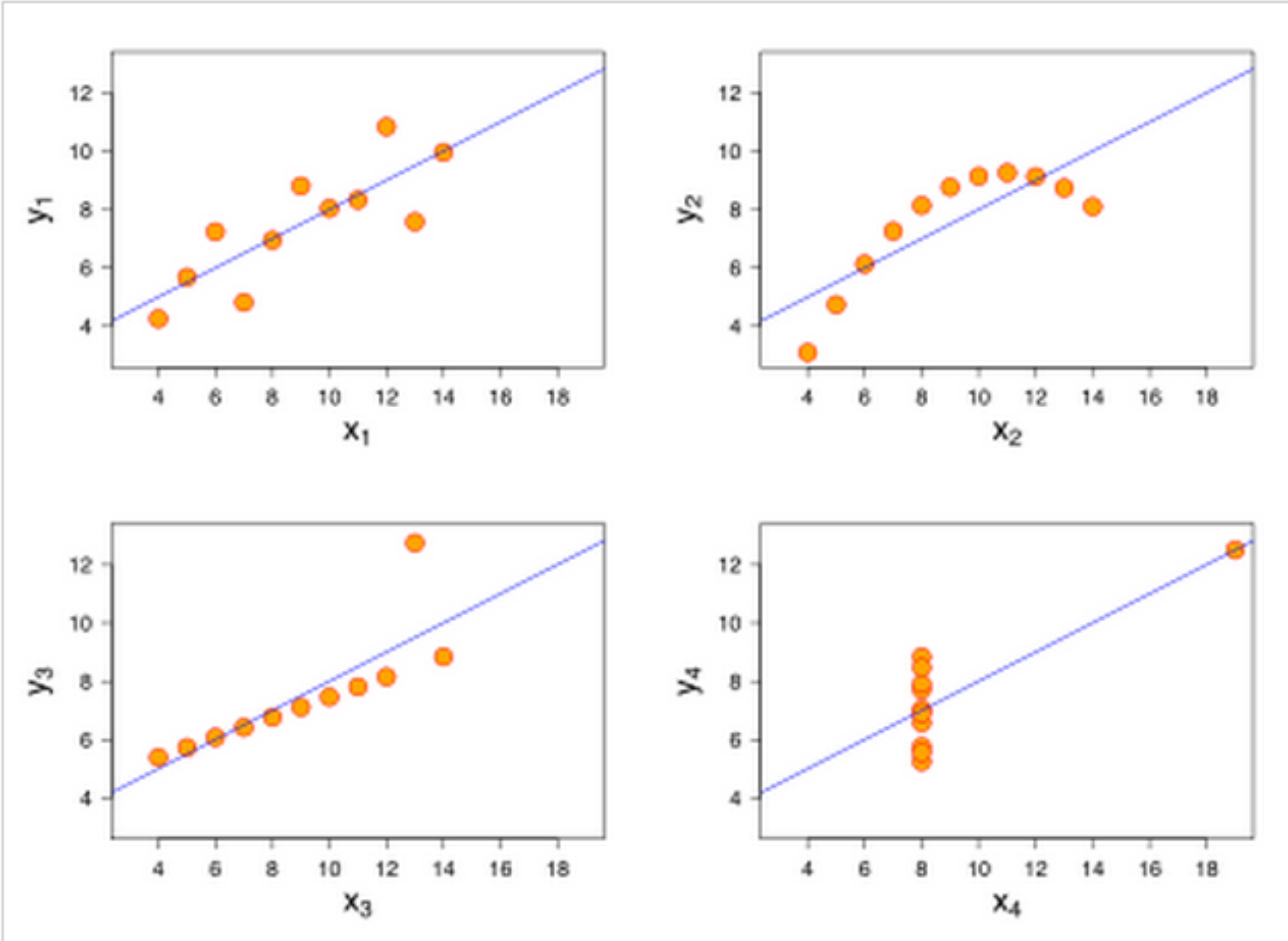
Retrospective, Wrap-Up, What's Next

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)

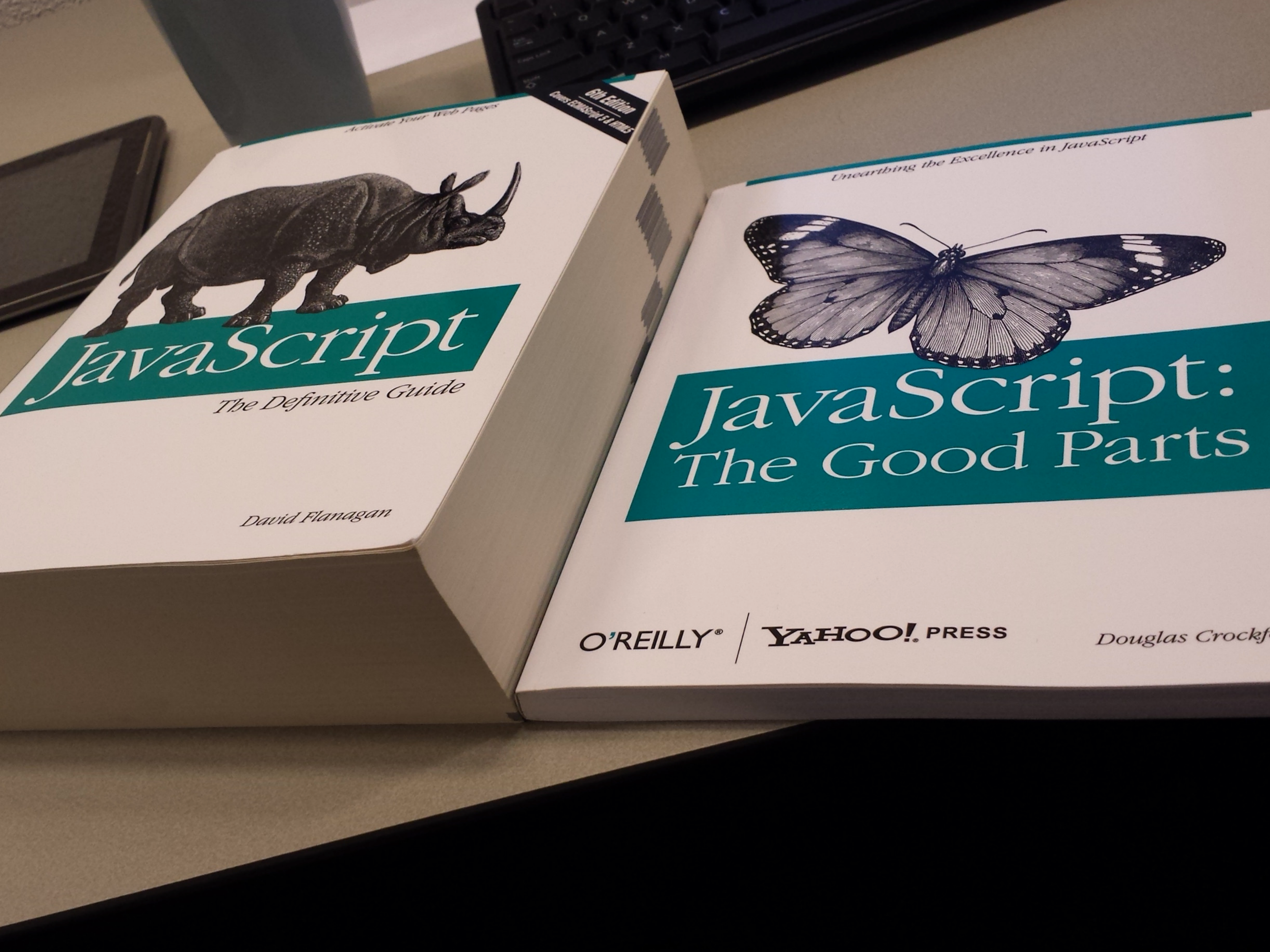
http://en.wikipedia.org/wiki/Anscombe%27s_quartet



12
10
8
6
4
y1
4
6
8
10
12
14
16
18
x1
y2
x2
y3
x3
y4
x4

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

Mechanics


Activate Your Web Pages
6th Edition
Covers ECMAScript 5 & HTML5
JavaScript
The Definitive Guide
David Flanagan
Unearthing the Excellence in JavaScript
JavaScript:
The Good Parts
O'REILLY
YAHOO! PRESS
Douglas

Overview Examples Documentation Source

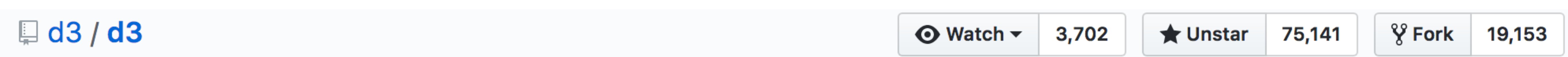
Data-Driven Documents

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



- 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

```
function WebGLCircleRenderer(glowContext, circleCount, colors, radii, alpha) {
    this.context = glowContext;
    this.count = circleCount;

    var vertShader = [
        "uniform mat4 u_matrix;",
        "attribute float a_x;",
        "attribute float a_y;",
        "attribute float a_radius;",
        "attribute vec3 a_color;",
        "varying vec3 v_color;",

        "void main() {",
        "    gl_PointSize = a_radius;",
        "    gl_Position = u_matrix * vec4(a_x, a_y, 1.0, 1.0);",
        "    v_color = a_color;",
        "}"
    ].join("\n");

    var fragShader = [
        "precision mediump float;",
        "uniform float u_alpha;",
        "varying vec3 v_color;",

        "void main() {",
        "    float centerDist = length(gl_PointCoord - 0.5);",
        "    float radius = 0.5;",
        // works for overlapping circles if blending is enabled
        "    gl_FragColor = vec4(v_color, u_alpha * step(centerDist, radius));",
        "}"
    ].join("\n");

    var circleShaderInfo = {
        vertexShader: vertShader,
        fragmentShader: fragShader,

        data: {
            // uniforms
            // Use a transformation matrix that makes 1 unit 1 pixel.
            u_matrix: { value: new Float32Array([
                2 / this.context.width, 0, 0, 0,
                0, 2 / this.context.height, 0, 0,
                0, 0, 1, 0,
                -1, -1, 0, 1
            ])},
            u_alpha: { value: new Float32Array([alpha]) },

            // attributes
            a_color: new Float32Array(colors),
            a_radius: new Float32Array(radii),
            a_x: new Float32Array(circleCount),
            a_y: new Float32Array(circleCount)
        },

        primitives: this.context.GL.POINTS,

        interleave: {
            a_x: false,
            a_y: false
        },

        usage: {
            a_x: this.context.GL.DYNAMIC_DRAW,
            a_y: this.context.GL.DYNAMIC_DRAW
        }
    };

    this.shader = new GLOW.Shader(circleShaderInfo);
}

WebGLCircleRenderer.prototype.setPositions = function(xs, ys) {
    this.shader.attributes.a_x.bufferSubData(xs);
    this.shader.attributes.a_y.bufferSubData(ys);
};

WebGLCircleRenderer.prototype.draw = function() {
    this.shader.draw();
};

WebGLCircleRenderer.prototype.dispose = function() {
    delete this.context;
    this.shader.dispose();
    delete this.shader;
};
```

```
    var context, stats, animationID, circleRenderer;
    function initPage() {
        var container = document.getElementById("container");

        context = new GLOW.Context({
            width: container.offsetWidth,
            height: container.offsetHeight,
            alpha: false
        });
        if (null === context.GL) {
            alert("no WebGL");
            return false;
        }

        container.appendChild(context.domElement);
        context.setupClear( { red: 0, green: 0, blue: 0 } );
        context.GL.enable(context.GL.BLEND);
        context.GL.blendFunc(context.GL.SRC_ALPHA,
                             context.GL.ONE_MINUS_SRC_ALPHA);


        stats = new Stats();
        stats.setMode(0);
        stats.domElement.style.position = 'absolute';
        stats.domElement.style.left = '0px';
        stats.domElement.style.top = '0px';
        document.body.appendChild( stats.domElement );

        return true;
    }

    function initCircles() {
        if (animationID !== undefined) {
            cancelAnimationFrame(animationID);
            circleRenderer.dispose();
        }

        var numPoints = parseInt(document.getElementById("numCircles").value);
        var minRadius = 5;
        var maxRadius = parseInt(document.getElementById("maxRadius").value);
        var alpha = parseFloat(document.getElementById("alpha").value);
        var maxVelocity = 1.5;
        var bands = 3;
        var bandWidth = 0.75
        var pointsPerBand = (numPoints / bands) | 0;

        var colors = new Float32Array(numPoints * 3);
        var xs = new Float32Array(numPoints);
        var ys = new Float32Array(numPoints);
        var radii = new Float32Array(numPoints);
        var phase = new Float32Array(numPoints);

        for (var band = 0; band < bands; band++) {
            for (var i = 0; i < pointsPerBand; i++) {
                var point = (band * pointsPerBand) + i;
                colors[(point * 3) + ((band + 0) % 3)] = 0.8 * (i / pointsPerBand);
                colors[(point * 3) + ((band + 1) % 3)] = 1;
                colors[(point * 3) + ((band + 2) % 3)] = 0.8 * (1 - (i / pointsPerBand));

                xs[point] = (i / pointsPerBand) * context.width;
                ys[point] = ((band / bands) * context.height) + (Math.random() * ((context.height * bandWidth) / bands));
                radii[point] = minRadius + (Math.random() * (maxRadius - minRadius));
                phase[point] = Math.random() * Math.PI * 2;
            }
        }
```

```
        circleRenderer = new WebGLCircleRenderer(context, numPoints,
                                                 colors, radii, alpha);


        var theta = 0;
        var dTheta = 0.01;
        var multiplier = 1.5;
        function step() {
            stats.begin();

            theta = (theta + dTheta) % (Math.PI * 2);
            for (var i = 0; i < numPoints; i++) {
                ys[i] += Math.sin(theta + phase[i]) * multiplier;
            }
            circleRenderer.setPositions(xs, ys);

            context.cache.clear();
            context.clear();
            circleRenderer.draw();
            animationID = requestAnimationFrame(step);

            stats.end();
        }

        animationID = requestAnimationFrame(step);
    }

    if (initPage()) {
        var drawButton = document.getElementById("drawButton");
        drawButton.onclick = initCircles;
        initCircles();
    }
```

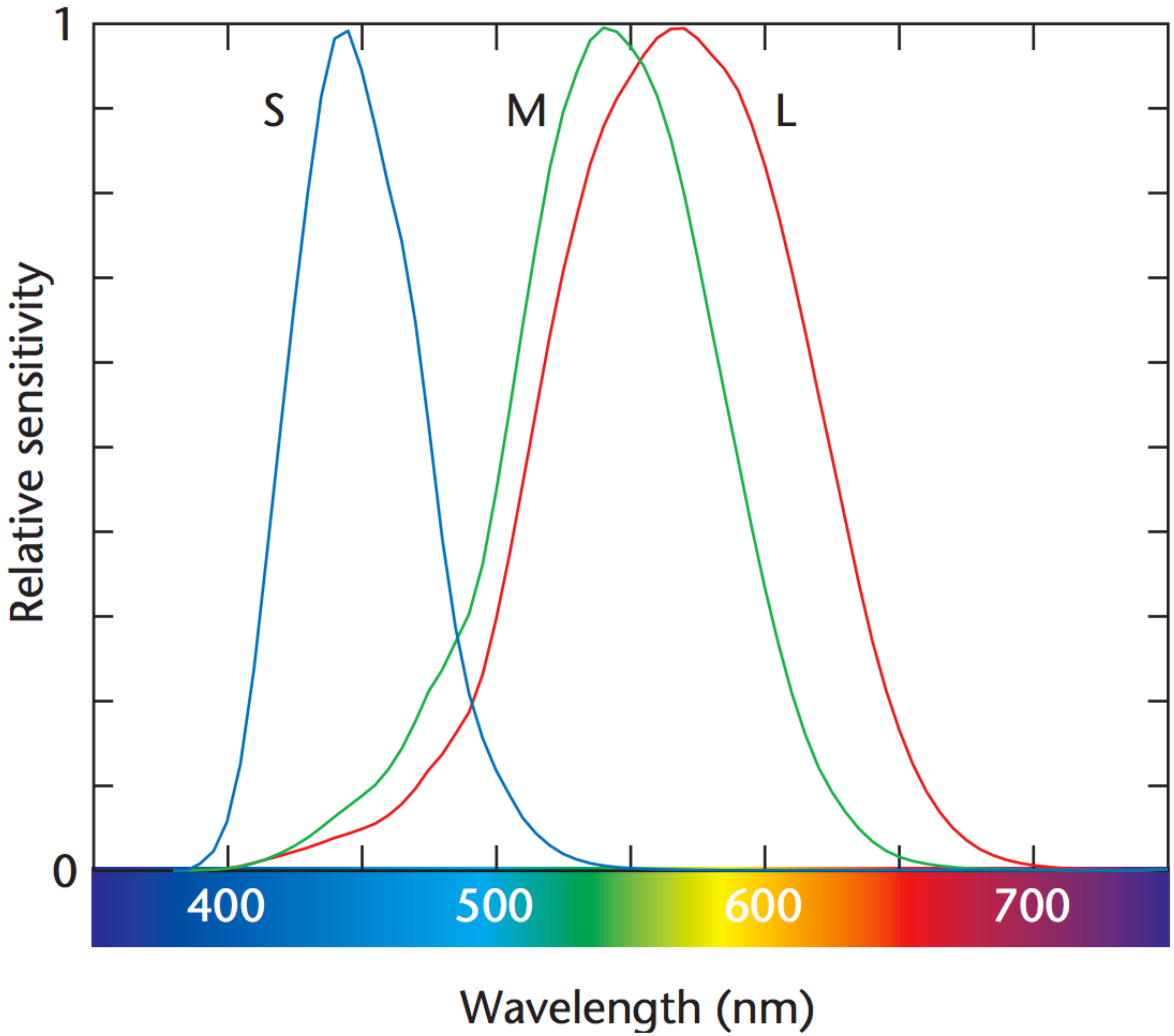
CUDA/OpenGL: 32M points

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

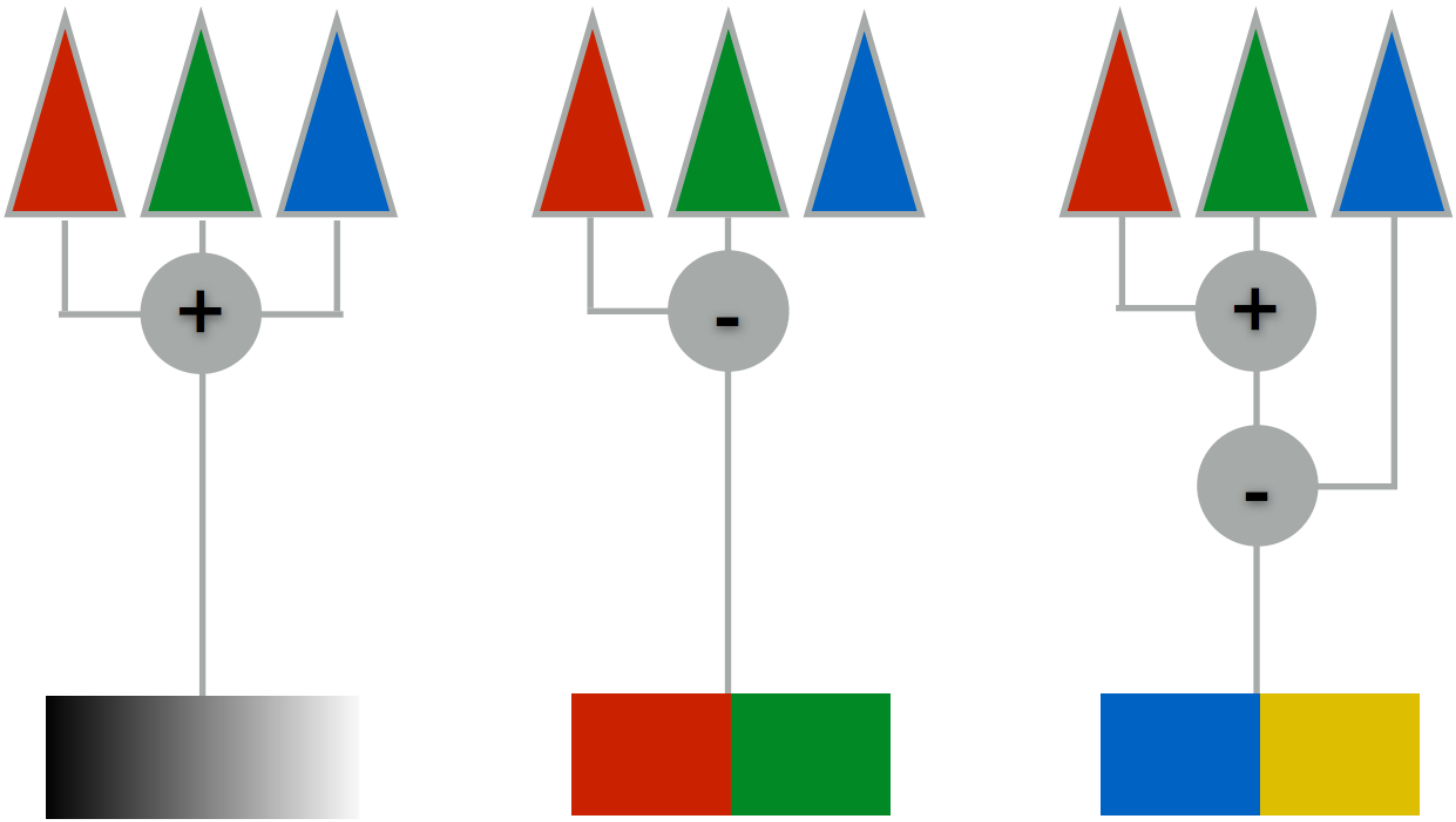
Principles

Color Vision

How does your eye work?



OPPONENT PROCESS MODEL

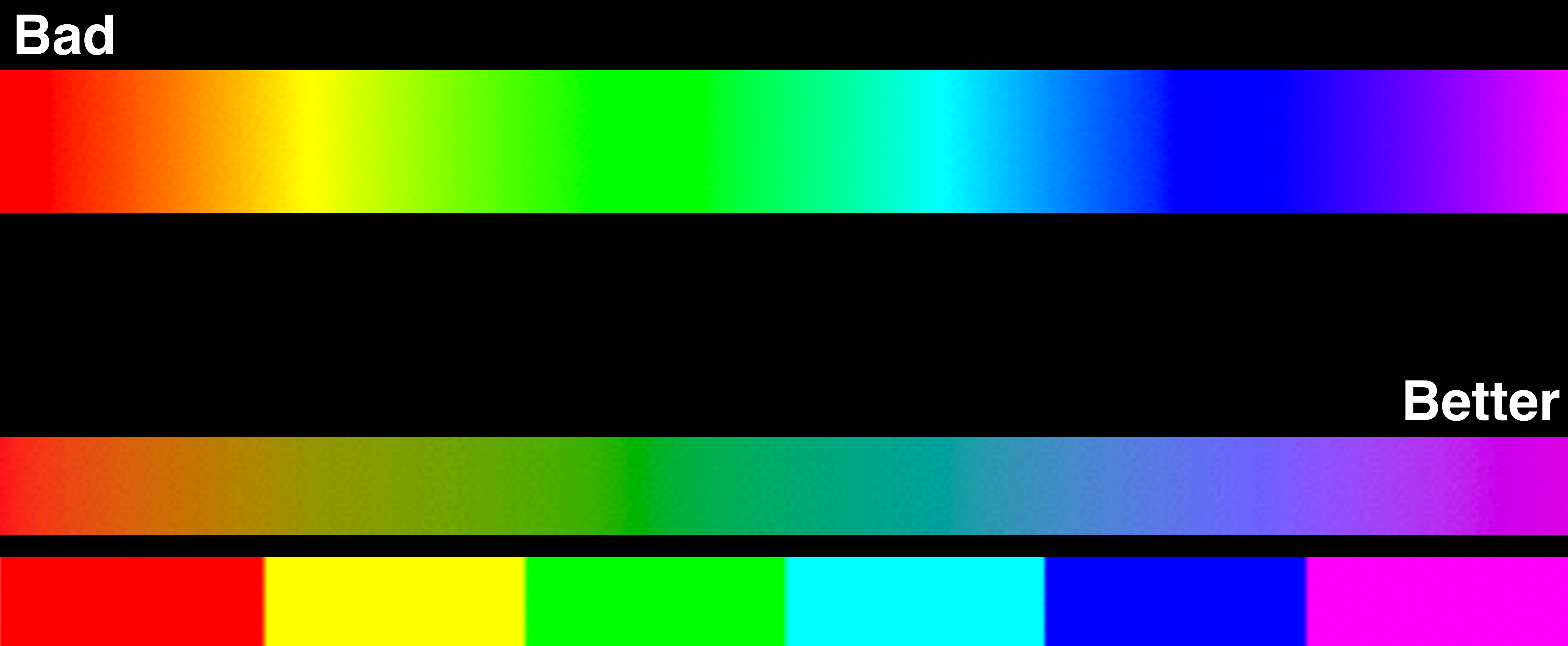


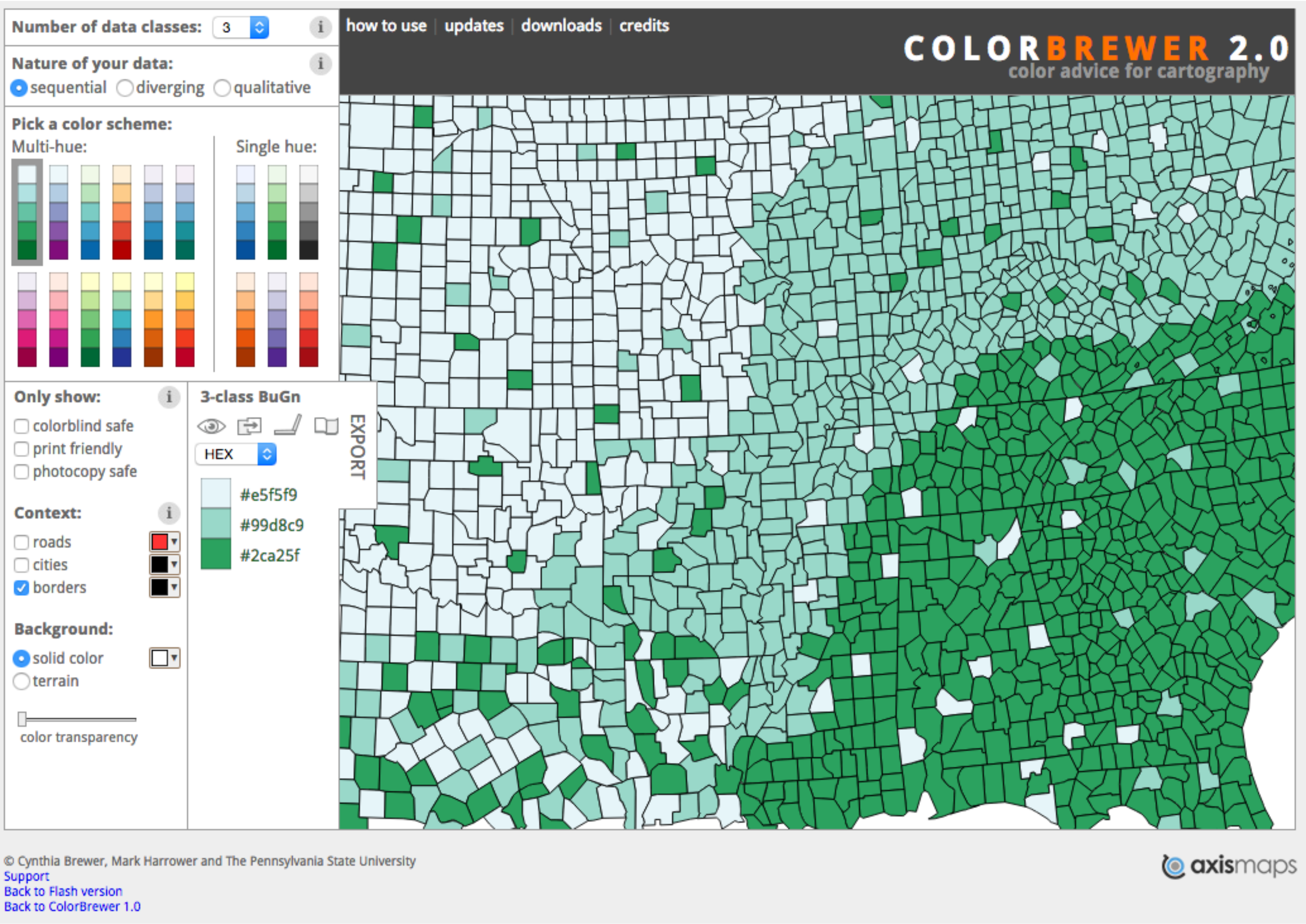
Polar Lab (or HCL)

- “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**

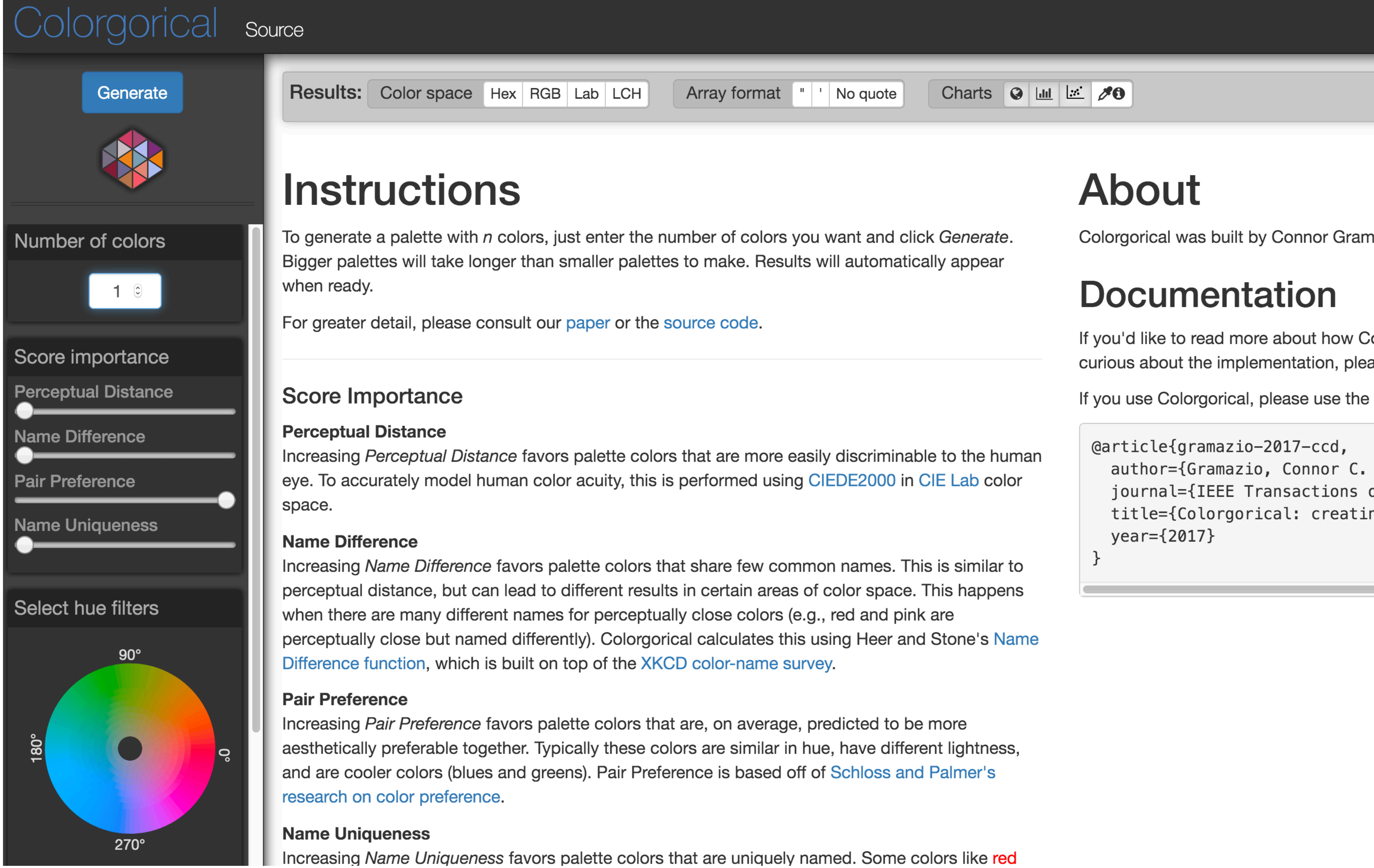
http://cscheid.net/static/20120216/hcl_frame.html

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



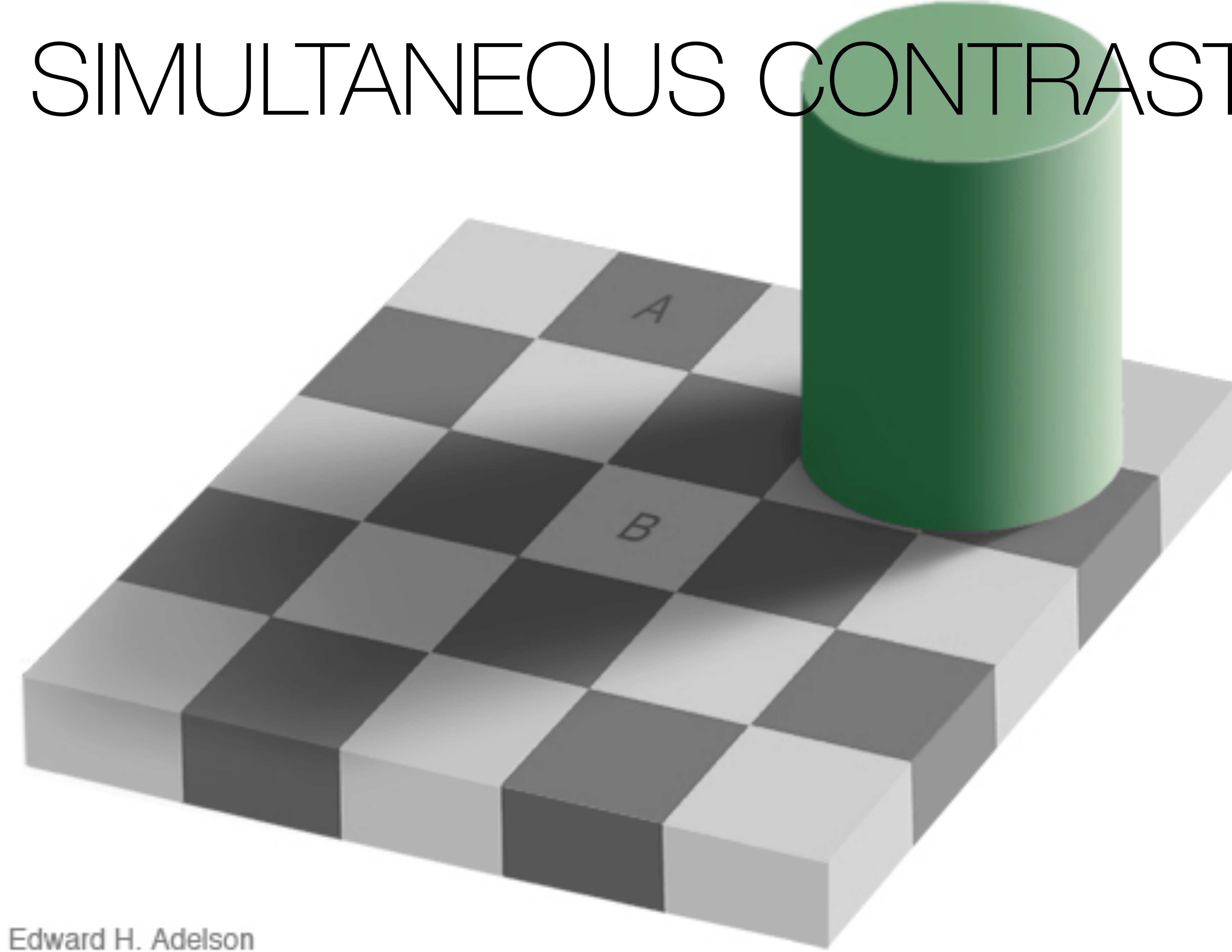


COLORBREWER

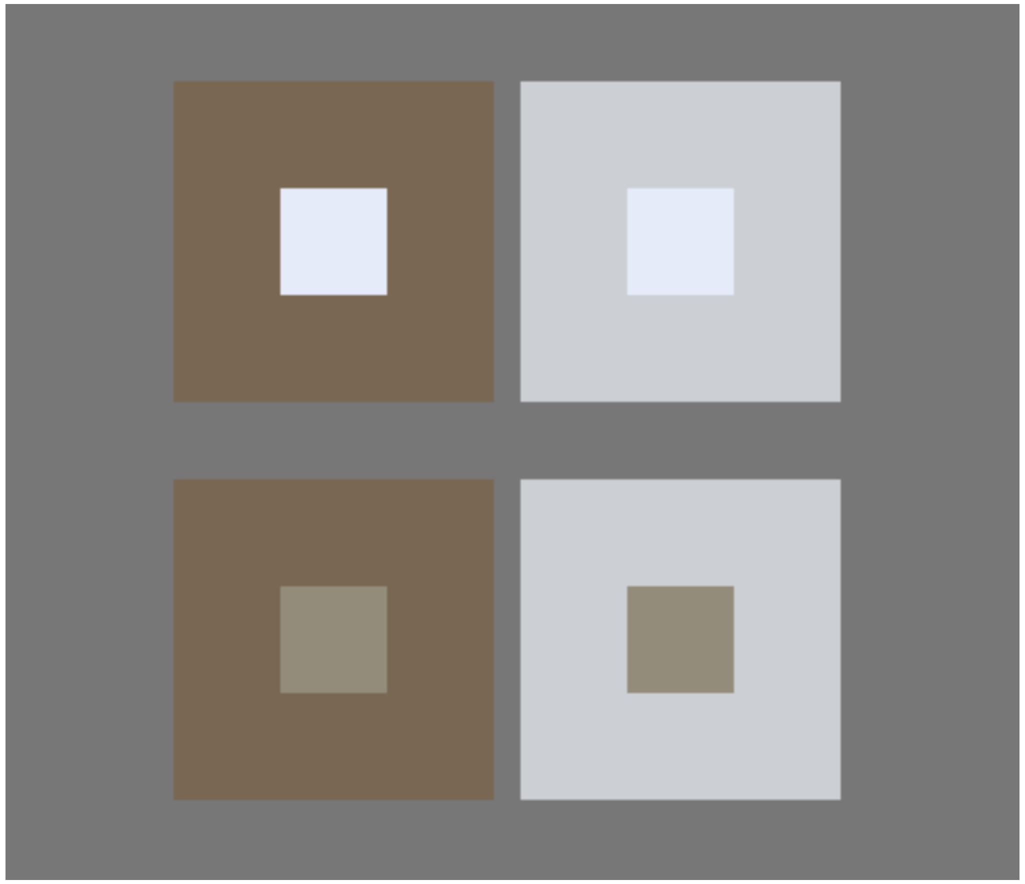


COLORGORICAL

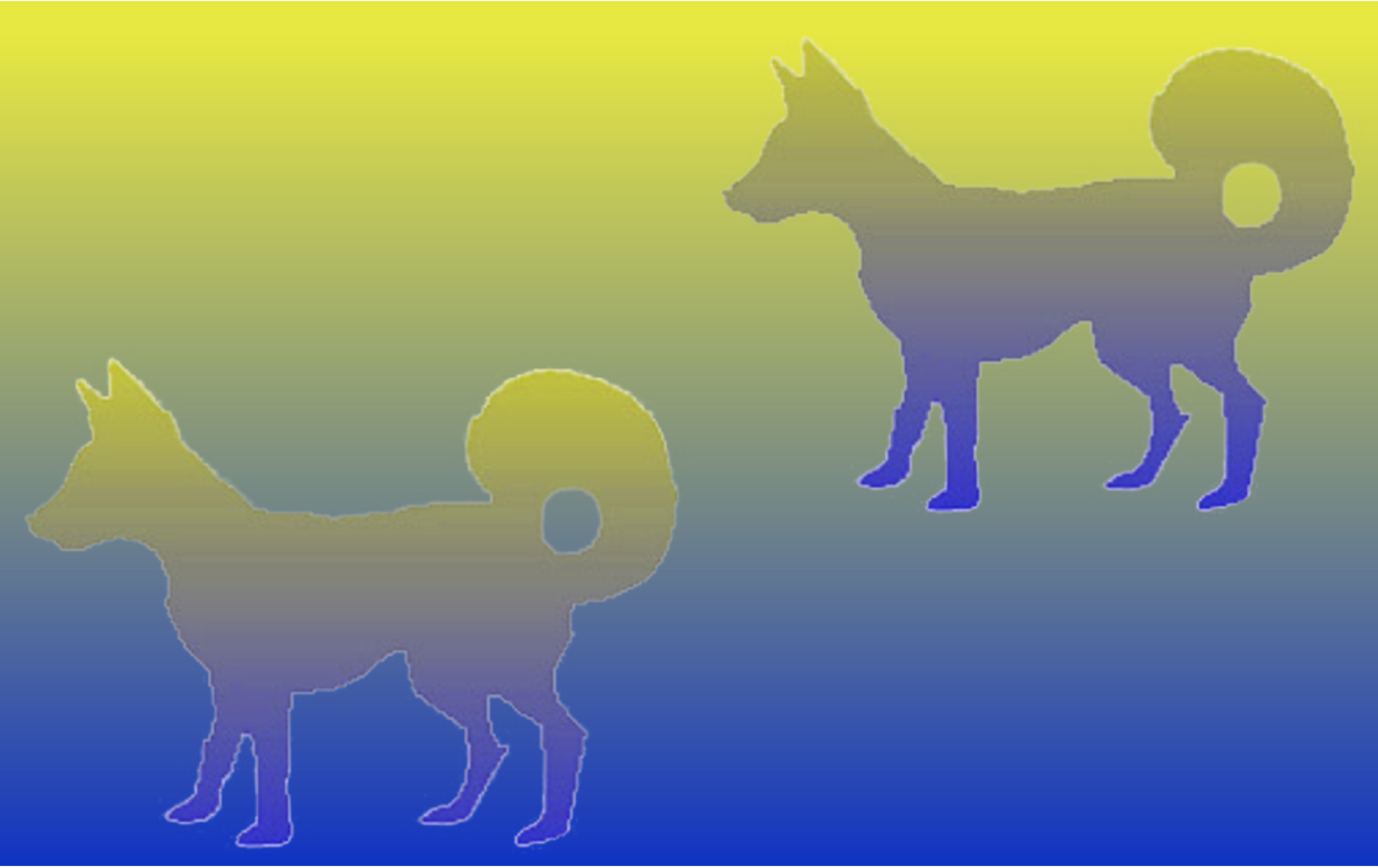
SIMULTANEOUS CONTRAST



SIMULTANEOUS CONTRAST

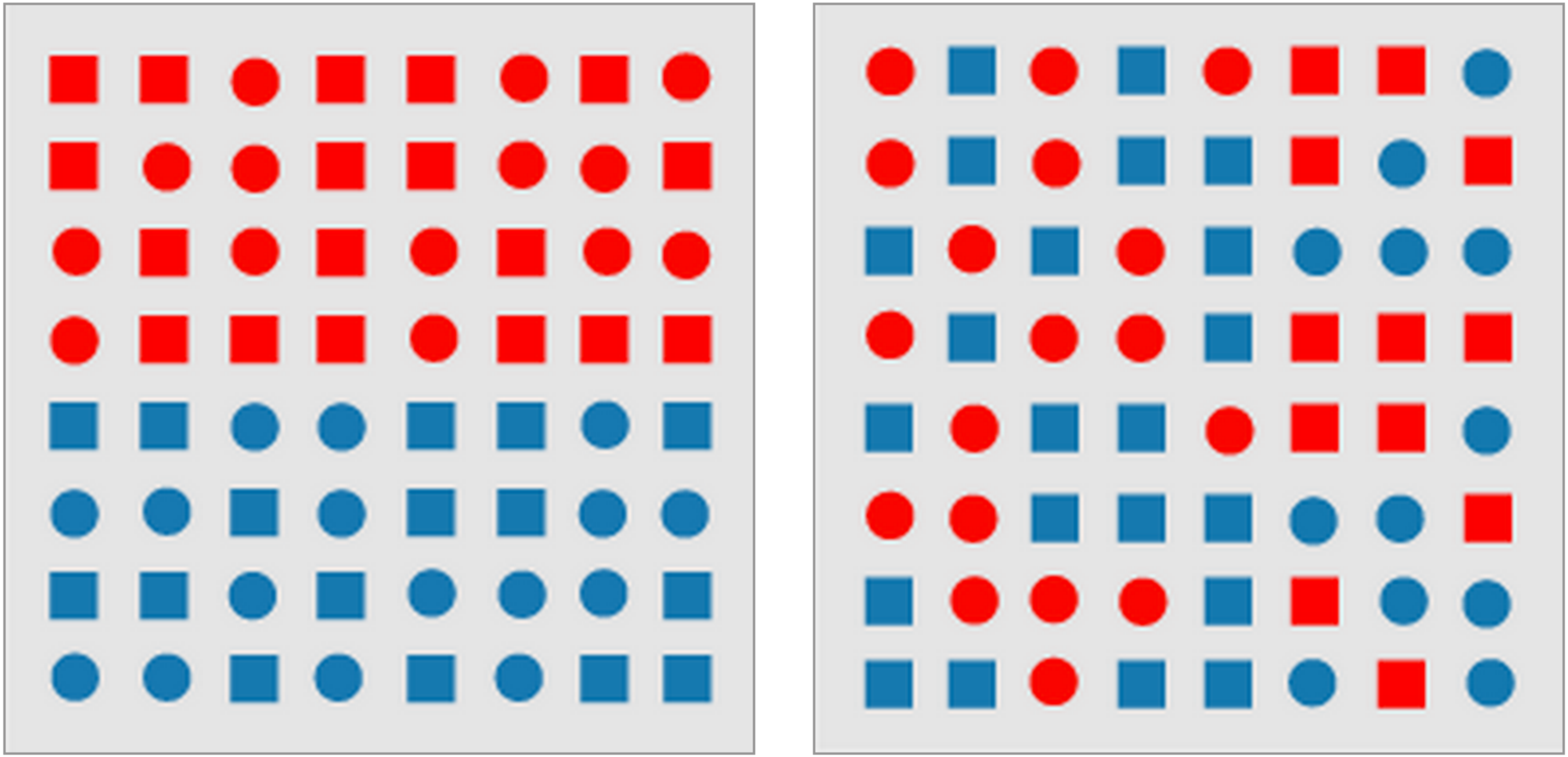


http://www.handprint.com/HP/WCL/tech13.html



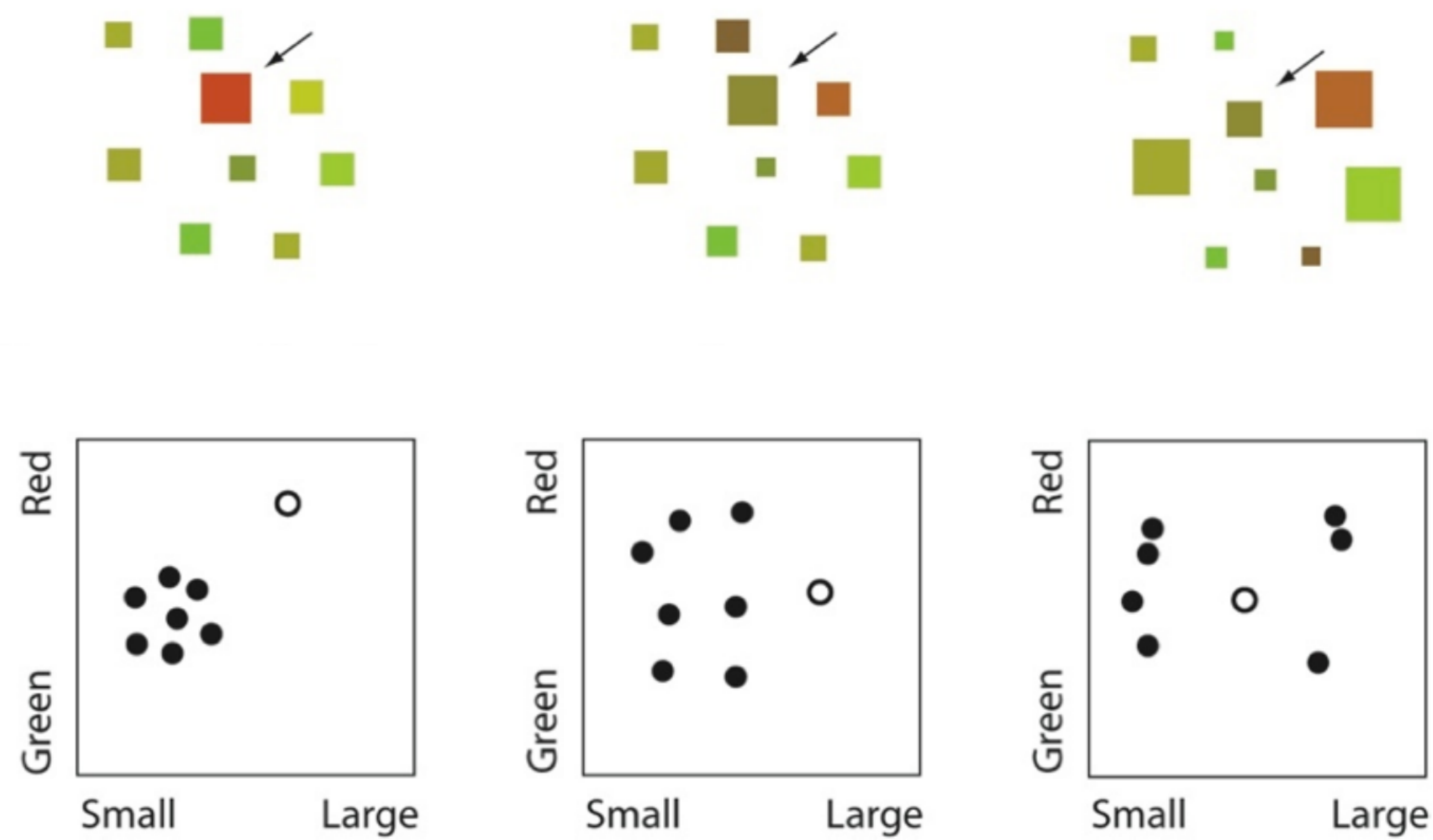

PREATTENTIVENESS,

OR “VISUAL POP-OUT”

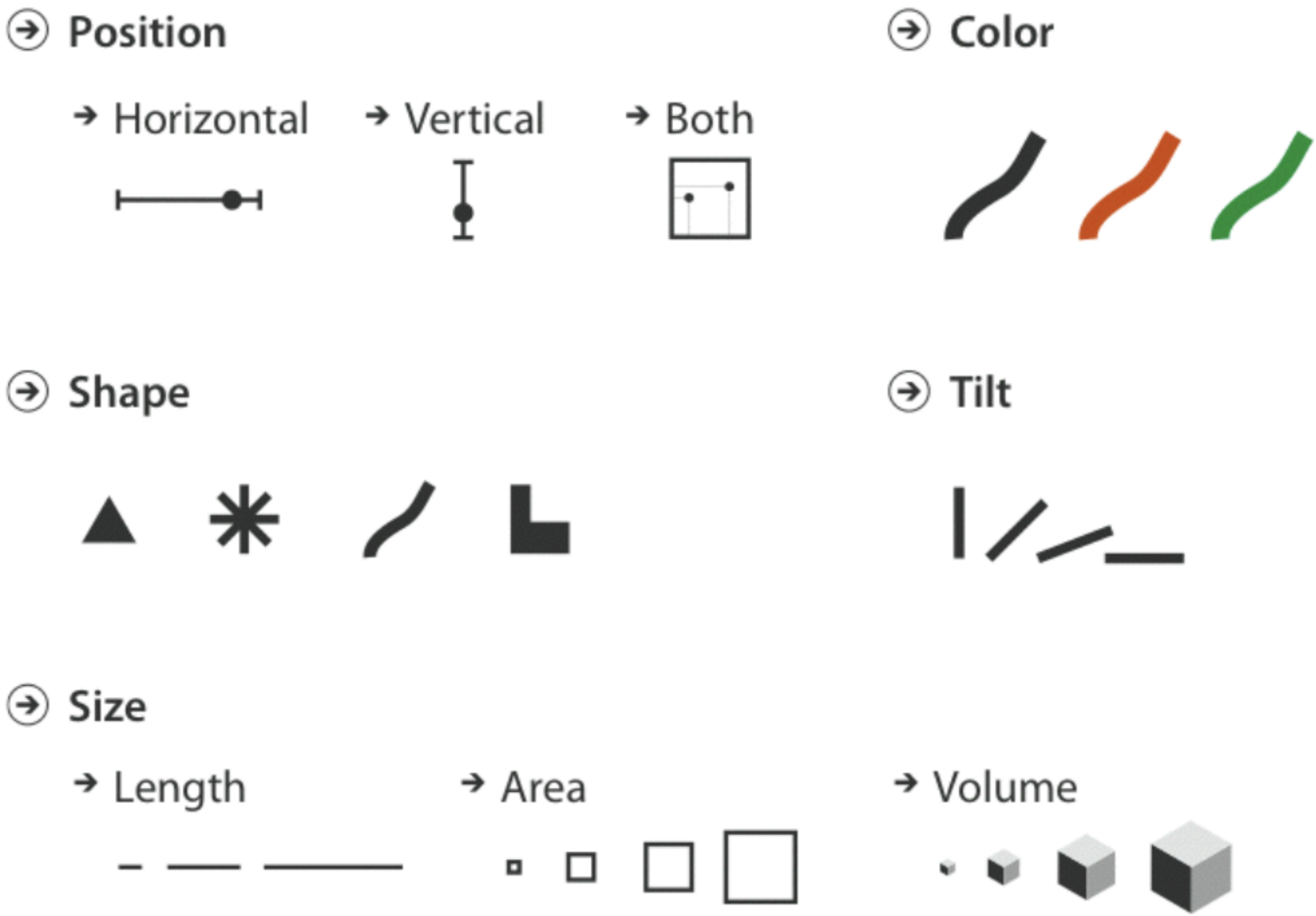


(a) (b)

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

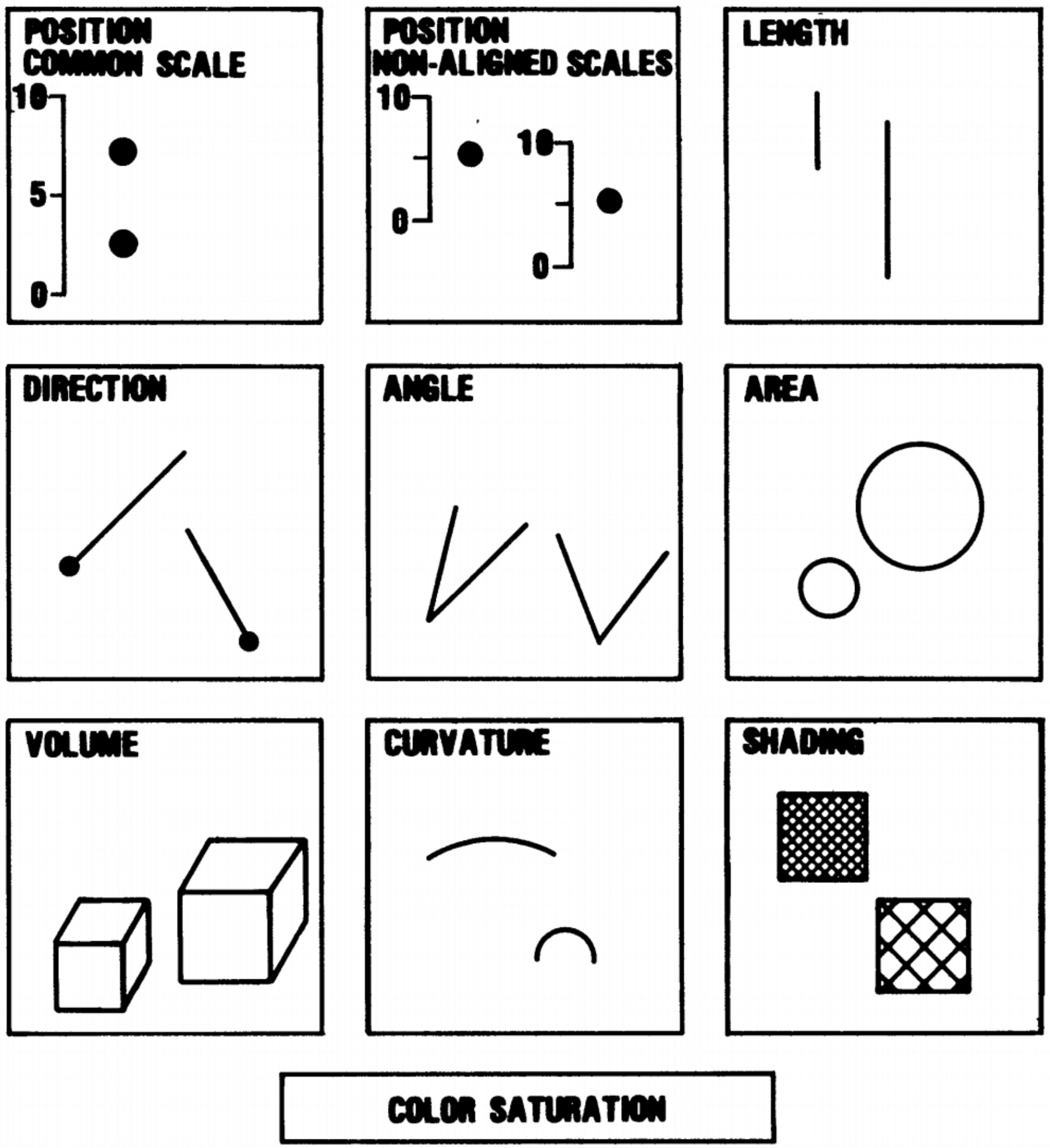


Figure 1. Elementary perceptual tasks.

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

Pie Chart Bad, Scatterplot Good

Cleveland/McGill perception papers

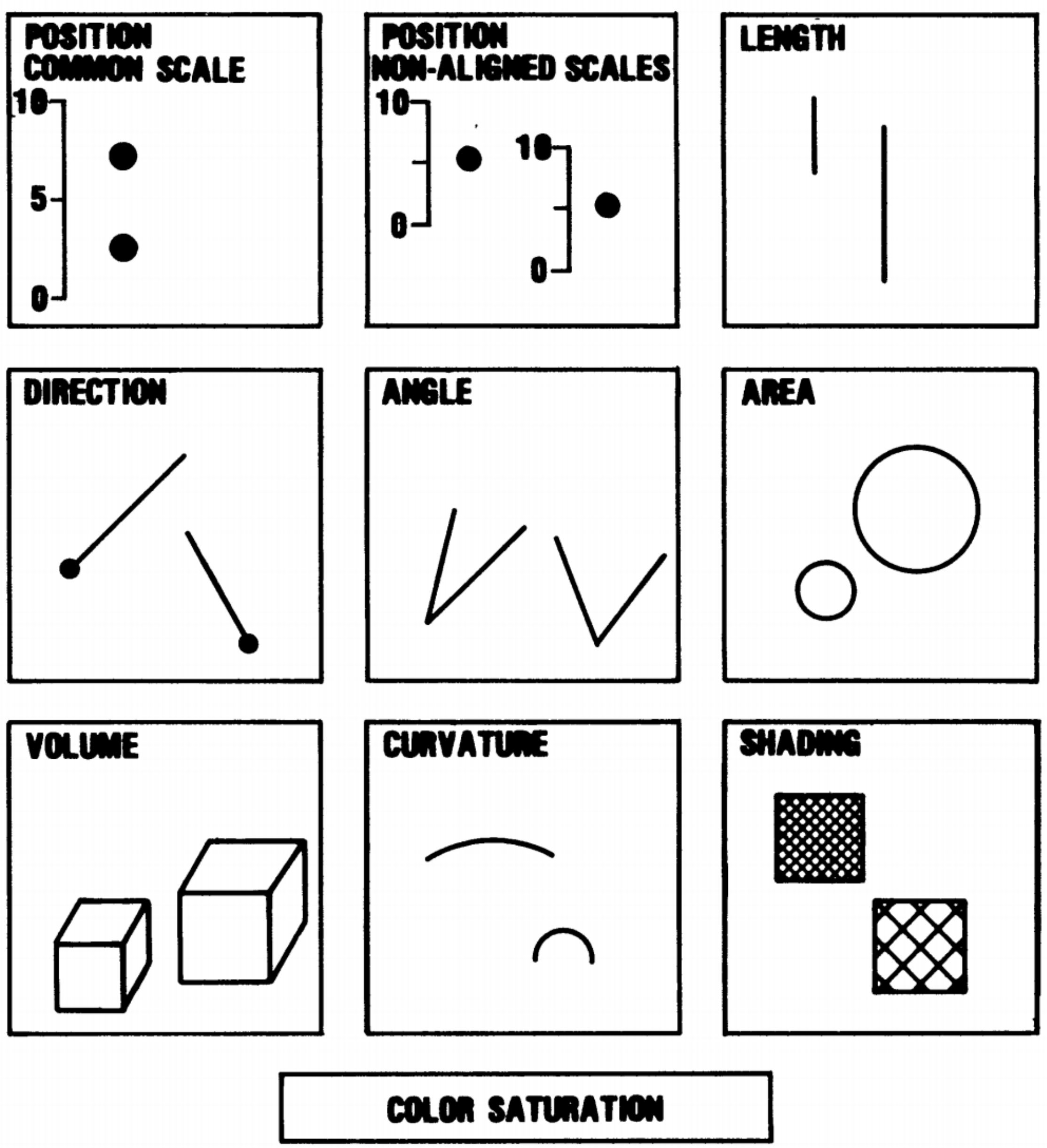
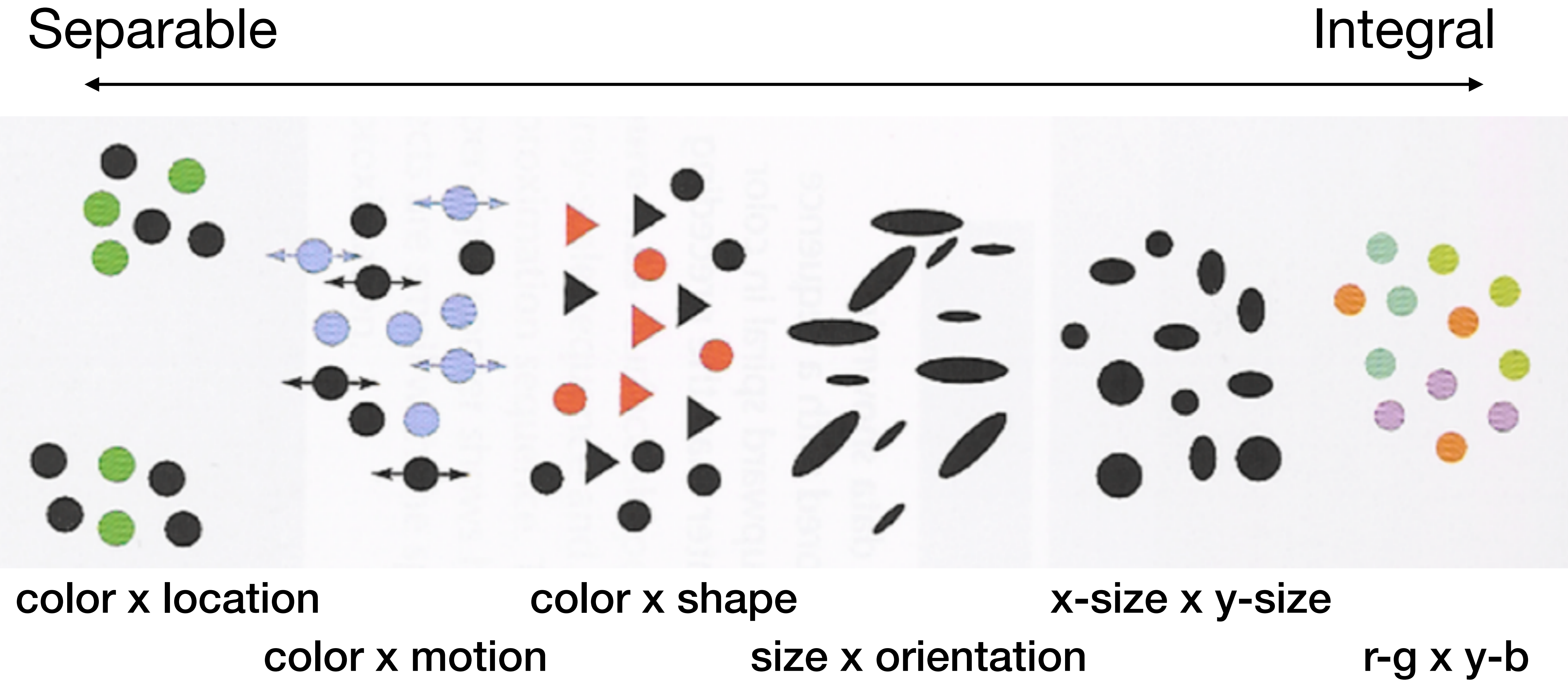


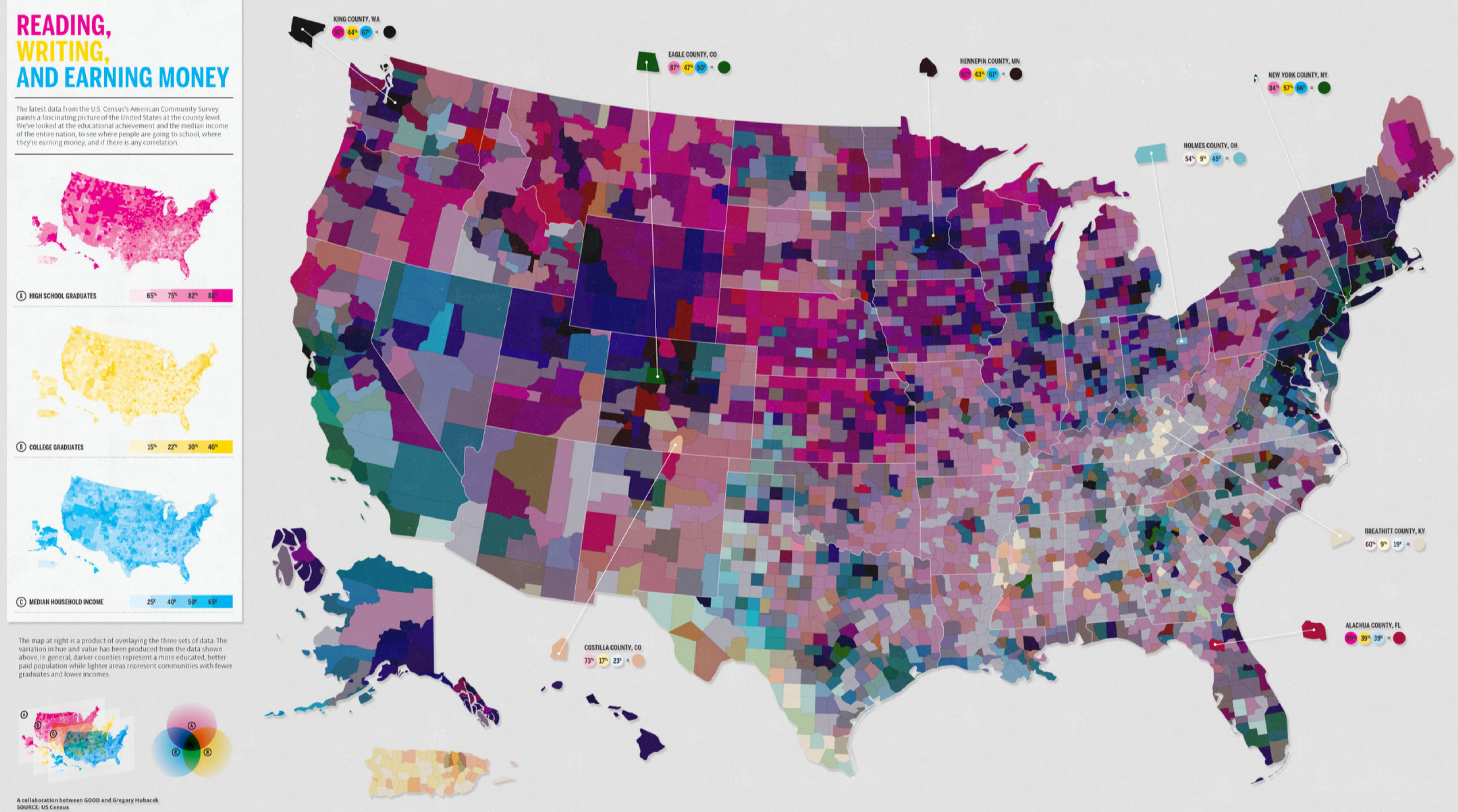
Figure 1. Elementary perceptual tasks.

- Notice the "elementary perceptual tasks"
- What about higher-level tasks?

Integral vs. Separable Channels



Colin Ware, 2004, p180

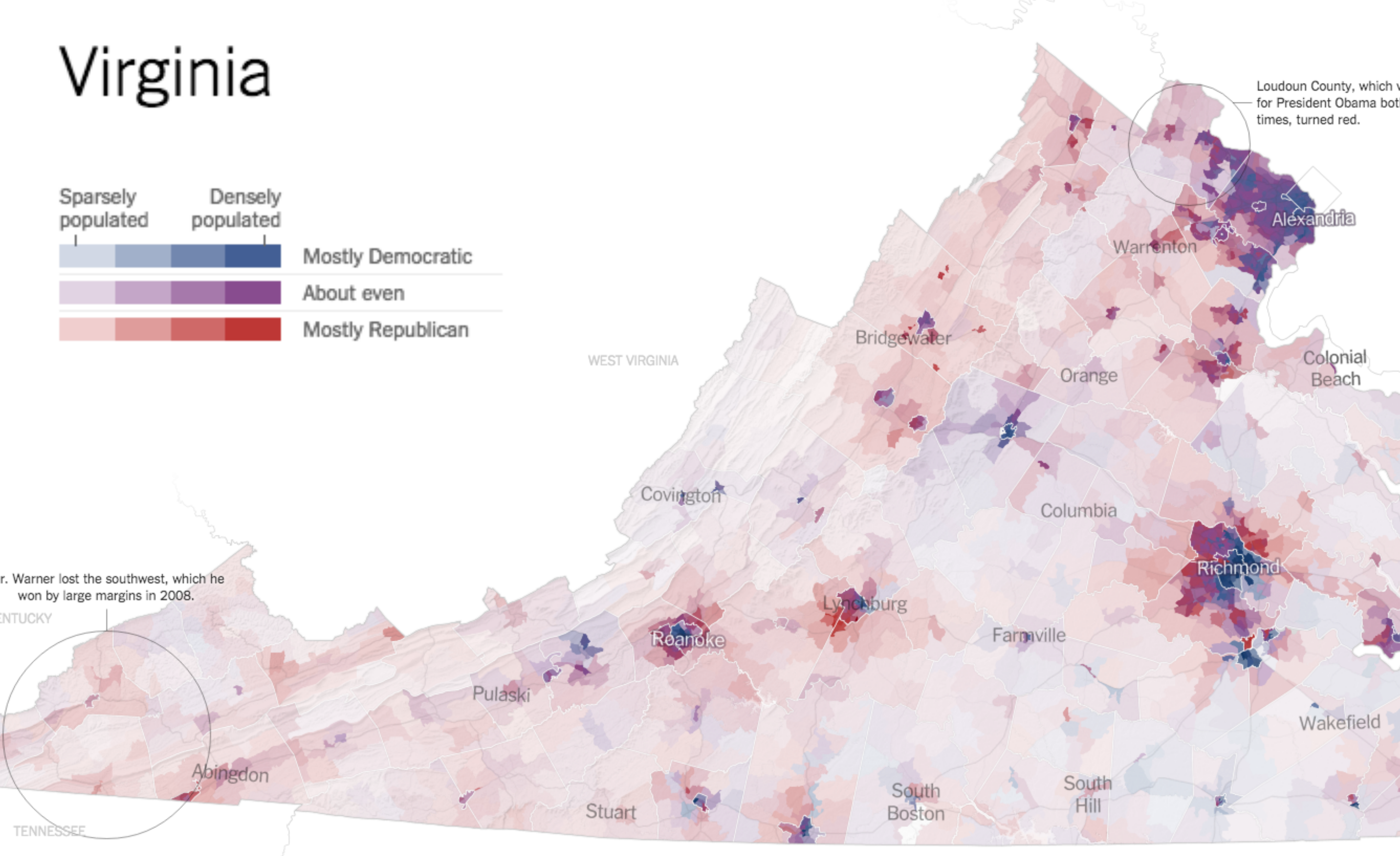


Trivariate (!) Color Map (terrible, terrible idea)

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

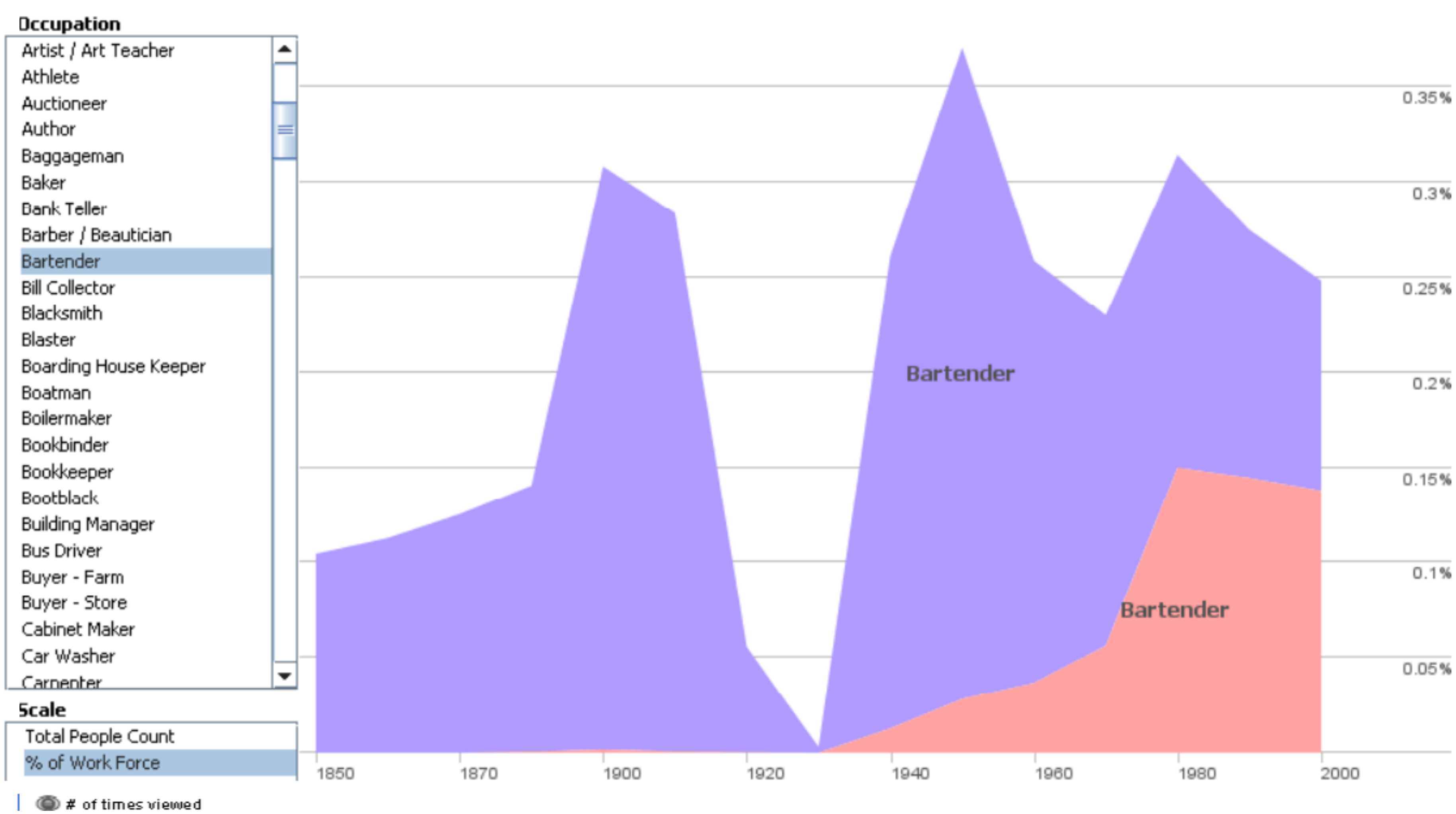


Bivariate Color Maps are Possible, but Hard

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

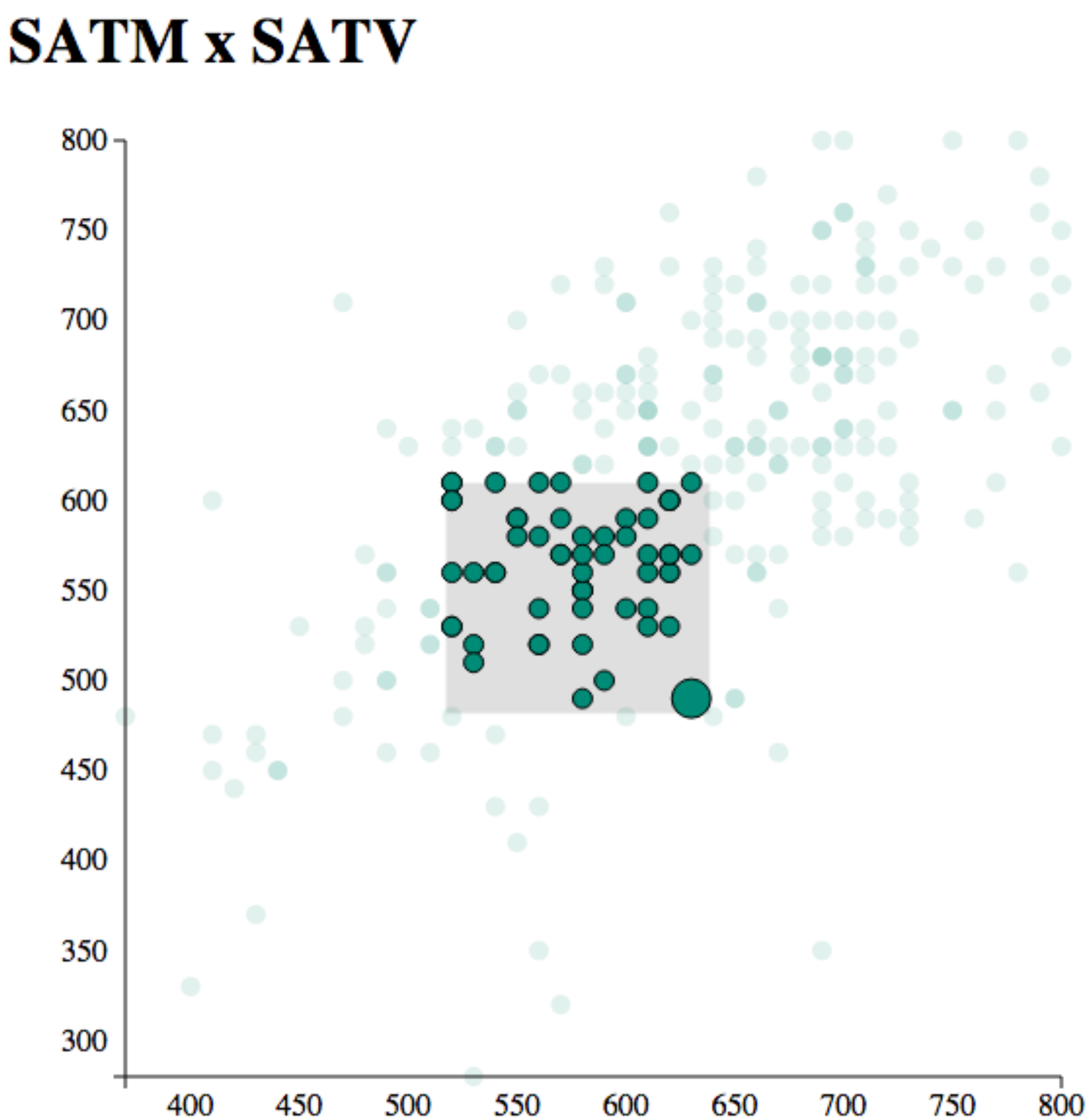
Interaction

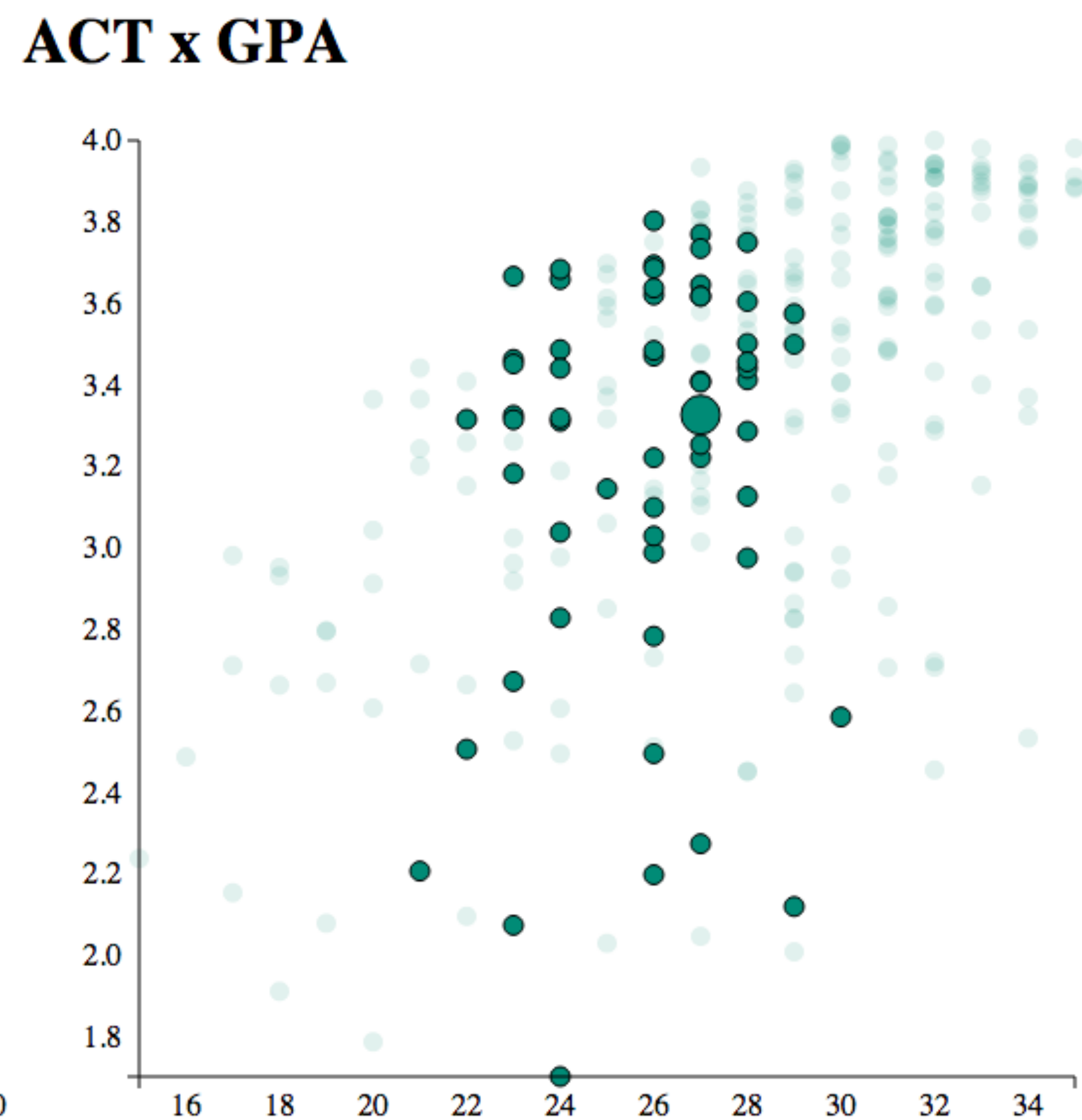
- Interpret the state of elements in the UI as a **clause** in a **query**. As UI changes, update data



Willett et al., TVCG 2007 (*)

Linked Brushing





Shneiderman’s “Visual information seeking mantra”

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

Techniques

Regular Scatterplots

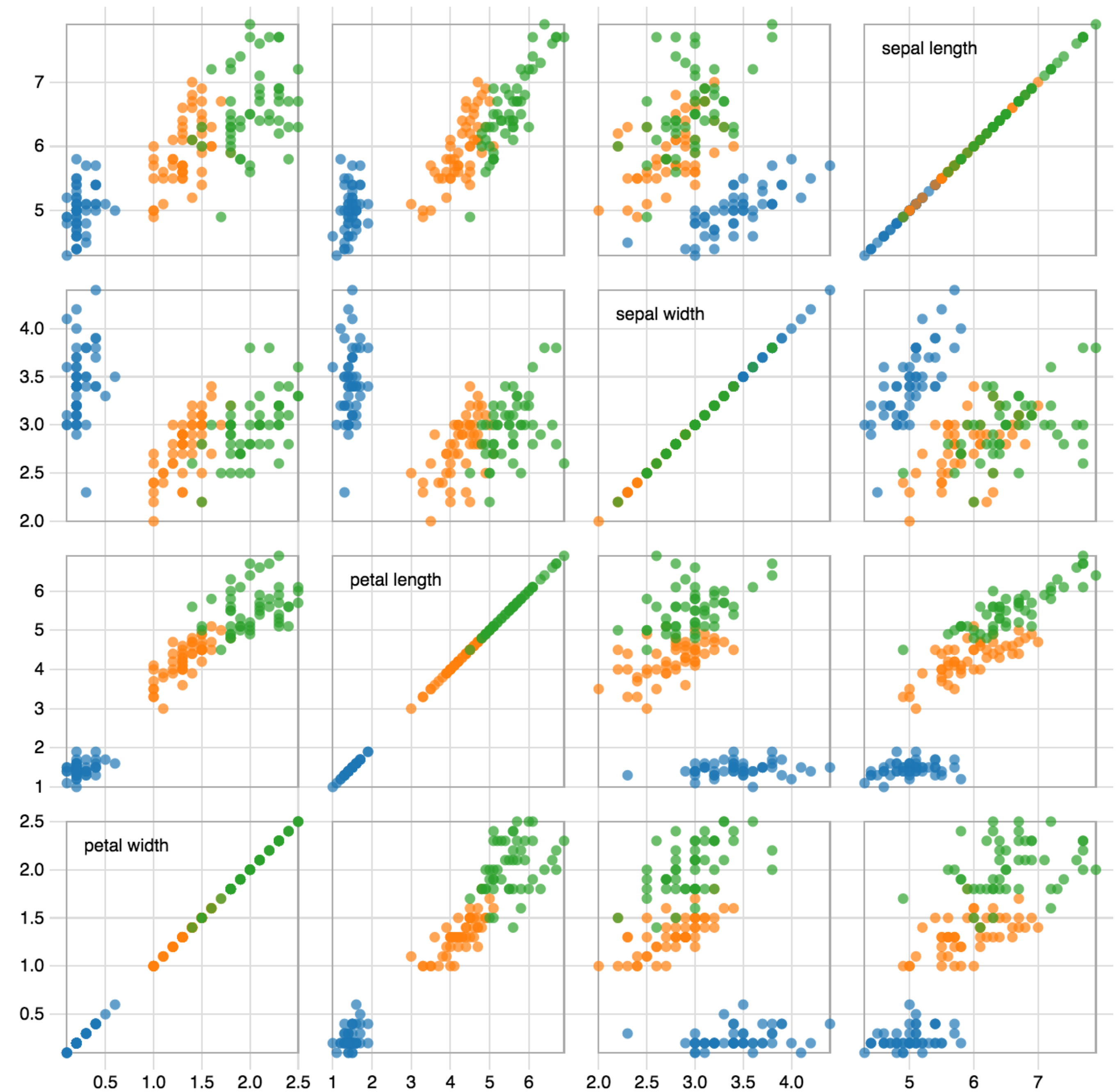
- Every data point is a vector:

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

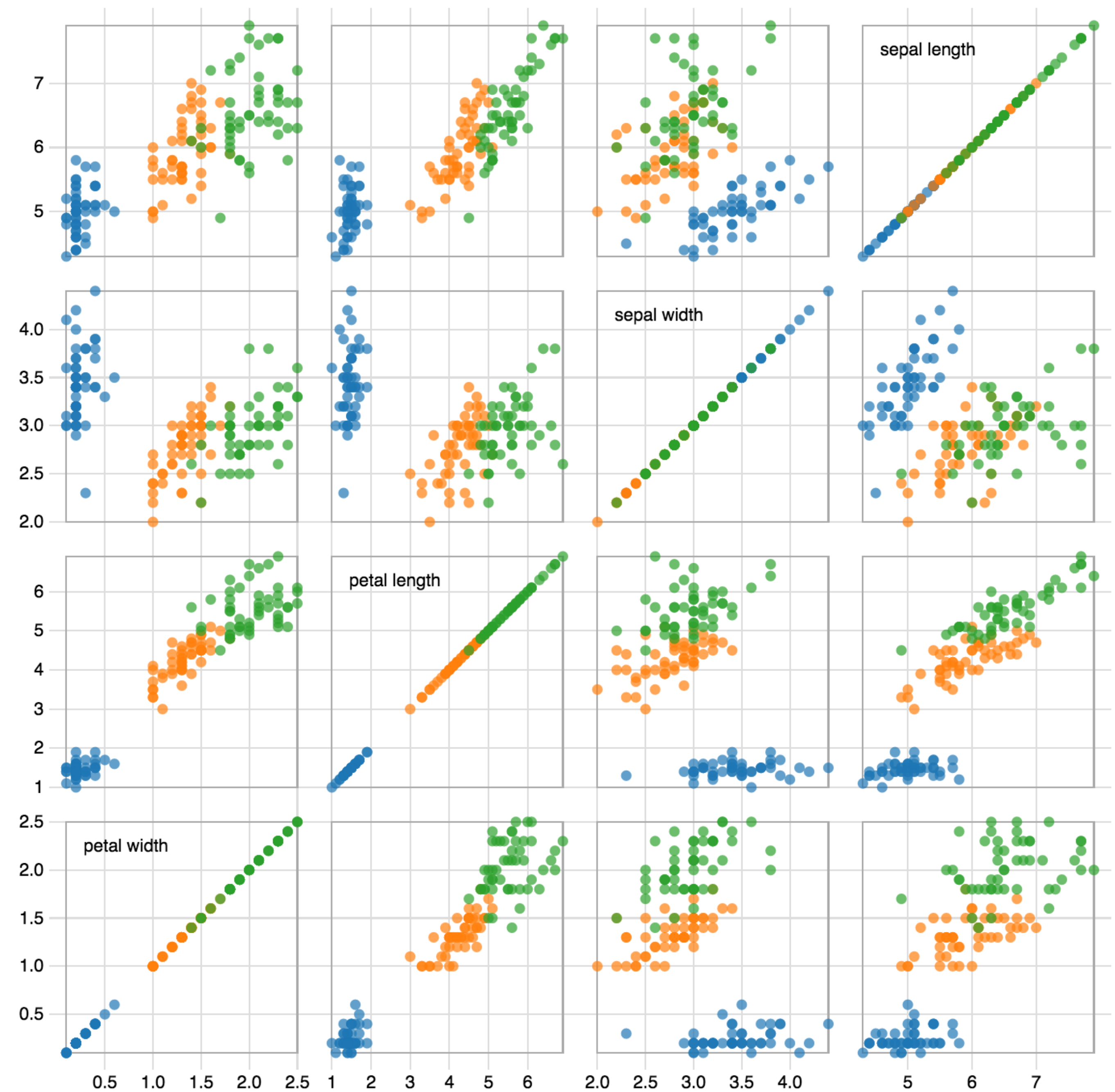
- Every scatterplot is produced by a very simple matrix:

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

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

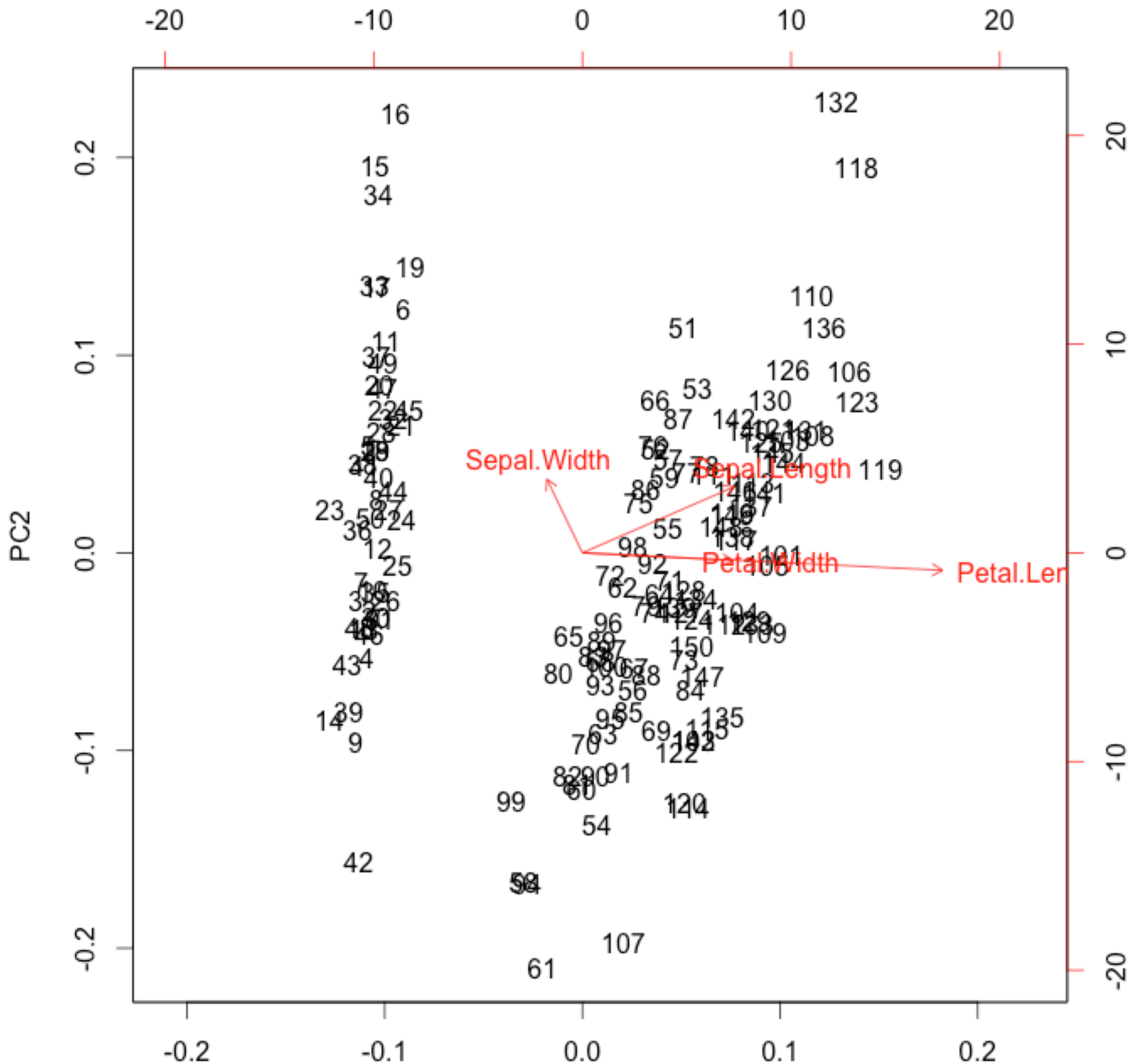


What about other matrices?

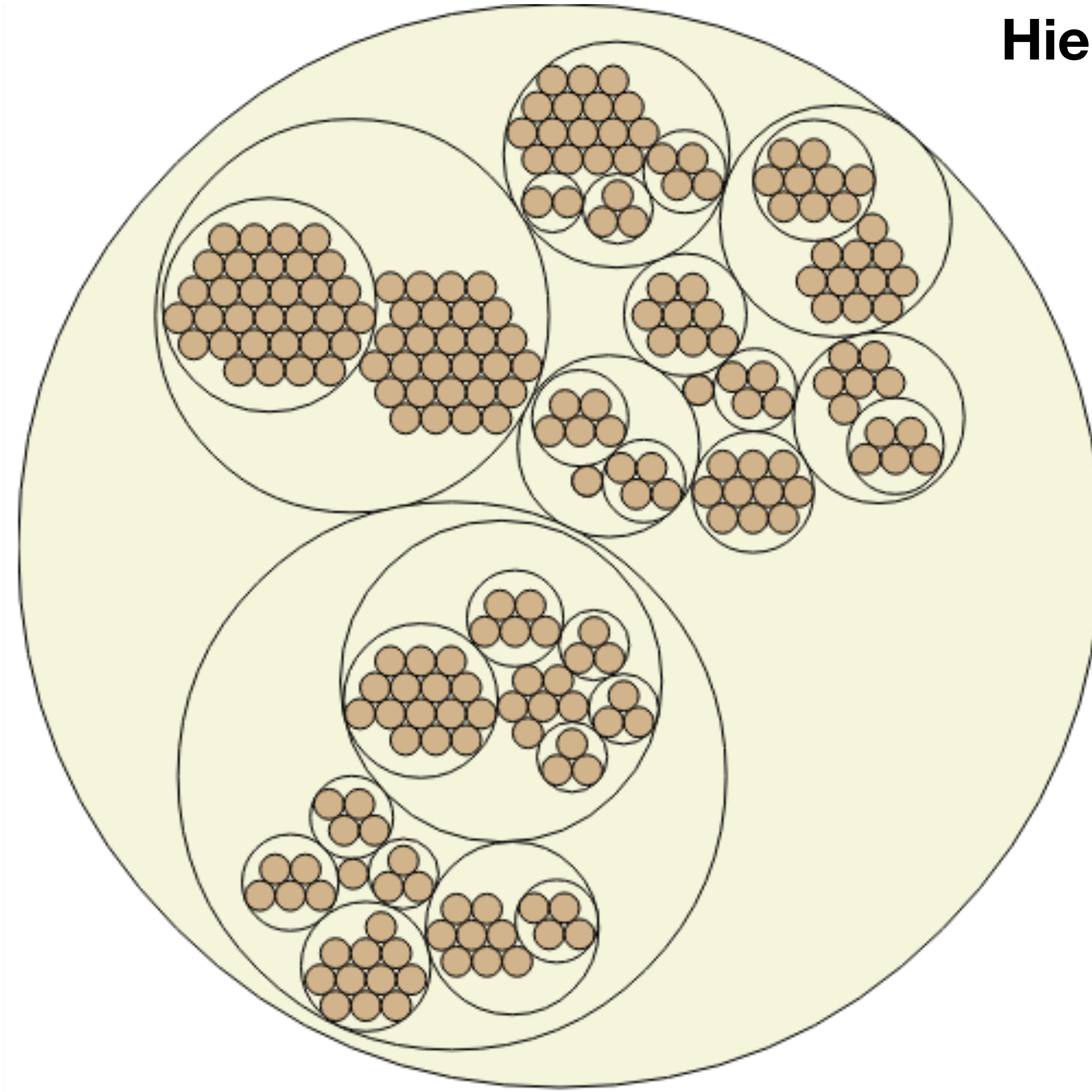


Dimensionality Reduction

Principal Component Analysis

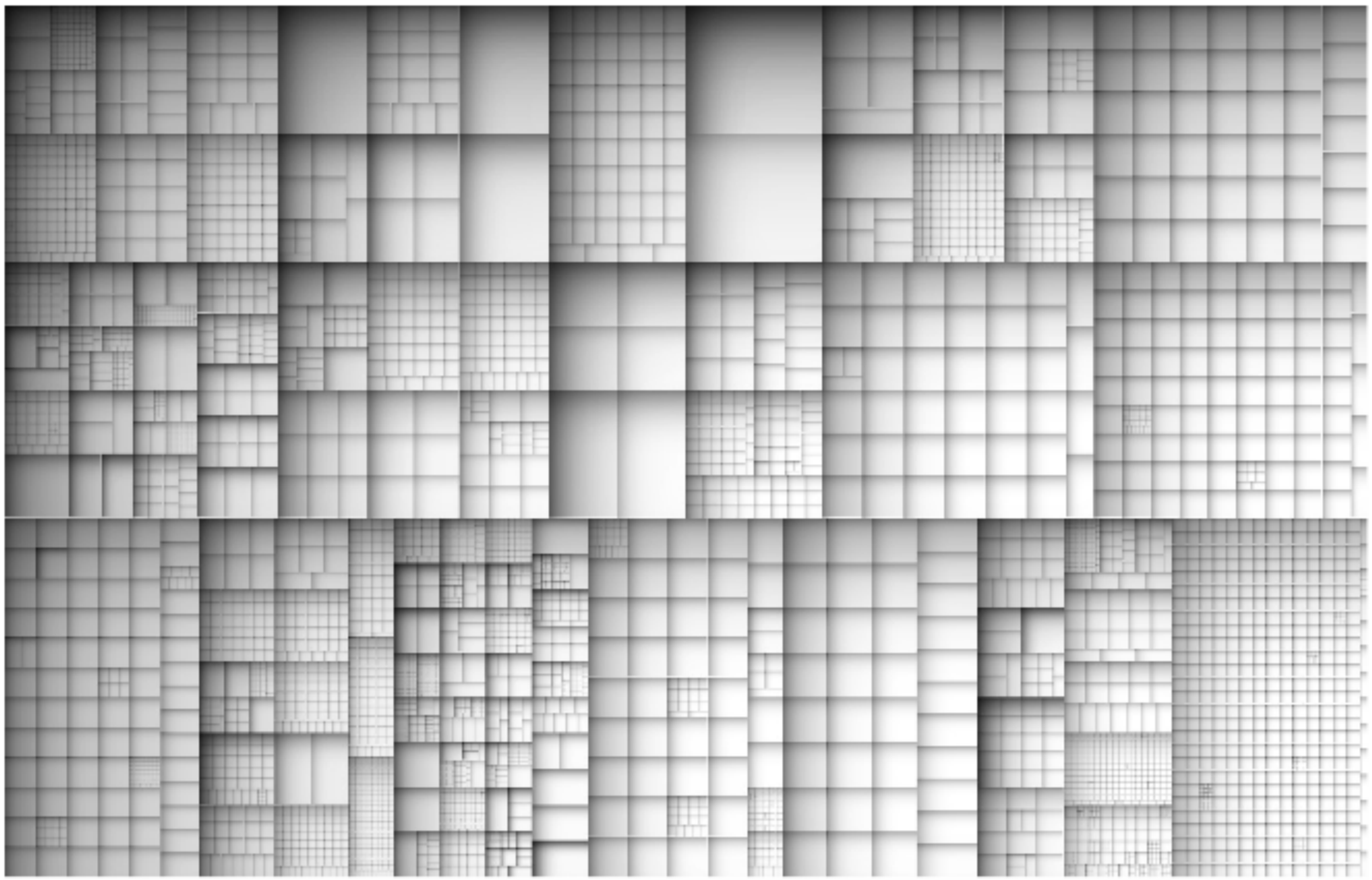


Hierarchies



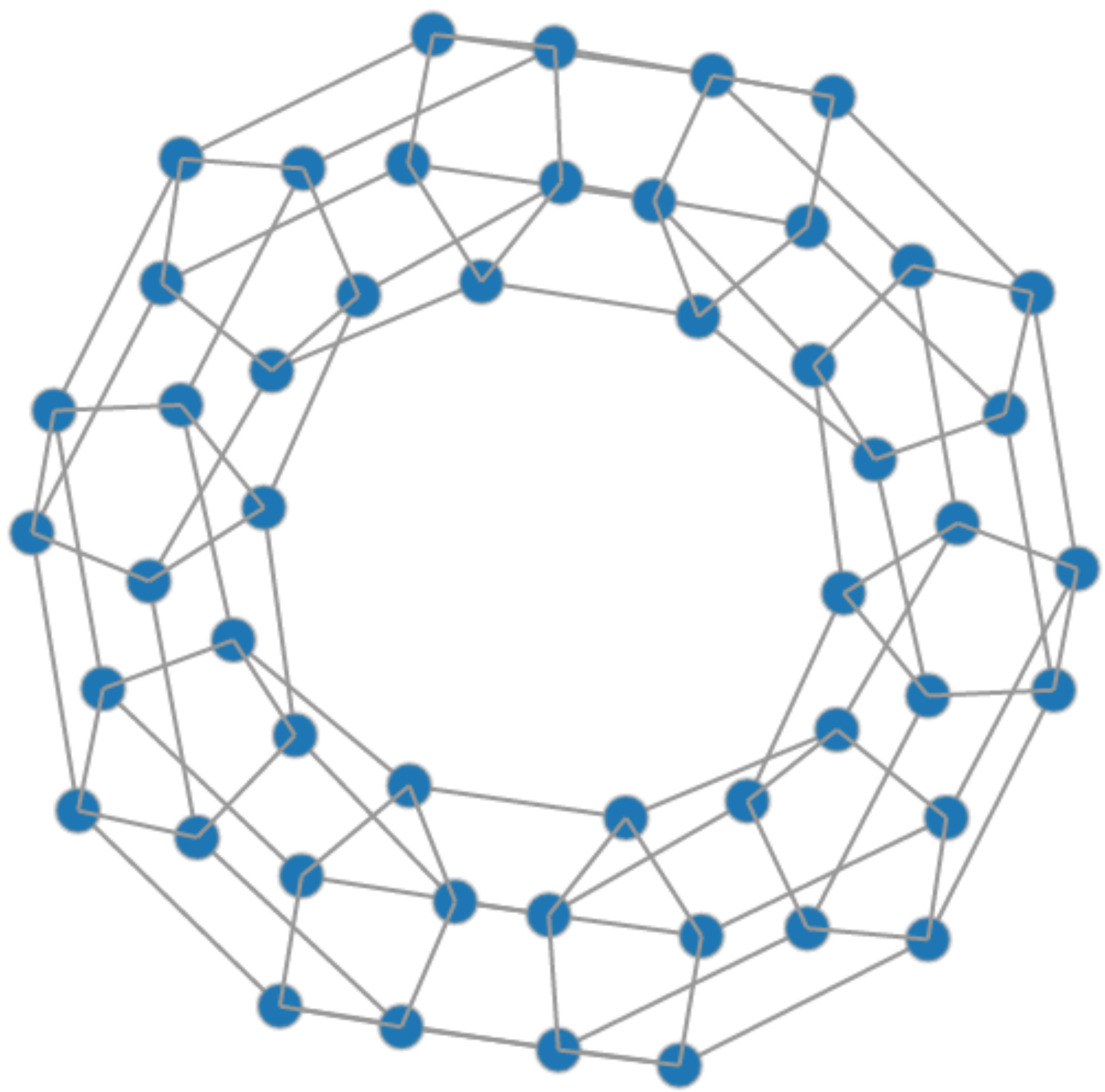
http://jsfiddle.net/VividD/WDCpq/8/

Hierarchies



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

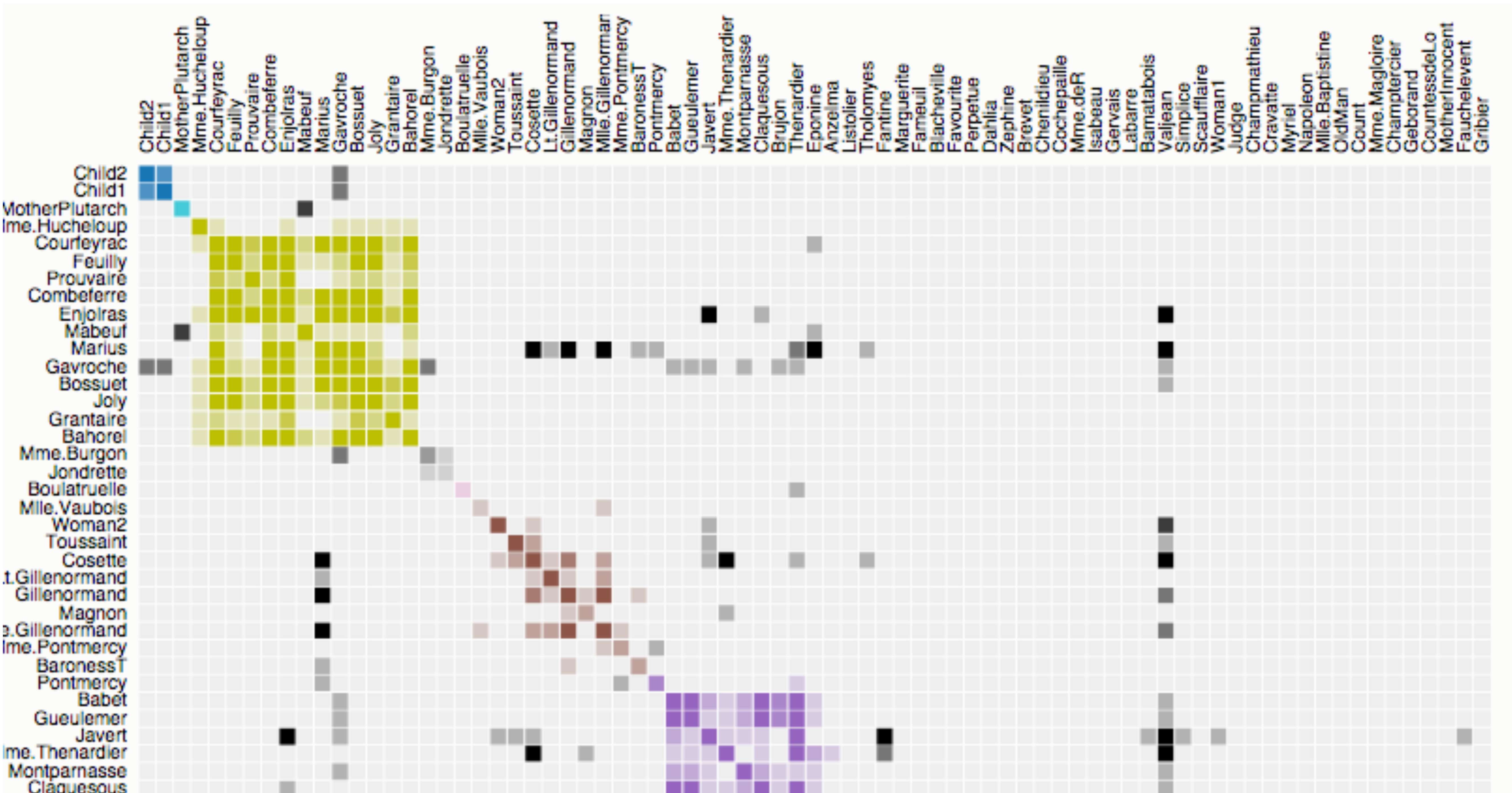
Node-link diagrams



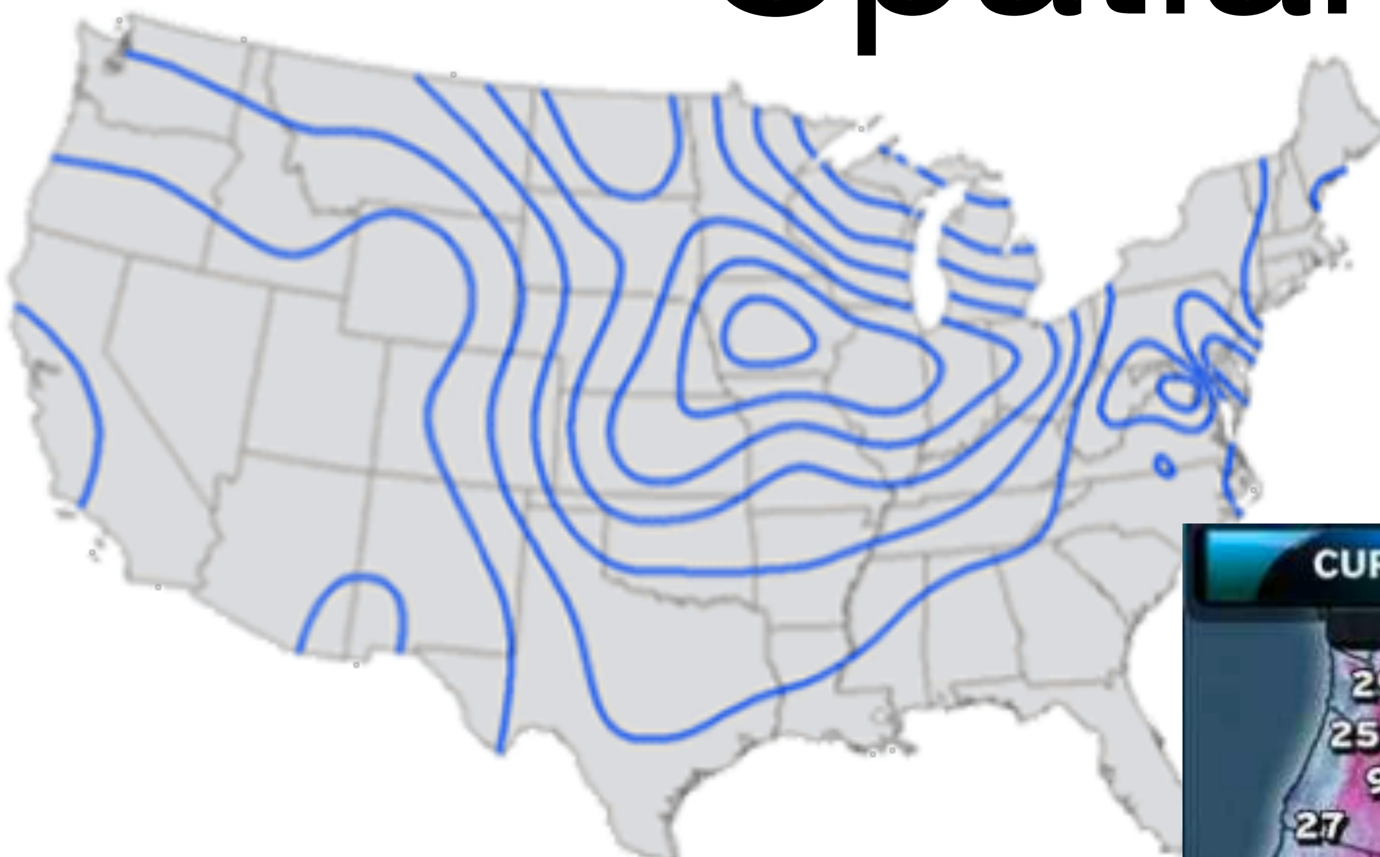
http://christophermanning.org/gists/1703449/#/%5B10%5D50/1/0

Matrix Diagrams

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



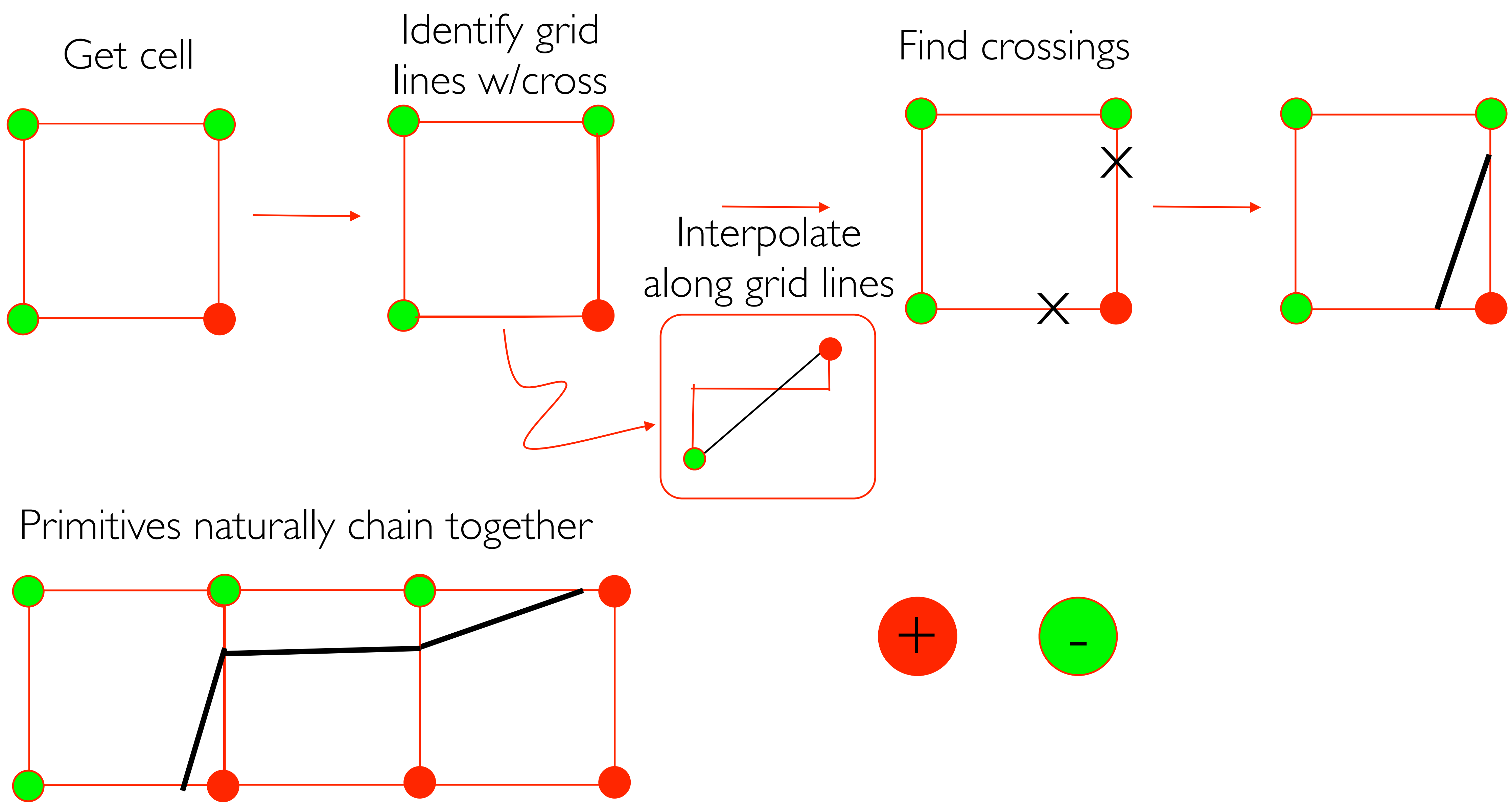
Spatial Data



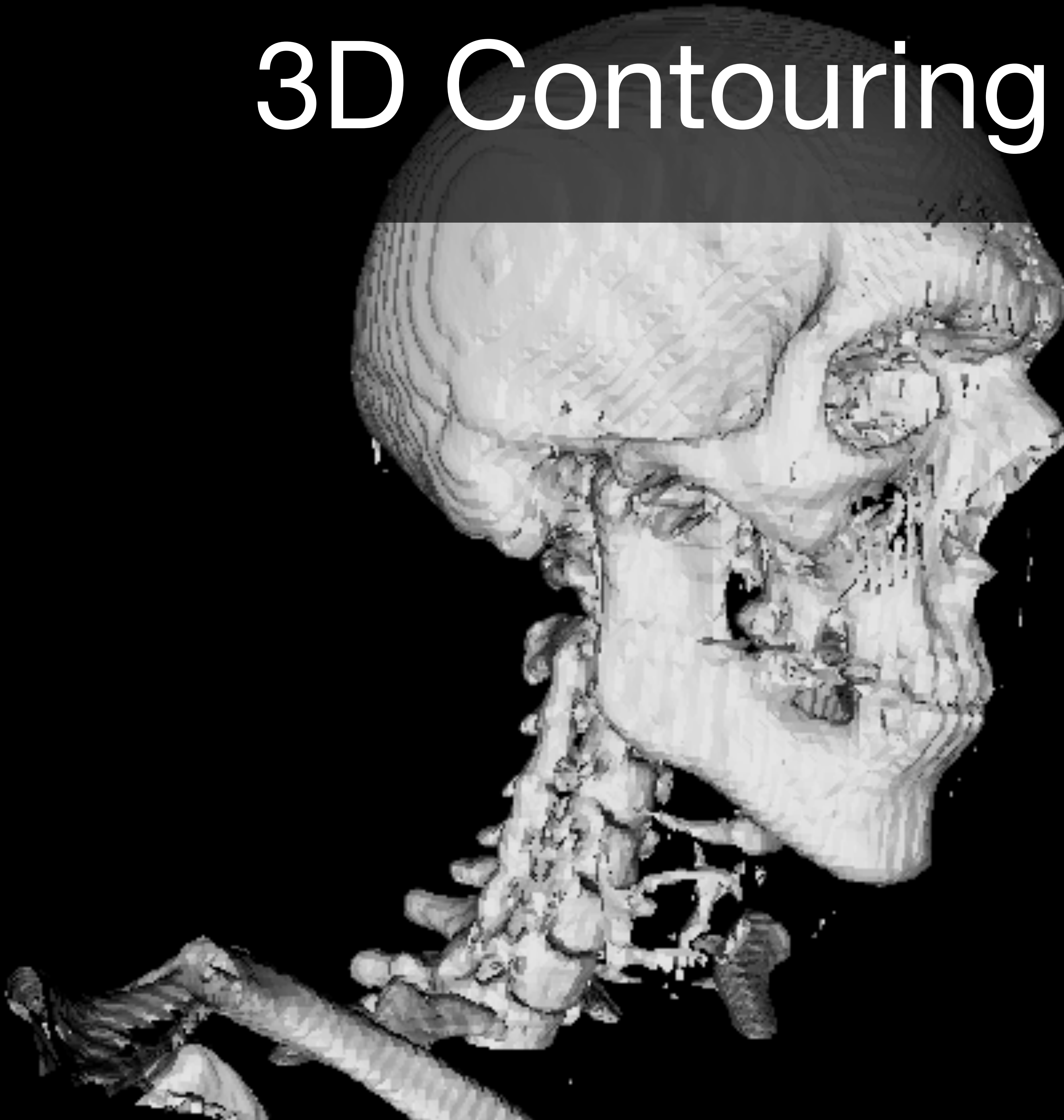
http://ryanhill1.blogspot.com/2011/07/isoline-map.html

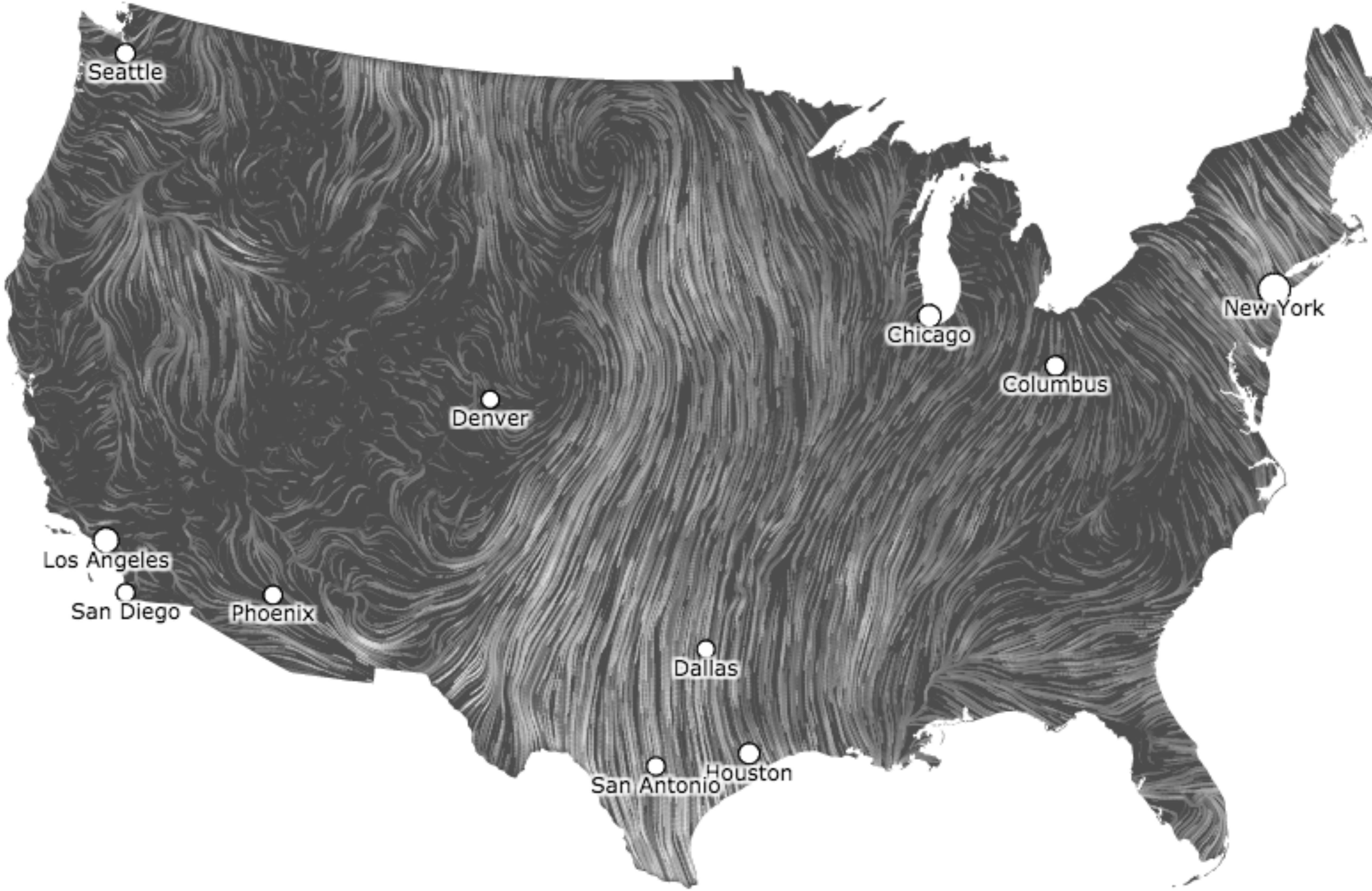
Approach to Contouring in 2D

- 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!