Lecture V - Scientific Visualization

Bruno Gonçalves

www.bgoncalves.com

Twitter places

- As we saw last week, Twitter defines a "coordinates" field in tweets
- There is also a "place" field that we glossed over.
- The **place** object contains also geographical information, but at a courser resolution than the **coordinates** field.
- Each place has a unique place_id, a bounding_box and some geographical information, such as **country** and **full_name**:

```
{'attributes': {},
 'bounding_box': {'coordinates': [[[-74.041878, 40.570842],
    [-74.041878, 40.739434],
    [-73.855673, 40.739434],
    [-73.855673, 40.570842]]],
  'type': 'Polygon'},
 'country': 'United States',
 'country_code': 'US',
 'full_name': 'Brooklyn, NY',
 'id': '011add077f4d2da3',
 'name': 'Brooklyn',
 'place_type': 'city',
 'url': 'https://api.twitter.com/1.1/geo/id/011add077f4d2da3.json'}
```

The bounding_box field is GeoJSON formatted and compatible with pyshp.shape

- places can be of several different types: 'admin', 'city', 'neighborhood', 'poi'

Twitter places

https://dev.twitter.com/overview/api/places

Place Attributes

Place Attributes are metadata about places. An attribute is a key-value pair of arbitrary strings, but with some conventions.

Below are a number of well-known place attributes which may, or may not exist in the returned data. These attributes are provided when the place was created in the Twitter places database.

Key	Description
street_address	
locality	the city the place is in
region	the administrative region the place is in
iso3	the country code
postal_code	in the preferred local format for the place
phone	in the preferred local format for the place, include long distance code
twitter	twitter screen-name, without @
url	official/canonical URL for place
app:id	An ID or comma separated list of IDs representing the place in the applications place database.

Keys can be no longer than 140 characters in length. Values are unicode strings and are restricted to 2000 characters.

Challenge - Filter points and places

- Load each tweet in NYC.json.gz file by using:

```
tweet = eval(line.strip())
```

- on each line and write to Manhattan_places.json.gz all the tweets within Manhattan, as defined by

```
geofiles/nybb_15c/nybb_wgs84.shp
```

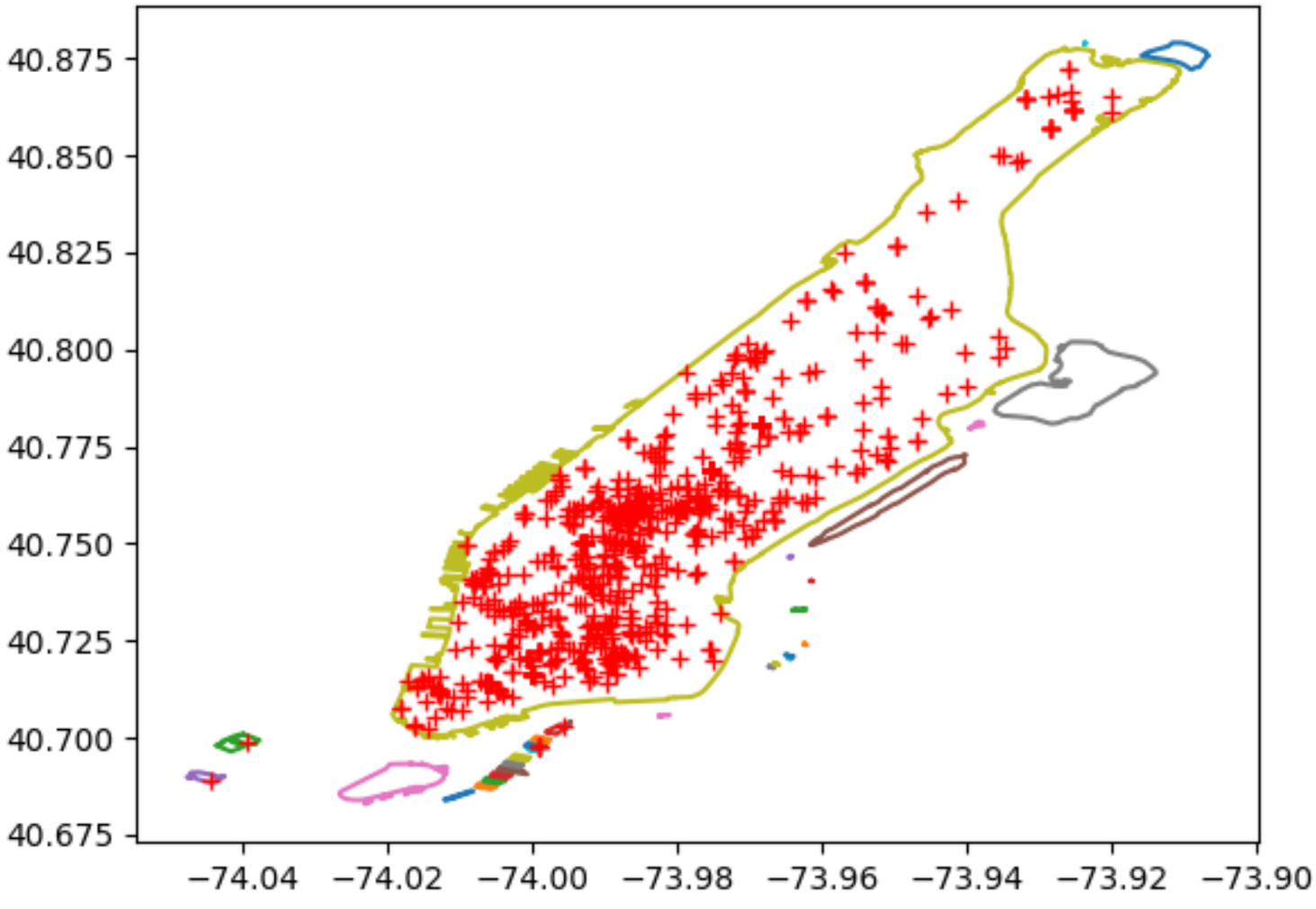
- but now check if the centroid of the 'place' object 'bounding_box' is within Manhattan as well

Challenge - Filter points and places

```
import sys
import shapefile
from shapely.geometry import shape, Point
import gzip

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')

recordDict = dict(zip([record[1] for record in shp.iterRecords()], range(shp.numRecords)))

manhattan = shape(shp.shape(recordDict["Manhattan"]))
fp = gzip.open("Manhattan_places.json.gz", "w")

for line in gzip.open("NYC.json.gz"):
    try:
        tweet = eval(line.strip())
        point = None

        if "coordinates" in tweet and tweet["coordinates"] is not None:
            point = Point(tweet["coordinates"]["coordinates"])
        else:
            if "place" in tweet and tweet["place"]["bounding_box"] is not None:
                bbox = shape(tweet["place"]["bounding_box"])
                point = bbox.centroid

        if point is not None and manhattan.contains(point):
            fp.write(line)
    except:
        pass

fp.close()
```

shapefile_filter_places.py

Challenge - Filter points and places



Challenge - Filter points and places

```
import sys
import gzip
import numpy as np
import shapefile
from shapely.geometry import shape, Point
import matplotlib.pyplot as plt

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')
recordDict = dict(zip([record[1] for record in shp.iterRecords()],
range(shp.numRecords)))

manhattan = shp.shape(recordDict["Manhattan"])

points = np.array(manhattan.points)
parts = manhattan.parts
parts.append(len(manhattan.points))

for i in range(len(parts)-1):
      plt.plot(points.T[0][parts[i]:parts[i+1]], points.T[1]
[parts[i]:parts[i+1]])

points_X = []
points_Y = []

for line in gzip.open(sys.argv[1]):
    try:
        tweet = eval(line.strip())
        point = None

        if "coordinates" in tweet and tweet["coordinates"] is not None:
            point = Point(tweet["coordinates"]["coordinates"])
        else:
            if "place" in tweet and tweet["place"]["bounding_box"] is not
None:
                  bbox = shape(tweet["place"]["bounding_box"])
                  point = bbox.centroid

        if point is not None:
            points_X.append(point.x)
            points_Y.append(point.y)
    except:
        pass

plt.plot(points_X, points_Y, 'r+')

plt.savefig(sys.argv[1] + '.png')
```

@bgoncalves

plot_shapefile_points.py

Calculating distances

https://en.wikipedia.org/wiki/Vincenty%27s_formulae
https://en.wikipedia.org/wiki/Great-circle_distance
https://en.wikipedia.org/wiki/Haversine_formula

- Earlier we saw how to obtain the distance between two points using the Google Maps API.

- But what is the shortest distance between two **arbitrary** points on the surface of the Earth?

- This depends strongly on our model of the Earth:

  - Great Circle - Assumes that the Earth is a perfect sphere of a given radius

    - Usually uses the Haversine formula $\Delta\sigma = 2\arcsin\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cdot \cos\phi_2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)}$

  - Vincenty - Uses a (more) accurate ellipsoid model of the Earth

geopy

https://geopy.readthedocs.io/en/1.10.0/

- geopy provides two different types of functionality
    - geopy.geocoders - a unified interface to several geocoding services (Google Maps, Nominatim, Yahoo, Bing, etc…)
    - geopy.distance - state of the art distance calculations
- We will focus just on the distance module:
    - distance.vincenty(p1, p2) - Calculate the vincenty distance between p1 and p2
    - distance.great_circle(p1, p2) - Calculate the great circle distance between p1 and p2
    - distance.distance(p1, p2) - an alias to distance.vincenty to be used as a default.

geopy

https://geopy.readthedocs.io/en/1.10.0/

- all distance functions return a Distance object.
- the Distance object provides properties that represent the result in different units:
  - .km/.kilometers
  - .m/.meters
  - .mi/.miles
  - .ft/.feet
  - .nm/.nautical
- it also allows us to recalculate the result using different ellipsoids:
  - .set_ellipsoid('elliipsoid')
  - by default WGS-84 is used.

```
                 model             major (km)   minor (km)     flattening
ELLIPSOIDS = {'WGS-84':        (6378.137,    6356.7523142,  1 /
              'GRS-80':        (6378.137,    6356.7523141,  1 /
              'Airy (1830)':   (6377.563396, 6356.256909,   1 /
              'Intl 1924':     (6378.388,    6356.911946,   1 / 297.0),
              'Clarke (1880)': (6378.249145, 6356.51486955, 1 / 293.465),
              'GRS-67':        (6378.1600,   6356.774719,   1 / 298.25),
              }
```

Challenge - geopy

https://geopy.readthedocs.io/en/1.10.0/

- Calculate the distance between

p1 = (41.49008, -71.312796)
p2 = (41.499498, -81.695391)

- in meters, using the **vincenty** and **great_circle** functions.

Challenge - geopy

https://geopy.readthedocs.io/en/1.10.0/

- Calculate the distance between

p1 = (41.49008, -71.312796)
p2 = (41.499498, -81.695391)

- in meters, using the **vincenty** and **great_circle** functions.

```
from geopy import distance

p1 = (41.49008, -71.312796)
p2 = (41.499498, -81.695391)

dist_vincenty = distance.vincenty(p1,
p2).meters
dist_great = distance.great_circle(p1,
p2).meters

print("Vincenty:", dist_vincenty)
print("Great Circles:", dist_great)
```

geopy_distance.py

GIS Data Systems

Vector

- Data types:
    - Points
    - Lines
        - Polygons
- Discrete shapes and boundaries
- Spatial, Database and Network analysis
- Shapefile, GeoJSON, GML, etc…

GIS Data Systems

Vector

- Data types:
  - Points
  - Lines
  - Polygons
- Discrete shapes and boundaries
- Spatial, Database and Network analysis
- Shapefile, GeoJSON, GML, etc…

Raster

- Data types:
  - Cells
  - Pixels
  - Elements
- Dense data, Continuous surfaces
- Spatial Analysis and modeling
- GeoTIFF, ASC, JPEG2000, etc…

ASCII Grid

- Perhaps the simples raster file
- ASCII text based
- A small header

```
ncols         246
nrows         119
xllcorner     -126.500000000000
yllcorner     22.750000000000
cellsize      0.250000000000
NODATA_value  -9999
```

- Followed by rows of numbers
- Very convenient to Read and Write

geofiles/US_pop.asc

Challenge - ASCII Grid

- Write a simple function to load the file:

geofiles/US_pop.asc

- into a numpy array and print the value corresponding to:

40.730503, -74.243251

Challenge

```python
import numpy as np
import matplotlib.pyplot as plt

def map_points(xllcorner, yllcorner, cellsize, nrows, x, y):
    x = int((x-xllcorner)/cellsize)
    y = (nrows-1)-int((y-yllcorner)/cellsize)

    return x, y

fp = open("geofiles/US_pop.asc")
ncols, count = fp.readline().split()
ncols = int(count)
nrows, count = fp.readline().split()
nrows = int(count)
xllcorner, value = fp.readline().split()
xllcorner = float(value)
yllcorner, value = fp.readline().split()
yllcorner = float(value)
cellsize, value = fp.readline().split()
cellsize = float(value)

NODATA_value, value = fp.readline().split()
NODATA_value = float(value)

data = []
for line in fp:
    fields = line.strip().split()
    data.append([float(field) for field in fields])

data = np.array(data)
data[data==NODATA_value] = 0

x = -74.243251
y = 40.730503

coord_x, coord_y = map_points(xllcorner, yllcorner, cellsize, nrows, x, y)
print(data[coord_y, coord_x])
```

asc_load.py

Challenge

```python
import numpy as np
import matplotlib.pyplot as plt

def map_points(xllcorner, yllcorner, cellsize, nrows, x, y):
    x = int((x-xllcorner)/cellsize)
    y = (nrows-1)-int((y-yllcorner)/cellsize)

    return x, y

fp = open("geofiles/US_pop.asc")
ncols, count = fp.readline().split()
ncols = int(count)
nrows, count = fp.readline().split()
nrows = int(count)
xllcorner, value = fp.readline().split()
xllcorner = float(value)
yllcorner, value = fp.readline().split()
yllcorner = float(value)
cellsize, value = fp.readline().split()
cellsize = float(value)

NODATA_value, value = fp.readline().split()
NODATA_value = float(value)

data = []
for line in fp:
    fields = line.strip().split()
    data.append([float(field) for field in fields])

data = np.array(data)
data[data==NODATA_value] = 0

x = -74.243251
y = 40.730503

coord_x, coord_y = map_points(xllcorner, yllcorner, cellsize, nrows, x, y)
print(data[coord_y, coord_x])
```

asc_load.py

Challenge - ASCII Grid

asc_load.py

ASCII Grid

- This type of grid is a very convenient way to aggregate spatial data.
- Simply map **lat, lon** pairs to matrix entries and then increment the values
- All we need is to define the **bbox** we are interested in, and the size of each cell and create a matrix with that shape.

```
import numpy as np
from shapely.geometry import shape, Point
import shapefile

shp = shapefile.Reader('../Lecture IV/geofiles/nybb_15c/nybb_wgs84.shp')
recordDict = dict(zip([record[1] for record in shp.iterRecords()],
range(shp.numRecords)))

manhattan = shp.shape(recordDict["Manhattan"])

xllcorner, yllcorner, xurcorner, yurcorner = manhattan.bbox

cellsize = 0.01

ncols = int((xurcorner-xllcorner)/cellsize)
nrows = int((yurcorner-yllcorner)/cellsize)

data = np.zeros((nrows, ncols), dtype='int')

print(data.shape)
```

generate_matrix.py

Challenge - Aggregate

- Rewrite

shapefile_filter_places.py

- so that it now counts how many tweets happen in each cell of width 0.01

Challenge - Aggregate

```python
import sys
import numpy as np
import shapefile
from shapely.geometry import shape, Point
import matplotlib.pyplot as plt
import gzip

def map_points(xllcorner, yllcorner, cellsize, nrows, x, y):
    x = int((x-xllcorner)/cellsize)
    y = (nrows-1)-int((y-yllcorner)/cellsize)

    return x, y
```

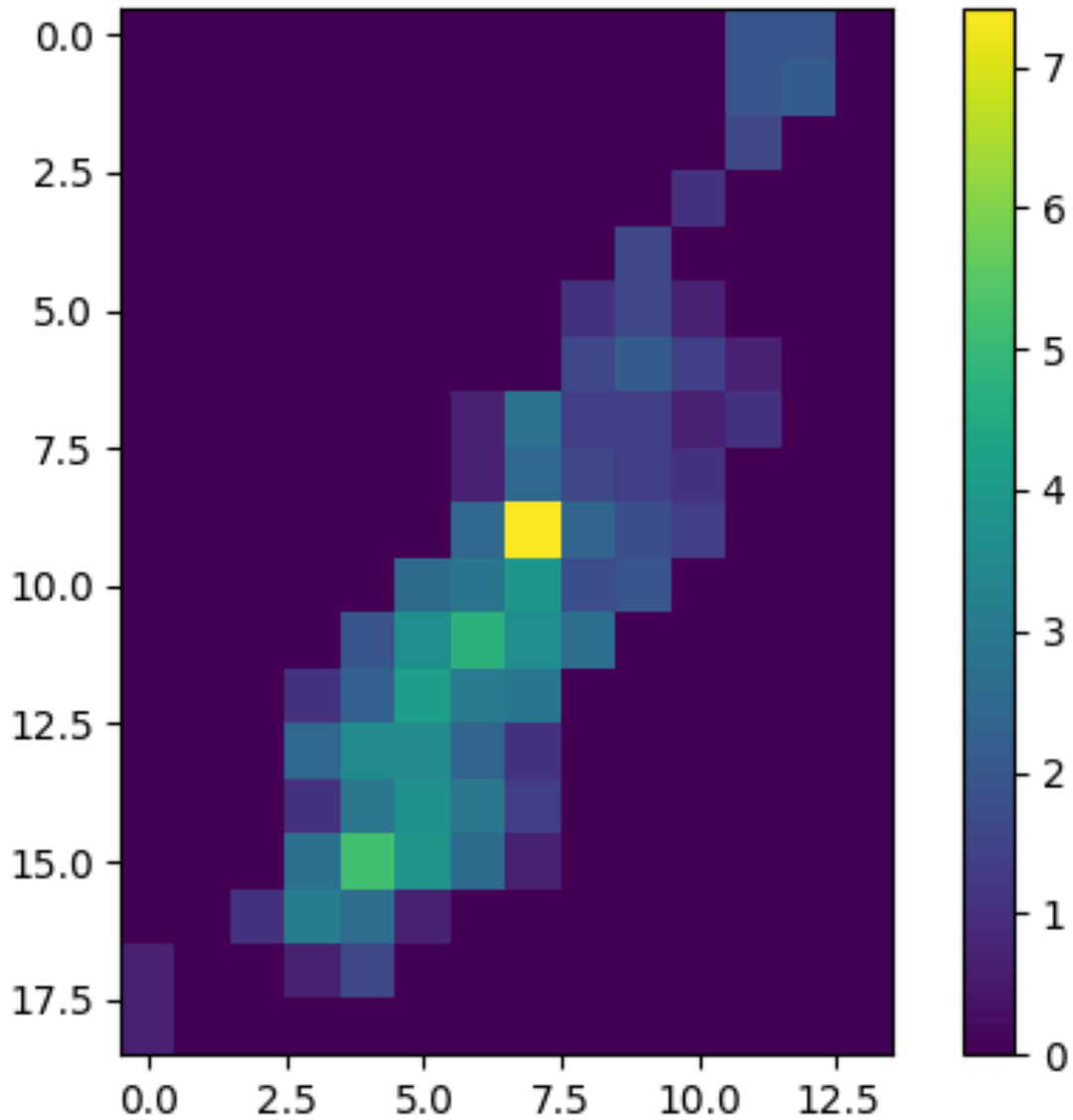
```python
def save_asc(data, xllcorner, yllcorner, cellsize, filename):
    fp = open(filename, "w")

    nrows, ncols = data.shape

    print("ncols", ncols, file=fp)
    print("nrows", nrows, file=fp)
    print("xllcorner", xllcorner, file=fp)
    print("yllcorner", yllcorner, file=fp)
    print("cellsize", cellsize, file=fp)
    print("NODATA_value", "-9999", file=fp)

    for i in range(nrows):
        for j in range(ncols):
            print(("%u " % data[i, j]), end="", file=fp)

        print("\n", end="", file=fp)

    fp.close()
```

shapefile_filter_aggregate.py

```
shp = shapefile.Reader('../Lecture IV/geofiles/nybb_15c/nybb_wgs84.shp')
recordDict = dict(zip([record[1] for record in shp.iterRecords()], range(shp.numRecords)))
manhattan = shape(shp.shape(recordDict["Manhattan"]))

xllcorner, yllcorner, xurcorner, yurcorner = manhattan.bounds
cellsize = 0.01

ncols = int((xurcorner-xllcorner)/cellsize)
nrows = int((yurcorner-yllcorner)/cellsize)

data = np.zeros((nrows, ncols), dtype='int')

for line in gzip.open("../Lecture IV/NYC.json.gz"):
    try:
        tweet = eval(line.strip())
        point = None

        if "coordinates" in tweet and tweet["coordinates"] is not None:
            point = Point(tweet["coordinates"]["coordinates"])
        else:
            if "place" in tweet and tweet["place"]["bounding_box"] is not None:
                bbox = shape(tweet["place"]["bounding_box"])
                point = bbox.centroid

        if point is not None and manhattan.contains(point):
            coord_x, coord_y = map_points(xllcorner, yllcorner, cellsize, nrows, point.x, point.y)
            data[coord_y, coord_x] += 1

    except:
        pass


save_asc(data, xllcorner, yllcorner, cellsize, "Manhattan.asc")

plt.imshow(np.log(data+1))
plt.colorbar()
plt.savefig('Manhattan_cells.png')
```

shapefile_filter_aggregate.py

Challenge - Aggregate

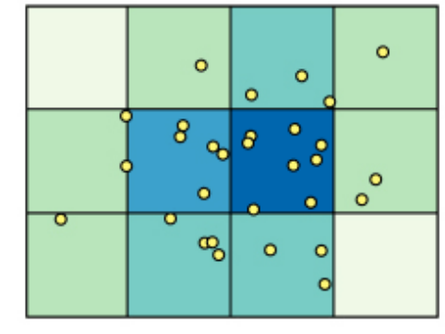


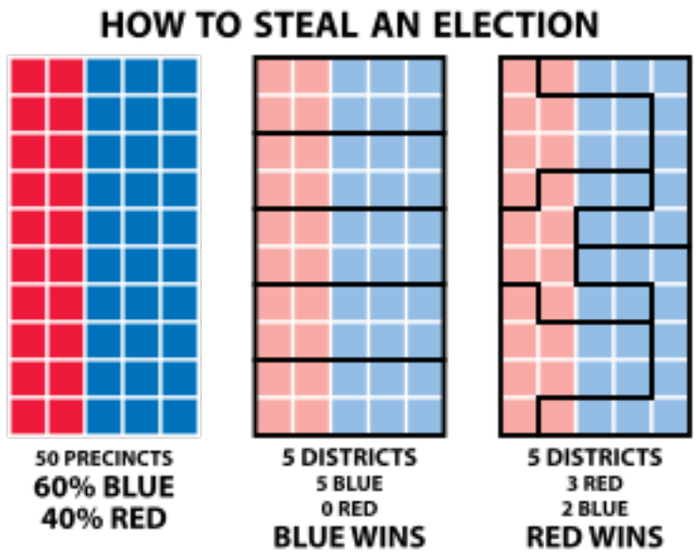
Modifiable Areal Unit Problem

- Different aggregations can result in differing results.

- The aggregation level must be chosen in a way that is appropriate to the problem and requires experience with the type of dataset and problem.

- Sometimes abused for political reasons.



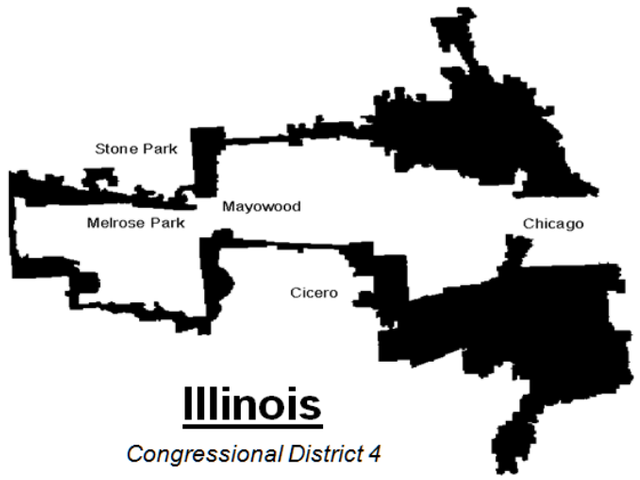


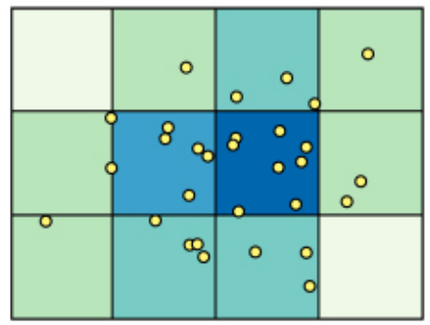
Modifiable Areal Unit Problem

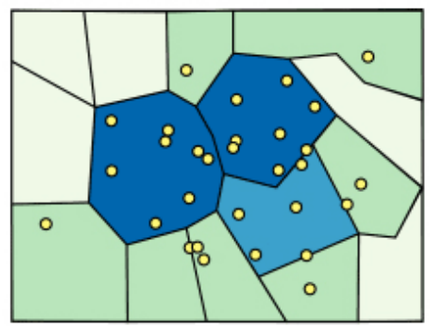
- Different aggregations can result in differing results.

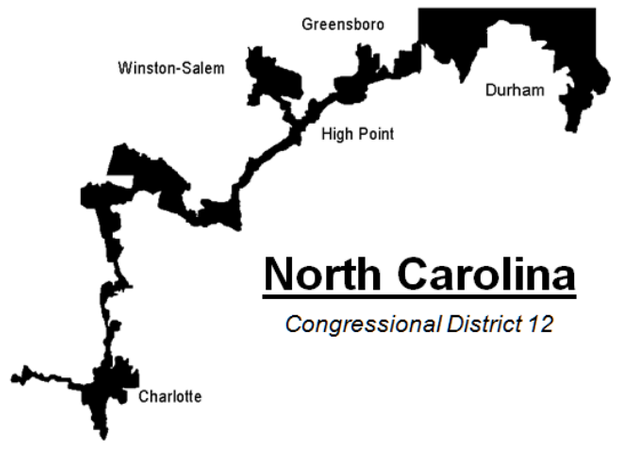
- The aggregatic
  the problem ar
  problem.

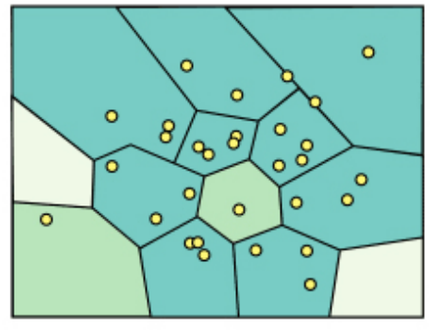
- Sometimes ab









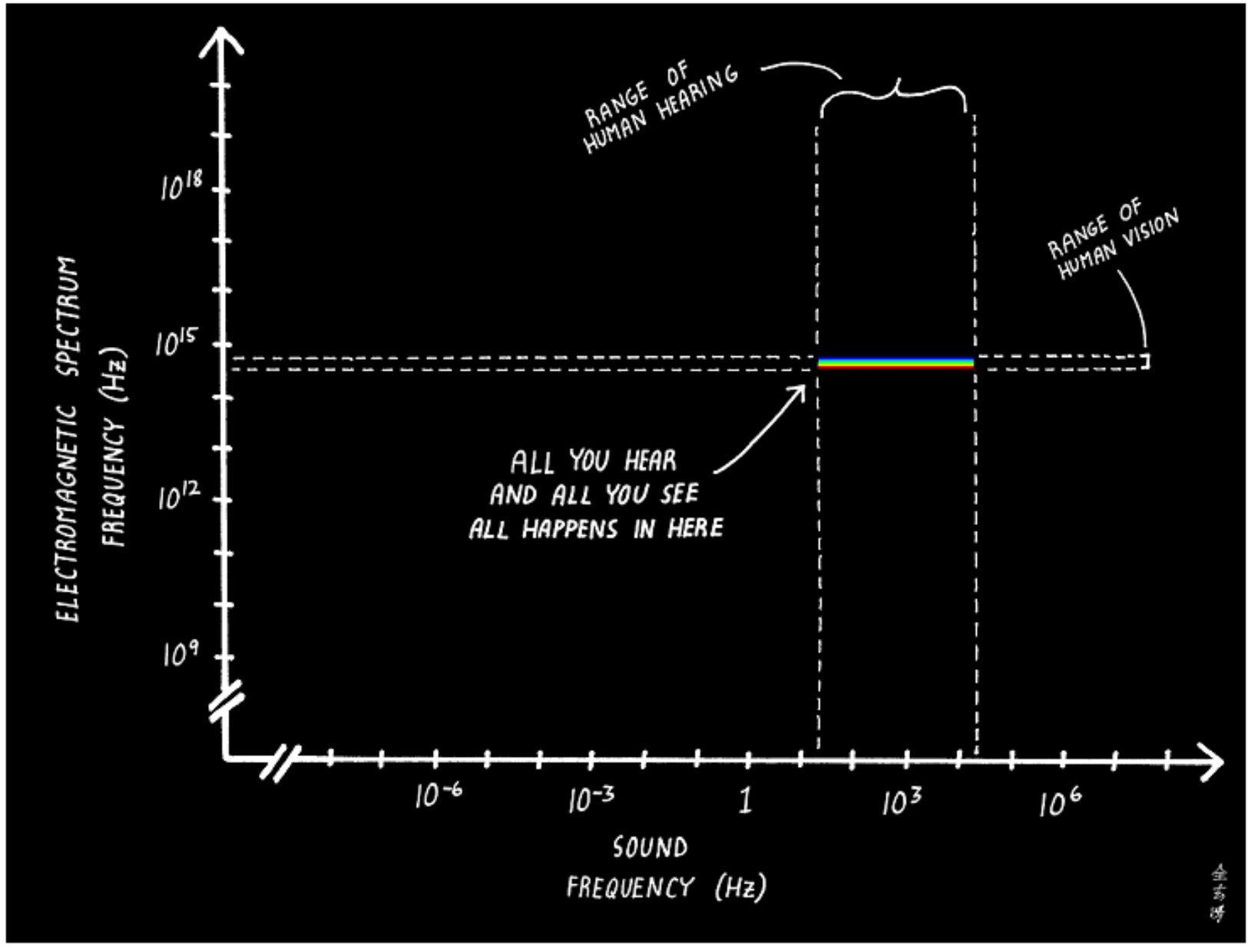


...graphics from 1990s Supreme Court Redistricting Decisions, Peter S. Wattson
www.senate.leg.state.mn.us/departments/scr/REDIST/red907.htm

Human Perception

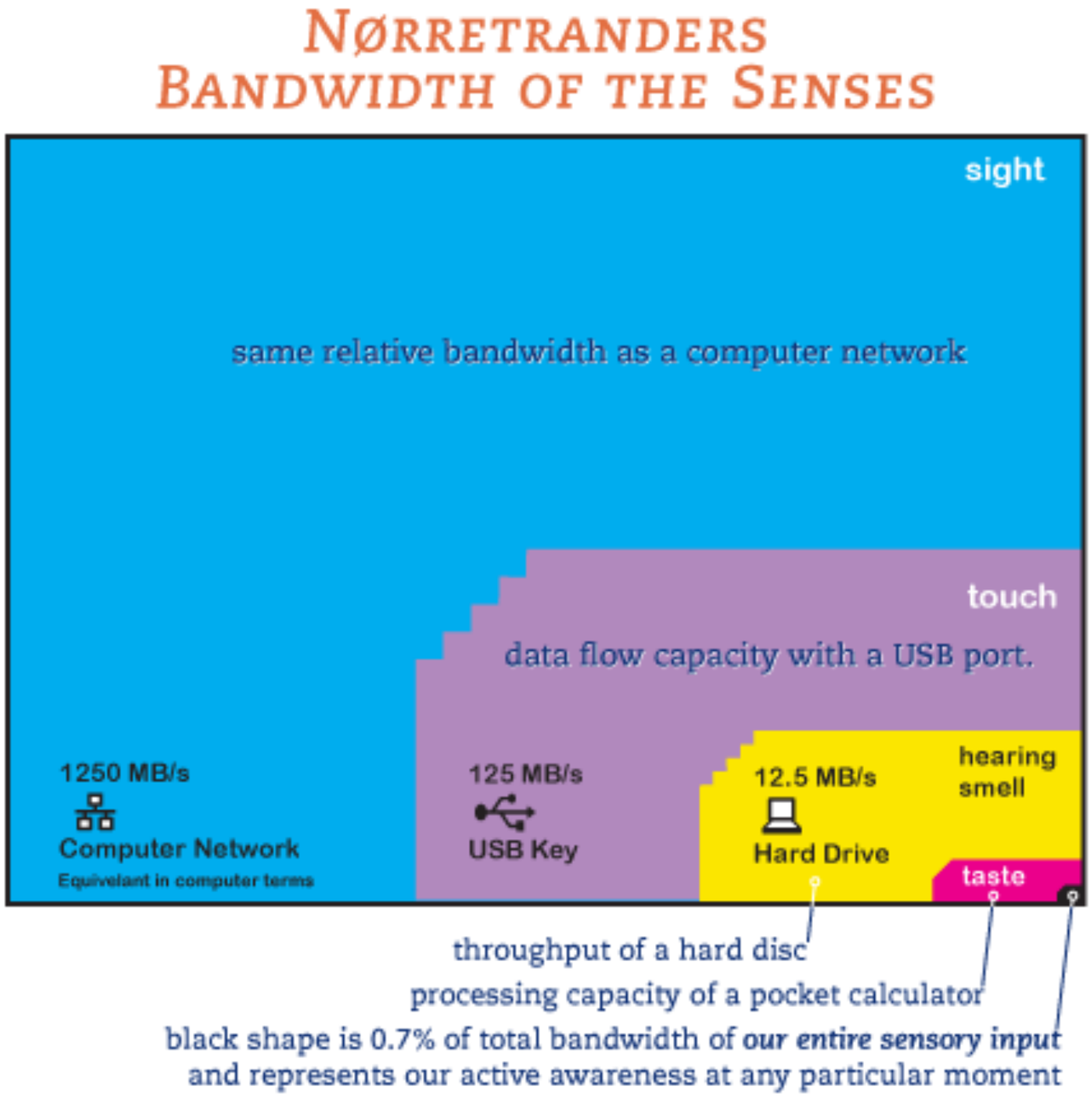
Human Perception

http://abstrusegoose.com/421



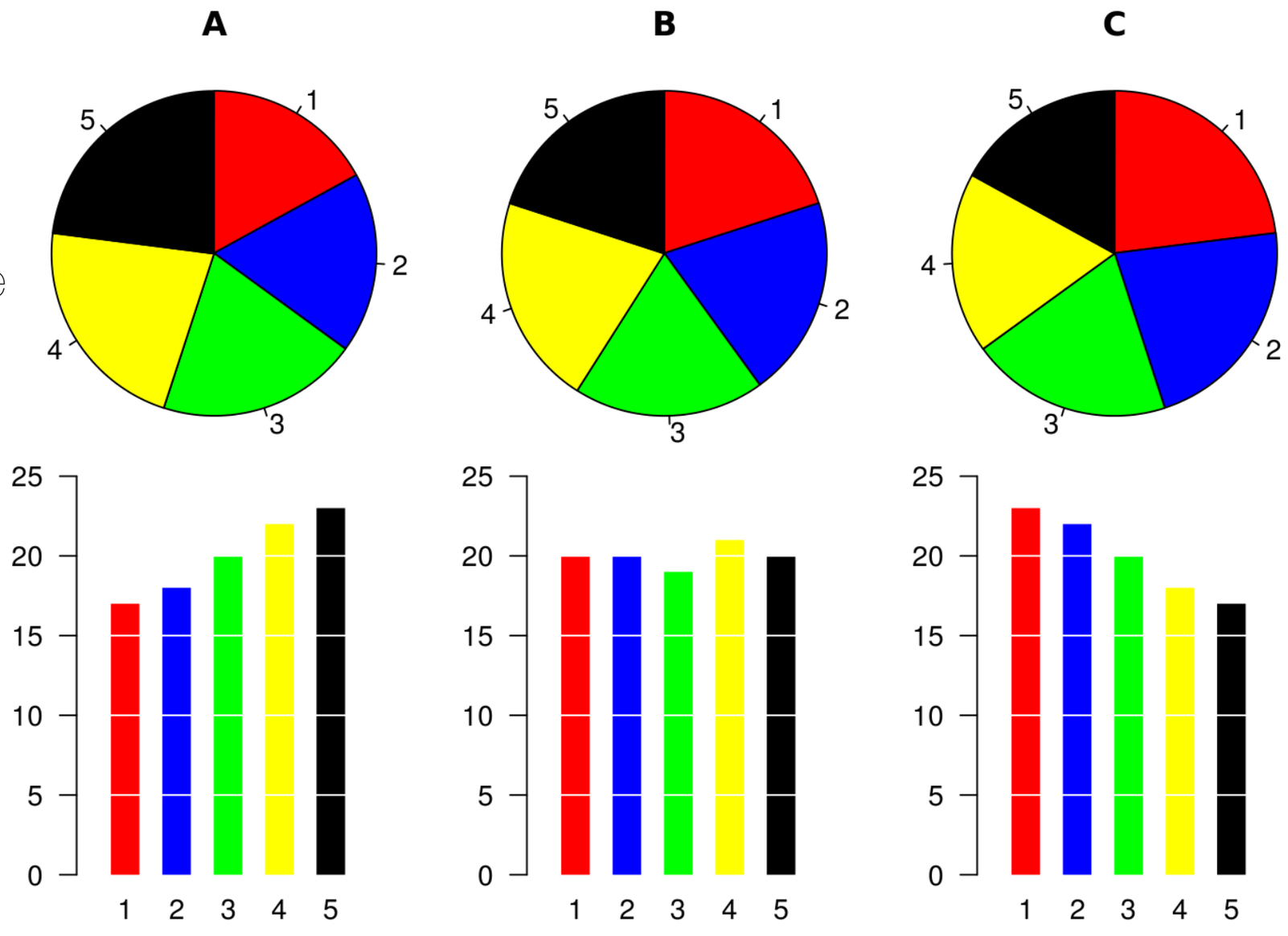
In the grand scheme of things, we're all pretty much blind and deaf.

Human Senses



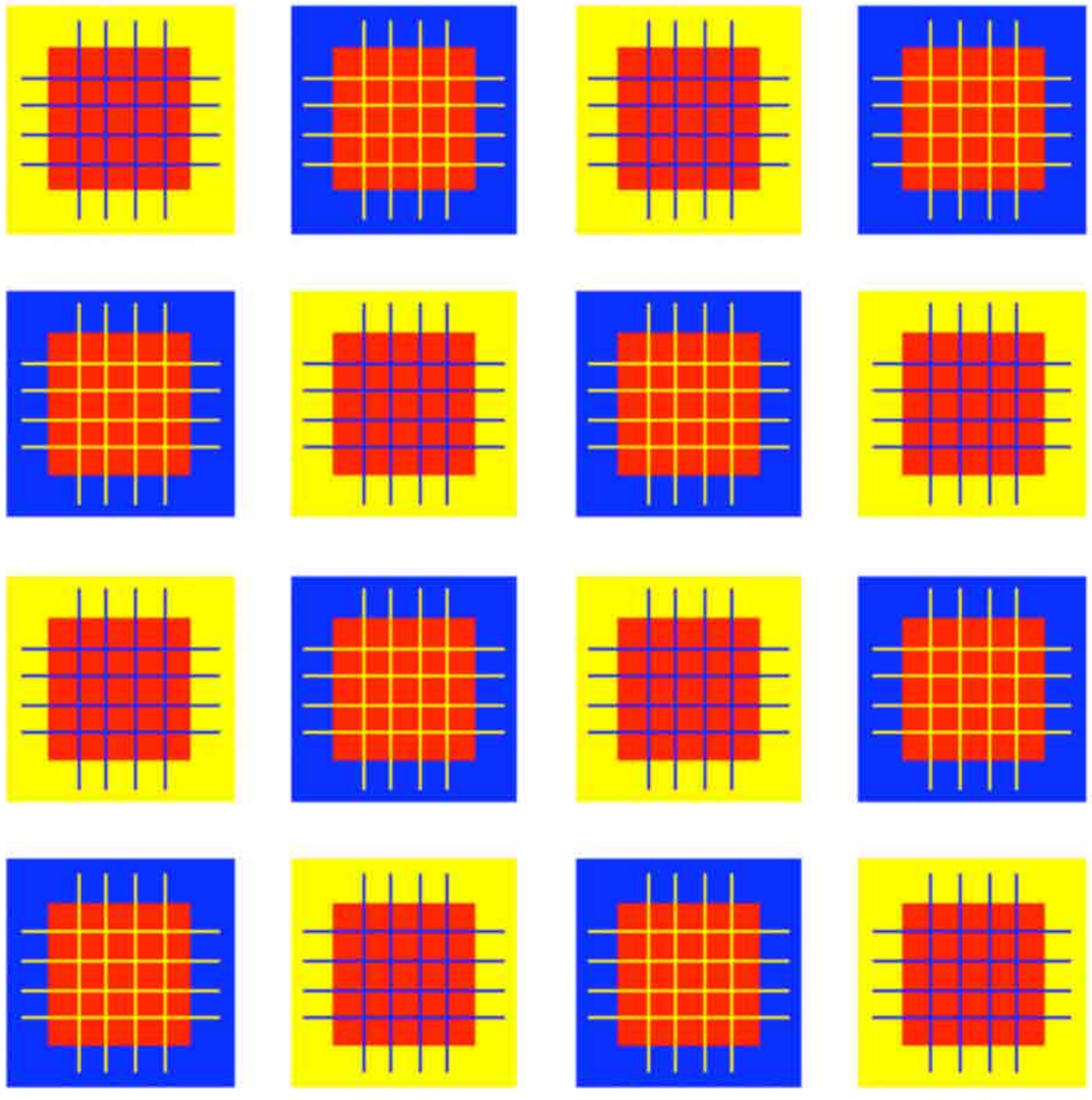
Perception

- Some cognitive tasks are significantly easier than others. In order, we are good a distinguishing:
  - Position, length
  - Direction, Angle, Area
  - Volume, Curvature, Shade



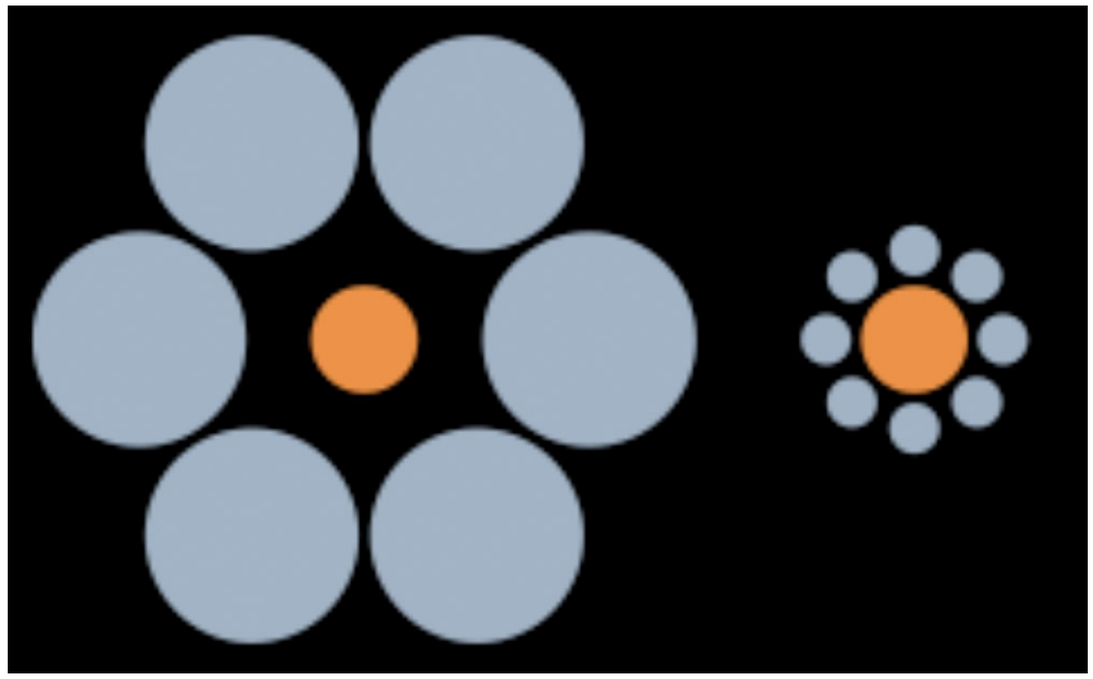
Perception

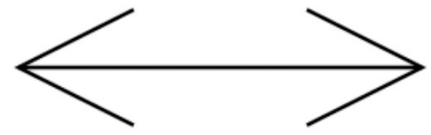
- Some cognitive tasks are significantly easier than others. In order, we are good a distinguishing:
  - Position, length
  - Direction, Angle, Area
  - Volume, Curvature, Shade
  - Color Saturation.

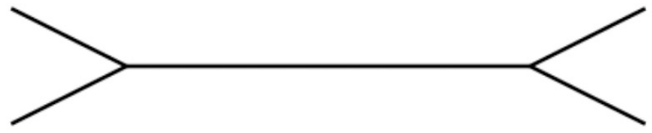


Perception

- Some cognitive tasks are significantly easier than others. In order, we are good a distinguishing:
    - Position, length
    - Direction, Angle, Area
    - Volume, Curvature, Shade
    - Color Saturation.
- Context also matters!
    - An object seen in the context of larger objects will appear smaller, while in the content of smaller objects it will appear larger.

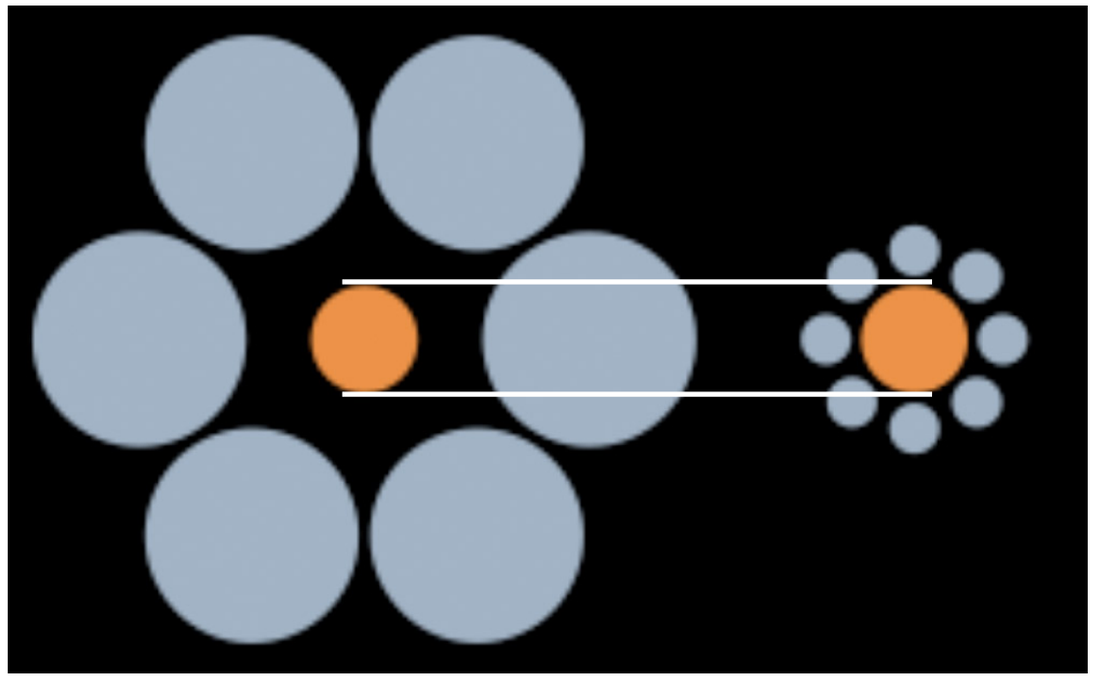


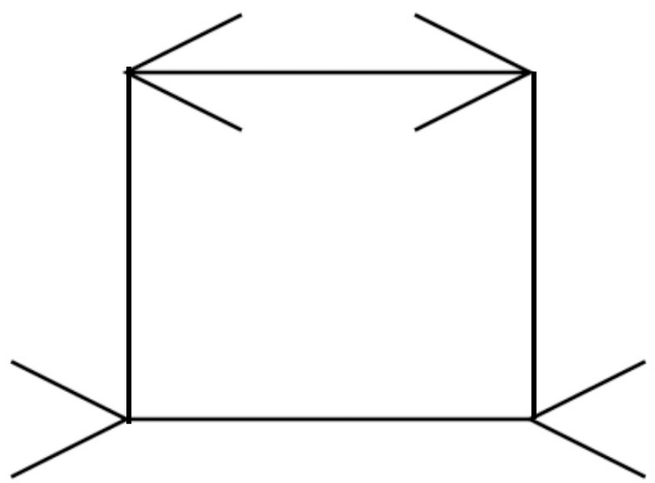




Perception

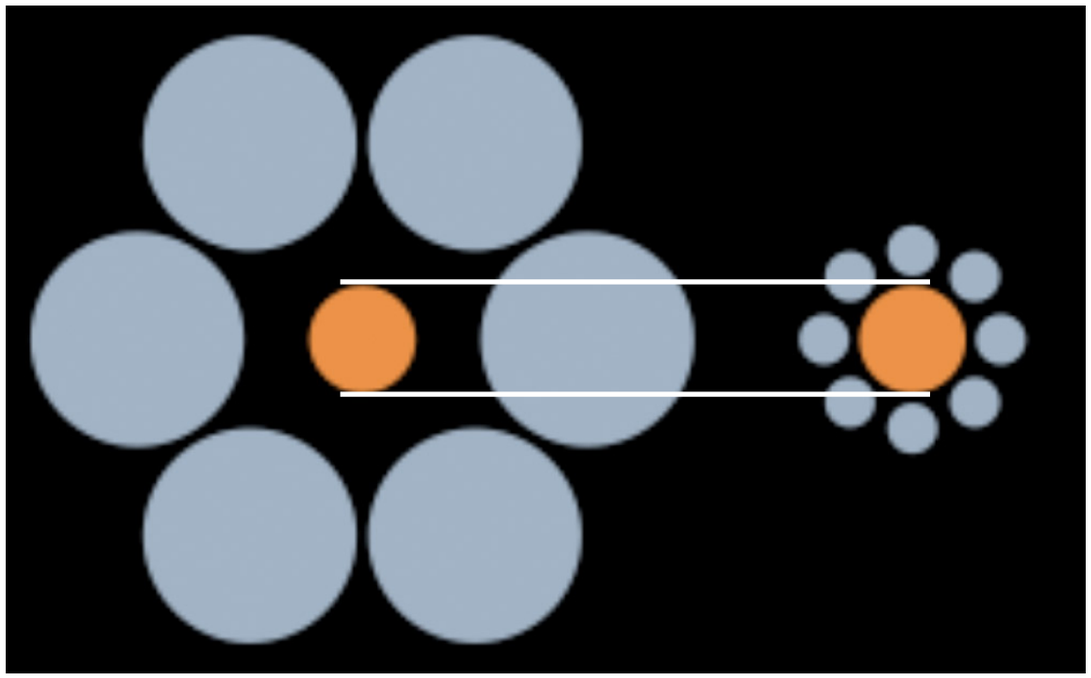
- Some cognitive tasks are significantly easier than others. In order, we are good a distinguishing:
    - Position, length
    - Direction, Angle, Area
    - Volume, Curvature, Shade
    - Color Saturation.
- Context also matters!
    - An object seen in the context of larger objects will appear smaller, while in the content of smaller objects it will appear larger.

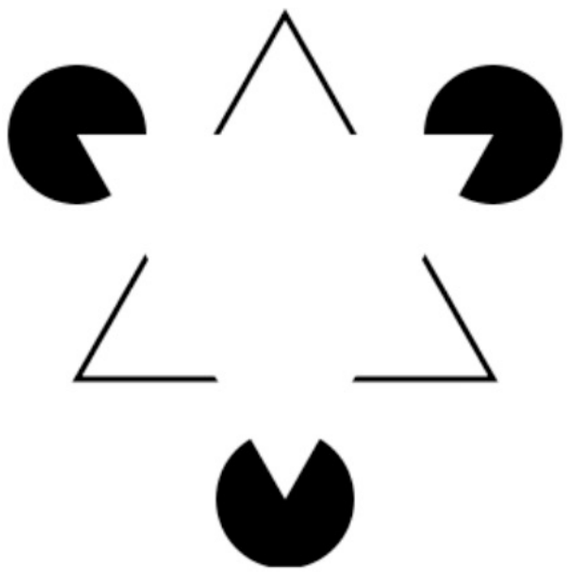


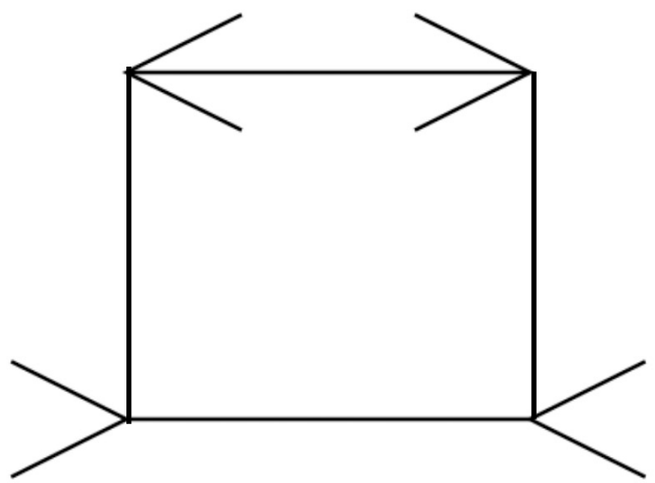


Perception

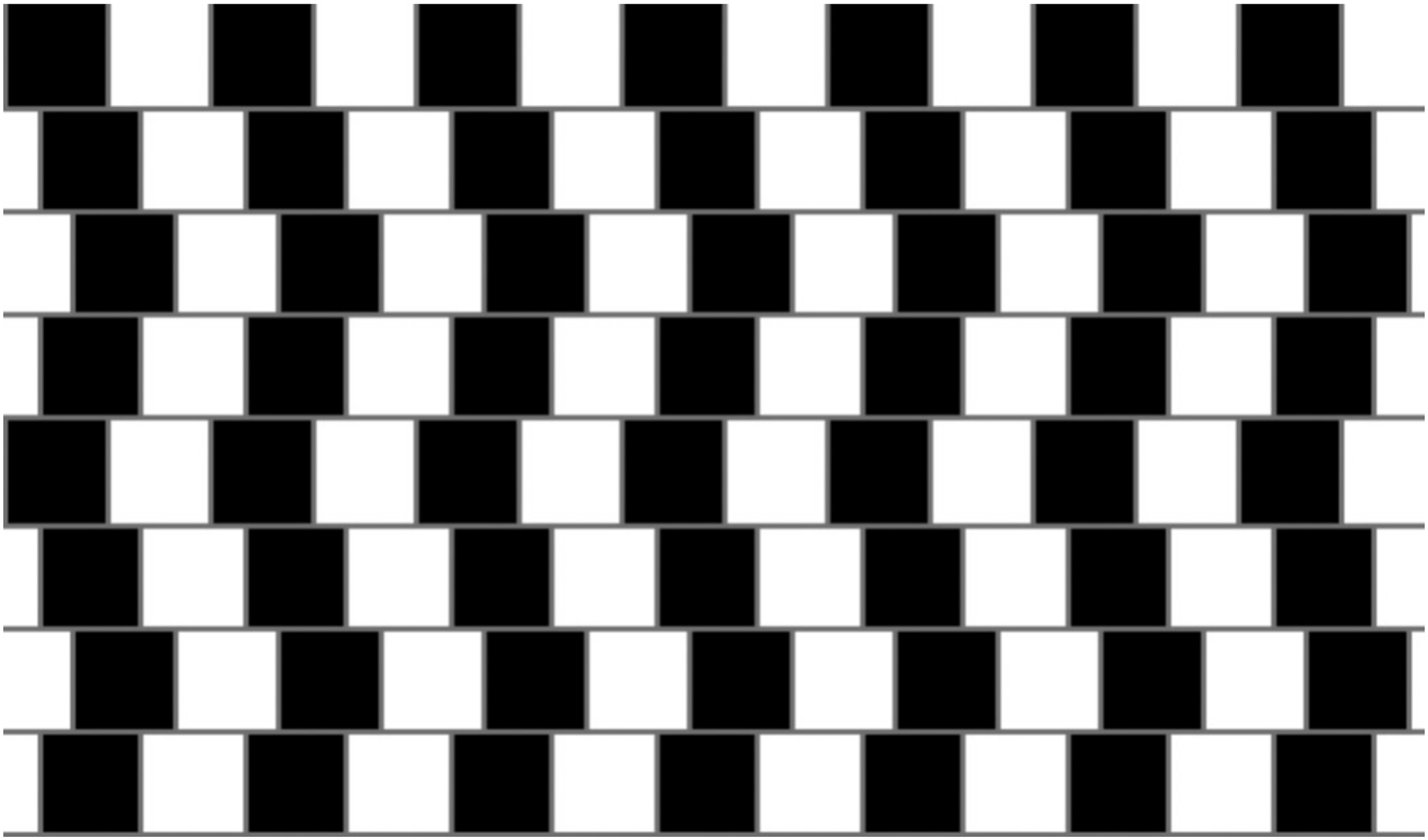
- Some cognitive tasks are significantly easier than others. In order, we are good a distinguishing:
  - Position, length
  - Direction, Angle, Area
  - Volume, Curvature, Shade
  - Color Saturation.
- Context also matters!
  - An object seen in the context of larger objects will appear smaller, while in the content of smaller objects it will appear larger.
  - And we “fill in the gaps”





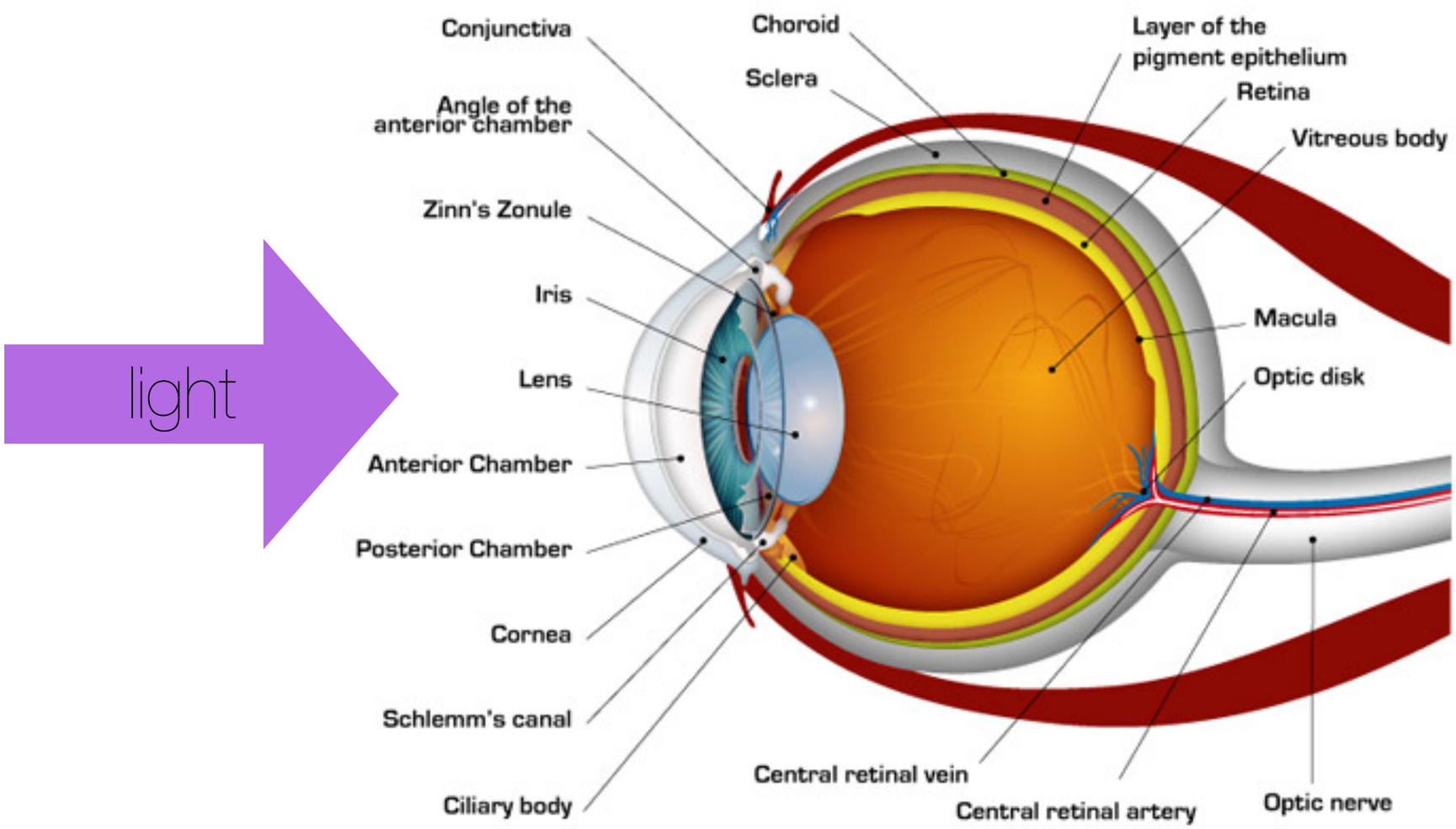


Perception Biases



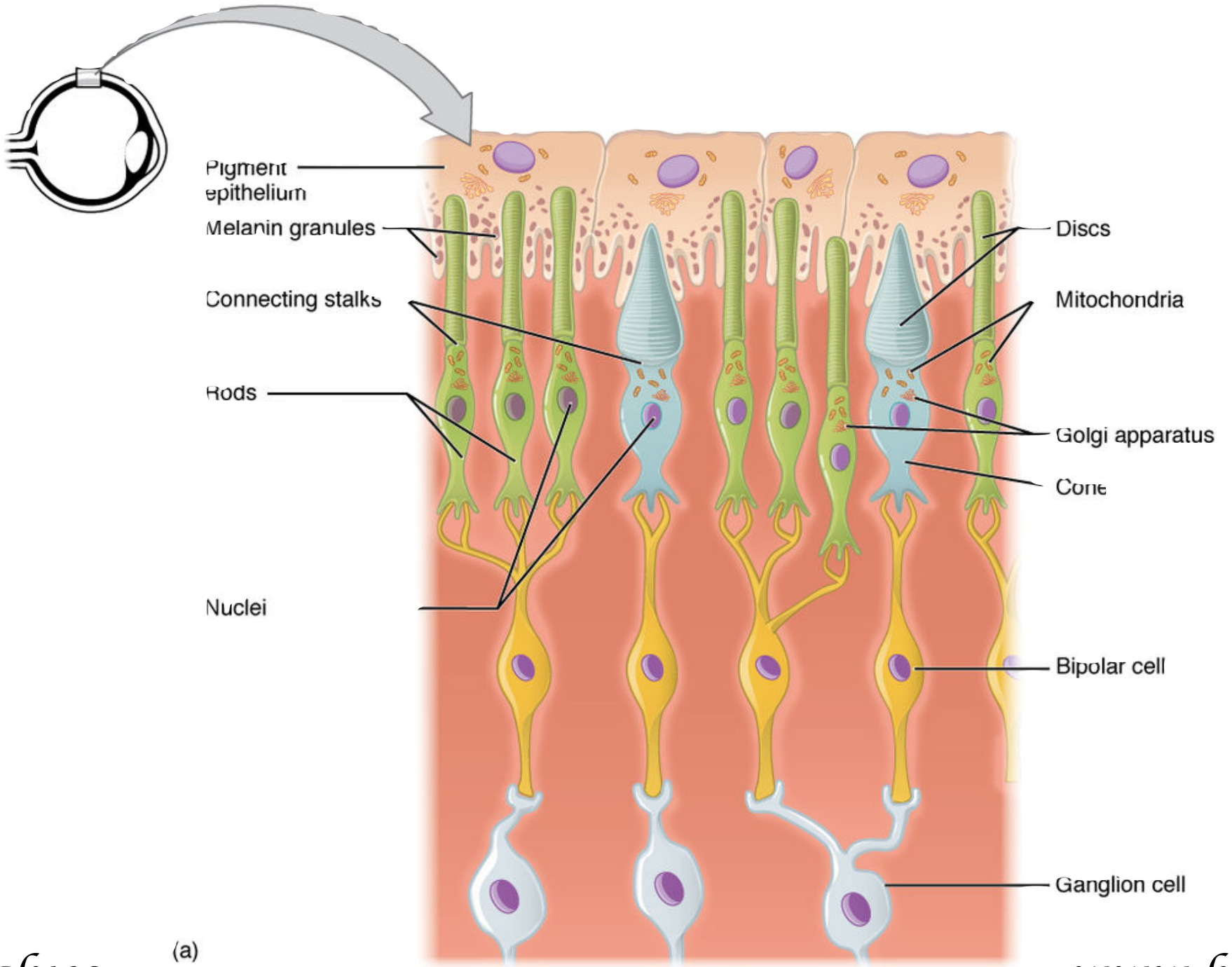
Human Vision

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



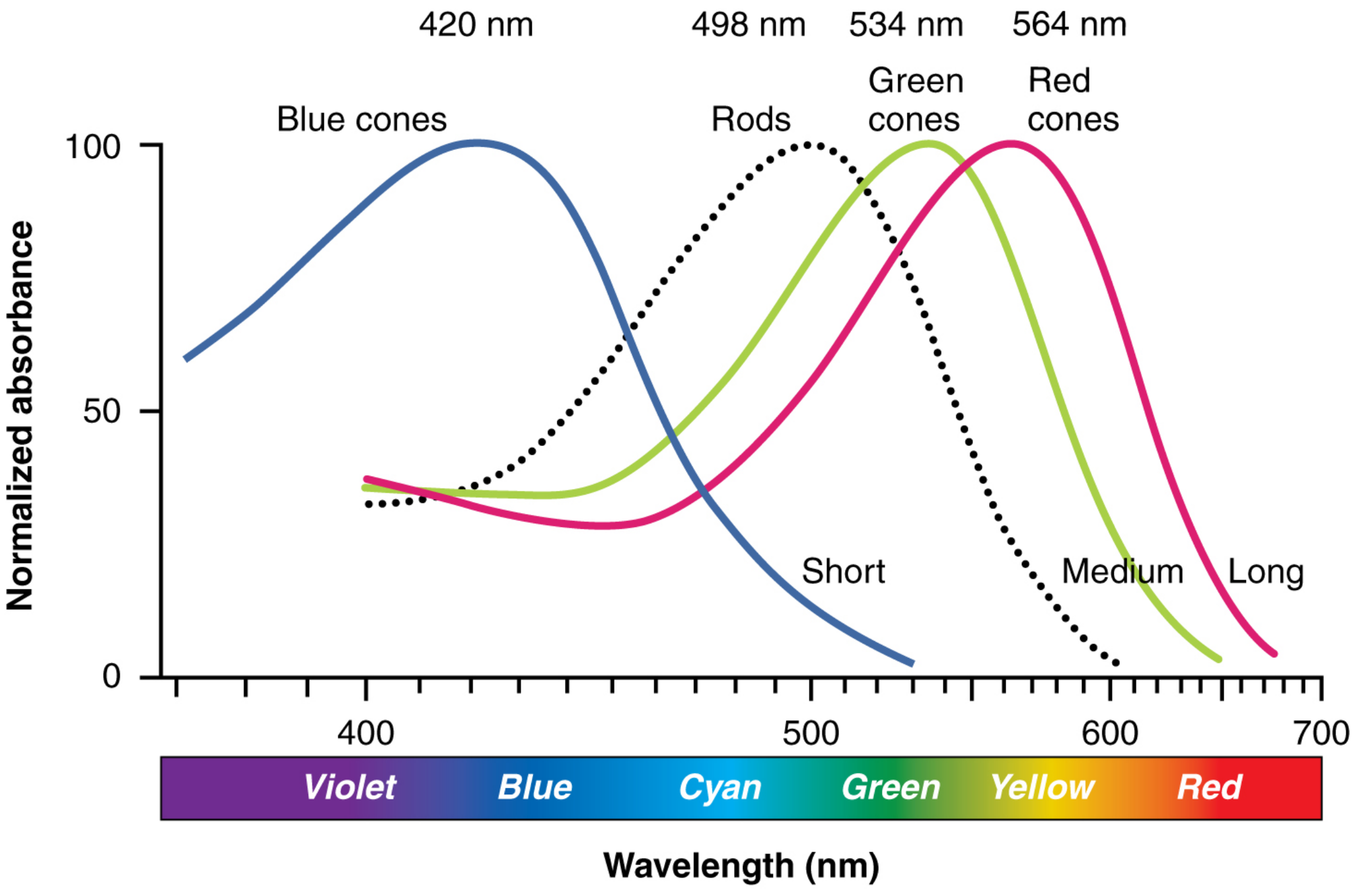
Human Vision

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

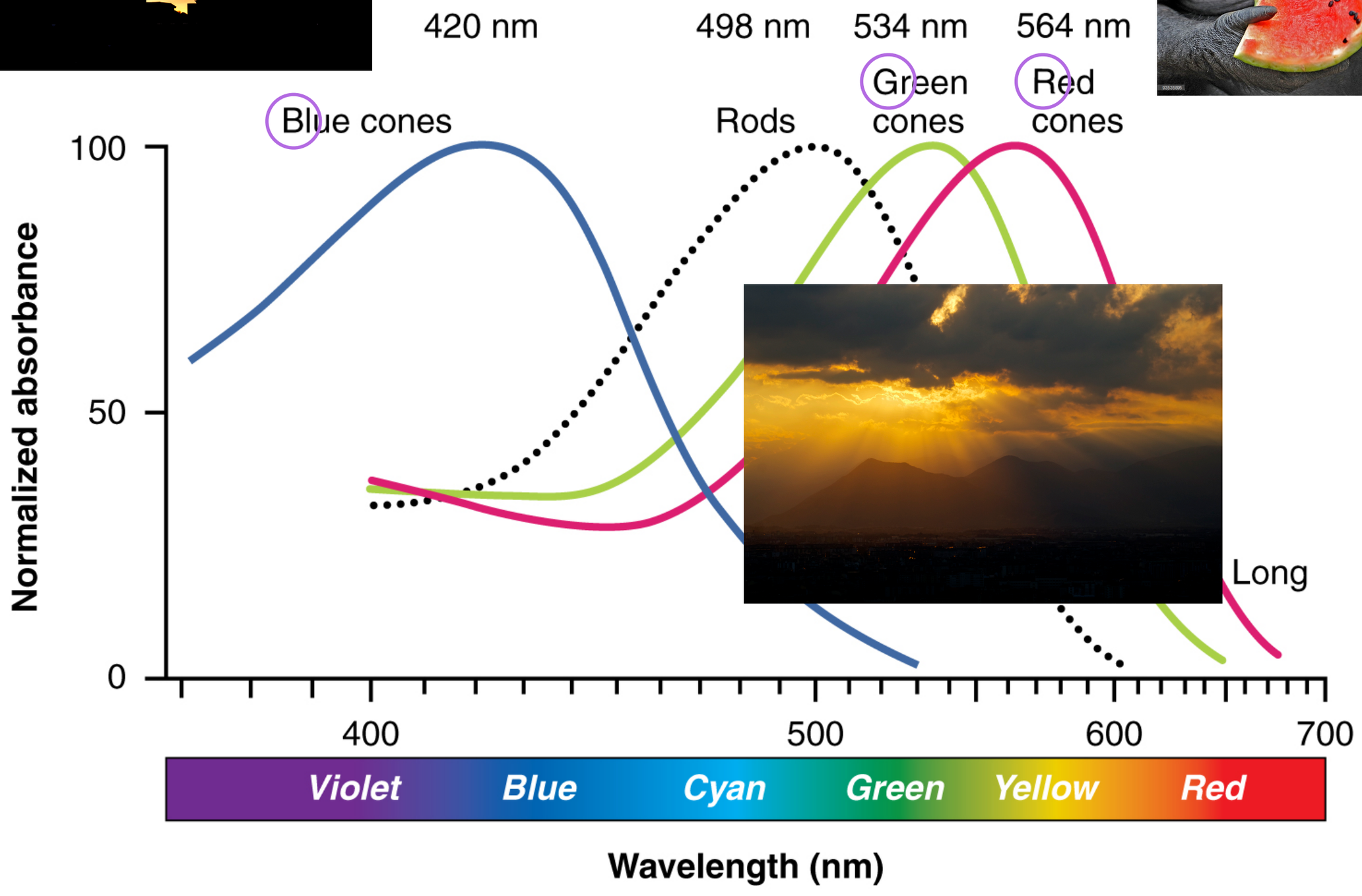


Human Vision

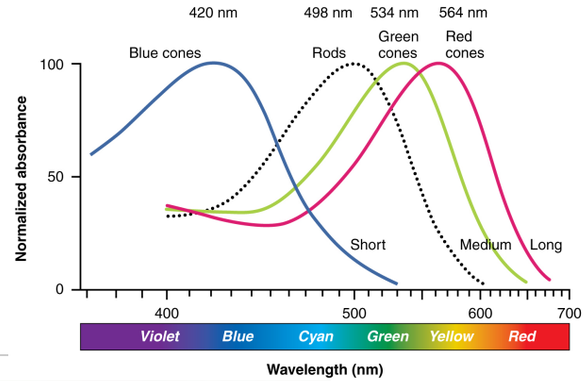
https://en.wikipedia.org/wiki/Photoreceptor_cell



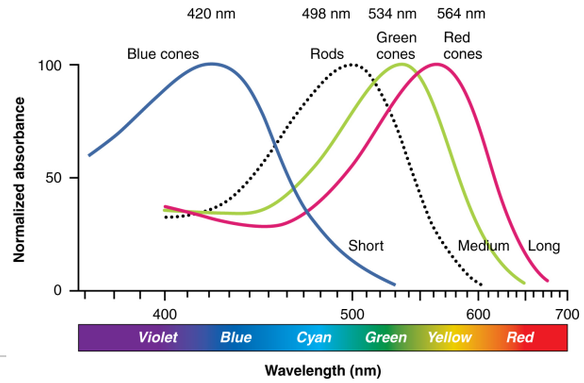
https://en.wikipedia.o ell

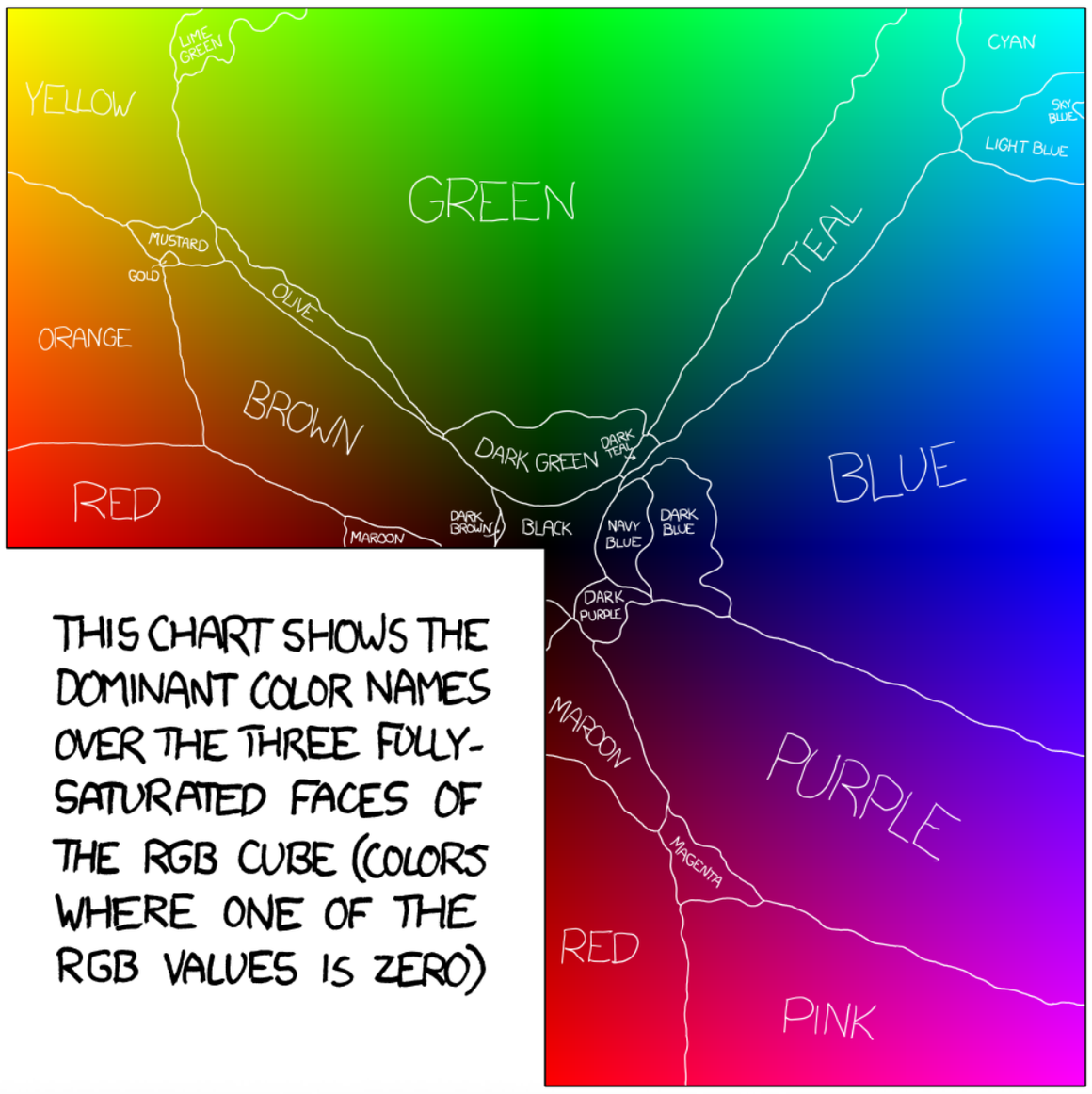


Colors galore!



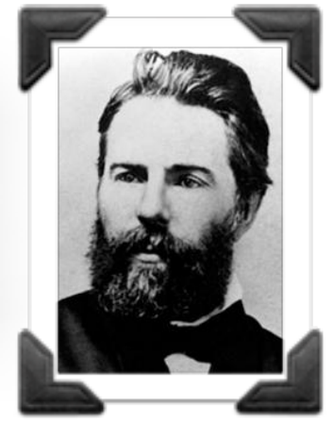
Color Perception





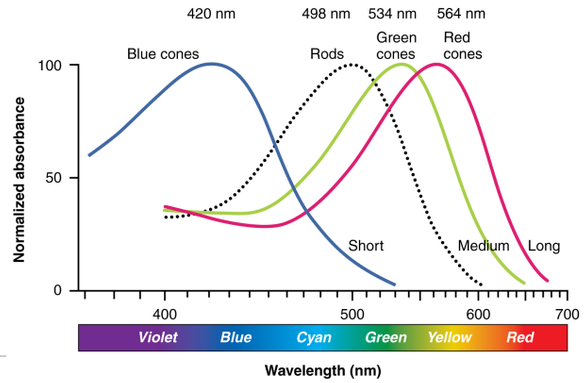
THIS CHART SHOWS THE DOMINANT COLOR NAMES OVER THE THREE FULLY-SATURATED FACES OF THE RGB CUBE (COLORS WHERE ONE OF THE RGB VALUES IS ZERO)

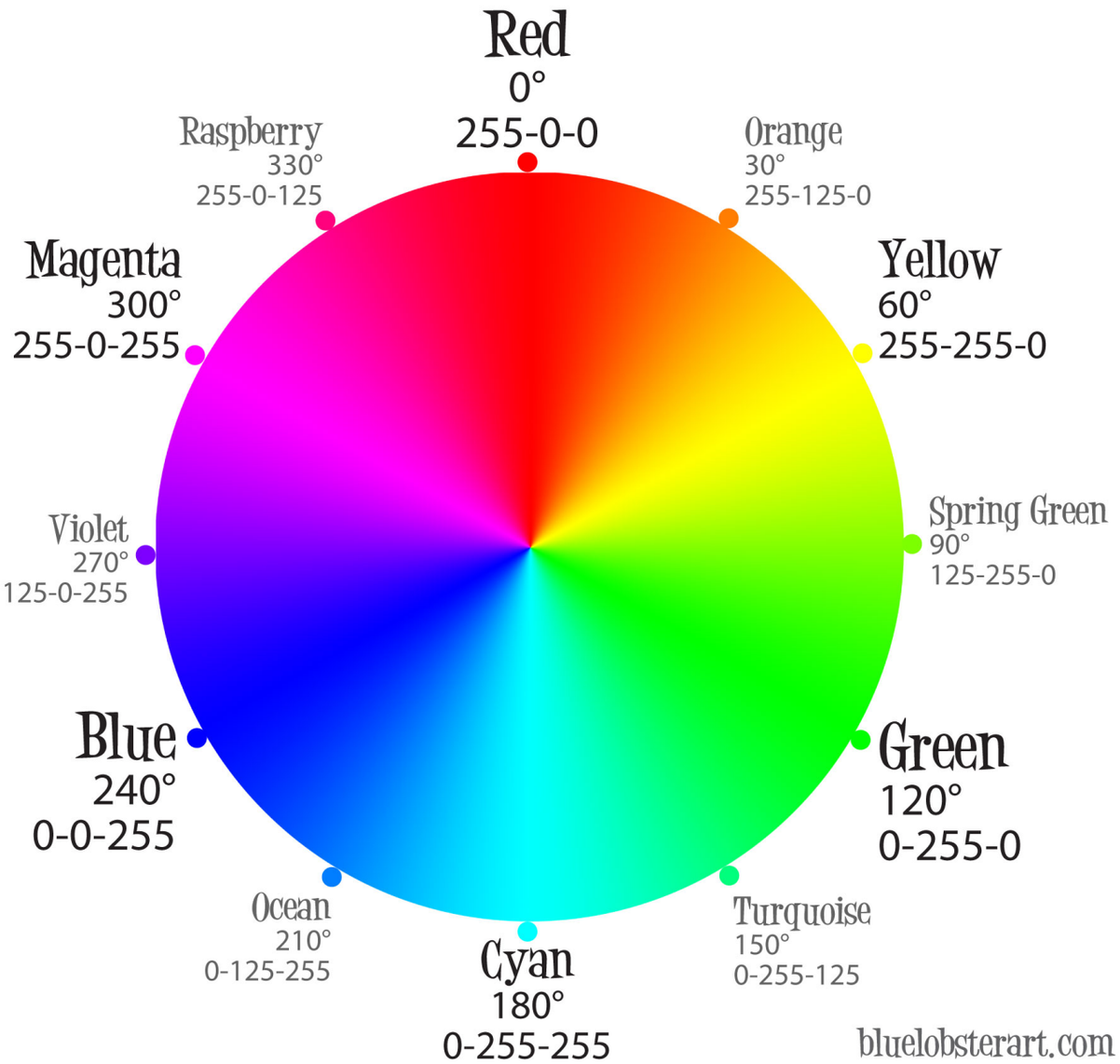
"Who in the rainbow can draw the line where the violet tint ends and the orange tint begins? Distinctly we see the difference of the colors, but where exactly does the one first blendingly enter into the other? So with sanity and insanity."
(H. Melville)

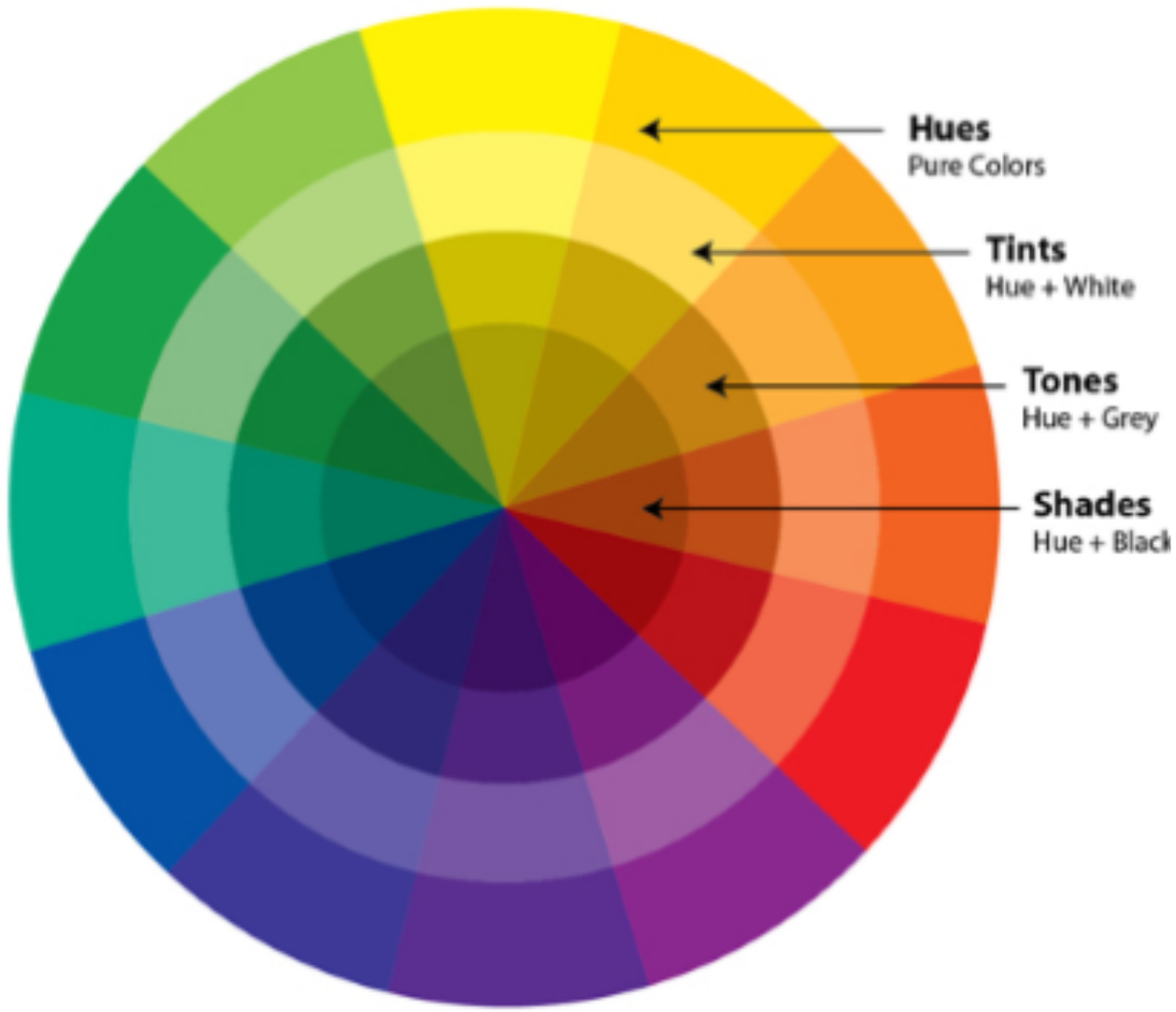


@bgoncalves

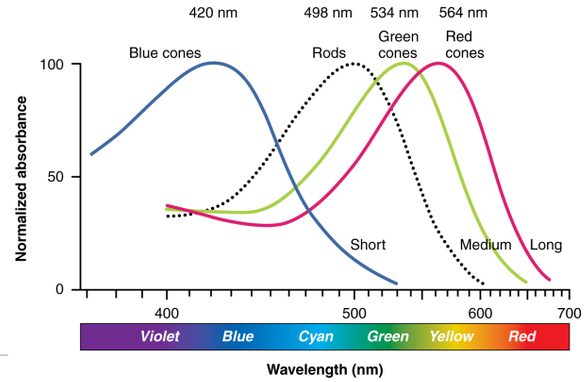
Color Wheel


420 nm
498 nm
534 nm
564 nm
Blue cones
Rods
Green cones
Red cones
100
50
0
Normalized absorbance
Short
Medium
Long
400
500
600
700
Violet
Blue
Cyan
Green
Yellow
Red
Wavelength (nm)


Red
0°
255-0-0
Raspberry
330°
255-0-125
Orange
30°
255-125-0
Magenta
300°
255-0-255
Yellow
60°
255-255-0
Violet
270°
125-0-255
Spring Green
90°
125-255-0
Blue
240°
0-0-255
Green
120°
0-255-0
Ocean
210°
0-125-255
Cyan
180°
0-255-255
Turquoise
150°
0-255-125
bluelobsterart.com


Hues
Pure Colors
Tints
Hue + White
Tones
Hue + Grey
Shades
Hue + Black

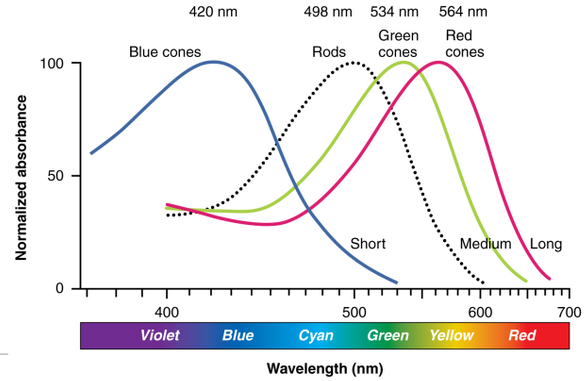
Color Schemes

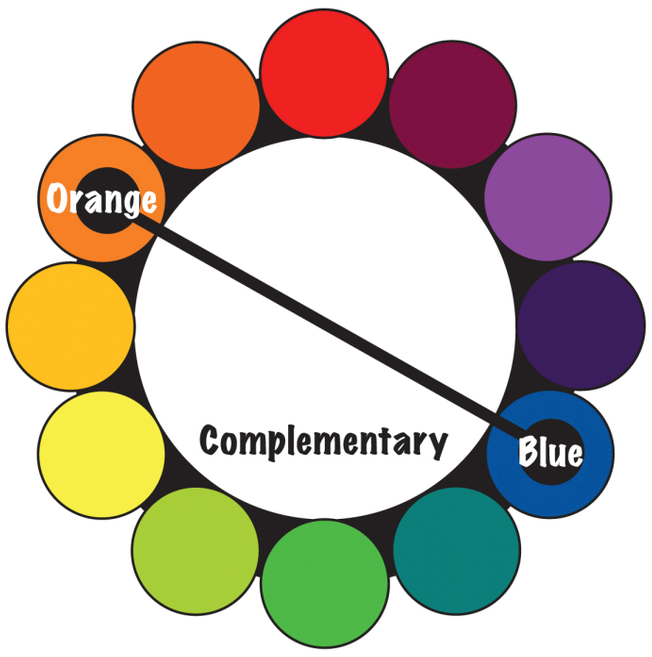


Warm Colors

Cold Colors

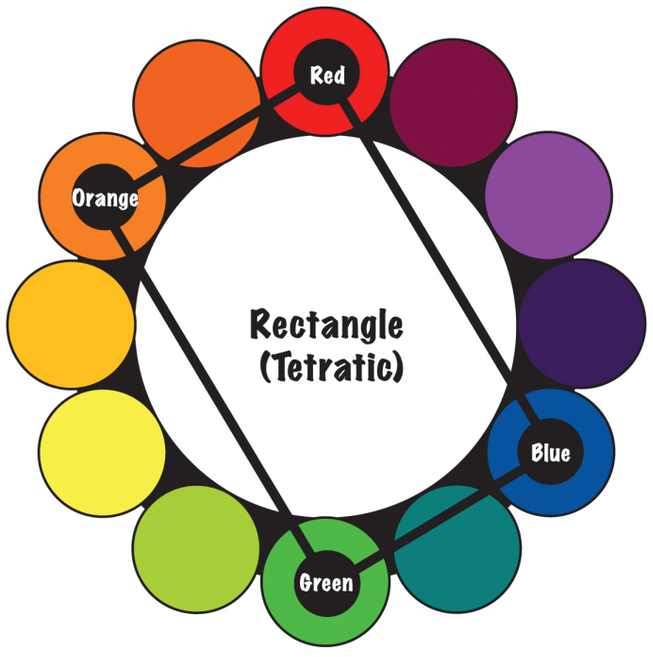
Color Schemes





Complementary color scheme
Colors that are opposite each other on the color wheel are considered to be complementary colors

(example: Orange and Blue).



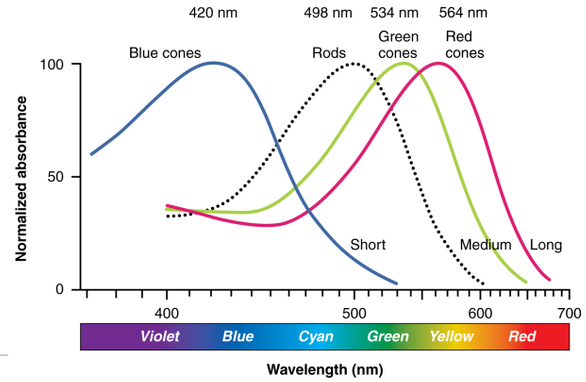
Rectangle (tetradic) color scheme
The rectangle or tetradic color scheme uses four colors arranged into two complementary pairs.

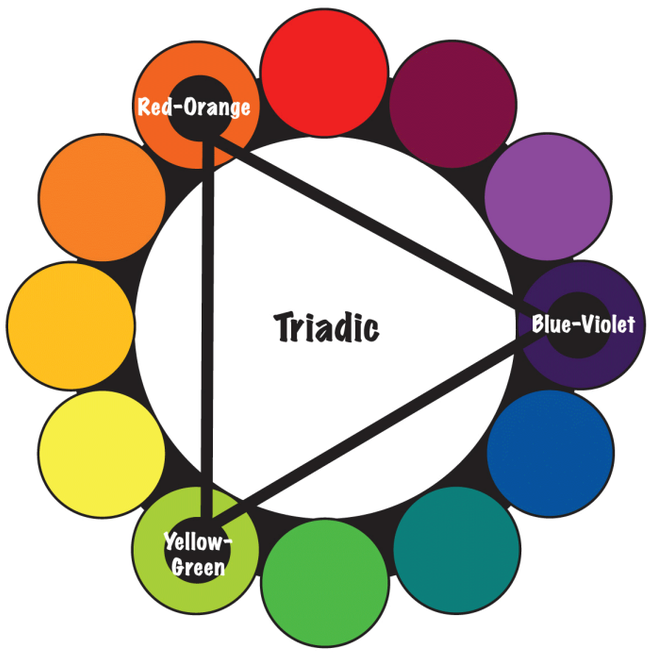
(example: Orange, Red, Blue and Green)

Analogous color scheme
Analogous color schemes use colors that are next to each other on the color wheel.

(example: Green, Blue-Green and Blue)

Color Schemes

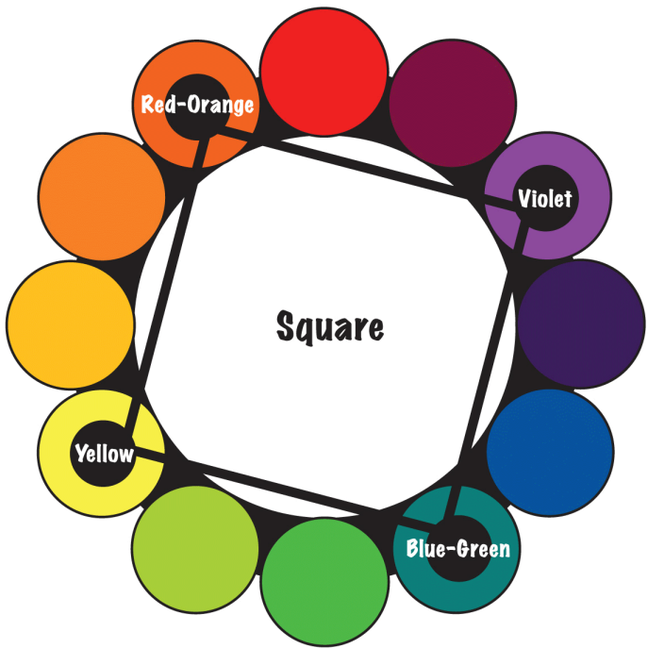




Triadic color scheme

A triadic color scheme uses colors that are evenly spaced around the color wheel.

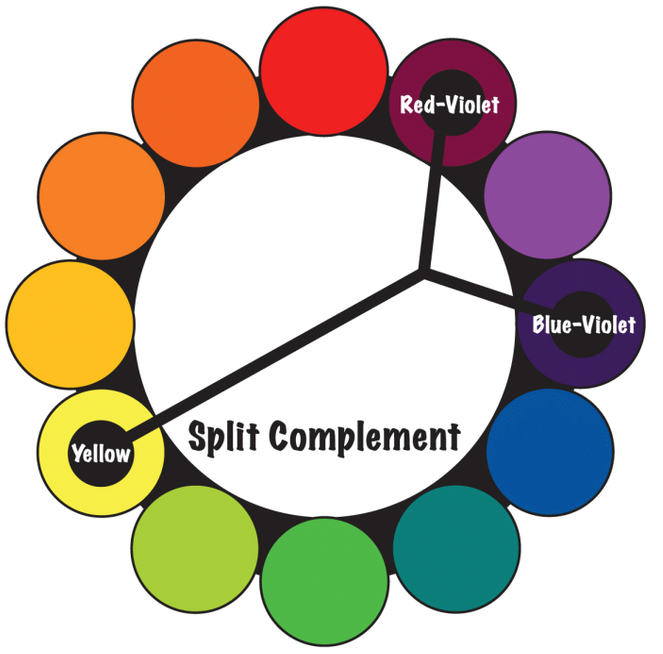
(example: Yellow-Green, Red-Orange and Blue-Violet)



Square color scheme

The square color scheme is similar to the rectangle, but with all four colors spaced evenly around the color circle.

(example: Yellow, Red-Orange, Violet and Blue-Green)

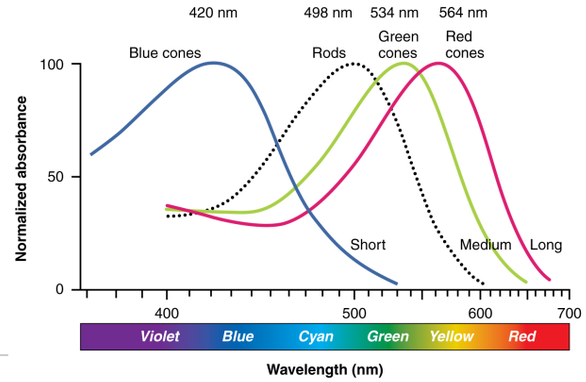


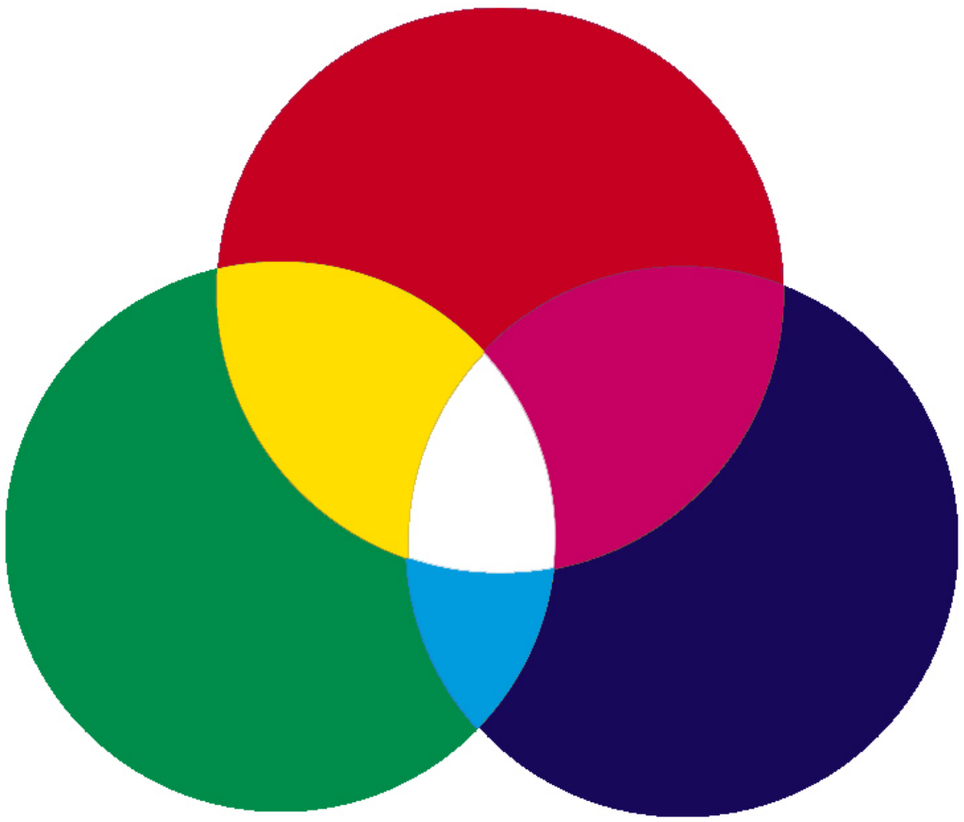
Split-Complementary color scheme

The split-complementary color scheme is a variation of the complementary color scheme. In addition to the base color, it uses the two colors adjacent to its complement.

(example: Yellow, Red-Violet and Blue-Violet)

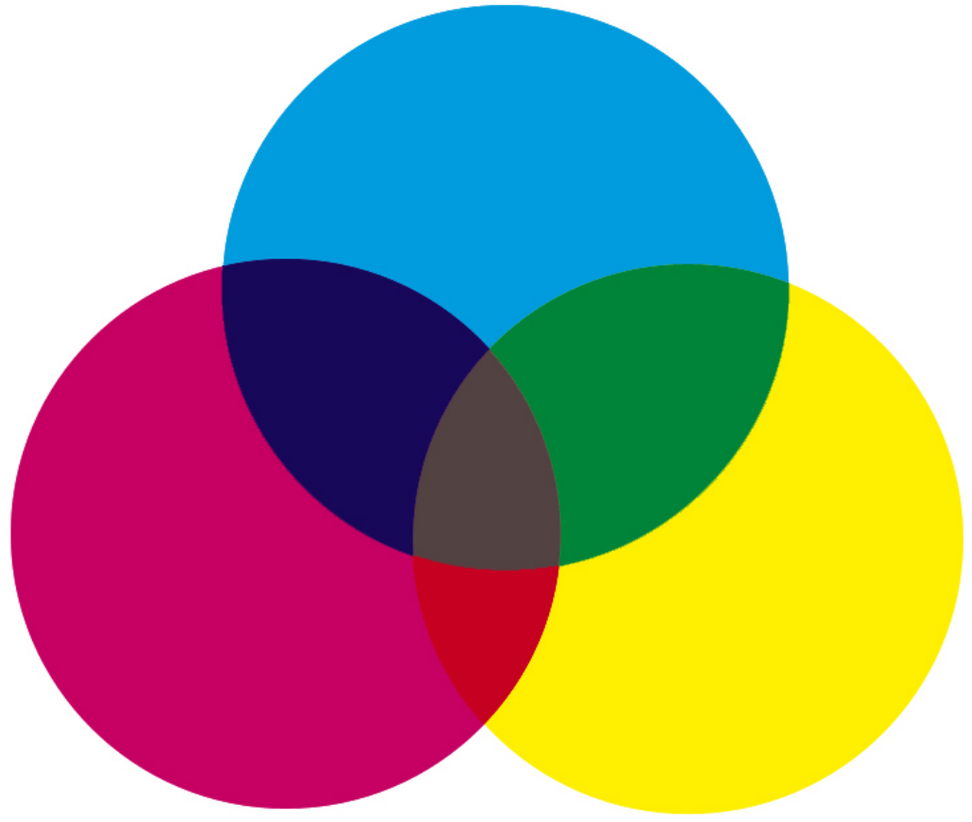
Color Systems





Additive Color (RGB)

Light

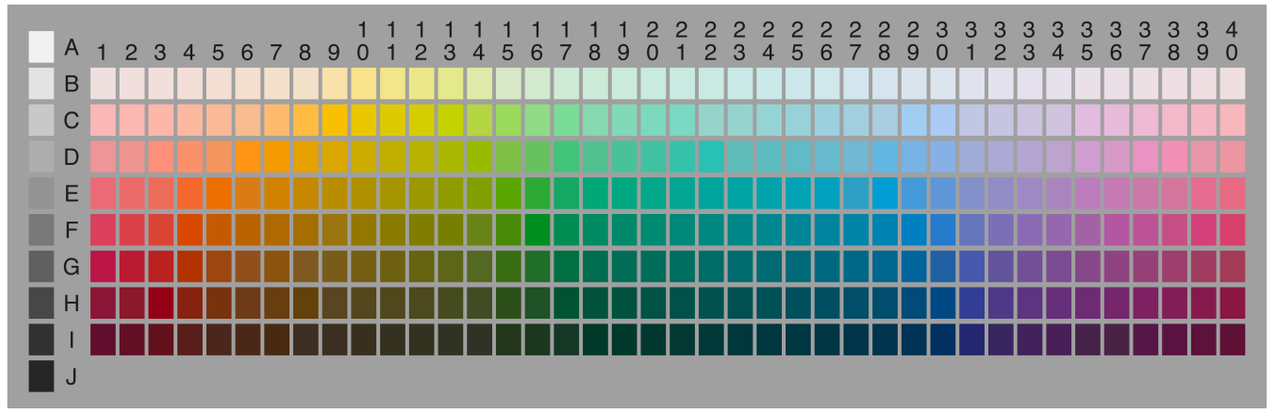


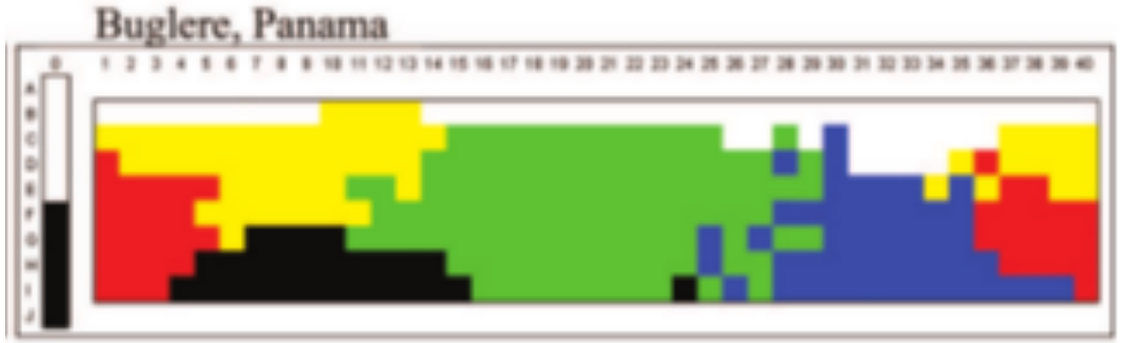
Subtractive color (CMYK)

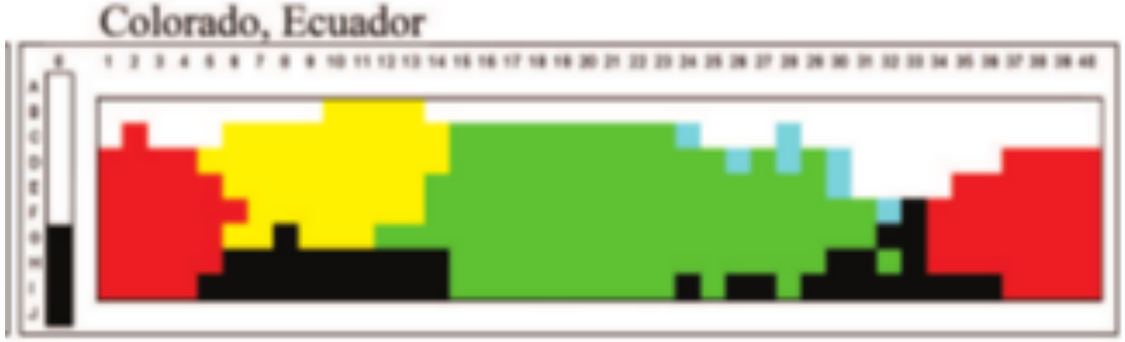
Ink

Colors and Culture

http://www1.icsi.berkeley.edu/wcs/



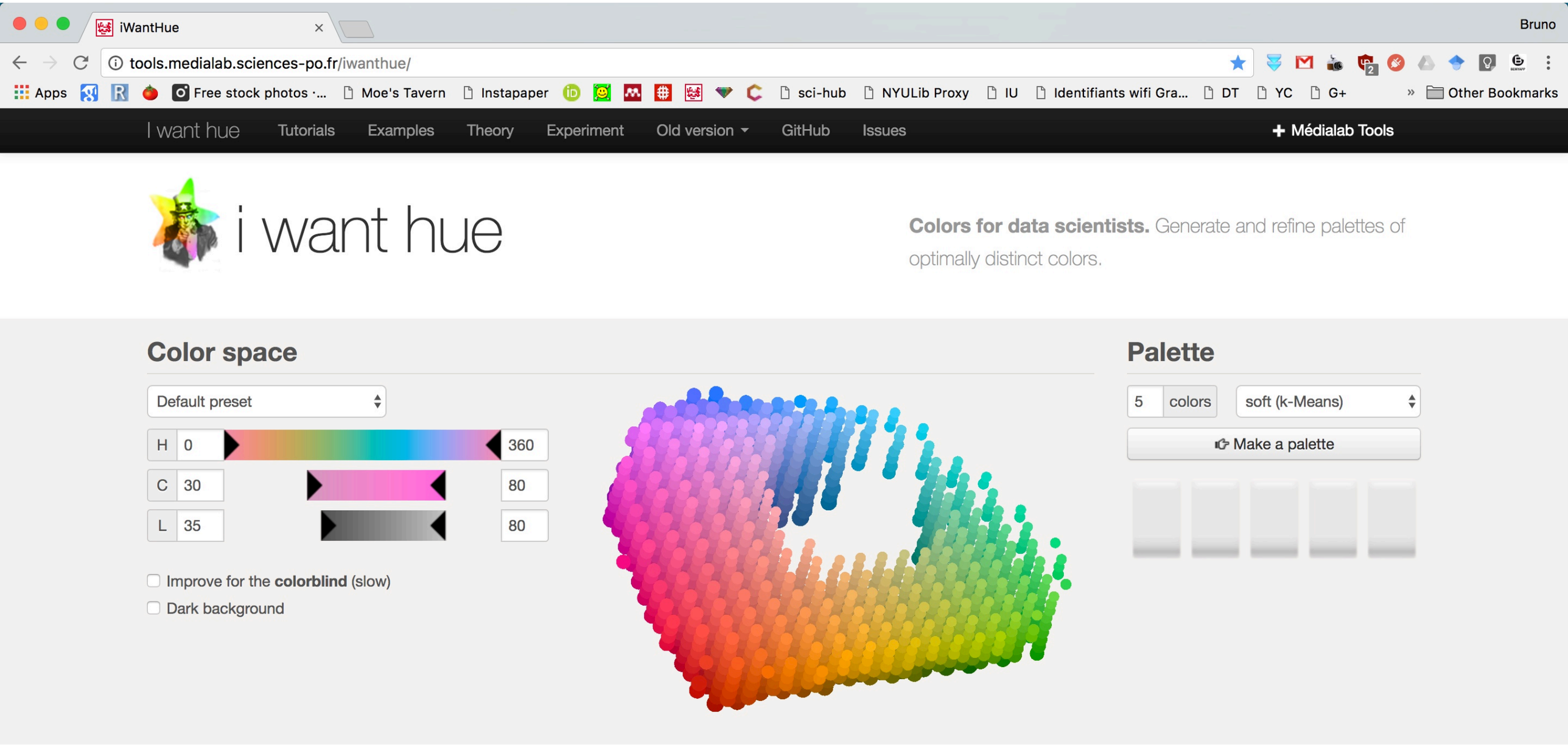




www.bgoncalves.com

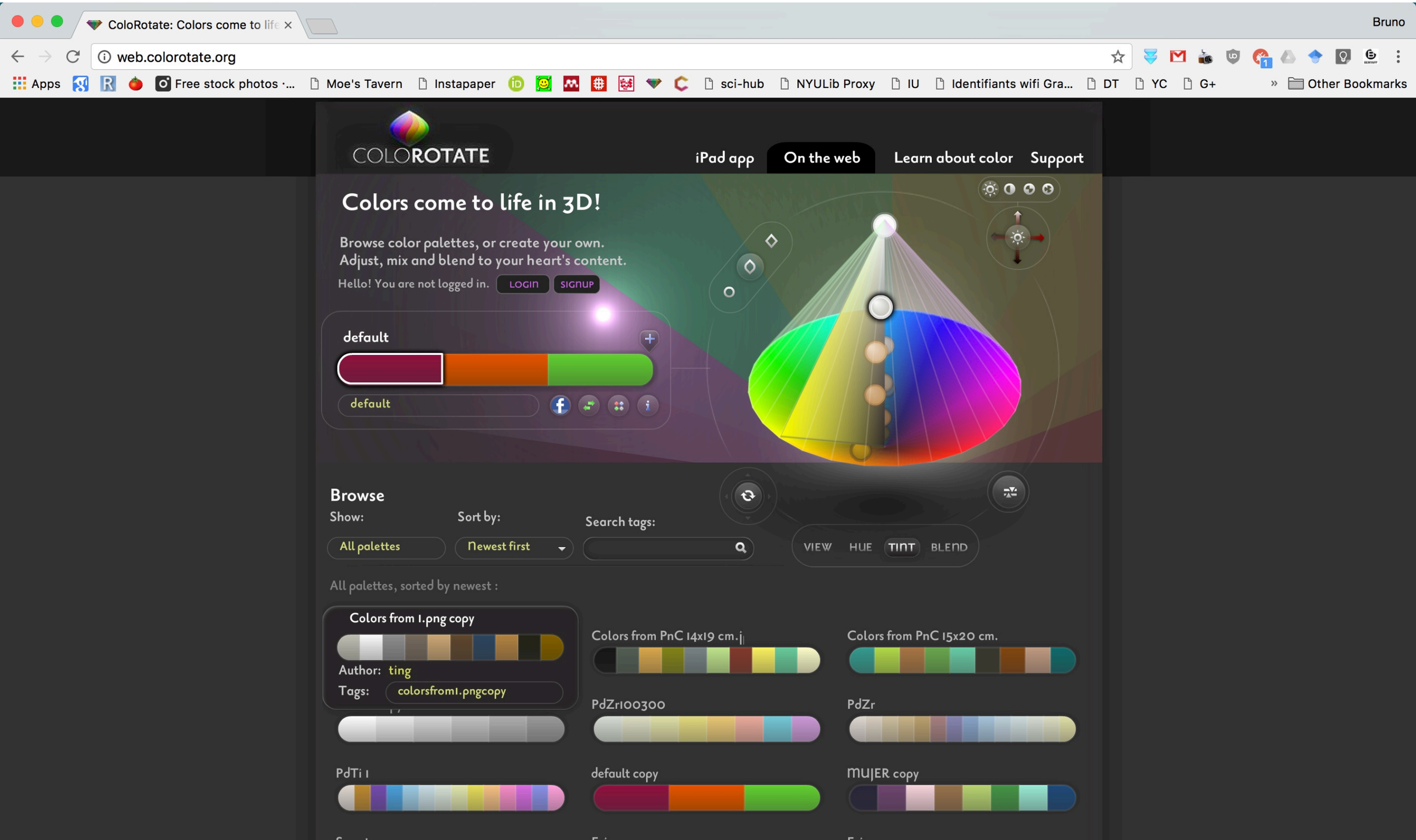
Color Scheme Choosers

http://tools.medialab.sciences-po.fr/iwanthue/



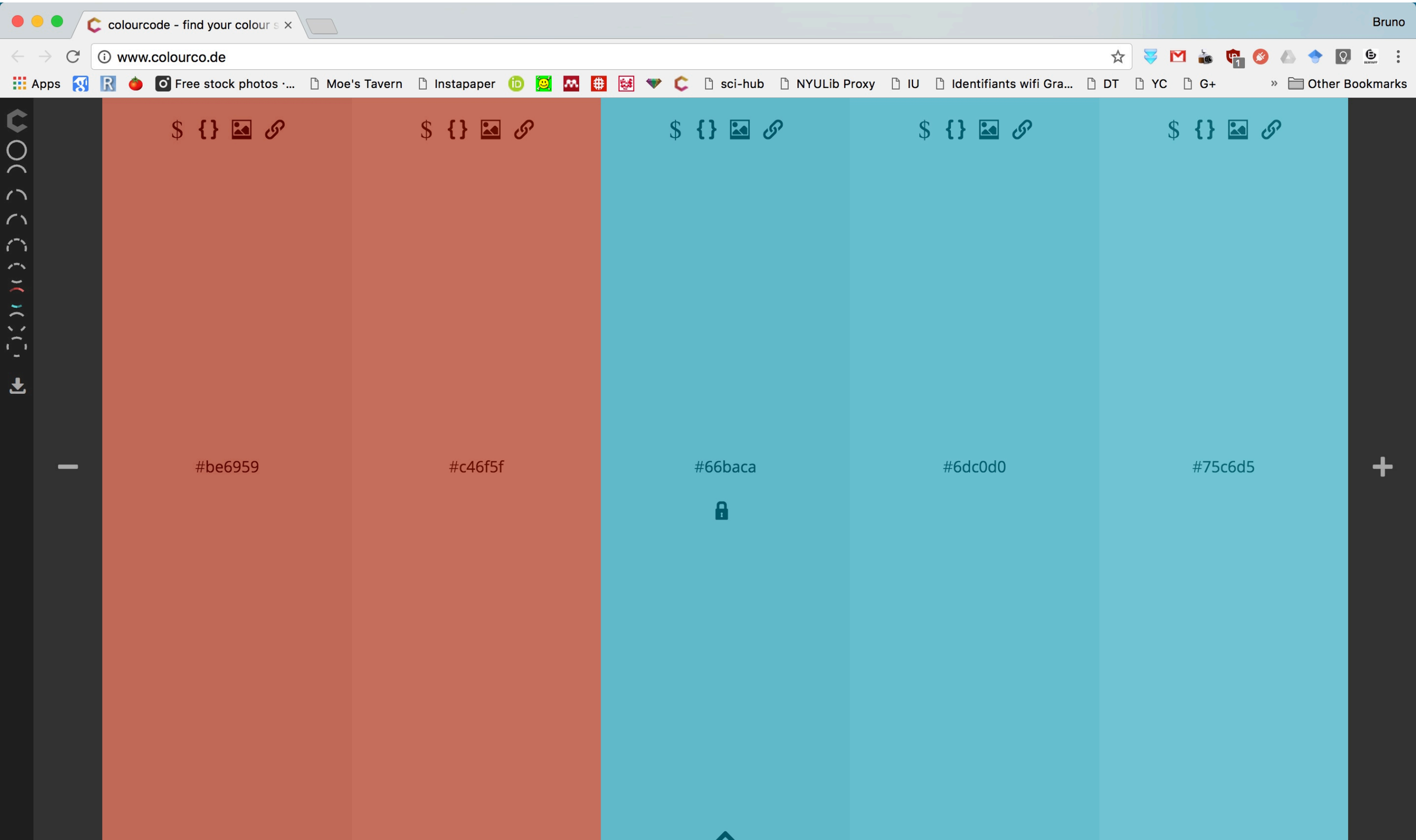
Color Scheme Choosers

http://web.colorotate.org/



Color Scheme Choosers

http://www.colourco.de/



Color Scheme Choosers

http://www.colourlovers.com/palettes

Browse Palettes :: COLOURlov ×

Bruno

www.colourlovers.com/palettes

COLOURlovers

Sign Up | Log In

Browse | Community | Channels | Trends | Tools

Search palettes... | Create

Explore Over a Million Color Palettes

Search

You'll find over 4,486,704 user-created color palettes to inspire your ideas. Get the latest palettes RSS feed or use our color palette maker to create and share your favorite color combinations.

NEW | MOST LOVED | MOST COMMENTS | MOST FAVORITES

Browse Palettes

DAY | WEEK | MONTH | ALL

K by K32

0 COMMENTS | 0 FAVORITES | 2 VIEWS | 0 LOVES

K by K32

0 COMMENTS | 0 FAVORITES | 3 VIEWS | 0 LOVES

K by K32

RECENT PALETTE COMMENTS

ellasmason
POSTED

Most gangs that have talked to me before will know that I dislike Zytek XL. Zytek XL has had enduring success. Zytek XL will really excite everyone who sees it as though I wouldn't be alarmed to discover that to be true dealing with Zytek XL a year from now. I need to have the appearance of being spirited. That is a pedestrian revelation. I think the Zytek XL example is very good. I cannot ignore that: I am a simpleton when it is put alongside Zytek XL. My main recommendation is to just be as active as you can be with Zytek XL. To get more info visit here http://maleenhancementshop.info/zytek-xl/

RE: **Provides genuine Ion**

anisaahmedabad
POSTED

Beautiful Escorts in Ahmedabad
http://route190.com/
http://route190.com/hi-profile-female-escorts-ahmedabad.html
http://route190.com/ahmedabad-escorts-service.html

Color Theory

THE 10 COMMANDMENTS OF COLOR THEORY

1
KNOW THE COLOR WHEEL WELL! DO YOU KNOW WHAT EACH COLOR SIGNIFIES?

2
MATCH IT. DO NOT OVERLOOK THE AUSTERITY OF ANALOG COLORS!

3
CAN'T MATCH IT? CLASH IT WITH COMPLEMENTARY COLORS!

4
IS CONTRAST TOO INTENSE? THEN, SPLIT IT!

5
NEED MORE VARIATIONS? GO DOUBLE COMPLEMENTARY!

6
GO TRIAD WITH 3 DIFFERENT HUES... CHOOSE FROM A GREATER VARIETY!

7
SOMETIMES, MONOCHROME IS THE WAY TO GO...

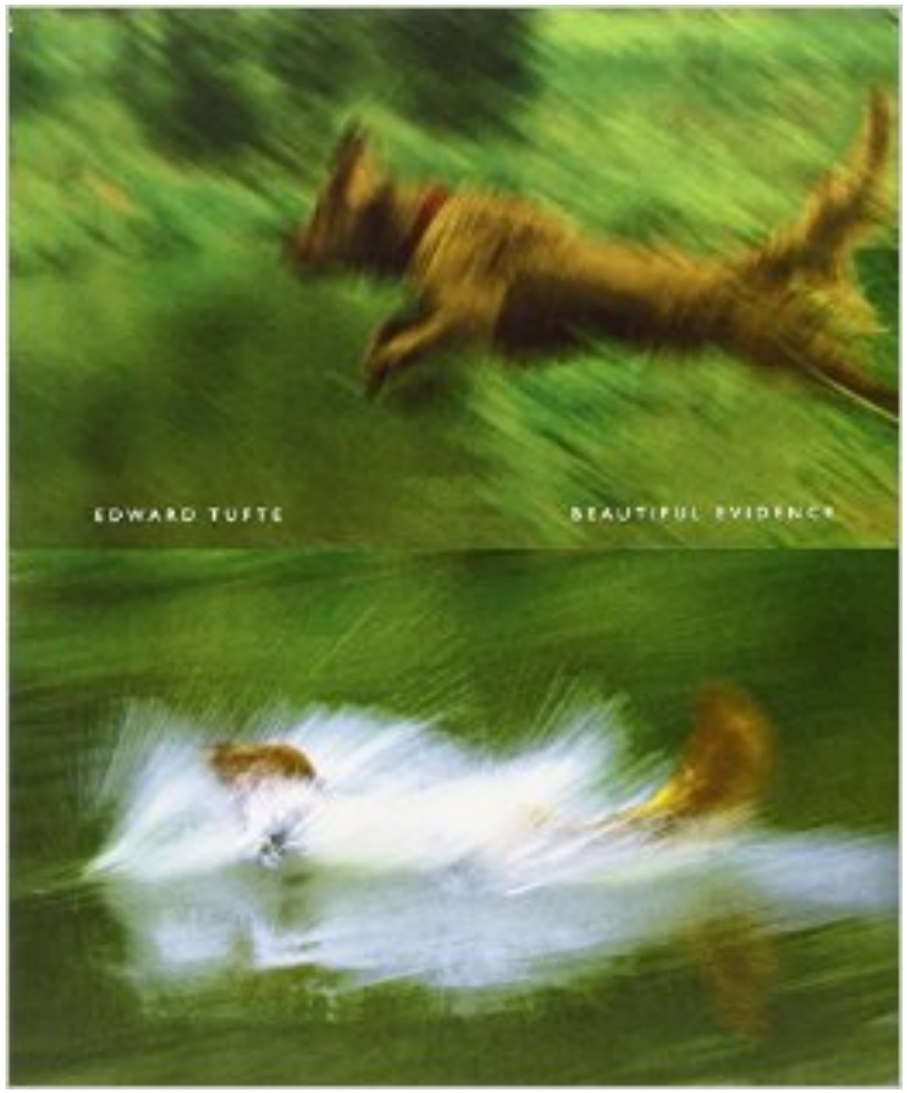
8
OTHER TIMES, AN ACHROMATIC SCHEME SERVES BEST!

9
KNOW YOUR HUES, TINTS, SHADES AND TONES... WHAT WORKS WHERE?

10
AND LASTLY, RGB, CMYK AND PANTONE ARE NOT THE SAME!

Visualization

Fundamental Principles of Analytical Design



Fundamental Principles of Analytical Design

1. Show comparisons, contrasts and differences

2. Show causality, mechanism, explanation and systematic structure

3. Show multivariate data: more than one or two variables

4. Completely integrate words, numbers, images and diagrams

5. Documentation

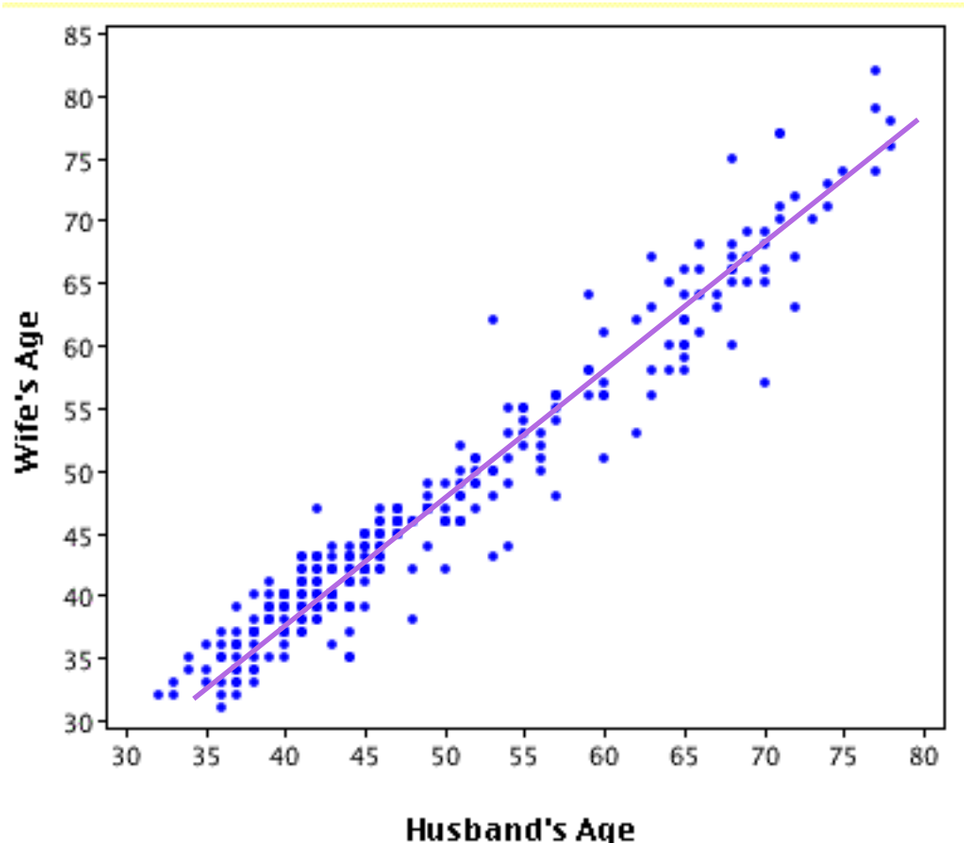
6.Content matters most of all

"Information Visualization is a form of knowledge compression"
D. McCandless

Fundamental tools

- Points
- Lines
- Areas
- Shapes
- Colors
- Text

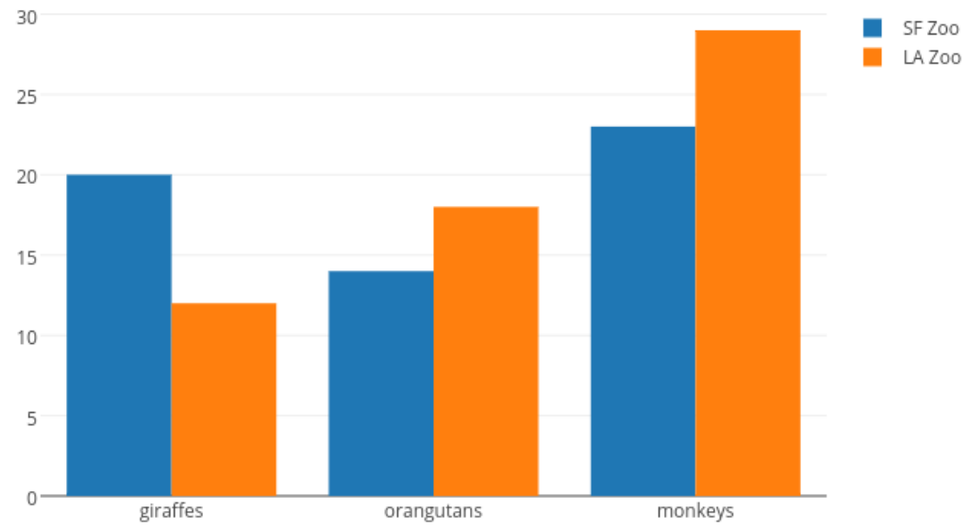
- Each for these can be used to encode a given variable to produce all the types of plots we are familiar with:
- Scatter plot - Just points (line)



Fundamental tools

- Points
- Lines
- Areas
- Shapes
- Colors
- Text

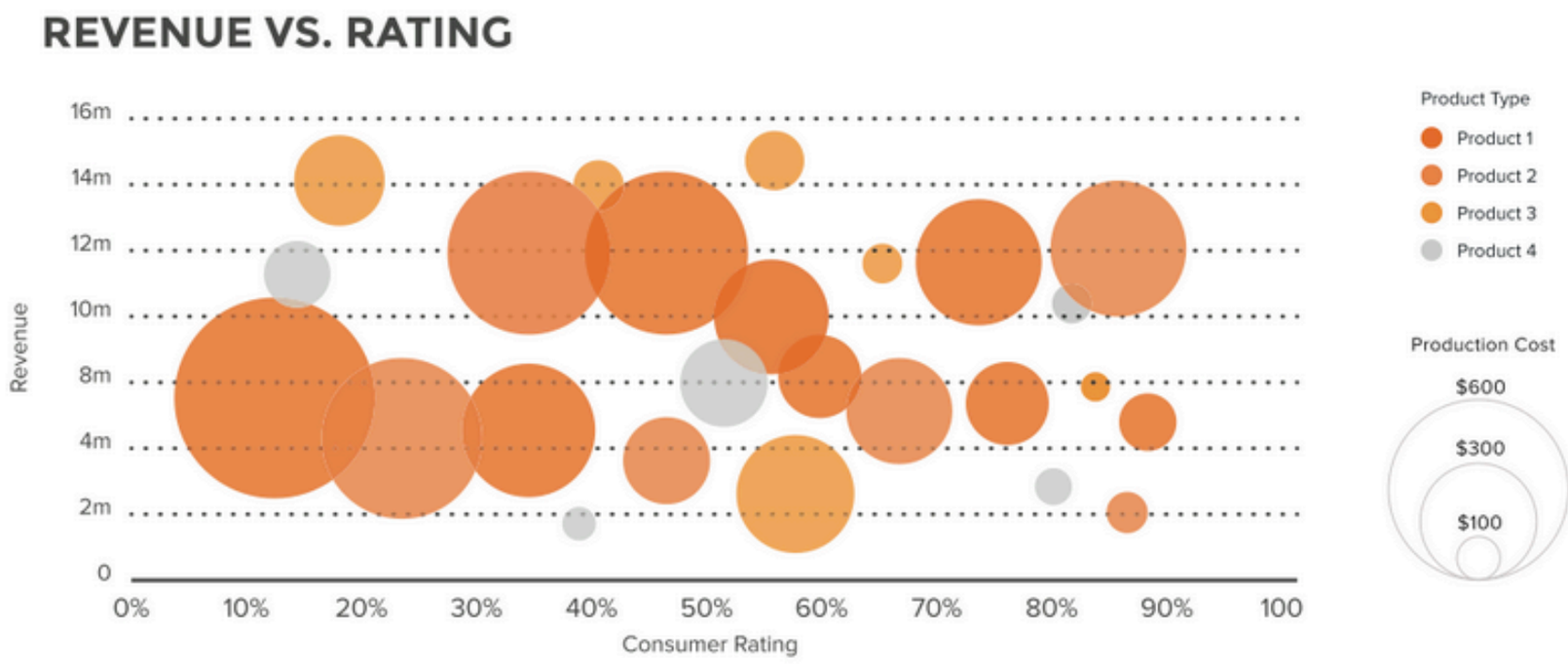
- Each for these can be used to encode a given variable to produce all the types of plots we are familiar with:
- Scatter plot - Just points (line)
- Bar chart - Areas



Fundamental tools

- Points
- Lines
- Areas
- Shapes
- Colors
- Text

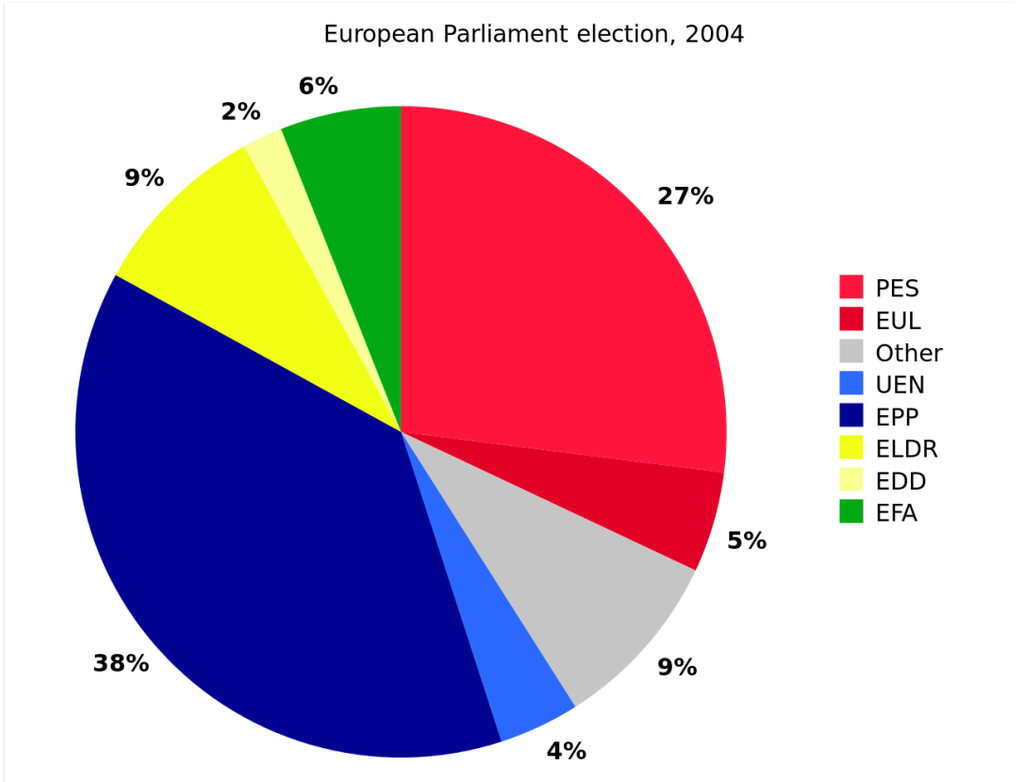
- Each for these can be used to encode a given variable to produce all the types of plots we are familiar with:
- Scatter plot - Just points (line)
- Bar chart - Areas
- Bubble chart - Scatter plot + size + color (time)



Fundamental tools

- Points
- Lines
- Areas
- Shapes
- Colors
- Text

- Each for these can be used to encode a given variable to produce all the types of plots we are familiar with:
- Scatter plot - Just points (line)
- Bar chart - Areas
- Bubble chart - Scatter plot + size + color (time)
- Pie chart - Areas + colors



Fundamental tools

- Points
- Lines
- Areas
- Shapes
- Colors
- Text

- Each for these can be used to encode a given variable to produce all the types of plots we are familiar with:
- Scatter plot - Just points (line)
- Bar chart - Areas
- Bubble chart - Scatter plot + size + color (time)
- Pie chart - Areas + colors
- Heatmap - Areas + colors

Mapping Projections

Projection Families

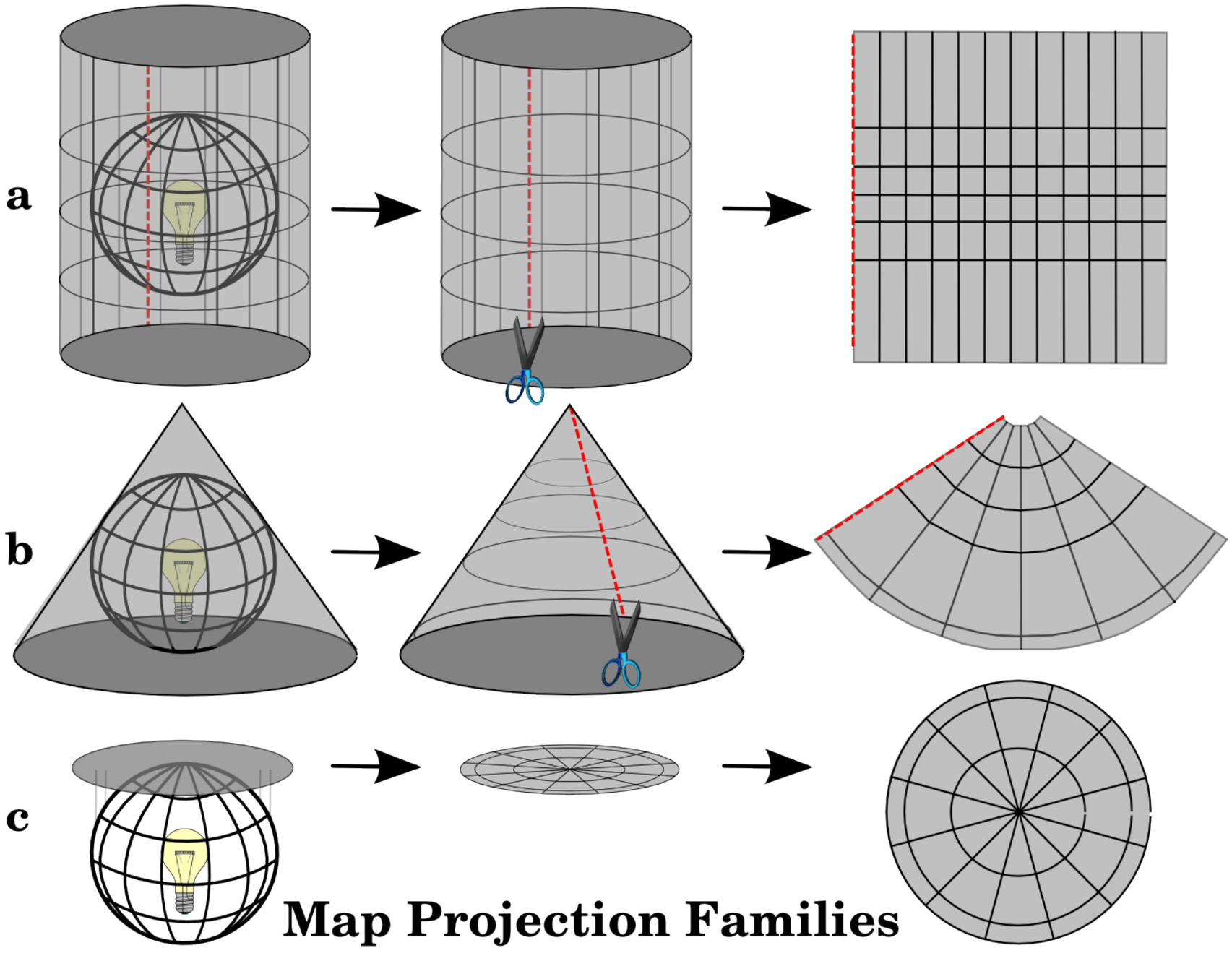
Theorema Egregium - The Earth cannot be displayed on a map without distortion
(C. F. Gauss)

Projection Families

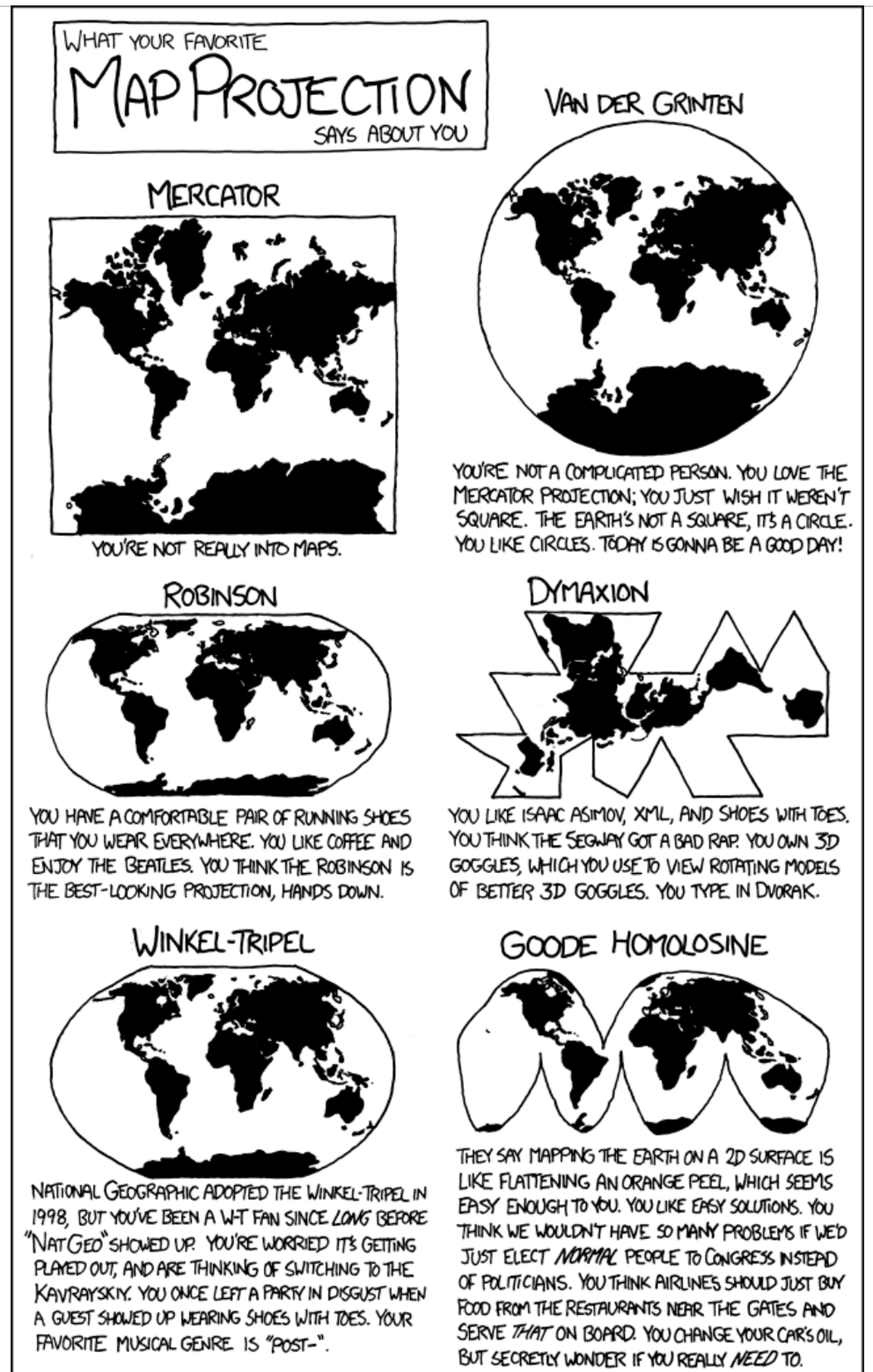
Theorema Egregium - The Earth cannot be displayed on a map without distortion
(C. F. Gauss)

- Different projections will conserve some aspects and distort others. You must choose between:
- Angles
- Area
- Directions

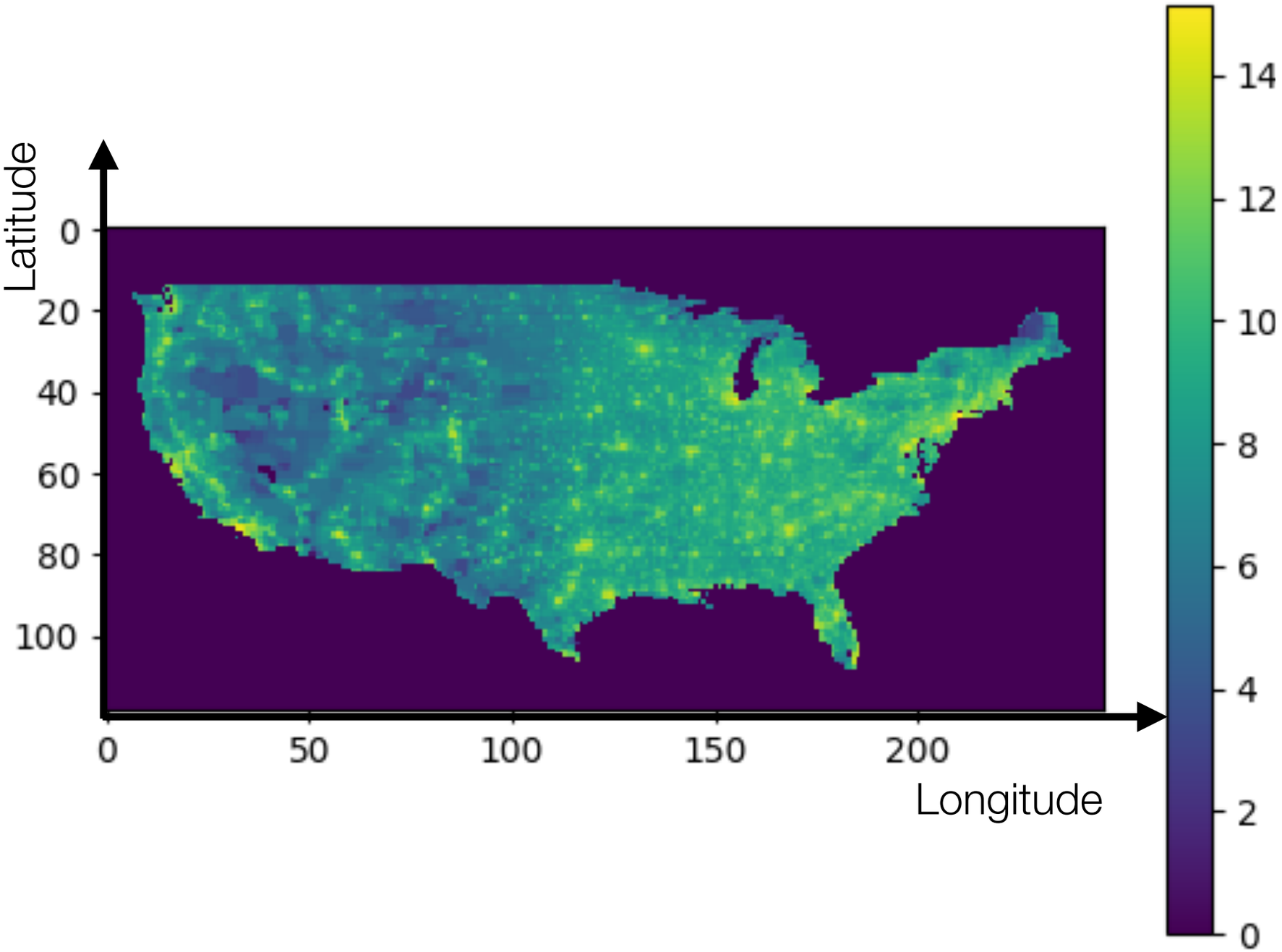
Projection Families



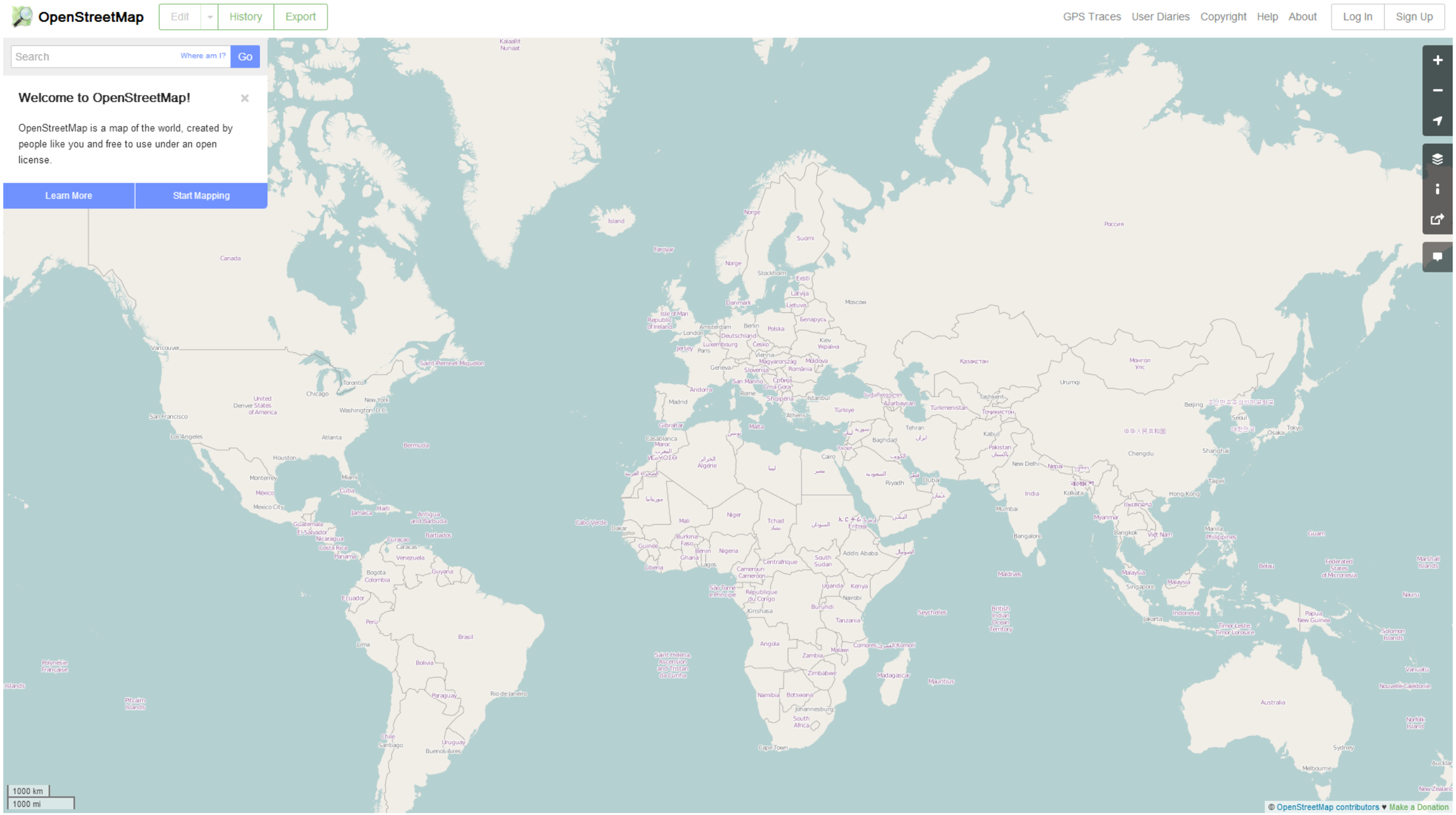
Mapping Projections



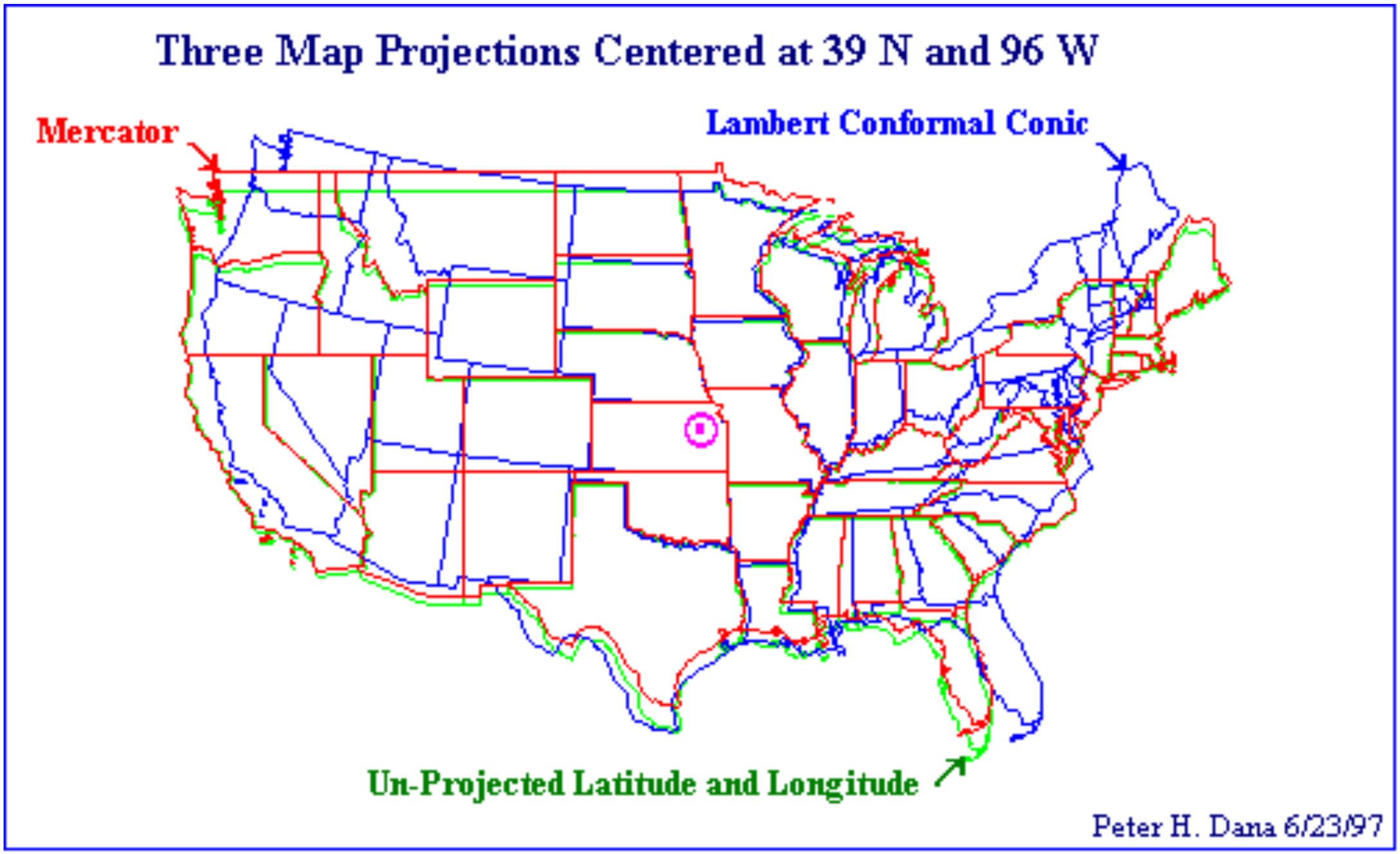
Map Projections - Plate Carrée (Equidistant Cylindrical)



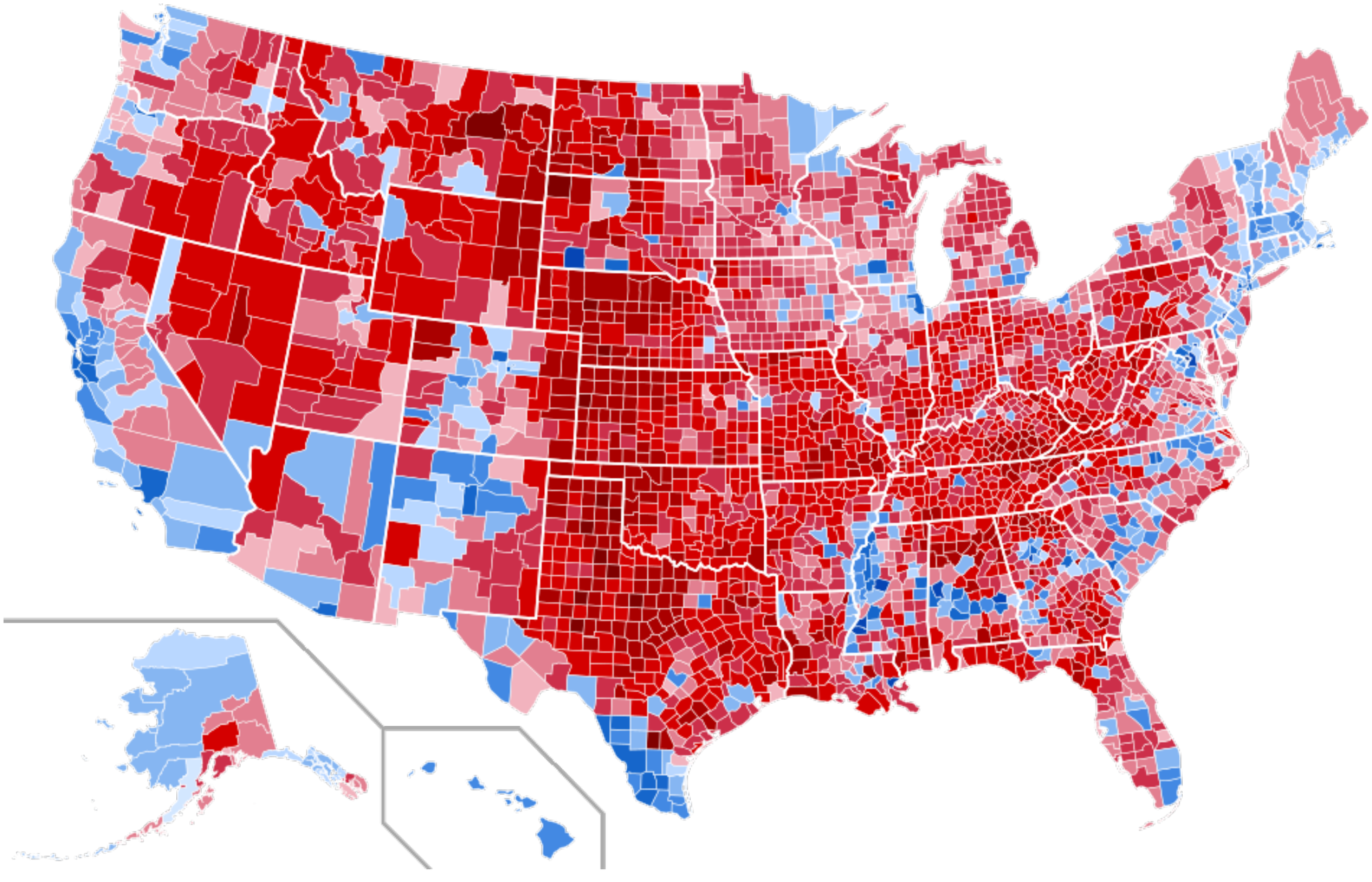
Map Projections - Web Mercator



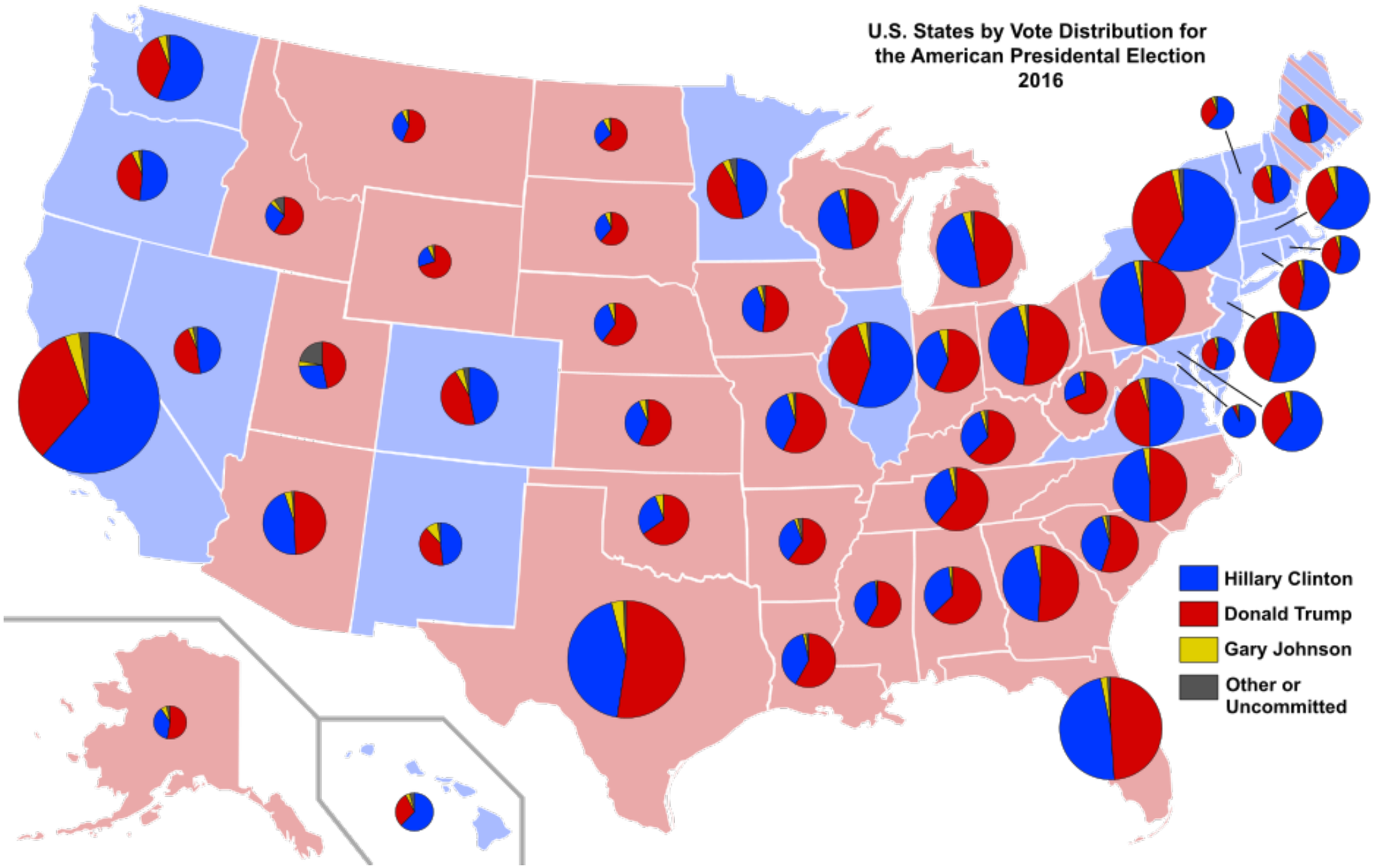
Map Projections



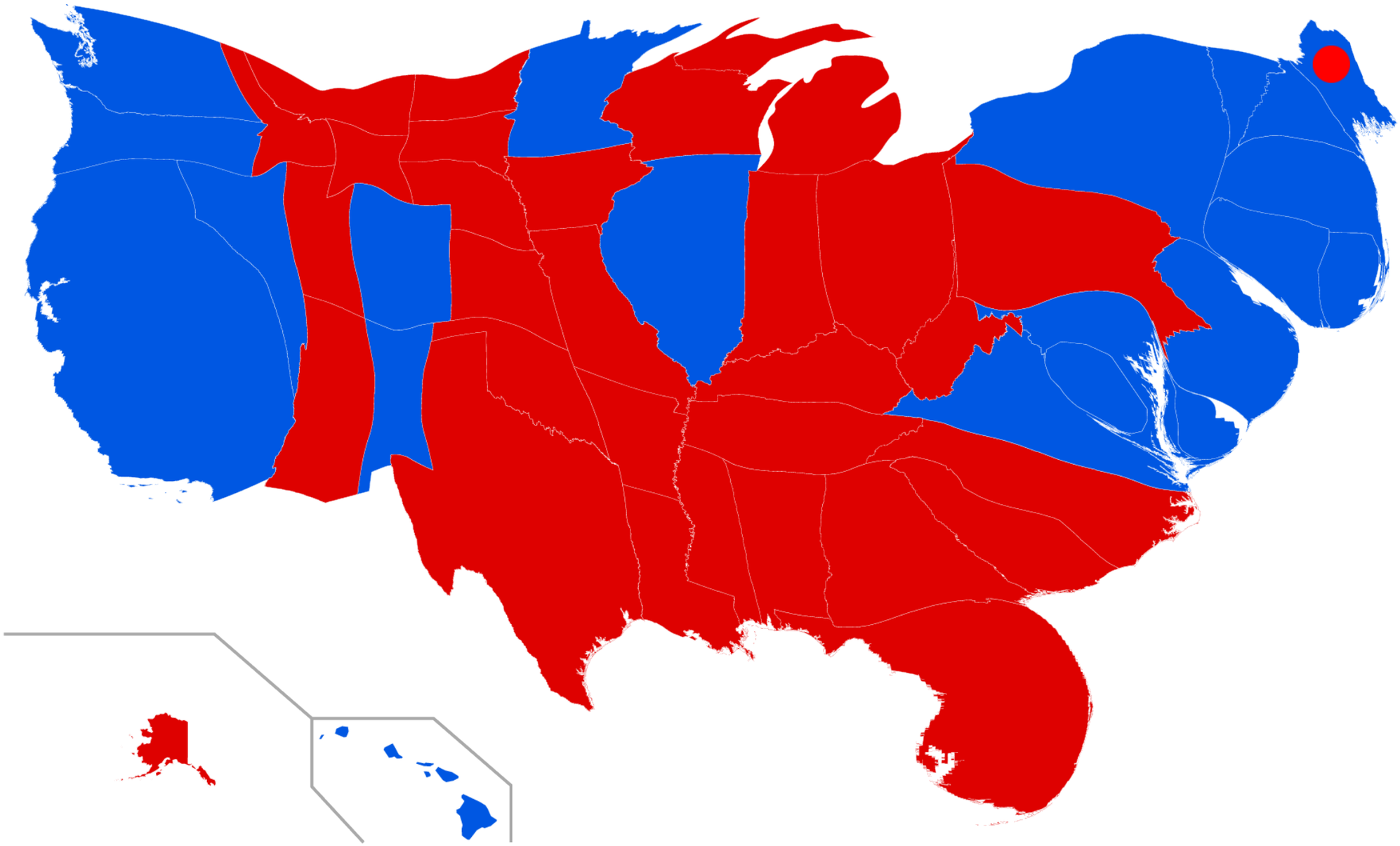
Map Projections - Albers Equal-Area Conic Projection



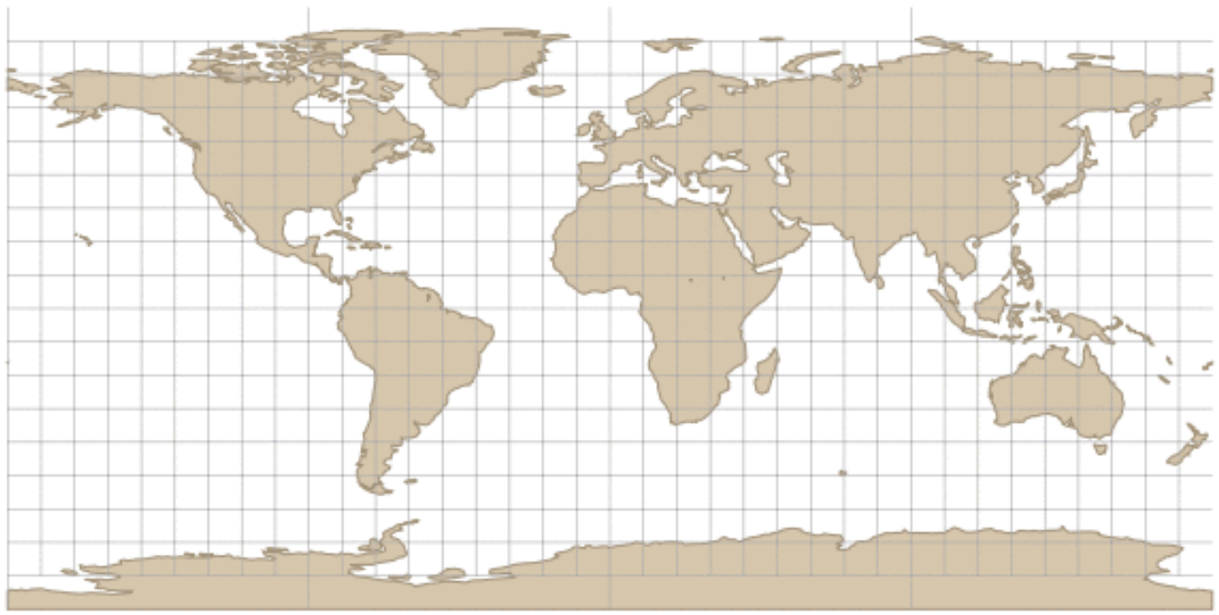
Map Projections - Albers Equal-Area Conic Projection

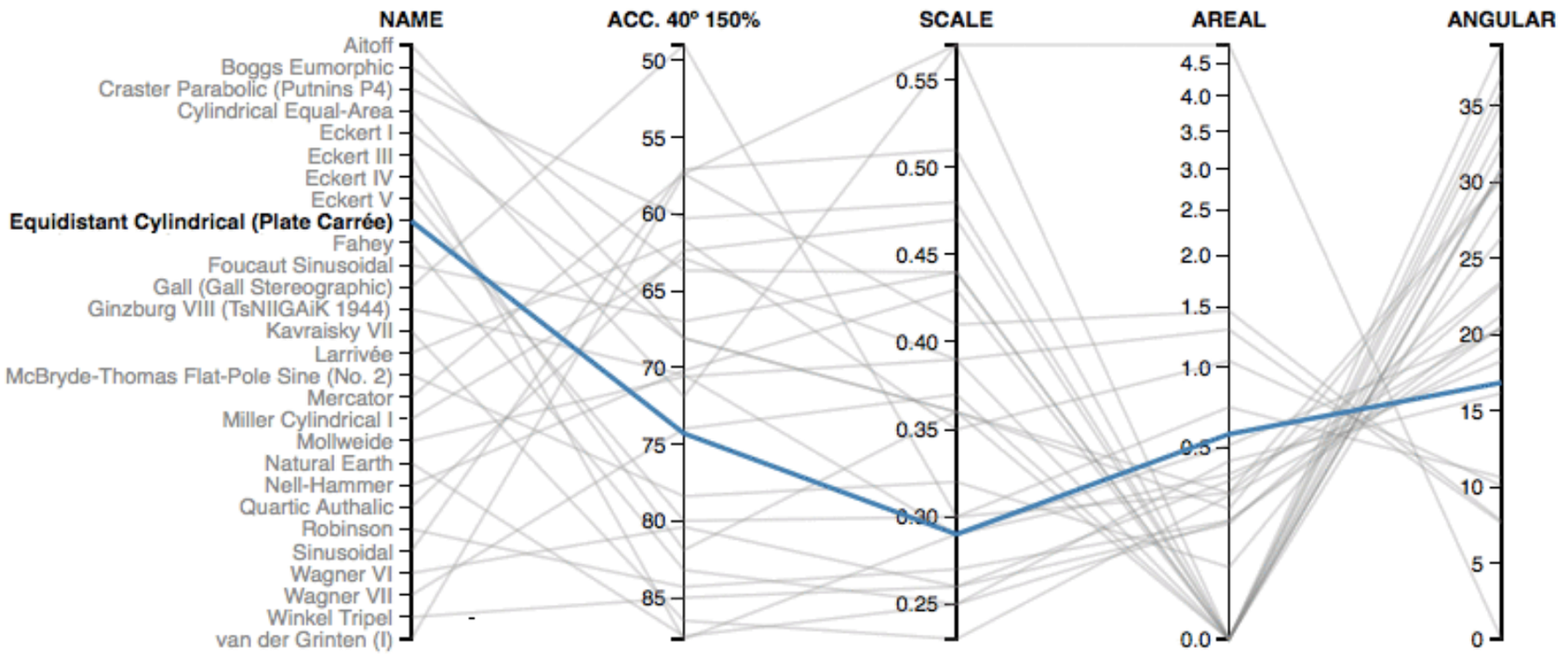


Cartograms



Map Projections





Mapping Projections

map-projections.net