Grafica computazionale

Lezione 1

(slide parzialmente basate su
Computer Graphics - MIT Opencourseware
Grafica Computazionale - Massimiliano Corsini – Università di Siena)

Che cos'è la Computer Graphics?

- Produzione di immagini 2D o 3D a partire da dati.
  - I dati sono ottenuti a partire da acquisizione o modellazione oppure possono essere il risultato di altre elaborazioni (ad esempio esperimenti scientifici)

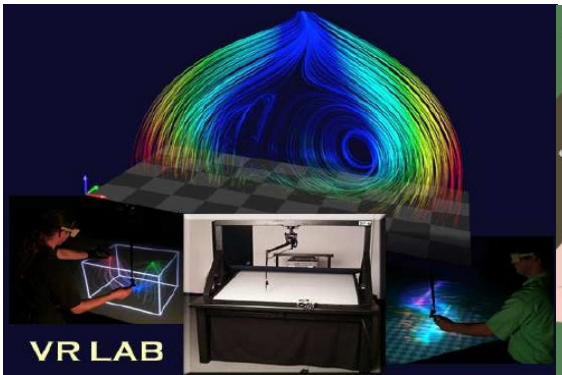
Film

Videogiochi

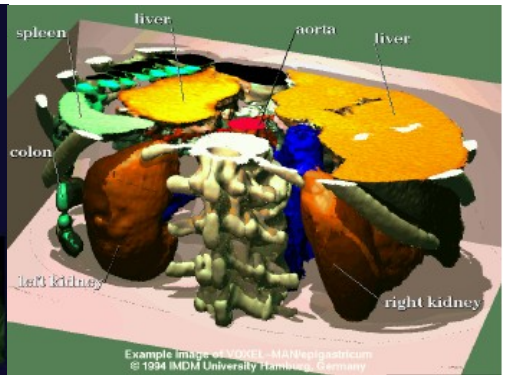
Architettura

CAD/CAM

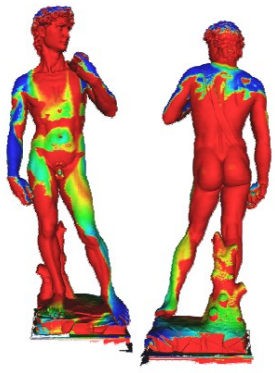
Visualizzazione Scientifica



Visualizzazione Medica



Beni Culturali



Storia

- **Preistoria** Fondamenta teoriche
  - Aritmetica Geometria (Babilonesi ed Egizi)
  - Prospettiva matiematica (Leon Battista Alberti 1435)
  - Geometria analitica (Rene Descartés 1596-1650)
  - Ottica (Eulcide, Alhazen, Snellius, Huygenes, Newton, Fresnel, ...)
- **Anni '60** Basata su **Frame Buffer** (pixel <-> vector)
  - Primi sistemi: Whirlwind, SAGE, Sketchpad ('63)
  - Algoritmi: Bresenham (linee), Coons e Bézier (superfici parametriche), FFT (reinventata da Cooley e Tukey)
  - Precursori: Appel (superfici nascoste e ombre 3D (Ray Casting))
- **Anni '70**
  - Algoritmi 3D: Shading (Gouraud e Phong), zbuffer e texturing (Cattmull), Recusrive Ray tracing (Whitted)
  - Hardware: Evans & Sutherland Picture System (supporto clipping e projection), Stampanti Laser (Xerox PARC), successo primi personal (Apple I & II)
  - Software: “Paint” (Xerox PARC), sviluppo C e Unix (Bell labs), primi arcade popolari (Pong e Pac Man)
- **Anni '80**
  - Algoritmi: Binary Space Partitioning, Montecarlo Ray Tracing, Radiosity
  - Hardware: Silicon Graphics (trasformazioni e rasterizzazione), VGA
  - Software: Postscript e Photoshop (Adobe),

- **Primi anni '90**
  - Software: OpenGL, MPEG, Mosaic (Web)
  - Hardware: SGI (16MB RAM, Framebuffer 24b, hardware z-buffer e Gouraud shading, dal 94 texture mapping), PC decenti ma no 3D, prime console 3D (Sony Playstation, Nintendo 64, ...)
- **Fine anni '90**
  - Hardware: esplode il mercato del hardware grafico per PC (orientato su texture più che trasformazioni). In pochi anni soppianta il mercato Workstation e porta SGI al fallimento.
  - Software: esplode il mercato videogiochi 3D (Doom, Quake,...). Postproduzione digitale diventa pervasiva nel cinema.
- **Anni 2000**
  - Hardware/Software: continua trend di sviluppo.
    Schede ad alte prestazioni, scanner, stampanti e fotocamere digitali e connessione internet virtualmetne in ogni casa.
    Uso di Visione Artificiale e postproduzione realtime in TV
  - Algoritmi: passaggio da pipeline fissa a shaders programmabili, GPGPU.

Tecniche

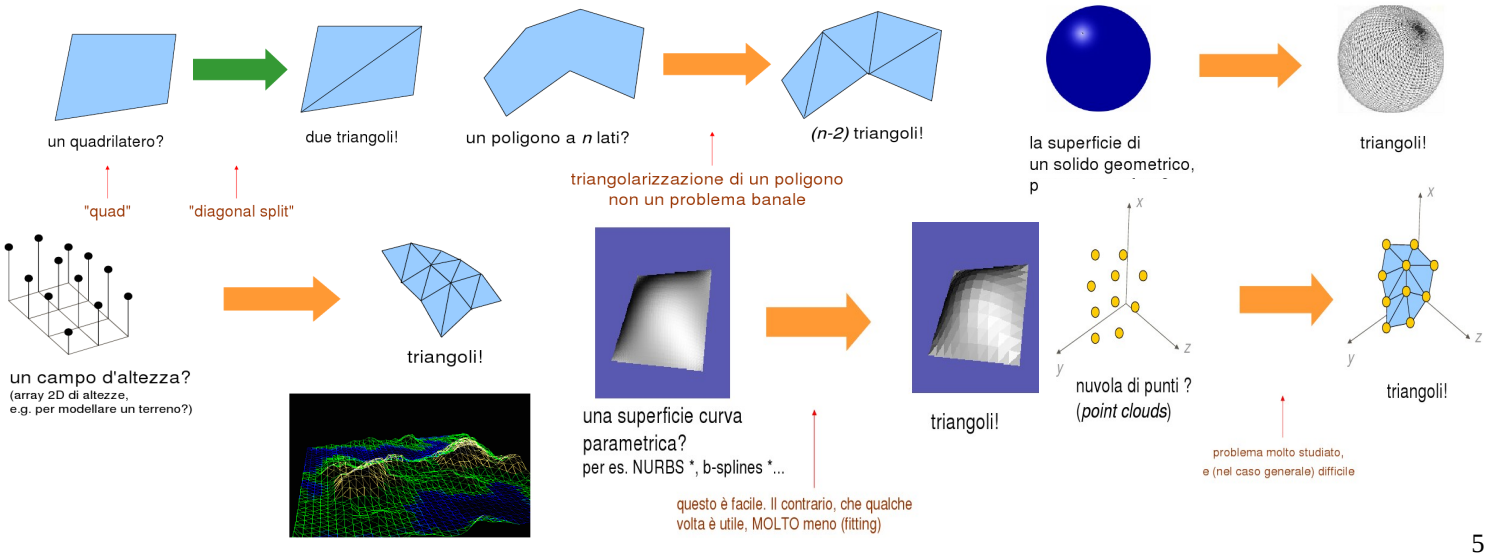
- Raytracing
- **Rasterization based**
- Image based (per es. light filed)
- Radiosity
- Photon Mapping

Proiettiamo la geometria del modello sul frame-buffer (dalla geometria allo schermo)

Usata dal hardware grafico interattivo moderno

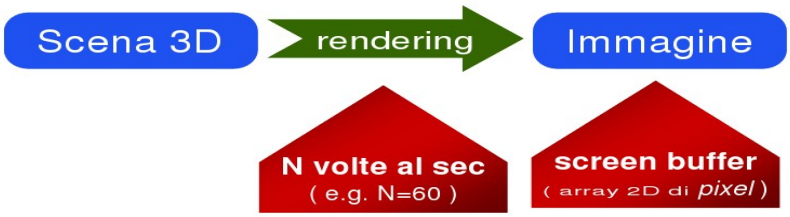
Rasterizzazione di triangoli

- Tutto sia composto da triangoli (3D) (...o al massimo puni e linee)



Hardware Grafico interattivo

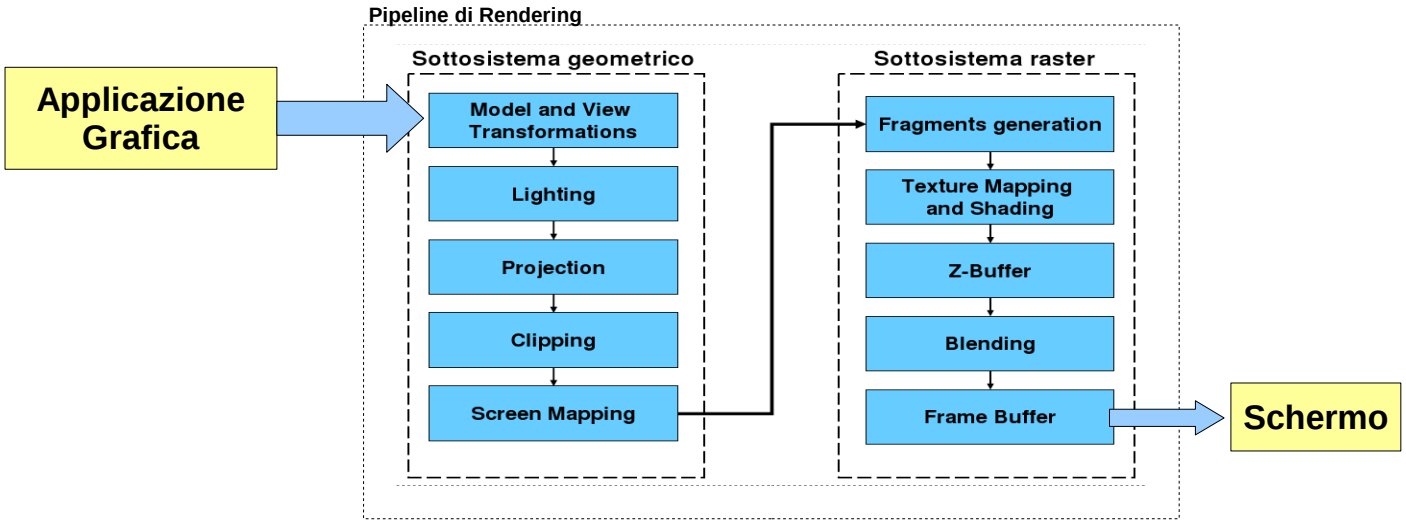
- Caratterizzato dalla cosiddetta pipeline di rendering
  - La pipeline di rendering è la serie di stages di elaborazione che I dati della scena attraversano per diventare immagine



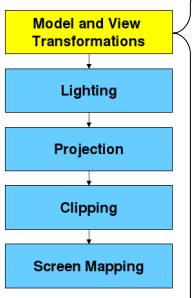
- Vantaggio di disporre di hardware specializzato per la grafica: efficienza
  - computazioni più ripetute hard-wired nel chipset
  - resto dell'applicazione libera di utilizzare la CPU e RAM base
  - sfruttamento del parallelismo implicito nel problema di rendering
- Svantaggio: rigidità
  - vincola l'approccio usato per fare rendering...
- La moderna pipeline di rendering è programmabile
  - Vertex shaders
  - Pixel (Fragment) shaders

Pipeline di Renering

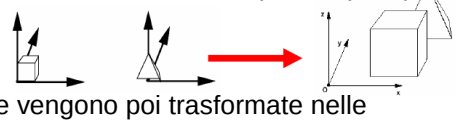
- La pipeline di rendering si suddivide in due parti
  - **Sottosistema geometrico:** porta la geometria delmodello nelle coordinate dello schermo
  - **Sottosistema raster:** accende I pixel dello schermo del giusto colore in funzione della geometria, dell'illuminazione e delle texture.
- Nei sistemi programmabili queste due parti sono affidate a due *shader*:
  - **Vertex shader:** fa i conti delle trasformate geometriche per ogni vetice del modello
  - **Pixel (Fragment) shader:** calcola il colore per ogni pixel



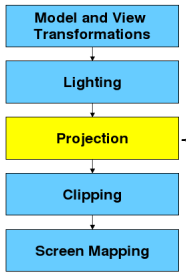
Sottosistema Geometrico



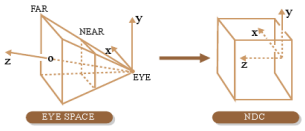
Oggetti definiti nel loro sistema di coordinate *(object space)*. Le trasformazioni di modellazione portano il modello prima nel sistema comune del mondo *(world space)*

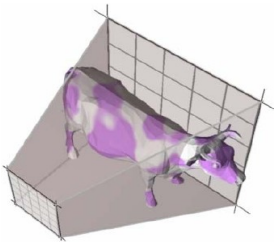


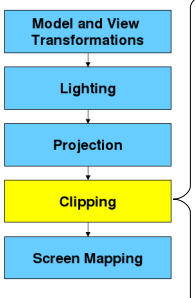
Queste vengono poi trasformate nelle coordinate del punto di vista dell'osservatore *(View coordinate system)*



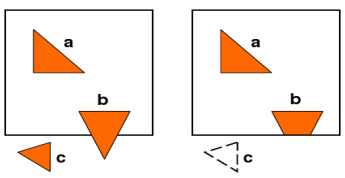
Tutto quello che è all'interno del volume di vista (view frustum) viene proiettato. Il view frustum viene così trasformato in un cubo di lato unitario *(normalized device coordinates).*

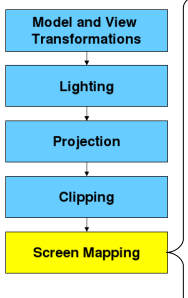




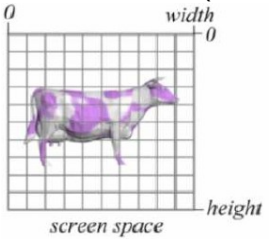


Tutte le primitive geometriche al di fuori del cubo unitario (e quindi al di fuori del volume di vista) sono scartate e non proseguono l'elaborazione. Quelle che intersecano il cubo sono modificate secondo l'intersezione.



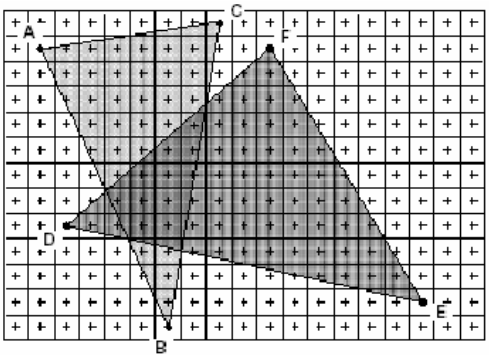


La scena viene mappata dalle NDC sulle coordinate dello schermo (screen space)



Sottosistema Raster

- Si occupa di passare dalla proiezione continua in screen space ai pixel dell'immagine visualizzata.
- Più precisamente si parla di frammenti poichè alcuni frammenti diventeranno pixel dell'immagine finale mentre altri no
  - I frammenti sono quindi dei pixel potenziali
- Si occupa inoltre di rimuovere le superfici nascoste (Z-buffer) e applicare tessiture
- Nei sistemi moderni parte dell'illuminazione è deferita dal sistema geometrico al sistema raster



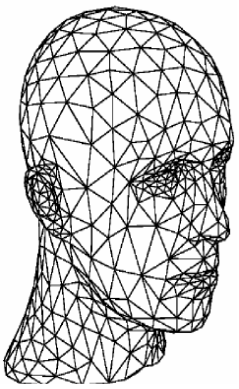
Programma del corso (in evoluzione)

- I settimana: sottosistema geometrico (Vertex shared)
  - Oggi e domani parleremo di geometria e del funzionamento delle trasformazioni
  - Venerdi in laboratorio (prima esercitazione)
- II settimana: sottosistema Raster (Pixel shader)
  - Lunedi e martedi ottica e modelli di illuminazione e interpolazione
  - Venerdi in laboratorio (seconda esercitazione)
- Pausa
- Venerdi 9/10 Laboratorio (terminiamo le prime due esercitazioni e discutiamo una soluzione)
- III settimana (12/10 a 16/10): Algoritmi di rasterizzazione e visibilità. Strutture dati spaziali
- IV settimana (19/10 a 23/10): Textures, image based rendering e ombre
- V settimana (26/10 e 27/10): cenni a modelli di illuminazione globale (Ray tracing, Radiosity e Photon tracing)

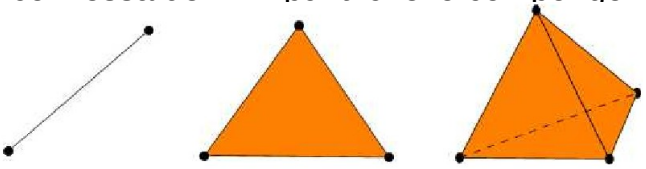
Rappresentazioni

- La rappresentazione di modelli 3D si può suddividere in due categorie:
  - Boundary-based: è la superfice dell'oggetto 3D ad essere rappresentata (boundary representation or b-rep)
    Esempi: mesh poligonali, rapp. Implicite e parametriche
  - Volume-based: è il volume ad essere rappresentato
    Esempi: voxels, CSG.
- Modellazione Geometrica può essere:
  - Automatica (a partire da immagini Computer Vision)
  - Manuale (Maya©, Rhino3D©, 3DStudio Max©)
  - Procedurale (frattali, grammatiche)
- Una volta che la geometria di un oggetto è rappresentata questa può essere renderizzata per produrre l'immagine finale.

Mesh Poligonali



- La superfice dell'oggetto 3D è rappresentata da un insieme di
- poligoni nello spazio.
- Una mesh è un insieme di vertici, lati e facce.
- Matematicamente, una mesh è una tupla (K,V) dove V è l'insieme dei punti nello spazio (vertici), e K, insieme dei complessi simpliciali. In pratica contiene le informazioni sulla connettività (topologia) tra I punti contenuti in V (ossia lati e facce).
- Un simplesso di ordine k è la combinazione convessa dei k+1 punti che lo compongono
  - Simplesso di ordine 1 -> segmento
  - Simplesso di ordine 2 -> triangolo
  - Simplesso di ordine 3 -> tetraedro

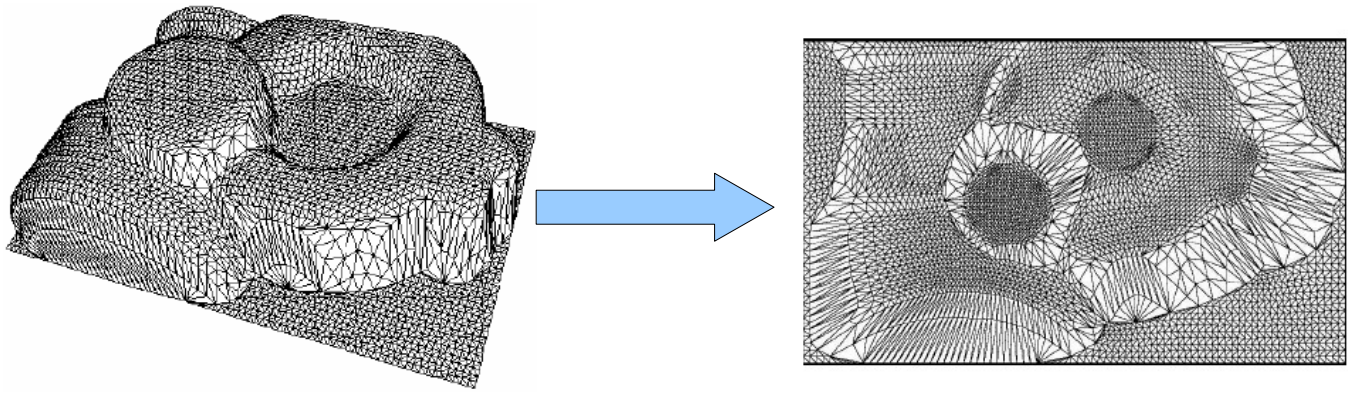


- L'insieme dei simplessi forma un complesso simpliciale.
- Le mesh poligonali sono un'approssimazione discreta di una superficie.
- Può essere vista come il minimo comune denominatore di tutte le altre rappresentazioni.
- Tipicamente vengono utilizzate particolari tipi di mesh:
  - Mesh triangolari: ossia mesh le cui facce sono triangoli (moderne schede grafiche).
  - Mesh quadrangolari: ossia mesh le cui facce sono quadrilateri (es: Geographic Information System (GIS)).

Limitazioni

- Facce Planari
- Non sono una rappresentazione compatta – i.e. modelli con elevatissimo dettaglio richiedono grandi quantità di dati
- Editing diretto è “difficile”
- L’elaborazione con algoritmi geometrici è intrinsicamente onerosa
- Non hanno una parametrizzazione naturale
  (la parametrizzazione di mesh è un settore ancora attivo di ricerca)

- La parametrizzazione di una mesh consiste nel trovare una funzione capace di mappare i vertici della mesh in un dominio planare.
  - La parametrizzazione introduce inevitabilmente distorsioni. Le tecniche odierne cercano di preservare una o più proprietà della superifice come angoli o aree.

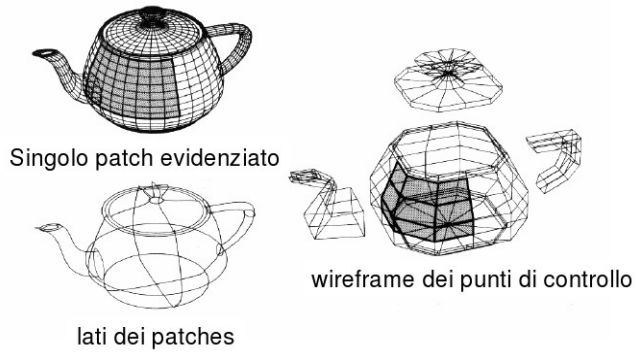


Rappresentazione implicita

- La superficie viene definita in termine delle sue coordinate cartesiane: f (x, y, z) = 0
  - Esempi : Sfera di raggio r ($x^2 + y^2 + z^2 - r^2 = 0$) Piano generico : $ax + by + cz + d = 0$
  - Rappresentazione estremamente compatta (!!)
  - Pochi utilizzi (design meccanico)

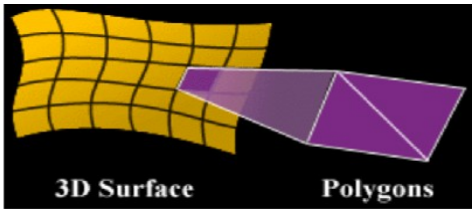
Rappresentazione Parametrica

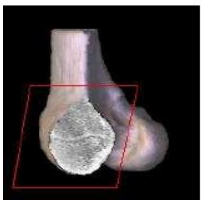
- La superficie viene definita in forma parametrica utilizzando tre funzioni bi-variate:
  $$S(u,v) = (X(u,v), Y(u,v), Z(u,v))$$
  dove u e v variano tra 0 ed 1
  - Esempi: Bézier, B-Spline, NURBS
  - Le curve parametriche sono molto flessibili
  - Non sono vincolate ad essere funzioni
  - La parametrizzazione è ovvia.
  - L’elaborazione geometrica è analitica.
  - Possono essere convertite facilmente a mesh poligonali

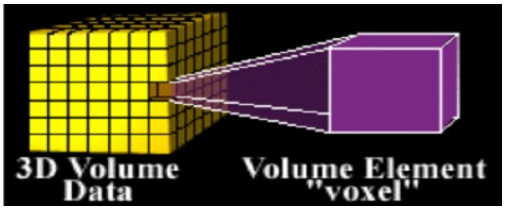


  - Le superfici che si ottengono sono smooth e praticamente qualsiasi forma può essere rappresentata con un buon grado di approssimazione.
  - E’ una rappresentazione efficiente dal punto di vista della memoria (è sufficiente memorizzare i punti di controllo).

Modellazione di Volumi (Voxels)







- Sono prevalentemente utilizzati quando è importante l'informazione volumetrica (ad esempio nelle applicazioni mediche).
- Richiedono enormi quantità di memoria (e.g. 512x512x512 = 128 MBits).
- Voxels sono utili per rappresentare isosuperfici.
- La conversione da voxel a mesh poligonale è "facile" (algoritmo Marching Cubes). Il viceversa è più complesso e tipicamente computazionalmente costoso.

Geometria Solida Costruttiva (CSG)

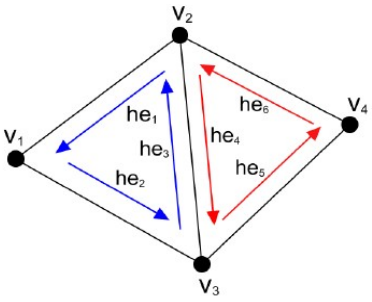
- La forma finale dell'oggetto viene ottenuta combinando I volumi occupati da un insieme sovrapposto di forme tridimensionali tramite operazioni booleane (unione, differenza, intersezione).

Rappresentazione delle Mesh

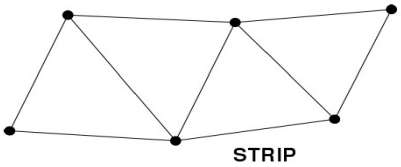
- **Rappresentazione immediata**: Tutte le facce sono memorizzate come terne di vertici.
  - Ad esempio il triangolo T si può rappresentare come T = {(v1x , v1y , v1z), (v2x, v2y, v2z), (v3x, v3y, v3z)}.
  - E' semplice ma non è efficiente, ad esempio i vertici sono ripetuti.
  - Le query sono particolarmente onerose.
- **Lista dei vertici**: Si utilizza una lista dei vertici senza ripetizione ed una lista delle facce che riferiscano la lista dei vertici.
  - La faccia T riferisce (tramite puntatore o indice) ai vertici da cui è composta. In questo modo si elimana la duplicazione dei vertici ma non quella dei lati.
  - Le query sono onerose anche in questo tipo di rappresentazione.
- **Lista dei lati**: Si utilizza una lista dei vertici (senza ripetizioni) ed una lista dei lati. Ogni lato riferisce i vertici che lo compongono. Le facce sono descritte riferendo i lati che le compongono.
  - Vertici e lati non sono ripetuti.
  - Le query sulla mesh cominciano a farsi più semplici ed efficienti.
- **Lista dei lati estesa**: Per rendere alcune query più efficienti si inserisce esplicitamente nella rappresentazione le due facce incidenti sul lato.
  - Ovviamente a beneficiarne sono le query che sfruttano l'ncidenza lato-faccia.

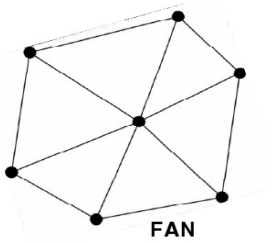
Rappresentazione delle Mesh

- **Winged edge**: I dati di incidenza vanno a far parte ancora più pesantemente della rappresentazione.
  - Ogni lato contiene, oltre ai vertici, due puntatori alle facce adiacenti più I puntatori ad i lati uscenti.
  - Ogni vertice contiene un puntatore ad uno dei lati che incide su di esso e le sue coordinate.
  - Una faccia è definita da un puntatore ad uno dei lati che vi incide.
- **Half edge**: Ogni lato viene diviso in due semi-lati orientati in modo opposto (da cui il nome).
  - Ciascun half-edge contiene un puntatore al vertice iniziale, al mezzo lato gemello (secondo un ordinamento dato) ed al mezzo-lato associato.
  - Ogni vertice contiene un puntatore ad uno qualsiasi dei mezzi lati uscenti e le coordinate.
  - Ogni faccia contiene uno dei suoi mezzi lati.

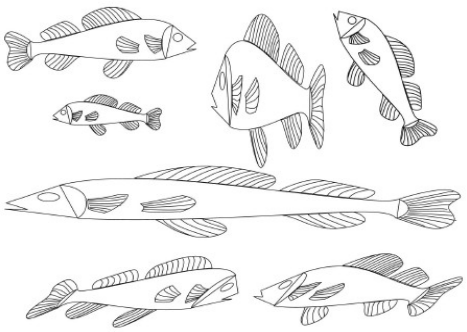


- **Fan e Strip di triangoli**: Sono particolari gruppi di triangoli utili per ottimizzare le performances di rendering.
  - Fan: è un gruppo di triangoli con un vertice in comune.
  - Strip: è un gruppo di triangoli con un lato in comune.

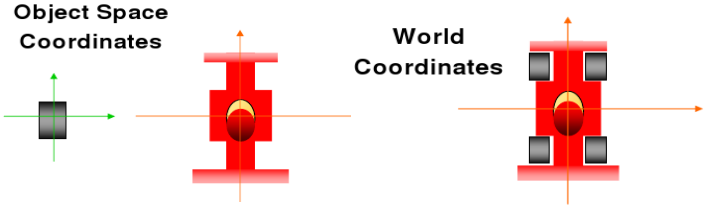




Trasformazioni Geometriche



- Le trasformazioni geometriche permettono di istanziare una stessa geometria con attributi (posizione, orientamento, fattori di scala) diversi.
- Ci permettono, di definire un oggetto tridimensionale componendolo con altri oggetti. Ogni oggetto, a partire dal proprio sistema di riferimento (object space), viene trasformato opportunamento in un sistema di riferimento comune (world space) per andare a far parte dell'oggetto finale.



- Le trasformazioni geometriche sono lo strumento che consente di manipolare punti e vettori all'interno dell'applicazione grafica;
- Le trasformazioni geometriche sono funzioni che mappano un punto (o un vettore) in un altro punto (o un altro vettore);
- La trasformazione di una mesh poligonale si riduce alla trasformazione dei vertici che la compongono **nel rispetto della connettività originale** e di altre invarianti geometriche (per esempio la collinearità)
-

Trasformazioni e invarianti

- Le trasformazioni geometriche di base (Euclidee) sono:
  - Traslazione
  - Rotazione
  - Riflessione
  - Conservano le lunghezze e gli angoli
- Aggiungiamo
  - scala (conserva angoli e rapporto tra lunghezze)
  - Aspect ratio (conserva rapporto tra lunghezze)
- Deformazioni di vista (ortogonali e prospettiche)

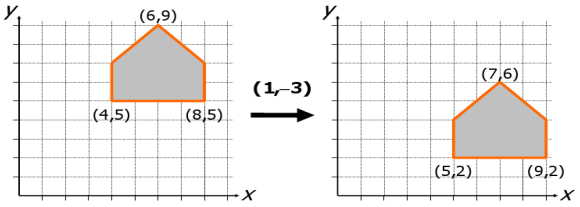
	Lunghezze	Angoli	Rapp. lunghezze	Colinearità
Traslazione	V	V	V	V
Rotazione	V	V	V	V
Scalatura uniforme	X	V	V	V
Scaltura non uniforme	X	X	V	V
Shearing	X	X	V	V
Proiezione Ortogonale	X	X	V	V
Proiezione Prospettica	X	X	X	V

Trasformazioni

- **Traslazione:** Traslare una primitiva geometrica nel piano significa muovere ogni suo punto P(x,y) di dx unità lungo l'asse x e di dy unità lungo l'asse y fino a raggiungere la nuova posizione P'(x', y') dove:

$$x' = x + d\,x \qquad y' = y + d\,y$$

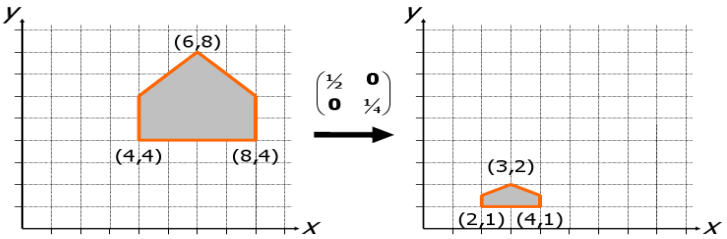
- In notazione matriciale: **P'=P+T**



- **Scala:** Scelto un punto **C** (punto fisso) di riferimento, scalare una primitiva geometrica significa riposizionare rispetto a **C** tutti i suoi punti in accordo ai fattori di scala sx (lungo l'asse x) e sy (lungo l'asse y) scelti. Se il punto fisso è l'origine O degli assi, la trasformazione di P in P' si ottiene con:

$$x' = s\,x \cdot x, \qquad y' = s\,y \cdot y$$

- In notazione matriciale: **P'=S·P** $P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$

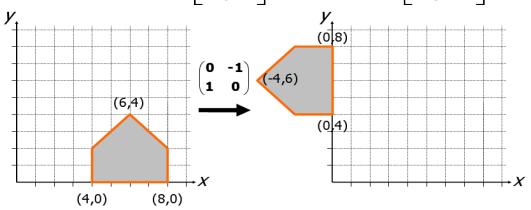


Trasformazioni

- Fissato un punto **C** (pivot) di riferimento ed un verso di rotazione (orario o antiorario), ruotare una primitiva geometrica attorno a **C** significa muovere tutti i suoi punti nel verso assegnato in maniera che si conservi, per ognuno di essi, la distanza da **C**;
- Una rotazione di θ attorno all'origine O degli assi è definita come:

$$x' = x \cdot \cos\theta - y \cdot \text{sen}\,\theta\,,$$
$$y' = x \cdot \text{sen}\,\theta + y \cdot \cos\theta$$

- In notazione matriciale **P'=R·P** $P = \begin{bmatrix} x \\ y \end{bmatrix}$ $P' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ $R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$



- Osservazioni:
  - Gli angoli sono considerati positivi quando misurati in senso antiorario;
  - Per ruotare attorno ad un punto diverso dall'origine occorre comporre la rotazione con una traslazione: **P'= R·(P-C)+C=R·P + (I-R)·C**

- Tutte le trasformazioni sono esprimibili come prodotto matriciale (lineari) tranne la traslazione

Coordinate omogenee

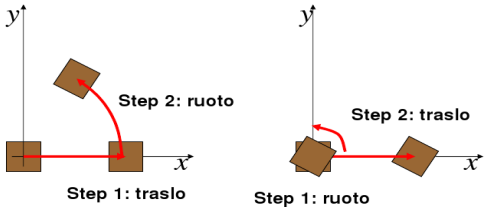
- Il punto P di coordinate (x,y) è rappresentato in coordinate omogenee come (xh,yh,w), dove:
- Due punti di coordinate (x, y, w) e (x′, y′, w′) rappresentano lo stesso punto del piano se e solo se le coordinate di uno sono multiple delle corrispondenti coordinate dell'altro;
- Almeno uno dei valori x, y, o w deve essere diverso da 0;
- Quando w = 1 (forma canonica) coordinate cartesiane ed omogenee coincidono.
- Con (x, y, w ≠ 0) si rappresentano punti, con (x, y, 0) si rappresentano punti all'infinito e quindi direzioni.
- Nella notazione in coordinate omogenee possiamo riscrivere le trasformazioni geometriche di base come operazioni matriciali
- Traslazione: $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
- Scalatura: $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
- Rotazione: $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

Concatenazione Trasformazioni

- La rappresentazione in coordinate omogenee permettono di gestire facilmente la concatenazione di trasformazioni;
- L'ordine di concatenazione è importante perché le trasformazioni geometriche sono associative ma non sono (di solito) commutative;
- La corretta sequenza delle trasformazioni T1, T2, T3 e T4 si ottiene componendo T come:

  T = T4·T3·T2·T1

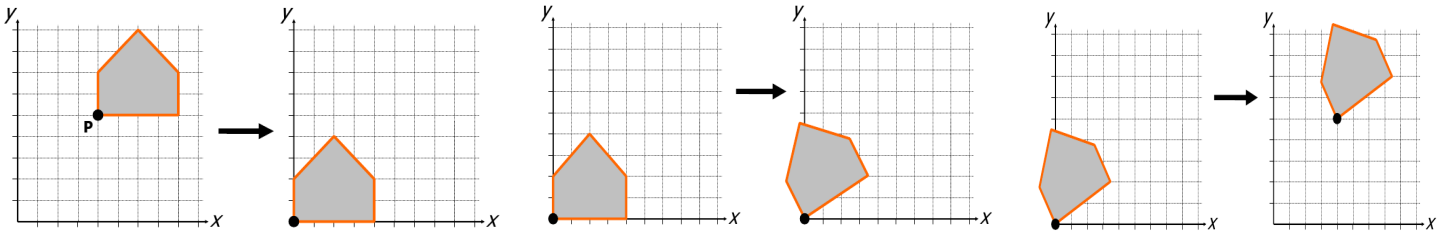
- Esempio di non commutatività: traslazione seguita da rotazione attorno all'origine (sinistra) e rotazione intorno all'origine seguita da traslazione (destra).



- XX

-
- La rotazione antioraria di un angolo θ attorno ad un punto P generico si ottiene componendo le seguenti trasformazioni:
  - Traslazione che muove P nell'origine degli assi;
  - Rotazione attorno all'origine;
  - Traslazione opposta alla precedente che riporta P nella sua posizione originale.

$$R_P = \begin{bmatrix} 1 & 0 & P_x \\ 0 & 1 & P_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -P_x \\ 0 & 1 & -P_y \\ 0 & 0 & 1 \end{bmatrix}$$

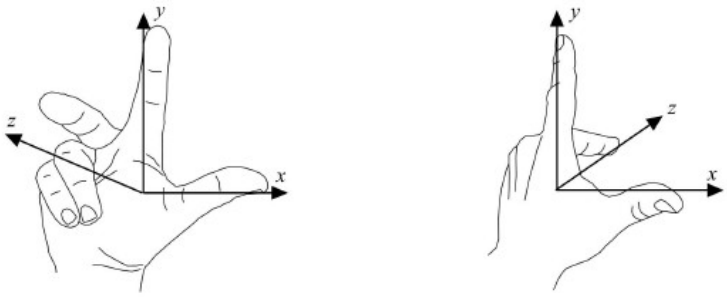


- La rotazione scalatura attorno ad un punto P generico:
  - Traslazione che muove P nell'origine degli assi;
  - Scalatura attorno all'origine;
  - Traslazione opposta alla precedente che riporta P nella sua posizione originale.

$$S_P = \begin{bmatrix} 1 & 0 & P_x \\ 0 & 1 & P_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -P_x \\ 0 & 1 & -P_y \\ 0 & 0 & 1 \end{bmatrix}$$

Passaggio al 3D

- Il passaggio dal piano allo spazio introduce una ambiguità per quanto concerne la scelta del sistema di riferimento cartesiano;
- Sistema destrorso (right-handed system) oppure sinistrorso (left-handed system);



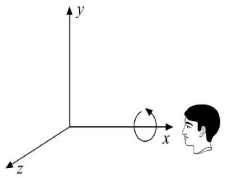
- Le trasformazioni geometriche nel piano possono essere rappresentate, in coordinate omogenee, mediante matrici 3×3;
- In modo analogo, le trasformazioni geometriche nello spazio possono essere rappresentate da matrici 4×4;
- Nello spazio, un punto in coordinate omogenee è rappresentato dalla quadrupla (x, y, z, w).

Traslazione: $\begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$ Scala: $\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

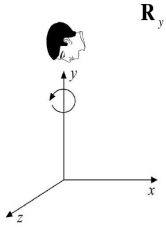
Rotazione

- La trasformazione di rotazione generica nello spazio (cioè attorno ad un asse qualsiasi) è invece complessa e non direttamente estendibile dal caso 2D (in cui l'asse di rotazione è perpendicolare al piano xy);
- Una generica rotazione nello spazio può essere ottenuta come composizione di 3 rotazioni attorno agli assi cartesiani (Euler angles)

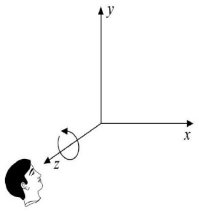
$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -sen\theta & 0 \\ 0 & sen\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \text{sen}\,\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\text{sen}\,\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\text{sen}\,\theta & 0 & 0 \\ \text{sen}\,\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



- Gli angoli di Eulero richiedono tre moltiplicazioni per ogni rotazione (inefficiente) Moltiplicazioni sucessive accumulano errori (deforamzioni)
- Teorema di Eulero: ogni trasformazione rigida 3D in cui un punto rimane fisso è equivalente ad una rotazione attorno ad un asse
  - Quindi ogni composizione di rotazioni può essere espresso come una unica rotazione attorno ad un asse (non necessariamente allineato agli assi principali)

Quaternioni

- I quaternioni sono una estensione dei numeri complessi
  - Si aggiungono le unità complesse j e k con $j^2=-1$ e $k^2=-1$ con le regole di moltiplicazione
    - Ij = -ji = k
    - jk = -kj = I
    - Ki = -ik = j
  - Un quaternione ha la forma: a +ib + jc + kd = a + **i**·**v** $\quad \mathbf{i} = \begin{bmatrix} i \\ j \\ k \end{bmatrix} \quad \mathbf{v} = \begin{bmatrix} b \\ c \\ d \end{bmatrix}$
  - Il prodotto di due quaternioni $q_1 = a_1 + ib_1 + jc_1 + kd_1$ e $q_1 = a_2 + ib_2 + jc_2 + kd_2$ è:

$$\begin{aligned} q_1q_2 = {} & (a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2) + i(a_1b_2 - b_1a_2 - c_1d_2 - d_1c_2) + j(a_1c_2 - b_1d_2 - c_1a_2 - d_1b_2) + \\ & k(a_1d_2 - b_1c_2 - c_1b_2 - d_1a_2) \\ = {} & a_1a_2 - \mathbf{v}_1\cdot\mathbf{v}_2 + a_1\mathbf{i}\cdot\mathbf{v}_2 + a_2\mathbf{i}\cdot\mathbf{v}_1 + \mathbf{i}\cdot(\mathbf{v}_1\times\mathbf{v}_2) \end{aligned}$$

  - $\mathbf{v}_1\times\mathbf{v}_2$ è il prodotto esterno tra due vettori, ed è definito come:

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad \mathbf{a}\times\mathbf{b} = \begin{bmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{bmatrix}$$

    - Da questo otteniamo l'identità $\det([a|b|c]) = (a\times b)\cdot c$

Quaternioni e rotazioni

- L'algebra dei quaternioni di lunghezza unitaria è isomorfa all'algebra SO(3) delle rotazioni in 3D
- Un quaternione unitario può essere espresso come: $\cos\theta + \sin\theta\, \mathbf{i}\cdot\mathbf{v}$ con $\|\mathbf{v}\|=1$
  - θ angolo di rotazione, **v** asse di rotazione
- Rappresentiamo un vettore P (in coordinate cartesiane) come il quaternione $0 + \mathbf{i}\cdot P$
- Una rotazione di un vettore P per un angolo attorno all'asse v può essere espresso per mezzo del quaternione $q=\cos\theta + \sin\theta\, \mathbf{i}\cdot\mathbf{v}$ come

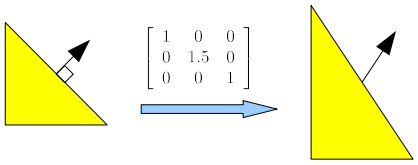
$$\mathbf{i}\cdot P' = q(\mathbf{i}\cdot P)q^{-1} = \mathbf{i}\cdot\left((\cos^2\theta - \mathbf{v}^T\mathbf{v})P + 2\cos\theta\mathbf{v}\times P + 2(\mathbf{v}\cdot P)\mathbf{v}\right)$$

  - Con $q^{-1} = \bar{q} = \cos\theta - \sin\theta\mathbf{i}\cdot\mathbf{v}$
- Da questo otteniamo la matrice di rotazione (in coordinate omogenee)

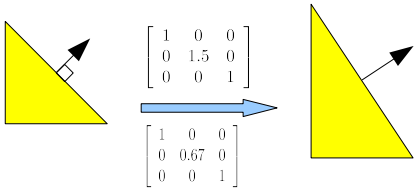
$$R_q = \begin{bmatrix} 1-2c^2-2d^2 & 2bc-2ad & 2bd+2ac & 0 \\ 2bc+2ad & 1-2b^2-2d^2 & 2cd-2ab & 0 \\ 2bd-2ac & 2cd+2ab & 1-2b^2-2c^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Trasformazioni e Normali

- Il vettore normale rappresenta una informazione compatta dell'orientamento di una superficie
  - Fondamentale per calcolare l'illuminazione
- È importante capire come una trasformazione modifichi una normale
  - Un vettore trasformato può finire per puntare in una direzione sbagliata!



  - Sia **N** la normale e **T** il vettore tangente, abbiamo **N**·**T**=0
  - Sia **M** una trasformazione, **T** viene trasformato in **T'**=**MT**.
  - Cerchiamo una trasformazione G per il vettore normale per cui valga (**GN**)·(**MT**)=0 ma (**GN**)·(**MT**)=$\mathbf{N}^T\mathbf{G}^T\mathbf{MT}$ per cui $\mathbf{G}^T\mathbf{M}=\mathbf{I}$
    - $\mathbf{G}=(\mathbf{M}^{-1})^T$

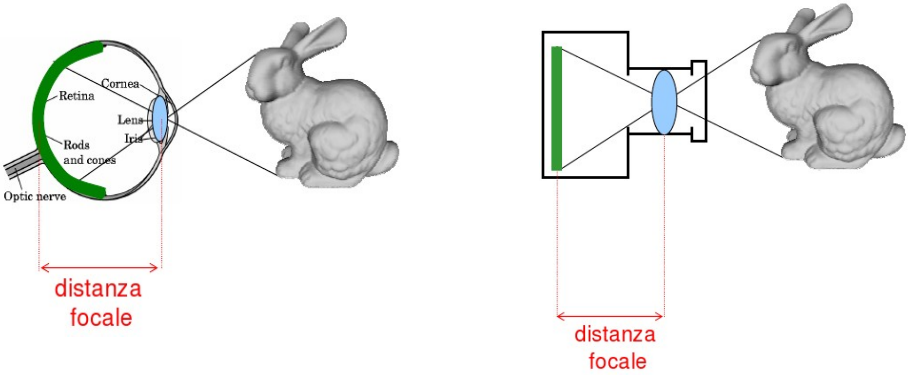


  - Trasformazione *controvariante*

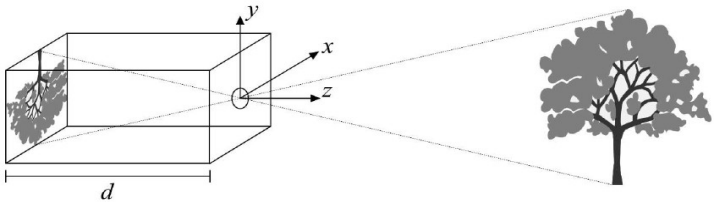
Trasformazioni di vista

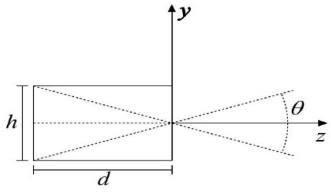
- Il processo di visione in tre dimensioni;
- Le trasformazioni di proiezione;
- I parametri della vista 3D;
- I sistemi di coordinate

- Il processo di rendering (visualizzazione) nello spazio 2D si riduce alla definizione di una window nello spazio dell'applicazione grafica, una viewport nello spazio delle coordinate del dispositivo di output ed alla applicazione di una trasformazione "window-to-viewport" dopo aver effettuato il clipping (rimozione) delle primitive (o parte di esse) esterne alla window.
- Se il dispositivo di output è 2D, il processo di rendering 3D è assimilabile al processo di formazione di un'immagine da parte di un sistema ottico, quale ad esempio una macchina fotografica.
- La visualizzazione consiste nel creare una particolare vista della scena 3D (relazione scena/osservatore).

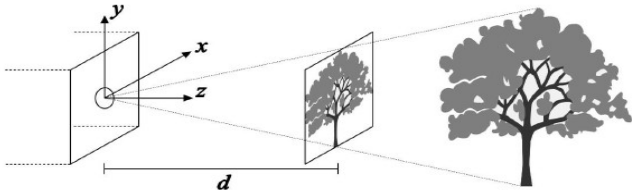


Macchina fotografica virtuale

- La metafora utilizzata per descrivere le relazioni scena/osservatore è quella della macchina fotografica virtuale (synthetic camera).
- 

- **Pinhole Camera**: La macchina fotografica virtuale è modellata considerando un parallelepipedo in cui la faccia anteriore presenta un foro di dimensioni infinitesime (pinhole camera) e sulla faccia posteriore si formano le immagini;
  - Immagini nitide, nessun problema di luminosità, l'angolo θ di vista può essere modificato variando il rapporto tra la distanza focale (d) e la dimensione del piano immagine.
- 

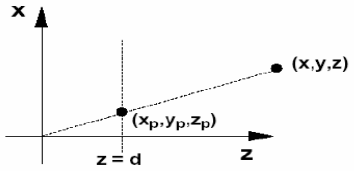
- Per evitare l'effetto di ribaltamento si assume l'esistenza di un piano immagine tra la scena ed il centro di proiezione



Proiezione prospettica

- Proietta tutti I punti lungo l'asse z sul piano z=d (punto di vista sul'origine)

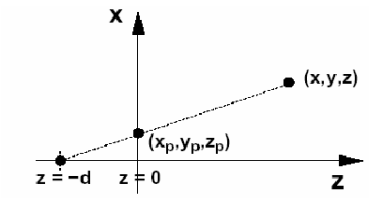
$$x_p = \frac{d \cdot x}{z} = \frac{x}{z/d}$$
$$y_p = \frac{d \cdot y}{z} = \frac{y}{z/d}$$
$$z_p = d$$



homogenize

$$\begin{pmatrix} x * d / z \\ y * d / z \\ d \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z / d \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- Alternativa: Proietta tutti I punti lungo l'asse z sul piano z=0 (punto di vista $[0,0,-d]^T$)
- $$x_p = \frac{d \cdot x}{z + d} = \frac{x}{(z/d) + 1}$$
$$y_p = \frac{d \cdot y}{z + d} = \frac{y}{(z/d) + 1}$$



homogenize

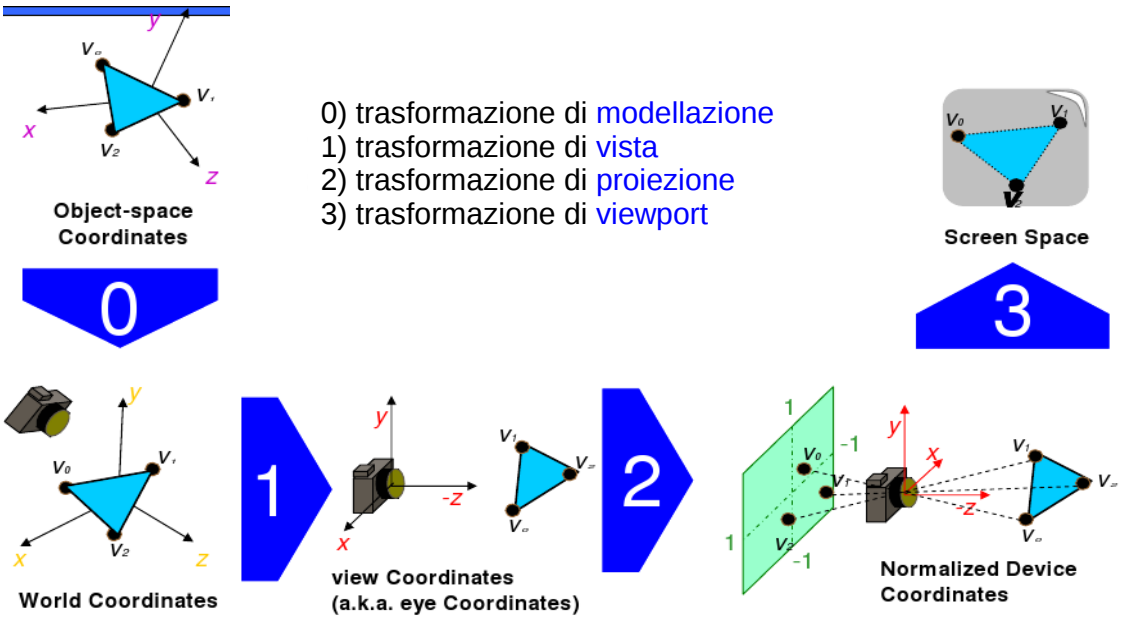
$$\begin{pmatrix} x * d / (z + d) \\ y * d / (z + d) \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ (z + d) / d \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Riepilogo pipeline di trasformazione

- Prima mettere in view coordinates
  - È sufficiente ruotare e traslare tutti i vertici del modello (in world coordinates) prima di fare la proiezione (con la pinhole):

$$P_{eye} = M_{vw} P_{world}$$

  - M_{vw} è una matrice 4x4 che modella la roto-traslazione opportuna



0) trasformazione di modellazione
1) trasformazione di vista
2) trasformazione di proiezione
3) trasformazione di viewport

Non solo Pinhole...

- Non soltanto la proiezione prospettica è utilizzata ma anche altri tipi di proiezione, ad esempio la proiezione ortogonale. Per ottenerla basta rendere la z=0 (o la lunghezza focale infinita)

$$P_{prosp} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{d} & 1 \end{bmatrix} \qquad P_{ort} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Distanza focale

***d* piccolo** → ***d* grande** — ***d* infinito** (diventa una proiezione ortogonale)

Più distorsione prospettica.

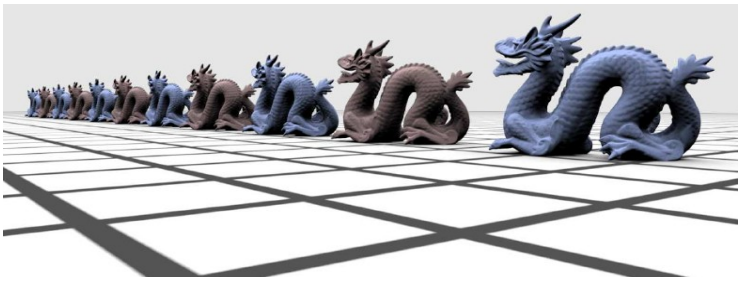
Effetto "fish-eye" (grandangolo)

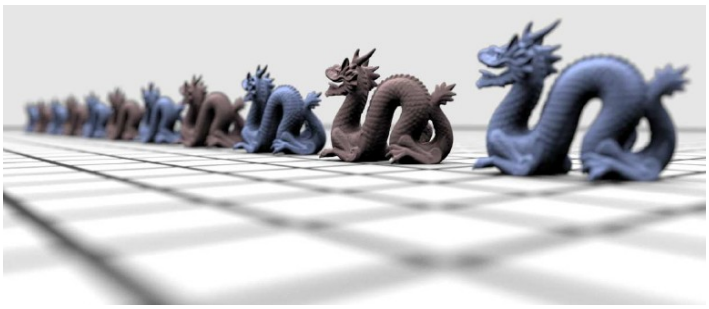
Proporzioni più mantenute

Effetto "zoom" (eg. vista dal satellite)

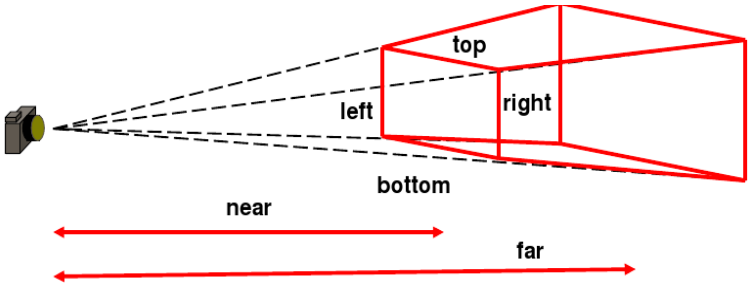
Limiti Pinhole

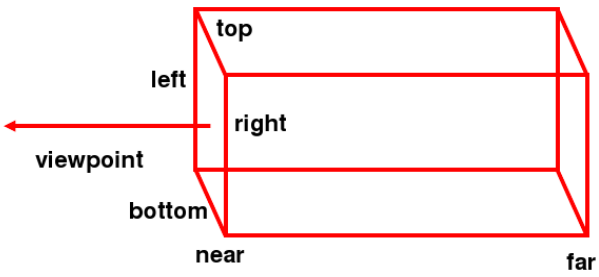
- La pinhole è semplice ma poco realistica.
- Ad esempio la modellazione di eventuali lenti sarebbe utile per “simulare” una camera più realistica.
- Con la pinhole abbiamo:
  - range di fuoco infinito (apertura infinitesima)
  - no flares (la luce non rimbalza nelle lenti…)
  - no distorsioni radiali (sempre dovute alle lenti)
-





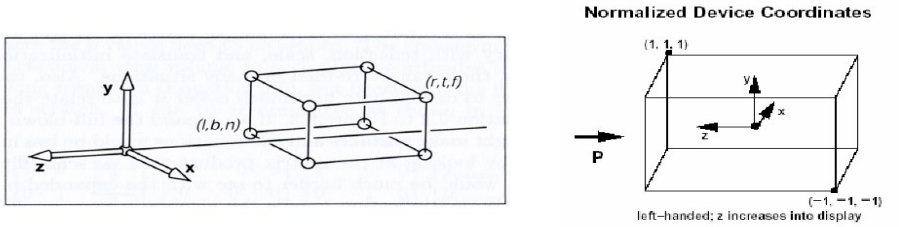
- Normalized Device Coordinates
- Abbiamo visto la proiezione ortogonale e prospettica
- Queste devono essere tali da mappare il volume di vista (viewfrustum) relativo nelle cosiddette Normalized Device Coordinates ([-1 , 1] x [-1 , 1] x [-1 , 1])





- In queste coordinate è facile verificare quel che è dentro o fuori dal view frustum

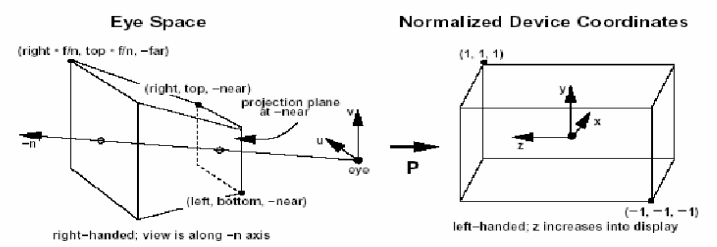
Proiezione Ortografica Canonica



Orthographic

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2}{\text{right}-\text{left}} & 0 & 0 & \frac{-(\text{right}+\text{left})}{\text{right}-\text{left}} \\ 0 & \frac{2}{\text{bottom}-\text{top}} & 0 & \frac{-(\text{bottom}+\text{top})}{\text{bottom}-\text{top}} \\ 0 & 0 & \frac{2}{\text{far}-\text{near}} & \frac{-(\text{far}+\text{near})}{\text{far}-\text{near}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Proiezione Prospettica Canonica



Perspective

$$\begin{bmatrix} x'w \\ y'w \\ z'w \\ w \end{bmatrix} = \begin{bmatrix} \frac{2 \cdot \text{near}}{\text{right}-\text{left}} & 0 & \frac{-(\text{right}+\text{left})}{\text{right}-\text{left}} & 0 \\ 0 & \frac{2 \cdot \text{near}}{\text{bottom}-\text{top}} & \frac{-(\text{bottom}+\text{top})}{\text{bottom}-\text{top}} & 0 \\ 0 & 0 & \frac{\text{far}+\text{near}}{\text{far}-\text{near}} & \frac{-2 \cdot \text{near} \cdot \text{far}}{\text{far}-\text{near}} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Screen Mapping

- La coordinata z non è interessata da questa trasformazione (è comunque passata allo stage di rasterizzazione per il test di visibilità)
- Le coordinate normalizzate (x,y) vengono mappate in coordinate schermo (xscreen , yscreen) essenzialmente attraverso un'operazione di traslazione e una di scalatura.
- Talvolta le screen coordinates vengono chiamate anche window coordinates (xwindow , ywindow).

$$\begin{pmatrix} x_{window} \\ y_{window} \end{pmatrix} = \begin{pmatrix} (w/2)x_{ndc} + o_x \\ (h/2)y_{ndc} + o_y \end{pmatrix}$$

  - w, h: larghezza ed altezza della viewport in pixel o_x , o_y : centro della viewport in pixel

Proiezioni nella pipeline