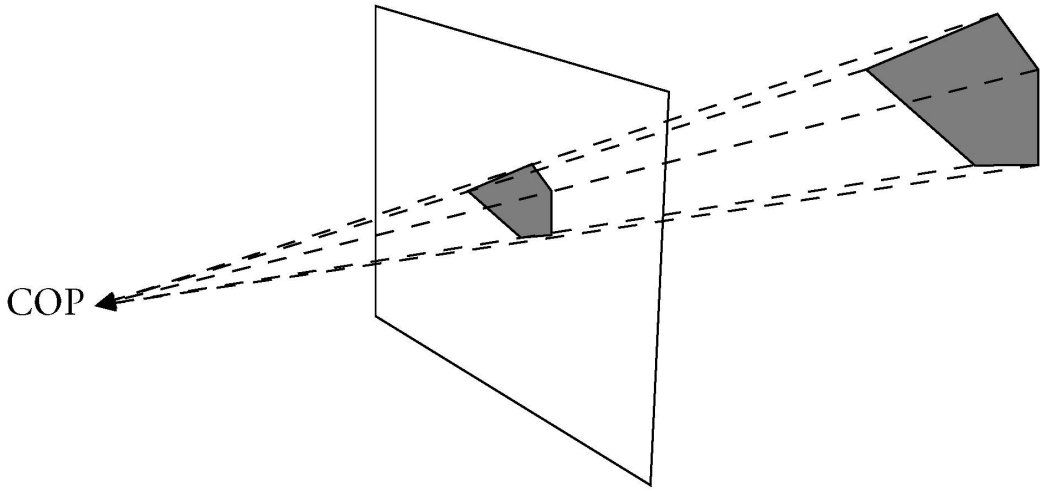
Pipeline Rendering

Dove si introduce un metodo di rendering adatto alle applicazioni in real-time.

- **Introduzione**
- **Trasformazioni geometriche**
- **Clipping**
- **Rimozione delle superfici nascoste**
- **Scan conversion**
- **Shading**
- **La rendering pipeline di OpenGL**

Introduzione

- Abbiamo visto come fare il rendering secondo il paradigma del ray casting (ray tracing).
- Vogliamo poter fare rendering in tempo reale (diciamo 30 fps).
- Anche il solo ray casting con un modello di illuminazione locale è troppo oneroso in termini computazionali, a causa dei calcoli delle intersezioni raggi-oggetti (per non parlare del ray tracing).
- Bisogna abbandonare completamente l'idea del ray casting e rivolgersi ad un diverso paradigma: il **pipeline rendering** di maglie poligonali.
- Ci concentriamo sulle **maglie poligonali**, perché:
  - sono la rappresentazione più diffusa
  - le altre rappresentazioni si possono ricondurre a questa.
- **Idea:**
  - Invece che gettare (*cast*) raggi sulla scena, proietto la scena sul piano immagine immagine.
  - La scena è composta di poligoni.
  - La proiezione di un poligono è ancora un poligono che ha per vertici le proiezioni dei vertici.



©Slater et al.

- Per proiettare un poligono basta proiettare i suoi vertici. Dunque invece di considerare tanti raggi quanti sono i pixel, si considerano tanti raggi quanti sono i vertici dei poligoni e si intersecano con il piano immagine (idealmente).
- Vantaggio: meno raggi da considerare
- Problemi:
  - **Visibilità**: nel ray casting il problema di stabilire quali parti della scena sono visibili e quali nascoste viene implicitamente risolto: è la prima intersezione lungo il raggio che conta. Nel nuovo approccio, quando proietto i poligoni, devo stabilire quali sono visibili (e dunque da disegnare) e quali no.
  - **Clipping**: nel ray casting gli oggetti al di fuori del volume di vista vengono implicitamente scartati (non vengono considerati e non influenzano il calcolo). Nel

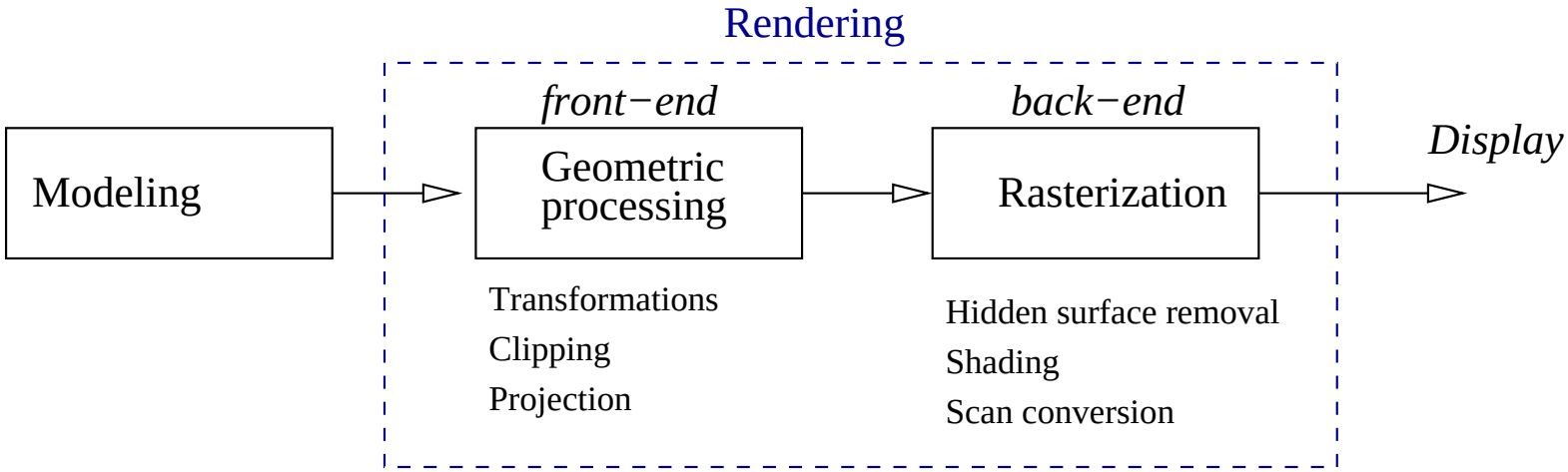
nuovo approccio si deve stabilire esplicitamente quali poligoni entrano nel volume di vista e quali no. Quelli a cavallo vanno tagliati.

- **Scan-conversion**: la proiezione dei poligoni sul piano immagine non tiene conto della natura discreta dell'immagine. Alla fine devo disegnare un poligono su una matrice di pixel, decidendo dunque quali pixel vi appartengono e quali no.
- **Shading interpolativo**: Nel ray casting il colore di ogni pixel deriva dall'applicazione del modello di illuminazione al corrispondente punto della scena. Nel nuovo paradigma gli unici punti collegati esplicitamente a punti della scena sono i vertici dei poligoni. Per questi ultimi si può assegnare un colore, e per quelli interni bisogna interpolare.
- **Ombre**: Nel ray casting era facile ottenere le ombre tracciando gli shadow rays. Ora invece le dovremo aggiungere con dei "trucchi" (oppure se ne fa a meno).

- Sembra che abbiamo fatto un passo in avanti e molti all'indietro. In realtà, come vedremo nelle lezioni seguenti, i nuovi problemi hanno soluzioni efficienti, e nel complesso la rendering pipeline risulta più veloce del ray casting. Le odierne applicazioni grafiche in tempo reale funzionano in questo modo.
- Nota: Mentre il ray casting è trasversale rispetto alla modellazione della scena, il pipeline rendering si applica solo a maglie poligonali.

Rendering di maglie poligonali

- L'ingresso è una lista di poligoni, e l'uscita è una immagine, ovvero una matrice di pixels a ciascuno dei quali è associato un colore.



- La primitiva geometrica che viene processata sono i poligoni
- I primi stadi sono **vertex bound** mentre gli ultimi sono **pixel bound**.
- Ci occuperemo ora della prima fase della rendering pipeline per le maglie poligonali, che viene chiamata *front-end* o *geometric processing*.
- Il **geometric processing** consiste delle operazioni di trasformazione (affine), proiezione (prospettica o ortografica) e clipping.

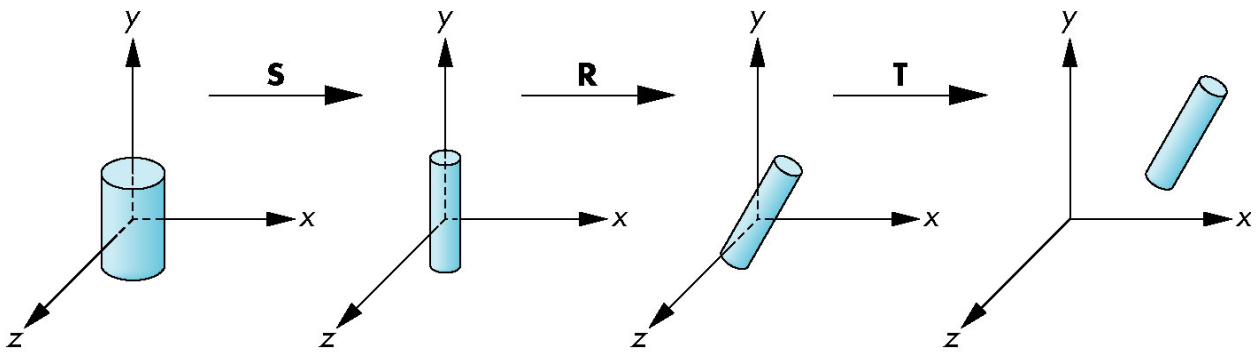
Trasformazioni geometriche

- Le trasformazioni affini e le proiezioni sono alla base dell'elaborazione geometrica (geometric processing) che viene compiuta nella rendering pipeline.
- Le proiezioni modellano la formazione della immagine 2D a partire dalla descrizione del mondo 3D.
- Le trasformazioni servono per cambiare la posizione, l'orientazione e la forma degli oggetti. Sono fondamentali per semplificare il processo di modellazione geometrica.
- Per esempio, consentono di posizionare nello spazio oggetti presi da una libreria (es. teiera), o copie di un oggetto definito una volta sola (es. auto sul traghetto).
- Un ulteriore uso delle trasformazioni è nella animazione, in particolare di oggetti articolati.

Trasformazioni affini

- Le trasformazioni affini sono usate nella rappresentazione del mondo 3D e nella sua manipolazione.
- Per quanto visto in precedenza, applicando la composizione di trasformazioni, una generica trasformazione di similitudine (rigida + scala), composta da una scalatura (s_x, s_y, s_z), una rotazione $R(\theta, \mathbf{u})$ e da una traslazione $\mathbf{t} = (t_x, t_y, t_z)$ è data dalla seguente matrice (notare l'ordine di applicazione), dove $c = \cos\theta$ e $s = \sin\theta$.

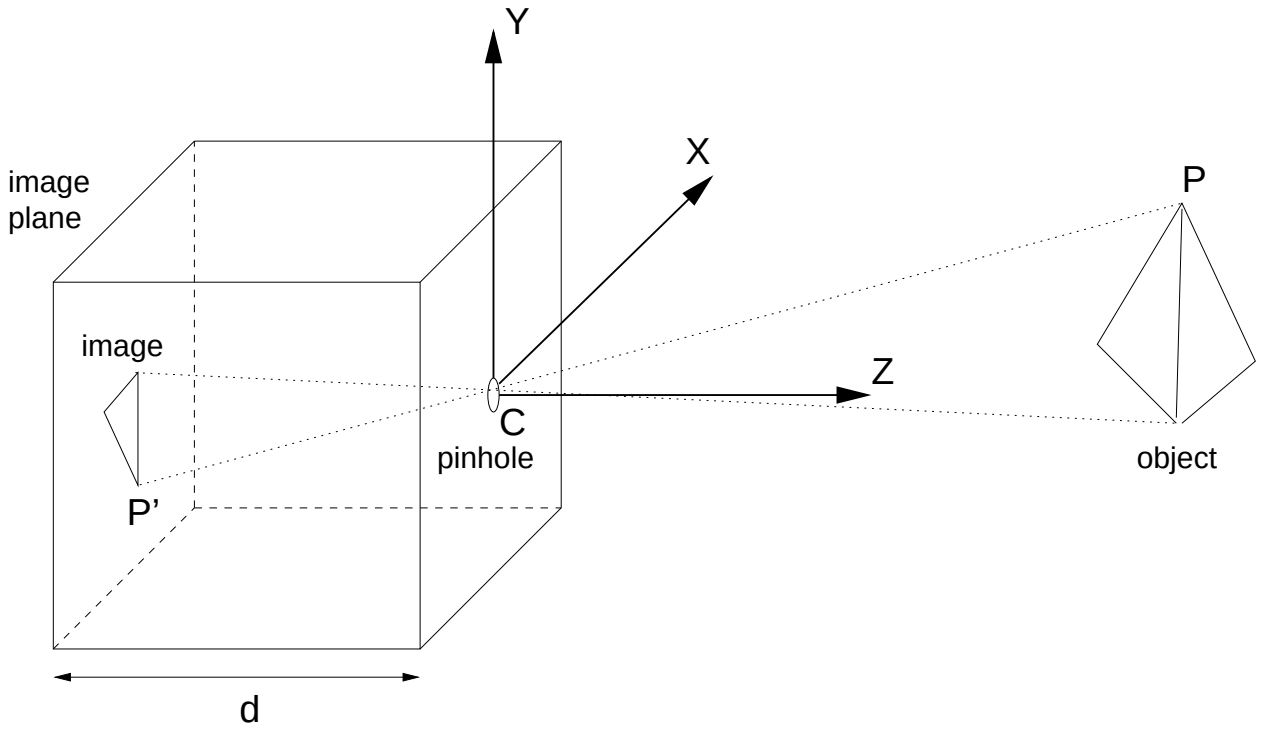
$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} (1-c)u_x^2 + c & (1-c)u_x u_y - s u_z & (1-c)u_x u_z + s u_y & 0 \\ (1-c)u_x u_y + s u_z & (1-c)u_y^2 + c & (1-c)u_y u_z - s u_x & 0 \\ (1-c)u_x u_z - s u_y & (1-c)u_y u_z + s u_x & (1-c)u_z^2 + c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



©E. Angel

Proiezioni

- Proiezione del mondo 3D sulla immagine (piano 2D).
- Il modello geometrico più semplice della formazione dell'immagine è la *pinhole camera* (letteralmente: macchina fotografica a foro di spillo.)



- Sia P un punto della scena, di coordinate (x, y, z) e sia P' la sua proiezione sul piano vista (o immagine), di coordinate (x', y', z'). Se d è la distanza del foro (o centro di proiezione) C dal piano immagine (distanza focale), allora dalla similarità dei triangoli si ottiene:

$$\frac{-x'}{d} = \frac{x}{z} \quad \text{e} \quad \frac{-y'}{d} = \frac{y}{z} \tag{1}$$

  e quindi

$$x' = \frac{-dx}{z} \qquad y' = \frac{-dy}{z} \qquad z' = -d \tag{2}$$

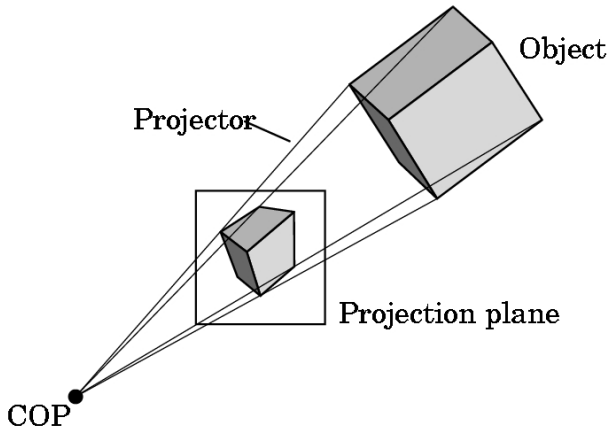
- Si noti che l'immagine è invertita rispetto alla scena, sia destra-sinistra che sopra-sotto, come indicato dal segno meno. Queste equazioni definiscono il processo di formazione dell'immagine che prende il nome di **proiezione prospettica**.

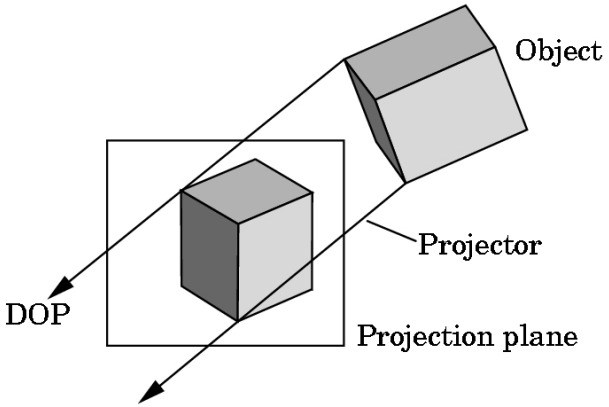
- In forma matriciale si può effettuare la proiezione prospettica, applicando ai punti P rappresentati in coordinate omogenee, $\tilde{P} = (x, y, z, 1)$, la matrice di proiezione prospettica 3×4:

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/d & 0 \end{pmatrix}$$

  dove d è la distanza tra il COP ed il piano vista (distanza focale).

- La proiezione P' del punto P sul piano vista si trova in due passi:
  1. Si applica a P la matrice M ottenendo una 3-pla $\tilde{P} = (x, y, -z/d)$; da notare che **non** rappresenta un punto (la terza componente non è pari a 1)
     item Si "normalizza" $\tilde{P}'$ dividendo le sue componenti per la terza ed ottenendo $\tilde{P}' = (-\frac{x}{(z/d)}, -\frac{y}{(z/d)}, 1)$ e dunque $P' = (-\frac{x}{(z/d)}, -\frac{y}{(z/d)})$
- Si ottiene cosìuna tripla $\tilde{P}' = (x, y, -z/d)$, che rappresenta in coordinate omogenee il punto di coordinate cartesiane $P' = (-\frac{x}{(z/d)}, -\frac{y}{(z/d)})$. P' è la proiezione di P sul piano vista.
- La **divisione prospettica** (o normalizzazione proiettiva) indica il passaggio da coordinate omogenee a coordinate cartesiane e consiste nel dividere per l'ultima coordinata omogenea e rimuovere l"'1" rimanente.
- Da notare che nella proiezione si perde l'informazione di **profondità** di un punto, ovvero punti con z iniziale di partenza diversa vengono proiettati nello stesso punto sul piano proiettivo





©E. Angel

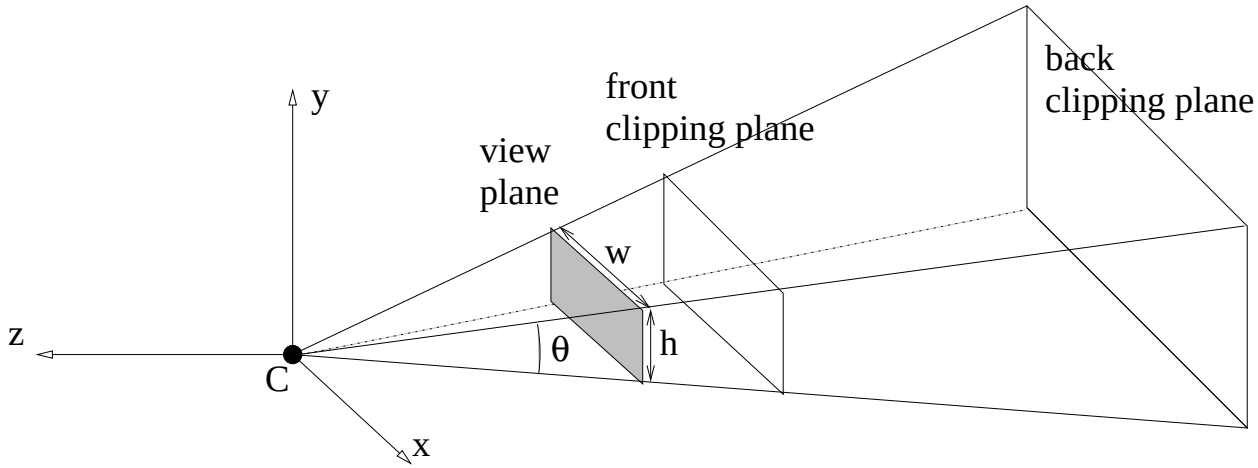
- Alle volte può risultare utile usare un tipo diverso di proiezione denominata **proiezione ortogonale** (od ortografica).
- Tale proiezione è definita come l'interesezione del piano proiettivo con la retta perpendicolare a tale piano e passante per il punto P che si vuol proiettare
- È un caso particolare di **proiezione parallela**
- La proiezione ortogonale di P si ottiene applicando la seguente matrice:

$$M_{\perp} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- In sostanza l'effetto della matrice è quello di rimuovere la componente z.

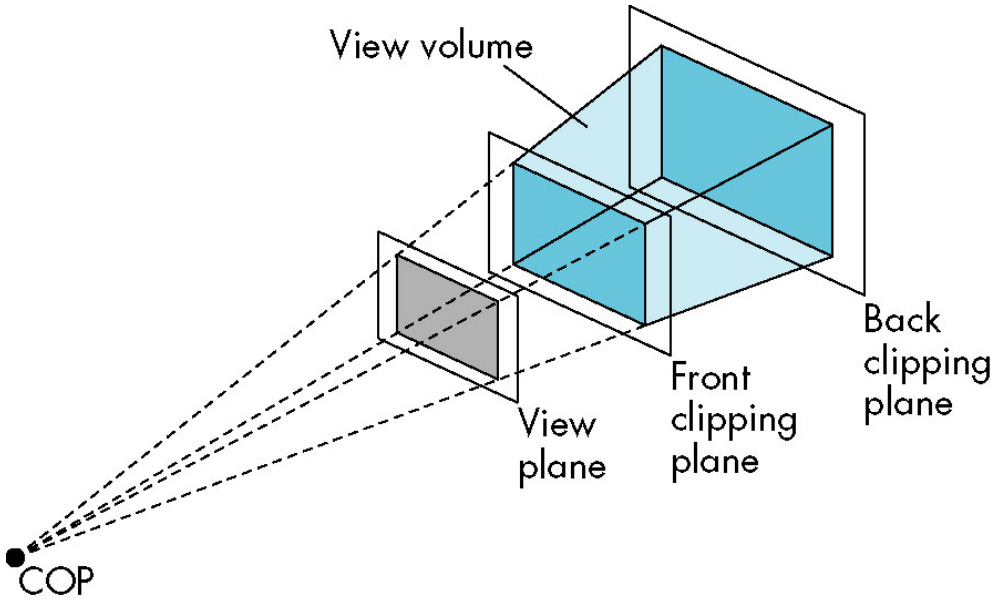
Trasformazione di vista

- Consideriamo ora un sistema di riferimento centrato nel COP, con l'asse z verso dietro, la x a destra dell'osservatore e y verso l'alto (spazio vista o *view space*).
- Il piano immagine è davanti alla telecamera a distanza 1 (tanto d è solo un fattore di scala globale).
- L'angolo di vista (solido) si assegna mediante l'angolo di vista (verticale) θ ed il fattore di aspetto $a = w/h$ della finestra di vista.



Volume di vista

- La piramide di vista, in principio semi-infinita, viene limitata da due piani paralleli al piano di vista: il **piano di taglio anteriore** (front clippig plane) ed il **piano di taglio posteriore** (back clipping plane).
- Il solido risultante è un frustum, e prende il nome specifico di **frustum di vista** (*view frustum*), o volume di vista (*view volume*).
- Il frustum di vista è la regione di spazio nel mondo che può apparire sulla finestra di vista [a]



©E. Angel

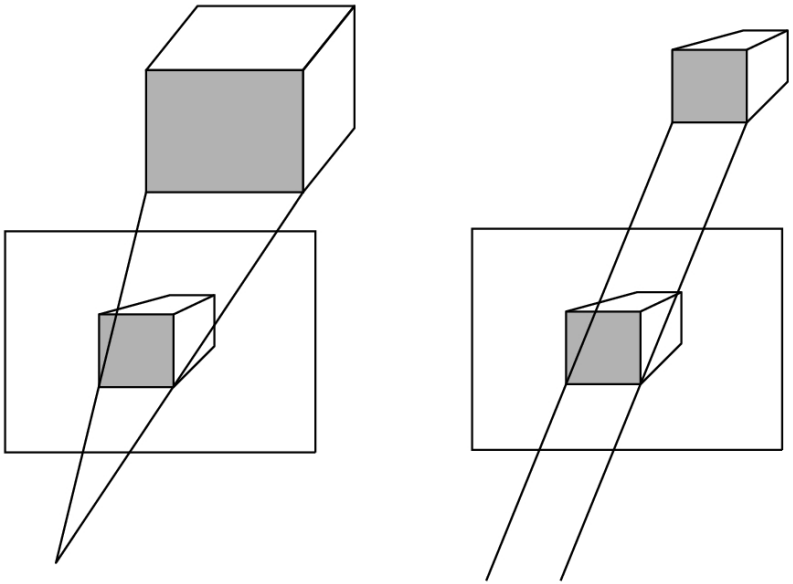
[a] http://en.wikipedia.org/wiki/Viewing_frustum

- Gli oggetti che cadono dentro il volume di vista vengono disegnati.
- Per proiettarli con M o ($M_\perp$) le coordinate devono essere trasformate nello spazio vista, con una trasformazione rigida, detta **trasformazione di vista**.
- Si noti che abbiamo messo il piano vista davanti al centro di proiezione, ma l'asse Z punta "indietro", quindi la matrice di proiezione rimane la stessa vista prima (con $d = 1$):

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Trasformazione prospettica

- In linea di principio si potrebbe effettuare la proiezione prospettica, applicando la matrice M ai punti P espressi nello spazio vista.
- Invece la proiezione viene effettuata in modo più contorto (apparentemente) di quello appena visto, applicando la **trasformazione prospettica**, che trasforma il frustum in un parallelepipedo retto (volume di vista canonico), seguita dalla proiezione ortografica.
- Il motivo principale per l'introduzione del volume di vista canonico è semplificare il **clipping** (vedi più avanti).



proiezione prospettica = trasformazione prospettica + proiezione ortografica

Pseudo-profondità

- Consideriamo la matrice 4×4 non singolare (simile alla M) (che prende il nome di matrice di **matrice di trasformazione prospettica**[a]):

$$N = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

- Applicando N a P si avrà quindi la 4-pla $(x, y, \alpha z + \beta, -z)$ che, dopo la divisione prospettica fornisce $P' = (-\frac{x}{z}, -\frac{y}{z}, -\alpha - \beta/z)$
- Le prime due componenti sono identiche alla proiezione standard, ma la terza componente (pseudo-profondità)

$$z_s = -(\alpha + \beta/z).$$

  per valori di α e β fissati è una funzione monotona di z. La relazione tra z e z_s è non lineare, ma l'ordinamento sulla profondità è conservato.

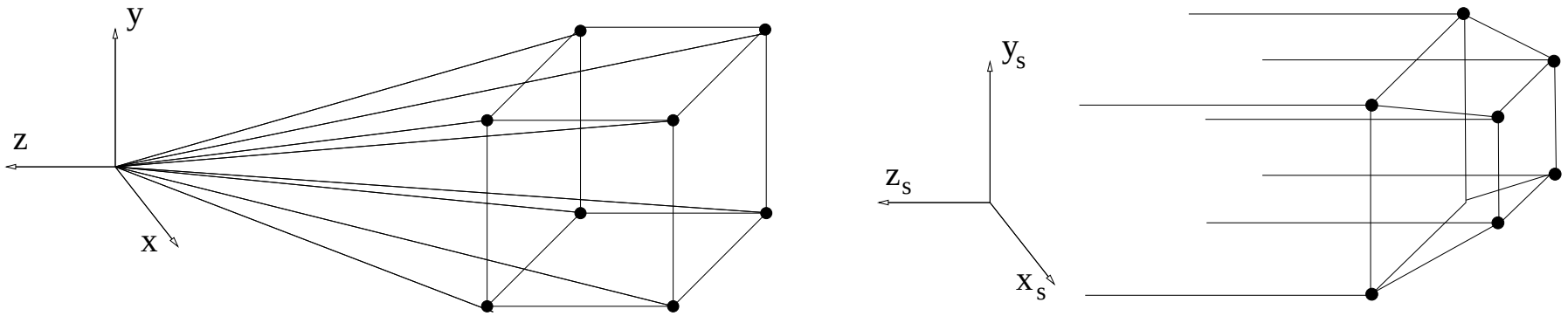
[a] Angel la chiama "matrice di normalizzazione prospettica"

- Quindi la trasformazione prospettica realizza un mapping che sortisce le stesse coordinate x ed y della proiezione prospettica, ma mantiene una terza coordinata che è funzione della z.
- Applicando la trasformazione N ai punti, seguita da una proiezione ortogonale, otteniamo lo stesso risultato che avremmo ottenuto con la proiezione prospettica.
- Infatti, applicando la proiezione ortografica dopo la trasformazione prospettica otteniamo:

$$M_{\perp}N = M$$

Volume di vista canonico

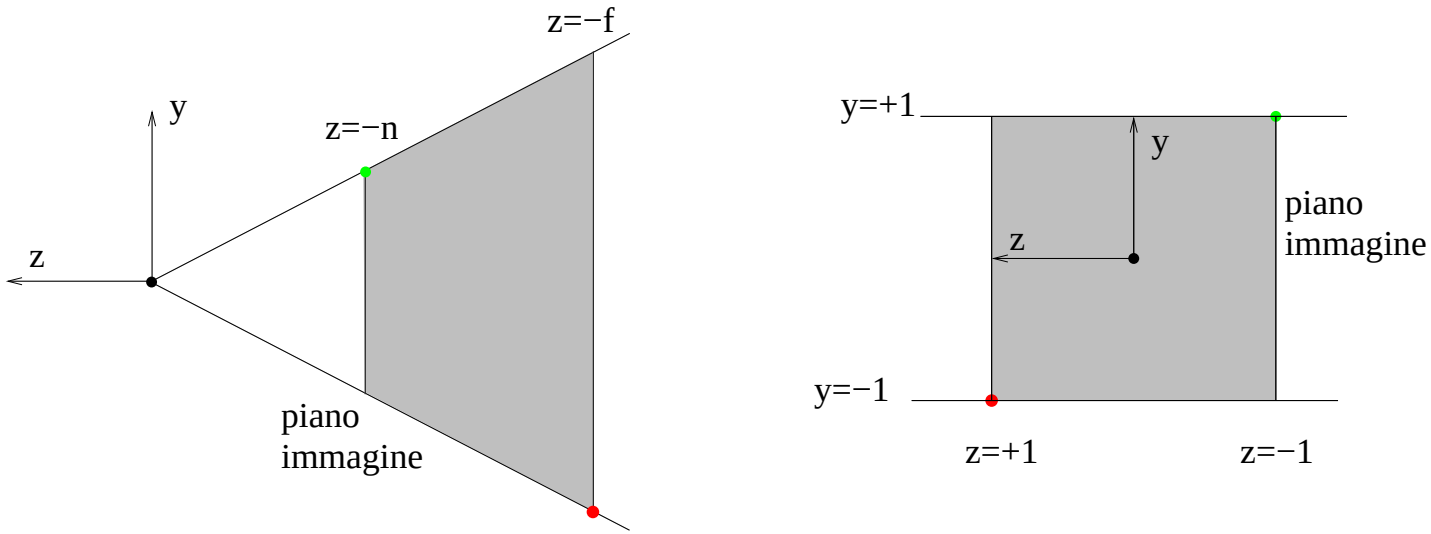
- La trasformazione (normalizzazione) prospettica N mappa punti dello spazio 3D in punti dello spazio 3D, e scegliendo opportunamente α be β possiamo fare in modo che mappi il frustum di vista in un parallelepipedo chiamato **volume di vista canonico**. Gli oggetti vengono distorti di conseguenza.
- Proiettando questo parallelepipedo ortogonalmente (ovvero si elimina la terza coordinata cartesiana, z_s nel nostro caso) si ottiene la proiezione prospettica desiderata.



- Diversi sistemi (PHIGS, OpenGL, Renderman, ...) adottano convenzioni diverse per le dimensioni del volume di vista canonico.
- In OpenGL il volume di vista canonico è un cubo di lato unitario, definito come:

$$-1 \leq x \leq +1, \quad -1 \leq y \leq +1, \quad -1 \leq z \leq +1$$

- Nel volume di vista canonico il back clipping plane ha equazione $z_s = 1$, ed il front clipping plane $z_s = -1$.
- Nel sistema di riferimento camera il front clipping plane di trova a distanza n dall'origine ed il back clipping plane a distanza f.
- Vogliamo dunque scegliere α e β in modo che l'intervallo di profondità $z \in [-n, -f]$ venga mappato in $z_s \in [-1, 1]$ (notare l'inversione di z).



- Risolvendo per α e β si ottiene:

$$\alpha = -\frac{f+n}{n-f} \quad \beta = \frac{2fn}{n-f}$$

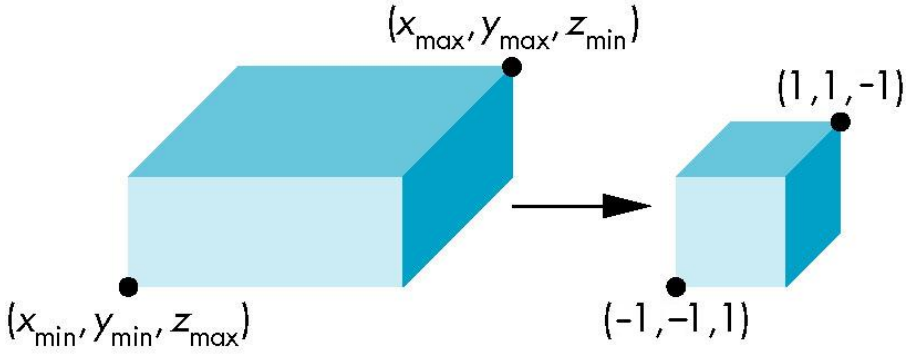
- Se l'angolo di vista $\theta = 90^o$ ed il fattore d aspetto $a = 1$, allora la matrice N con α e b calcolati come sopra trasforma il frustum nel volume di vista canonico (si può verificare trasformando i vertici del frustum di vista).
- Nel caso più generale, bisogna tenere conto dell'angolo di vista θ e del fattore di aspetto a, moltiplicando N per la seguente matrice di scalatura:

$$\begin{pmatrix} \gamma/a & 0 & 0 & 0 \\ 0 & \gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  che ha l'effetto di trasformare la piramide generica in piramide retta a base quadrata.
- La matrice N viene chiamata anche (in terminologia OpenGL) matrice di proiezione (projection matrix) anche se, a rigore, non effettua una proiezione dello spazio 3D, ma una sua trasformazione.
- Si noti che abbiamo sempre considerato la trasformazione operata da una matrice in coordinate omogenee composta da: moltiplicazione matrice-vettore seguita da divisione prospettica. Risulterà utile per il *clipping* separare le due operazioni (vedi coordinate di clipping).

Telecamera ortografica

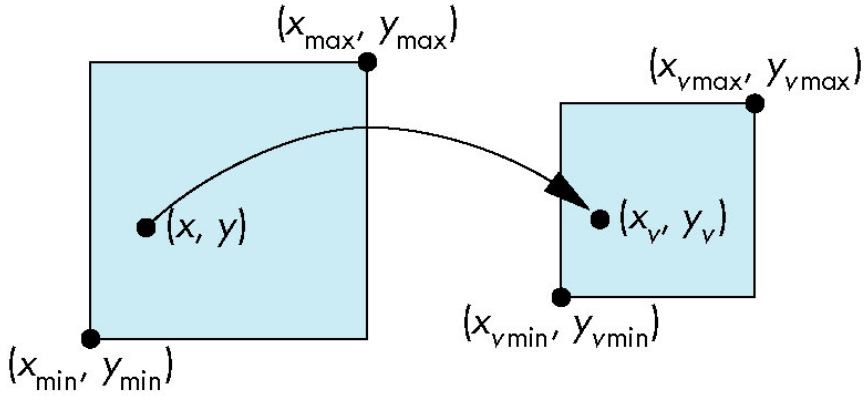
- Se invece si vuole effettuare una proiezione ortogonale (ortografica), basta sostituire la trasformazione prospettica con una trasformazione (affine) che mappa il volume di vista (un parallelepipedo in questo caso) nel volume di vista canonico.



©E. Angel

- Nel caso della proiezione ortografica, il volume di vista è già un parallelepipedo, e basta trasformarlo in quello canonico con una opportuna matrice di scalatura.

Trasformazione viewport

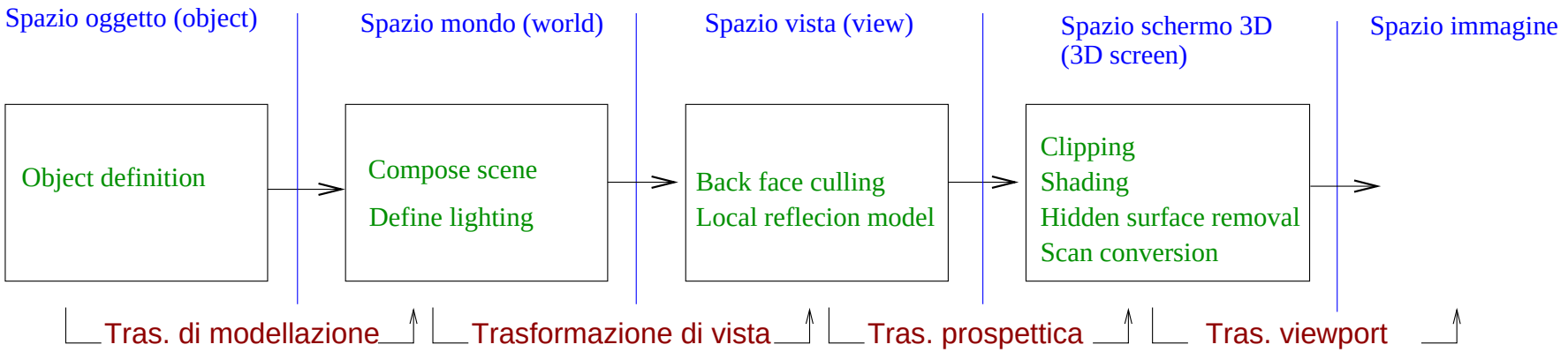


©E. Angel

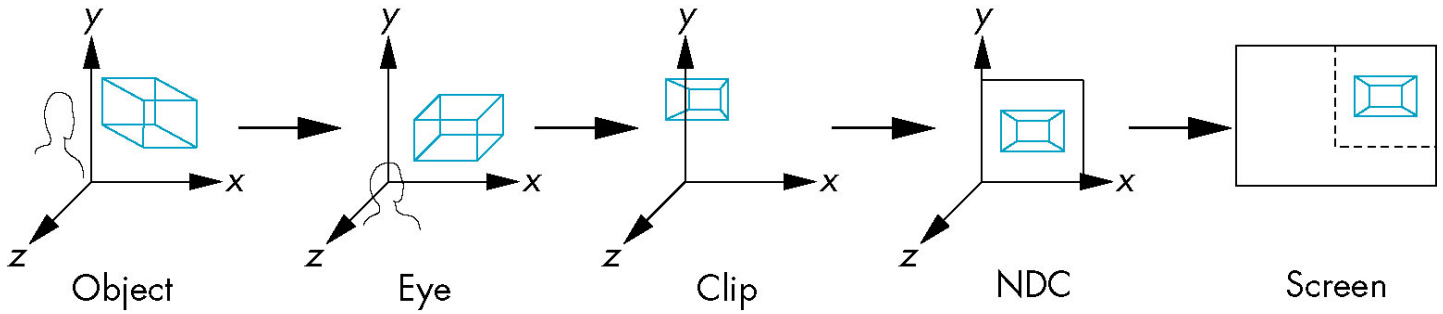
- Viewport: porzione del display all'interno della quale viene visualizzato il risultato del rendering
- La trasformazione viewport si applica dopo la proiezione ortografica.
- Dipende dalle caratteristiche fisiche del *display* (es. i pixel).
- Ai punti proiettati del 3D screen viene applicata una matrice di trasformazione affine che :
  - ripristina il fattore di aspetto corretto per l'immagine (distorto dalla trasformazione prospettica)
  - scala e trasla l'immagine per aggiustarla al viewport (es. 640 $\times$ 480)

Rassegna dei sistemi di coordinate

- Abbiamo visto come vengono gestite dal punto di vista geometrico le trasformazioni 3D e la proiezione prospettica.
- Il processo coinvolge diversi sistemi di riferimento e trasformazioni tra di essi.



- Autori diversi definiscono spazi diversi. L'esempio sotto è tratto da Angel.



©E. Angel

- **Spazio Oggetto** (*object space*): è lo spazio dove ciascun singolo oggetto viene definito. Si chiama anche spazio locale (*local space*) o spazio di modellazione (*modeling space*)
- **Spazio Mondo** (*world space*): è lo spazio dove la scena o l'oggetto completo è rappresentato. Si passa dallo spazio oggetto allo spazio mondo mediante la trasformazione di modellazione. Si chiama anche *application space*.
- **Spazio Vista** (*view space*): è un sistema di riferimento centrato sulla telecamera virtuale, che definisce, assieme alla finestra di vista ed ai piani di taglio, il frustum di vista. Si chiama anche *camera space* o *eye space*.
- **Spazio schermo 3-D** (*3D-screen space*): è lo spazio che corrisponde al volume di vista canonico, che si ottiene trasformando (con deformazione) il frustum di vista in un parallelepipedo. Molte operazioni del processo di rendering avvengono qui. Si chiama anche *3D normalized device coordinate system* (NDC) o *normalized projection coordinate system*.
- **Spazio Immagine** (*image space*) è il sistema di coordinate nel display fisico (pixel). Si ottiene proiettando ortogonalmente il volume di vista canonico ed applicando la trasformazione di viewport. Si chiama anche *(physical) device coordinate system*, o *screen coordinate system*.

("spazio" viene usato come sinonimo di "sistema di coordinate")

Clipping

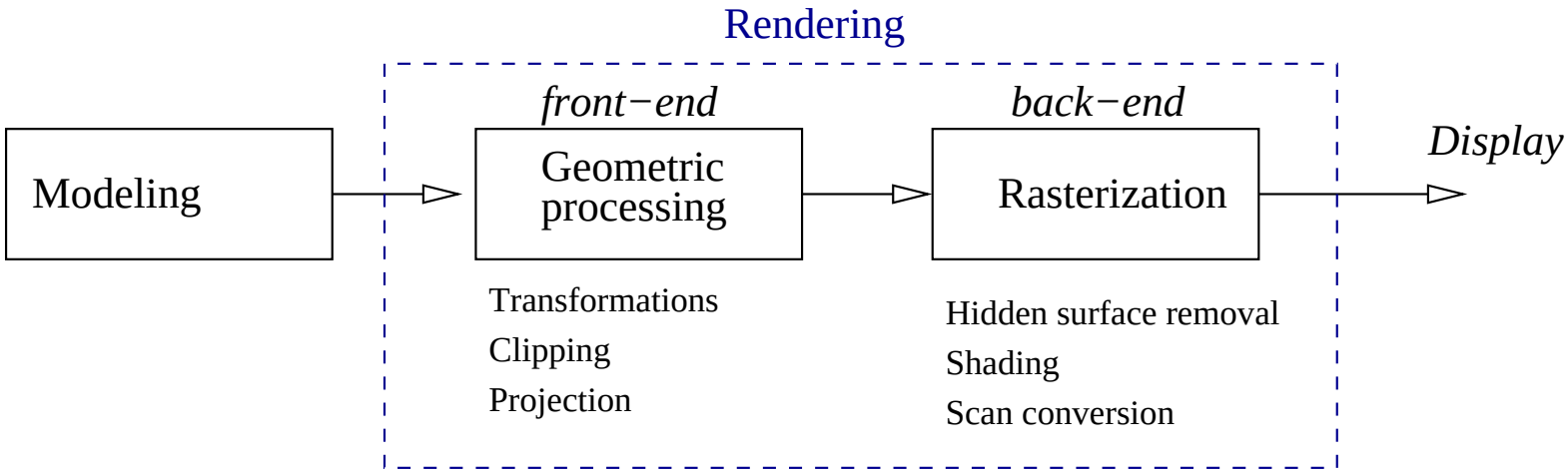
- Il **clipping** consiste nell'eliminare i poligoni esterni al volume di vista e nel ritagliare le parti dei poligoni che vi sono solo parzialmente contenuti.
- Il clipping avviene nello spazio 3D screen: infatti lavorando con il volume di vista canonico (in NDC), l'operazione risulta semplificata, poiché le facce del volume di vista sono ortogonali.
- In particolare, si opera in coordinate omogenee prima della divisione prospettica (*clip coordinates*). Questo per ragioni abbastanza tecniche, tra cui la possibilità che vi siano punti con coordinata omogenea negativa.
- Ci limitiamo a citare l'algoritmo di clipping di poligoni in 3D di Sutherland-Hodgeman (il lettore interessato potrà trovare dettagli in Angel pg. 293, SCMS pg. 113).

Un piano di taglio anteriore distinto dal piano vista consente di sezionare la scena, evidenziando l'azione del clipping.

©Alan Watt

Ricapitoliamo...

- Dopo che i poligoni hanno attraversato tutte le trasformazioni geometriche sono stati scartati i poligoni al di fuori del view frustum e sono stati tagliati (*clipping*), si può procedere a disegnarli sul display (o *viewport*).



- In questa lezione studieremo la parte finale della rendering pipeline, il back-end o **rasterization**.

Rimozione delle superfici nascoste

- Non tutti i poligoni sopravvissuti, però, devono essere disegnati. Alcuni possono non essere visibili dall'osservatore perché nascosti (totalmente o parzialmente) da altri poligoni.
- Problema: dati un insieme di poligoni in 3D ed un punto di vista, si vogliono disegnare solo i poligoni visibili (o porzioni di essi). Ogni poligono si assume essere piatto ed opaco.
- Vi sono essenzialmente due approcci:
  - **object-precision**: l'algoritmo lavora sui poligoni stabilendo relazioni di occlusione reciproca. Il costo è quadratico nel numero dei poligoni. Però la precisione è elevata (precisione macchina).
  - **image-precision**: l'algoritmo stabilisce occlusioni a livello del pixel. è più veloce ma la precisione è limitata.
- Nota: a rigore i metodi object-precision fanno parte del geometric processing, mentre solo quelli image-precision fanno parte della rasterizzazione.
- La rimozione delle superfici nascoste (*Hidden Surface Removal*, HSR) viene anche indicata come **determinazione delle superfici visibili** (*Visible Surface Determination*,VSD).

Back-face culling

- Non è un algoritmo generale di HSR, ma solo una tecnica (object space) utile per eliminare subito poligoni ovviamente invisibili.
- L'eliminazione delle facce posteriori (o **back-face culling**), elimina i poligoni che, a causa della loro orientazione, non possono essere visti.
- Viene effettuato nello spazio vista.
- Se $\mathbf{v}$ è la direzione di vista (punta verso l'osservatore) ed $\mathbf{n}$ è la normale al poligono, è facile rendersi conto che il poligono è visibile solo se:

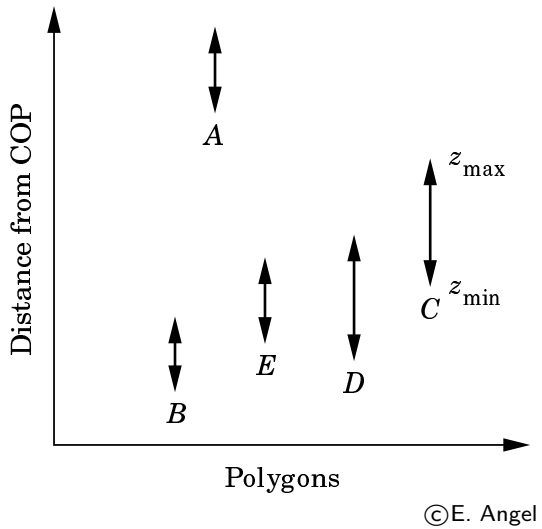
$$\mathbf{n} \cdot \mathbf{v} > 0$$

- **Nota**: se la scena è composta da un solo solido convesso, il culling risolve anche il problema della eliminazione della HSR.

Depth-sort

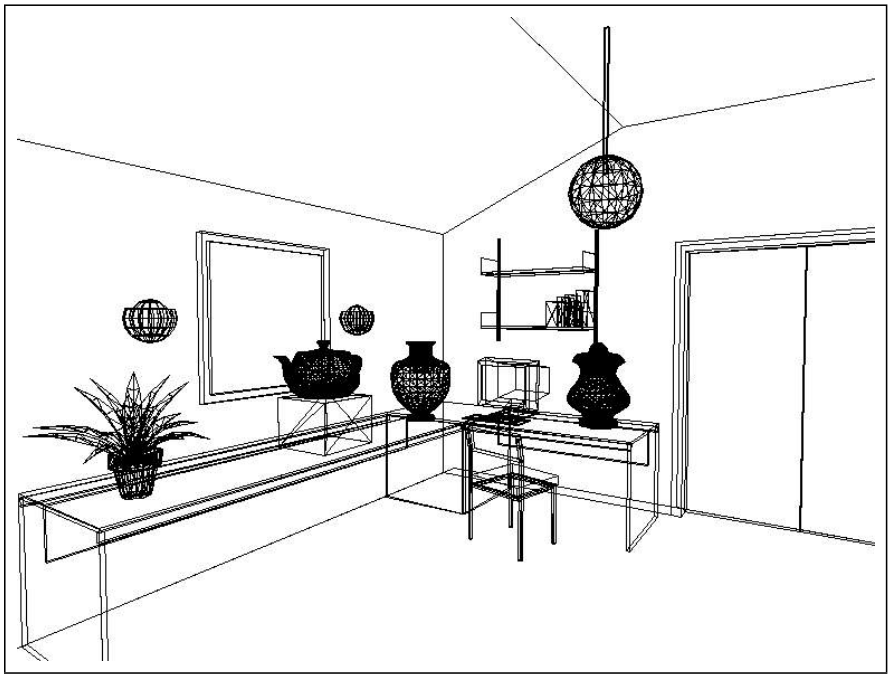
- Opera in object-precision (ovvero a livello di poligono)
- **Idea**:
  - i poligoni vengono ordinati per profondità (o pseudo-profondità) decrescente rispetto all'oservatore
  - si effettua il rendering dei poligoni della lista, dal più lontano al più vicino (*back-to-front rendering*).
- In questo modo i poligoni più lontani vengono sovrascritti da quelli più vicini.
- Non sempre è possibile ordinare i poligoni per profondità. Allora bisogna spezzarli.
- La versione semplificata del depth-sort che ignora le ambiguità si chiama ance **algoritmo del pittore**.

- **Depth-sort**: dopo avere ordinato i poligoni in base al vertice di massima distanza dall'osservatore, vengono rilevate le ambiguità e risolte spezzando i poligoni coinvolti.
- Dati due poligoni P e Q le cui z-estensioni si sovrappongono, è corretto disegnare P prima di Q solo se almeno uno di questi test è vero:
  1. le x-estensioni di P e Q sono disgiunte (veloce);
  2. le y-estensioni di P e Q sono disgiunte (veloce);
  3. Si consideri il piano contenente Q. P è contenuto completamente nel semispazio opposto a quello dove è l'osservatore (abbastanza veloce)?
  4. Si consideri il piano contenente P. Q è contenuto completamente nello stesso semispazio dove è l'osservatore (abbastanza veloce)?
  5. Le proiezioni di P e Q sullo schermo sono disgiunte (pesante)?



- Se tutti i test falliscono, controllo se è corretto disegnare Q prima di P. Se anche questo fallisce Q viene tagliato usando il piano contenente P, ed i pezzi vengono collocati nella lista al posto giusto.

Esempio di visualizzazione wireframe di una scena, con e senza rimozione della superfici nascoste.



©Alan Watt

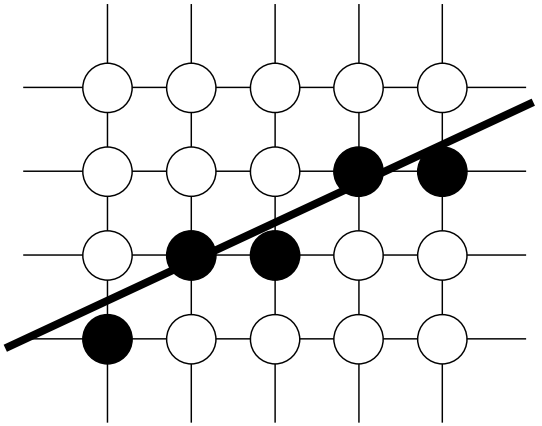
Depth-Buffer

- È un algoritmo image-precision, ma processa un poligono alla volta
- Opera nel 3D screen space
- Fa parte del processo di rasterizzazione
- Il *depth-buffer* o **z-buffer** è una matrice (grande come l'immagine) che contiene, per ciascun pixel, il più piccolo valore di profondità (z) incontrato finora.
- Durante la rasterizzazione, per ciascun poligono che viene processato:
  - si calcola la profondità (z) dei punti interni con interpolazione della z dei vertici (con interpolazione scan-line, come il colore in Gouraud shading).
  - se la z del pixel è inferiore a quella contenuta nello z-buffer, allora la sua intensità viene scritta nell'immagine e la z viene viene aggiornata.
- **Vantaggio**: semplicità di implementazione
- **Svantaggio**: occupazione di memoria: servono almeno 20-32 bits per pixel per avere una discretizzazione accettabile delle profondità.

Scan conversion

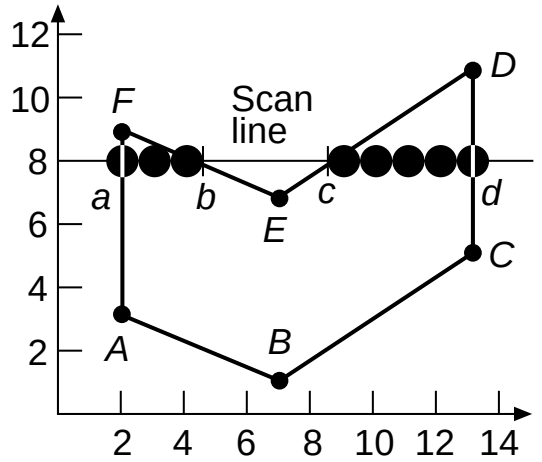
- La **scan conversion** consiste nel disegnare i poligoni sulla immagine, riempendoli con il colore determinato dallo shading (vedi dopo). Questo equivale a risolvere i seguenti due problemi:
  - Determinare i pixel da accendere per disegnare un segmento
  - Determinare i pixel interni al poligono
- Nota. Per noi la scan conversion è uno stadio della rasterizzazione, che comprende anche HSR e Shading. Si faccia però attenzione al fatto che questo uso non è comunemente accettato: molti autori usano scan conversion e rasterization come sinonimi.

- **Algoritmo di Bresenham**. Il classico algoritmo per disegnare un segmento è quello di Bresenham. Esso genera una sequenza connessa di pixel. Dopo avere disegnato un pixel l'algoritmo sceglie tra i suoi 8-vicini quale accendere in base all'equazione della retta, **usando solo aritmetica intera.**



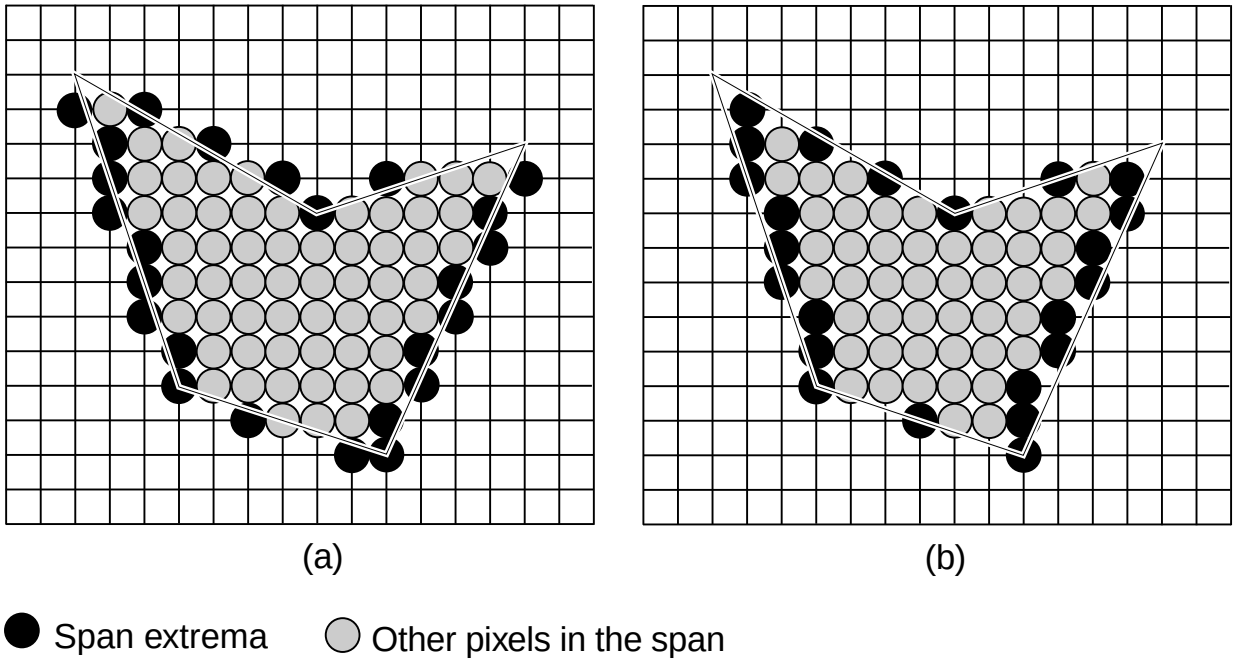
©Foley et al.

- **Algoritmo scan-line**. è l'algoritmo standard per riempire i poligoni. Il poligono viene riempito considerando una linea che lo scandisce riga dopo riga dal basso verso l'alto. Per ciascuna riga si effettua una scansione da sinistra a destra, e quando si incontra un edge del poligono si inizia a riempire, quando si incontra un'altro edge si smette. Ci sono casi speciali da gestire con accortezza.



©Foley et al.

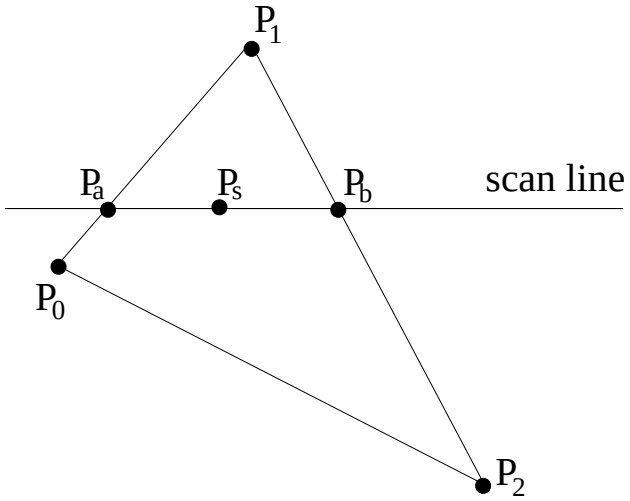
- Nota: i segmenti disegnati da Bresenham non vanno bene per la scan conversion di un poligono. Infatti (a) ce ne vuole uno solo per riga e (b) vogliamo che siano interni al poligono.



- Non ci occupiamo nel dettaglio di questi algoritmi, che sono ormai dei classici della Grafica, poiché per il programmatore la scan-conversion è un processo ormai completamente trasparente.
- Il lettore interessato potrà trovare maggiori dettagli in Buss sez. II.4, oppure Mount, Lecture 25 e 26, Angel, sez. 7.8, SCMS saez. 5.3.)

Interpolazione scan-line

- Vediamo come si si possono interpolare all'interno di un triangolo valori definiti sui vertici (colore, profondità, coordinate texture,... lo vedremo)
  - Dato un triangolo sui cui vertici P_0, P_1, P_2, sono definite tre quantità c_0,c_1,c_2 .
  - Se la scan line interseca due lati del triangolo in P_a e P_b (come in figura), calcoliamo il valore c_a per interpolazione lineare tra i valori c_0 e c_1, ed il valore c_b per interpolazione lineare tra i valori c_1 e c_2.
  - calcoliamo quindi il valore c_s per tutti i punti del segmento di scan line (P_a,P_b) per interpolazione lineare tra i valori agli estremi, c_a e c_b.



- è più efficiente calcolare il valore interpolato in modo incrementale.
- **Nota.** L'interpolazione effettuata in NDC è una approssimazione di una operazione che andrebbe effettuata in coordinate non distorte (es. in spazio vista).

Shading

- Gli algoritmi di **shading** consentono di colorare i poligoni, ovvero di assegnare un colore ai pixels sui quali il poligono viene proiettato.
- A monte serve un **modello di illuminazione** che consente di assegnare un colore ai punti 3D di una superficie.
- Ricordiamo solo che per applicare un modello di illuminazione locale servono: la normale $\mathbf{n}$ della superficie nel punto, il vettore $\mathbf{v}$ che punta verso l'osservatore (direzione di vista) ed il vettore $\mathbf{l}$ che punta verso la sorgente luminosa.
- Gli algoritmi di **shading** sono essenzialmente schemi di **interpolazione** di valori calcolati sui vertici dei poligoni, che si applicano durante il processo di scan-conversion del poligono.

- L'applicazione del modello di illuminazione ai vertici di un poligono (appartenente ad una maglia poligonale) richiede di specificare un normale nel vertice $\mathbf{n}$.
- Tipicamente:
  - Se la superficie da rappresentare è poliedrica (es. cubo) allora ciascun vertice di una faccia ha la medesima normale, ovvero *la* normale della faccia, che si ottiene come prodotto esterno di due lati consecutivi (non collineari) del poligono. Attenzione che lo stesso vertice deve poter avere associate molte normali...
  - Se invece la maglia è una approssimazione di una superficie soggiacente nota, a ciascun vertice viene associata la (unica) normale alla superficie, ottenuta dall'equazione parametrica o implicita della superficie.
  - Se la maglia approssima una superficie liscia ma incognita, la normale nel vertice può essere ottenuta prendendo la media delle normali dei poligoni adiacenti al vertice:

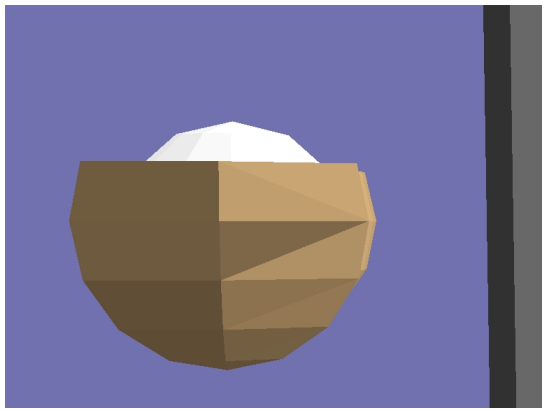
$$\mathbf{n} = \frac{\sum_i \mathbf{n}_i}{||\sum_i \mathbf{n}_i||}$$

- Struttura dati: ogni faccia ha una lista di vertici e di normali di vertice (quindi lo stesso vertice può essere associato a normali diverse)

Flat shading

- Il flat shading è il più semplice di tutti: si colora una faccia con un colore.
- Se la normale fosse definita per la faccia, si colorerebbe il poligono con il colore ottenuto usando la sua normale $\mathbf{n}$ ed il modello di illuminazione locale.
- Poiché la normale è definita sul vertice, si calcola il colore su un solo vertice del poligono (il primo che si incontra) usando il modello di illuminazione locale, e si applica quel colore a tutto il poligono.
- Nota: Per una superficie poliedrica, se il punto di vista e la sorgente luminosa sono lontani, i vettori $\mathbf{v}$ ed $\mathbf{l}$ sono costanti su una faccia, quindi è corretto, secondo il modello di illuminazione locale, assegnare un solo colore a tutta la faccia.

- **Vantaggi**: semplicità e velocità
- **Svantaggi**: Si percepiscono distintamente i poligoni. Questo effetto è indesiderato quando la maglia poligonale una superficie continua. La normale, infatti, varia e quindi anche il colore associato a triangoli vicini.



©Alan Watt

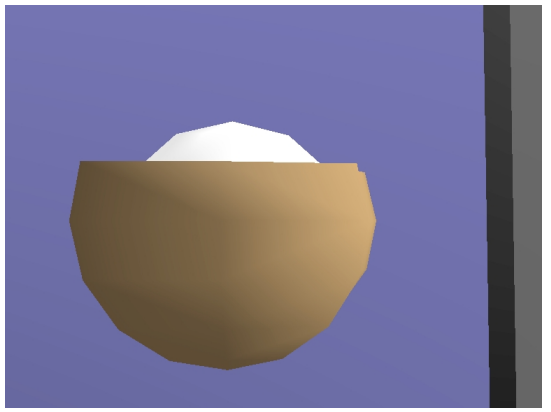
- Anche nel caso di superfici piatte, se osservatore e luce sono vicini, mentre $\mathbf{n}$ è costante su tutta la superficie, $\mathbf{v}$ ed $\mathbf{l}$ cambiano da triangolo a triangolo, quindi i triangoli hanno colori diversi, e si può percepire la transizione.

©Alan Watt

Gouraud shading

- È la tecnica di shading interpolative più antica, e la più semplice (veloce) che produce una variazione dell'intensità atraverso il poligono.
  - si considerano le normali di vertice;
  - si calcola il colore in ogni vertice usando il modello di illuminazione locale;
  - si calcola l'intensità luminosa dei punti interni al poligono con interpolazione scan-line.
- Nota: se le normali di vertice di un poligono sono tutte uguali (come nel caso di una superficie poliedrica), si ha lo stesso effetto del flat shading, visto che ai vertici del poligono viene associato il medesimo colore (assumendo che i vettori $\mathbf{v}$ ed $\mathbf{l}$ siano costanti).

- **Vantaggi**: attenua il salto di colore tra facce adiacenti, è semplice e veloce (è solo poco più oneroso di flat).
- **Svantaggi**: Non elimina completamente la percezione dei poligoni e non rende bene le riflessioni speculari (highlights).



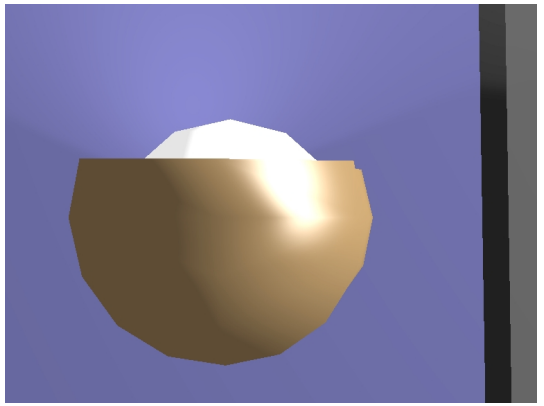
©Alan Watt

©Alan Watt

Phong shading

- Maggiore realismo si ottiene con il **metodo di interpolazione di Phong** (da non confondersi con il modello di illuminazione locale di Phong).
- Le normali dei vertici vengono interpolate all'interno del poligono (con interpolazione scan-line + normalizzazione).
- La normale interpolata viene usata nel modello di illuminazione locale per calcolare il colore dei punti interni al poligono.
- Il metodo "cattura" le riflessioni speculari poiché usa il modello di illuminazione locale anche sui punti interni del poligono.

- **Vantaggio**: buon realismo.
- **Svantaggi**:
  - è circa 5 volte più lento di Gouraud. Per questo spesso confinato all'uso off-line.
  - bisogna "tirarsi dietro" le normali fino alla fase di scan-conversion (quando vengono dipinti i poligoni). Infatti OpenGL non lo supporta per questo motivo: le normali vengono abbandonate dopo la proiezione prospettica

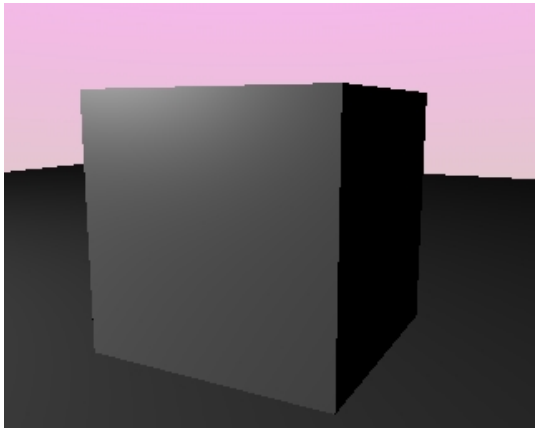


©Alan Watt

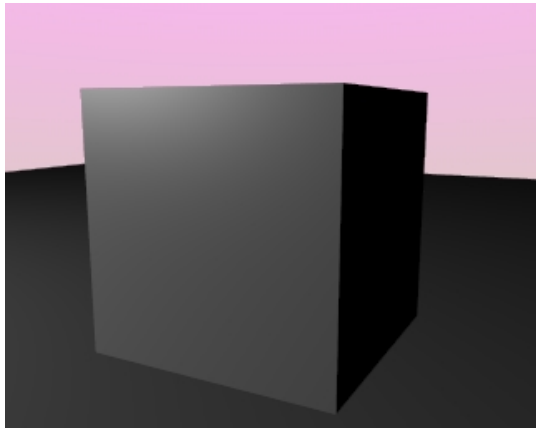
©Alan Watt

Problema dell'aliasing

- Un argomento di cui non abbiamo parlato, e di cui non parleremo, è quello dell'**aliasing**
- Nella costruzione di una immagine per la natura discreta del *raster display* (o del *frame buffer*, che è lo stesso) è inevitabile la comparsa di artefatti che degradano la qualità dell'immagine
- L'esempio più tipico sono le "scalette" che si formano quando si disegnano linee rette
- Vi sono vari metodi di **anti-aliasing** e i più comuni sono basati sul calcolo di medie di shading tra pixel vicini



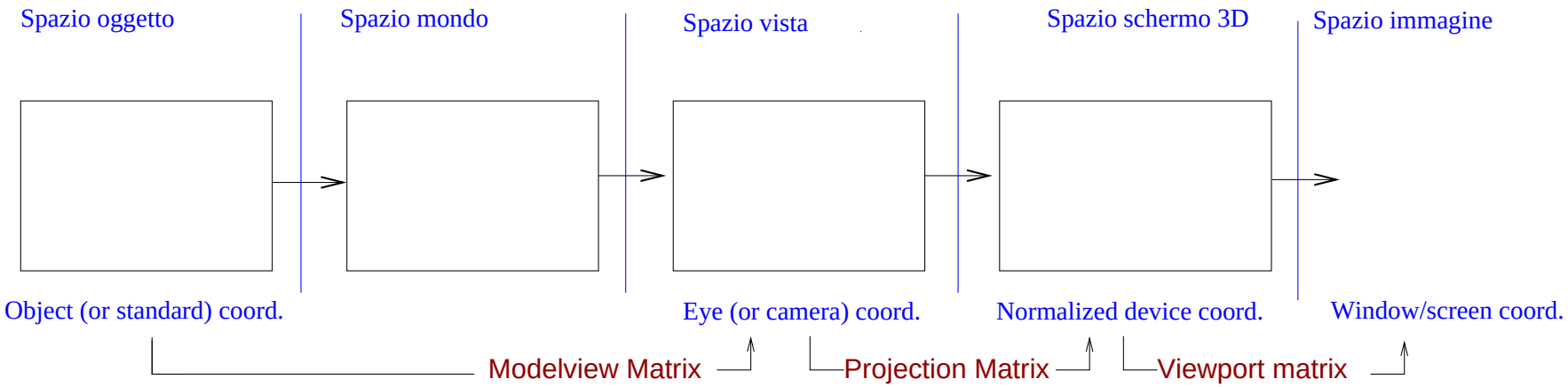
(1) Aliasing



(2) Anti aliasing

La rendering pipeline di OpenGL

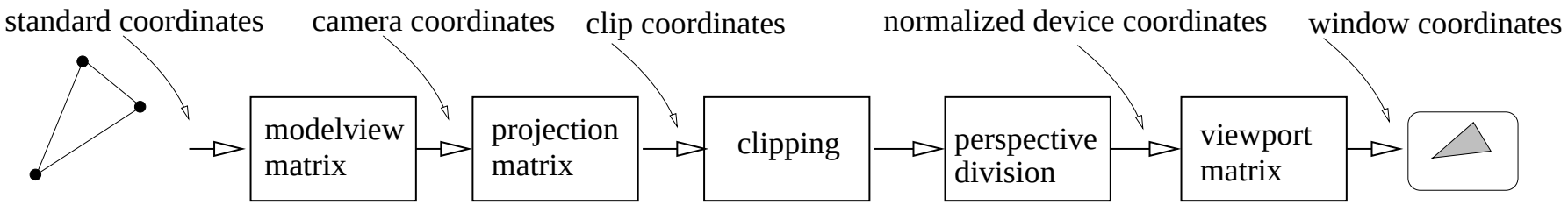
- In OpenGL i sistemi di riferimento non si mappano esattamente su quelli che abbiamo visto precedentemente:



- In OpenGL non esiste lo spazio mondo. L'idea e che questa è una nozione che attiene alla applicazione.
- Tra le coordinate camera e le NDC ci sono le clip coordinates.
- Nel nostro sistema le coordinate immagine sono 2D, mentre in OpenGL si distingue tra le *window coordinates* che conservano la pseudo-profondità le *screen coordinates*, che sono 2D e coincidono con queste ultime a meno della pseudo-profondità.

Storia di un poligono

- Vediamo ora sommariamente cosa accade ad un poligono quando attraversa la pipeline grafica di OpenGL



- Una maglia poligonale viene inserita nella pipeline grafica, assieme alle **normali** associate ai suoi vertici.
- I vertici vengono quindi trasformati dalla **modelview matrix**, che li trasforma dal sistema di riferimento mondo al sistema di riferimento telecamera (eye).
- Qui il **modello di illuminazione locale** viene applicato ai vertici, usando le normali, in modo da assegnare un colore a ciascun vertice.

- Andando avanti nella pipeline viene applicata la **projection matrix**. Le coordinate risultanti prendono il nome di "clip coordinates".
- Si effettua il **clipping** dei poligoni.
- I vertici attraversano quindi la **divisione prospettica**, passando da una rappresentazione 3D ad una rappresentazione 2D + pseudo-profondità. Le normali si perdono, ma si mantiene il colore associato ai vertici.
- Siamo in **normalized device coordinates** , che variano tra -1 ed 1.
- Per mappare queste nella immagine finale (**window coordinates**) si applica la **viewport matrix** (la pesudodepth si mappa tra 0 ed 1).
- Si procede alla **rasterizzazione** del poligono: i valori di pseudo-profondità e colore sui vertici vengono interpolati all'interno del poligono nel corso della sua scan conversion (con interpolazione scan-line), per determinare
  - la visibilità dei pixel interni (depth-buffer)
  - il loro colore (Gouraud shading)

Frammenti

- Una precisazione: l'output della rasterizzazione tecnicamente non sono pixels ma **frammenti** (*fragments*) che sono entità individuate in NDC o window coordinates e possiedono tre attributi: colore, (pseudo)profonfità e coordinate texture. I pixel compaiono alla fine della pipeline, vivono nello screen space e possiedono solo il colore.
- Un frammento è la più piccola unità in cui viene discretizzato un poligono. Normalmente viene generato un frammento per cicun pixel, ma ci possono essere anche più frammenti per un pixel (supersampling).