Geometric Transformation

Monday, January 6, 2020 10:47 AM

- Images often need to be stretched, shrunk, shifted, magnified, or geometrically transformed in some other way.

Applications:

- correction of lens distortion
- correction for viewing angle
- Correction of nonlinear field in MRI
- image registration (lining up for comparison)
- projection onto nonplanar surfaces (or inverse)
- lens designed high-resolution middle for digital zoom

Two basic algorithms required:

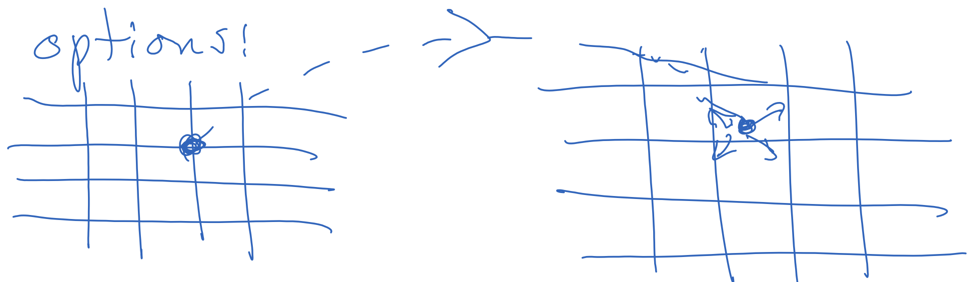
1) mapping that defines transformation from original to target coordinates

2) method of interpolating one set of sample values to another set

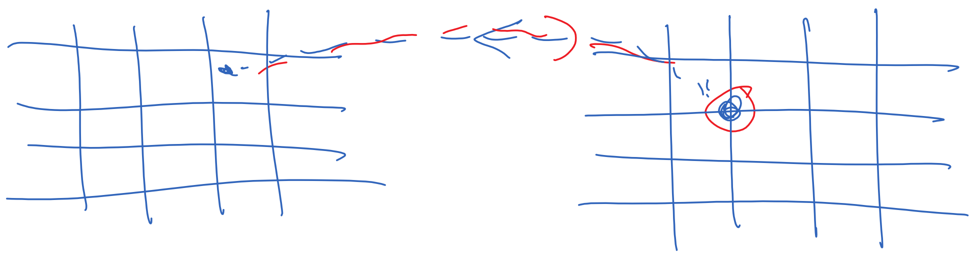
Interpolation

- integer grid points may not map to integer grid points in transformed image

two options:



forward mapping



backward mapping

Backward mapping is preferable:

- each output pixel is addressed exactly once, in line-by-line fashion
- forward mapping is wasteful — many pixel values may map outside target image

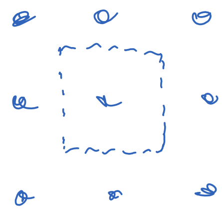
⟹ In practice, we must define the mapping that takes us from the target pixel locations back to original pixel locations.

Since original image location is generally between samples, we must interpolate.

Options:

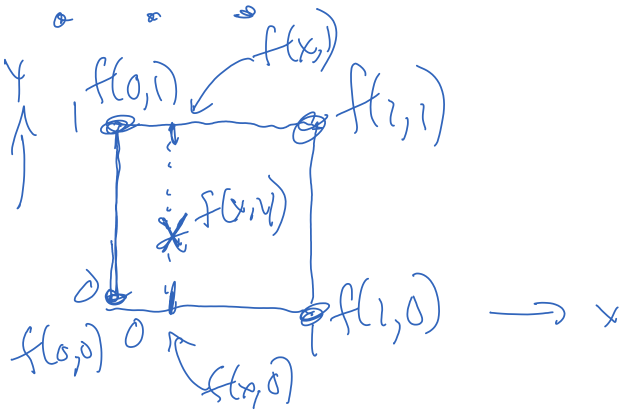
* nearest-neighbor — take value from

pixel that is closest to backward-mapped location.
Implicitly, the "continuous" original image is assumed to be square constant patches.



$\Rightarrow$ results sometimes blocky

* bilinear



- linearly interpolate to get $f(x,0)$ and $f(x,1)$

$$f(x,0) = f(0,0) + x\left[f(1,0) - f(0,0)\right]$$
$$f(x,1) = f(0,1) + x\left[f(1,1) - f(0,1)\right]$$

- then linearly interpolate between $f(x,0)$ and $f(x,1)$ to get $f(x,y)$

$$f(x,y) = f(x,0) + y\left[f(x,1) - f(x,0)\right]$$
$$= (1-x)(1-y)f(0,0) + x(1-y)f(1,0) + (1-x)y\,f(0,1) + xy\,f(1,1)$$

Read 5.1, 5.2, 5.5

Read ...

Project due Wed.

* higher-order
  - cubic interpolator
  - cubic splines
  - $\frac{\sin x}{x}$ — ideal for bandlimited images

(in 2-D, bicubic)

General idea: these higher-order methods use larger set of surrounding points to compute each interpolated point
- better results
- more computation

Spatial mappings

$x_o(x_i, y_i)$ + $y_o(x_i, y_i)$ map from input coordinates (x_i, y_i) to output coordinates (x_o, y_o)

Recall, however, that we need a backward mapping:

$$x_i(x_o, y_o) \;+\; y_i(x_o, y_o)$$

* translation $\begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$

* rotation around origin

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

* scaling

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

* affine transformation

(translation/rotation/scaling/shearing/reflection)

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

* polynomial warping (rubber sheet transformation)

$$x_i = \sum_{m=0}^{M} \sum_{n=0}^{N} a_{mn} x_0^m y_0^n$$

$$y_i = \sum_{m=0}^{M} \sum_{n=0}^{N} b_{mn} x_0^m y_0^n$$

$\{a_{mn}\}$ and $\{b_{mn}\}$ are chosen to specify a particular mapping

2nd order:

$$x_i = a_{00} + a_{01} y_0 + a_{10} x_0 + a_{11} x_0 y_0 + a_{20} x_0^2 + a_{02} y_0^2$$

$$y_i = b_{00} + b_{01} y_0 + b_{10} x_0 + b_{11} x_0 y_0 + b_{20} x_0^2 + b_{02} y_0^2$$

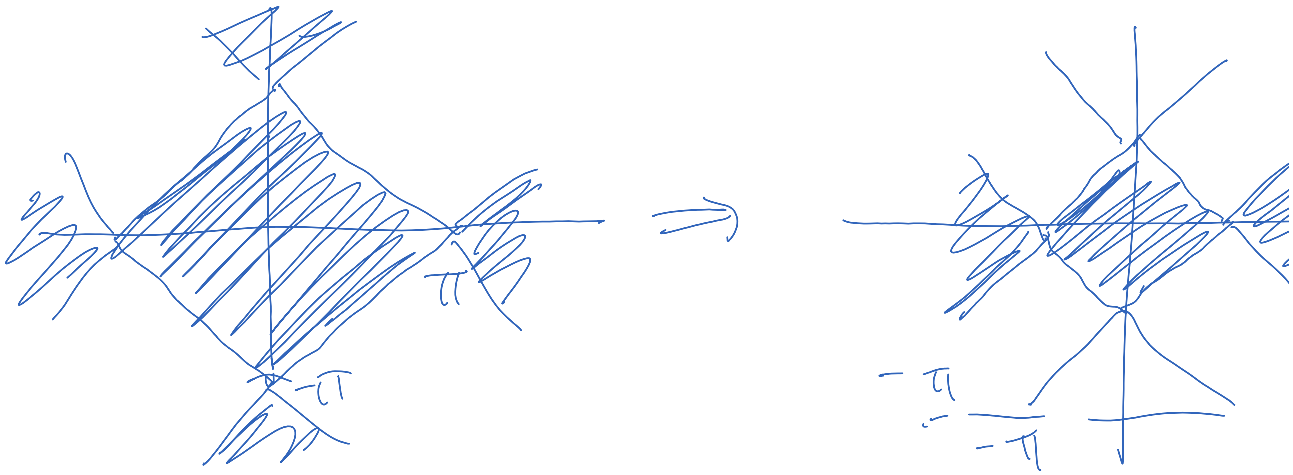
* 1st-order case is affine

~~Control point specification~~

A set of

$$\begin{bmatrix} x_{i1} \\ \vdots \end{bmatrix} \quad \begin{bmatrix} \quad \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \end{bmatrix}$$

$$\begin{bmatrix} x_{12} \\ \vdots \\ x_{in} \end{bmatrix} \qquad \vdots \qquad \begin{bmatrix} a_2 \\ a_3 \\ a_4 \end{bmatrix}$$


π
-π
-π
-π