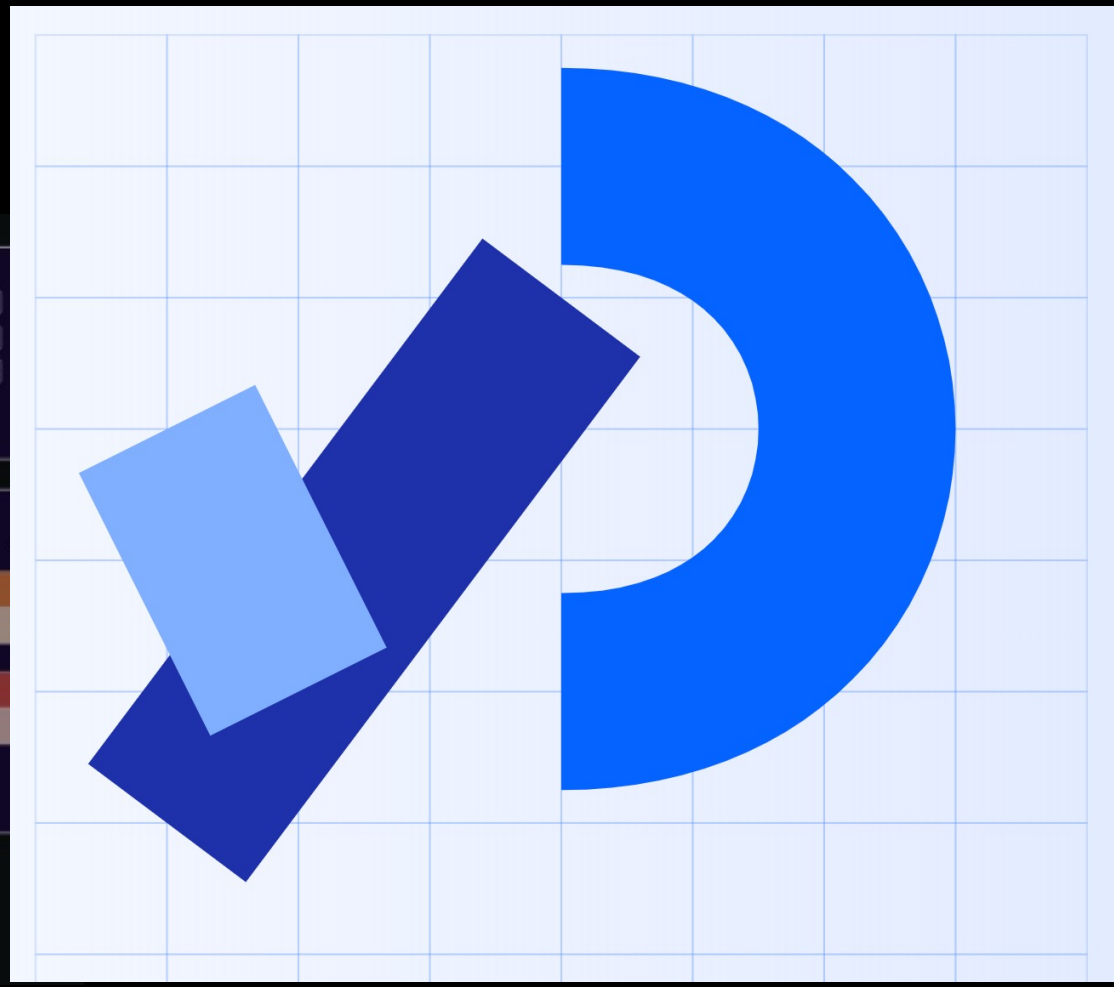
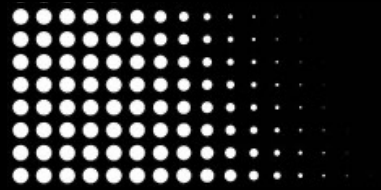
Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

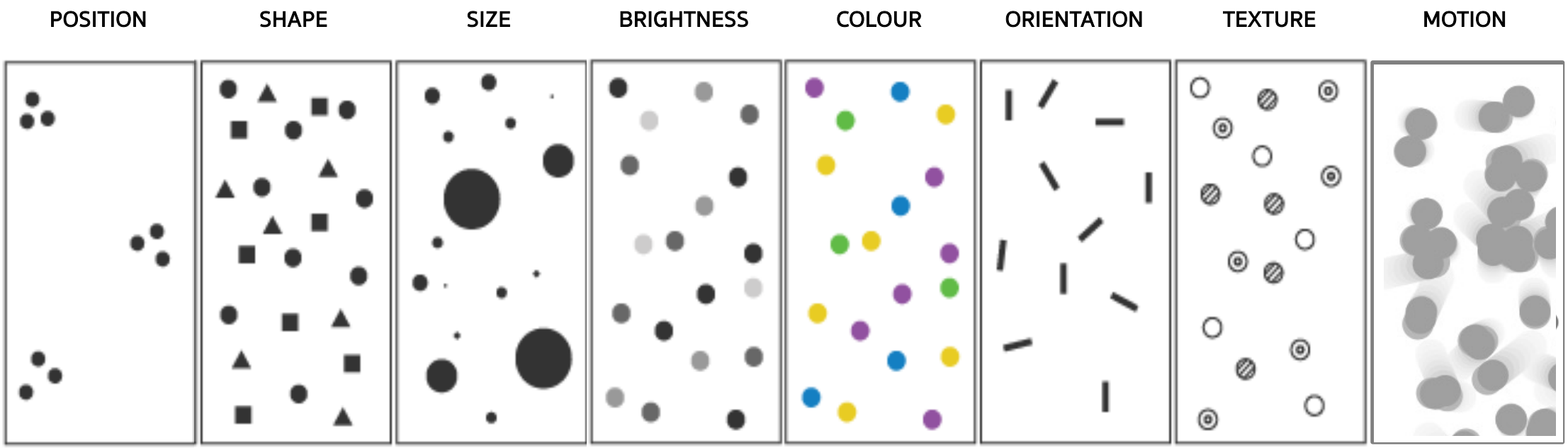


Graphical Encoding

Examples in Processing



Recall : Visual Encoding Channels



The Eight Visual Variables (from Ward et al 2010, after J. Bertin)

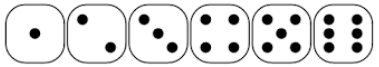
Graphics primitives used to depict data in visualisation

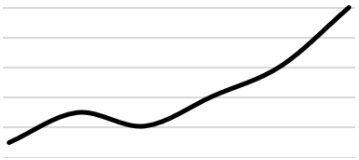
Preview : Common Data Attribute Types

N.B. A more comprehensive discussion of this in provided in future lectures; do not use this page as a reference in assignments

Quantitative

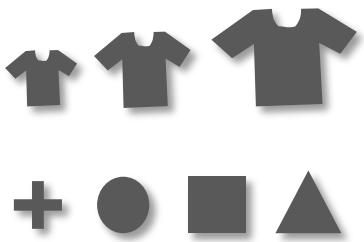
- Discrete : taking only integer values or from a specific subset of possible values e.g. 2, 4, 6, ..
- Continuous : real values e.g. numbers in the interval [0, 5]





Categorical

- Binary : true/false
- Ordinal / ranked: values that have an implied ordering, e.g. small, medium, large
- Nominal / Arbitrary : potentially infinite range and no ordering e.g. addresses



Loading Data : Table

One of many functions and classes in Processing for loading different types of data. For details see https://processing.org/reference/Table.html

```
Table table;
float[] x, y, v;
int[] c;
int N;

size(400, 400);

table = loadTable("sample.csv", "header");

N = table.getRowCount();
x = new float[N];
y = new float[N];
c = new int[N];
v = new float[N];

N = table.getRowCount();
for (int i=0; i<N; i++)
{
  TableRow row = table.getRow(i);
  x[i] = row.getFloat("X");
  y[i] = row.getFloat("Y");
  c[i] = row.getInt("Category");
  v[i] = row.getFloat("Value");
}
```

sample

id	X	Y	Value	Category
0	154.20749	127.546715	0.036984384	0
1	110.68671	103.770874	0.16846246	0
2	93.17792	309.17993	0.86499405	0
3	151.20421	310.14233	0.9057057	1
4	115.480446	250.57158	0.9894253	6
5	126.53544	46.5673	0.630883	4
97	108.75547	285.69888	0.41862965	0
98	392.61032	368.57758	0.2778448	5
99	23.039341	295.21283	0.090792656	1

A contrived data set just to test drawing functions:
X, Y are quantitative values in the range 0-400, Value is a quantitative scalar value between 0 and 1.
Category is a whole number between 0 an 7

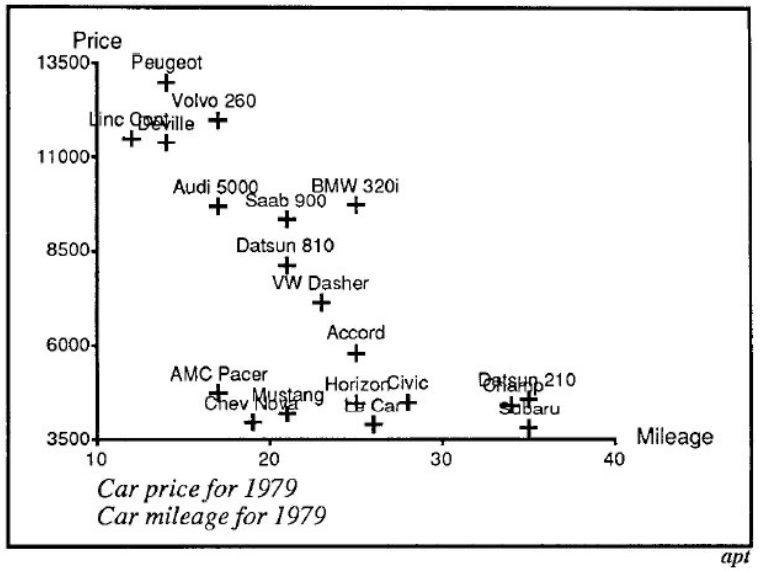
1. Position

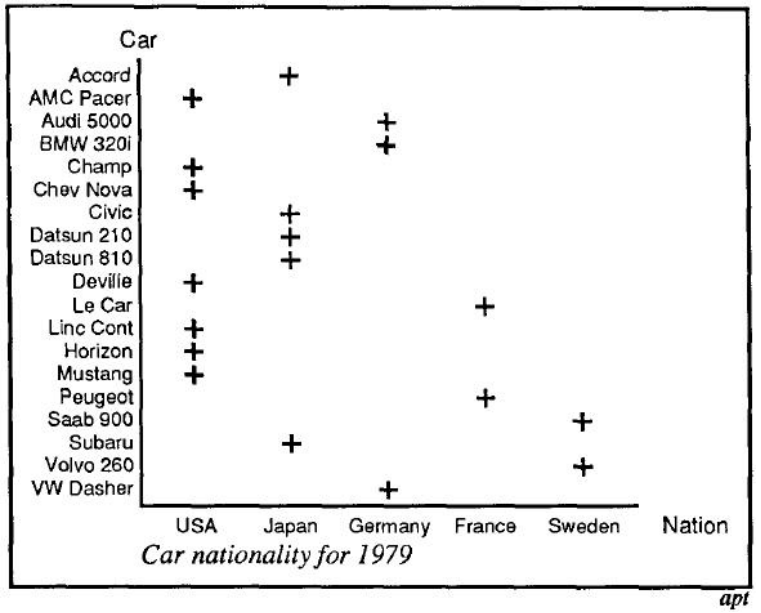
Placement of graphics within the display space

- Greatest impact in most visualizations.

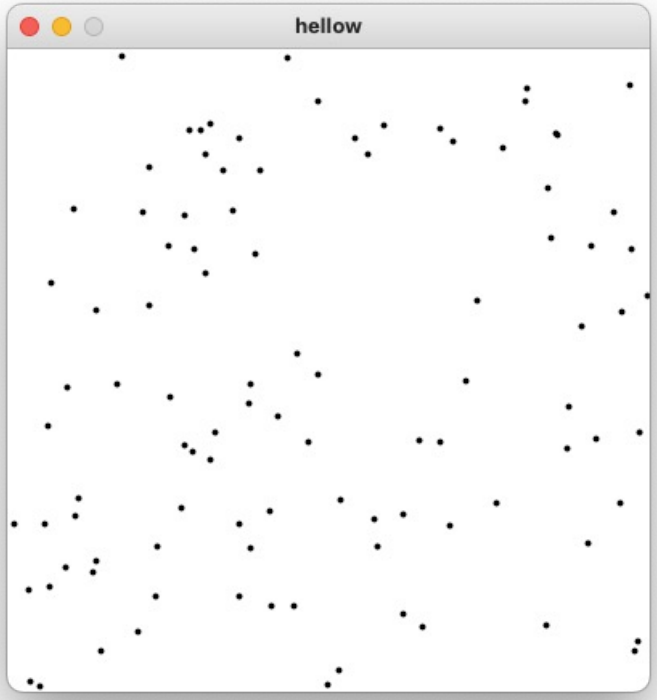
Position is used in ALL visualizations in some way

- expressing values
- categorizing classes
- Arranging the data for legibility e.g. sorting, alignment, layout





Position




```
Table table;
float[] x, y, v;
int[] c;
int N;

size(400, 400);
table = loadTable("sample.csv", "header");

N = table.getRowCount();
x = new float[N];
y = new float[N];
c = new int[N];
v = new float[N];



for (int i=0; i<N; i++)
{
  TableRow row = table.getRow(i);
  x[i] = row.getFloat("X");
  y[i] = row.getFloat("Y");
  c[i] = row.getInt("Category");
  v[i] = row.getFloat("Value");
}
```

```
background(255);
stroke(0);
strokeWeight(4);

for (int i=0; i<N; i++)
{
  point(x[i], y[i]);
}
```

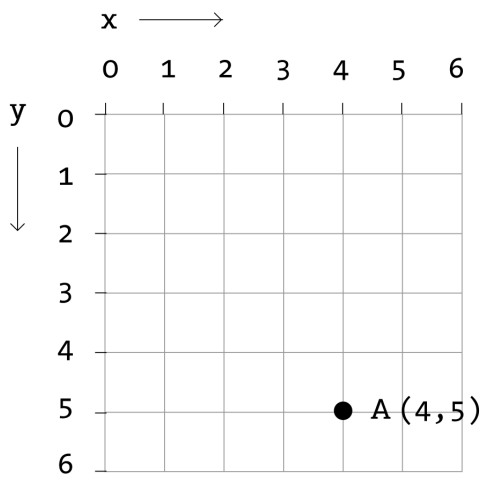
All the drawing is essentially here

Data load and prep as discussed earlier. Hereafter, we omit this for brevity. Assume the same for the following slides.

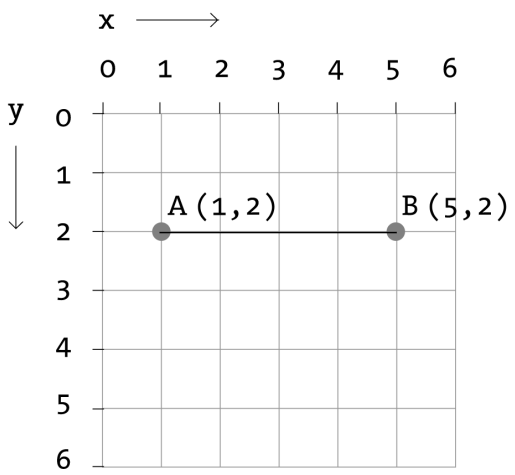
Processing Co-ordinate Space

The origin x=0, y=0 or <0,0> is located at the top left, positive x-axis points right and positive y-axis points down. [This is typical for most 2D graphics system but not all take this approach]

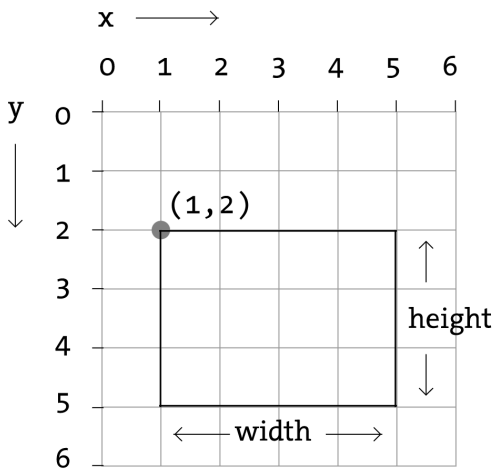
Below are examples of some simple vector drawing functions



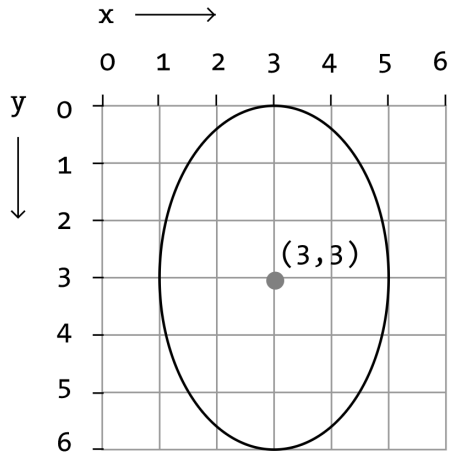
Example: `point(4,5);`



Example: `line(1,2,5,2);`



Example: `rect(1,2,4,3);`



Example:
```
ellipseMode(CENTER);
ellipse(3,3,4,6);
```

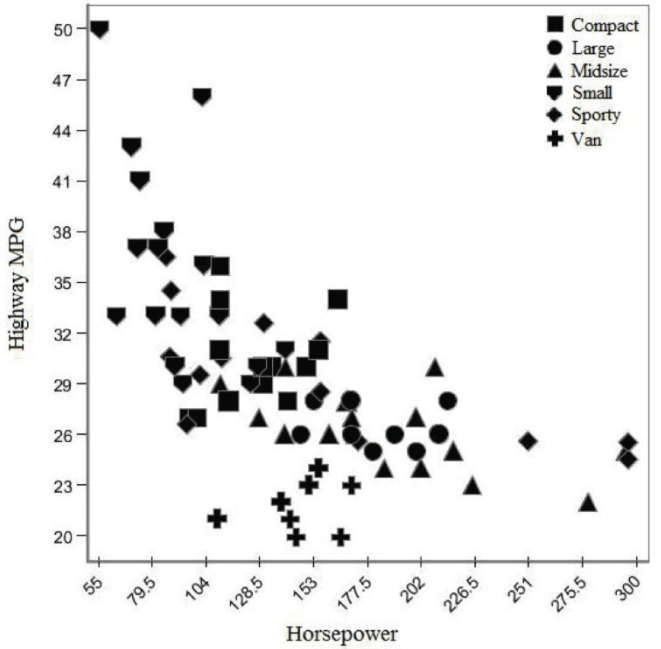
Images © Processing Foundation. Examples from: https://processing.org/tutorials/drawing/

2. Shape / Mark

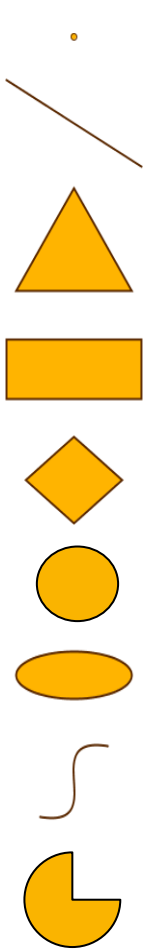
Represent data attribute through geometrical shape

- Great to recognize many classes but doesn't usually allow reliable ordering or grouping.

As with position, all visualizations incorporate marks in some way



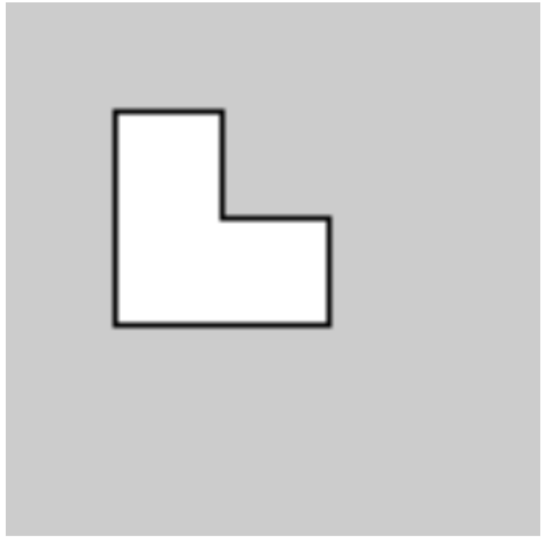
Basic Shape Functions in Processing



```
point (150, 40);  // position x, y

line(80, 80,    220, 160); //start x, y,   end x, y

triangle(90, 275,   150, 180,    210, 275); //3 vertices x,y

rect(80, 320, 140, 55); //top-left corner x, y, width, height

quad(150, 410,    200, 450,   150, 490,   100, 450); //4 vertices

circle(100, 100,   50); //center x, y, radius

ellipse(150, 540,   120, 40);  //center x, y, width height

bezier(185, 600,   110, 590,   190, 670,    115, 660);  //4 control points

arc(150, 740,     80, 80,    0, 3*HALF_PI );   //center x, y, width, height,
                                                //start & end angle
```

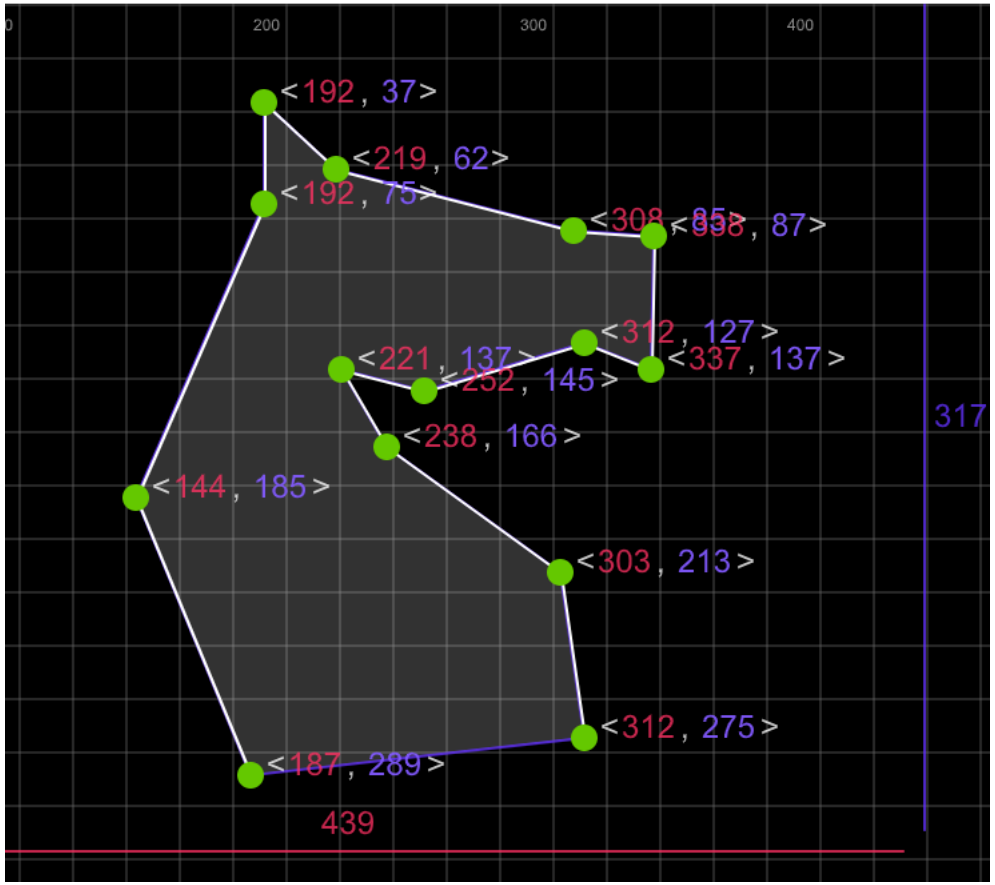
Generalised Shapes

beginShape lets you create arbitrary polygon/polyline shapes built using vertices



```
beginShape();
vertex(20, 20);
vertex(40, 20);
vertex(40, 40);
vertex(60, 40);
vertex(60, 60);
vertex(20, 60);
endShape(CLOSE);
```

Details at: https://processing.org/reference/beginShape_.html

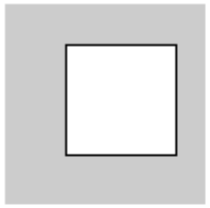


Try this at home: the above image is from a simple online tool (created in Processing) that lets you draw a vector obkect and saves the vertex positions as text. You might use this to create shapes to import into your own programs.
LINK: https://www.scss.tcd.ie/John.Dingliana/p5js/vectors/

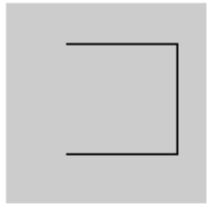
Generalised Shapes

`beginShape()` .. `endShape()` and `vertex( )` are used to construct complex shapes, using primitives e.g.

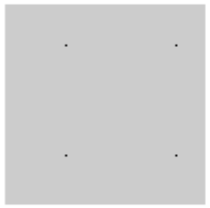
`POINTS, LINES, QUADS, TRIANGLES, TRIANGLE_STRIP, TRIANGLE_FAN, QUAD_STRIP`



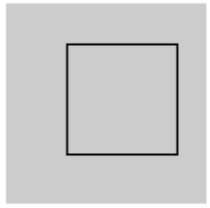
```
beginShape();
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape(CLOSE);
```



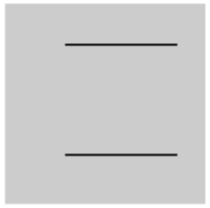
```
noFill();
beginShape();
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape();
```



```
beginShape(POINTS);
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape();
```



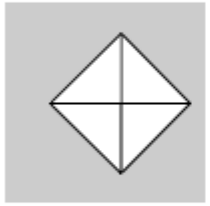
```
noFill();
beginShape();
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape(CLOSE);
```



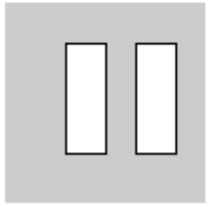
```
beginShape(LINES);
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape();
```

```
beginShape(TRIANGLES);
vertex(30, 75);
vertex(40, 20);
vertex(50, 75);
vertex(60, 20);
vertex(70, 75);
vertex(80, 20);
endShape();
```

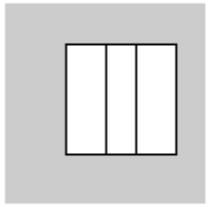
```
beginShape(TRIANGLE_STRIP);
vertex(30, 75);
vertex(40, 20);
vertex(50, 75);
vertex(60, 20);
vertex(70, 75);
vertex(80, 20);
vertex(90, 75);
endShape();
```



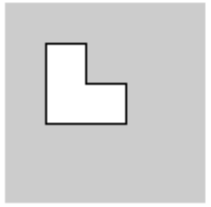
```
beginShape(TRIANGLE_FAN);
vertex(57.5, 50);
vertex(57.5, 15);
vertex(92, 50);
vertex(57.5, 85);
vertex(22, 50);
vertex(57.5, 15);
endShape();
```



```
beginShape(QUADS);
vertex(30, 20);
vertex(30, 75);
vertex(50, 75);
vertex(50, 20);
vertex(65, 20);
vertex(65, 75);
vertex(85, 75);
vertex(85, 20);
endShape();
```



```
beginShape(QUAD_STRIP);
vertex(30, 20);
vertex(30, 75);
vertex(50, 20);
vertex(50, 75);
vertex(65, 20);
vertex(65, 75);
vertex(85, 20);
vertex(85, 75);
endShape();
```



```
beginShape();
vertex(20, 20);
vertex(40, 20);
vertex(40, 40);
vertex(60, 40);
vertex(60, 60);
vertex(20, 60);
endShape(CLOSE);
```

More details at
https://processing.org/reference/beginShape_.html

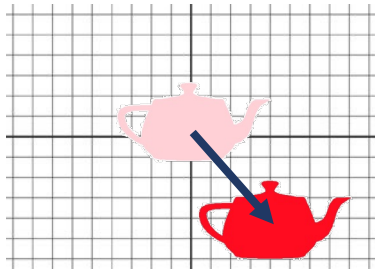
In processing and other graphical languages, it is easier (and typical) to define shapes in "Local Co-ordinates" e.g. centered around <0,0>, then transform the shape

Recall: Transformations

In processing and other graphical languages, it is easier (and typical) to define shapes in "Local Co-ordinates" e.g. centered around <0,0>, then transform the shape

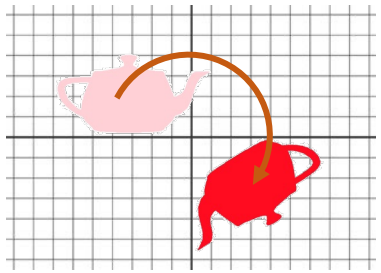
Translate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



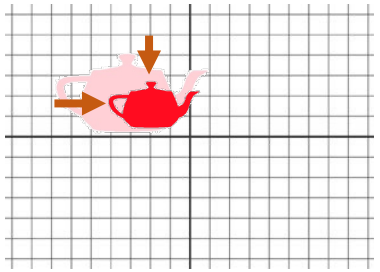
Rotate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

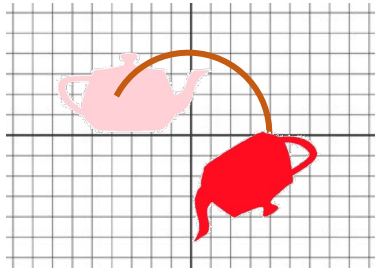


Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Each transform is associated with a specific matrix.
Each point on the object is multiplied by the relevant matrix to transform the whole object.

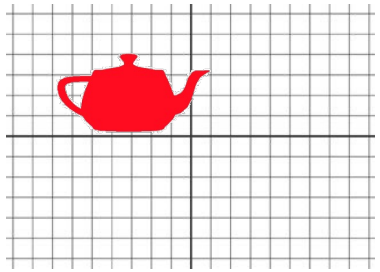


$$\begin{bmatrix} x'_1 \\ y'_1 \\ 1 \end{bmatrix}, \begin{bmatrix} x'_2 \\ y'_2 \\ 1 \end{bmatrix}, \begin{bmatrix} x'_3 \\ y'_1 \\ 1 \end{bmatrix} \cdots \begin{bmatrix} x'_n \\ y'_n \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}, \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}, \begin{bmatrix} x_3 \\ y_3 \\ 1 \end{bmatrix} \cdots \begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix}$$

List of transformed points in world coordinates

Transform

List of points in local coordinates

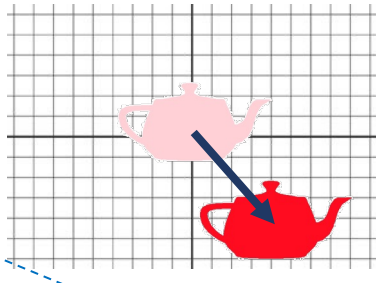


Recall: Transformations

In processing and other graphical languages, it is easier (and typical) to define shapes in "Local Co-ordinates" e.g. centered around <0,0>, then transform the shape

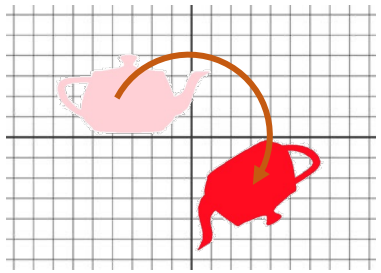
Translate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



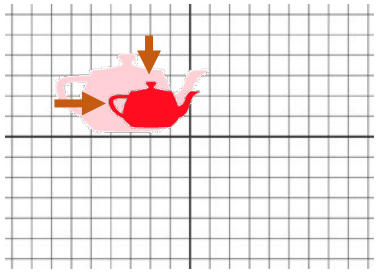
Rotate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

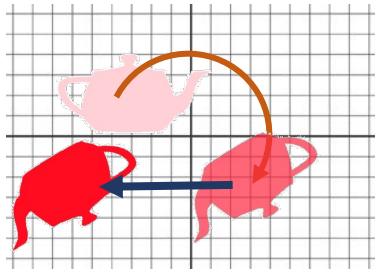


Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Each transform is associated with a specific matrix.
Each point on the object is multiplied by the relevant matrix to transform the whole object.

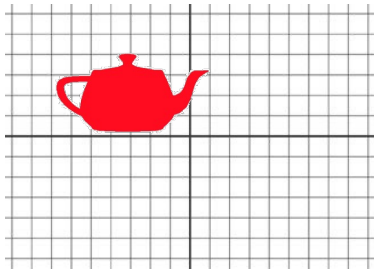


$$\begin{bmatrix} x'_1 \\ y'_1 \\ 1 \end{bmatrix}, \begin{bmatrix} x'_2 \\ y'_2 \\ 1 \end{bmatrix}, \begin{bmatrix} x'_3 \\ y'_1 \\ 1 \end{bmatrix} \cdots \begin{bmatrix} x'_n \\ y'_n \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}, \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}, \begin{bmatrix} x_3 \\ y_3 \\ 1 \end{bmatrix} \cdots \begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix}$$

List of transformed points in world coordinates

Combined transform

List of points in local coordinates



Multiple transforms achieved by concatenating several matrices
But order matters (not commutative)

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \neq \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Transformations

Transforms in Processing:

```
translate(..); //move
rotate(..); //orient/turn
scale(..); //resize
```

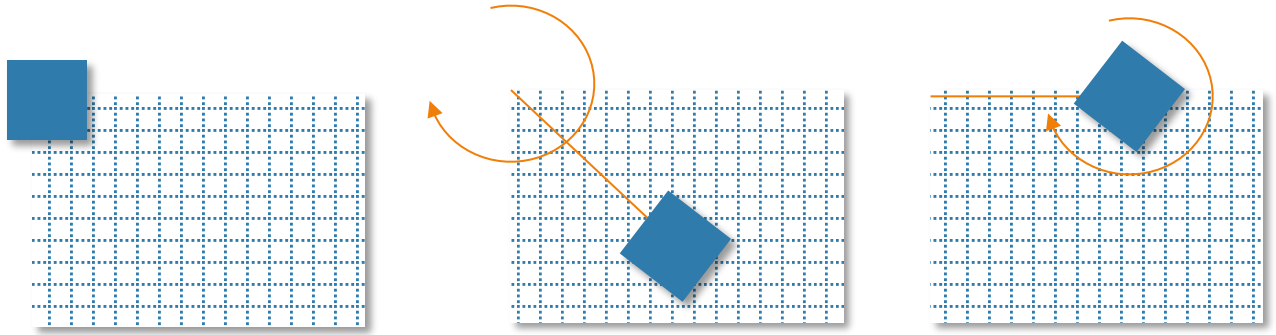
In Processing, these are called before an object is drawn (the combined transform applies to everything drawn after wards)

Additional transforms are stacked upon each other

N.B. Order is important when combining transforms

Transform stack operations

```
pushMatrix(); //save current transform state
popMatrix();  //retrieve last-saved transf. state
```

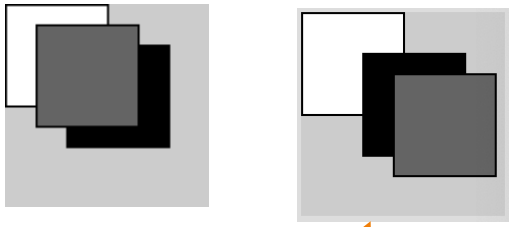


```
rect(-10, -10, 20, 20);
```

```
rotate(PI/4);
translate(40, 0);
rect(-10, -10, 20, 20);
```

```
translate(40, 0);
rotate(PI/4);
rect(-10, -10, 20, 20);
```

```
fill(255); // White rectangle
rect(0, 0, 50, 50);

pushMatrix();
translate(30, 20);
fill(0); );  // Black rectangle
rect(0, 0, 50, 50
popMatrix();

translate(15, 10);
fill(100); // Gray rectangle
rect(0, 0, 50, 50);
```



The image on the right shows the result if the `pushMatrix / popMatrix` calls are removed

Shape and Transforms

Note that the code for loading the data is omitted here for brevity and is identical to that in Slide 5.


```
background(255);
noStroke();
fill(0, 100);

for (int i=0; i<N; i++)
{
  pushMatrix();
     //save current transform state
  translate(x[i], y[i]);

  switch(c[i]) //map categories to shape
  {
  case 0:
    circle(0, 0, 20);
    break;
  case 1:
    rect(-5, -10, 10, 20);
    break;
  case 2:
    rect(-10, -5, 20, 10);
    break;
  case 3:
    triangle(-10, 5, 0, -10, 10, 5);
    break;
  case 4:
    quad(-10, 0, 0, 10,
         10, 0, 0, -10);
    break;
  case 5:
    beginShape();
    vertex(-10, -10);
    vertex( 10, -10);
    vertex( 10, 10);
    vertex(0, 0);
    vertex(-10, 10);
    endShape();
  case 6:
    arc(0, 0, 20, 20, 0, PI );
    break;
  case 7:
    arc(0, 0, 20, 20, PI/2, PI );
    break;
   }
  popMatrix();
    //retrieve saved transform state
}
```

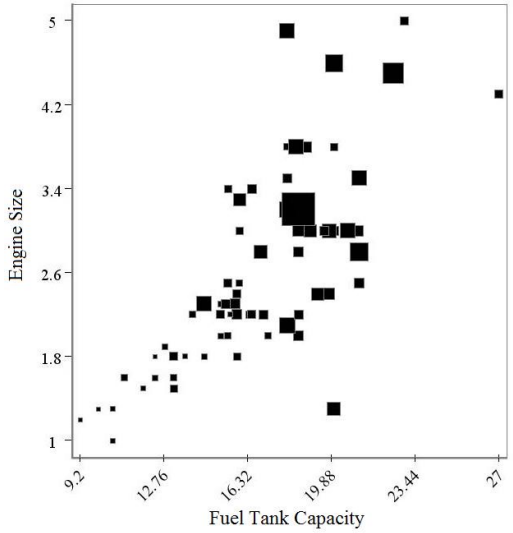
Without transforms, we would need to explicitly re-calculate and enter the coordinates of each point on each shape

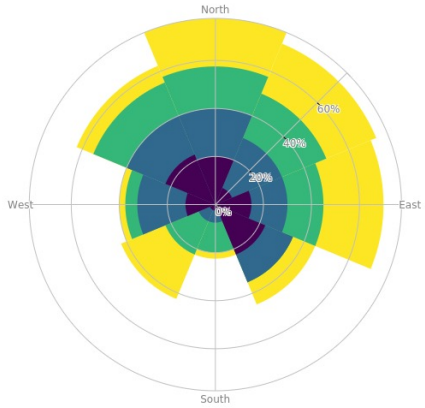
Without **`push/popMatrix`**, each successive translation would be cumulative (relative to the previous position) which is not what we want

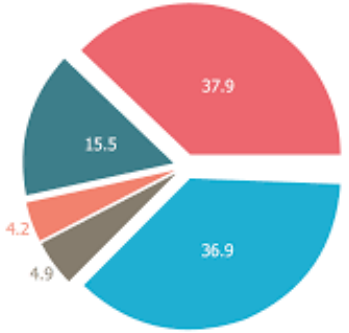
3. Size

Represent data through how large/small a mark is drawn

- Data values expressed as length, area or volume
- Easy to see whether one is bigger
- Good for interval and continuous variables





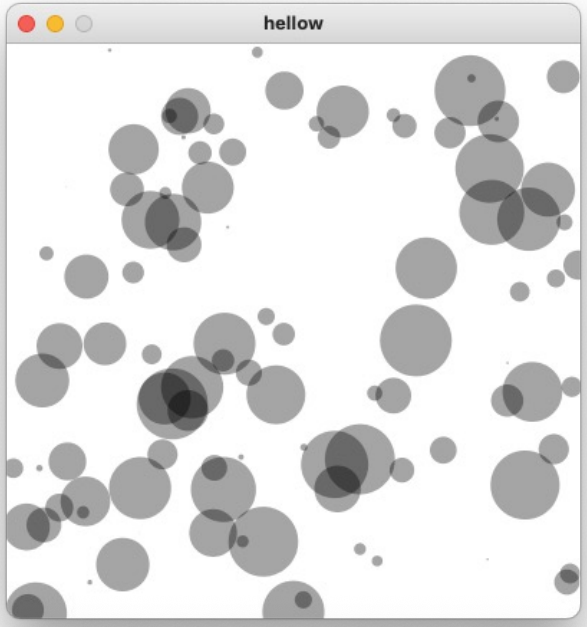


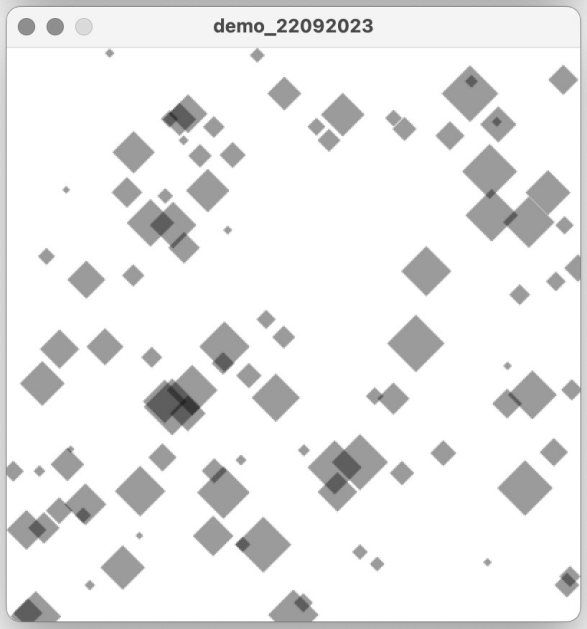
SIZE

```
background(255);
noStroke();
fill(0, 100);

for (int i=0; i<N; i++)
{
  float rad = map(v[i], 0, 1, 0, 50);
  circle(x[i], y[i], rad);
}
```

There are various ways to encode size.

In the example above (and result shown in the top figure), the radius of the circle is mapped to a quantitative attribute in the data





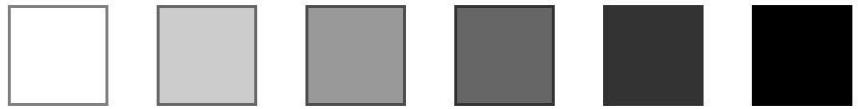
Note that the code for loading the data is omitted here for brevity and is identical to that in Slide 5.
Code is written for clarity not efficiency.

In the more generalized example below, transformations are used to (a) scale the shape based on a the attribute v, and (b) to translate the shape based on attributes x, and y.

```
background(255);
noStroke();
fill(0, 100);

for (int i=0; i<N; i++)
{
  float s = map(v[i], 0, 1, 0.25, 2);

  pushMatrix();
  translate(x[i], y[i]);
  scale(s);
  quad(-10, 0, 0, -10, 10, 0, 0, 10);
  popMatrix();
}
```

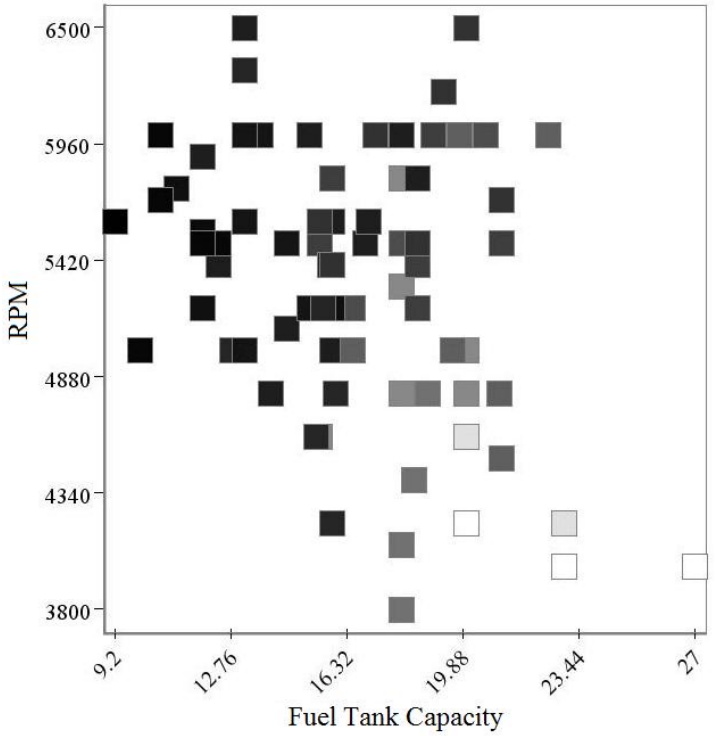
The **map** function, rescales a value from one range to another. See: https://processing.org/reference/map_.html

4. Brightness / Value



Represent data through how a mark is shaded

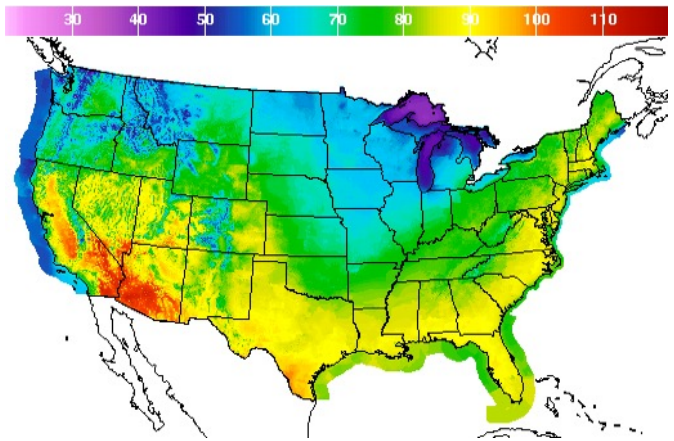
- Also called Value/ Intensity / Luminance
- OK for quantitative data
- But not very many shades distinguishable to human eye
- Visual response is not (always) linear

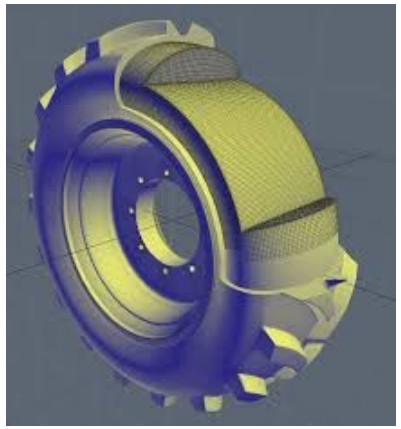


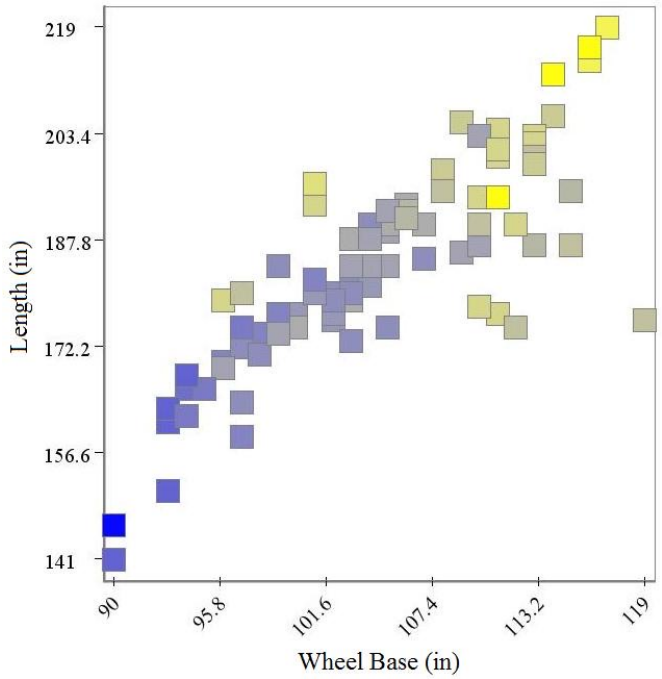
5. Colour

Represent variables through colour

- In visualisation, this often implies hue and saturation (n.b. brightness is also a dimension of colour)
- Good for qualitative data (identity channel)
- Does not work well for quantitative data! Lots of pitfalls! Be careful!





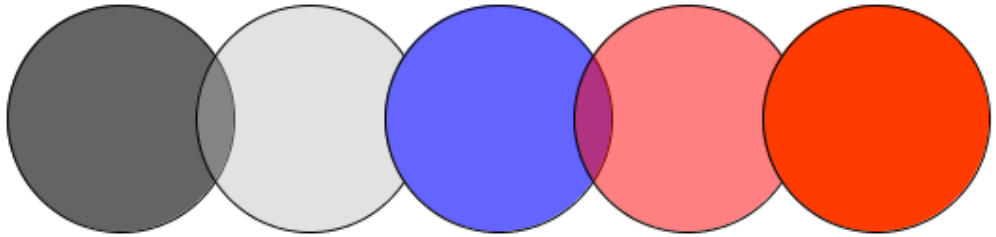


Colour in Processing

Processing provides a built-in variable type for dealing with color

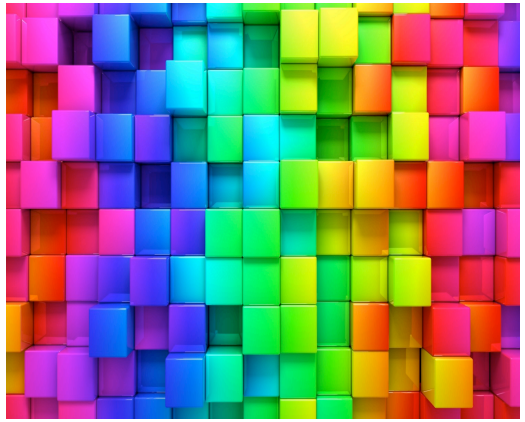
- 32-bit Colour Supported: 1 byte for three components and 1 byte for opacity/alpha
- By default colors represented as a “Red, Green, Blue” (RGB) triple (each in the range 0-255)
- If four values are used, the fourth is “alpha” or an opacity/transparency value
- If a single value is provided it is considered grey scale : red=green=blue=grey value
- Also supports Hue, Saturation, Brightness (HSB) color mode

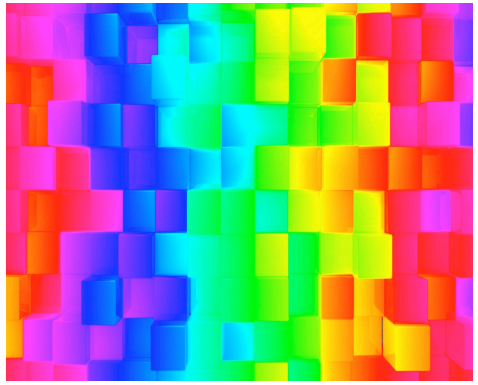
```
color c1 = color (100); //equivalent to color c1 = (100, 100, 100);
color c2 = color (180, 100); //gray = 180, alpha = 100
color c3 = color (100, 100, 255); //red = 100, green = 100, blue = 255
color c4 = color (255, 0, 0,. 127); //red = 255, green = 0, blue = 0, aplha = 127
colorMode(HSB, 255); color c5 = color (10, 255, 255);
                                    //hue = 10, saturation = 255, brightness = 255
```



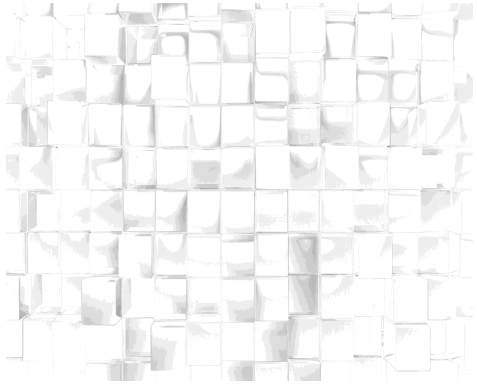
N.B. Color will be the topic of more extensive later lecture. In the meantime, for more details see: https://processing.org/tutorials/color/

Hue Saturation and Brightness (HSB) Model

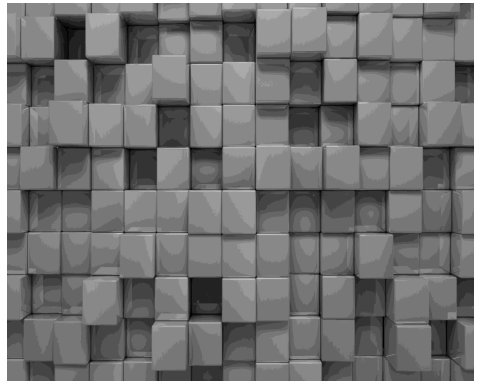




Hue
(Principal wavelength)



Saturation
(purity : pure to washed out)

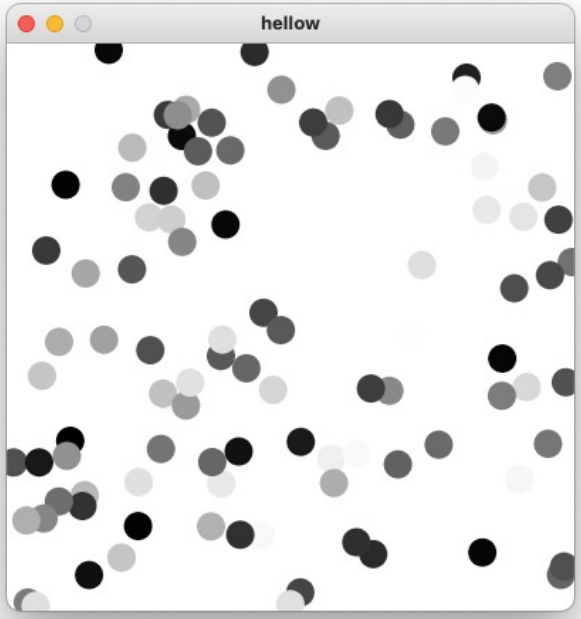


Brightness
(Overall intensity)

Typical Interface for choosing colour in Modelling tools:
Hue chosen by angle, saturation by radial distance, brightness on a separate slider

Brightness

Note that the code for loading the data is omitted here for brevity and is identical to that in Slide 5.

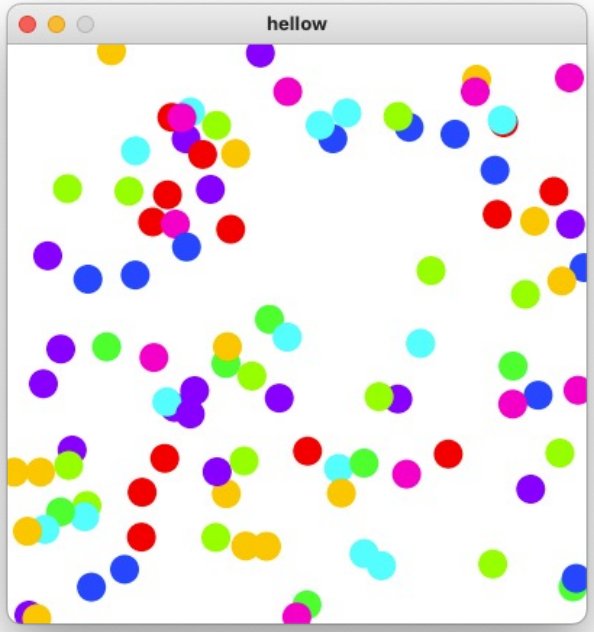



```
background(255);
noStroke();

float grayvalue;

for (int i=0; i<N; i++)
{
  grayvalue = map(v[i], 0, 1, 0, 255);
  fill(grayvalue);
  ellipse(x[i], y[i], 20, 20);
}
```

In this example the attributes x and y are mapped to positions in 2D, and the quantitative attribute v is mapped to the brightness of the circles.

Colour (Hue)

Note that the code for loading the data is omitted here for brevity and is identical to that in Slide 5.



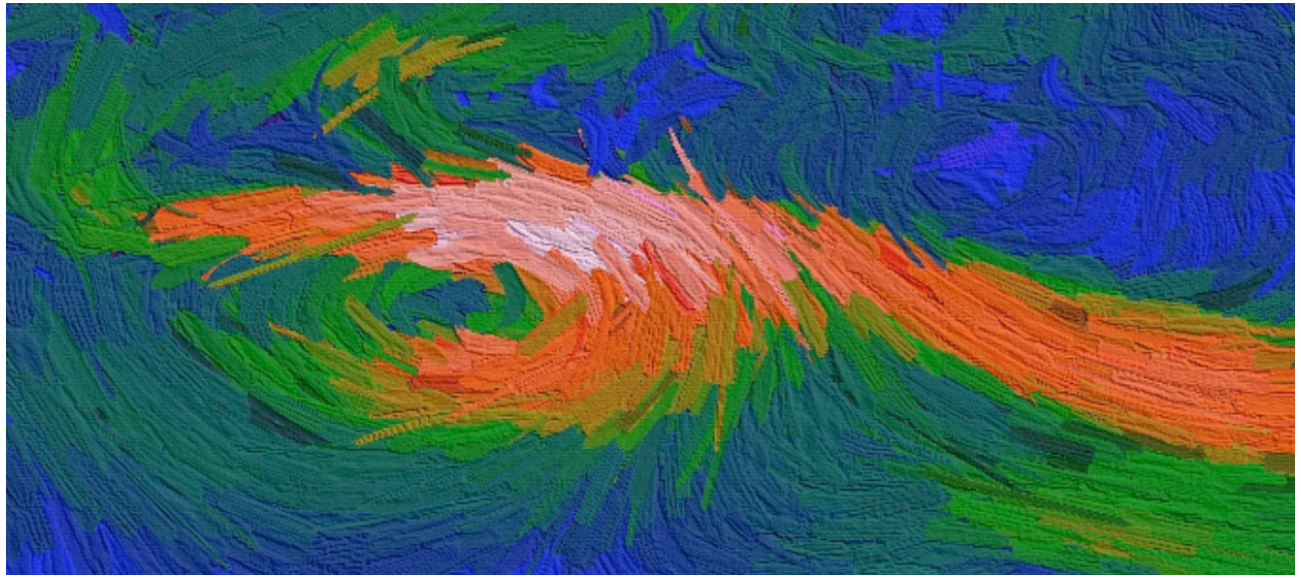

```
background(255);
noStroke();

colorMode(HSB);
float huevalue;

for (int i=0; i<N; i++)
{
 huevalue = c[i] * 255/8;
 fill(huevalue, 255, 255);
 ellipse(x[i], y[i], 20, 20);
}
```

In this example the attributes x and y are mapped to positions in 2D, and the categorical attribute c is mapped to the brightness of the circles.

6. Orientation/Direction

Mark is rotated in connection to the data variable

- Good sense of similarities/distinct orientations BUT limited precision for quantitative data



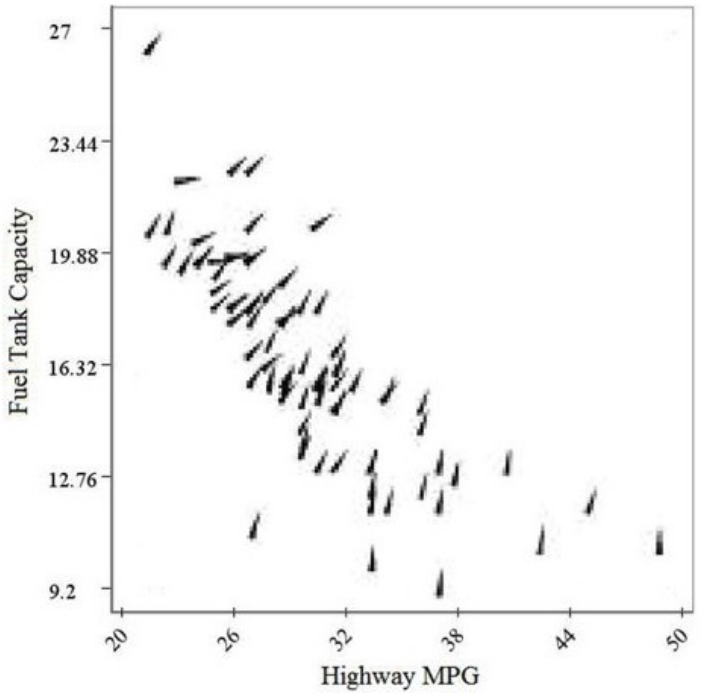
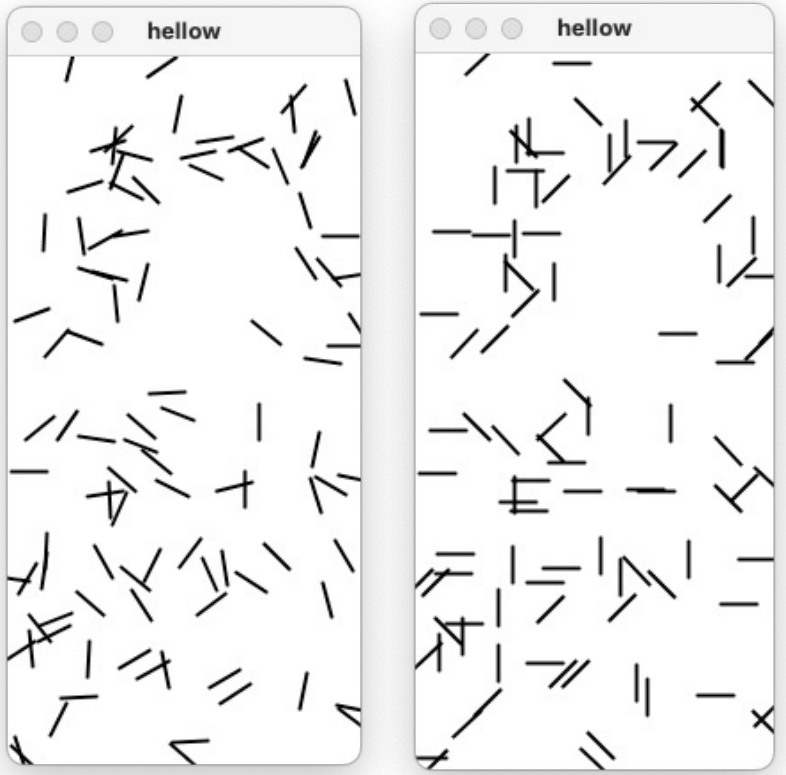


Image from Ward et al 2014

Orientation

Note that the code for loading the data is omitted here for brevity and is identical to that in Slide 5.

```
background(255);
stroke(0);
strokeWeight(2);

float angle;

for (int i=0; i<N; i++)
{
  pushMatrix(); //save transform

    translate(x[i], y[i]);
    angle = map(v[i], 0, 1, 0, 360);
    rotate(angle);
    line(0, -10, 0, 10);

  popMatrix(); //recall saved transform
}
```



Encoding Values by angle

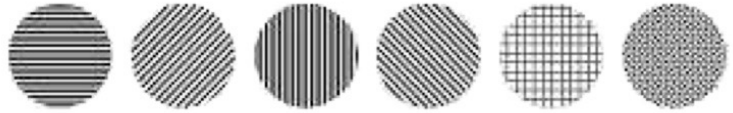
Encoding Categories by angle

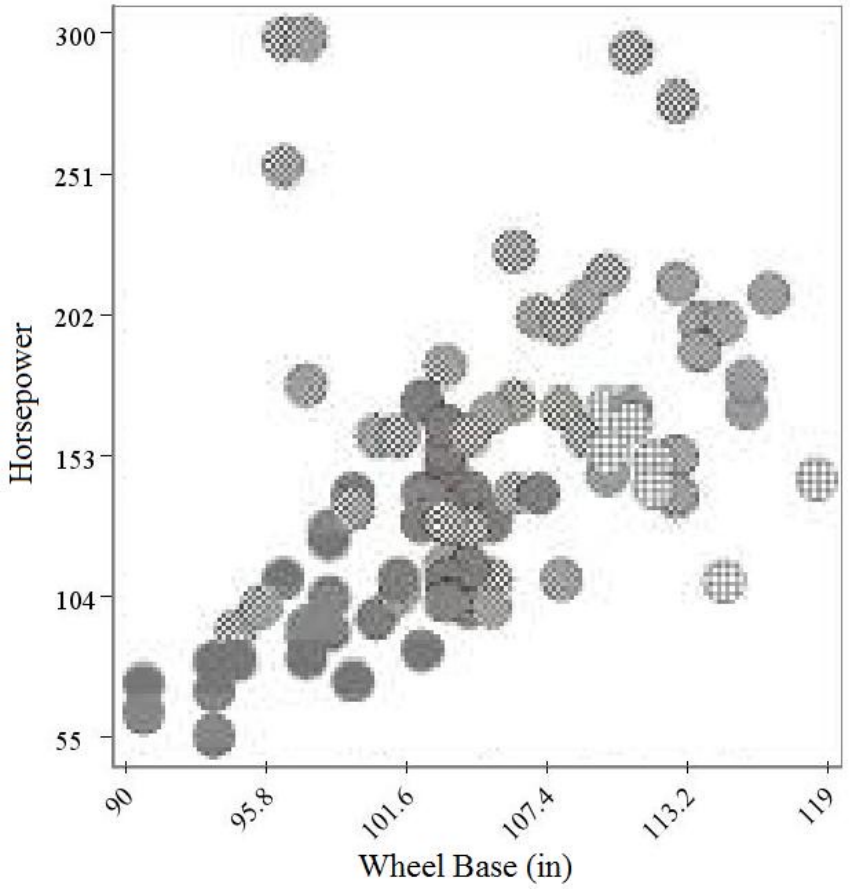
Replace this with
`angle = c[i] * PI/4;`
to encode categories c[i] instead of values v[i]

7. Texture

Visual pattern to communicate data

- Can be considered a combination of other variables (mark, color, orientation)
- Most commonly associated with polygon, region or surface





Raster Images

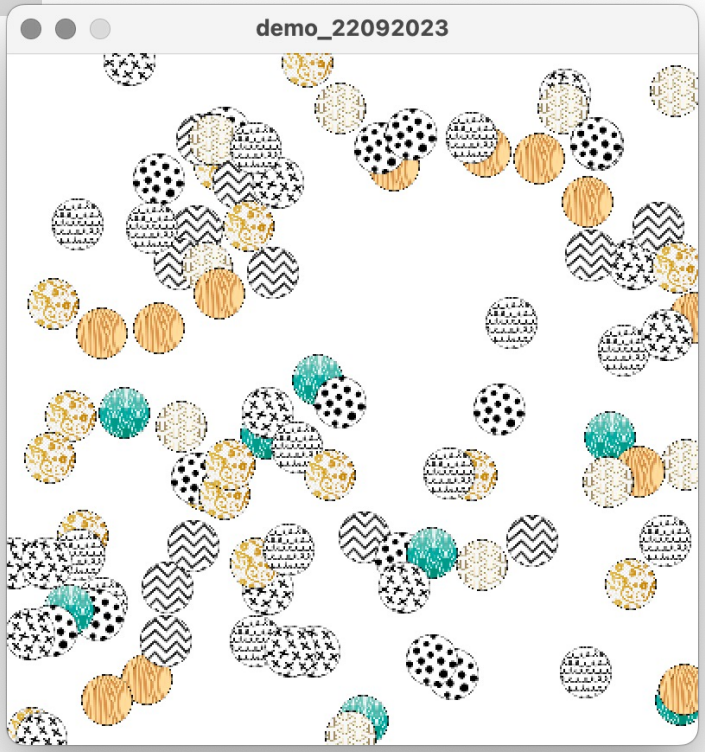
Processing provides a built-in image class, **`PImage`**, for dealing with images:

- Numerous powerful raster related operations can be applied to this (e.g. filters, transforms, etc.)
- Basic example shown below

```
PImage photo; //declare variable for PImage

photo = loadImage("laDefense.jpg"); //load image from file
size(100, 100); //set the window/canvas size
image(photo, 0, 0); //draws image with top left corner at origin <0,0>
```

More details: https://processing.org/reference/PImage.html

Texture / Pattern



In this example, for simplicity we are merely using 8 images of circles with pre-rendered texture patterns

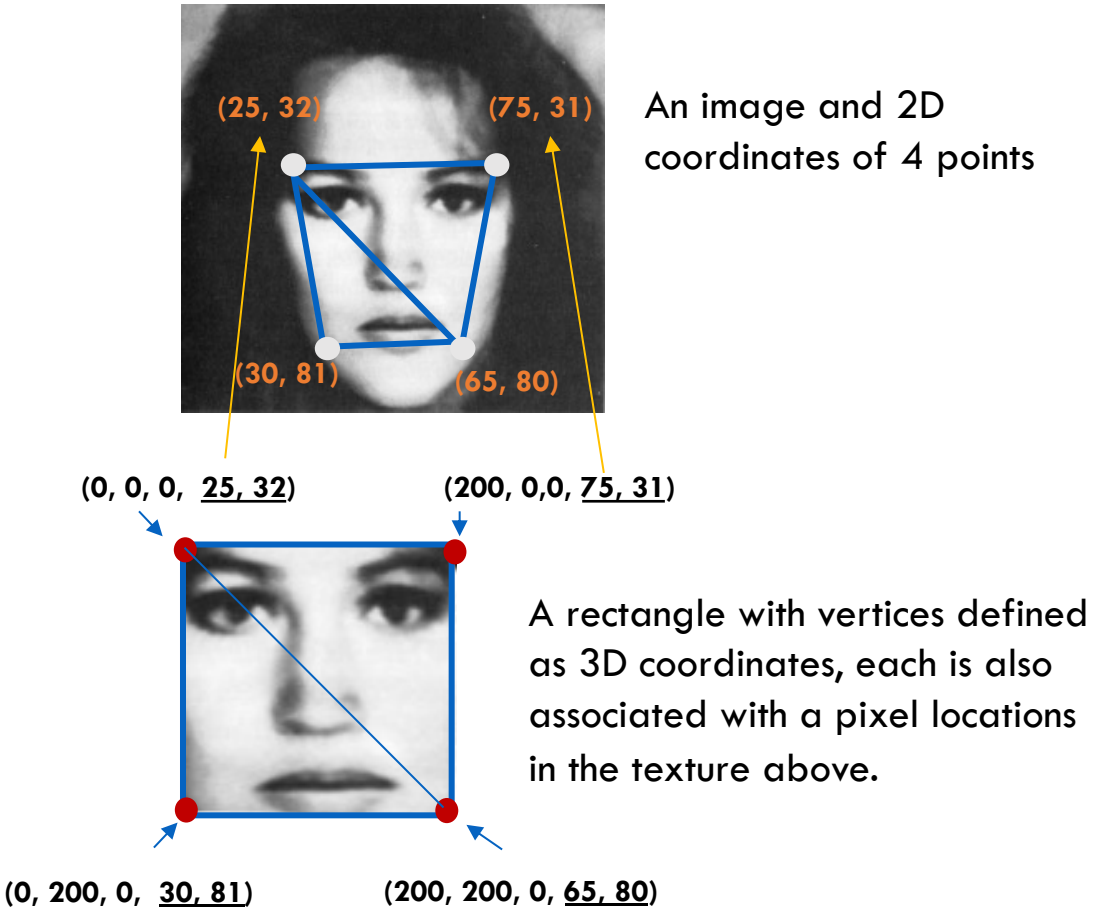
More generally, shapes created using the **beginShape()** function can be "texture mapped" with any image.

Note that the code for loading the data is omitted here for brevity and is identical to that in Slide 5.

```
PImage[] img;

img = new PImage[8];
img[0] = loadImage("tex0.png");
img[1] = loadImage("tex1.png");
img[2] = loadImage("tex2.png");
img[3] = loadImage("tex3.png");
img[4] = loadImage("tex4.png");
img[5] = loadImage("tex5.png");
img[6] = loadImage("tex6.png");
img[7] = loadImage("tex7.png");

background(255);
noStroke();
imageMode(CENTER);
  //[optional] this causes positioning by center of
  //image rather than the default by top-left corner

for (int i=0; i<N; i++)
{
  image(img[c[i]], x[i], y[i], 30, 30);
}
```

Texture Mapping Example

```
PImage a = loadImage("face.jpg");

beginShape(QUADS);
 texture(a);
 vertex(0,0,0,    25, 32);
 vertex(200, 0, 0,  75, 31);
 vertex(200, 200, 0,   65, 80);
 vertex(0, 200, 0,    30, 81);

endShape();
```



An image and 2D coordinates of 4 points

A rectangle with vertices defined as 3D coordinates, each is also associated with a pixel locations in the texture above.

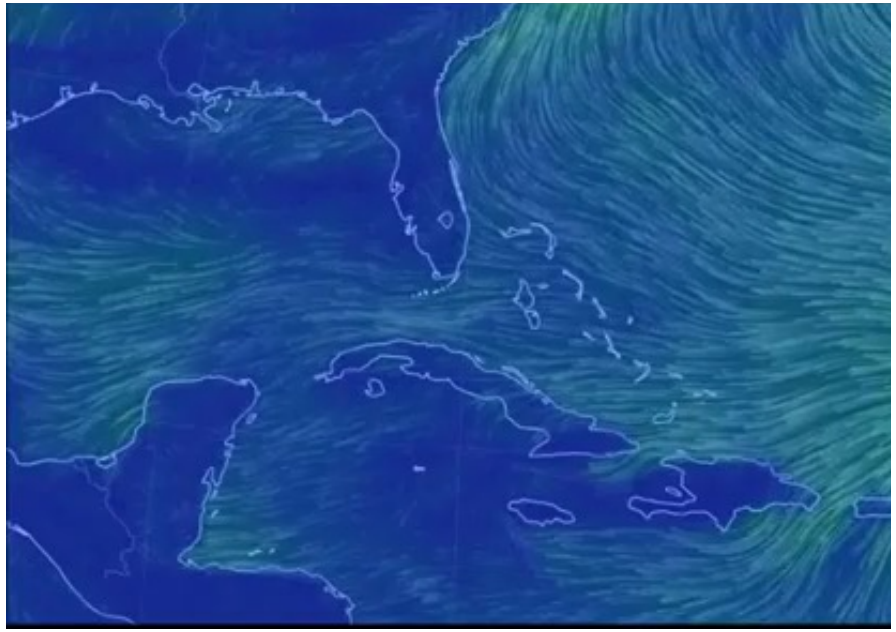
N.B. The **texture()** function must be called between **beginShape()** and **endShape()** and before any calls to **vertex()**.

More details: https://processing.org/reference/texture_.html

8. Motion

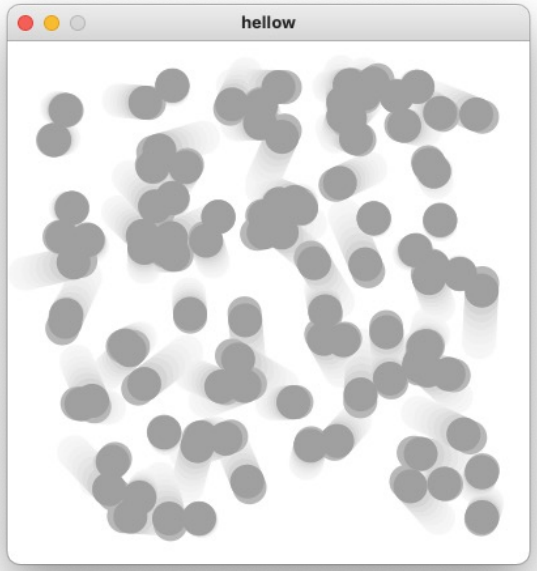
Variables represented by motion

- Can be associated with any other visual variable e.g. position change, flashing (varying opacity)
- Information conveyed through changes over time
  - Speed of change
  - Direction of motion
- Outliers tend to pop-out, grouping due to similarities



Direct data encoding to motion [https://earth.nullschool.net/]

Motion



For convenience, the previous examples have been in Processing's ***static mode*** (a sequential list of functions executed once).

For motion (animation), we need to use processing's **interactive mode**. This follows an event–driven paradigm, where we specify call-back functions called in response to various system events e.g. setup is called at launch of a program, draw is called every time the program needs to be drawn (by default a repetitive loop).

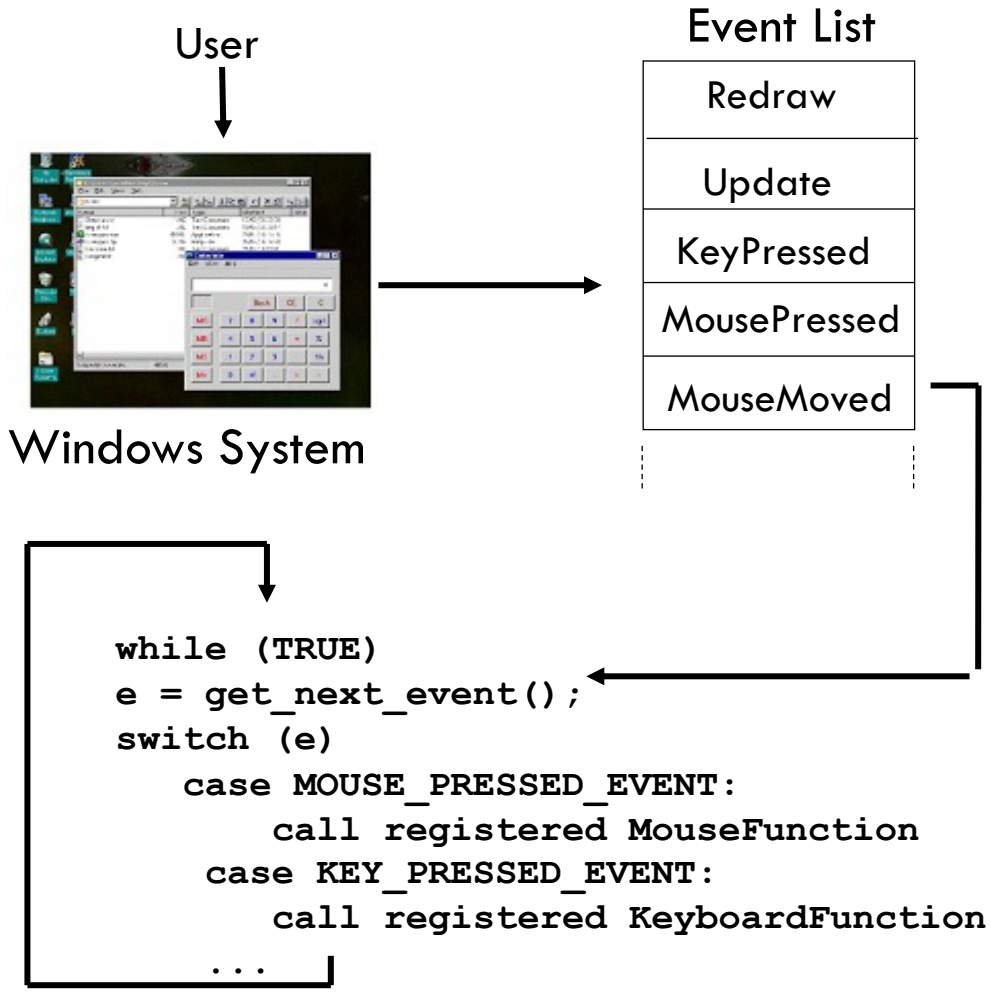
```
Table table;
float[] x, y, v;
int[] c;
int N;
float dir = 1;
float phase = -8;
void setup()
{
  size(400, 400);
  table = loadTable("sample.csv", "header");

  N = table.getRowCount();
  x = new float[N];
  y = new float[N];
  c = new int[N];
  v = new float[N];

  for (int i=0; i<N; i++)
  {
    TableRow row = table.getRow(i);
    x[i] = row.getFloat("X");
    y[i] = row.getFloat("Y");
    c[i] = row.getInt("Category");
    v[i] = row.getFloat("Value");
  }
}
```

```
void draw()
{
  background(255);
  fill(125);
  noStroke();
  float angle;

  for (int i=0; i<N; i++)
  {
    pushMatrix(); //save current tranform
    translate(x[i], y[i]);
    angle = map(c[i], 0, 8, 0, 2*PI);
    rotate(angle);
    translate(phase*v[i], 0);
    ellipse(0, 0, 26, 26);
    popMatrix(); //retrieve saved transform
  }

  phase += dir; //update animation stage

  //flip direction periodically
  if (phase>7 && dir>0)
    dir = -1;
  if (phase<-7 && dir<0)
    dir = 1;
}
```

This specific example draws a circle oscillating about the x-y position, at a speed proportional to the value v[i] and at an angle determined by r[i].

Event-driven Programming

```
void setup()
{
   size(800, 800);
}
void draw()
{
  rect(10, 10, 780, 780);
}
void keyPressed()
{
  if (key=='b')
    background(255, 0, 0);
    redraw();
}
void mouseMoved()
{
   ellipse(mouseX, mouseY, 10, 10);
}
```



Event Functions

- The **`setup()`** function is where you state operations you want performed once when your program begins
- The **`draw()`** function gets called repetitively (unless you stop it)
  - [optional] Within the program **`noloop()`** stops redrawing **`loop()`** restarts it
- Several mouse callbacks including **`mousePressed() , mouseReleased, mouseMoved(), mouseDragged()`** :
- Mouse positions are returned in **`mouseX, mouseY`**
- The **`keyPressed()`** function gets called when you press a key. You can get the key pressed using the global variable **`keyPressed`**

System Variables

Typically for use with event handlers

All global variables

```
void setup()
{
  size (400, 400);
  circle(width/2, height/2, width);
}

void draw()
{
}

void mouseDragged()
{
  circle(mouseX, mouseY, 10);
}
```

Type	Variable	Description
boolean	keyPressed	Set to true when a key is pressed, otherwise false
char	key	character that was typed or CODED if a special key was pressed
int	keyCode	If a special key is pressed this may be set to one of the following UP, DOWN, LEFT, RIGHT, CTRL, SHIFT or ALT
boolean	mousePressed	true (when mouse button is pressed) otherwise false
int	mouseX	Current X and Y position of mouse
int	mouseY	
int	pmouseX	Previous X and Y position of mouse
int	pmouseY	
int	mouseButton	Set to LEFT, RIGHT or CENTER when mouse is pressed
int	width	width of the viewport (or canvas)
int	height	height of the viewport

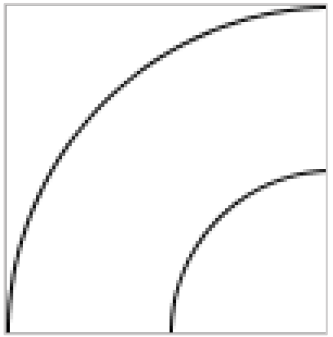
Other Relevant Functions in processing

Saving Output From Processing

See: https://processing.org/reference/save_.html
https://processing.org/reference/libraries/pdf/index.html

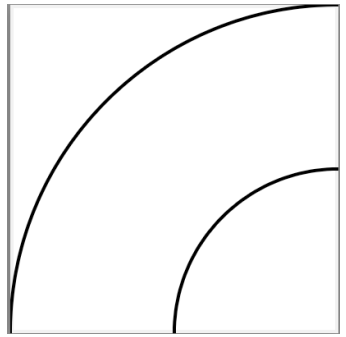
```
void setup() {
        size(100, 100);
}

void draw() {
  background(255);
  noFill();
  circle( width, height,    width*2);
  circle( width, height,    width);
  save("filename.png");
  // Exit the program
  println("Finished.");
  exit();
}
```

By default, saving a file in Raster format creates an image that looks fine in its native resolution but when scaled up, starts to look blocky (aliasing)



```
import processing.pdf.*;

void setup() {
  size(100, 100, PDF, "filename.pdf");
}

void draw() {
  background(255);
  noFill();
  circle( width, height,    width*2);
  circle( width, height,    width);

  // Exit the program
  println("Finished.");
  exit();
}
```

In contrast vector objects saved in a vector format such as PDF remain smooth at any resolution



Output to SVG

Processing provides a library for outputting visualizations in vector format as SVG files, which may come in handy for lossless export of visualizations

```
import processing.svg.*;

void setup() {
  size(100, 100, PDF, "filename.svg");
}

void draw() {
  background(255);
  noFill();
  circle( width, height,    width*2);
  circle( width, height,    width);

  // Exit the program
  println("Finished.");
  exit();
}
```

The PShape class in processing can be used for loading and drawing SVG formatted images (amongst other things)

```
PShape s;

void setup() {
  size(240, 320);
  s = loadShape("Marilyn-Monroe.svg");
}

void draw() {
 shape(s, 10, 10, 220, 300);
}
```

Text And TextMode in processing

Text in processing:
https://processing.org/reference/text_.html

```
size(400,400);
PFont mono;

mono = createFont("arial", 128);
background(0);
textFont(mono);
text("word", 48, 240);
```

The TextMode function allows saving text as vector format

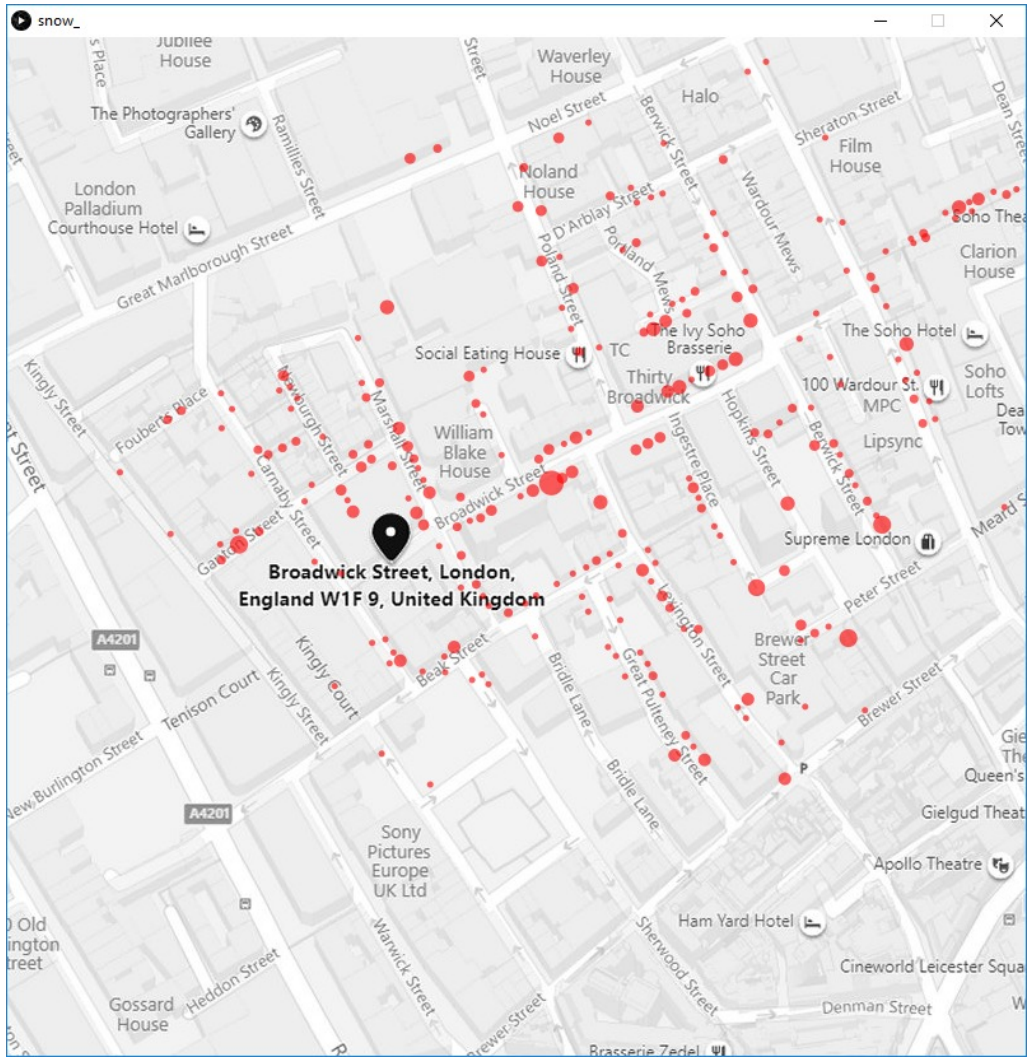
See: https://processing.org/reference/textMode_.html

```
import processing.pdf.*;

void setup() {
  size(500, 500, PDF, "TypeDemo.pdf");
  textMode(SHAPE);
  textSize(180);
}

void draw() {
  text("ABC", 75, 350);
  exit();  // Quit the program
}
```

John Snow's Map

Solution Walkthrough

Informal Practical



This is a taster assignment for practice. No marks or deadline for this. No need to submit.

But it may help with the next assignment and lecture to have tried this.

Motivations:

- Practical aspects: try out Processing (p5), get started visualizing stuff
- Visualization aspects: positional encoding, spatial data

Objectives:

- Create a version of John Snow's famous visualization of the cholera outbreak in Broad Street (now broadwick street).
- Essentially, you should load an image of a blank map and draw circles (ellipses) at every location where a death is recorded. N.B. you don't need to emulate Snow and draw black rectangles
- Optionally you may play with colour and size (e.g. scale circle based on number of deaths) to improve the visualization

Getting Started

Get the program from:

- http://www.processing.org/download/
- includes windows, mac and linux versions
- OR run it online at https://processing.org/

Look at some examples/tutorials from the following slide

```
int u = 60;
boolean showGrid = true;

void setup() {
  size(600, 600);
}

void draw() {
  background(255);
  if (showGrid) drawGrid();
  strokeCap(SQUARE);
  strokeWeight(1.5 * u);

  stroke(5, 100, 255);
  bezier(4 * u, 1 * u, 7 * u, 1 * u, 7 * u, 5 * u, 4 * u,
5 * u);

  stroke(30, 50, 170);
  line(1 * u, 6 * u, 4 * u, 2 * u);

  stroke(130, 175, 255);
  line(1 * u, 3 * u, 2 * u, 5 * u);
}

void drawGrid() { ··· }
```

Reset

Close Editor

Tutorials

Some basic tutorials available at: https://processing.org/tutorials/

The following are relevant for this assignment

- https://processing.org/tutorials/gettingstarted/
- https://processing.org/tutorials/overview
- https://processing.org/tutorials/color/
- https://processing.org/tutorials/text/
- https://processing.org/tutorials/transform2d/

Reference

Language Reference here: https://processing.org/reference/

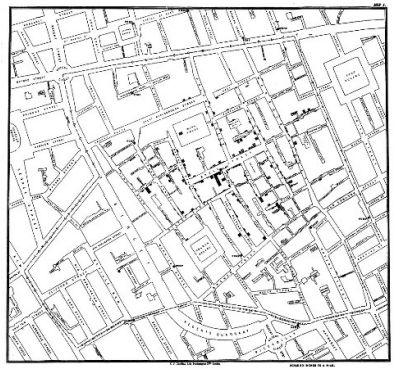
- You will probably need to look up the following functions: `fill, stroke, loadTable, ellipse, circle, rect, loadImage, Pimage`
- For saving to PDF see: https://processing.org/reference/libraries/pdf/index.html

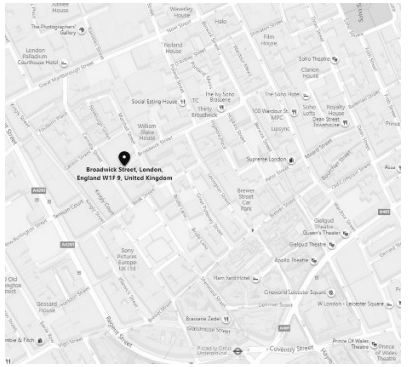
Write some basic code to do the following [N.B. the links provide explanation and examples]

- Open and **size** a window: https://processing.org/reference/size_.html
- Load and draw an image: https://processing.org/reference/loadImage_.html
- Load and read data a CSV file: https://processing.org/reference/loadTable_.html
- Draw some basic shapes: https://processing.org/reference/ellipse_.html

Snow : Background and Assets

Available on blackboard and at https://www.scss.tcd.ie/John.Dingliana/cs7ds4





You may want to read a bit about Snow's famous visualization: https://ralucanicola.github.io/cholera-map-3D/

map.jpg is an image of Snow's Visualization already marked with deaths

mapclean.jpg is a blank and more modern mapping of the area (in the same scale as above)

Cholera data set : snowpixelcoords.csv

- count of deaths at a particular location x and y.
- Original data is included (columns 2 and 3) but you should ignore this as the exact scale is unknown
- Instead look at columns 4 and 5 (x_screen and y_screen) which are a cleaned up version in the pixel coordinates that already match the maps images above
- A count of -999 indicates that this is the location of a water pump (optional to visualize this)

Don't use these columns

count	geometryx	geometryy	x_screen	y_screen
3	-0.13793	51.513418	314.4917326	321.4091085
2	-0.137883	51.513361	319.4455326	331.0991085
1	-0.137853	51.513317	322.6075326	338.5791085
1	-0.137812	51.513262	326.9289326	347.9291085
4	-0.137767	51.513204	331.6719326	357.7891085
1	-0.13678	51.514058	435.7017326	212.6091085
3	-0.136696	51.514148	444.5553326	197.3091085
1	-0.136712	51.513961	442.8689326	229.0991085
2	-0.13617	51.513945	499.9957326	231.8191085
5	-0.135485	51.513821	572.1947326	252.8991085
5	-0.135374	51.513999	583.8941326	222.6391085
3	-0.135582	51.513795	561.9709326	257.3191085
3	-0.135679	51.513766	551.7471326	262.2491085
1	-0.135814	51.513726	537.5181326	269.0491085
...	...	...	...	...
-999	-0.133962	51.510019	732.7189326	899.2391085
-999	-0.138199	51.511295	286.1391326	682.3191085

Basic Steps

Create a canvas

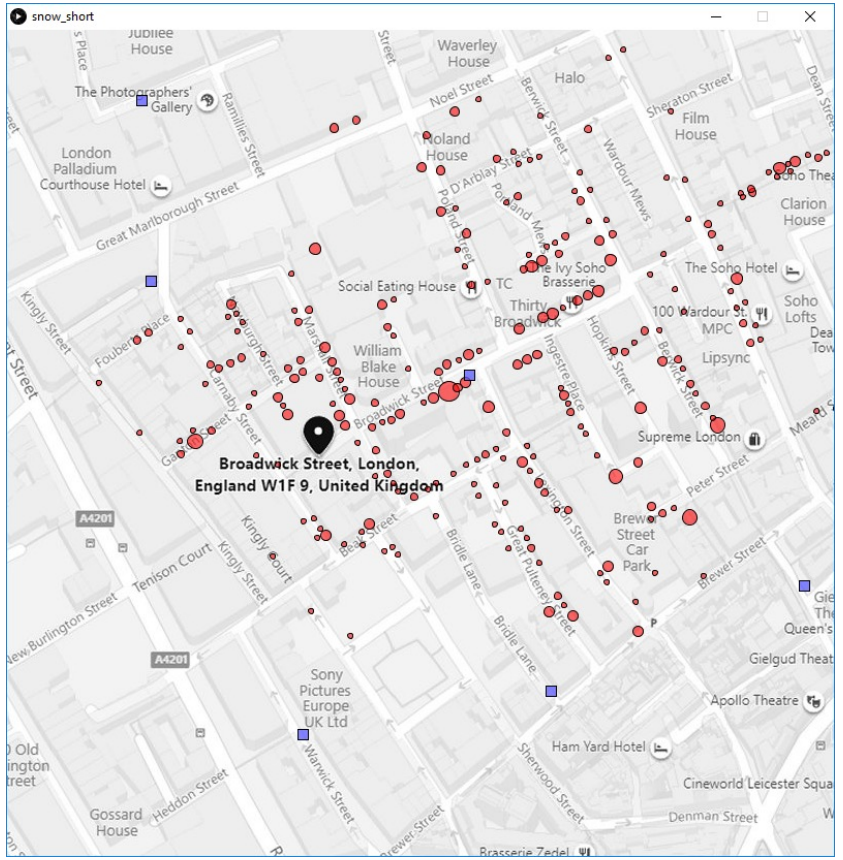
Load and draw the map image

Load the data file

For each row

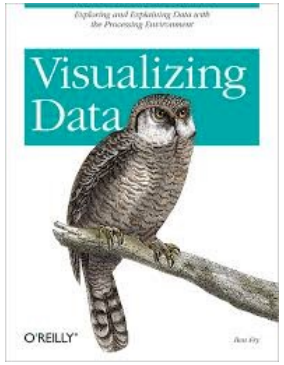
if num_deaths >0 draw circle in red with radius proportional to deaths

if numdeaths == -999 draw blue square

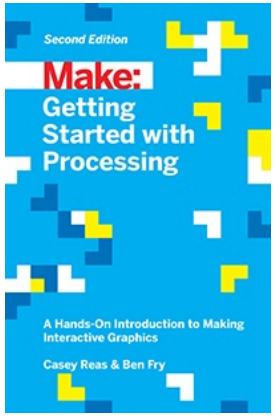


Other References

Visualizing Data: Exploring and Explaining Data with the Processing Environment 1st Edition by Ben Fry



Make: Getting Started with Processing, Second Edition Casey Reas and Ben Fry.



Tutorials: https://processing.org/tutorials/

Reference manual online: http://www.processing.org/reference/

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

The End

For now