Lezione 2 12/03/2025

Matematica

La matematica è fondamentale: punti, vettori, versori, trasformazioni spaziali; per l'engine di rendering, l'engine fisico, AI, suoni 3D...

Punti

Bisogna rappresentare per esempio dove un personaggio si ritrova nella scena, o trovare il centro di una sfera.

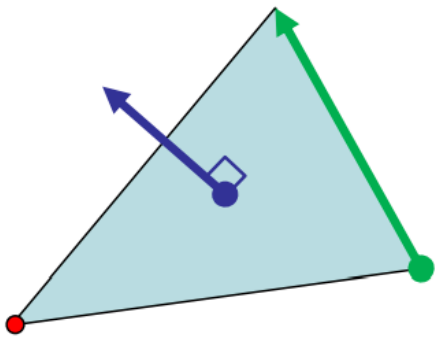
Vettori

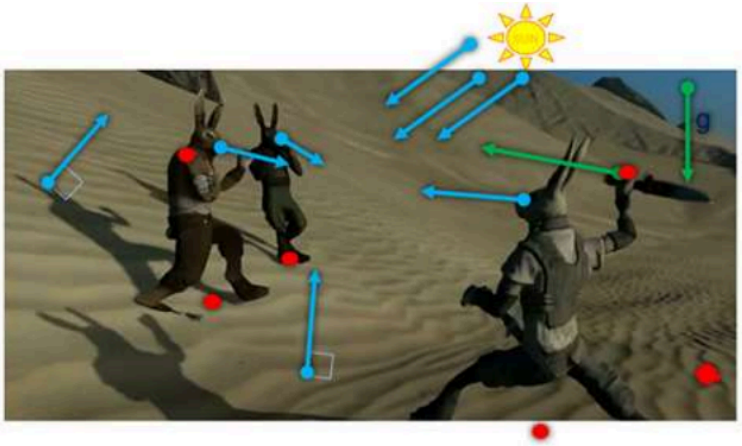
La differenza tra 2 punti, per esempio il movimento di un robot, la velocità, l'accelerazione. Possiamo immaginare queste cose come una freccia, dove **la lunghezza è importante**.

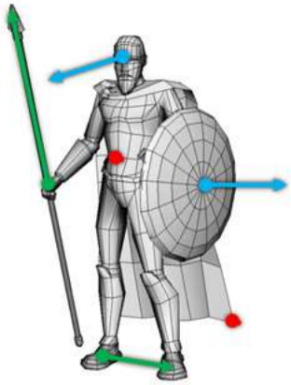
Versori

Rappresentano un vettore normalizzato, una **direzione nello spazio**. Per esempio la direzione in cui guarda un personaggio, la normale di un piano, un punto di una sfera. Possiamo immaginarlo come un **vettore di lunghezza unitaria** (unit length).

- A point
  - A vertex of the triangle
- A vector
  - A side of the triangle
- A versor
  - The normal of the triangle

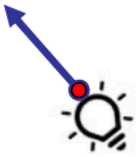


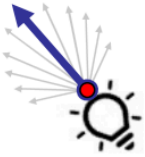




Un punto può rappresentare la posizione di un punto di vista (view), e un versore rappresenta la direzione della vista.

- Sounds
- (spot) lights





intuitive
know how to imagine it spatially, in 3D

operational
know how to compute from data

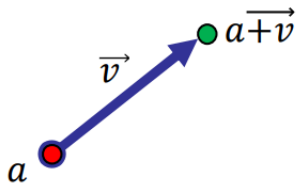
syntactic
know how to write down, (in paper or code)

plus rules:
know how to manipulate equations

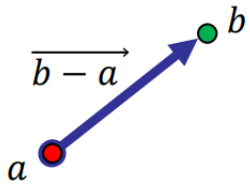
Oggi vediamo questi aspetti nel mondo dei videogames.

Recap algebra lineare

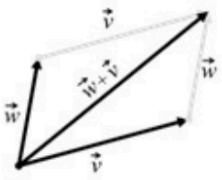
- Addition
  POINT + VECTOR = POINT

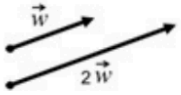


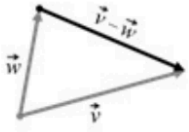
- Difference
  POINT – POINT = VECTOR



- addition (vector + vector = vector)
- product with a scalar (scaling)
  (vector * scalar = vector)
- therefore: interpolation
  $\text{mix}(\overrightarrow{v_0}, \overrightarrow{v_1}, t) = (1-t)\,\overrightarrow{v_0} + t\,\overrightarrow{v_1}$
- therefore: opposite (flip verse)
  (how to: multiply by – 1)
- therefore: difference
  (vector – vector = vector)



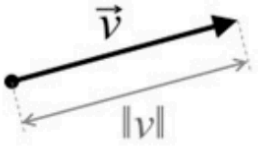


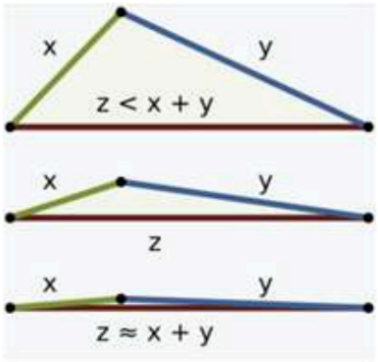


L'interpolazione la rivediamo dopo.

- Norm means length / magnitude /
  Euclidean norm / 2-norm
- distance between points:
  length of vector (a – b) = distance between a and b
- Rules: triangle inequality:

$$\|\vec{v}+\vec{w}\| \leq \|\vec{v}\| + \|\vec{w}\|$$

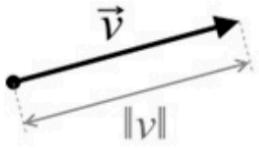




La norma di un vettore è la sua lunghezza.

Normalizzazione di un vettore: porta un vettore ad essere un versore (avrà norma 1)

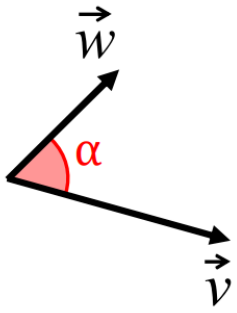
- Input: a vector
- Result: a versor
- how to: scale the vector by (1.0 / length)



Dot product: tra vettori, non sempre rappresentato da un dot, anche detto prodotto interno.

$$\vec{v} \cdot \vec{w} \qquad < \vec{v}, \vec{w} > \qquad (\vec{v}^{\mathrm{T}}\, \vec{w})$$

Dati 2 vettori, ci restituisce uno scalare.



$$\vec{v} \cdot \vec{w} \quad = \quad \|\vec{v}\| \cdot \|\vec{w}\| \cdot \cos(\alpha)$$

- dot is zero: vectors are orthogonal (or, either vector is degenerate)
- positive dot: acute angle
- negative dot: obtuse angle

valid with both vectors & versors

- versor dot vector: extension of vector along direction
- versor dot versor: cosine of angle
- versor dot versor: also, a similarity measure (in -1 +1)
- any vector dot itself: its squared length

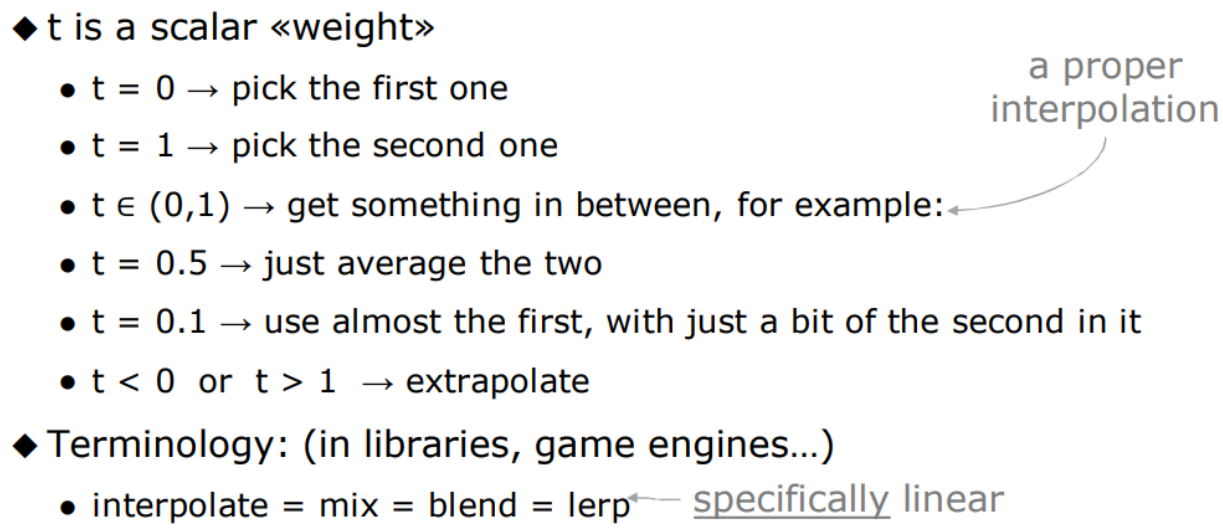
Vettori ortogonali: hanno direzioni perpendicolari.

Interpolazione

Possiamo interpolare una qualsiasi coppia di cose dello **stesso tipo**:

- mix(point , point , t) → point
- mix(vector , vector , t) → vector
- mix(versor , versor , t) → versor

Li mischiamo rispetto ad un parametro che ci dice quanto più stiamo verso uno dei due parametri. Vogliamo quindi trovare qualcosa nel mezzo dei due parametri.



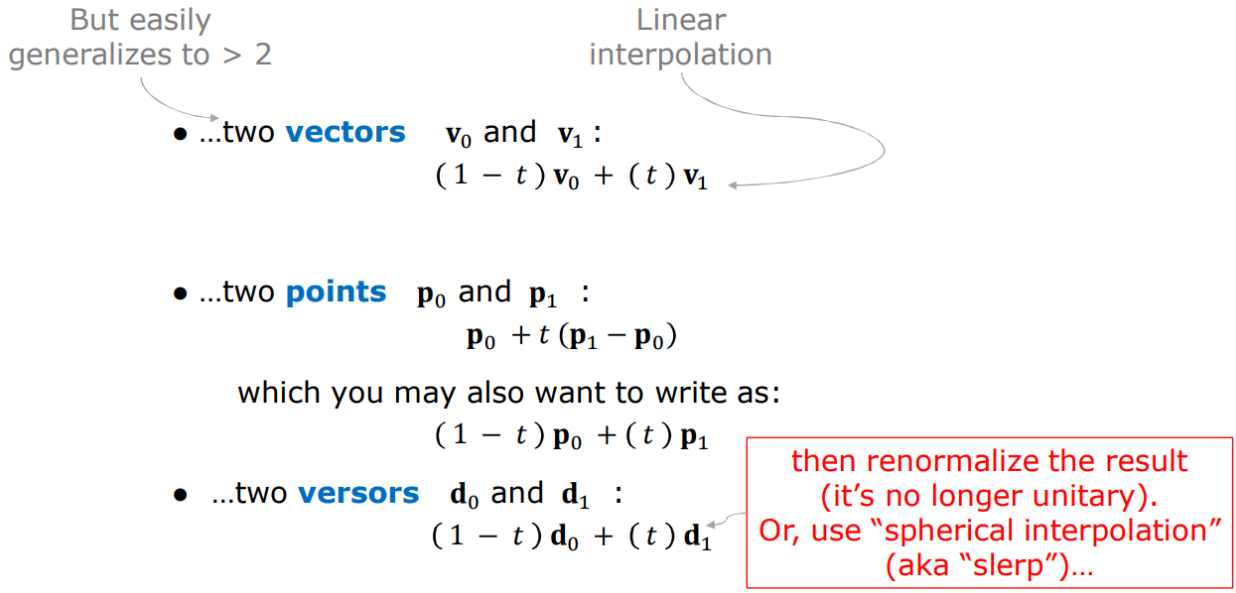
lerp è un tipo di interpolazione specificatamente lineare.

◆ Terminology:

- $a\,\mathbf{x} + b\,\mathbf{y}$: a **linear combination** of **x** and **y**
- if $a+b=1$ and $a,b \in [0,1]$: a **(linear) interpolation** of **x** and **y**
- if $a+b=1$ but $a,b \notin [0,1]$: a **(linear) extrapolation** of **x** and **y**
- a , b : the used **weights**
- $a + b = 1$: weights are a **partition of unity**

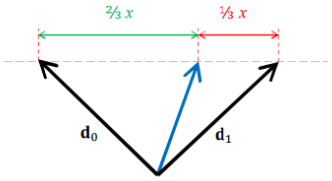
◆ Generalizes to > 2 objects $(a\,\mathbf{x} + b\,\mathbf{y} + c\,\mathbf{z})$

Però se facciamo **interpolazione tra 2 oggetti**, possiamo dare un unico peso **t** dato che l'altro è dato a differenza. L'interpolazione è spesso scritta nei linguaggi di programmazione come mix(x, y, t).



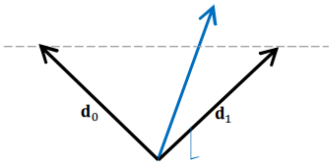
L'ultimo non è effettivamente un versore, deve essere normalizzato.

Linear interpolation:

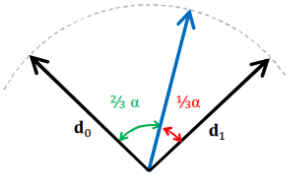


$\mathbf{d} = \text{lerp}(\mathbf{d}_0, \mathbf{d}_1, ⅔)$

Then, renormalize:



Spherical interpolation:



$\mathbf{d} = \text{slerp}(\mathbf{d}_0, \mathbf{d}_1, ⅔)$

Not the same result!

- But, close enough
- Even closer when:

  $\mathbf{d}_0$, $\mathbf{d}_1$ similar OR t close to ½

Is it worth the extra computation cost?

Al posto di considerare la combinazione lineare dei vettori (in questo caso versori), possiamo andare a considerare l'angolo tra i due versori, e definire il versore interpolazione come uno dei versori che sta all'interno dell'angolo definito dai due versori.

Rimaniamo quindi con la lunghezza 1, perchè stiamo usando solo la direzione. Si usa slerp perchè la combinazione lineare di versori non garantisce di avere un versore.

I due risultati non sono esattamente uguali, ma sono molto simili, si preferisce slerp. è una questione di costo computazionale, la normalizzazione è un po' costosa.

- LERP + normalization:

$$(1 - t)\,\mathbf{d_0} + t\,\mathbf{d_1}$$

then re-normalize } aka "NLERP"

- or SLERP:

angle between $\mathbf{d}_0$ and $\mathbf{d}_1$

$$\frac{\sin((1 - t)\,\alpha)}{\sin(\alpha)}\mathbf{d}_0 + \frac{\sin(t\,\alpha)}{\sin(\alpha)}\mathbf{d}_1$$

- Applicable to any versor (unit vector) including 2D, 3D, and quaternions (see later)
- SLERP can even be used on general vectors:
  - Compute magnitudes of vectors
  - Compute directions of vectors (divide by magnitude, i.e., normalize)
  - new direction = SLERP of the directions (unit vectors)
  - new magnitude = LERP of the magnitudes (scalars)
  - multiply new dir with new mag to get the final result

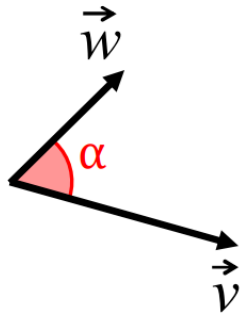
Cross product

$$\vec{v} \times \vec{w} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \times \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix} = \begin{pmatrix} v_y\ w_z - v_z\ w_y \\ v_z\ w_x - v_x\ w_z \\ v_x\ w_y - v_y\ w_x \end{pmatrix}$$

è quell'operazione che ci permette, dati due vettori, di ottenere un vettore.

rappresenta il prodotto tra i moduli dei vettori per il seno dell'angolo compreso.

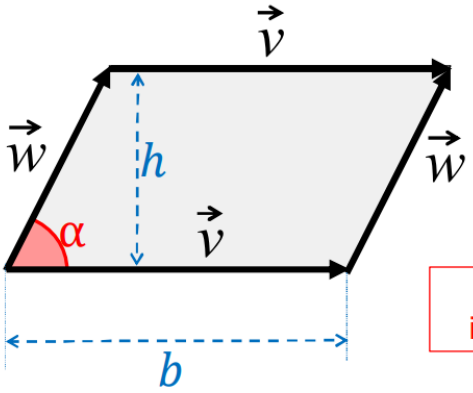
Dot product vs cross product:



$$\vec{v} \cdot \vec{w} = \|\vec{v}\| \cdot \|\vec{w}\| \cdot \cos(\alpha)$$

$$\|\vec{v} \times \vec{w}\| = \|\vec{v}\| \cdot \|\vec{w}\| \cdot \sin(\alpha)$$

Il cross product è definito solo in 3D. Tutto il resto visto fin ora generalizza.



cross product
is the parallelogram area

$$\|\vec{v} \times \vec{w}\| = \underbrace{\|\vec{v}\|}_{b} \cdot \underbrace{\|\vec{w}\| \cdot \sin(\alpha)}_{h}$$

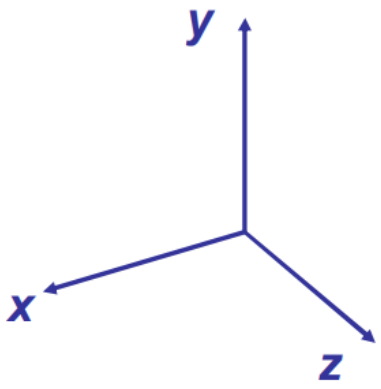
Il modulo del vettore cross product è l'area del parallelogramma costruito sui due vettori.

$$\vec{u} = \vec{v} \times \vec{w}$$

- Length of $\vec{u} = \|\vec{v}\| \cdot \|\vec{w}\| \cdot \sin(\alpha)$
- Direction of $\vec{u}$ = orthogonal to both $\vec{v}$ and $\vec{w}$
- Verse of $\vec{u}$ = use the «right-hand rule» or the «left-hand rule»
  - whicever hand you are using to imagine your vector space! (and reference frame)

La regola della mano destra dipende da come è stato definito il sistema di riferimento. Pollice sulla x, indice sulla y, la direzione di z ci deve dare quella del dito medio.

Questo per esempio ha come riferimento la mano sinistra invece:

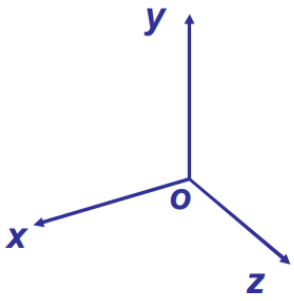


Gli assi sono l'insieme dei versori linearmente indipendenti (x,y,z) che stanno nello spazio n-dimensionale.

Ogni vettore v in questo spazio può essere espresso come una combinazione lineare di questi vettori, in esattamente 1 modo.

I pesi da dare a questa combinazione lineare sono le **coordinate dei punti nello spazio rappresentato da questi assi**.

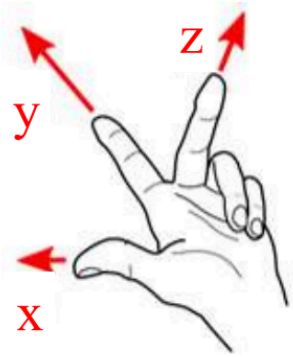
Se prendiamo gli assi e l'origine, questo è detto **sistema di riferimento dello spazio vettoriale**.



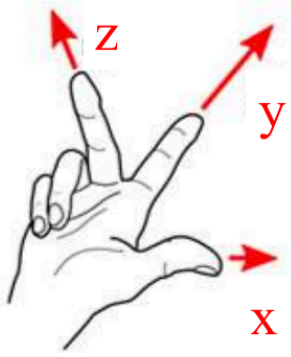
- The set of axes plus the origin is called the reference frame of the vector space
- *n* axes (vectors) + 1 origin (point)
- Any vector ***v*** :
  one linear comb. of the axes
- Any point ***p*** :
  origin + one linear comb. of axes

Gli assi non devono per forza essere ortogonali, basta che siano indipendenti. Quelli ortogonali sono quelli più comodi.

- They can be right- or left-handed



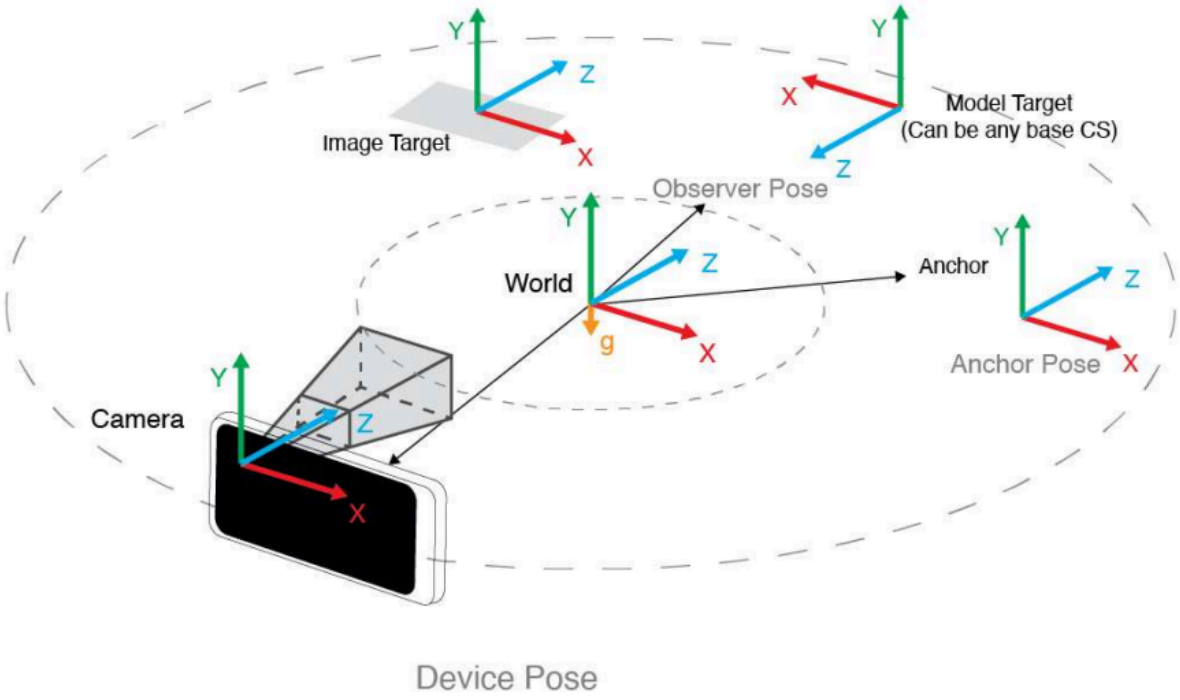
$x \times y = z$



$x \times y = z$
regardless!

Use the same hand to *imagine* a cross product

La regola ci dice qual è la direzione del cross product facendo la combinazione del primo e del secondo asse (x e y).



Nei videogames ogni scena, oggetto... ha uno spazio di riferimento. Quindi per operare negli oggetti dello spazio virtuale, bisogna trovare le trasformazioni corrette tra i sistemi di riferimento.

- **Unity:** left-handed: **X-right**, **Y-up**, **Z-forward**
- **Unreal:** left-handed: **X-forward**, **Y-right**, **Z-up**
- **3ds-Max:** right-handed, Z-up
- **Blender:** left-handed, Z-up
- **most VR systems:** right-handed, Y-up
- **OpenGL:** (clip space) right-handed, Y-up
- **DirectX:** (clip space) left-handed, Y-down

Unity e unreal hanno la regola della mano sinistra, ma poi cambiano il ruolo assegnato agli assi.

Note sulla programmazione

- n-tuple of scalar values (n is the dimension)
- for us (usually): n = 3 (at times, 2 or 4)
- they are the Cartesian coordinates of the point/vector
- e.g.:

```
class Vector3 {
  // fields:
  float x, y, z;

  // methods:
  …
}
```

Questa è una classe che esiste in Unity.

La stessa struttura può essere usata per rappresentare punti, vettori o versori, ma anche altro.

Esempio:

- Concept ("on paper"): $\mathbf{p} = \mathbf{p} + k\,\hat{\mathbf{d}}$

- Code:
  - Data types:

```
Point3D dragonPos = …;
Versor3D dir = …;
float k = …;
```

  - Beginner's style code:

```
dragonPos.x = dragonPos.x + dir.x * k ;
dragonPos.y = dragonPos.y + dir.y * k ;
dragonPos.z = dragonPos.z + dir.z * k ;
```

  - What you should do:

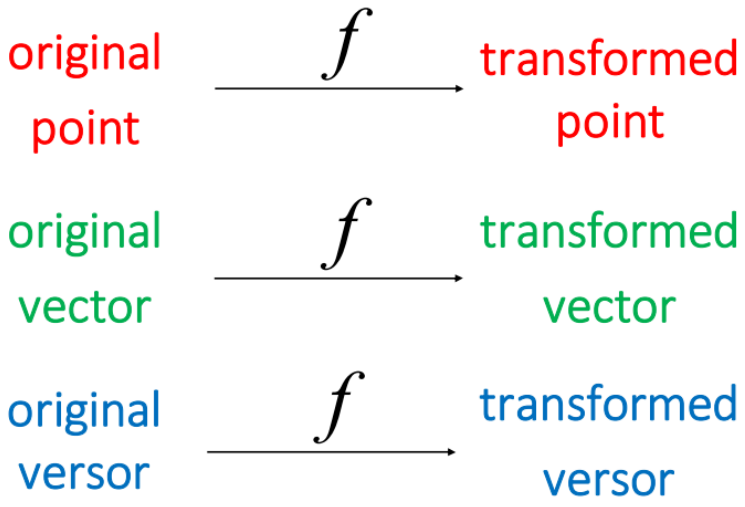
```
dragonPos += dir * k ;
```

Operazione di aggiornamento del punto p di modo che diventi se stesso + k volte lo spostamento in una direzione (d è un versore). In unity si fa nel terzo modo

---

Trasformazioni spaziali

Una trasformazione nello spazio è una funzione che possiamo applicare ad un punto, un vettore o un versore per ottenere rispettivamente un punto un vettore o un versore.



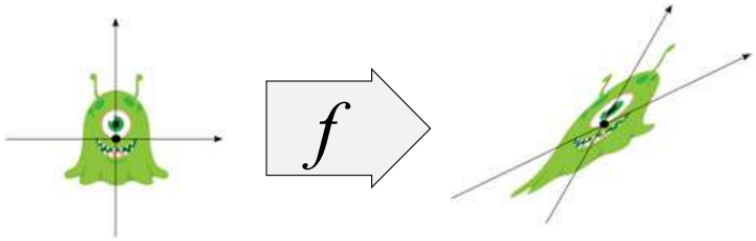
Ogni oggetto è posizionato nella scena. Il mondo virtuale è condiviso da tutti gli oggetti.

Questo viene fatto applicando una trasformazione all'oggetto. Quindi la trasformazione viene applicata a tutti punti, vettori e versori dell'oggetto.

La trasformazione associata ad un oggetto del gioco, è una trasformazione che va dallo spazio-oggetto (dove esiste solo lui, spazio locale) allo spazio-mondo (spazio globale). Questa trasformazione viene detta **modelling transform**.

In computer grafica è comune usare le trasformate affini che vengono salvate in matrici 4D, che rappresentano tutte le trasformazioni degli oggetti. Nei videogames questa non è la rappresentazione più comoda, useremo un sottoinsieme: la classe delle similarità, poi la vedremo.

Una trasformazione affine è una trasformazione che ri-definisce il sistema di riferimento, ovvero porta il sistema di riferimento in un altro. Quindi la **trasformazione affine** porta da un sistema di riferimento all'altro.



Lo spazio mondo e lo spazio oggetto sono definiti dal **sistema di riferimento**.

◆ Given

the local reference frame
- three "axis" vectors $\vec{x}$ $\vec{y}$ $\vec{Z}$
- one "origin" point $\mathbf{p}$

and

- a point $\mathbf{a} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}$ or vector $\vec{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$ on the model — Expressed in local coords

◆ Write an expression to find

- the corresponding point $\mathbf{a}'$ or vector $\vec{\mathbf{V}}'$ but expressed in world space

Note: $\vec{x}$ $\vec{y}$ $\vec{z}$ and $\mathbf{p}$ are points/vectors expressed in WORLD SPACE coords

$\mathbf{a}$, $\vec{v}$ are points/vectors expressed in LOCAL SPACE coord

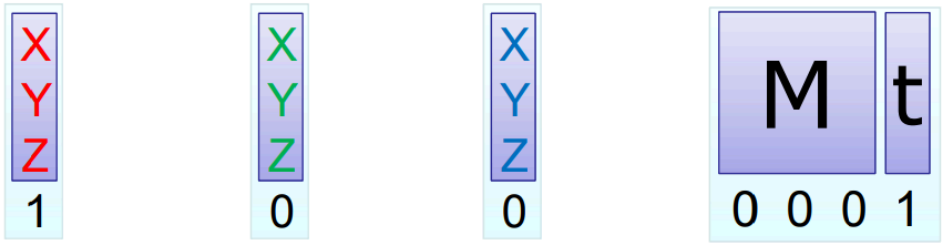
Dato un sistema di riferimento e un punto origine, e dato un punto e un vettore, possiamo scrivere un'espressione che ci dice quel punto e quel vettore dello spazio oggetto a che punto e che vettore corrispondono nello spazio mondo.

p c'è nel primo (punto), è una traslazione, c'è solo nel punto e non nel vettore sotto perchè quando facciamo gaming i vettori partono tutti dall'origine, quindi non c'è traslazione.

$$\mathbf{a}' = \vec{\mathbf{p}} + a_x \vec{\mathbf{x}} + a_y \vec{\mathbf{y}} + a_z \vec{\mathbf{z}}$$
$$\vec{\mathbf{v}}' = v_x \vec{\mathbf{x}} + v_y \vec{\mathbf{y}} + v_z \vec{\mathbf{z}}$$

these equations can be written concisely using *matrix notation*...

points: vectors: versors: transforms:



Si usano le **coordinate omogenee**.

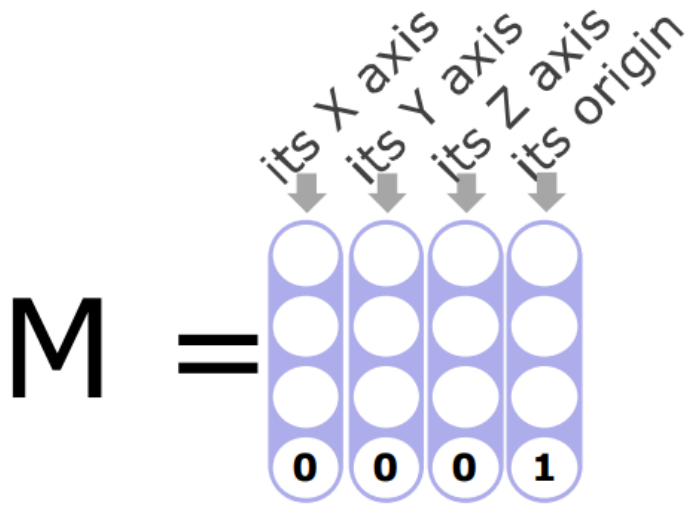
Quante coordinate ci servono per rappresentare uno spazio n-dimensionale? n. Ora vogliamo rappresentare lo spazio delle trasformazioni, da 3D a 3D con una matrice 4×4, perchè abbiamo bisogno di includere nella trasformazione anche una parte addizionale, una “traslazione”.

Per fare questo usiamo le coordinate omogenee, ovvero l’aggiunta di un “1” come quarta coordinata a tutte le coordinate 3D. quindi diventa x y z 1. Questo valore della quarta coordinata può assumere qualunque valore, se è diverso da 0 allora il punto che sta rappresentando è dato dalla prima seconda e terza coordinata, ma divise dal valore della quarta coordinata.

Lo 0 in fondo ci serve per rappresentare “tutte le direzioni” ovvero tutti i punti all’infinito, che ci dicono dove puntano i vettori e i versori.

Quindi la trasformazione è data da una matrice 4×4, che ha una colonna per ogni asse, più una colonna dedicata all’origine (vedremo il perchè).

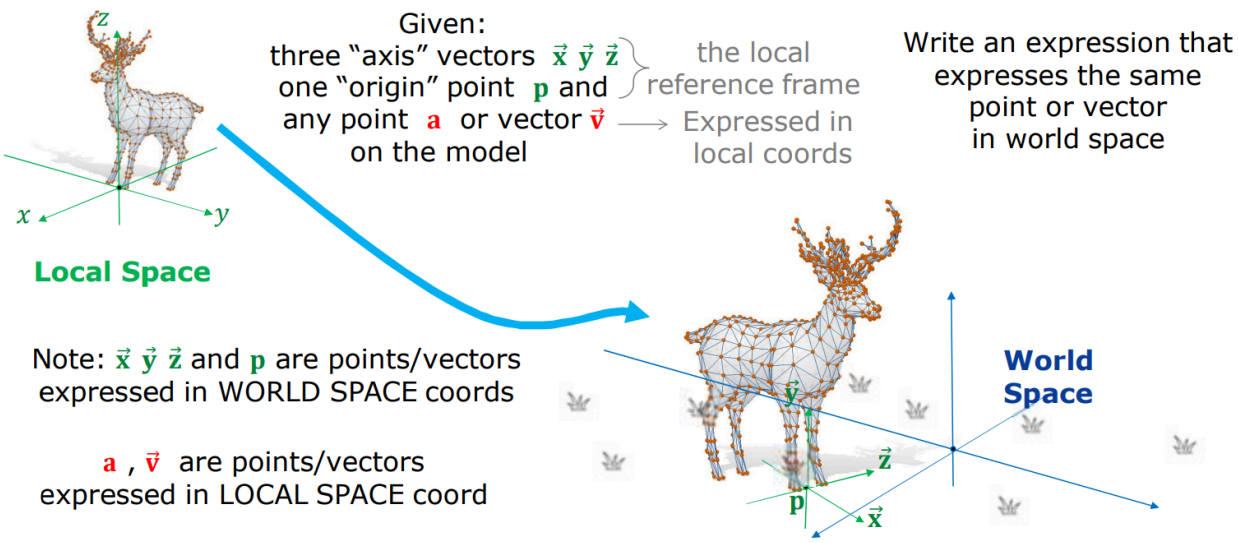
L’ultima riga della matrice è data da un vettore fisso che è sempre 0 0 0 1. Questa matrice è una diretta descrizione del sistema di riferimento di partenza. Se applichiamo questa matrice ad un vettore in coordinate omogenee vedremo vediamo che ciascuno degli elementi è definito da una particolare riga della matrice per il vettore che stiamo analizzando.



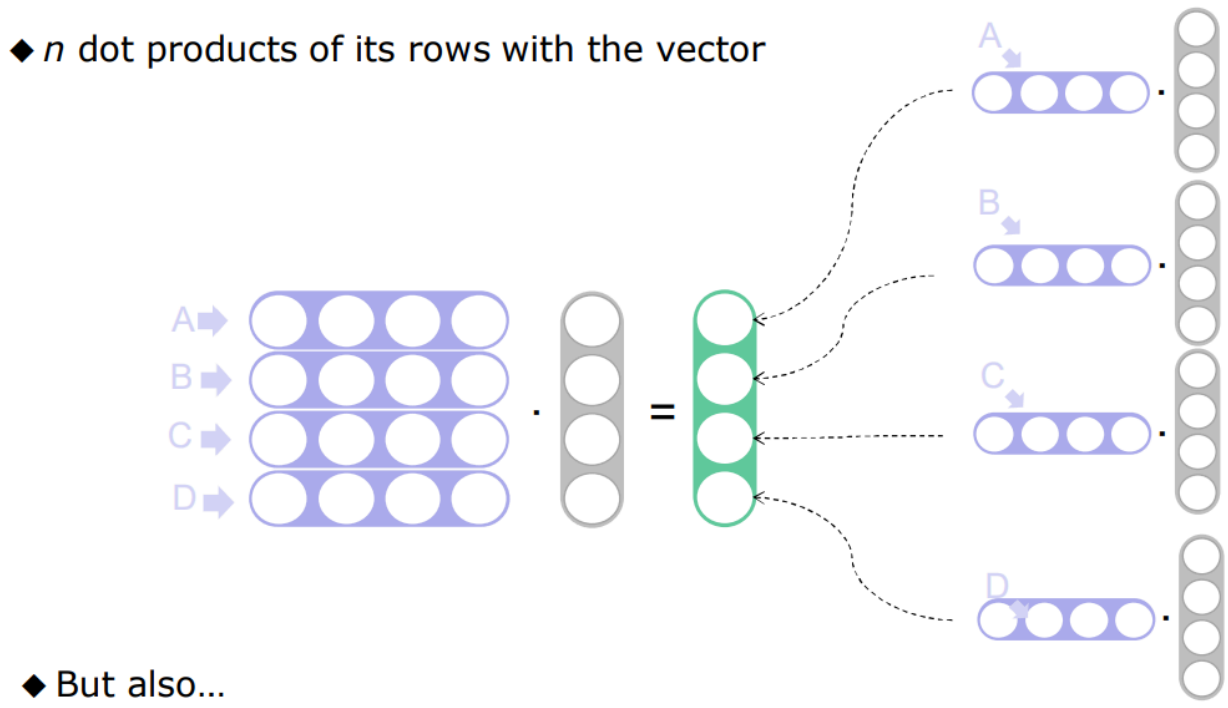
Nelle colonne ci sono i valori della x y e z intesi come assi del nuovo sistema di riferimento locale. Quindi abbiamo le coordinate locali (verde) e devo trasformarle dallo spazio oggetto allo spazio mondo, spostando quindi lo spazio di riferimento. Quindi la trasformazione “codifica” quella riga azzurra.

Quindi la prima colonna rappresenta i 3 valori dell’asse x, etc. La quarta colonna è p. Quindi dati x y z p sistema di riferimento locale dello spazio mondo, possiamo calcolare la matrice di trasformazione che porta un oggetto dal suo sistema di riferimento allo spazio mondo, rispetto a questo sistema di riferimento.

L’ultima riga è 0 0 0 1 perchè le prime tre colonne sono versori, e quindi la loro ultima coordinata omogenea è 0, mentre invece p è un (vettore penso) e quindi ha 1.

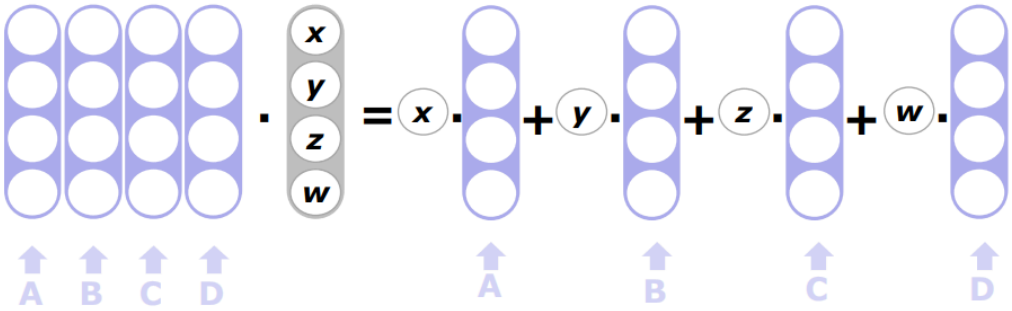


Quindi possiamo vedere l'applicazione della matrice in questo modo, ovvero matrice per vettore, ma non è il modo giusto di vederla:

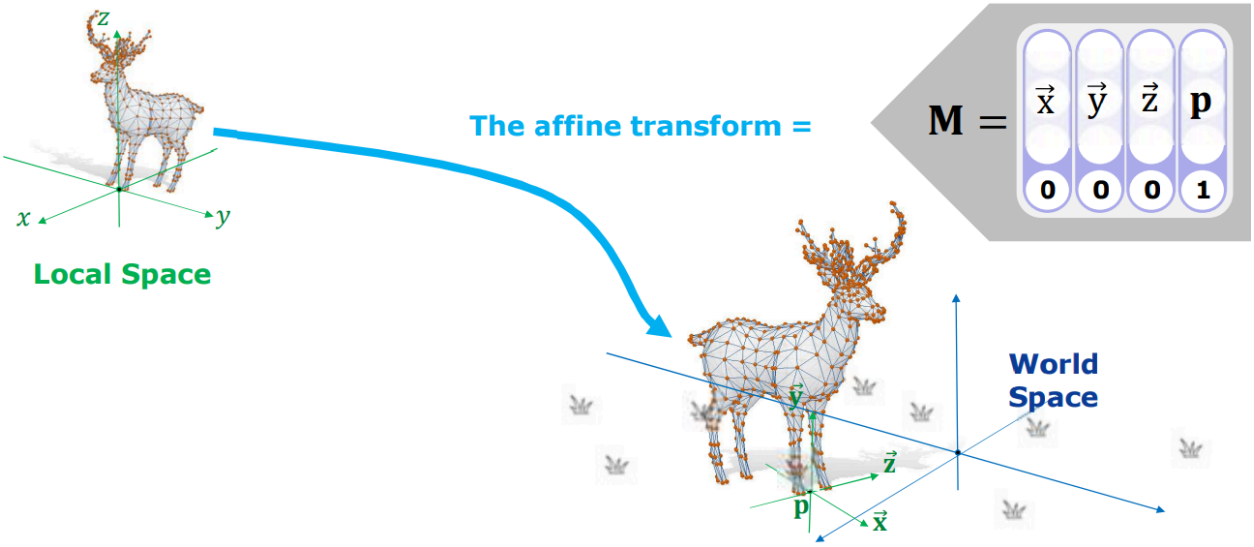


Preferiamo vederla come una combinazione lineare. Vediamo quindi il vettore che dai i coefficienti della combinazione lineare tra l'x y e z del nuovo sistema di riferimento locale. Quindi ci dice dove andare a mettere ogni punto in quel sistema di riferimento.

◆ ...a linear combination of its columns



Ogni trasformazione affine può essere calcolata conoscendo il primo sistema di riferimento e il secondo, perchè possiamo calcolare la matrice che porta gli oggetti dal primo spazio al secondo, che è questa matrice M.



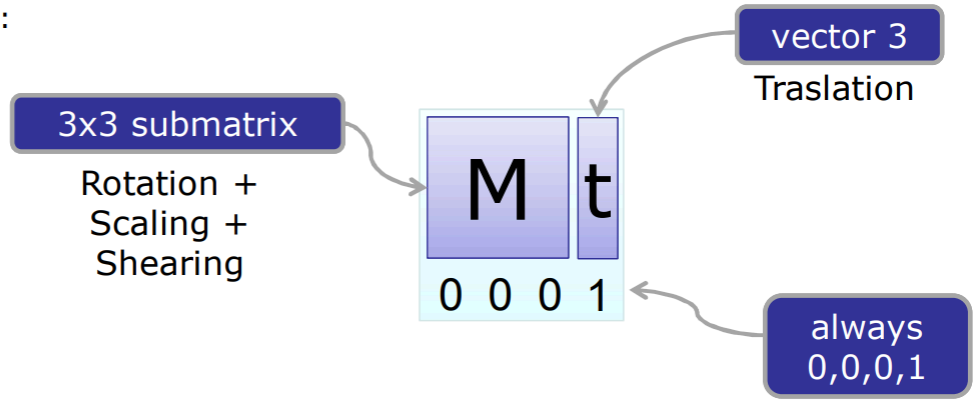
Questa trasformazione da spazio oggetto a spazio mondo è universale a prescindere dall'oggetto.

La matrice 4×4 possiamo scomporla in 3 pezzi:

- il pezzo sotto 0 0 0 1 che non varia
- una matrice 3×3 dove possiamo trovare questo tipo di trasformazione (rotazione, scaling sharing)

- nell'ultima colonna troviamo la traslazione

◆ General case :



◆ Equivalently, can be stored as:
Mat3x3 M and Vec3 t