Interpolation

Prof. Noah Snavely
CS1114
http://www.cs.cornell.edu/courses/cs1114

Cornell University
Computer Science

Administrivia

- Assignment 3 due tomorrow by 5pm
  - Please sign up for a demo slot
- Assignment 4 will be posted tomorrow evening
- Quiz 3 next Thursday

Today: back to images

- This photo is too small:

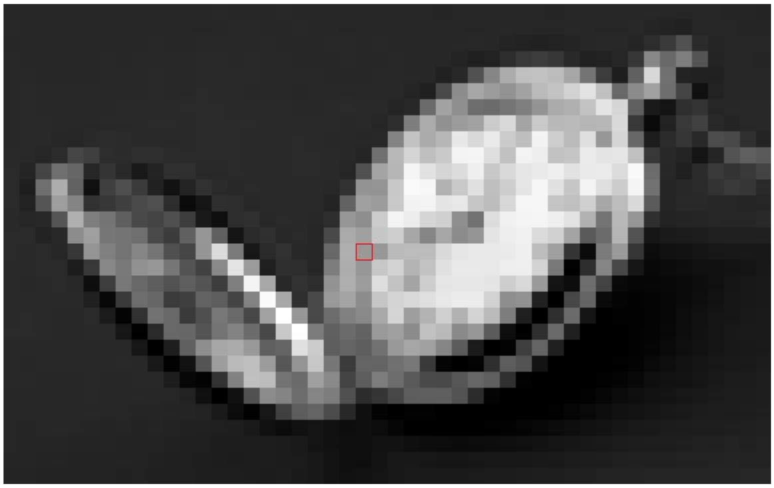
- Might need this for forensics:
  http://www.youtube.com/watch?v=3uoM5kfZIQ0

Zooming

- First consider a black and white image
  (one intensity value per pixel)

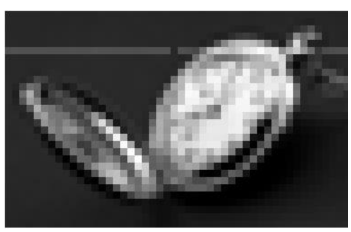
- We want to blow it up to poster size (say, zoom in by a factor of 16)
- First try: repeat each row 16 times, then repeat each column 16 times

Zooming: First attempt



Interpolation

- That didn't work so well
- We need a better way to find the in between values
- Let's consider one horizontal slice through the image (one *scanline*)



Interpolation

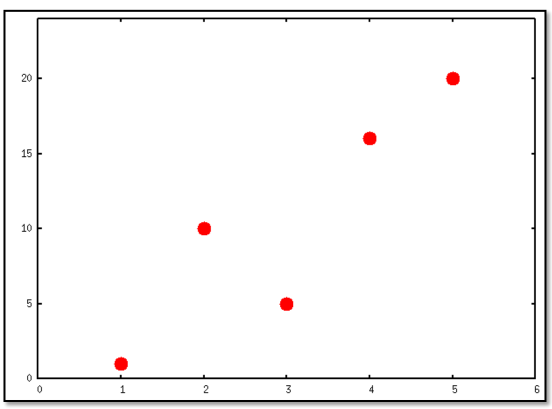
- Problem statement:
- We are given the values of a function *f* at a few locations, e.g., *f*(1), *f*(2), *f*(3), …
- Want to find the rest of the values
  - What is *f*(1.5)?

- This is called *interpolation*
- We need some kind of model that predicts how the function behaves

Interpolation

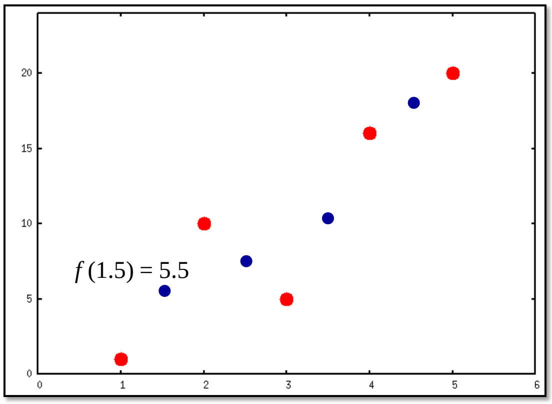
- Example:

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



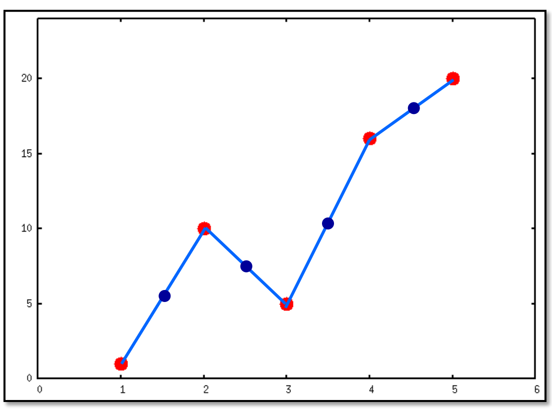
Interpolation

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



Linear interpolation (lerp)

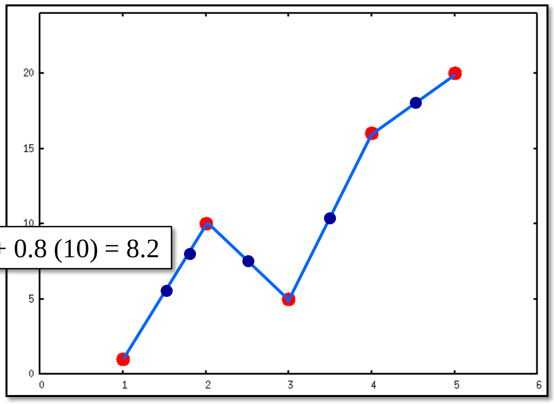
- Fit a line between each pair of data points



Linear interpolation

- What is $f(1.8)$?

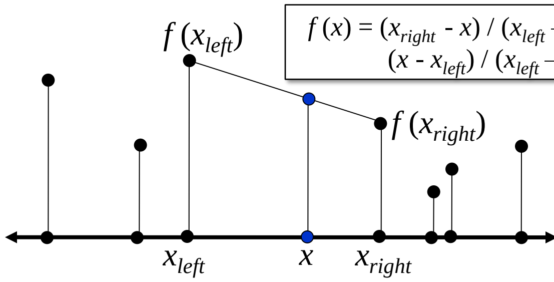
Answer: 0.2 $f(1)$ + 0.8 $f(2)$



$f(1.8) = 0.2\ (1) + 0.8\ (10) = 8.2$

Linear interpolation

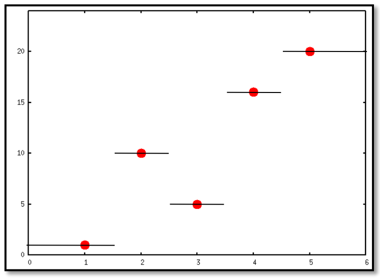
- To compute $f(x)$, find the two points x_{left} and x_{right} that x lies between



$$f(x) = (x_{right} - x) / (x_{left} - x_{right})\ f(x_{left}) + (x - x_{left}) / (x_{left} - x_{right})\ f(x_{right})$$

Nearest neighbor interpolation

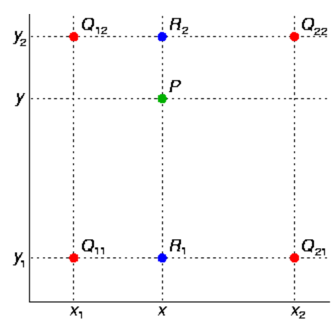
- The first technique we tried
- We use the value of the data point we are closest to



- This is a fast way to get a bad answer

Bilinear interpolation

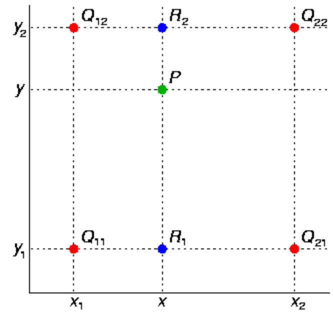
- What about in 2D?
  - Interpolate in x, then in y
- Example
  - We know the red values
  - Linear interpolation in x between red values gives us the blue values
  - Linear interpolation in y between the blue values gives us the answer



http://en.wikipedia.org/wiki/Bilinear_interpolation

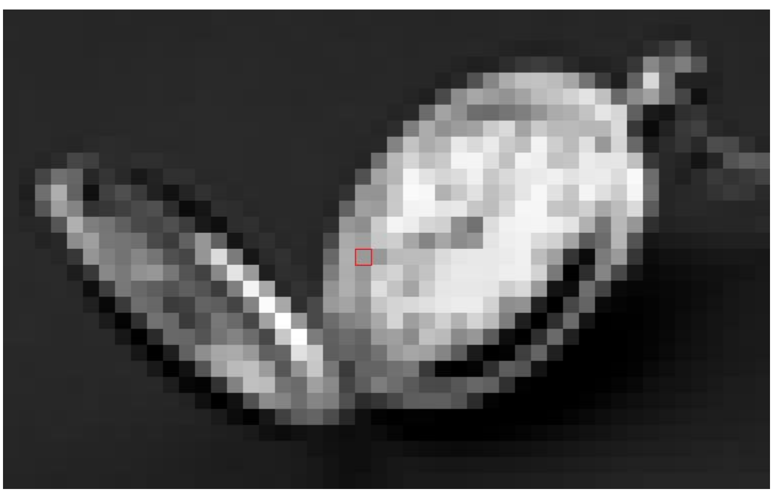
Bilinear interpolation

$$
\begin{aligned}
f(x,y) \approx\ & \frac{f(Q_{11})}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y_2-y) \\
& + \frac{f(Q_{21})}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y_2-y) \\
& + \frac{f(Q_{12})}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y-y_1) \\
& + \frac{f(Q_{22})}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y-y_1).
\end{aligned}
$$



http://en.wikipedia.org/wiki/Bilinear_interpolation

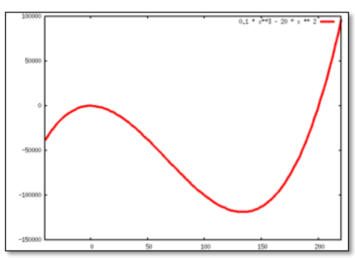
Nearest neighbor interpolation



Bilinear interpolation

Beyond linear interpolation

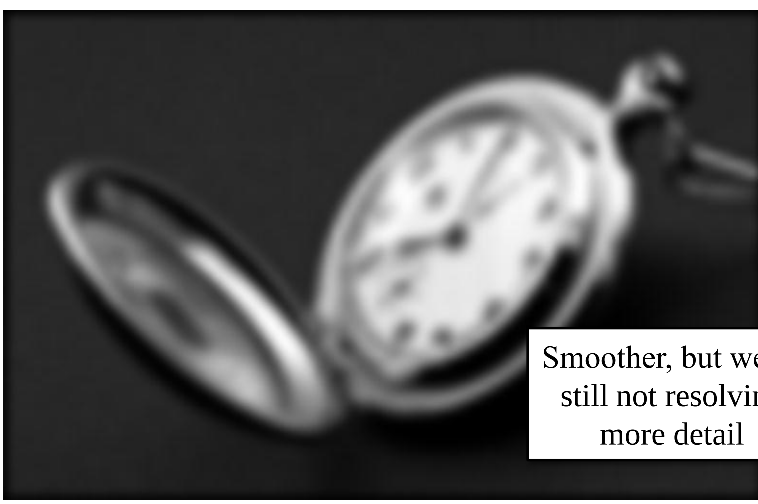
- Fits a more complicated model to the pixels in a neighborhood
- E.g., a cubic function



http://en.wikipedia.org/wiki/Bicubic_interpolation

Bilinear interpolation

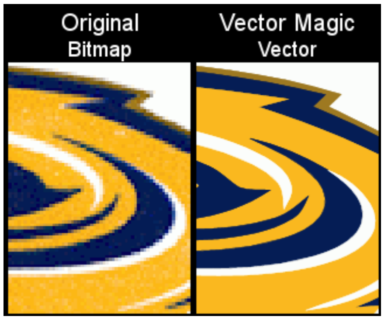
Bicubic interpolation



Smoother, but we're still not resolving more detail

Even better interpolation

- Detect curves in the image, represents them analytically



Even better interpolation

nearest-neighbor interpolation

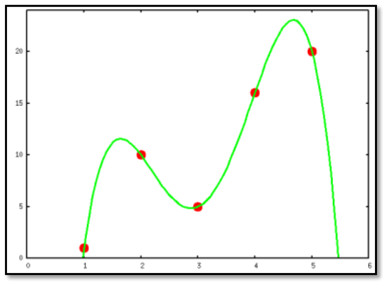
hq4x filter

SNES resolution: 256x224
Typical PC resolution: 1920x1200

As seen in ZSNES

Polynomial interpolation

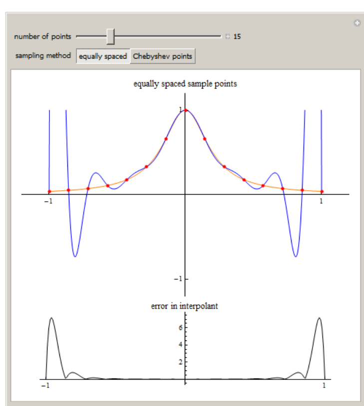
- Given n points to fit, we can find a polynomial $p(x)$ of degree $n - 1$ that passes through every point exactly



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

Polynomial interpolation

- For large n, this doesn't work so well…



Other applications of interpolation

- Computer animation (keyframing)

Gray2Color

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

Limits of interpolation

- Can you prove that it is impossible to interpolate correctly?
- Suppose I claim to have a correct way to produce an image with 4x as many pixels
  - Correct, in this context, means that it gives what a better camera would have captured
  - Can you *prove* this cannot work?
- Related to impossibility of compression

Example algorithm that can't exist

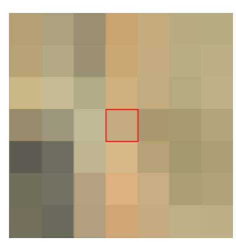
- Consider a compression algorithm, like zip
  - Take a file F, produce a smaller version F'
  - Given F', we can uncompress to recover F
  - This is lossless compression, because we can "invert" it
    - MP3, JPEG, MPEG, etc. are not lossless
- Claim: there is no such algorithm that always produces a **smaller** file F' for every input file F

Proof of claim (by contradiction)

- Pick a file F, produce F' by compression
  - F' is smaller than F, by assumption
- Now run compression on F'
  - Get an even smaller file, F''
- At the end, you've got a file with only a single byte (a number from 0 to 255)
  - Yet by repeatedly uncompressing this you can eventually get F
- However, there are more than 256 different files F that you could start with!

Conclusions

1. Some files will get larger if you compress them (usually files with random data)
2. We can't (always) correctly recover missing data using interpolation
3. A low-res image can represent multiple high-res images

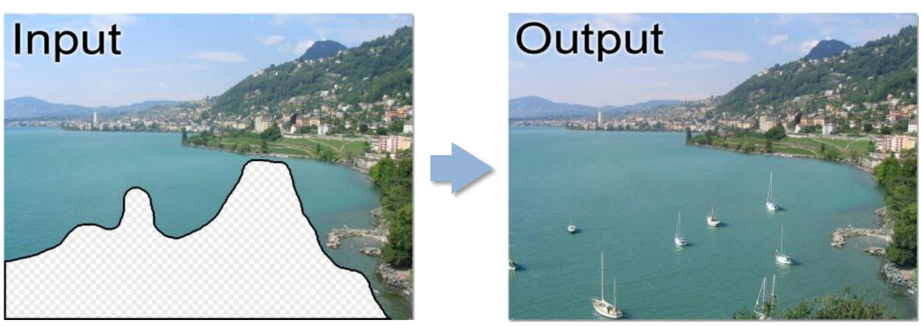


Extrapolation

- 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

Image extrapolation



http://graphics.cs.cmu.edu/projects/scene-completion/
Computed using a database of millions of photos

Questions?