2. Fourier analysis and sampling theory

Reading

Required:

- Watt, Section 14.1

Recommended:

- Ron Bracewell, The Fourier Transform and Its Applications, McGraw-Hill.
- Don P. Mitchell and Arun N. Netravali, "Reconstruction Filters in Computer Computer Graphics ," Computer Graphics, (Proceedings of SIGGRAPH 88). *22 (4)*, pp. 221-228, 1988.

What is an image?

We can think of an **image** as a function, *f*, from R^2 to R:

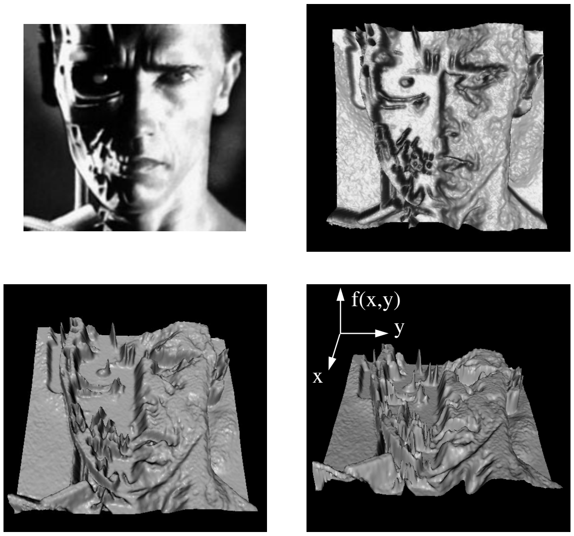
- $f(x, y)$ gives the intensity of a channel at position (x, y)
- Realistically, we expect the image only to be defined over a rectangle, with a finite range:
  - $f: [a,b] \mathrm{x} [c,d] \rightarrow [0,1]$

A color image is just three functions pasted together. We can write this as a "vector-valued" function:

$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$

We'll focus in grayscale (scalar-valued) images for now.

Images as functions



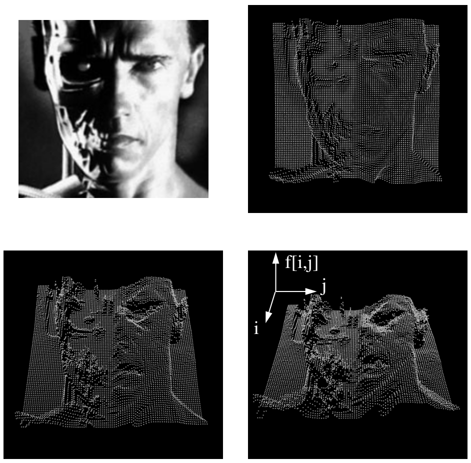
Digital images

In computer graphics, we usually create or operate on **digital** (**discrete**) images:

- **Sample** the space on a regular grid
- **Quantize** each sample (round to nearest integer)

If our samples are Δ apart, we can write this as:

$f[i,j] = \text{Quantize}\{ f(i\,\Delta, j\,\Delta) \}$



Motivation: filtering and resizing

What if we now want to:

- smooth an image?
- sharpen an image?
- enlarge an image?
- shrink an image?

Before we try these operations, it's helpful to think about images in a more mathematical way…

Fourier transforms

We can represent functions as a weighted sum of sines and cosines.

We can think of a function in two complementary ways:

- **Spatially** in the **spatial domain**
- **Spectrally** in the **frequency domain**

The **Fourier transform** and its inverse convert between these two domains:

Spatial domain → $F(s) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi sx} dx$ → Frequency domain

Spatial domain ← $f(x) = \int_{-\infty}^{\infty} F(s) e^{i2\pi sx} ds$ ← Frequency domain

Fourier transforms (cont'd)

Spatial domain → $F(s) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi sx} dx$ → Frequency domain

Spatial domain ← $f(x) = \int_{-\infty}^{\infty} F(s) e^{i2\pi sx} ds$ ← Frequency domain

Where do the sines and cosines come in?

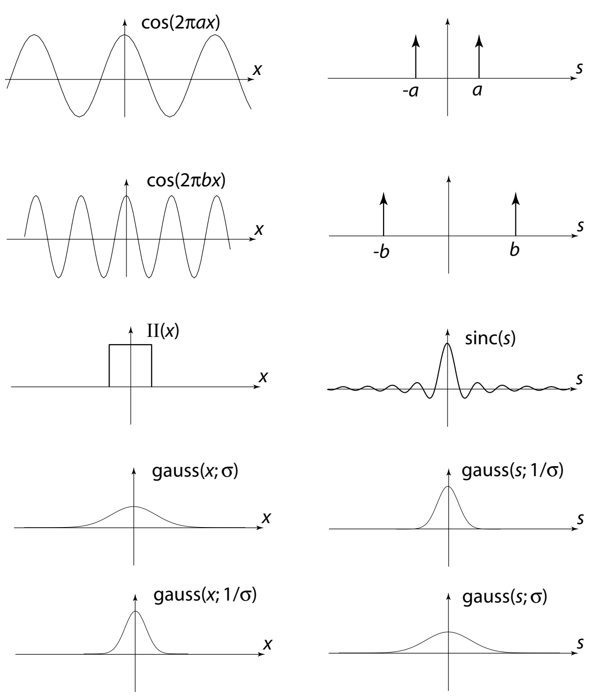
$f(x)$ is usually a real signal, but $F(s)$ is generally complex:

$$F(s) = A(s) + iB(s)$$
$$= |F(s)| e^{-i2\pi\theta(s)}$$

If $f(x)$ is symmetric, i.e., $f(x) = f(-x)$), then $F(s) = A(s)$. Why?

1D Fourier examples

Spatial domain | Frequency domain



2D Fourier transform

Spatial domain → $F(s_x, s_y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y) e^{-i2\pi(s_x x + s_y y)} dx dy$ → Frequency domain

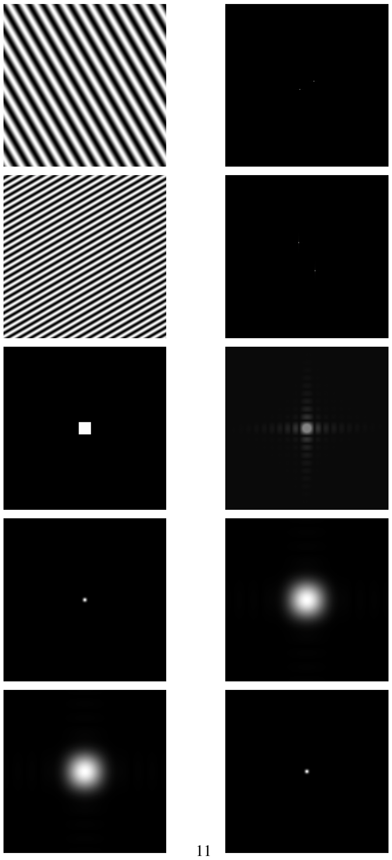
Spatial domain ← $f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(s_x, s_y) e^{i2\pi(s_x x + s_y y)} ds_x ds_y$ ← Frequency domain

Spatial domain | Frequency domain

$f(x,y)$ $\quad$ $|F(s_x, s_y)|$

2D Fourier examples

Spatial domain $f(x,y)$



Frequency domain $|F(s_x, s_y)|$

Convolution

One of the most common methods for filtering a function is called **convolution**.

In 1D, convolution is defined as:

$$g(x) = f(x) * h(x)$$
$$= \int_{-\infty}^{\infty} f(x')h(x - x')dx'$$
$$= \int_{-\infty}^{\infty} f(x')\tilde{h}(x' - x)dx'$$

where $\tilde{h}(x) \equiv h(-x)$.

Convolution properties

Convolution exhibits a number of basic, but important properties.

Commutativity:

$$a(x) * b(x) = b(x) * a(x)$$

Associativity:

$$[a(x) * b(x)] * c(x) = a(x) * [b(x) * c(x)]$$

Linearity:

$$a(x) * [k \cdot b(x)] = k \cdot [a(x) * b(x)]$$

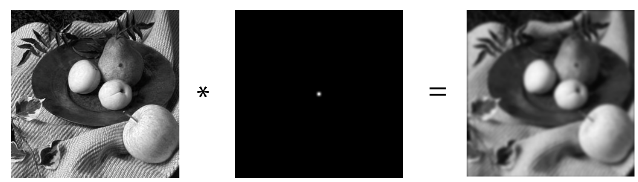
$$a(x) * (b(x) + c(x)) = a(x) * b(x) + a(x) * c(x)$$

Convolution in 2D

In two dimensions, convolution becomes:

$$\begin{aligned} g(x,y) &= f(x,y) * h(x,y) \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x',y')h(x-x',y-y')dx'dy' \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x',y')\tilde{h}(x'-x,y'-y)dx'dy' \end{aligned}$$

where $\tilde{h}(x,y) = h(-x,-y)$.



f(x,y) *h(x,y)* *g(x,y)*

Convolution theorems

Convolution theorem: *Convolution* in the *spatial* domain is equivalent to *multiplication* in the *frequency* domain.

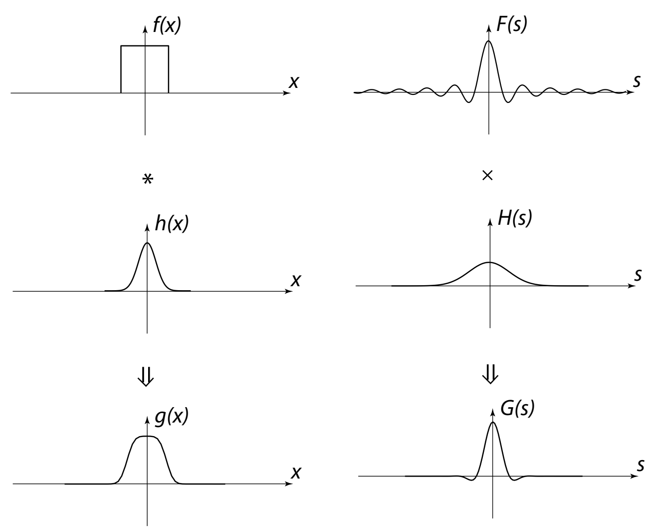
$$f * h \longleftrightarrow F \cdot H$$

Symmetric theorem: *Convolution* in the *frequency* domain is equivalent to *multiplication* in the *spatial* domain.

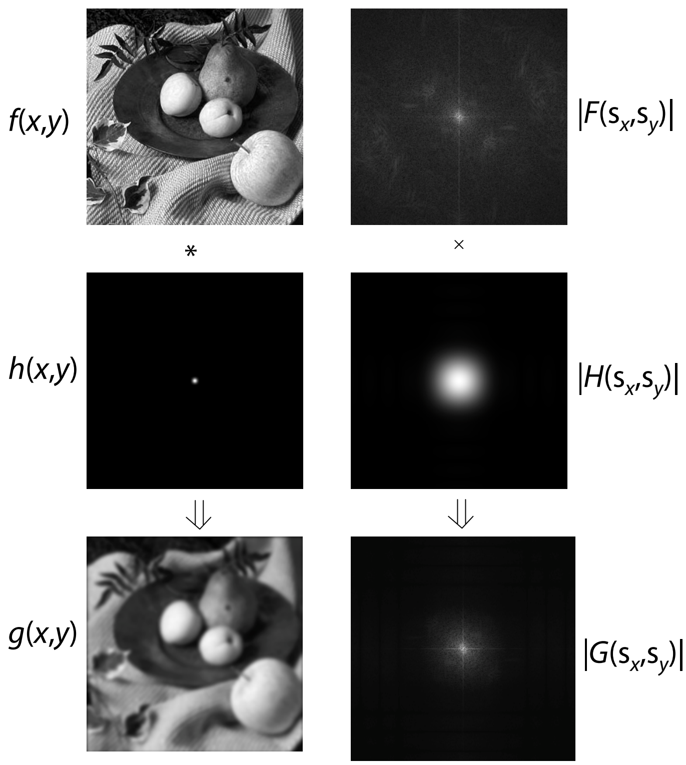
$$f \cdot h \longleftrightarrow F * H$$

1D convolution theorem example

Spatial domain Frequency domain



2D convolution theorem example

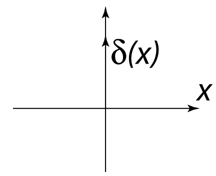


The delta function

The **Dirac delta function**, $\delta(x)$, is a handy tool for sampling theory.

It has zero width, infinite height, and unit area.

It is usually drawn as:

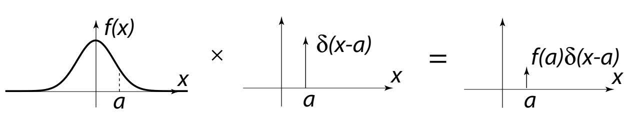


Sifting and shifting

For sampling, the delta function has two important properties.

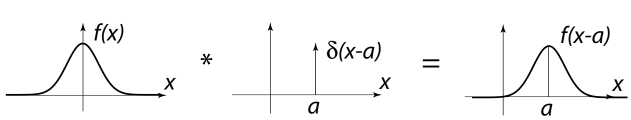
Sifting:

$$f(x)\delta(x-a) = f(a)\delta(x-a)$$



Shifting:

$$f(x) * \delta(x-a) = f(x-a)$$

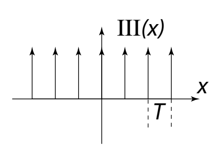


The shah/comb function

A string of delta functions is the key to sampling. The resulting function is called the **shah** or **comb** function:

$$\mathrm{III}(x) = \sum_{n=-\infty}^{\infty} \delta(x - nT)$$

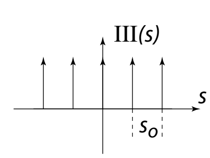
which looks like:



Amazingly, the Fourier transform of the shah function takes the same form:

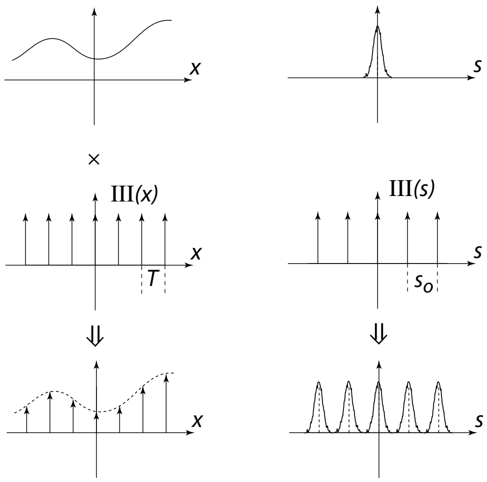
$$\mathrm{III}(s) = \sum_{n=-\infty}^{\infty} \delta(s - ns_o)$$

where $s_o = 1/T$.



Sampling

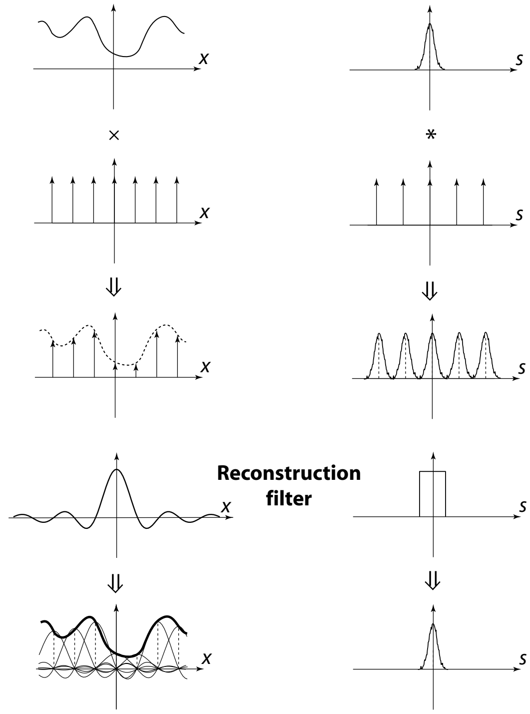
Now, we can talk about sampling.



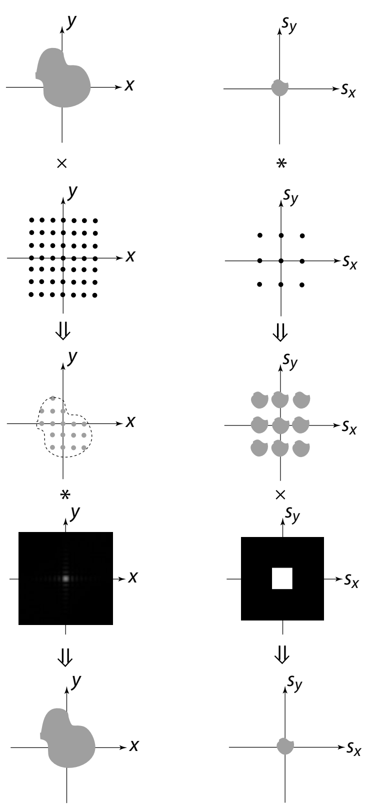
The Fourier spectrum gets *replicated* by spatial sampling!

How do we recover the signal?

Sampling and reconstruction



Sampling and reconstruction in 2D



Sampling theorem

This result is known as the **Sampling Theorem** and is due to Claude Shannon who first discovered it in 1949:

> A signal can be reconstructed from its samples without loss of information, if the original signal has no frequencies above ½ the sampling frequency.

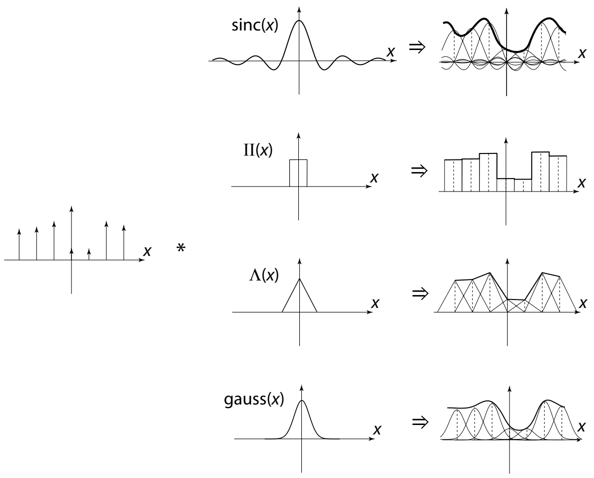
For a given **bandlimited** function, the minimum rate at which it must be sampled is the **Nyquist frequency**.

Reconstruction filters

The sinc filter, while "ideal", has two drawbacks:

- It has large support (slow to compute)
- It introduces ringing in practice

We can choose from many other filters…



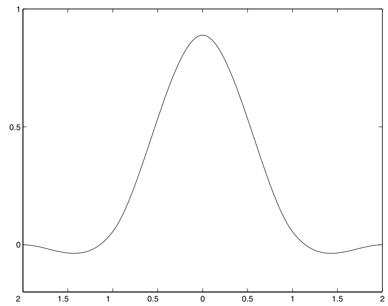
Cubic filters

Mitchell and Netravali (1988) experimented with cubic filters, reducing them all to the following form:

$$r(x) = \frac{1}{6}\begin{cases} (12-9B-6C)|x|^3 + (-18+12B+6C)|x|^2 + (6-2B) & |x| < 1 \\ ((-B-6C)|x|^3 + (6B+30C)|x|^2 + (-12B-48C)|x| + (8B+24C) & 1 \le |x| < 2 \\ 0 & \textit{otherwise} \end{cases}$$

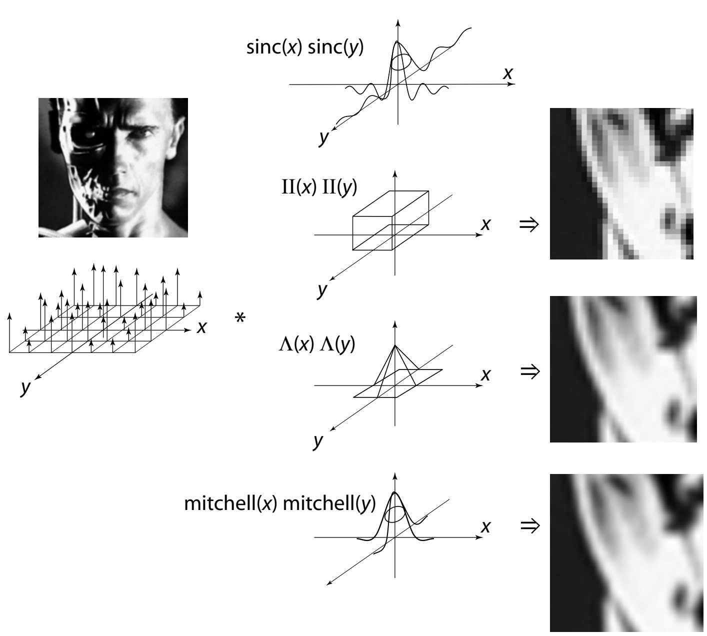
The choice of B or C trades off between being too blurry or having too much ringing. B=C=1/3 was their "visually best" choice.

The resulting reconstruction filter is often called the "Mitchell filter."



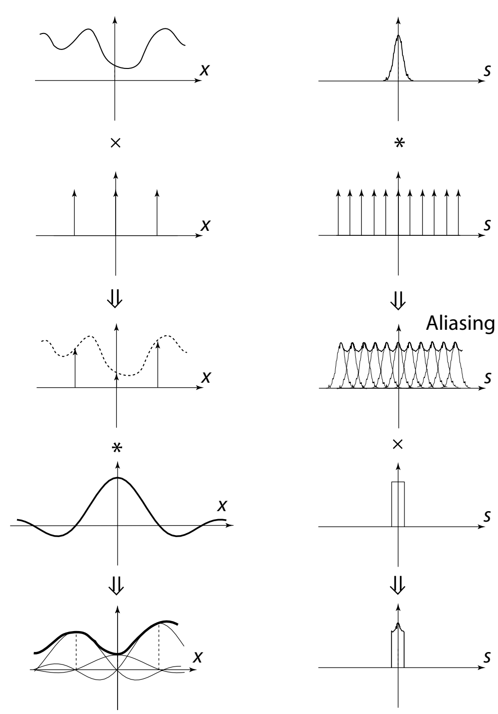
Reconstruction filters in 2D

We can also perform reconstruction in 2D…



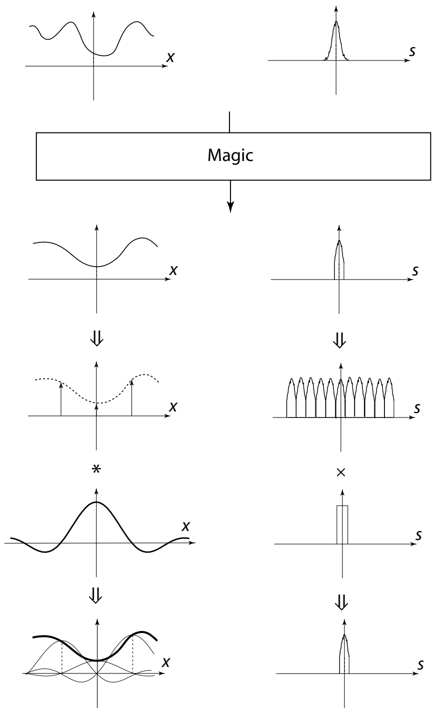
Aliasing

What if we go below the Nyquist frequency?



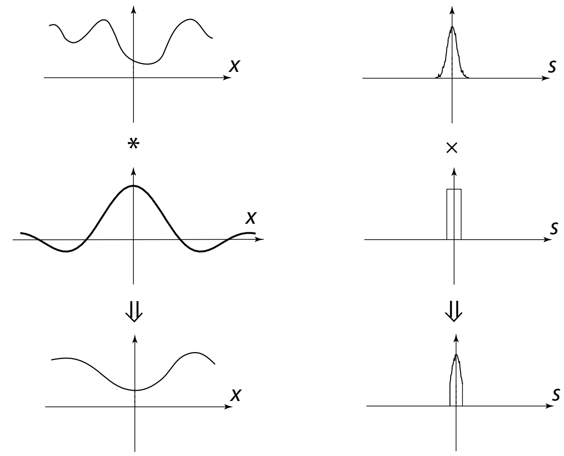
Anti-aliasing

Anti-aliasing is the process of *removing* the frequencies before they alias.



Anti-aliasing by analytic prefiltering

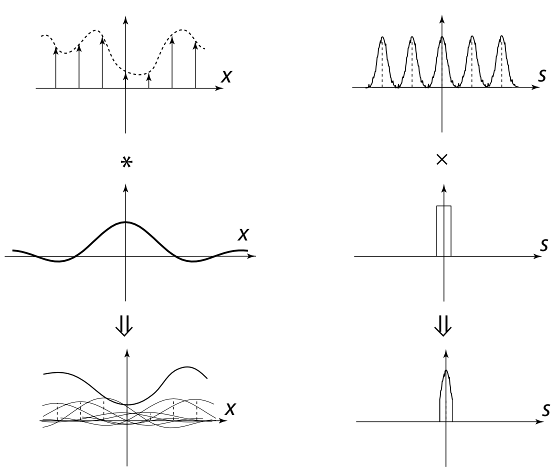
We can fill the "magic" box with analytic pre-filtering of the signal:



Why may this not generally be possible?

Filtered downsampling

Alternatively, we can sample the image at a higher rate, and then filter that signal:



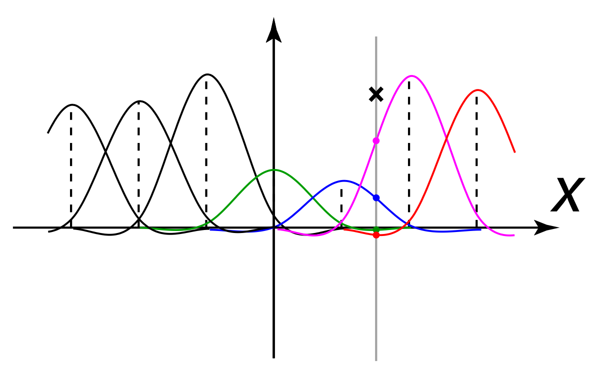
We can now sample the signal at a lower rate. The whole process is called filtered **downsampling** or **supersampling and averaging down**.

Practical upsampling

When resampling a function (e.g., when resizing an image), you do not need to reconstruct the complete continuous function.

For zooming in on a function, you need only use a reconstruction filter and evaluate as needed for each new sample.

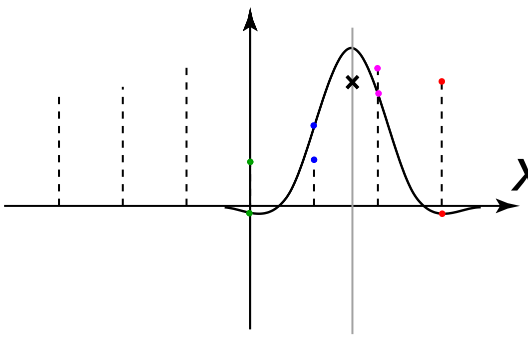
Here's an example using a cubic filter:



Practical upsampling

This can also be viewed as:

1. putting the reconstruction filter at the desired location
2. evaluating at the original sample positions
3. taking products with the sample values themselves
4. summing it up



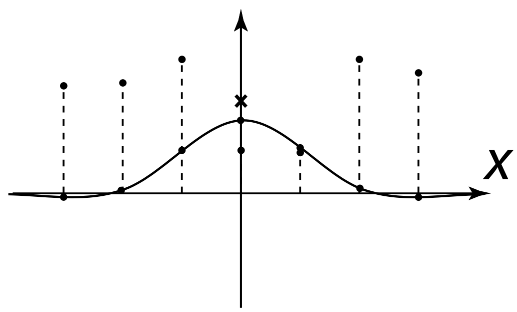
Important: filter should always be normalized!

Practical downsampling

Downsampling is similar, but filter has larger support and smaller amplitude.

Operationally:

1. Choose filter in downsampled space.
2. Compute the downsampling rate, *d*, ratio of new sampling rate to old sampling rate
3. Stretch the filter by 1/*d* and scale it down by *d*
4. Follow upsampling procedure (previous slides) to compute new values



2D resampling

We've been looking at **separable** filters:

$$r_{2D}(x,y) = r_{1D}(x) r_{1D}(y)$$

How might you use this fact for efficient resampling in 2D?