20-10-2022

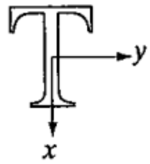
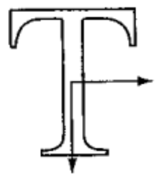
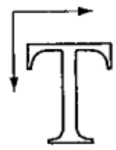
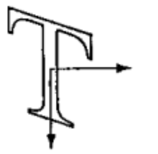
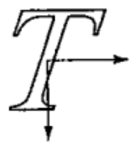
Le possibili operazioni che si fanno fra le matrici di un'immagine sono somma e prodotto:

- **Dalla somma fra 2 matrici** raster (immagini raster) posso ottenere la matrice risultante. Con la somma matriciale non ottengo un un risultato "interessante" perchè **non ottengo l'unione** fra le due immagini raster.
- Il **prodotto**, invece, mi da **un risultato molto più interessante**. Esso è riga per colonna. Per eseguire tale moltiplicazione matriciale è necessario che le colonne della prima matrice siano uguali al numero di righe della seconda matrice. Per esempio: 4x3 e 3x7 sono moltiplicabili perchè **i valori "medi" delle dimensioni sono uguali**

OPERAZIONI AFFINI

Presa un'immagine, un'**operazione affine** lavora solo sulla **posizione dei singoli pixel**. Serve per spostare i pixel nella nuova immagine. Non cambiano l'aspetto dell'immagine ma solo la posizione dei pixel e solitamente si fa sempre con le 3x3.

> Sostanzialmente si calcola, pixel per pixel, la nuova posizione che l'immagine deve assumere.

Nome della trasformazione	Matrice affine, T	Equazioni delle coordinate	Esempio
Identità	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v$ $y = w$	
Ridimensionamento	$\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = c_x v$ $y = c_y w$	
Rotazione	$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v\cos\theta - w\sin\theta$ $y = v\cos\theta + w\sin\theta$	
Traslazione	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$	$x = v + t_x$ $y = w + t_y$	
Distorsione verticale	$\begin{bmatrix} 1 & 0 & 0 \\ s_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v + s_v w$ $y = w$	
Distorsione orizzontale	$\begin{bmatrix} 1 & s_h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v$ $y = s_h v + w$	

Procedura:

- Prendo un punto nella matrice di coordinate generiche (v, w) e le metto in un **vettore riga** e con questo si vuole fare un'operazione;
- L'operazione vale per un solo pixel;
- Il numero di pixel non raddoppia perchè non aumento o diminuisco i pixel;
- **Scorro** tutti i pixel della matrice e dico dove andare a posizionare i nuovi pixel mediante le operazioni affini;
- Non è detto che nella matrice di output io abbia riempito tutti pixel.

Identità

$$[v \quad w \quad ?] \quad \overset{T}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}$$

Nel posto vuoto del vettore riga (?) metto 1 ottenendo:

$$[v \quad w \quad 1]$$

così da poter fare il prodotto riga per colonna.
Il risultato ci da una posizione.

$$[v \quad w \quad 1] \cdot T = [x \quad y \quad 1]$$

Dove:

- **x** è il risultato della nuova riga del nuovo pixel
- **y** è il risultato della nuova colonna del nuovo pixel
- **T** è la matrice
  Quindi le nuove coordinate che ottengo si possono scrivere come sistema di equazioni:

$$\begin{cases} x = v \\ y = w \end{cases}$$

Zooming (riscalaggio)

Con questa operazione si possono creare **pixel vuoti** ma che poi verranno sistemati con **l'interpolazione**.

$$\begin{bmatrix} v & w & 1 \end{bmatrix} \quad \begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Dove:

- **cx** e **cy** sono le **costanti** che permettono di **riscalare**
- cx lungo le **coordinate x** (righe)
- cy lungo le **coordinate y** (colonne).
- se entrambe le costanti **sono uguali** a 2 allora raddoppiamo l'immagine (zoom);
- se $cx > 1$ o $cy > 1$ allora l'**immagine aumenta** su x o y
- se $0 < cx < 1$ o $0 < cy < 1$ allora l'**immagine diminuisce** su y o su x

  Risultato:

  $$\begin{bmatrix} v & w & 1 \end{bmatrix} \cdot T = [c_x v, c_y w, 1]$$

  Output:

  $$\begin{cases} x = c_x \cdot v \\ y = c_y \cdot w \end{cases}$$

Rotazione

Anche in questo caso si possono creare pixel vuoti ma si sistema con l'interpolazione.
Ruota la matrice prendendo il **perno** l'angolo in alto a sinistra, quindi l'**origine degli assi**.

> Una rotazione di 30° gradi vuol dire che ruota in senso antiorario.

θ sarà l'angolo di rotazione

$$\begin{bmatrix} v & w & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \operatorname{sen}\theta & 0 \\ -\operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Risultato:

$$\begin{bmatrix} v & w & 1 \end{bmatrix} \cdot T = [v\cos\theta - w\operatorname{sen}\theta,\ v\operatorname{sen}\theta + w\cos\theta,\ 1]$$

Output:

$$\begin{cases} x = v\cos\theta - w\operatorname{sen}\theta \\ y = v\operatorname{sen}\theta + w\cos\theta \end{cases}$$

Traslazione

$$[v \quad w \quad 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

Risultato:

$$[v \quad w \quad 1] \cdot T = [v+t_x, w+t_y, 1]$$

Output:

$$\begin{cases} x = v + t_x \\ y = w + t_y \end{cases}$$

- Si **sommano** se **tx** e **ty se sono positivi** e viceversa se sono negativi.

> ***In caso si diano origini a risultato in numeri negativi, pezzi di immagini lì non esistono più e si perdono i pixel.***
> Se traslo un'immagine verso sinistra fino a -324930, non la vedo più e quindi perdo pixel.
> Per sistemare tale problema si possono fare dei blocchi `if` per vedere se si presentano indici negativi e quindi evitare tali operazioni.

Shear (verticale)

$$[v \quad w \quad 1] \begin{bmatrix} 1 & 0 & 0 \\ S_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Risultato:

$$[v \quad w \quad 1] \cdot T = [v + sw, w, 1]$$

Output:

$$\begin{cases} x = v + sw \\ y = w \end{cases}$$

Shear (orizzontale)

$$[v \quad w \quad 1] \begin{bmatrix} 1 & sh & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Risultato:

$$[v \quad w \quad 1] \cdot T = [v, w + vsh, 1]$$

Output:

$$\begin{cases} x = v \\ y = w + v\,sh \end{cases}$$

FORWARD MAPPING

Il fenomeno che crea buchi all'interno della nuova immagine è detto Forward Mapping

- (v, w) è il pixel di input, (x, y) quello di output e T la matrice affine.
- Si applica l'operazione affine per ottenere il valore $[x\ y\ 1] = [v\ w\ 1] \times T$
- In questo caso **si fa scorrere l'immagine di input** e **per ogni pixel (v,w)** si calcola la posizione della nuova immagine (x,y)
- I risultati sono i seguenti:


buchi

INVERSE MAPPING

Vediamo bene la formula del forward mapping.

[x y 1] = [v w 1] * T

> Questa formula crea i buchi. Allora scorro la matrice di output e per ogni pixel ci si pone la domanda "**quale valore devo copiare dall'input**?".

Quindi, applicando l'inverse mapping, la formula diventa
$[v\ w\ 1] = [x\ y\ 1 \times inversa(T)]$

L'**output** allora è:

- $o(x, y) = A(v, w)$ con A matrice di partenza.

Test su MATLAB

```
A=rgb2gray(imread('lena.jpg'));
A=double(A);
figure,imshow(uint8(A));
[m,n]=size(A);
theta=-45;

B=zeros(size(A));
T=[cosd(theta) sind(theta) 0; -sind(theta) cosd(theta) 0; 0 0 1];
 %scorre l'immagine di input e si stabilisce in quale punto finiranno i
 %nostri pixel in output

for v=1:m
    for w=1:n
        vett=round([v w 1]*T);
        x=vett(1);
        y=vett(2);
        if (x>0 & x<=m) & (y>0 & y<=n)
            B(x,y)=A(v,w);
        end
    end
end

figure,imshow(uint8(B));
```

Osservazione:

- Le operazioni affini **non sono commutativi**.

Combinazione di operazioni

- In questo caso **faccio il prodotto riga per colonna** fra le matrici di cui devo eseguire le operazioni e **il risultato lo uso per fare le mie operazioni nella moltiplicazione per il vettore riga**.
- **L'ordine** del prodotto fra matrici **è importante** perchè **NON** si parla di operazioni commutative.