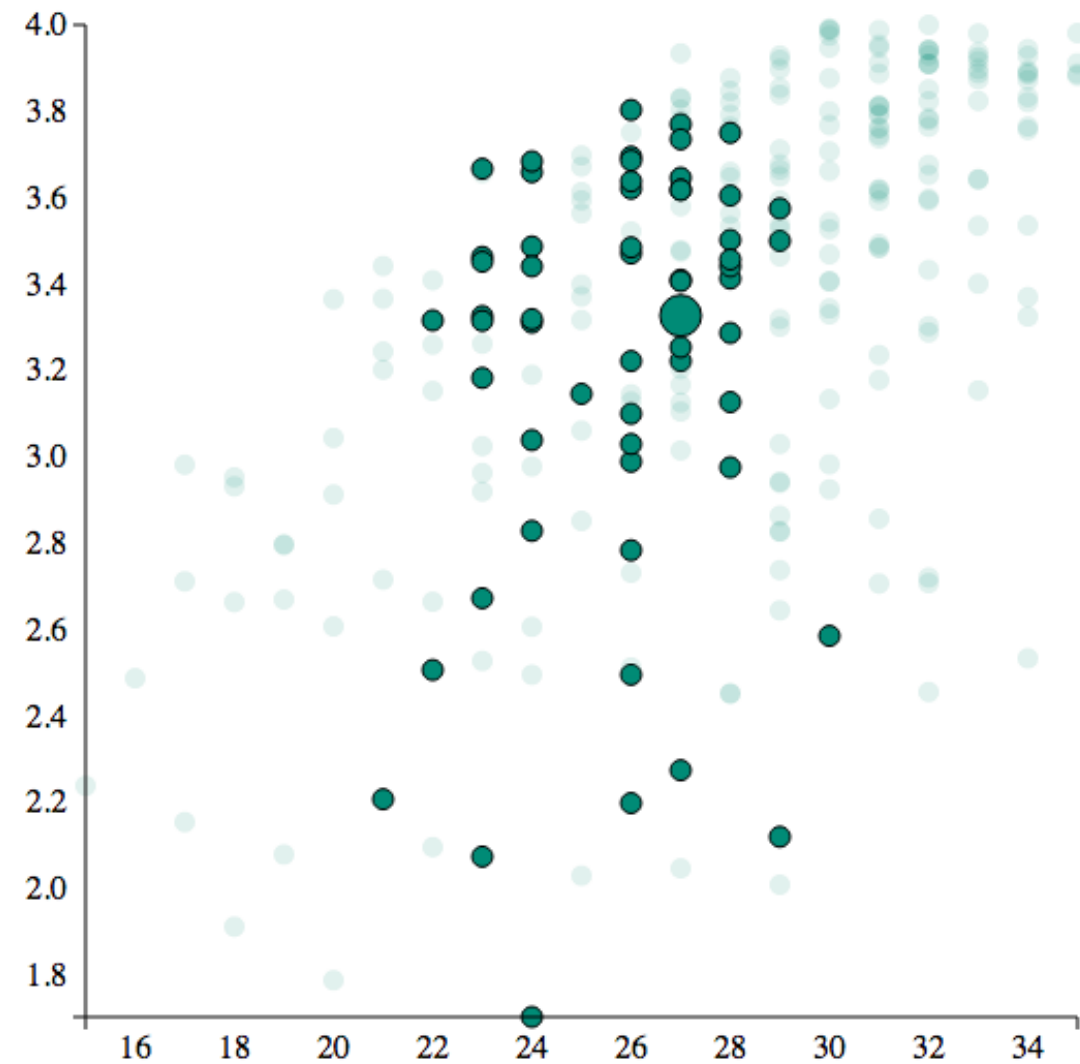
Willett et al., TVCG 2007 (\*)

# Linked Brushing

**SATM x SATV**



**ACT x GPA**



# Shneiderman's “Visual information seeking mantra”

**Overview first,  
zoom and filter,  
then details-on-demand**

# Techniques



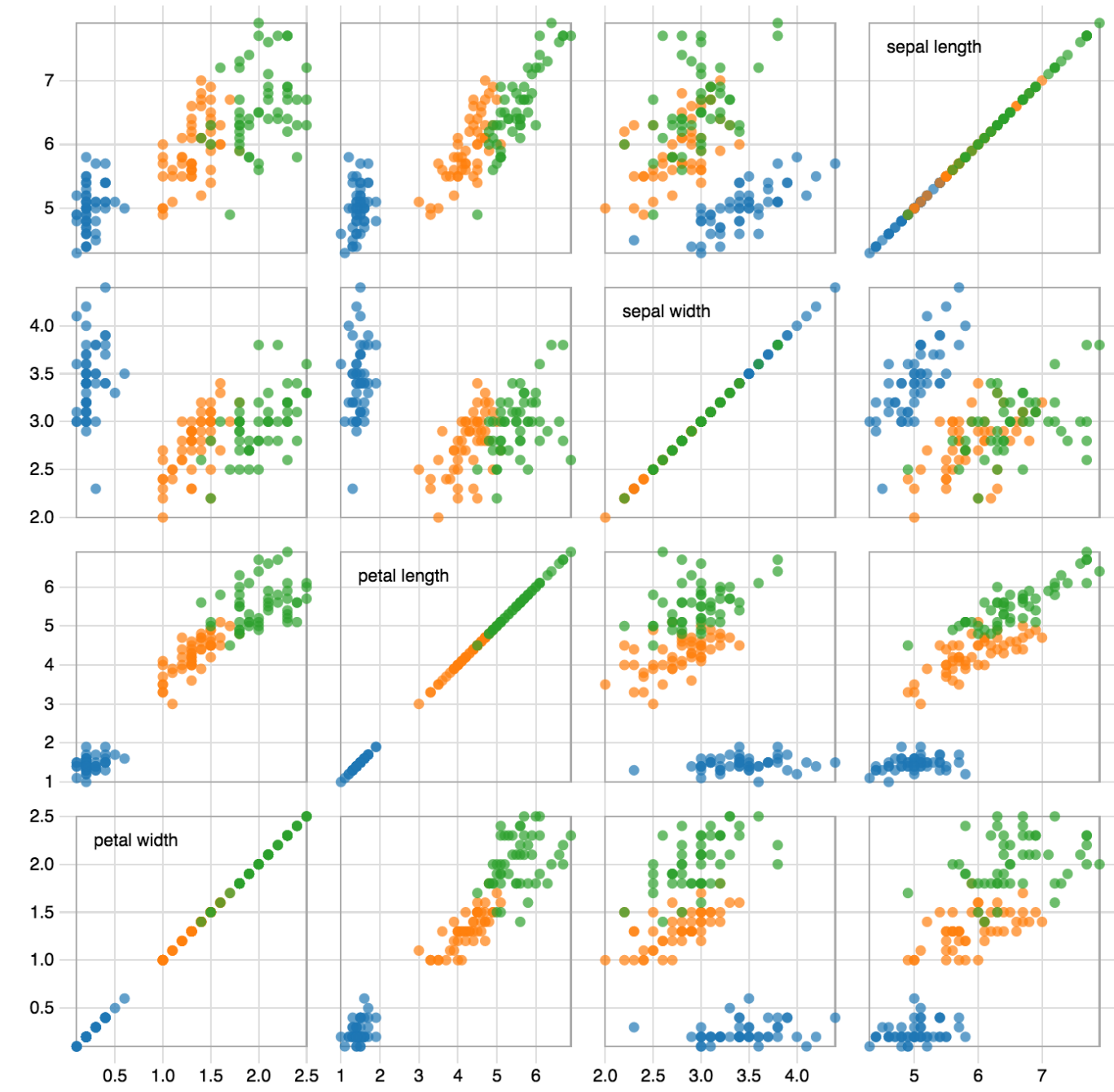
# Regular Scatterplots

- Every data point is a vector:

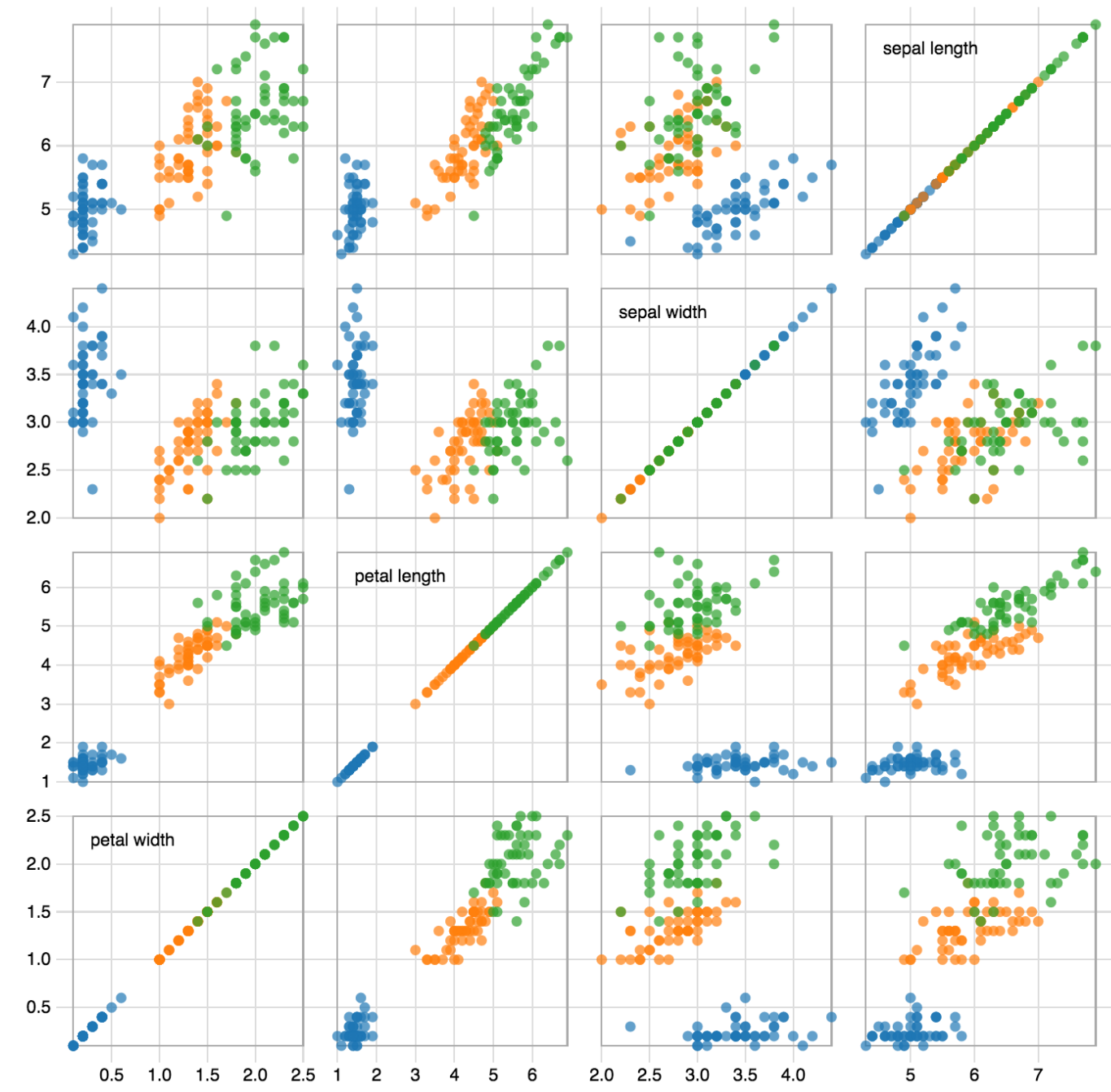
$$\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

- Every scatterplot is produced by a very simple matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

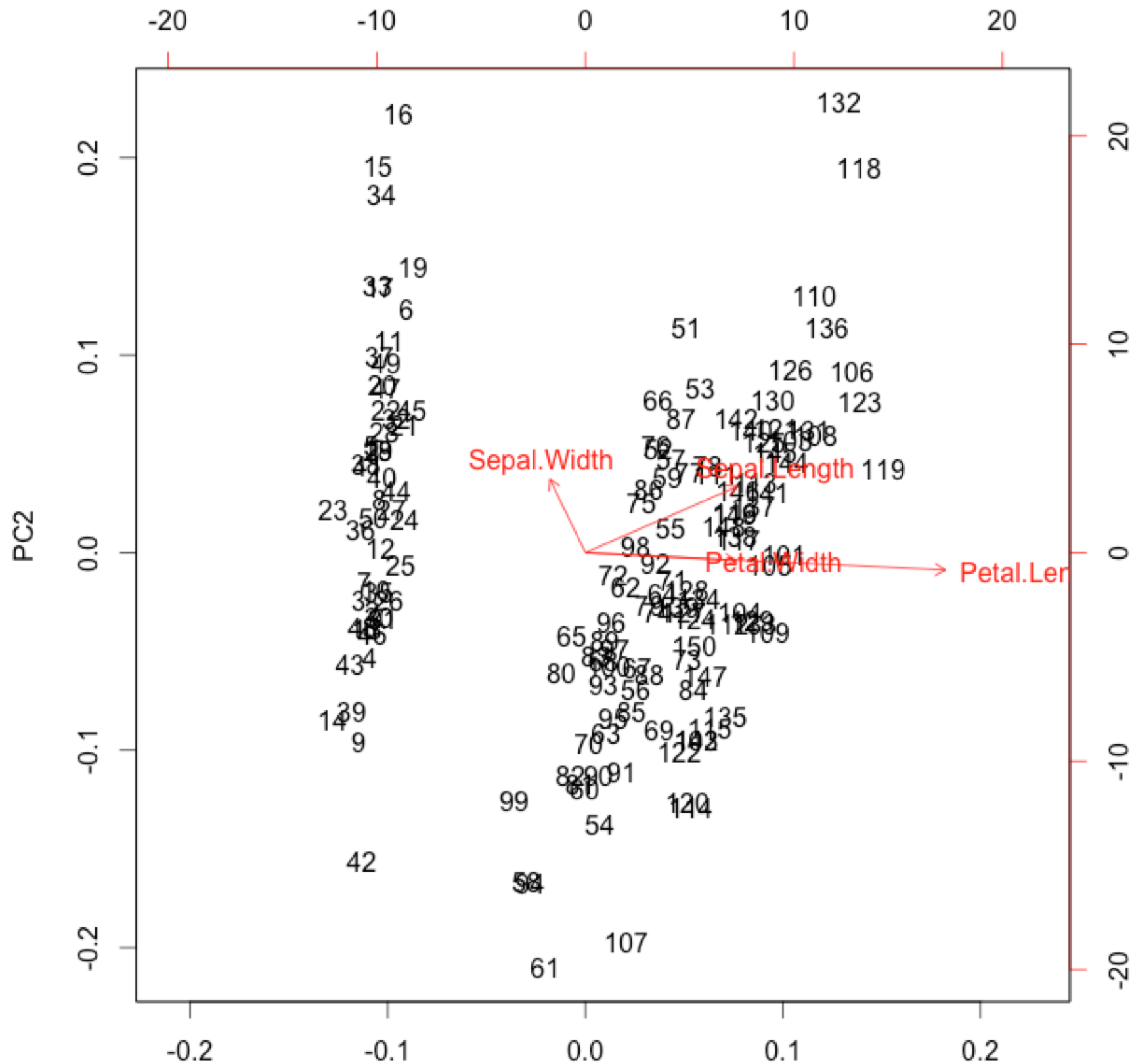


# What about other matrices?

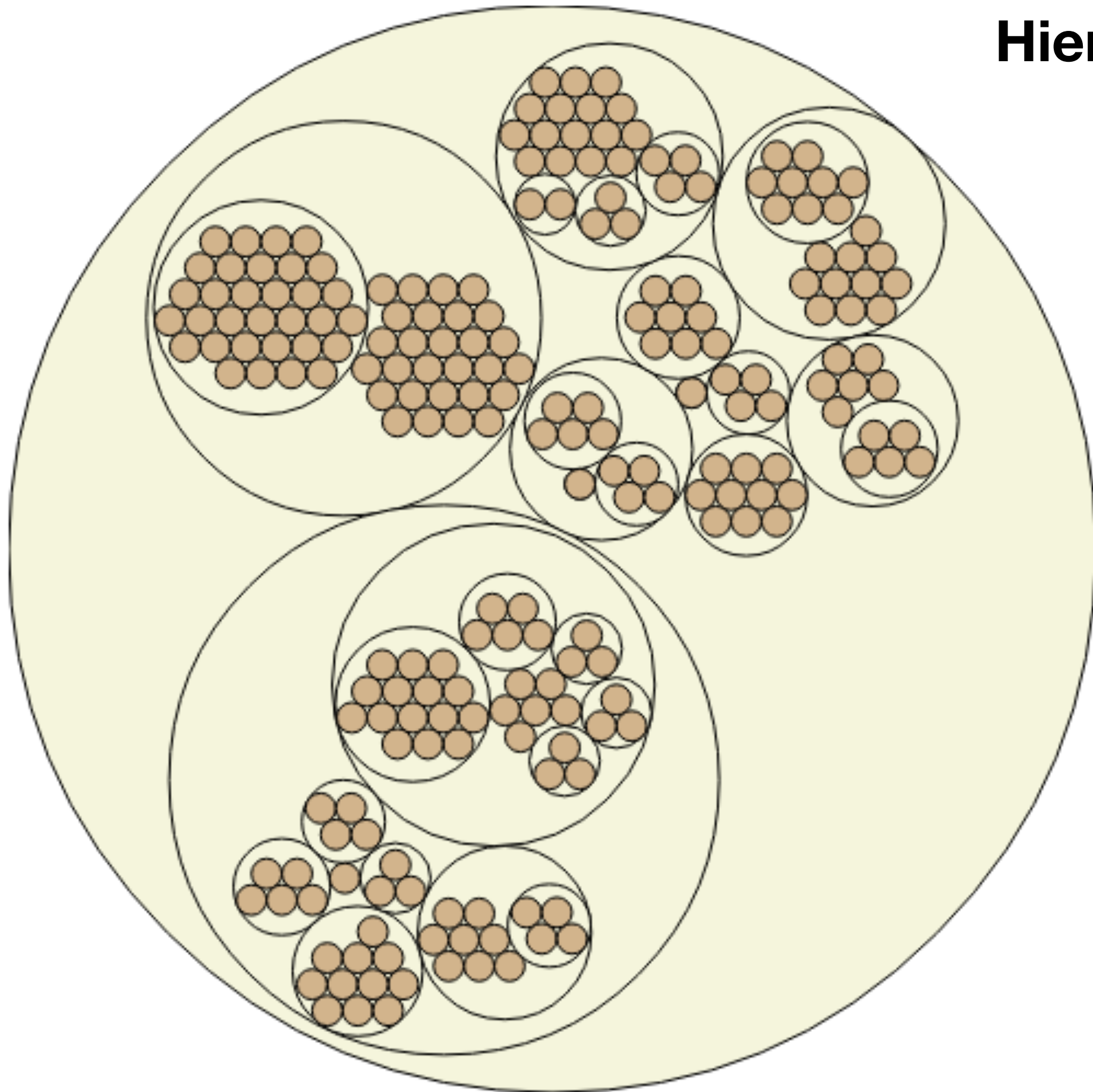


# Dimensionality Reduction

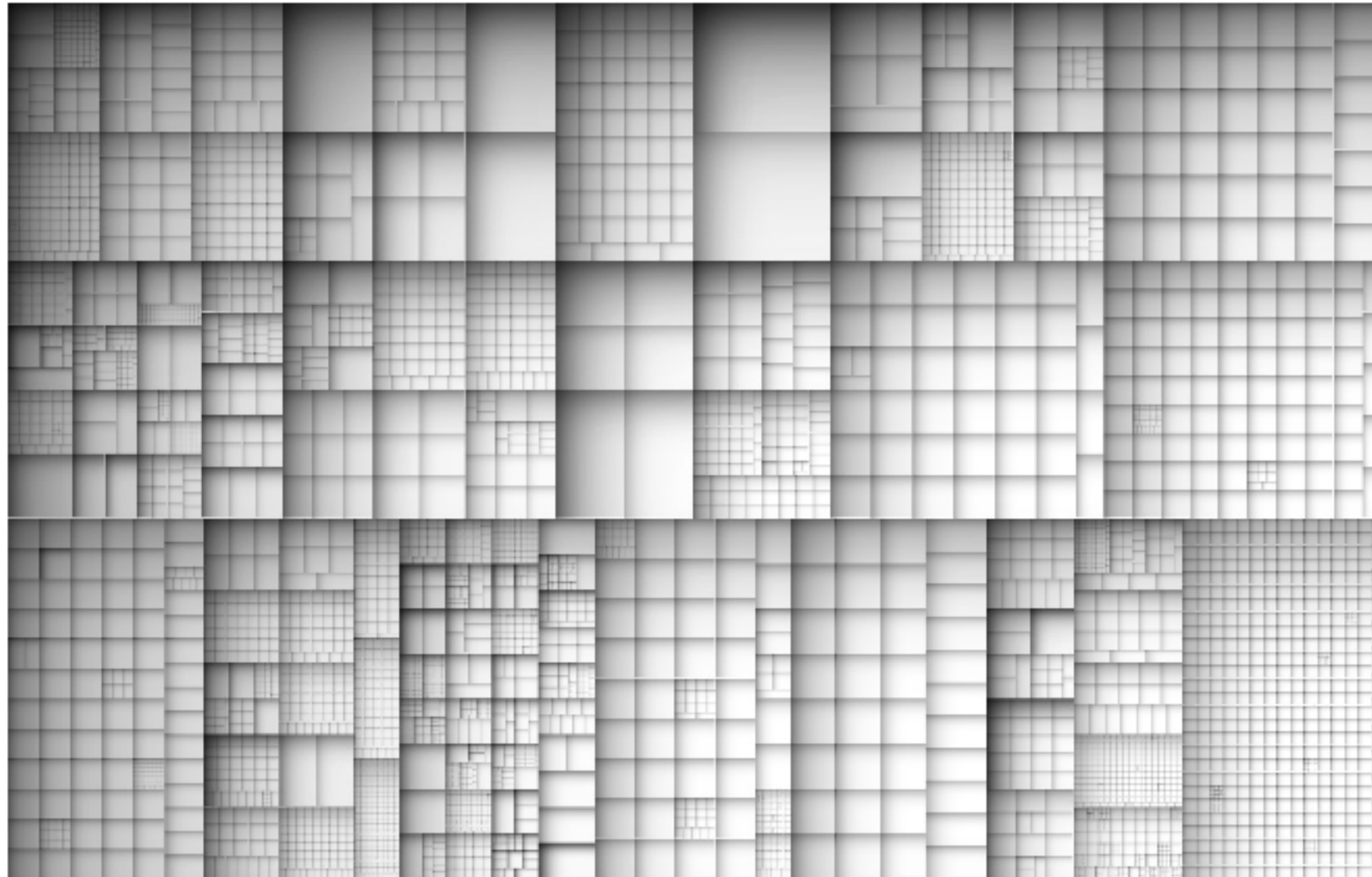
# Principal Component Analysis



# Hierarchies



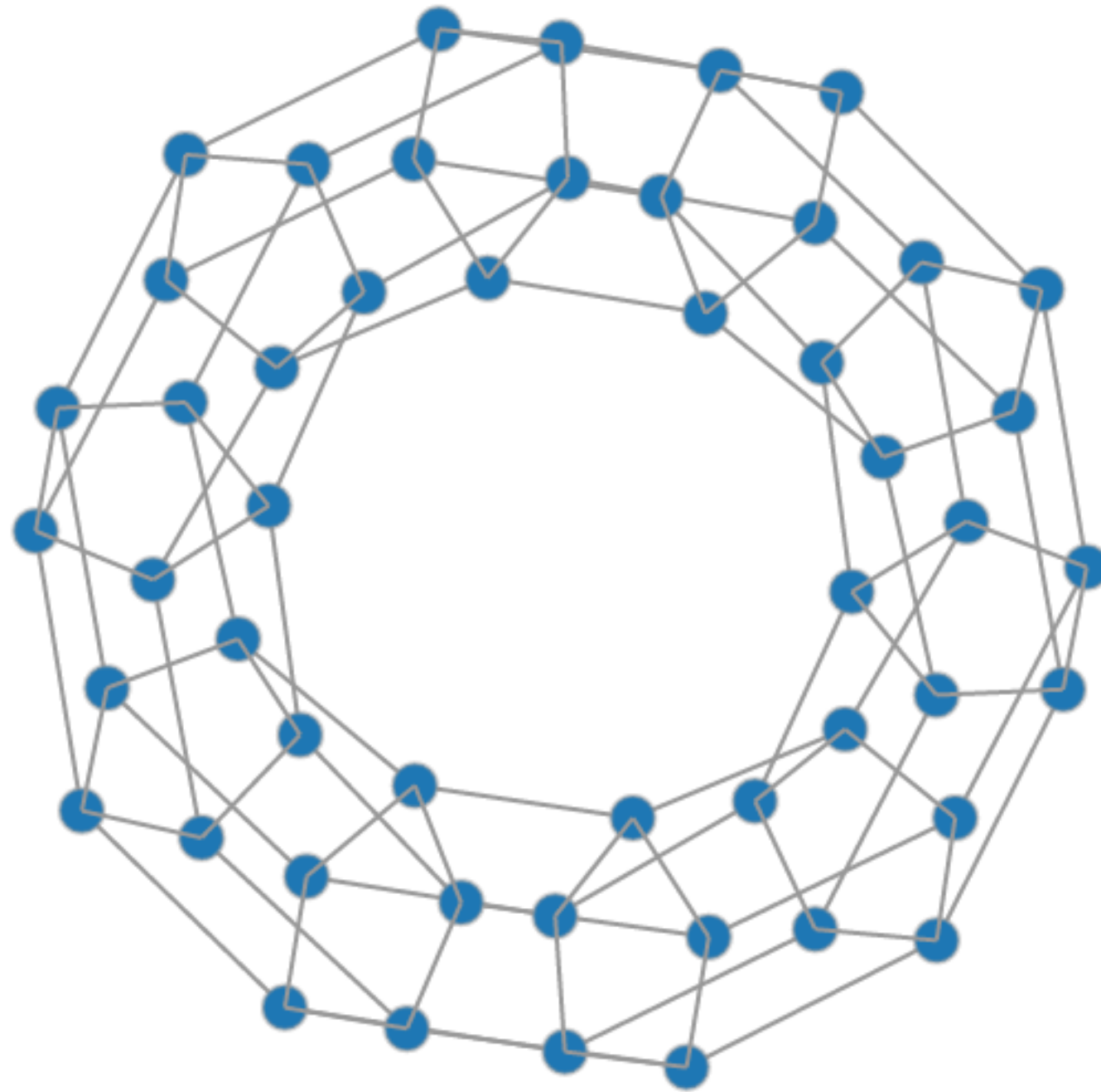
# Hierarchies



<http://www.cs.rug.nl/svcg/SoftVis/ViewFusion>

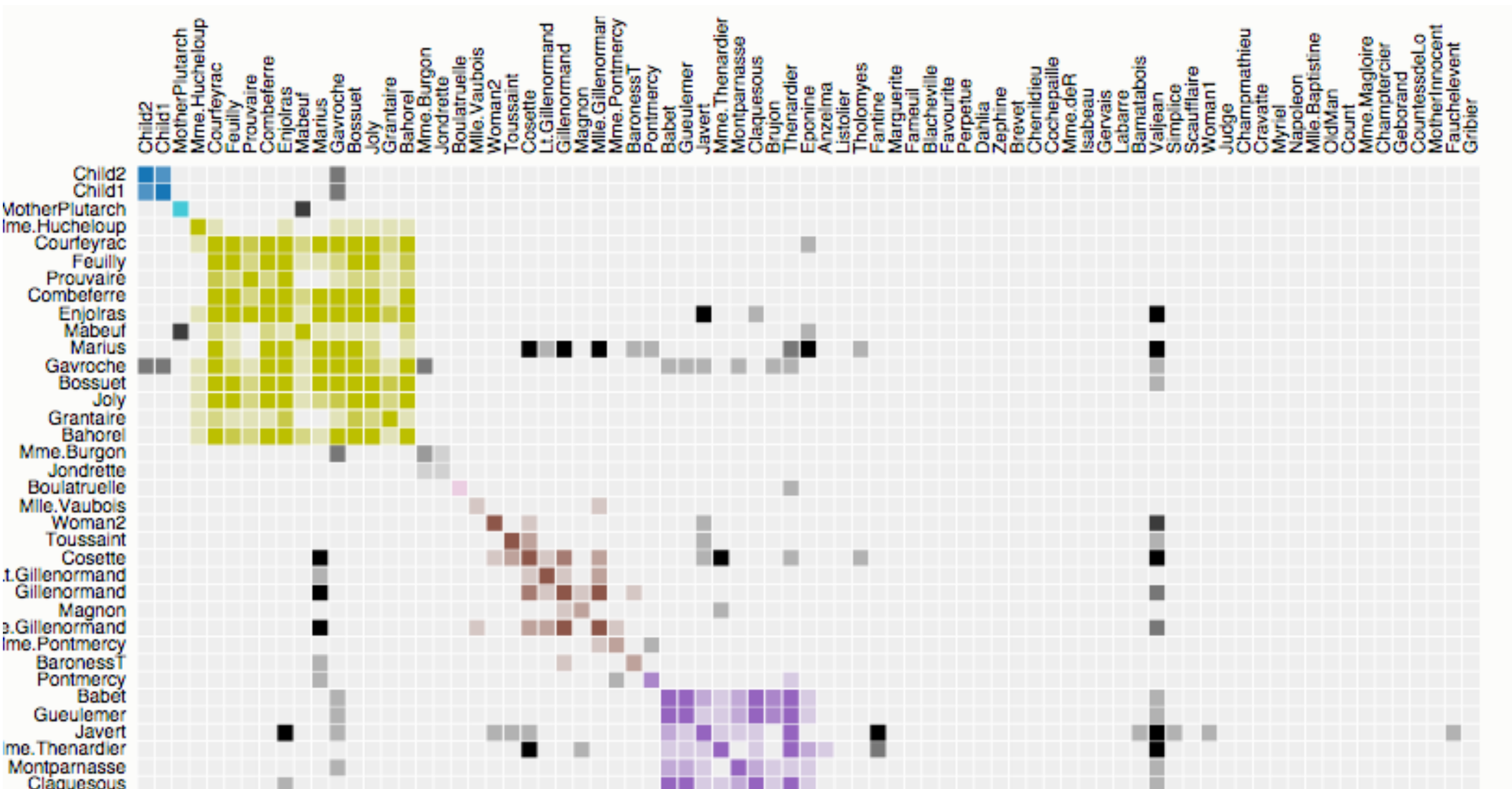


# Node-link diagrams

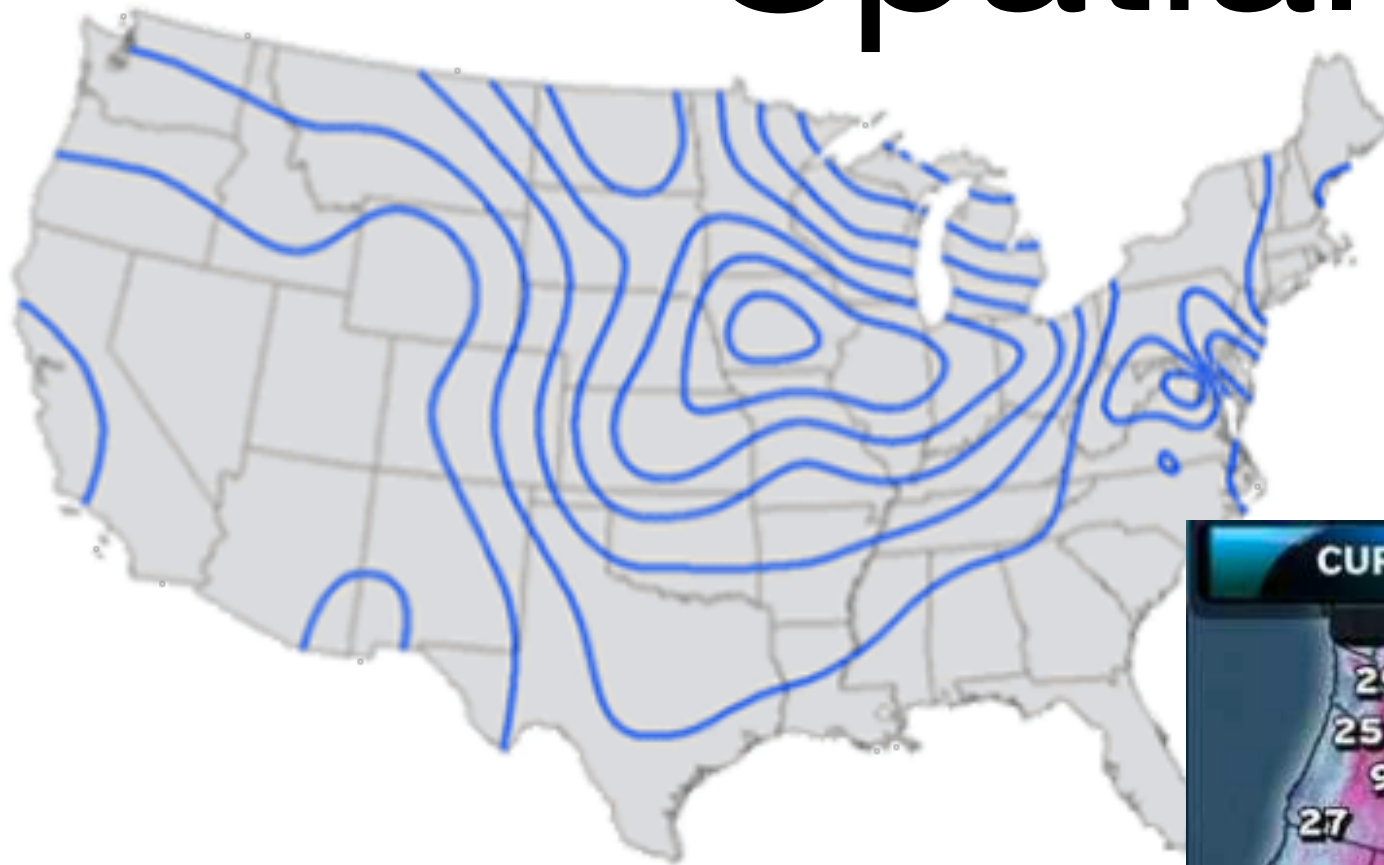


# Matrix Diagrams

<http://bost.ocks.org/mike/miserables/>



# Spatial Data

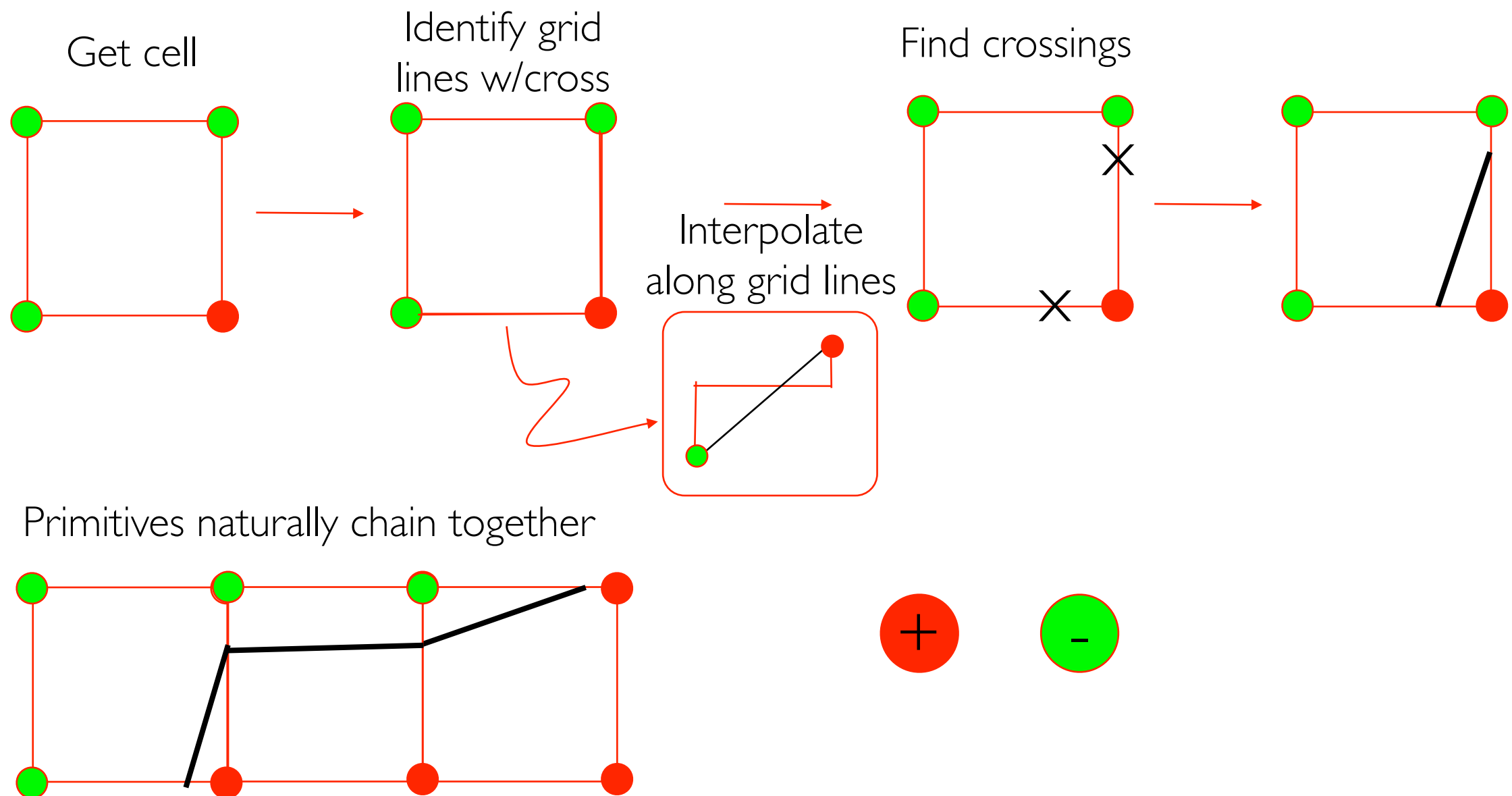


<http://ryanhill1.blogspot.com/2011/07/isoline-map.html>

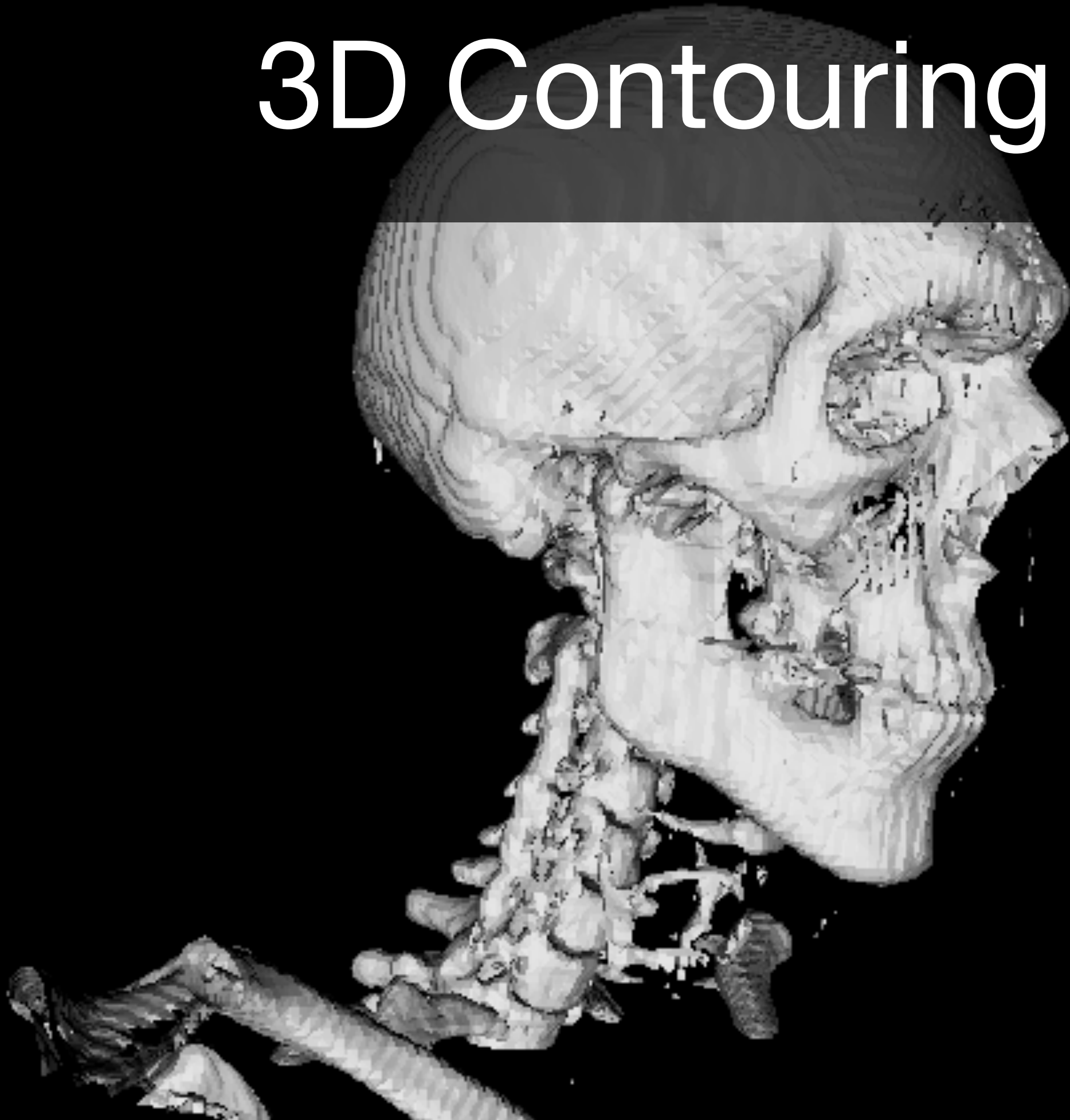


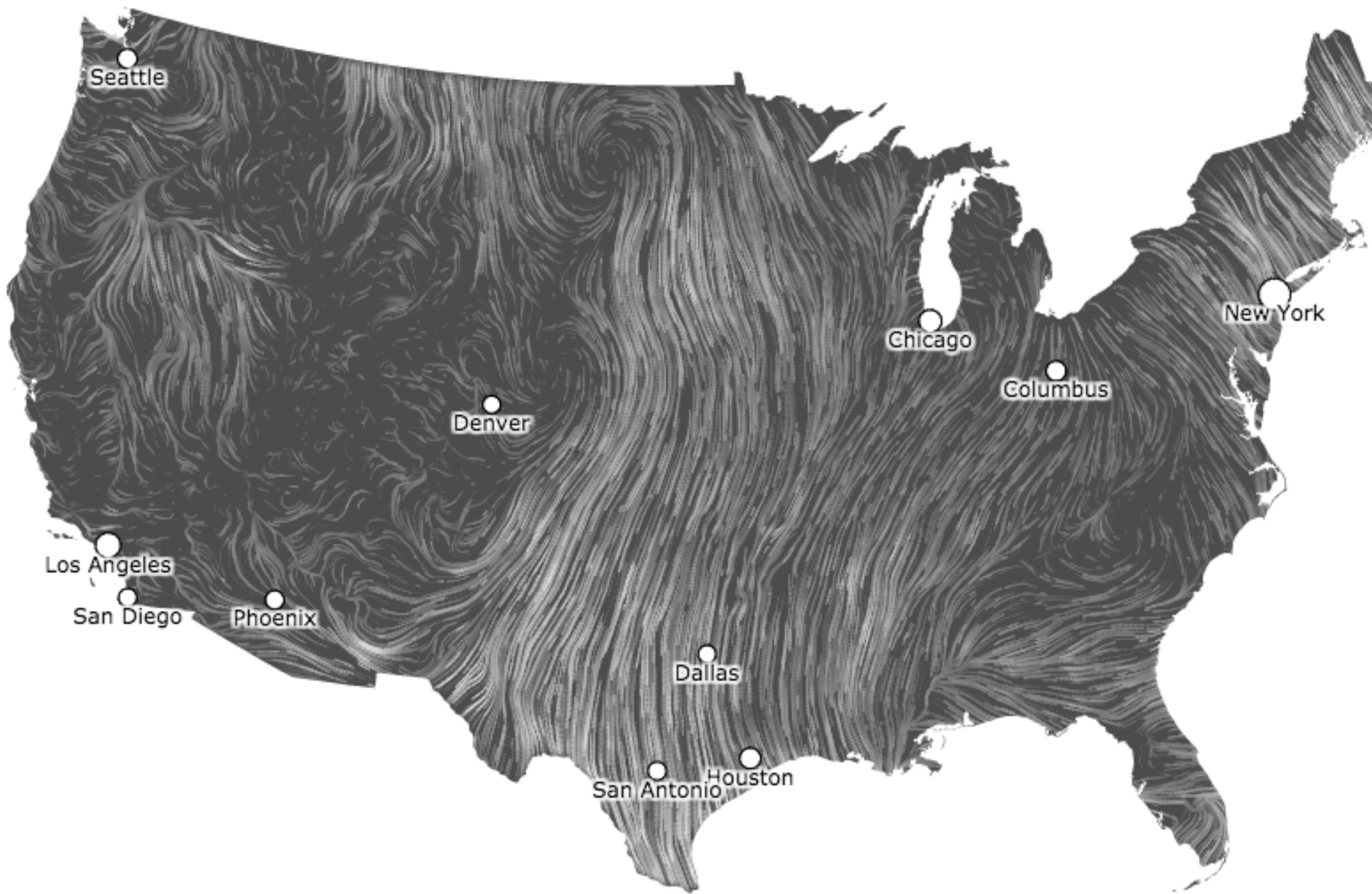
# Approach to Contouring in 2D

- Contour must cross every grid line connecting two grid points of opposite sign



# 3D Contouring





# Spatial Data: Vector Fields



# CS444: Data Visualization

- Now you know **why, how and how not to** create visualizations for your data!