Guessing intelligently (interpolation)

Prof. Graeme Bailey

http://cs1114.cs.cornell.edu

(notes modified from Noah Snavely, Spring 2009)

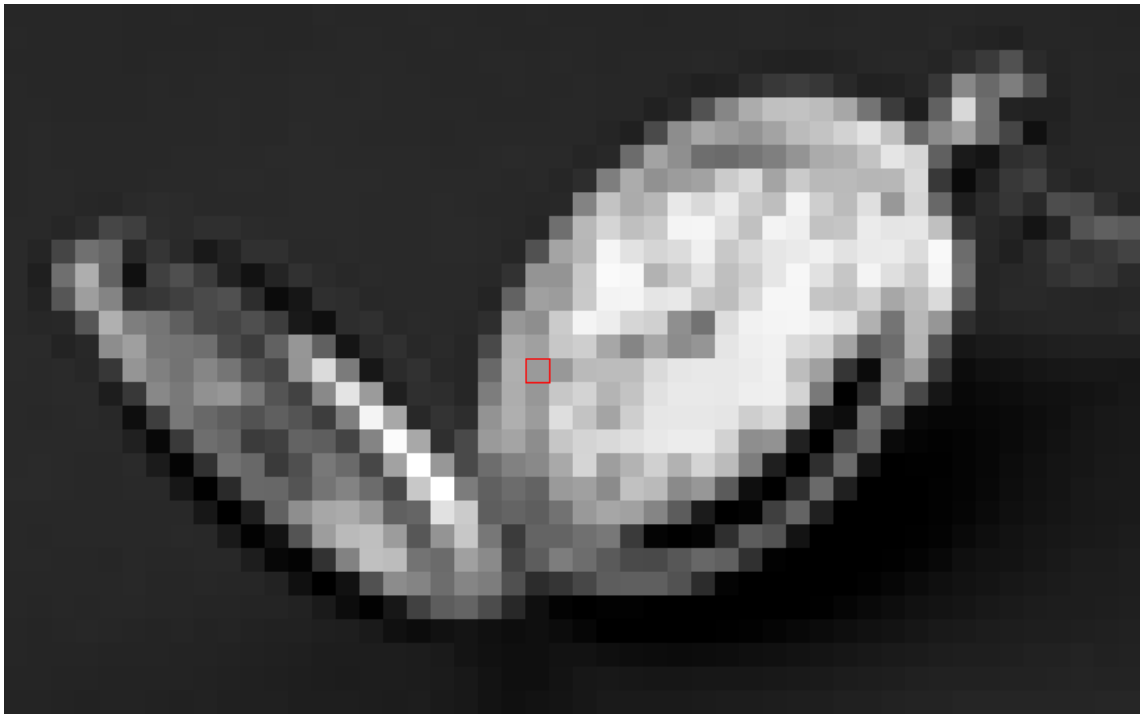
Cornell University
Computer Science

Manipulating images

- These photos are too small:

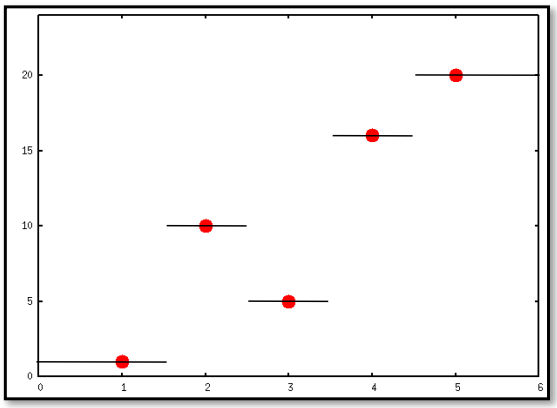
- How can we enlarge them?
- What should be the 'missing' values? Constructing these is called interpolation – there are many ways to do this.
- Simple method 1: repeat the rows and columns – *constant* interpolation.

Zooming: First attempt



Constant Interpolation

- We used the value of the data point we were closest to
- Suppose we know values of y=f(x) as graphed in red:



- We simply make the intermediate values be those of their nearest neighbour.

Interpolation

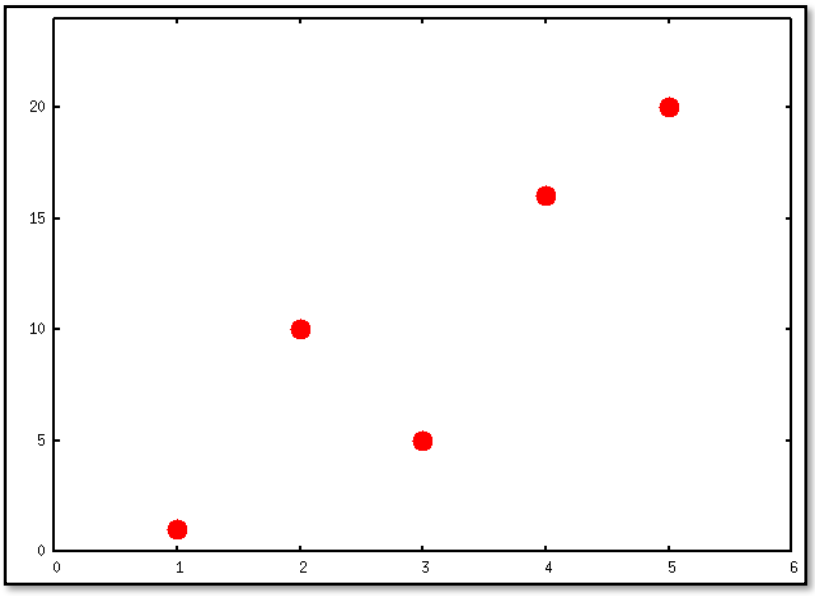
- Problem statement:
  - We are given the values of a function *f* at some scattered locations, e.g., *f*(1), *f*(2), *f*(3), …
  - We want to choose the 'missing' values intelligently
  - For example, what is *f*(1.5)?

- This is called *inter*polation
- Guessing values outside our range is called *extra*polation
- We need some kind of model that predicts how the function might behave – there will be many we can choose from, all better than constant!

Linear Interpolation

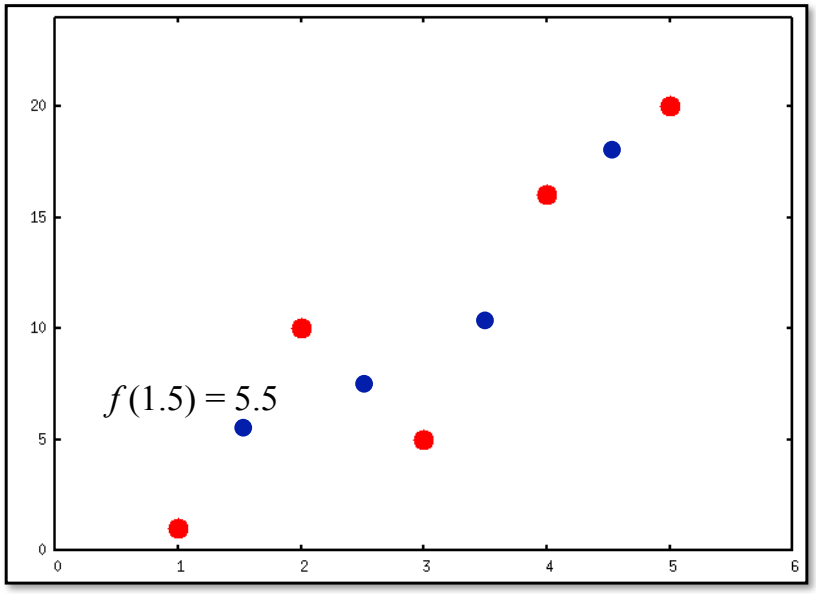
- Example, suppose we're given:

f(1) = 1, f(2) = 10, f(3) = 5 , f(4) = 16, f(5) = 20



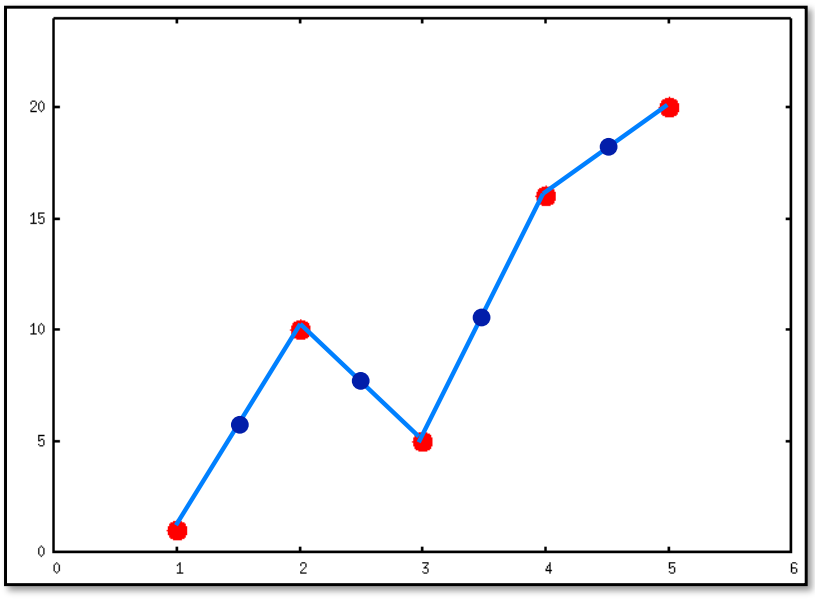
Linear Interpolation

- How can we find f(1.5)?
- One approach: take the average of f(1) and f(2)



Linear Interpolation

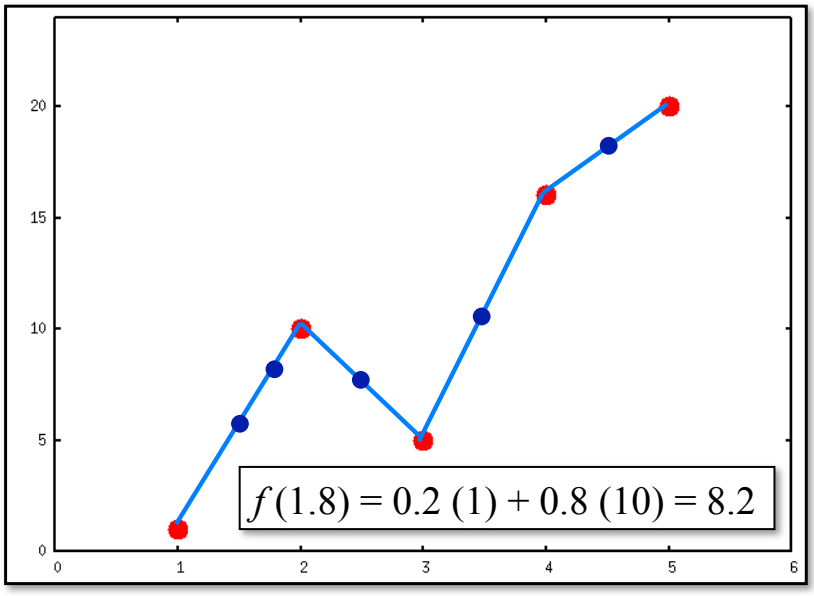
- More generally, fit a line between each pair of data points
- This piecewise linear approach is simple to code and gives quite good results



Linear Interpolation

- What is *f*(1.8)? *Answer:* 0.2 *f*(1) + 0.8 *f*(2)

- More generally, if A and B are two points, then A (1-t) + B t is a straight line running from A to B as t runs from 0 to 1



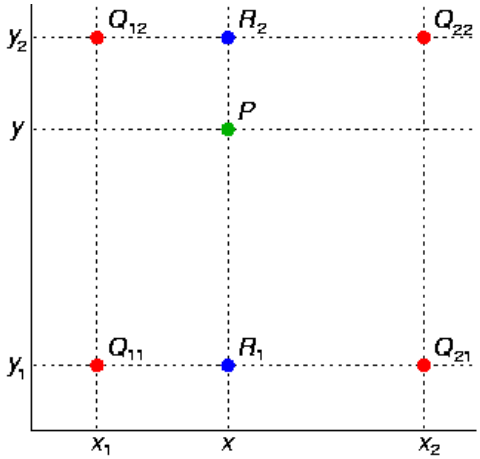
Linear Interpolation

- So how do we apply this averaging trick to 2D images?

- Let's consider one *horizontal* slice through the image (one *scanline*)

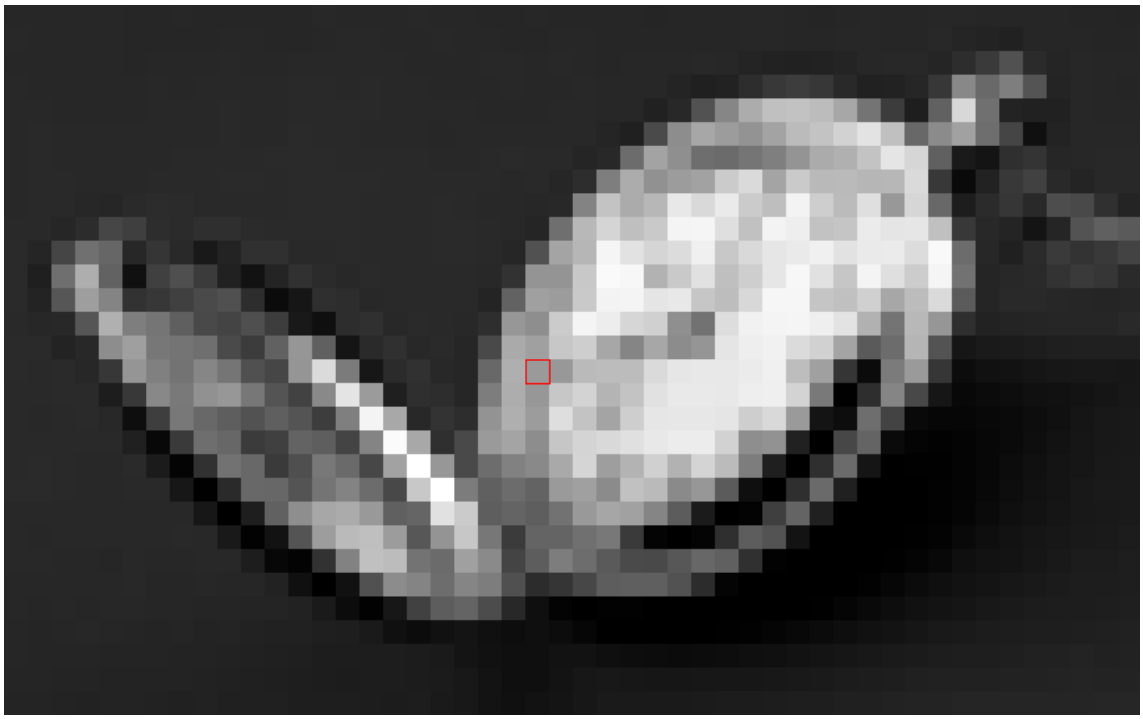
Bilinear interpolation

- So for 2D …
  - Interpolate in x, then in y
- For example,
  - We know the red values
  - Linear interpolation in x between red values gives us the blue values
  - Linear interpolation in y between the blue values gives us an answer
  - Do we get the same answer doing this in the other order ?



Notice that the
A (1-t) + B t
formula works equally well for points A and B in 2D, or 3D, or ….

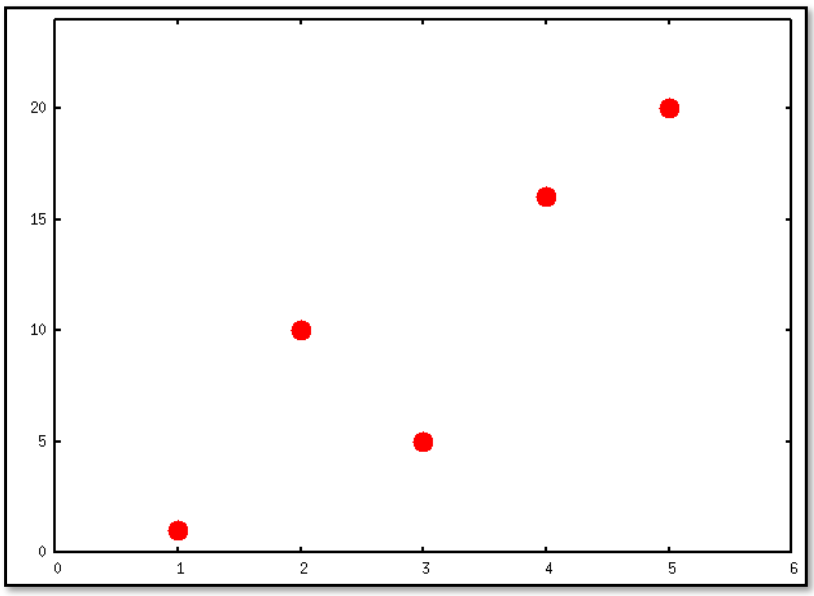
Constant Interpolation



Bilinear interpolation

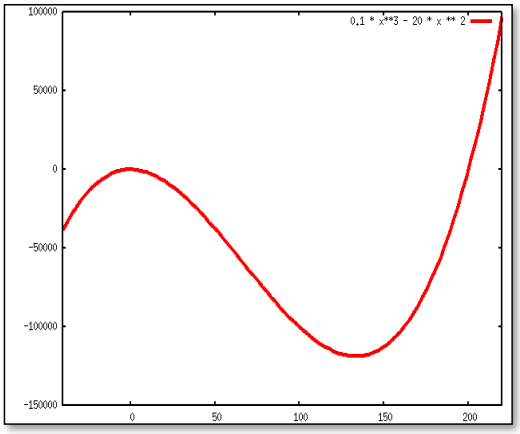
Beyond linear interpolation

- More generally, we could fit polynomials of degree higher than 1 (= linear)
  - Quadratics?
  - Cubics?
  - Quintics?
- Why only polys?
  - Trigs?
  - Hypergeometric?



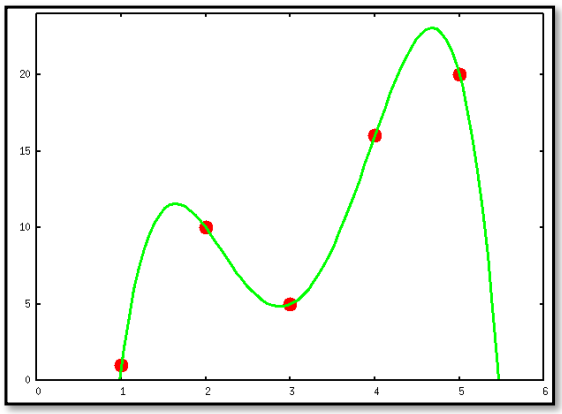
Beyond linear interpolation

- It's all about relating effectively to the nearby data.
- A linear function has 2 degrees of freedom, its slope and position, so can pass through A and B, but have no freedom left to avoid the sharp corners for the nextdoor linear functions
- Cubic functions have 4 degrees of freedom, so can pass through A and B *and* match the slopes of the nextdoor cubics – much smoother! (Odd degree polys are nice this way.)



Polynomial interpolation

- Given n points to fit, we can find a polynomial $p(x)$ of degree $n - 1$ that passes through every point.



$p(x) = -2.208\ x^4 + 27.08x^3 - 114.30\ x^2 + 195.42x - 104$

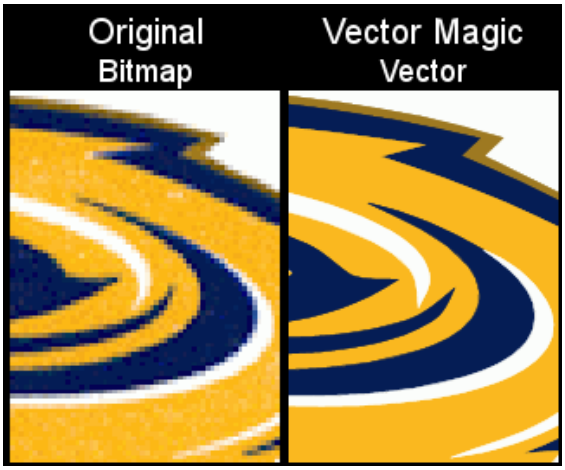
- Every interpolation technique has benefits and negatives; no one trick is ideal for everything!

Bilinear interpolation

Bicubic interpolation

Even better interpolation

- Detect curves in the image, represent them analytically, so can generate afresh the pixels from formulae for the new resolution.

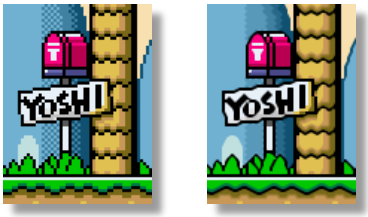


Even better interpolation

nearest-neighbor interpolation

hq4x filter

SNES resolution: 256x224
Typical PC resolution: 1920x1200



As seen in ZSNES

Other applications of interpolation

- Computer animation (keyframing), ie temporal interpolation

Extrapolation

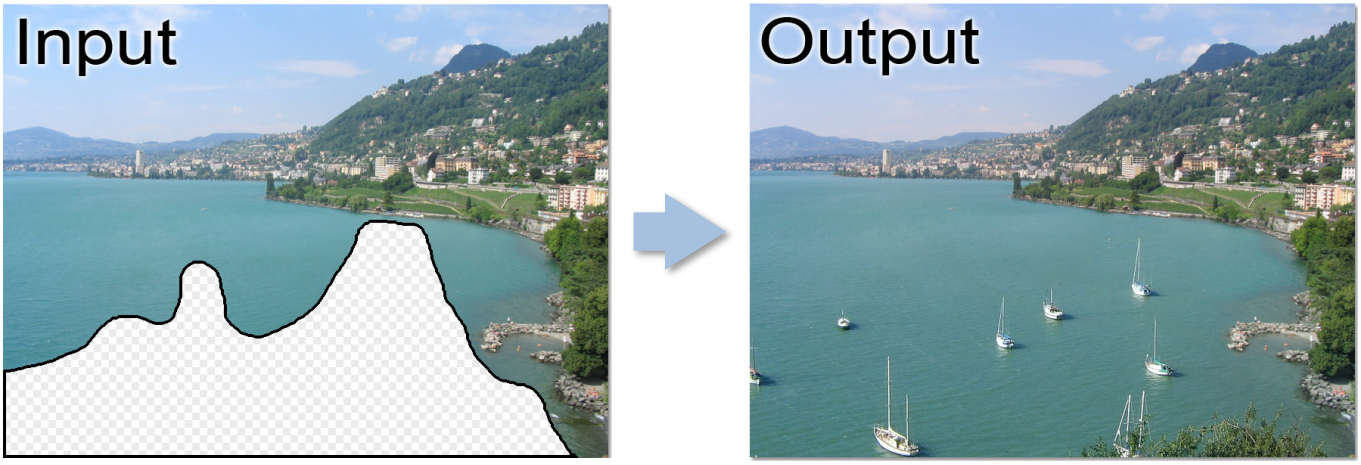
- Suppose you only know the values f(1), f(2), f(3), f(4) of a function
  - What is f(5)?

- This problem is called extrapolation
  - Much harder than interpolation: what is outside the image?
  - For the particular case of temporal data, extrapolation is called prediction (what will the value of MSFT stock be tomorrow?)
  - If you have a good model, this can work well (though perhaps that's the definitions of a good model!!)

Gray2Color

http://www.cs.huji.ac.il/~yweiss/Colorization/
(Matlab code available)

Image extrapolation



http://graphics.cs.cmu.edu/projects/scene-completion/
Computed using a database of millions of photos – though in some sense this is interpolation amongst images! True extrapolation would yield only ocean and coastline.

Image transformations

2D Transformations

- 2D Transformation:
  - Function from 2D → 2D

    $f(x, y) = (x', y')$
  - We'll apply this function to every pixel to get a new pixel location

- Examples:

  $f(x, y) = (0.5x, 1.5y)$

  $f(x, y) = (y, x)$

  $f(x, y) = (yx, x^2 + y^2)$

  $f(x, y) = (x \cos\theta + y \sin\theta, -x \sin\theta + y \cos\theta)$,

  for some fixed value of θ.

2D Transformations examples

$f(x, y) = (0.5x, 2y)$

2D Transformations examples

$f(x, y) = (y, x)$

2D Transformations examples

- Can be non-linear:

2D Transformations examples

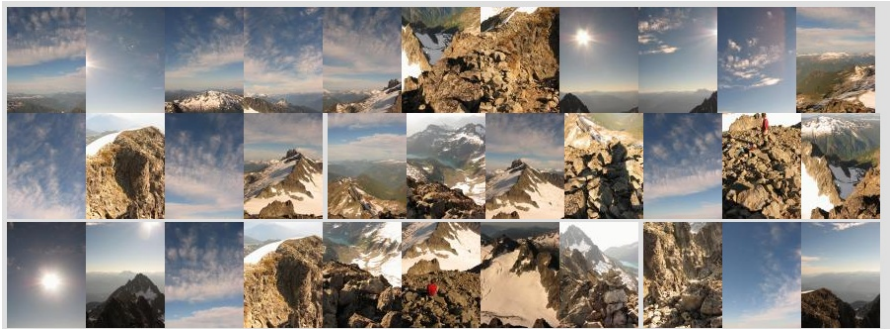


image credit: Matt Brown

Linear Transformations

- We will focus on *linear* transformations
  - 1D: f(x) = ax
  - 2D: f(x, y) = (ax + by, cx + dy)
- Examples
  1. f(x, y) = (0.5x, 1.5y)
  2. f(x, y) = (y, x)
- These all have the property that, writing points as vectors,
  1. **f**(**v** + **u**) = **f**(**v**) + **f**(**u**) and
  2. **f**(a**v**) = a **f**(**v**), for a any scalar (number)

*Notice that since a**v** simply stretches (or contracts, or reverses) **v**, then property 2 means that **f** simply takes the whole 'line' through the origin along **v** and moves it to a new line through the origin along **f**(**v**).*

2D Linear Transformations

- We usually describe 2D space via a pair of coordinate axes, so let
  - **i** be the vector of length 1 pointing along the x-axis and
  - **j** be the unit vector for the y-axis
- Then any vector **v** can be written in terms of how much x and y it has…
  - **v** = a**i** + b**j** (here, **i** and **j** are called *basis* vectors)
- So if **f** is linear, then
  - **f**(**v**) = **f**(a**i** + b**j**) = **f**(a**i**) + **f**(b**j**) = a **f**(**i**) + b **f**(**j**)
- Which means that if we know what **f** does to the basis vectors, then we know what it does to every vector! We can use this to make life simple…

2D Linear Transformations

- Consider our second example: f(x,y) = (y,x)
- What's **f**(**i**) = ? ans **j** and **f**(**j**) = ? ans **i**
- So then

$$\mathbf{f}(1\mathbf{i} + 0\mathbf{j}) = 0\mathbf{i} + 1\mathbf{j} \text{ and } \mathbf{f}(0\mathbf{i} + 1\mathbf{j}) = 1\mathbf{i} + 0\mathbf{j}$$

Writing $\mathbf{i} = \begin{bmatrix}1\\0\end{bmatrix}$ and $\mathbf{j} = \begin{bmatrix}0\\1\end{bmatrix}$ then $\mathbf{f}(1\begin{bmatrix}1\\0\end{bmatrix} + 0\begin{bmatrix}0\\1\end{bmatrix}) = 0\begin{bmatrix}1\\0\end{bmatrix} + 1\begin{bmatrix}0\\1\end{bmatrix}$

This gives us a shorthand notation for **f**(**v**) = **u** , namely $\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}\begin{bmatrix}a\\b\end{bmatrix} = \begin{bmatrix}b\\a\end{bmatrix}$

2D Linear Transformations

- More generally, we can represent any 2D linear transformation by a 2x2 matrix …

$$T = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

- Doing the calculation using matrix multiplication

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

- Note that the actual numbers appearing in the matrix depend critically on the choice of basis – choosing cleverly is the main thing in life.

Examples

- f(x, y) = (0.5x, 1.5y)

$$T = \begin{bmatrix} 0.5 & 0 \\ 0 & 1.5 \end{bmatrix}$$

- f(x, y) = (y, x)

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

Common linear transformations

- Uniform scaling:

$$S = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \qquad \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} sx \\ sy \end{bmatrix}$$

Common linear transformations

- Rotation anticlockwise by angle θ

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Remember that for images, the positive y *direction is* ***downwards****!!*

Common linear transformations

- Shear

$$H = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}$$

Remember again that for images, the positive y *direction is* ***downwards****!!*

Composing linear transformations

- What if we want to scale *and* rotate?
- Answer: multiply the matrices together

$$S = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \qquad R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

scale *rotation*

$$RS = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} = \begin{bmatrix} s\cos\theta & -s\sin\theta \\ s\sin\theta & s\cos\theta \end{bmatrix}$$

scale and rotation

- Does the order of multiplication matter? *yes*