Image Processing : Transformations, Warping and Morphing

Last Week

Courtesy:
Irfan Essa, Georgia Tech
James Hayes, Georgia Tech
Alexei Efros, UC Berkeley
Steve Seitz, U Washington

Image Transformations

- image **filtering**: change ***range*** of image
  - *g(x) = h(f(x))*

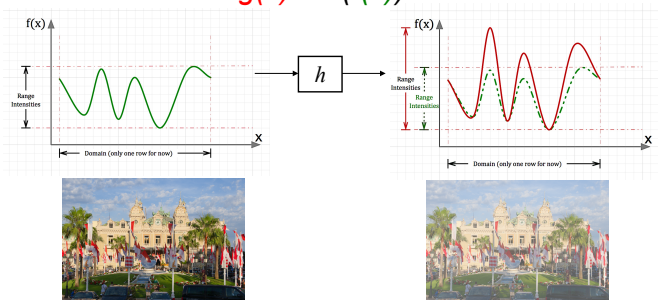
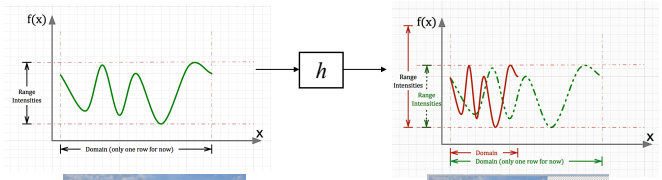
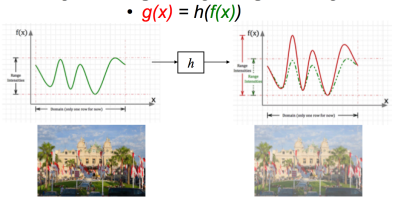


Image Transformations

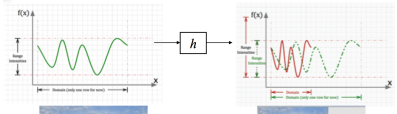
- image **filtering**: change ***domain*** of image
  - *g(x) = f(h(x))*



- image **filtering**: change ***range*** of image
  - *g(x) = h(f(x))*



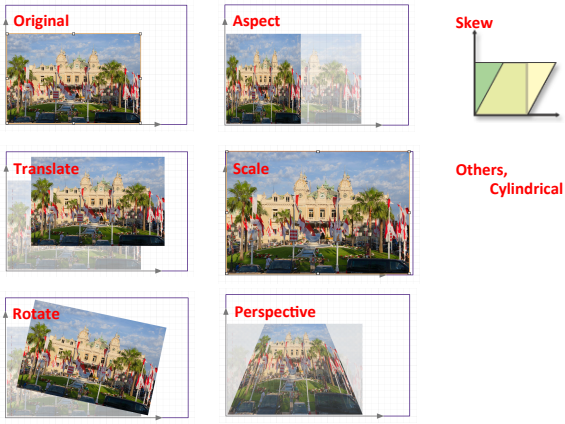
- image **filtering**: change ***domain*** of image
  - *g(x) = f(h(x))*



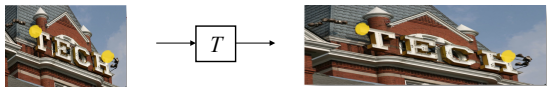
Parametric (global) warping

- Examples:
  - Original
  - Translation
  - Rotation

Parametric (global) warping



Parametric Global warping



$\mathbf{p} = (x,y)$ $\quad\quad$ $\mathbf{p'} = (x',y')$

- Transformation T is a **coordinate**-changing machine of pixels at x, y.
  p' = T(p)
- Global and parametric T:
  - Is the **same** for any point p
  - Described **parameters**
- *T can be represented* as a matrix:
  p' = **M**p

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix}$$

Image Scaling

- Multiply each components by a scalar
- **Uniform scaling**: Scalar same for x, y
- **Non-uniform**: Not same

$$\begin{aligned} x' &= ax + by \\ y' &= cx + dy \end{aligned} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

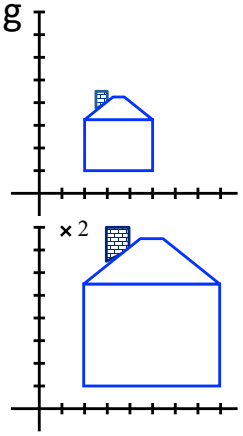


Image Scaling

- Multiply each components by a scalar
- **Uniform scaling**: Scalar same for x, y
- **Non-uniform**: Not same

$$\begin{aligned} x' &= ax + by \\ y' &= cx + dy \end{aligned} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

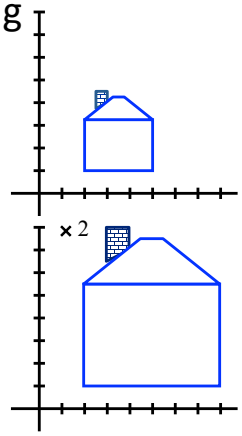


Image Scaling

- Multiply each components by a scalar
- **Uniform scaling**: Scalar same for x, y (s_x=s_y)
- **Non-uniform**: Not same (s_x≠s_y)

$$\begin{aligned} x' &= s_x x \\ y' &= s_y y \end{aligned} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

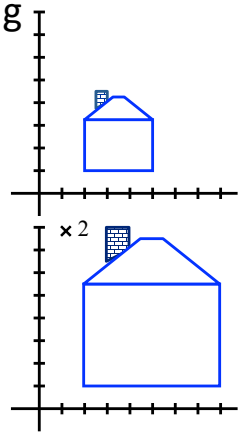
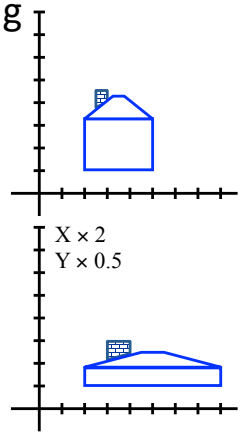


Image Scaling

- Multiply each components by a scalar
- **Uniform scaling**: Scalar same for x, y (s_x=s_y)
- **Non-uniform**: Not same (s_x≠s_y)

$$\begin{aligned} x' &= s_x x \\ y' &= s_y y \end{aligned} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



2D Image Transformation

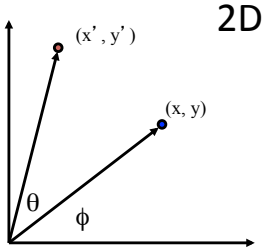
- Scaling:

$$\begin{aligned} x' &= s_x x \\ y' &= s_y y \end{aligned} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Mirror

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Rotation

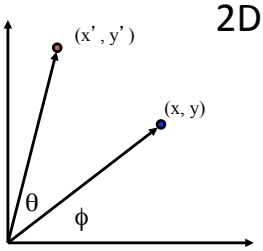


- X' is a linear combination of x and y (even if sin(θ) and cos(θ) themselves are non-linear)
- Inverse transformation?
  - Rotation by –θ
  - For rotation matrices $R^{-1} = R^T$

$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Rotation



$x = r\cos(\phi)$
$y = r\sin(\phi)$
$x' = r\cos(\phi + \theta)$
$y' = r\sin(\phi + \theta)$

Trig Identity

$x' = r\cos(\phi)\cos(\theta) - r\sin(\phi)\sin(\theta)$

$y' = r\sin(\phi)\cos(\theta) + r\cos(\phi)\sin(\theta)$

Substitute

$x' = x\cos(\theta) - y\sin(\theta)$
$y' = x\sin(\theta) + y\cos(\theta)$

$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Linear Transformations (LD)

Linear Transformations

- Mirror
- Identity
- Scale
- Shear
- Rotate

$$x' = -x,\ y' = -y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$x' = x,\ y' = y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$x' = s_x x,\ y' = s_y y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$x' = x + sh_x * y,\ y' = sh_y * x + y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ sh_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Linear Combination

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = M \begin{bmatrix} x \\ y \end{bmatrix} \Leftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

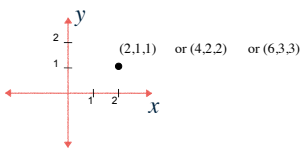
- Only Linear Combinations can be represented by a 2 x 2 matrix
  - Mirror, Identity, Shear, Scale, and Rotate
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition
- What about Translate?

$$x' = x + t_x$$
$$y' = y + t_y$$

Homogenous Coordinate

- **Idea**: Add an extra dimension
  - In 2D use a 3-vector and 3x3 matrices.
  - In 3D use a 4-vector and 5x4 matrices.
- The extra coordinate (3rd for 2D images) is an arbitrary value w, added to each 2D point (x,y) ⇒ (x,y,w)
- (x, y, w) ⇒ (x/w, y/w)
- (x, y, 0) ⇒ infinity
- (0, 0, 0) is not allowed

$$\begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

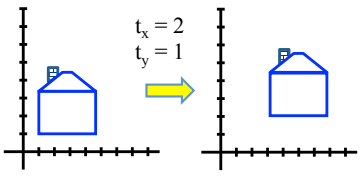


Basic 2D Transformation

$$x' = x + t_x,\ y' = y + t_y \qquad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

$$Translation = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$Identity = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



Basic 2D Transformations

- Basic 2D transformations as 3x3 matrices

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Shear

Basic 2D Transformations

Transformations can be combined

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Affine Transformations

- Affine transformations combines:
  - Linear transformations, and
  - Translations
- Properties:
  - Origin **does not necessarily** map to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Projective Transformations

- Combination of Affine transformations, and **Projective warps**
- Properties:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Parallel lines do not necessarily remain parallel
  - Ratios are not preserved

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Recovering Transformations

- What if we know *f* and *g* and want to recover the transform *T*?
- How many points correspondences would we need?
  - How many degrees of Freedom?
- Lets do some examples:

$f(x,y) \rightarrow \boldsymbol{T(x,y)} \rightarrow g(x',y')$

For Each Transformation:

a) How many points of correspondence are needed?
b) How many degrees of freedom?

Translation:
- a) 1
- **b) 2 (t_x, t_y)**

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation (Images moves and is rotated)
- a) 2 (the translation point away from origin + a pt to give you θ)
- b) **3 (tx, ty, θ)**

Affine (combines linear and translation)
- a) 3
- b) **6 (think about the matrix)**

Projective:
- a) 4
- b) 8

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Demo in OpenCV

- translation.py
- rotation.py
- shear.py
- perspective.py
- affine.py

Transformations and Warping (non-rigid)

- **Transformation**: Lines remain lines
- **Warping**: Points are mapped to points – lifting the line constraint)
  - Need to find a Mathematical function for warping from a plane to the plane

Image (non-rigid) Warping

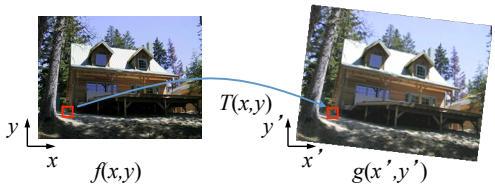
Methods/Approaches:

- Distorted through simulation of optical aberrations
  - Fish Eye Lens bulging the surface
- Projected onto a curved or mirrored surface **– texture mapping**
- Partitioned into **polygons** and then distort each polygon distorted
- Distorted using morphing
  - Transform an image to make look another image.

Warping: Two Methods

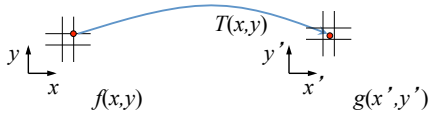
- Forward &Inverse Warping

Forwarding Warping



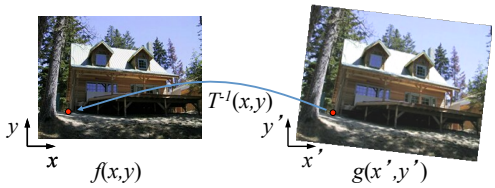
- Send each pixel *f(x,y)* to its corresponding location *(x',y') = T(x,y)* in the second image
- Q: what if pixel lands "between" two pixels?

Forward Warping



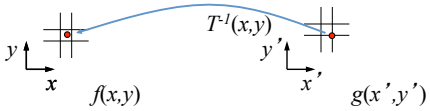
- Send each pixel f(x,y) to its corresponding location (x',y') = T(x,y) in the second image
- Q: what if pixel lands "between" two pixels?
- A: distribute color among neighboring pixels (x',y')

Inverse Warping



- Get each pixel *g(x',y')* from its corresponding location *(x,y) = T-1(x',y')* in the first image
- Q: what if pixel comes from "between" two pixels?

Inverse Warping

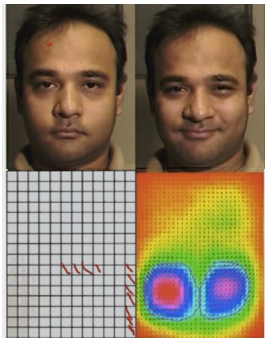


- Get each pixel *g(x',y')* from its corresponding location
  - *(x,y) = T-1(x',y')* in the first image
- Q: what if pixel comes from "between" two pixels?
- A: Interpolate color value from neighbors

Forward vs. Inverse Warping

- Q: Which trade-offs are OK?
- A: usually inverse ⇒ eliminates holes
  however, it requires an invertible warp
- function ⇒ not always possible...

Mesh-Based (Triangular) Warping



Specifying the Warp Field

- Use a sparse set of **corresponding** point and with a displacement field
  - **Triangulate** the set of points on Source
  - Use the affine model for each triangle
  - Triangulate Target with displaced Points
    - Find the transformations
  - Use inverse mapping
  - Generate the warp field.

Reading

- Reading: Book Chapter 2-3
  - 2.1.1, 2.1.2 Transformations
  - 3.6.2 Mesh-based warping
- Beier and Neely (1992) "Feature-based Image Metamorphosis"ACM SIGGRAPH 1992
- http://www.graphicsmill.com/docs/gm5/Transformations.htm
- Michael Jackson Morphing Video
  - https://www.youtube.com/watch?v=3ZHtL7CirJA

Optional Assignments

Warping/Morphing:

- http://www.cs.otago.ac.nz/cosc450/assignments/assignment2_2016.pdf
- http://www.learnopencv.com/face-morph-using-opencv-cpp-python/
  - Animate the morphing
- Perspective Corrector – find alignment points
  - Berlin Wall use that as an example

Create a facial mapper (HW or Project):

- http://www.dailymail.co.uk/femail/article-3691691/Are-beautiful-Amber-Heard-Face-mapping-expert-puts-FEMAIL-s-faces-test-compare-perfect-listers-surprising-results.html

Panorama –Extend this approach of buildling panoramas by enabling multiple images

- http://www.pyimagesearch.com/2016/01/11/opencv-panorama-stitching/