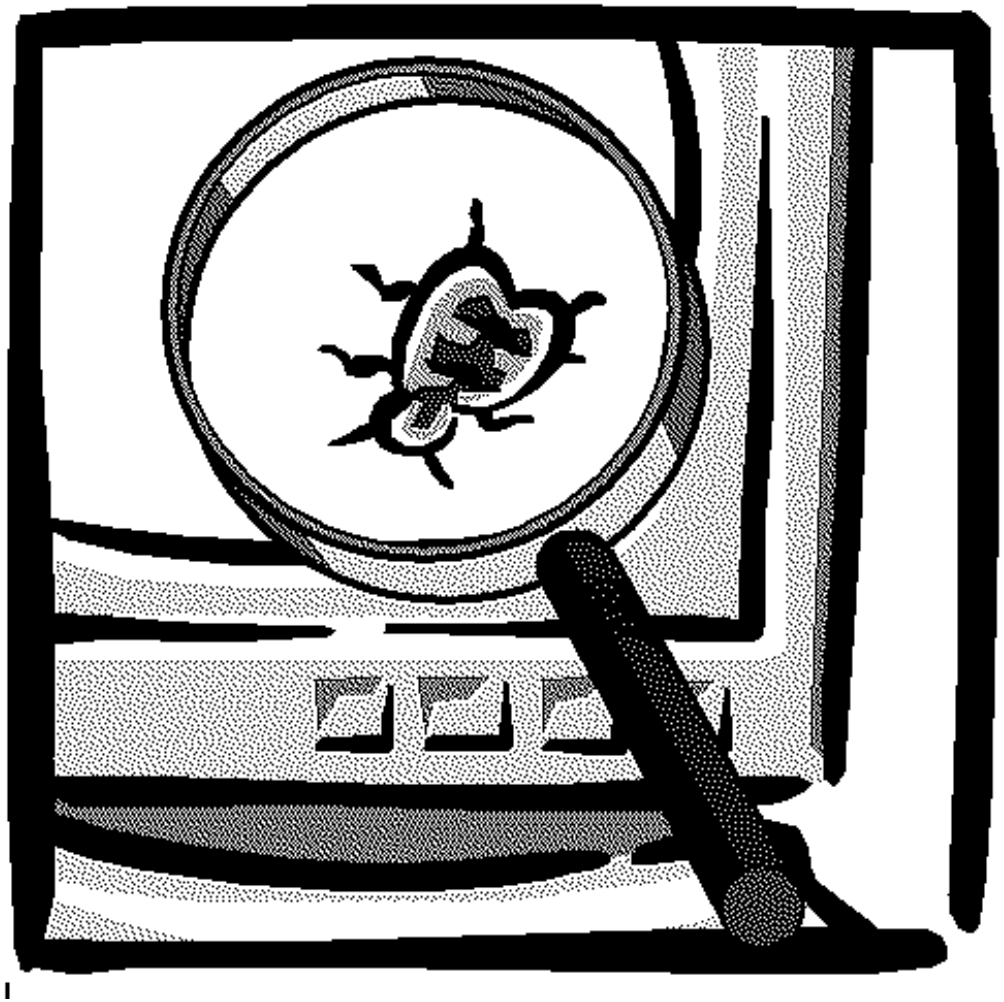
6.837 Lecture 5

Antialiasing and Resampling

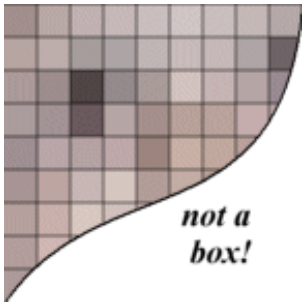
- What is a pixel?
- Signal Processing
- Image Resampling



What is a Pixel?

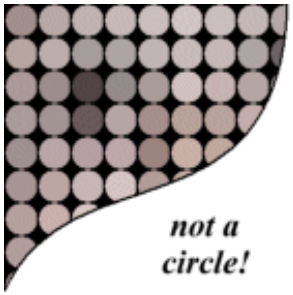
A pixel is not...

- a box
- a disk
- a teeny tiny little light



A pixel is a point...

- it has no dimension
- it occupies no area
- it cannot be seen
- it can have a coordinate



A pixel is *more* than just a point, it is a *sample*

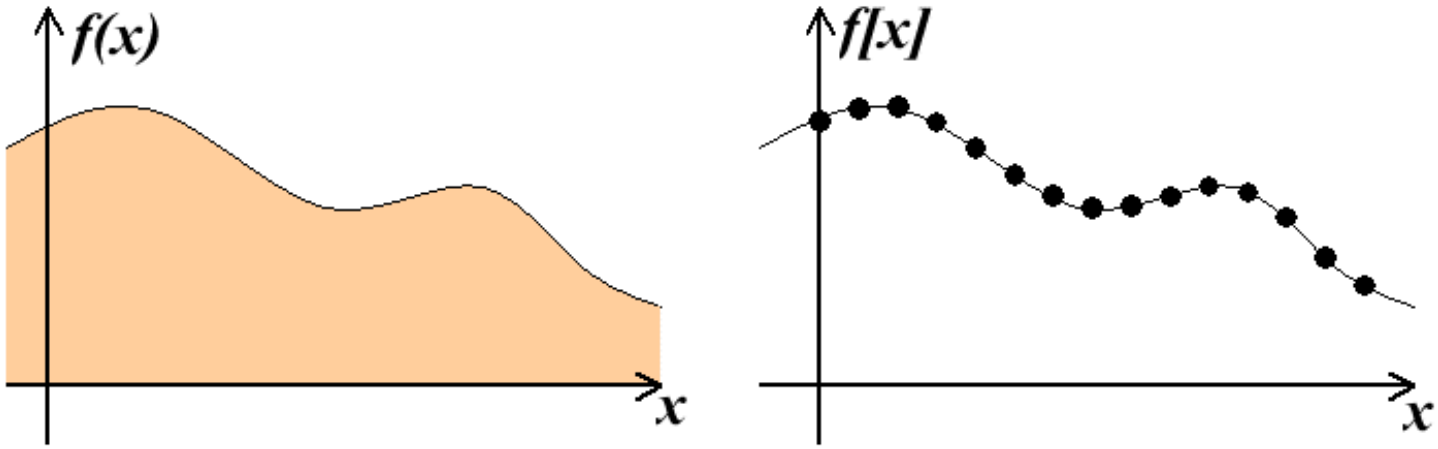
More on Samples

We think of most things in the real world as *continuous*,
yet, everything in a computer is discrete

The process of mapping a continuous *function* to a discrete one is called *sampling*

The process of mapping a continuous *variable* to a discrete one is called *quantization*

When we represent or render an image using a computer we must both sample and quantize

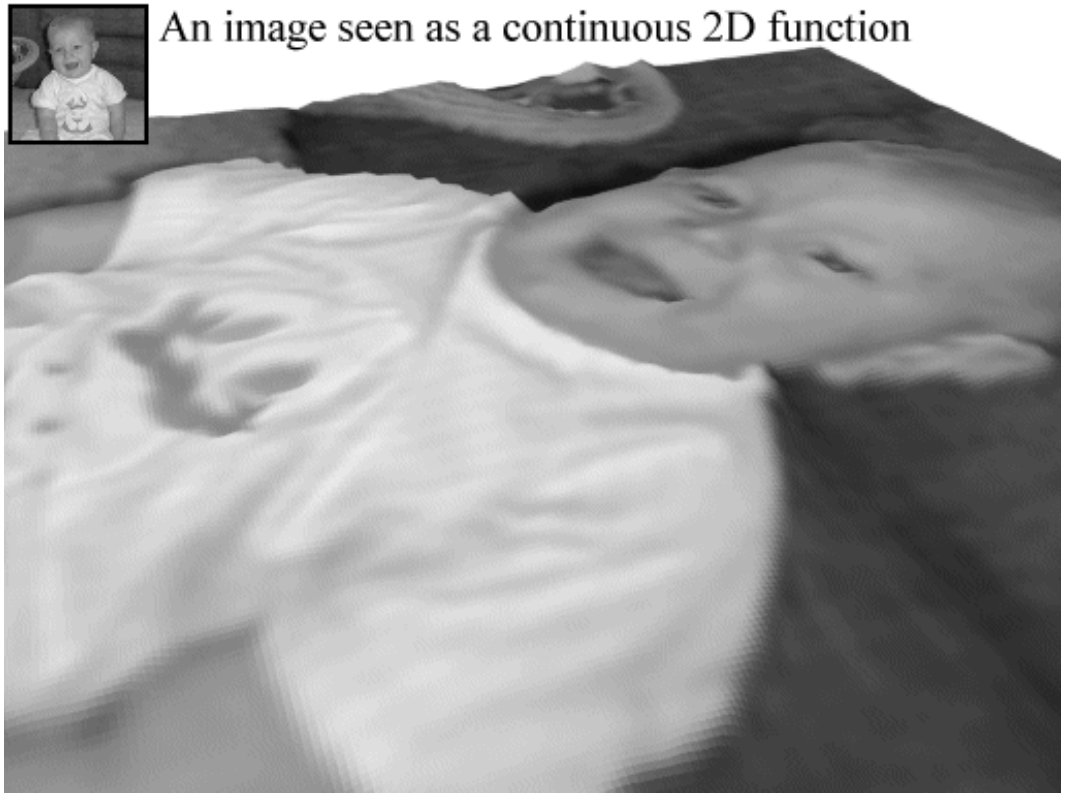


Picturing an Image as a 2D Function

An *ideal* image can be viewed as a function, $I(x, y)$, that gives an intensity for any given coordinate (x, y). We could plot this function as a height field. This plot would lowest at dark points in the image and highest at bright points.

Most of the functions that we encounter are *analytic* and, therefore, can be expressed as simple algebraic relations. An image, in general, cannot be represented with such a simple form. Instead, we represent images as tabulated functions.

So how do we fill this table?



Sampling Grid

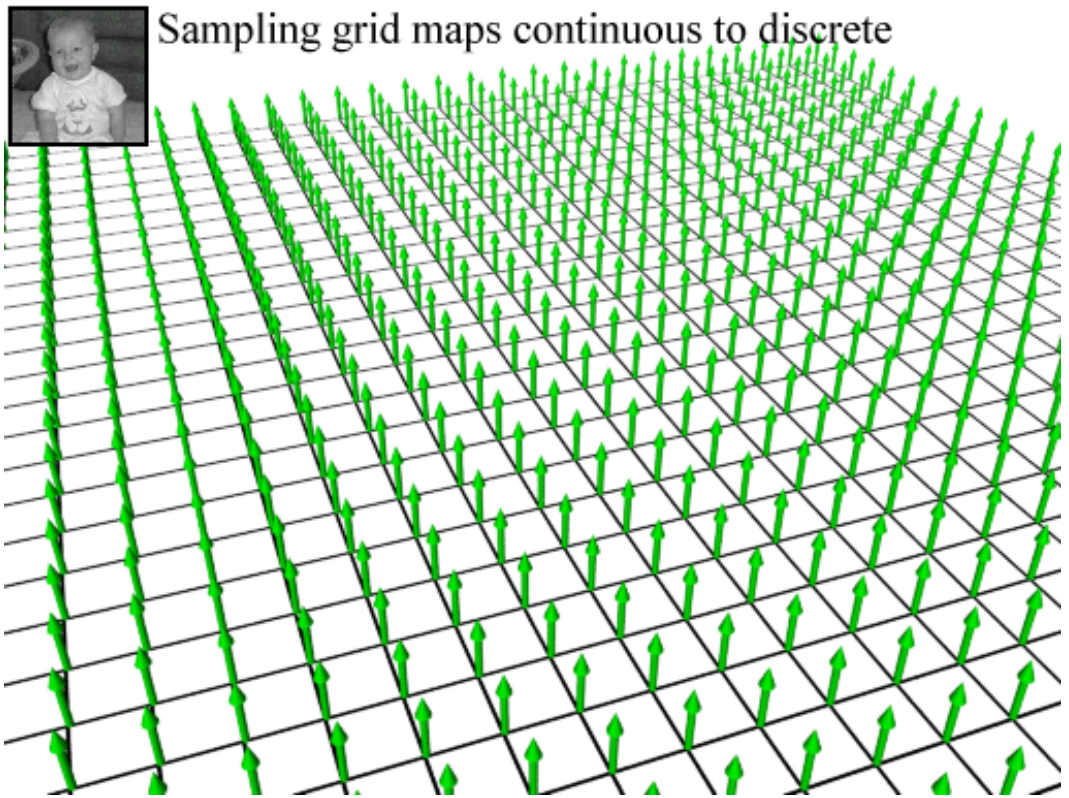
The most common (but not the only) way to generate the table values necessary to represent our function is to multiply the function by a *sampling grid*. A sampling grid is composed of periodically spaced Kronecker delta functions.

The definiton of the 2-D Kronecker delta is:

$$\delta(x,y) = \begin{cases} 1, & (x,y) = (0,0) \\ 0, & \text{otherwise} \end{cases}$$

And a 2-D sampling grid:

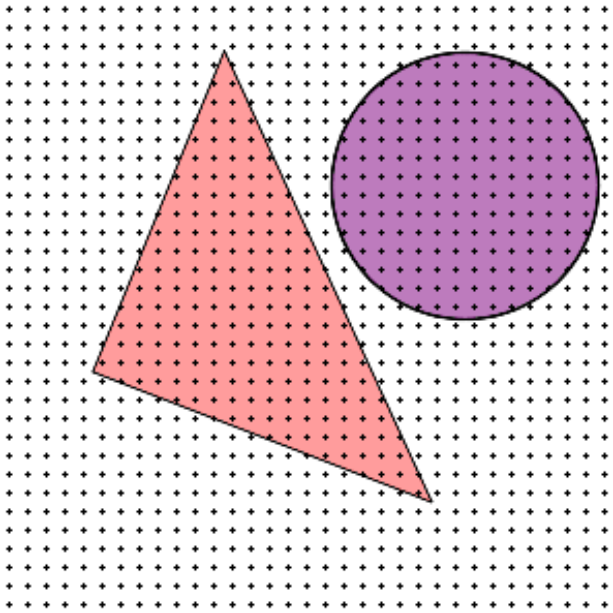
$$\sum_{j=0}^{h-1} \sum_{i=0}^{w-1} \delta(u-i, v-j)$$

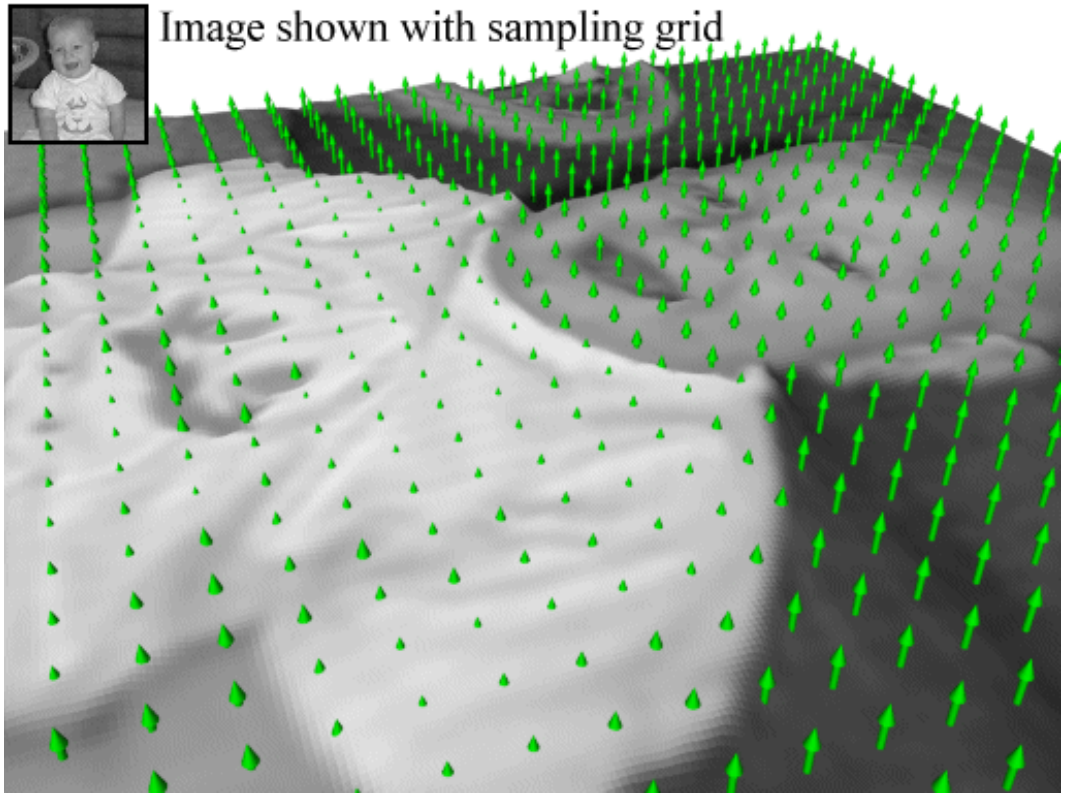


Sampling an Image

When a continuous image is multiplied by a sampling grid a discrete set of points are generated. These points are called samples. These samples are pixels. We store them in memory as arrays of numbers representing the intensity of the underlying function.

The same analysis can be applied to geometric objects:





The Big Question

How densely must we sample an image in order to capture its essence?

Since our sampling grid is *periodic* we can appeal to Fourier analysis for an answer. Fourier analysis states that all periodic signals can be represented as a summation of sinusiodal waves. Thus every image function that we understand as a height field in the *spatial domain*, has a dual representaion in the *frequency domain*.

We can transform signals from one domain to the other using the Fourier transform.

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\, e^{-i2\pi(ux+vy)}\, dxdy$$

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)\, e^{\,i2\pi(ux+vy)}\, dudv$$

Convolution

In order to simplify our analysis we will consider 1-D signals for the moment. It will be straightforward to extend the result to 2-D.

Some operations that are diffcult to compute in the spatial domain are simplified when the function is tranformed to its dual representation in the frequency domain. One such function is *convolution*.

Convolution describes how a system with impulse response, *h(x)*, reacts to a signal, *f(x)*.

$$f(x) * h(x) = \int_{-\infty}^{\infty} f(\lambda) h(x - \lambda) d\lambda$$

This integral evaluation is equivalent to multiplication in the frequency domain

$$f(x) * h(x) \rightarrow F(u) H(u)$$

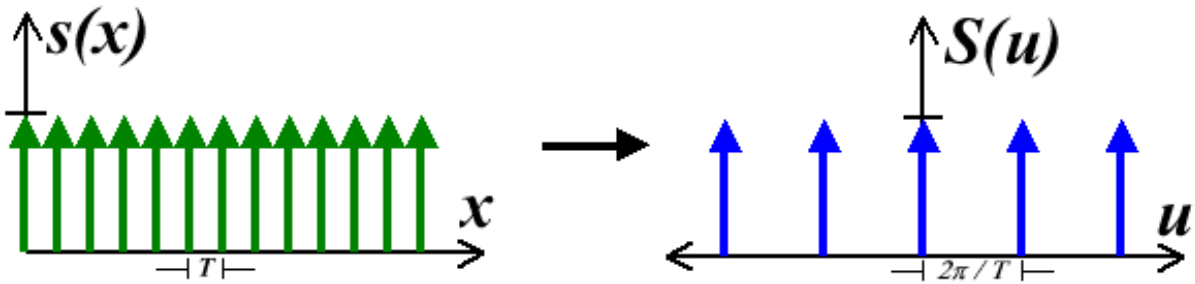
The converse is also true

$$F(u) * H(u) \rightarrow f(x) h(x)$$

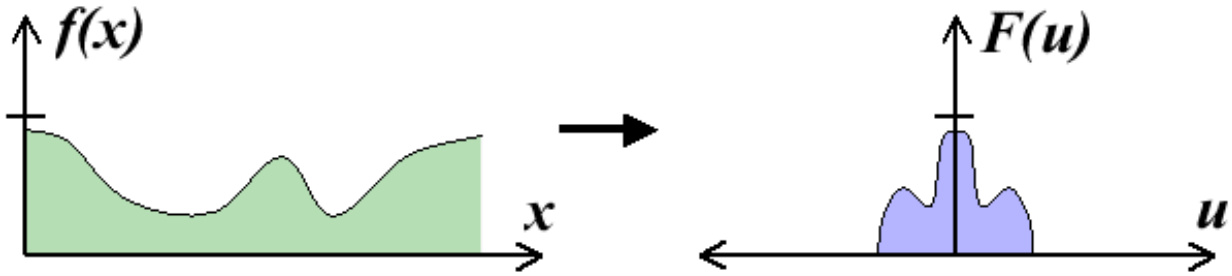
Sampling in the Frequency Domain

Image sampling was defined as multiplying a periodic series of delta functions by the continuous image. This is the same as convolution in the frequency domain.

Consider the sampling grid:

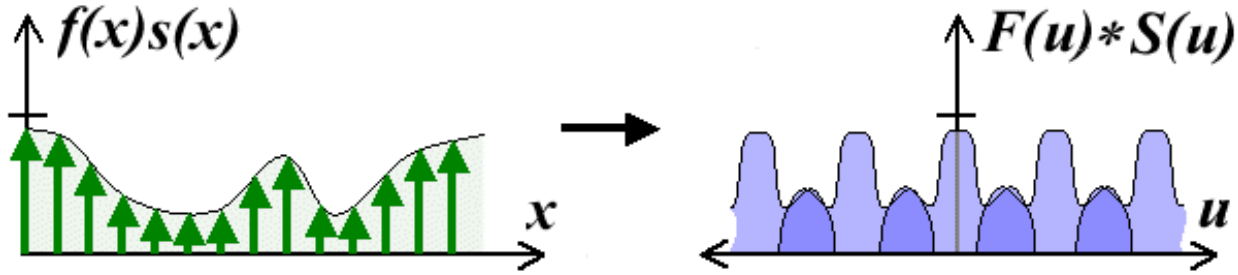


And the function being sampled

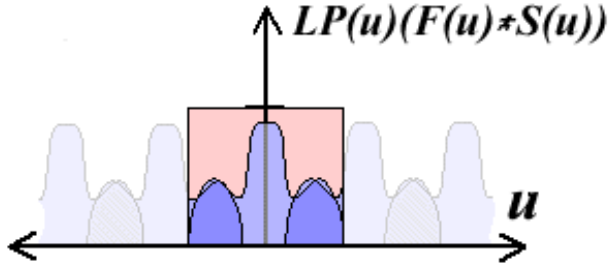


Reconstruction

This amounts to accumulating copies of the function's spectrum centered at the delta functions of the sampling grid.



Remember the goal of a sampled represention is to faithfully represent the underlying function. Ideally we would apply a *low-pass filter* to our sampled representation to *reconstruct* our original function. We will call this processing a *reconstruction filter*.

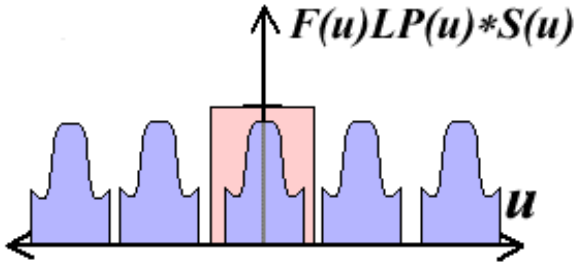


In this example we mixed together copies of our function (as seen in the darker overlap regions). In this case subsequent processing does not allow us to separate out a representative copy of our function.

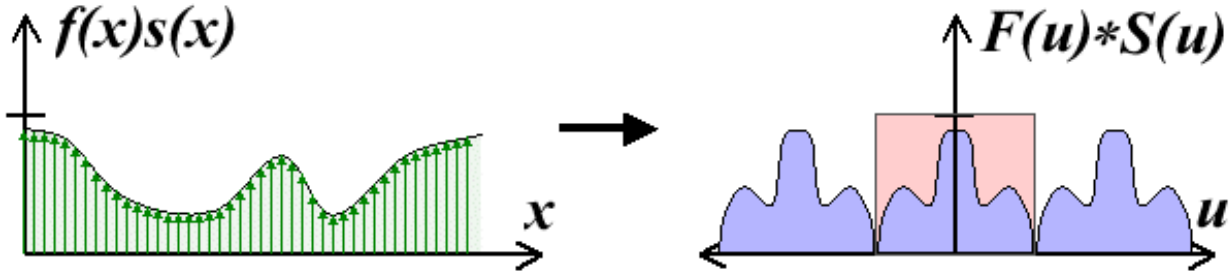
Aliasing

This mixing of spectrums is called *aliasing*. It has the effect of introducing high-frequencies into our original function. There are two ways of dealing with aliasing.

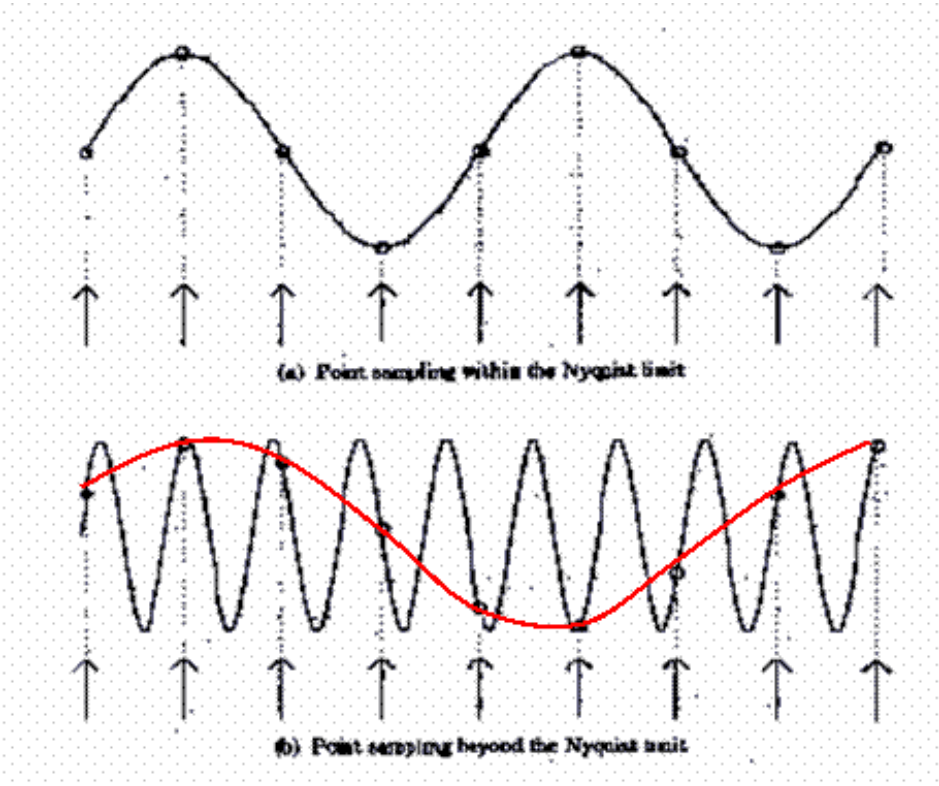
The first is to low pass filter our signal before we sample it.



The second is to increase the sampling frequency



Sampling Frequency



From Robert L. Cook, "Stochastic Sampling and Distributed Ray Tracing", An Introduction to Ray Tracing, Andrew Glassner, ed., Academic Press Limited, 1989

Sampling Theorem

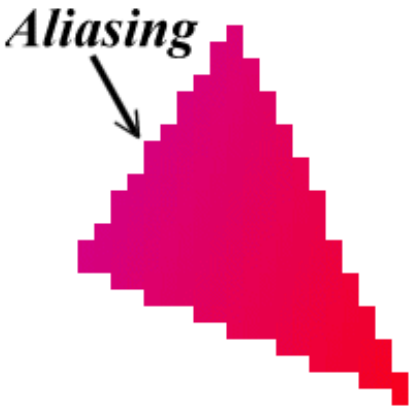
In order to have any hope of accurately reconstructing a function from a periodically sampled version of it, two conditions must be satisfied:

1. The function must be bandlimited.
2. The sampling frequency, f_s, must be at least twice the maximum frequency, f_{max}, of the function.

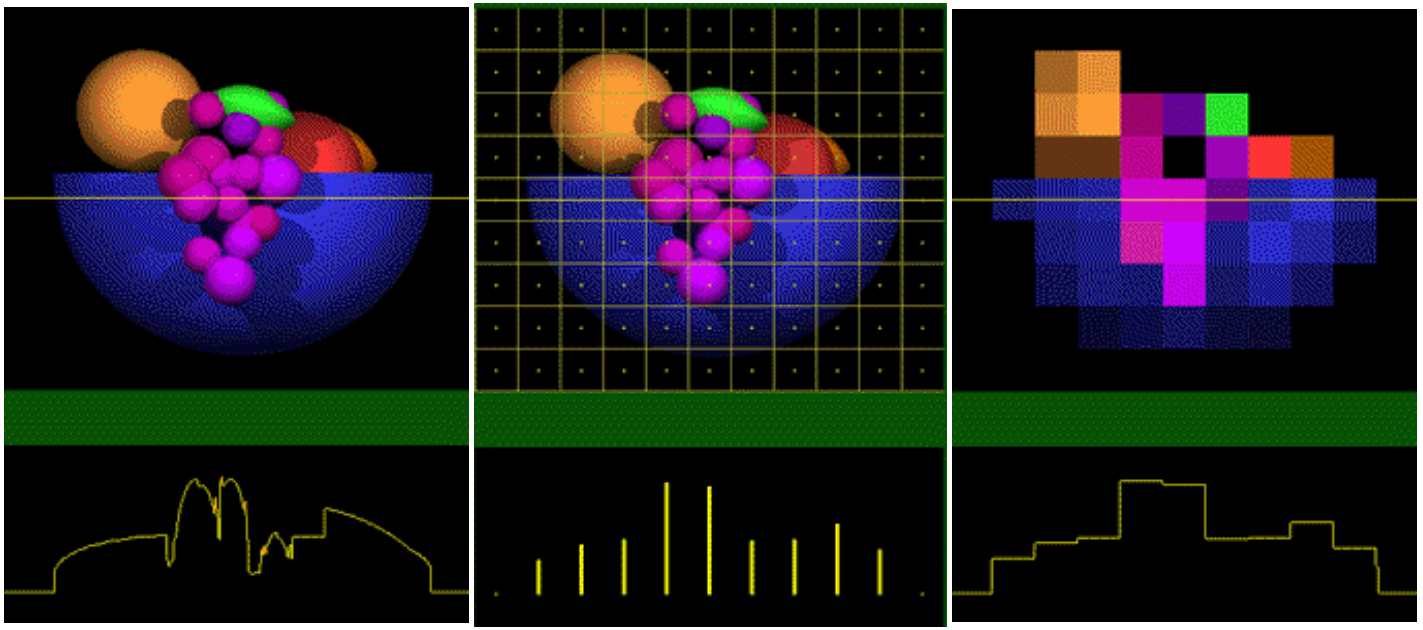
Satisfying these conditions will eliminate aliasing.

In practice:

- "Jaggies" are aliasing
- Both of the techniques discussed are used
  1. Super-sampling (more samples than pixels)
  2. Low-pass prefiltering (averaging of super-samples)



Sampling Example

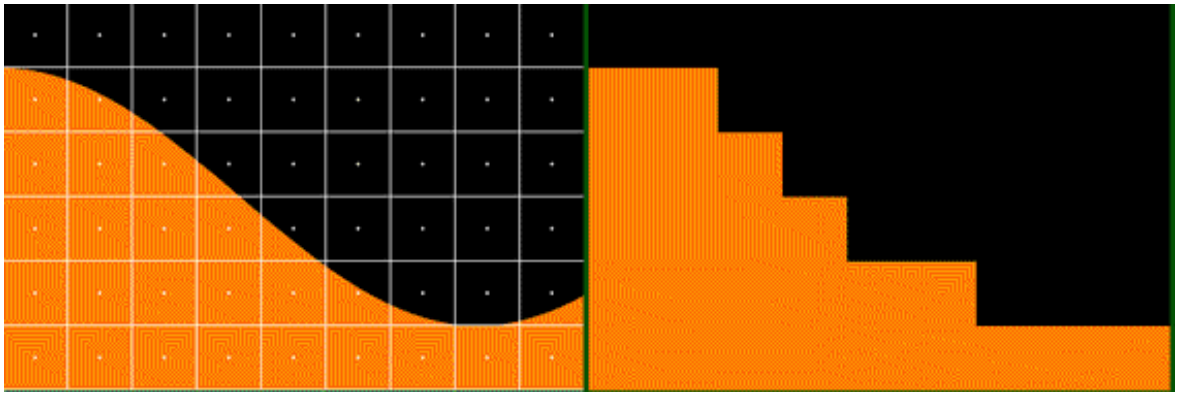


Original Image | Samples | Reconstruction

Source of images

Aliasing Effects

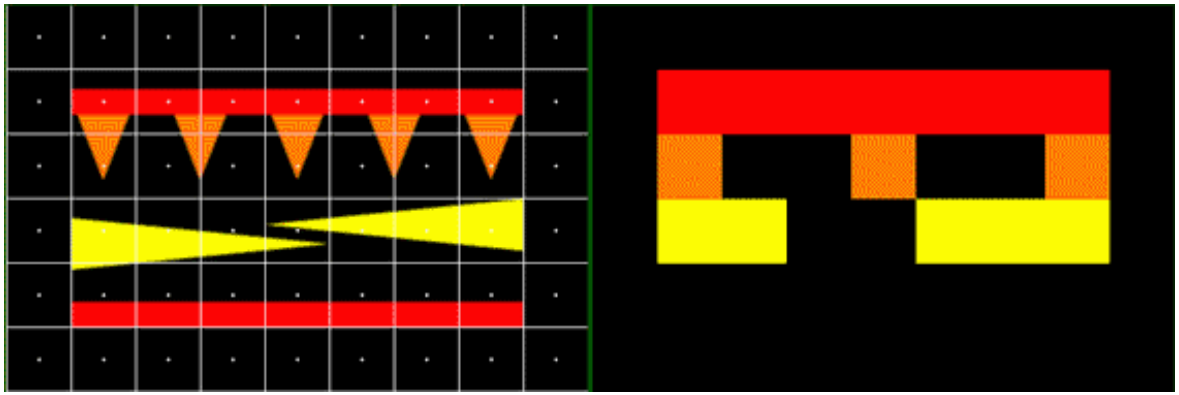
Jagged boundaries



Source of images

Aliasing Effects

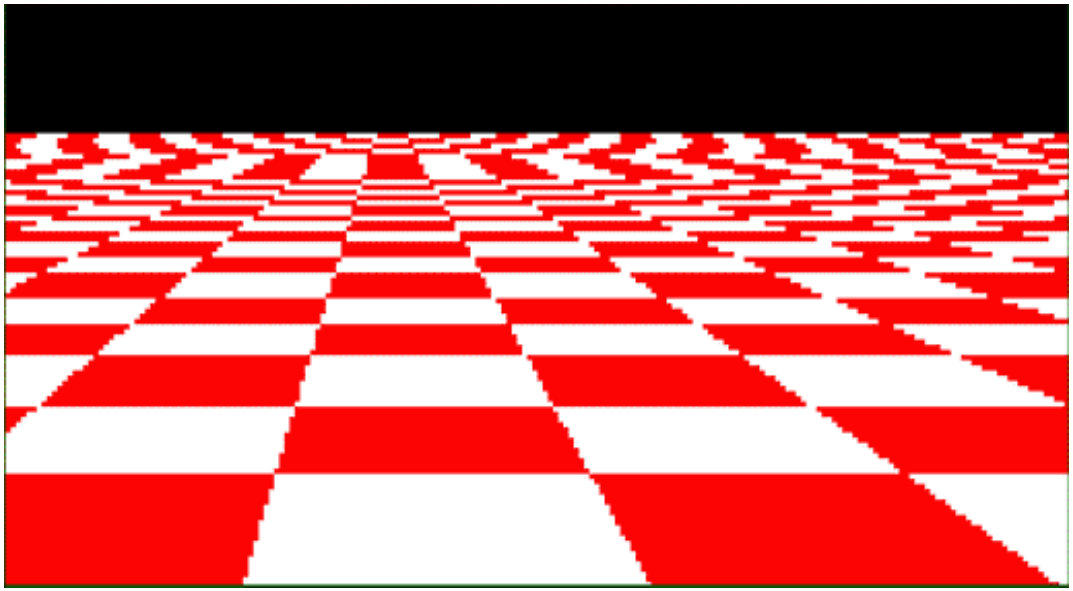
Improperly rendered detail



Source of images

Aliasing Effects

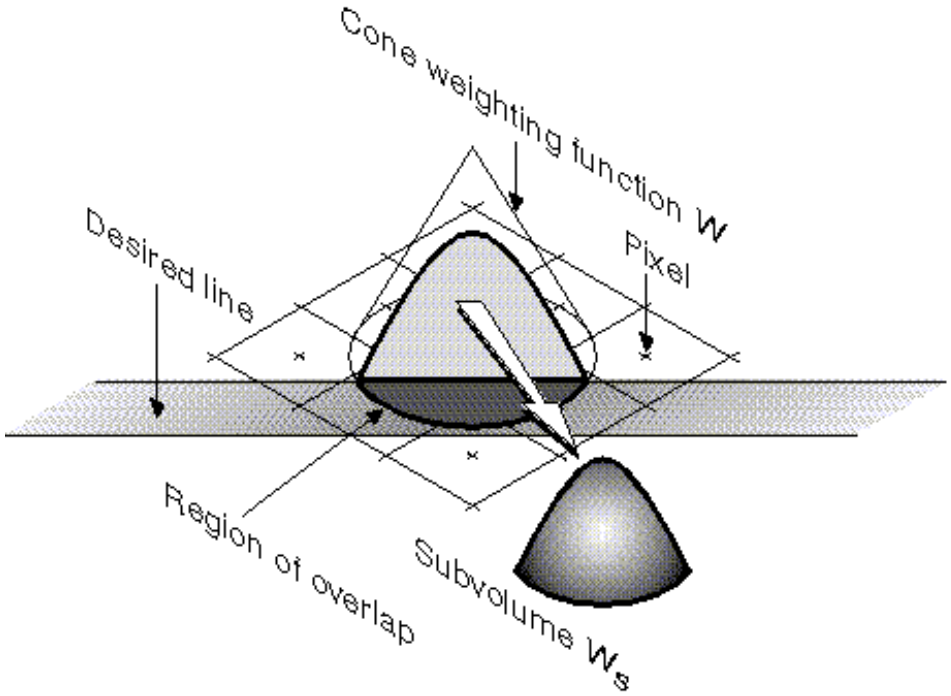
Texture errors



Source of images

Pre-Filtering

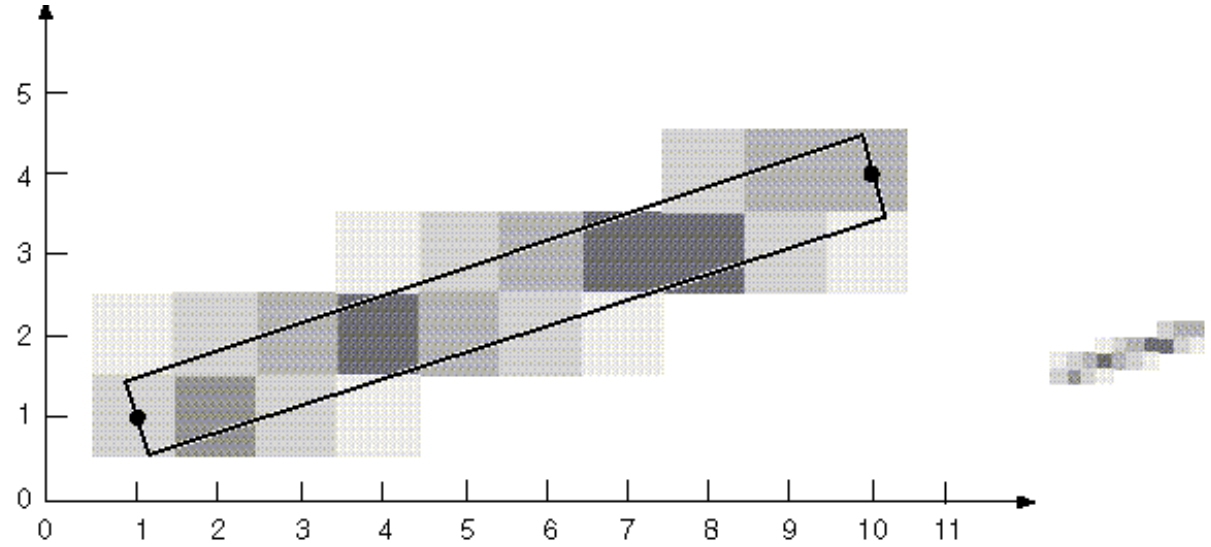
Prefiltering methods treat a pixel as an area, and compute pixel color based on the amount of overlap of the scene's objects with a pixel's area.



Source: Foley, VanDam, Feiner, Hughes - Computer Graphics, Second Edition, Addison Wesley

Anti-Aliased Lines

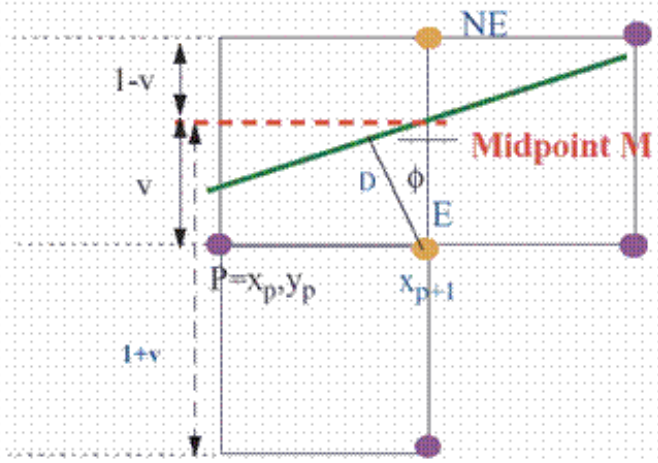
Multiple pixels with varying intensity are used to represent the line. Each pixel intensity set as a function of the distance of the pixel to the line.



Source: Foley, VanDam, Feiner, Hughes - Computer Graphics, Second Edition, Addison Wesley

Anti-Aliased Lines

Compute area of overlap incrementally as a function of Bressenham algorithm "error" (fraction).



$$D = v\cos\phi = \frac{vdx}{\sqrt{dx^2+dy^2}}$$

Can compute D incrementally in Bressenham style (f = fraction).

$$D_E = \frac{f+dx}{\sqrt{dx^2+dy^2}}$$

$$D_{NE} = \frac{f-dx}{\sqrt{dx^2+dy^2}}$$

Source: Foley, VanDam, Feiner, Hughes - Computer Graphics, Second Edition, Addison Wesley

AntiAliased line algorithm (uses floating point)

```
public void lineAA(int x0, int y0, int x1, int y1) {
    int pix = color.getRGB();
    int dy = y1 - y0;
    int dx = x1 - x0;
    int stepx, stepy;
    double invD = ((dx==0)&&(dy==0)) ? 0.0 : 1.0/(2.0*Math.sqrt(dx*dx+dy*dy));
    if (dy < 0) { dy = -dy;  stepy = -1; } else { stepy = 1; }
    if (dx < 0) { dx = -dx;  stepx = -1; } else { stepx = 1; }
    dy <<= 1;                                                  // dy is now 2*dy
    dx <<= 1;                                                  // dx is now 2*dx
    if (dx > dy) {
      int fraction = dy - (dx >> 1);
      int v2dx = 0;
      double dd, invD2dx = dx * invD;
      setPixelAA(x0, y0, 0);
      setPixelAA(x0, y0 + stepy, invD2dx);
      setPixelAA(x0, y0 - stepy, invD2dx);
      while (x0 != x1) {
        v2dx = fraction;
            if (fraction >= 0) {
              y0 += stepy;
              v2dx -= dx;
              fraction -= dx;
              }
            x0 += stepx;
            v2dx += (dx >> 1);
            fraction += dy;
            dd = v2dx * invD;
            setPixelAA(x0, y0, dd);
            setPixelAA(x0, y0 + stepy, invD2dx - dd);
            setPixelAA(x0, y0 - stepy, invD2dx + dd);
            }
            else {
              ...}
}
```

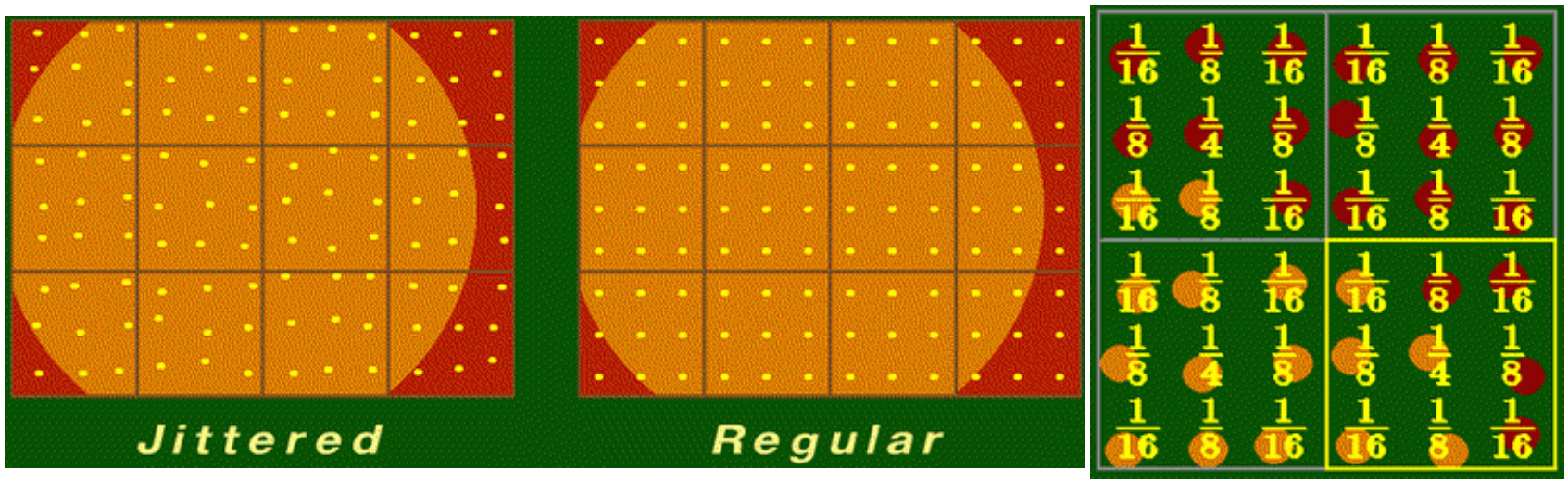
AntiAliased Line Demonstration

Draw a line by clicking and dragging on the pixel grid shown with the left mouse button. An *ideal* line is displayed until the left button is released. Upon release a discrete approximation of the line is drawn on the display grid using an anti-aliased line algorithm. An *ideal* line is then overlaid for comparison. A right click clears the screen.

Select a line type, then draw a line by clicking and dragging with the left mouse button. A gray line trace will follow the mouse. A final line will be drawn upon release. Compare anti-aliased lines with, for example, Bressenham lines.

Post-Filtering

The more popular approach to antialiasing. For each pixel displayed, a set of samples at "super-resolution" are combined.



Source of images

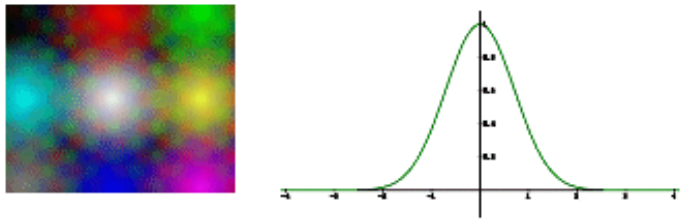
Reconstruction Revisited

Once we've satified the sampling theorem we can theoretically reconstruct the function. This requires low-pass filtering the samples that are used to represent the image.

Remember the samples that we store in memory are *points*; they have no area only positions. It is this *reconstruction filter* that gives a pixel area.

Question: If what I say is true how can we just stick pixel values in our frame buffer and see a picture? If they were just points (like I claim) we would see anything.

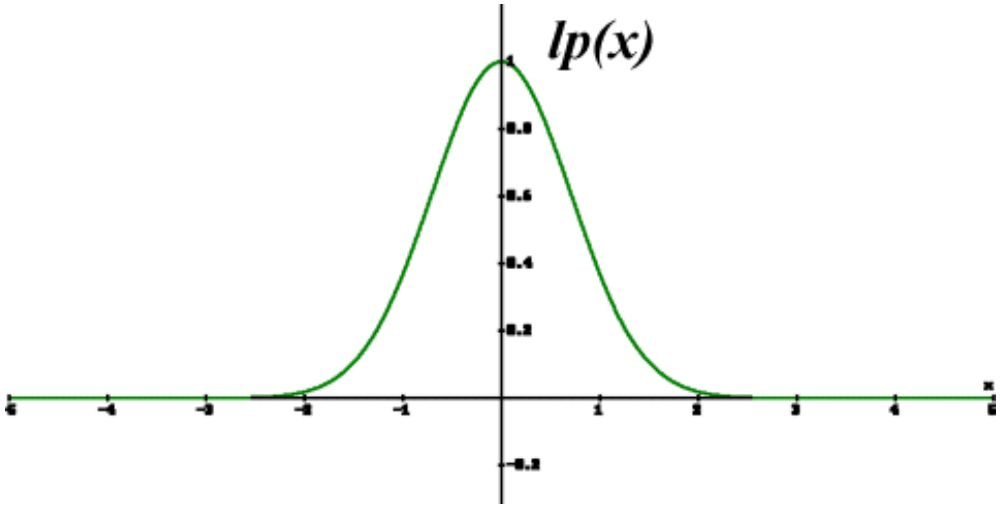
The fact is that a CRT has a built in reconstruction filter. Most real world devices do because they do not have an infinite frequency response.



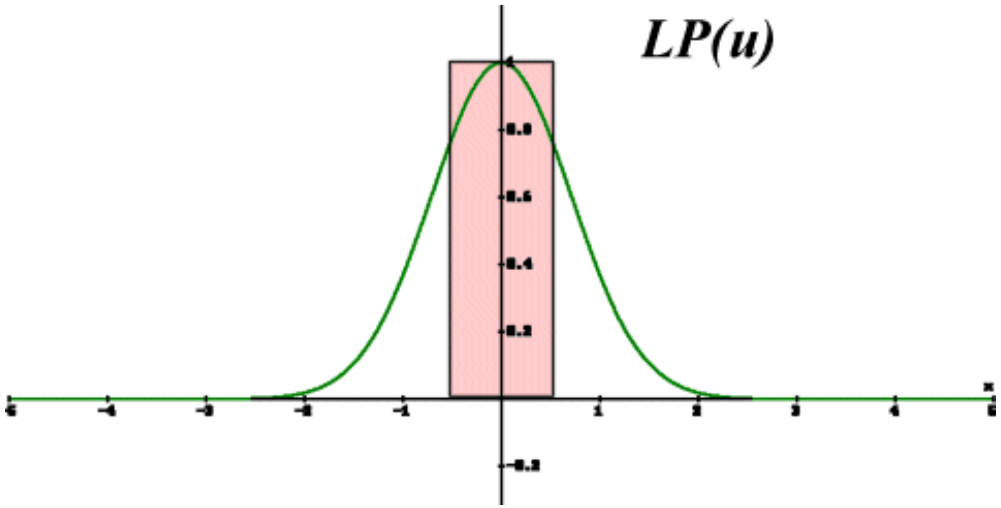
Each pixel illuminates a spot on the screen whose intensity fall off is well approximated by a gaussian.

Gaussian Reconstruction

So how does a Gaussian do as a reconstruction filter? In the spatial domain:

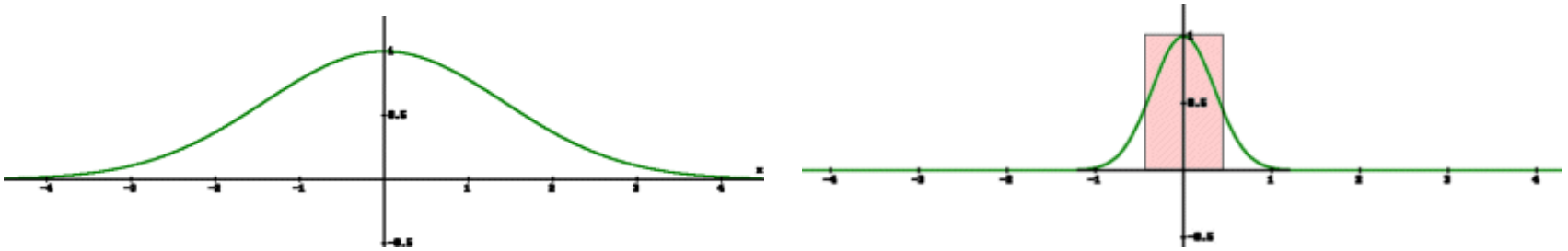


In the frequency domain:



Gaussian Verdict

- Wider in spatial domain corresponds to narrower in the frequency domain



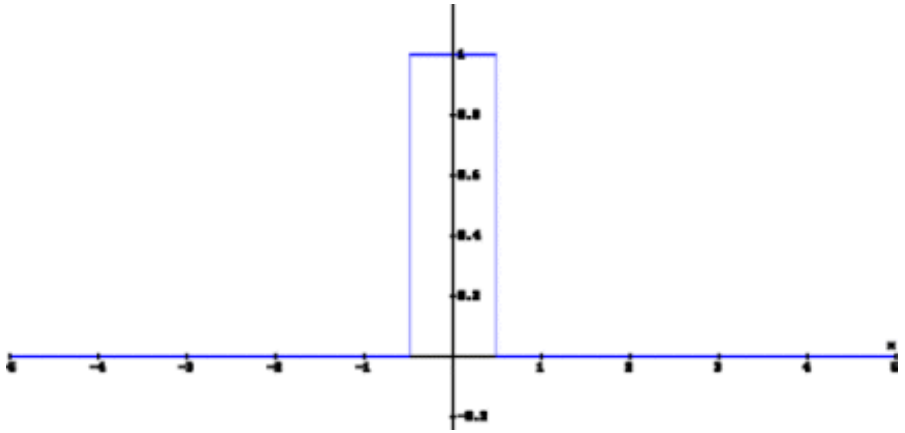
- We are forced to live with either *low-frequency attenuation* or *high-frequency leakage*
- Live with blur
- Infinite extent
- CRT resolution
- Only reconstruction kernel that is both *linearly separable* and *radially symmetric*

For a CRT images we basically have to live with the reconstruction filters that we are given. However, when we write an magnification program we are in control of the magnification.

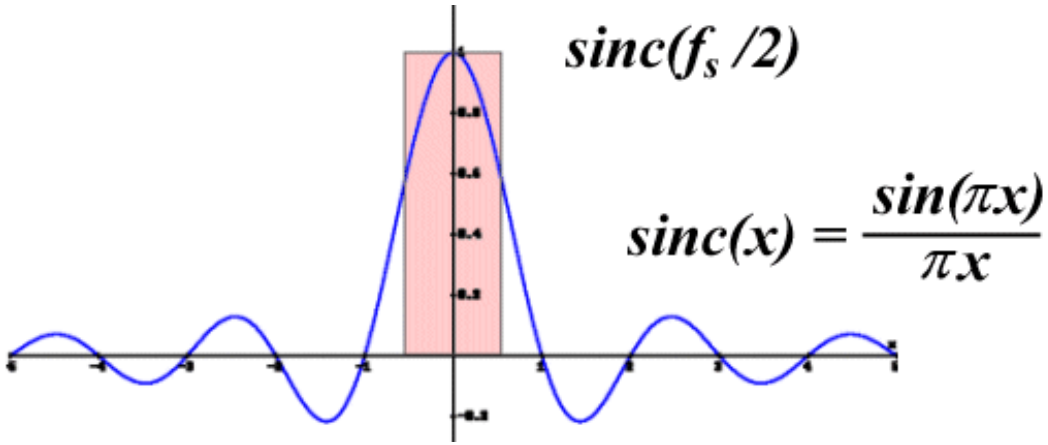
Nearest-Neighbor Reconstruction

One possible reconstruction filter has the shape of a simple box in the spatial domain.
Note: This is not the shape of the pixel but rather the shape of the reconstruction filter!

In the spatial domain:



In the frequency domain:



Nearest-Neighbor Example

Blockiness is due to high-frequency leakage in the reconstruction filter

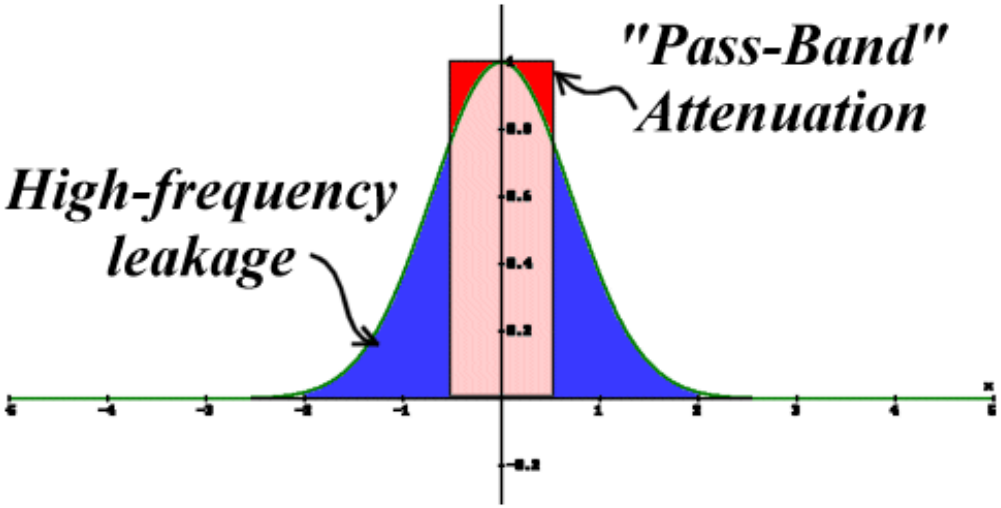
Even properly sampled images are rendered as blocky

"Post-sampling aliasing"

This is a rather poor reconstruction

Problems with Reconstruction Filters

Many of the "blocky" artifacts visible in resampled images are not due to classical aliasing, but instead caused by poor reconstruction filters.



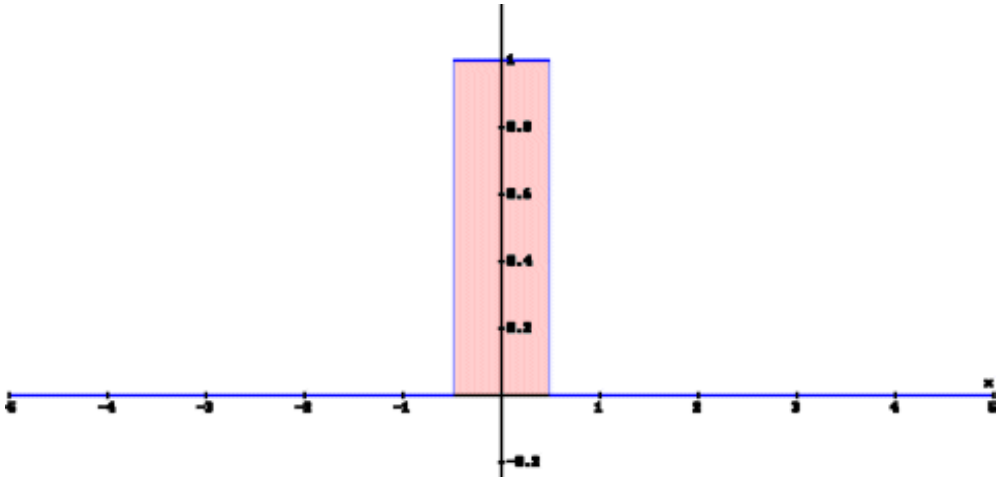
Excessive pass-band attenuation results in images that are noticeably blurry.

Excessive high-frequency leakage can lead to ***ringing*** or it can accentuate the sampling grid which is sometimes called ***anisotropy***.

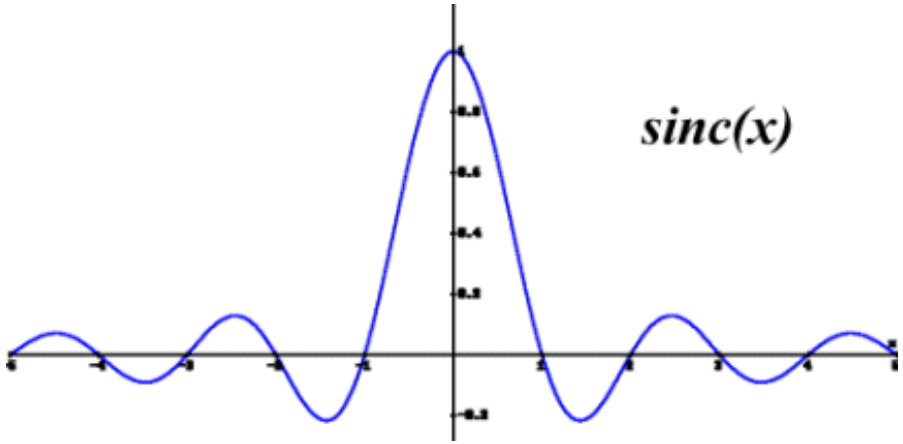
Is there an Ideal Reconstruction Filter?

Theoretically yes, practically no

We can work backwards from our desired frequency domain response:

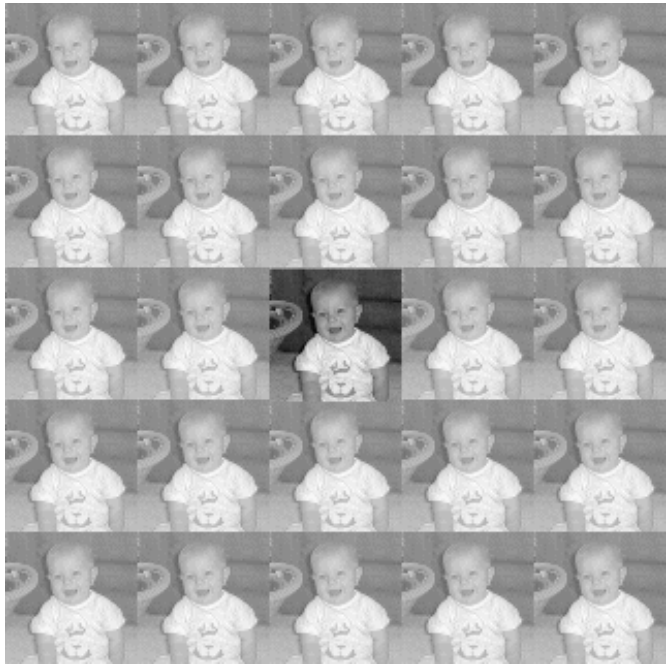


To compute the ideal reconstruction filter in the spatial domain:



Problems with a Sinc Reconstruction Filter

- It has infinite spatial extent.
  Every pixel contributes to every point that we interpolate.
- Hard to compute in the spatial domain.
- Assumes the samples of the source image tile an infinite plane

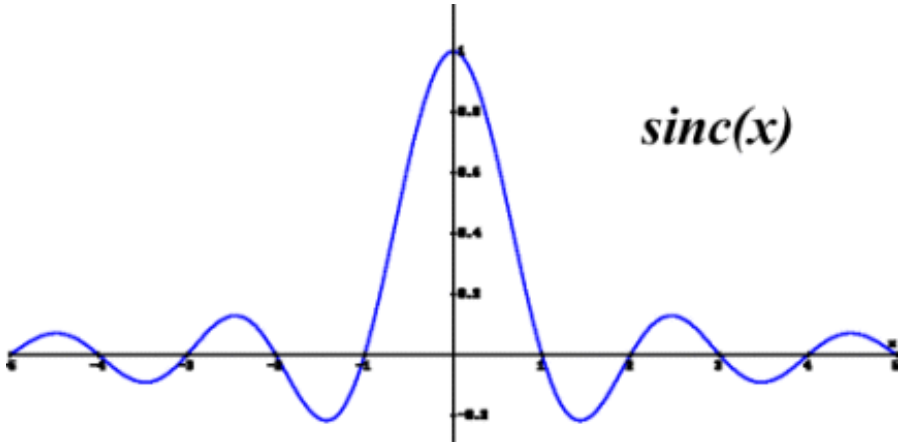


- Every interpolated point requires an infinite summation
- Truncating this summation leads to Gibb's phenomenon

Lessons from the Sinc Function

Notice the values of the sinc function at the sample points (integer values).

It has a value of one at the point being reconstructed and a zero everywhere else.

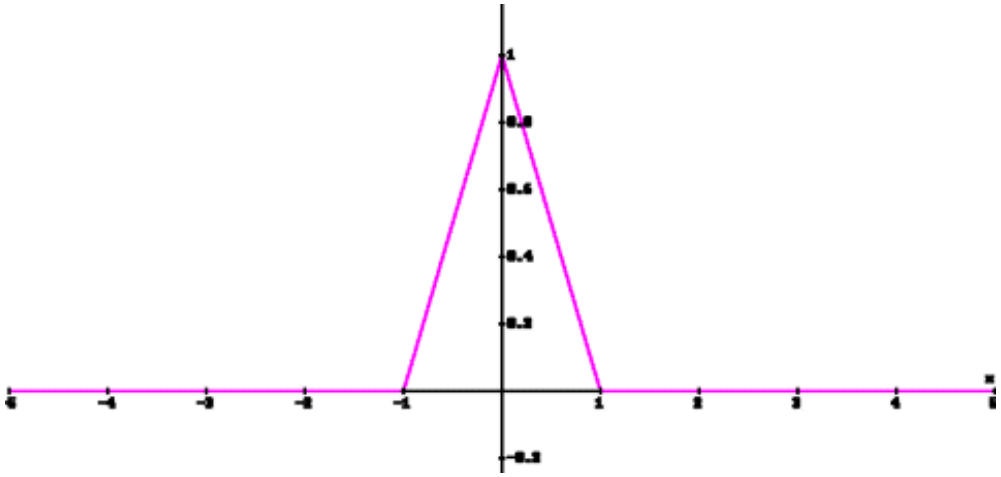


This means that a sinc will exactly reconstruct the sample values at the sample points.

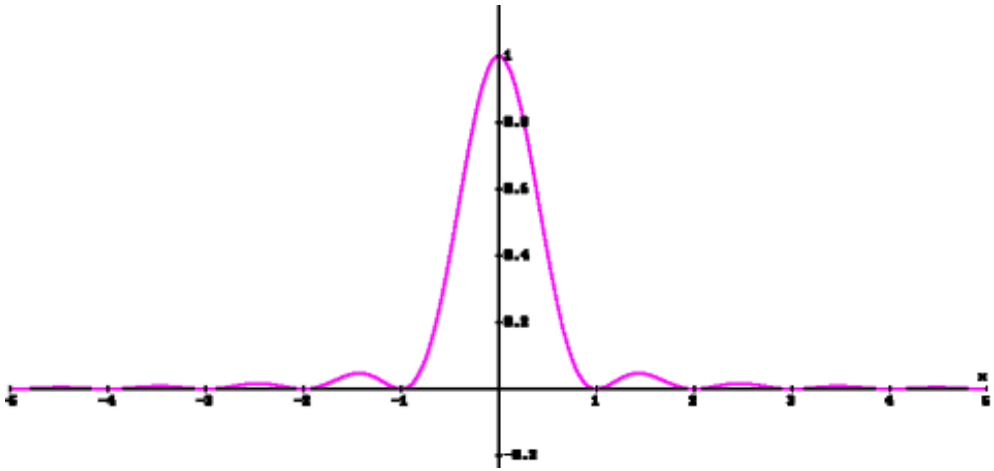
Also, notice that it has even symmetry (like a cosine function) and a slope of 0 at the sample point.

Approximations with Finite Extent

What sort of reconstruction we can do if we limit ourselves to considering only the pixels immediately surrounding a given point? One possibility is bilinear reconstruction:



Which has the frequency response:



Bilinear Example

Sometimes called a
tent filter

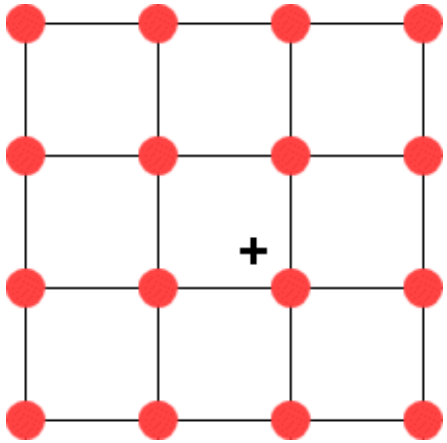
High-frequencies are significantly attenuated

Easy to implement
(just linearly interpolate between samples)

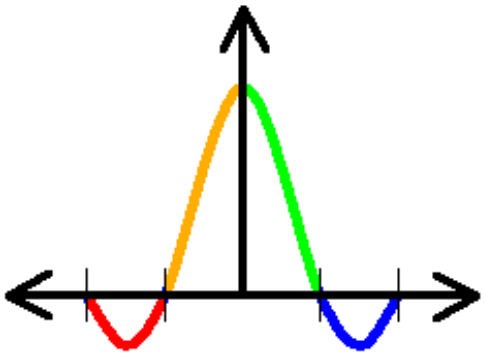
Still can produce artifacts

Larger Extents

We can use our knowledge about splines to design filters with arbitrary extents. Here we will consider a 4 by 4 neighborhood surrouding the desired pixel.



We can constrain our design usin the knowledge that we have gleaned from the ideal **sinc** reconstruction filter. First, we divide our spline into four piecewise sections sections.



Spline Constraints

The piecewise cubic spline should satisfy the following:

1. The function must be symmetric:

$$k(x) = k(-x)$$

   We can reformulate our general cubic such that it is symmetric as follows:

$$k(x) = A|x|^3 + Bx^2 + C|x| + D$$

   This also allows us to consider only 2 segments instead of 4.

2. $\mathbf{C_0}$ and $\mathbf{C_1}$ continuity between spline segments

$$k_1(1) = k_2(1), \quad k_2(2) = 0$$

$$k'_1(0) = 0, \quad k'_1(1) = k'_2(1), \quad k'_2(2) = 0$$

3. In order to assure that a uniform array of samples is reconstructed as a uniform continuous function it should satisfy the following convex relationship:

$$\sum_{n=-\infty}^{\infty} k(x-n) = 1$$

Bicubic Derivation

Starting with the equations of our two spline segments and their derivitives we can construct the two desired spline segments.

The resulting family of reconstruction filters is:

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

These bicubic reconstruction splines are commonly called

Mitchell's 2-parameter spline family

Notes:

k(0)=(3 - B)/3, k(1)=B/6
thus, if *B = 0* then k(0) = 1, k(1) = 0 like *sinc(x)*

Bicubic Example

Mitchell's 2-parameter spline family

Very low leakage

Includes:

- Cardinal splines
  (B=0, C=-a)

- Catmull-Rom Spline
  (B=0, C=0.5)

- Cubic B-spline
  (B=1, C=0)

Next Time


ACTIVE-EDGES
Edge-Equations
Plane-Equations
Half-Spaces
Triangles
(psst... What IS he talking about?)