

Lecture 2: Linear Algebra Primer

Harbin Institute of Technology (Shenzhen)

Jingyong Su

sujingyong@hit.edu.cn

Outline

- Vectors and matrices
 - Basic Matrix Operations
 - Special Matrices
- Transformation Matrices
 - Homogeneous coordinates
 - Translation
- Matrix inverse
- Matrix rank
- Singular Value Decomposition (SVD)
 - Use for image compression
 - Use for Principal Component Analysis (PCA)
 - Computer algorithm



Vector

- A column vector $\mathbf{v} \in \mathbb{R}^{n \times 1}$ where

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

- A row vector $\mathbf{v}^T \in \mathbb{R}^{1 \times n}$ where

$$\mathbf{v}^T = [v_1 \quad v_2 \quad \dots \quad v_n]$$

T denotes the transpose operation



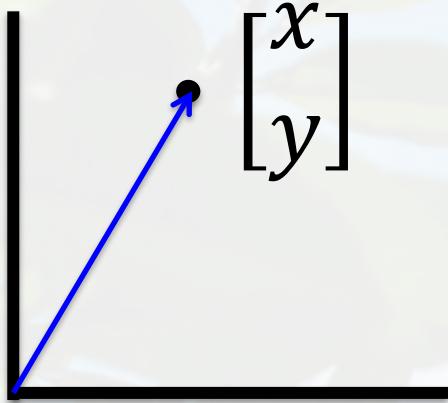
Vector

- We'll default to column vectors in this class

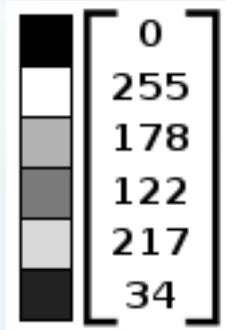
$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

- You'll want to keep track of the orientation of your vectors when programming
- You can transpose a vector V in MATLAB by writing \mathbf{V}' . (But in class materials, we will **always** use \mathbf{V}^T to indicate transpose, and we will use \mathbf{V}' to mean “ \mathbf{V} prime”)

Vectors have two main uses



- Data (pixels, gradients at an image keypoint, etc) can also be treated as a vector



- Vectors can represent an offset in 2D or 3D space
- Points are just vectors from the origin

- Such vectors don't have a geometric interpretation, but calculations like "distance" can still have value

Matrix

- A matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is an array of numbers with m rows and n columns.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & & & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}$$

- If $m = n$, we say that \mathbf{A} is square.

Images



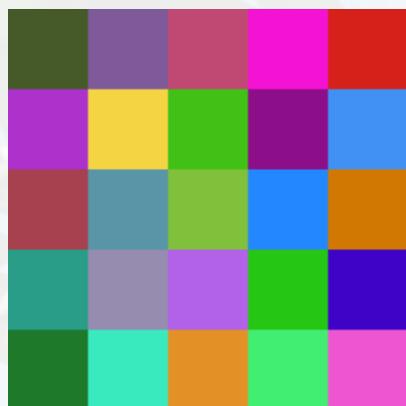
=

$$\begin{bmatrix} 193 & 180 & 210 & 112 & 125 \\ 189 & 8 & 177 & 97 & 114 \\ 100 & 71 & 81 & 195 & 165 \\ 167 & 12 & 242 & 203 & 181 \\ 44 & 25 & 9 & 48 & 192 \end{bmatrix}$$

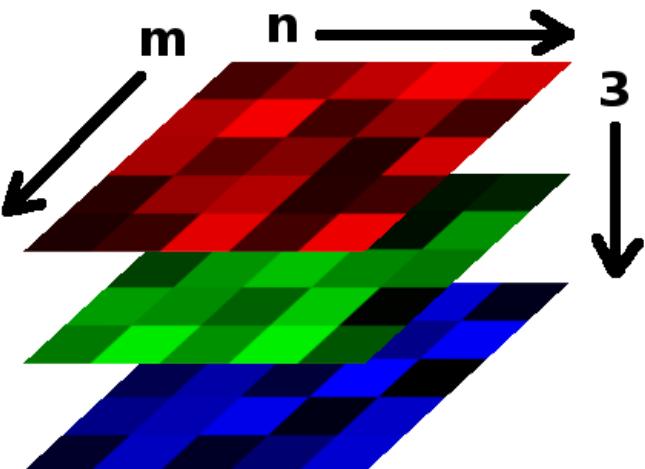
- MATLAB represents an image as a matrix of pixel brightness
- Note that matrix coordinates are NOT Cartesian coordinates. The upper left corner is $[y, x] = (1,1)$

Color Images

- Grayscale images have one number per pixel, and are stored as an $m \times n$ matrix.
- Color images have 3 numbers per pixel – red, green, and blue brightness
- Stored as an $m \times n \times 3$ matrix



=



Basic Matrix Operations

- We will discuss:
 - Addition
 - Scaling
 - Dot product
 - Multiplication
 - Transpose
 - Inverse / pseudoinverse
 - Determinant / trace



Matrix Operations

- Addition

$$\begin{bmatrix} \alpha & b \\ c & d \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} \alpha + 1 & b + 2 \\ c + 3 & d + 4 \end{bmatrix}$$

- Can only add a matrix with matching dimensions, or a scalar.

$$\begin{bmatrix} \alpha & b \\ c & d \end{bmatrix} + 7 = \begin{bmatrix} \alpha + 7 & b + 7 \\ c + 7 & d + 7 \end{bmatrix}$$

- Scaling

$$\begin{bmatrix} \alpha & b \\ c & d \end{bmatrix} \times 3 = \begin{bmatrix} 3\alpha & 3b \\ 3c & 3d \end{bmatrix}$$



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

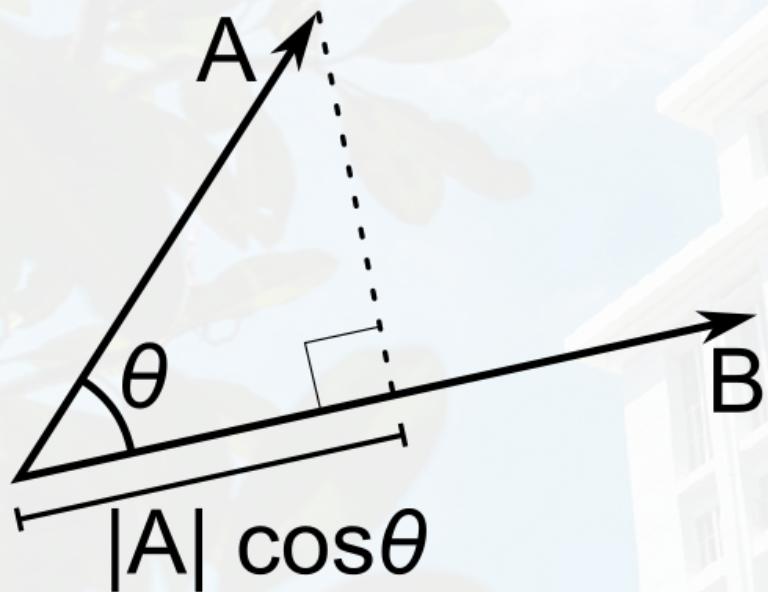
Matrix Operations

- Inner product (dot product) of vectors
 - Multiply corresponding entries of two vectors and add up the result
 - $\mathbf{x} \cdot \mathbf{y}$ is also $|\mathbf{x}| |\mathbf{y}| \cos(\text{the angle between } \mathbf{x} \text{ and } \mathbf{y})$

$$\mathbf{x}^T \mathbf{y} = [x_1 \quad \dots \quad x_n] \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^n x_i y_i \quad (\text{scalar})$$

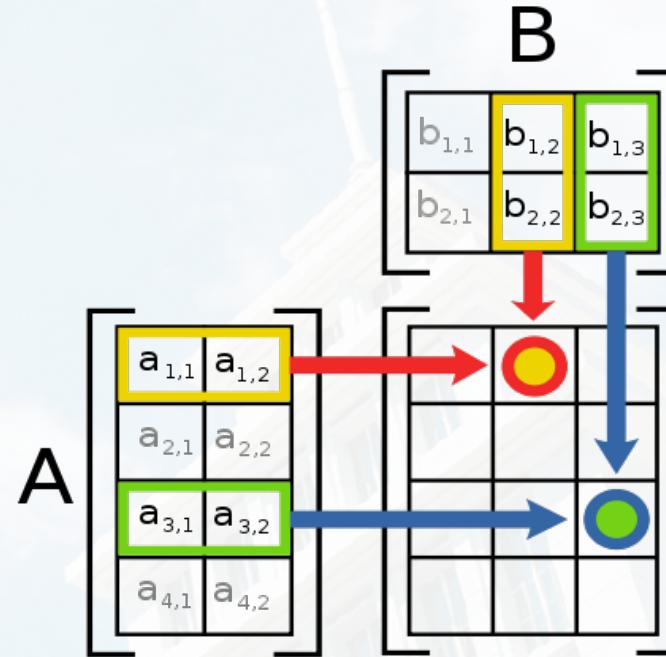
Matrix Operations

- Inner product (dot product) of vectors
 - If B is a unit vector, then $A \cdot B$ gives the length of A which lies in the direction of B



Matrix Operations

- Multiplication
- The product AB is:



- Each entry in the result is (that row of A) dot product with (that column of B)
- Many uses, which will be covered later



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

Matrix Operations

- Multiplication example:

$$\begin{array}{c} A \times B \\ \downarrow \\ \left[\begin{array}{cc} 0 & 2 \\ 4 & 6 \end{array} \right] \quad \left[\begin{array}{cc} 10 & 14 \\ 34 & 54 \end{array} \right] \\ \text{---} \\ 0 \cdot 3 + 2 \cdot 7 = 14 \end{array}$$

– Each entry of the matrix product is made by taking the dot product of the corresponding row in the left matrix, with the corresponding column in the right one.



Matrix Operations

- Powers
 - By convention, we can refer to the matrix product AA as A^2 , and AAA as A^3 , etc.
 - Obviously only square matrices can be multiplied that way



Matrix Operations

- Transpose – flip matrix, so row 1 becomes column 1

$$\begin{bmatrix} 0 & 1 & \dots \\ \downarrow & \nearrow & \\ \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}^T = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

- A useful identity:

$$(ABC)^T = C^T B^T A^T$$

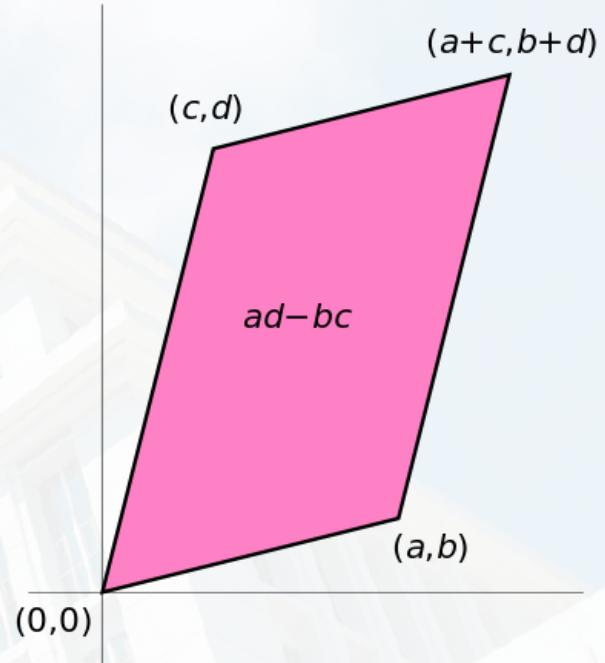


哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

Matrix Operations

- Determinant
 - $\det(\mathbf{A})$ returns a scalar
 - Represents area (or volume) of the parallelogram described by the vectors in the rows of the matrix
 - For $\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $\det(\mathbf{A}) = ad - bc$
 - Properties:
 - $\det(\mathbf{AB}) = \det(\mathbf{BA})$
 - $\det(\mathbf{A}^{-1}) = \frac{1}{\det(\mathbf{A})}$
 - $\det(\mathbf{A}^T) = \det(\mathbf{A})$
 - $\det(\mathbf{A}) = 0 \Leftrightarrow \mathbf{A}$ is singular



Matrix Operations

- Trace

$\text{tr}(\mathbf{A}) = \text{sum of diagonal elements}$

$$\text{tr}\left(\begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix}\right) = 1 + 7 = 8$$

- Invariant to a lot of transformations, so it's used sometimes in proofs. (Rarely in this class though.)
- Properties:

$$\text{tr}(\mathbf{AB}) = \text{tr}(\mathbf{BA})$$

$$\text{tr}(\mathbf{A} + \mathbf{B}) = \text{tr}(\mathbf{A}) + \text{tr}(\mathbf{B})$$



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

Special Matrices

- Identity matrix \mathbf{I}
 - Square matrix, 1's along diagonal, 0's elsewhere
 - $\mathbf{I} \cdot [\text{another matrix}] = [\text{that matrix}]$
- Diagonal matrix
 - Square matrix with numbers along diagonal, 0's elsewhere
 - A diagonal \cdot [another matrix] scales the rows of that matrix

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

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 2.5 \end{bmatrix}$$



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

Special Matrices

- Symmetric matrix

$$\mathbf{A}^T = \mathbf{A}$$

$$\begin{bmatrix} 1 & 2 & 5 \\ 2 & 1 & 7 \\ 5 & 7 & 1 \end{bmatrix}$$

- Skew-symmetric matrix

$$\mathbf{A}^T = -\mathbf{A}$$

$$\begin{bmatrix} 1 & -2 & -5 \\ 2 & 1 & -7 \\ 5 & 7 & 1 \end{bmatrix}$$



哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

Outline

- Vectors and matrices
 - Basic Matrix Operations
 - Special Matrices
- Transformation Matrices
 - Homogeneous coordinates
 - Translation
- Matrix inverse
- Matrix rank
- Singular Value Decomposition (SVD)
 - Use for image compression
 - Use for Principal Component Analysis (PCA)
 - Computer algorithm

Transformation

- Matrices can be used to transform vectors in useful ways, through multiplication: $x' = Ax$
- Simplest is scaling:

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

(Verify to yourself that the matrix multiplication works out this way)

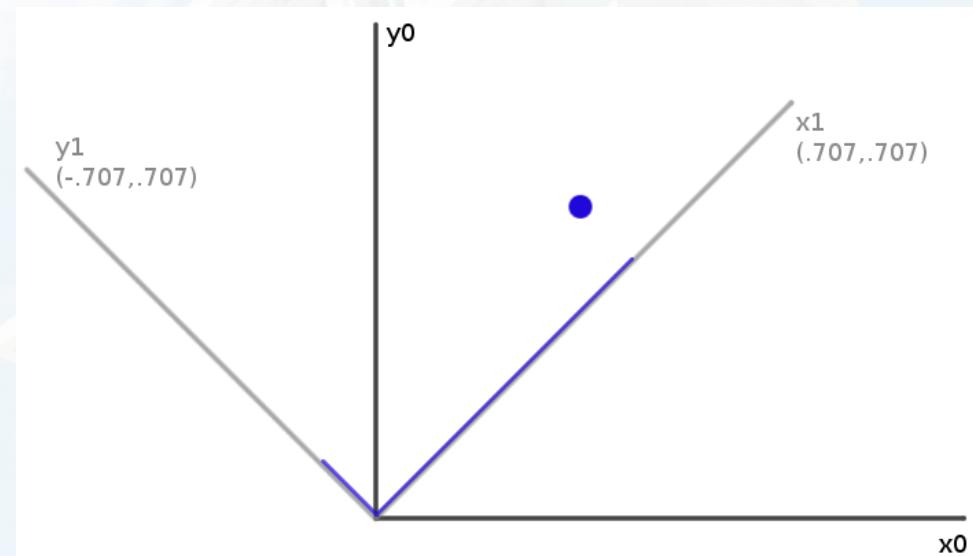


哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

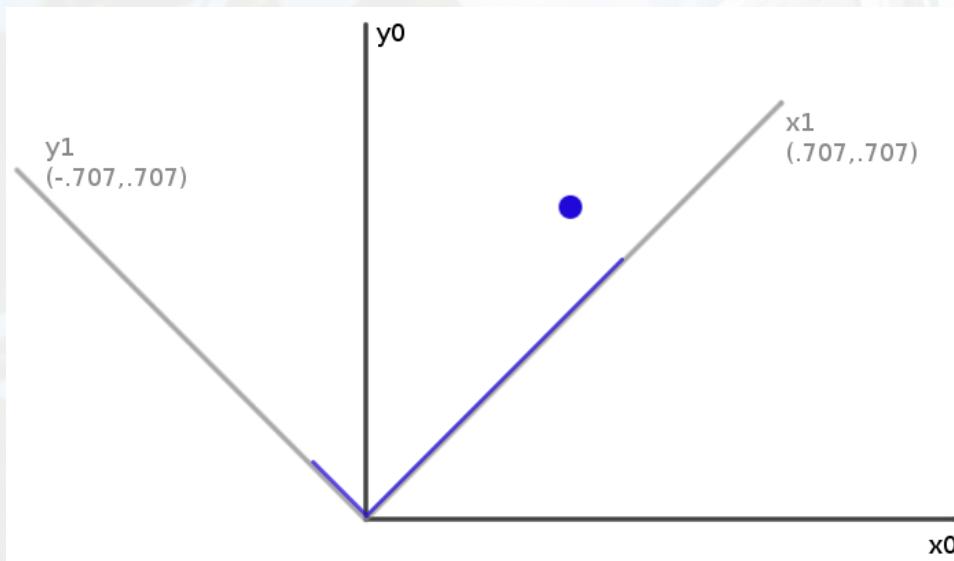
Rotation

- How can you convert a vector represented in frame “0” to a new, rotated coordinate frame “1”?
- Remember what a vector is: [component in direction of the frame’s x axis, component in direction of y axis]



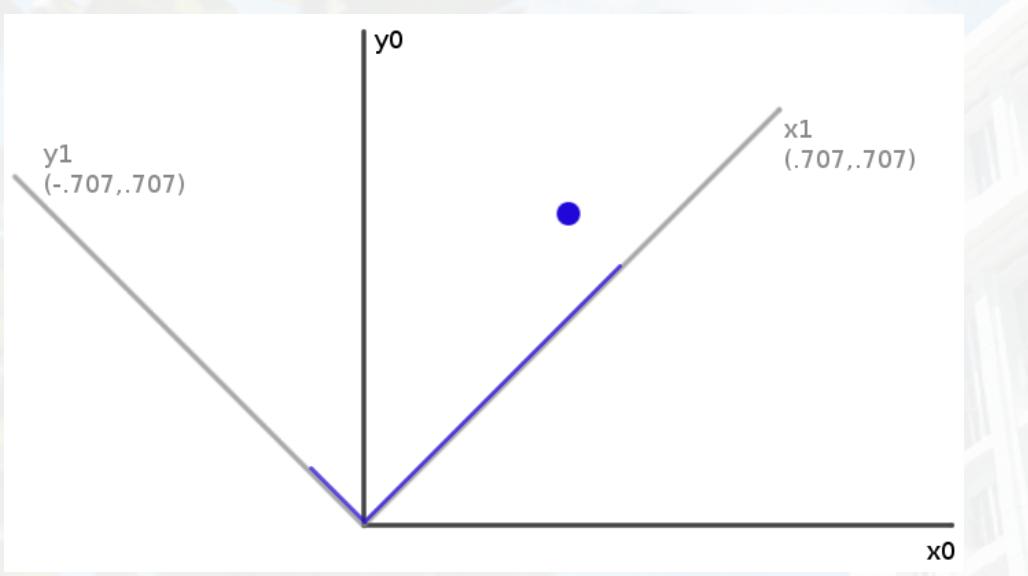
Rotation

- So to rotate it we must produce this vector: [component in direction of **new x axis**, component in direction of **new y axis**]
- We can do this easily with dot products!
- New x coordinate is [original vector] **dot** [the new x axis]
- New y coordinate is [original vector] **dot** [the new y axis]



Rotation

- Insight: this is what happens in a matrix*vector multiplication
 - Result x coordinate is [original vector] **dot** [matrix row 1]
 - So matrix multiplication can rotate a vector p:



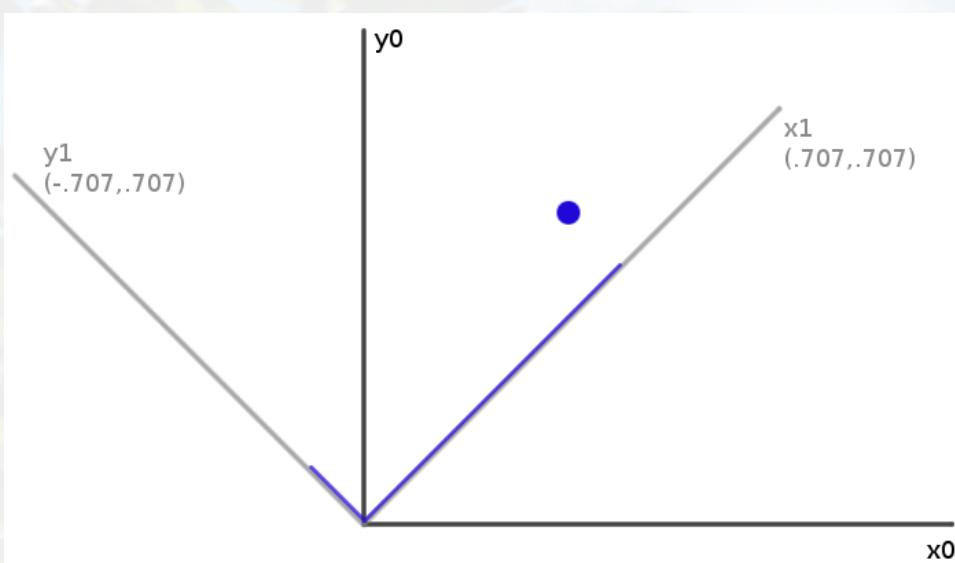
$R \times p =$
rotated p'

$$R = \begin{bmatrix} .707 & .707 \\ -.707 & .707 \end{bmatrix}$$
$$\begin{bmatrix} p \\ px \\ py \end{bmatrix} \rightarrow \begin{bmatrix} p \\ px' \\ py' \end{bmatrix}$$



Rotation

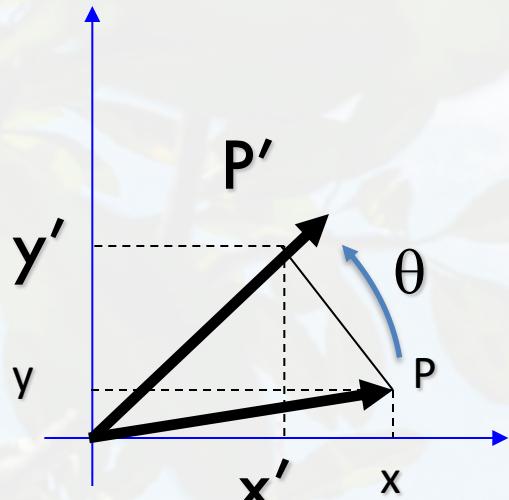
- Suppose we express a point in a coordinate system which is rotated left
- If we use the result in the **same** coordinate system, we have rotated the point right



- Thus, rotation matrices can be used to rotate vectors. We'll usually think of them in that sense-- as operators to rotate vectors

2D Rotation Matrix Formula

Counter-clockwise rotation by an angle θ



$$x' = \cos \theta x - \sin \theta y$$

$$y' = \cos \theta y + \sin \theta x$$

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

$$\mathbf{P}' = \mathbf{R} \mathbf{P}$$



Transformation Matrices

- Multiple transformation matrices can be used to transform a point:
 $p' = R_2 R_1 S p$
- The effect of this is to apply their transformations one after the other, from **right to left**.
- In the example above, the result is
 $(R_2 (R_1 (S p)))$
- The result is exactly the same if we multiply the matrices first, to form a single transformation matrix:
 $p' = (R_2 R_1 S) p$



Homogeneous system

- In general, a matrix multiplication lets us linearly combine components of a vector

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

- This is sufficient for scale, rotate, skew transformations.
- But notice, we can't add a constant! ☹

Homogeneous system

- The (somewhat hacky) solution? Stick a “1” at the end of every vector:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix}$$

- Now we can rotate, scale, and skew like before, AND translate (note how the multiplication works out, above)
- This is called “homogeneous coordinates”

Homogeneous system

- In homogeneous coordinates, the multiplication works out so the rightmost column of the matrix is a vector that gets added.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix}$$

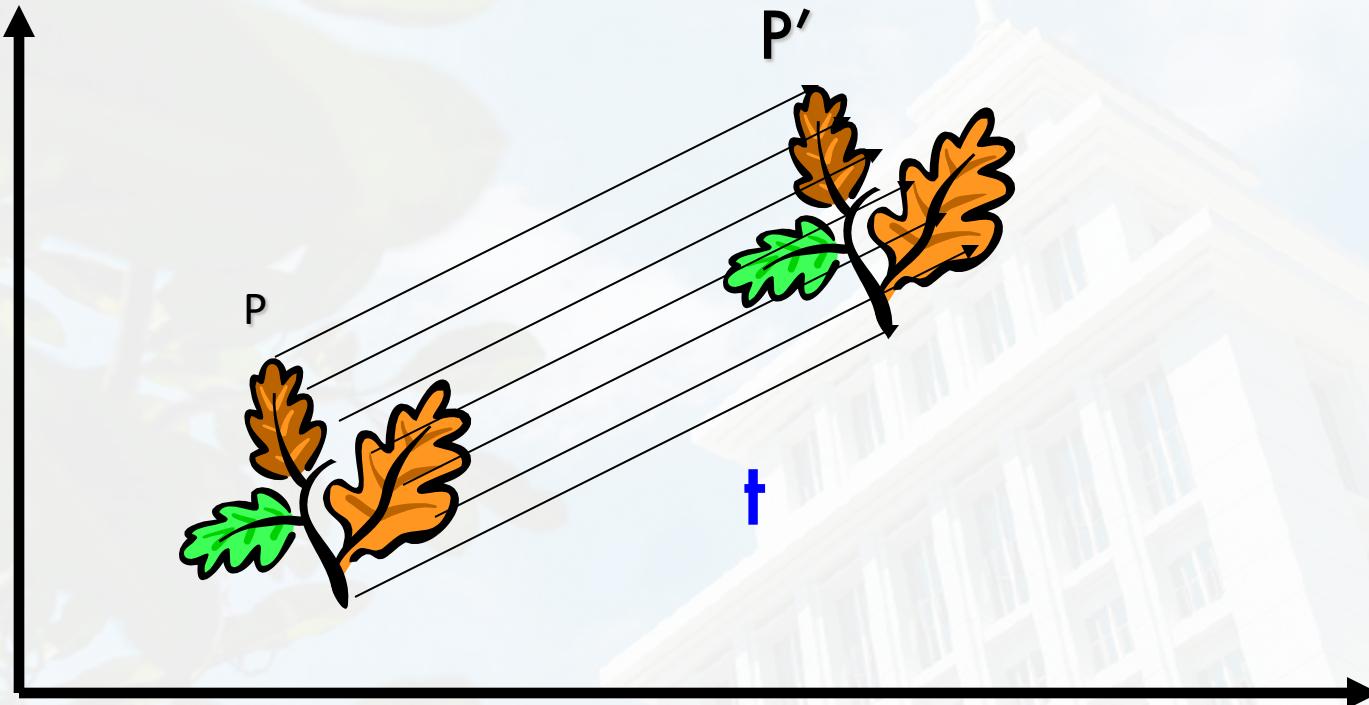
- Generally, a homogeneous transformation matrix will have a bottom row of [0 0 1], so that the result has a “1” at the bottom too.

Homogeneous system

- One more thing we might want: to divide the result by something
 - For example, we may want to divide by a coordinate, to make things scale down as they get farther away in a camera image
 - Matrix multiplication can't actually divide
 - So, **by convention**, in homogeneous coordinates, we'll divide the result by its last coordinate after doing a matrix multiplication

$$\begin{bmatrix} x \\ y \\ 7 \end{bmatrix} \Rightarrow \begin{bmatrix} x/7 \\ y/7 \\ 1 \end{bmatrix}$$

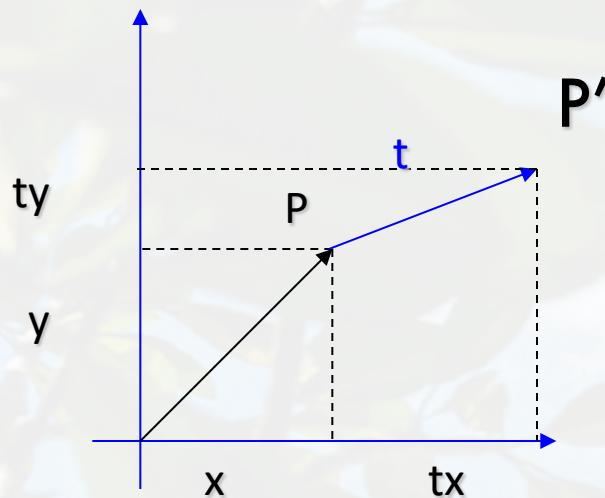
2D Translation



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

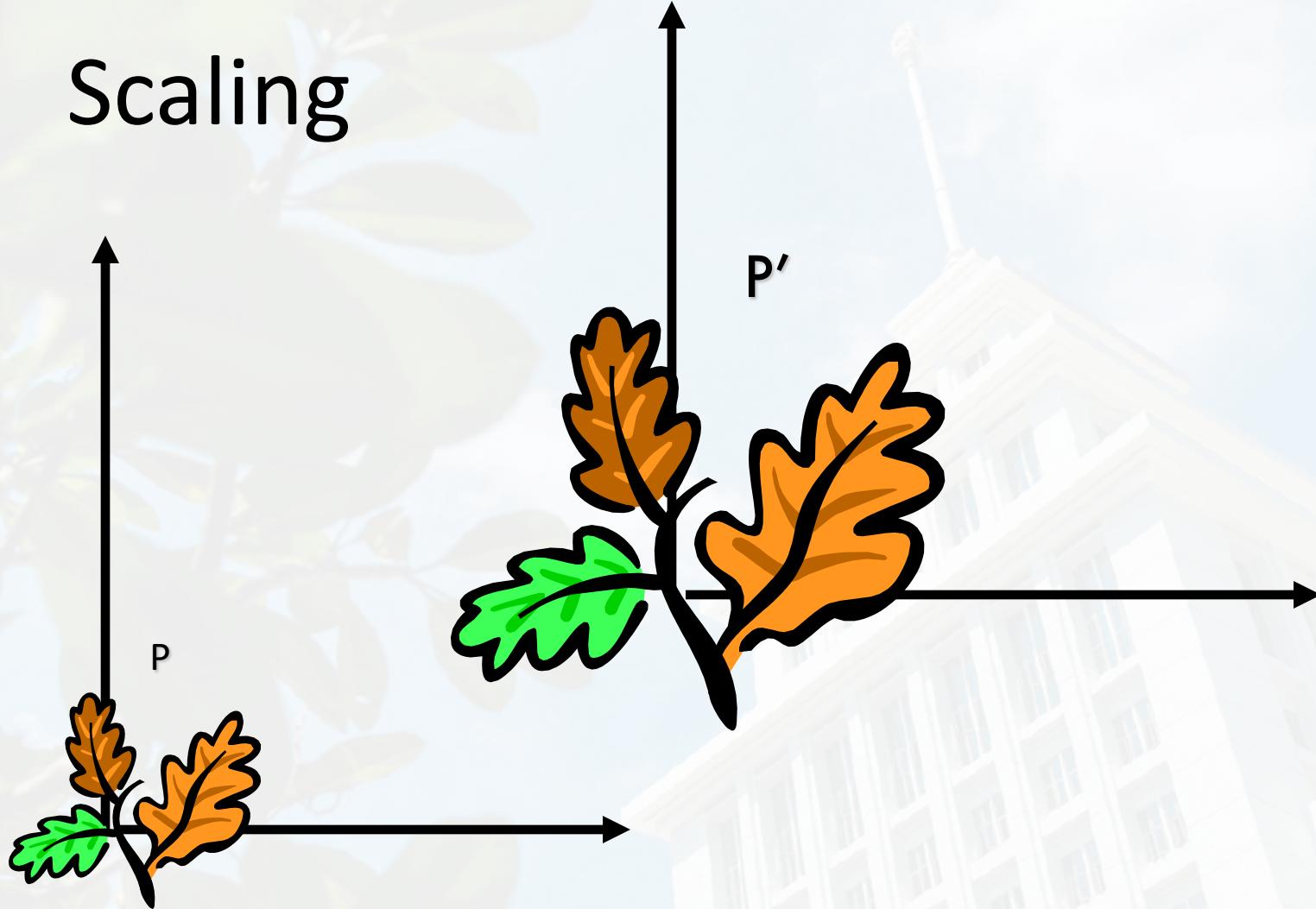
2D Translation using Homogeneous Coordinates



$$\begin{aligned}\mathbf{P} &= (x, y) \rightarrow (x, y, 1) \\ \mathbf{t} &= (t_x, t_y) \rightarrow (t_x, t_y, 1) \\ \mathbf{P}' &\rightarrow \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \cdot \mathbf{P} = \mathbf{T} \cdot \mathbf{P}\end{aligned}$$



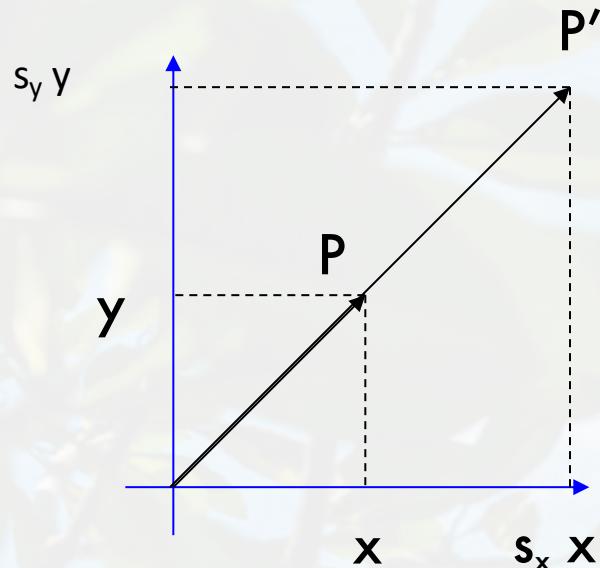
Scaling



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

Scaling Equation



$$P = (x, y) \rightarrow P' = (s_x x, s_y y)$$

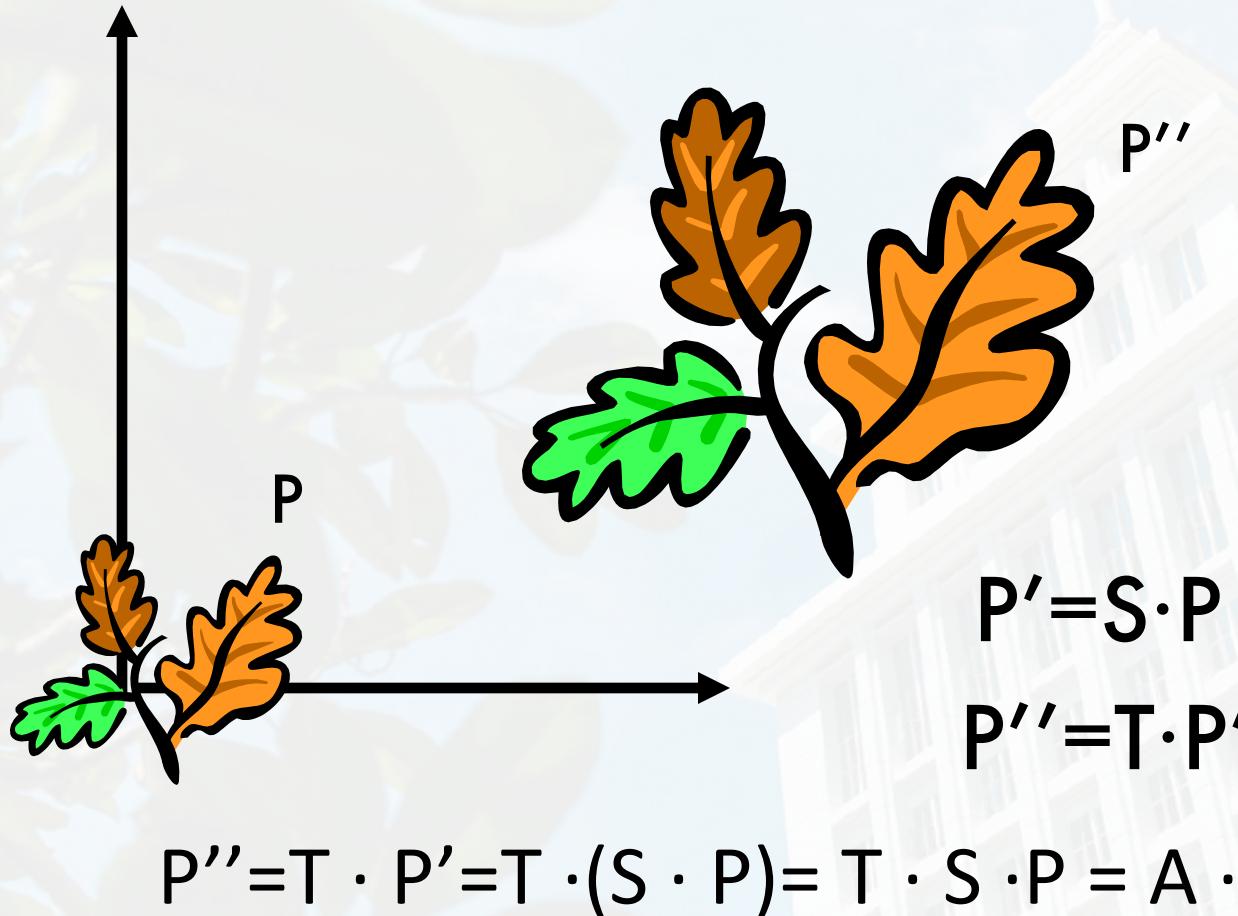
$$P = (x, y) \rightarrow (x, y, 1)$$

$$P' = (s_x x, s_y y) \rightarrow (s_x x, s_y y, 1)$$

$$P' \rightarrow \begin{bmatrix} s_x x \\ s_y y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}}_S \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} S' & 0 \\ 0 & 1 \end{bmatrix} \cdot P = S \cdot P$$



Scaling & Translating



Scaling & Translating

$$\mathbf{P}' = \mathbf{T} \cdot \mathbf{S} \cdot \mathbf{P} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$$
$$= \underbrace{\begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + t_x \\ s_y y + t_y \\ 1 \end{bmatrix} = \begin{bmatrix} s & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Translating & Scaling != Scaling & Translating

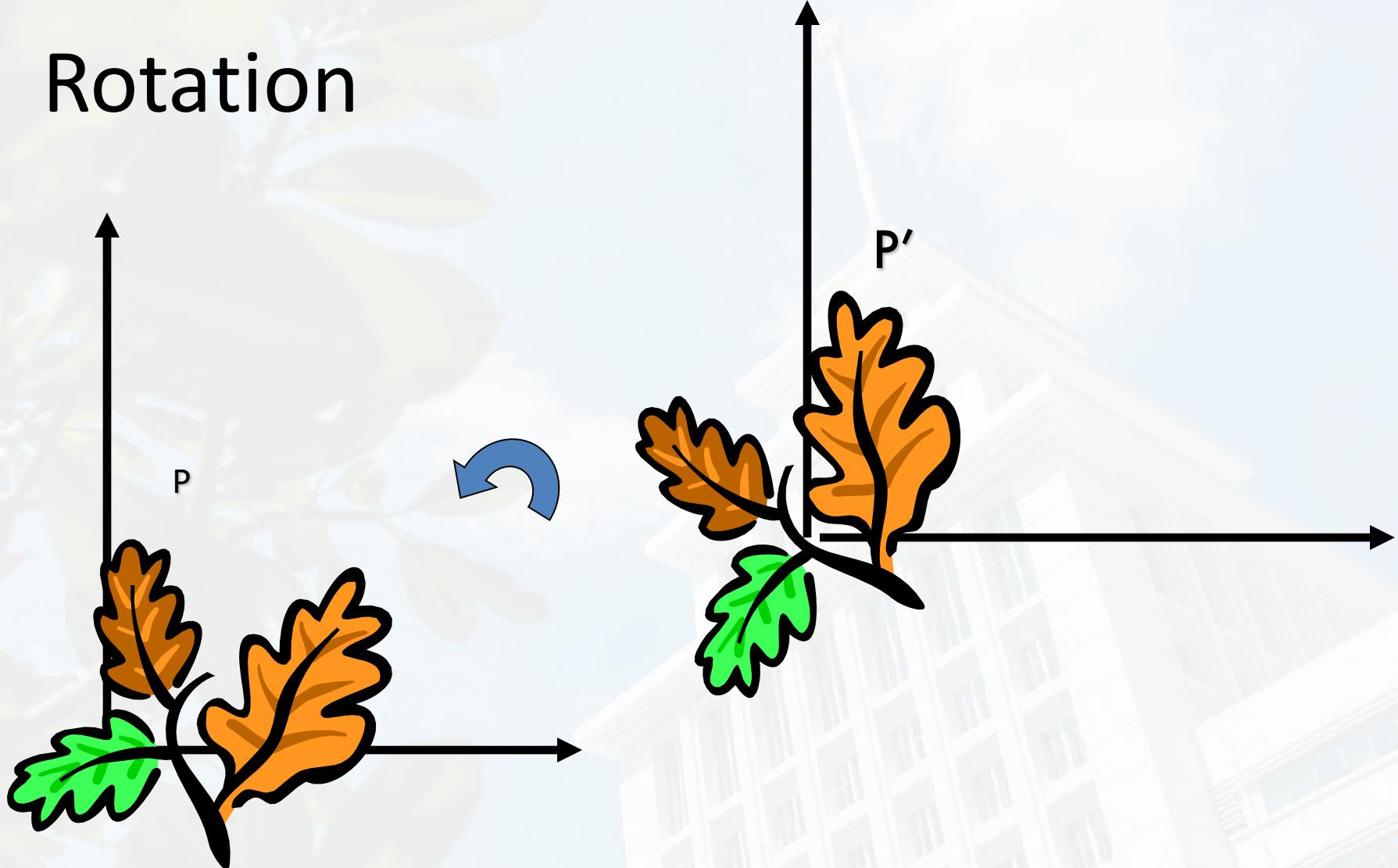
$$\mathbf{P}''' = \mathbf{T} \cdot \mathbf{S} \cdot \mathbf{P} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + t_x \\ s_y y + t_y \\ 1 \end{bmatrix}$$

$$\mathbf{P}''' = \mathbf{S} \cdot \mathbf{T} \cdot \mathbf{P} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$$

$$= \begin{bmatrix} s_x & 0 & s_x t_x \\ 0 & s_y & s_y t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + s_x t_x \\ s_y y + s_y t_y \\ 1 \end{bmatrix}$$



Rotation

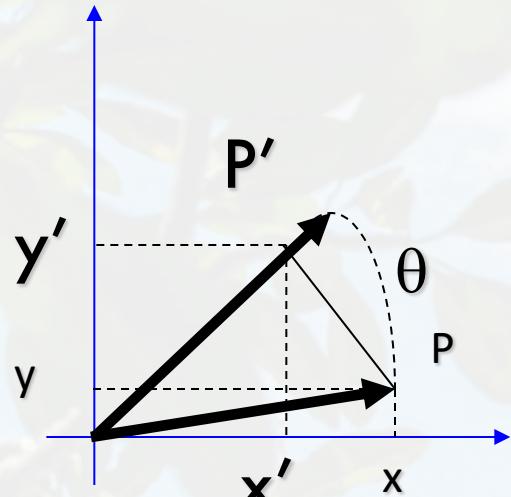


哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

Rotation Equations

Counter-clockwise rotation by an angle θ



$$x' = \cos \theta x - \sin \theta y$$

$$y' = \cos \theta y + \sin \theta x$$

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

$$\mathbf{P}' = \mathbf{R} \mathbf{P}$$



Rotation Matrix Properties

- Transpose of a rotation matrix produces a rotation in the opposite direction

$$\mathbf{R} \cdot \mathbf{R}^T = \mathbf{R}^T \cdot \mathbf{R} = \mathbf{I}$$

$$\det(\mathbf{R}) = 1$$

- The rows of a rotation matrix are always mutually perpendicular (a.k.a. orthogonal) unit vectors
 - (and so are its columns)

Properties

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

A 2D rotation matrix is 2x2

Note: R belongs to the category of *normal* matrices and satisfies many interesting properties:

$$\mathbf{R} \cdot \mathbf{R}^T = \mathbf{R}^T \cdot \mathbf{R} = \mathbf{I}$$

$$\det(\mathbf{R}) = 1$$



Rotation+ Scaling +Translation

$$\mathbf{P}' = (\mathbf{T} \mathbf{R} \mathbf{S}) \mathbf{P}$$

$$\mathbf{P}' = \mathbf{T} \cdot \mathbf{R} \cdot \mathbf{S} \cdot \mathbf{P} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \cos \theta & -\sin \theta & t_x \\ \sin \theta & \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$$

This is the form of the general-purpose transformation matrix

$$= \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} S & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \boxed{\begin{bmatrix} RS & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}$$



Outline

- Vectors and matrices
 - Basic Matrix Operations
 - Special Matrices
- Transformation Matrices
 - Homogeneous coordinates
 - Translation
- **Matrix inverse**
- Matrix rank
- Singular Value Decomposition (SVD)
 - Use for image compression
 - Use for Principal Component Analysis (PCA)
 - Computer algorithm

Inverse

- Given a matrix \mathbf{A} , its inverse \mathbf{A}^{-1} is a matrix such that $\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$
- E.g.
$$\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{3} \end{bmatrix}$$
- Inverse does not always exist. If \mathbf{A}^{-1} exists, \mathbf{A} is *invertible* or *non-singular*. Otherwise, it's *singular*.
- Useful identities, for matrices that are invertible:

$$(\mathbf{A}^{-1})^{-1} = \mathbf{A}$$

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$$

$$\mathbf{A}^{-T} \triangleq (\mathbf{A}^T)^{-1} = (\mathbf{A}^{-1})^T$$



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

Matrix Operations

- Pseudoinverse
 - Say you have the matrix equation $AX=B$, where A and B are known, and you want to solve for X
 - You could use MATLAB to calculate the inverse and premultiply by it: $A^{-1}AX=A^{-1}B \rightarrow X=A^{-1}B$
 - MATLAB command would be **inv(A)*B**
 - But calculating the inverse for large matrices often brings problems with computer floating-point resolution (because it involves working with very small and very large numbers together).
 - Or, your matrix might not even have an inverse.

Matrix Operations

- Pseudoinverse
 - Fortunately, there are workarounds to solve $AX=B$ in these situations. And MATLAB can do them!
 - Instead of taking an inverse, directly ask MATLAB to solve for X in $AX=B$, by typing **A\B**
 - MATLAB will try several appropriate numerical methods (including the pseudoinverse if the inverse doesn't exist)
 - MATLAB will return the value of X which solves the equation
 - If there is no exact solution, it will return the closest one
 - If there are many solutions, it will return the smallest one

Matrix Operations

- MATLAB example:

$$AX = B$$

$$A = \begin{bmatrix} 2 & 2 \\ 3 & 4 \end{bmatrix}, B = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

```
>> x = A\B  
x =  
    1.0000  
   -0.5000
```



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

Outline

- Vectors and matrices
 - Basic Matrix Operations
 - Special Matrices
- Transformation Matrices
 - Homogeneous coordinates
 - Translation
- Matrix inverse
- **Matrix rank**
- Singular Value Decomposition (SVD)
 - Use for image compression
 - Use for Principal Component Analysis (PCA)
 - Computer algorithm



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

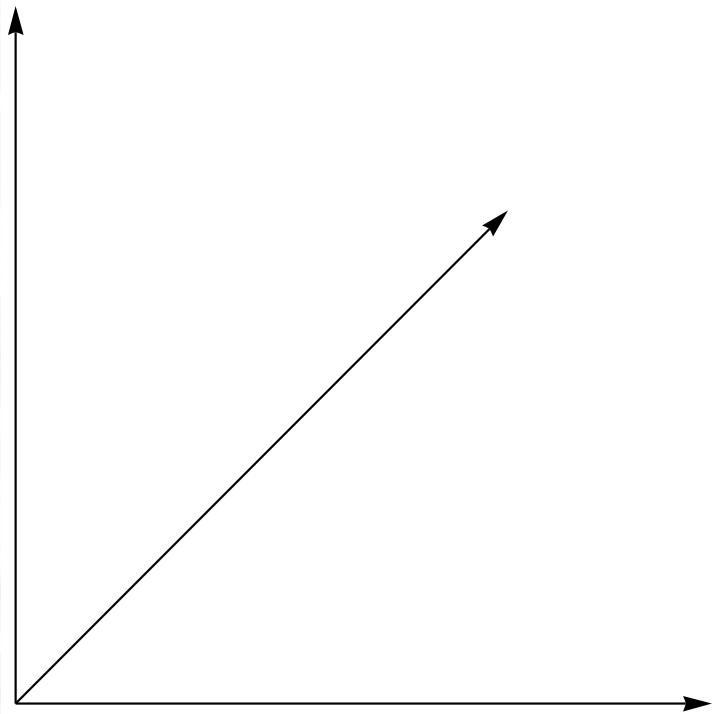
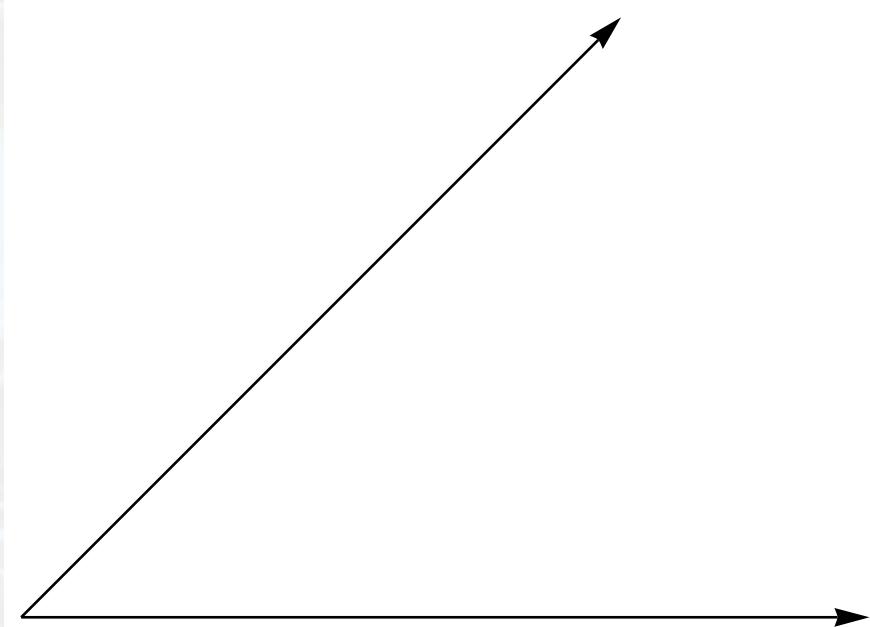
Linear independence

- Suppose we have a set of vectors $\mathbf{v}_1, \dots, \mathbf{v}_n$
- If we can express \mathbf{v}_1 as a linear combination of the other vectors $\mathbf{v}_2 \dots \mathbf{v}_n$, then \mathbf{v}_1 is linearly *dependent* on the other vectors.
 - The direction \mathbf{v}_1 can be expressed as a combination of the directions $\mathbf{v}_2, \dots, \mathbf{v}_n$. (E.g. $\mathbf{v}_1 = .7 \mathbf{v}_2 - .7 \mathbf{v}_4$)
- If no vector is linearly dependent on the rest of the set, the set is linearly *independent*.
 - Common case: a set of vectors $\mathbf{v}_1, \dots, \mathbf{v}_n$ is always linearly independent if each vector is perpendicular to every other vector (and non-zero)

Linear independence

Linearly independent set

Not linearly independent



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

Matrix rank

- Column/row rank

$\text{col-rank}(\mathbf{A})$ = the maximum number of linearly independent column vectors of \mathbf{A}

$\text{row-rank}(\mathbf{A})$ = the maximum number of linearly independent row vectors of \mathbf{A}

- Column rank always equals row rank

- Matrix rank

$$\text{rank}(\mathbf{A}) \triangleq \text{col-rank}(\mathbf{A}) = \text{row-rank}(\mathbf{A})$$



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

Matrix rank

- For transformation matrices, the rank tells you the dimensions of the output
- E.g. if rank of \mathbf{A} is 1, then the transformation

$$\mathbf{p}' = \mathbf{Ap}$$

maps points onto a line.

- Here's a matrix with rank 1:

$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + y \\ 2x + 2y \end{bmatrix}$$

All points get mapped to the line $y=2x$



Matrix rank

- If an $m \times m$ matrix is rank m , we say it's "full rank"
 - Maps an $m \times 1$ vector uniquely to another $m \times 1$ vector
 - An inverse matrix can be found
- If rank $< m$, we say it's "singular"
 - At least one dimension is getting collapsed. No way to look at the result and tell what the input was
 - Inverse does not exist
- Inverse also doesn't exist for non-square matrices

Outline

- Vectors and matrices
 - Basic Matrix Operations
 - Special Matrices
- Transformation Matrices
 - Homogeneous coordinates
 - Translation
- Matrix inverse
- Matrix rank
- Singular Value Decomposition (SVD)
 - Use for image compression
 - Use for Principal Component Analysis (PCA)
 - Computer algorithm



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

Singular Value Decomposition (SVD)

- There are several computer algorithms that can “factor” a matrix, representing it as the product of some other matrices
- The most useful of these is the Singular Value Decomposition.
- Represents any matrix \mathbf{A} as a product of three matrices: $\mathbf{U}\Sigma\mathbf{V}^T$
- MATLAB command: `[U,S,V]=svd(A)`

Singular Value Decomposition (SVD)

$$\mathbf{U}\Sigma\mathbf{V}^T = \mathbf{A}$$

- Where \mathbf{U} and \mathbf{V} are rotation matrices, and Σ is a scaling matrix. For example:

$$U \Sigma V^T = A$$
$$\begin{bmatrix} -.40 & .916 \\ .916 & .40 \end{bmatrix} \times \begin{bmatrix} 5.39 & 0 \\ 0 & 3.154 \end{bmatrix} \times \begin{bmatrix} -.05 & .999 \\ .999 & .05 \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ 1 & 5 \end{bmatrix}$$



Singular Value Decomposition (SVD)

- Beyond 2D:
 - In general, if \mathbf{A} is $m \times n$, then \mathbf{U} will be $m \times m$, Σ will be $m \times n$, and \mathbf{V}^T will be $n \times n$.
 - (Note the dimensions work out to produce $m \times n$ after multiplication)

$$\begin{bmatrix} U \\ -.39 & -.92 \\ -.92 & .39 \end{bmatrix} \times \begin{bmatrix} \Sigma \\ 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix} \times \begin{bmatrix} V^T \\ -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} = \begin{bmatrix} A \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Singular Value Decomposition (SVD)

- \mathbf{U} and \mathbf{V} are always rotation matrices.
 - Geometric rotation may not be an applicable concept, depending on the matrix. So we call them “unitary” matrices – each column is a unit vector.
- Σ is a diagonal matrix
 - The number of nonzero entries = rank of \mathbf{A}
 - The algorithm always sorts the entries high to low

$$\begin{matrix} U \\ \left[\begin{matrix} -.39 & -.92 \\ -.92 & .39 \end{matrix} \right] \end{matrix} \times \begin{matrix} \Sigma \\ \left[\begin{matrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{matrix} \right] \end{matrix} \times \begin{matrix} V^T \\ \left[\begin{matrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{matrix} \right] \end{matrix} = \begin{matrix} A \\ \left[\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix} \right] \end{matrix}$$



SVD Applications

- We've discussed SVD in terms of geometric transformation matrices
- But SVD of an image matrix can also be very useful
- To understand this, we'll look at a less geometric interpretation of what SVD is doing



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

SVD Applications

$$U \begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix} \times \begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix} \times \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

- Look at how the multiplication works out, left to right:
- Column 1 of \mathbf{U} gets scaled by the first value from Σ .

$$U\Sigma \begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix} \times \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \quad A_{partial} \begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}$$

- The resulting vector gets scaled by row 1 of \mathbf{V}^T to produce a contribution to the columns of \mathbf{A}



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

SVD Applications

$$\begin{array}{c} U\Sigma \quad V^T \\ \left[\begin{matrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{matrix} \right] \times \left[\begin{matrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{matrix} \right] \quad A_{partial} \\ + \quad \left[\begin{matrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{matrix} \right] \times \left[\begin{matrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{matrix} \right] \quad A_{partial} \\ = \quad \left[\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix} \right] \end{array}$$

- Each product of (*column i of U*)·(*value i from Σ*)·(*row i of V^T*) produces a component of the final A .



SVD Applications

$$\begin{bmatrix} -3.67 & -8.8 \end{bmatrix} \begin{matrix} U\Sigma \\ \begin{bmatrix} -.71 & .30 & 0 \end{bmatrix} \end{matrix} \times \begin{bmatrix} V^T \\ \begin{bmatrix} -.42 & .81 & .41 \\ -.57 & .11 & -.82 \\ -.70 & -.58 & .41 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} A_{partial} \\ \begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} -3.67 & -8.8 \end{bmatrix} \begin{matrix} U\Sigma \\ \begin{bmatrix} -.71 & .30 & 0 \end{bmatrix} \end{matrix} \times \begin{bmatrix} V^T \\ \begin{bmatrix} -.42 & .81 & .41 \\ -.57 & .11 & -.82 \\ -.70 & -.58 & .41 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} A \\ \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \end{bmatrix}$$

- We're building \mathbf{A} as a linear combination of the columns of \mathbf{U}
- Using all columns of \mathbf{U} , we'll rebuild the original matrix perfectly
- But, in real-world data, often we can just use the first few columns of \mathbf{U} and we'll get something close (e.g. the first $A_{partial}$, above)

SVD Applications

$$\begin{bmatrix} -3.67 & U\Sigma & V^T \\ -8.8 & -.71 & 0 \end{bmatrix} \times \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} = \begin{bmatrix} 1.6 & A_{partial} \\ 3.8 & 2.1 \\ 6.2 & 2.6 \end{bmatrix}$$

$$\begin{bmatrix} -3.67 & U\Sigma & V^T \\ -8.8 & -.71 & 0 \end{bmatrix} \times \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} = \begin{bmatrix} -.6 & A_{partial} \\ .2 & -.1 \\ 0 & .4 \\ -.2 & -.2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & A \\ 4 & 2 \\ 5 & 3 \\ 6 & \end{bmatrix}$$

- We can call those first few columns of U the *Principal Components* of the data
- They show the major patterns that can be added to produce the columns of the original matrix
- The rows of V^T show how the *principal components* are mixed to produce the columns of the matrix

SVD Applications

$$U \begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix} \times \begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix} \times \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} = V^T \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

We can look at Σ to see that the first column has a large effect

while the second column has a much smaller effect in this example



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

Principal Component Analysis

$$\begin{matrix} U \\ \begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix} \end{matrix} \times \begin{matrix} \Sigma \\ \begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix} \end{matrix} \times \begin{matrix} V^T \\ \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \end{matrix} = \begin{matrix} A \\ \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \end{matrix}$$

- Remember, columns of \mathbf{U} are the *Principal Components* of the data: the major patterns that can be added to produce the columns of the original matrix
- One use of this is to construct a matrix where each column is a separate data sample
- Run SVD on that matrix, and look at the first few columns of \mathbf{U} to see patterns that are common among the columns
- This is called *Principal Component Analysis* (or PCA) of the data samples

Principal Component Analysis

- Often, raw data samples have a lot of redundancy and patterns
- PCA can allow you to represent data samples as weights on the principal components, rather than using the original raw form of the data
- By representing each sample as just those weights, you can represent just the “meat” of what’s different between samples.
- This minimal representation makes machine learning and other algorithms much more efficient



PCA for Images - Setup

- Suppose a data set comprising N observations, each of M variables (dimensions).
- The aim: to reduce the dimensionality of the data so that each observation can be usefully represented with only L variables, $L < M$.
- Data are arranged as a set of N column data vectors, each representing a single observation of M variables: the n -th observations is a column vector.
- We thus have an $M \times N$ data matrix \mathbf{X} .

PCA for images

Let us consider a 321×261 image.



The image is considered as a very long 1D vector by concatenating image pixels column by column (or alternatively row by row), i. e. $321 \times 261 = 83781$.

The huge number 83781 is the dimensionality of our vector space.

The intensity variation is assumed in each pixel of the image.

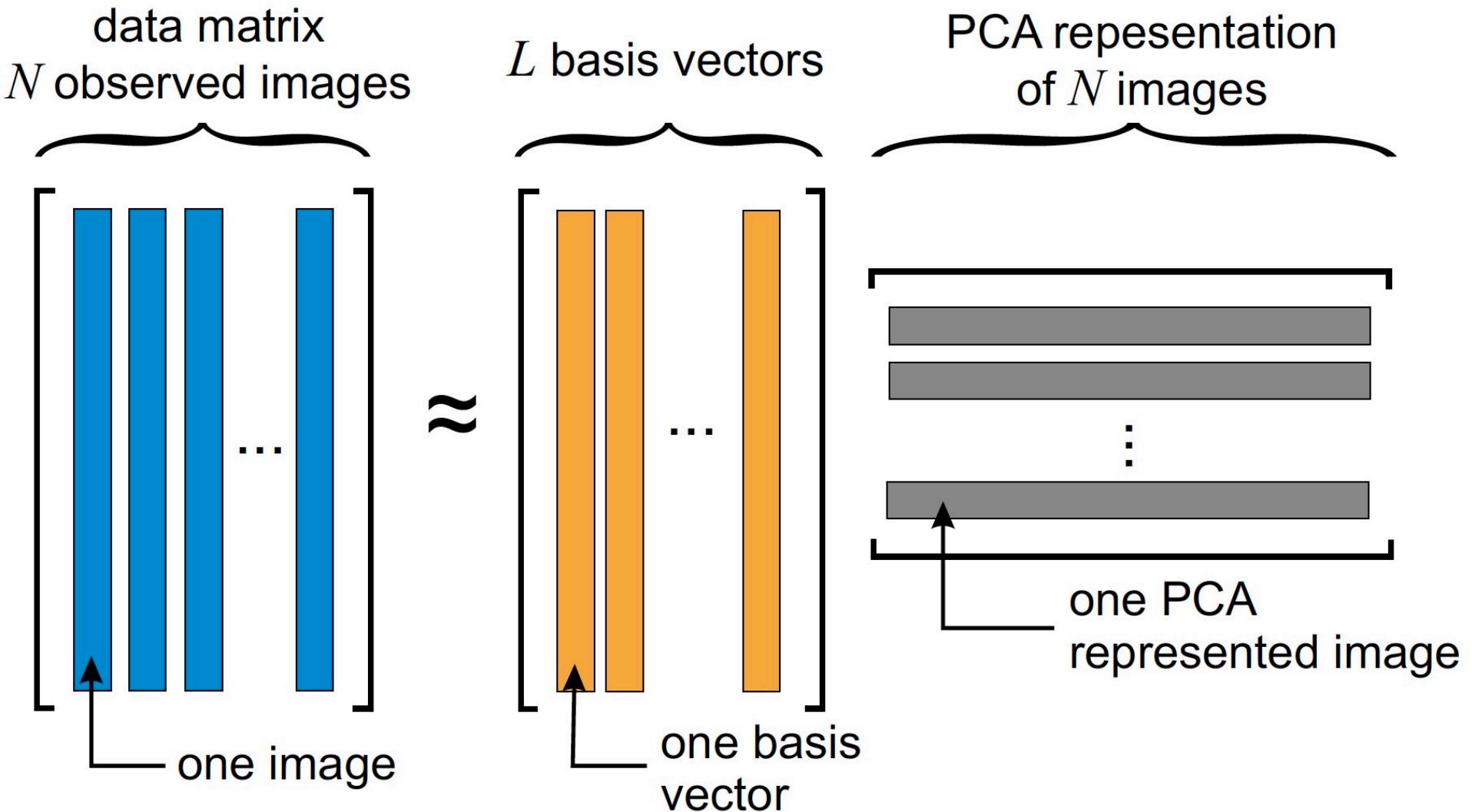
What if we have 32 instances of images



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

PCA: graphical illustration

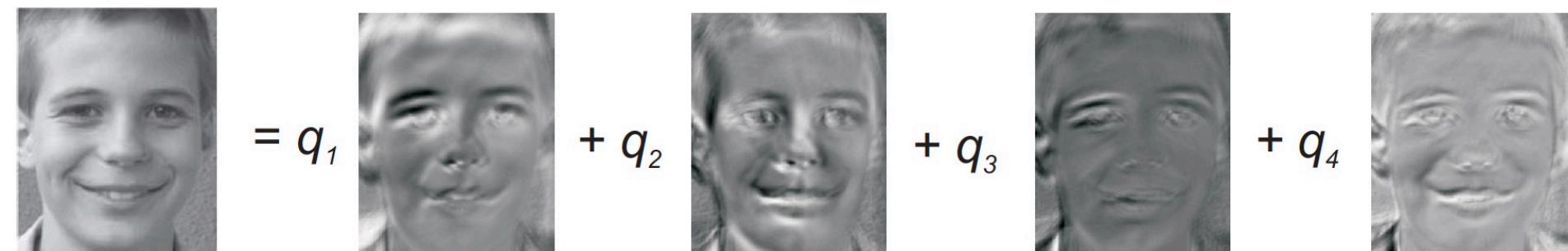


PCA for images

- We have 32 observations and 83781 unknowns!
- The number of principle components is less than or equal to 32.
- The eigen-vectors we derive are called eigen-images, after rearranging back from the 1D vector to a rectangular image.
- Let us perform the dimensionality reduction from 83781 to 4.

Approximation by 4 principal components only

- Reconstruction of the image from four basis vectors \mathbf{b}_i , $i = 1, \dots, 4$ which can be displayed as images.
- The linear combination was computed as $q_1\mathbf{b}_1 + q_2\mathbf{b}_2 + q_3\mathbf{b}_3 + q_4\mathbf{b}_4 = 0.078\mathbf{b}_1 + 0.062\mathbf{b}_2 - 0.182\mathbf{b}_3 + 0.179\mathbf{b}_4$.



Reconstruction fidelity, 4 components



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

Reconstruction fidelity, original



哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

PCA for Images - Drawbacks

- By rearranging pixels column by column to a 1D vector, relations of a given pixel to pixels in neighboring rows are not taken into account.
- Another disadvantage is in the global nature of the representation; small change or error in the input images influences the whole eigen-representation. However, this property is inherent in all linear integral transforms.

Addendum: How is SVD computed?

If you're interested, one computer algorithm to do it makes use of Eigenvectors

Eigenvector definition

- Suppose we have a square matrix \mathbf{A} . We can solve for vector x and scalar λ such that $\mathbf{Ax} = \lambda x$
- In other words, find vectors where, if we transform them with \mathbf{A} , the only effect is to scale them with no change in direction.
- These vectors are called eigenvectors (German for “self vector” of the matrix), and the scaling factors λ are called eigenvalues
- An $m \times m$ matrix will have $\leq m$ eigenvectors where λ is nonzero

Finding eigenvectors

- Computers can find an x such that $Ax = \lambda x$ using this iterative algorithm:
 - $x = \text{random unit vector}$
 - while(x hasn't converged)
 - $x = Ax$
 - normalize x
- x will quickly converge to an eigenvector
- Some simple modifications will let this algorithm find all eigenvectors

Finding SVD

- Eigenvectors are for square matrices, but SVD is for all matrices
- To do $\text{svd}(A)$, computers can do this:
 - Take eigenvectors of AA^T (matrix is always square).
 - These eigenvectors are the columns of \mathbf{U} .
 - Square root of eigenvalues are the singular values (the entries of Σ).
 - Take eigenvectors of A^TA (matrix is always square).
 - These eigenvectors are columns of \mathbf{V} (or rows of \mathbf{V}^T)

Finding SVD

- Moral of the story: SVD is fast, even for large matrices
- It's useful for a lot of stuff
- There are also other algorithms to compute SVD or part of the SVD