Notes on Sampling (L3-…)

What is a pixel

- One of these finite measurements
- At a particular position
- Point sample – value at a specific place
  - Infinitesimally small place
- Finite region of constant value (little square)
  - Doesn't actually model things better (inconsistent)
  - Mathematically less convenient
  - Useful for some thought experiments later on

A pixel is not a little square!

- Sensors average over region
  - Doesn't mean its really peicewise constant
  - Don't really know what went on in the square
- Point Samples (paradoxically) fit better with the finite case (the buckets, screen dots)
  - Sensing – estimation of what happens at the point from the neighborhood
  - Display – neighborhood is created based on the points inside of it (splats, bleeding, …)

Point Sampling Has Problems

- Miss small things
- Problem: discretization throws away information
- Don't know what happens between samples
- Sampling loses information – you cannot get back the information once its lost!

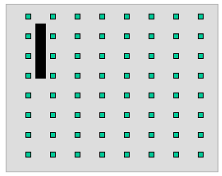
Aliasing

- Technical term for sampling problems
- If you lose information and "make it up" wrong, you get weird effects

Why do we care?

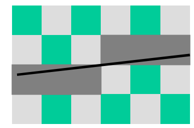
Bad sampling is bad

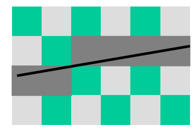
- Miss small things between samples



Ugly

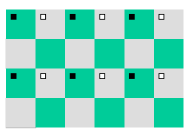
- Imagine line drawing
- Jaggies
- Crawlies
  - Small change causes jump
  - Smooth motion becomes jumpy

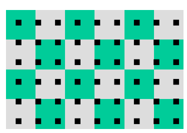




Get really weird results

- Sample a checkerboard
  - Look at a sampled picture
- Too few samples
  - Get all black
  - Get all white
  - Get weird patterns
    - Aliasing
    - Moire'
  - Arbitrary algorithm decision gives very different answers!
- Imagine resampling





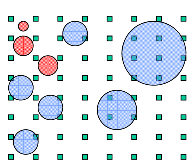
Demonstration ratios: 4/6 (here) = 2/3

Dealing with discretization

- Sampling
  - Understand what information we are throwing away
- Reconstruction
  - Recreate as well as possible from the samples
- Re-Sampling
  - Sample a sampled image
  - Transform the image
- Signal Processing / Image Processing
- Consider the 1D case first since its easier

Intuition

- Too few samples = BAD
- Sampling rate depends on the thing you're sampling
- Need to sample close enough to get smallest object
- Need to limit small objects to be big enough that they aren't missed

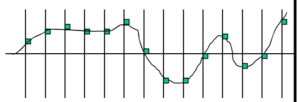


A different intuition

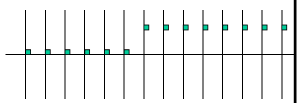
- Not really point sampling
  - Mesurements average over a finite range
  - Displays make finite dots
- Need to model these
  - Sampling filters, reconstruction filters
  - Averages over regions -> Convolution (generalized)
- Need to be realistic about what they mean
  - Can't see everything (too small, …)
- Sampling theory gives a nice mathematics for this!

Point sampling in 1D

- Only record samples
- Don't know what happens in between samples

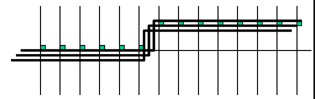


- Given the samples, don't know what really happened!

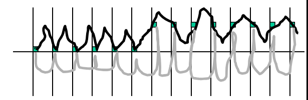


Reconstruction from Sampling

- Can't localize events
  - Bigger problems than that



- No idea! Signal could be anything
- Without additional information, we're guessing as to what the signal is



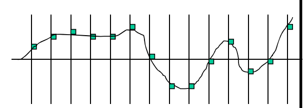
- But what additional info?

Sampling Intuitions

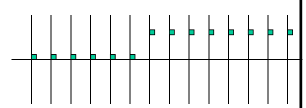
- Reconstruct the "smoothest" signal that makes sense from samples
- If signal is "smooth enough", sampling will give something we can reconstruct
- If signal is not "smooth", sampling will give something that will reconstruct to something else
  - Aliasing
- But how do we define "smooth"

Point sampling in 1D

- Only record samples
- Don't know what happens in between samples

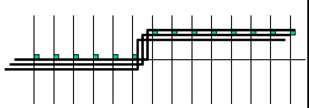


- Given the samples, don't know what really happened!

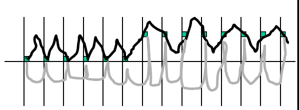


Reconstruction from Sampling

- Can't localize events
  - Bigger problems than that



- No idea! Signal could be anything
- Without additional information, we're guessing as to what the signal is



- But what additional info?

Sampling Intuitions

- Reconstruct the "smoothest" signal that makes sense from samples
- If signal is "smooth enough", sampling will give something we can reconstruct
- If signal is not "smooth", sampling will give something that will reconstruct to something else
  - Aliasing

Signal processing

- Need better "language" for talking about signals
- Idea: represent signals in a different way
- Up till now: time domain (graph against time)
  - Good for asking "what does signal do at time X"
- New idea: frequency domain
  - Good for talking about how smooth signals are
- Different view of the same thing

Frequency Domain

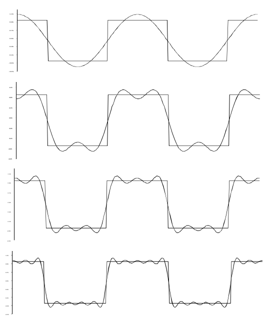
- Fourier Theorem:
  - Any periodic signal can be represented as a sum of sine and cosine waves with harmonic frequencies
  - If one function has frequency *f*, then its harmonics are function with frequency *nf* for integer *n*
  - Extensions to non-periodic signals later
  - Also works in any dimension (e.g. 2 for images, 3, …)
- Example: box

Example: Box (Square Wave)

- 1 cosine – bad
- More cosines, better approx

$$f(x) = \begin{cases} 1 & |x| \le \tfrac{1}{2} \\ 0 & |x| > \tfrac{1}{2} \end{cases}$$

$$S_{boxes}(x) = \frac{1}{2} + \frac{2}{\pi}\sum_{k=1}^{\infty}(-1)^{k-1}\frac{\cos(2k-1)\omega x}{2k-1}$$
$$= \frac{1}{2} + \frac{2}{\pi}\left(\cos\omega x - \frac{1}{3}\cos 3\omega x + \frac{1}{5}\cos 5\omega x - \cdots\right)$$



Intuitions

- Low frequencies are smooth
  - High frequencies change fast, are not smooth
- If a signal can be made of only low frequencies, it is smooth
- If a signal has sharp changes, it will require high frequencies to represent

General Functions

- A non-periodic function can be represented as a sum of sin's and cos's of (possibly) all frequencies:

$$f(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{i\omega x}d\omega$$
$$e^{i\omega x} = \cos\omega x + i\sin\omega x$$

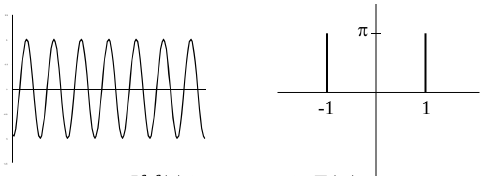
- $F(\omega)$ is the *spectrum* of the function $f(x)$
  - The spectrum is how much of each frequency is present in the function
  - We're talking about functions, not colors, but the

Fourier Transform

- $F(\omega)$ is the Fourier Transform of f(t)
  - A different representation of the same signal
- To get f(t) back you use the Inverse Fourier Transform
- You don't need to know how to compute them

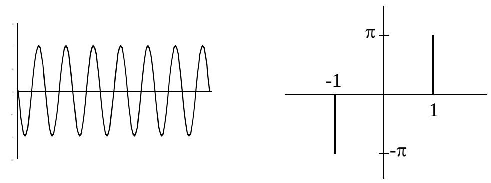
$$F(\omega) = \int_{-\infty}^{\infty} f(x)e^{-i\omega x}dx$$

Cosine and Its Transform



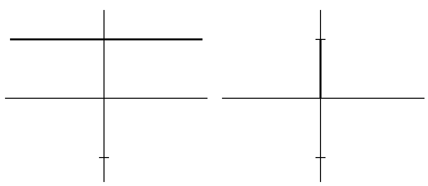
If $f(x)$ is even, so is $F(\omega)$

Sine and Its Transform



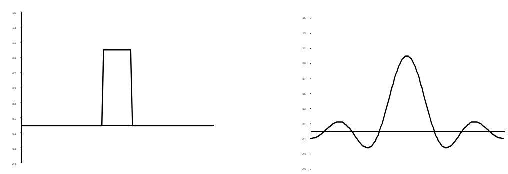
If $f(x)$ is odd, so is $F(\omega)$

Constant Function and Its Transform

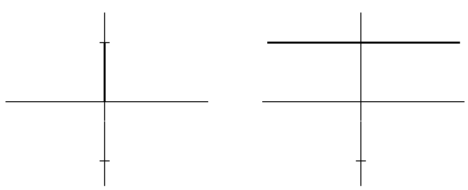


The constant function only contains the 0^{th} frequency – it has no wiggles

Box Function and Its Transform

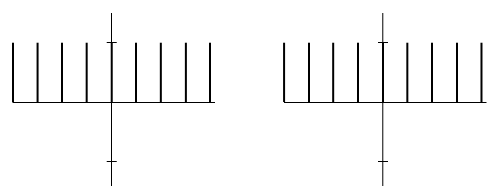


Delta Function and Its Transform



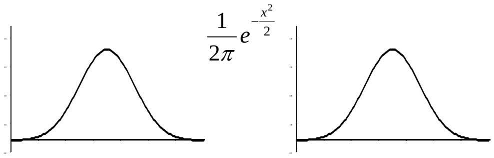
Fourier transform and inverse Fourier transform are qualitatively the same, so **knowing one direction gives you the other**

Shah (Spikes) and Its Transform



Gaussian and Its Transform

$$\frac{1}{2\pi} e^{-\frac{x^2}{2}}$$



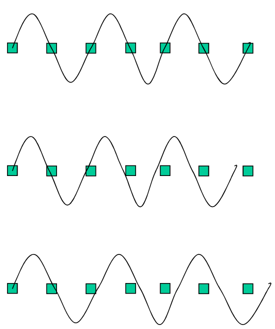
They are the same

Qualitative Properties

- The spectrum of a functions tells us the relative amounts of high and low frequencies
  - **Sharp edges give high frequencies**
  - **Smooth variations give low frequencies**
- A function is *bandlimited* if its spectrum has no frequencies above a maximum limit
  - sin, cos are band limited
  - Box, Gaussian, etc are not
- To band-limit a signal we *low-pass filter* it

Sampling Theorem (intuition)

- High frequencies get lost
  - Can only sample band limited signals
- Sampling rate must be 2 times higher than signal
- Signal must be half frequency of sample rate
  - Otherwise, signal can "turn around" between samples
- Nyquist rate
  - 2x highest frequency in signal



Sampling Theorem

- If your signal is bandlimited
- And you know what the band limit is
- And you sample at (at least) twice that frequency
  - Above the Nyquist rate
- Then – you can reconstruct your signal EXACTLY!
- Caveat
  - Ideal reconstruction requires perfect band

Sampling theory in practice

- When you're sampling- PREFILTER
  - Make sure no high frequencies
  - Need to remove them BEFORE sampling
  - Otherwise, aliasing
  - Filtering effectively means blurring
- When you're reconstructing – FILTER
  - View as a spike chain (remove HF)
  - Filtering effectively means interpolating

Sampling Theory

- Given a set of samples (at a sampling rate):
  - There is exactly one band-passed signal that goes through those samples
  - Where the band-pass is less than half the sampling rate
- Ideal reconstruction
  - View samples as spike chain, low-pass filter
  - Need an ideal low-pass filter
  - Approximate ideal low-pass filter

Sampling Theory (2)

- If we sample a band-passed signal
  AND the sampling rate is > 2*highest freq
  THEN we can do ideal reconstruction
- If you know the highest frequencies you care about, you know how fast you need to sample!
  - CD Audio Example: human hearing isn't so great after 22Khz, so sample at 44.1Khz

Sampling Theory (3)

- If your signal is not bandpassed
  (i.e. has HF >= 2*sampling rate)
  THEN you will get aliasing when you sample
- Once you've aliased – you can't go back!
- You have no idea what the original was!
- Need to PREFILTER the signal before sampling to make it bandpassed

Filtering: Convolutions

- A general filter is a function on an image that produces another image
- Many common filters are simpler in the Fourier domain
- Choice:
  - Transform image, filter, inverse transform image
  - Inverse transform operator, apply in spatial domain

Theory vs. Practice

Theory
- Properly sampled original
- Know bandlimit
- Band-limit signals
- Use Ideal Filters
- Ideal Reconstructions

Practice
- Who knows about source?
- Assume that its OK?
- Ideal LPF not practical
- Use approximations
- Tradeoffs for "ideal"
  - Might look blurry
  - Might want aliasing (sharpness)
  - Care about efficiency

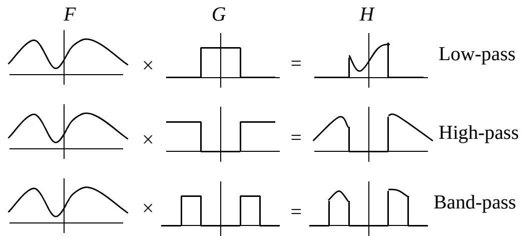
What is a filter anyway?

- Frequency filters
  - Add remove different frequencies
- Multiplication in frequency means CONVOLUTION in time/space
- Continuous and Discrete Convolutions

Filters

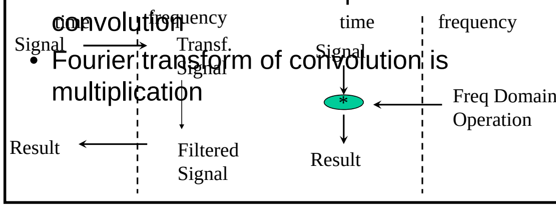
- A *filter* is something that attenuates or enhances particular frequencies
- Easiest to visualize in the frequency domain, where filtering is defined as multiplication:
  $$H(\omega) = F(\omega) \times G(\omega)$$
- Here, *F* is the spectrum of the function, *G* is the spectrum of the filter, and *H* is the filtered function. Multiplication is point-wise

Qualitative Filters



Can you transform an operator?

- Many filters are multiplication in frequency domain
- Fourier transform of multiplication is convolution
- Fourier transform of convolution is multiplication



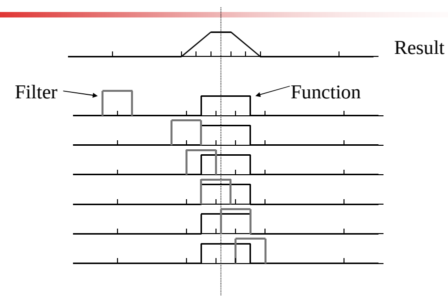
Filtering in the Spatial Domain

- Filtering the spatial domain is achieved by *convolution*

$$h(x) = f \otimes g = \int_{-\infty}^{\infty} f(u)g(x-u)du$$

- Qualitatively: Slide the filter to each position, *x*, then sum up the function multiplied by the filter at that position

Convolution Example



Convolution Theorem

- Convolution in the spatial domain is the same as multiplication in the frequency domain
  - Take a function, *f*, and compute its Fourier transform, *F*
  - Take a filter, *g*, and compute its Fourier transform, *G*
  - Compute $H=F \times G$
  - Take the inverse Fourier transform of *H*, to get *h*
  - Then $h=f \otimes g$
- Multiplication in the spatial domain is the same as convolution in the frequency

What's a filter?

- Generic – an operation that maps a signal to another signal
- Specifically: a LOW-PASS filter
  - Attenuates high frequencies
  - Easy to describe in frequency domain (give frequency response)
  - Multiply certain values

Need to know about convolutions

- We need to have band limited signals
  - Need low pass filters
  - Which are implemented as **convolutions**
- Reconstruction requires low-pass filtering
  - Which is implemented as **convolution**
- Need to see Sampling theory in Fourier domain
  - Need **convolution**
- **Convolution is the mathematical**

Convolution

- Multiplication in frequency is convolution in time (space)
- Convolution is the generalization of averaging
- Continuous convolution
  Discrete convolution

Convolution

- Operator on 2 signals
  - f(t) * g(t)   (f and g are both signals)
- Specifically
  - One signal is "our signal"
  - The other is the filter (called a kernel)

Filtering in the Spatial Domain

- Filtering the spatial domain is achieved by *convolution*

$$h(x) = f \otimes g = \int_{-\infty}^{\infty} f(u)g(x-u)du$$

- Qualitatively: Slide the filter to each position, *x*, then sum up the function multiplied by the filter at that position

Discrete Convolution

- h(t) = (f*g)(t) = SUM f(i) g(t-i)
  - Notice that we flip g backwards as we slide it
  - Often g is symmetric, so this is easy to forget
- g = [1 2] f = [1 3 1 2 0] (outside range is 0)
- Zero centering of g ([1/3 1/3 1/3])
  - Weighted average

Dealing with boundaries

- Pretend data outside boundaries is 0
  - Dims edges
- Reflect about ends
- Keep constant values at edges
- Renormalize kernel

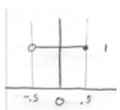
Convolution in 2D

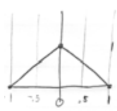
- Show box moving around
- Seperable filters
  - Can do as 1D convolution in both directions
  - Not all filters can do this
  - Useful to find ones that can

Reconstruction in Practice

- Sample a sample – no problem!
- Issue is samples between samples
- Theory: LPF a spike chain
  - Convolve "resonstruction kernel" with samples
  - Only really need to evaluate at places where you'll sample
- Another view: interpolation
  - Different interpolations are different filters

Some reconstruction kernels
Crude approximations to LPF


Constant


Triangle
(Bartlet)


Interpolating
Cubic
(Catmull-Rom)

Spacing (1 unit = sample distance)

Scaling issues

Interpolating (non-interpolating kernels exist as well)

Approx to Ideal LPF

Reconstruction Example

- Sample at sample
- Sample between samples
- Bartlett filter
  - Width correct for sample spacing
- See how we get linear interpolation
- Could do this as linear interpolation
  - Generalizes nicely this way
- Need to evaluate filter for various values
- Convolve reconstruction kernel with sampling kernel (LPF for frequency limit)
- Easier ways to implement nearest neighbor

Re-Sampling

- Choose different samples of the same image

1 2 3 4 5 6 7 8 9 10 11 12 13
1 2 3 4 5 6 7 8 9

- Resizing is a special case
  - Scale image down (and have same sample rate)
  - Keep image the same size (but new sample rate)
- General Version:
  - (x,y) = F (x,y) R^n->R^n
  - Called a warp – could be anything

The obvious problems

- New samples in between old samples
  - Need to *interpolate*
- New sampling rate too low for signal
  - Might alias – need to pre-filter

Re-Sampling in Theory

- Reconstruct (get the continuous signal)
- Filter (to make sure band limited for sampling)
- Sample

- Note that we apply 2 low-pass filters
- Once HF are cut, no need to repeat
- Can pick the one that has lowest cutoff

Re-Sampling

- Need to reconstruct and sample
- f*g*h = f*(g*h) (can put the filters together)
  - Order you do things in doesn't matter

- If new sample rate is higher – old signal is sufficiently bandpassed, just reconstruct
  Upsampling, enlarging, …
- If new sample rate is lower – need to pre-filter
  - Do pre-filtering first (since its discrete)

Re-Sampling in Practice

- No need to create continuous signal
  - How would we represent it?

- Can apply filters in either order
  - Blur (LPF) to remove HF (if necessary)
  - Interpolate (LPF) to find values of samples

- Only compute things at the samples
  - Don't compute the convolution at all places
  - Use one filter that does both

Why pre-filter?

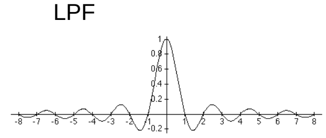
- Consider triangle [0 1 2 3 2 1 0 …]
  - Warning! Not band passed, will alias no matter what
- Downsample by 2
  - (twice as fast, half as many samples)
  - Just pick every other sample
- Is this a resampling kernel?
  - Yes – it is any *interpolating* kernel – if you sample at the sample, you get the sample
  - Not all reconstruction kernels are interpolating,

Downsampling the triangle

```
•   0 1 2 3 2 1 0 1 2 3 2 1 0 1 2 3 2 1 0 1 2 3 2 1 0 1 2 3 2 1 0 1 2 3 2 1 0 1 2 3 2 1 0

•   every other (1/2 sample rate, speed up/shrink by factor of 2)
•   0   2   2   0   2   2   0   2   2   0   2   2   0   2   2   0   2   2   0   2   2   0
•     1   3   1   1   3   1   1   3   1   1   3   1   1   3   1   1   3   1   1   3   1

•   every third (1/3 sample rate, speed up/shrink by factor of 3)
•   0     3     0     3     0     3     0     3     0     3     0     3     0     3     0
•     1     2     1     2     1     2     1     2     1     2     1     2     1     2

•   every forth (1/4 sample rate, speed up/shrink by factor of 4)
•   0       2       2       0       2       2       0       2       2       0       2
•     1       1       3       1       1       3       1       1       3       1       1
•   (notice that this looks just like every other)

•   every firth (1/5 sample rate, speed up/shrink by factor of 5)
•   0         1         2         3         2         1         0         1         2
•     1         0         1         2         3         2         1         0         1
•   (notice that this looks just like the original!)
```

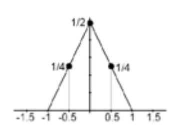
What kernel to use?

- LPF (and LPF-like) filters look like bumps
  - Width of bump is inversely proportional to cutoff
  - Shape of bump says how closely approximates LPF



Sinc = sin(x)/x = ideal LPF

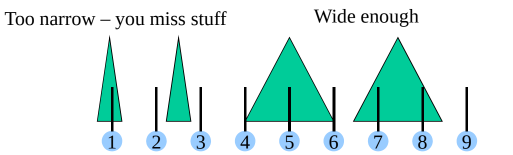
g r



Tent

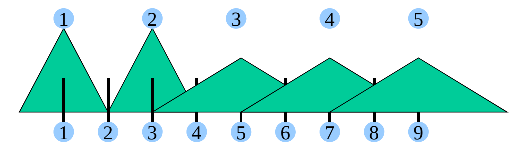
How wide a bump?

- Wide enough to cover between original samples
  - Otherwise, won't have enough to interpolate
- Equivalent to the reconstruction filter cutoff frequency



How wide for Pre-Filtering

- Needs to be wide enough that you don't miss any original samples when you try the new samples
- This is the cutoff from the resampling filter



One filter for resampling

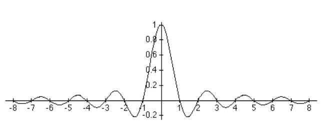
- Only one of the limits will be the lowest – use that one
- Why not be conservative (use an extra big filter)?
  - Cuts out too much stuff
  - Removes too much details
  - Looks blurry
- Things are easier if your samples are

What filters?

- Filters need to be normalized (so they sum to 1)
- Since we're sampling them, renormalize AFTER sampling
- Interpolating – must be one at the sample
  - Must be zero at the other samples
  - Must sum to zero at the other samples
  - Generally, have negative lobes (ringing)
    - But a real LPF will ring
- Reconstruction filters – generally use

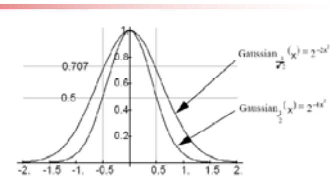
A note on "support"

- Bump width is not the same as support!
- Ideal LPF has infinite support
- Wider the support, more info to use



Gallery of filters

- Box
- Tent
- B-Spline
- Gaussian
- Lanczos



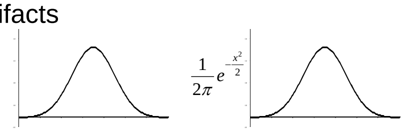
- Differ in how closely they approximate LPF
  - Get rid of lower frequencies accidentally
  - Let in high frequencies accidentally

Some good filters

- B-Splines
  - Repeated convolution of the unit box
  - ½ [1 1] (keep convolving with this)
  - ¼ [1 2 1]
  - 1/8 [1 3 3 1]
  - 1/16 [1 4 5 6 1]
- Wider (bigger) filter = lower frequency limit
- Pick something so overlap at new samples
  - (e.g. the 3 wide one is good for downsample by 2)

Gaussian Filter

- Attenuates high frequencies even further
- In 2d, rotationally symmetric, so fewer artifacts

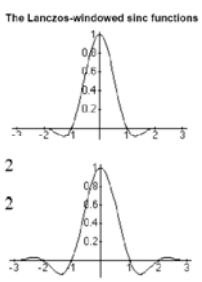


$$\frac{1}{2\pi} e^{-\frac{x^2}{2}}$$

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Lanczos (windowed sinc)

- Chop off far away lobes
- Renormalize
- Chop to preserve derivativ



$$\text{Lanczos2}(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x} \frac{\sin(\pi \frac{x}{2})}{\pi \frac{x}{2}}, & |x| < 2 \\ 0, & |x| \geq 2 \end{cases}$$

Simple example

- Have signal [1 2 1 3 1 2 1]
- Resample @ a rate 2.5 times the original
- Pick the filters?
  - Bartlett (tent) filter = linear interpolation
  - B-Spline ¼ [1 2 1] = common choice (probably not "big" enough)
- Implement?
  - First do the pre-filter, then use the tent at the samples
  - Figure out which samples you need for Bartlet, compute those using B-Spline
  - Use the continuous B-Spline and sample
  - Only need a small set of phases

In 2D

- Everything in 1D extends to ND – just hard to draw
- 2D convolution
  - Kernel is a 2D function, or a matrix in the discrete case
  - Slide around in 2D
  - Centered, Boundaries
- Seperability
  - Some operations can be done in 1 dimension, then the other
  - Sampling can be
  - Filtering can be – if the kernel is seperable

If there's time…
(or things to try at home)

- Show the need for a fixed set of phases for reconstruction kernels
- Show 2D B-Splines
- Do a 2D resampling example

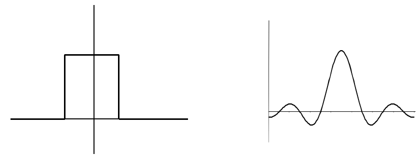
Filtering Images

- Work in the discrete spatial domain
- Convert the filter into a matrix, the *filter mask*
- Move the matrix over each point in the image, multiply the entries by the pixels below, then sum
  - eg 3x3 box filter
  - Effect is averaging

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Box Filter

- Box filters smooth by averaging neighbors
- In frequency domain, keeps low frequencies and attenuates (reduces) high frequencies, so clearly a low-pass filter



Spatial: Box

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Frequency: sinc

Filter Widths

- Fourier Transform of a Time scaling:
  - f(k t) -> F(1/k omega)
  - As time gets scaled, frequency gets scaled by the inverse
- Box filter: wider box in frequency domain = narrower filter in time domain
- To filter higher frequencies use a narrow (in time/space) filter
- Lower Frequency cutoff (in a High-pass

Handling Boundaries

$$I_{output}[x][y] = \sum_{i=-k/2}^{k/2} \sum_{j=-k/2}^{k/2} I_{input}[x+i][y+j]M[i+k/2][j+k/2]$$

- At (0,0) for instance, you might need pixel data for (-1,-1), which doesn't exist
- Option 1: Make the output image smaller – don't evaluate pixels you don't have all the input for
- Option 2: Replicate the edge pixels
  - Equivalent to: posn = x + i; if (posn < 0) posn = 0; and so on for other indices
- Option 3: Reflect image about edge

Seperable Filters

- Some 2D filters can be implemented as 2 1D filters
- Each dimension at a time
- Much easier
  - Don't need to build 2D filter kernel
  - Much faster (O(mn) not O(m^2 n))
- Box filters are separable

Constructing Masks: 2D

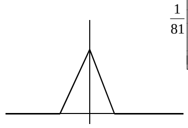
- Multiply 2 1D masks together using *outer product* $M[i][j] = m[i]m[j]$
- *M* is 2D mask, *m* is 1D mask

	0.2	0.6	0.2
0.2	0.04	0.12	0.04
0.6	0.12	0.36	0.12
0.2	0.04	0.12	0.04

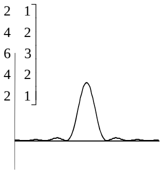
Bartlett Filter

- Triangle shaped filter in spatial domain
- In frequency domain, product of two box filters, so attenuates high frequencies more than a box

$$\frac{1}{81}\begin{bmatrix}1 & 2 & 3 & 2 & 1\\ 2 & 4 & 6 & 4 & 2\\ 3 & 6 & 9 & 6 & 3\\ 2 & 4 & 6 & 4 & 2\\ 1 & 2 & 3 & 2 & 1\end{bmatrix}$$



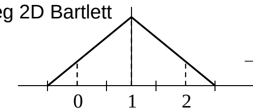
Spatial: Triangle (Box⊗Box)



Frequency: sinc^2

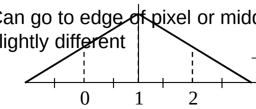
Constructing Masks: 1D

- Sample the filter function at matrix "pixels", then normalize
- eg 2D Bartlett



$\frac{1}{5}$ | 1 | 3 | 1 |

- Can go to edge of pixel or middle of next: results are slightly different

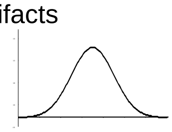


$\frac{1}{4}$ | 1 | 2 | 1 |

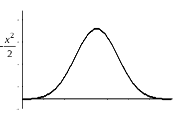
Gaussian Filter

$$\frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix}$$

- Attenuates high frequencies even further
- In 2d, rotationally symmetric, so fewer artifacts



$$\frac{1}{2\pi}e^{-\frac{x^2}{2}}$$



Constructing Gaussian Mask

- Use the binomial coefficients
  – Central Limit Theorem (probability) says that with more samples, binomial converges to Gaussian

$\frac{1}{4}$ | 1 | 2 | 1 |

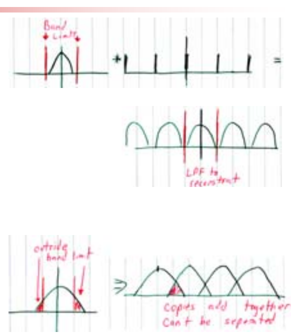
$\frac{1}{16}$ | 1 | 4 | 6 | 4 | 1 |

$\frac{1}{64}$ | 1 | 6 | 15 | 20 | 15 | 6 | 1 |

```
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

Sampling Theory

- Sampling is multiply by spike chain in time domain
  – Fourier transform of spike chain is spike chain
  – Fourier transform of multiply is convolution
- Sampling is convolution by spike chain in frequency
- Makes infinite copies of signal
- Reconstruction low-pass filters to remove all but one
- Non-band limited, things "spill"



Sampling / Reconstruction

- Both sampling and reconstruction require
  Low Pass Filtering
- Sampling:
  – Low pass filter signal to make sure is band-limited
- Reconstruction:
  – Low pass filter spike chain to figure out what happens between samples
- Resampling:

Resizing = Resampling

- Same image – different number of samples
- Issues:
  – New samples are in between old samples
  – Too few new samples to capture all the frequency
- Basic idea (in theory)
  – Reconstruct original signal (LPF the samples)
  – Low-pass filter (so sampling works)

Resampling – Little Square Model

- Region of source = Region of Dst
- Pixel is a region
  – Dest region might be bigger than pixel in source
  – Average over the region (convolution gives us the weights)
- In-between pixels is piecewise constant
  – Chunky look is what the model says is right

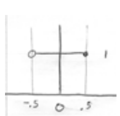
Pre-Filtering

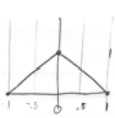
- If SRC is bigger than DST it may have HF
  – If its close, might need it anyway because of imperfect reconstruction
- Need to LPF
- LPF before sampling?
  – Requires you to do a complete reconstruction
  – Only really need to do it at points you will sample
- Pre-Filtering
  – Do LPF before reconstruction / as part of

Reconstruction in Practice

- Sample a sample – no problem!
- Issue is samples between samples
- Theory: LPF a spike chain
  – Convolve "resonstruction kernel" with samples
  – Only really need to evaluate at places where you'll sample
- Another view: interpolation
  – Different interpolations are different filters

Some reconstruction kernels


Constant


Triangle
(Bartlet)


Interpolating
Cubic
(Catmull-Rom)

Spacing (1 unit = sample distance)

Scaling issues

Interpolating (non-interpolating kernels exist as well)

Approx to Ideal LPF

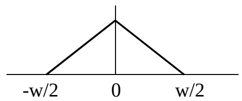
Reconstruction Example

- Sample at sample
- Sample between samples
- Bartlett filter
  – Width correct for sample spacing
- See how we get linear interpolation
- Could do this as linear interpolation
  – Generalizes nicely this way
- Need to evaluate filter for various values
- Convolve reconstruction kernel with sampling kernel (LPF for frequency limit)
- Easier ways to implement nearest neighbor

Functional Form for Filters

- Consider the Bartlett in 1D:

$$H_w(s) = \frac{2}{w}\left(1 - \frac{2|s|}{w}\right)$$



- To apply it at a point x_{orig} and find the contribution from point x where the image has value I(x)

$$f(x) = \frac{2}{w}\left(1 - \frac{2|x - x_c|}{w}\right) I(x)$$

- Extends naturally to 2D:

$$f(x, y) = \frac{4}{w^2}\left(1 - \frac{2|x - x_c|}{w}\right)\left(1 - \frac{2|y - y_c|}{w}\right) I(x, y)$$

General Resampling

- Could be any transformation on x,y
- X',y' = f(x,y)
- Scale, translate, rotate, something weird
- Kernel should get warped too
  - Little square -> some weird shape
  - Little circle/square (of kernel) -> some weird shape
  - In practice, stick with squares

Reverse Warping

- Note we generally need the INVERSE:
  - X', y' = f(x,y) (x' = dst, x = src)
  - Know x', need to find x is inverse
- Reverse warping is easier (scan over each pixel in the dst, figure out where it comes from)
- Forward warping is tricker
  - Usually can invert function, but if you can't
  - Need to worry about holes
- Lots of fun warps to do!