Interpolation & Resampling

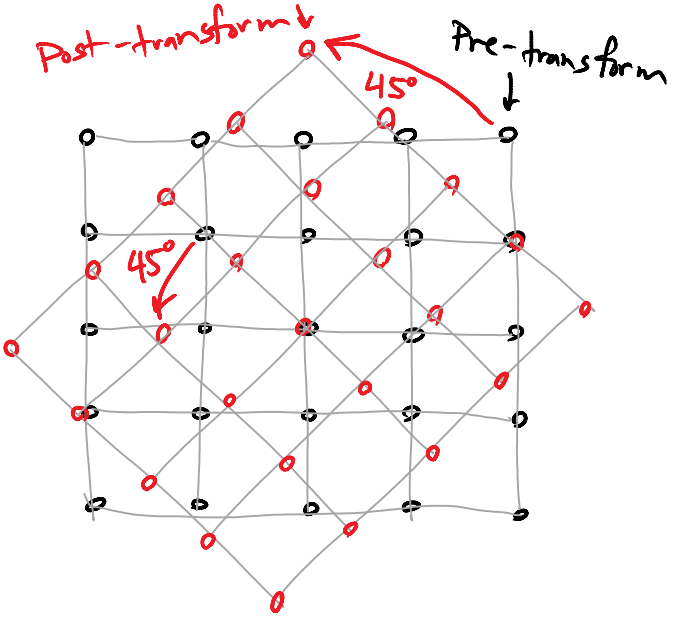
L10

Goal: To learn when & how to resample an image. We also want to have a way to understand imperfections in our resampling.

Resampling

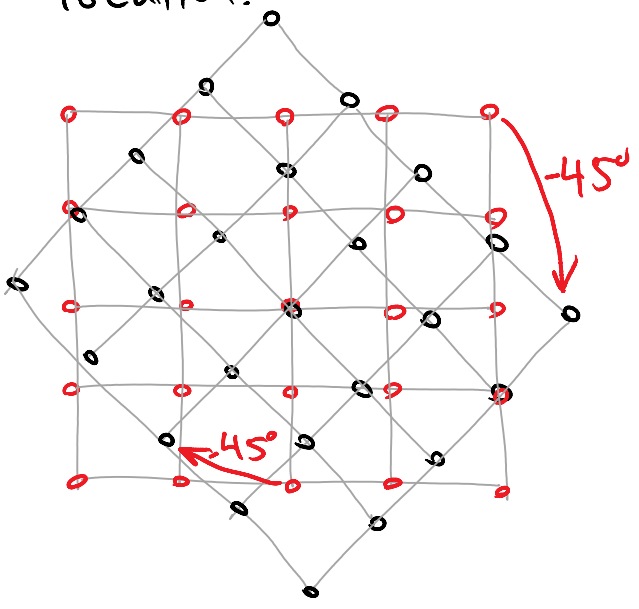
Suppose we want to rotate an image by 45°.

Method 1: For each pre-transform pixel, rotate it 45° to its post-transform location.



Problem: Pixels tend to land between post-transform pixels.

Method 2: For each post-transform pixel, counter-rotate it 45° to its pre-transform location.



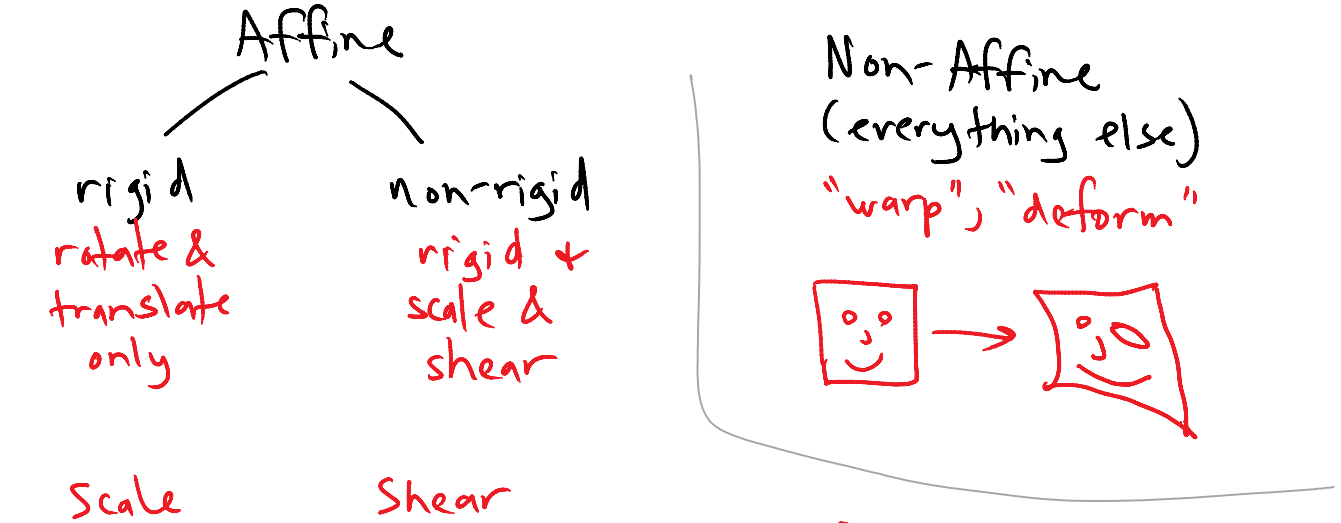
Problem: Pixels tend to come from locations between pre-transform pixels.

The vast majority use Method 2. The reason becomes more obvious when you consider non-affine transformations.

Pseudocode: to rotate an image f by 45°

```
for each (r,c) in g (post-transf. image)
    rotate (r,c) by -45° → (r',c')
    sample f at (r',c') → g(r,c)
next (r,c)
```

Types of Spatial Transforms

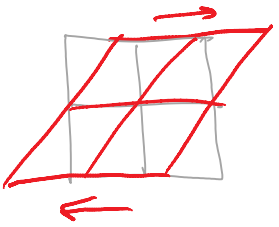


Scale

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Shear

eg. $\begin{bmatrix} 1 & a & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$



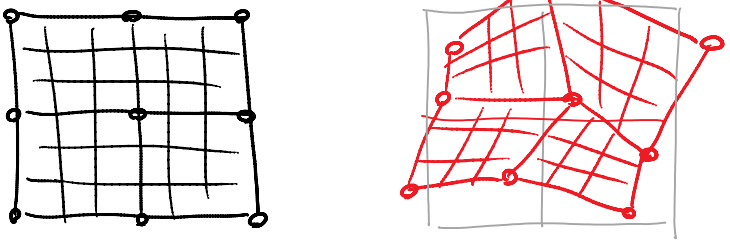
$x' = x + ay$
$y' = y$
$z' = z$

There are many ways to represent non-affine (a.k.a. "non-linear") transformations. Ultimately, one needs to define a displacement

Ultimately, one needs to define a displacement map... a set of vectors defining how each pixel moved. To implement it, we need to know the inverse of the deformation. For each post-transform pixel, we need a displacement vector that tells us where it came from.

Piecewise Linear

The displacement of a grid of control points defines the transform. Between those ctrl. points, we use a linear transformation.



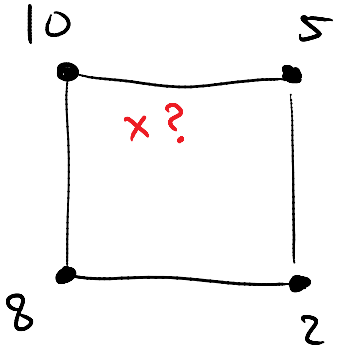
Basis Functions

Use a linear combination of mathematical functions to define the x- and y-displacements.

eg. trig, polynomials (splines, etc), Gaussians, etc.

Interpolation

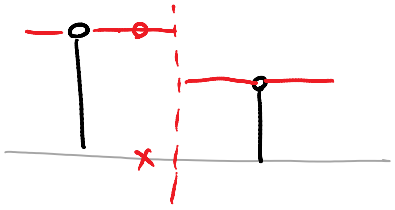
This is the operation of estimating the value of an image between its pixels.

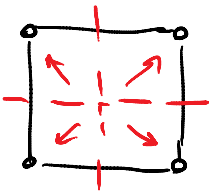


Nearest Neighbour

The simplest method is to round to the nearest sample, and use its value. The problem is that this creates discontinuities in the result.

eg.

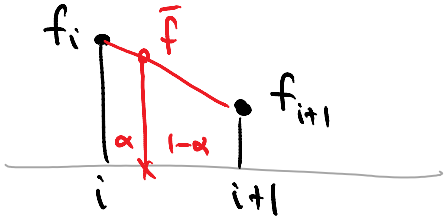




Linear Interpolation

You've heard of this before.

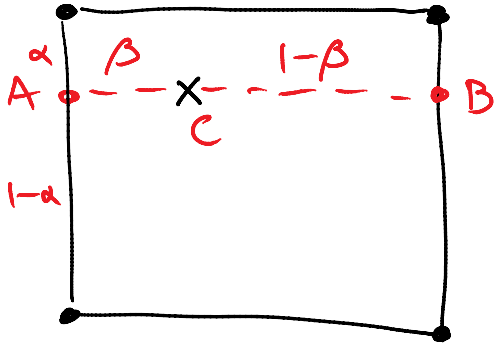
i.e.



$$\bar{f} = \alpha f_{i+1} + (1-\alpha) f_i$$

It's a weighted sum ("convex combination" in fact) of the two nearest values.

What about in 2D, for images?



Use 1D linear interp. along each dimension sequentially.

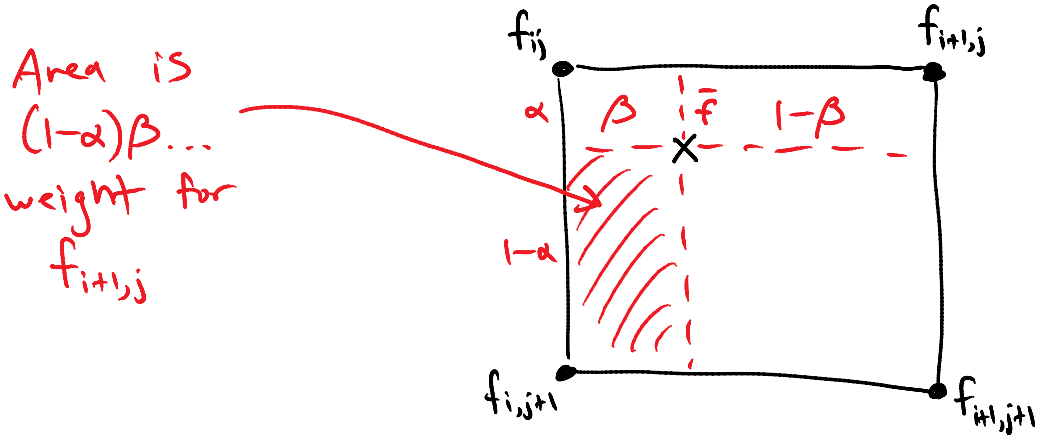
- compute A & B
- compute C

This is called "bilinear interpolation".

In 3D, "trilinear..."

Trick: Weight each sample by the area (length, volume) of its opposite region.

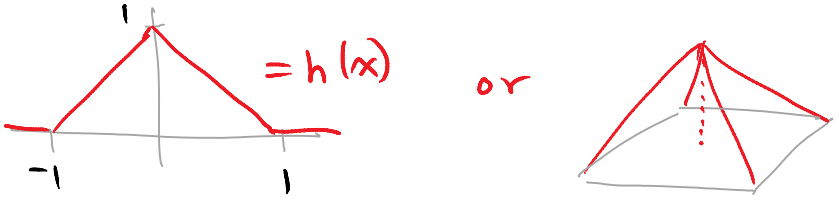
volume) of its opposite region.



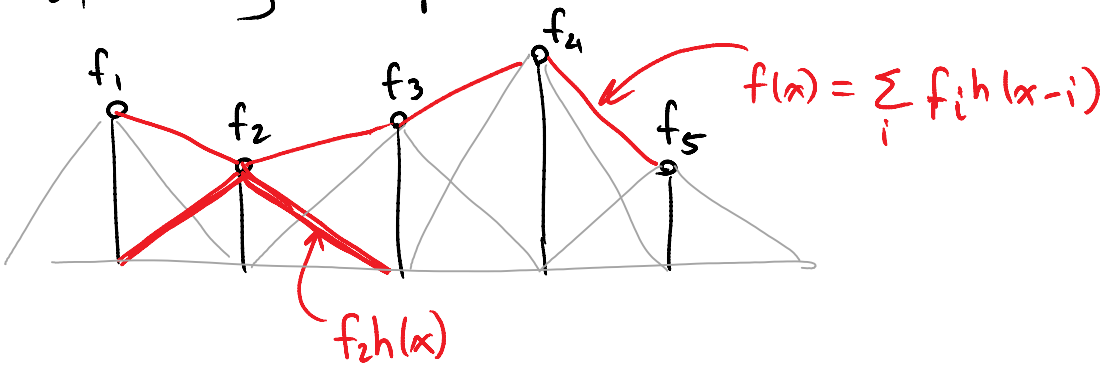
$$\bar{f} = f_{ij}(1-\alpha)(1-\beta) + f_{i+1,j}(1-\alpha)\beta + f_{i,j+1}\alpha(1-\beta) + f_{i+1,j+1}\alpha\beta$$

Convolution

Another way to think of (bi) linear interpolation is as convolving the samples with a continuous-domain interpolation kernel.



This is equivalent to placing a weighted copy of h(x) at every sample.



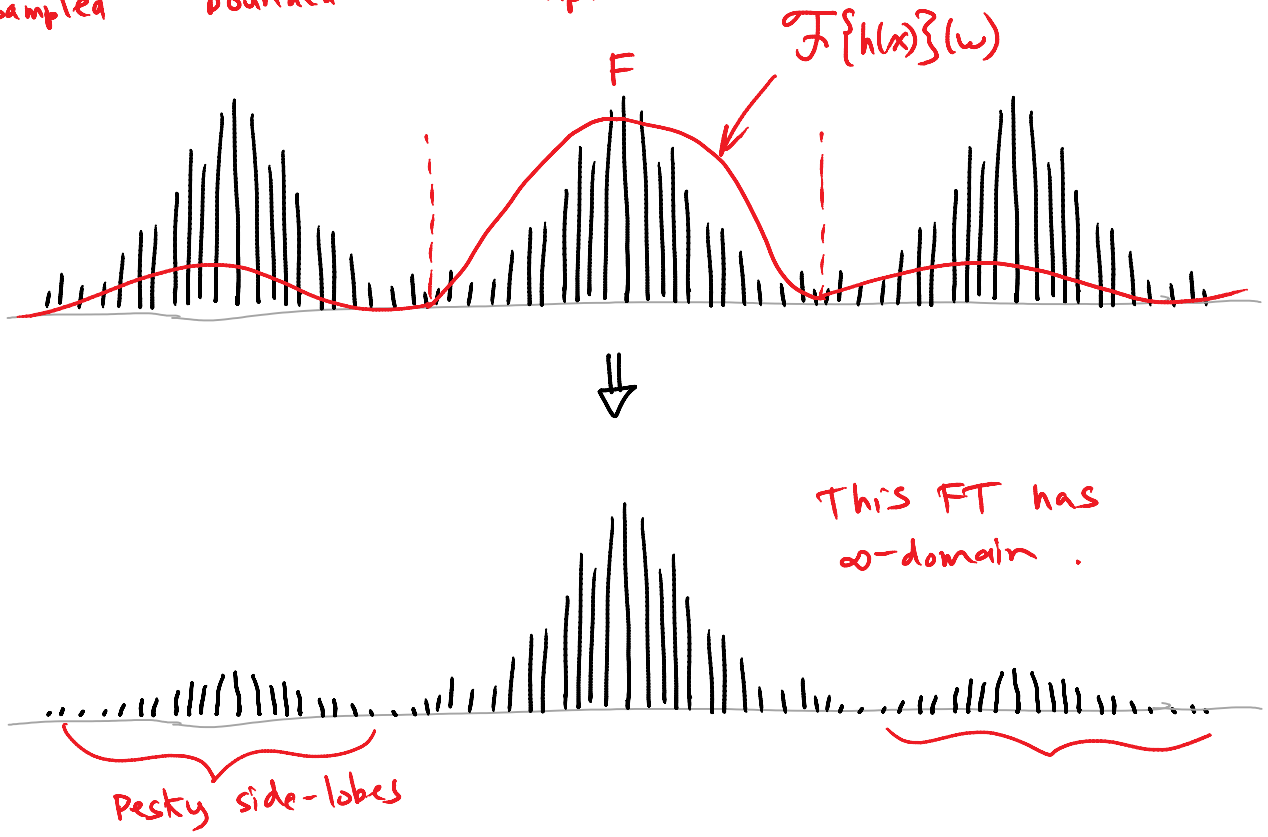
Sampling Theory (revisited)

Hence, interp. can be thought of as

Hence, interp. can be thought of as

$$(f * h(x))(x) \iff F\,H$$

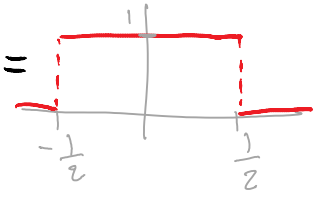
periodic, sampled — continuous, bounded — periodic, sampled — continuous, ∞-domain



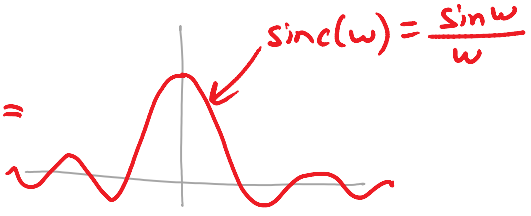
But, since we sample $(f * h(x))$, its FT becomes periodic, and those pesky side-lobes interfere with adjacent copies. This is another example of aliasing, and results in a fundamental loss of information.

For nearest-neighbour, the side-lobes are even bigger than for linear interp.

kernel $h(x) =$



$H(\omega) =$



There are lots of other kernels we can use. But there is a trade-off:

There are lots of other kernels we can use. But, there is a tradeoff:

- You want a large kernel because the scaling property says that spatially wide functions have a tighter FT, & the side lobes will be smaller. (accuracy)
- You want a small kernel so it is cheaper to interpolate (efficiency)