EEE-6512: Image Processing and Computer Vision

Sept 27, 2017

Lecture #5: Point and Geometric Transformations

Damon L. Woodard, Ph.D.
Dept. of Electrical and Computer Engineering
dwoodard@ece.ufl.edu

Warping

Image Warping

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

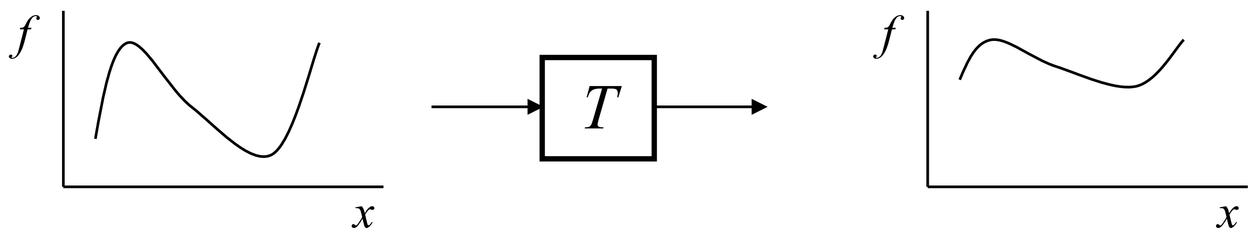


image warping: change ***domain*** of image

$$g(x) = f(T(x))$$

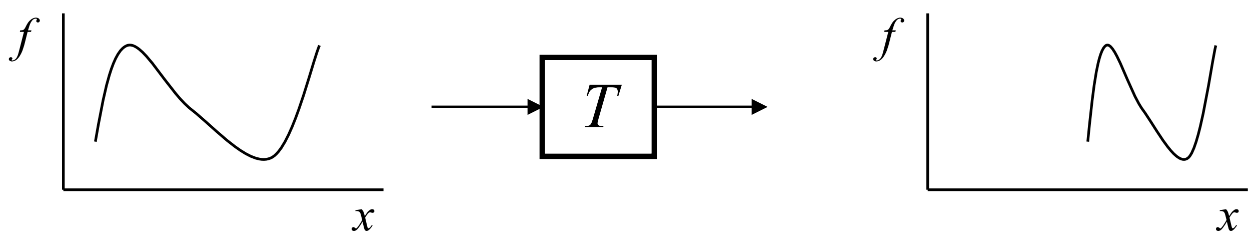


Image Warping

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

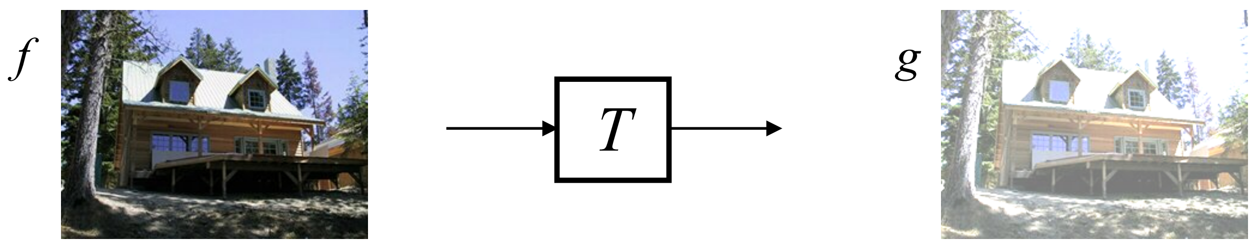
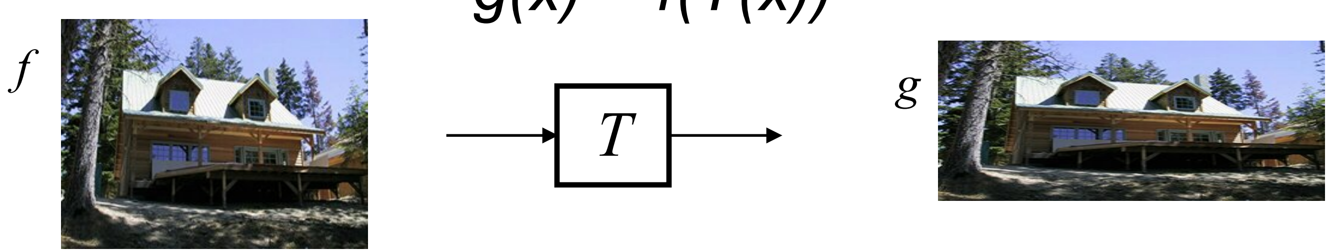


image warping: change ***domain*** of image

$$g(x) = f(T(x))$$



Parametric (global) warping

- Examples of parametric warps:

translation

rotation

aspect

affine

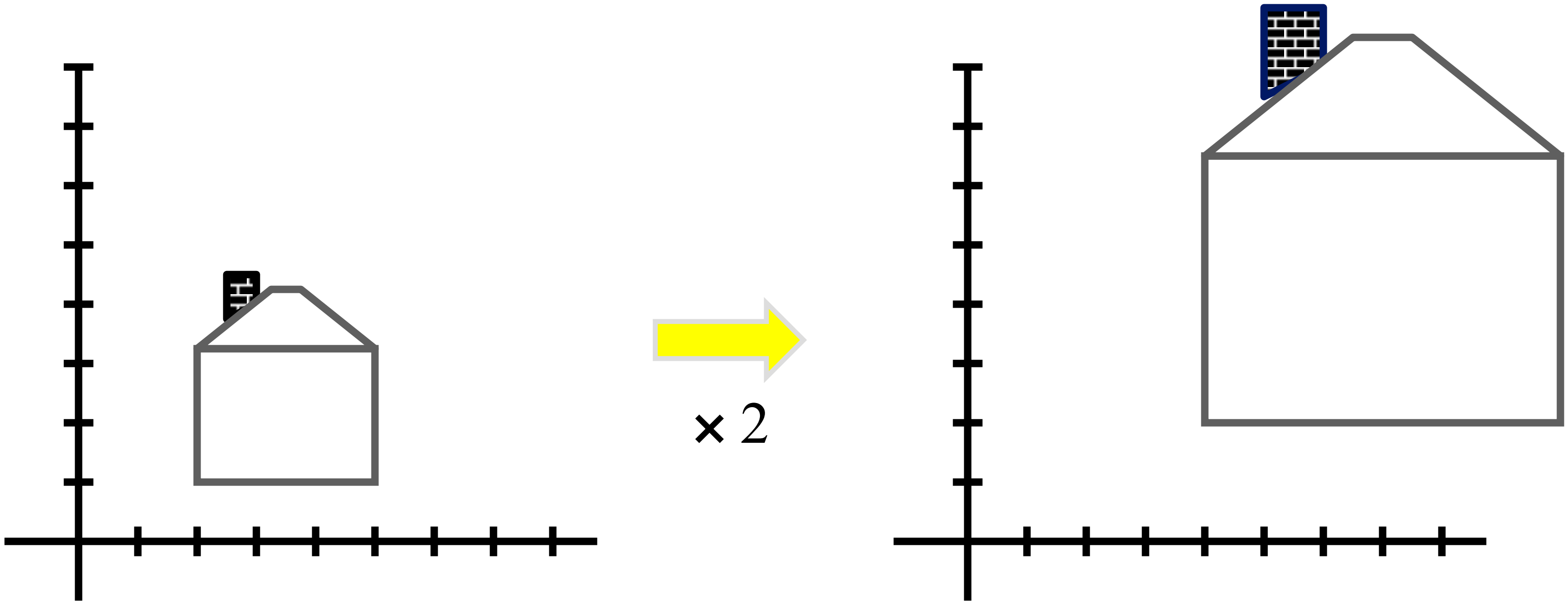
perspective

cylindrical

Non-Global Warping

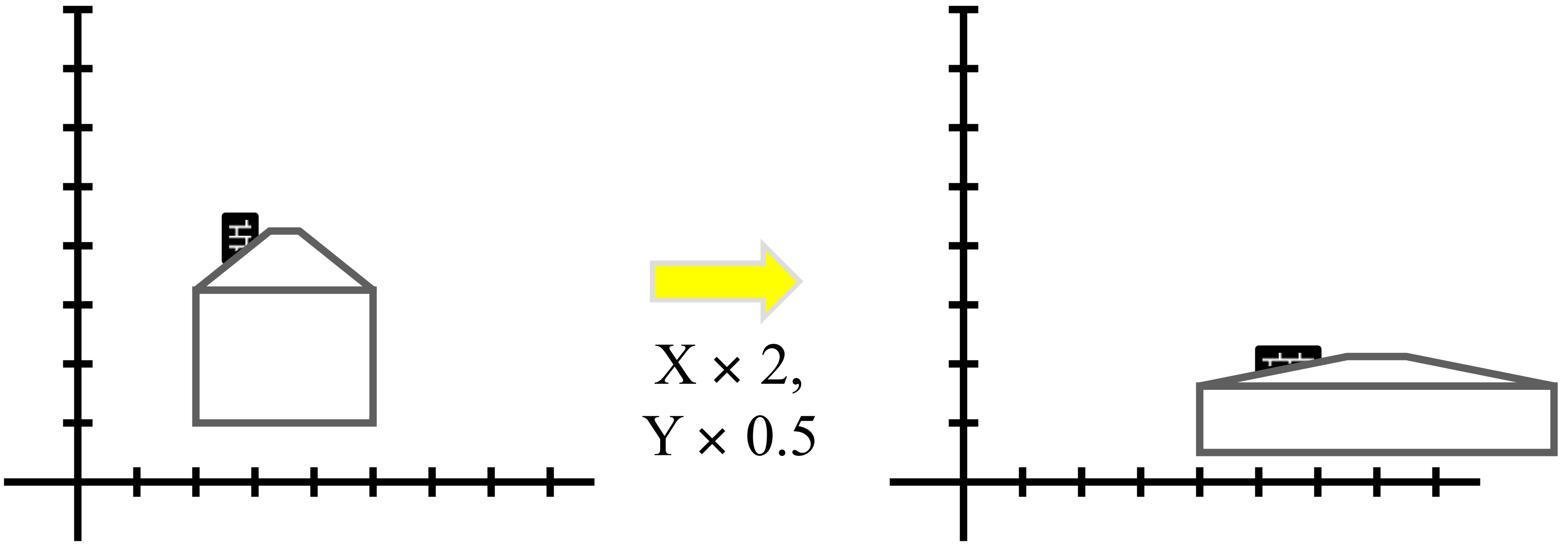
Scaling

- *Scaling* a coordinate means multiplying each of its components by a scalar
- *Uniform scaling* means this scalar is the same for all components:



Scaling

- *Non-uniform scaling*: different scalars per component:



Warping

- Consider arbitrary geometric transformations from real-valued coordinates (x, y) to real-valued coordinates (x', y'):

$$I'(x', y') = I(x, y)$$

- The **mapping function** $f: \mathbb{R}^2 \rightarrow \mathbb{R}^2$ specifies the transformation, or **warping**, from the input coordinates to the output coordinates:

$$(x', y') = f(x, y) \qquad\qquad (x, y) = f^{-1}(x', y')$$

Euclidean Transformations

- Can either be a translation, a rotation, or a reflection
- Do not change lengths and angle measures (**shape of geometric object will not change**)
- Translation and rotation can be combined into a single **Euclidean transformation**:

$$\mathbf{x}' = \mathbf{R}(\mathbf{x} - \mathbf{c}) + \mathbf{c} + \mathbf{t} = \mathbf{R}\mathbf{x} + \tilde{\mathbf{t}}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \tilde{t}_x \\ \tilde{t}_y \end{bmatrix}$$

Where:

$$\tilde{\mathbf{t}} \equiv \begin{bmatrix} \tilde{t}_x & \tilde{t}_y \end{bmatrix}^{\mathsf{T}} = -\mathbf{R}\mathbf{c} + \mathbf{c} + \mathbf{t}$$

Homogenous Coordinates

Represent coordinates in 2D with a 3D vector

$$\begin{bmatrix} x \\ y \end{bmatrix} \xrightarrow{\text{homogeneous coords}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homogeneous Coordinates

Add a 3rd coordinate to every 2D point

- (x, y, w) represents a point at location (x/w, y/w)
- (x, y, 0) represents a point at infinity
- (0, 0, 0) is not allowed

Convenient coordinate system to represent many useful transformations

Similarity Transformations

- **Similarity transformations:** a superset of Euclidean transformations.

  - They include not only translations and rotations, but also **uniform scaling**:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} k\cos\theta & -k\sin\theta & k\tilde{t}_x \\ k\sin\theta & k\cos\theta & k\tilde{t}_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Affine Transformations

- Removing the constraint to allow for any arbitrary, invertible 2 x 2 matrix leads to an **affine transformation:**

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_{13} \\ a_{23} \end{bmatrix}$$

- This can be rewritten as a single matrix multiplication using homogeneous coordinates:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{12} & a_{23}a_{12} - a_{22}a_{13} \\ -a_{21} & a_{11} & -a_{23}a_{11} + a_{21}a_{13} \\ 0 & 0 & a_{11}a_{22} - a_{12}a_{21} \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

Affine Transformations

- Generalization of Euclidean Transformations
- Special case of similarity transformation
- Lines map to Lines
- Circles become ellipses
- Lengths and angles are not preserved
- Include rotations, translation, uniform/**non-uniform scaling**, and **shear**.

Projective Transformations

- A 2D **projective transformation** relaxes the constraint that the bottom row of the matrix be [0 0 1]^T, leading to an invertible 3 x 3 matrix **H** known as a **homography:**

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \propto \underbrace{\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}}_{\mathbf{H}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- With projective transformations, homogeneous coordinates become considerably more difficult to visualize and understand.

Projective Transformations

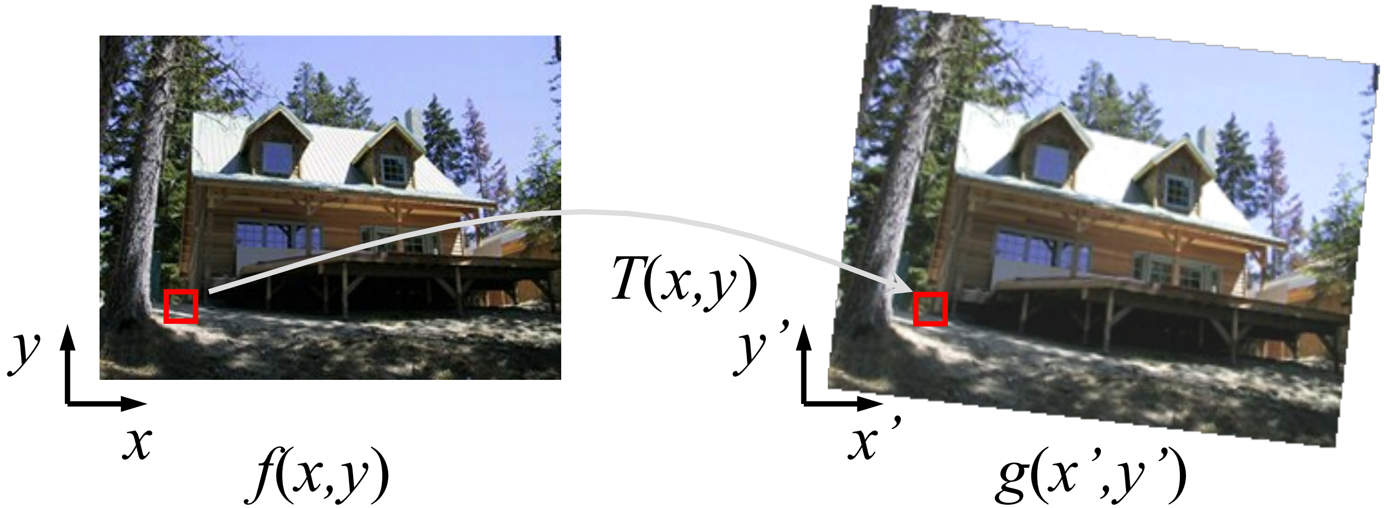
- Projective transformations …
  - Affine transformations, and
  - Projective warps

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- Properties of projective transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Parallel lines do not necessarily remain parallel
  - Ratios are not preserved
  - Closed under composition

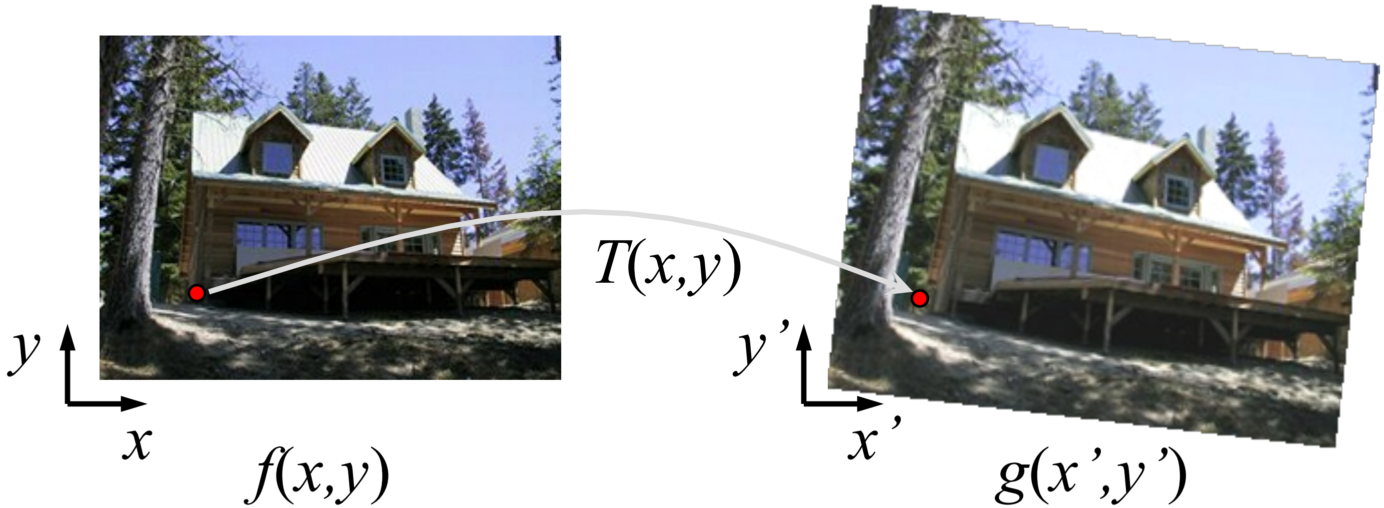
Forward / Backward Mapping

Image warping



- Given a coordinate transform $(x',y') = h(x,y)$ and a source image $f(x,y)$, how do we compute a transformed image $g(x',y') = f(T(x,y))$?

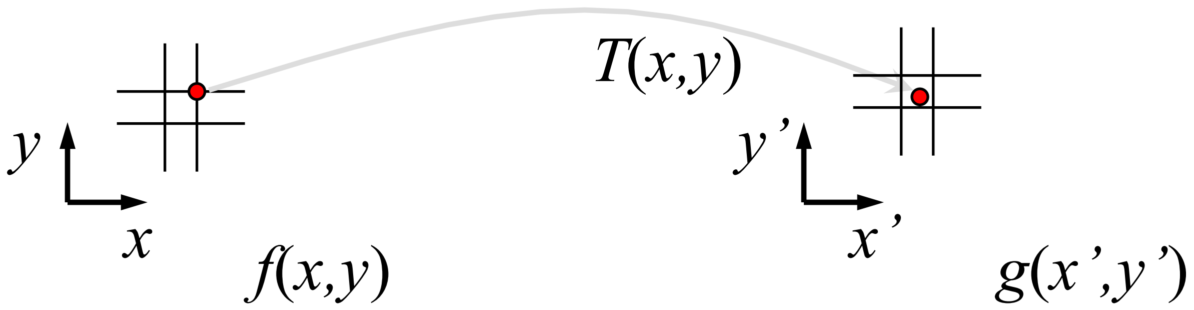
Forward warping



- 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?

Forward warping



Send each pixel $f(x,y)$ to its corresponding location

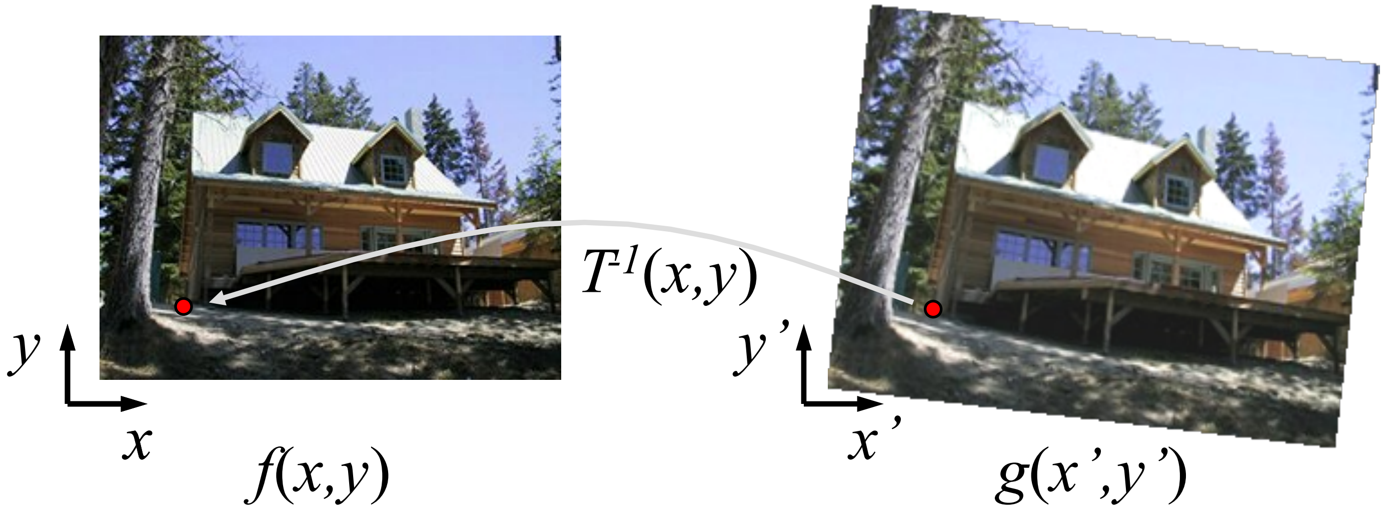
$$(x',y') = T(x,y) \text{ in the second image}$$

Q: what if pixel lands "between" two pixels?

A: distribute color among neighboring pixels (x',y')

- Known as "splatting"

Inverse warping

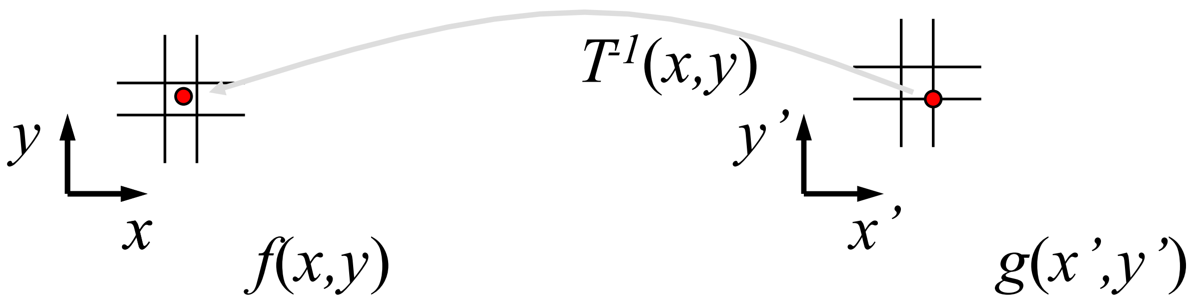


Get each pixel $g(x',y')$ from its corresponding location

- $(x,y) = T^{-1}(x',y')$ 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

$$(x,y) = T^{-1}(x',y') \text{ in the first image}$$

Q: what if pixel comes from “between” two pixels?

A: *Interpolate* color value from neighbors

- nearest neighbor, bilinear, bicubic

Forward vs. inverse warping

Q: Which is better?

A: usually inverse—eliminates holes

- however, it requires an invertible warp function—not always possible...

Questions?

Slide Credits

Image Processing and Analysis by Stan Birchfield

Some slides from Alexei Efros