

ECE5554 – Computer Vision

Lecture 1c – Coordinate Transformations

Creed Jones, PhD

Today's Objectives

Geometric Primitives

2D Coordinate Transformations

- Homogeneous Coordinates
- Affine (three-point) Mappings
- Projective (four-point) Mappings
- Other Mappings

3D Coordinate Transformations

- Transformations
- Rotations
- Quaternions

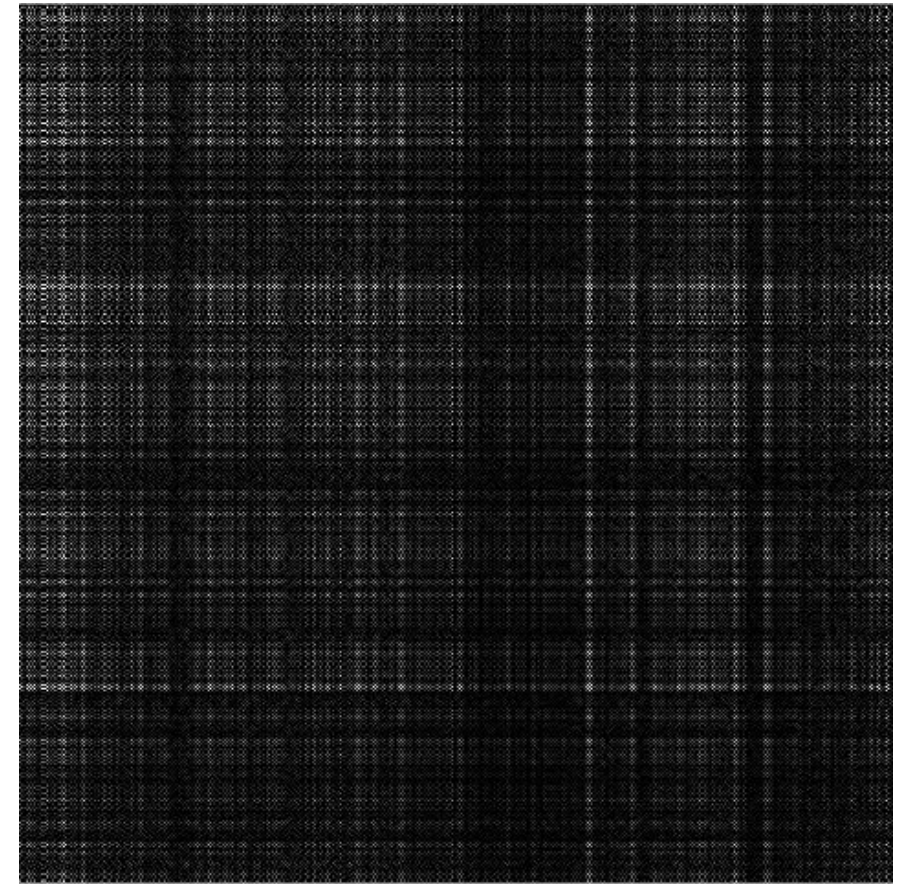
Camera Calibration

Although we treat a digital image as a matrix, many of the standard matrix operations make no sense – we need new ways of thinking of the spatial nature of an image

$M =$



$M^{-1} =$



To represent places, regions and objects in images, we can define a set of useful geometric primitives

- Points in 2D, *pixel locations*, are given by a 2D vector $\bar{x} = \begin{bmatrix} x \\ y \end{bmatrix}$ or $\bar{x} = (x, y)$
 - In *homogeneous coordinates* $\tilde{x} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{bmatrix}$, which is the same point as $\bar{x} = \begin{bmatrix} \tilde{x}/\tilde{w} \\ \tilde{y}/\tilde{w} \end{bmatrix}$
 - That same point can be written as the *augmented vector* $\bar{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \tilde{x}/\tilde{w} \\ \tilde{y}/\tilde{w} \\ 1 \end{bmatrix}$
- These homogeneous coordinates are convenient because they allow coordinate transformations through matrix multiplication

Other geometric primitives are defined, to specify simple shapes in image space

- 2D lines are determined by the equation $\bar{x} \cdot \tilde{l} = ax + by + c = 0$
 - The vector $\tilde{l} = (a, b, c)$ specifies the line in space
 - If we normalize so that $l = (\hat{n}, d)$ and $\|\hat{n}\| = 1$, then \hat{n} is the unit normal vector perpendicular to the line, and d is the perpendicular distance to the origin
 - The intersection of two lines is given by $\tilde{p} = \tilde{l}_1 \times \tilde{l}_2$
 - The line between two points is given by $\tilde{l} = \tilde{p}_1 \times \tilde{p}_2$
- 3D points in homogeneous form look like $\tilde{x} = (\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w})$
- 3D planes are determined by $\bar{x} \cdot \tilde{m} = ax + by + cz + d = 0$
 - The vector $\tilde{m} = (a, b, c, d)$ specifies the plane in space
- 3D lines are usually written in terms of any two points on the line

Often we modify an image geometrically by changing the locations at which pixel values are stored; operations to do this are called geometric transformations

- In the general case, each pixel value is replaced by the pixel value from another location in the image: $I'(c', r') \leftarrow I(c, r)$
- The relation between the old pixel location (c, r) and the new location (c', r') is given by a transformation function T , which is a mapping from 2D space to 2D space: $T: \mathbb{R}^2 \rightarrow \mathbb{R}^2$
- The general mathematical definition assumes continuous image dimensions x and y
 - Complication can arise from the discrete nature of the image coordinates

Many simple transformations can be expressed as separate functions operating on the x and y coordinates

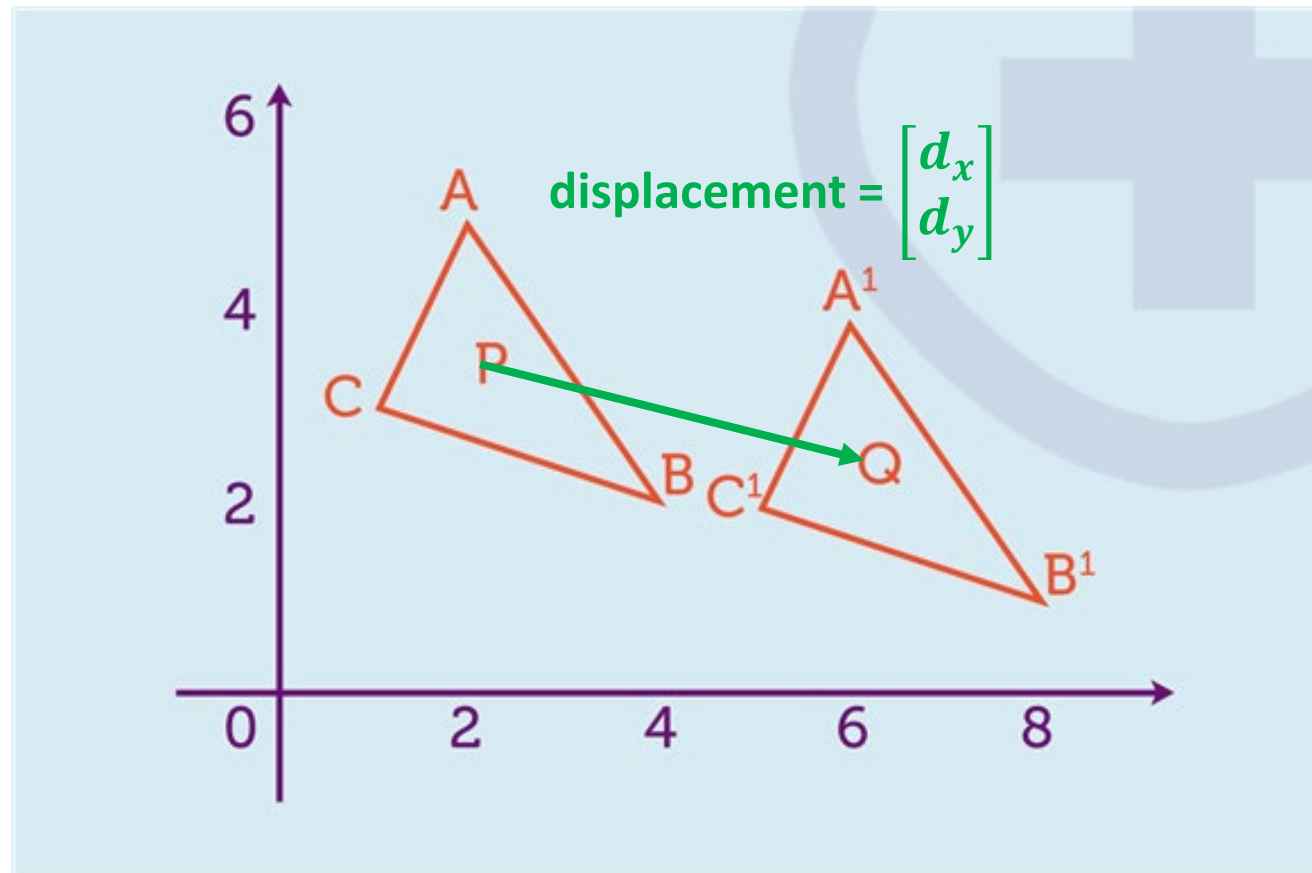
- In general, $\begin{bmatrix} x' \\ y' \end{bmatrix} = T \left(\begin{bmatrix} x \\ y \end{bmatrix} \right)$
- For simplicity, $x' = T_x(x)$ and $y' = T_y(y)$
- As an example, translation – shifting an image by a constant amount:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

- The image is shifted by a constant vector $\begin{bmatrix} d_x \\ d_y \end{bmatrix}$

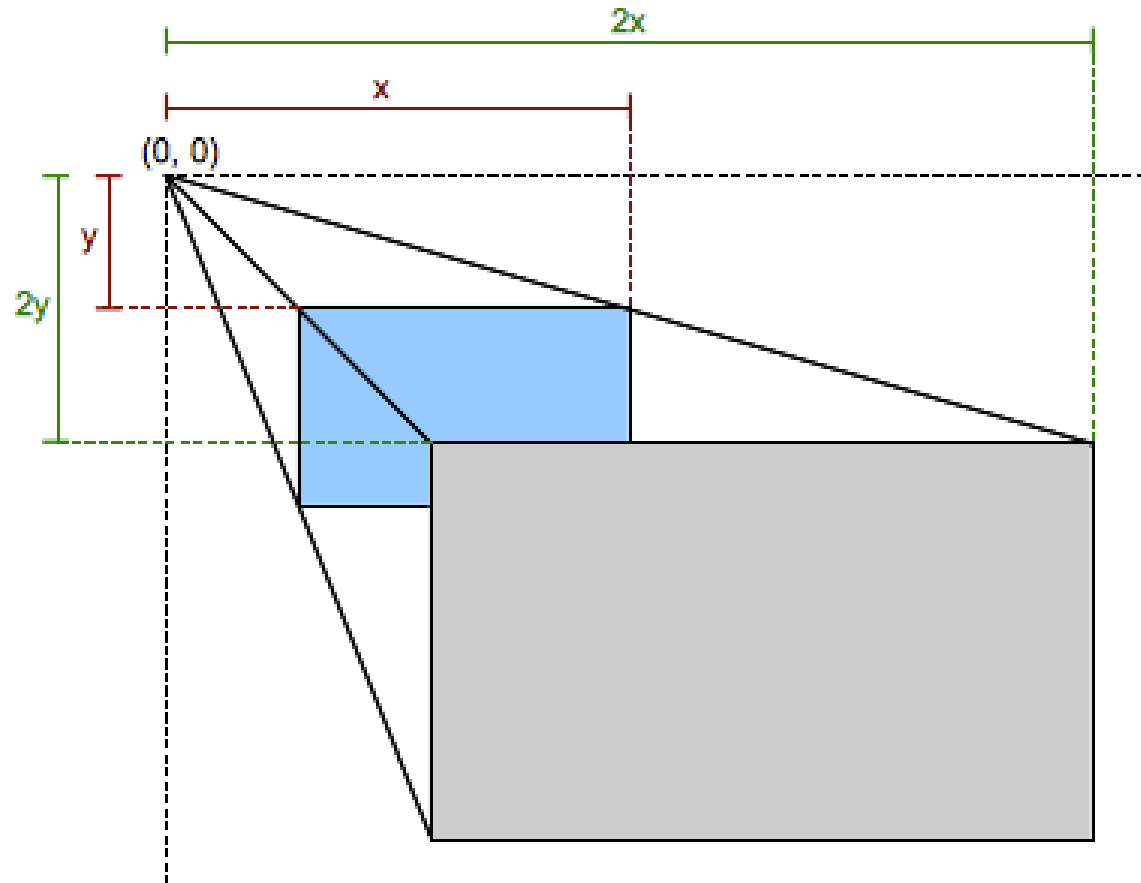
Translation shifts an image by a constant amount:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$



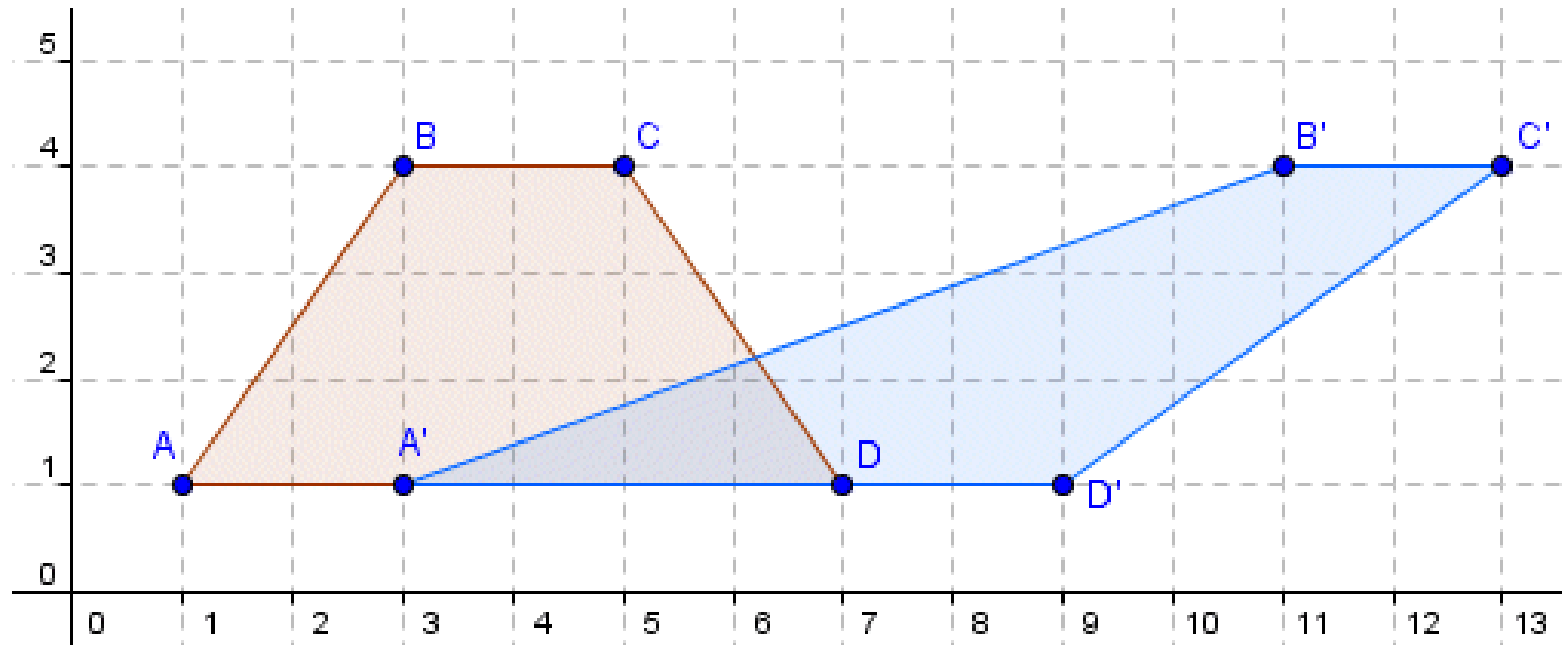
Scaling changes the size of an image by a constant amount
(may be different in x and y):

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



Shearing offsets each image coefficient by a factor times the other coefficient (different in x and y):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x + b_x y \\ y + b_y x \end{bmatrix} = \begin{bmatrix} 1 & b_x \\ b_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



In this example, $b_x = 2$, $b_y = 0$

Rotation is performed around the origin of the coordinate system
(upper left unless we adjust to the image center) :

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



In this example, the image was rotated by 60°

Recall, we can represent Cartesian coordinates as homogeneous coordinates, append a third coordinate with a constant value of 1 to each point's location (creating the augmented vector)

- $hom(\bar{x}) = hom\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \tilde{x}$ and $hom^{-1}\left(\begin{bmatrix} a \\ b \\ c \end{bmatrix}\right) = \frac{1}{c} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a/c \\ b/c \end{bmatrix}$
- homogeneous coordinates allow translations to be expressed by a matrix multiplication

$$\begin{aligned} \bullet \quad \begin{bmatrix} x' \\ y' \end{bmatrix} &= hom^{-1}\left(\begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} hom\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)\right) = hom^{-1}\left(\begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}\right) \\ &= hom^{-1}\left(\begin{bmatrix} x + d_x \\ y + d_y \\ 1 \end{bmatrix}\right) = \begin{bmatrix} x + d_x \\ y + d_y \end{bmatrix} \end{aligned}$$

The other transformations that we have seen can be performed in homogeneous coordinates

- Scaling: $\begin{bmatrix} x' \\ y' \end{bmatrix} = hom^{-1} \left(\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} hom \left(\begin{bmatrix} x \\ y \end{bmatrix} \right) \right) = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix}$
- Shear: $\begin{bmatrix} x' \\ y' \end{bmatrix} = hom^{-1} \left(\begin{bmatrix} 1 & b_x & 0 \\ b_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} hom \left(\begin{bmatrix} x \\ y \end{bmatrix} \right) \right) = \begin{bmatrix} x + b_x y \\ y + b_y x \end{bmatrix}$
- Rotation: $\begin{bmatrix} x' \\ y' \end{bmatrix} = hom^{-1} \left(\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} hom \left(\begin{bmatrix} x \\ y \end{bmatrix} \right) \right) = \begin{bmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + y \cos \theta \end{bmatrix}$

These transformations can also be expressed using non-square matrices applied to augmented position vectors

- Translation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = [\mathbf{I} \quad \mathbf{t}] \tilde{\mathbf{x}}$$






Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	4	angles	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{2 \times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{3 \times 3}$	8	straight lines	

Table 2.1 Hierarchy of 2D coordinate transformations. Each transformation also preserves the properties listed in the rows below it, i.e., similarity preserves not only angles but also parallelism and straight lines. The 2×3 matrices are extended with a third $[0^T \ 1]$ row to form a full 3×3 matrix for homogeneous coordinate transformations.

Szeliski

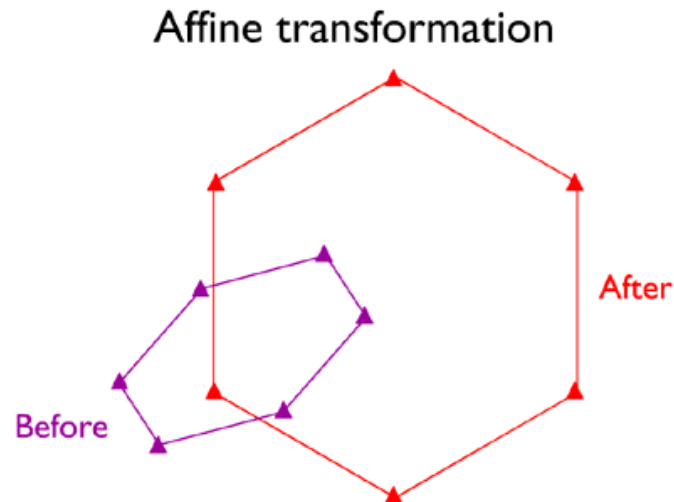
In homogeneous representation, combinations of operations can be done in series, or by multiplying the transformation matrices in the proper order

- To scale an image and then shear it,

$$\begin{aligned} \bullet \quad \begin{bmatrix} x' \\ y' \end{bmatrix} &= hom^{-1} \left(\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tilde{x} \right) hom^{-1} \left(\begin{bmatrix} 1 & b_x & 0 \\ b_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} hom \left(\begin{bmatrix} x \\ y \end{bmatrix} \right) \right) \\ &= hom^{-1} \left(\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & b_x & 0 \\ b_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} hom \left(\begin{bmatrix} x \\ y \end{bmatrix} \right) \right) \\ &= hom^{-1} \left(\begin{bmatrix} s_x & s_x b_x & 0 \\ s_y b_y & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \right) = \begin{bmatrix} s_x x + s_x b_x y \\ s_y y + s_y b_y x \end{bmatrix} \end{aligned}$$

The general combination of translation, scaling and rotation is called an Affine (three-point) Mapping

- $\begin{bmatrix} x' \\ y' \end{bmatrix} = hom^{-1} \left(A \, hom \left(\begin{bmatrix} x \\ y \end{bmatrix} \right) \right) = hom^{-1} \left(\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \right)$
- Straight lines remain straight over any affine mapping

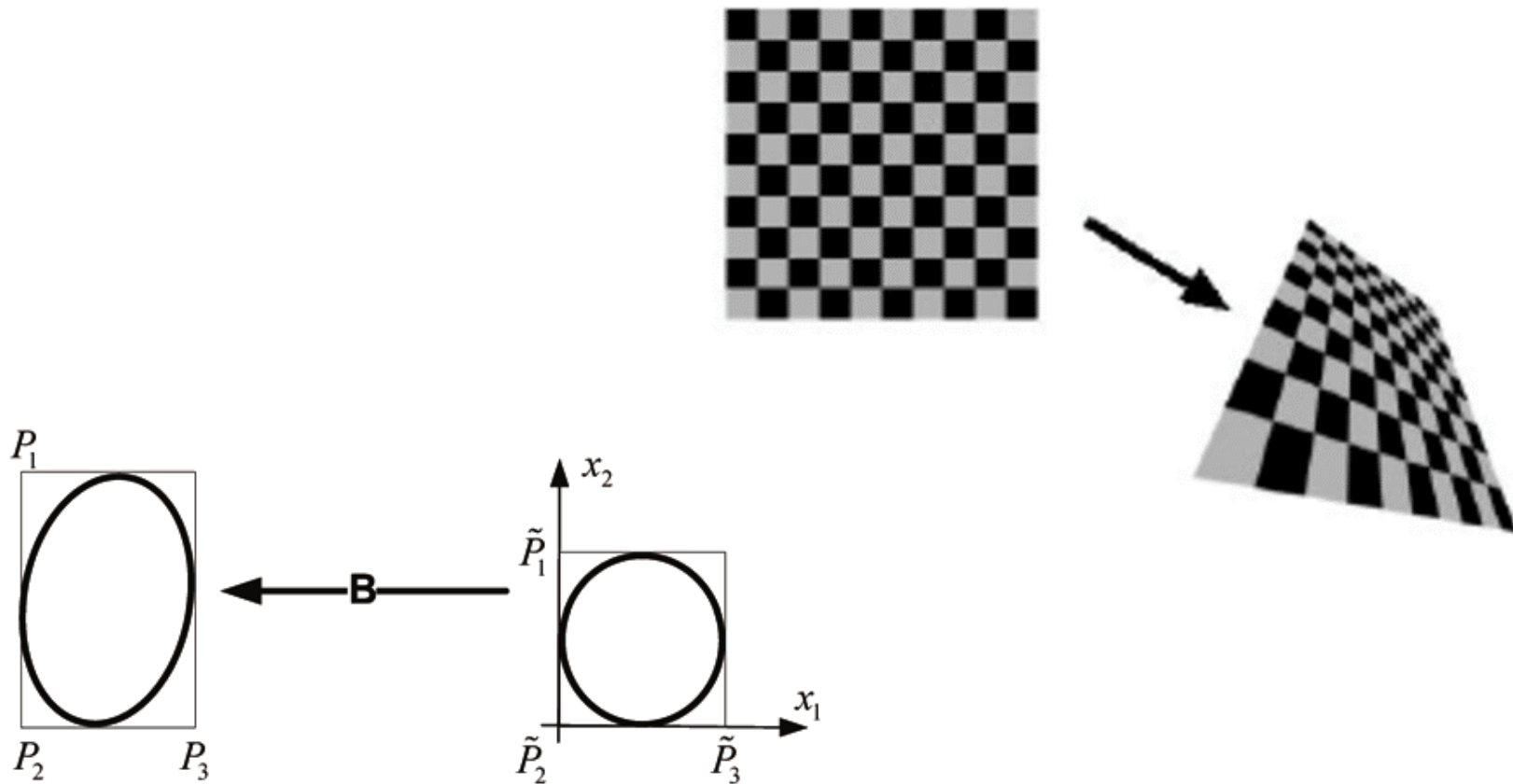


a) source image



b) image after transformation

Under an affine transform, lines remain lines, circles become ellipses, rectangles become parallelograms and polygons keep the same number of edges

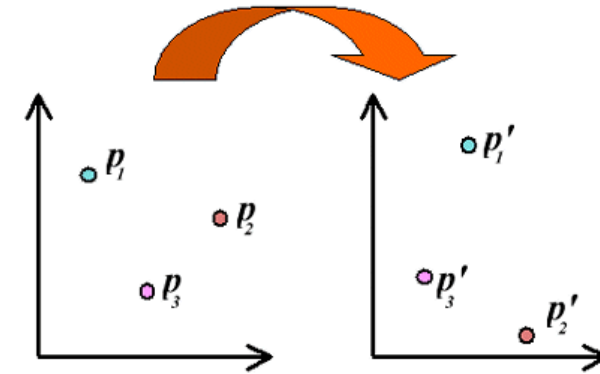


An affine transform is determined by three points in the original image and their corresponding points in the result image

- The resulting affine transform is:

$$A = \frac{1}{d} \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix}$$

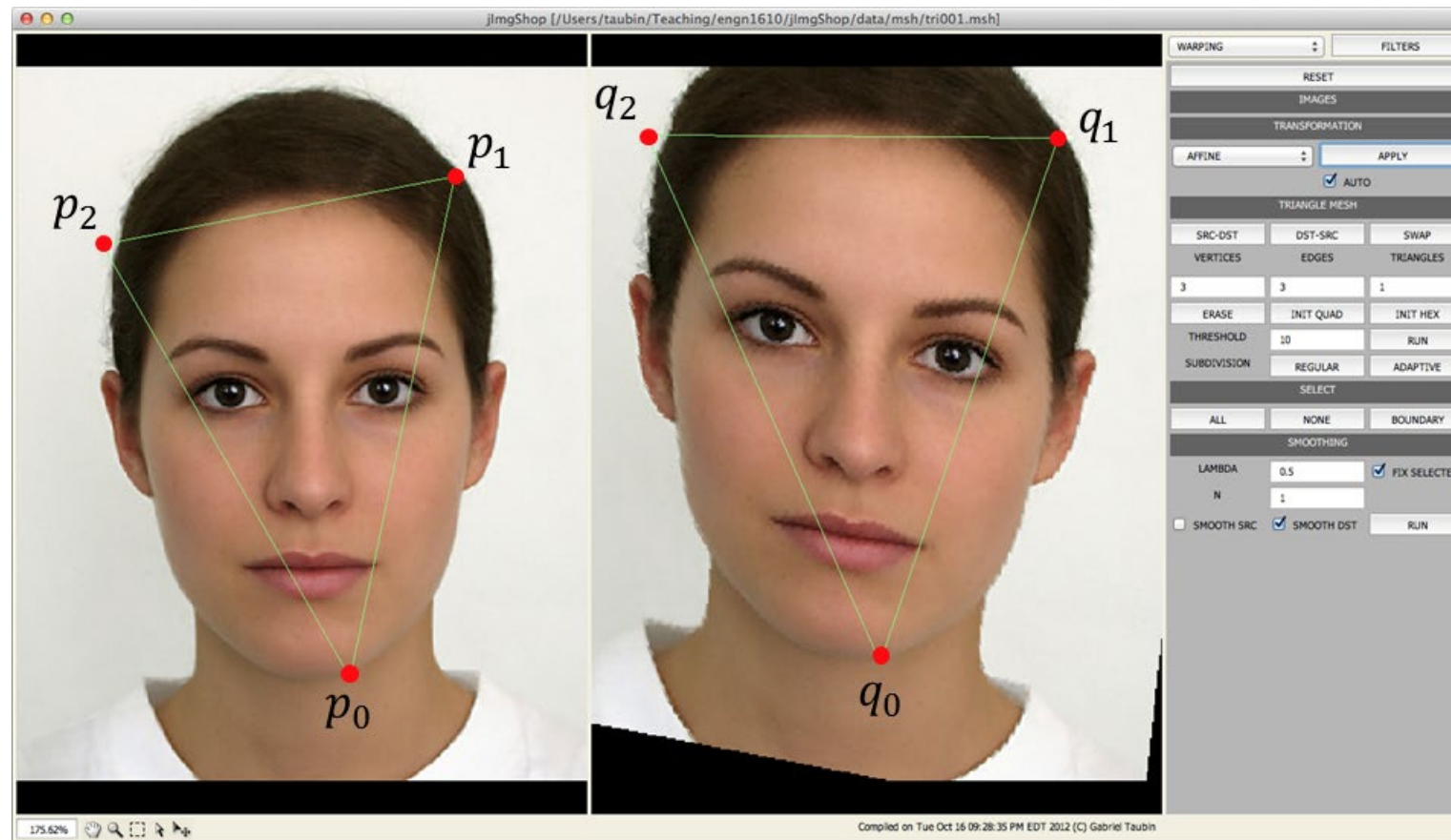
- $a_{00} = y_0(x'_1 - x'_2) + y_1(x'_2 - x'_0) + y_2(x'_1 - x'_2)$
- $a_{01} = x_0(x'_2 - x'_1) + x_1(x'_0 - x'_2) + x_2(x'_1 - x'_0)$
- $a_{02} = x_0(y_2x'_1 - y_1x'_2) + x_1(y_0x'_2 - y_2x'_0) + x_2(y_2x'_0 - y_0x'_1)$
- $a_{10} = y_0(y'_1 - y'_2) + y_1(y'_2 - y'_0) + y_2(y'_0 - y'_1)$
- $a_{11} = x_0(y'_2 - y'_1) + x_1(y'_0 - y'_2) + x_2(y'_1 - y'_0)$
- $a_{12} = x_0(y_2y'_1 - y_1y'_2) + x_1(y_0y'_2 - y_2y'_0) + x_2(y_2y'_0 - y_0y'_1)$
- $d = x_0(y_2 - y_1) + x_1(y_0 - y_2) + x_2(y_1 - y_0)$



Take a moment and pick up a rectangular object near you – a piece of paper, a phone, a small card...

- Try various motions with the object and pay attention to the shape of the object in the image formed by your visual system
 - Translation
 - Rotating the paper clockwise
 - Moving toward and away from you
 - Moving only one side away from you
 - Moving only one corner away from you

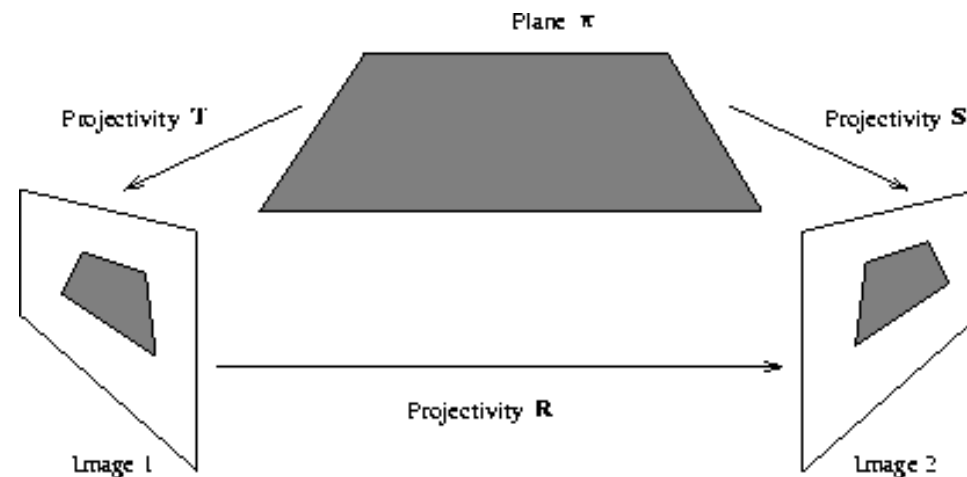
A common use of the affine transform is to align various face images to the same three-point reference locations (chin and eyes, for example) for face recognition



A projective (four-point) mapping is defined by the relationship between two quadrilaterals; it can introduce projections

- $$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & 1 \end{bmatrix}$$

- The relative spacing (ratio of distances) of points on a line is not preserved



Bilinear mappings cannot be represented by matrix multiplications, even using homogenous coordinates

- $x' = a_0x + a_1y + a_2xy + a_3$
- $y' = b_0x + b_1y + b_2xy + b_3$
- Straight lines are not preserved
 - lines map to quadratic curves
- Circles don't necessarily map to ellipses



There are a few other non-linear mappings that are used on occasion – consider the case where the image is reflected by a complex surface

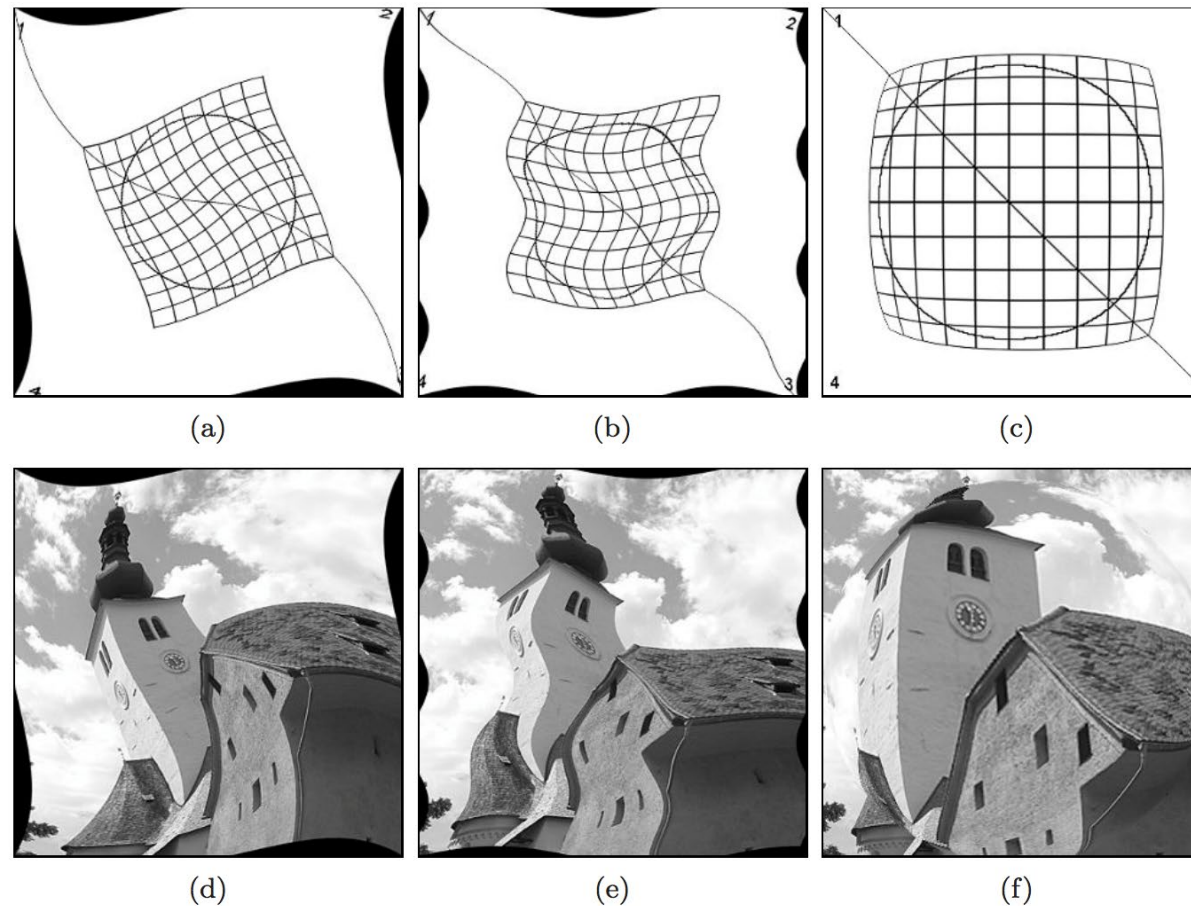


Figure 10.7 Various nonlinear image deformations: *twirl* (a,d), *ripple* (b,e), and *sphere* (c,f) transformations. The original (source) images are shown in Fig. 10.6 (a) and Fig. 10.1 (a), respectively.

3D COORDINATE TRANSFORMATIONS

In three dimensions, similar transformations are achievable by (slightly larger) matrices






Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} I & t \end{bmatrix}_{3 \times 4}$	3	orientation	
rigid (Euclidean)	$\begin{bmatrix} R & t \end{bmatrix}_{3 \times 4}$	6	lengths	
similarity	$\begin{bmatrix} sR & t \end{bmatrix}_{3 \times 4}$	7	angles	
affine	$\begin{bmatrix} A \end{bmatrix}_{3 \times 4}$	12	parallelism	
projective	$\begin{bmatrix} \tilde{H} \end{bmatrix}_{4 \times 4}$	15	straight lines	

Table 2.2 Hierarchy of 3D coordinate transformations. Each transformation also preserves the properties listed in the rows below it, i.e., similarity preserves not only angles but also parallelism and straight lines. The 3×4 matrices are extended with a fourth $[0^T \ 1]$ row to form a full 4×4 matrix for homogeneous coordinate transformations. The mnemonic icons are drawn in 2D but are meant to suggest transformations occurring in a full 3D cube.

Szeliski

Quaternions are four-dimensional vectors that can be used to cleanly represent points and rotations in 3D

- Unit quaternions have the form $\mathbf{q} = (q_x, q_y, q_z, q_w)$, where $\|\mathbf{q}\| = 1$
- To rotate a 3D vector, starting at the origin and ending at point p , by an angle specified by a unit quaternion q :

- Form $\tilde{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$

- Find q^{-1} (identical to q except the sign of the last component is flipped)

- Calculate $\tilde{p}' = q\tilde{p}q^{-1}$

Simple 3D translation (offset)

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Rotation in 3D

- about z axis

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

- about x axis

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

- about y axis

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

Rotation

about arbitrary axis through the origin:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$\mathbf{u}_1 = (u_{11}, u_{12}, u_{13})^T$ is a unit vector
in the direction
of the new x' -axis (etc.)

CAMERA CALIBRATION

It's possible to write a transformation that will convert from a point in 3D space to its destination in a 2D image

- f is the focal length of the optical system
- (c_0, r_0) is the optical center or the *principal point*
- s is the *skew coefficient*

$$\begin{bmatrix} sc \\ sr \\ s \end{bmatrix} = \begin{bmatrix} f & 0 & c_0 & 0 \\ 0 & -f & r_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

The (row, column) location of an image point

Intrinsic camera parameters;
Sometimes this is called the
“camera matrix” or the “projection matrix”

A scene point in a 3D
“camera-coordinate”
reference frame

It's also useful to split the 3D to 2D transformation into two pieces – one related to the camera (focal length, for example) and one that converts between two 3D frames of reference

$$\begin{bmatrix} sC \\ sR \\ s \end{bmatrix} = \mathbf{K} \mathbf{M} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Intrinsic camera parameters (3x4 matrix)

Extrinsic camera parameters (4x4 matrix), which map any world-coord. point to the camera-coordinate ref. frame

A scene point in a 3D "world-coordinate" reference frame

Camera calibration

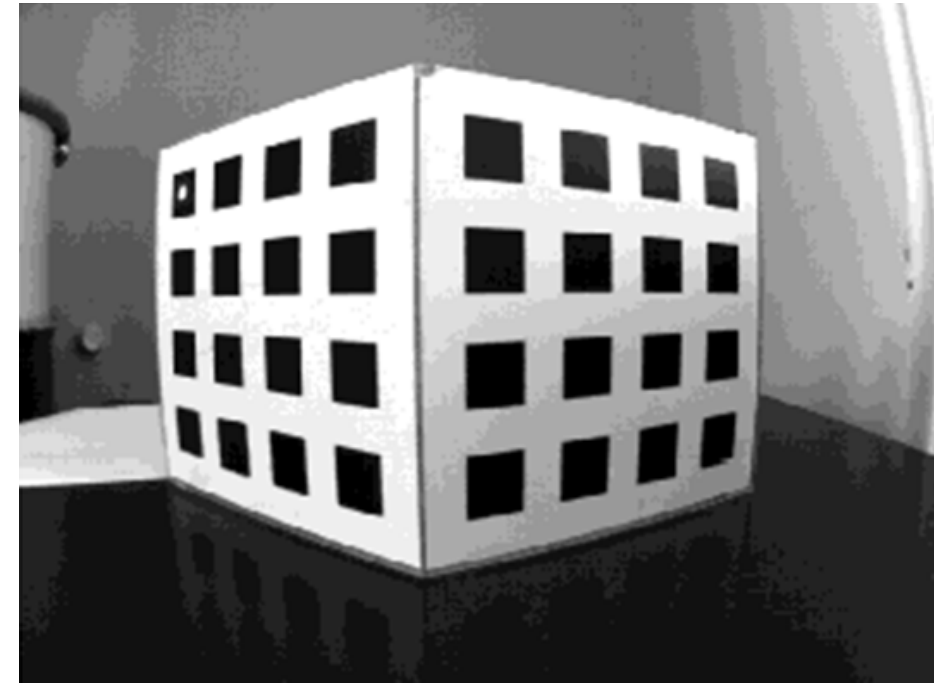
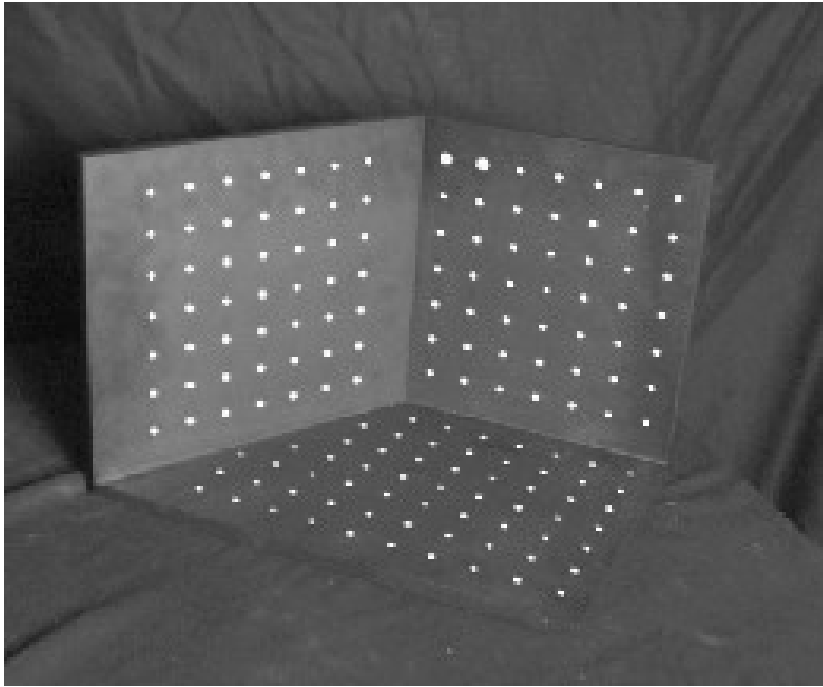
$$\begin{bmatrix} sc \\ sr \\ s \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

One approach

First: Collect some known 3D-to-image correspondences

Next: Solve for the a_{ij} values in the A matrix above

Finally: Decompose (if needed) into intrinsics and extrinsics



- Example: use a calibration target with easily-detectable points at known locations

$$c = \frac{a_{11}x + a_{12}y + a_{13}z + a_{14}}{a_{31}x + a_{32}y + a_{33}z + a_{34}}$$

$$r = \frac{a_{21}x + a_{22}y + a_{23}z + a_{24}}{a_{31}x + a_{32}y + a_{33}z + a_{34}}$$

One approach

First: Collect some known 3D-to-image correspondences

Next: Solve for the a_{ij} values in the A matrix above

Finally: Decompose (if needed) into intrinsics and extrinsics

$$c = \frac{a_{11}x + a_{12}y + a_{13}z + a_{14}}{a_{31}x + a_{32}y + a_{33}z + a_{34}}$$

$$r = \frac{a_{21}x + a_{22}y + a_{23}z + a_{24}}{a_{31}x + a_{32}y + a_{33}z + a_{34}}$$

$$c(a_{31}x + a_{32}y + a_{33}z + a_{34}) = a_{11}x + a_{12}y + a_{13}z + a_{14}$$

$$r(a_{31}x + a_{32}y + a_{33}z + a_{34}) = a_{21}x + a_{22}y + a_{23}z + a_{24}$$

One approach

First: Collect some known 3D-to-image correspondences

Next: Solve for the a_{ij} values in the A matrix above

Finally: Decompose (if needed) into intrinsics and extrinsics

$$c = \frac{a_{11}x + a_{12}y + a_{13}z + a_{14}}{a_{31}x + a_{32}y + a_{33}z + a_{34}}$$

$$r = \frac{a_{21}x + a_{22}y + a_{23}z + a_{24}}{a_{31}x + a_{32}y + a_{33}z + a_{34}}$$

One approach

First: Collect some known 3D-to-image correspondences

Next: Solve for the a_{ij} values in the A matrix above

Finally: Decompose (if needed) into intrinsics and extrinsics

$$c(a_{31}x + a_{32}y + a_{33}z + a_{34}) = a_{11}x + a_{12}y + a_{13}z + a_{14}$$

$$r(a_{31}x + a_{32}y + a_{33}z + a_{34}) = a_{21}x + a_{22}y + a_{23}z + a_{24}$$

$$a_{11}x + a_{12}y + a_{13}z + a_{14} - c(a_{31}x + a_{32}y + a_{33}z + a_{34}) = 0$$

$$a_{21}x + a_{22}y + a_{23}z + a_{24} - r(a_{31}x + a_{32}y + a_{33}z + a_{34}) = 0$$

One approach

First: Collect some known 3D-to-image correspondences

Next: Solve for the a_{ij} values in the A matrix above

Finally: Decompose (if needed) into intrinsics and extrinsics

$$a_{11}x_1 + a_{12}y_1 + a_{13}z_1 + a_{14} - c_1(a_{31}x_1 + a_{32}y_1 + a_{33}z_1 + a_{34}) = 0$$

$$a_{21}x_1 + a_{22}y_1 + a_{23}z_1 + a_{24} - r_1(a_{31}x_1 + a_{32}y_1 + a_{33}z_1 + a_{34}) = 0$$

⋮

$$a_{11}x_i + a_{12}y_i + a_{13}z_i + a_{14} - c_i(a_{31}x_i + a_{32}y_i + a_{33}z_i + a_{34}) = 0$$

$$a_{21}x_i + a_{22}y_i + a_{23}z_i + a_{24} - r_i(a_{31}x_i + a_{32}y_i + a_{33}z_i + a_{34}) = 0$$

⋮

$$a_{11}x_n + a_{12}y_n + a_{13}z_n + a_{14} - c_n(a_{31}x_n + a_{32}y_n + a_{33}z_n + a_{34}) = 0$$

$$a_{21}x_n + a_{22}y_n + a_{23}z_n + a_{24} - r_n(a_{31}x_n + a_{32}y_n + a_{33}z_n + a_{34}) = 0$$

One approach

First: Collect some known 3D-to-image correspondences

Next: Solve for the a_{ij} values in the A matrix above

Finally: Decompose (if needed) into intrinsics and extrinsics

$$\begin{bmatrix} x_1 & y_1 & z_1 & 1 & 0 & 0 & 0 & 0 & -x_1 c_1 & -y_1 c_1 & -z_1 c_1 & -c_1 \\ 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & -x_1 r_1 & -y_1 r_1 & -z_1 r_1 & -r_1 \\ & & & & . & . & . & & & & & \\ x_n & y_n & z_n & 1 & 0 & 0 & 0 & 0 & -x_n c_n & -y_n c_n & -z_n c_n & -c_n \\ 0 & 0 & 0 & 0 & x_n & y_n & z_n & 1 & -x_n r_n & -y_n r_n & -z_n r_n & -r_n \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{24} \\ a_{31} \\ a_{32} \\ a_{33} \\ a_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \dots \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x_1 & y_1 & z_1 & 1 & 0 & 0 & 0 & 0 & -x_1 c_1 & -y_1 c_1 & -z_1 c_1 & -c_1 \\ 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & -x_1 r_1 & -y_1 r_1 & -z_1 r_1 & -r_1 \\ & & & & \cdot & \cdot & \cdot & & & & & \\ x_n & y_n & z_n & 1 & 0 & 0 & 0 & 0 & -x_n c_n & -y_n c_n & -z_n c_n & -c_n \\ 0 & 0 & 0 & 0 & x_n & y_n & y_n & 1 & -x_n r_n & -y_n r_n & -z_n r_n & -r_n \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{24} \\ a_{31} \\ a_{32} \\ a_{33} \\ a_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \dots \\ 0 \\ 0 \end{bmatrix}$$

- This is of the form $Q a = 0$
- The method of least squares can be used: solve for the unit vector a that minimizes $|Qa|^2$
- Solution is the eigenvector associated with the smallest eigenvalue of the matrix $Q^T Q$

Today's Objectives

Geometric Primitives

2D Coordinate Transformations

- Homogeneous Coordinates
- Affine (three-point) Mappings
- Projective (four-point) Mappings
- Other Mappings

3D Coordinate Transformations

- Transformations
- Rotations
- Quaternions

Camera Calibration