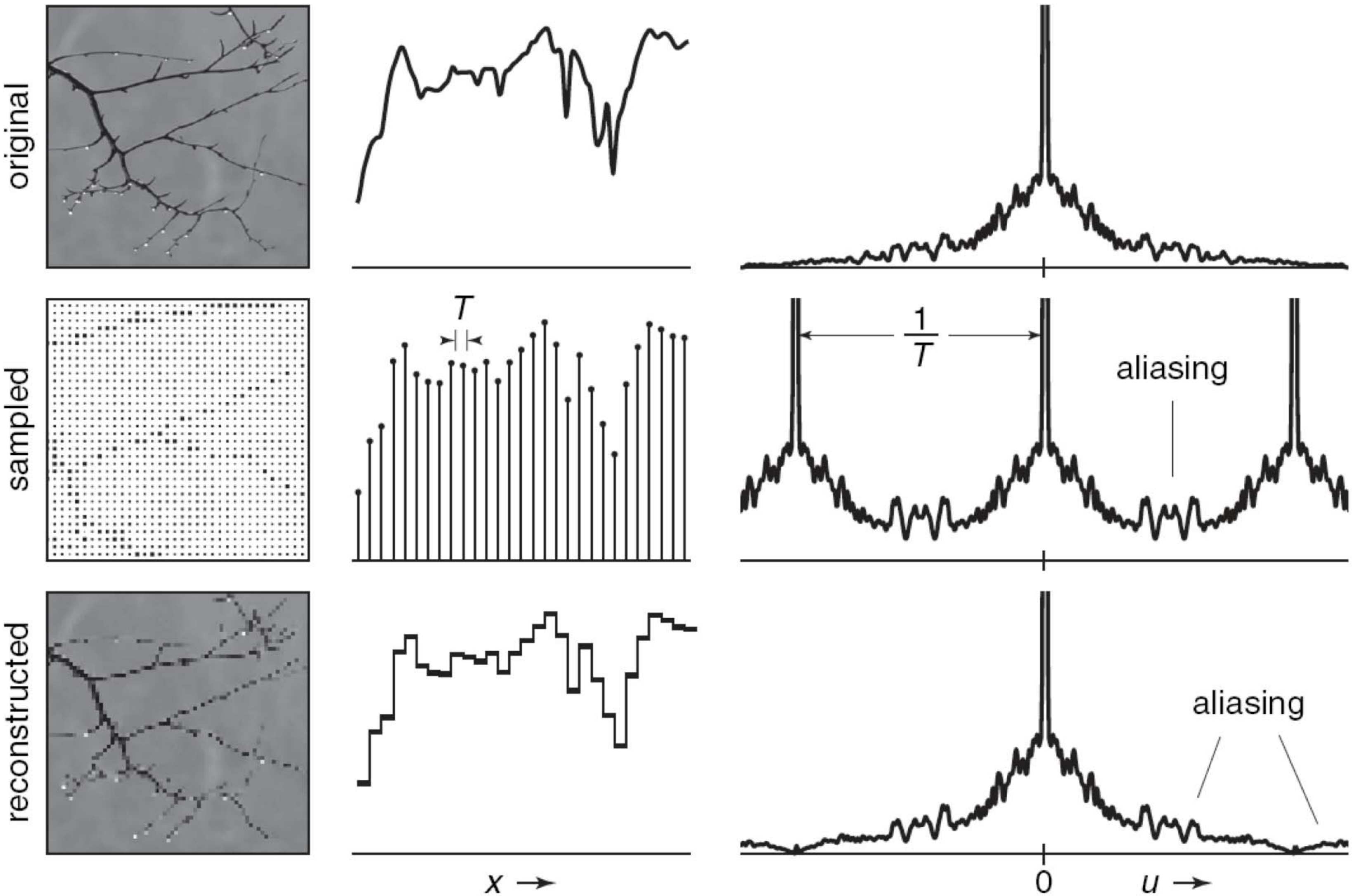
Anti-Aliasing

Jian Huang

CS456

Aliasing?

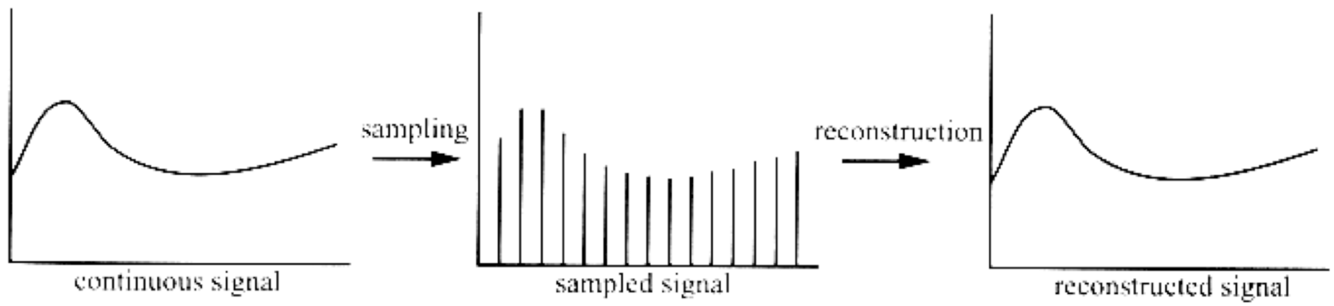


original
sampled
reconstructed
T
1/T
aliasing
aliasing
x →
0
u →

Aliasing

- Aliasing comes from in-adequate sampling rates of the continuous signal
- The theoretical foundation of anti-aliasing has to do with frequency analysis
- It’s always easier to look at 1D cases, so let’s first look at a few of those.

Example of Sampling



Fourier Analysis

- By looking at F(u), we get a feel for the “frequencies” of the signal.
- We also call this frequency space.
- Intuitively, you can envision, the sharper an edge, the higher the frequencies.
- From a numerical analysis standpoint, the sharper the edge the greater the tangent magnitude, and hence the interpolation errors.

Fourier Analysis

- Bandlimited
  - We say a function is bandlimited, if F(u)=0 for all frequencies u>c and u<-c.
- Amplitude Spectrum
  - The magnitude, |F(u)|, is called the amplitude spectrum or simply the spectrum.
- Phase Spectrum or Phase

$$\Phi(u) = \tan^{-1}\left(\frac{\mathrm{Im}(u)}{\mathrm{Re}(u)}\right)$$

Fourier Properties

- Linearity

$$af(x) + bg(x) \Leftrightarrow aF(u) + bG(u)$$

- Scaling

$$f(ax) \Leftrightarrow \frac{1}{a} F(\frac{u}{a})$$

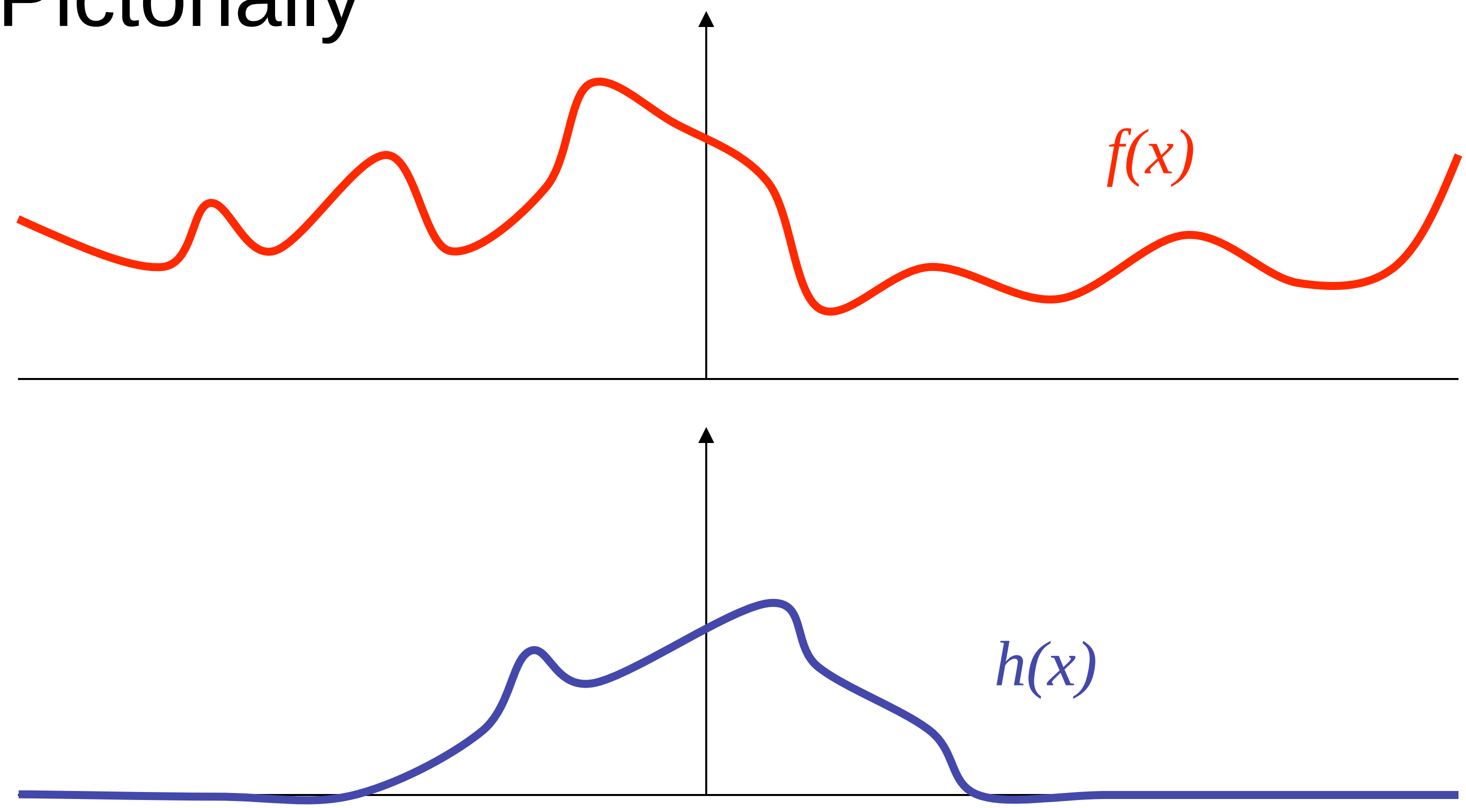
Convolution

- Definition:

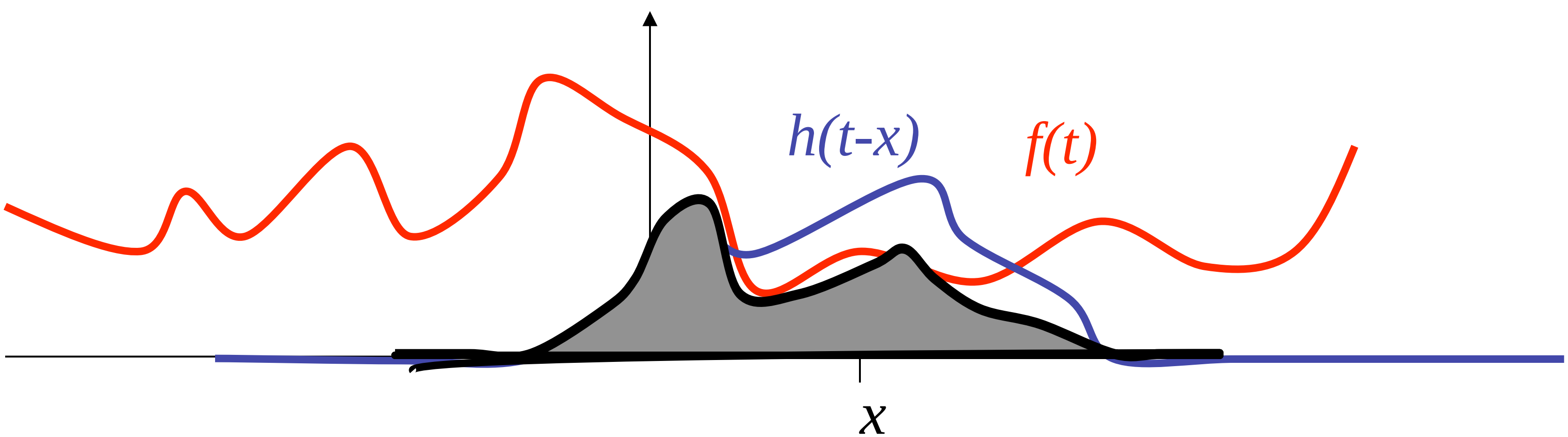
$$f(x) \otimes h(x) = \int_{-\infty}^{\infty} f(t)h(t-x)dt$$

Convolution

- Pictorially



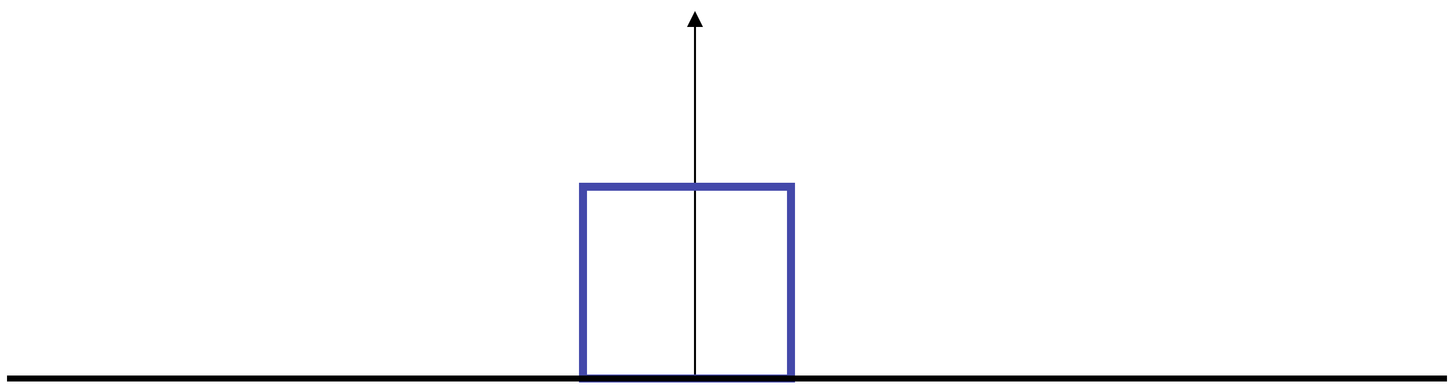
Convolution



Convolution

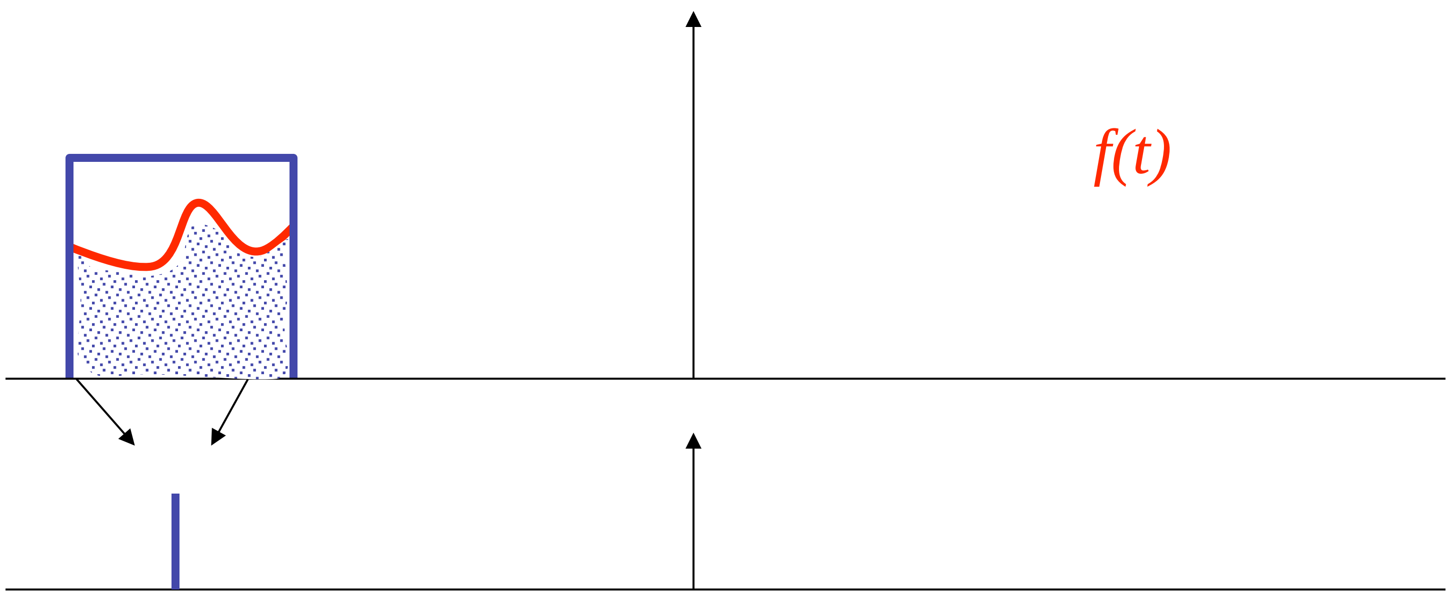
- Consider the function (box filter):

$$h(x) = \begin{cases} 0 & x < -\frac{1}{2} \\ 1 & -\frac{1}{2} \le x \le \frac{1}{2} \\ 0 & x > \frac{1}{2} \end{cases}$$



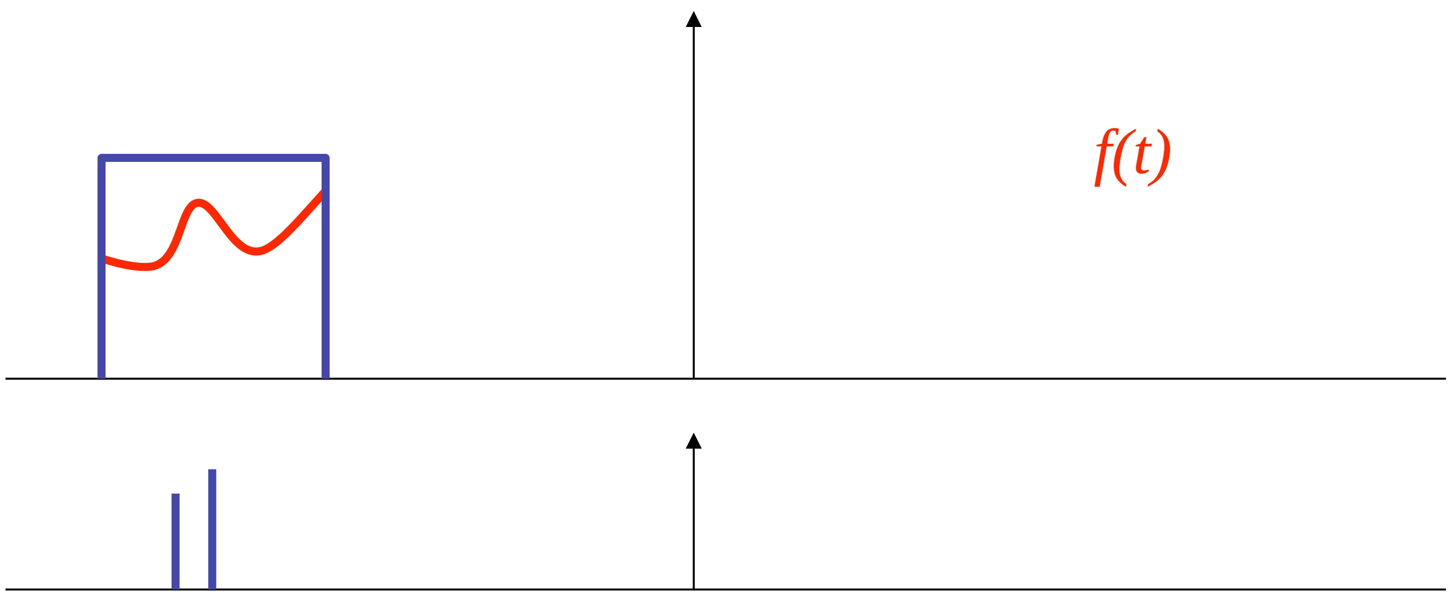
Convolution

- This function ***windows*** our function *f*(*x*).



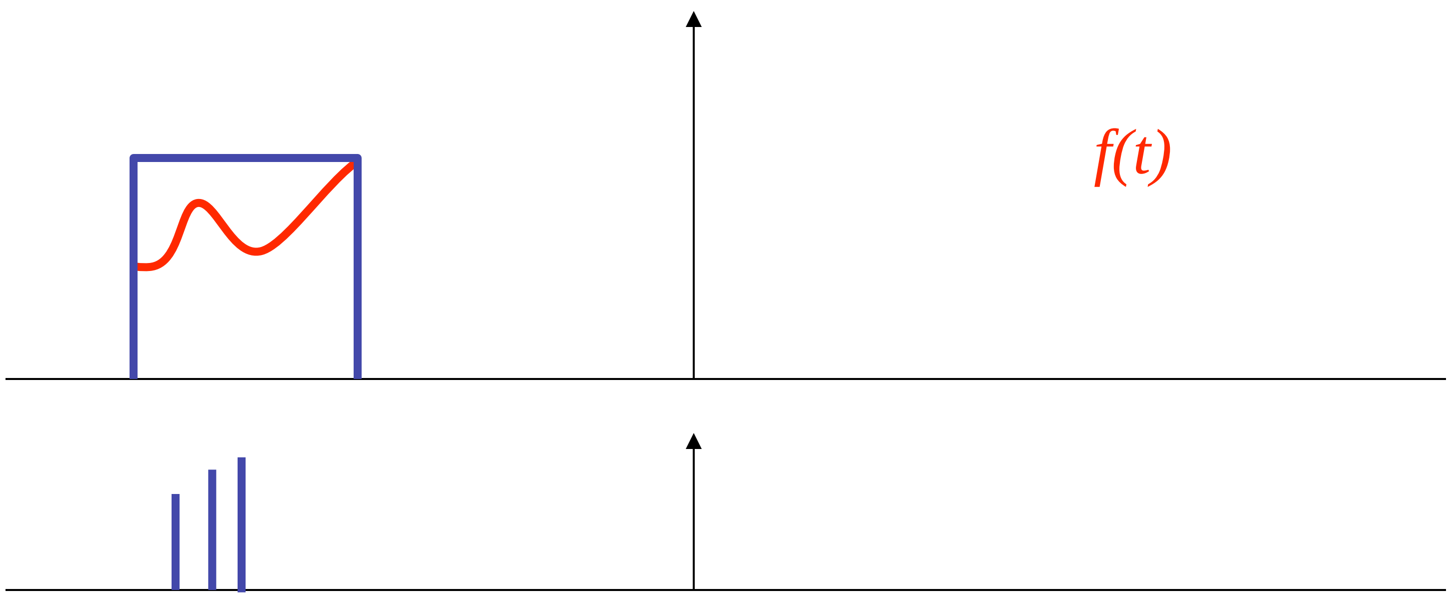
Convolution

- This function ***windows*** our function $f(x)$.



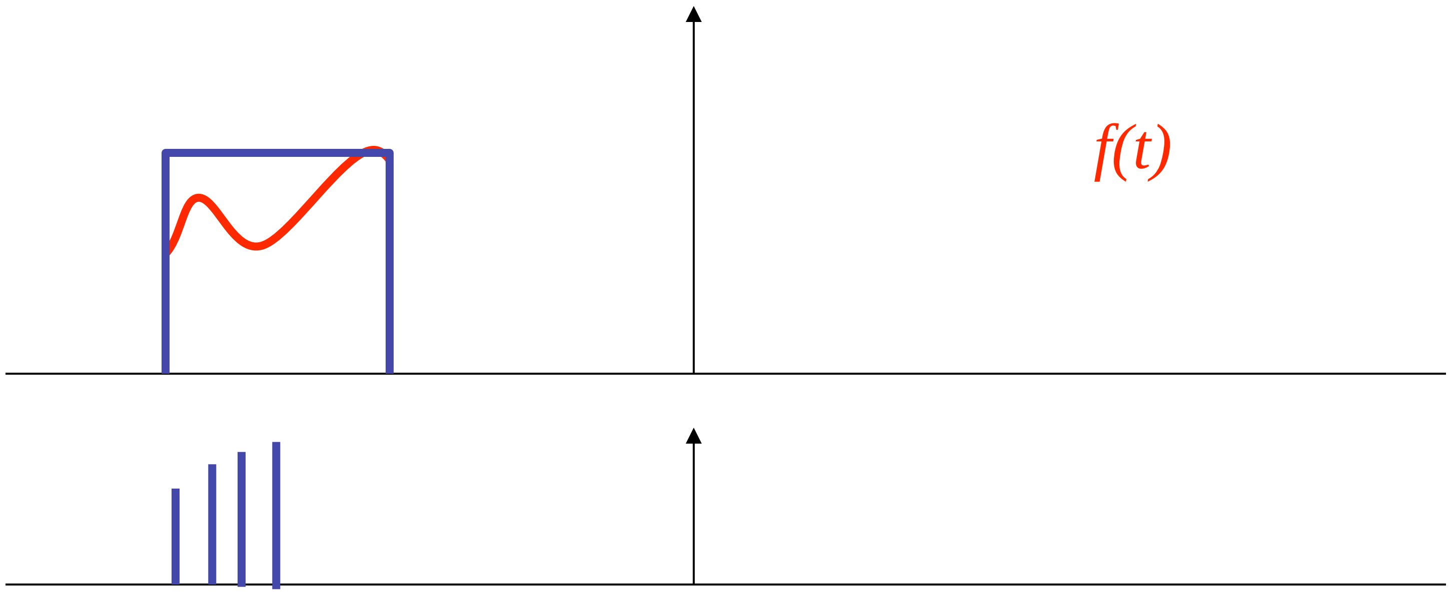
Convolution

- This function ***windows*** our function *f(x)*.



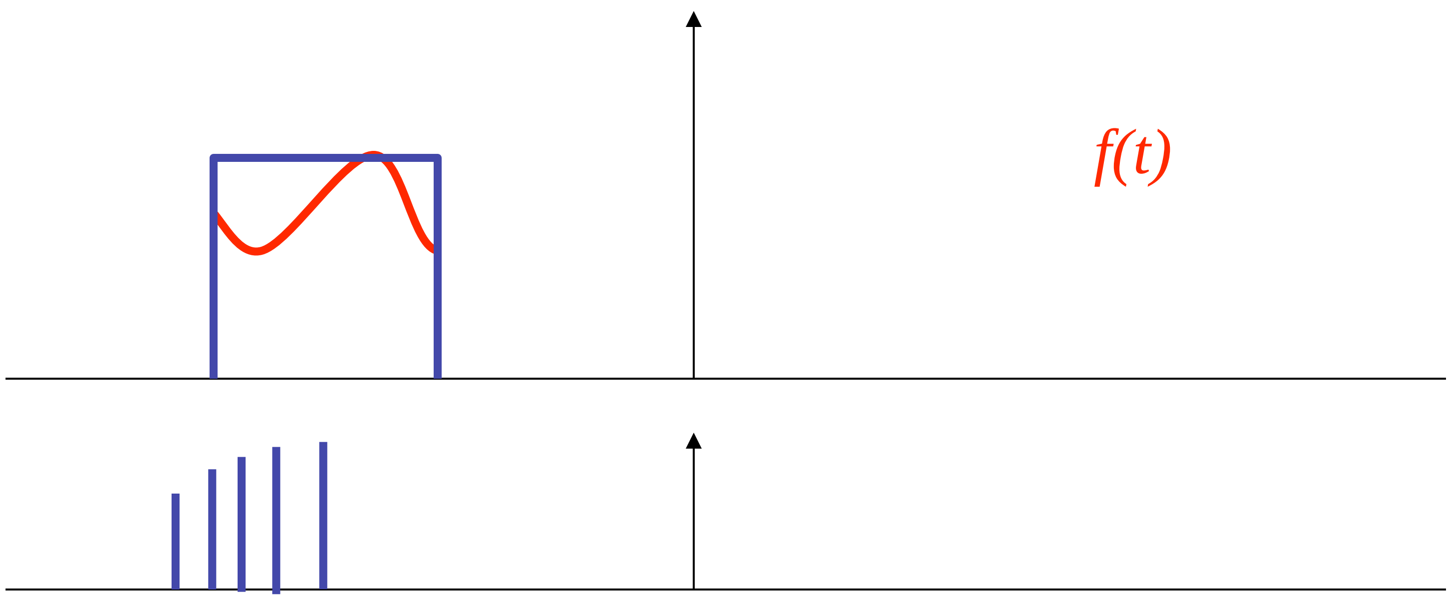
Convolution

- This function ***windows*** our function *f(x)*.



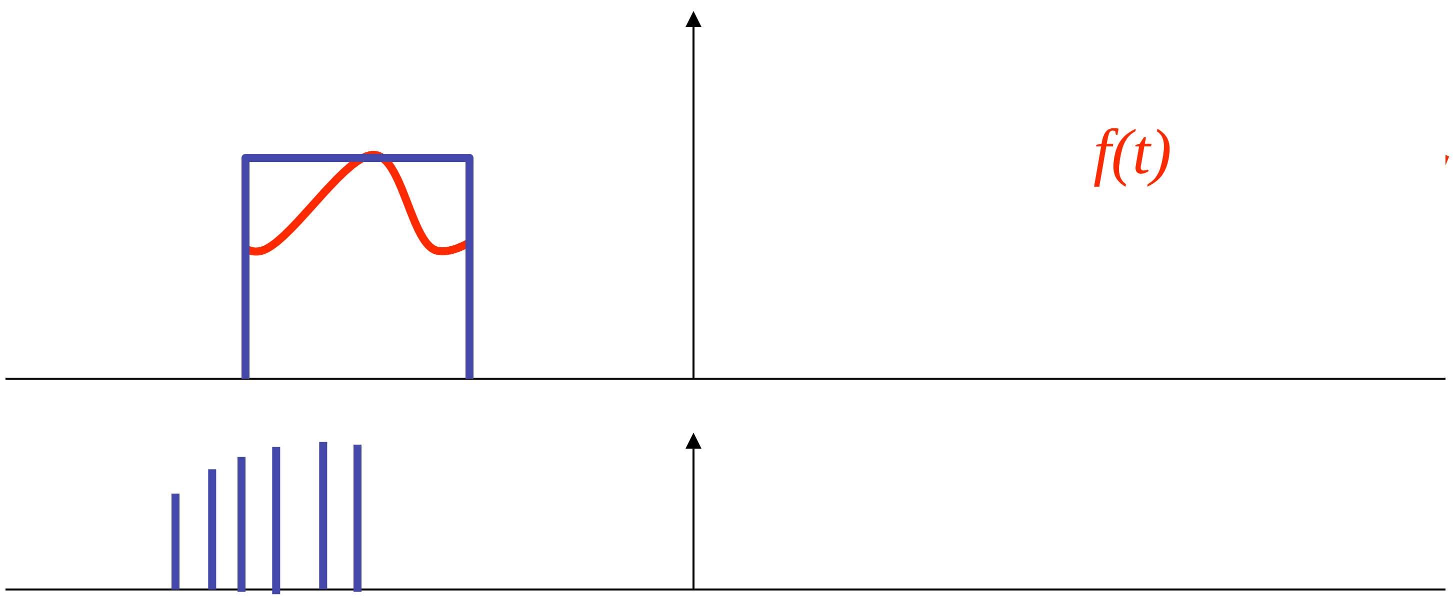
Convolution

- This function ***windows*** our function *f(x)*.



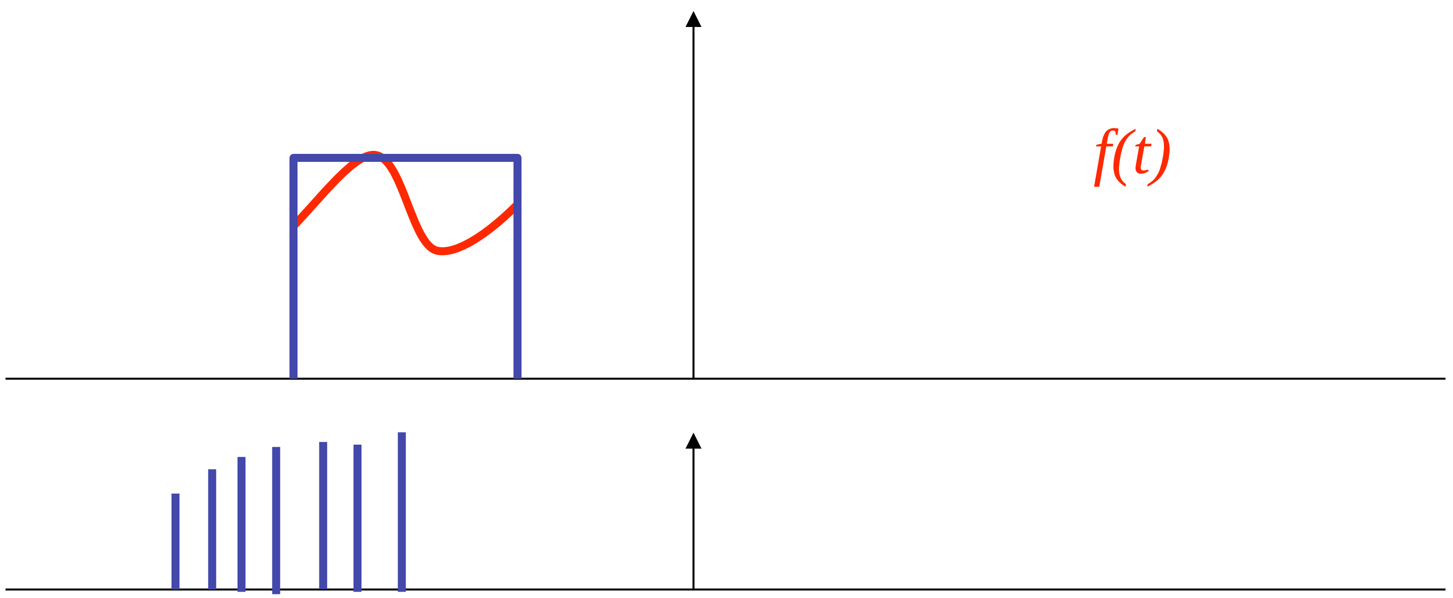
Convolution

- This function ***windows*** our function *f*(*x*).



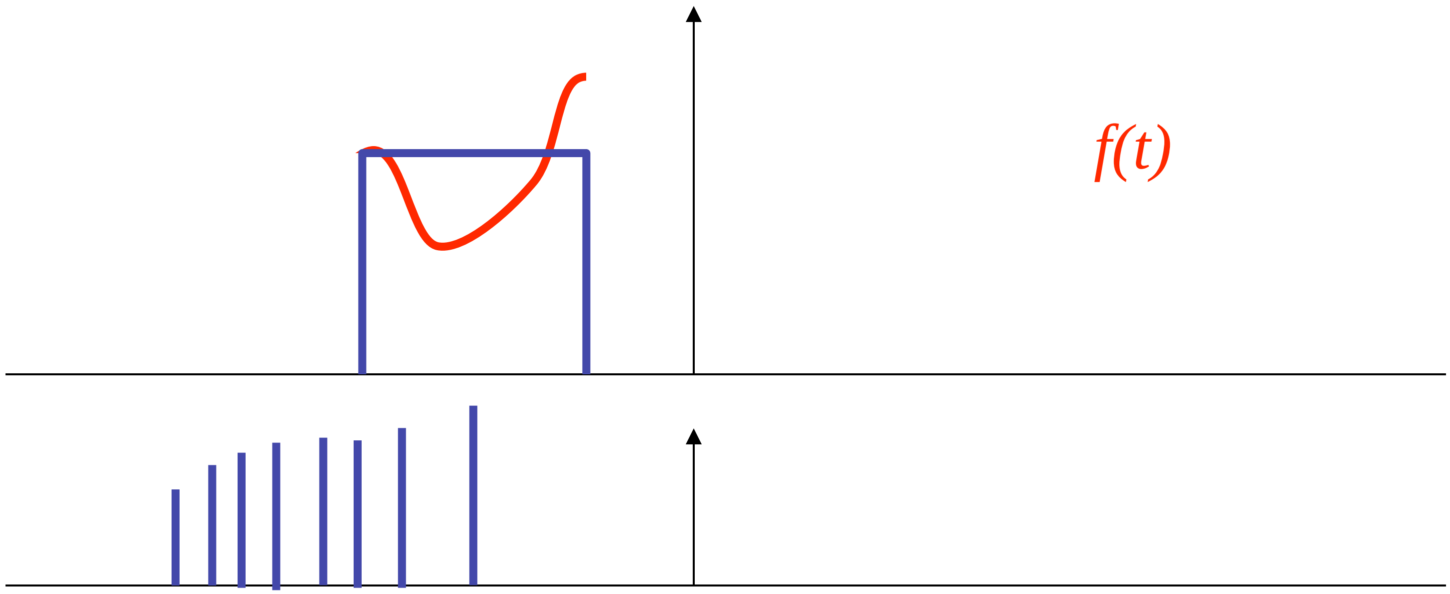
Convolution

- This function ***windows*** our function *f(x)*.



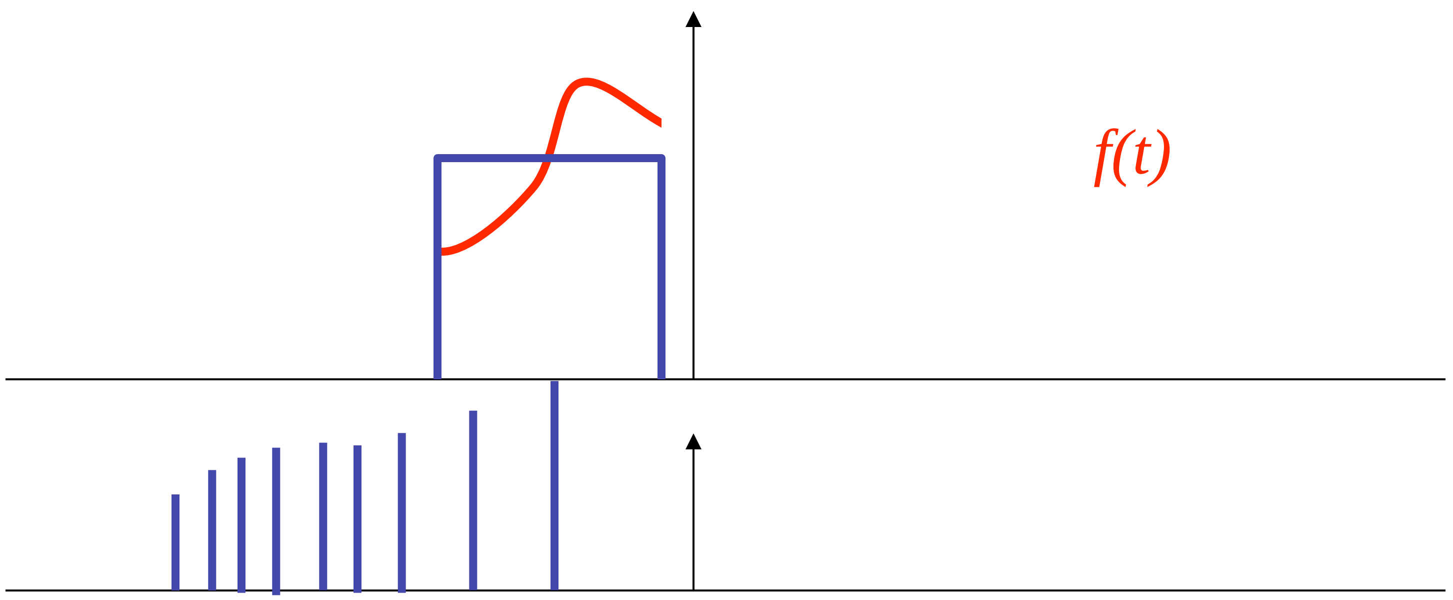
Convolution

- This function ***windows*** our function *f(x)*.



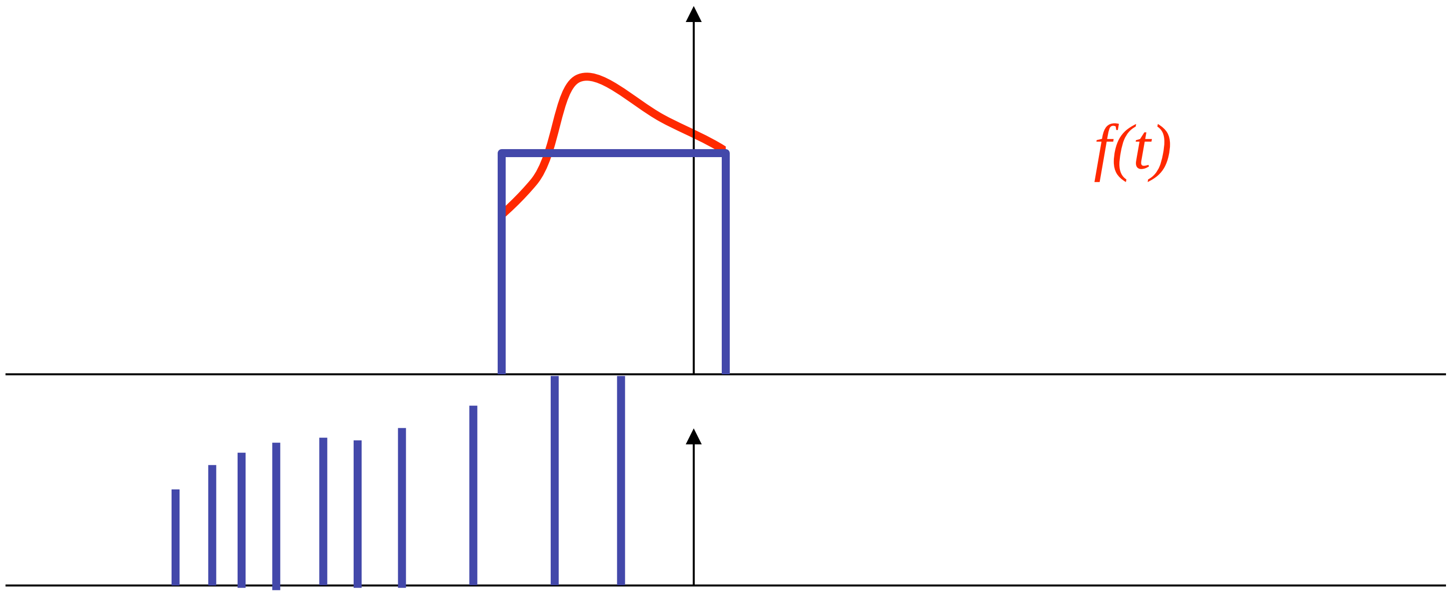
Convolution

- This function ***windows*** our function *f(x)*.



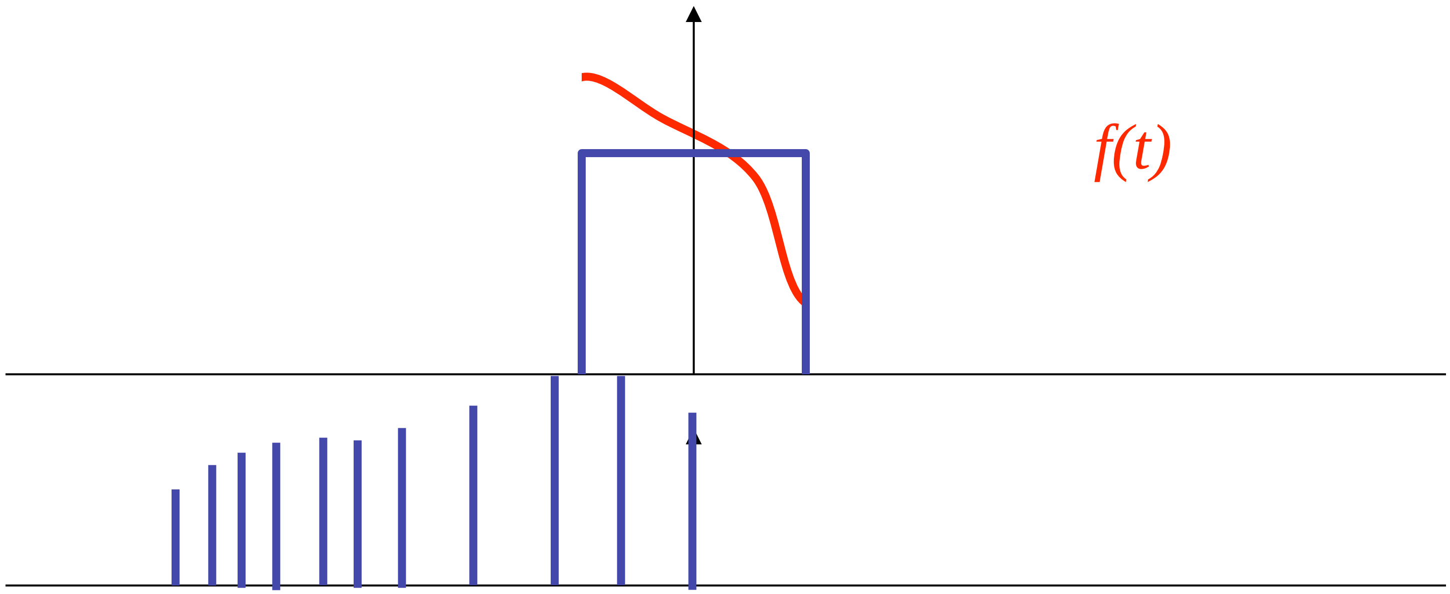
Convolution

- This function ***windows*** our function *f(x)*.



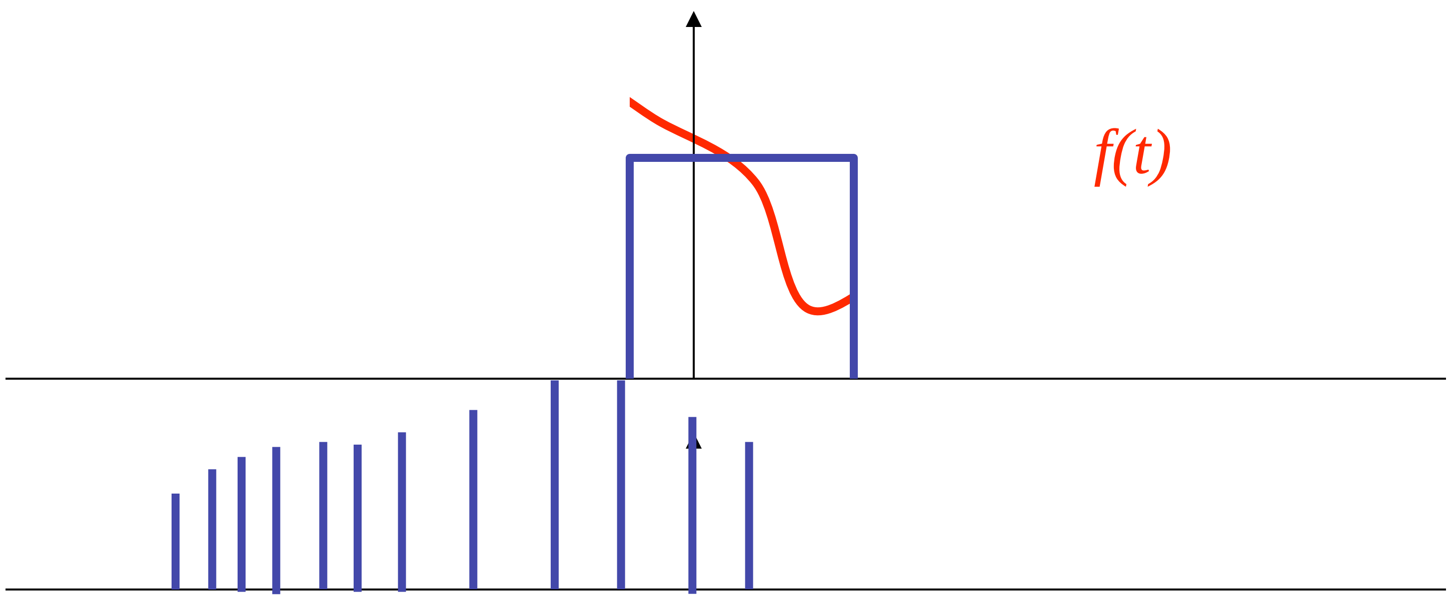
Convolution

- This function ***windows*** our function *f(x)*.



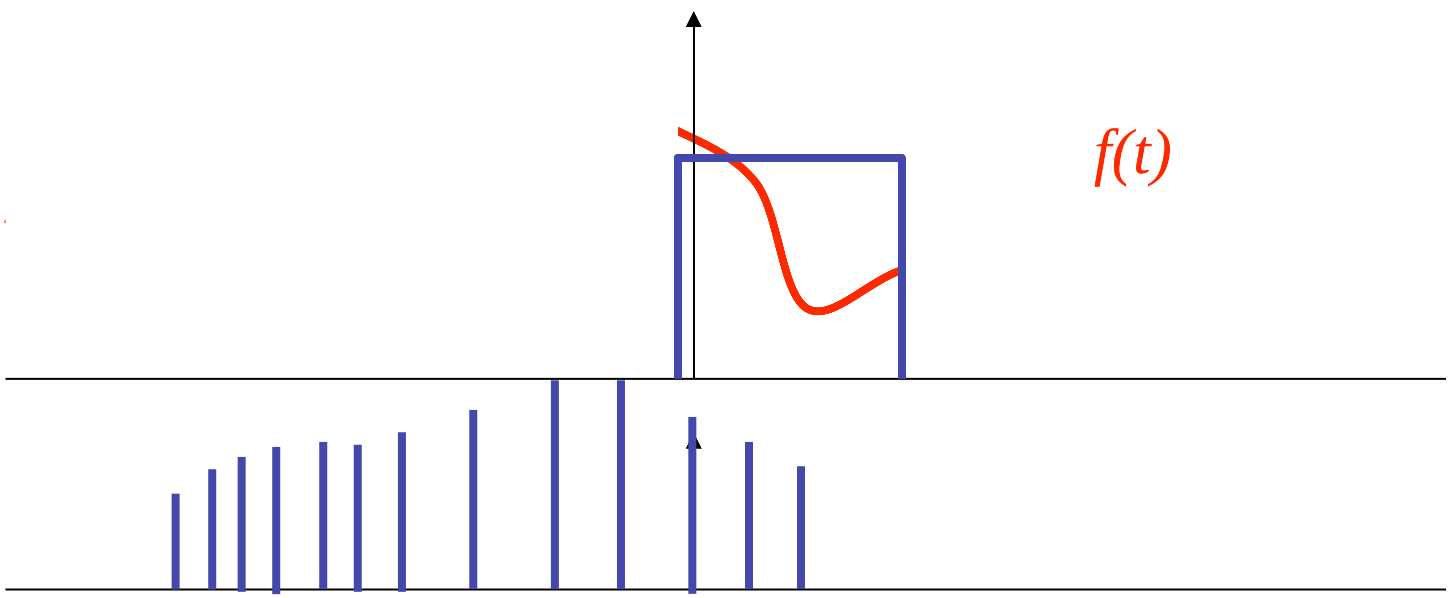
Convolution

- This function ***windows*** our function *f(x)*.



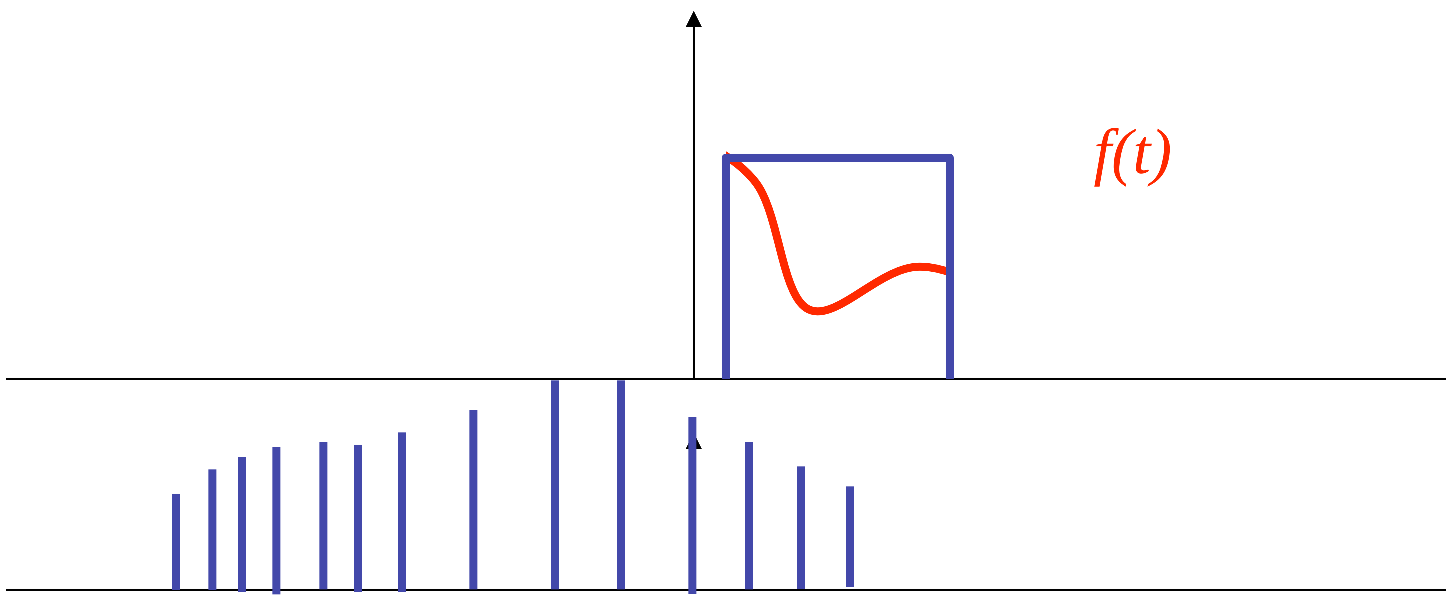
Convolution

- This function ***windows*** our function *f(x)*.



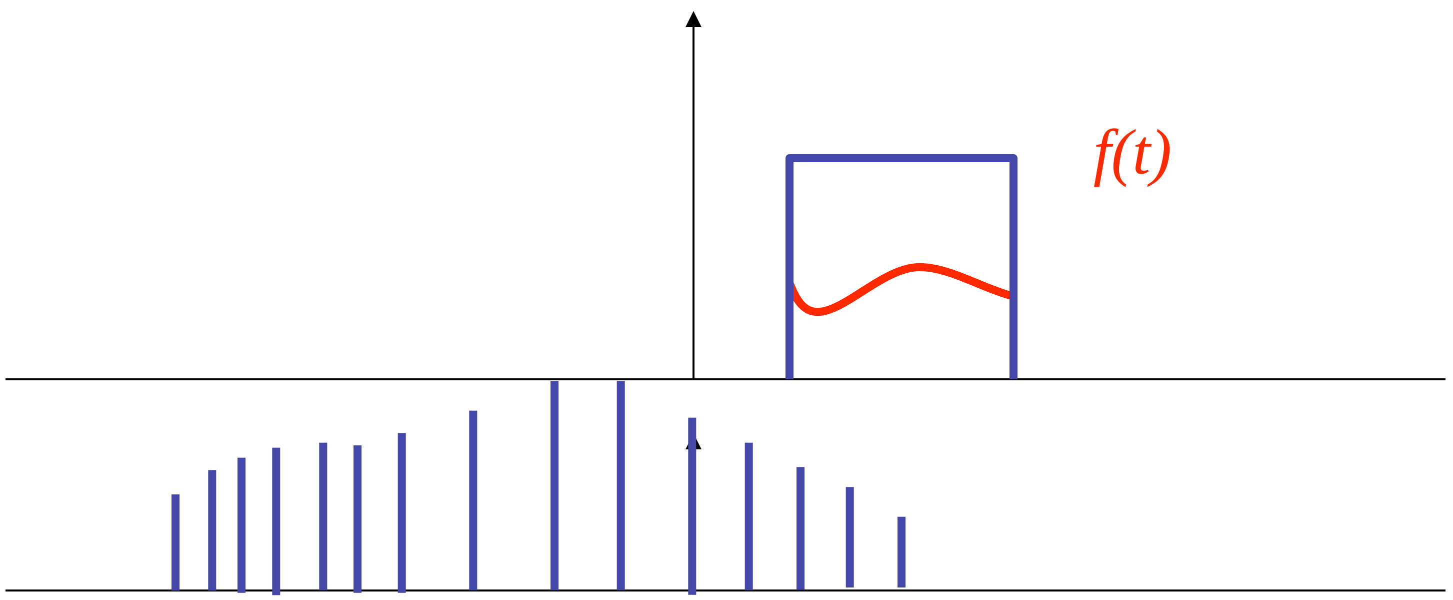
Convolution

- This function ***windows*** our function *f(x)*.



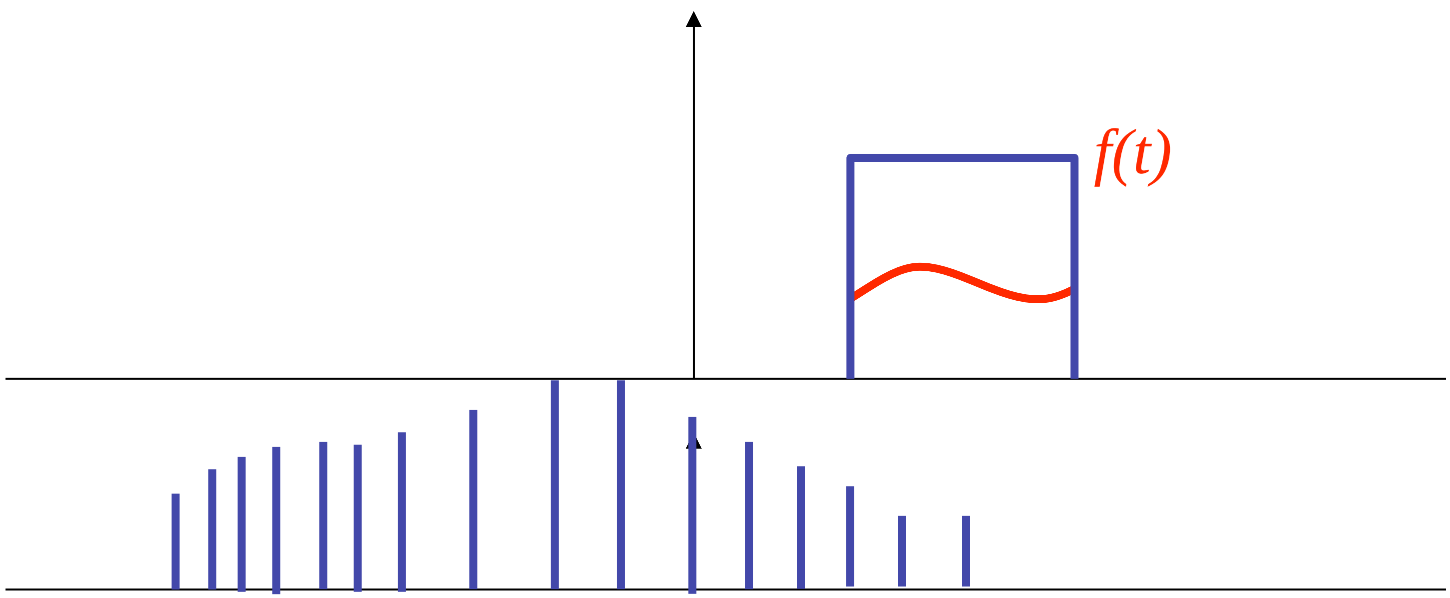
Convolution

- This function ***windows*** our function *f(x)*.



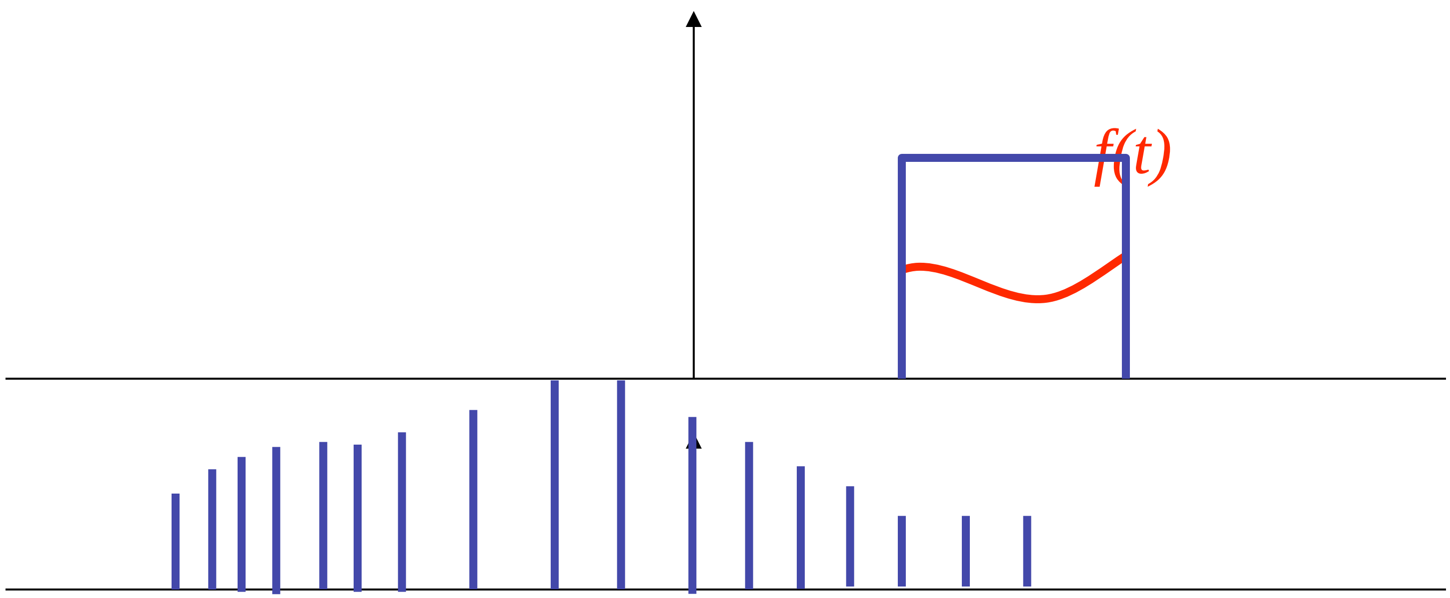
Convolution

- This function ***windows*** our function *f(x)*.



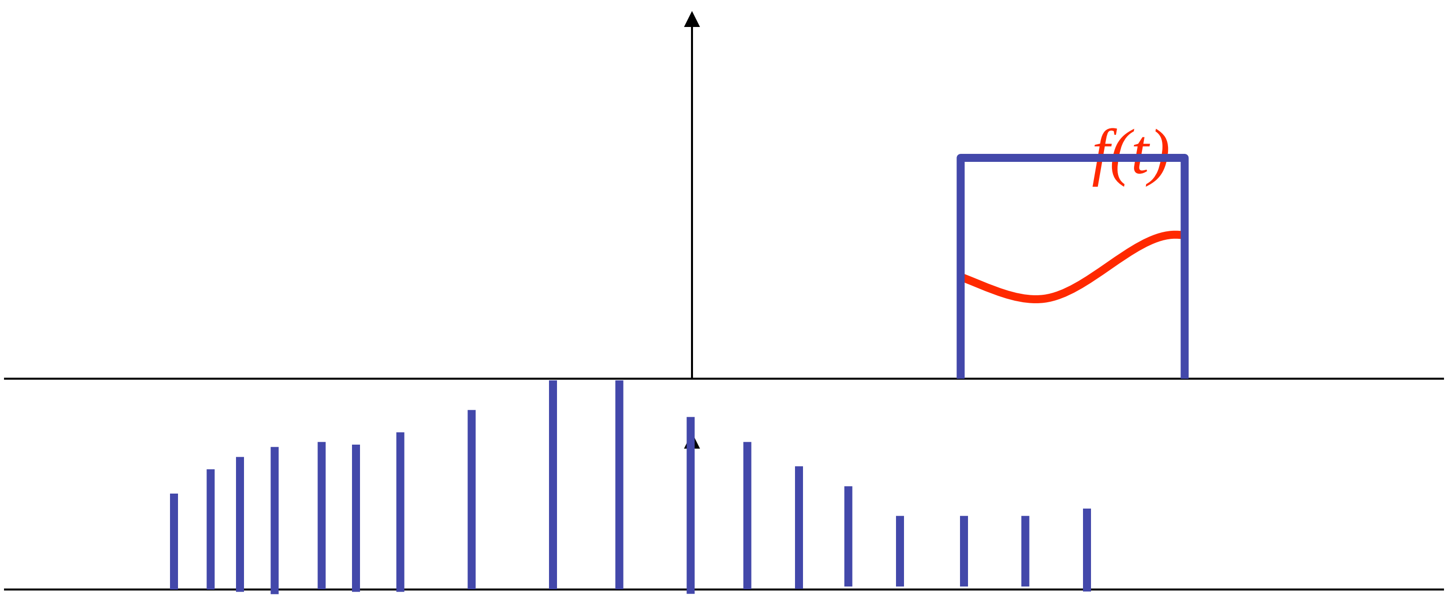
Convolution

- This function ***windows*** our function *f(x)*.



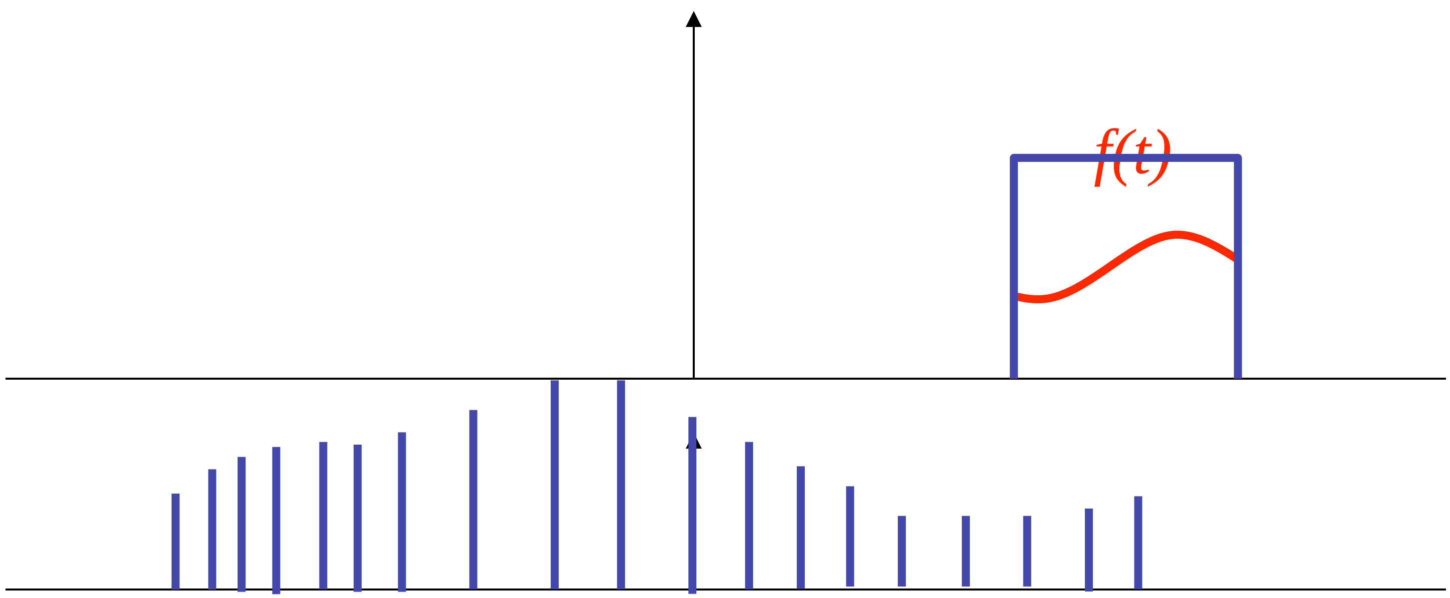
Convolution

- This function ***windows*** our function *f(x)*.



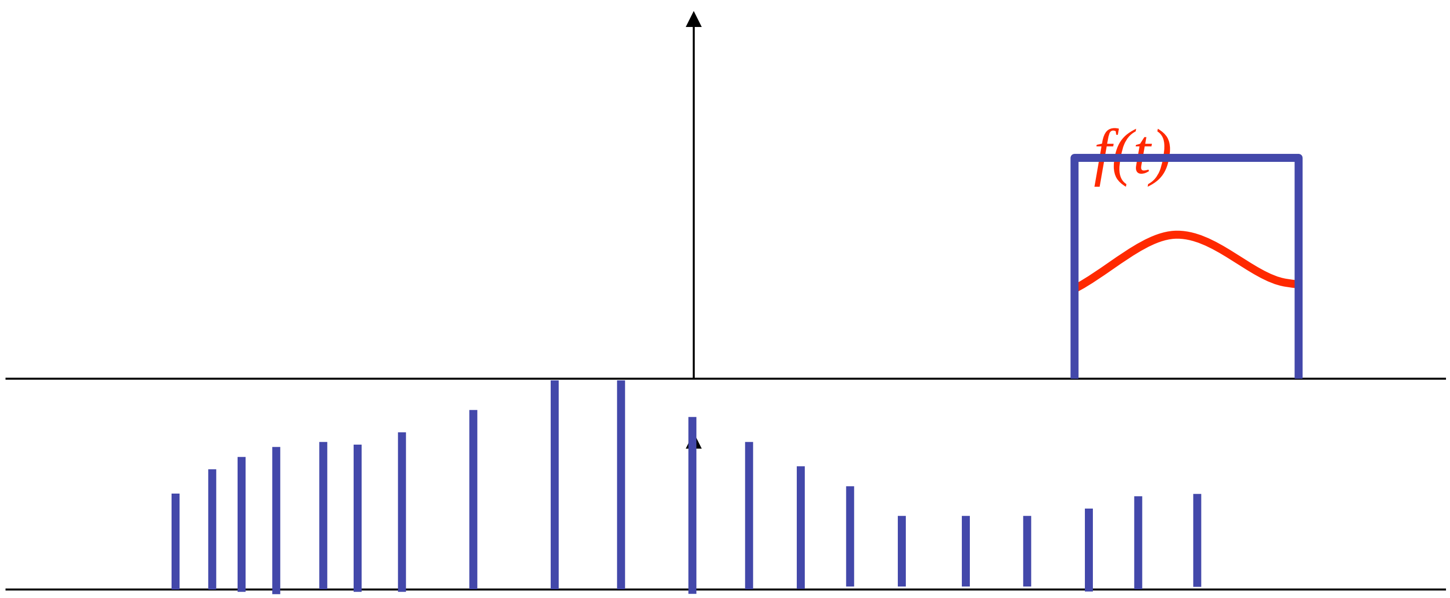
Convolution

- This function ***windows*** our function *f(x)*.



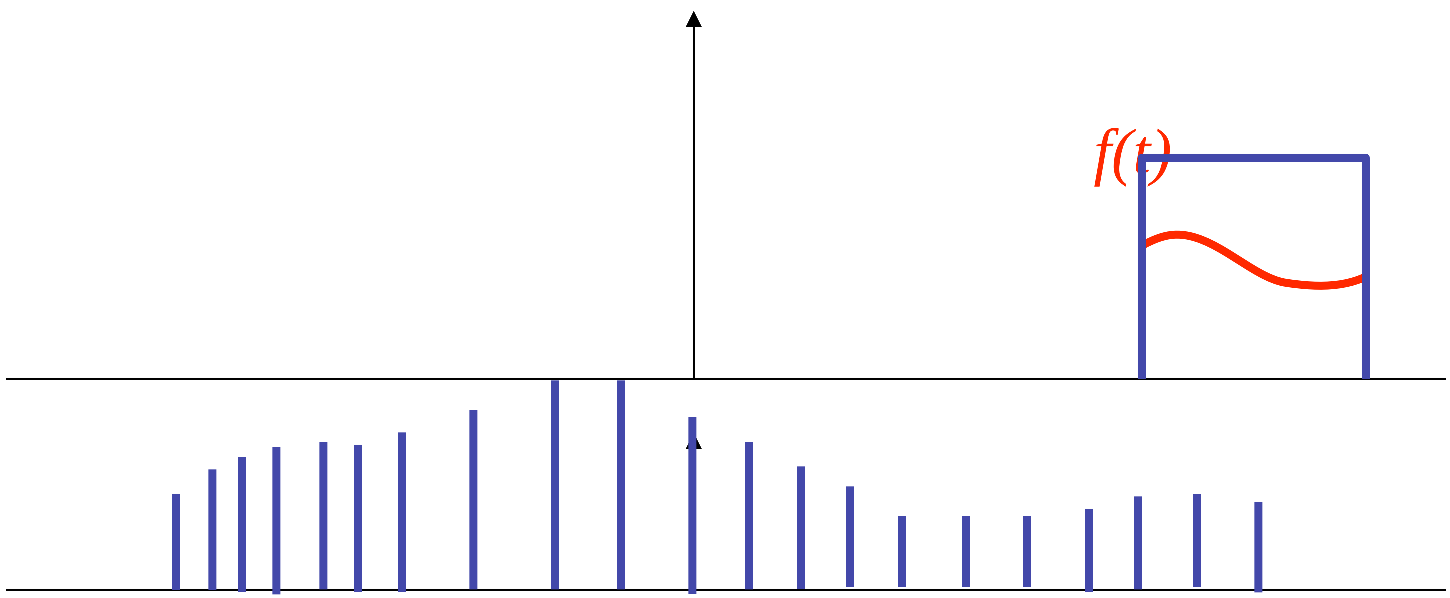
Convolution

- This function ***windows*** our function *f(x)*.



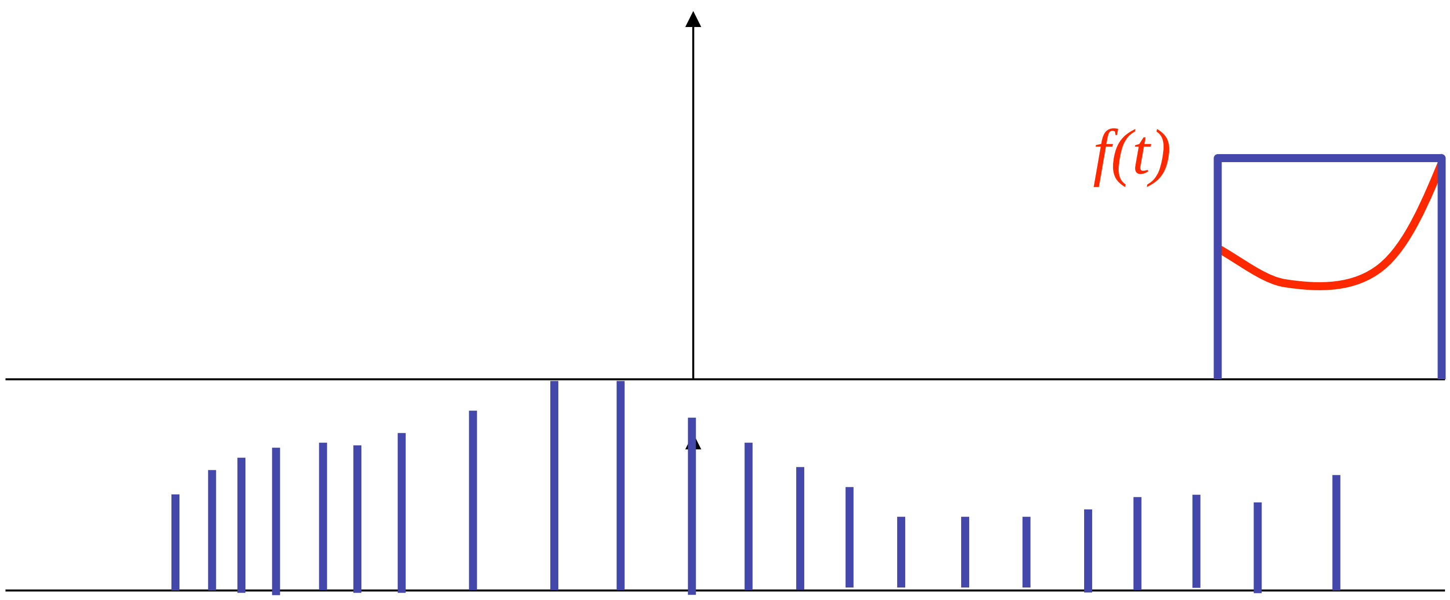
Convolution

- This function ***windows*** our function *f(x)*.



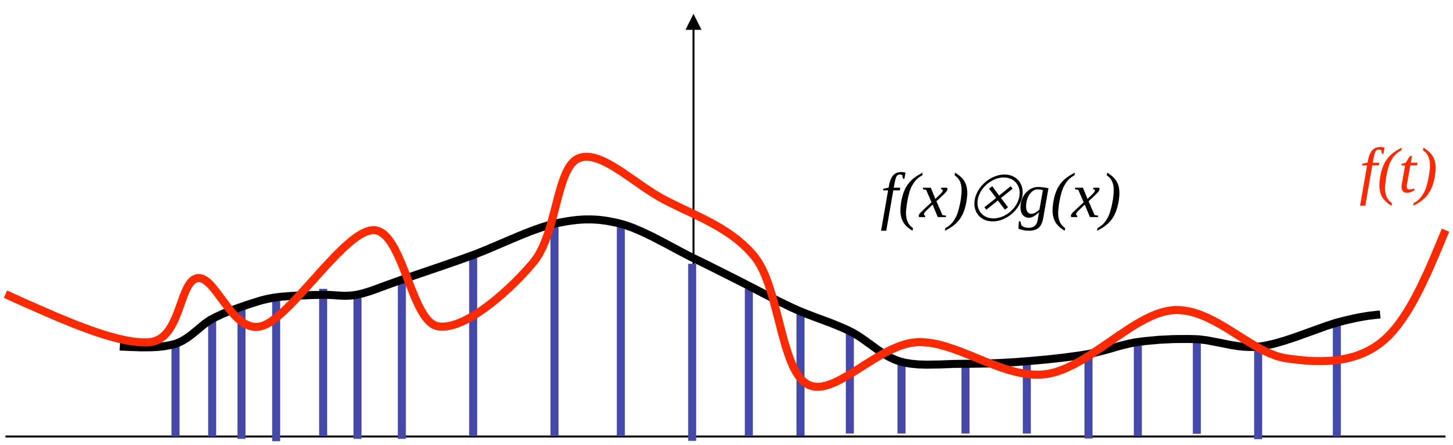
Convolution

- This function ***windows*** our function *f*(*x*).

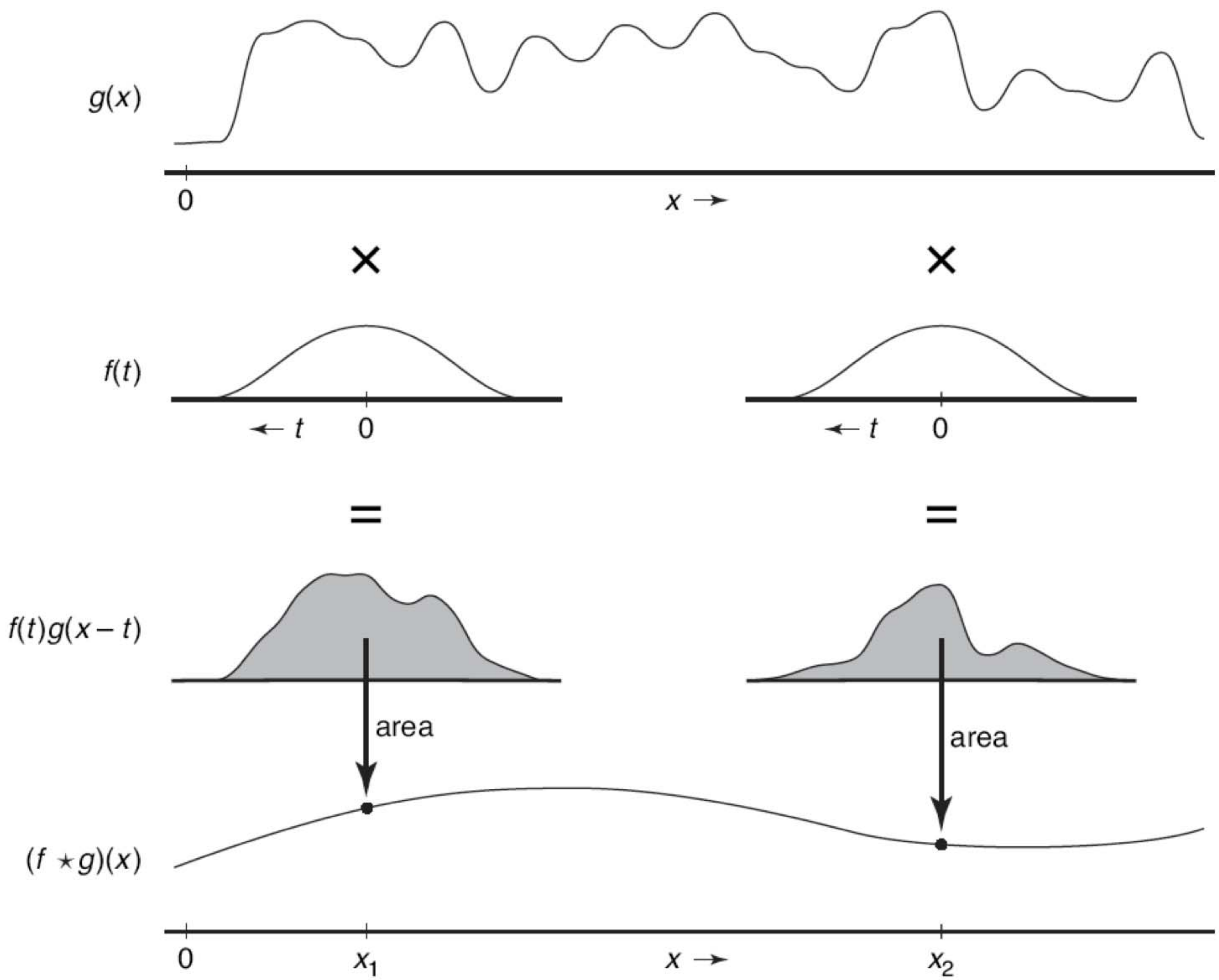


Convolution

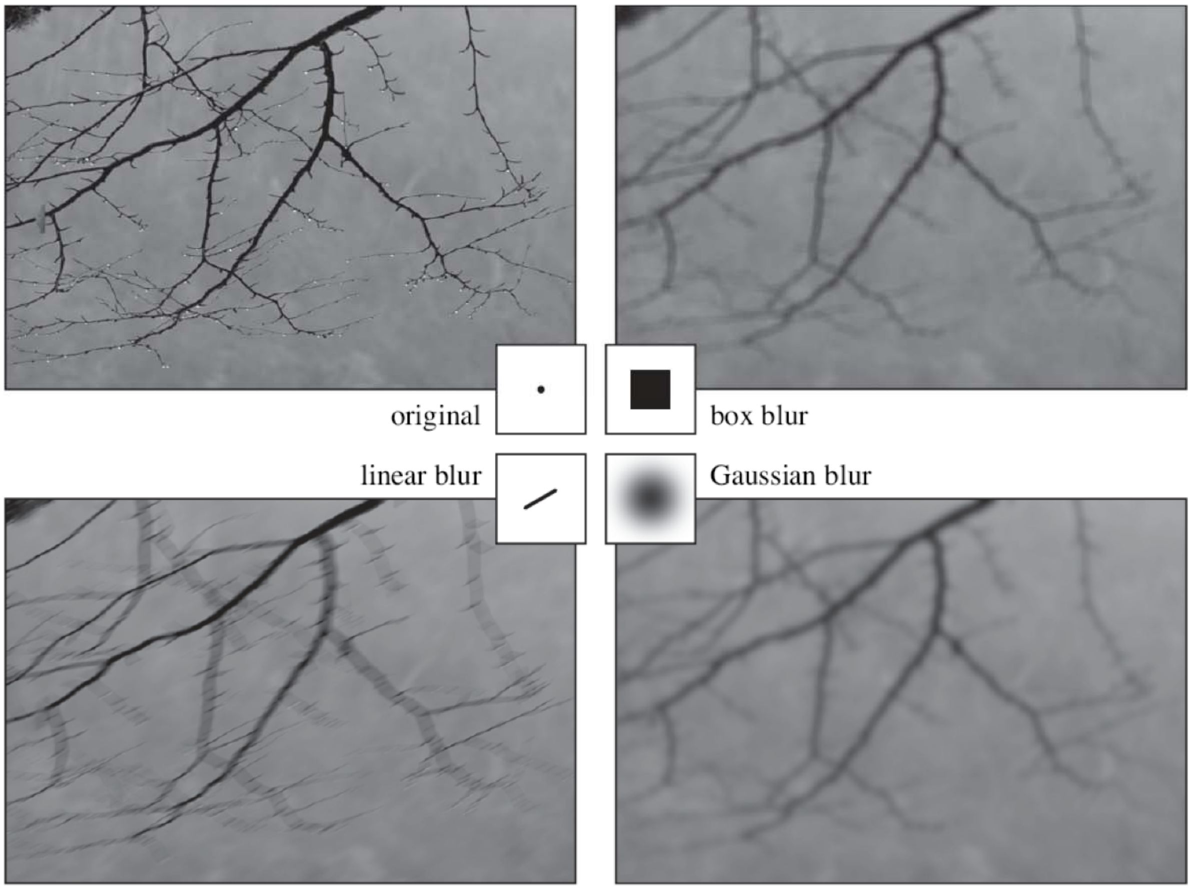
- This particular convolution smooths out some of the high frequencies in *f(x)*.



Another Look At Convolution



Filtering and Convolution



Different functions achieve different Results.

Aliasing

- What this says, is that any frequencies greater than a certain amount will appear intermixed with other frequencies.
- In particular, the higher frequencies for the copy at 1/T intermix with the low frequencies centered at the origin.

Aliasing and Sampling

- Note, that the sampling process introduces frequencies out to infinity.
- We have also lost the function *f(x),* and now have only the discrete samples.
- This brings us to our next powerful theory.

Sampling Theorem

- **The Shannon Sampling Theorem**

  A band-limited signal *f(x),* with a cutoff frequency of λ, that is sampled with a sampling spacing of T may be perfectly reconstructed from the discrete values *f[nT]* by convolution with the *sinc(x)* function, provided:

$$\lambda < \frac{1}{2T}$$

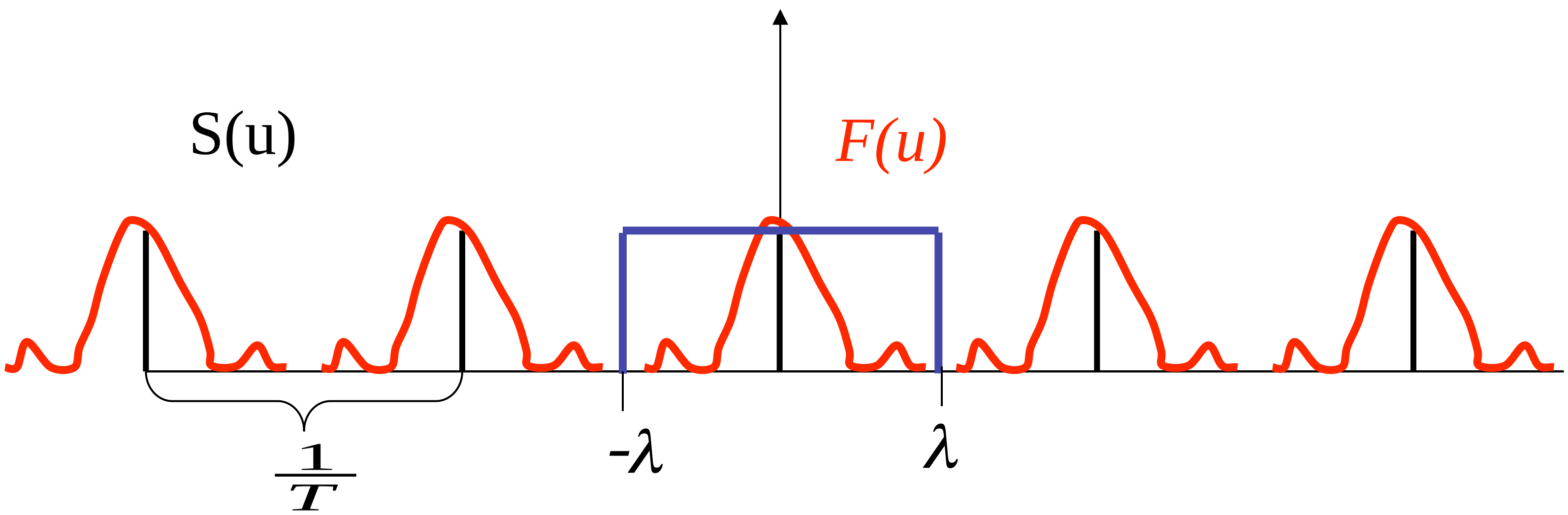
  λ is called the ***Nyquist limit.***

Sampling Theory

- Why is this?
- The Nyquist limit will ensure that the copies of $F(u)$ do not overlap in the frequency domain.
- I can completely reconstruct or determine $f(x)$ from $F(u)$ using the Inverse Fourier Transform.

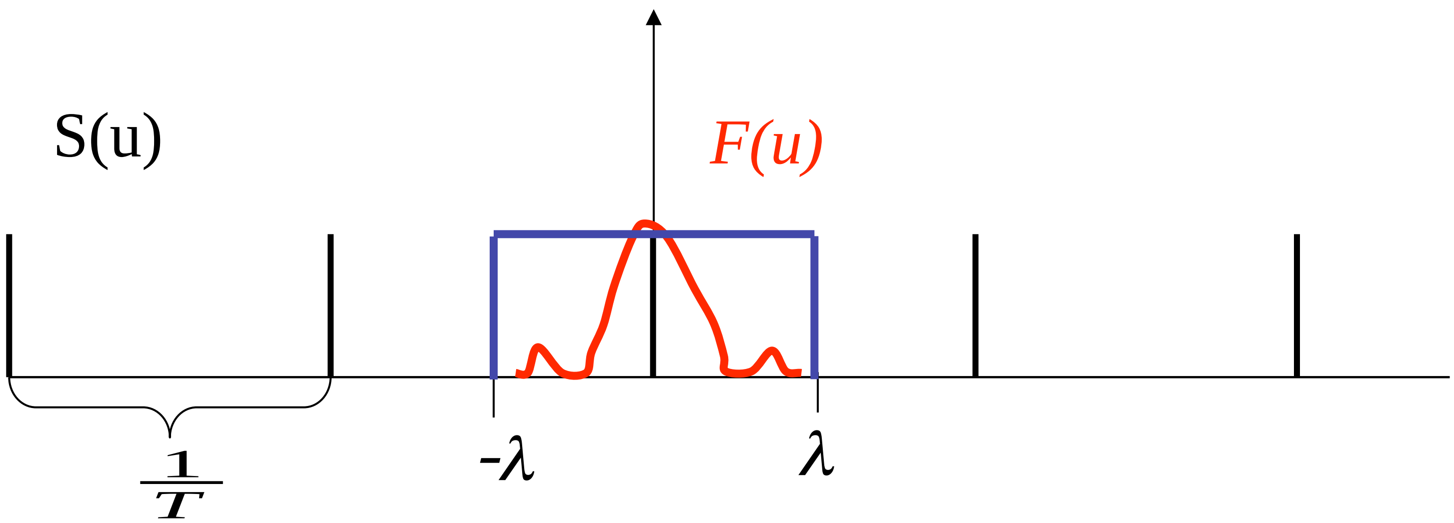
Sampling Theory

- In order to do this, I need to remove all of the shifted copies of *F(u)* first.
- This is done by simply multiplying *F(u)* by a box function of width 2λ.

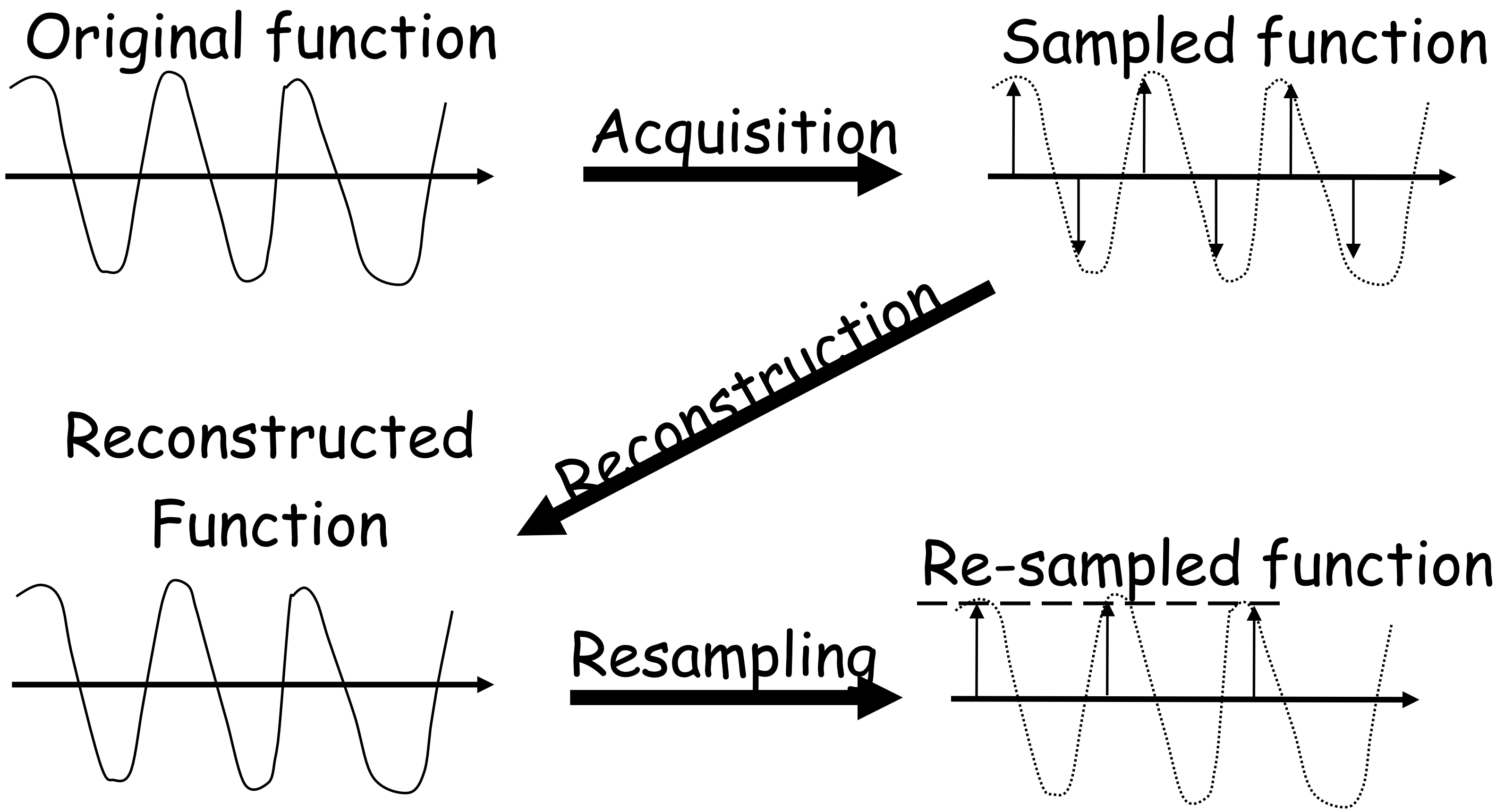


Sampling Theory

- In order to do this, I need to remove all of the shifted copies of *F(u)* first.
- This is done by simply multiplying *F(u)* by a box function of width 2λ.



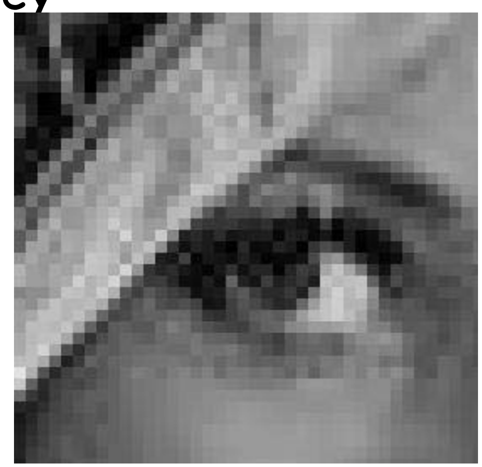
General Process



Interpolation (an example)

- Very important; regardless of algorithm
- expensive => done very often for one image
- Requirements for good reconstruction
  - performance
  - stability of the numerical algorithm
  - accuracy

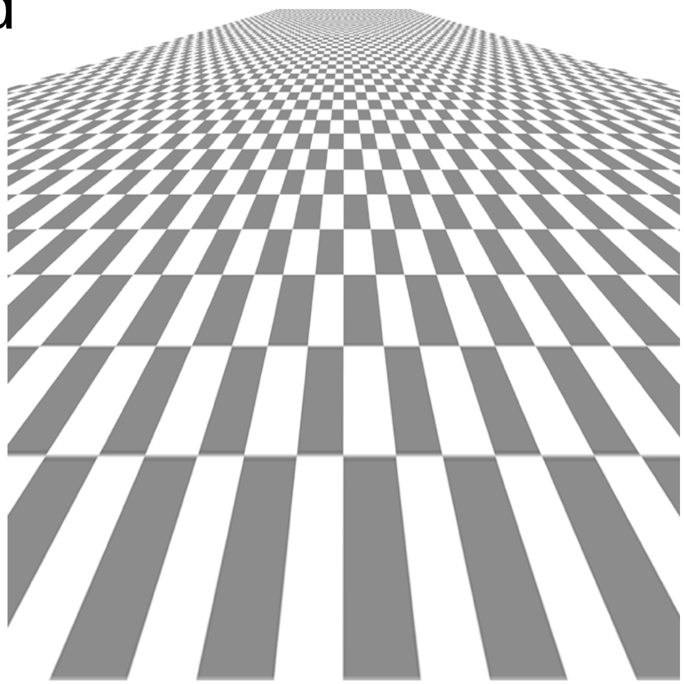
Nearest neighbor



Linear

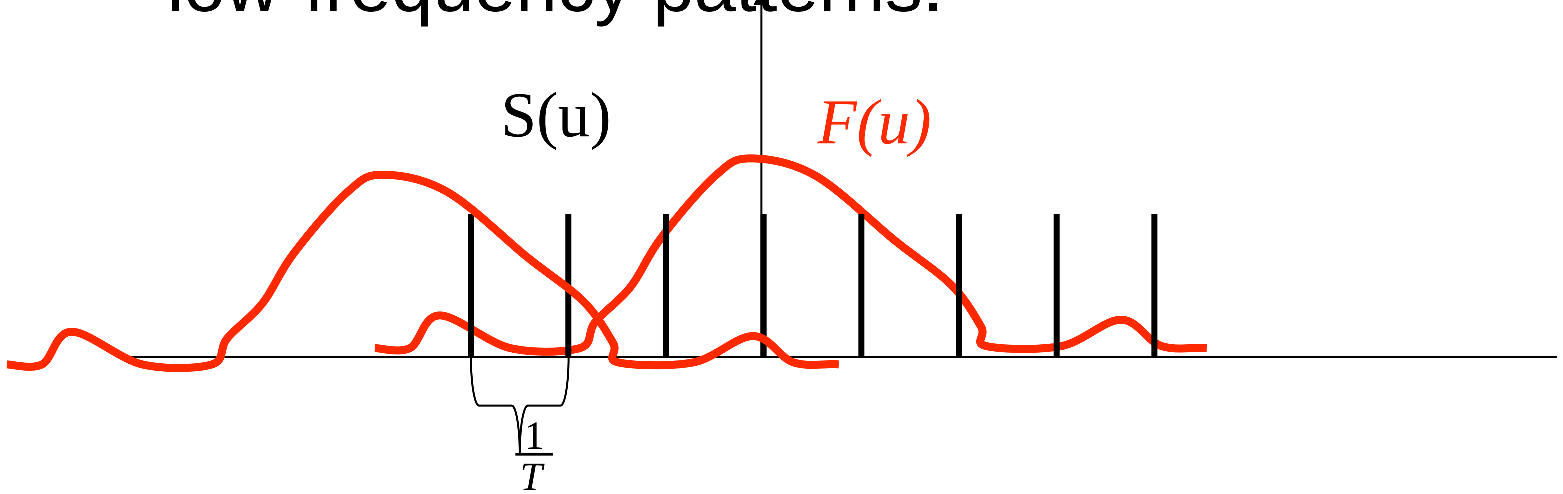
Sampling and Anti-aliasing

- The images were calculated as follows:
  - A 2Kx2K image was constructed and smoothly rotated into 3D.
  - For Uniform Sampling, it was downsampled to a 512x512 image.
  - Noise was added to the image, sharpened and then downsampled for the other one.
  - Both were converted to B&W.



Sampling and Anti-aliasing

- The problem:
  - The signal is not band-limited.
  - Uniform sampling can pick-up higher frequency patterns and represent them as low-frequency patterns.

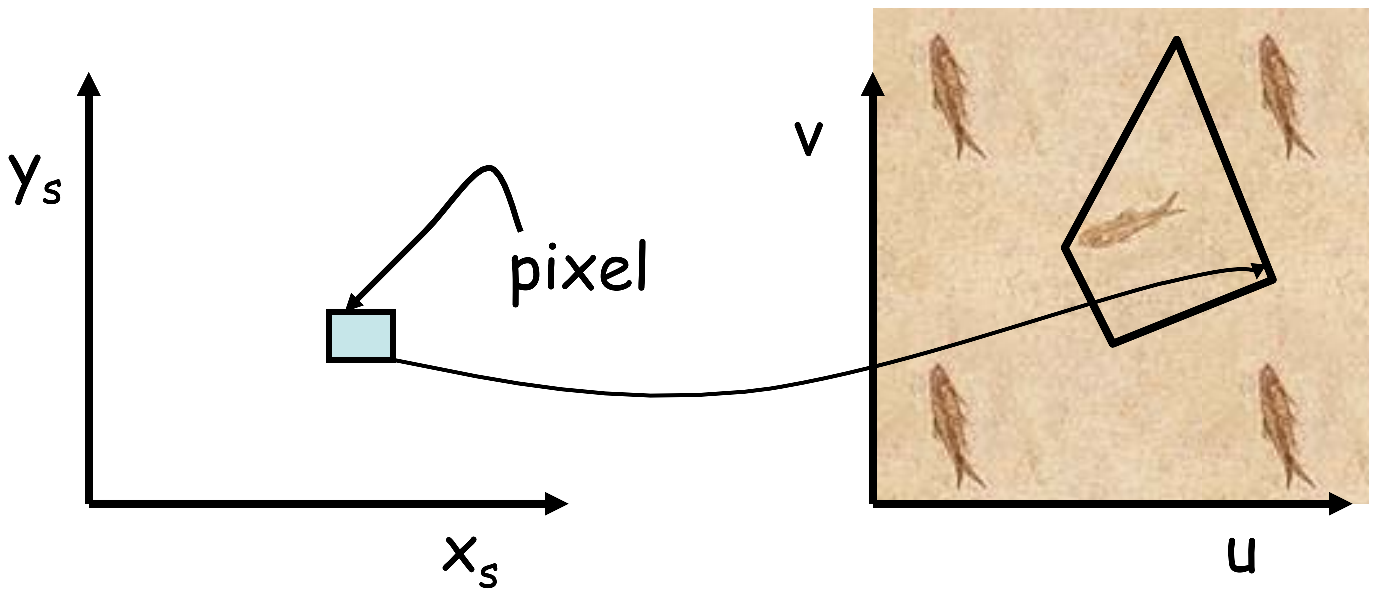


Quality considerations

- So far we just mapped one point
- results in bad aliasing (resampling problems)
- we really need to integrate over polygon
- super-sampling is not a very good solution (slow!)
- most popular (easiest) - mipmaps

Quality considerations

- Pixel area maps to "weird" (warped) shape in texture space



Quality considerations

- We need to:
  - Calculate (or approximate) the integral of the texture function under this area
  - Approximate:
    - Convolve with a wide filter around the center of this area
    - Calculate the integral for a similar (but simpler) area.

Quality considerations

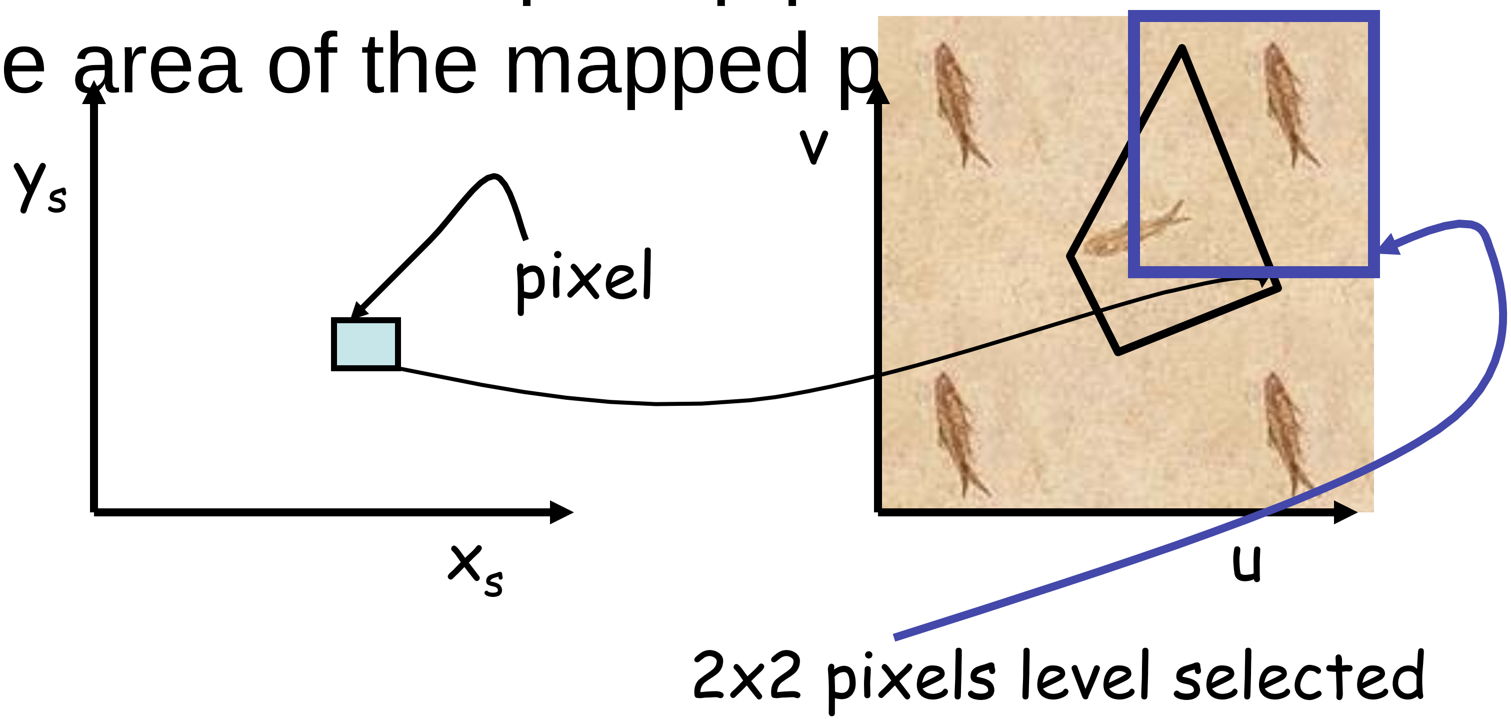
- the area is typically approxiated by a rectangular region (found to be good enough for most applications)
- filter is typically a box/averaging filter - other possibilities
- how can we pre-compute this?

Mip-maps

- An image-pyramid is built.

Mip-maps

- Find level of the mip-map where the area of each mip-map pixel is closest to the area of the mapped p



Mip-maps

- Pros
  - Easy to calculate:
    - Calculate pixels area in texture space
    - Determine mip-map level
    - Sample or interpolate to get color
- Cons
  - Area not very close – restricted to square shapes (64x64 is far away from 128x128).
  - Location of area is not very tight.

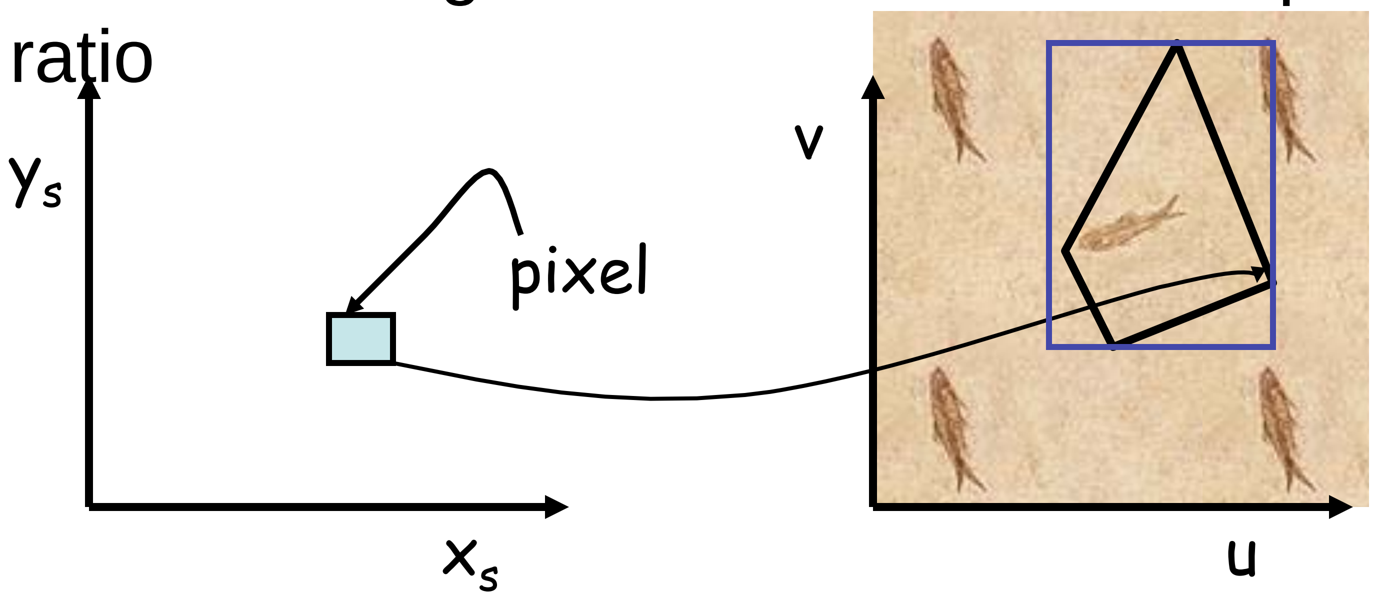
Summed Area Table (SAT)

- Use an axis aligned rectangle, rather than a square
- Precompute the sum of all texels to the left and below for each texel location
  - For texel (u,v), replace it with:

    sum (texels($i=0 \ldots u, j=0 \ldots v$))

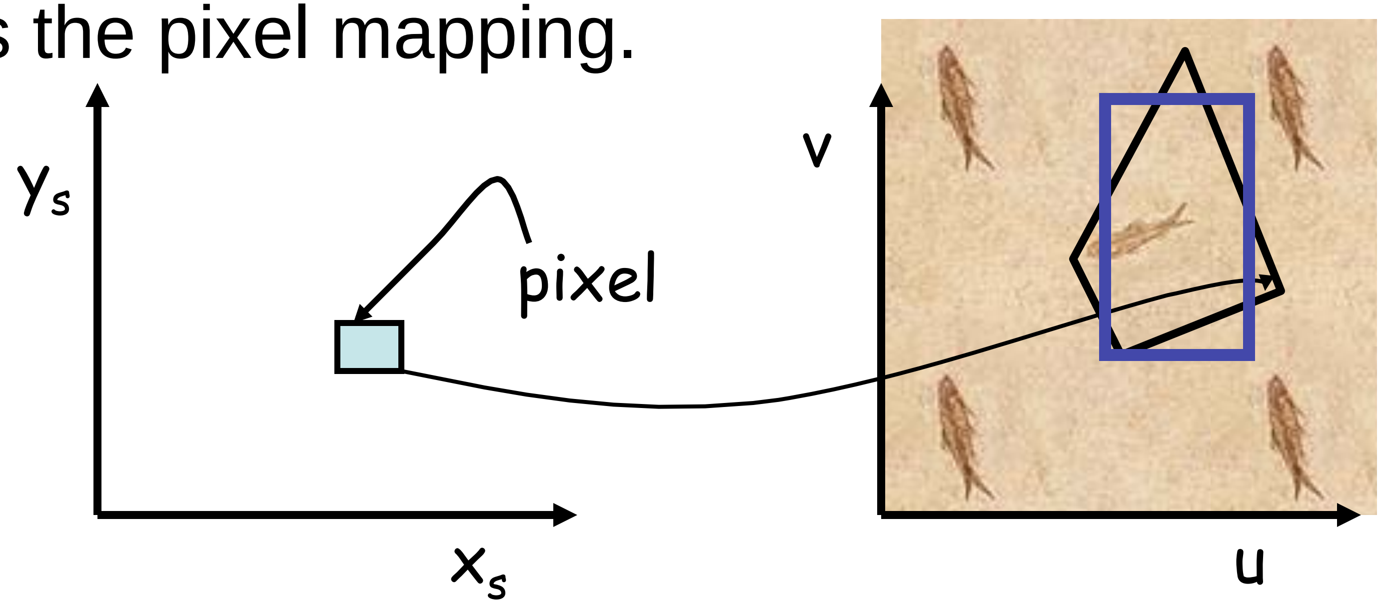
Summed Area Table (SAT)

- Determining the rectangle:
  - Find bounding box and calculate its aspect ratio



Summed Area Table (SAT)

- Determine the rectangle with the same aspect ratio as the bounding box and the same area as the pixel mapping.

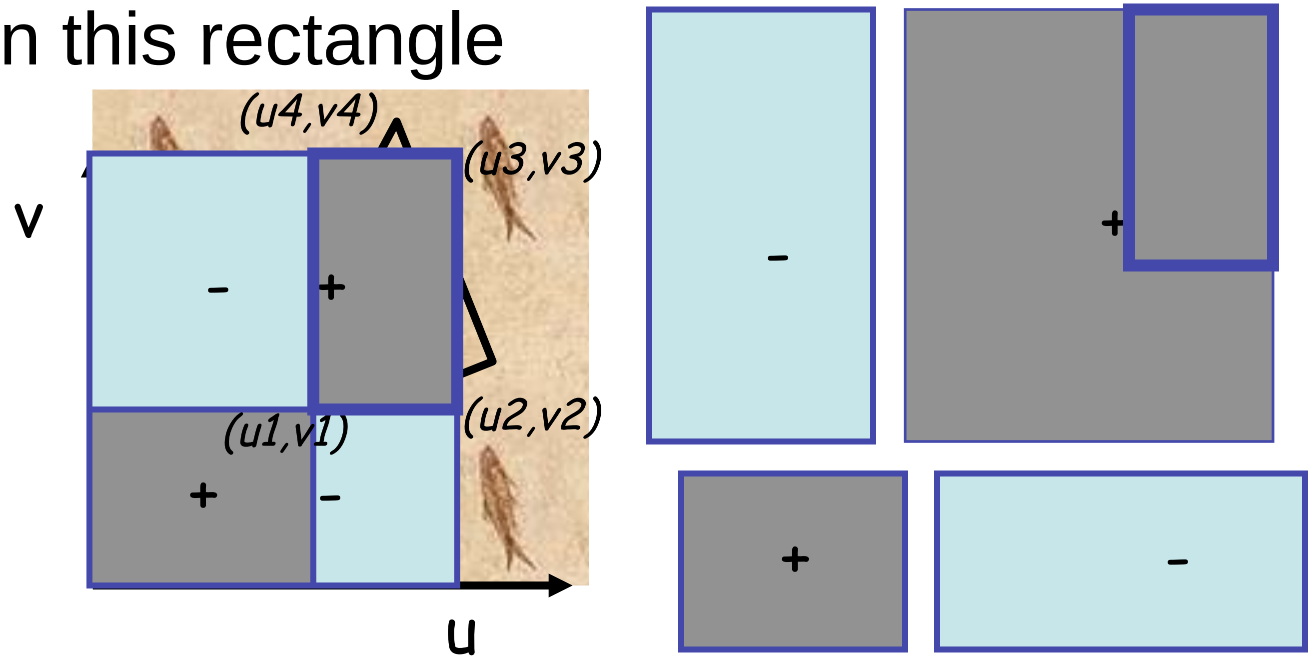


Summed Area Table (SAT)

- Center this rectangle around the bounding box center.
- Formula:
  - Area = aspect_ratio*x*x
  - Solve for x – the width of the rectangle
- Other derivations are also possible using the aspects of the diagonals, …

Summed Area Table (SAT)

- Calculating the color
  - We want the average of the texel colors within this rectangle



Summed Area Table (SAT)

- To get the average, we need to divide by the number of texels falling in the rectangle.
  - Color = SAT(u3,v3)-SAT(u4,v4)-SAT(u2,v2)+SAT(u1,v1)
  - Color = Color / ((u3-u1)*(v3-v1))
- This implies that the values for each texel may be very large:
  - For 8-bit colors, we could have a maximum SAT value of 255*nx*ny
  - 32-bit pixels would handle a 4kx4k texture with 8-bit values.
  - RGB images imply 12-bytes per pixel.

Summed Area Table (SAT)

- Pros
  - Still relatively simple
    - Calculate four corners of rectangle
    - 4 look-ups, 5 additions, 1 mult and 1 divide.
  - Better fit to area shape
  - Better overlap
- Cons
  - Large texel SAT values needed
  - Still not a perfect fit to the mapped pixel.