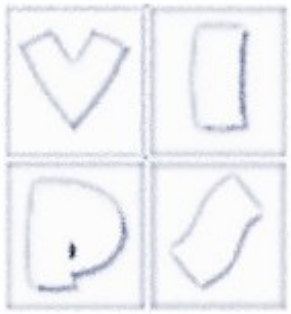


Grafica al calcolatore
-
Computer Graphics

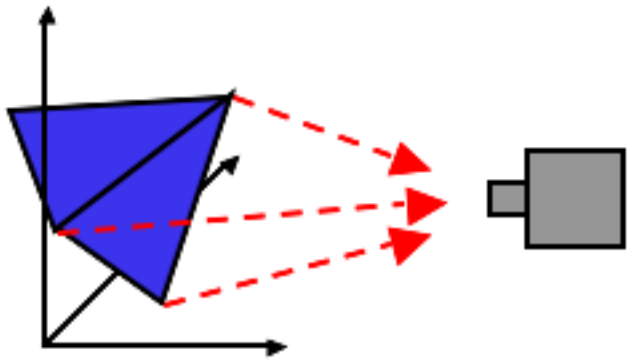
Pipeline di rasterizzazione

Rasterization pipeline

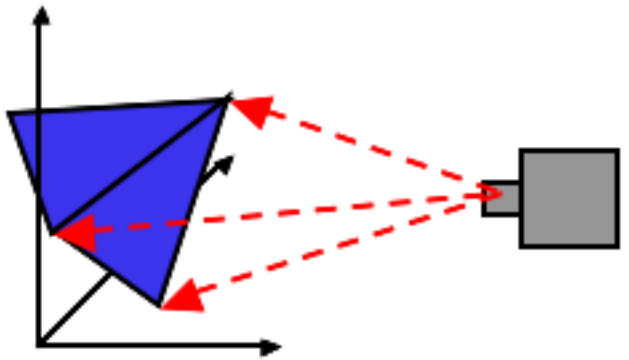
- Abbiamo descritto la procedura intuitiva del ray-casting
- Abbiamo tuttavia già visto che le applicazioni grafiche usano un metodo diverso. Perché?
  - Anche il solo ray casting con un modello di illuminazione locale è troppo oneroso in termini computazionali, a causa dei calcoli delle intersezioni raggi-oggetti (per non parlare del ray tracing)
- Si usa un diverso paradigma: la rasterizzazione.
- Non più modelli generici, ma maglie poligonali, le altre rappresentazioni si possono ricondurre a questa.
  - ray casting è invece trasversale rispetto alla modellazione della scena

Rasterization pipeline

- Invece che gettare (to cast) raggi sulla scena, proietto la scena (composta di poligoni) sul piano immagine.
- La proiezione di un poligono è ancora un poligono, che ha per vertici le proiezioni dei vertici
- La rasterizzazione è un esempio di rendering in object-order, mentre ray-casting è un esempio di rendering in image-order..



Rasterization:
Project geometry forward



Ray casting:
Project image samples backwards

Problemi

- Eliminato il calcolo delle intersezioni raggio-oggetto. Ma ho molti altri problemi:
  - Superfici nascoste: 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. Nella rasterizzazione, 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. Nella rasterizzazione devo 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 stessa. Alla fine devo disegnare un poligono su una matrice di pixel, decidendo dunque quali pixel vi appartengono e quali no.

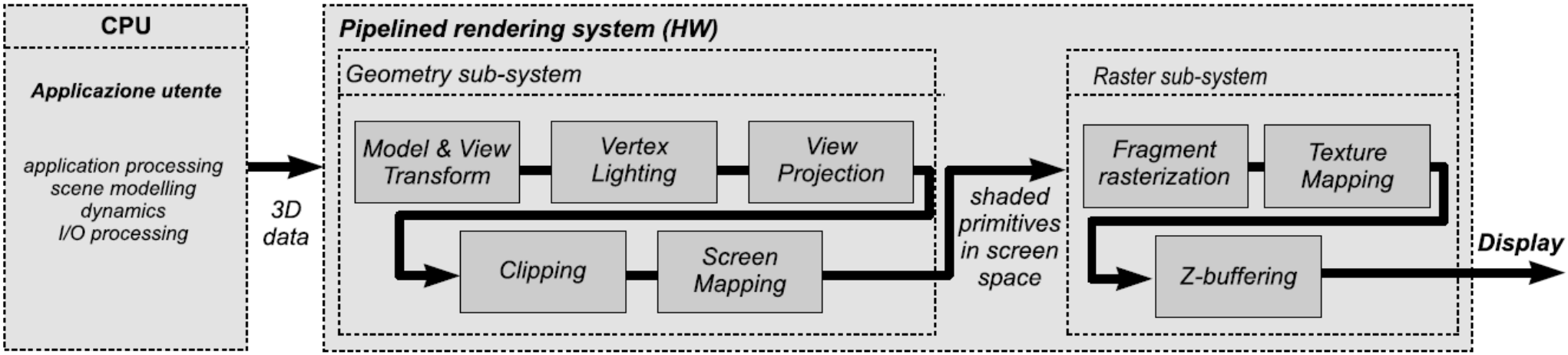
Problemi

- 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 (abbiamo visto come...).
- Effetti globali di illuminazione: Nel ray casting era facile ottenere le ombre tracciando gli shadow rays. Più in generale, con il ray-tracing si possono ottenere effetti di illuminazione globale che invece nella rasterizzazione richiedono trucchi più o meno elaborati (oppure se ne fa a meno).

- La soluzione di questi problemi è implementata nella pipeline in modo efficiente e la rasterizzazione risulta più veloce del ray casting.

Pipeline di rasterizzazione

- Avevamo già visto questo schema
  - Si parte da una lista di primitive (triangoli)
  - Si operano una serie di processi a catena (pipeline), parte in CPU, parte in GPU
  - Pipeline divisa in due parti: la prima geometrica, la seconda "raster" che lavora sui pixel dell'immagine (la vera rasterizzazione)

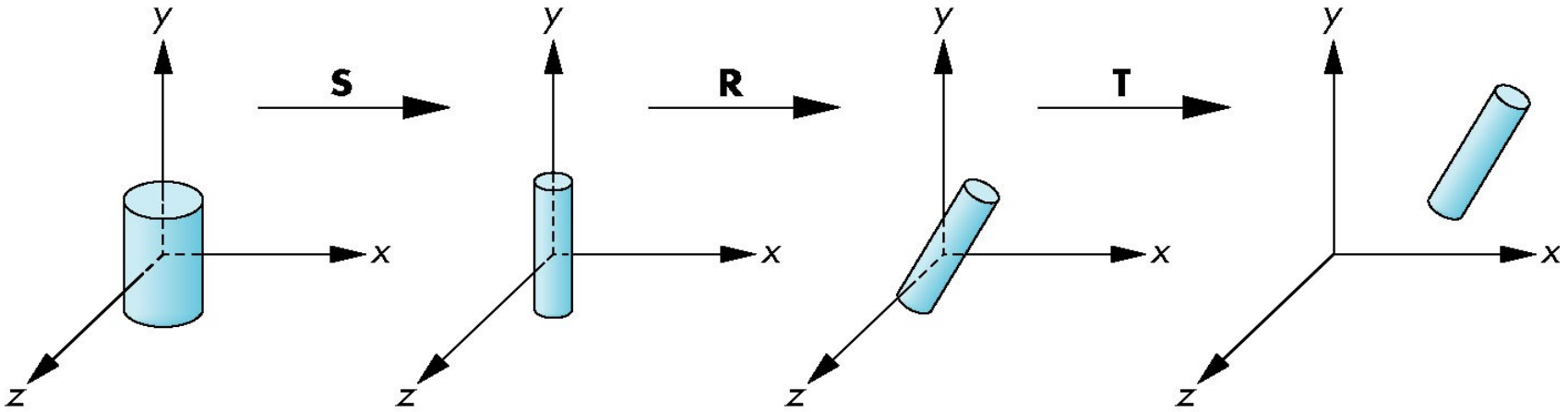


Implementazione ed evoluzione

- In OpenGL possiamo considerare la pipeline grafica come una macchina a stati ove si definiscono delle primitive e delle operazioni che le modificano
- Il programma definisce queste e le passa alla pipeline che dà in ouput le immagini
- Primitive di due tipi: geometriche e raster
  - Vediamo ora quali sono le operazioni base per realizzare la visualizzazione di una scena triangolata

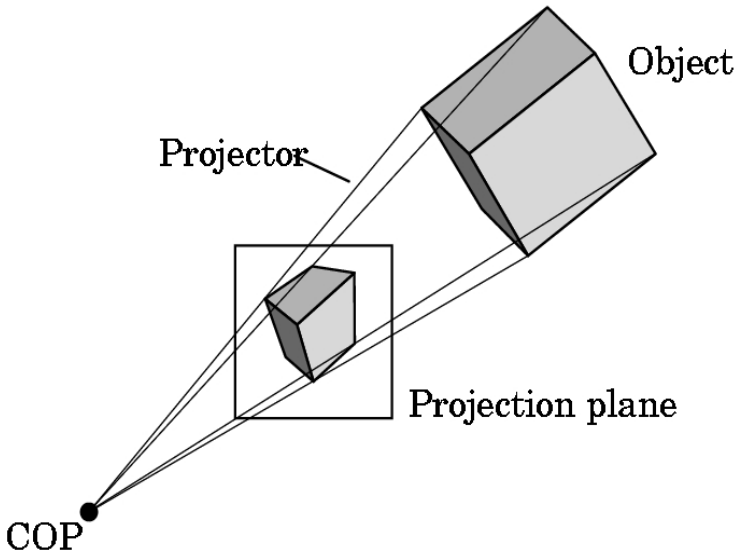
Pipeline geometrica

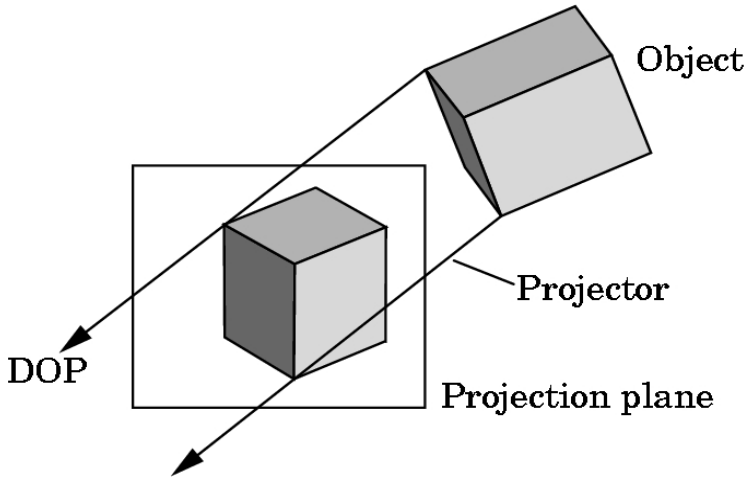
- Devo
  - mettere assieme gli oggetti della scena
  - rappresentare gli oggetti nel sistema di riferimento della telecamera virtuale
  - Proiettarli
  - Posso applicare le varie trasformazioni geometriche



Proiezione

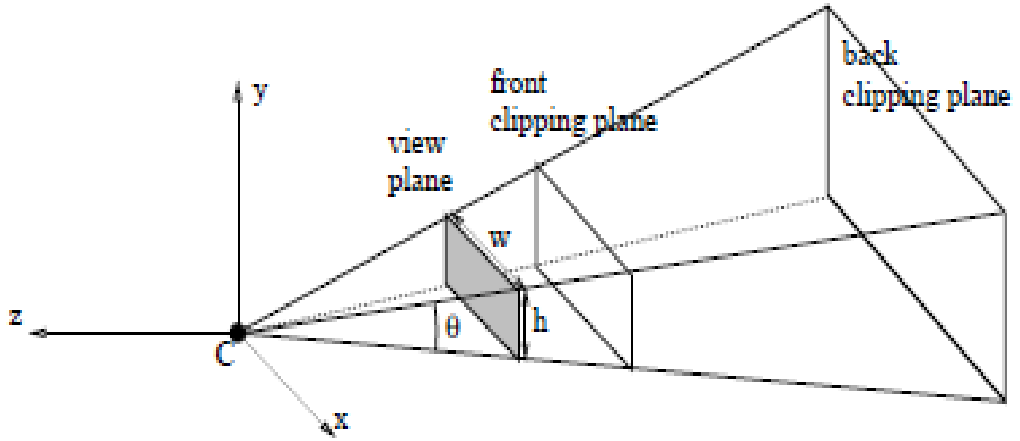
- Quindi metto tutti gli oggetti con trasformazioni "di modellazione" (traslazioni, rotazioni, scalature) in un riferimento "coordinate mondo"
- Poi devo proiettare sul piano immagine
  - Proiezione prospettica o affine





Trasformazione di vista

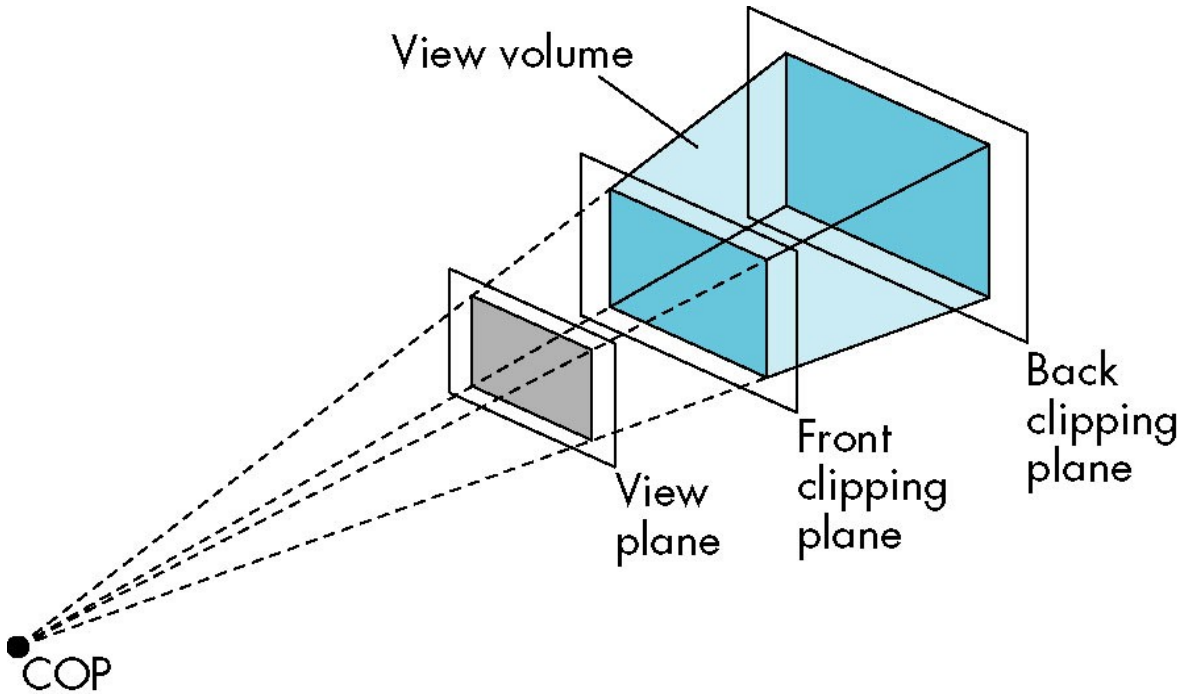
- Supponiamo di usare la prospettica.
- Devo passare in coordinate solidali con la camera (view space, come in figura) e definire cosa "vedo" dato che lavorando in object order devo buttare via il resto (clipping)



- Convenzioni
  - Il piano immagine è davanti alla telecamera a distanza 1 (d era 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) e quello 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



Volume di vista

- Gli oggetti che cadono dentro il volume di vista vengono disegnati.
- Per proiettarli con M o $(\mathrm{M}_{\perp})$ 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):

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

Trasformazione prospettica

- La proiezione prospettica M potrebbe essere applicata ai punti P (vertici dei triangoli) espressi nello spazio vista ottenendo le coordinate sul piano immagine
- Invece si fa una cosa più contorta: si applica la "trasformazione prospettica", che mappa il frustum in un parallelepipedo retto (volume di vista canonico), poi mappato sul piano immagine con una proiezione ortografica.
- Motivo: semplificare le operazioni di clipping (buttar via ciò che sta fuori)

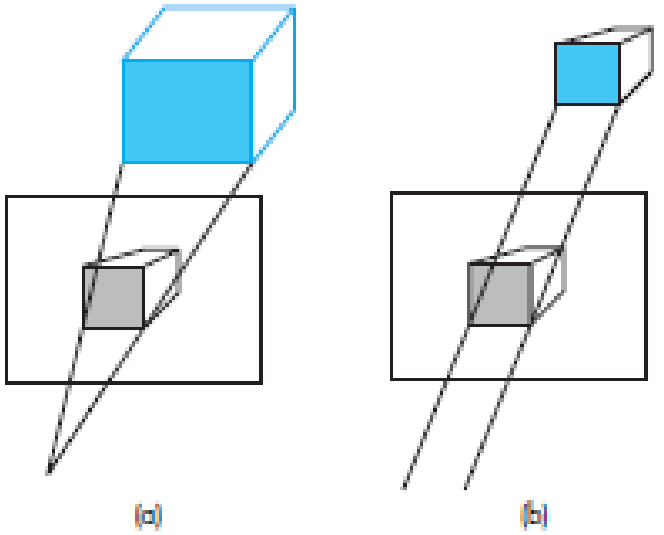


FIGURE 4.23 Predistortion of objects. (a) Perspective view. (b) Orthographic projection of distorted object.

Trasformazione prospettica

- Consideriamo la matrice 4 × 4 N, non singolare (simile alla M) (che prende il nome di matrice di matrice di trasformazione o di normalizzazione prospettica).

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

- Applicando N a P si ottiene la 4-pla $(x, y, \alpha z+\beta, -z)$ che, dopo la divisione prospettica diventa $(-x/z, -y/z, -\alpha-\beta/z)$
- Le prime due componenti sono identiche alla proiezione standard, ma la terza componente (pseudo-profondità) è diversa: $z_s = \alpha z+\beta, -z$.
- per valori di α, 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.

Volume di vista canonico

La trasformazione (normalizzazione) prospettica N mappa punti 3D in punti 3D, e scegliendo opportunamente α, β possiamo fare in modo che mappi il frustum di vista in un parallelepipedo con coordinate arbitrarie. Gli oggetti vengono distorti di conseguenza.
Proiettando questo parallelepipedo ortogonalmente (eliminando la terza coordinata, cioè z_s) si ottiene la proiezione prospettica desiderata. In OpenGL il volume di vista canonico è un cubo:

$$-1 \le x \le 1 \quad -1 \le y \le 1 \quad -1 \le z \le 1$$

Volume di vista canonico

- 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, β in modo che l'intervallo di profondità z $[-n,-f]$ venga mappato in z_s $[-1, 1]$ (notare l'inversione di z).
- Si ottiene

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

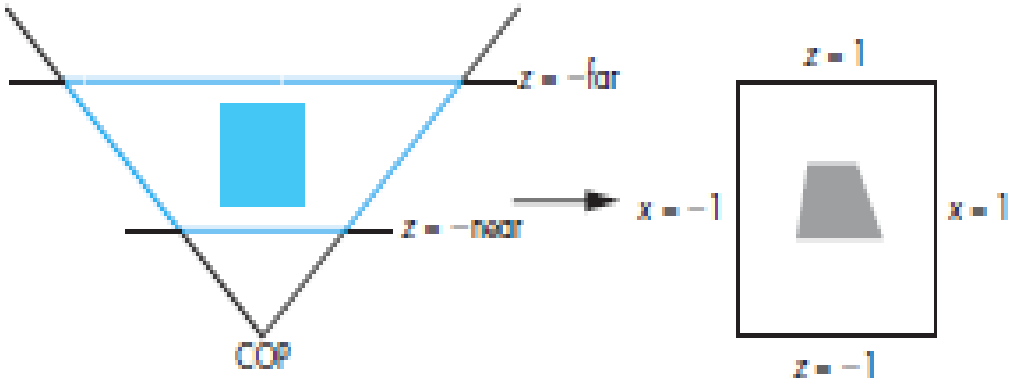


FIGURE 4.39 Perspective normalization of view volume.

Volume canonico

- Se l'angolo di vista $\theta = 90$ ed il fattore d'aspetto $a = 1$, allora la matrice N con α, β calcolati come sopra trasforma il frustum nel volume di vista canonico
- Nel caso generale, bisogna considerare θ e a, moltiplicando N per la seguente matrice di scalatura, dove $\gamma = 1/\tan(\theta/2)$

che ha l'effetto di trasformare la piramide generica in piramide retta a base quadrata.

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

- Attenzione: questa matrice agisce anche sulle coordinate x ed y.
- 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.

Caso ortogonale

- 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.
- 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.

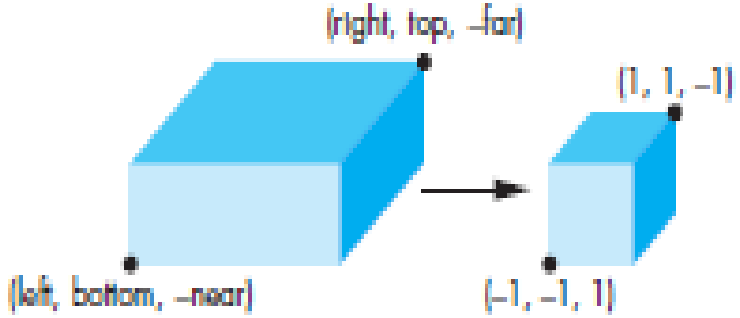
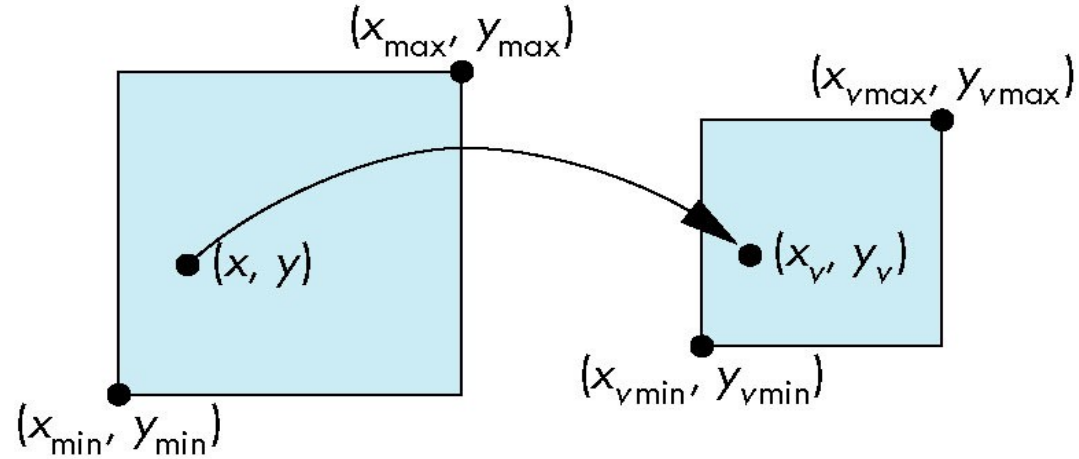


FIGURE 4.25 Mapping a view volume to the canonical view volume.

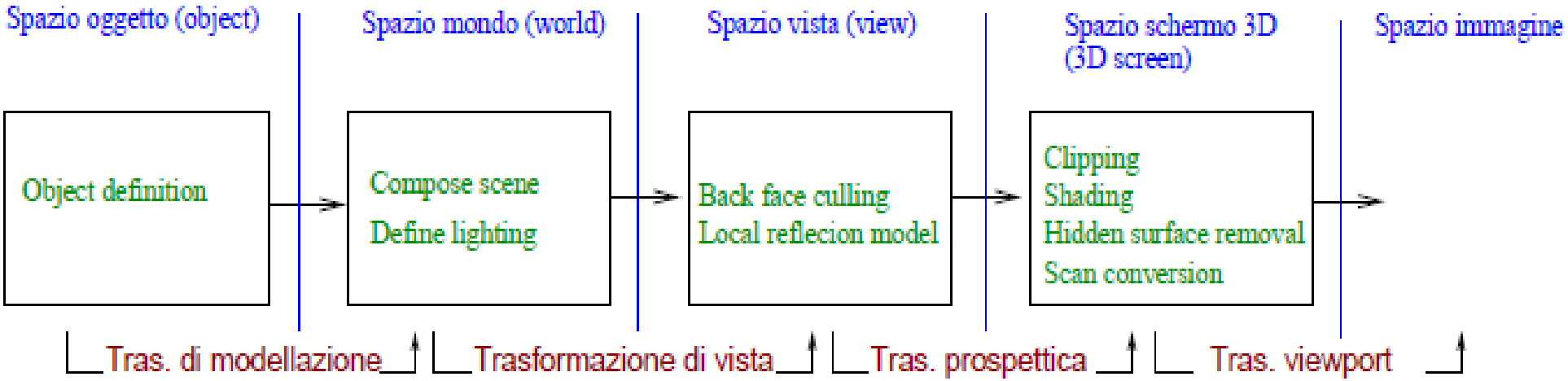
Trasformazione viewport

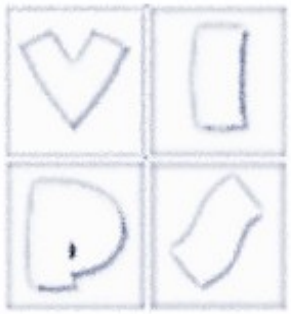
- Viewport: porzione del display all'interno della quale viene visualizzato il risultato del rendering
- La trasformazione viewport si applica dopo la proiezione ortografica e dipende dalle caratteristiche fisiche del display
- Ai punti proiettati dal volume di vista canonico viene applicata una matrice di trasformazione affine che:
  - ripristina il fattore di aspetto corretto per l'immagine (distorto dalla trasformazione prospettica)
  - scala e trasla l'immagine



Pipeline geometrica

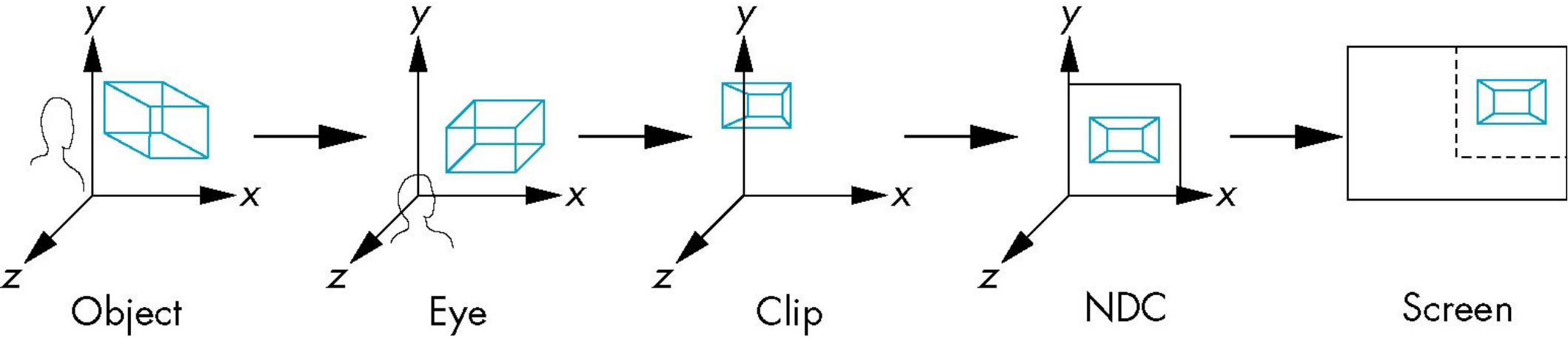
- Quindi la pipeline geometrica coinvolge diversi sistemi di riferimento e trasformazioni tra di essi.
- In ciascuno di essi avvengono delle operazioni, da parte del codice o della pipeline hardware





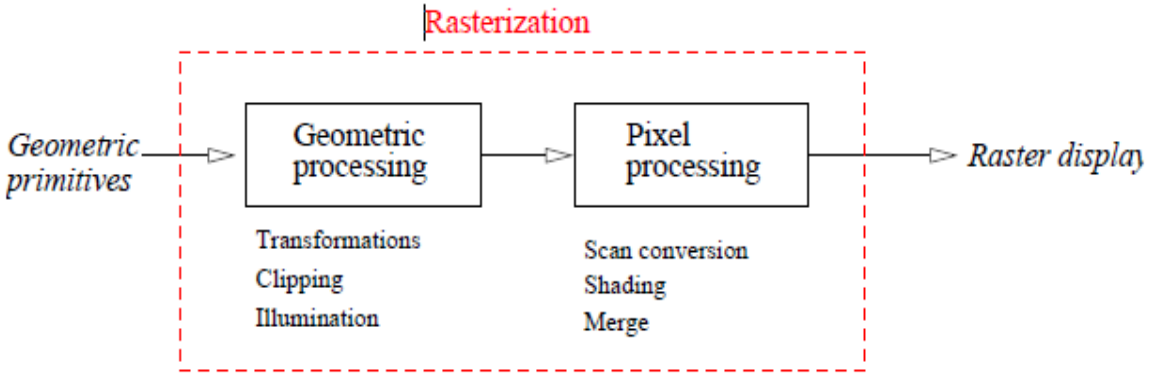
Nomi

- I nomi possono variare, ma il senso è quello
- Es da Angel



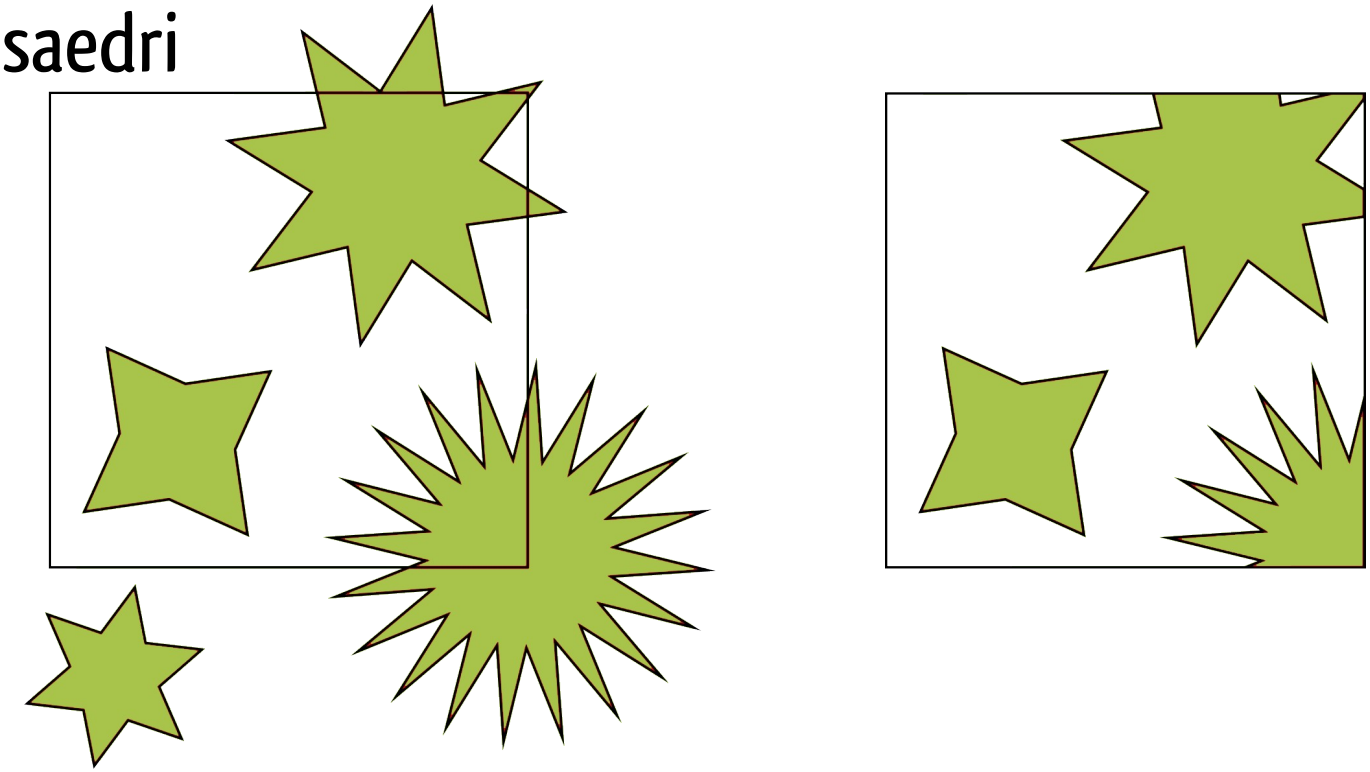
Operazioni

- Clipping: devo eliminare le parti fuori dal frustum e tagliare i poligoni a metà: ricavo lista di poligoni ridotta
- Illumination: calcolo il colore con l'equazione semplificata del rendering sul vertice (alternativa: farlo dopo)
- Scan conversion: trovo i pixel sovrapposti alla proiezione del triangolo: genero i frammenti (pixel)
- Illumination/shading: calcolo il colore del frammento
  - Interpolando i colori dei vertici
  - Oppure calcolando a questo punto l'illuminazione
- Rimuovo superfici nascoste (Back-face culling)
- Mappatura texture, effetti di miscelazione di immagini diverse, ecc.



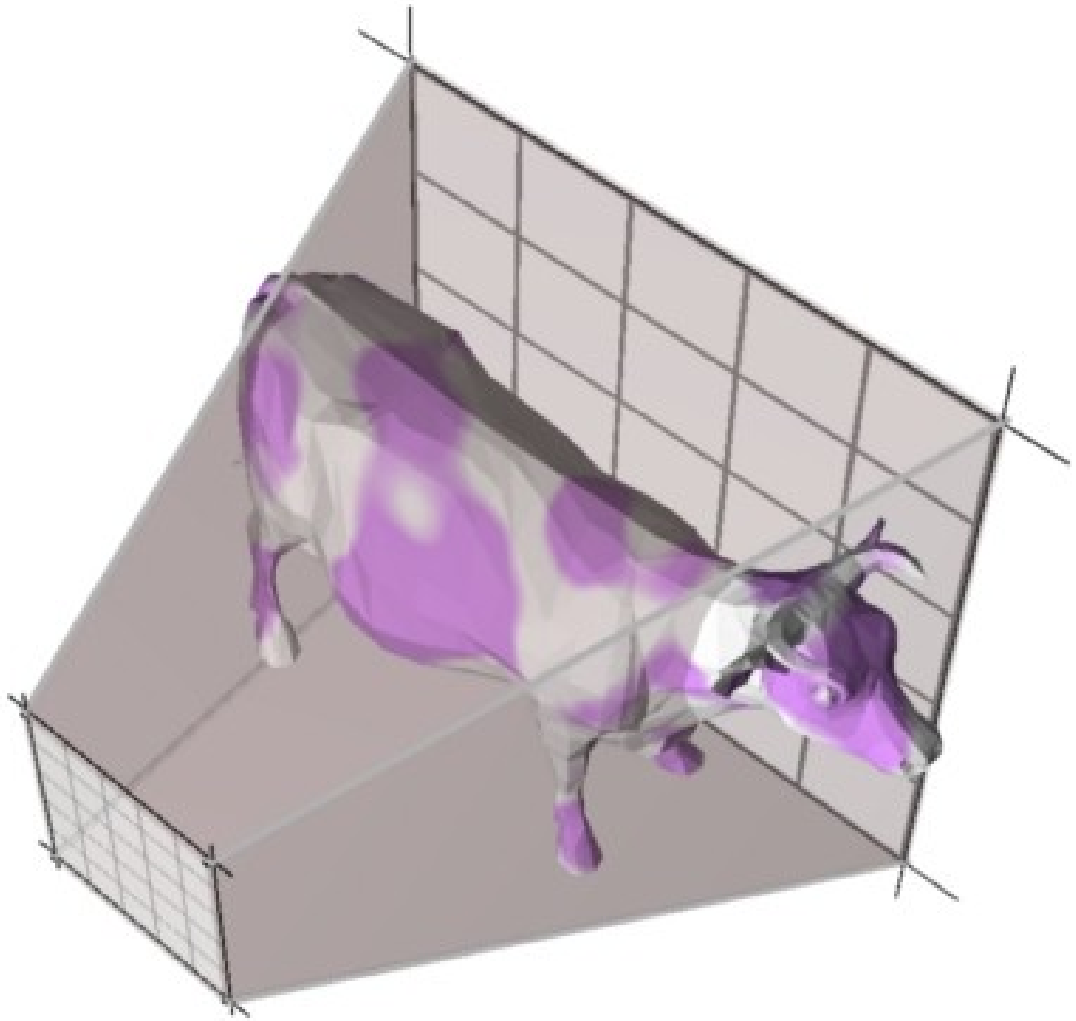
Clipping

- L'operazione di clipping consiste nell'individuare (e rimuovere) le primitive grafiche (o parti di esse) esterne ad una finestra rettangolare o esaedrale oppure, più in generale, esterne ad un poligono o poliedro convesso.
- In computer graphics si è interessati al clipping rispetto a rettangoli o esaedri



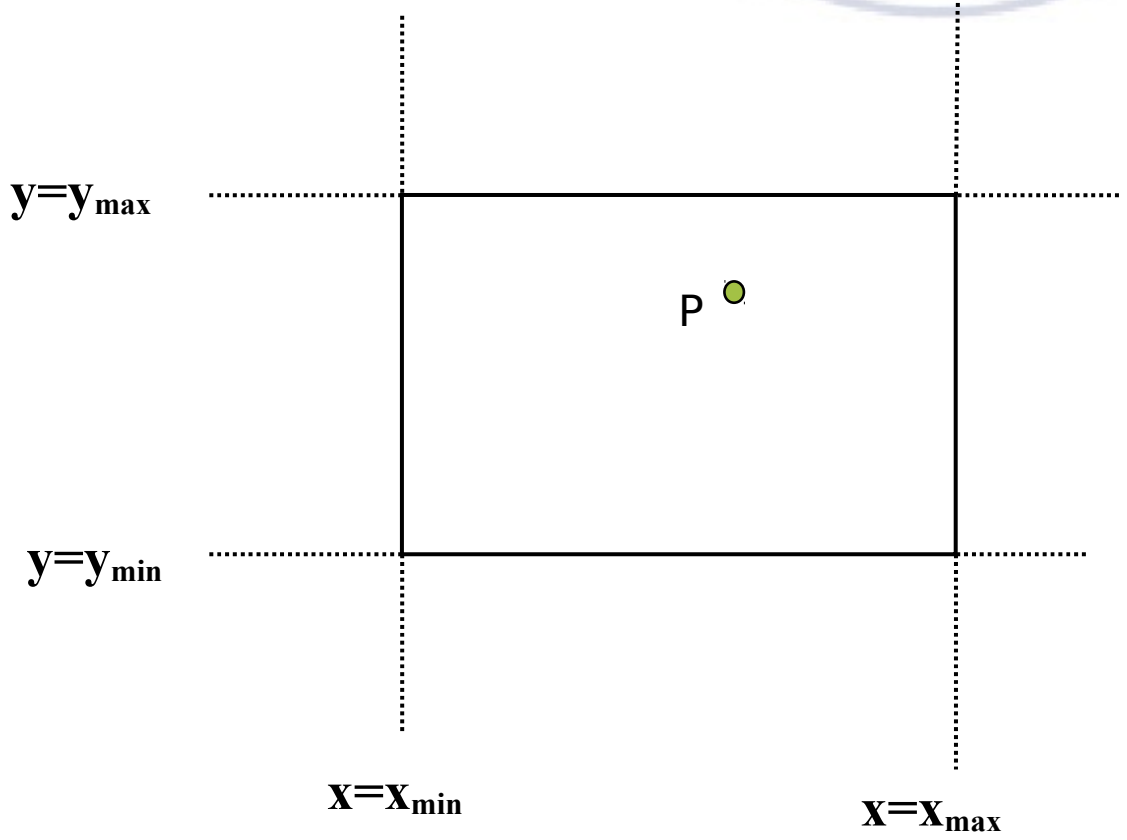
Clipping (generalità)

- Non interessa tutto ciò che non è nel view frustum



Clipping di un punto

- Clipping di un punto: un punto è all'interno del rettangolo di clipping se e solo se sono soddisfatte le 4 disuguaglianze:
  - $x_{\min} \leq x \leq x_{\max}, y_{\min} \leq y \leq y_{\max}$

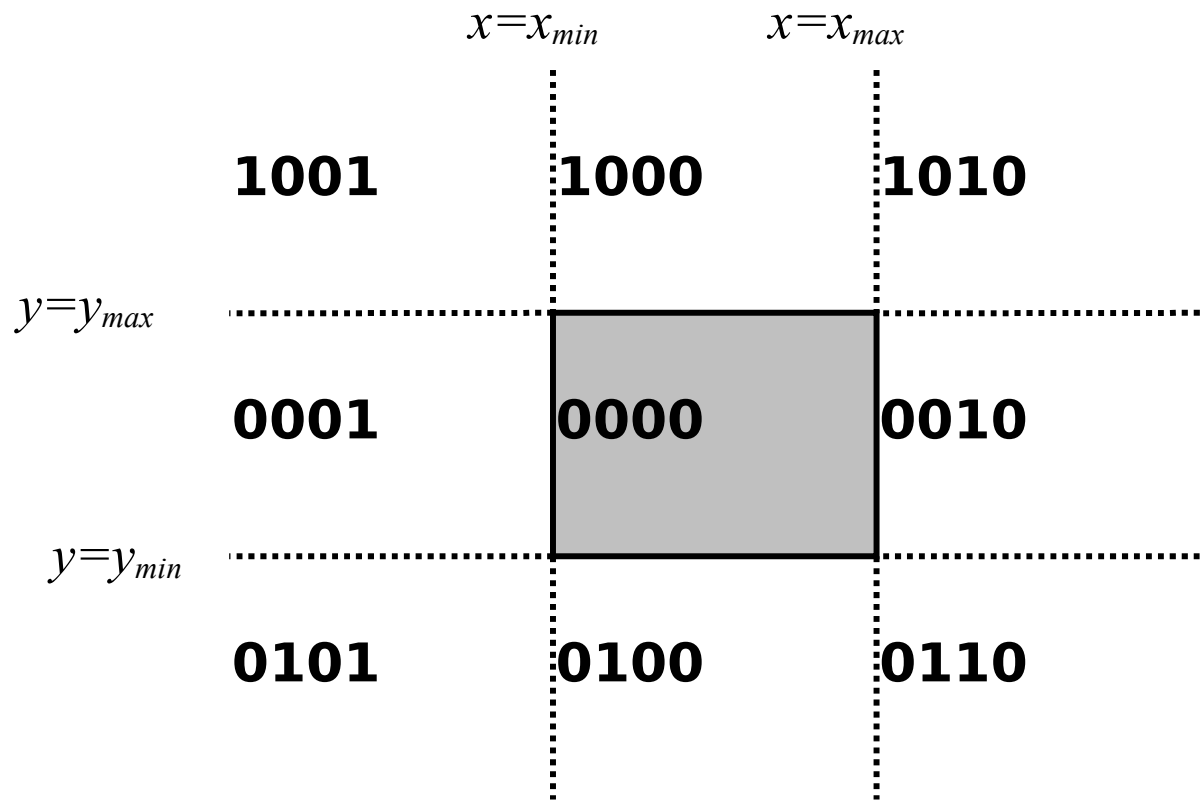


Clipping di un segmento

- Clipping di un segmento: necessario analizzare le posizioni dei suoi punti estremi.
  - Se gli estremi sono entrambi interni al rettangolo di clipping, il segmento è interno;
  - Se un estremo è interno e l'altro esterno, allora il segmento interseca il rettangolo di clipping ed è necessario determinare l'intersezione;
  - Se entrambi gli estremi sono esterni al rettangolo, il segmento può intersecare o meno il rettangolo di clipping e si rende necessaria una analisi più accurata per individuare le eventuali parti interne del segmento.

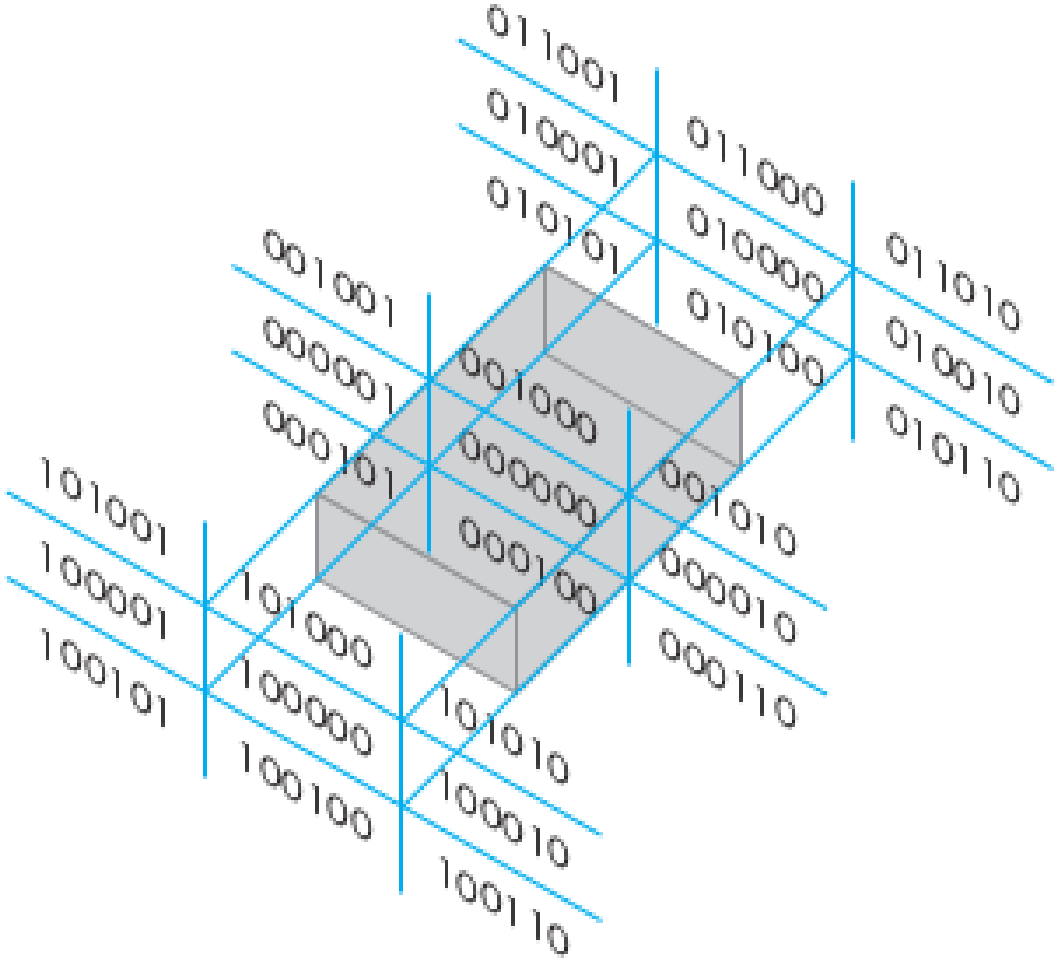
Algoritmi

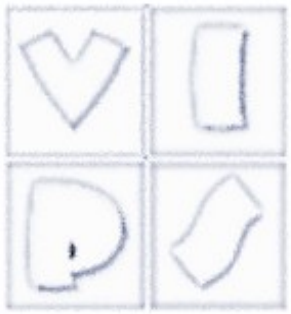
- Diretto: calcolo le intersezioni retta/rettangolo di clipping: inefficiente
- Cohen-Sutherland: codifica binaria regioni, in base ai codici degli estremi (and logico) capisco le intersezioni



3D Cohen-Sutherland

- Easily extended





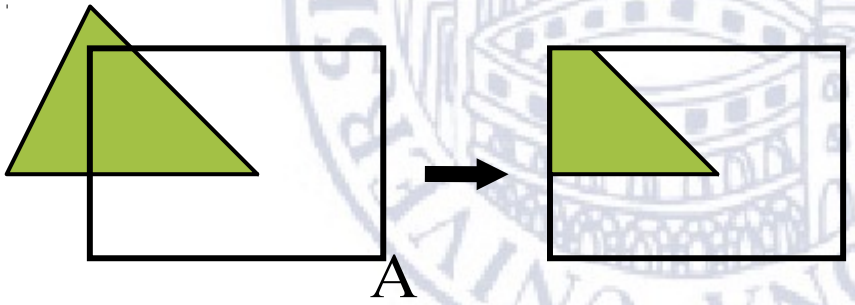
Segmenti: Liang-Barsky

- Un po' più efficiente
- Si scrive il segmento in forma parametrica (t in [0 1])

  $P_0+(P_1-P0)t=P_0+t\mathbf{v}$
- Si trovano le intersezioni con le rette/piani di clipping
- Si calcolano i valori di t di entrata e uscita dalla regione interna
- Si trova il segmento dopo clipping prendendo il massimo t dei punti di entrata se >0 e il minimo in uscita (se <1)
- Direttamente applicabile anche in 3D

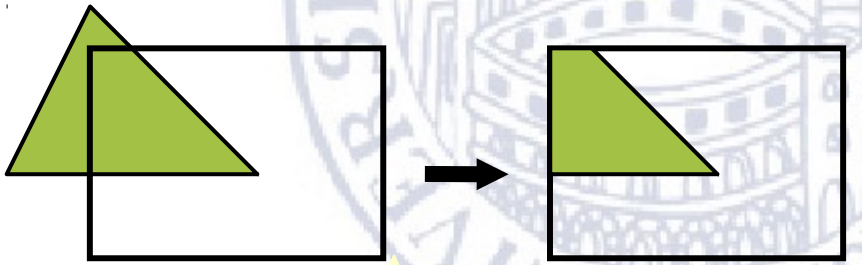
Clipping di un poligono

- Il clipping di un poligono è un'operazione più complessa rispetto al clipping di un segmento per diversi aspetti:
- Dal semplice poligono convesso (A);

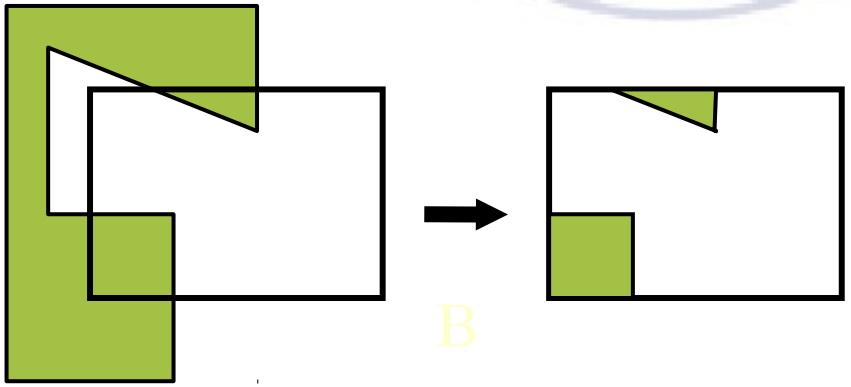


Clipping di un poligono

- Il clipping di un poligono è un'operazione più complessa rispetto al clipping di un segmento per diversi aspetti:
- Dal semplice poligono convesso (A);
- Al poligono concavo che origina più componenti connesse (B);



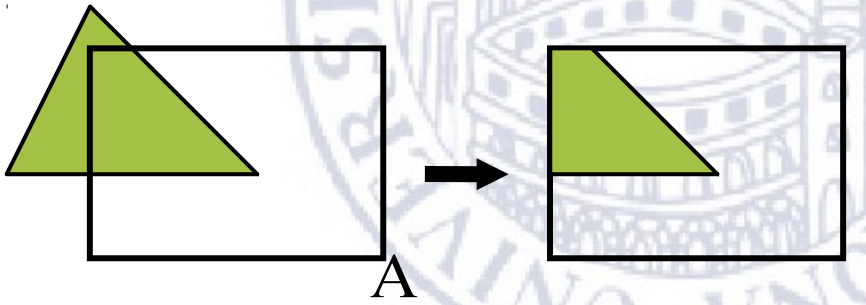
A



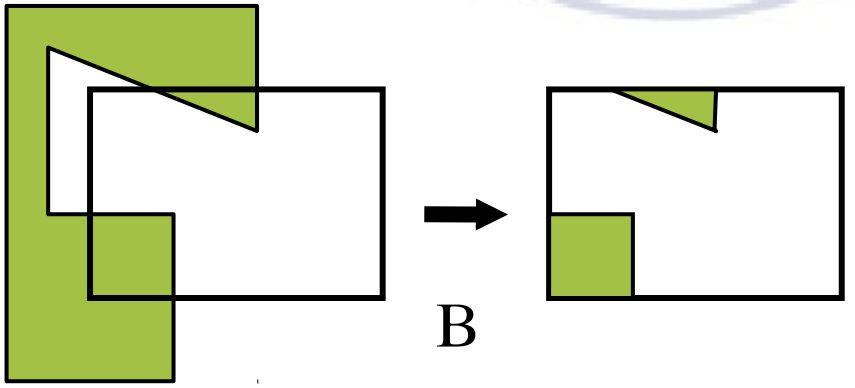
B

Clipping di un poligono

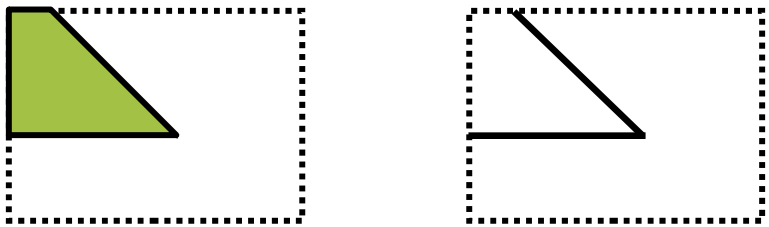
- Il clipping di un poligono è un'operazione più complessa rispetto al clipping di un segmento per diversi aspetti:
- Dal semplice poligono convesso (A);
- Al poligono concavo che origina più componenti connesse (B);
- In ogni caso il risultato consta di uno o più poligoni e non solo segmenti sconnessi (C).



A



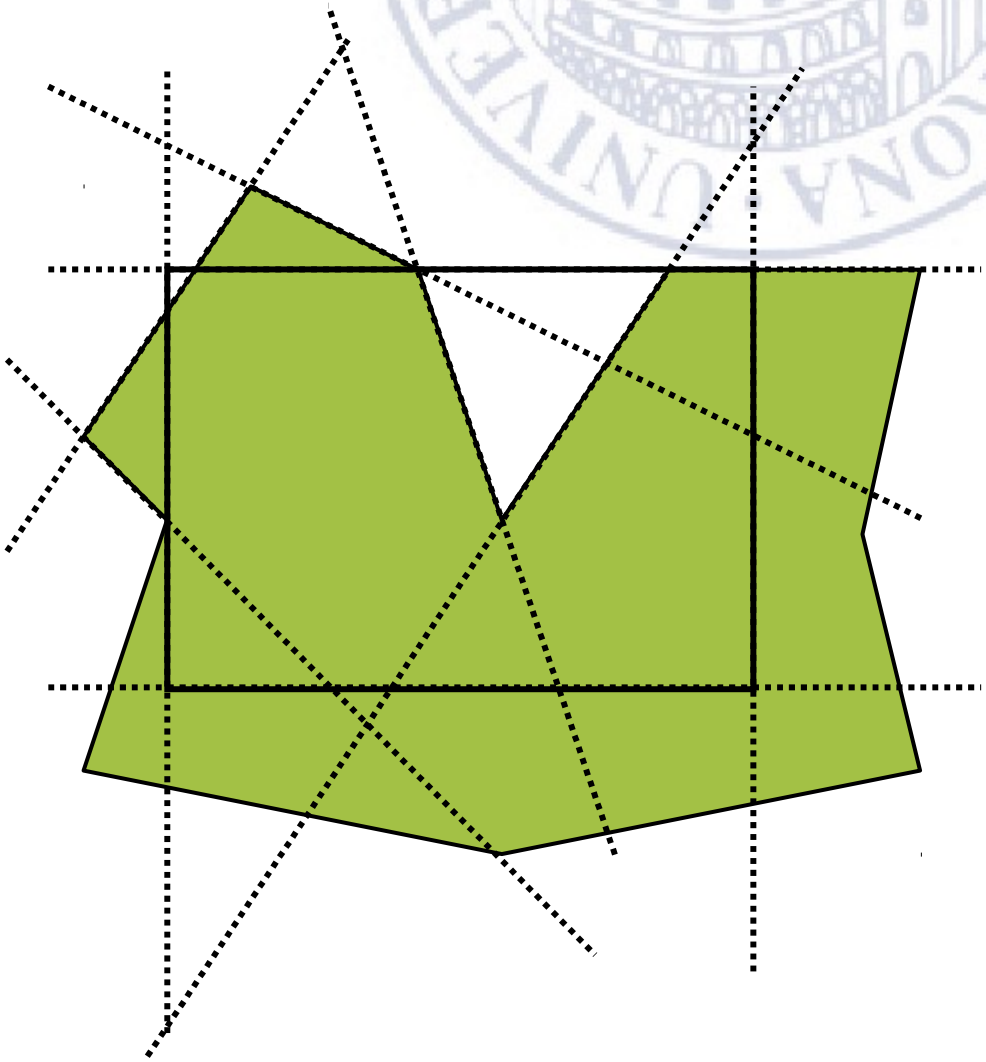
B



C

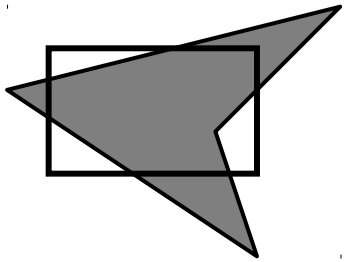
Clipping di un poligono

- L'approccio diretto consiste nel confrontare ogni lato del poligono con le 4 rette che delimitano il rettangolo di clipping;
- Questo approccio implica l'esecuzione di operazioni costose (la determinazione di intersezioni) e spesso inutili.

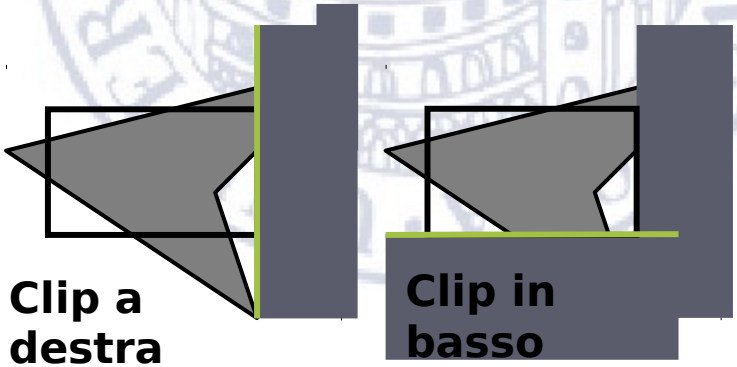


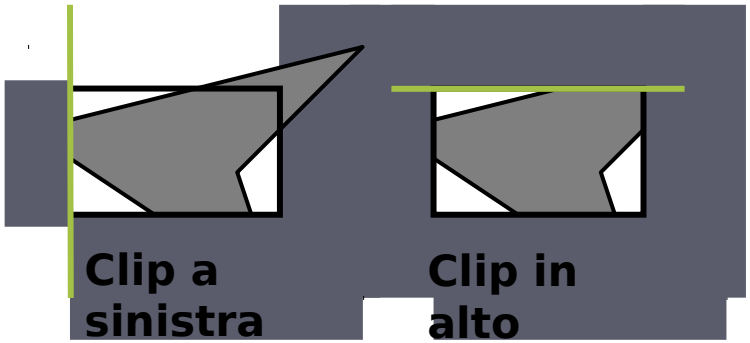
Clipping di un poligono (Sutherland-Hodgman)

- Approccio divide et impera;
- Il problema è ricondotto al clipping di un poligono generico rispetto ad una retta;
- La procedura è applicata sequenzialmente alle 4 rette che definiscono il rettangolo di clipping.



Poligono originale





Clipping con bounding box

- Altro approccio: calcolo la bounding box della primitiva
- Considero intersezione tra bounding box e regione di vista
- Solo se le regioni si intersecano faccio clipping (sull'intersezione)

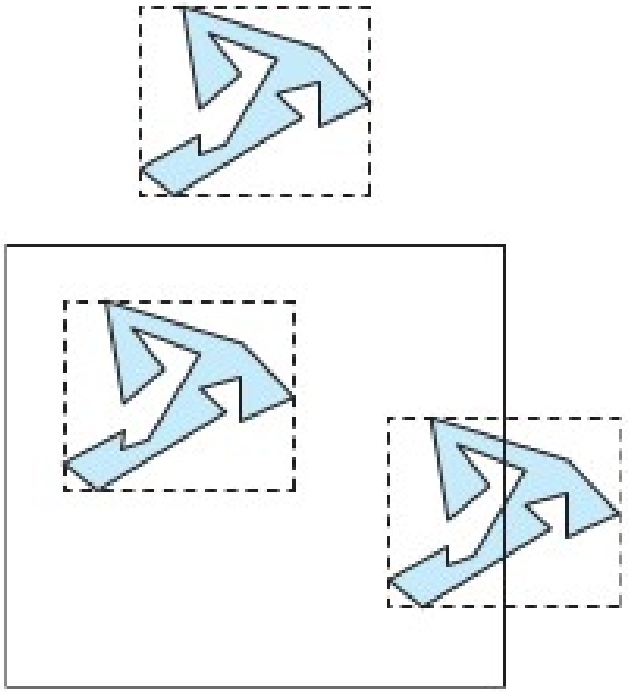


FIGURE 6.19 Clipping with bounding boxes.

Rimozione superfici nascoste (HSR)

- Gli oggetti della scena sono generalmente opachi;
- Gli oggetti più vicini all'osservatore possono nascondere (occludere) la vista (totale o parziale) di oggetti più lontani;
- Il problema della rimozione delle superfici nascoste (HSR, Hidden surface removal) consiste nel determinare le parti della scena non visibili dall'osservatore;
- La rimozione delle superfici nascoste non è solo dipendente dalla disposizione degli oggetti nella scena ma anche dalla relazione esistente tra oggetti e posizione dell'osservatore.
- Quindi in una applicazione interattiva se sposto il punto di vista cambia la visibilità degli oggetti

Rimozione superfici nascoste

- Stiamo analizzando la parte “geometrica” della pipeline grafica. La rimozione però non è necessariamente fatta qui:
  - algoritmi che operano in **object-space** determinano, per ogni primitiva geometrica della scena, le parti della primitiva che non risultano oscurate da altre primitive nella scena. Gli algoritmi operano nello spazio di definizione delle primitive;
  - algoritmi che operano in image-space determinano, per ogni punto “significativo” del piano di proiezione (ogni pixel del piano del piano immagine), la primitiva geometrica visibile “attraverso” quel punto. Gli algoritmi operano nello spazio immagine della scena proiettata.

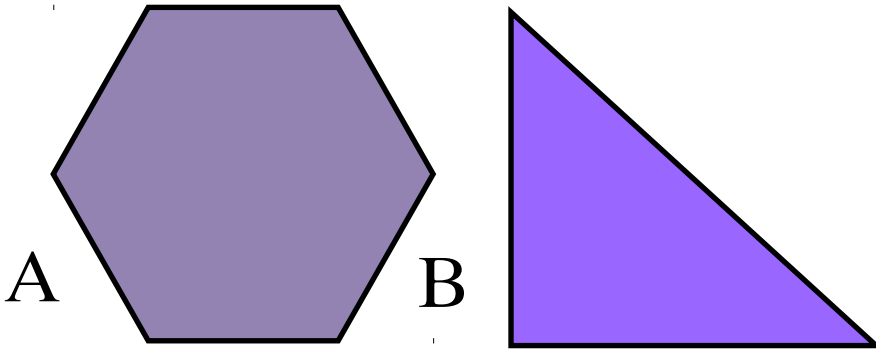
Applicazione o Geometry subsystem

Raster subsystem

Conviene operare in object space?

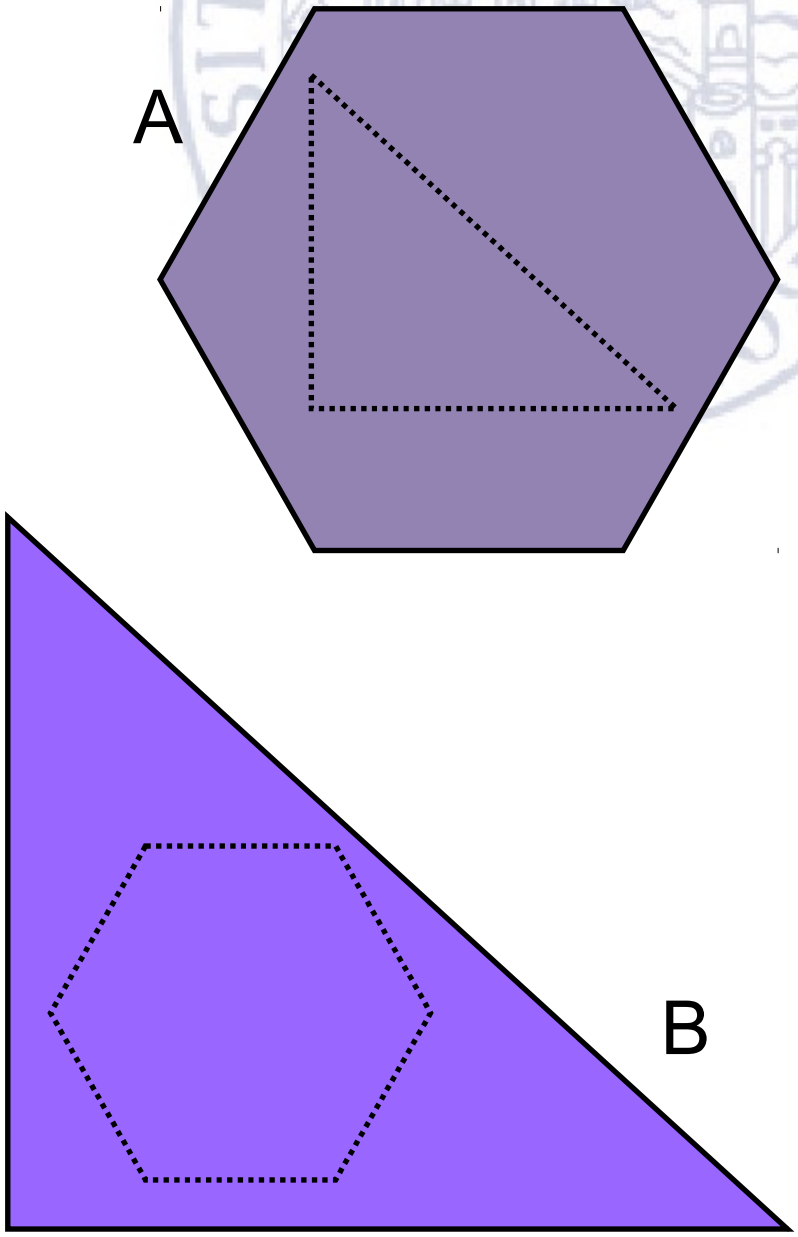
- Nell'ipotesi di una scena 3D composta da k primitive geometriche planari ed opache, si può derivare un generico algoritmo di tipo object-space analizzando gli oggetti a coppie;
- Fissato un punto di vista, le relazioni spaziali di due primitive geometriche A e B possono essere:
  - A [B] oscura B [A]; solo A [B] è visualizzata;
  - A e B sono completamente visibili; entrambe le primitive sono visualizzate;
  - A [B] occlude parzialmente B [A]

è necessario individuare le parti visibili di B [A].



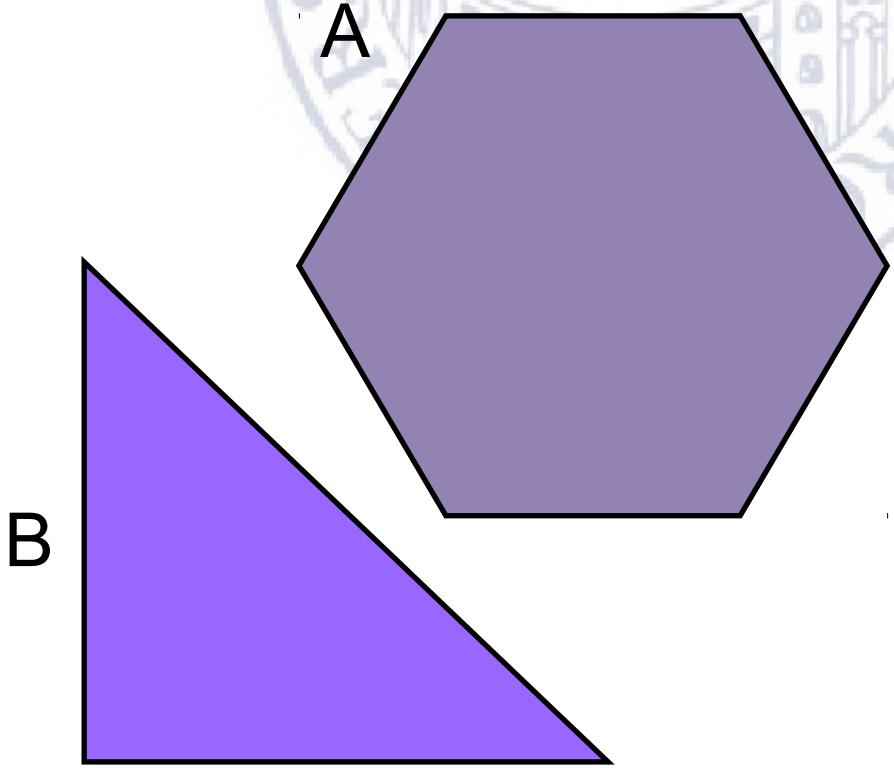
HSR: Object-space

- A oscura B: tutti i punti di A sono più vicini all'osservatore di tutti i punti di B e la proiezione di B sul piano di vista ricade all'interno della proiezione di A. Visualizza A;
- B oscura A: tutti i punti di B sono più vicini all'osservatore di tutti i punti di A e la proiezione di A sul piano di vista ricade all'interno della proiezione di B. Visualizza B.



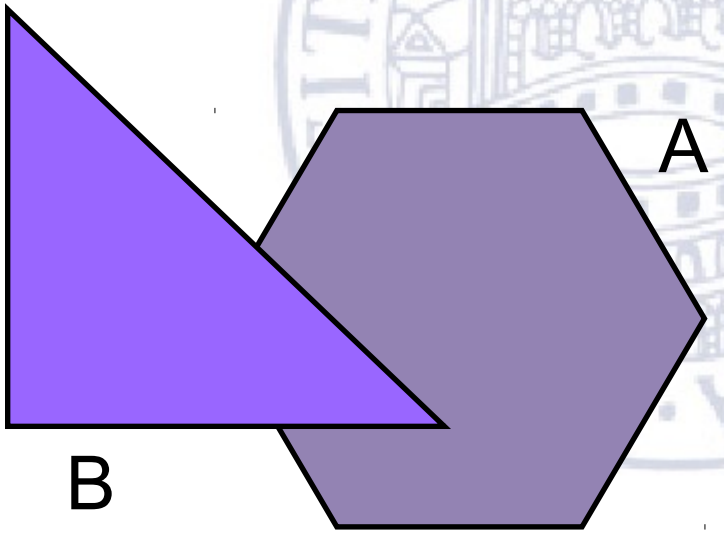
HSR: Object-space

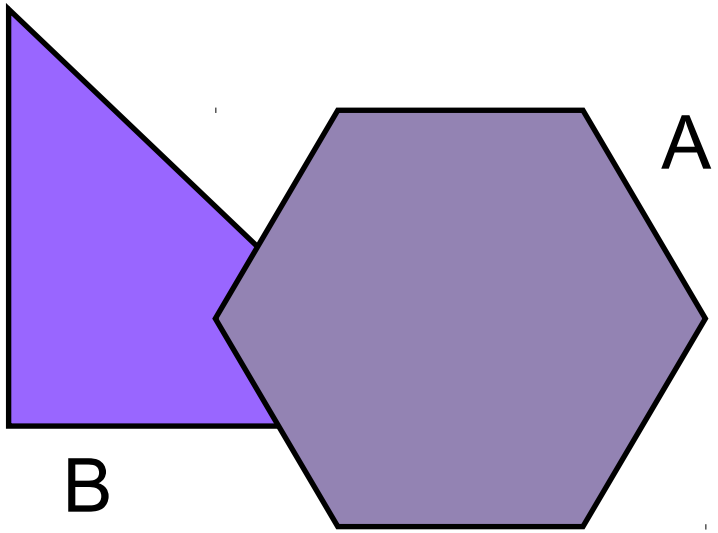
- A e B non si occludono: le due proiezioni di A e B sul piano di vista sono disgiunte;
- Visualizza A e B.

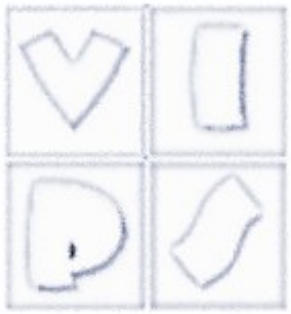


HSR: Object-space

- A e B si oscurano parzialmente: l'intersezione tra le due proiezioni di A e B sul piano di vista è non nulla e diversa sia dalla proiezione di A che da quella di B.
- Individua le parti visibili di ciascuna primitiva.

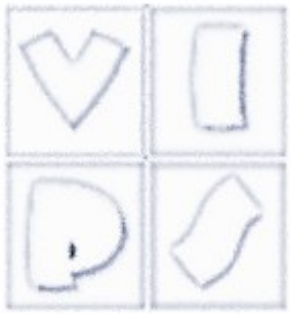






HSR: Object-space

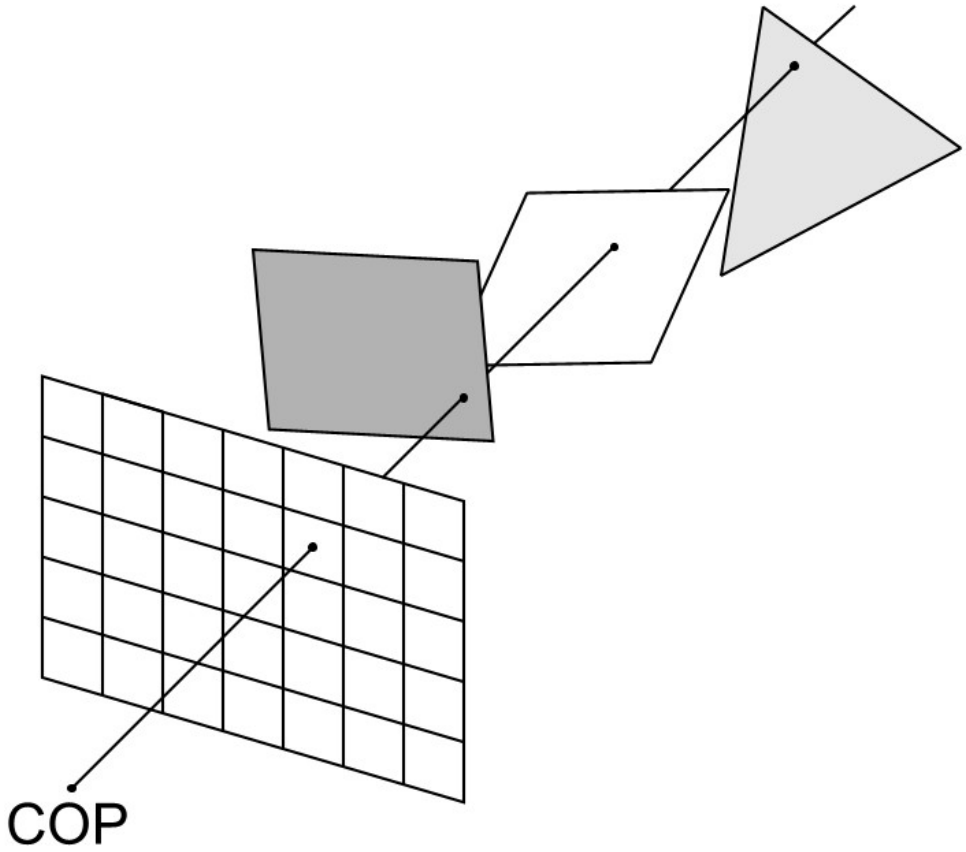
- Un possibile algoritmo risolutivo:
  - Proiettare le *k* primitive geometriche;
  - Al generico passo analizzare la *i-esima* primitiva (i=1, …, k-1) con le rimanenti $k - i$ in modo da individuare le parti visibili.
- La complessità dell'approccio object-space risulta di ordine $O(k^2)$
- L'approccio object-space è consigliabile solo quando le primitive nella scena sono relativamente poche.
- Si può aumentare l'efficienza?

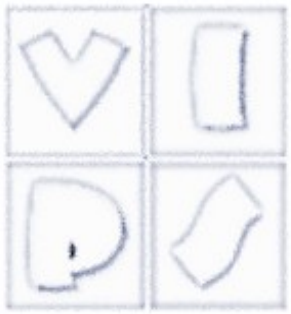


- Sì ricordiamo le tecniche di suddivisione spaziale
  - Riorganizzare gli oggetti della scena (o le loro proiezioni sul piano immagine) in gruppi coerenti dal punto di vista spaziale;
  - Si effettua di solito mediante griglie 2D o 3D che servono a limitare i confronti in profondità tra le primitive della scena.
  - Ma per oggetti con molti triangoli resterebbe
- Quindi normalmente l'operazione viene fatta a livello in image space, cioé nella parte di pipeline che vedremo nella prossima lezione, cioè a livello dei pixel

HSR: Image-space

- E' quello che normalmente si fa nella pipeline grafica
- In pratica si tratta di ordinare per profondità le proiezioni delle primitive su ciascun pixel dell'immagine finale e tenere solo la proiezione che avviene da più vicino
- La rimozione delle superfici occluse è sempre un ordinamento per distanza lungo il proiettore

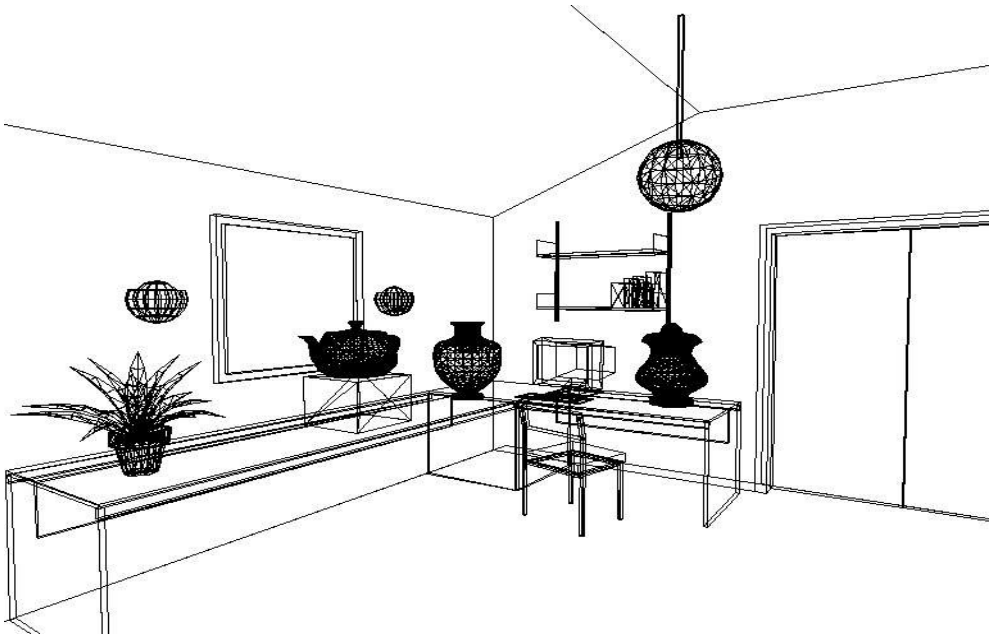


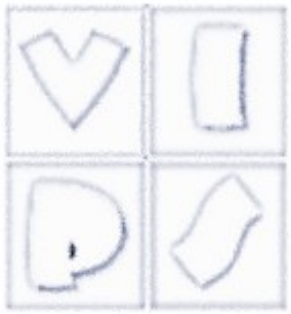


HSR: Image-space

- L'operazione fondamentale dell'approccio image-space è il calcolo delle intersezioni tra semirette e piani di appartenenza delle primitive (per ogni semiretta al più k intersezioni);
- Anche se per un display n x m, questa operazione deve essere eseguita n x m x k volte, la complessità risulta comunque di ordine O(k)
- Sia nell'approccio object-space che in quello image-space la rimozione delle superfici nascoste è riconducibile ad un problema di ordinamento (in profondità).

Con e senza rimozione superfici nascoste





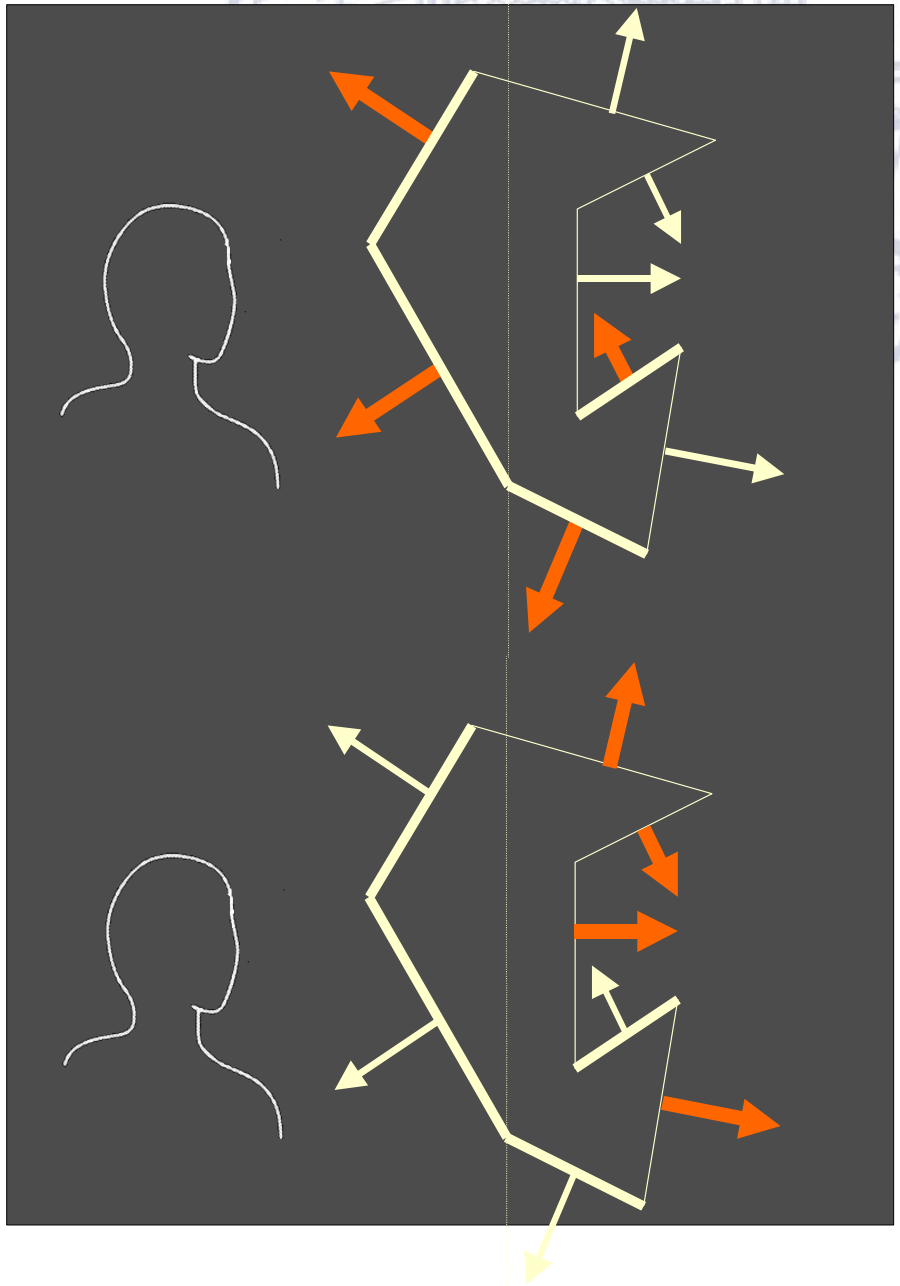
Back-face culling

- Alcune facce invisibili sono più facilmente rimosse
- Back-face culling (Eliminazione delle facce posteriori)
  - Se gli oggetti della scena sono rappresentati da poliedri solidi chiusi (cioè le facce poligonali dei poliedri delimitano completamente i volumi dei solidi);
  - Se ogni faccia poligonale è stata modellata in modo tale che la normale ad essa sia diretta verso l'esterno del poliedro di appartenenza;
  - Se nessuna parte del poliedro viene tagliata dal front clipping plane ...

Back-face culling

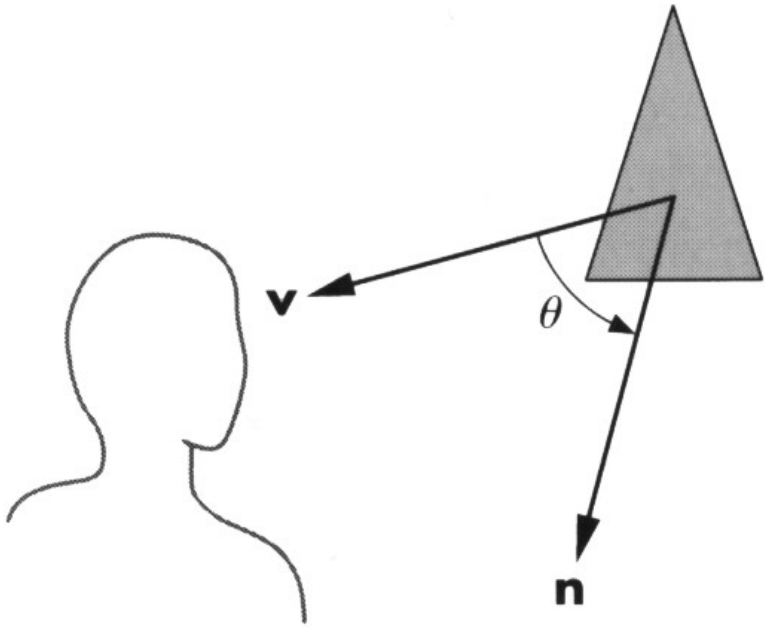
- Nelle ipotesi precedenti...
- Le facce la cui normale forma angoli superiori a ±90° con la direzione di vista possono essere visibili;

- Le facce la cui normale forma angoli inferiori a ±90° con la direzione di vista certamente non sono visibili;



Back-face culling

- Per ridurre il carico di lavoro richiesto per la rimozione delle superfici nascoste può essere quindi opportuno eliminare inizialmente tutti le primitive geometriche la cui normale è orientata verso il semispazio opposto all'osservatore, non visibile all'osservatore;
- Indicato con θ l'angolo tra la normale e l'osservatore, la primitiva in esame deve essere rimossa se $-90^\circ \leq \theta \leq 90^\circ$, cioè se $\cos\theta \geq 0$.
- Invece di calcolare la quantità $\cos\theta$ possiamo valutare il prodotto scalare $n \cdot v \geq 0$

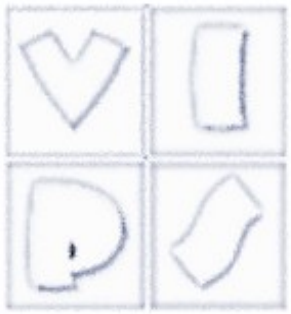


Back-face culling

- Se l'operazione è eseguita in coordinate normalizzate di vista (dopo aver eseguito la proiezione) la determinazione delle facce back-facing si riduce ad un controllo del segno della coordinata z delle normali: ad un segno positivo corrispondono facce front-facing, ad un segno negativo facce back-facing;
- Questo procedimento (detto back-face culling) consente, in media, di dimezzare il tempo necessario per il rendering di oggetti solidi dato che, sempre in media, circa metà delle facce di un poliedro sono back-facing e quindi la loro visualizzazione sarebbe comunque inutile.

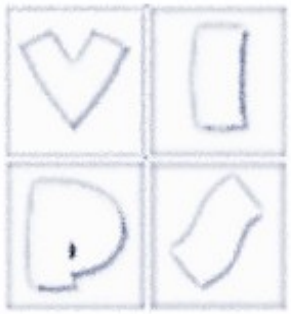
Riepilogo

- Spazio Oggetto (object space): dove ciascun singolo oggetto viene definito. Detto anche local space o modeling space
- Spazio Mondo (world space): dove si costruisce. Si passa dallo spazio oggetto allo spazio mondo mediante la trasformazione di modellazione. Si chiama anche application space.
- Spazio Vista (view space): centrato sulla telecamera virtuale, che definisce, assieme alla finestra di vista ed ai piani di taglio, il frustum di vista. Detto anche camera space o eye space
- Spazio schermo 3-D (3D-screen space): volume di vista canonico, che si ottiene trasformando il frustum di vista in un parallelepipedo. Molte operazioni del processo di rendering avvengono qui. Detto anche 3D normalized device coordinate system (NDC) o normalized projection coordinate system.
- Spazio Immagine (image space): 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.



A questo punto

- Abbiamo mappato i triangoli sul piano immagine nella regione del “sensore” della telecamera virtuale
- Abbiamo eliminato le parti esterne al volume di vista (e in parte eliminato le parti occluse)
- Ora dobbiamo sovrapporre i triangoli ai pixel dell'immagine discretizzata e colorare i pixel corrispondenti sulla base di modello di illuminazione e caratteristiche della geometria
- Questo è compito della seconda parte della pipeline cioè del cosiddettto “raster” subsystem



Riferimenti

- Angel Cap. 6
- Scateni cap. 5