

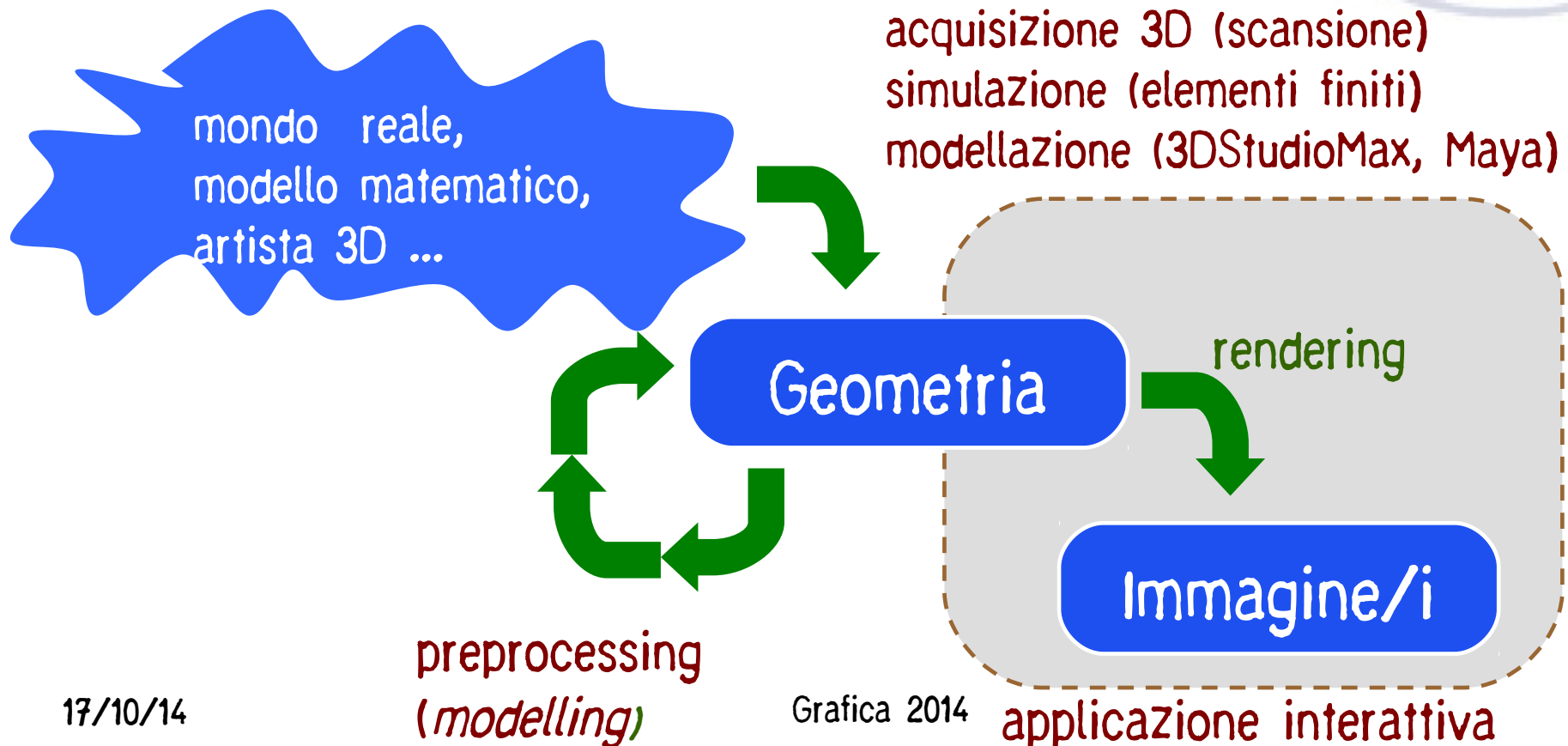
# Grafica al calcolatore - Computer Graphics

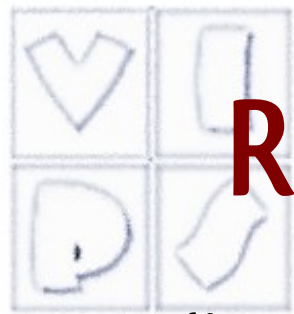


## 4 - Modellazione

# Modellazione

- Definiamo ora possibili strutture dati per modellare gli oggetti nello spazio.
- Poi vedremo come modellare anche la formazione delle immagini attraverso il “rendering”



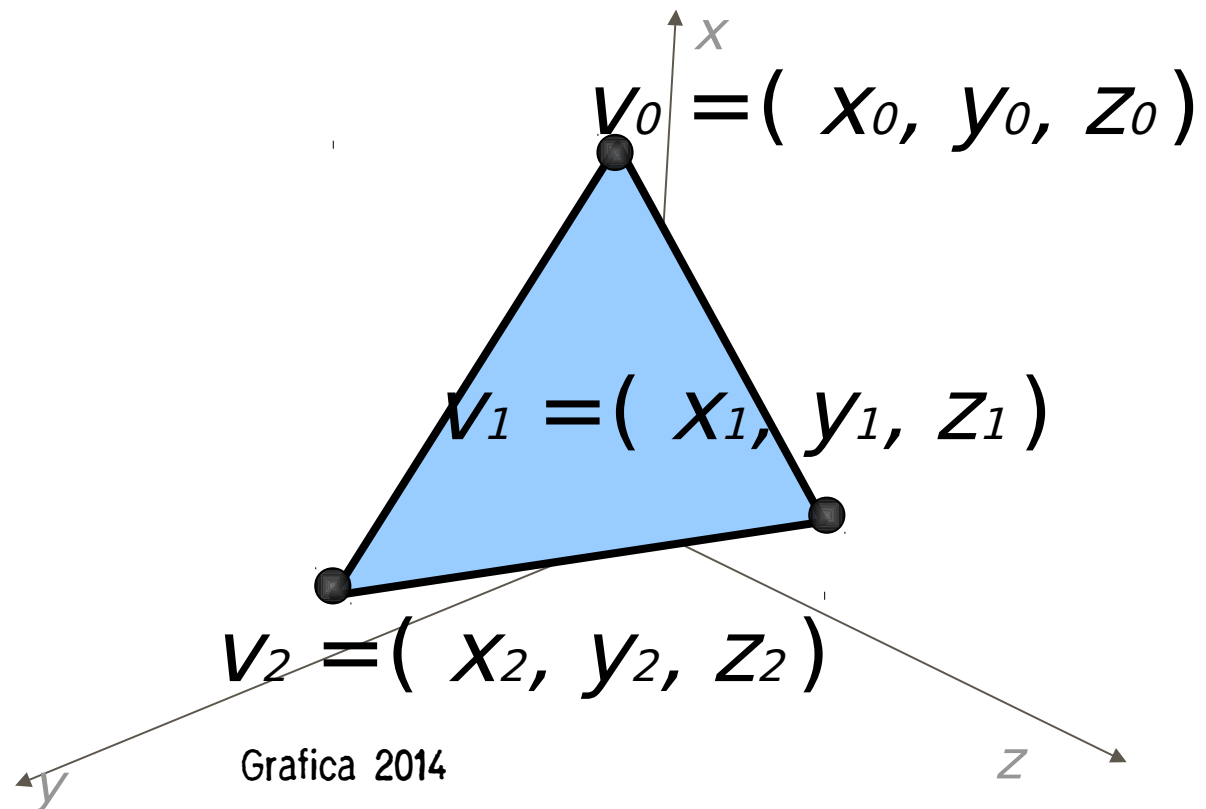


# Rappresentazione degli oggetti

- Gli oggetti che si vogliono rappresentare in una applicazione grafica hanno di solito caratteristiche particolari
  - Sono finiti
  - Sono chiusi (non sempre)
  - Sono continui
- Le rappresentazioni di oggetti (regioni dello spazio, in generale) si suddividono in
  - basate sul contorno (boundary): descrivono una regione in termini della superficie che la delimita (boundary representation, o b-rep).
  - basate sullo spazio occupato (o volumetriche).

# Rappresentazione comune: triangoli

- E' la rappresentazione più usata nella grafica interattiva per motivi di efficienza
  - La pipeline delle schede grafiche è ottimizzata per il rendering dei triangoli





# Esistono però alternative

- Boundary
  - Superfici parametriche (lisce, non hanno problemi di “tessellazione” cioè visibilità degli spigoli tra le facce)
- Volumetriche
  - Rappresentazione “voxellizzata” (a cubetti)
  - Geometria costruttiva solida
- Image based rendering:
  - Non si modella effettivamente la scena, ma si memorizzano campionamenti della luce, rendendo però possibile una visualizzazione da più punti di vista, interattiva
  - Light fields





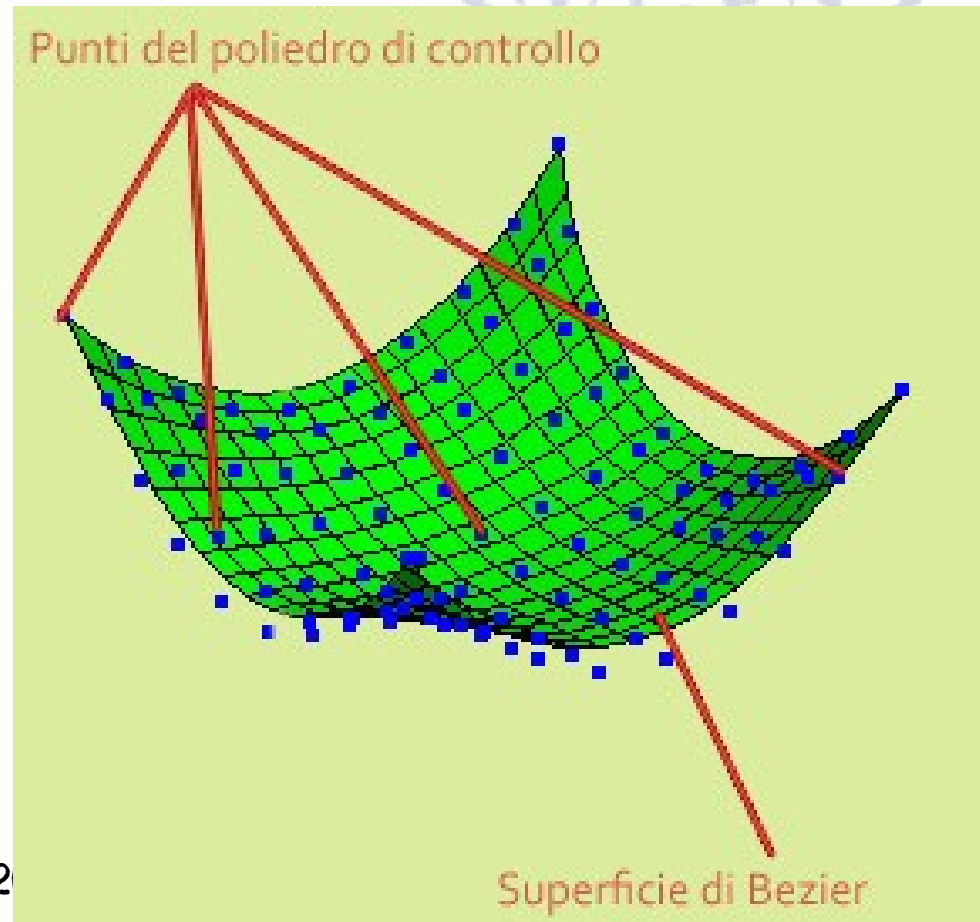
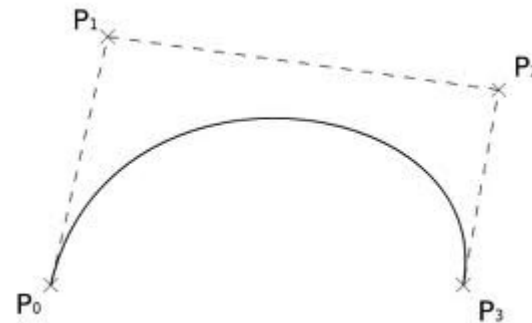
# Curve e superfici lisce

- Il vantaggio principale nell'uso di superfici per modellare un oggetto sarebbe l'assenza del problema della tessellazione visibile (cioè approssimo una superficie liscia coi triangoli, ma vedo poi i triangoli evidenti, effetto ridotto di solito con trucchi opportuni nel rendering)
- Uso di superfici parametriche pesante per applicazioni in tempo reale; per lo più utilizzate in fase di modellazione o per rendering non interattivo
- Negli ultimi tempi le cose sono cambiate ed oggi cominciano ad apparire applicazioni di grafica avanzata che usano superfici curve anche in tempo reale

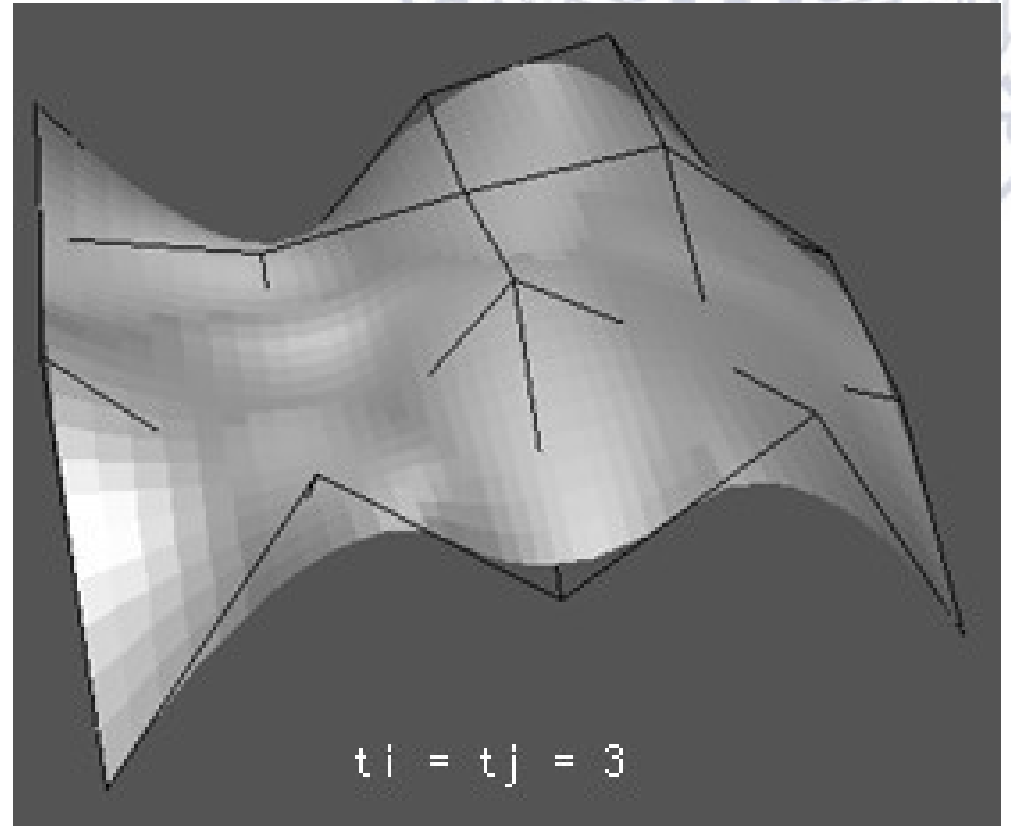
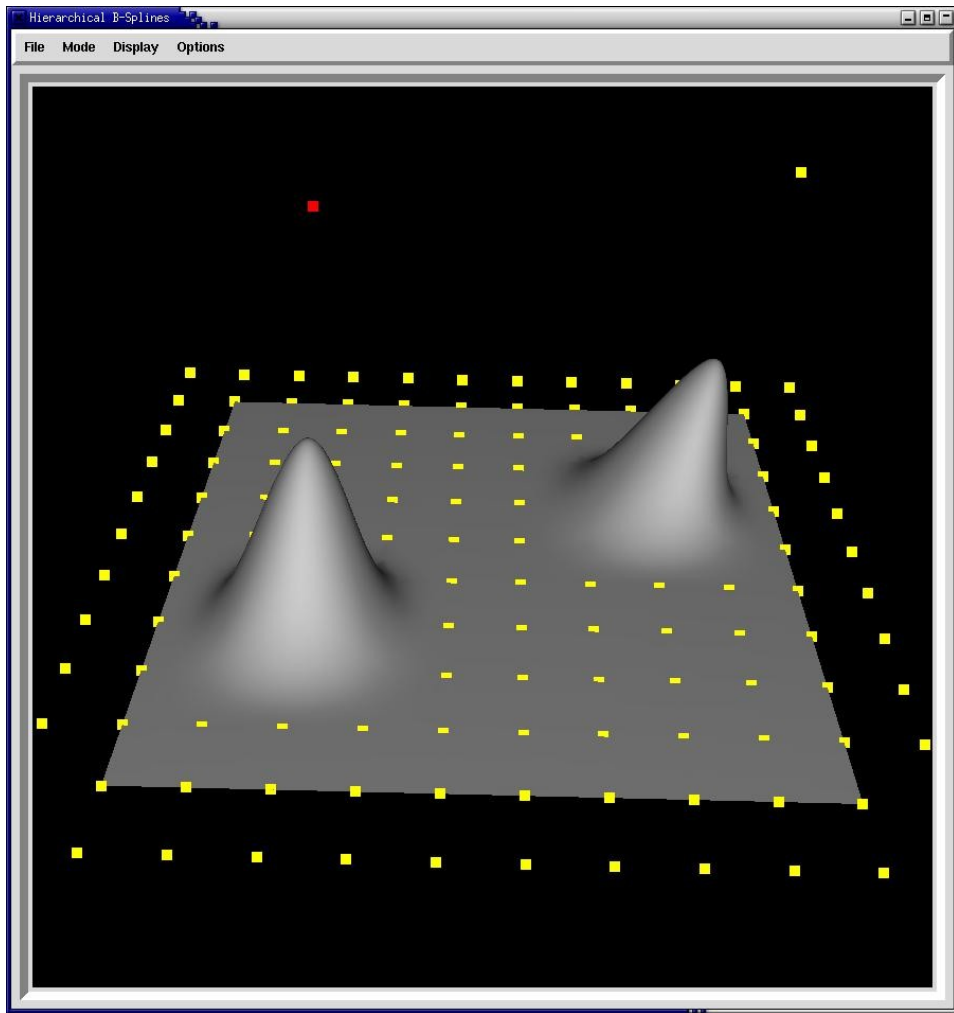
# Curve e superfici lisce



- Esempio: curve/superfici di Bezier
  - Dati  $N$  punti di controllo la curva passa per il primo e l'ultimo e approssima gli altri con una funzione da essi dipendente
  - Con una griglia si generano superfici



# Superfici parametriche - Esempi

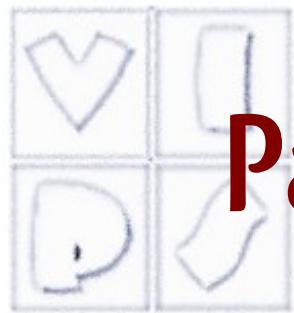




# Partizionamento spaziale (voxel)

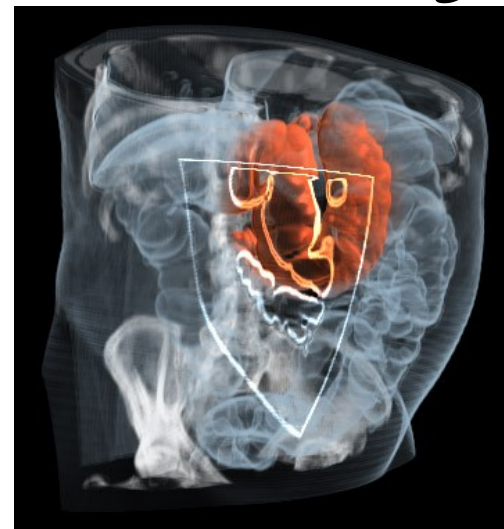
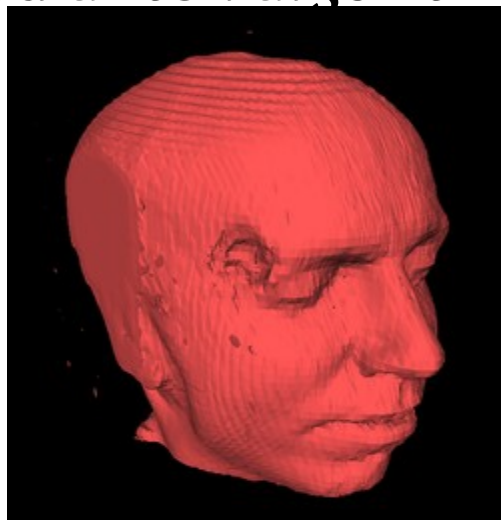
- Lo spazio viene suddiviso in celle adiacenti dette in 3D voxel (equivalente dei pixel delle immagini): una cella è “piena” se ha intersezione non vuota con la regione, è detta vuota in caso contrario. Oppure contiene un valore di densità (tipico dei dati diagnostici es. TAC)
- Una rappresentazione di una scena complessa ad alta risoluzione richiederebbe l'impiego di un numero enorme numero di voxel, per cui questa rappresentazione è in genere limitata a singoli oggetti.
- Ma le cose stanno cambiando grazie a progressi nell'HW e nel SW





# Partizionamento spaziale (voxel)

- Rendering di modelli voxel-based:
  - algoritmi ad hoc (tecniche direct volume rendering)
  - conversione da voxel a rappresentazione per superfici (triangle-based)
- Da una rappresentazione volumetrica voxelizzata si può passare efficientemente a una rappresentazione poligonale della superficie mediante l'algoritmo detto marching cubes.

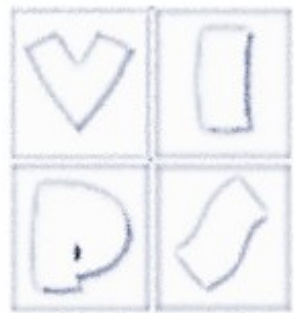




# Rappresentazioni compatte

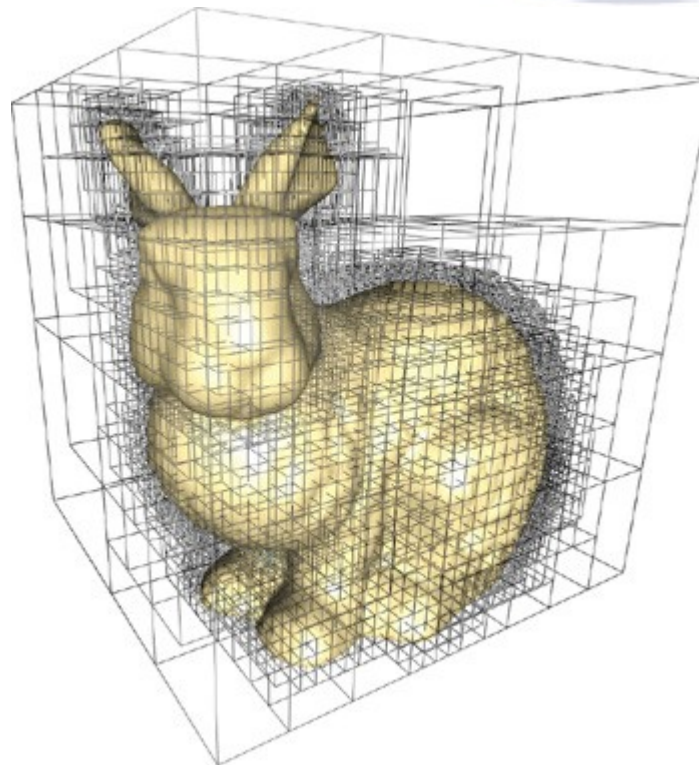
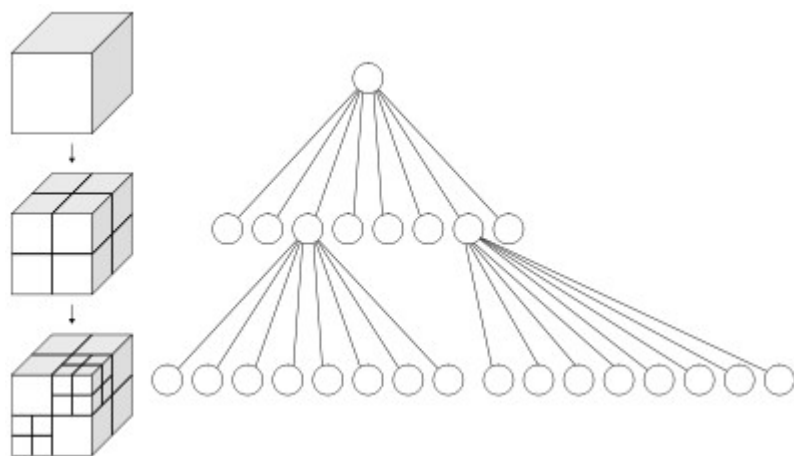
- Se ho solo i valori pieno/vuoto, posso rappresentare in modo compatto il volume con una struttura octree
- Si parte con un cubo contenente la regione e si suddivide ricorsivamente. Ci si ferma ogni volta che un ottante contiene tutte celle piene o tutte celle vuote.
- Più economica rispetto alla enumerazione delle singole celle, poiché grandi aree uniformi (piene o vuote) vengono rappresentate con una sola foglia (anche se nel caso peggiore il numero delle foglie è pari a quello delle celle)

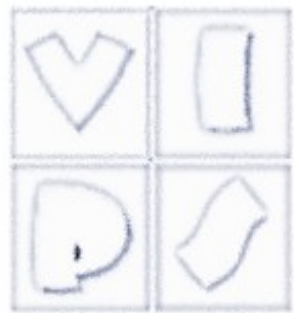




# Octree

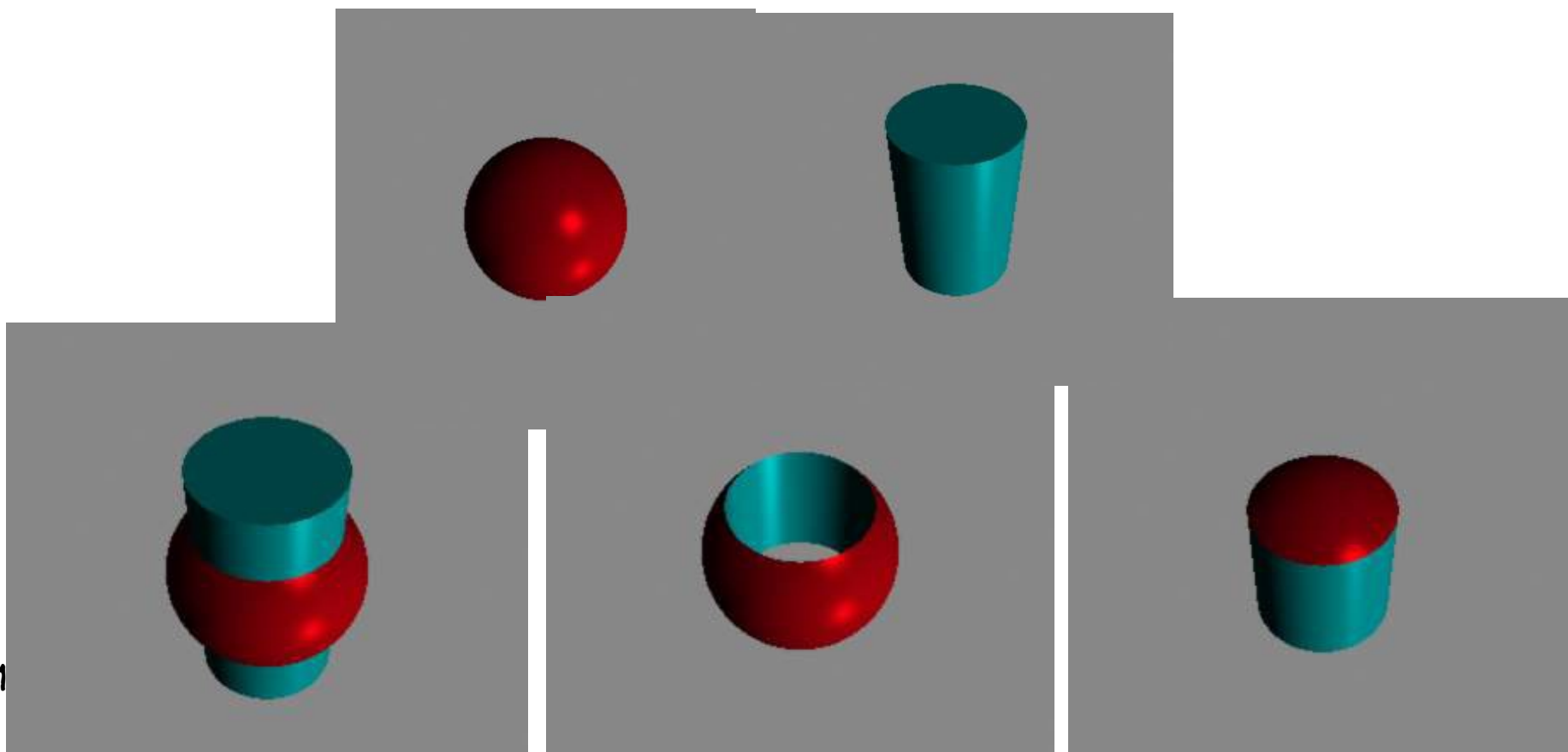
- Naturalmente può servire anche come struttura di supporto per il calcolo delle intersezioni con mesh





# Geometria costruttiva solida

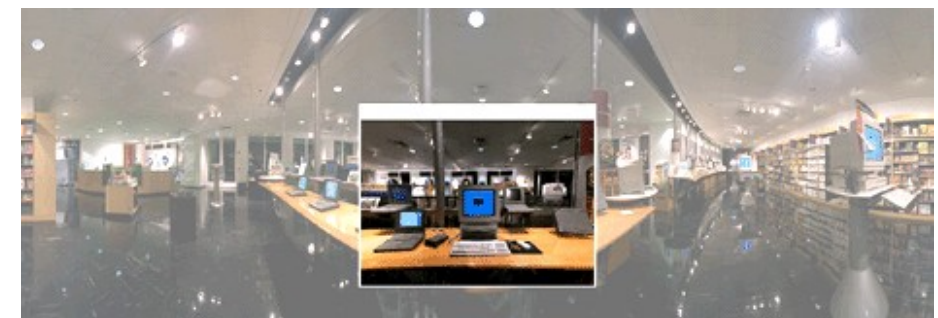
- Altra rappresentazione particolarmente adatta per il modeling (diffusa nel settore CAD), ma poco efficiente per il rendering.
- Si tratta, essenzialmente, di costruire degli oggetti geometrici complessi a partire da modelli base con operazioni booleane





# Immagini panoramiche

- Una scena 3D o un singolo oggetto rappresentati elaborando una serie di immagini 2D, in modo che sia garantita continuità nella rappresentazione della scena e la scelta interattiva della posizione di osservazione
- Uno dei formati più comuni è il QuickTimeVR
- Dal punto di vista centrale l'utente può guardare (ma senza spostarsi) in ogni direzione e può zoomare a piacimento



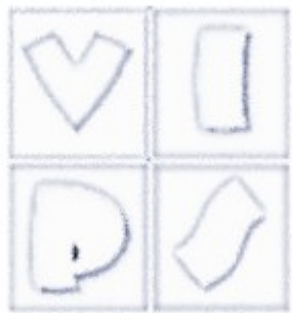
pre-processing

real-time

# Light field - Esempio



- Catturo il campo di illuminazione che deriva dall'oggetto e ne riproduco l'effetto sulle viste scelte
- In figura, esempio di strumentazione per acquisizione della radianza (light field) e immagine di sintesi ottenuta dai dati acquisiti (P. Debevec, USC)



# Poligoni e triangoli

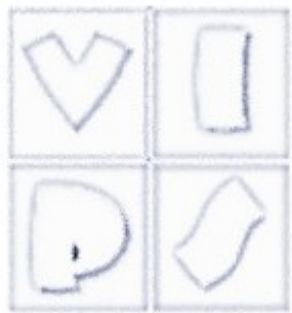
- Nella grafica 3D interattiva si usa l'approssimazione poligonale degli oggetti (del loro contorno).
- Si tratta di approssimare una superficie 2D con un insieme di poligoni convessi opportunamente connessi gli uni agli altri.
- Nella pipeline di rendering si lavora in genere con i soli triangoli
  - tutte le altre rappresentazioni eventualmente usate nel programma sono convertite prima del rendering in triangoli
- Possiamo usare la geometria definita finora per definire rigorosamente le proprietà dei modelli triangolati
  - Un insieme di triangoli è matematicamente un 2-complesso simpliciale puro





# Simplessi

- Un semplice è l'analogo  $n$ -dimensionale del triangolo.
- Specificamente, un semplice di ordine  $n$  (o  $n$ -semplice) è il guscio convesso di  $n+1$  punti affinementemente indipendenti in  $\mathbb{R}^d$ .
- Uno 0-semplice è un punto, un 1-semplice è un segmento, un 2-semplice è un triangolo, un 3-semplice è un tetraedro.
- Il guscio convesso di un qualunque sottoinsieme degli  $n+1$  punti che definiscono il  $n$ -semplice si chiama **faccia** del semplice. Le facce sono a loro volta semplici (di ordine  $n$ ).
- Se il sottoinsieme è proprio, anche la faccia si dice propria.
- Le facce di ordine 0 sono i punti stessi, chiamati vertici. Le facce di ordine 1 si chiamano spigoli. La faccia di ordine  $n$  è l' $n$ -semplice stesso.

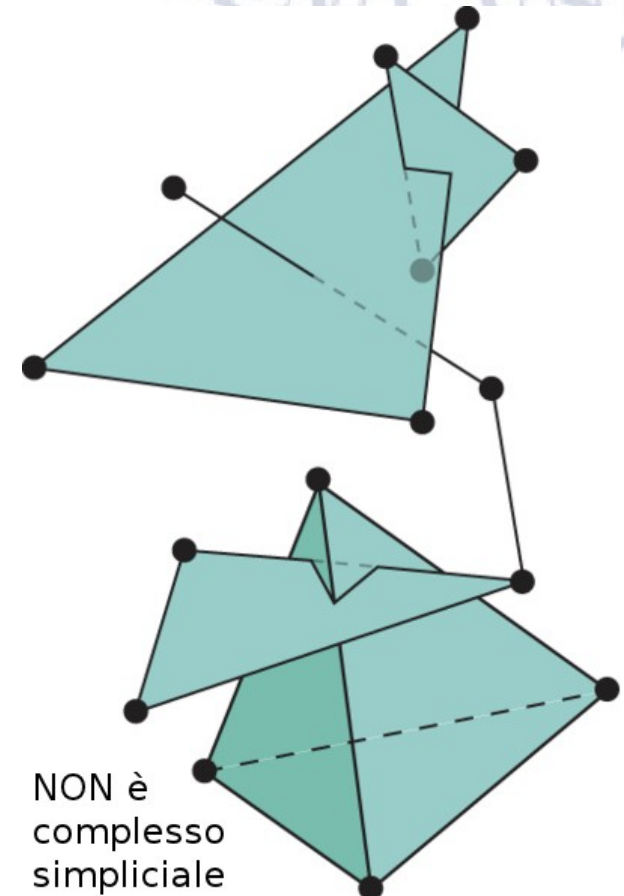
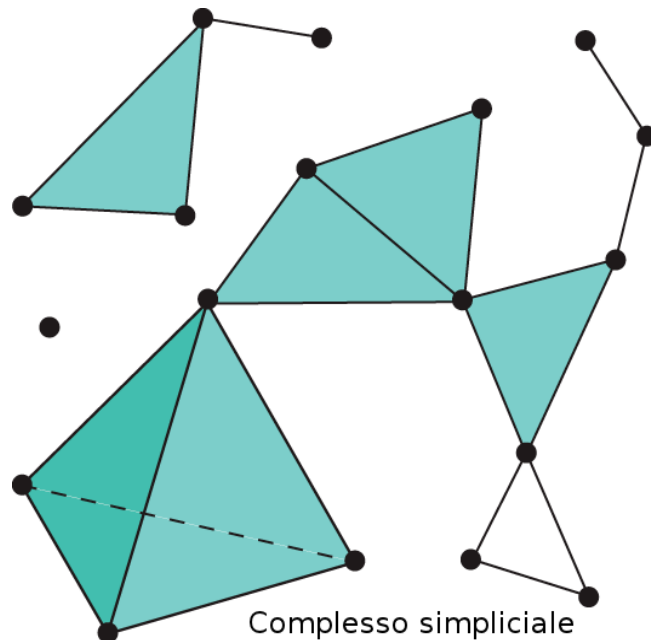


# Complessi simpliciali

- Un complesso simpliciale  $K$  è un insieme di semplici che soddisfano le seguenti condizioni:
  - Ogni faccia di un semplice in  $K$  appartiene a sua volta a  $K$ .
  - L'intersezione di due semplici 1 e 2 è una faccia comune a 1 e 2 oppure è vuota.
- Se l'ordine massimo dei semplici è  $k$ ,  $K$  prende il nome di  $k$ -complesso simpliciale
  - Per esempio, un 2-complesso simpliciale deve contenere almeno un triangolo e nessun tetraedro.
  - Un  $k$ -complesso simpliciale è puro se ogni semplice di ordine  $< k$  è la faccia di un  $k$ -simple.



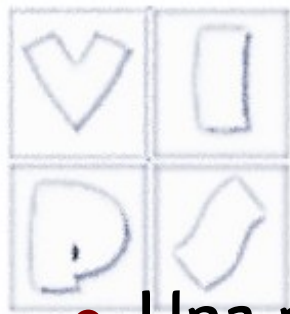
- Per esempio, un 2-complesso simpliciale puro è fatto solo di triangoli (non ci sono vertici o spigoli “orfani”).
- Due semplici 1 e 2 sono incidenti se 1 è una faccia propria di 2 o vale il viceversa.
- Due  $k$ -semplici sono  $(k-1)$ -adiacenti se esiste un  $(k-1)$  semplice che è una faccia propria di entrambi.





# Varietà

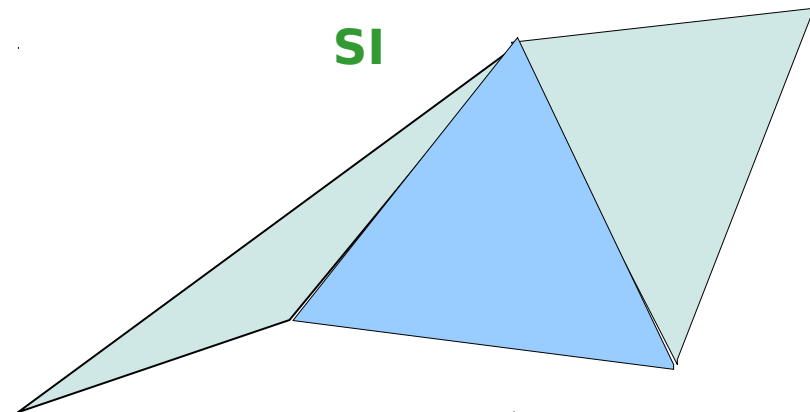
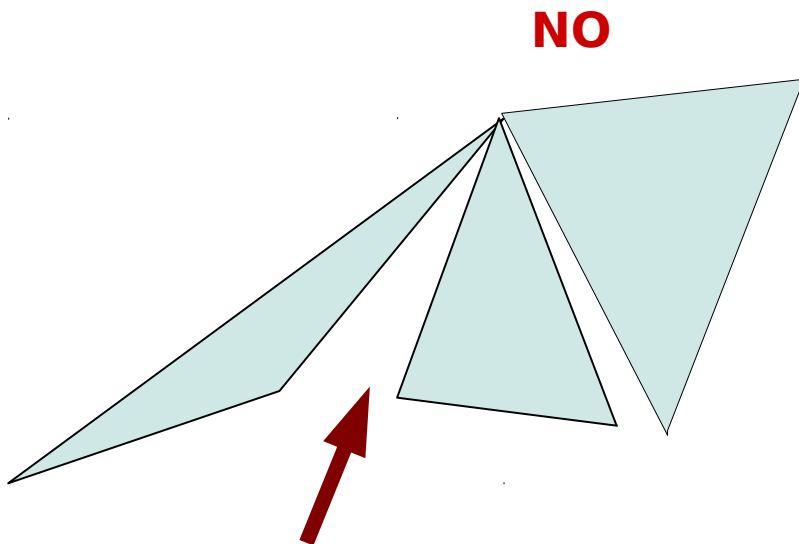
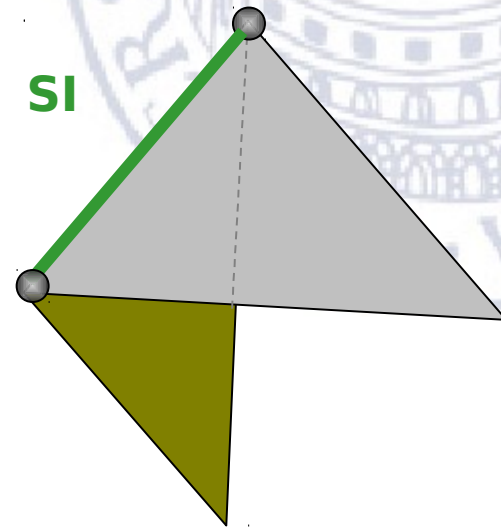
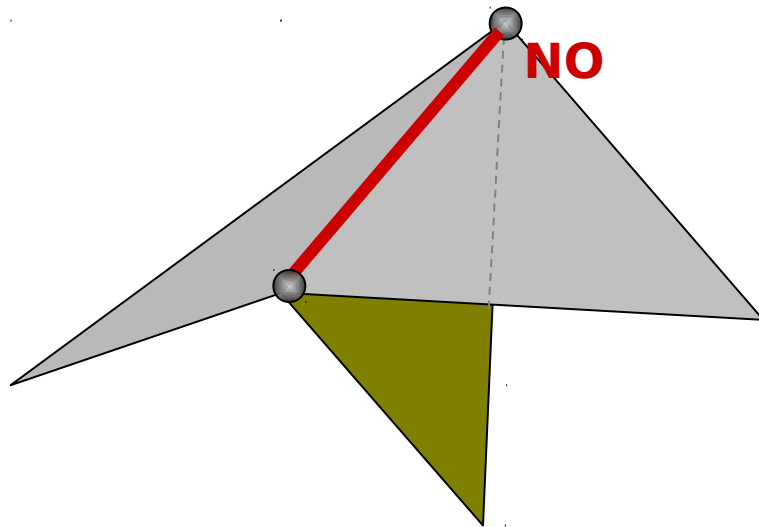
- Una varietà  $k$ -dimensionale  $X$  è un sottoinsieme di  $\mathbb{R}^d$  in cui ogni punto ha un intorno omeomorfo alla sfera aperta di  $\mathbb{R}^k$ .
- In generale le superfici degli oggetti solidi (sfere, poliedri, ecc.) sono varietà bidimensionali.
- Omeomorfismo: applicazione biiettiva, continua, con inversa continua. Intuizione: trasformazione senza "strappi".
  - In una varietà  $k$ -dimensionale **con bordo** ogni punto ha un intorno omeomorfo alla sfera aperta o alla semisfera aperta di  $\mathbb{R}^k$ .
  - Il bordo di  $X$  è l'insieme dei punti che hanno un intorno omeomorfo alla semisfera aperta.
  - Una varietà è sempre una varietà con bordo, eventualmente vuoto.
  - Il bordo, se non è vuoto, è a sua volta una varietà  $k-1$  dimensionale senza bordo.

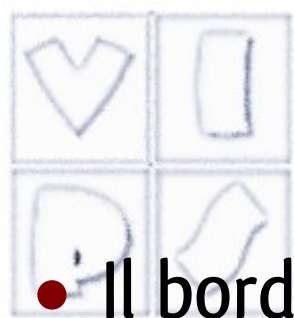


# Mesh poligonali

- Una maglia (mesh) triangolare è 2-complesso simpliciale puro che è anche una varietà bidimensionale con bordo.
- I triangoli della maglia si chiamano anche facce.
- La condizione di essere varietà si traduce nei seguenti vincoli ulteriori sulla struttura del complesso simpliciale:
  - uno spigolo appartiene al massimo a due triangoli (quelli eventuali che appartengono ad uno solo formano il bordo della maglia)
  - se due triangoli incidono sullo stesso vertice allora devono appartenere alla chiusura transitiva della relazione di 1-adiacenza, ovvero devono formare un ventaglio o un ombrello.
- Si usa il termine condizione 2-manifold (varietà)

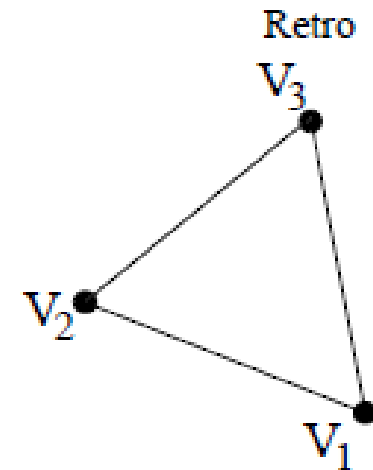
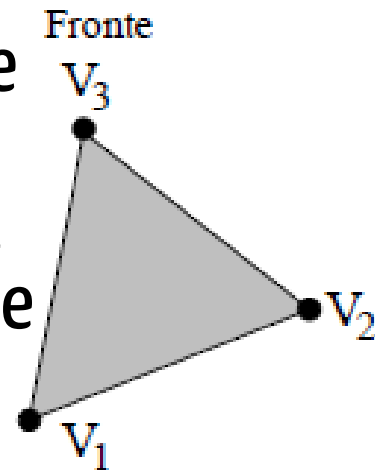
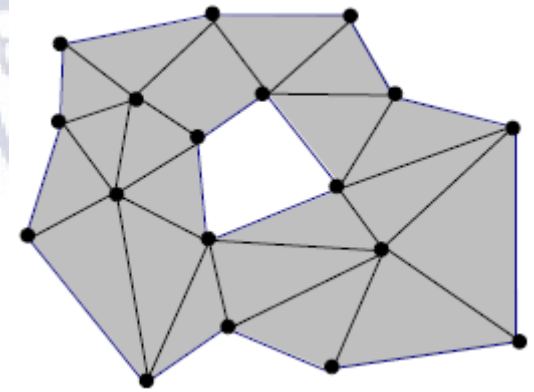
# Condizione 2-manifold



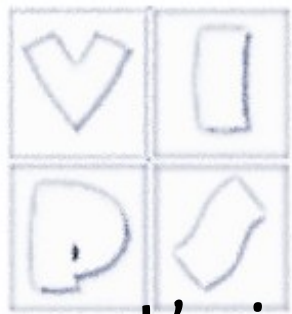


# Orientazione

- Il bordo della maglia consiste di uno o più anelli (sequenza chiusa di spigoli) o loop.
- Se non esistono spigoli di bordo la maglia è chiusa (come quelle che rappresentano la superficie di una sfera).
- L'orientazione di una faccia è data dall'ordine ciclico (orario o antiorario) dei suoi vertici incidenti. L'orientazione determina il fronte ed il retro della faccia. La convenzione (usata anche da OpenGL) è che la faccia mostra il fronte

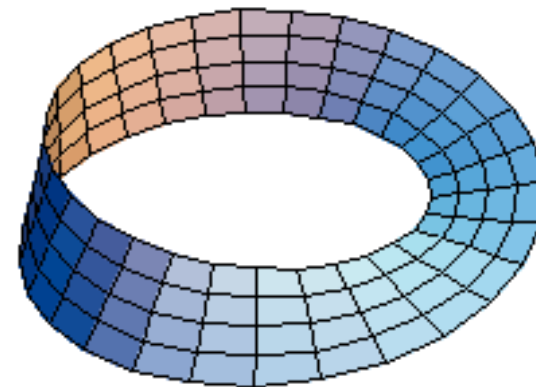


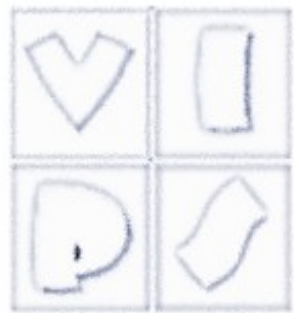




# Mesh orientabili

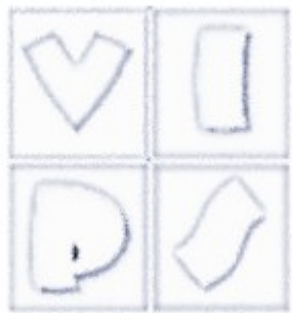
- L'orientazione di due facce adiacenti è compatibile se i due vertici del loro spigolo in comune sono in ordine inverso. Vuol dire che l'orientazione non cambia attraversando lo spigolo in comune.
- La maglia si dice orientabile se esiste una scelta dell'orientazione delle facce che rende compatibili tutte le coppie di facce adiacenti.
  - Non tutte le mesh 2-manifold sono orientabili (es. anello di Moebius)





# Maglie/Mesh generiche

- Abbiamo definito la maglia triangolare. In maniera analoga si può estendere la definizione a maglie poligonali generiche
  - Maglie poligonali generiche: i poligoni possono avere qualsiasi numero di spigoli e non è detto che ci sia un solo tipo di poligono. Sono raramente utilizzate in grafica al calcolatore
  - Quadrangolari (quad meshes): gli elementi poligonali sono tutti quadrilateri. Sono alle volte usate, per esempio se si vuole fare il rendering di un terreno descritto da un array di altezze. In una maglia quadrangolare bisogna imporre un vincolo aggiuntivo di planarità per ogni quadrilatero che la compone.
- OpenGL consente di descrivere maglie poligonali generiche, ma per disegnarle li suddivide usualmente in triangoli.

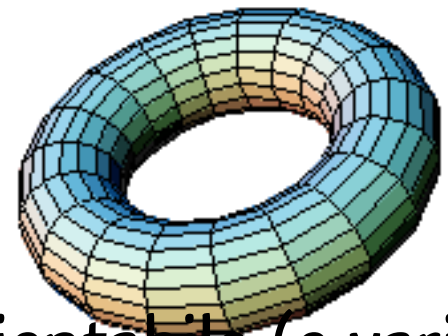


# Equazione di Eulero

Se  $V$  è il numero di vertici,  $L$  il numero di spigoli ed  $F$  il numero di facce della maglia poligonale orientabile chiusa di genere  $G$ , allora vale la Formula di Eulero  $V - L + F = 2 - 2G$

Una superficie ha genere  $G$  se può essere tagliata lungo  $G$  linee semplici chiuse senza disconnetterla (intuitivamente, ma non rigorosamente “numero di buchi”)

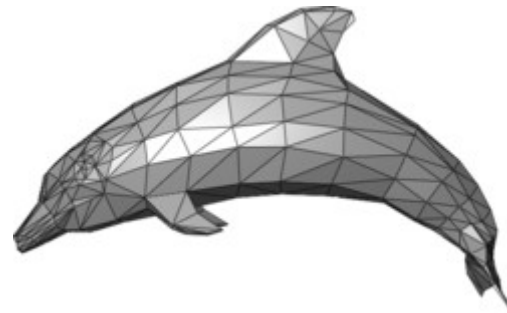
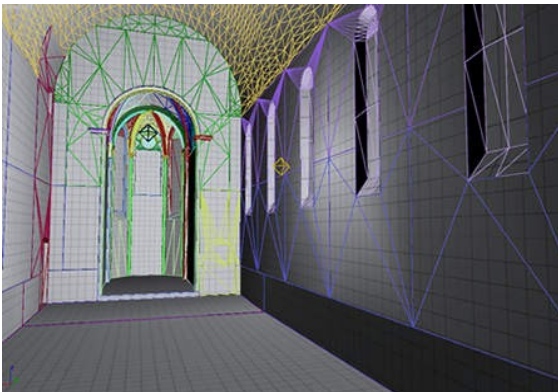
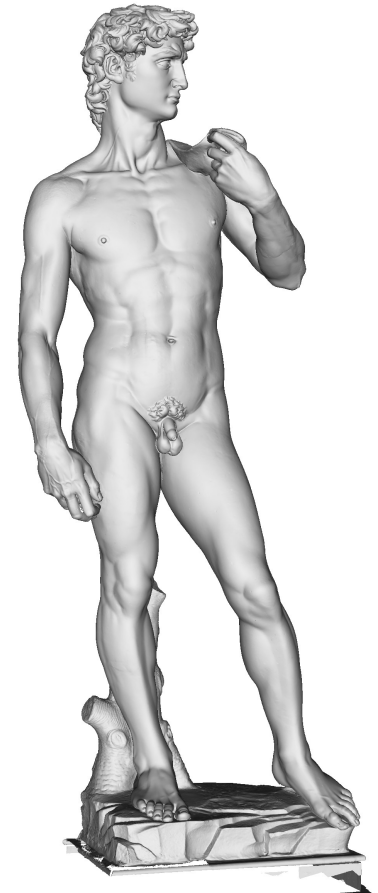
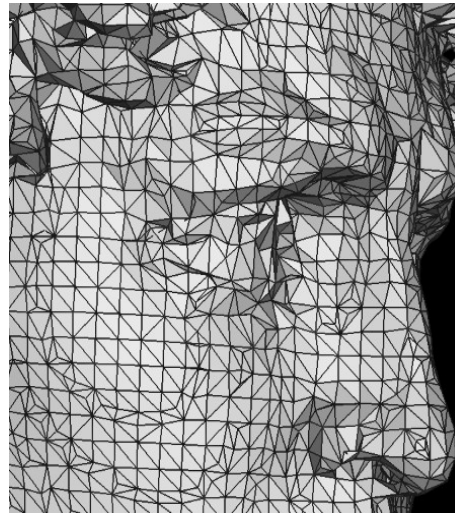
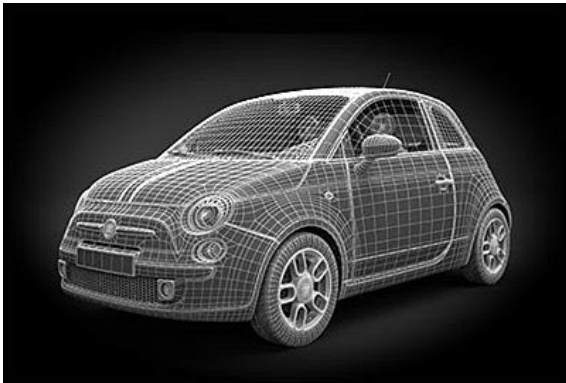
- Il genere di una superficie determina la sua topologia; per una sfera, per esempio,  $G = 0$ , mentre per un toro (una ciambella)  $G = 1$ .
- Più in generale, per una maglia poligonale orientabile (e varietà bidimensionale) vale la formula  $V - L + F = 2(S - G) - B$ 
  - $S$  numero di componenti connesse,  $B$  è il numero di anelli di bordo





# Mesh di triangoli

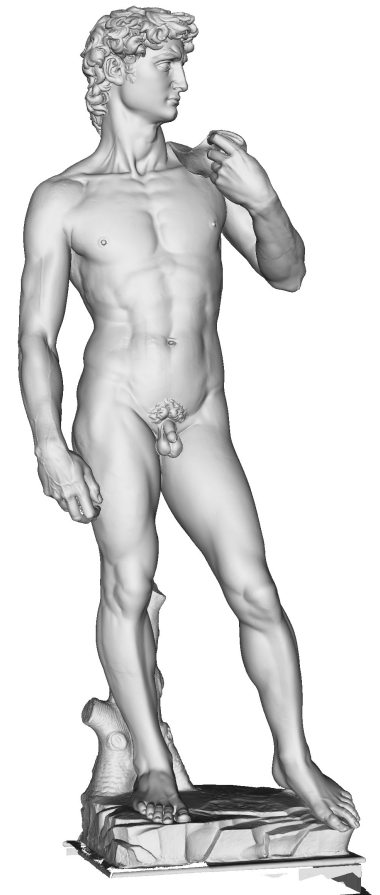
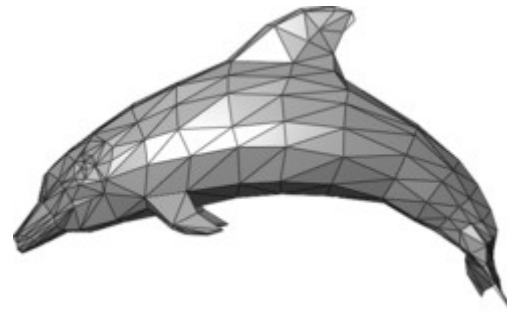
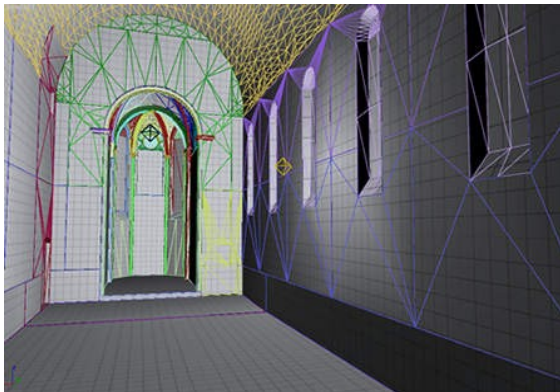
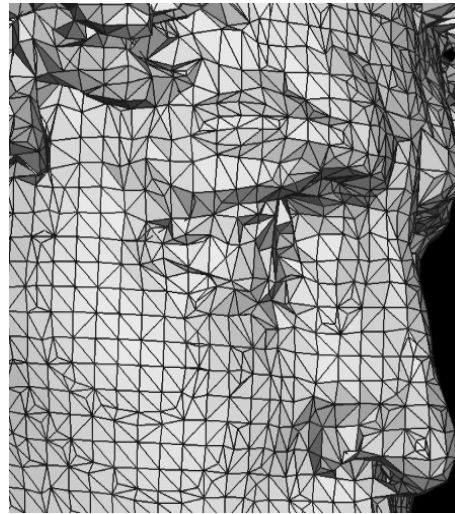
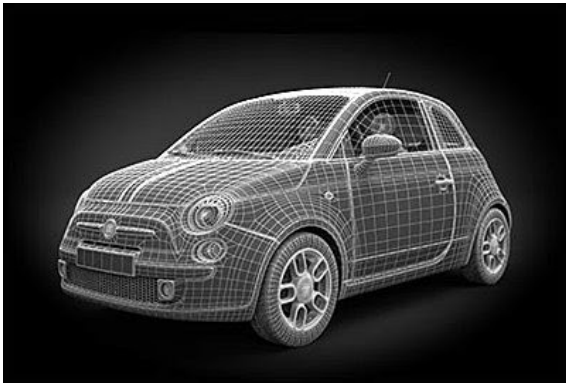
- Nella pratica sono il tipo di modello dominante
- Usato nella gran parte delle applicazioni interattive
  - Dato che il rendering è ottimizzato in hardware



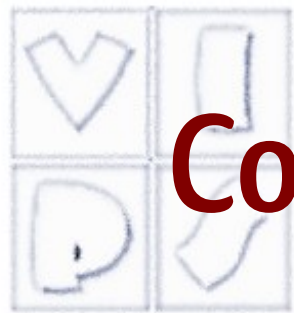
Gratica 2014

# Mesh di triangoli

- Generate da modellazione CAD, acquisizione con scanner, ricostruzione da immagini (Computer Vision)
- Anche molto voluminose: vedremo come **semplificarle**

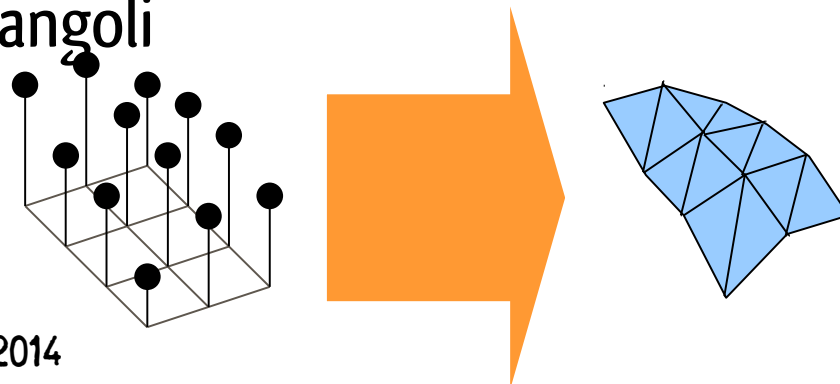
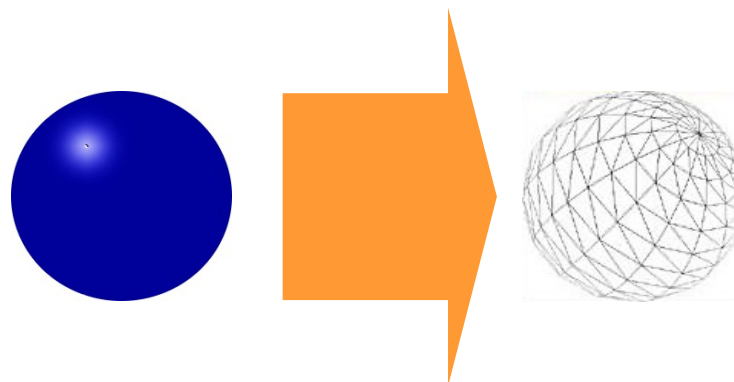
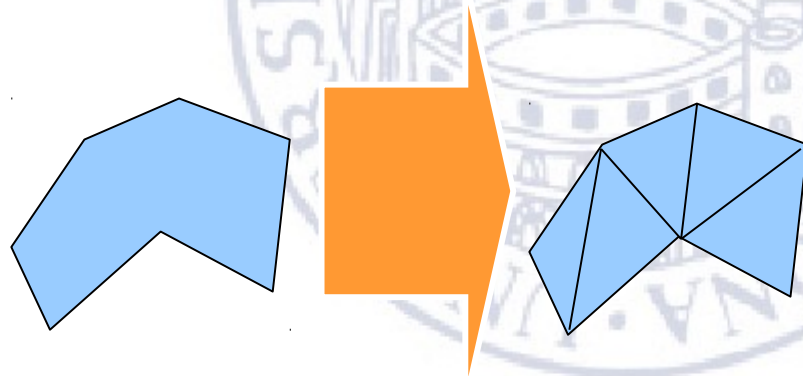






# Costruzione della mesh triangolare

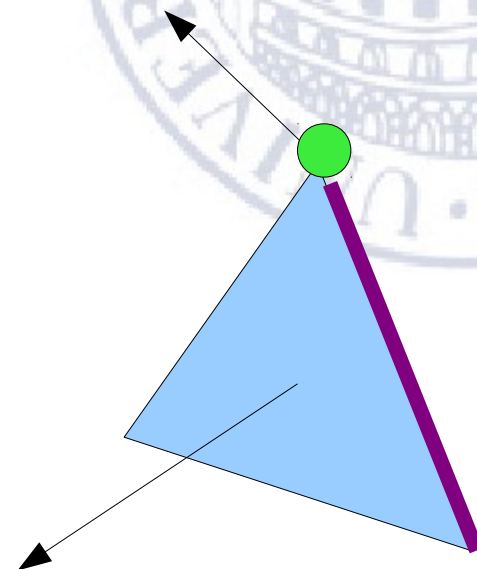
- Conversione da altri formati:
  - Poligoni  $\rightarrow$  Triangoli
  - Superf. Quadriche  $\rightarrow$  Triangoli
  - Campi di altezze o Punti  $\rightarrow$  Triangoli





# Mesh triangolare - Attributi

- Posso definirli:
  - per vertice
    - esplicito un attributo per ogni vertice
  - per faccia
    - esplicito un attributo per ogni faccia
  - per wedge (vertice di faccia)
    - esplicito tre attributi per ogni faccia
- Attributi più comuni:
  - colore
  - coordinate texture





# Mesh triangolare - Limiti

- Non è sempre semplice modellare le entità da rappresentare con triangoli...
  - Esempi:
    - Nuvole
    - Fiamme
    - Capelli, pelliccia



by Niniane Wang  
17/10/14 (non real time)



by N. Adabala Florida Uni  
(non real time) Grafica 2014



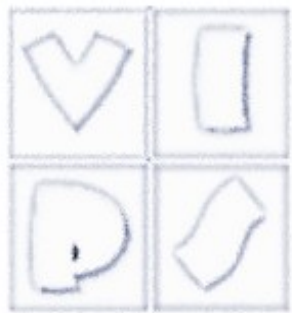
by M. Turitzin and J. Jacobs  
Stanford Uni (real time!)



# Evitare ridondanze

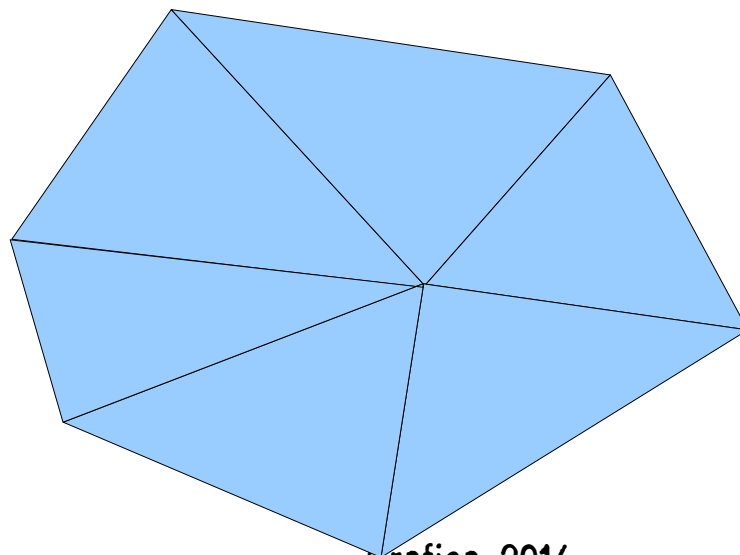
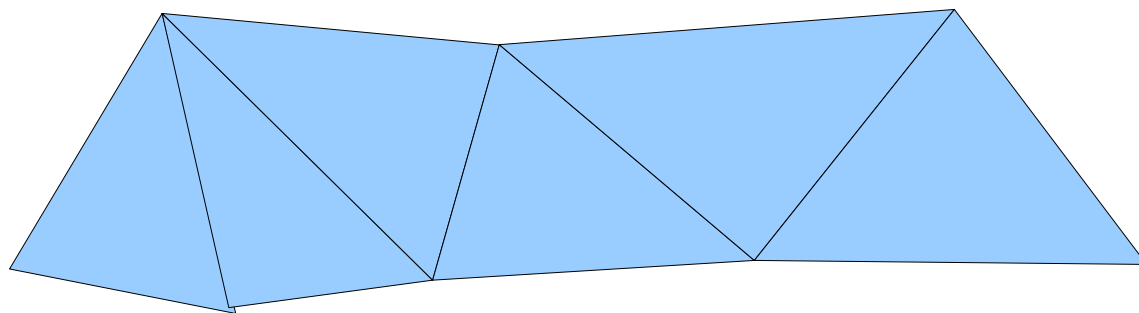
- Quando si devono disegnare due triangoli con uno spigolo in comune, questo viene disegnato due volte. Questo introduce un certo grado di ridondanza che può incidere sulle prestazioni
- Si preferisce quindi raggruppare i triangoli di una maglia in opportuni gruppi che possono essere elaborati in maniere più efficienti. Si possono ad esempio utilizzare
  - **Fan di triangoli:** è un gruppo di triangoli che hanno in comune un vertice. Il primo viene specificato completamente, per i successivi basta dare il nuovo vertice. Efficiente, ma i triangoli che incidono su un vertice sono in genere pochi
  - **Strip di triangoli:** gruppo di triangoli che posseggono a due a due uno spigolo in comune. Di nuovo il primo triangolo viene specificato normalmente, per i successivi basta specificare il nuovo vertice. Meno efficiente, ma le strip in genere contengono più triangoli delle fan





# Strip e fan

- Esistono algoritmi per creare queste rappresentazioni dalle mesh





# Mesh e rendering



- Per determinare l'effetto di una qualsiasi trasformazione affine su un oggetto (traslazione, rotazione, scalatura, composizioni varie di queste), basta applicare la trasformazione ai vertici (che sono punti); le informazioni connettive date dagli spigoli non cambiano in questo tipo di trasformazioni
  - Questo rende piuttosto semplice il rendering di oggetti descritti in termini di maglie poligonali. qualsiasi trasformazione viene eseguita sui vertici, cioè si tratta di applicare trasformazioni affini su punti
  - L'affermazione precedente è vera anche per la proiezione; per vedere come si proietta la forma di una maglia su un piano (l'immagine), basta seguire la proiezione dei vertici.



# Memorizzazione

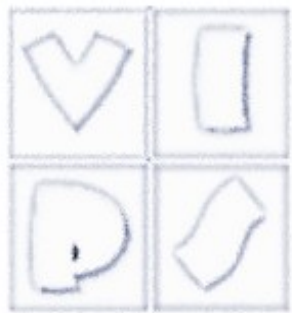
- Alla creazione dei modelli vengono in genere generati nodi, connettività e attributi. Gli algoritmi di processing e rendering devono accedere in vario modo a tale informazione
- Diversi modi di rappresentare questa informazione
- Nei programmi applicativi sono quindi necessarie delle procedure per convertire una rappresentazione in un'altra
- Progettare ed implementare tali procedure è un ottimo modo per capire nel dettaglio le varie rappresentazioni utilizzate per descrivere maglie poligonali
- Nei disegni e negli esempi ci concentreremo sul caso di maglia triangolare, ma il discorso è valido in generale per tutti i poligoni convessi



# Elementi base

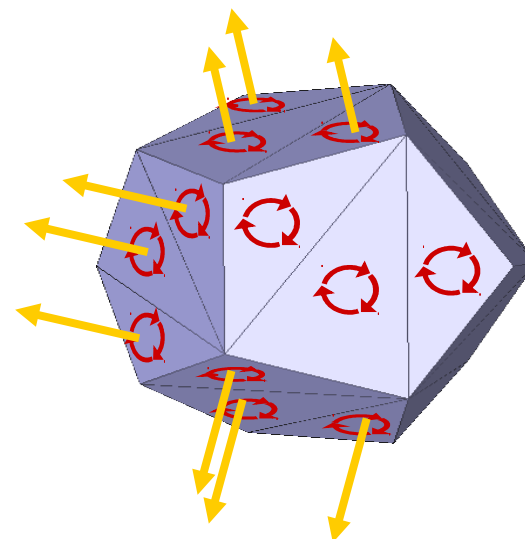
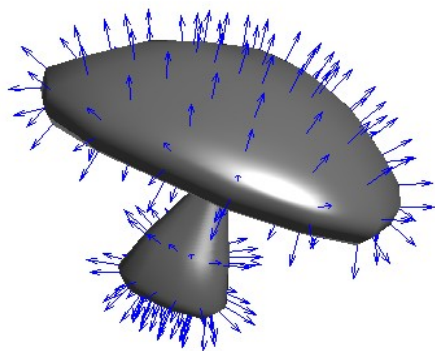
- Vertici: sono gli elementi 0 dimensionali e sono identificabili con punti nello spazio 3D (essenzialmente tre coordinate); alle volte può essere utile associare ai vertici altre caratteristiche oltre alla posizione (tipo il colore)
- Spigoli: sono elementi 1 dimensionali e rappresentano un segmento che unisce due vertici. Di solito non contengono altre informazioni.
- Facce: sono i poligoni bidimensionali, formati da un certo numero di spigoli e di vertici (dimostrare che sono in numero uguale). I vertici o gli spigoli si usano per identificare la faccia; possono contenere altre informazioni (tipo il colore)





# Elementi base

- **Normali:** è fondamentale sapere quale è l'esterno della superficie e quale l'interno; a tal scopo si associa spesso ad una maglia poligonale anche l'informazione sulla normale uscente. Come vedremo questa informazione può essere associata alle facce, come sarebbe naturale, oppure ai vertici (per ragioni che saranno chiare nel seguito).





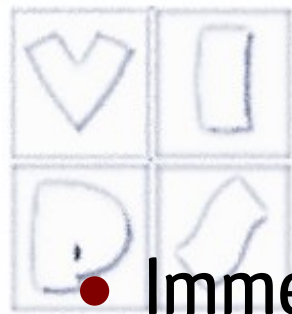
# Struttura della mesh

- I vertici danno informazioni di tipo posizionale, gli spigoli informazioni di tipo connettivo (non c'è informazione spaziale)
- Gli spigoli connettono i vertici, permettendo di introdurre un concetto di “vicinanza” tra vertici e dando le informazioni di tipo topologico (ovvero definiscono un grafo)
- Le facce sono determinate una volta dati i vertici e gli spigoli, quindi non introducono nulla
  - Al più possono avere associati attributi, anche se è raro
- La normale  $\mathbf{n}$  ad una faccia è data dal prodotto vettore di due suoi spigoli consecutivi non collineari
  - attenti al verso: la normale è uscente dal fronte della faccia
- Per un triangolo  $(V_1, V_2, V_3)$  si ha:  $\mathbf{n} = (V_3 - V_2) \times (V_1 - V_2)$



# Ricerca di incidenza

- Una ricerca di incidenza è una procedura che determina tutti gli elementi di un dato tipo che incidono su un certo elemento
- Ad esempio può essere interessante e utile sapere, data una faccia, quali siano i vertici della maglia che incidono su tale di faccia.
- Gli algoritmi di processing dei modelli si basano su calcoli sul vicinato (calcolo curvature, smoothing, ecc.)
  - Dobbiamo renderli efficienti
  - A volte si applicano anche a runtime
- Le strutture dati possono semplificare tali ricerche



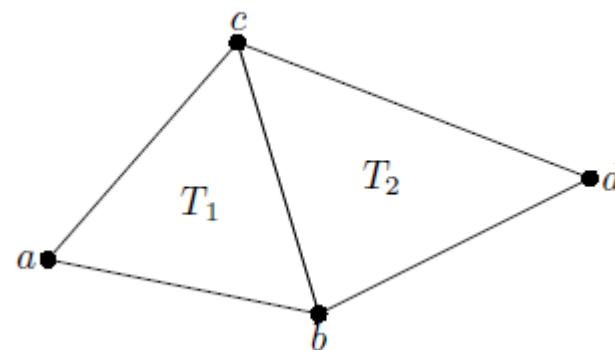
# Strutture dati<sub>t</sub>

- Immediata: specificare tutte le facce della maglia come terne di triplette di coordinate cartesiane. Es.

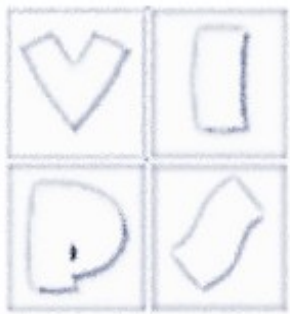
$$T_1 = (a_x, a_y, a_z); (b_x, b_y, b_z); (c_x, c_y, c_z), \dots$$

- Inefficiente; ad esempio i vertici vengono ripetuti nella lista dei poligoni.
- Ricerche di incidenza sono inoltre particolarmente onerose (e a volte non definibili)

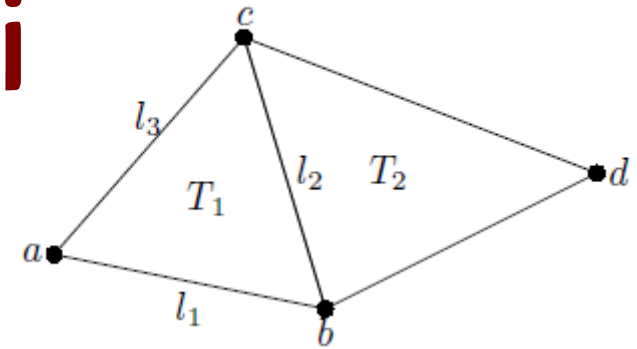
```
typedef struct {  
    float v1[3];  
    float v2[3];  
    float v3[3];  
} faccia;
```







# Lista dei vertici

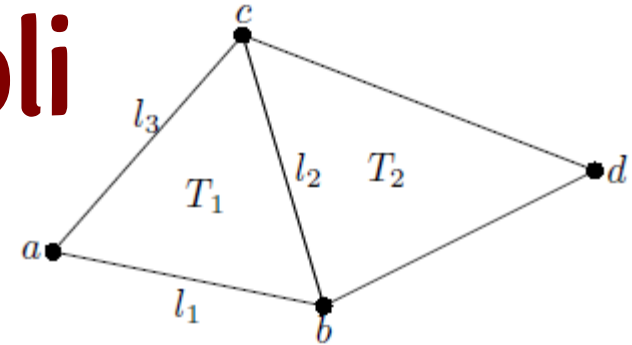


- Si costruisce una lista dei vertici senza ripetizioni
- Le facce sono descritte dai puntatori ai vertici che la compongono (in genere in senso antiorario)
  - In questo modo si elimina la duplicazione dei vertici, ma non quello sugli spigoli; uno spigolo appartenente a due triangoli viene immagazzinato due volte
- Le ricerche di incidenza continuano ad essere complesse

```
typedef  
struct {  
    float  
    x,y,z;  
} vertice;  
typedef  
struct {  
    vertice*  
    v1,v2,v3;  
} faccia;
```



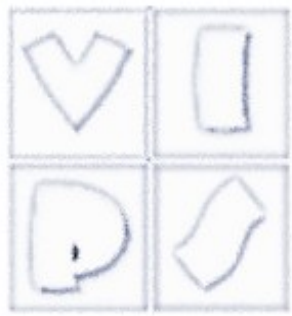
# Lista degli spigoli



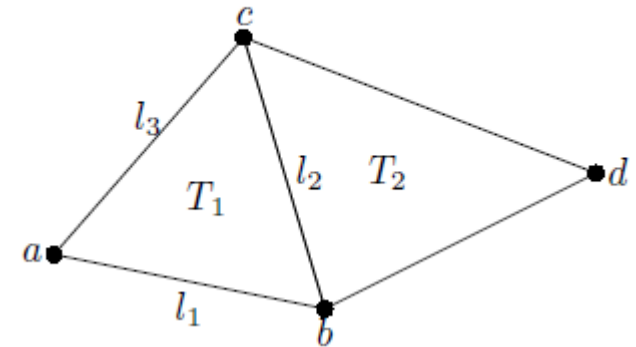
- Lista dei vertici (senza ripetizioni) ed una lista degli spigoli, ciascuno composto dai due puntatori ai vertici incidenti sullo spigolo;
- ciascuna faccia viene descritta infine dai puntatori degli spigoli che la compongono (in genere in senso antiorario)
- In questo modo si elimina la ripetizione sui vertici e sugli spigoli
- Le ricerche di incidenza sono più semplici, ma non tutte

```
typedef struct
{
    float x,y,z;
} vertice;
typedef struct
{
    vertice* in,
    fin;
} spigolo;
typedef struct
{
    spigolo*
    l_1,l_2,l_3;
} faccia;
```

# Lista degli spigoli estesa



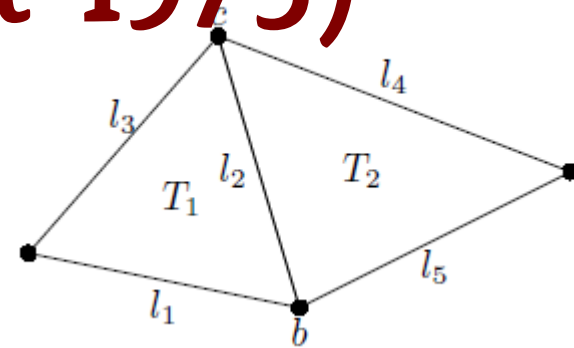
- Un terzo passo può essere ottenuto aggiungendo alla struttura dati degli spigoli anche i due puntatori alle facce incidenti sullo spigolo
- Ad esempio lo spigolo  $l_2$  punta a  $T_1$  e  $T_2$ , oltre che a  $b$  e  $c$  come prima.
- In questo modo si semplificano di molto le ricerche di incidenza spigolo-faccia



```
typedef struct {  
    float x,y,z;  
} vertice;  
typedef struct {  
    vertice* in, fin;  
    faccia* sin,  
    dest;  
} spigolo;  
typedef struct {  
    spigolo* a,b,c;  
} faccia;
```

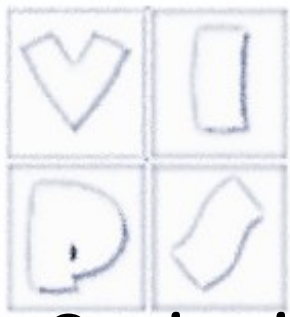
# Winged edge (Baugmart 1975)

- Si aggiungono dei puntatori allo spigolo per rendere più semplice l'analisi delle incidenze.
- L'elemento base è lo spigolo (edge) con le sue due facce incidenti (wings)
  - Lo spigolo  $l_2$  contiene un puntatore ai due vertici su cui incide ( $b$ ;  $c$ ), alle due facce su cui incide ( $T_1$ ,  $T_2$ ) ed ai due spigoli uscenti da ciascun vertice
  - Un vertice contiene un puntatore ad uno degli spigoli che incide su di esso, più le coordinate (ed altro)
  - La faccia contiene un puntatore ad uno degli spigoli che vi incide (ed altro).

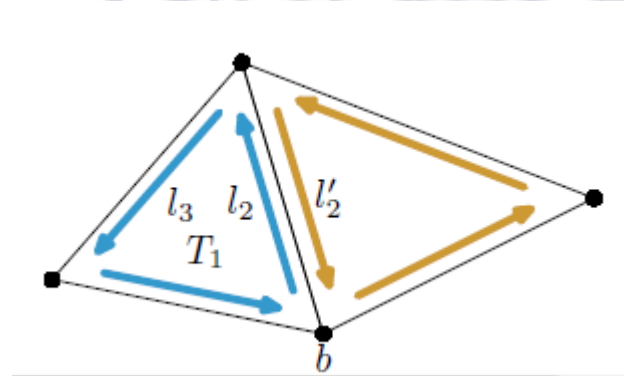


```
typedef struct {  
    we_vertice* v_ini,  
    v_fin;  
    we_spigolo* vi_sin,  
    vi_dstr;  
    we_spigolo* vf_sin,  
    vf_dstr;  
    we_faccia* f_sin,  
    f_dstr;  
} we_spigolo;  
typedef struct {  
    float x, y, z;  
    we_spigolo* spigolo;  
} we_vertice;  
typedef struct {  
    we_spigolo* spigolo;  
} we_faccia;
```





# Half edge



- Ogni spigolo viene diviso in due spigoli orientati in modo opposto
  - Ciascun mezzo spigolo contiene un puntatore al vertice iniziale, alla faccia a cui “appartiene”, al mezzo spigolo successivo (seguendo l'ordinamento) ed al mezzo spigolo gemello
  - Ogni vertice, oltre alle coordinate (e attributi) contiene un puntatore ad uno qualsiasi dei mezzi spigoli che esce da tale vertice
  - Ogni faccia contiene uno dei suoi mezzi spigoli (oltre ad altre caratteristiche quali, ad esempio, la normale)

```
typedef struct {
    he_vertice*
    origine;
    he_spigolo*
    gemello;
    he_faccia* faccia;
    he_spigolo*
    successivo;
} he_spigolo;
typedef struct {
    float x, y, z;
    he_spigolo*
    spigolo;
} he_vertice;
typedef struct {
    he_spigolo*
    spigolo;
} he_faccia;
```



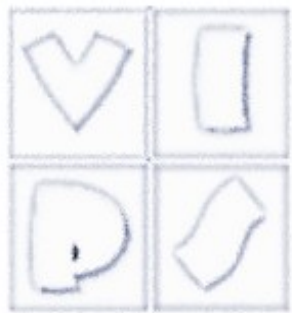
# Note

- La stessa applicazione grafica può far uso di più di una struttura dati
- La rappresentazione con la lista di vertici essendo semplice e leggera è tipicamente usata come formato per i file contenenti la geometria di oggetti
- Le applicazioni grafiche in genere caricano tali file ed usano l'informazione contenuta in essi per riempire una struttura dati più utile ai fini algoritmici (per esempio la half-edge)



# Esercizi

- descrivere una procedura che, data una rappresentazione winged-edge di una maglia triangolare, “stampi” tutti le facce incidenti su un dato vertice  $v$ , assumendo data una procedura `stampaFaccia(we faccia *f)`
- descrivere una procedura che, data una rappresentazione half-edge di una maglia triangolare, “stampi” tutti i mezzi-spigoli uscenti dal vertice  $v$  con una procedura `stampaSpigolo(he spigolo *l)`



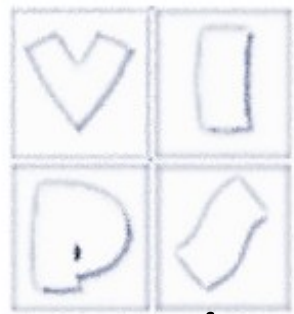
# Formati per mesh

- Esempio .off, obj, .ply, .wrl, x3d, .mesh
- Possono supportare anche diversi tipi di primitive, scene graph
- Esempio .off

```
OFF
4 4 0
-1 -1 -1
1 1 -1
1 -1 1
-1 1 1
3 1 2 3
3 1 0 2
3 3 2 0
3 0 1 3
```

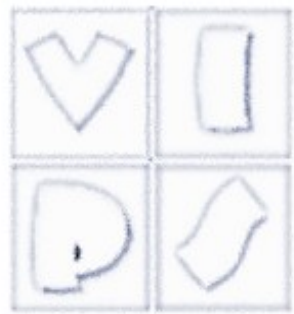






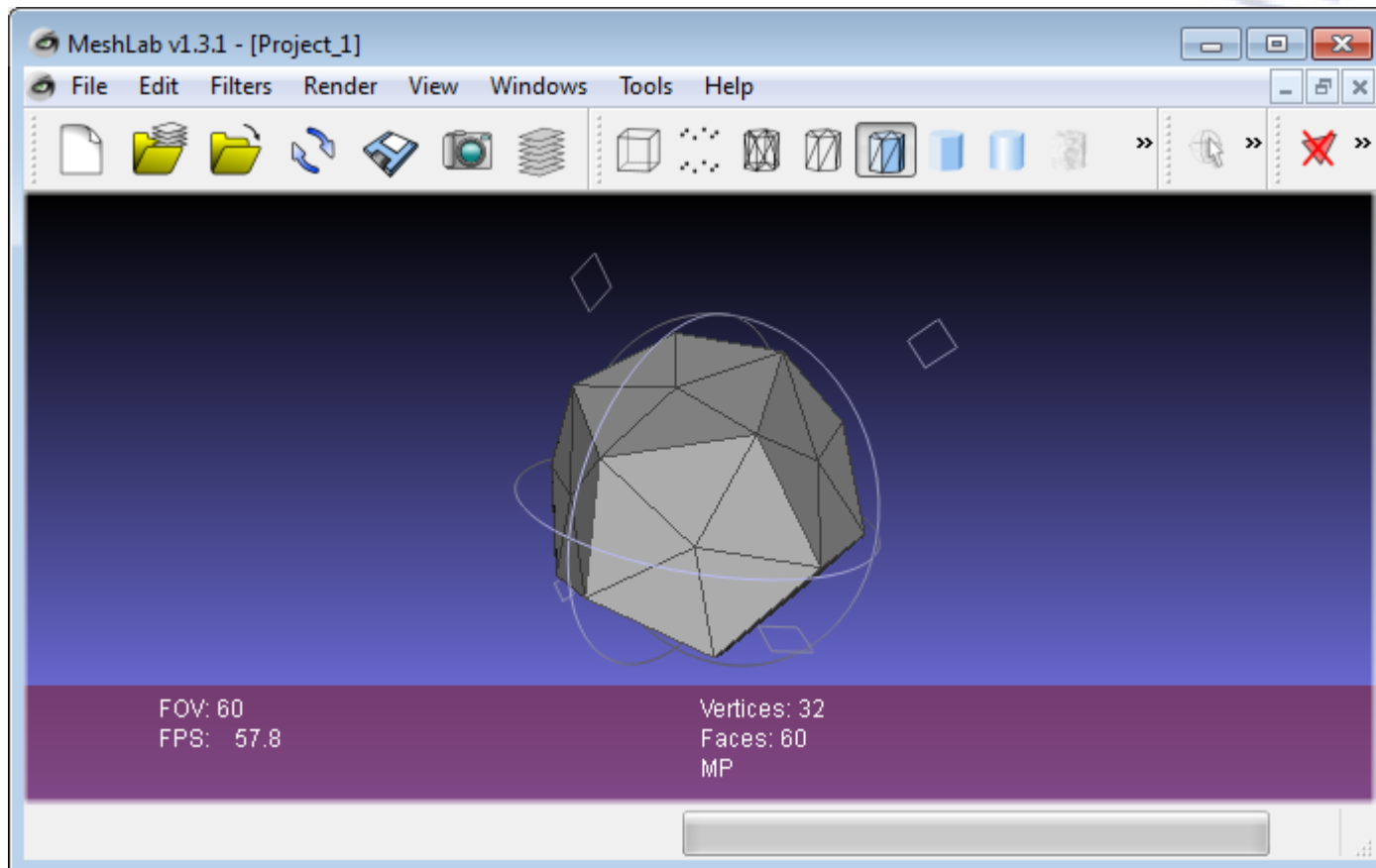
# Mesh (geometry) processing

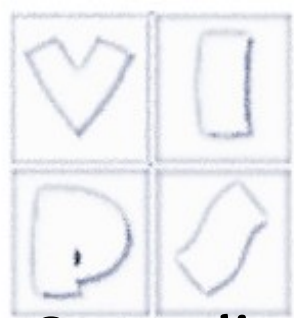
- Esistono molti algoritmi sviluppati in ambito geometria computazionale per il processing dei dati dei modelli
  - Meshing/remeshing: generare nuove rappresentazioni poligonali dalle nuvole di punti: per i modelli acquisiti da scanner non è banale
  - Smoothing (fairing, denoising): eliminare il rumore
  - Riparazione: le mesh derivate da ricostruzione algoritmica possono avere buchi e problemi topologici (autointersezioni, edge e vertici non manifold): algoritmi per tapparli/eliminarli
  - Decimazione: algoritmi per semplificare i modelli e risparmiare memoria, o generare modelli multirisoluzione
  - Densificazione: si cerca di aumentare il dettaglio
  - Shape analysis: riconoscere oggetti/parti, trovare simmetrie, ecc.



# Meshlab

- Strumento open source ottimo per il processing di modelli con algoritmi moderni e aggiornato
  - <http://meshlab.sourceforge.net/>





# Curvatura

Se taglio con piano in tutte le direzioni ho linee con curvatura: i valori massimo e minimo  $k_1$   $k_2$  sono le curvature principali

- Gaussian curvature: prodotto delle curvature principali

$$K(v) = k_1 k_2$$

- Mean curvature

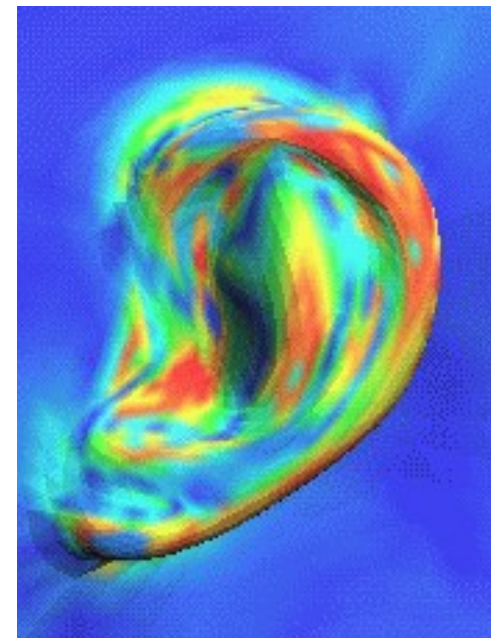
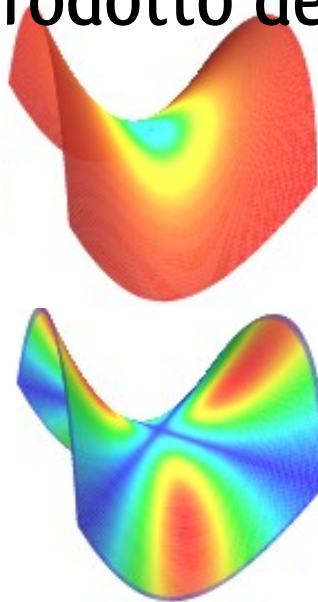
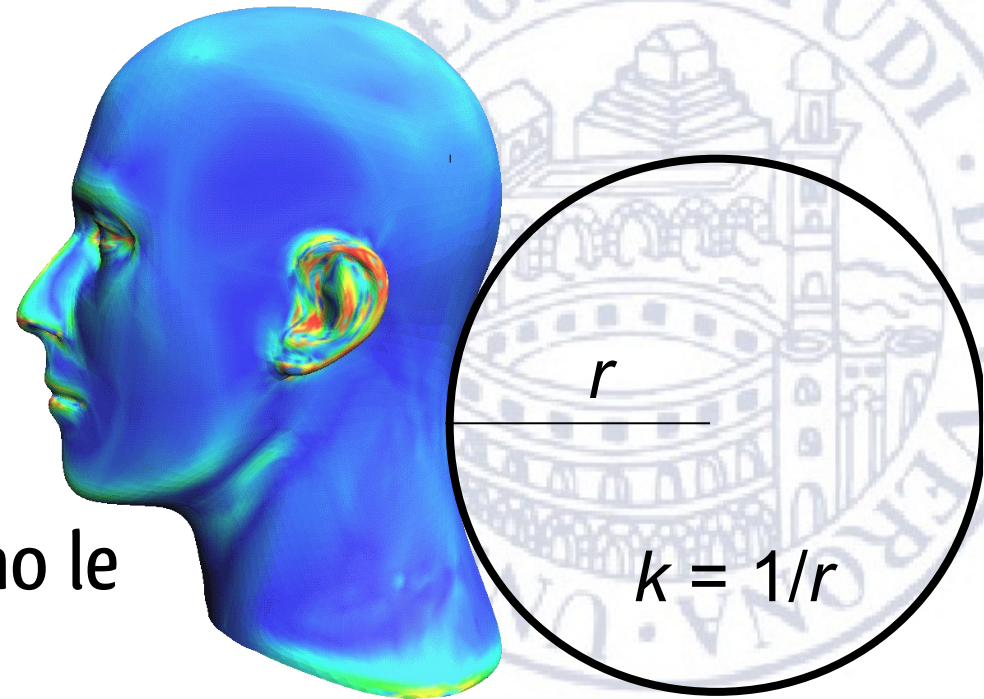
$$H(v) = (k_1 + k_2)/2$$

Si possono stimare sulle mesh e ne caratterizzano le

17/10/14

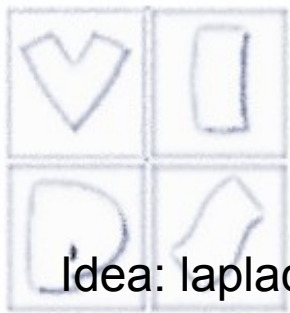
proprietà

Grafica 2014



Meyer, Desbrun, Schroeder, Barr





# Smoothing

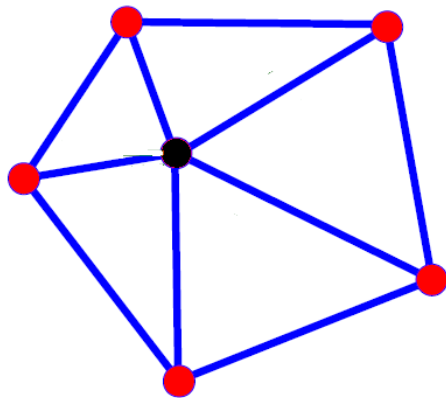
Idea: laplaciano: uso la media dei vicini

Problemi: si contrae

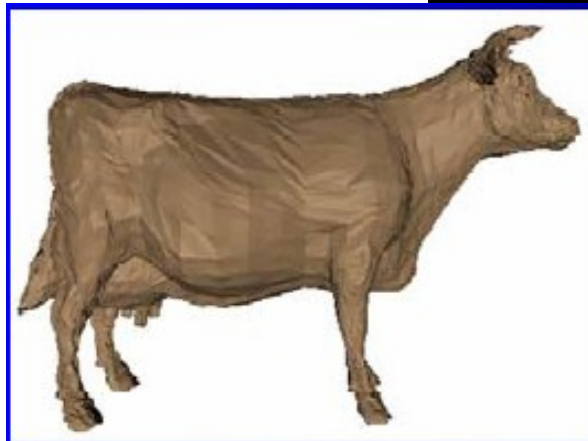
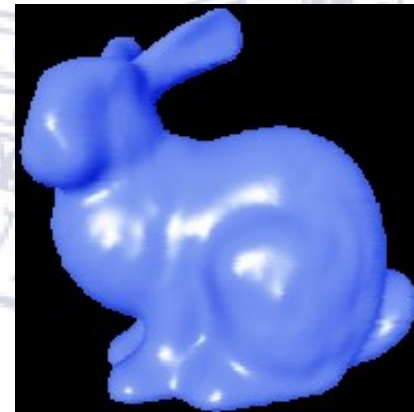
Dipende dalla discretizzazione

$$\Delta v_i = \frac{1}{|N(v_i)|} \sum_{j \in N(v_i)} (v_j - v_i)$$

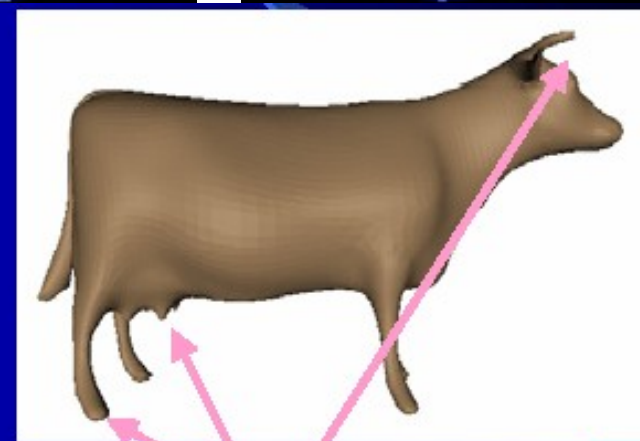
$$v_i = v_i + \lambda \Delta v_i$$



One-ring  
Neighborhood of  $v_i$



Noisy cow model  
with 20K+ faces



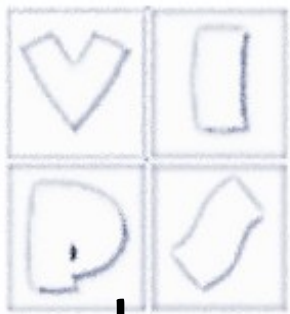
Over-smoothing with  
Laplacian smoothing





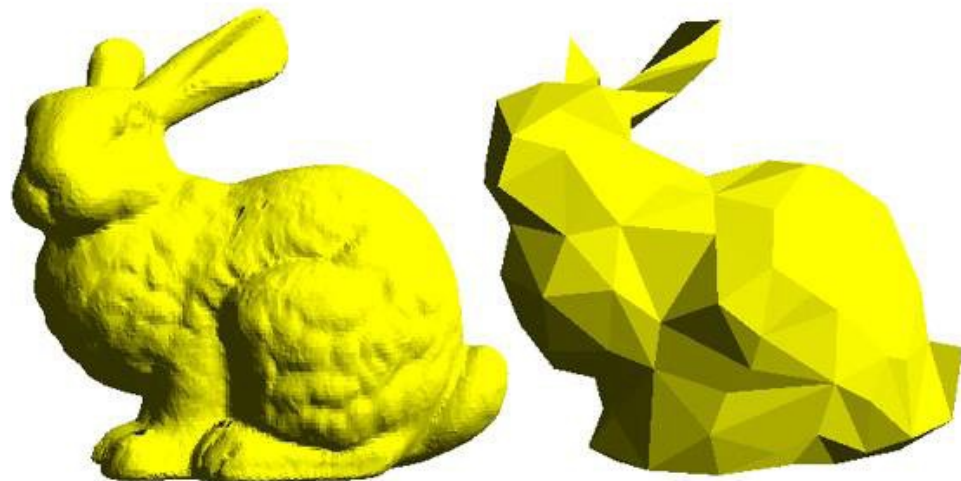
# Smoothing

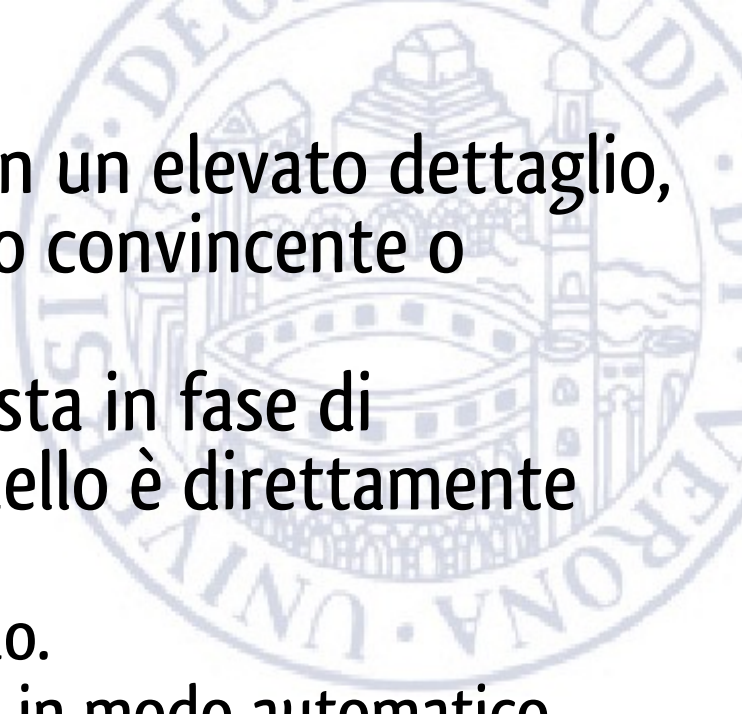

- Per il calcolo ho bisogno dei vicini (1-ring): calcolo di incidenza
- Shrinking: riduce il volume
- Metodo iterativo: quante iterazioni?
- Non preserva discontinuità
  - Stessi problemi dell'image processing (in fondo l'immagine può essere considerata una mesh “strutturata”)
  - Esistono soluzioni

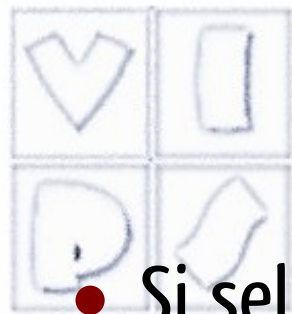


# Semplificazione

- Le maglie poligonali, se molto accurate, possano essere computazionalmente troppo onerose da gestire.
- E' quindi importante poterle semplificare, se necessario.
- Idea semplice: rimuovo un vertice alla volta e riparo il buco.
- Idealmente vogliamo rimuovere il maggior numero possibile di vertici per cui la risultante maglia semplificata sia una buona approssimazione della maglia fine originale (devo calcolare l'errore introdotto!).

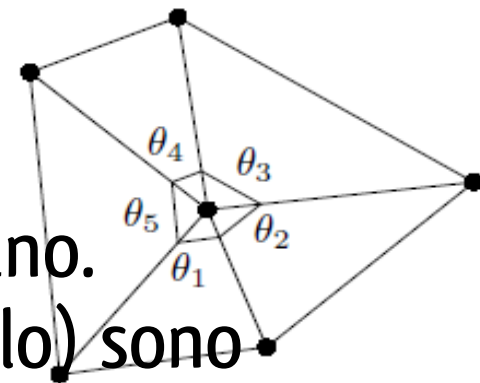


- 
- 
- Molte applicazioni richiedono modelli con un elevato dettaglio, al fine di mantenere un livello di realismo convincente o accuratezza di calcolo.
  - La complessità però non è sempre richiesta in fase di rendering: il costo per disegnare un modello è direttamente legato alla sua complessità
    - Utile avere versioni semplificate del modello.
    - vorremmo che la semplificazione avvenisse in modo automatico.
  - Un algoritmo di semplificazione prende in ingresso una maglia triangolare e ne produce una versione approssimata con meno triangoli. Vi sono due categorie di algoritmi:
    - Decimazione dei vertici.
    - Contrazione iterativa degli spigoli.

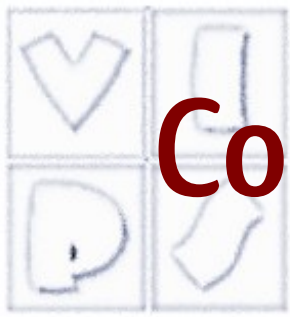


# Algoritmi

- Si seleziona iterativamente con euristiche locali il vertice meno , si rimuove e si ri-triangola il buco. Criteri?
- Esempio: diradare le zone a bassa curvatura.
  - Calcolo sui vertici un'approssimazione della curvatura (Gaussiana), che prende il nome di angle deficit uguale a  $2\pi$  meno la somma degli angoli interni di tutti i triangoli incidenti sul punto
- E' dimostrato che se  $\epsilon(v) = 2\pi - \sum_i \theta_i(v)$  allora il vertice  $v$  e tutti i vertici ad esso connessi giacciono su un piano.
- Si vede che vertici con  $\epsilon(v)$  nullo (o molto piccolo) sono essenzialmente “inutili”







# Contrazione iterativa degli spigoli

- Ad ogni iterazione viene eliminato per contrazione lo **spigolo** di costo minore, ed i costi dei vicini vengono aggiornati.
  - I vari metodi differiscono per la metrica di errore impiegata.
- Una metrica semplice:

$$\varepsilon(\overline{P_1 P_2}) = \frac{\|P_2 - P_1\|}{|n_\ell \cdot n_r|} \quad \text{e } n_\ell \text{ e } n_r \text{ le}$$

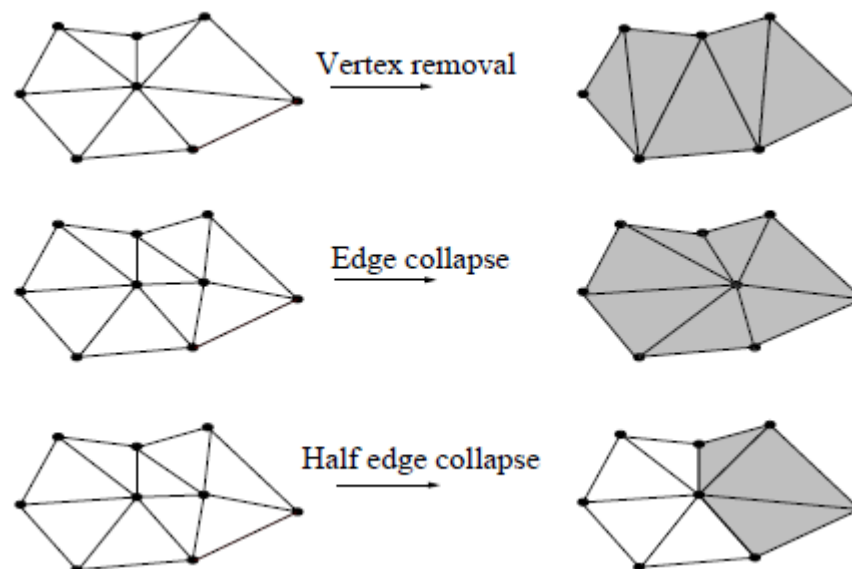
dove  $P_1$  e  $P_2$  sono i vertici  
normali delle facce incidenti.

R.



# Rimozione vertici/spigoli

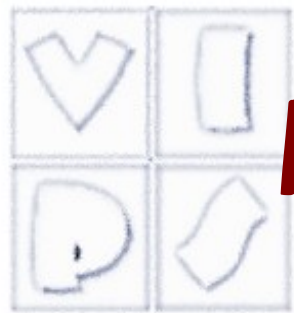
- Varie possibilità:
- vertex removal rimuove un vertice e triangola la cavità
- edge collapse rimuove uno spigolo fondendo due vertici in uno nuovo
- half edge collapse uno dei due vertici rimane fermo.
  - In figura, in grigio i triangoli modificati



# Metodo di Garland-Heckbert

- Procede per contrazione iterativa degli spigoli.
- A ciascun vertice  $v$  è associato un insieme di piani,  $\Pi(v)$ : inizialmente sono i piani definiti dai triangoli incidenti sul vertice. Dopo una contrazione, al nuovo vertice si associa l'unione degli insiemi dei vertici che sono stati contratti
- A ciascun vertice  $v$  è associato un errore  $\Delta(v)$ , pari alla somma dei quadrati delle distanza di  $v$  dai piani di  $\Pi(v)$ .
  - Inizialmente l'errore è 0 per costruzione.

$$\Delta(v) = \sum_{p \in \Pi(v)} (p^T v)^2 = \sum_{p \in \Pi(v)} (v^T p)(p^T v) = \sum_{p \in \Pi(v)} v^T (pp^T) v = v^T \underbrace{\left( \sum_{p \in \Pi(v)} pp^T \right)}_Q v$$



# Metodo di Garland-Heckbert

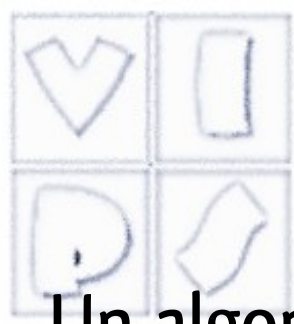
- Per calcolare l'errore di un vertice basta mantenere la matrice  $Q$  (l'insieme dei piani è solo concettuale).
- Il costo di uno spigolo è l'errore  $(\mathbf{v}_1, \mathbf{v}_2) = \bar{\mathbf{v}}^T (Q_1 + Q_2) \bar{\mathbf{v}}$ , dove  $\bar{\mathbf{v}}$  è la posizione del vertice risultante dalla contrazione di  $(\mathbf{v}_1, \mathbf{v}_2)$
- Come scegliere  $\bar{\mathbf{v}}$ ?
  - Scelta migliore: la posizione  $\mathbf{v}$  che rende minima  $\Delta(\mathbf{v})$  (si risolve con un sistema lineare).





# Schema dell'algoritmo:

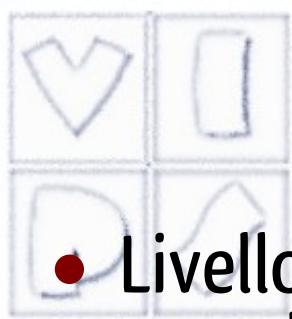
- 1. Calcola le matrici  $Q$  per tutti i vertici della maglia.
- 2. Per ciascuno spigolo  $(\mathbf{v}_1, \mathbf{v}_2)$  calcola la posizione ottima per il vertice dopo l'ipotetica contrazione .
  - L'errore  $\bar{\mathbf{v}}^T (Q_1 + Q_2) \bar{\mathbf{v}}$  è il costo associato allo spigolo.
- 3. Costruisci uno heap con gli spigoli e chiave pari al costo.
- 4. Rimuovi lo spigolo  $(\mathbf{v}_1, \mathbf{v}_2)$  di minor costo dalla cima dello heap, effettua la contrazione  $(\mathbf{v}_1, \mathbf{v}_2) \leftarrow \bar{\mathbf{v}}$
- 5.  $Q_1 = Q_1 + Q_2$  e aggiorna i costi degli spigoli incidenti in  $\mathbf{v}_1$
- 6. Ripeti da 4.
  - Bisogna prestare attenzione a non creare incoerenze nella maglia, come p.es. il fold-over.



# Multirisoluzione

Un algoritmo di semplificazione come questo genera una piramide di modelli a complessità crescente, ovvero una rappresentazione multirisoluzione dell'oggetto.

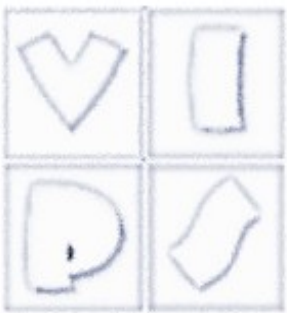
- Si parte da un modello  $M_0$  (maglia triangolare)
- Si semplifica il modello, ottenendone uno nuovo  $M_1$  con meno triangoli
- Si itera il procedimento  $n$  volte, ottenendo una piramide di modelli  $M_0 \dots M_n$ 
  - Tale piramide può essere usata per svariati compiti
- Non archivio tutti i modelli, ma solo  $M_n$  e tutte le mosse di semplificazione inverse.



# Multirisoluzione

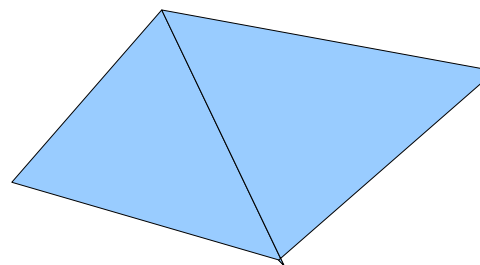
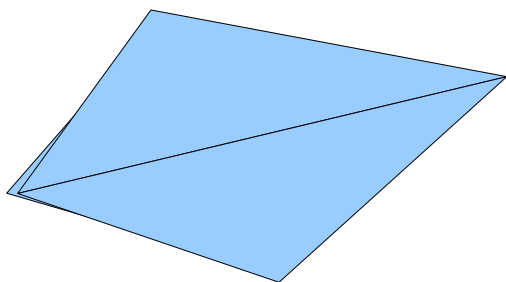
- Livello dinamico di dettaglio: utilizzo la piramide per variare la complessità dell'oggetto a seconda del punto di vista (inutile proiettare un modello di 5000 triangoli su 4 pixel).
- Compressione di una maglia: con una opportuna fase di codifica, lo spazio occupato da  $M_n$  e dalla lista di movimenti di ricostruzione è più piccolo dello spazio occupato da  $M_0$ .
- Rendering progressivo: se trasmetto il modello coarse e poi i dettagli posso fare un rendering progressivo, ovvero ottenere una prima immagine di  $M_n$ , e poi progressivamente “migliorarla” percorrendo la piramide dei modelli. Analogo a quanto siamo abituati a vedere nella trasmissione e visualizzazione di immagini JPEG su internet.

R



# Ottimizzazione mesh

- Miglioro il modello della maglia senza rimuovere o spostare i vertici, ma solo la connettività
- Passo elementare: l'edge flipping (o edge swapping)
- Il criterio di ottimizzazione può riguardare la minimizzazione degli angoli diedrali penalizzando triangoli adiacenti le cui normali sono troppo diverse.





# Modelli nello spazio



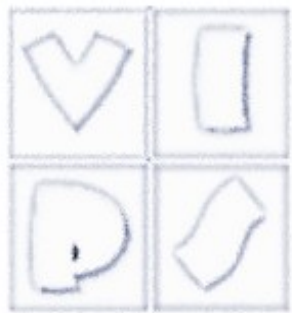


# Nello spazio euclideo

- Spazio tridimensionale
- I vettori identificano direzioni nello spazio (3 componenti)
- I punti determinano posizioni nello spazio. Concetto differente
- Operazioni ammesse:
  - somma e prodotto tra scalari, prodotto di scalari per vettori, somma di vettori, differenza di punti, somma di un punto con un vettore, combinazioni affini.
- Il prodotto scalare permette di determinare la lunghezza dei vettori, la distanza tra punti e l'angolo tra due vettori
- Si possono rappresentare i vettori in una base ortonormale;
  - in questo caso il prodotto scalare tra due vettori è particolarmente semplice: somma dei prodotti delle componenti
- Si può definire un'**origine** del **sistema di riferimento**  $O$  per definire tutti i punti mediante addizione di vettori

R



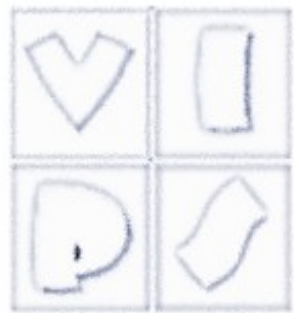


# Prodotto vettore

- E' utile introdurre un'ulteriore operazione tra vettori: **il prodotto vettore**
- Si tratta di un caso particolare di prodotto denominato **esterno**; in tre dimensioni particolarmente semplice:

$$\mathbf{u} \times \mathbf{v} = (u_y v_z - u_z v_y, u_z v_x - u_x v_z, u_x v_y - u_y v_x)$$

- Si dimostra che il prodotto vettore di due vettori  $\mathbf{u}$  e  $\mathbf{v}$  è un vettore ortogonale al piano contenente i due vettori e di modulo pari all'area definita da  $\mathbf{u}$  e  $\mathbf{v}$ . Il verso è scelto in modo tale che  $(\mathbf{u}, \mathbf{v}, \mathbf{u} \times \mathbf{v})$  formino una terna destrorsa

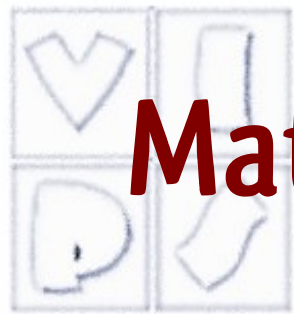


# Matrici come trasformazioni

Data una base ortonormale, si possono rappresentare i vettori con array di 3 elementi.

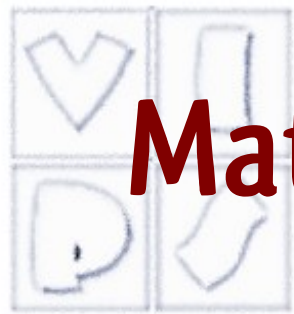
- Le matrici quadrate rappresentano quindi delle applicazioni lineari di uno spazio vettoriale in sé (formano un gruppo non abeliano)
- Tutte le applicazioni lineari di uno spazio vettoriale in sé sono esprimibili tramite matrici quadrate
- L'applicazione di più di una matrice ad un vettore si effettua sfruttando l'algebra delle matrici; ad esempio applicare prima A, poi B ed infine C equivale ad applicare la matrice CBA





# Matrici come cambiamento di base

- Siano  $(e_1, e_2, e_3)$  e  $(e'_1, e'_2, e'_3)$  due basi ortonormali per lo spazio vettoriale
- Sia  $T$  la trasformazione che manda  $e_i$  in  $e'_i$   $i = 1; 2; 3$
- Si può dimostrare che tale trasformazione esiste e che la corrispondente matrice ha  $t_{ij} = e'_i \cdot e_j$
- Il generico vettore  $v = (v_1, v_2, v_3)$  viene trasformato da  $T$  nel vettore  $v' = Tv$
- Fino ad ora abbiamo parlato di vettori in modo indipendente dalle componenti e quindi dalla base



# Matrici come cambiamento di base

- Abbiamo detto che dato uno spazio vettoriale esistono infinite basi. Nella rappresentazione concreta il cambiamento da una base ad un'altra è descritto da una matrice
  - Attenzione: prima abbiamo parlato di applicare una matrice ad un vettore per ottenere un nuovo vettore trasformato;
  - adesso invece vogliamo una matrice che si applichi alle componenti di un vettore per ottenere le nuove componenti rispetto ad un'altra base dello stesso vettore.
- Studiamo il caso concreto di uno spazio vettoriale in 3D

- Le componenti di  $\mathbf{v}'$  nella base  $\{\mathbf{e}_i'\}$  sono uguali alle componenti di  $\mathbf{v}$  nella base  $\{\mathbf{e}_i\}$  (perché lo trasformo assieme alla base).


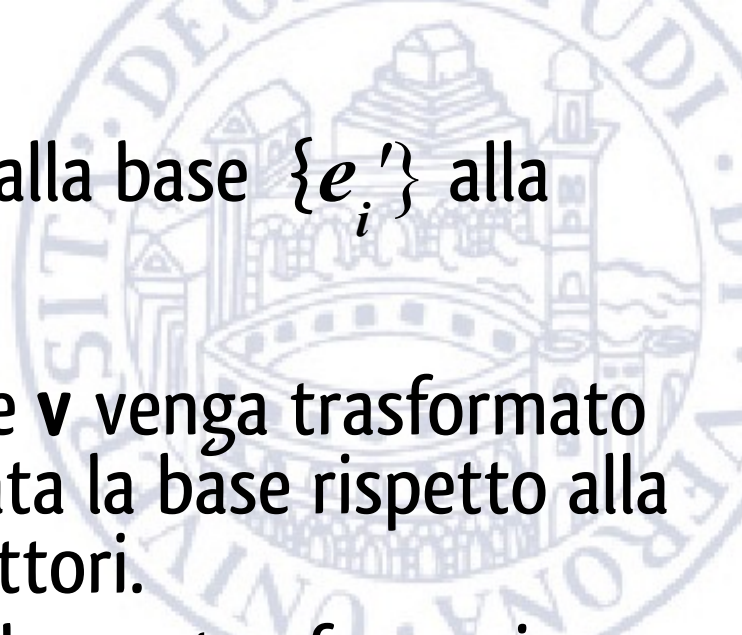
- Sia quindi  $A$  la matrice del cambiamento di coordinate cercata, avremo:

$$\mathbf{v}' = T\mathbf{v}$$

$$A\mathbf{v}' = AT\mathbf{v}$$

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = AT \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

- Dovendo questa relazione vale
  - Abbiamo dunque trovato la matrice  $A$  che trasforma le coordinate di un vettore dalla base  $\{\mathbf{e}_i\}$  alla base  $\{\mathbf{e}_i'\}$

- 
- 
- La matrice che trasforma le coordinate dalla base  $\{e_i'\}$  alla base  $\{e_i\}$  sarà ovviamente  $A^{-1}$ , ovvero  $T$
  - Le componenti di  $v$  cambiano non perché  $v$  venga trasformato in un nuovo vettore, ma perché è cambiata la base rispetto alla quale sono definite le componenti dei vettori.
  - In generale dato un vettore  $(v_1, v_2, v_3)$ , la sua trasformazione in  $(v'_1, v'_2, v'_3)$  tramite la matrice  $B$  può essere vista o come una trasformazione identificata da  $B$  del vettore fissata la base, oppure come un cambiamento di base indotto dalla matrice  $B^{-1}$  tenendo fisso il vettore
  - Nel primo caso si parla di **trasformazione attiva** sullo spazio, nel secondo caso di **trasformazione passiva**





# Coordinate omogenee

- Definiamo il prodotto di un punto per 1 e per 0

$$P \cdot 1 = P \quad P \cdot 0 = 0$$

- In questo modo possiamo definire le coordinate omogenee di un punto e di un vettore rispetto al riferimento  $(e_1, e_2, e_3, O)$

$$V = (v_1, v_2, v_3, 0)$$

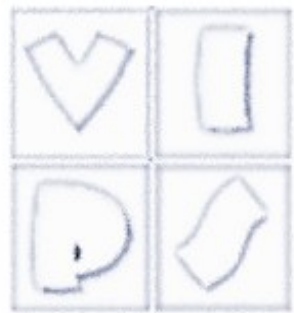
$$P = (p_1, p_2, p_3, 1)$$

- La scelta di 0 e 1 come ultima coordinata per vettori e punti è arbitraria, andrebbe bene qualsiasi valore
  - Tale scelta però permette il **type checking**: si trattano le 4-ple delle coordinate omogenee come vettori quando si effettua una qualsiasi combinazione lineare di punti e vettori, usando le usuali regole, se l'ultima coordinata del risultato è 0, allora il risultato è un vettore; se è pari a 1 allora il risultato è un punto!
  - Se non è 0 né 1, allora si è effettuata una operazione non lecita



# Trasformazioni affini

- Per definire una trasformazione in genere studieremo come si trasforma un punto generico e da questo ricaveremo la matrice di ordine 4 che agisce sulle coordinate omogenee del punto.
- Una trasformazione affine preserva le combinazioni affini, cioè rette parallele vengono trasformate in rette parallele.
- Usando le coordinate omogenee, si può rappresentare ogni trasformazione affine con una matrice (questo è uno dei motivi per usare le coordinate omogenee, l'altro è legato alle proiezioni).



# Traslazione



- Una traslazione determinata dal vettore  $t$  trasforma il punto  $P$  nel punto

$$P' = P + t$$

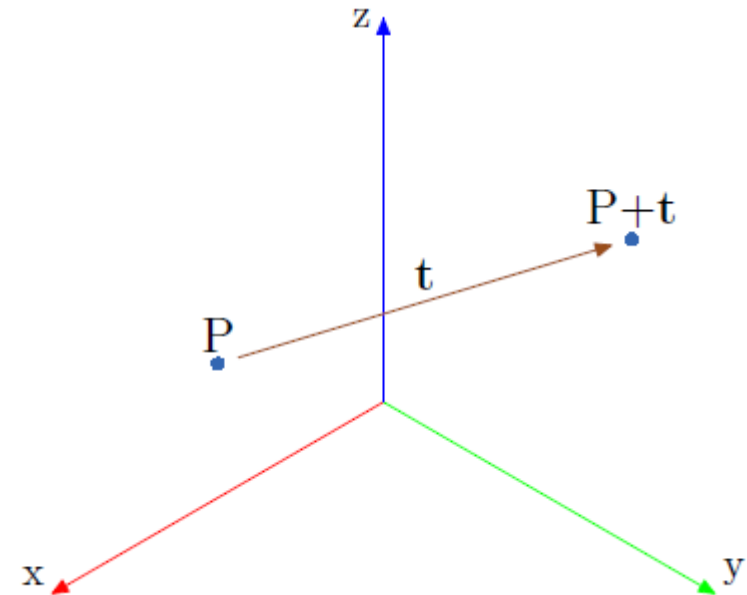
In termini di componenti

$$t = (t_x, t_y, t_z, 0)$$

$$P = (p_x, p_y, p_z, 1)$$

$$P' = (p_x + t_x, p_y + t_y, p_z + t_z, 1)$$

- E' facile vedere che la matrice di trasformazione  $T_t$  per le coordinate omogenee è quella qui a destra



$$T_t = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



# Traslazione



- Si vede subito da questa matrice che i vettori non vengono modificati da una traslazione
- $T_t^{-1} = T_{-t}$
- E' dimostrato che se non si fa uso delle coordinate omogenee, ovvero non si distinguono punti e vettori, non e possibile dare una rappresentazione matriciale alla traslazione.
- Questo giustifica parzialmente lo sforzo di introdurre i concetti di spazio affine e di punto rispetto a limitarsi ai soli spazi vettoriali.



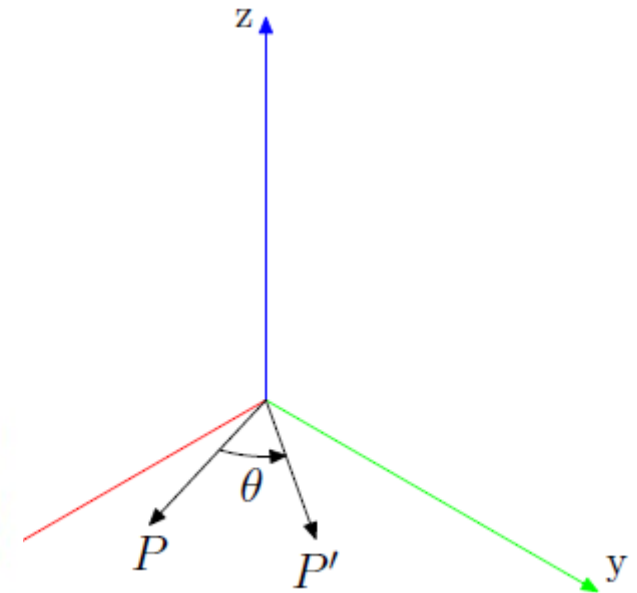
# Rotazione rispetto a un asse

- Una rotazione di un angolo  $\theta$  in senso antiorario (prima regola della mano destra) intorno all'asse  $z$  determina la seguente trasformazione di un punto  $P$  in  $P'$

$$p'_x = p_x \cos(\theta) - p_y \sin(\theta)$$

$$p'_y = p_x \sin(\theta) + p_y \cos(\theta)$$

$$p'_z = p_z$$



- Si può facilmente dimostrare che per rotazioni intorno all'asse  $x$  e  $y$  si hanno le seguenti espressioni:

$$p'_y = p_y \cos(\theta) - p_z \sin(\theta)$$

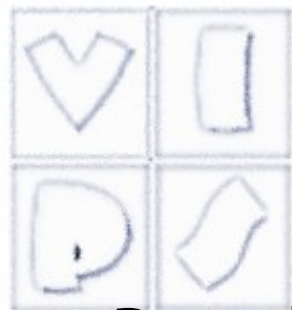
$$p'_z = p_y \sin(\theta) + p_z \cos(\theta)$$

$$p'_x = p_x$$

$$p'_z = p_z \cos(\theta) - p_x \sin(\theta)$$

$$p'_x = p_z \sin(\theta) + p_x \cos(\theta)$$

$$p'_y = p_y$$



# Rotazioni

- Dovrebbe a questo punto essere facile dimostrare che le matrici che rappresentano le rotazioni rispetto agli assi coordinati sono quelle qui riportate
- Da notare che un vettore viene trasformato da una rotazione (a differenza delle traslazioni che lasciano i vettori inalterati)
- Le matrici non commutano

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y(\theta) = \begin{pmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

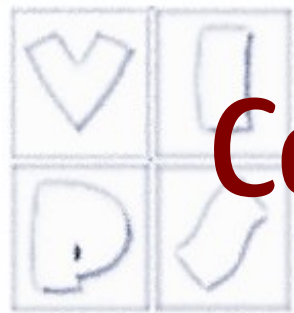
- Le rotazioni rispetto agli assi cartesiani non commutano; provare a ruotare un oggetto di 90 gradi prima rispetto all'asse x e poi rispetto all'asse y. Ripetete quindi l'operazione prima rispetto all'asse y e poi rispetto all'asse x. Risultato?
- Vedremo nel seguito come trattare una rotazione rispetto ad un asse qualsiasi, non solo rispetto ad uno degli assi cartesiani
- Da notare che le rotazioni lasciano inalterati i punti che si trovano sull'asse di rotazione.
- Si può dimostrare che  $R_x(\theta)^{-1} = R_x(-\theta)$  e similmente per gli altri assi
- Si può dimostrare che le matrici di rotazione date sopra sono ortogonali, cioè ad es per asse x:  $R_x(\theta)^{-1} = R_x(-\theta)^T$
- La proprietà di ortogonalità è vera per ogni rotazione, non solo per quelle rispetto agli assi coordinati



# Scalatura

- Traslazioni e rotazioni conservano la lunghezza dei vettori.
  - sottogruppo delle trasformazioni affini chiamato trasformazioni isometriche o rigide.
- Un altro tipo di trasformazione affine che non preserva le distanze è la **scalatura** (ve ne sono anche altre)
- Dato un punto  $P = (p_x, p_y, p_z, 1)$  la trasformazione di scala, o scalatura, lo trasforma nel punto  $P' = (s_x p_x, s_y p_y, s_z p_z, 1)$  dove i valori  $(s_x, s_y, s_z)$  sono i fattori di scala lungo gli assi
- Una scalatura è omogenea se  $s_x = s_y = s_z = s$ 
  - vettori semplicemente allungati ( $s > 1$ ) o accorciati ( $s < 1$ )
  - Un punto, in una scalatura omogenea, viene invece traslato lungo la retta che passa per l'origine e per il punto stesso



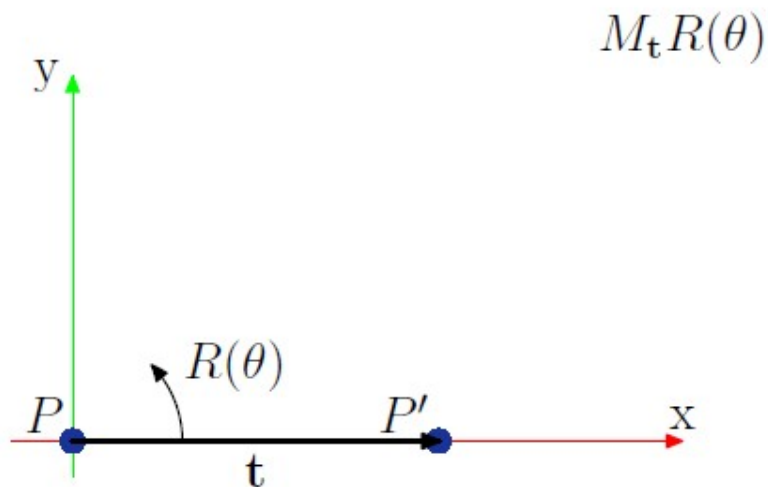
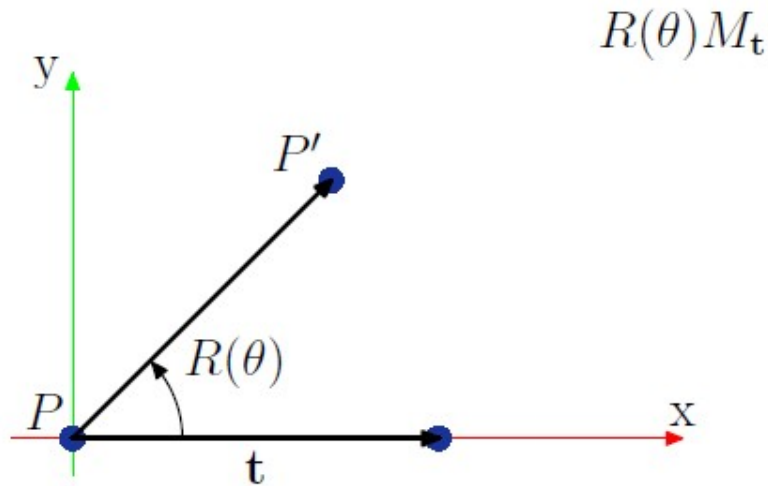
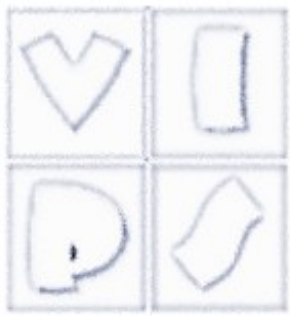


# Composizione di trasformazioni

- Le trasformazioni espresse come matrici si compongono usando semplicemente l'algebra delle matrici
- Date due trasformazioni rappresentate dalle matrici A e B, la composizione di A seguita da B sarà data dalla matrice BA.
  - Importante: notare l'ordine delle matrici; siccome si applica la matrice risultante a sinistra del vettore delle coordinate omogenee, la trasformazione che viene effettuata per prima va a destra.

La composizione di trasformazione si estende immediatamente al caso di più di due matrici  $T = T_n \dots T_1$

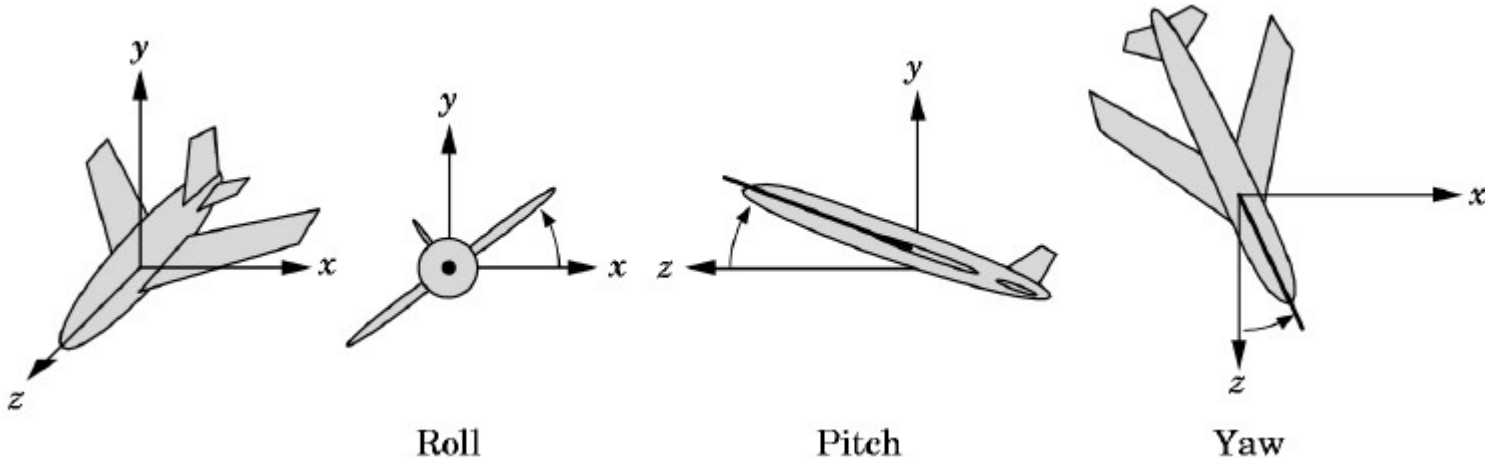
# Non commutatività

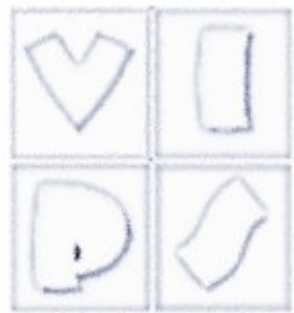


- Esempio: data una traslazione lungo il vettore  $t$  ed una rotazione di un angolo lungo l'asse  $z$ , si ottiene un risultato diverso effettuando prima la rotazione e poi la traslazione o viceversa
- Per rendersene conto basta guardare come viene trasformato nei due casi un punto che in partenza si trova nell'origine

# Rotazioni generiche

- Una rotazione qualsiasi rispetto ad un asse passante per l'origine può essere decomposta nel prodotto di tre rotazioni rispetto agli assi coordinati; i tre angoli prendono il nome di **angoli di Eulero**
- La rappresentazione con gli angoli di Eulero non è univoca, a terne diverse può corrispondere la stessa trasformazione.
  - Una delle rappresentazioni di Eulero impiega gli angoli roll (rollio), pitch (beccheggio) e yaw (imbardata), di derivazione aeronautica.
  - Per convenzione:  $R(\theta_y, \theta_p, \theta_R) = R_y(\theta_y)R_x(\theta_p)R_z(\theta_R)$





# Rotazione con asse generico

- Una rotazione  $R(\theta, \mathbf{u})$  di un angolo  $\theta$  attorno all'asse  $\mathbf{u}=(u_x, u_y, u_z)$  si rappresenta con la seguente matrice (vedere Buss, cap.2), dove  $c = \cos(\theta)$  e  $s = \sin(\theta)$  :

$$R(\theta, \mathbf{u}) = \begin{pmatrix} (1-c)u_x^2 + c & (1-c)u_xu_y - su_z & (1-c)u_xu_z + su_y & 0 \\ (1-c)u_xu_y + su_z & (1-c)u_y^2 + c & (1-c)u_yu_z - su_x & 0 \\ (1-c)u_xu_z - su_y & (1-c)u_yu_z + su_x & (1-c)u_z^2 + c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Per ruotare attorno ad un asse generico, bisogna traslare l'asse nell'origine, ruotare ed infine applicare la traslazione inversa.
  - Data una matrice di rotazione qualunque (ovvero ortogonale e con determinante positivo), si può risalire all'asse  $\mathbf{u}$  ed angolo (formula e dimostrazione sul Buss, cap.2)





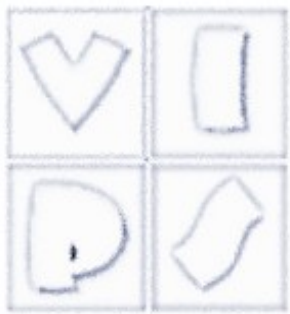
# Trasformazioni

- Ricapitolando: una trasformazione affine generica si scrive in coordinate 3D
- In coordinate omogenee
- Se  $t$  è il vettore di traslazione e  $R$  una matrice di rotazione, la trasformazione in coordinate omogenee è

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} a & b & c & u \\ d & e & f & v \\ g & h & i & w \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}$$



# Cambio di riferimento

- L'idea di cambiamento di base (trasformazione passiva) che abbiamo già affrontato si ripropone negli stessi termini per i sistemi di riferimento
- Dati due riferimenti  $(e_1, e_2, e_3, O)$  e  $(e'_1, e'_2, e'_3, O)$  si tratta di trovare una matrice  $4 \times 4$  che permetta di ottenere le coordinate di un punto rispetto al secondo riferimento date le coordinate dello stesso punto rispetto al primo
- Come nel caso dei cambiamenti di base di un riferimento, se  $T$  è la trasformazione attiva che manda il primo riferimento nel secondo, allora  $T^{-1}$  è la matrice che trasforma le coordinate rispetto al primo riferimento nelle coordinate rispetto al secondo riferimento



# Riferimenti

- Wikipedia voce polygon mesh
- <http://meshlab.sourceforge.net/>
- Angel cap 2.3





# Domande di verifica

- Cos'è un semplice?
- Cosa sono le relazioni di incidenza?
- Cosa si intende per varietà (2-manifold)?
- Come si può ridurre la complessità di una mesh?