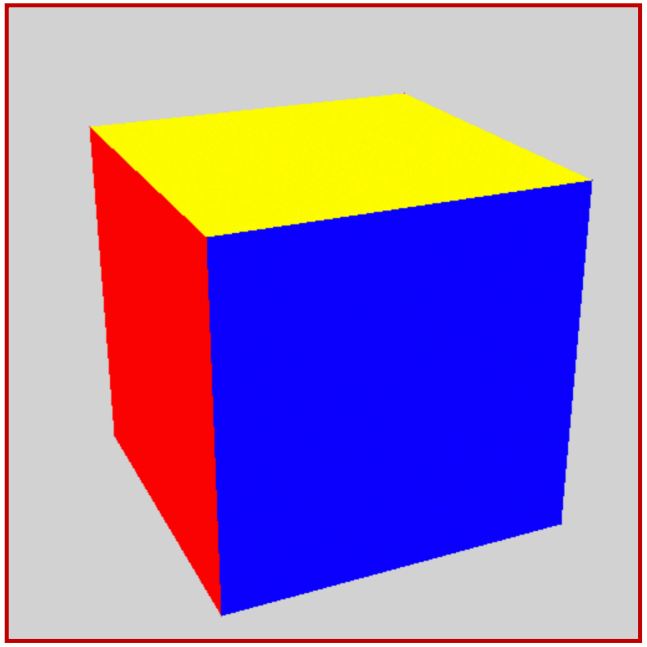
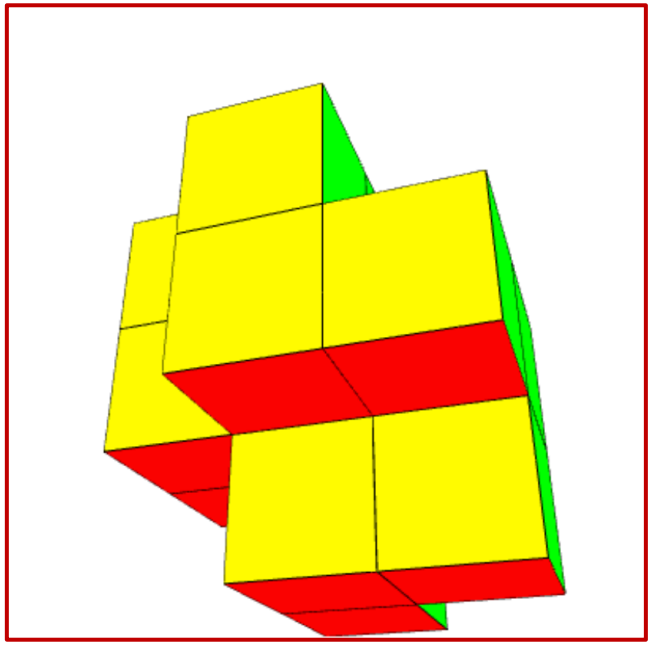
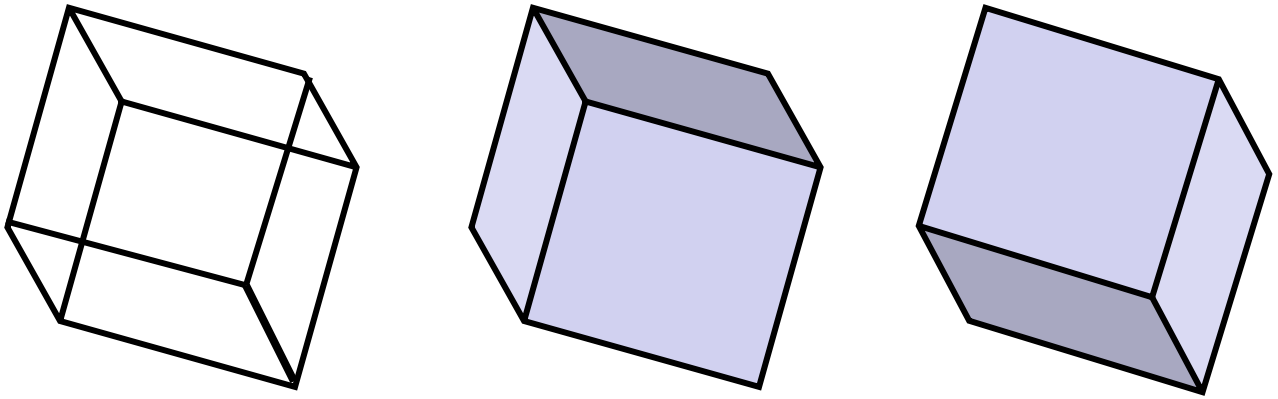
Algoritmi di real-time Rendering





Wire Frame

Ambiguità del wire frame:



Con rendering ci si riferisce alla resa di una immagine a partire dalla descrizione geometrica di una scena 3D (modelli 3D), simulando quello che vedrebbe un osservatore.

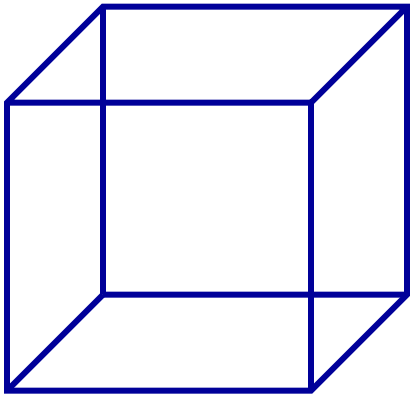
Nel processo di rendering il problema di determinare le linee e superfici nascoste, insieme ad una visualizzazione real-time, è uno dei problemi più interessanti della Computer Graphics.

Gli algoritmi relativi, cercano di determinare quali lati o superfici (facce triangolari) siano visibili da un determinato punto di vista in un tempo utile per una visualizzazione real-time.

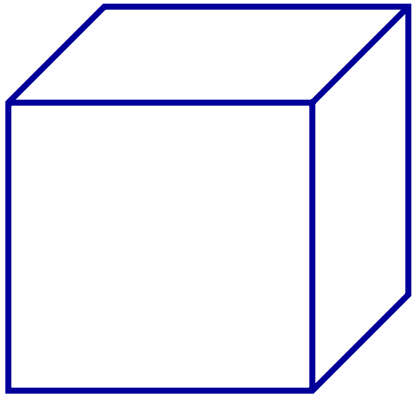
Classificazione e storia

Eliminazione delle Parti Nascoste

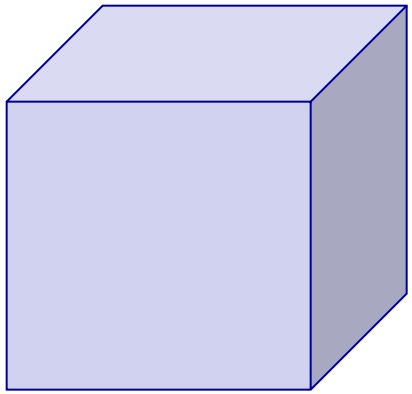
- Hidden Line Removal (HL)
- Hidden Surface Removal (HS)



WireFrame



Hidden Line



Hidden Surface

Eliminazione Parti Nascoste

Gli algoritmi che si basano sull'eliminazione delle parti nascoste implicano un ordinamento.

L'ordinamento principale è basato sulla distanza tra un lato (Edge) o una superficie (Face) e il punto di osservazione (Vp).

Un oggetto vicino all'osservatore avrà più probabilità di essere visibile di un oggetto lontano; dopo aver determinato la distanza si procede all'ordinamento in senso locale per determinare se l'oggetto è nascosto da quelli più vicini.

L'efficienza di questi algoritmi dipende quindi dall'efficienza delle tecniche di ordinamento.

Per aumentare questa efficienza si usa il concetto di coerenza, cioè la tendenza di parti della scena a rimanere costanti nello spazio.

Hidden Line/Surface Removal

I lati o le facce o loro parti che risultano visibili all' osservatore (non nascosti/e) devono essere disegnati/e. Sappiamo cosa vuol dire disegnare un lato, ma cosa vuol dire …

… disegnare una faccia?

Può voler dire disegnarne i lati, o meglio i lati visibili della faccia (rappresentazione wire-frame), oppure colorare la parte visibile della; cioè disegnare con un colore tutti e soli i pixel interni (rappresentazione solida); ma prima di disegnarli devono essere determinati; si tratta di applicare un algoritmo di rasterizzazione.

Hidden Line/Surface Removal

Classificazione: Sutherland, Sproull, Schumaker (1974):

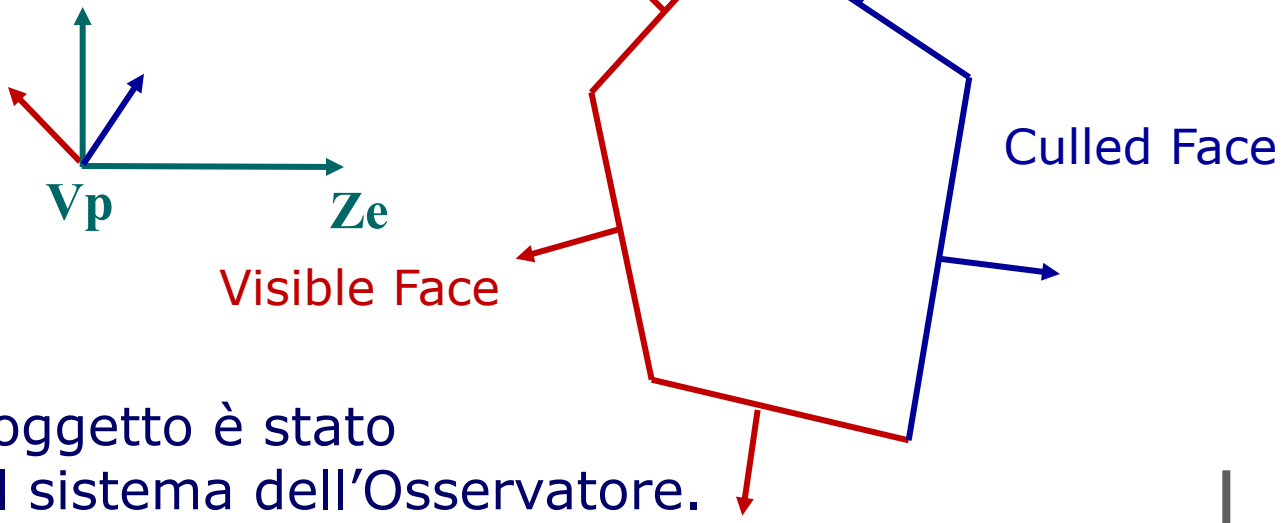
- **Object Space**
  - Calcoli geometrici su poligoni nello spazio 3D
  - Precisione Floating point
  - Si deve processare la scena nell'ordine degli oggetti
- **Image Space**
  - Visibilità al pixel
  - Precisione Intera
  - Si deve processare la scena nell'ordine delle immagini

Algoritmo Back Face Culling

Si applica ad oggetti (Mesh 3D) convessi:

- Le Facce siano definite dai Vertici in senso antiorario rispetto a chi guarda dall'esterno; la normale calcolata punta allora verso l'esterno;
- Si testi la componente Z della normale di ogni faccia;

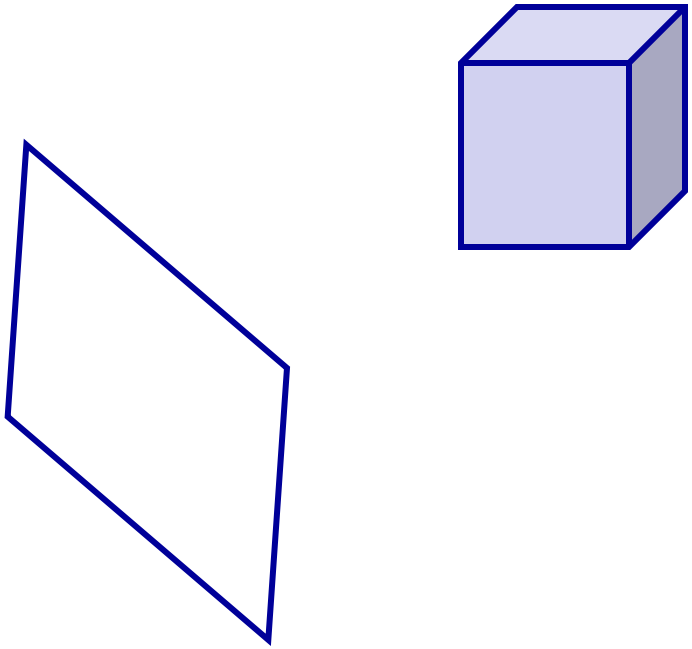
se positiva la faccia non dovrà essere disegnata perché non visibile dall'Osservatore;



Esempio 2D; l'oggetto è stato trasformato nel sistema dell'Osservatore.

Backface Culling

Più in generale:

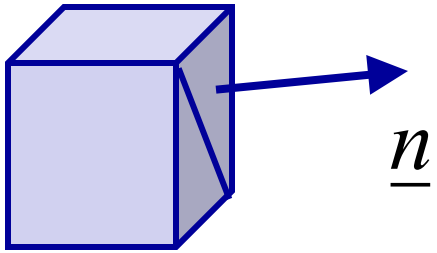


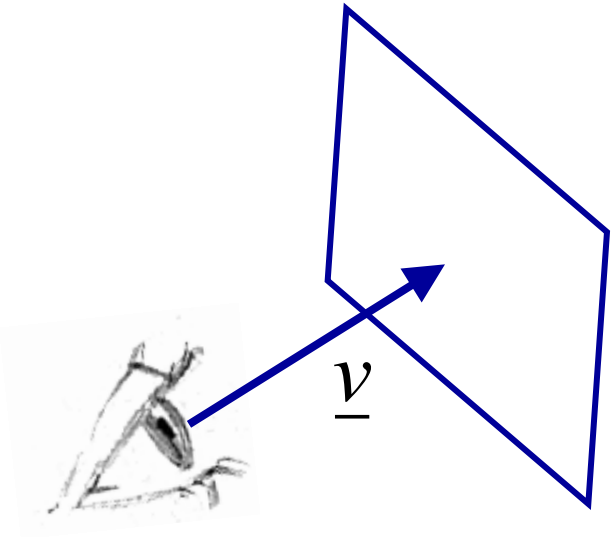
Backface Culling

Più in generale:

$$\underline{n} \cdot \underline{v} \geq 0$$ allora scarta la faccia

(no proiezione e no disegno)

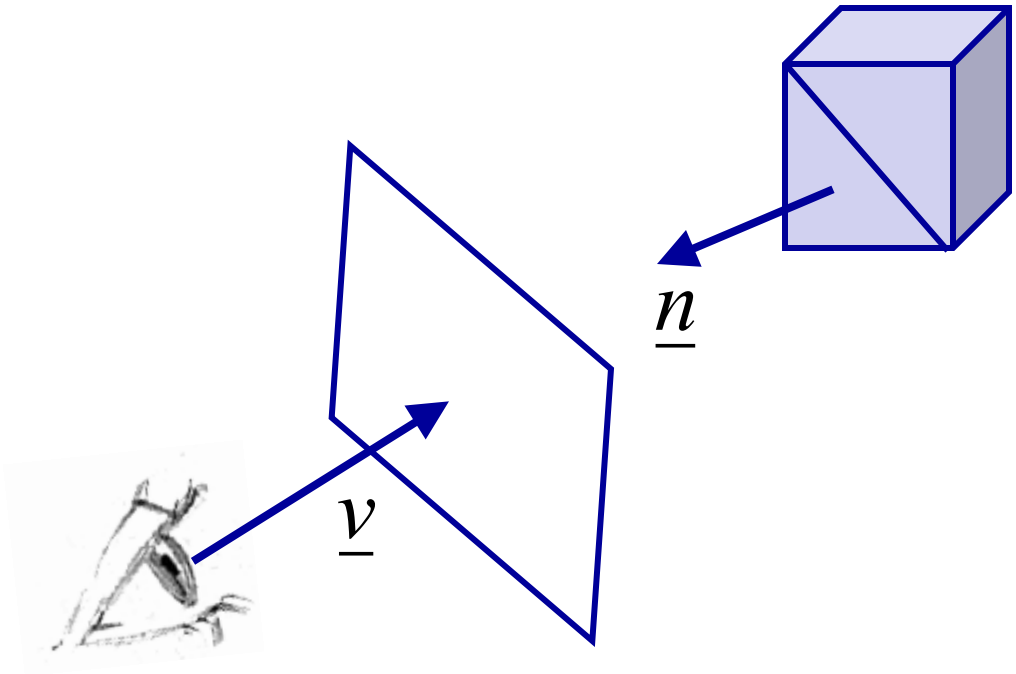




Backface Culling

Più in generale:

$\underline{n} \cdot \underline{v} < 0$ allora processa la faccia
(proiezione e disegno)

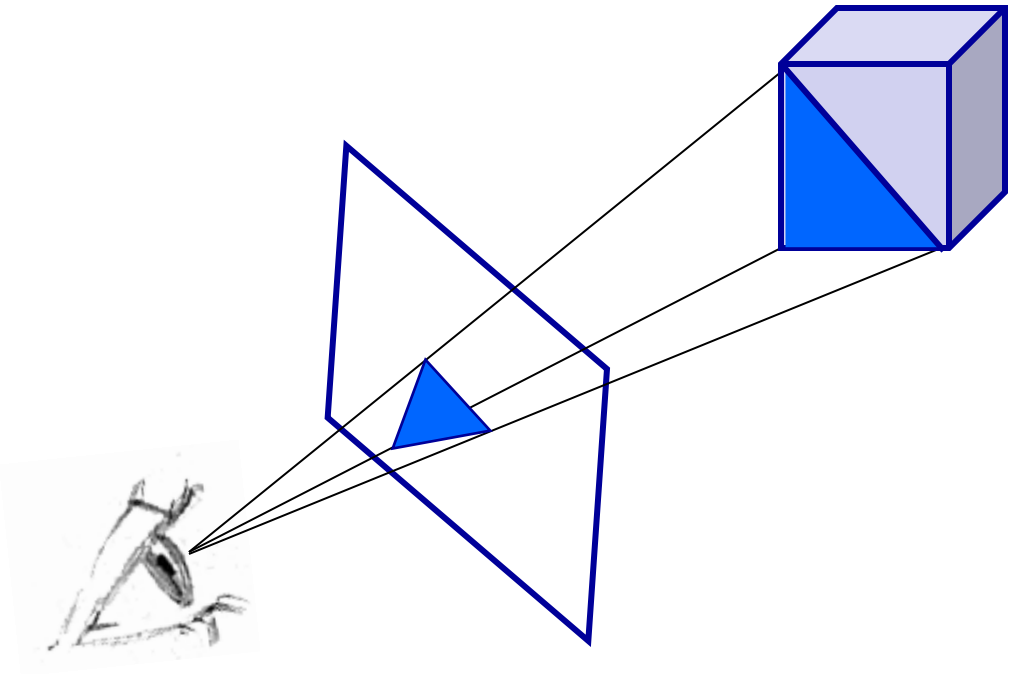


Backface Culling

Più in generale:

… proiezione e disegno

disegno = algoritmo di rasterizzazione della faccia



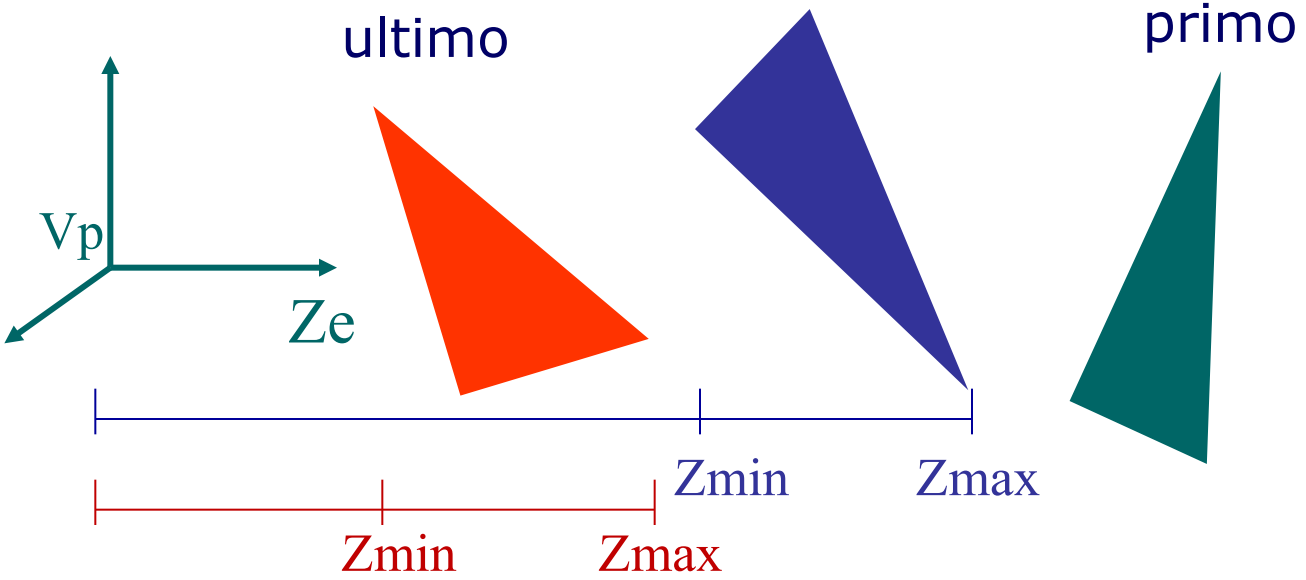
Backface Culling

- Vantaggi
  - Aumenta la velocità di rendering rimuovendo circa la metà delle facce, che quindi non verranno processate (proiezione e disegno/rasterizzazione)
  - Non serve l'ordinamento delle facce

- Svantaggi
  - Funziona solo per superfici chiuse, convesse e senza buchi;
  - Non può essere considerato un vero algoritmo di Hidden Surfaces

Algoritmo del Pittore

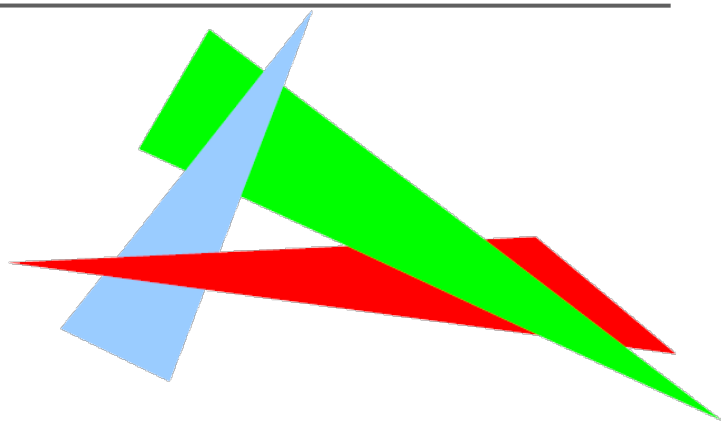
- Si ordinano i triangoli dal più lontano al più vicino rispetto all'Osservatore (usando la coordinata Ze nel sistema dell'Osservatore)
- Si rasterizzano i triangoli secondo l'ordinamento così determinato



se Zmin > Zmax allora l'ordine è blu e rosso

Algoritmo del Pittore

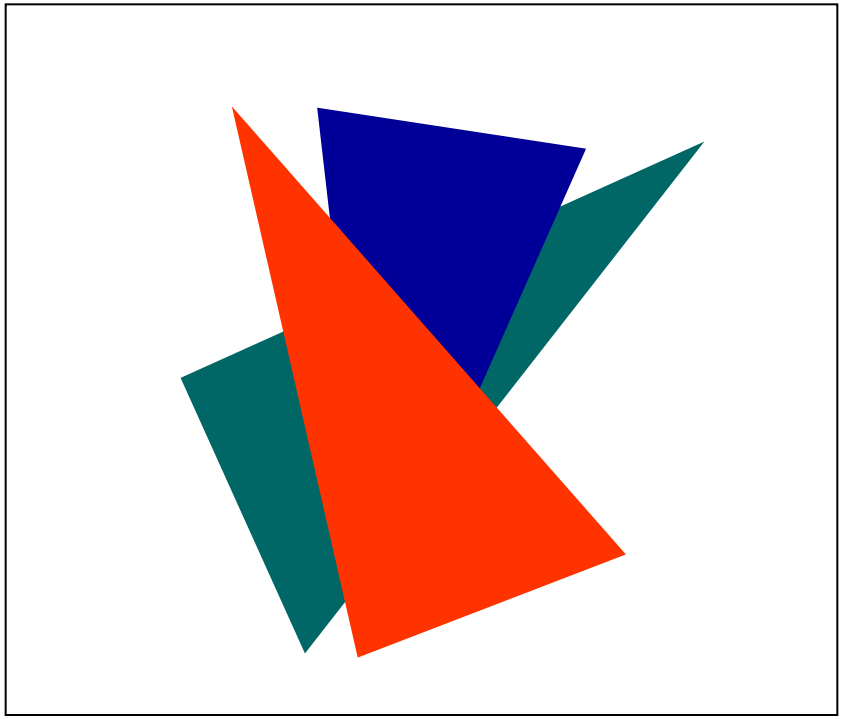
Casi in cui non funziona:
- Intersezioni
- Cicli



Questi casi si risolvono suddividendo i triangoli, ma questo è difficile e costoso;

Complessità:
- Ordinamento

Algoritmo del Pittore



Fase di disegno

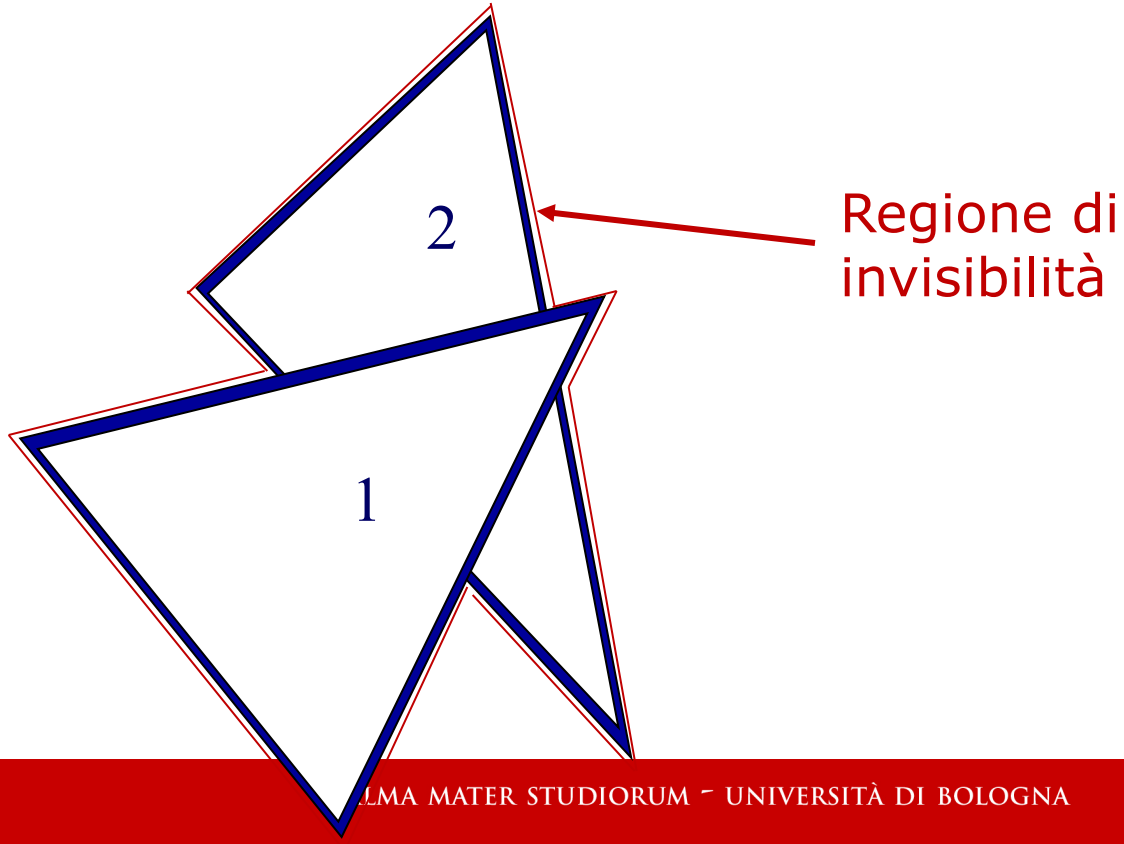
Nota: vengono disegnati tutti pixel di tutte le facce (rasterizzazione di tutte le facce)

Algoritmo del Pittore

- Vantaggi
  - Si basa su un semplice agoritmo di ordinamento di poligoni
- Svantaggi
  - E' difficile definire un criterio di ordinamento
  - Ridisegna certi pixel molte volte
  - L'ordinamento può essere costoso

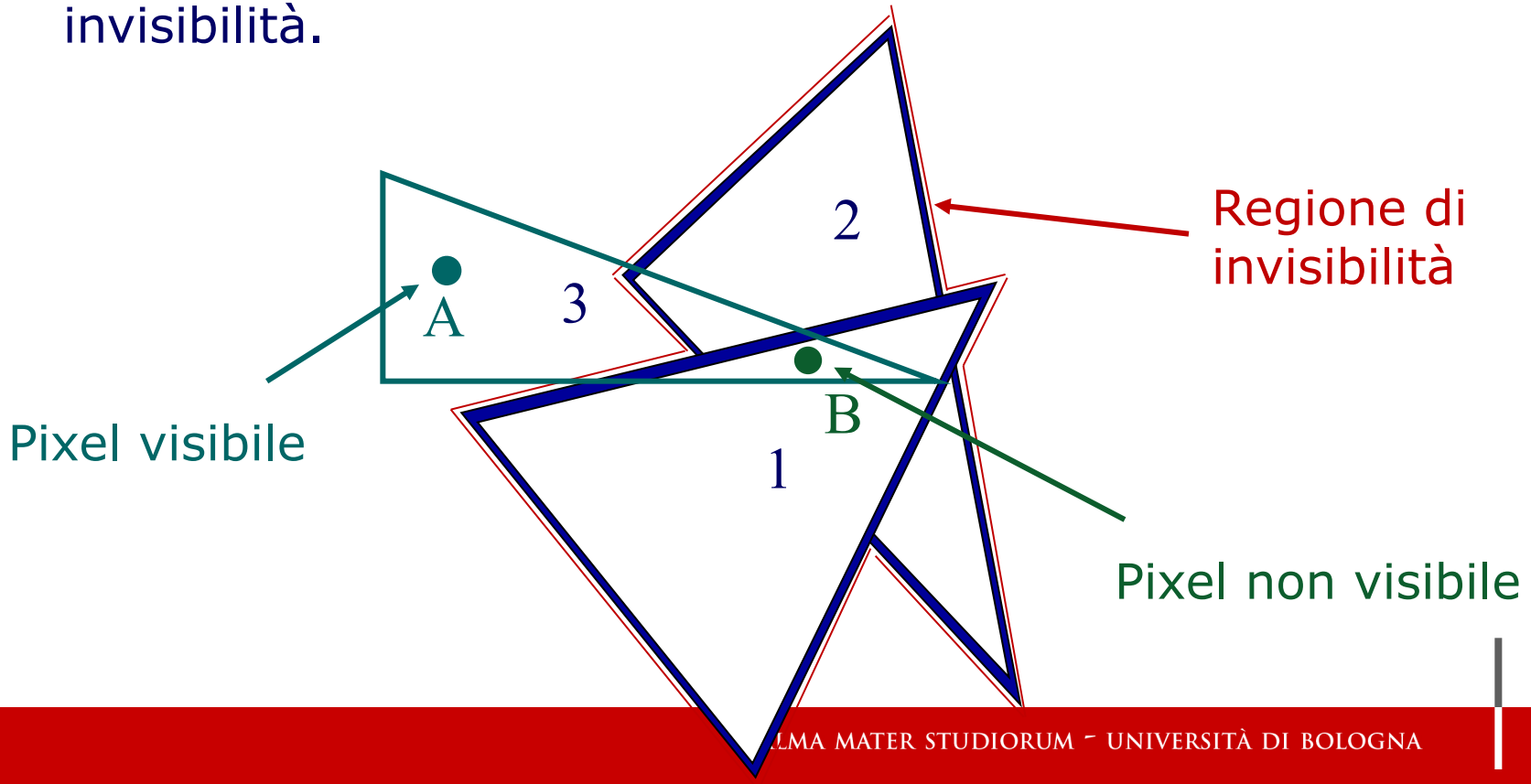
Visibility Buffer (HS)

Si ordinano i triangoli dal più vicino al più lontano rispetto all'Osservatore (usando la coordinata Ze nel sistema dell'Osservatore)
Si rasterizzano i triangoli secondo l'ordinamento così determinato pixel per pixel, testando la regione di invisibilità.



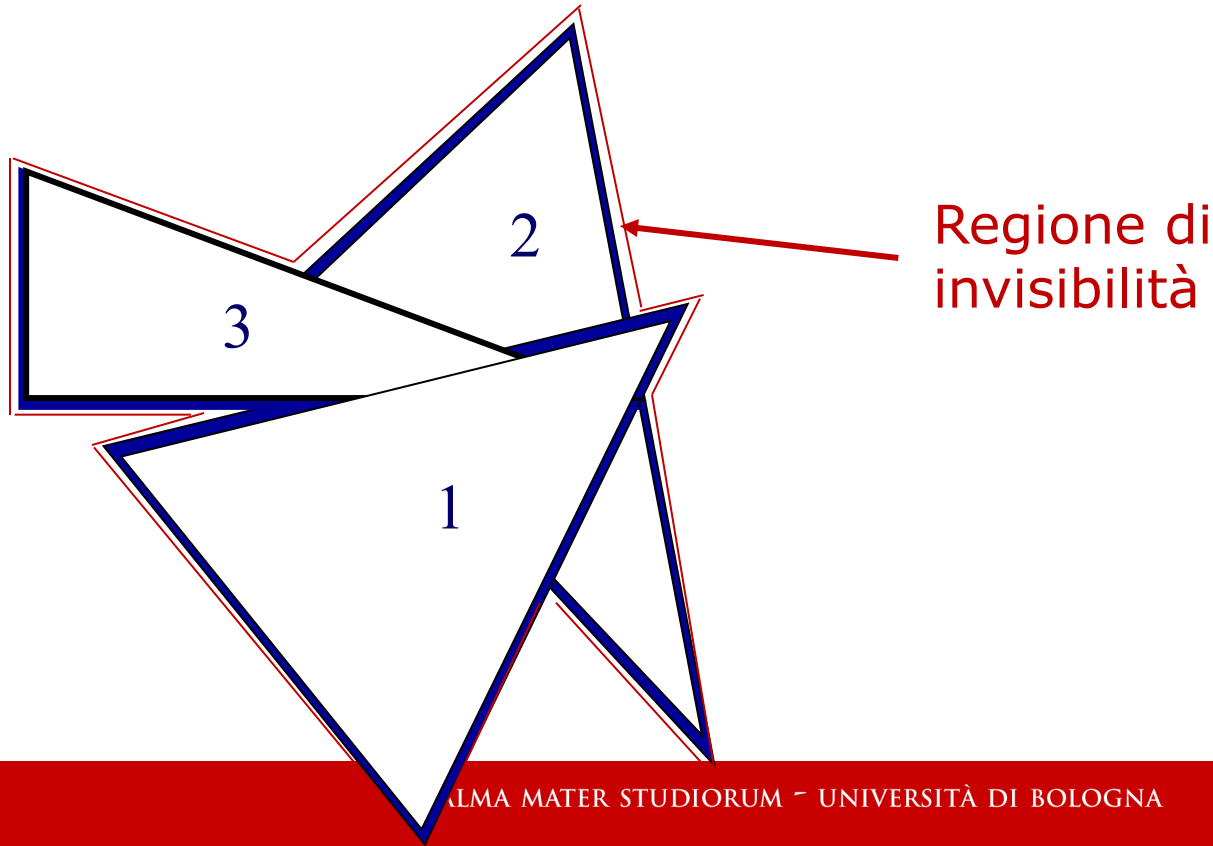
Visibility Buffer

Si ordinano i triangoli dal più vicino al più lontano rispetto all'Osservatore (usando la coordinata Ze nel sistema dell'Osservatore)
Si rasterizzano i triangoli secondo l'ordinamento così determinato pixel per pixel, testando la regione di invisibilità.



Visibility Buffer

Si ordinano i triangoli dal più vicino al più lontano rispetto all'Osservatore (usando la coordinata Ze nel sistema dell'Osservatore)
Si rasterizzano i triangoli secondo l'ordinamento così determinato pixel per pixel, testando la regione di invisibilità.



Visibility Buffer

- Vantaggi
  - Si basa su un semplice algoritmo di ordinamento di poligoni
  - Ogni pixel viene disegnato una sola volta
- Svantaggi
  - E' difficile definire un criterio di ordinamento
  - L'ordinamento può esere costoso
  - Si deve memorizzare la regione di invisibilità

Algoritmo Z-buffer

E' l'algoritmo di eliminazione di superfici nascoste implementato sulle GPU.
Fu proposto originariamente da Catmull nel 1975 ed è un algoritmo Image Space.

E' una semplice estensione del concetto di Frame Buffer;

- Viene usato un frame buffer (di interi) per memorizzare le intensità/colori di ogni pixel;

- Viene usato un buffer, detto Z-buffer (di floating point) per memorizzare le coordinate Z o profondità (da cui anche Depth-buffer) di ogni pixel visibile nello spazio immagine;

Algoritmo Z-buffer

- Si considera una faccia triangolare (i tre vertici);
- Si applica la trasformazione di vista per ottenere la sua immagine sulla viewport;
- Si rasterizza la faccia e per ogni pixel considerato, si determina la profondità Z del punto 3D che il pixel rappresenta, in coordinate dell'osservatore;
- Si confronta tale profondità con quella memorizzata nello Z-buffer in corrispondenza di quel pixel;
- Se dal confronto risulta che il nuovo pixel (punto) ha profondità minore, allora si memorizza la sua profondità nello Z-buffer e la sua intensità/colore nel frame buffer.

Algoritmo Z-buffer

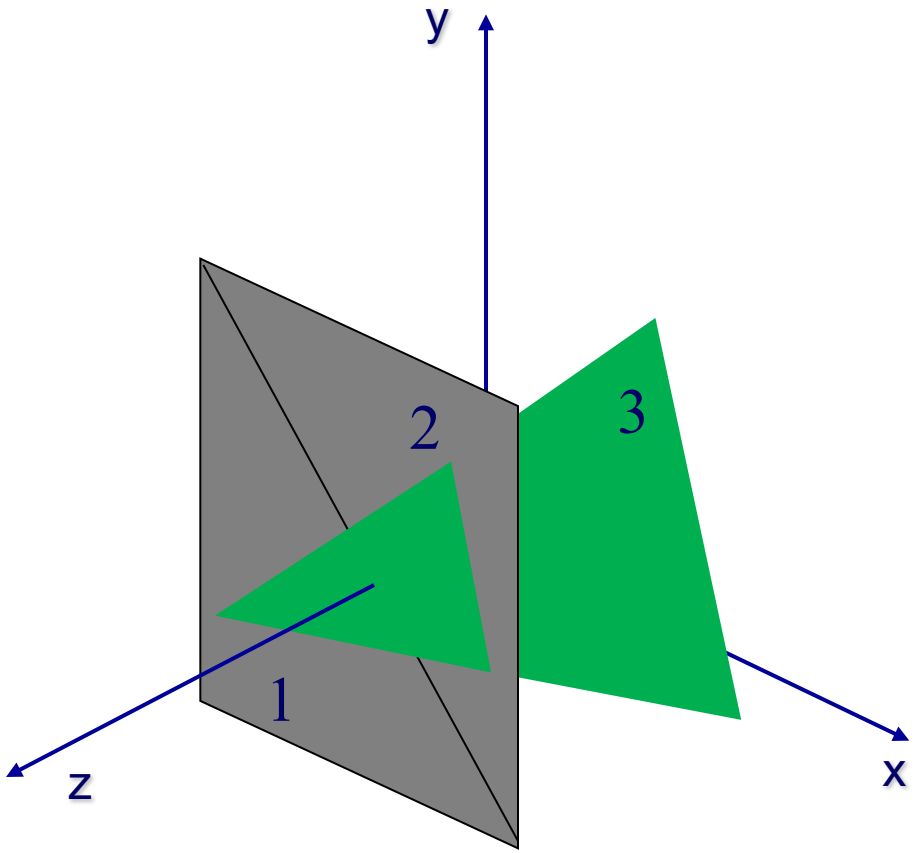
```
void zbuffer() {

for (y = 0; y < Vymax; y++)
  for(x = 0; x < Vxmax; x++) {
      FrameBuffer(x,y) = BACK_Col;
      ZBuffer(x,y,INF_Val);
 }

 for (every triangle)
   for (all pixel (px,py) of the projected triangle) {

      pZ = computeZ(px,py);
      if (pZ < ZBuffer(px,py))  //pixel is visible
      {
         FrameBuffer(px,py) = computeColor(px,py);
         ZBuffer(px,py) = pZ;
      }
   }
}
```

Esempio



Facce 3D

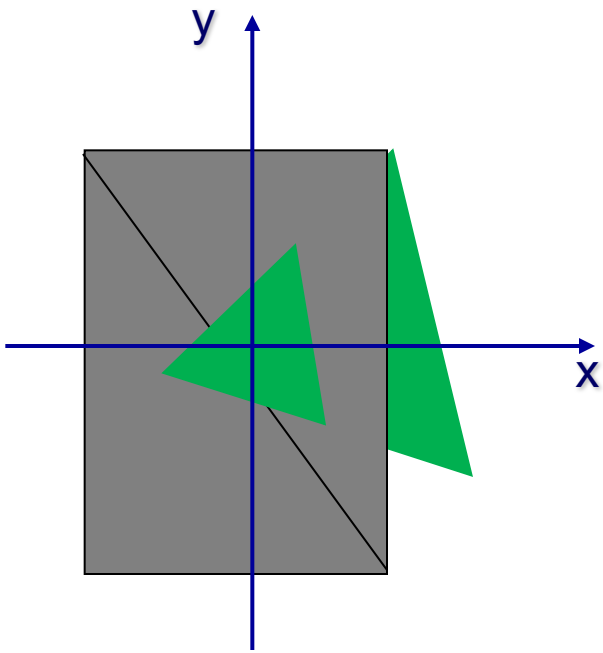
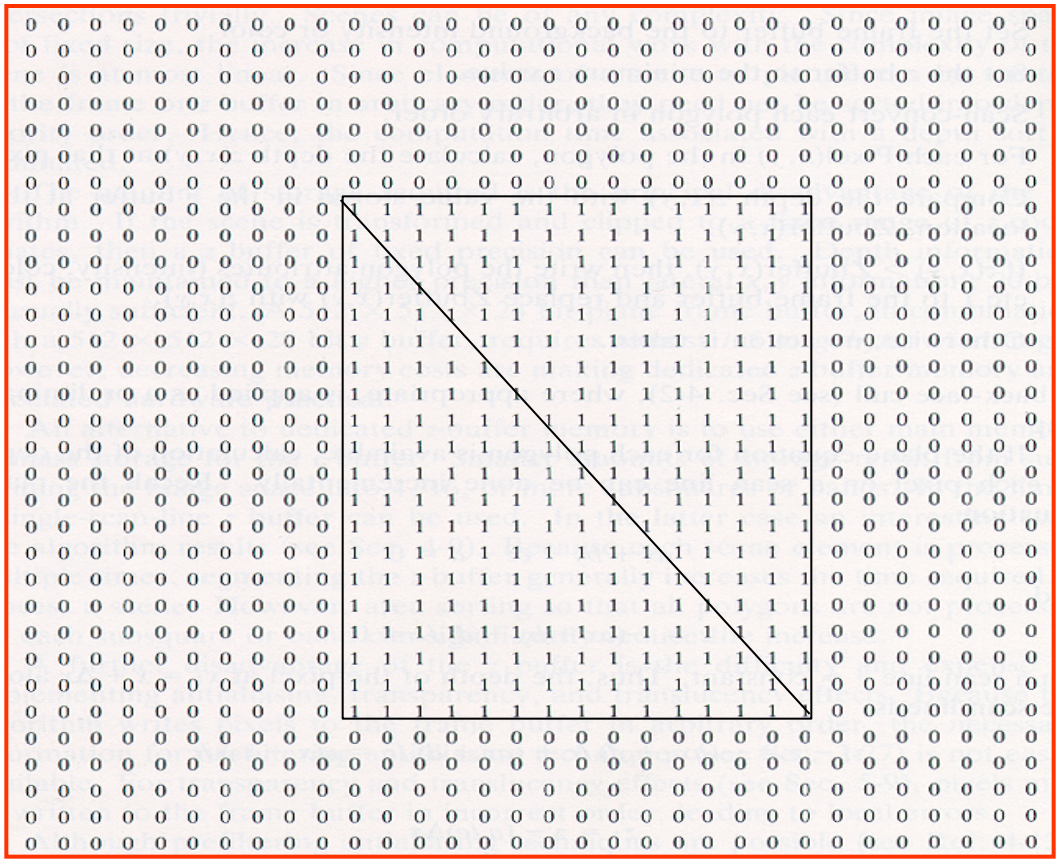


Immagine schermo

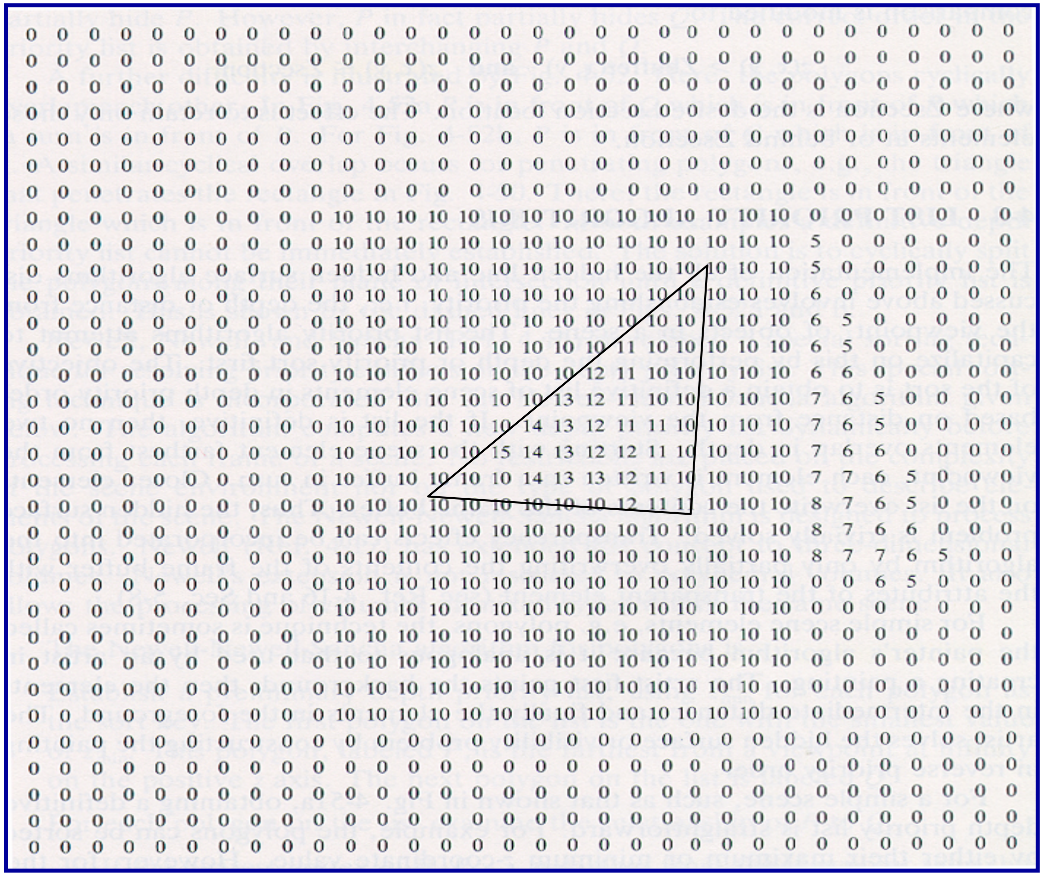
Z-Buffer dopo triangoli 1 e 2

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

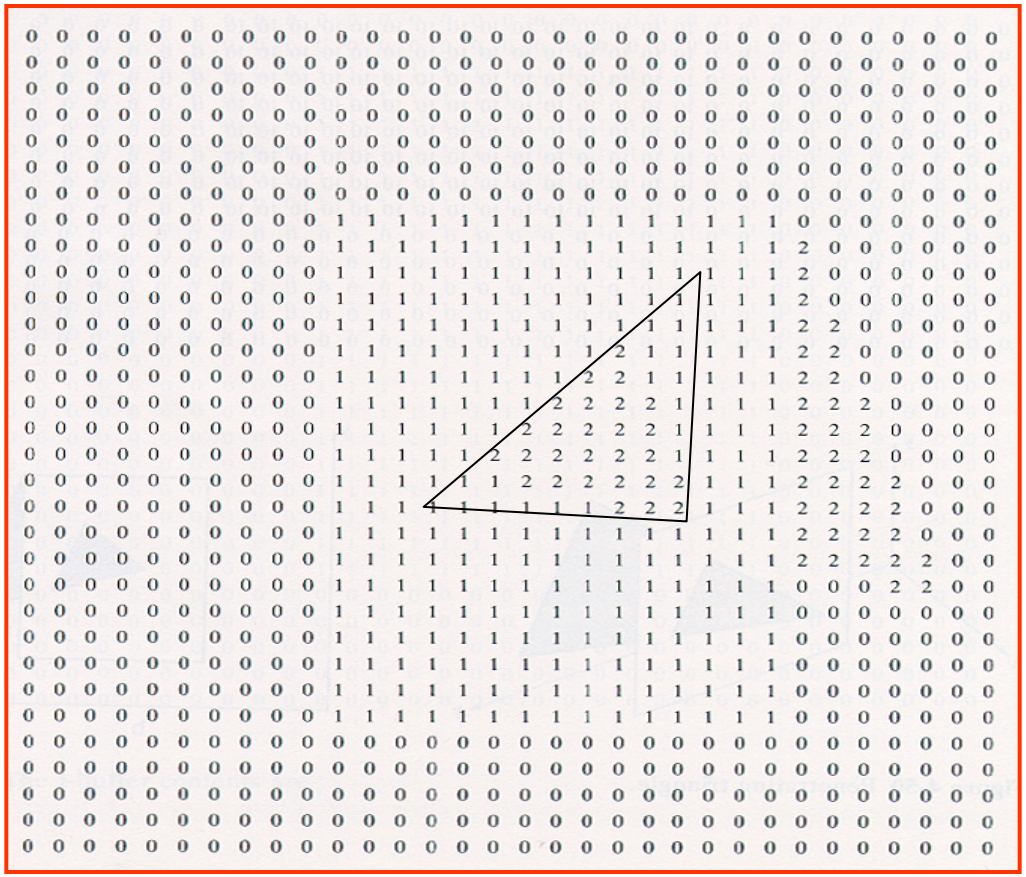
Frame Buffer dopo triangoli 1 e 2



Z-Buffer dopo il triangolo 3



Frame Buffer dopo il triangolo 3



Algoritmo Z-buffer

- Vantaggi
  - Non prevede alcun ordinamento, infatti il colore di un pixel è determinato dal punto 3D (di Ze minore) di cui lui è immagine
  - Semplice da implementare

- Svantaggi
  - Richiede spazio di memoria aggiuntivo
  - C'è ancora una sorta di ridisegno anche se solo in memoria (copia del frame buffer)

Prossimi Argomenti

Per essere in grado di implementare un algoritmo Z-buffer dovremo affrontare i seguenti argomenti:

- Rasterizzazione (Coordinate Baricentrice o Scan Conversion);
  - con colore
  - con texture
- Proiezione con correzione e Z-buffer (Depth)
- Rivisitare la Pipeline Grafica con Z-buffer

Prossimi Argomenti

Successivamente per completare le nostre conoscenze sugli algoritmi di real-time rendering dovremo sapere come funzionano:

- Algoritmi di Clipping di punti, segmenti e triangoli/poligoni piani;
- Modelli di illuminazione;
- Algoritmi di shading.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Giulio Casciola

Dip. di Matematica

giulio.casciola at unibo.it