

INFORMATICA GRAFICA – SSD ING-INF/05

Sistemi di elaborazione delle informazioni

a.a. 2007/2008

CAP 6. Rendering grafico

Texture mapping

Texture mapping

- ❖ Shading: funziona finché modello geometrico ha complessità della composizione del materiale
- ❖ La descrizione di un materiale non uniforme è corretta a partire da una rappresentazione geometrica con suddivisione (*tassellazione*) in primitive collegata alle discontinuità del materiale da rappresentare
- ❖ Altrimenti? Texture mapping!

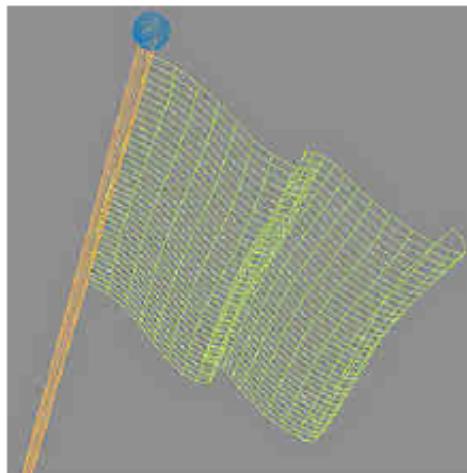
Texture Mapping

- ❖ Nelle operazioni per frammento si può accedere ad una RAM apposita: la **Texture RAM** strutturata in un insieme di **Textures** (“**tessiture**”)
- ❖ Ogni tessitura è un array 1D, 2D o 3D di **Texels** (campioni di tessitura) dello stesso tipo

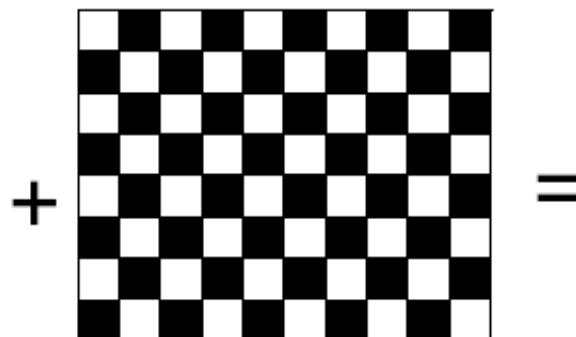
Texels

- ❖ Sono esempi di texels:
 - ❖ Ogni texel un colore (componenti: R-G-B, o R-G-B-A): la tessitura è una “color-map”
 - ❖ Ogni texel una componente alpha: la tessitura è una “alpha-map”
 - ❖ Ogni texel una normale (componenti: X-Y-Z): la tessitura è una “normal-map” o “bump-map”
 - ❖ Ogni texel contiene un valore di specularità: la tessitura è una “shininess-map”

Rimappare immagini sulla geometria



geometria 3D
(mesh di quadrilateri)



RGB texture 2D
(color-map)



Rimappare immagini sulla geometria



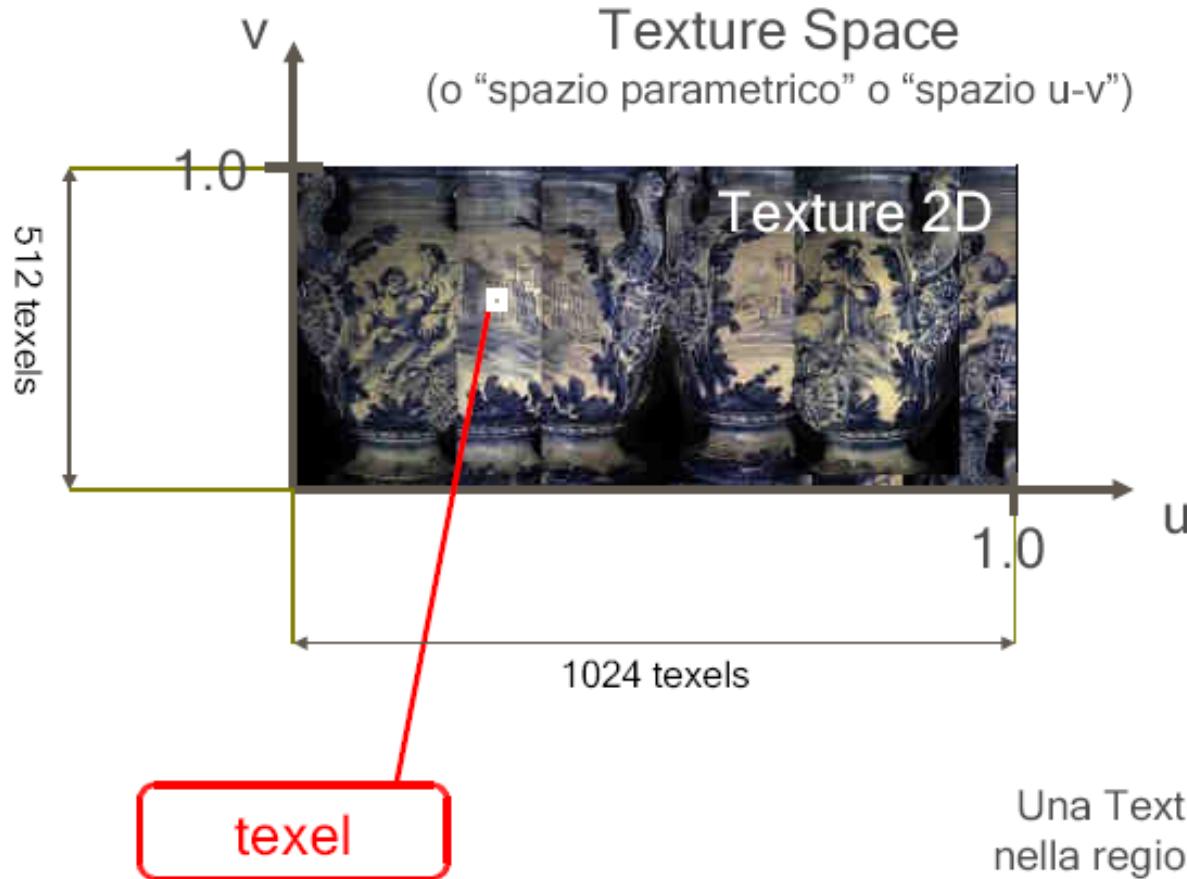
Texture Mapping: storia

- ❖ 1974 introdotto da Ed Catmull
 - ❖ nella sua Phd Thesis
- ❖ Solo nel 1992 (!) si ha texture mapping in hardware
 - ❖ Silicon Graphics RealityEngine
- ❖ Dal 92 a oggi ha avuto aumento rapidissimo della diffusione
 - ❖ strada intrapresa soprattutto da low end graphic boards
- ❖ Oggi è una delle più fondamentali tecniche di rendering
 - ❖ Il re indiscusso delle tecniche image based



Ed Catmull

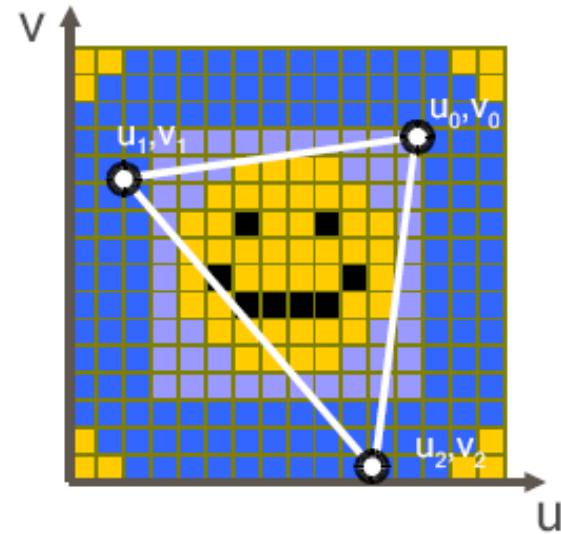
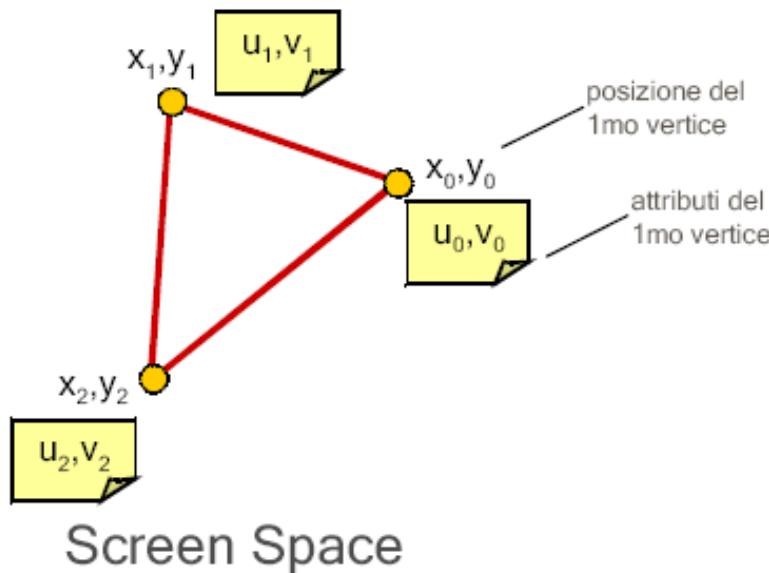
Notazione



Una Texture è definita
nella regione $[0,1] \times [0,1]$
dello “spazio parametrico”

Texture Mapping

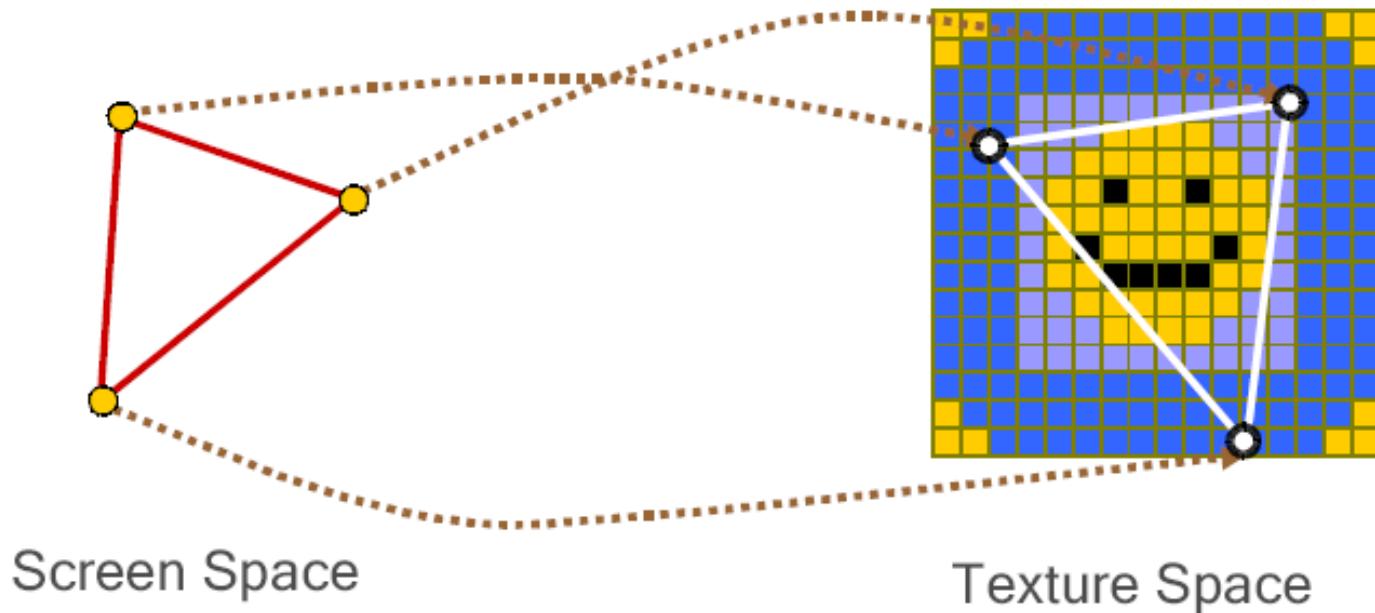
- ❖ Ad ogni **vertice** (di ogni triangolo) assegno le sue coordinate u, v nello **spazio tessitura**



Texture Space

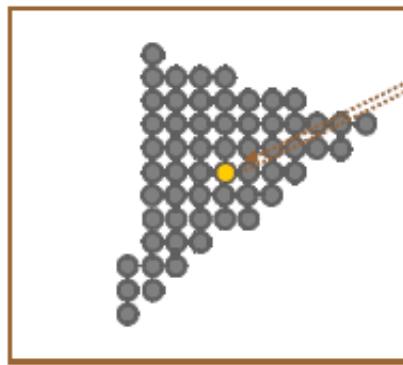
Texture Mapping

- ❖ Così in pratica definisco un **mapping** fra il triangolo e un triangolo di tessitura



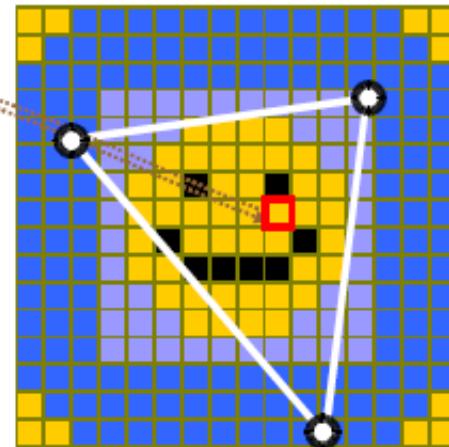
Texture Mapping

- ❖ Ogni vertice (di ogni triangolo) ha le sue coordinate u, v nello **spazio tessitura**



Screen Space

texture look-up

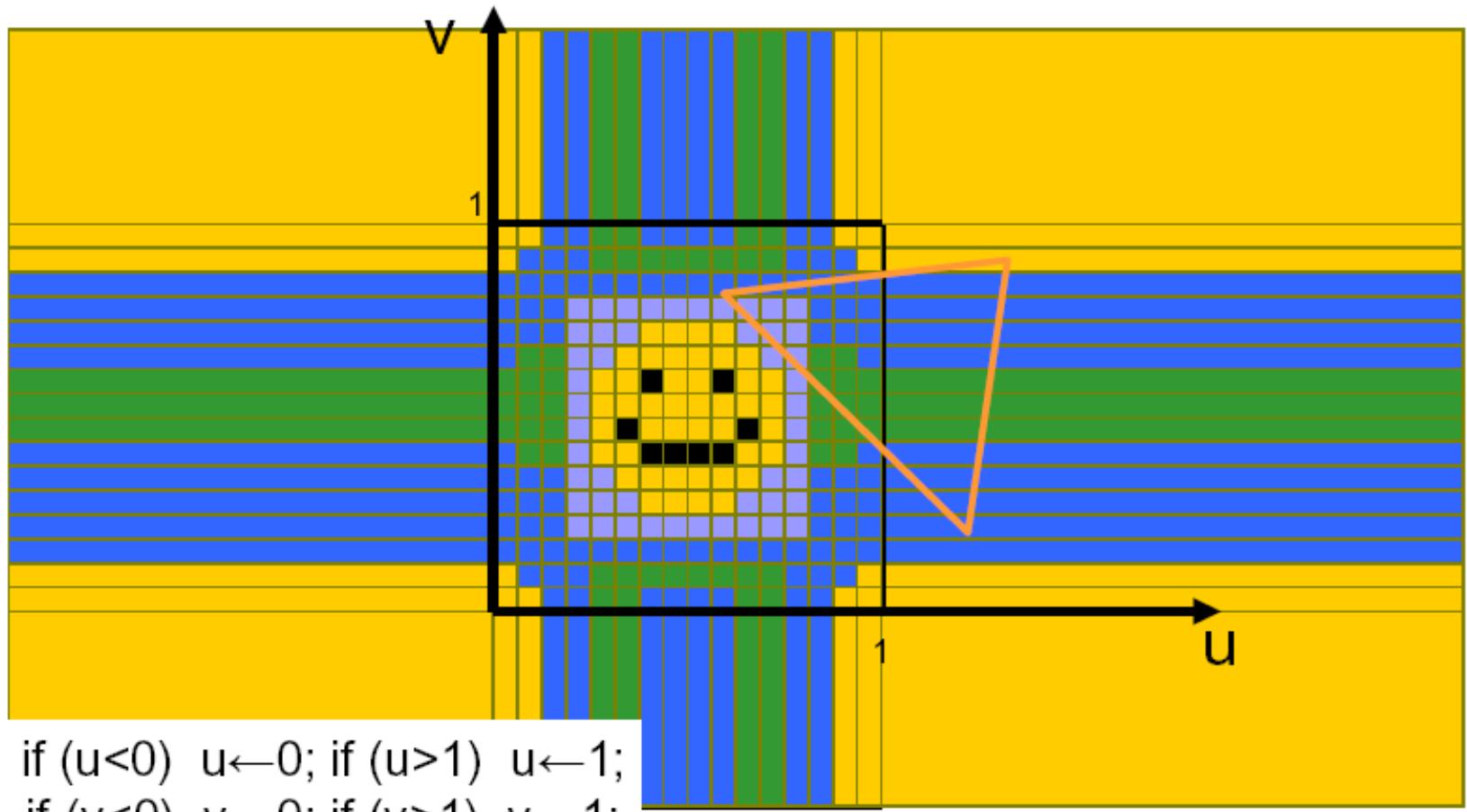


Texture Space

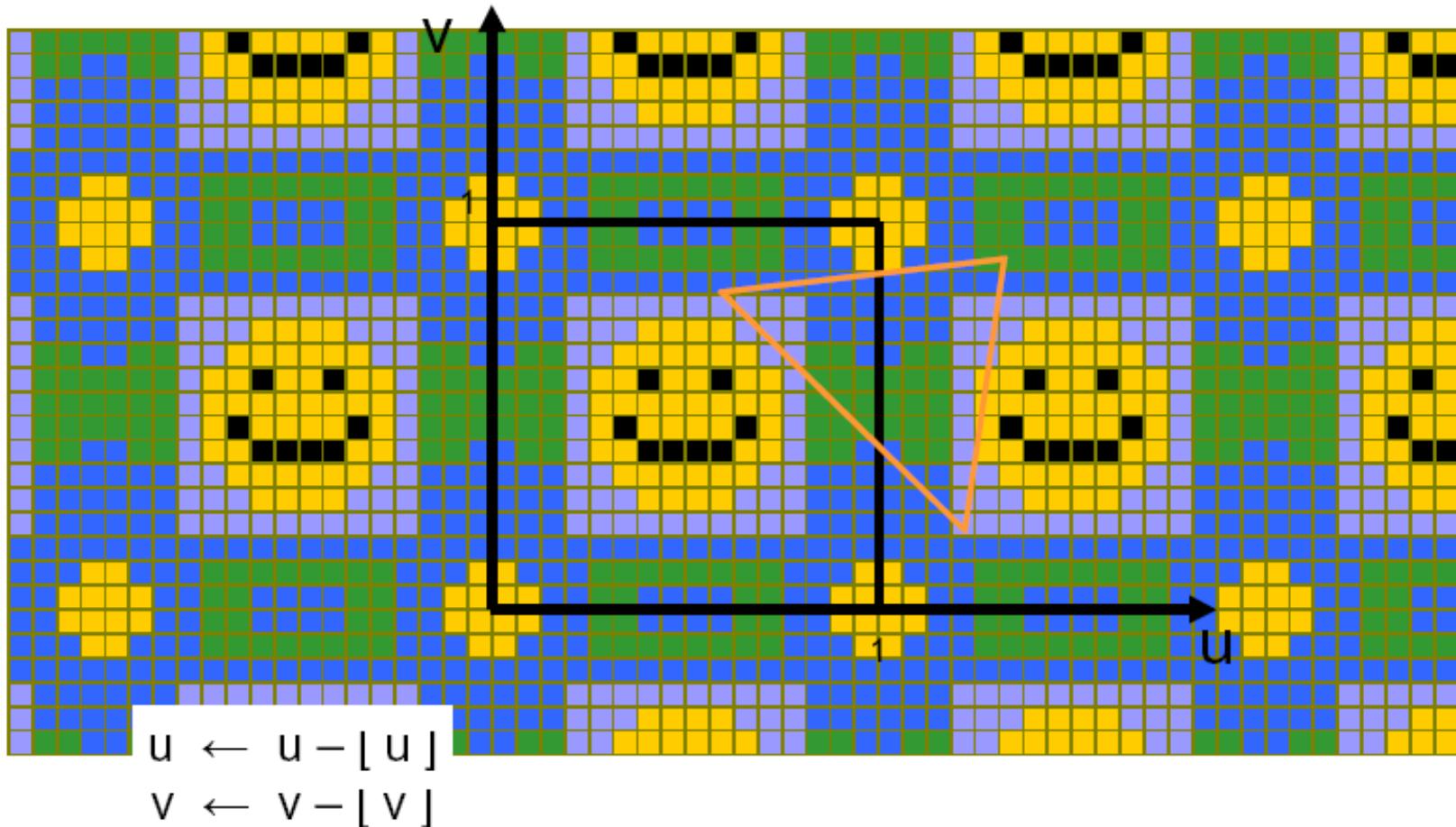
Nota: la tessitura va caricata

- ❖ Da disco a memoria RAM main (sulla scheda madre)
- ❖ Da memoria RAM main a Texture RAM (on board dell'HW grafico)
- ❖ Entrambe le operazioni sono piuttosto lente e sono impossibili da fare una volta per frame quindi nel progetto dell'applicazione si devono utilizzare strategie per la gestione delle texture

Texture fuori dai bordi: modo *clamp*



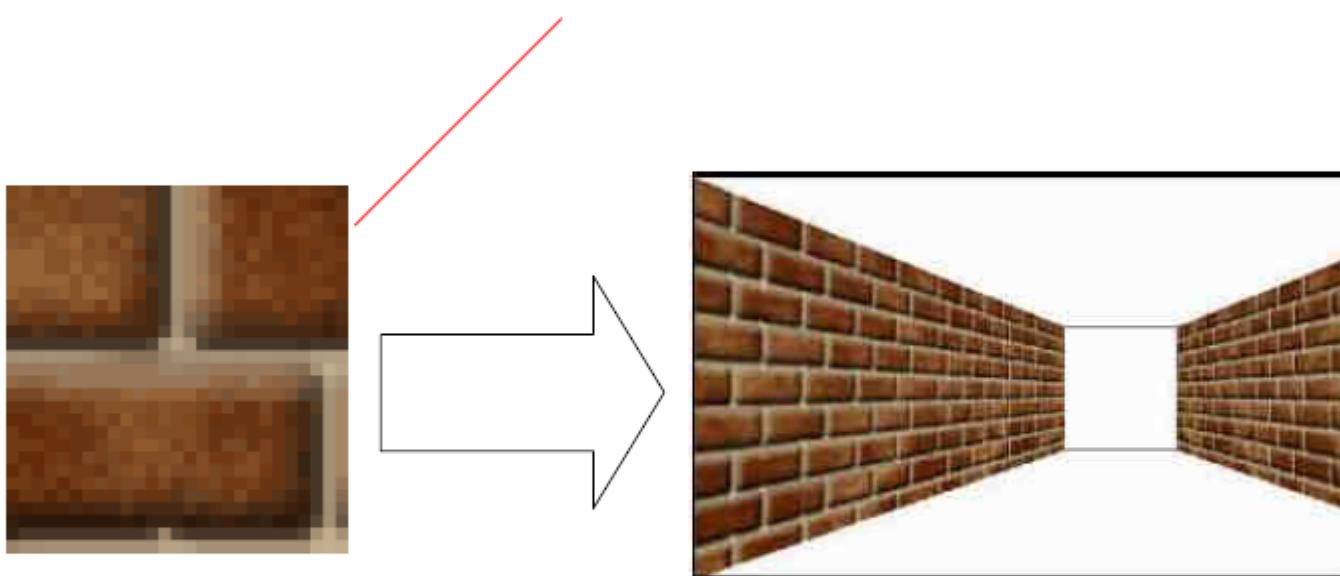
Texture fuori dai bordi: modo *repeat*



Tessiture ripetute

- ❖ Tipico utilizzo:

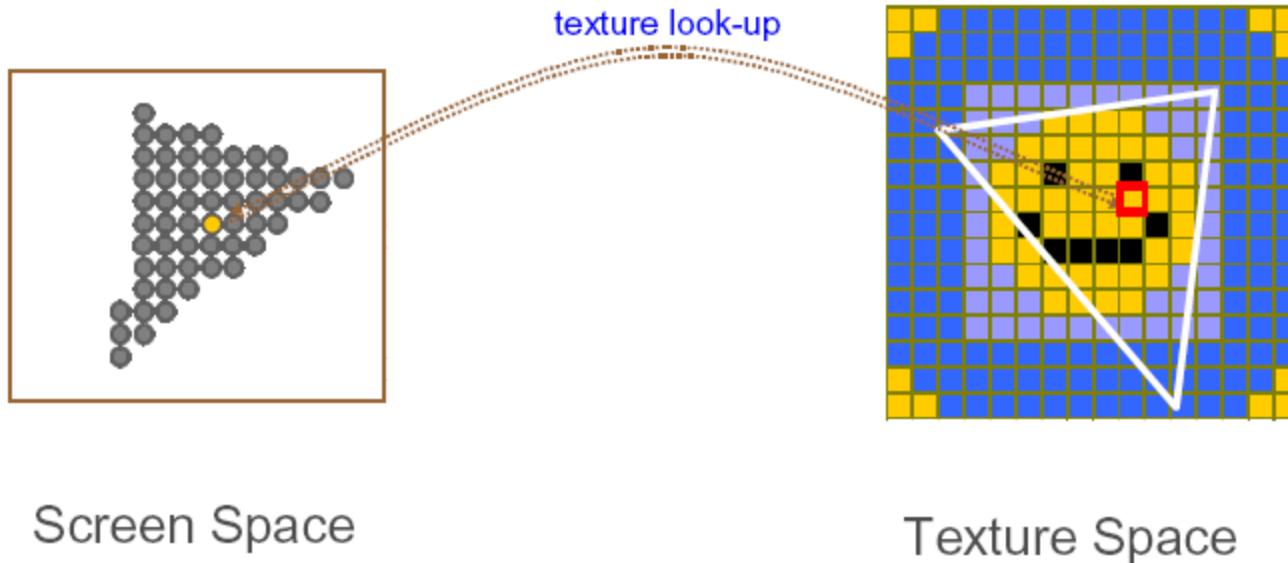
Nota: deve essere **TILABLE**



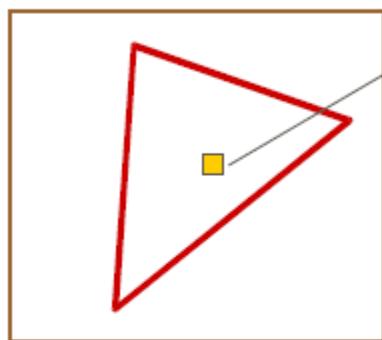
Molto efficiente in spazio: una sola texture mappa su molti triangoli

Texture Look-up

- ❖ Un frammento ha coordinate non intere (in texels)

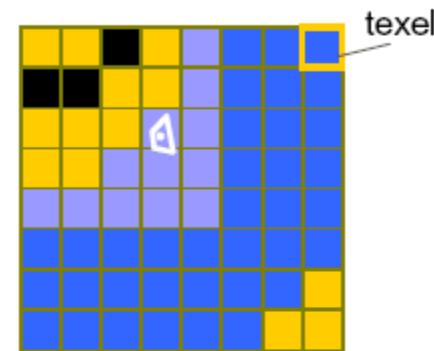


Texture Look-up

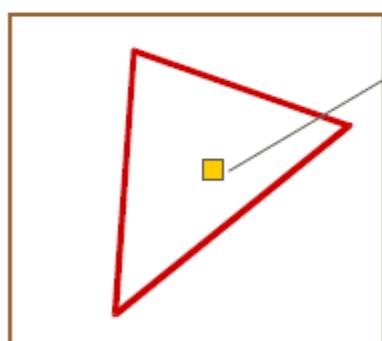


pixel
un pixel = meno di un texel
magnification

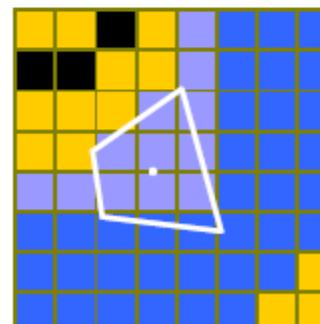
Screen Space



Texture Space



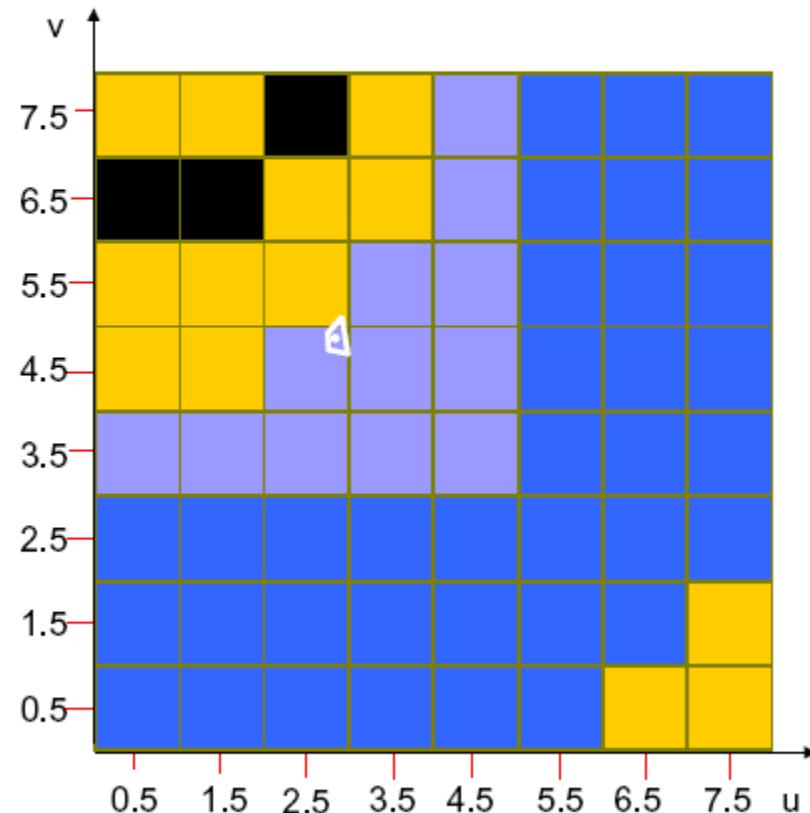
pixel
un pixel = più di un texel
minification



Texture maps are square or rectangular, but after being mapped to a polygon or surface and transformed into screen coordinates, the individual texels of a texture rarely correspond to individual pixels of the final screen image.

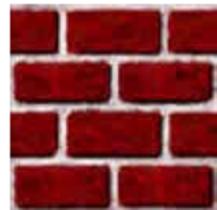
Caso Magnification

Soluzione 1:
prendo il texel in cui sono
(equivale a prendere
il texel più vicino)
equivale ad arrotondare
alle coordinate texel
interne
"Nearest Filtering"



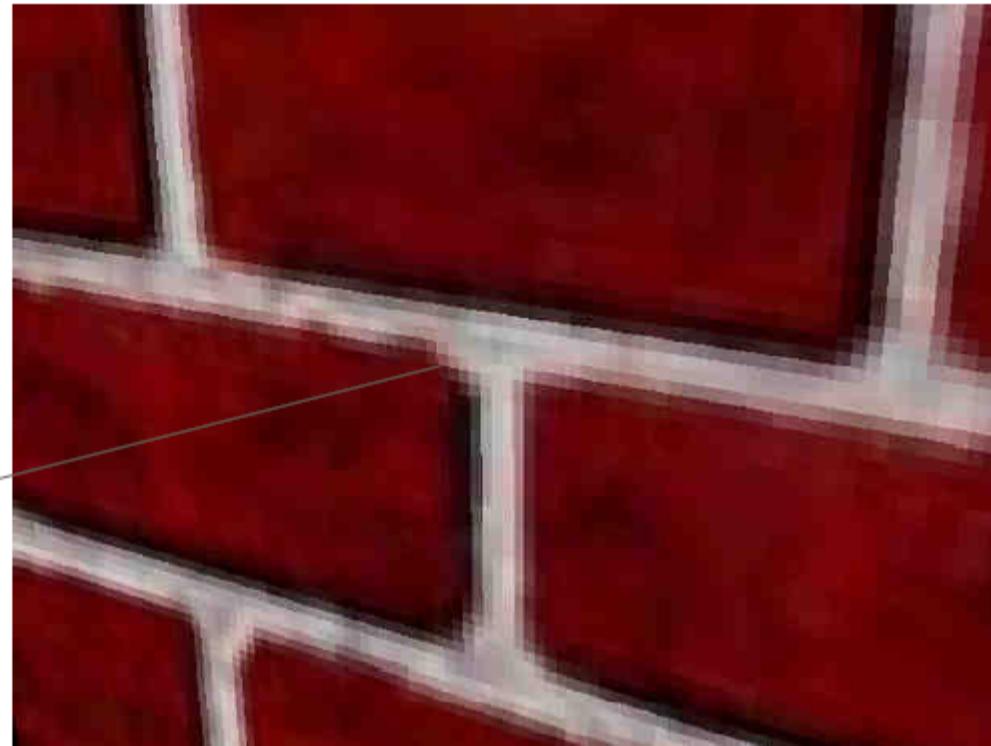
Caso Magnification

Nearest Filtering: risultato visivo



texture 128x128

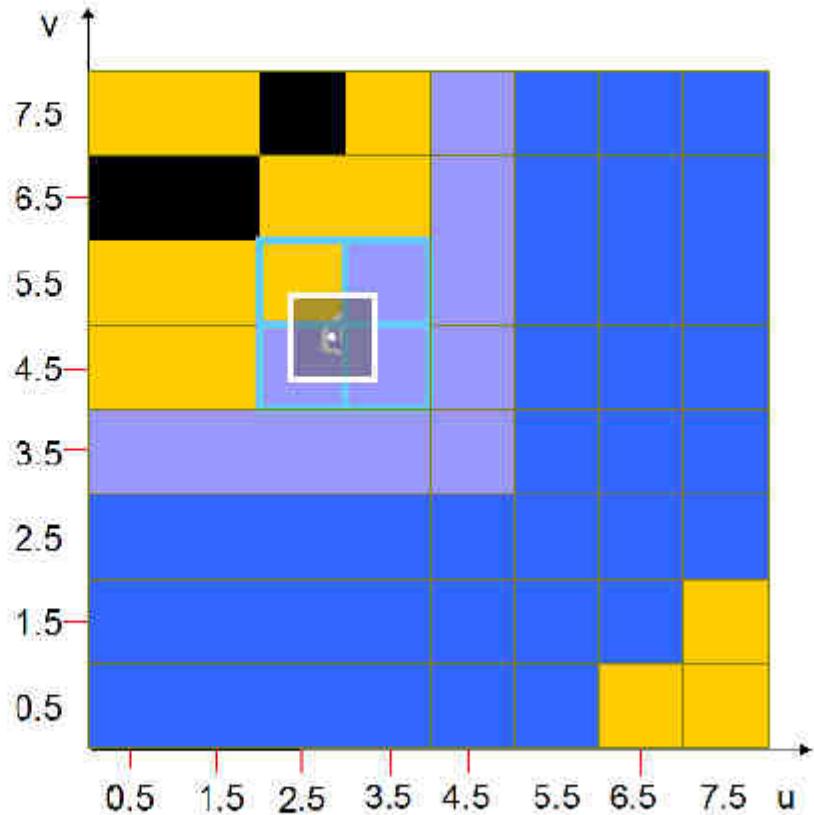
"si vedono i texel!"



Caso Magnification

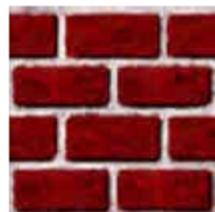
Soluzione 2:
Medio il valore dei quattro texel
più vicini

Interpolazione Bilineare

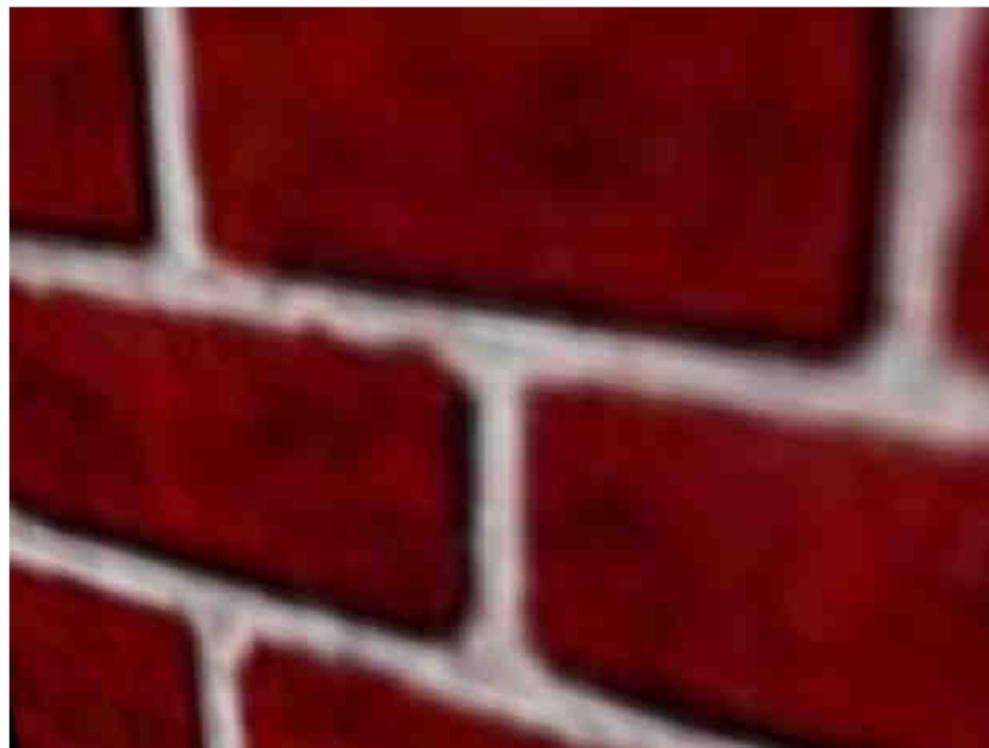


Caso Magnification

Bilinear Interpolation: risultato visivo



texture 128x128

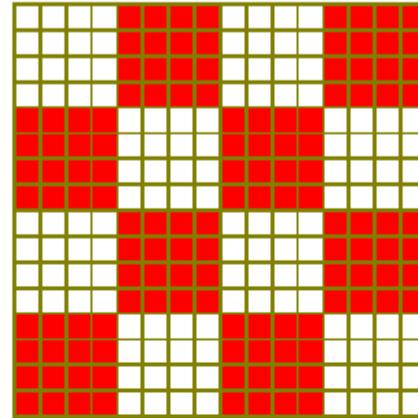


Caso Magnification

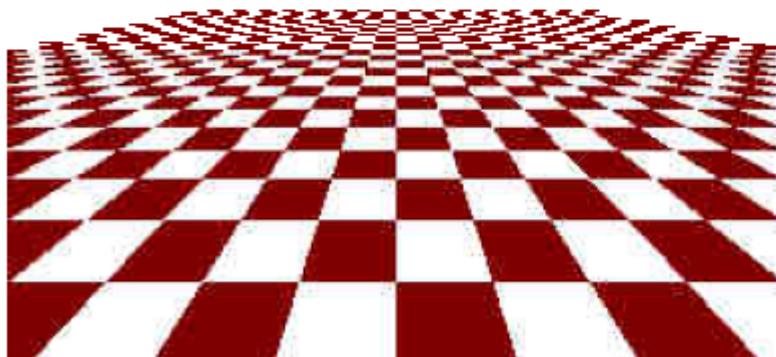
- ❖ Modo Nearest:
 - ❖ si vedono i texel
 - ❖ va bene se i bordi fra i texel sono utili
 - ❖ più veloce

- ❖ Modo Interpolazione Bilineare
 - ❖ di solito qualità migliore
 - ❖ può essere più lento
 - ❖ rischia di avere un effetto "sfuocato"

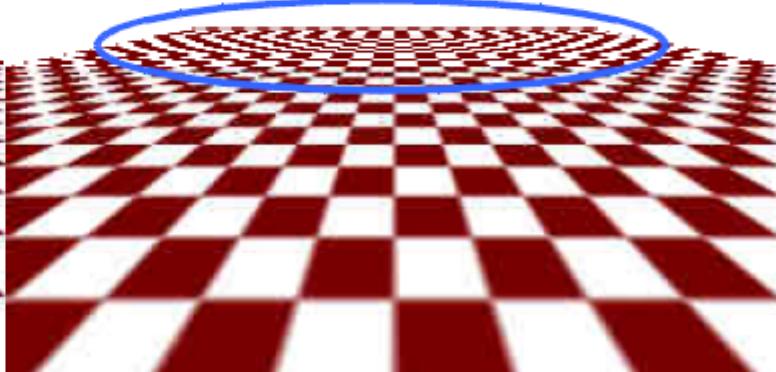
Caso Minification



Nearest Filtering



Bilinear interpolation
non risolve il problema

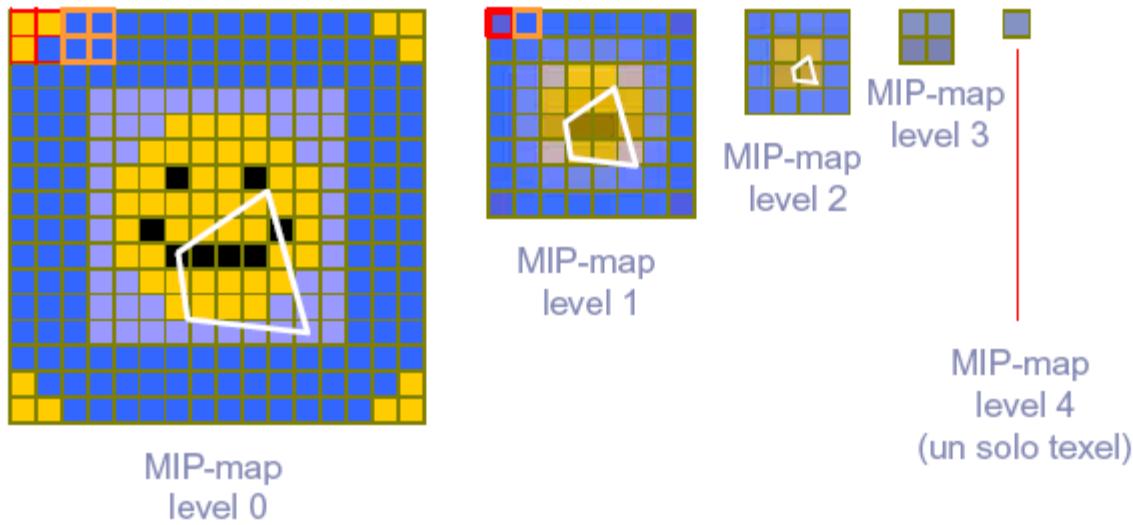


Minification - MIP Mapping

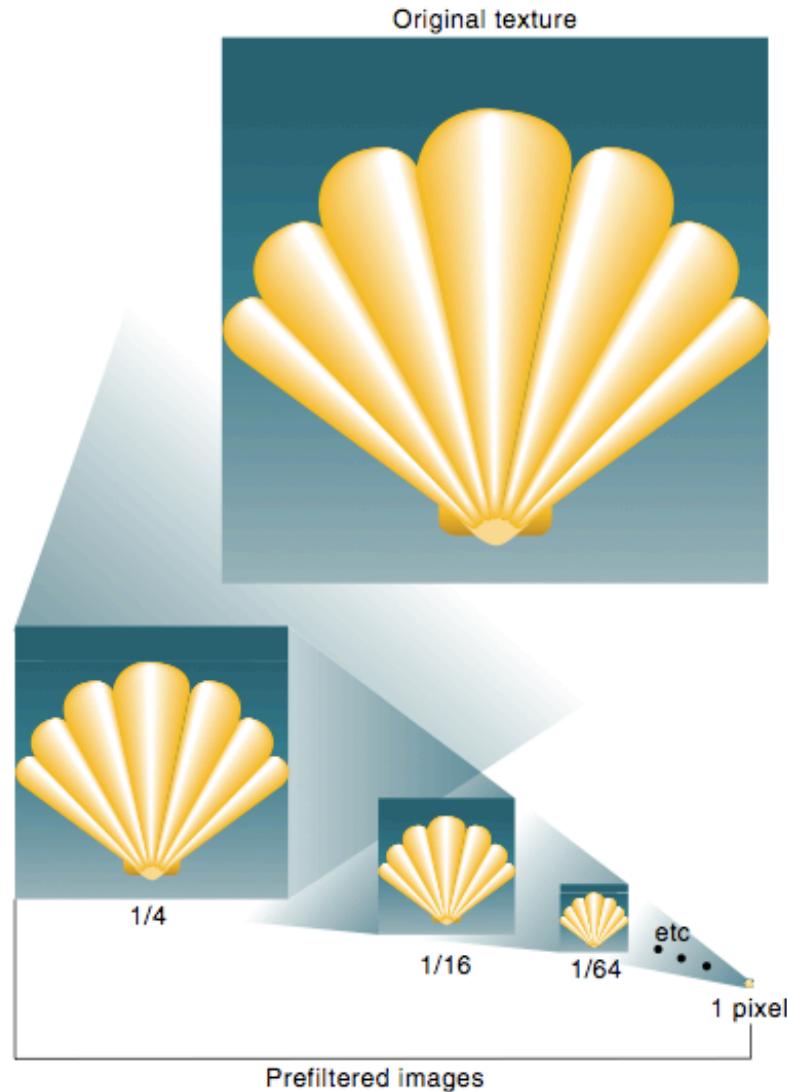
- Textured objects can be viewed, like any other objects in a scene, **at different distances** from the viewpoint.
- In a dynamic scene, as a textured object moves farther from the viewpoint, **the texture map must decrease in size** along with the size of the projected image.
- To accomplish this, OpenGL has to filter the texture **map down to an appropriate size** for mapping onto the object, without introducing visually disturbing artifacts
- When using mipmapping, OpenGL has to determine **which texture map to use** based on the size (in pixels) of the object being mapped.
- With this approach, the level of detail in the **texture map is appropriate for the image** that's drawn on the screen—as the image of the object gets smaller, the size of the texture map decreases.
- Mipmapping uses some **clever methods to pack image data** into memory.

Caso Minification: MIP-mapping

MIP-mapping: "Multum In Parvo"



Minification - MIP Mapping



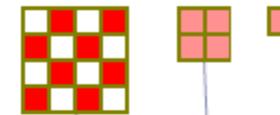
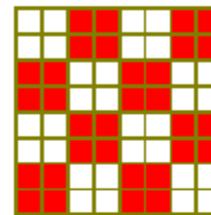
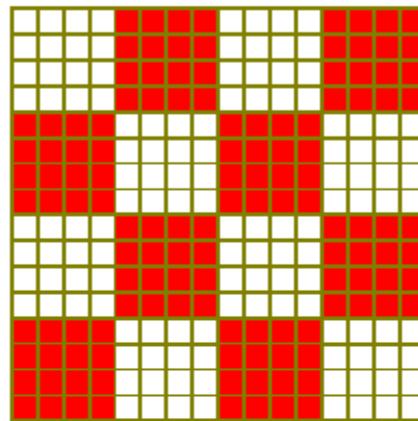
Mipmap Math

- ❖ Definiamo un **fattore di scala**, $\rho = \text{texels/pixel}$ come valore massimo fra ρ_x e ρ_y , che può variare sullo stesso triangolo, può essere derivato dalle matrici di trasformazione ed è calcolato nei **vertici**, interpolato nei **frammenti**
- ❖ Il **livello di mipmap** da utilizzare è: $\log_2 \rho$ dove il livello 0 indica la massima risoluzione
- ❖ Il livello non è necessariamente un numero intero e può quindi essere arrotondato

★ To use mipmapping, you must provide all sizes of your texture in powers of 2 between the largest size and a 1×1 map.

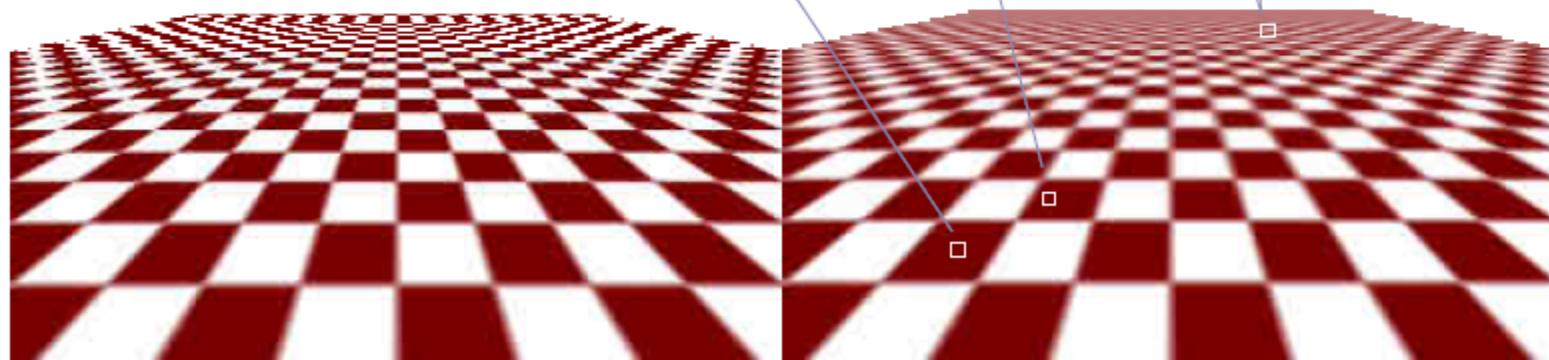
★ For example, if your highest-resolution map is 64×16 , you must also provide maps of size 32×8 , 16×4 , 8×2 , 4×1 , 2×1 , and 1×1 .

Caso Minification: MIP-mapping



Bilinear interpolation
non risolve il problema

MIP-mapping

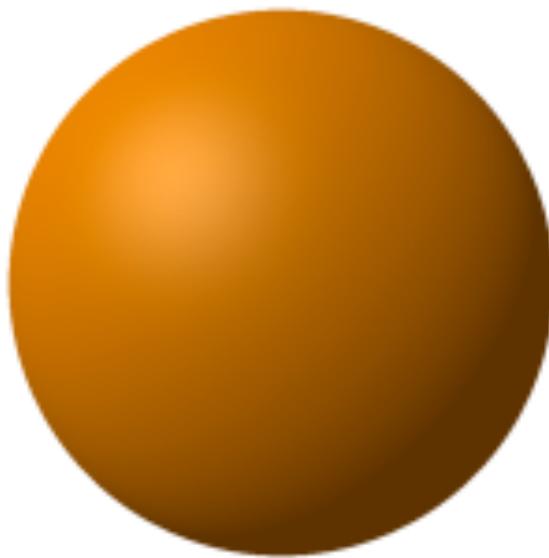


Bump Mapping

While Texture Mapping added colour to a polygon, **Bump Mapping** adds what appears to be *surface roughness*.

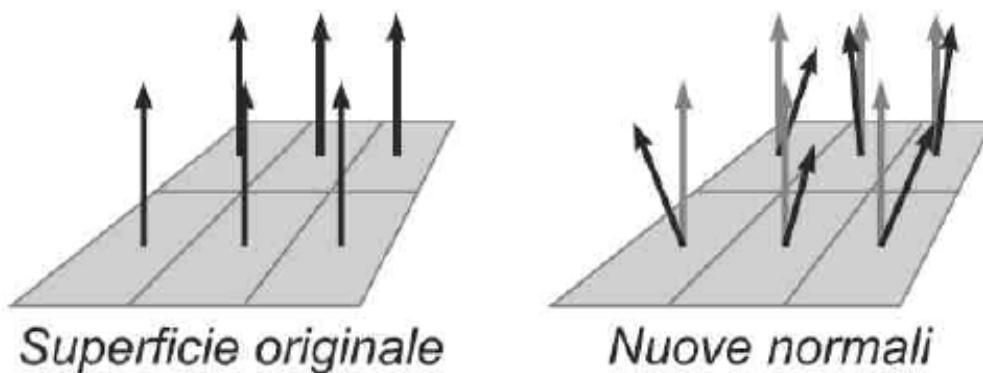
Bump Mapping can have a dramatic effect on the look of a polygonal object, adding a minute detail to an object which would otherwise require a large number of polygons.

Note that the **polygon is still physically flat**, but **appears** to be bumpy.



Bump mapping

- ❖ L'effetto che si ottiene è un perturbazione del valore delle normali che altera il rendering senza modificare la geometria



Rimozione linee nascoste

Hidden Surface Removing

Esempio PLaSM su oggetti convessi

```
DEF Jewel (arg::IsPol) = arg1 & arg2 & arg3
```

WHERE

```
arg1 = R:<2,3>:(PI/4):arg,
```

```
arg2 = R:<3,1>:(PI/4):arg,
```

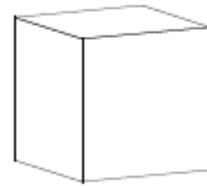
```
arg3 = R:<1,2>:(PI/4):arg
```

END;

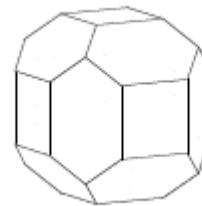
```
DEF convex1 = T:<1,2,3>:<-0.5,-0.5,-0.5>:(CUBOID:<1,1,1>);
```

```
DEF convex2 = Jewel:convex1;
```

```
DEF convex3 = Jewel:convex2;
```



convex1



convex2



convex3

Rimozione delle superfici nascoste

- ❖ Il problema della **rimozione delle superfici nascoste** consiste nel determinare se un oggetto è visibile all'osservatore, oppure rimane oscurato da altri oggetti della scena
- ❖ Non è quindi un problema legato solo alla disposizione degli oggetti nella scena, ma alla relazione che esiste tra oggetti e posizione dell'osservatore

Rimozione delle superfici nascoste

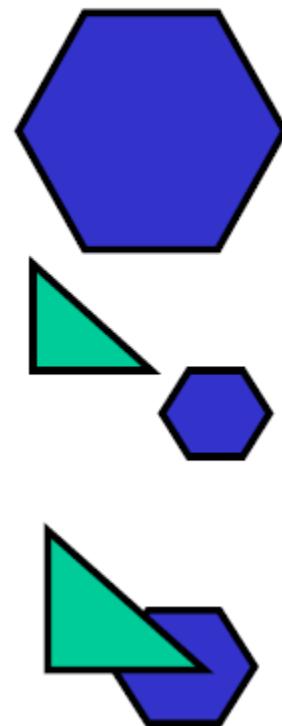
- ❖ Gli algoritmi per la rimozione delle superfici nascoste si possono dividere in due classi:
 - ❖ gli algoritmi **object-space** determinano, **per ogni oggetto**, quali parti dell'oggetto non sono oscurate da altri oggetti nella scena
 - ❖ gli algoritmi **image-space** determinano, **per ogni pixel**, quale è l'oggetto più vicino all'osservatore

Object-space

- ❖ Data una scena tridimensionale composta da n poligoni piatti ed opachi, si può derivare un generico algoritmo di tipo object-space considerando gli oggetti a coppie

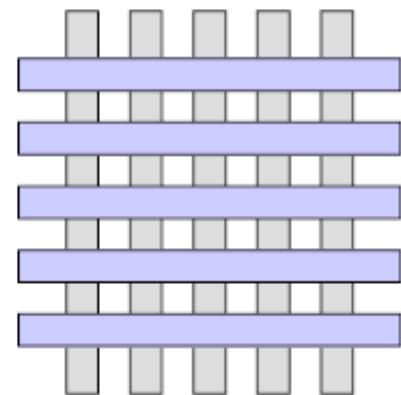
Object-space

- ❖ Data una coppia di poligoni, ad esempio A e B, ci sono quattro casi da considerare:
 - ❖ A oscura B: visualizzeremo solo A
 - ❖ B oscura A: visualizzeremo solo B
 - ❖ A e B sono completamente visibili: visualizzeremo sia A che B
 - ❖ A e B si oscurano parzialmente l'uno l'altro: dobbiamo calcolare le parti visibili di ciascun poligono



Object-space

- ❖ Si prende uno dei n poligoni e lo si confronta con tutti i restanti $n - 1$
- ❖ In questo modo si determina quale parte del poligono sarà visibile
- ❖ Questo processo è ripetuto con gli altri poligoni
- ❖ La complessità di questo approccio risulta di ordine $O(n^2)$
- ❖ L'approccio object-space è consigliabile solo quando gli oggetti nella scena sono pochi



L'algoritmo *depth sort*

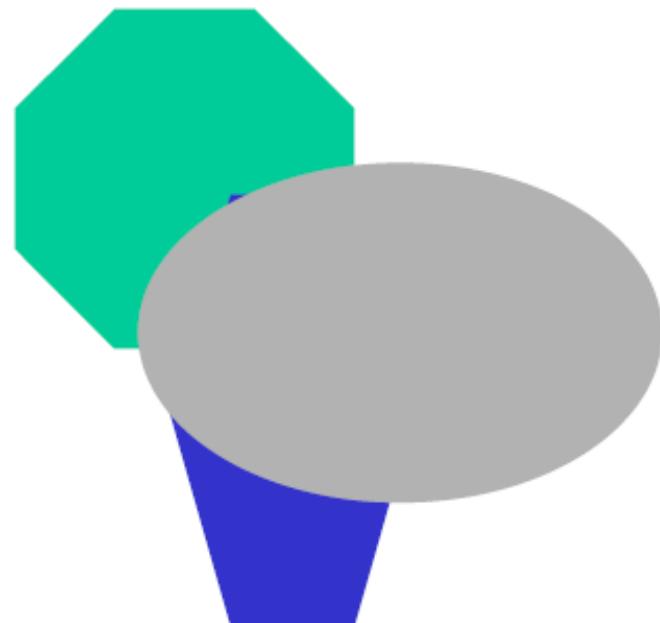
- ❖ Consideriamo una scena composta da poligoni planari
- ❖ L'algoritmo depth sort è una variante di un algoritmo ancora più semplice, chiamato **algoritmo del pittore**
- ❖ Supponiamo che i poligoni siano ordinati sulla base della loro distanza dall'osservatore

L'algoritmo *depth sort*

- ❖ Per rappresentare correttamente la scena, potremmo individuare la parte visibile del poligono più distante, e predisporla nel frame buffer
- ❖ Se i poligoni sono solo due, questa operazione richiede l'esecuzione del clipping di un poligono rispetto all'altro

L'algoritmo *depth sort*

- ❖ Un'altra possibilità è invece quella di seguire un approccio analogo a quello usato da un pittore:
- ❖ Dipingere prima il poligono più lontano interamente, e poi dipingere il poligono più vicino, dipingendo sopra le parti del poligono più lontano non visibili all'osservatore

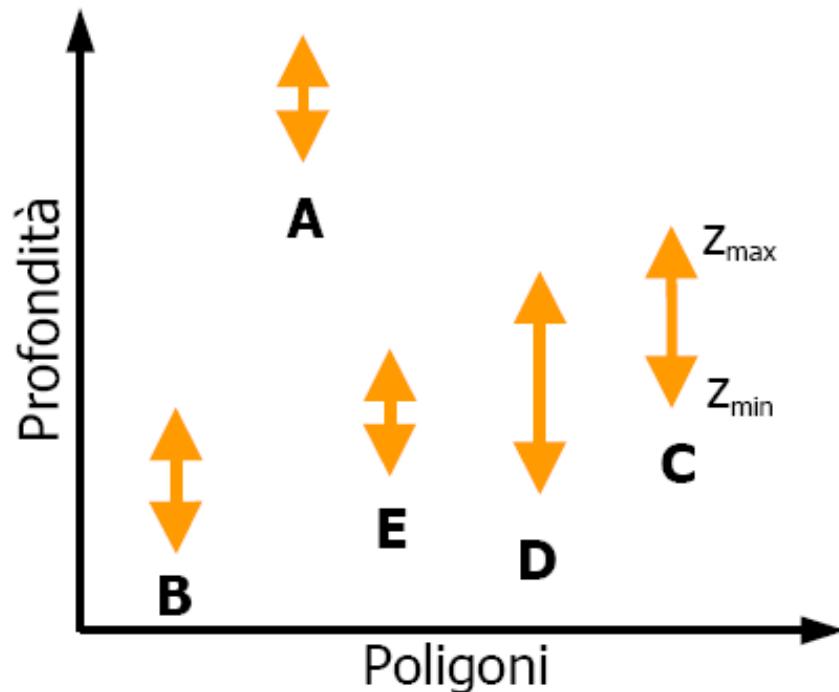


L'algoritmo *depth sort*

- ❖ I problemi da risolvere per implementare questo approccio riguardano
 - ❖ l'ordinamento in profondità dei poligoni
 - ❖ la situazione di sovrapposizione tra poligoni

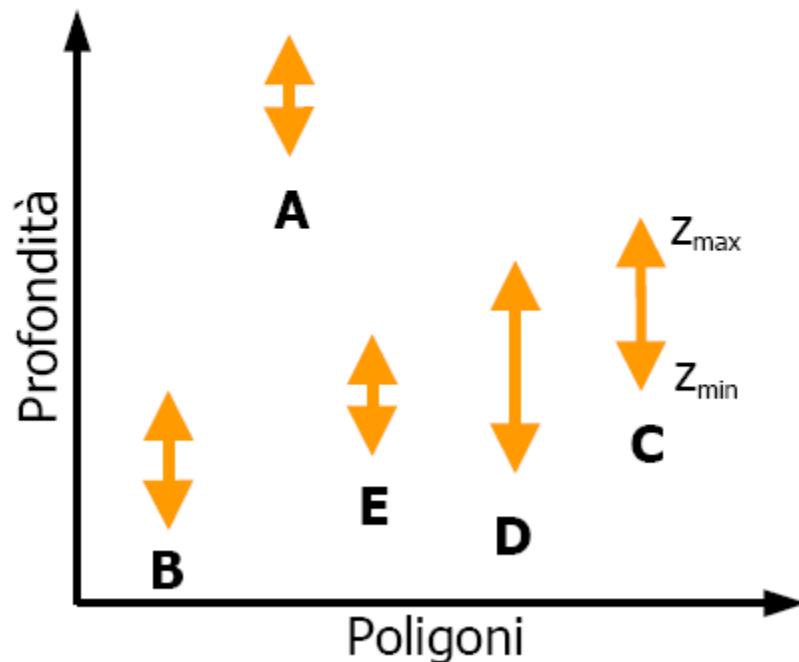
L'algoritmo *depth sort*

- ❖ Devo ordinare i poligoni in base alla distanza della loro coordinata z massima dall'osservatore
- ❖ Più precisamente, si considera l'estensione nella direzione z di ogni poligono



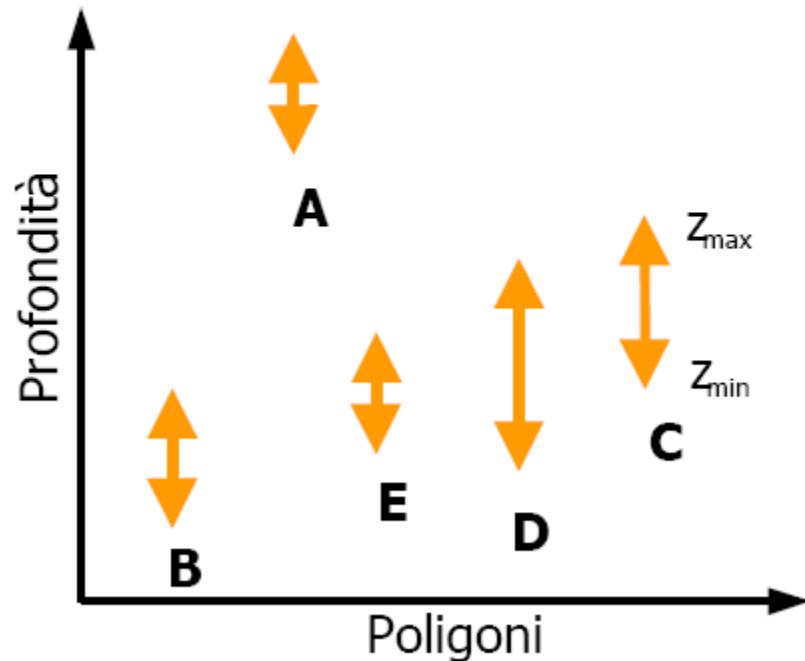
L'algoritmo *depth sort*

- ❖ Se la profondità minima di ogni poligono è maggiore della profondità massima del poligono situato sul retro, possiamo visualizzare i poligoni partendo da quello più in profondità



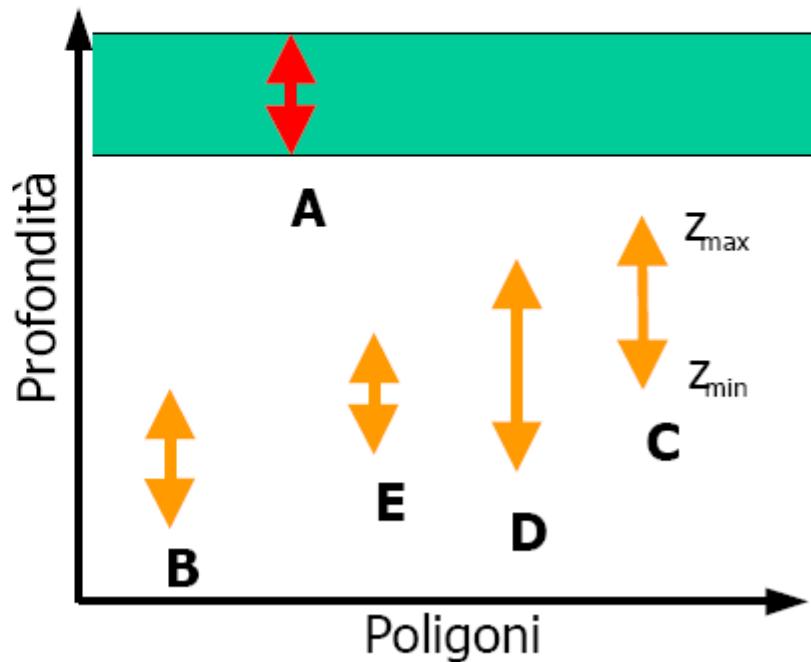
L'algoritmo *depth sort*

- ❖ E' il caso del poligono A, che è situato dietro a tutti gli altri poligoni e può essere visualizzato per primo



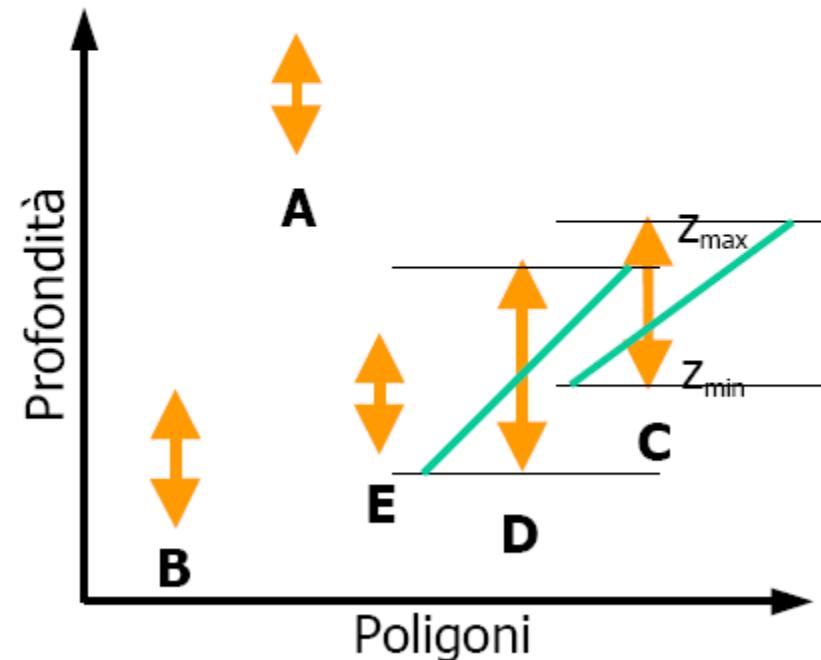
L'algoritmo *depth sort*

- ❖ E' il caso del poligono A, che è situato dietro a tutti gli altri poligoni e può essere visualizzato per primo



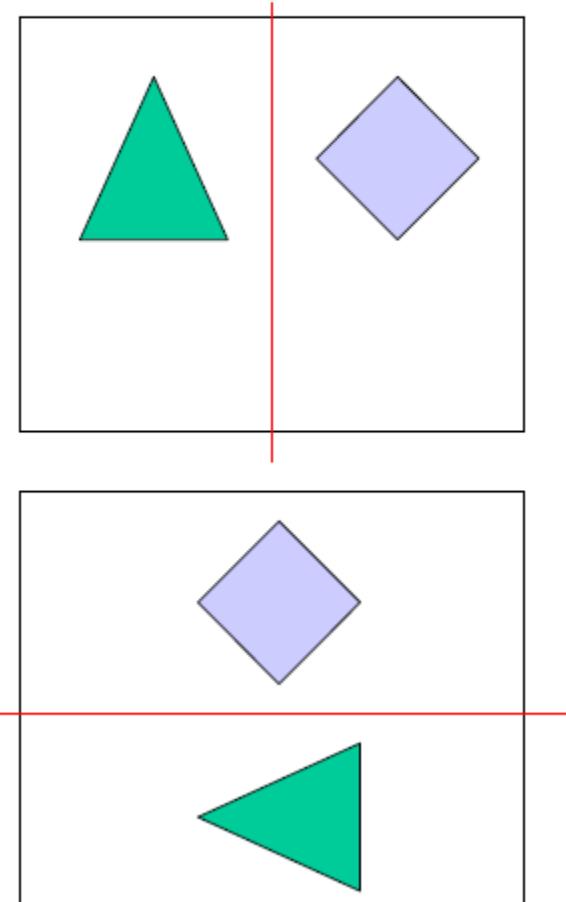
L'algoritmo *depth sort*

- ❖ Gli altri poligoni, tuttavia, non possono essere visualizzati basandosi solo sulla loro estensione lungo z
- ❖ Se le estensioni z di due poligoni si sovrappongono, dobbiamo determinare un ordine per visualizzarli individualmente che permetta di ottenere l'immagine corretta



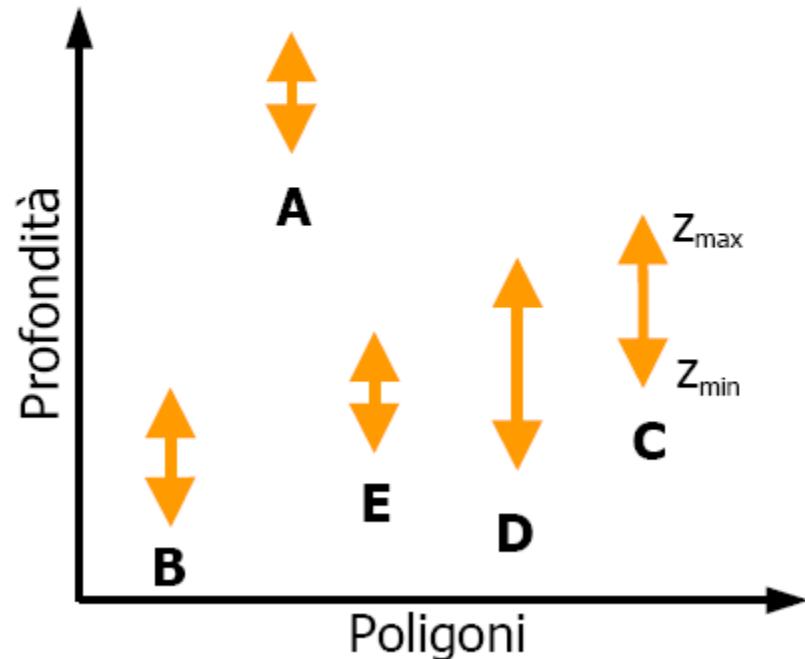
L'algoritmo *depth sort*

- ❖ Il test più semplice consiste nel controllare le estensioni lungo x e lungo y
- ❖ Se non c'è sovrapposizione in almeno una delle due direzioni, allora sicuramente nessuno dei due poligoni può oscurare l'altro, ed essi possono essere visualizzati in un ordine qualsiasi



L'algoritmo *depth sort*

- ❖ Se anche questo test fallisce, può essere ancora possibile trovare un ordine corretto per visualizzare i poligoni, ad esempio se tutti i vertici di un poligono cadono dalla stessa parte del piano determinato dall'altro poligono



L'algoritmo *depth sort*

- ❖ Rimangono da considerare due situazioni problematiche, per cui non esiste un ordine corretto di rappresentazione

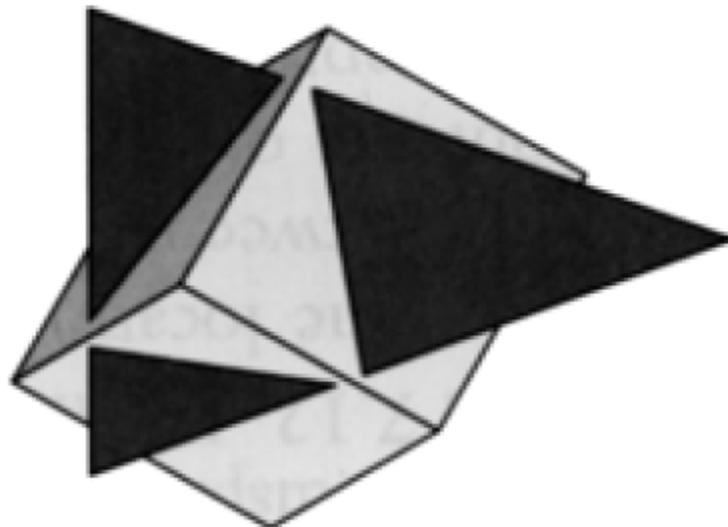
L'algoritmo *depth sort*

- ❖ La prima si verifica quando tre o più poligoni si sovrappongono ciclicamente



L'algoritmo *depth sort*

- ❖ La seconda situazione si verifica invece quando un poligono penetra nell'altro



L'algoritmo *depth sort*

- ❖ In entrambi i casi, è necessario spezzare i poligoni in corrispondenza dei segmenti di intersezione, e cercare l'ordine corretto di rappresentazione del nuovo insieme di poligoni

Formulazione rigorosa: coerenza della scena

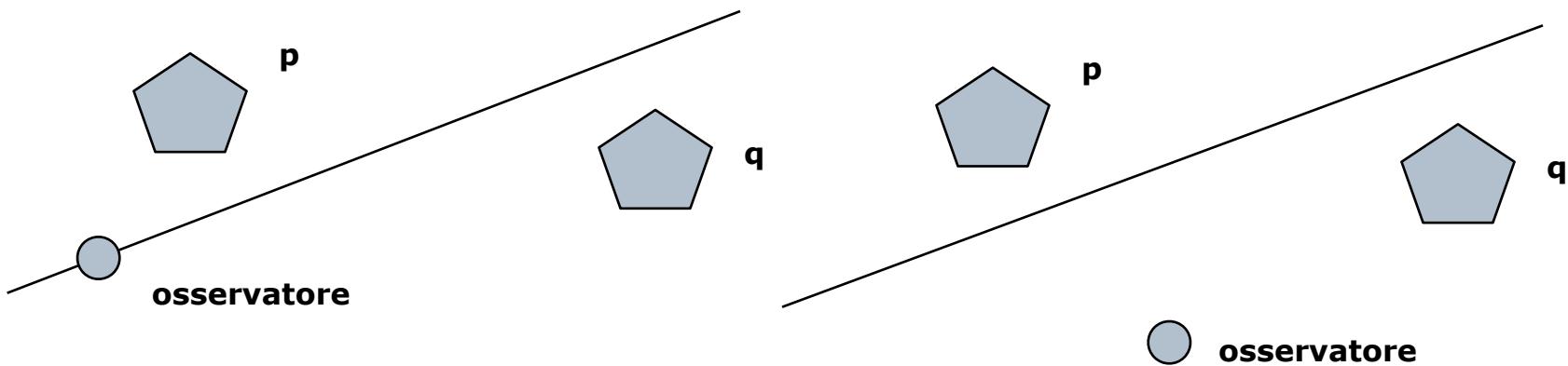
Consideriamo una scena con due soli poligoni 3D.

Quali sono le condizioni affinchè l'osservatore possa:

- vedere entrambi i poligoni
- qualcuno di essi possa coprire l'altro (o una sua parte) rendendolo non visibile all'osservatore
- Evidenziare le **condizioni formali** che consentano di scomporre un problema HSR in sottoproblemi che possano essere risolti indipendentemente.

Formulazione rigorosa: Relazione di visibilità

- Data una coppia di poligoni 3D, qualunque **piano h** che **lasci i due poligoni in semispazi differenti** prende il nome di **piano di separazione**
- La notazione **$p \leq q$** , dove p e q sono poligoni, significa che " **p non può coprire q** " rispetto all'osservatore se esiste un piano **h** di separazione che :
 - (a) contenga l'osservatore, oppure
 - (b) contenga p e l'osservatore in semispazi opposti.



Relazione di visibilità

Si consideri poi che, dati due poligoni **p**, **q** e un piano di separazione **h** ci sono tre possibili casi

- il piano **h** contiene l'osservatore

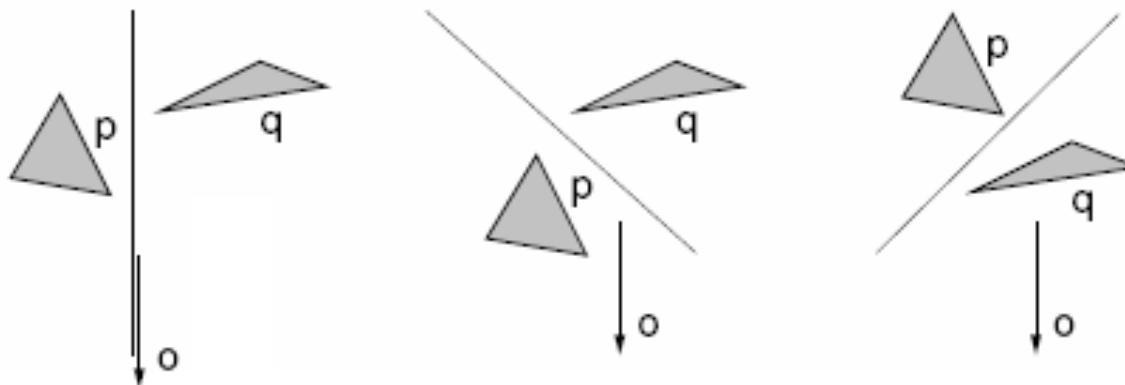
$$\mathbf{p} \leq \mathbf{q} \quad \mathbf{q} \leq \mathbf{p}$$

- l'osservatore appartiene allo stesso semispazio di **p**

$$\mathbf{q} \leq \mathbf{p}$$

- l'osservatore appartiene allo stesso semispazio di **q**

$$\mathbf{p} \leq \mathbf{q}$$



Grafo di visibilità

Il grafo della relazione di visibilità sull'insieme dei poligoni è chiamato
grafo di visibilità

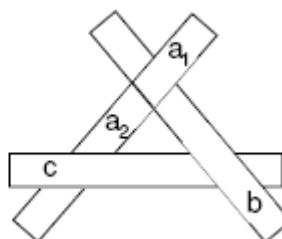
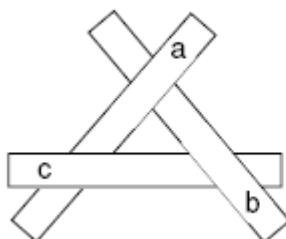
La relazione di visibilità **non è un ordine totale** ... vogliamo renderla tale!

Definizione di Ordine Totale

- **riflessiva** ed **antisimmetrica**. Se $a \leq b$ e $b \leq a$ allora $a = b$
- **transitiva** Se $a \leq b$ e $b \leq c$ allora $a \leq c$
- Per ogni coppia (a,b) deve valere $a \leq b$ o $b \leq a$

Grafo di visibilità

- Gli algoritmi in spazio oggetto devono calcolare un **ordine totale** dei poligoni, che sia compatibile con tale relazione.
- Rendere la relazione **antisimmetrica** è facile. Per ogni situazione $a \leq b$ e $b \leq a$ è sufficiente scegliere una delle coppie (es ordine lessicografico sugli identificativi)
- Più difficile è rendere la relazione **transitiva**
 - A questo scopo è necessario “aprire i cicli” del tipo **$a \leq c \leq b \leq a$** (SOLUZIONE frammentare un poligono)



Grafo di visibilità

- Un ordine totale derivato dal grafo di visibilità di una scena è detto **ordine in profondità**
- relativo algoritmo costruttivo è chiamato ordinamento in profondità ***depth-sort***.
- Gli algoritmi HSR in **spazio oggetto** si basano sulla relazione di visibilità
- Ordino in profondità i poligoni della scena, e sul successivo rendering di questi in ordine diretto (**back to front**) o inverso (**front to back**)

Uso del grafo di visibilità per il depth sort

- **depth-sort** (o algoritmo di Newell) è uno degli approcci più noti al problema HSR
- calcola un **ordine totale** sull' insieme di poligoni risultante dalle numerose fasi di pre-elaborazione
- ne fa uso per **rasterizzare** i poligoni dal più lontano al più vicino all'osservatore.

Modalità di processamento *back to front*

Descrizione:

- Quando i poligoni d'ingresso siano stati ordinati in profondità , l'algoritmo di Newell rimuove le parti nascoste della scena **rasterizzando** per primo il poligono più lontano.
- Poi viene rasterizzato il secondo poligono in distanza decrescente dall'osservatore, e questo può ricoprire alcune parti del precedente, e così via, fino a presentare sul dispositivo il poligono più vicino all'osservatore.
- Alla fine restano disegnate le sole parti della scena viste dall'osservatore

Svantaggi:

- può funzionare solo con dispositivi **raster**
- una immagine di una scena **complessa** la quantità dei dati può costituire un problema

Modalità di processamento *front to back*

- disegno i poligoni partendo dal più vicino
- il primo poligono è disegnato interamente ed assunto come valore iniziale del cosiddetto **contorno apparente** della scena
- L' **occlusore** è sottratto dai poligoni successivo (sequenza ordinata in profondità) aggiornando il contorno apparente
- perfetto per dispositivi vettoriali quali i grandi **plotter a penna**
- **riduce** la quantità dei dati grafici

Algorithm Front-to-back (input :: sequenza di poligoni 2D immersi in 3D))

{

```
    inverte la sequenza di input ;
    B := ∅; (inizializza il contorno apparente)
    for each poligono p ∈ input:
        p := p – B;
        disegna o rasterizza p;
        B := Unione(B,p);
```

}

Svantaggi:

- oneroso da calcolare
- Incorre in problemi numerici

Image-space

- ❖ Per ogni pixel, si considera un raggio che parte dal centro di proiezione e passa per quel pixel
- ❖ Il raggio è intersecato con ciascuno dei piani determinati dai k poligoni per determinare per quali piani il raggio attraversa un poligono
- ❖ L'intersezione più vicina al centro di proiezione è quella visibile

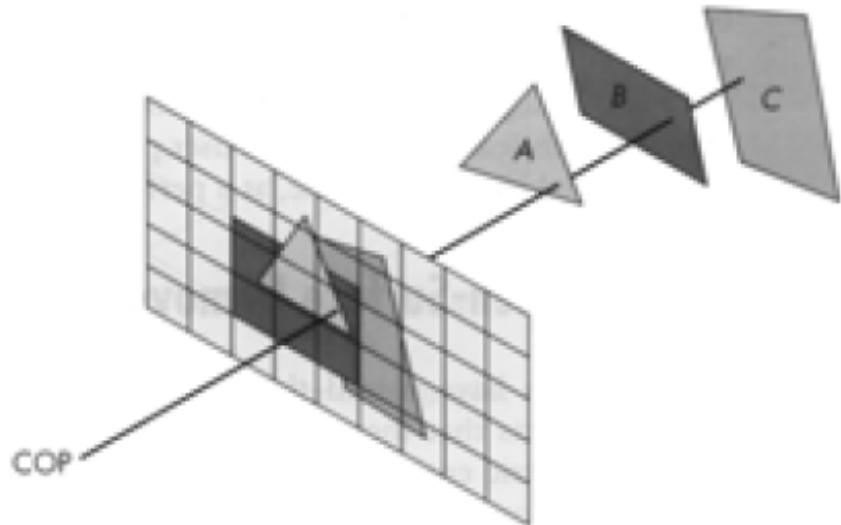


Image-space

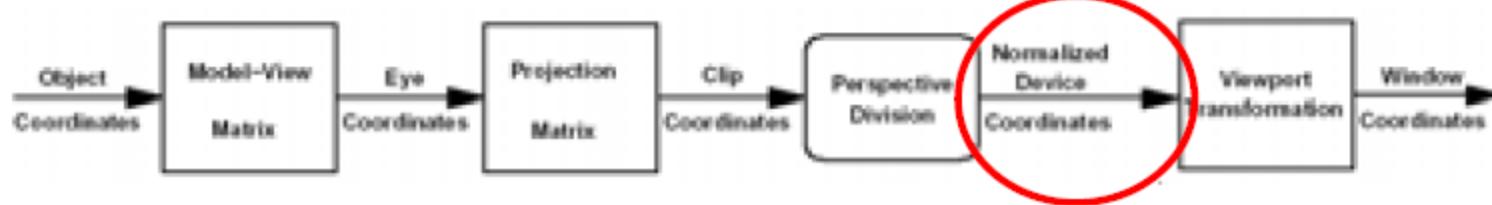
- ❖ L'operazione fondamentale dell'approccio image-space è il calcolo delle intersezioni dei raggi con i poligoni
- ❖ Per un display $w \times h$, questa operazione deve essere eseguita $w \cdot h \cdot n$ volte, e la complessità risulta di ordine $O(n)$, considerando quindi la risoluzione dello schermo una costante.

L'algoritmo *z-buffer*

- ❖ L'algoritmo **z-buffer** è un algoritmo di tipo image-space, basato su una logica molto semplice, e facile da implementare
- ❖ Lavora in accoppiamento con l'algoritmo di scan conversion, e necessita, oltre alla memoria di display (frame buffer), anche di un'area di memoria in cui memorizzare le **informazioni di profondità** relative ad ogni pixel
- ❖ Quest'area addizionale di memoria è chiamata **z-buffer**

L'algoritmo z-buffer

- ❖ La rasterizzazione dei poligoni avviene subito dopo la proiezione sul piano di vista in NDC
- ❖ Ad ogni pixel dello schermo possono coincidere 0, 1 o più primitive
- ❖ Nel corso dell'esecuzione della scan conversion, possiamo pensare al processo di proiezione come al calcolo del colore da associare ad ogni punto di intersezione tra una retta che passa dal centro di proiezione ed un pixel e le primitive

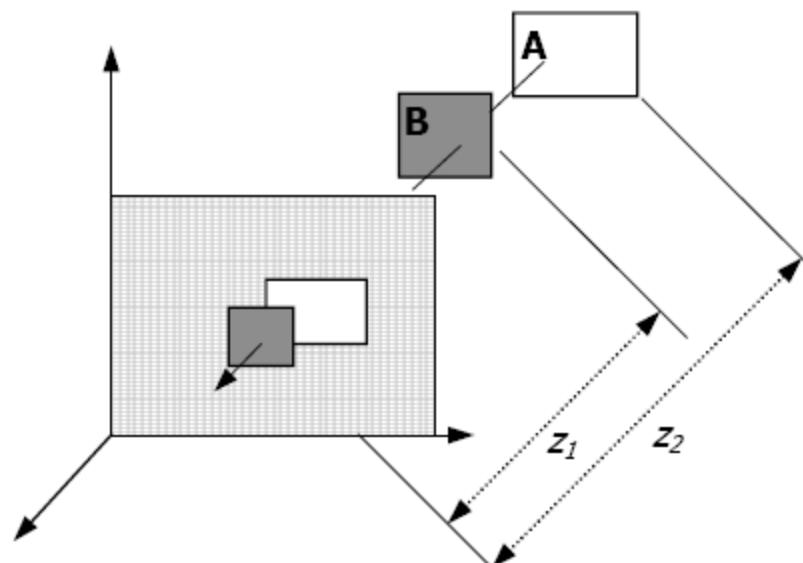


L'algoritmo z-buffer

- ❖ Nell'effettuare questa operazione, si può facilmente controllare se il punto di intersezione è visibile o meno (dall'osservatore), secondo la regola che stabilisce che
il punto è visibile se è il punto di intersezione più vicino al centro di proiezione

L'algoritmo z-buffer

- ❖ Quando si esegue la scan conversion del poligono B, il suo colore apparirà sullo schermo poiché la distanza z_1 è minore della distanza z_2 relativa al poligono A
- ❖ Al contrario, quando esegue la scan conversion del poligono A, il pixel che corrisponde al punto di intersezione non apparirà sul display



L'algoritmo z-buffer

- ❖ Poiché si procede poligono per poligono, non è possibile disporre di tutte le informazioni relative agli altri poligoni
- ❖ Dobbiamo disporre di una memoria, detta appunto z-buffer, che abbia la stessa risoluzione del frame buffer e con una profondità sufficiente per memorizzare le informazioni sulla risoluzione che si vuole ottenere per le distanze
- ❖ Ogni elemento dello z-buffer è inizializzato al valore della distanza massima dal centro di proiezione (il back-clipping plane)

L'algoritmo z-buffer

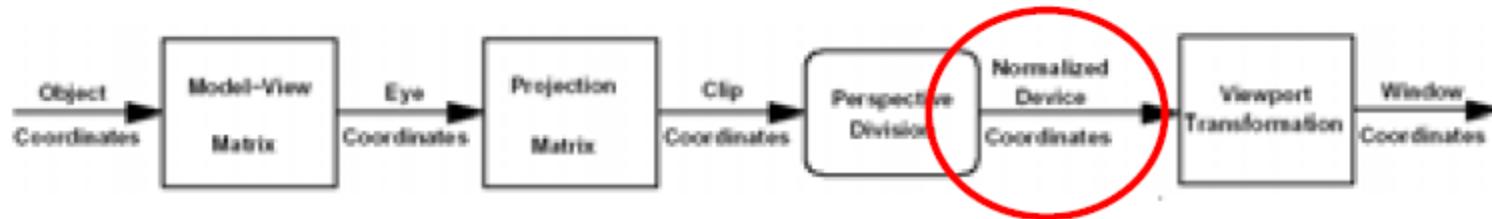
- ❖ Con questo approccio i poligoni possono essere rasterizzati in qualsiasi ordine (non è necessario alcun ordinamento preventivo dei poligoni in object-space, ovvero in 3D)

```
for y:=0 YMAX
    for x:=0 XMAX
        WritePixel(x,y,colore del background);
        WriteZ(x,y,0).
    for ogni poligono
        for ogni pixel nella proiezione del poligono
            pz:= valore della z nel pixel di coordinate (x,y)
            if pz>= ReadZ(x,y) then
                WriteZ(x,y,pz)
                WritePixel(x,y,colore del poligono nel pixel di coordinate (x,y))
```

L'algoritmo *z-buffer*

L'algoritmo z-buffer

- ❖ Se ripensiamo a come avviene la proiezione da 3 a 2 dimensioni, una volta trasformato il view frustum nel cubo in NDC possiamo pensare all'algoritmo di z-buffer come a quel metodo che anziché scartare la z al momento della proiezione la memorizza (nello z-buffer appunto) assieme all'informazione sul colore (nel frame buffer)

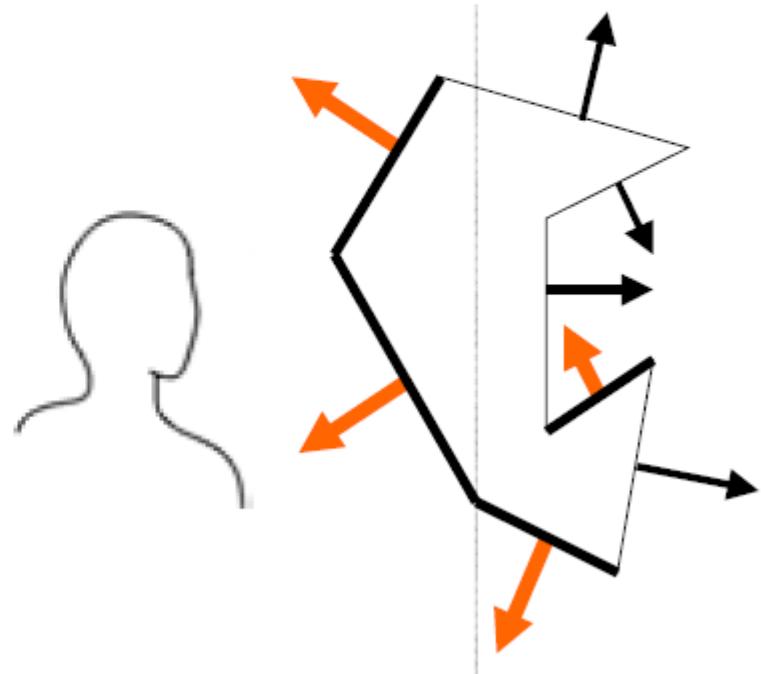


Eliminazione delle *back face*

- ❖ Oltre alla rimozione delle superfici nascoste esistono metodi opzionali che consentono di eliminare dalla pipeline di rendering primitive che si può decidere che saranno comunque invisibili
- ❖ Se un oggetto è rappresentato da un poliedro solido chiuso, le facce poligonali del poliedro delimitano completamente il volume del solido
- ❖ Modelliamo i poligoni in maniera tale che le normali alle loro superfici siano tutte dirette verso l'esterno del poligono

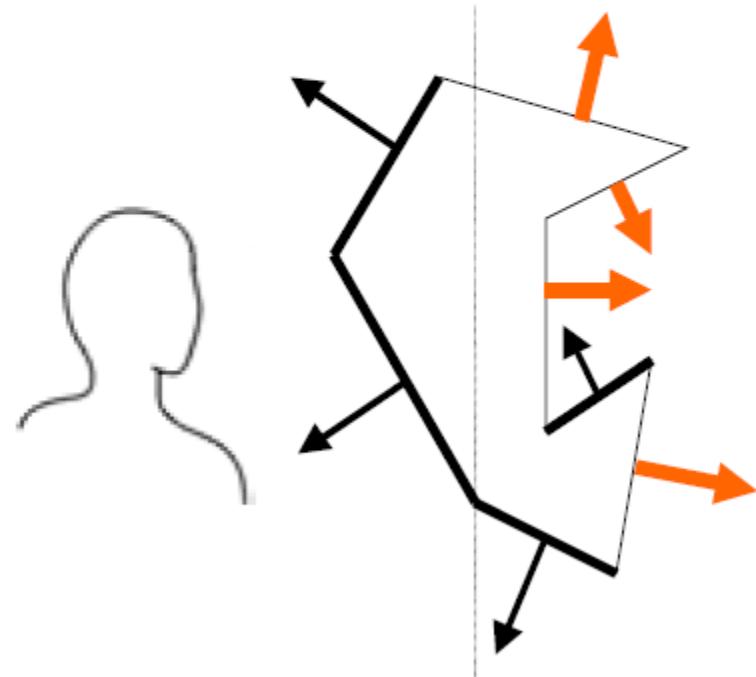
Eliminazione delle *back face*

- ❖ Se nessuna parte del poliedro viene tagliata dal *front clipping plane*:
 - ❖ le facce che hanno una normale che punta *verso* l'osservatore **possono** essere visibili



Eliminazione delle *back face*

- ❖ Se nessuna parte del poliedro viene tagliata dal *front clipping plane*:
 - ❖ quelle con normale che punta *via* dall'osservatore **sicuramente non lo sono**



Eliminazione delle *back face*

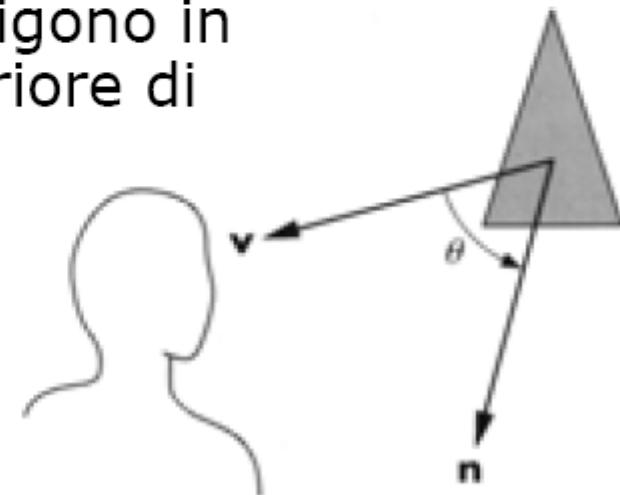
- ❖ Per determinare se un poligono deve essere eliminato, dobbiamo verificare se la sua normale è diretta o meno verso l'osservatore
- ❖ Se indichiamo con θ l'angolo tra la normale e l'osservatore, il poligono in esame definisce la parte anteriore di un oggetto se e solo se

$$-90^\circ \leq \theta \leq 90^\circ,$$

$$\cos \theta \geq 0$$

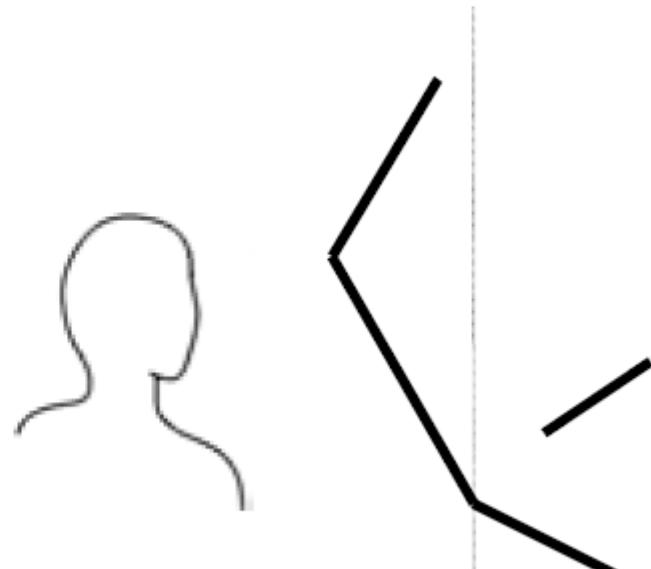
- ❖ Invece di calcolare il coseno, si usa il prodotto scalare

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



back face culling

- ❖ Le facce sicuramente invisibili non vengono più considerate nelle fasi successive del processo di rendering
- ❖ In media, circa metà delle facce di un solido chiuso sono inutili

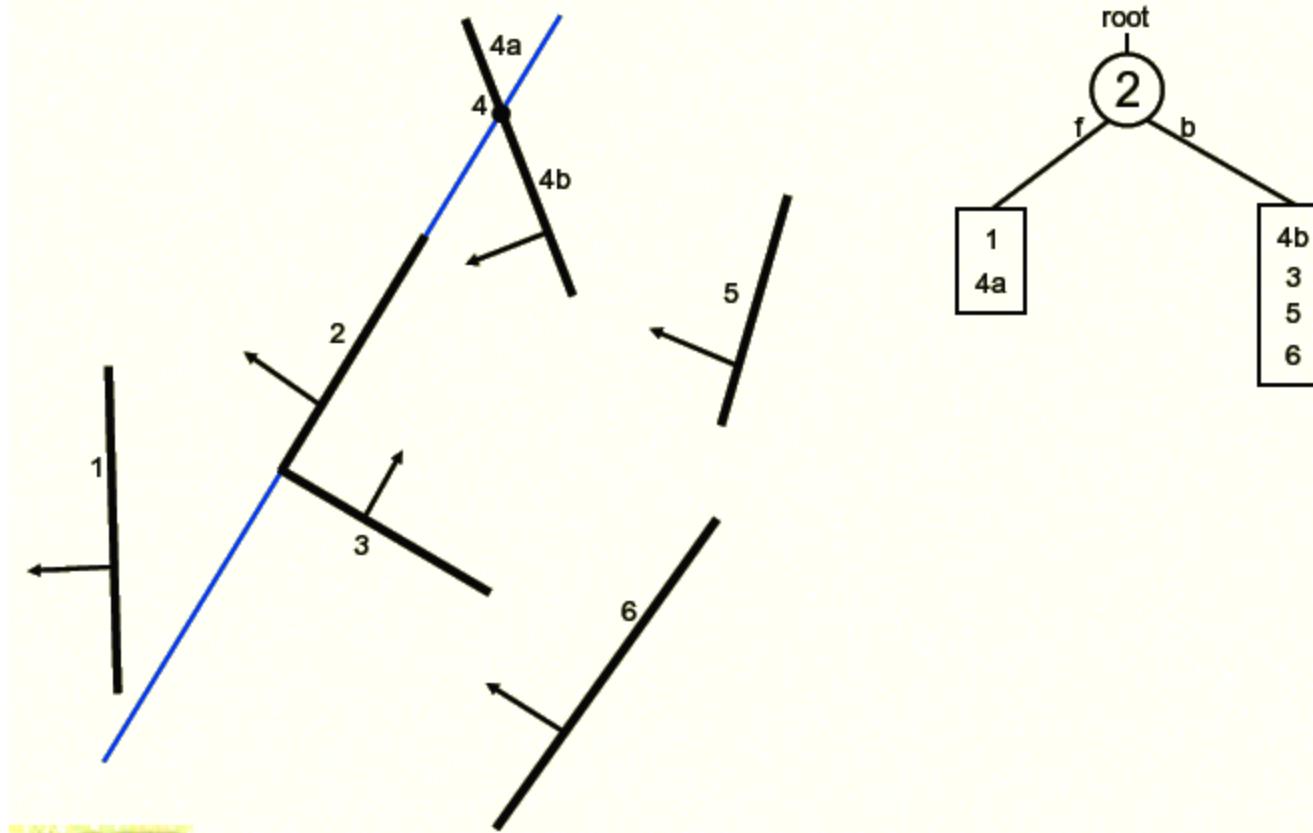


Binary Space Partitioning

- Dato un insieme di **iperpiani** nello spazio euclideo E^d , un albero BSP definito su tali iperpiani stabilisce un **partizionamento gerarchico** dello spazio E^d
- Ogni nodo **v** di un tale albero binario rappresenta una regione **R** convessa ed eventualmente illimitata di E^d
- I due figli di ogni nodo interno v sono denotati come **below(v)** e **above(v)** rispettivamente
- Ogni nodo **interno v** dell'albero è associato con un iperpiano di partizionamento **h**, che interseca l'interno di **R**

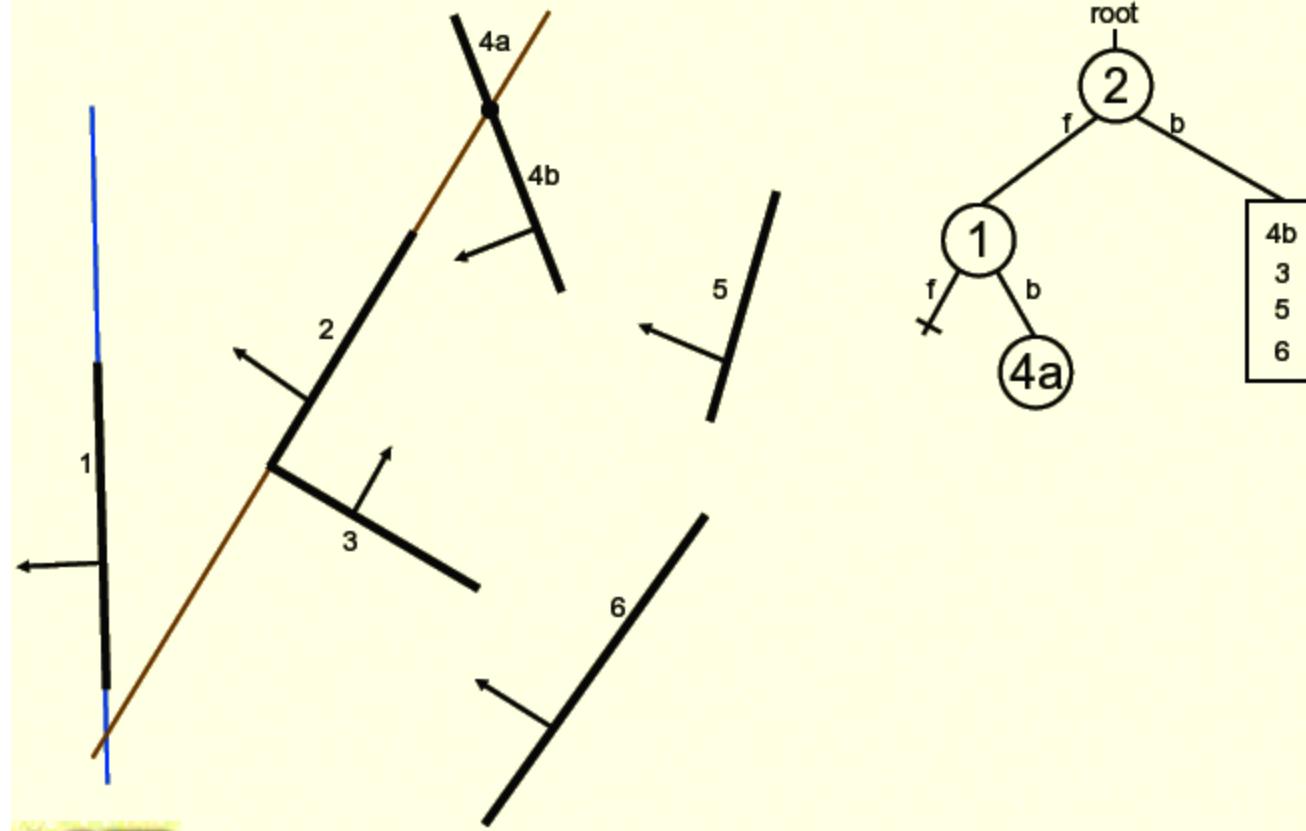
Binary Space Partitioning

Binary space partitioning



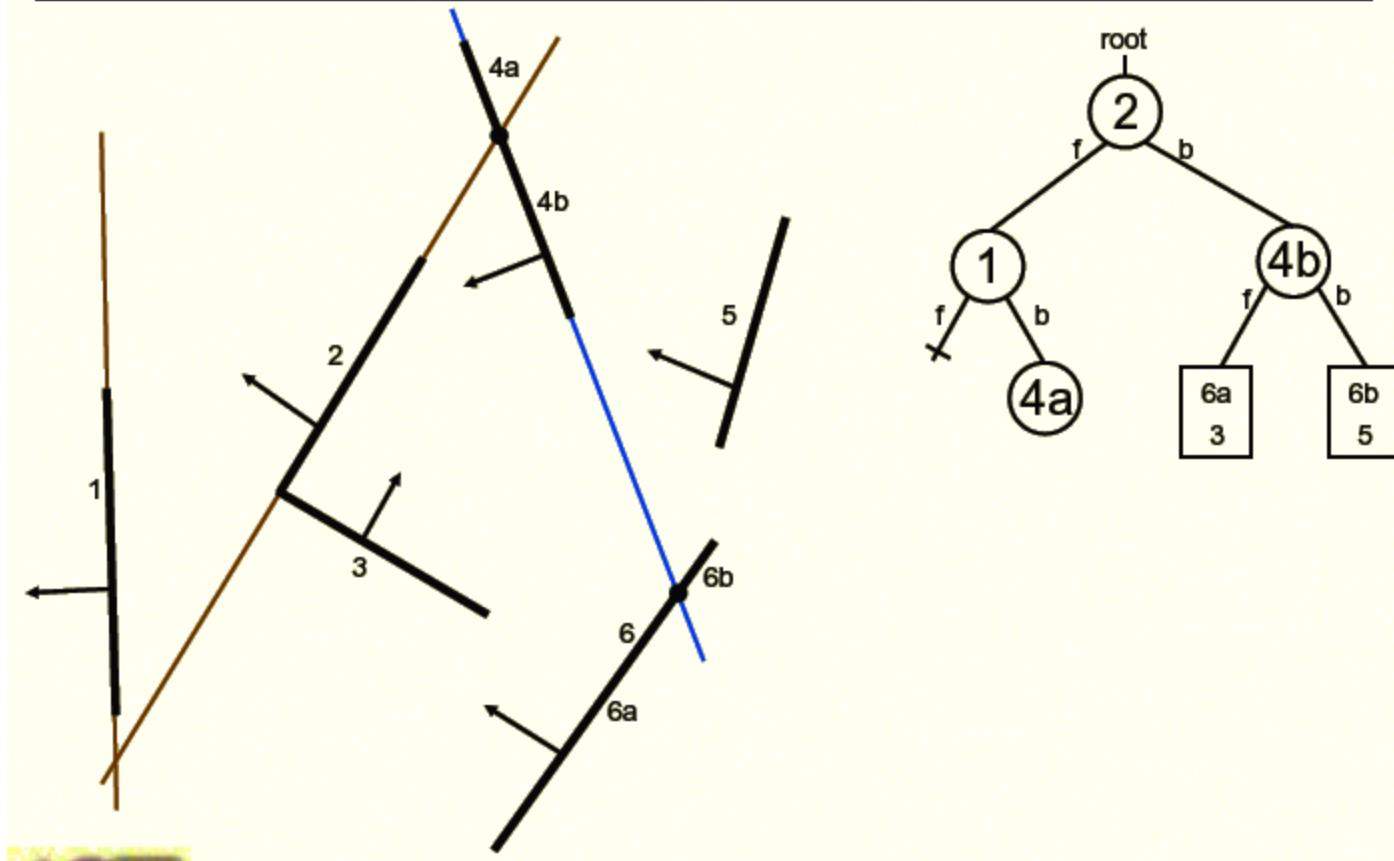
Binary Space Partitioning

Binary space partitioning



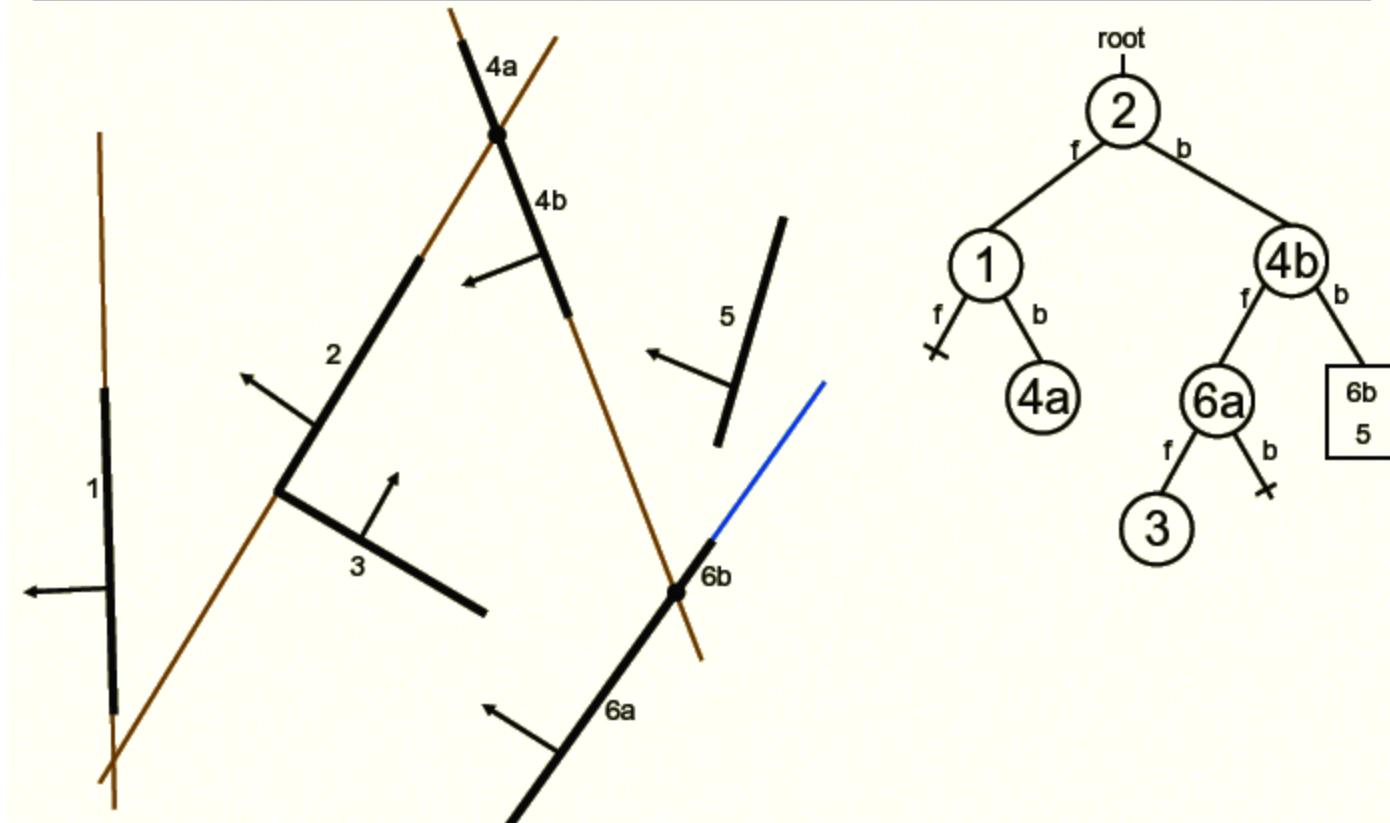
Binary Space Partitioning

Binary space partitioning



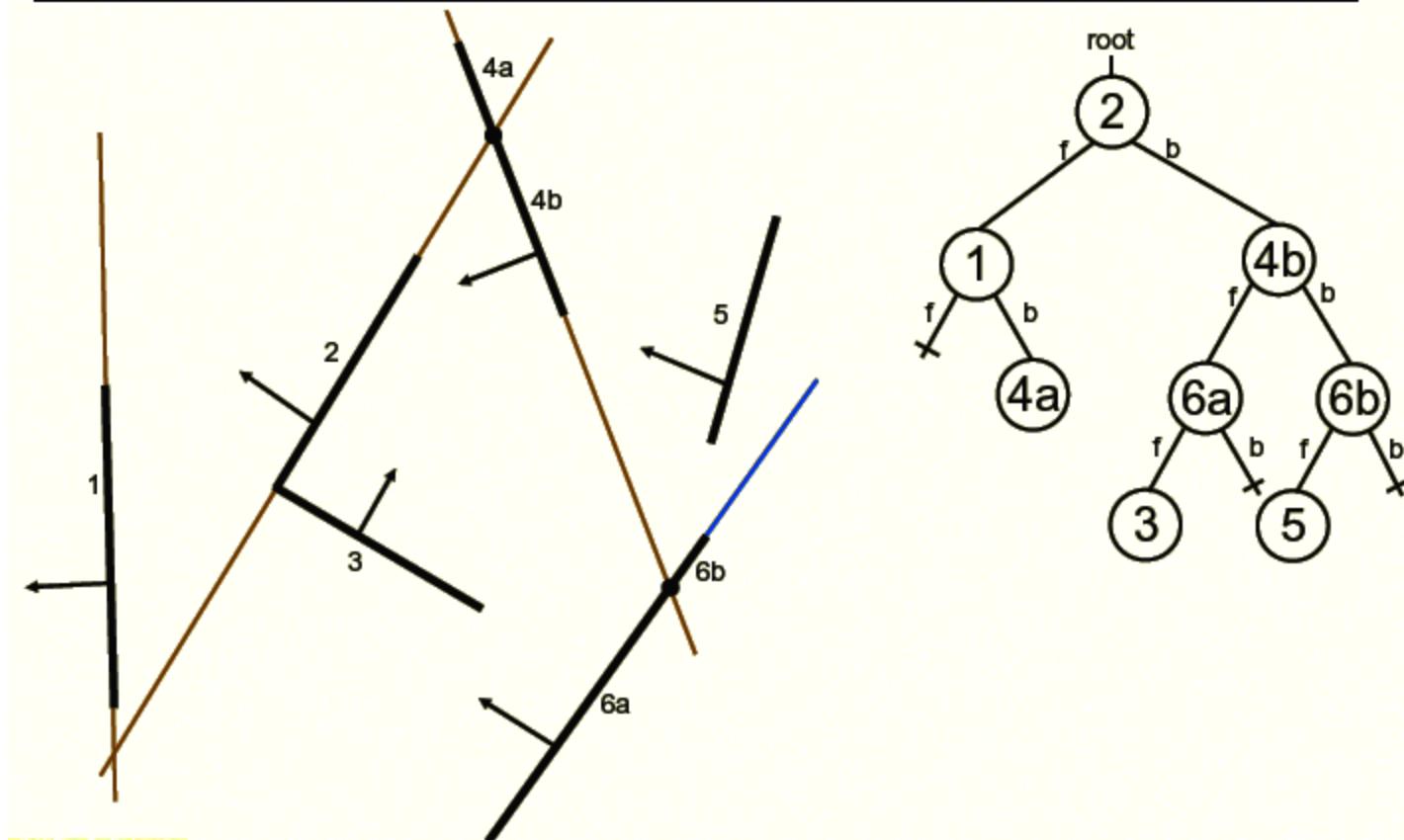
Binary Space Partitioning

Binary space partitioning



Binary Space Partitioning

Binary space partitioning



Binary Space Partitioning

- L'algoritmo di **partizione binaria dello spazio** (BSP) rimuove le parti nascoste della scena lavorando in spazio oggetto
- E' stato usato con successo nei **giochi** (per esempio nel famoso DOOM)
- Costruisce, in una fase di **pre-elaborazione**, una struttura dati specializzata, che è una sorta di *indice spaziale*.
- Tale struttura dati è un **albero binario** chiamato albero di partizione binaria dello spazio, o semplicemente **albero BSP**
- Per ogni possibile **posizione** dell'osservatore, un **ordinamento** in profondità dei poligoni e' possibile in tempo lineare con il loro numero.

Binary Space Partitioning

- BSP può essere usato per ordinare i poligoni rispetto a **qualunque posizione** dell'osservatore
- Un nuovo ordinamento in profondità deve essere realmente calcolato solo quando l'osservatore esce dalla cella
- Una tale importante proprietà delle animazioni fu scoperta negli anni '70 sotto il nome di **coerenza della scena**.
- L'algoritmo BSP può essere descritto in modo **indipendente dalla dimensione**.

Binary Space Partitioning

- **Approccio** Si possono distinguere tre fasi principali nell'algoritmo:
 1. pre-elaborazione, con **costruzione** dell'albero BSP;
 2. **visita** dell'albero BSP, con generazione dell'ordinamento in profondità;
 3. **rendering** della lista ordinata dei poligoni.
- Se la **scena è statica** la costruzione dell'albero è necessaria **una volta sola**.
- Al contrario, la visita dell'albero BSP è necessaria per ogni cella dello spazio attraversata dall'osservatore (i.e., quando l'osservatore attraversa il piano di partizionamento della regione R in cui si trova).

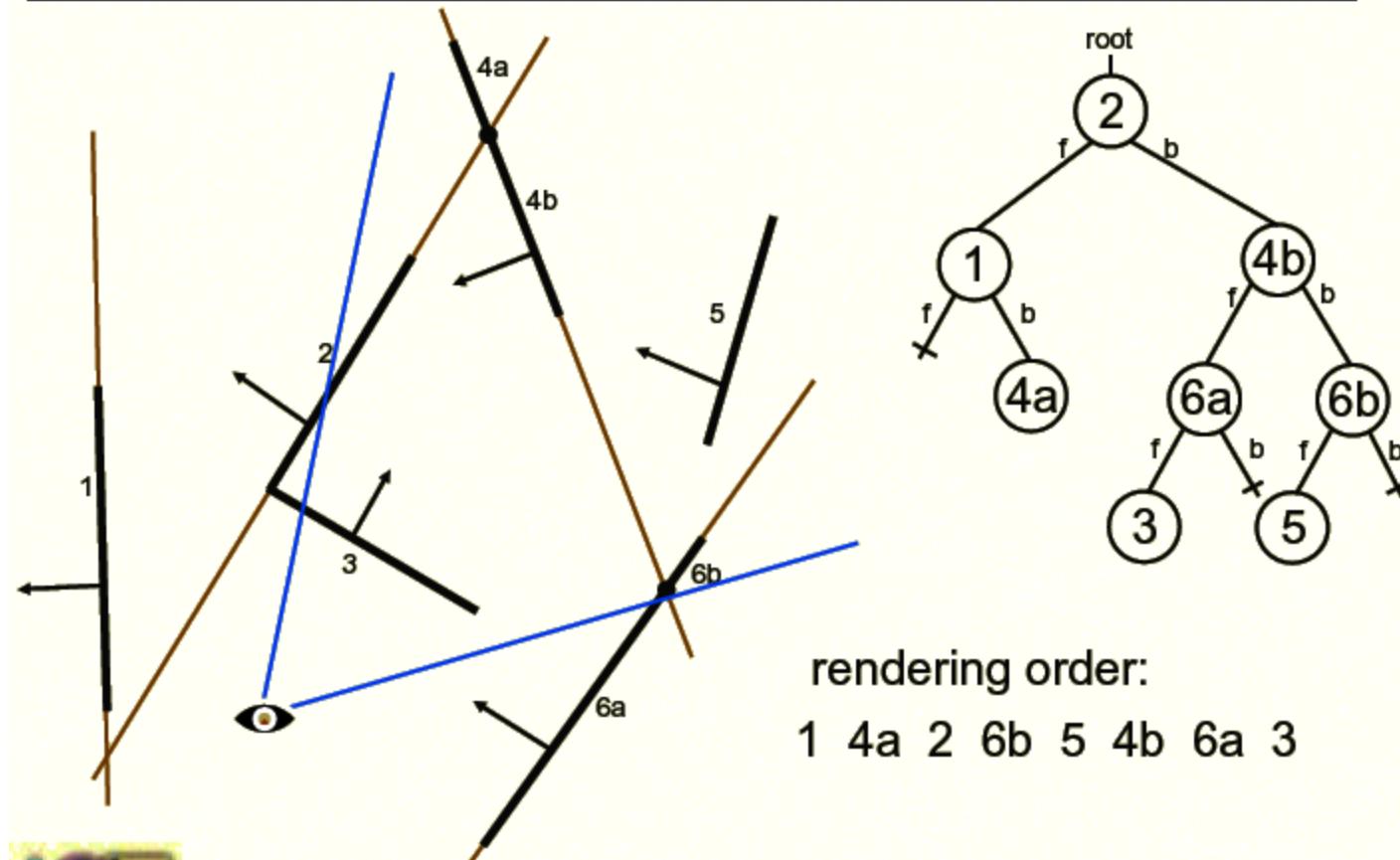
Binary Space Partitioning - Ordinamento back to front

Algorithm TraverseB2F ($v : \text{BSPtree}$)

```
{  
    if IsLeaf ( $v$ ) then  
        Output(  $v$  )  
    else  
    {  
        if  $\bullet \in R+(v)$  then {  
            TraverseB2F (below( $v$ ))  
            Output(  $v$  )  
            TraverseB2F (above( $v$ ))  
        }  
        else {  
            TraverseB2F (above( $v$ ))  
            Output(  $v$  )  
            TraverseB2F (below( $v$ ))  
        }  
    }  
}
```

Binary Space Partitioning - Ordinamento back to front

BSP traversal algorithm (back-to-front)



Binary Space Partitioning - Ordinamento back to front

BSP traversal algorithm (back-to-front)

