Pipeline di Output Grafica

Algoritmi

Algoritmi di rasterizzazione 2D

Disegno di primitive grafiche 2D

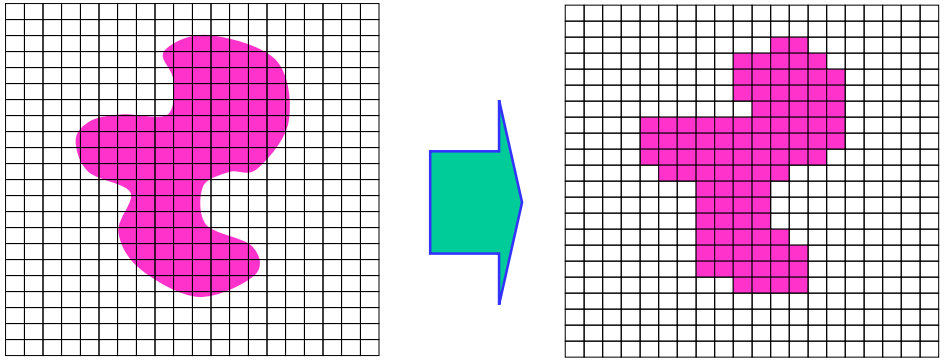
Algoritmi

- Scan conversion
  - Linee
  - Filling di Triangoli
  - Filling di Poligoni Convessi e Concavi

- Antialiasing

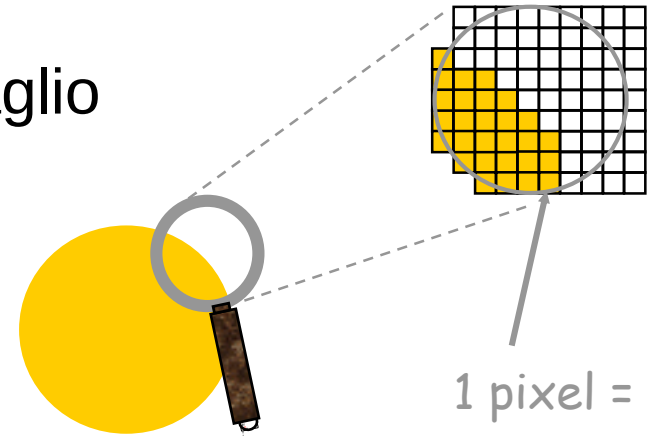
Rasterizzazione

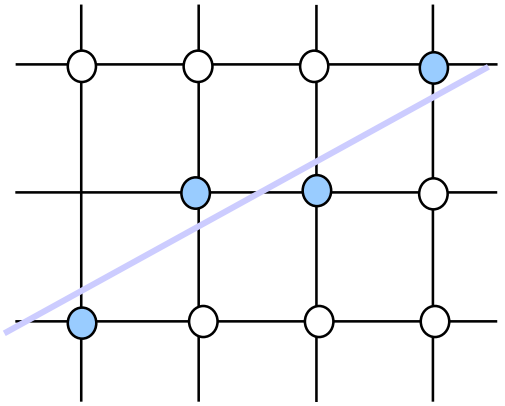
Traduzione dell'immagine 2D "continua" nell'immagine 2D "discreta" dovuta alla griglia di pixels



SCAN CONVERSION:
Tecnica/criterio con cui i pixel vengono colorati (o no) lungo i contorni delle immagini

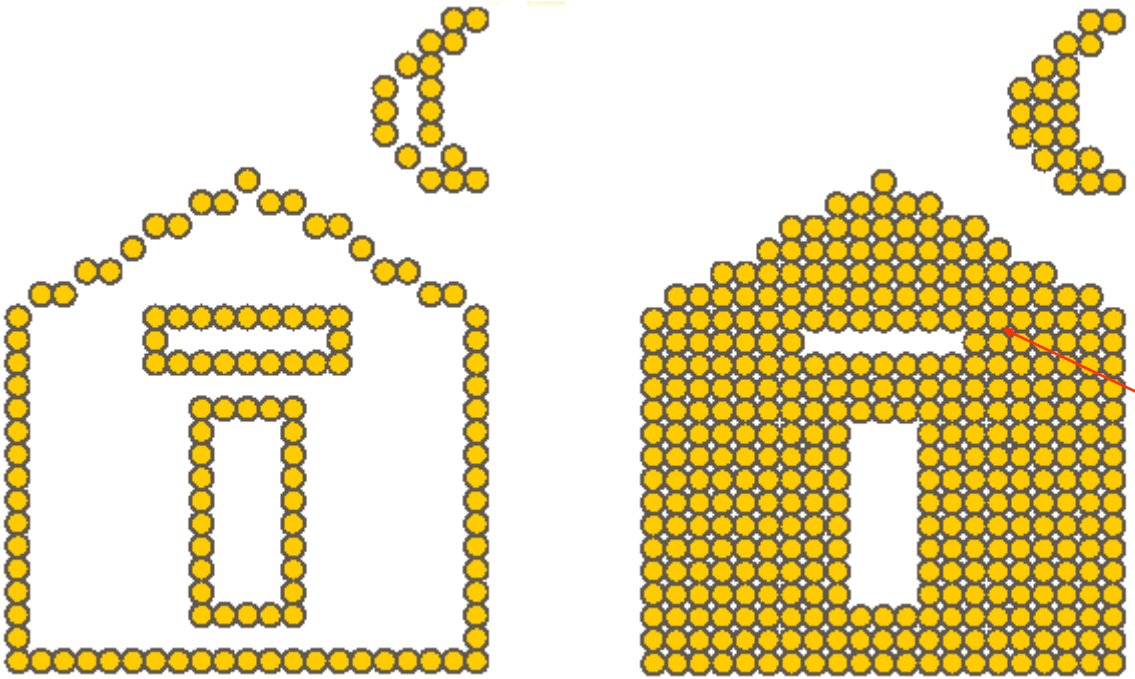
Dettaglio





Contorni e Area Filling

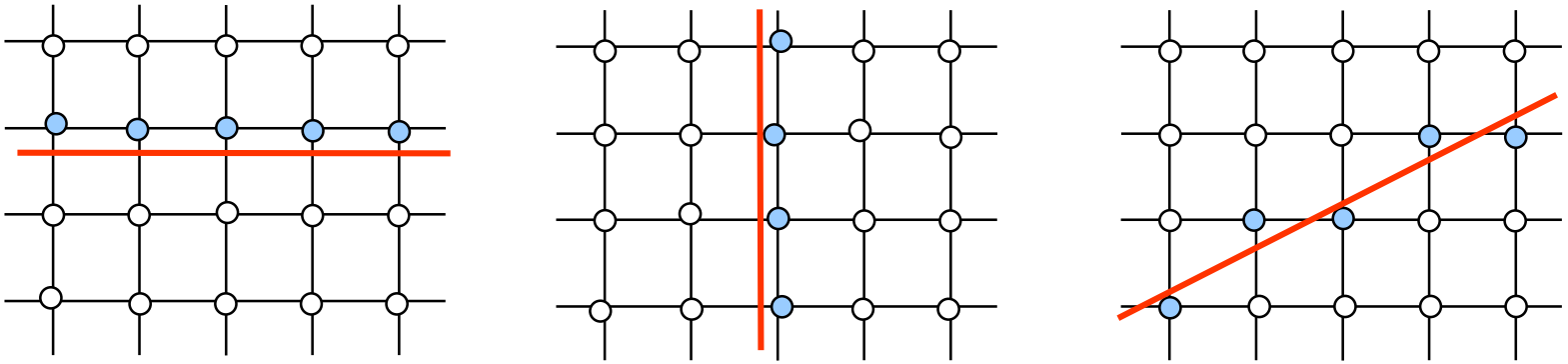
- Con tecnologia raster si possono disegnare su display grafico:
  - oggetti rappresentati dai soli contorni
  - oggetti “pieni”



Griglia discreta e distanza tra pixels non (o poco) percepita dalla retina a seconda della risoluzione

Scan Conversion: Linee

- Un algoritmo di scan-conversion per LINEE calcola le coordinate dei pixel che giacciono o “sono vicini” ad una linea ideale, infinitamente sottile



- La sequenza di pixel deve:
  - giacere “il più vicino possibile” alla linea ideale
  - essere il più diritta possibile

Scan Conversion: Linee

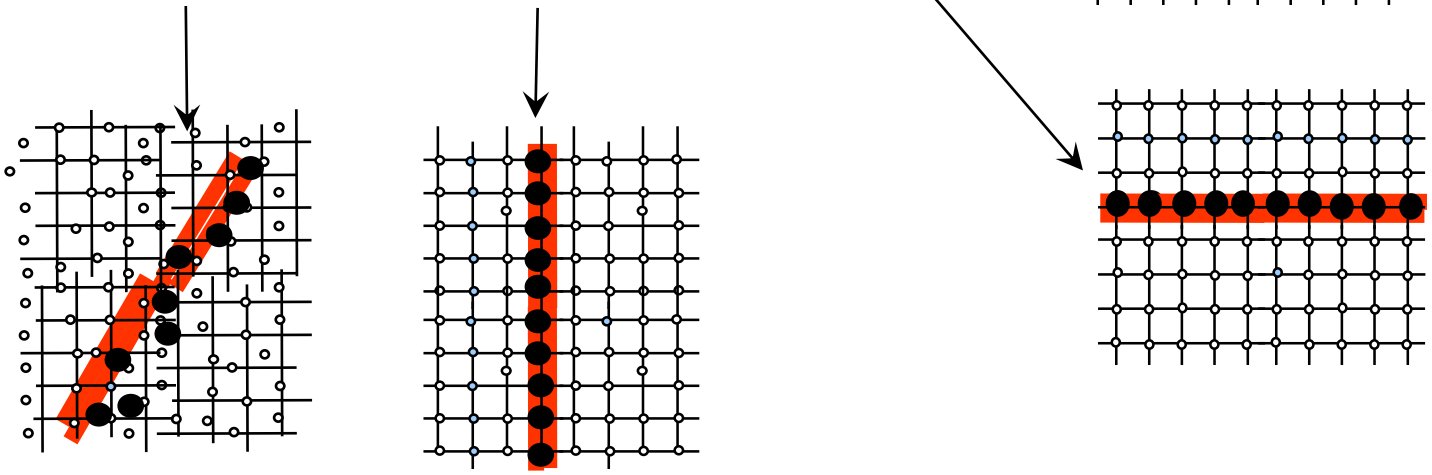
Algoritmi di Scan Conversion

- Algoritmo incrementale di Base (**Bresenham line algorithm**)
- Algoritmo Midpoint Line

Scan Conversion: Linee

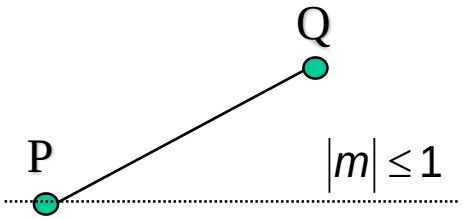
Proprietà di base

- consideriamo una linea larga 1 pixel
- equazione linea: ax+by+c=0 → y=mx+p (obliqua/orizzontale) o x=const (verticale)
- CASO $|m| \leq 1$
  - viene acceso un pixel di ciascuna colonna
- Altrimenti (CASI $|m| > 1$, $o\ m \to \infty \Leftrightarrow$ x=const)
  - viene acceso un pixel di ciascuna riga



Scan Conversion: Linee

Algoritmo incrementale di Base



OBIETTIVO: Rasterizzazione segmento $(x_P,y_P) \rightarrow (x_Q,y_Q)$

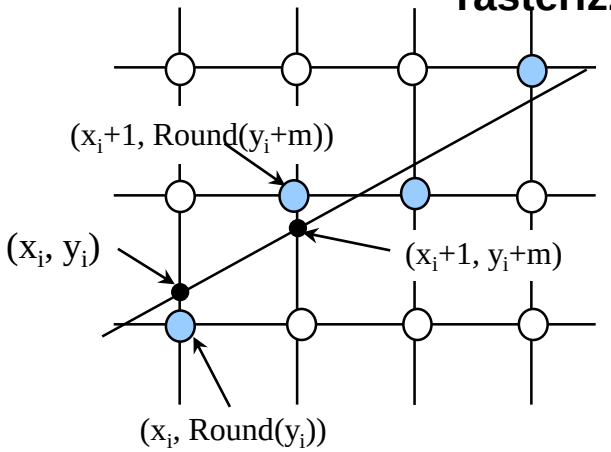
P e Q a coordinate intere (pixel) note

CASO $|m| \leq 1$

→ 1 ed 1 solo pixel per colonna

- Partenza: $x_0=x_P$ e $y_0=y_P$
- Calcola per ogni i=0,1,2,…
- $x_{i+1} = x_i + 1$ (1=Δx size pixel)
- $y_{i+1} = y_i + m$ ⇐ $y_{i+1} = mx_{i+1} + p = m(x_i+\Delta x)+p = y_i + m\Delta x = y_i + m$
- arrotonda y_{i+1} → Round(y_{i+1}) che è intero
- accendi il pixel di coordinate (x_{i+1}, Round(y_{i+1}))

Linea rasterizzata



(CASI $|m| > 1$, *o* $m \rightarrow \infty \Leftrightarrow$ x=const: ribaltando il ruolo di x ed y)

Scan Conversion: Linee

Algoritmo Incrementale di Base - pseudocodice
(detto anche Digital Differential Analyser, DDA)

```
Line (int x0, int y0, int x1, int y1, int value)        // HP: x0<x1
{       int  x; float dx, dy, y, m;
        if ( x1 EQUAL x0 ) // vertical line
        {    for ( y = min(y0,y1); y ≤ max(y0,y1); y++ )  WritePixel (x0, y,value}    }
        else
        {
           // x1 NOT EQUAL x0 i.e. oblique line
           dy = y1 - y0;     dx = x1 - x0;     m = dy/dx;
           if ( -1 ≤ m ≤ 1 ) {
                    y = y0;
                    for ( x = x0; x ≤ x1; x++ )
                    {       WritePixel (x, Round(y), value);        /* set pixel to value */
                            y+= m;
                    }
           }
            else // m>1 or  m<-1
           {    x = x0;
                for ( y = min(y0,y1); y ≤ max(y0,y1); y++ )
                 {        WritePixel (Round(x), y, value);        /* set pixel to value */
                          x+= 1/m;
                 }
           }
        }
}
```

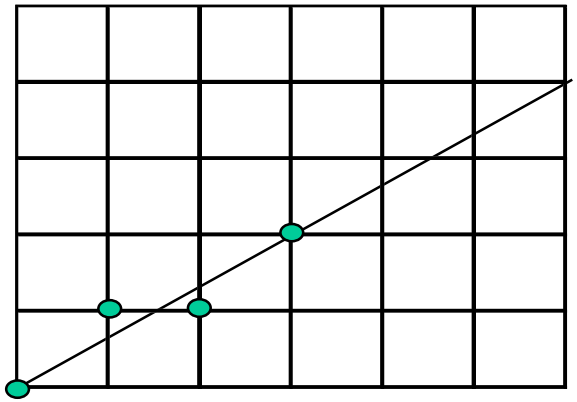
Scan Conversion: Linee

Esempio applicando DDA

- Tracciamo un segmento dal punto A=(0,0) a B=(6,4)
- Equazione retta: y = 2/3 x

→ m = 2/3

passo	x	y	attiva
1	0	0	(0,0)
2	1	2/3	(1,1)
3	2	4/3	(2,1)
4	3	2	(3,2)



Considerazioni

- *y* ed *m* sono valori reali, poichè la pendenza è calcolata come frazione
- il calcolo di *Round(y)* per ottenere un valore intero è dispendioso

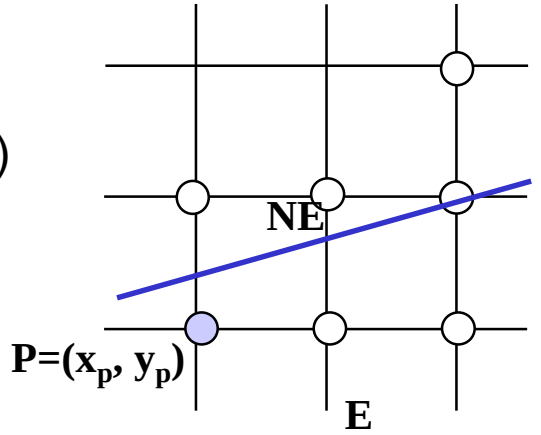
Scan Conversion: Linee

Algoritmo di Midpoint Line (1/3)

- è un algoritmo incrementale: valori (x_i, y_i) → valori (x_{i+1}, y_{i+1})
- incrementa di 1 una delle variabili
- l'altro incremento viene calcolato minimizzando l'errore, cioè la distanza tra la linea ideale ed il punto sulla griglia più vicino
- DIFFERENZA rispetto a DDA: opera sugli interi e non sui reali

…come opera

- OBIETTIVO: rasterizzare un segmento (x_0, y_0)→(x_1, y_1)
- equazione retta: $F(x,y)=ax+by+c=0$ (ipotesi: $0 \le m \le 1, a>0$)
- IPOTESI: già selezionato il pixel P (x_p, y_p)
- dobbiamo decidere se selezionare pixel E (incremento a destra) oppure NE (incremento a destra + uno verso l'alto)



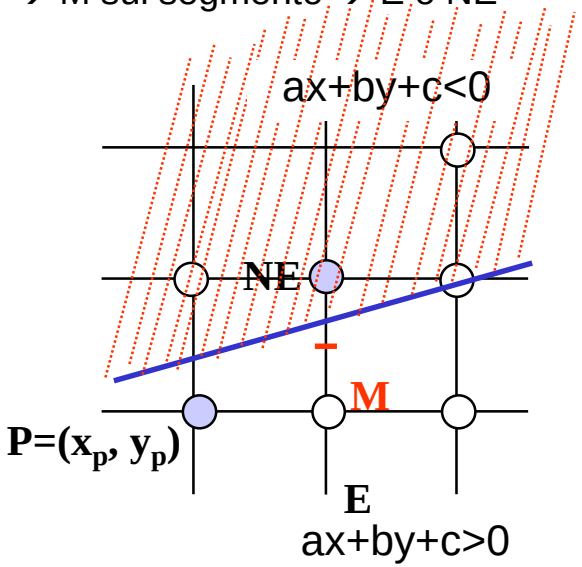
Scan Conversion: Linee

Algoritmo di Midpoint Line (2/3)

- sia M il punto di mezzo fra i punti E e NE
- dobbiamo decidere se M giace al di sopra o al di sotto della linea
- se M giace sopra la linea, si sceglie il pixel E
- se M giace al di sotto della linea, si sceglie il pixel NE
- "sopra"/"sotto" deciso sulla base di una variabile di decisione: d= F(M)=F(x_P+1,y_P+1/2)

  (1) d<O → M sopra → scelto E; (2) d>O → M sotto → scelto NE; (3) d=O → M sul segmento → E o NE

- VANTAGGIO: le operazioni usate nell'algoritmo sono *addizioni* computazionalmente poco costose
  vediamo meglio il perchè nella prossima slide



Scan Conversion: Linee

Algoritmo di Midpoint Line (3/3)

E' possibile valutare il valore della variabile di decisione d in modo incrementale

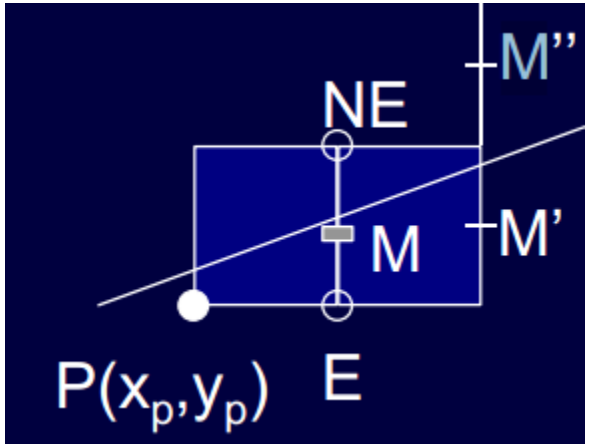
- Se si sceglie **EST**

$d_{new}= F(M')=F(x_P+2,y_P+1/2)$

$d_{new}= a(x_P+2)+b(y_P+1/2)+c$

$d_{old}= a(x_P+1)+b(y_P+1/2)+c$

$d_{new-}\ d_{old}= a$



- Se si sceglie **NORDEST**

$d_{new}= F(M'')=F(x_P+2,y_P+3/2)$

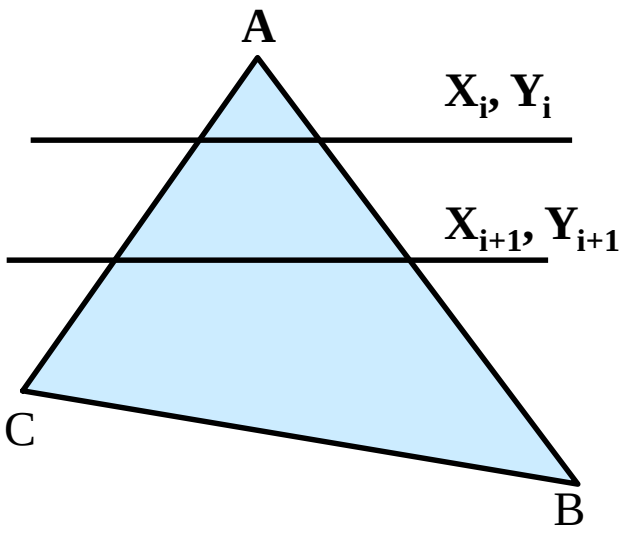
$d_{new}= a(x_P+2)+b(y_P+3/2)+c$

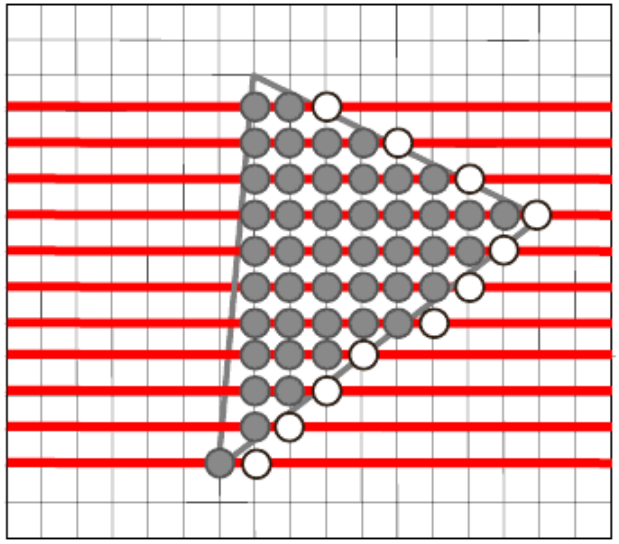
$d_{old}= a(x_P+1)+b(y_P+1/2)+c$

$d_{new-}\ d_{old}= a+b$

Per eliminare il problema della divisione per 2 nelle espressioni precedenti
si usa come funzione di valutazione:
F(x,y)=2(ax+by+c) risultato: i valori dei parametri calcolati sopra si moltiplicano per 2

Scan Conversion: Triangoli





Algoritmo di base per scan-line orizzontale:

- Creare la lista dei lati
- Ripetere
  - trovare i punti di intersezione tra:
    asse parallelo all'asse x (scan-line)
    e i lati del triangolo
  - se non null, disegnare/colorare la linea di pixel interni più prossimi agli estremi di "intersezione"

Calcolo delle intersezioni: sistema lineare

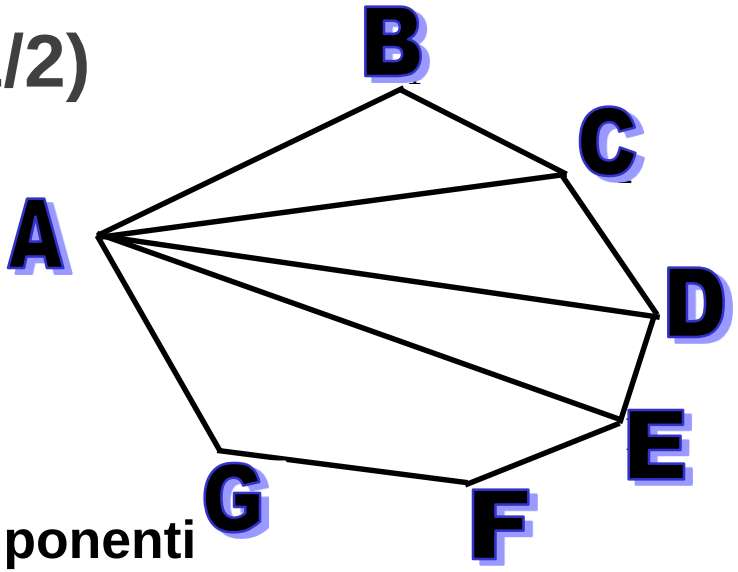
- y = const (linea di scansione)
- y = mx + p (lato del triangolo)

In particolare:

- Accendi pixel lungo la riga di scansione dal primo interno (incluso) al primo esterno (escluso) (vedi algoritmo più avanti per filling di poligoni)

Scan Conversion: Poligoni

Poligoni Convessi (1/2)

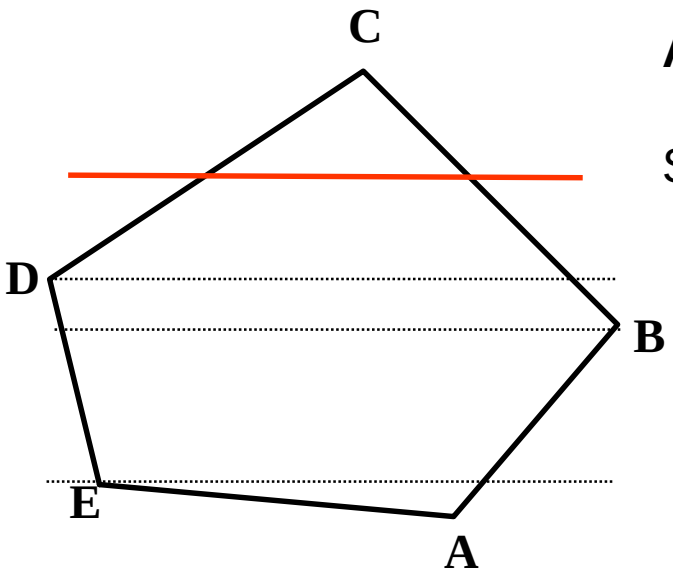


Algoritmo: per triangoli componenti

- trovare il vertice più a sinistra (con x minore), detto A
- trovare il triangolo formato da A e dai 2 vertici adiacenti B e C
  - attivare pixel del triangolo ABC (scan conversion triangolo)
  - escludere il triangolo ABC dal poligono
  - ripetere per triangolo successivo (ACD)
  - … fino ultimo vertice precedente A

Scan Conversion: Poligoni

Poligoni Convessi (2/2)



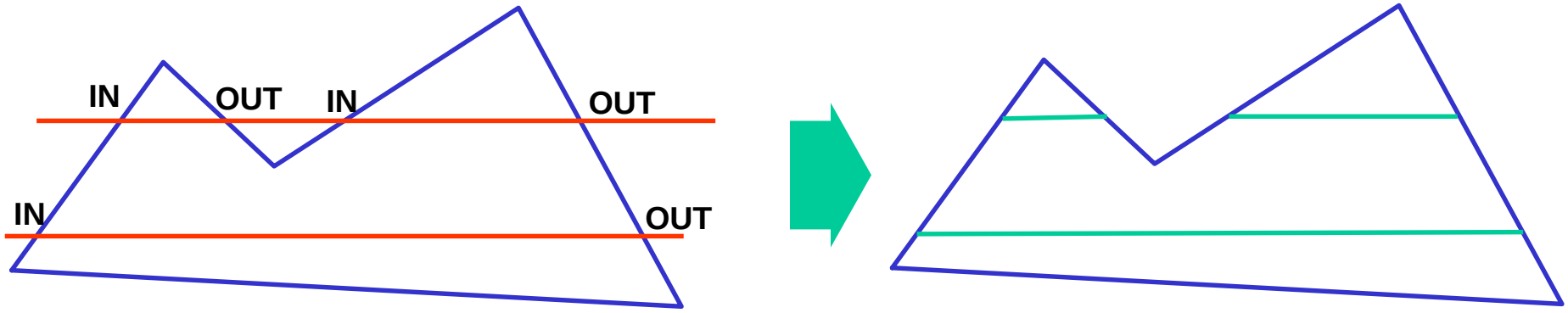
Algoritmo per scan-line orizzontale:

Simile al caso dei triangoli

Scan Conversion: Poligoni

Poligoni Concavi

- Scopo:
  - riempire l'area del poligono con pixel
- Procedimento:
  - tracciare una linea per ogni riga di pixel sullo schermo (scan line)
  - calcolare l'intersezione della linea con gli spigoli del poligono
  - accendere linee di pixel da ogni punto IN ad ogni punto OUT (regola di parità)
  - Si utilizzano strutture dati edge table (ET), active edge table (AET), (vedi algoritmo scan-line per la rimozione di superfici nascoste più avanti)

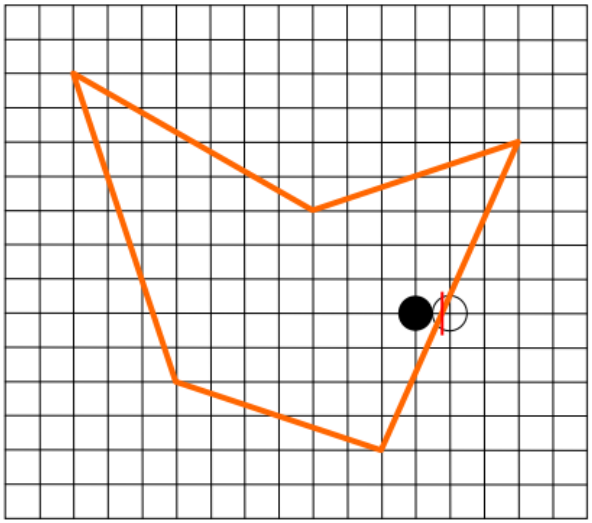


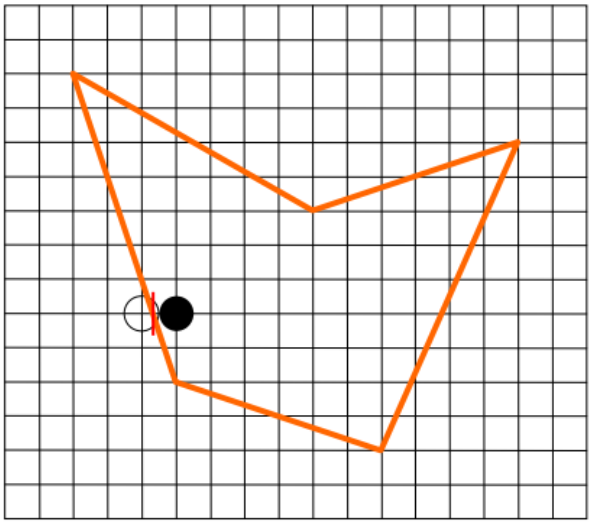
  - Esistono alcune regole specifiche e casi particolari

Scan Conversion: Poligoni

Regole specifiche

1) Se si incontra l'intersezione provenendo da dentro il poligono si arrotonda il valore dell'intersezione all'intero inferiore. Se si incontra l'intersezione provenendo da fuori il poligono si arrotonda all'intero superiore.



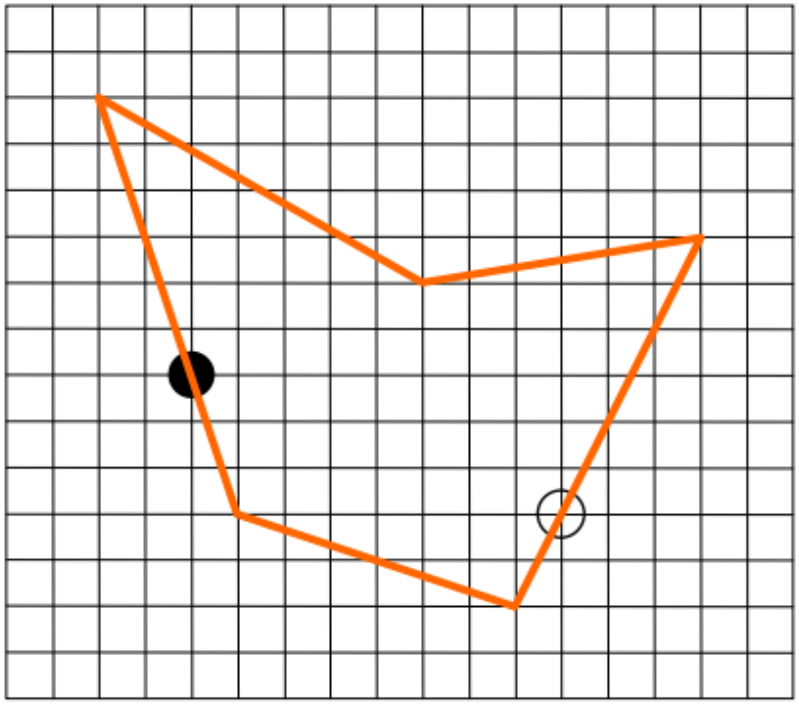


Pertanto i pixel (che hanno sempre coordinata intera) a sinistra dell'intersezione sinistra (arrotondata) ed a destra dell'intersezione destra (arrotondata) non vengono mai attribuiti al poligono.

Scan Conversion: Poligoni

Regole specifiche

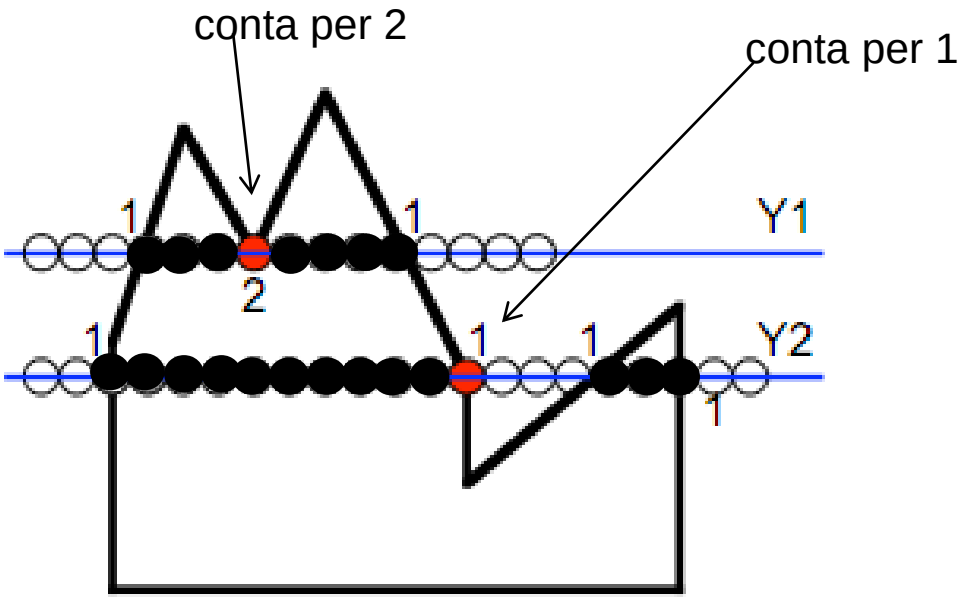
2) Per evitare conflitti di attribuzione di spigoli condivisi, si definisce che un'intersezione a coordinate intere all'estremo sinistro della scan-line di pixel è interna al poligono, all'estremo destro è esterna.



Scan Conversion: Poligoni

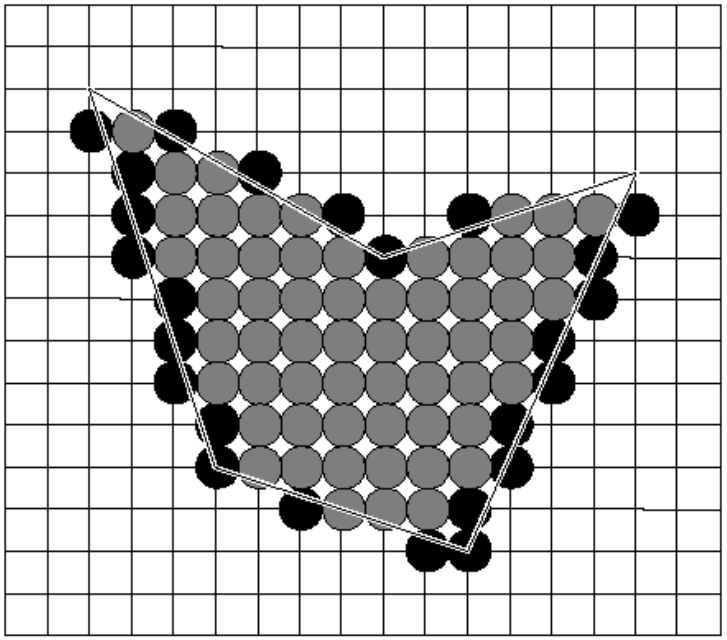
Regole specifiche

3) Caso particolare di intersezione della scan-line in un vertice della primitiva.
Se i vertici adiacenti stanno tutti dalla stessa parte rispetto alla scan-line conta il vertice per ‘’2”.
Se i vertici adiacenti stanno da parti diverse rispetto alla scan-line
conta il vertice per ‘’1”.



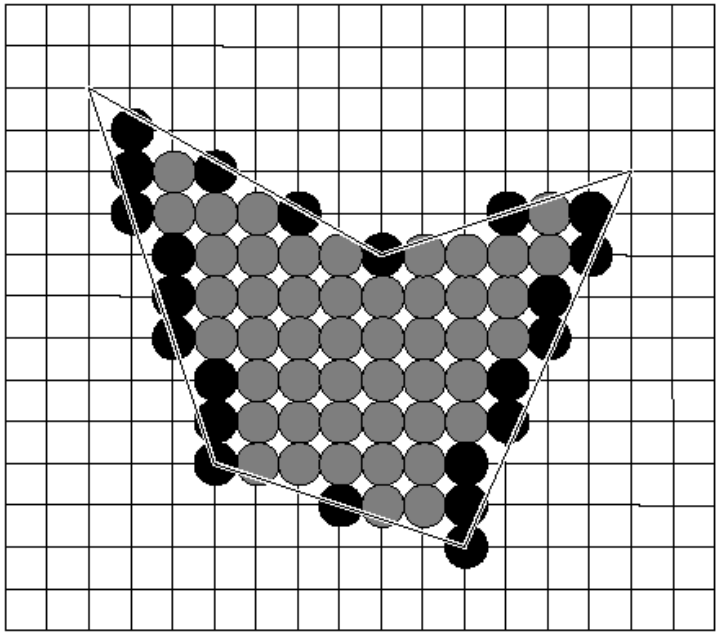
Scan Conversion: Poligoni

Differenze…



Scan conversion dei contorni, poi filling di pixel “interni”

meno realistico!
Scan conversion degli spigoli può introdurre pixel esterni che invadono poligoni adiacenti



Filling specifico con gli algoritmi visti in precedenza

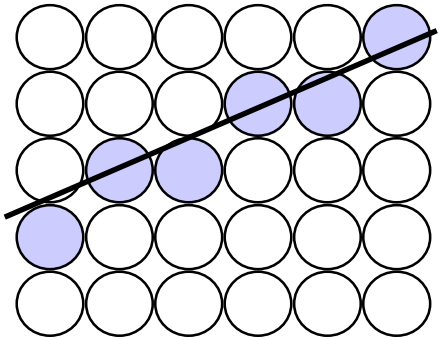
più realistico!
non considera pixel esterni

Algoritmi di rasterizzazione 2D:

Anti-Aliasing

Antialiasing

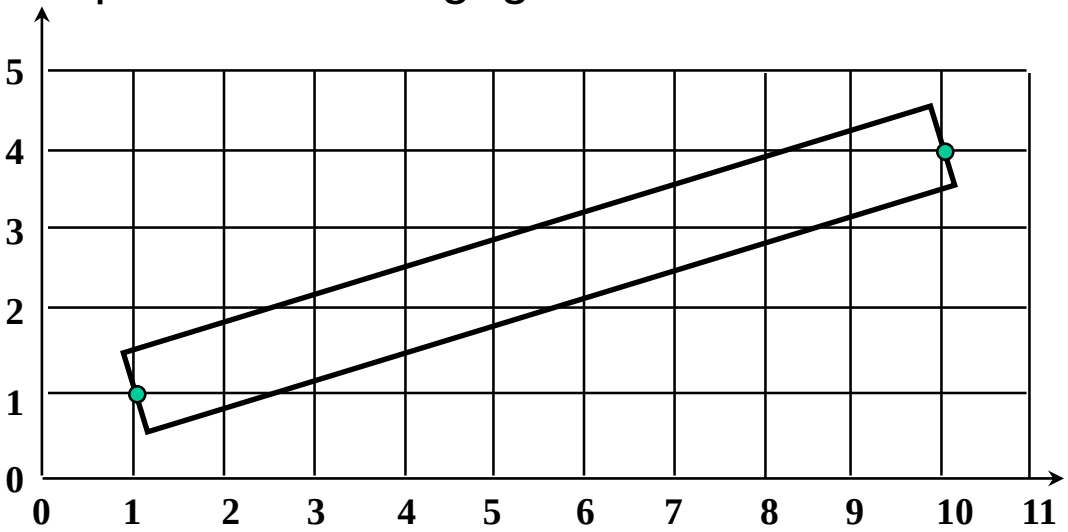
- Scan conversion di linee “brutale” produce **effetto di aliasing**
- Aliasing: effetto di salto/discontinuità tra pixel dovuto alla natura discreta del frame buffer
- Per griglie cartesiane di pixel (monitor): effetto maggiore per linee oblique, specialmente per *m* di valore assoluto molto diverso da 1



Antialiasing: tecniche che riducono od eliminano l’aliasing con l’obiettivo di migliore percezione “continua” della linea/curva

Tecnica Unweighted Area Sampling (1/2)

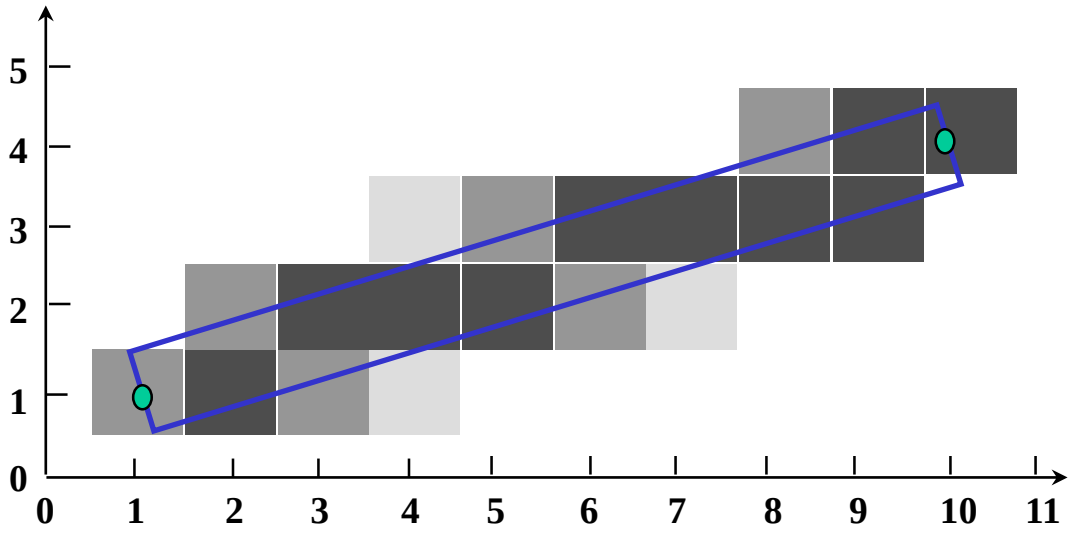
- **un segmento è pensato come un rettangolo** di un certo spessore che copre una porzione della griglia



- si assume che i pixel formino un array di piastrelle quadrate che coprono lo schermo …
- … e che una linea contribuisca all'intensità di ciascun pixel di una certa quantità proporzionale alla percentuale della piastrella del pixel che essa copre

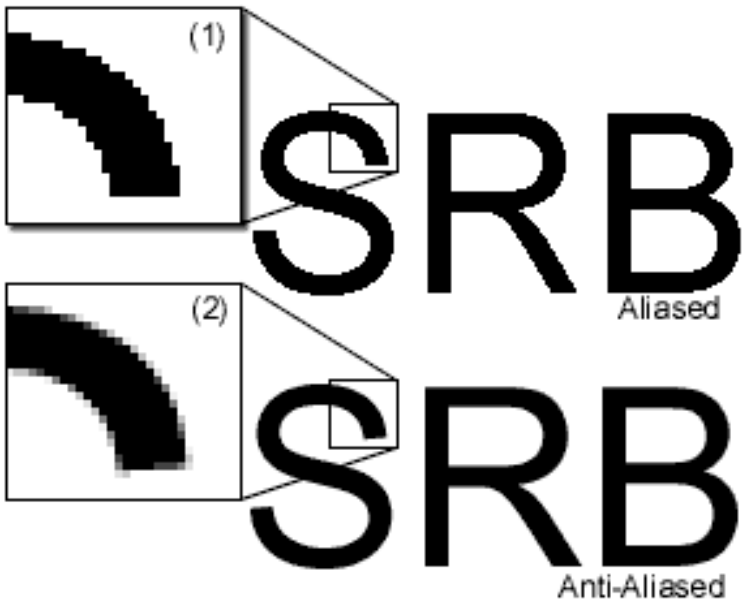
Tecnica Unweighted Area Sampling (2/2)

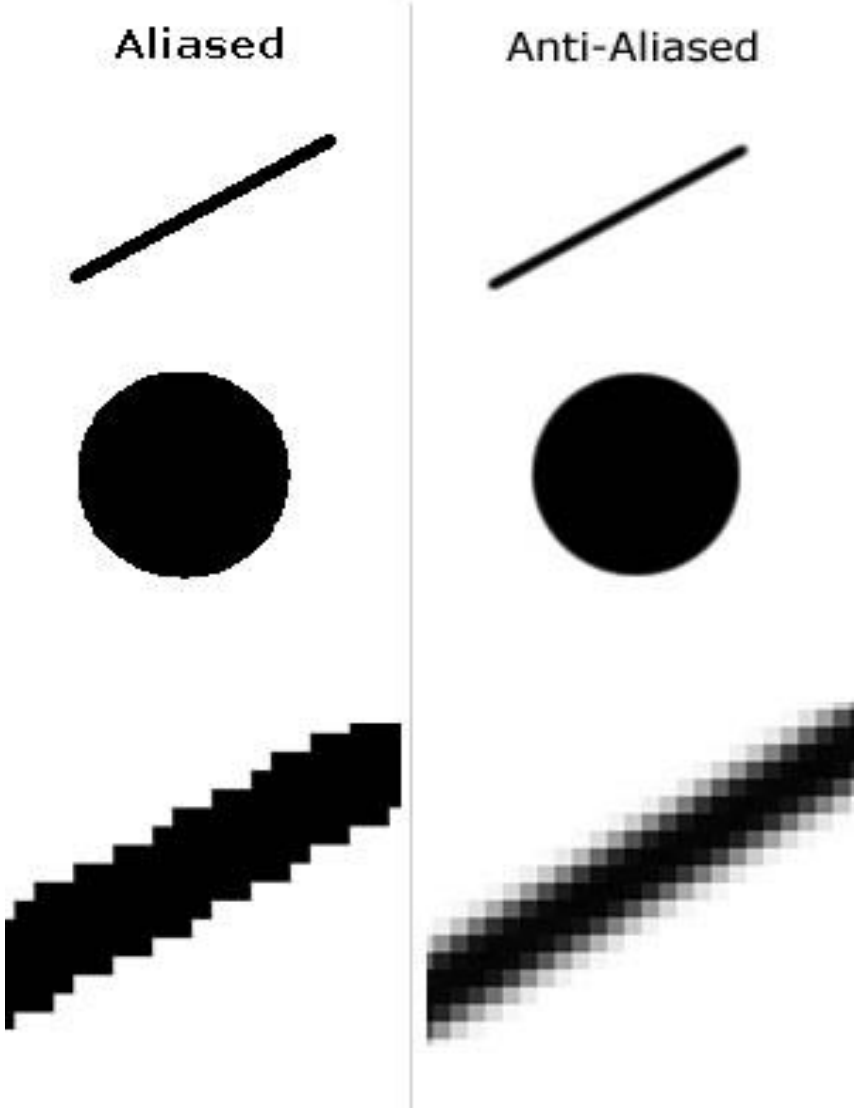
- definendo l'intensità di colore di ogni pixel in modo proporzionale alla quantità di area di esso ricoperta dalla linea si smussa l'immagine della linea



Esempi

Differenze (1/2)

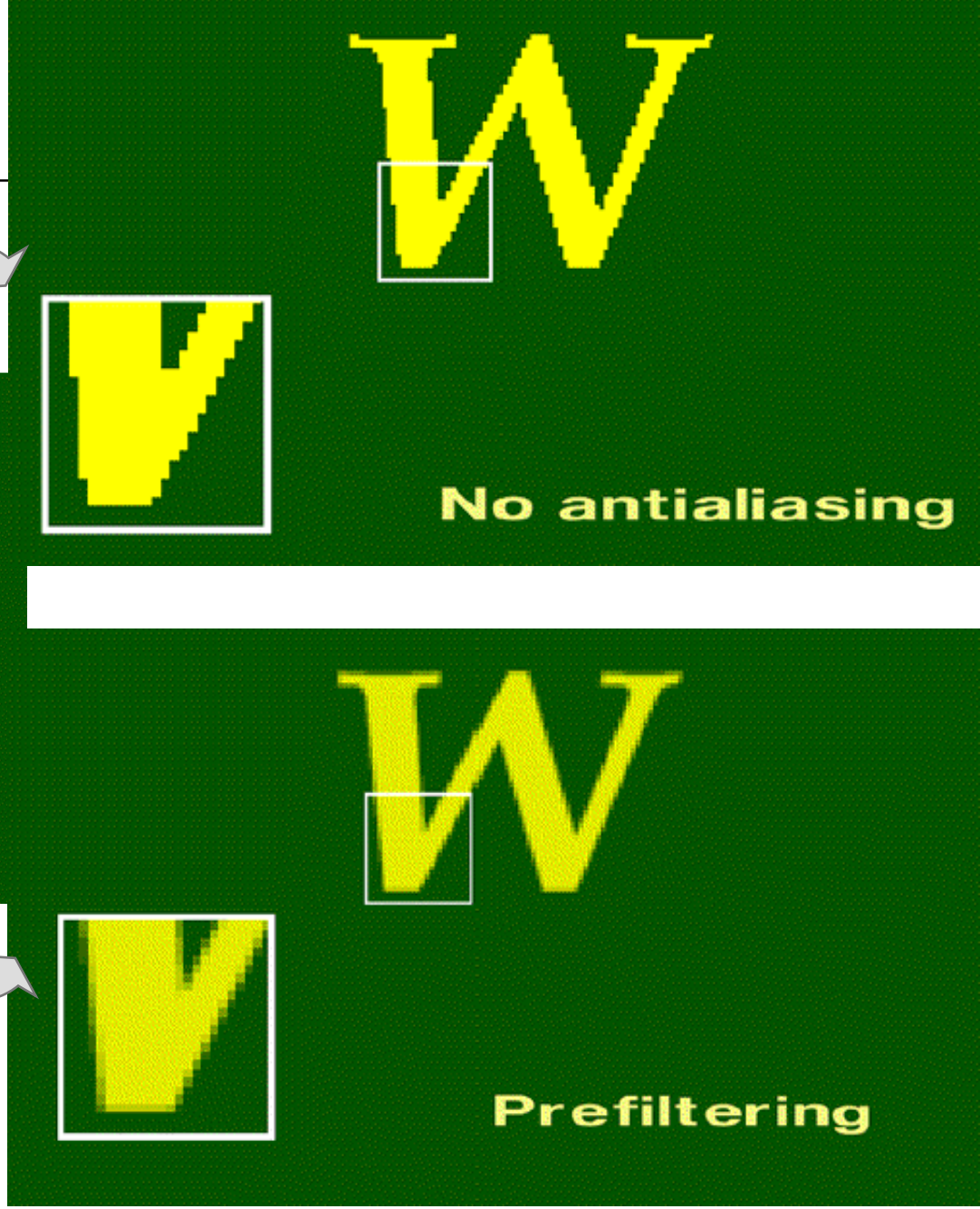




Linee orizzontali e verticali non necessitano di antialiasing

Esempi

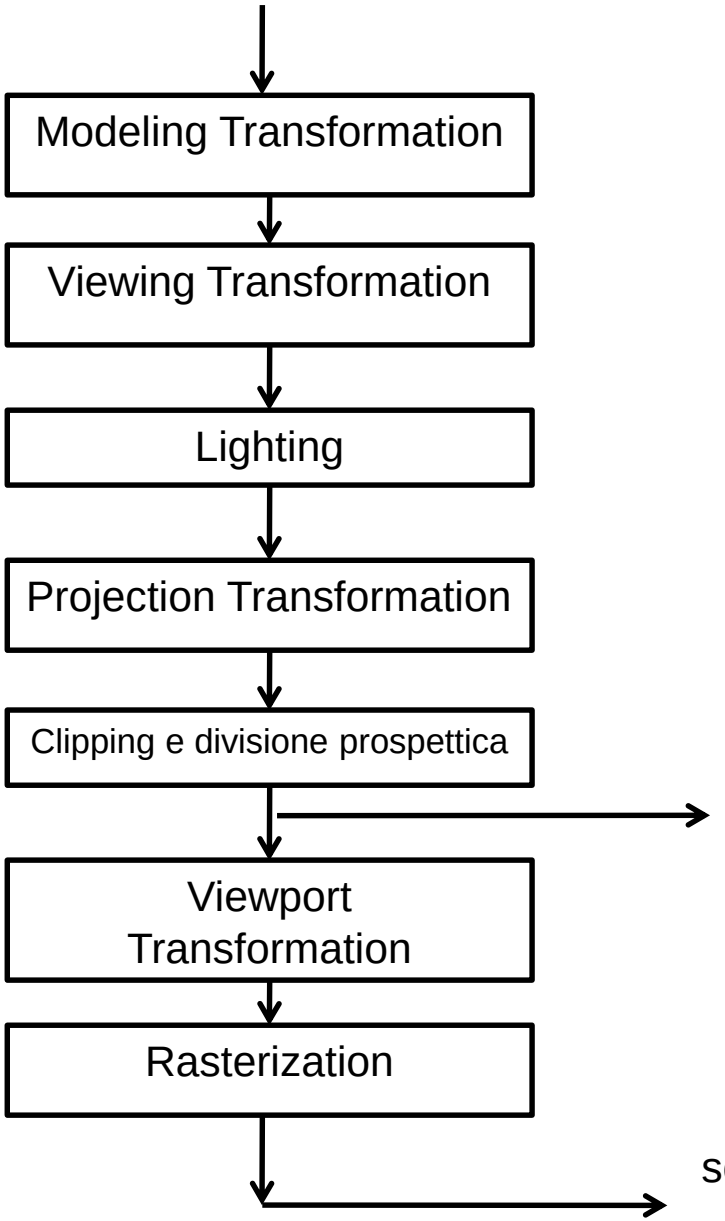
Differenze (2/2)



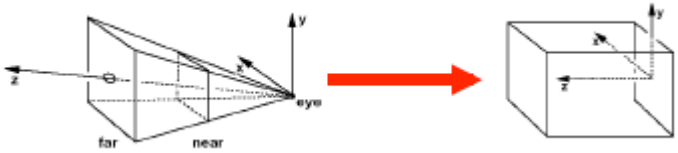
Clipping 2D e 3D

Eliminazione di superfici nascoste

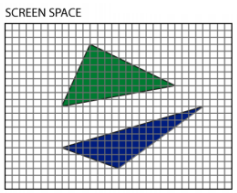
Schema completo della pipeline grafica



il clipping avviene in coordinate omogenee.
Successivamente avviene la divisione prospettica e si ottengono le 3D Normalized device coordinates (NDC) in un volume di vista canonico
(cubo con x,y,z comprese tra -1 e +1)



screen space

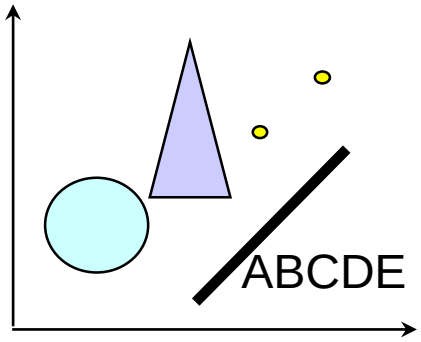


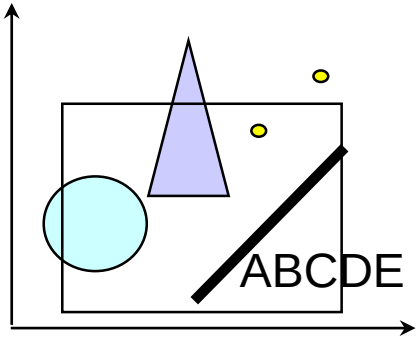
Clipping

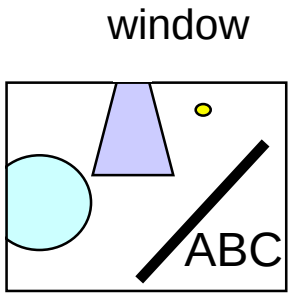
selezione della parte di scena da visualizzare

suddivide ciascuna entità del modello della scena in parti *visibili* e *non visibili*

eseguito prima della scan conversion (rasterizzazione): algoritmo di scan conversion viene applicato solo sulla porzione di immagine da rappresentare







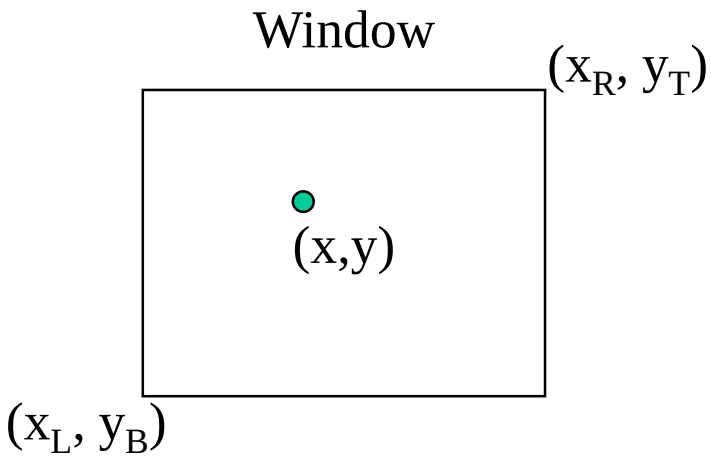
Clipping 2D

Distinguiamo tra….

- Point Clipping
- Line Clipping
- Polygon Clipping

Point Clipping

- Si valutano le coordinate x,y del punto rispetto ai punti “minimale” e “massimale” della finestra



$$x_L \leq x \leq x_R$$

$$y_B \leq y \leq y_T$$

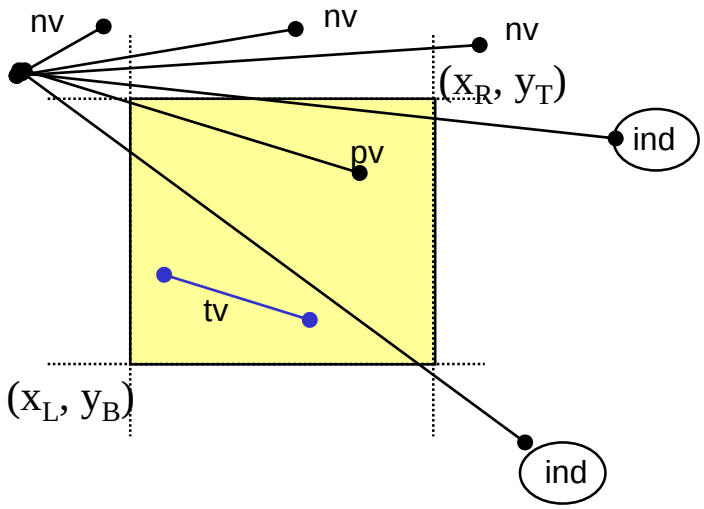
Line clipping (1/8)

- Si valutano le coordinate x,y dei **due punti estremi** del segmento rispetto ai bordi della finestra
- Per i segmenti parzialmente visibili si determinano le parti visibili

Fasi di un algoritmo di line clipping:

- Fase 1: individua “tipi” di segmenti rispetto alla window.
  - determinati:
    - totalmente/parzialmente visibili
      (almeno un vertice nella finestra)
    - non visibili
      x_1 and $x_2 < x_L$
      x_1 and $x_2 > x_R$
      y_1 and $y_2 < y_B$
      y_1 and $y_2 > y_T$
  - indeterminati
    - non visibili/parzialmente visibili
      (entrambi i vertici fuori dalla finestra)

TIPI DI LINEE



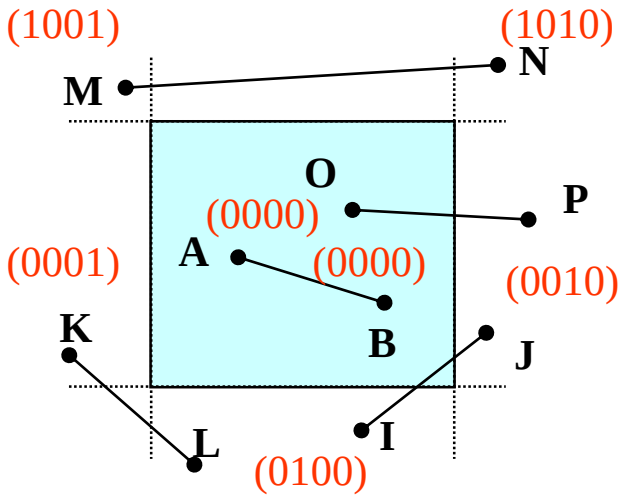
- Fase 2: effettua il clipping dei segmenti calcolati nel passo precedente calcolando le intersezioni con i lati della window

Line clipping (2/8)

Algoritmo di Cohen Sutherland

- **PASSO 1: viene assegnato un codice a 4 bit ad ogni estremo del segmento**
  - Bit 1 = 1 se estremo del segmento è **strettamente** sopra la window
  - Bit 2 = 1 se estremo del segmento è **strettamente** sotto la window
  - Bit 3 = 1 se estremo del segmento è **strettamente** a destra della window
  - Bit 4 = 1 se estremo del segmento è **strettamente** a sinistra della window

1001	1000	1010
0001	window 0000	0010
0101	0100	0110



Line clipping (3/8)

Algoritmo di Cohen Sutherland

- **PASSO 2: *Bitwise logical AND* viene utilizzato per determinare a quale categoria appartiene il segmento**

ESEMPIO MN

Estremo 1		Estremo 2		Bitwise logical AND
1	and	1	true	1
0	and	0	false	0
0	and	1	false	0
1	and	0	false	0

– **CLASSIFICAZIONE DEI SEGMENTI:**

- bitwise logical AND uguale a 0000

  -entrambi gli estremi hanno codice 0000 → interamente visibile, **TERMINA ALGORITMO**

  -un estremo ha codice 0000, l'altro diverso da 0000 → parzialmente visibile

  -i due estremi hanno entrambi codice diverso da 0000 → indeterminato (parz. vis. o inv.)

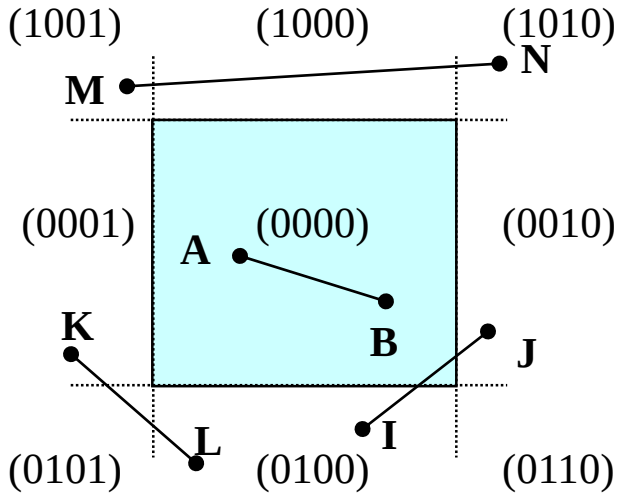
- bitwise logical AND diverso da 0000 → invisibile, **TERMINA ALGORITMO**

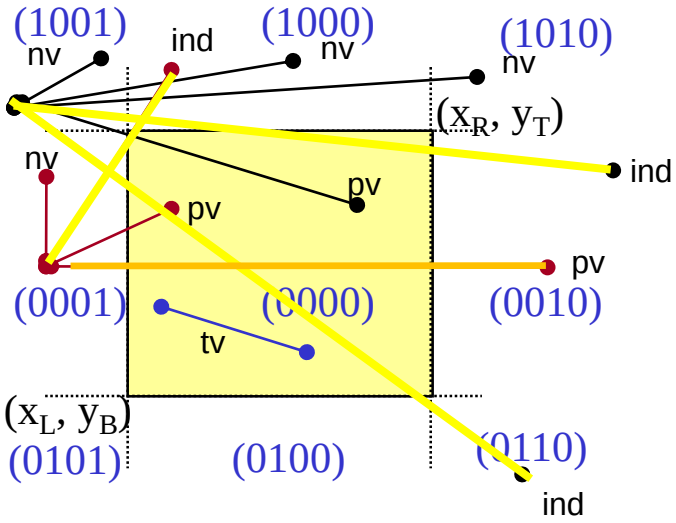
Line clipping (4/8)

Algoritmo di Cohen Sutherland

Esempi

Segmento	Codici	Estremi	Bitwise Logical AND	Categoria
AB	0000	0000	0000	Visibile
KL	0001	0100	0000	Indeterminato
IJ	0100	0010	0000	Indeterminato
MN	1001	1010	1000	Invisibile





Casi indeterminati:

possono essere sia invisibili, che parzialmente visibili

Line clipping (5/8)

- **PASSO 3: per i segmenti parzialmente visibili o indeterminati occorre determinare le porzioni visibili**

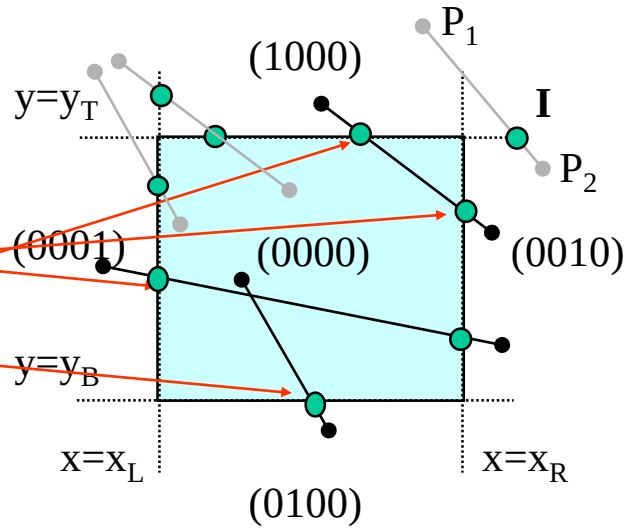
a) almeno uno dei due estremi si trova fuori dalla finestra, sia esso P_1

b) si esaminano i bit del codice di P_1 nell'ordine **bit4, bit3, bit2, bit1**

c) per il primo bit trovato che vale "1" si trova il punto di intersezione **I** del segmento P_1P_2 con la corrispondente linea di bordo della finestra (vedi immagine sotto)

d) sostituisci P_1 con **I** (si accorcia quindi il segmento) e ripeti l'algoritmo di Cohen Sutherland dal passo 1 con il segmento rimanente, finchè al passo 2 tale segmento non diventa interamente visibile o invisibile

- se Bit4=1 → intersezione con $x=x_L$
- se Bit3=1 → intersezione con $x=x_R$
- se Bit2=1 → intersezione con $y=y_B$
- se Bit1=1 → intersezione con $y=y_T$



Attenzione: intersezioni non è detto che siano sul contorno finestra!

Line clipping (6/8)

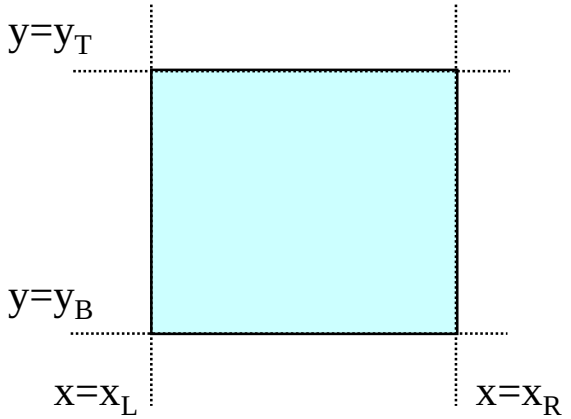
Calcolo delle intersezioni

- per trovare il punto di intersezione è possibile risolvere il sistema delle equazioni cartesiane del segmento e delle rette che delimitano la window
- **Sistema di equazioni composto da:**
  - Equazione parametrica del segmento con estremi $P_1(x_1, y_1)$ e $P_2(x_2, y_2)$

    $x = x_1 + t\,(x_2 - x_1)$

    $y = y_1 + t\,(y_2 - y_1) \qquad 0 \leq t \leq 1$

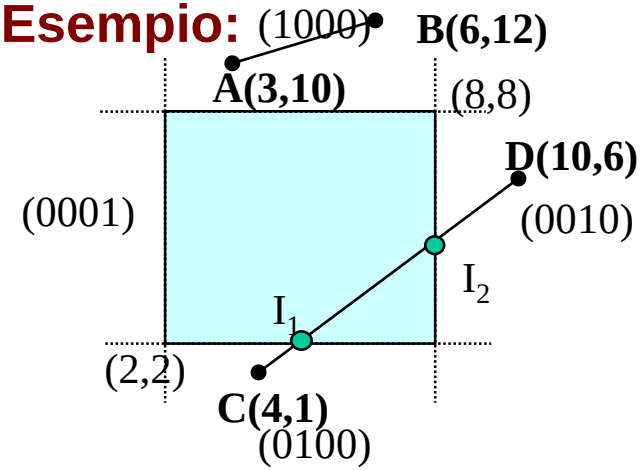
  - Equazione di una delle rette contenente i lati della window
    - $x = x_L$
    - $x = x_R$
    - $y = y_B$
    - $y = y_T$



Line clipping (7/8)

Calcolo delle intersezioni

Esempio:



			Bitwise logical AND	
AB	A(3,10)	1000	1000	Invisibile
	B(6,12)	1000		
CD	C(4,1)	0100	0000	Indeterminato
	D(10,6)	0010		

– clipping segmento CD

$x = x_1 + t(x_2 - x_1) = 4 + 6t$

$y = y_1 + t(y_2 - y_1) = 1 + 5t$

C: bit2 = 1 → interseca $y = y_B = 2$

$y = y_1 + t(y_2 - y_1) = 1 + 5t$

$t = 1/5 = 0.2 \rightarrow I_1 = (5.2, 2)$

D: bit3 = 1 → interseca $x = x_R = 8$

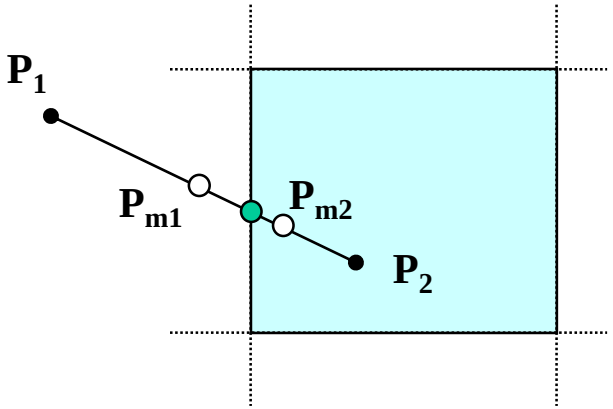
$x = x_1 + t(x_2 - x_1) = 4 + 6t$

$t = 2/3 = 0.66.. \rightarrow I_2 = (8, 4.33..)$

Line clipping (8/8)

Calcolo delle intersezioni – MIDPOINT SUBDIVISION

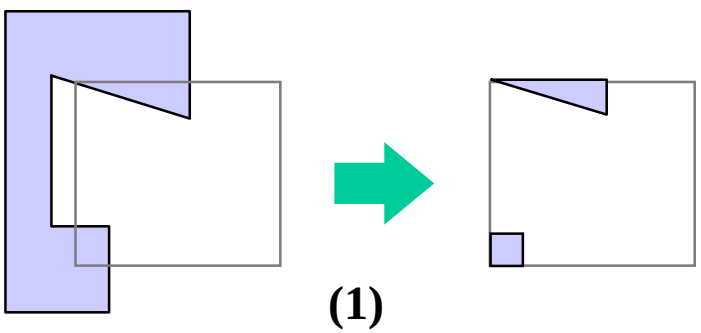
- metodo iterativo basato sul calcolo di punti medi
- per dato un segmento con estremi P_1 (x_1, y_1) e P_2 (x_2, y_2)



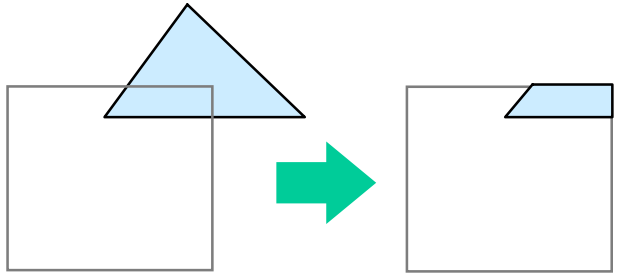
- si calcola il punto medio P_m (x_m, y_m) : $x_m = \frac{x_1 + x_2}{2}$ $y_m = \frac{y_1 + y_2}{2}$
- si suddivide il segmento nelle due metà
- si ripete la suddivisione per i segmenti ottenuti che sono ancora parzialmente visibili
- …P_m tenderà al punto di intersezione

Polygon Clipping

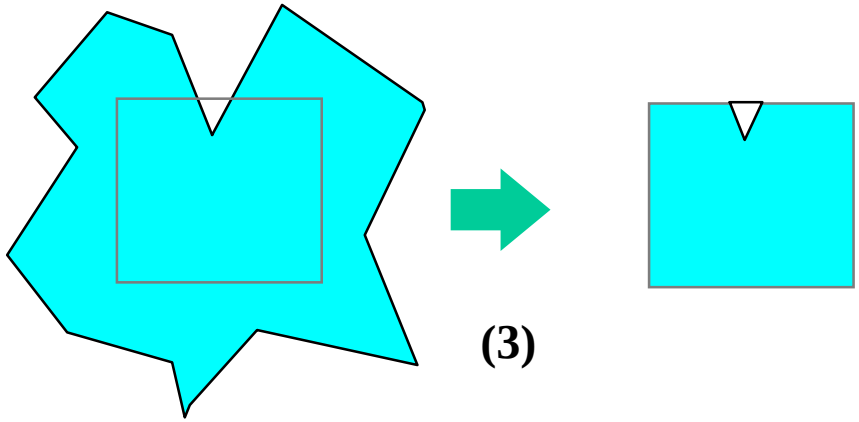
- Si individuano le sottoregioni dei poligoni contenute entro la finestra, che andranno visualizzate
- Le parti esterne alla finestra non saranno visualizzate



(1)



(2)

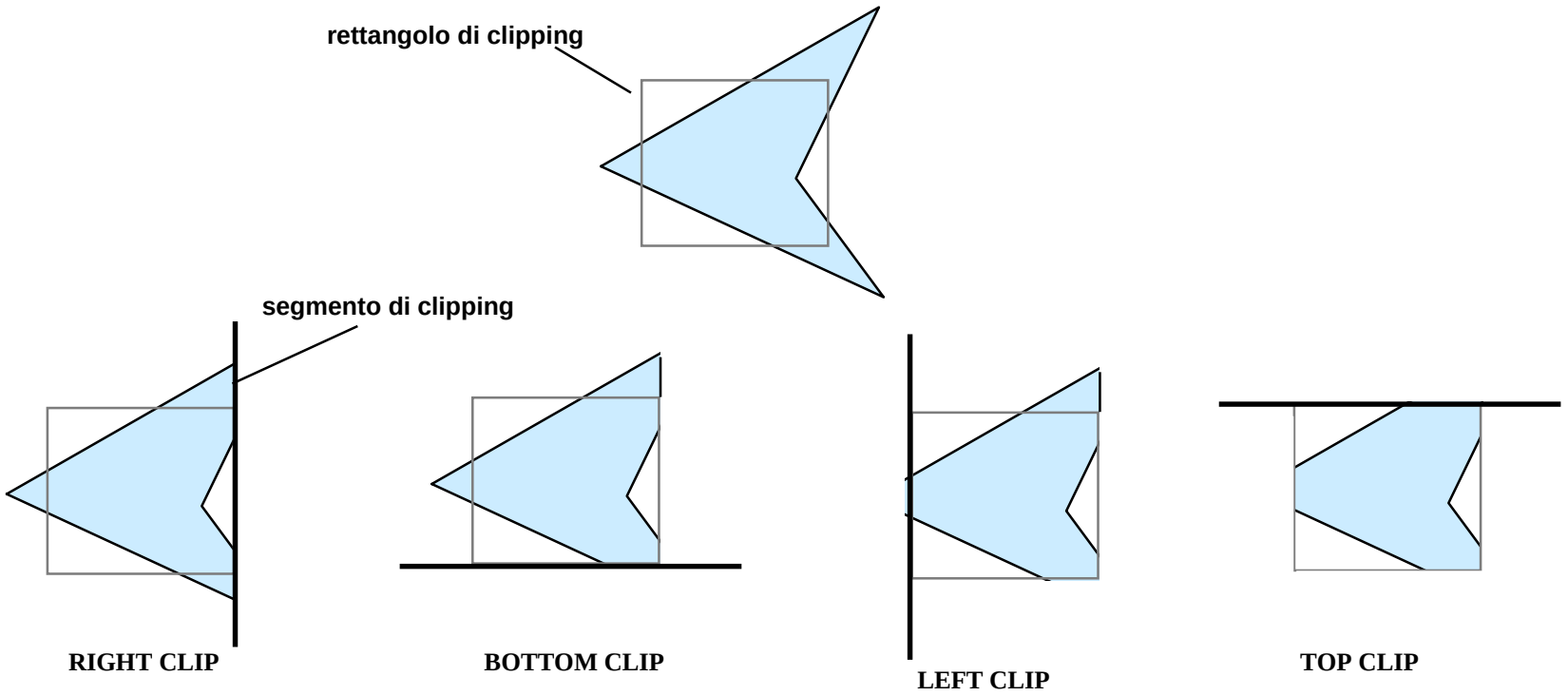


(3)

Polygon Clipping

Algoritmo di Sutherland-Hodgeman

- Strategia: clipping di ogni lato del poligono per ogni segmento del rettangolo di clipping



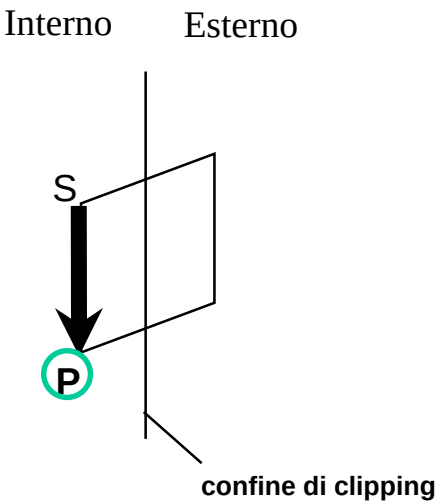
Polygon Clipping

... come opera (1)

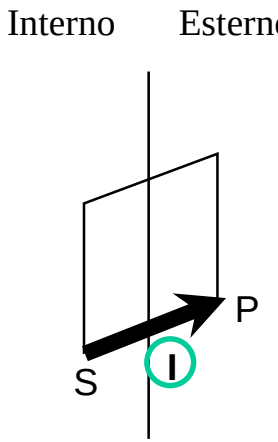
- ad ogni passo
  - INPUT: sequenza dei vertici che definiscono i segmenti del poligono:
    - $P_1, P_2, P_3, P_4, ..., P_n$
  - OPERAZIONE: esamina le relazioni tra tutte le coppie di vertici successivi ed uno dei lati del rettangolo di clipping secondo i criteri illustrati nella prossima slide
  - OUTPUT: sequenza di vertici che definiscono i segmenti del nuovo poligono:
    - $Q_1, Q_2, Q_3, Q_4, ..., Q_m$
- … si ripete per gli altri lati del rettangolo di clipping

Polygon Clipping

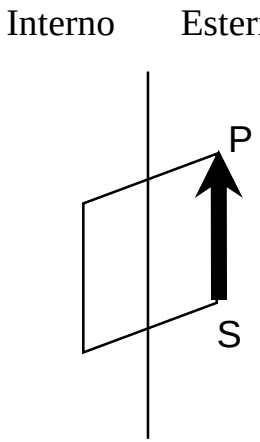
... come opera (2)



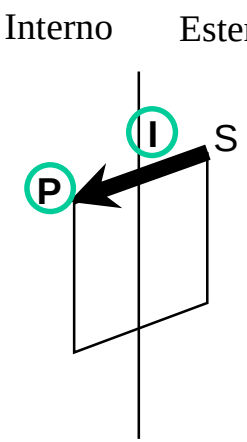
1° caso
output: 1 punto, P



2° caso
output: 1 punto, I

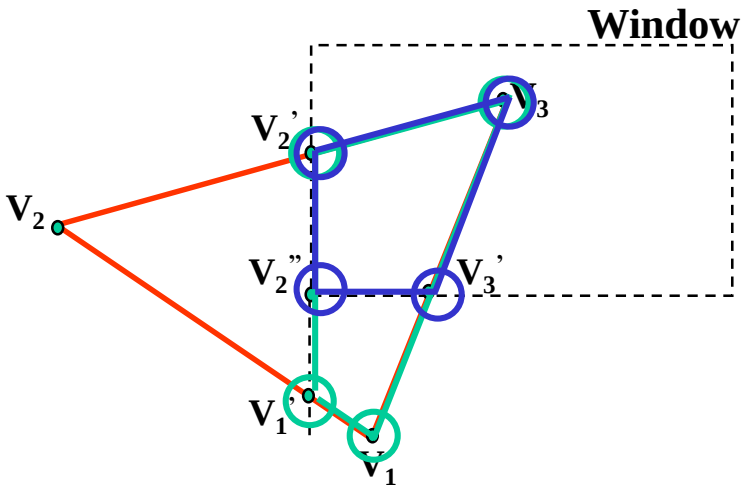


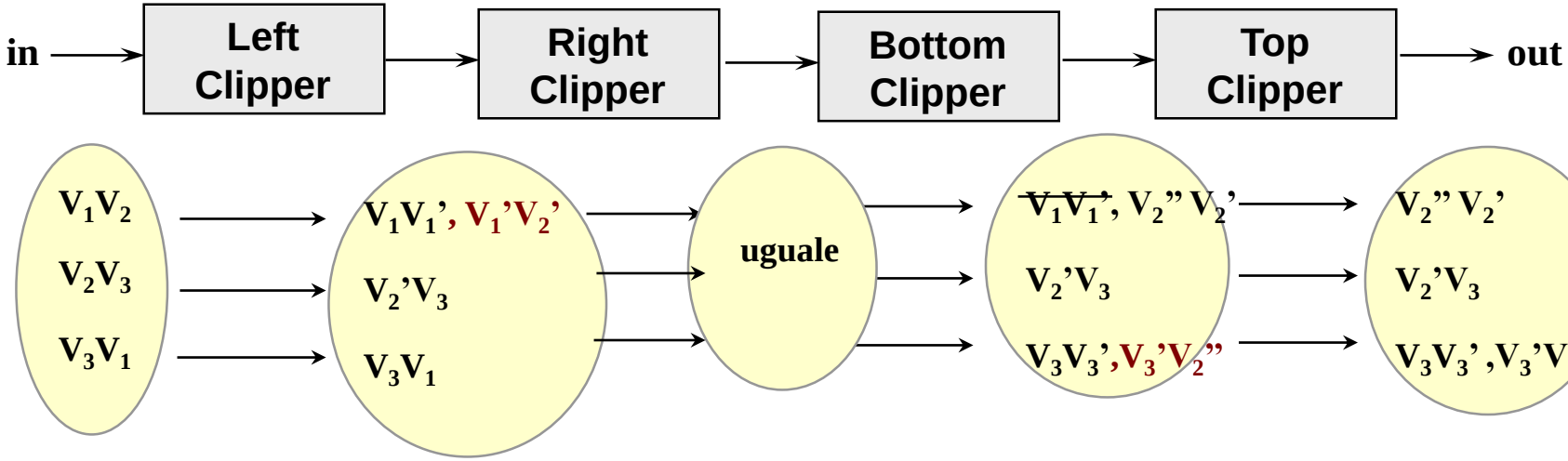
3° caso
output: 0 punti



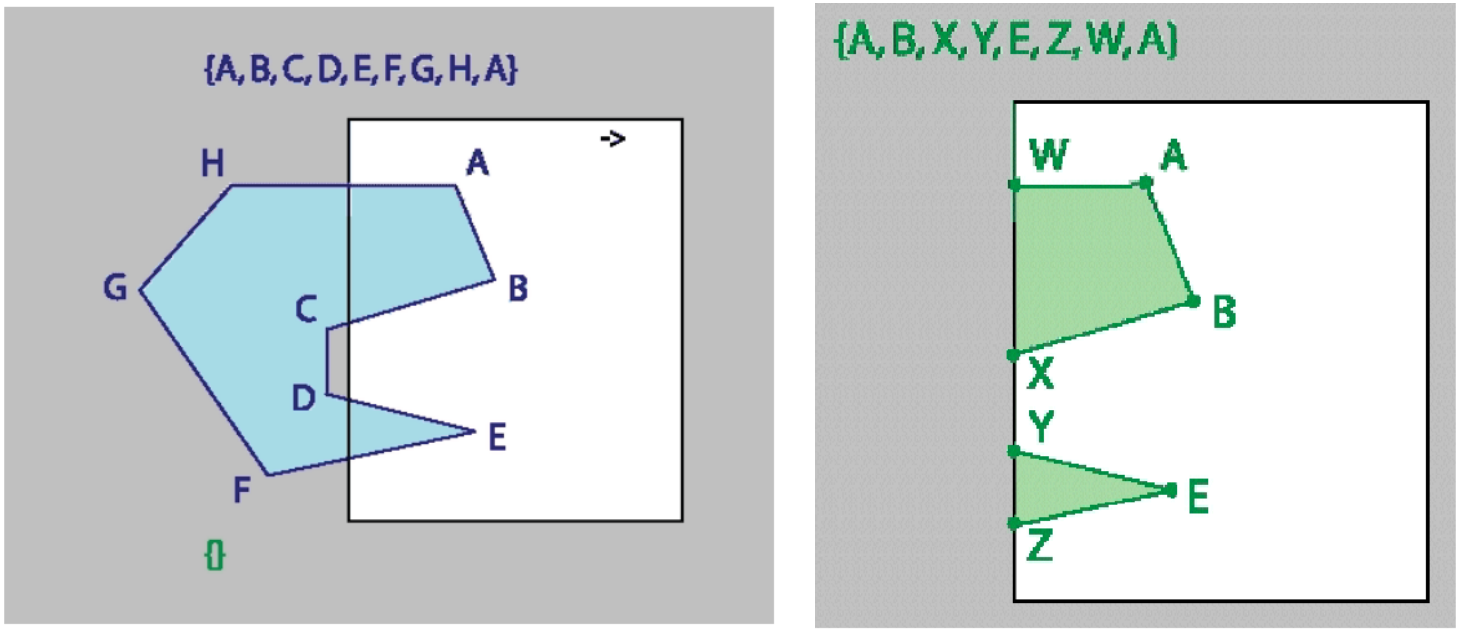
4° caso
output: 2 punti, I e P

Polygon Clipping





Polygon Clipping



Animated by Max Peysakhov @ Drexel University

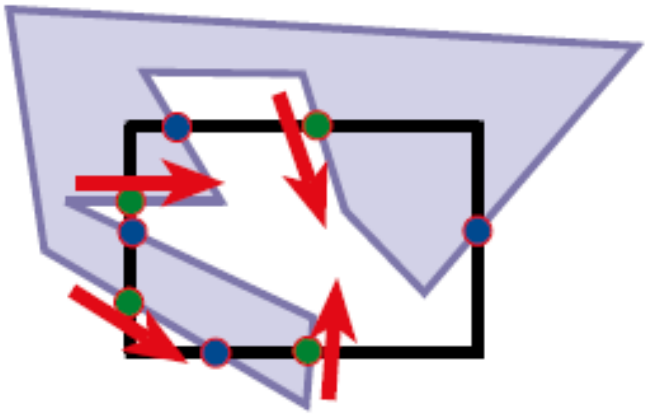
- Svantaggio dell'algoritmo di Sutherland-Hodgeman
  - nel caso di poligoni concavi si generano dei segmenti "spuri" per esempio XY e ZW
  - occorre una fase di post-elaborazione per eliminare questi segmenti

Polygon Clipping

Algoritmo di Weiler-Atherton

Strategia: percorrere i contorni passando dal poligono alla finestra di clipping e viceversa, applicabile a poligoni di forma arbitraria

1. Si calcolano le intersezioni e si marcano (in verde in figura) i vertici dove il poligono “entra” nella finestra di clipping



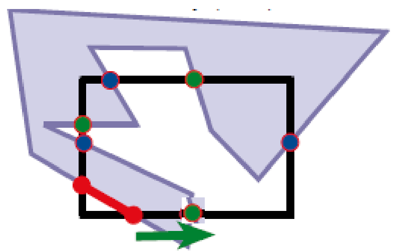
MIT EECS 6.837, Cutler and Durand

http://people.cs.vt.edu/~cpa/clipping/clipping_applet.html

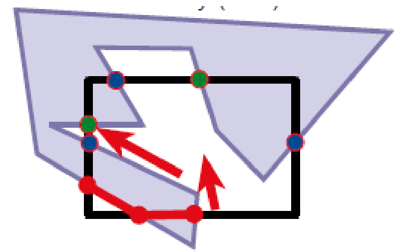
Polygon Clipping

Algoritmo di Weiler-Atherton

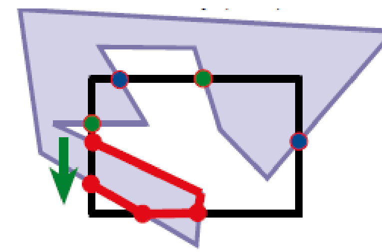
2. Si parte da un vertice entrante
3. Si attraversano i contorni **<u>sempre</u>** in senso antiorario applicando le seguenti regole
   - Se si incontra un vertice entrante (verde) lo si salva e si inizia a percorrere il bordo del poligono
   - Se si incontra un vertice uscente (blu) lo si salva e si inizia a percorrere il bordo della finestra
4. Quando si ritorna al vertice di partenza il poligono individuato supera la fase di clipping
5. Si riparte dal successivo vertice entrante non ancora visitato e si ripete la procedura al punto 3



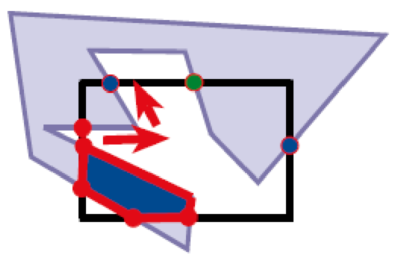
MIT EECS 6.837, Cutler and Durand



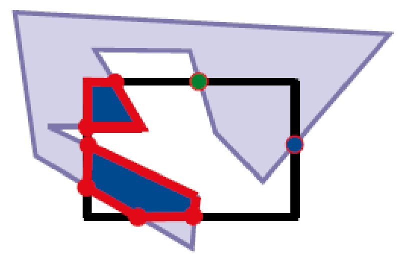
MIT EECS 6.837, Cutler and Durand



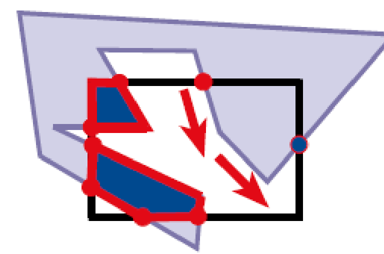
MIT EECS 6.837, Cutler and Durand



MIT EECS 6.837, Cutler and Durand

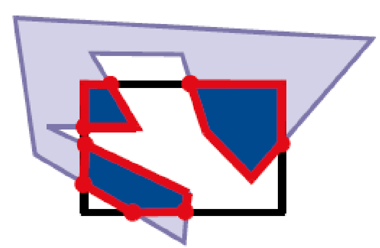


MIT EECS 6.837, Cutler and Durand



MIT EECS 6.837, Cutler and Durand

In questo modo si estraggono progressivamente i “sotto-poligoni” del poligono originale contenuti nella finestra di clipping.

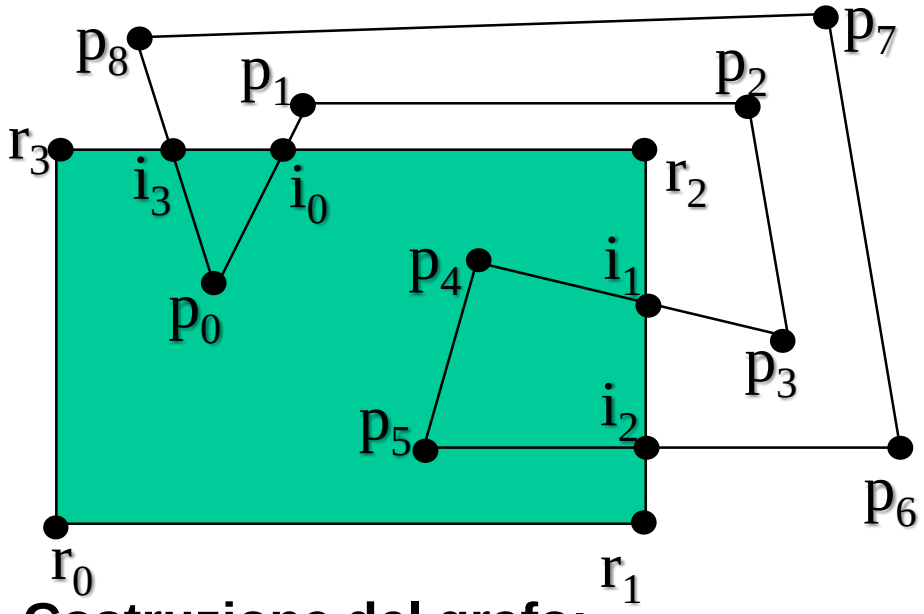


MIT EECS 6.837, Cutler and Durand

Polygon Clipping

Algoritmo di Weiler-Atherton

Tecnica di soluzione mediante grafo di connessione dei vertici del poligono, dei vertici poligono di clipping e dei punti di intersezione

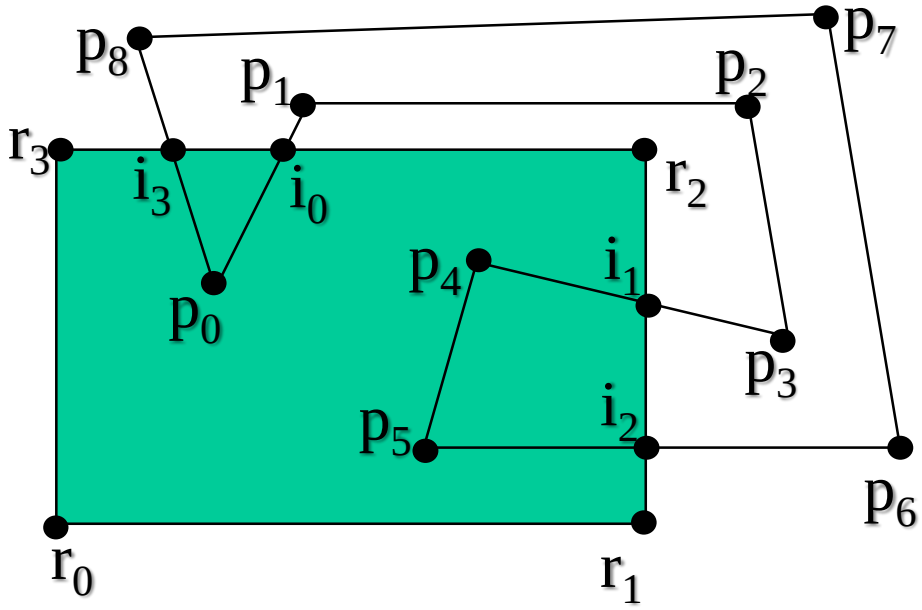


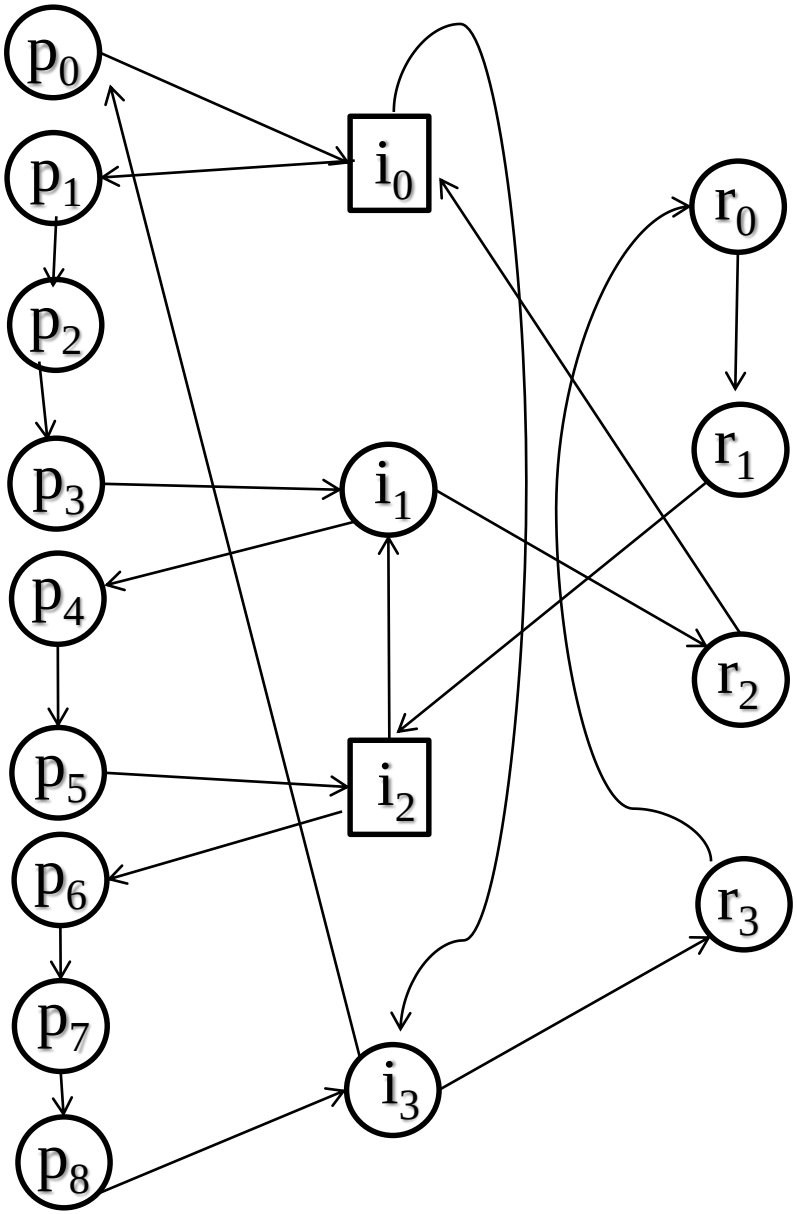
Costruzione del grafo:

1) inserire nodi in tre colonne (vertici poligono, vertici di intersezione distinguendo quelli entranti e quelli uscenti, vertici del poligono di clipping)
2) inserire archi percorrendo il poligono (includendo le intersezioni)
3) inserire archi percorrendo il poligono di clipping (includendo le intersezioni)

Polygon Clipping

Algoritmo di Weiler-Atherton





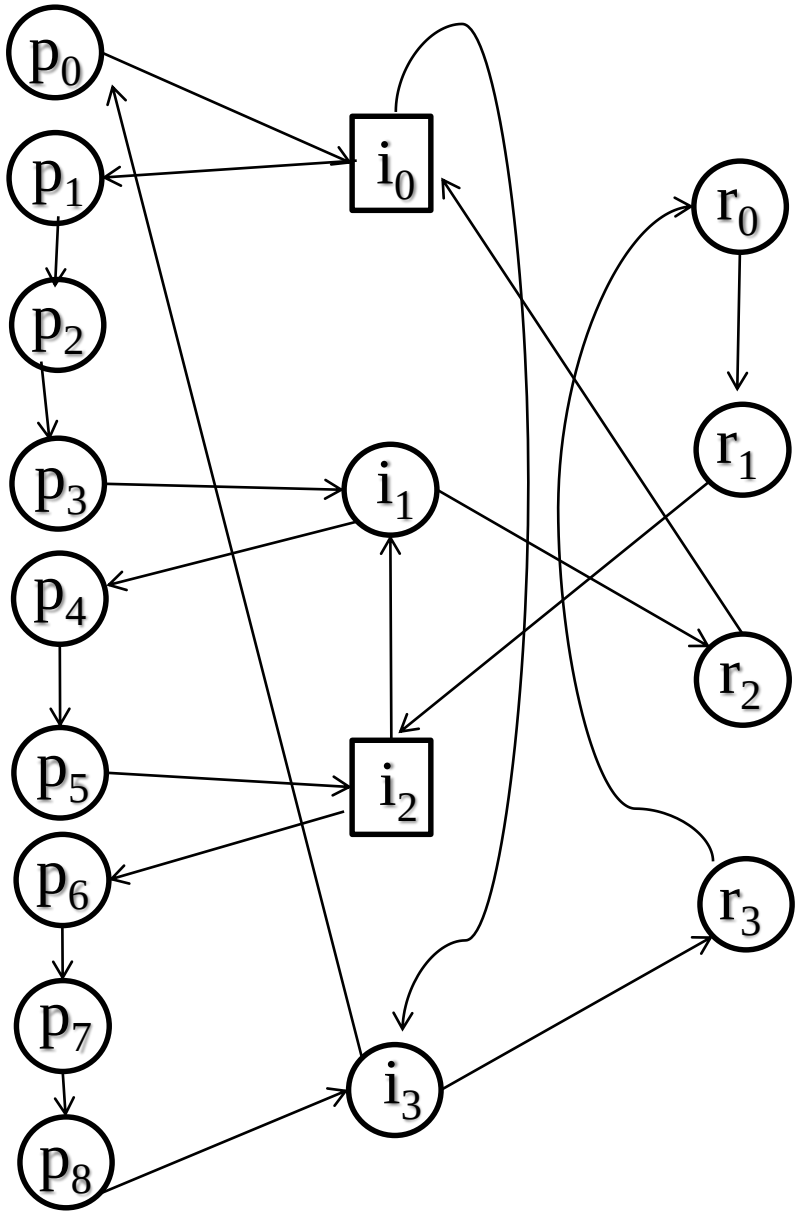
Polygon Clipping

Algoritmo di Weiler-Atherton

Uso del grafo:

1) Si parte dal primo vertice di intersezione entrante (i_1 nell'esempio)
2) Si percorrono i vertici del poligono, quando si incontra un vertice di intersezione entrante si passa a percorrere il poligono, quando si incontra un vertice uscente si torna a percorrere il poligono di clipping
3) Si continua fino a che non si ritorna nel vertice entrante di partenza e si ottiene così una porzione del poligono " clippato"
4) Si ricomincia l'algoritmo al punto 1 considerando il successivo vertice entrante non ancora visitato, si termina quando non ci sono altri vertici uscenti da visitare

Applicando l'algoritmo all'esempio si ottiene che i sotto-poligoni: $\mathbf{i_1 p_4 p_5 i_2}$ e $\mathbf{i_3 p_0 i_0}$ sono le porzioni visibili del poligono originale



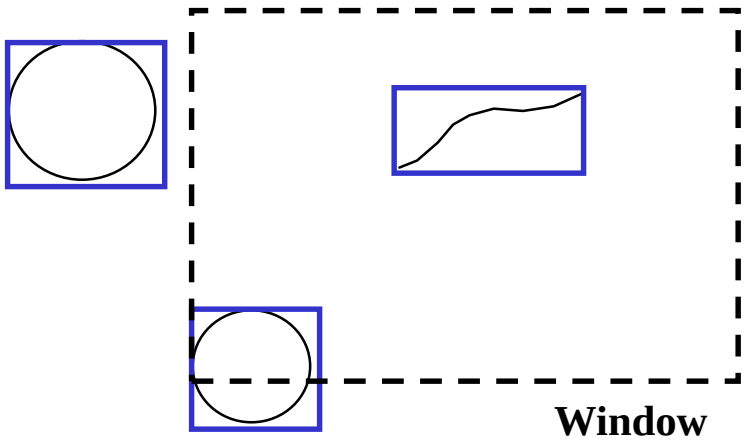
Clipping di curve e superfici

- Selezione e visualizzazione di parti di curve (in 2D) e superfici (in 3D) contenute dentro la finestra
- **TECNICA PRELIMINARE: uso di *bounding box***
  - si verifica prima l'intersezione fra bounding box e area di clipping
  - Se bounding box contenuta dentro rettangolo di clipping → curva è tutta visibile
  - Se bounding box esterna → curva è tutta invisibile
  - Vantaggio: calcolo di intersezione fra curva-area di clipping solo quando bounding box interseca la finestra

intersezioni curva/linee:

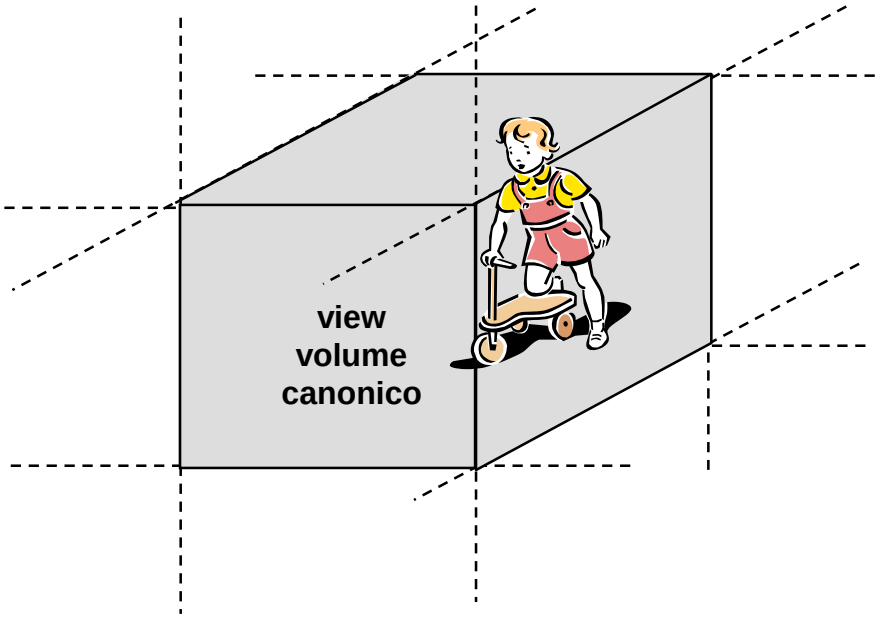
è un sistema algebrico NON lineare

$$\begin{cases} x=x(t) \\ y=y(t) \\ x=x_{L(R)} \text{ o } y=y_{B(T)} \end{cases} \quad \text{o} \quad \begin{cases} f(x,y)=0 \\ x=x_{L(R)} \text{ o } y=y_{B(T)} \end{cases}$$



Clipping 3D (1/3)

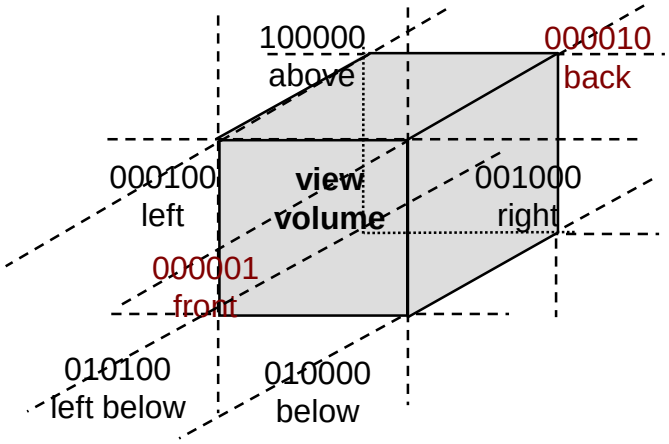
- Selezione di parti/regioni della scena visibili, fatta rispetto a View Volume Canonico
- Gli algoritmi visti in precedenza sono estendibili al caso 3D (es: Sutherland-Hodgeman si estende considerando intersezioni tra lati dei poligoni e i piani del volume canonico)



Clipping 3D (2/3)

Estensione dell'Algoritmo Cohen Sutherland (1/2)

- Per clipping 3D di linee
- Rispetto al view volume, si individuano 6 regioni "principali"
- Si associa un *codice a 6 bit* per ogni vertice della linea
  - bit 1 = 1 se vertice è sopra
  - bit 2 = 1 se vertice è sotto
  - bit 3 = 1 se vertice è destra
  - bit 4 = 1 se vertice è sinistra
  - bit 5 = 1 se vertice è dietro
  - bit 6 = 1 se vertice è davanti
- 27 regioni complessive



Clipping 3D (3/3)

Estensione dell'Algoritmo Cohen Sutherland (2/2)

- Bitwise logical AND per classificare i segmenti come:
  - totalmente/parzialmente visibili
  - invisibili
  - indeterminati
- Per i segmenti parz. visibili o indeterminati si calcola l'intersezione:

$$\begin{cases} x = x_1 + t\,(x_2 - x_1) \\ y = y_1 + t\,(y_2 - y_1) \quad 0 \le t \le 1 \\ z = z_1 + t\,(z_2 - z_1) \\ x=\text{const} \;\; o \;\; y=\text{const} \;\; o \;\; z=\text{const} \end{cases}$$

Rimozione di superfici nascoste

Rimozione di superfici nascoste

- L’immagine dipende dal punto di vista e dalle relazioni di posizione tra gli oggetti
  - Senza un adeguato “rendering” c’è ambiguità:
    - in che cosa è davanti e che cosa è dietro
  - Necessità di tecniche/algoritmi per:
    - Nascondere/non mostrare le parti che l’occhio non vede
- Il problema della rimozione delle superfici nascoste (HSR, Hidden Surface Removal) consiste nel determinare le parti della scena non visibili dall’osservatore

Rimozione di superfici nascoste

Motivazione

- migliorare l'effetto 3D della scena
- eliminare ambiguità nella percezione di profondità
- per dare ovviamente più realismo all'immagine

Aspetti del problema

Oggetti hanno zone occluse rispetto all'osservatore

Oggetti della scena sono opachi

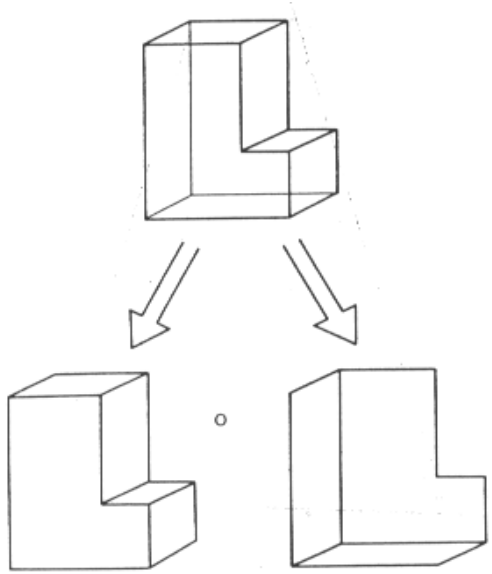
→ oggetti vicini "nascondono" parte degli oggetti lontani

Problematiche fondamentali

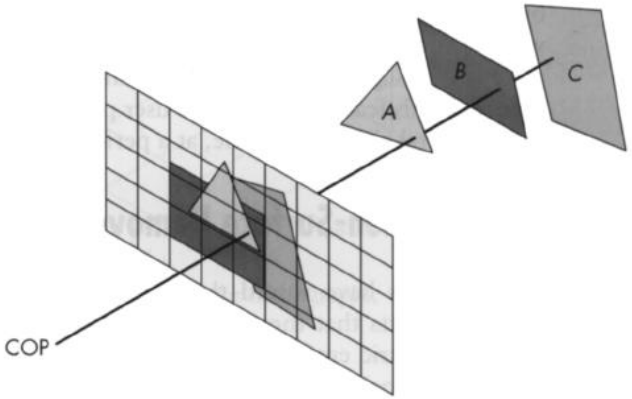
- BACK-FACE CULLING:
  Non mostrare le zone occluse degli oggetti

- RIMOZIONE DI SUPERFICI NASCOSTE:
  Non mostrare oggetti "coperti" da quelli davanti

Back-Face culling



Rimozione superfici nascoste



Back-face culling (1/3)

… una prima tecnica di filtraggio (pre-processing) che viene eseguita per togliere superfici/facce sicuramente non visibili

- BACK-FACE CULLING:
  tecnica di non-visualizzazione delle parti occluse degli oggetti rispetto all'osservatore. Vediamo la definizione e come viene realizzato in OpenGL.

Ipotesi

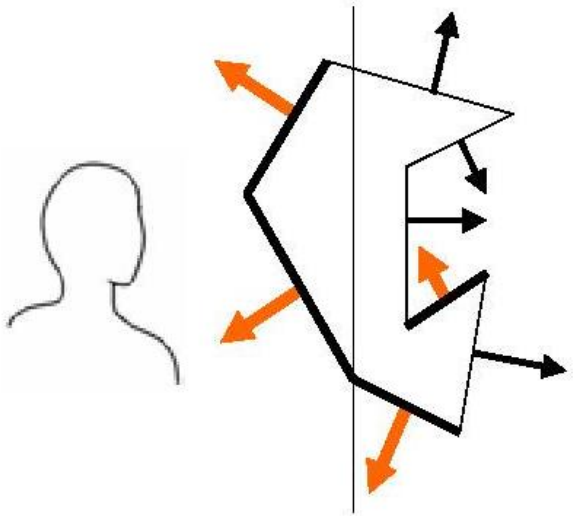
- Oggetto modello poliedrale
  - facce sono planari

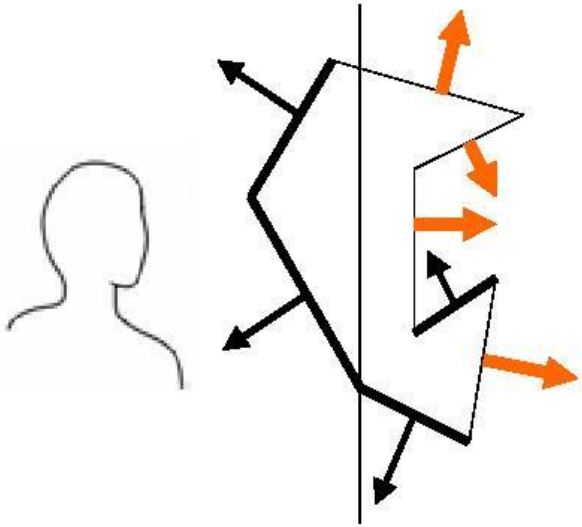
- Ipotesi di orientazione "coerente"
  - Le normali alle facce puntano tutte verso l'esterno dell'oggetto

Back-face culling (2/3)

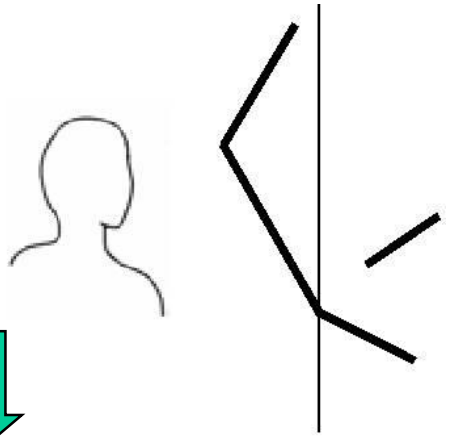
Se…

- 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
  - quelle con normale che punta *via* dall'osservatore **sicuramente non lo sono**





- Le facce sicuramente invisibili non vengono più considerate nelle fasi successive del processo di rendering



Vengono eliminate facce/patches la cui normale è diretta nel verso opposto all'osservatore

Back-face culling (3/3)

Come verificare se le facce devono essere eliminate?

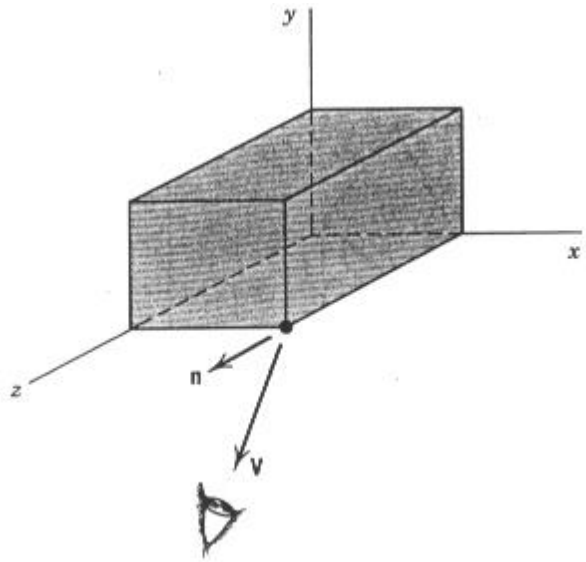
- Si determina la *normale uscente* n per ogni faccia dell'oggetto
- Si considera un *vettore* V da un qualsiasi punto della faccia diretto verso il punto di vista
- Si calcola $V \cdot n$ (prodotto scalare)
  - se $V \cdot n > 0$ → faccia può essere visibile
  - altrimenti → faccia non visibile/nascosta
  - in realtà poichè siamo in coordinate normalizzate è sufficiente valutare il segno della coordinata z del vettore normale

Vantaggi:

Circa metà delle facce sono "back" cioè non visibili

→ circa dimezzati I tempi per il rendering

<u>Osservazione:</u> OpenGL supporta il back-face culling (vedi prima esercitazione), ma il calcolo avviene in base all'ordinamento dei vertici dei poligoni, **non** in base alle normali dei poligoni



Rimozione di superfici nascoste

Il back-face culling non è sufficiente per la rimozione completa di tutte le superfici nascoste della scena.
E' necessaria una fase successiva di rimozione di superfici occluse (dovute alla presenza di più oggetti nella scena) mediante algoritmi più specifici.

Possibilità

- lavorare nello spazio oggetto (object-space)
- lavorare nello spazio immagine (image-space)

Object-space (1/3)

Significato:

- Si opera nello spazio delle primitive geometriche. Vengono determinate per ogni oggetto quali parti (primitive) non sono oscurate da altri oggetti della scena.

Idea dell'approccio

- Scena = unione di oggetti-geometrie 3D
- Confronto tra tutte le coppie di oggetti A e B

Relazioni possibili:

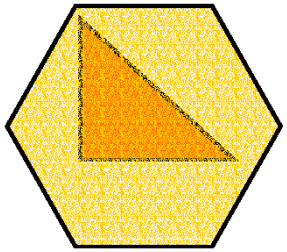
- A oscura B → visualizzato solo A
- B oscura A → visualizzato solo B
- A e B non si oscurano tra loro → visualizzati sia A che B
- A oscura parzialmente B
- B oscura parzialmente A

Vanno calcolate le sottoregioni visibili

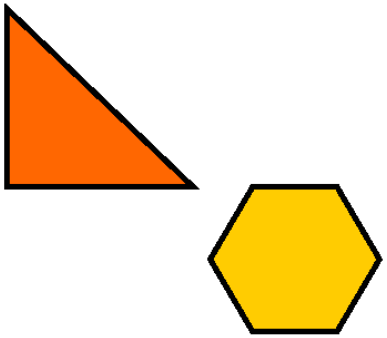
Object-space (2/3)

I vari casi

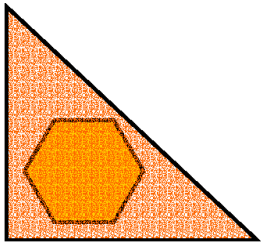
- A oscura B:
  tutti i punti di A sono più vicini all'osservatore di tutti i punti di B e la proiezione di B sul piano di vista ricade completamente all'interno della proiezione di A
- Visualizzeremo solo A



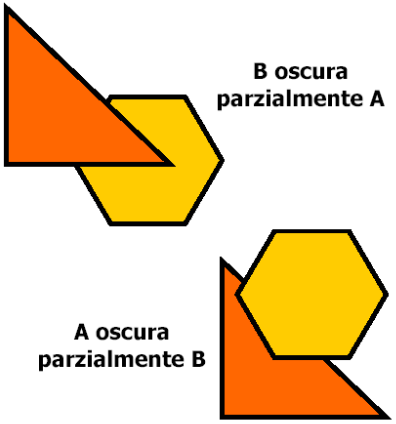
- A e B sono completamente visibili:
  le due proiezioni di A e B sul piano di vista sono disgiunte
- Visualizzeremo sia A che B



- 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 completamente all'interno della proiezione di B
- Visualizzeremo solo B



- A e B si oscurano parzialmente l'un l'altro:
  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
- Abbiamo la necessità di calcolare le parti visibili di ciascun poligono



Object-space (3/3)

Un possibile algoritmo:

- Scena con k poligoni
- Si applica la proiezione 3D→2D di tutti i k poligoni
- Per ogni passo i=1,2,..k:
  - Si confronta l'i-mo poligono con ognuno dei k-i poligoni restanti (se confronto i-j è già avvenuto, non si esegue confronto j-i)
  - Si determina la parte visibile di ogni poligono

Complessità: **$O(k^2)$**

Consigliabile quando:

i poligoni della scena sono pochi: k piccolo

Image-space (1/2)

Significato

- Si opera nello spazio immagine della scena proiettata. Viene determinato per ogni pixel quale oggetto è più vicino all'osservatore

Algoritmo generale (Ray Casting):

- Per ogni pixel della window:
  - si considera la retta che passa per esso e il centro di proiezione
  - si calcolano le intersezioni della retta con gli oggetti
  - si calcolano le distanze tra tali intersezioni con il centro di proiezione
  - il pixel viene colorato sulla base del colore dell'oggetto avente intersezione a distanza minima

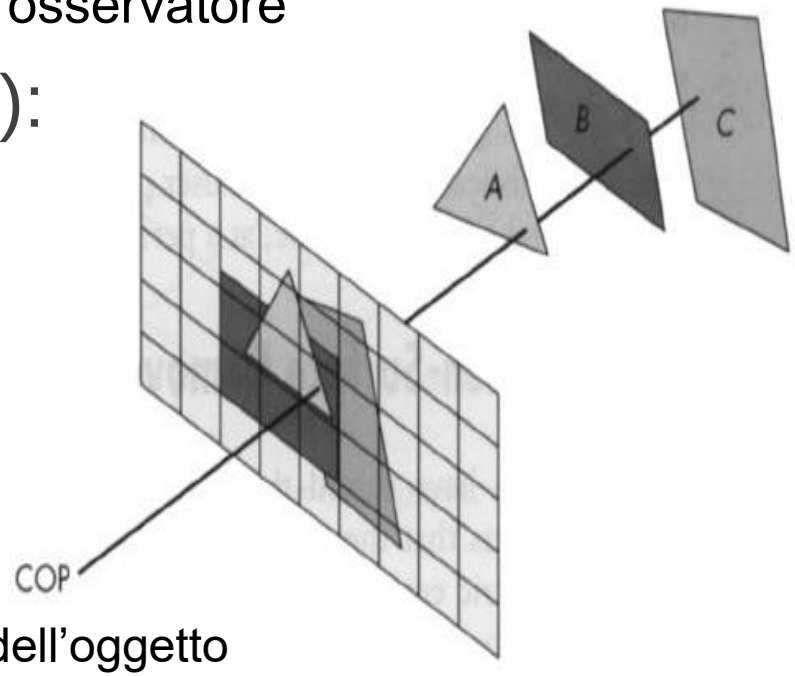


Image-space (2/2)

Caratteristiche

- operazione fondamentale: intersezioni tra rette e piani/poligoni
- k poligoni: da fare k volte

Complessità

- per un display $n_x m$ pixel: operazione viene eseguita $n_x m_x k$
- ma n,m sono costanti → complessità **O(k),** lineare ma con costante di proporzionalità $n_x m$ tipicamente molto elevata dovuta alla risoluzione dello schermo

Rimozione di superfici nascoste

Vedremo vari algoritmi per l'identificazione di superfici visibili (e quindi per la rimozione di superfici nascoste)

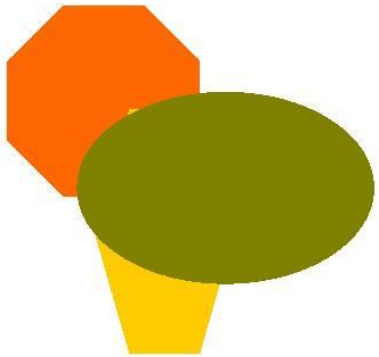
- Algoritmo Depth sort
- Algoritmo Z-Buffer
- Algoritmo di scan-line (Watkin)
- Ray Casting (già accennato nelle precedenti slides)

Algoritmo Depth-Sort (1/4)

- Esempio di algoritmo object-space
- Richiede:
  - poligoni siano ordinati per distanza decrescente dall'osservatore
- Variante dell' "**algoritmo del pittore**"
- Idea:

  Dipingere l'oggetto più lontano, poi i successivi progressivamente meno lontani, colorando sopra gli oggetti precedentemente dipinti

  1. Le primitive geometriche della scena sono ordinate in ordine decrescente di profondità rispetto all'osservatore
  2. Le primitive geometriche sono disegnate (rasterizzate) dalla più lontana alla più vicina

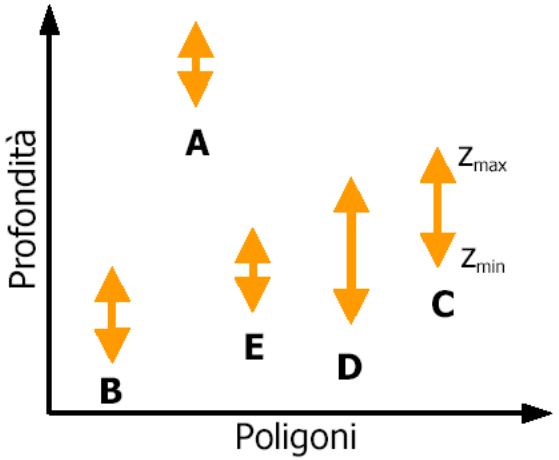


- Attenzione:

  ambiguità se ci sono zone di z sovrapposte

Algoritmo Depth-Sort (2/4)

Algoritmo

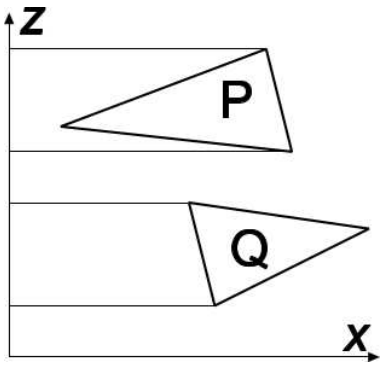


- Ordinare i poligoni/oggetti in base alla profondità dall'osservatore
- Casi di ordinamento:

1) **z min di oggetto P > z max di oggetto Q:**

→ nell'ordinamento, P precede Q

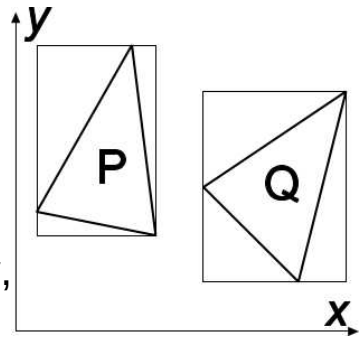
Oss: In questa slide l'asse z è invertito rispetto a come di solito lo definiamo nella pipeline e rappresenta la profondità



2) **se range delle z di P e Q si intersecano:**

Si controllano range delle x e delle y:

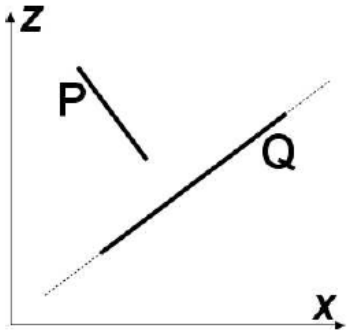
- se gli intervalli (range) non si intersecano su almeno un asse x o y, allora P e Q si visualizzano in ordine qualsiasi



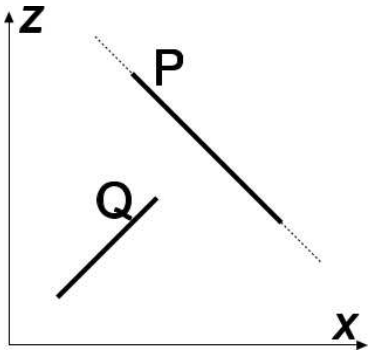
Algoritmo Depth-Sort (3/4)

Algoritmo

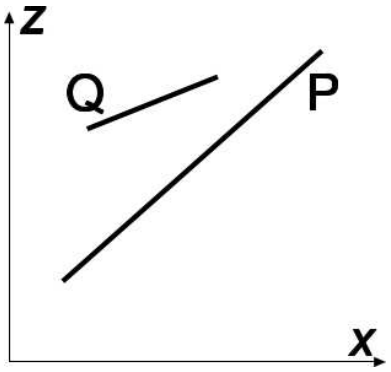
- Altrimenti, se tutti i vertici di P si trovano dalla parte opposta dell'osservatore rispetto al piano individuato da Q, allora P precede Q;



- Altrimenti, se tutti i vertici di Q si trovano dalla stessa parte dell'osservatore rispetto al piano individuato da P, allora P precede Q;

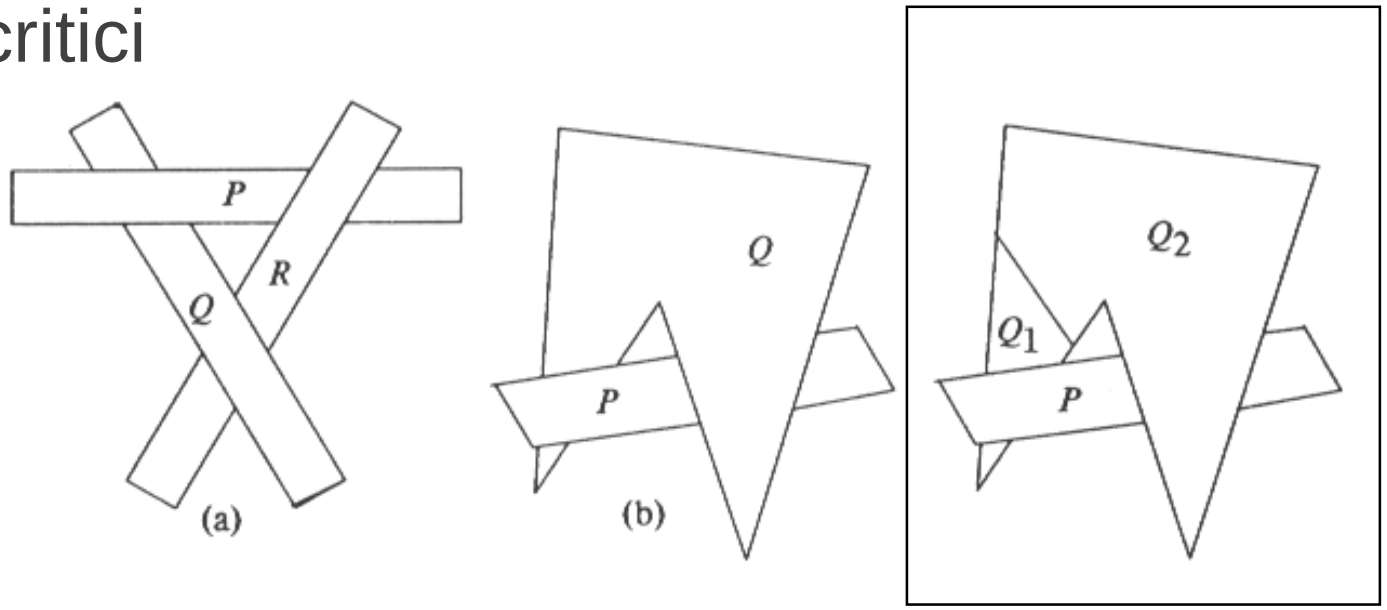


- Se tutti i test precedenti forniscono esito negativo allora si procede allo scambio di P con Q nell'ordinamento e, nuovamente, all'esecuzione dei test;



Algoritmo Depth-Sort (4/4)

Casi critici



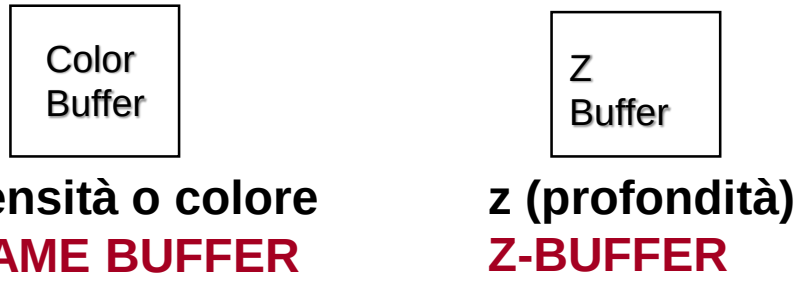
a) sovrapposizione ciclica: poligoni per i quali non esiste alcun ordine di priorità.

b) poligoni si intersecano/sovrappongono.

L' ordine delle priorità può essere stabilito: calcolando intersezioni tra essi,
o suddividendo in sottopoligoni

Algoritmo Z-Buffer (1/5)

- Esempio di algoritmo image-space, è l'algoritmo usato da OpenGL
- Lavora in coppia con algoritmi di scan conversion
- Richiede 2 aree di memoria



- Algoritmo:
  1. **si inizializzano i due buffer** (background color e z minima)
  2. **Per ogni poligono (in ordine casuale) si determinano i pixel corrispondenti alla sua proiezione sul piano di proiezione, utilizzando un algoritmo di scan conversion (rasterizzazione). Nella pipeline moderna si utilizza una tecnica basata sulle coordinate baricentriche.**

     Per ogni pixel del poligono in esame si calcola il valore di *z*

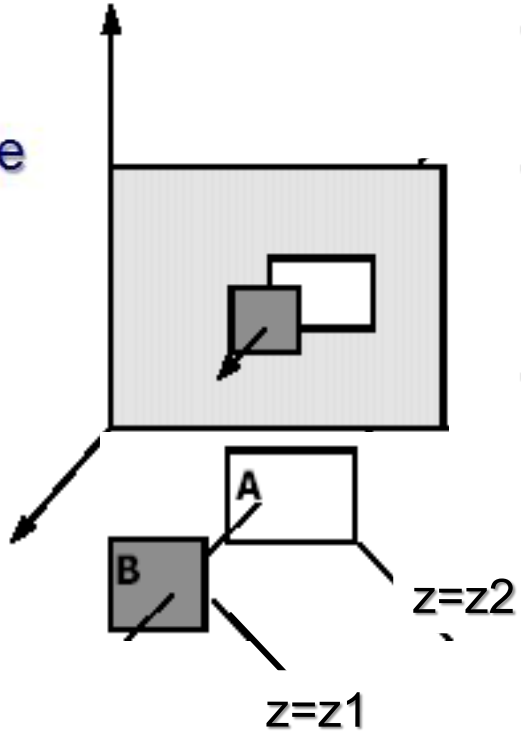
     –se *z* > del valore attuale nello zbuffer
     - si aggiorna il valore della z nello zbuffer
     - si aggiorna il valore dell'intensità nel color buffer

     –altrimenti non viene fatta nessuna operazione

Algoritmo Z-Buffer (2/5)

Esempio

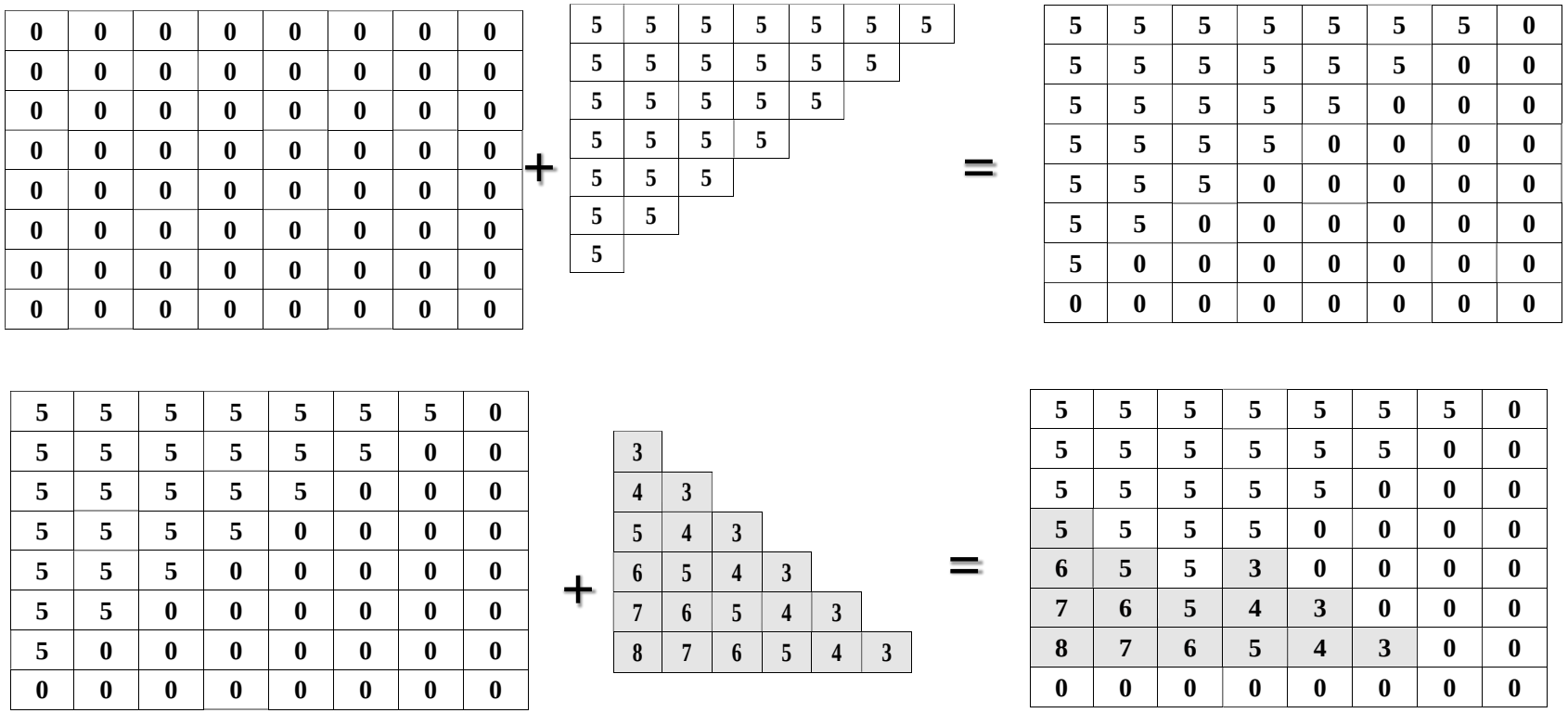
- Quando si esegue la scan conversion del poligono B, il suo colore apparirà sullo schermo poiché la sua quota z_1 è maggiore della quota z_2 relativa al poligono A



- Al contrario, quando esegue la scan conversion del poligono A, il pixel che corrisponde al punto di intersezione non apparirà sul display

Algoritmo Z-Buffer (3/5)

Z-Buffer: esempio di progressivo caricamento dei valori



- Stessa risoluzione del frame buffer (X-pixels x Y-pixels)
- Ogni cella contiene informazioni su profondità z di tipo reale (es. 24 bit)
- z quota: Livello z=0 è quello del back clipping plane

Algoritmo Z-Buffer (4/5)

Pseudo codice

```
void zBuffer
int pz;
{
   for(y=0; y<YMAX; y++){
      for(x=0; x<XMAX; x++){
                WritePixel(x,y, BACKGROUND_VALUE);     /* pulisce i due buffer */
                WriteZ(x,y,0);
             }
   }
   for(each polygon){
      for(each pixel in polygon's projection){
         pz=polygon's z-value at pixel coords(x,y);
         if(pz>=ReadZ(x,y){ /*New point is not farther*/
                        WriteZ(x,y,pz);
                        WritePixel(x,y,polygon's color at pixel coords (x,y));
               }
             }
    }
 }
```

Algoritmo Z-Buffer (5/5)

Calcolo del valore di z

Si sfrutta l'ipotesi di lavorare con poligoni.

il valore di z di un pixel si ottiene applicando una formula (non vediamo i dettagli) che interpola i valori di z dei vertici 3D del poligono che si proietta su quel pixel.

i valori di z dei vertici del poligono sono ottenuti applicando i vari stadi della pipeline (modelview+projection+viewport transformation).

i valori di z diminuiscono all'aumentare della distanza dall'osservatore.

Algoritmo di Scan Line

- Estensione dell'algoritmo di scan line per rasterizzare le parti di un poligono (vedi slides iniziali)
- Il problema della visibilità tra oggetti è ricondotto ad un problema di visibilità tra segmenti
- Noto anche come **Algoritmo di Watkins**
- Esempio di algoritmo image-space
- Per ogni linea di scansione orizzontale: si calcolano intersezioni dei poligoni proiettati sul piano immagine con essa → si determinano i segmenti visibili
- Si calcola la profondità per risolvere ambiguità di segmenti sovrapposti
- Mantiene una lista dinamica di lati attivi
- Si sfruttano ipotesi di coerenza sulla linea di scansione
  - es: se 2 scan lines successive intersecano stessi spigoli si sfruttano informazioni precedentemente calcolate

Algoritmo di Scan Line

- Utilizza 3 strutture dati

1. **Tabella degli spigoli (edge table, ET)** (contiene: gli estremi di ogni spigolo proiettato, il coefficiente angolare della retta di appartenenza e una etichetta che identifica il nome del poligono di appartenenza);

2. **Tabella delle primitive (polygon table, PT)** (contiene: coefficienti dell'equazione del poligono 3D, colore o informazioni di shading, flag ''In'' che assume valore True quando l'algoritmo ''entra'' in un poligono;

3. **Tabella degli spigoli attivi (active edge table, AET)** (contiene: gli spigoli "attivi", cioè gli spigoli non orizzontali intersecati dalla scanline corrente. Gli spigoli sono ordinati secondo valori crescenti dell'ascissa di intersezione con la scanline

Algoritmo di Scan Line

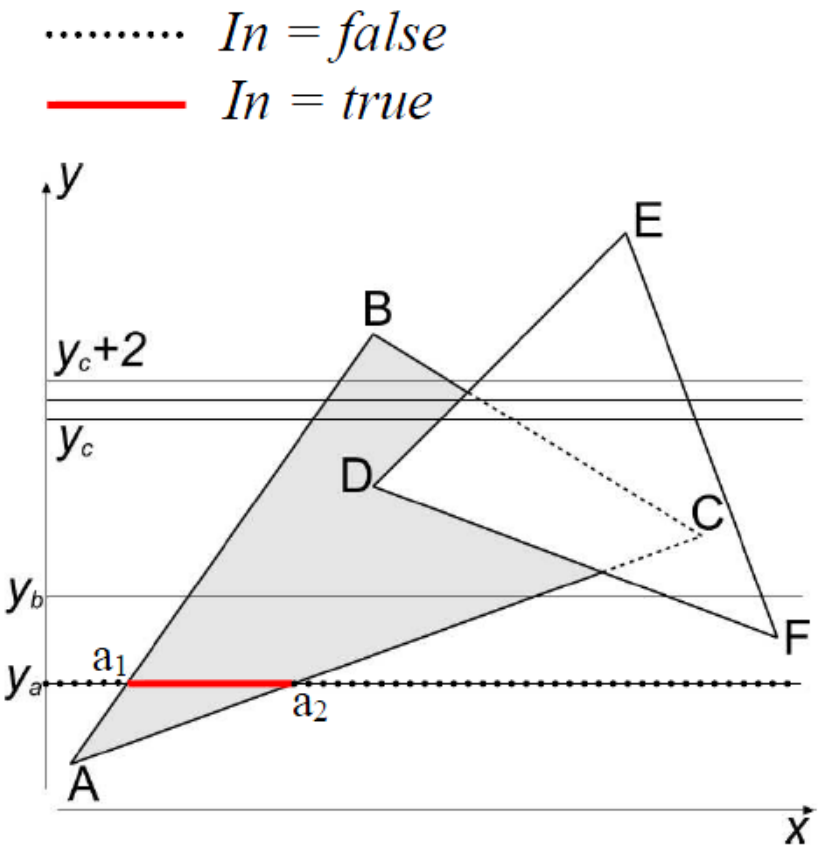
Algoritmo:

Scan-line $y=y_a$;

Spigoli attivi: AB, AC;

In=true tra a_1 e a_2 sullo scan line $y=y_a$;

In questo intervallo è l'unico flag attivo quindi i pixel di ordinata $y=y_a$ e ascissa compresa tra a_1 e a_2 sono visibili.



Algoritmo di Scan Line

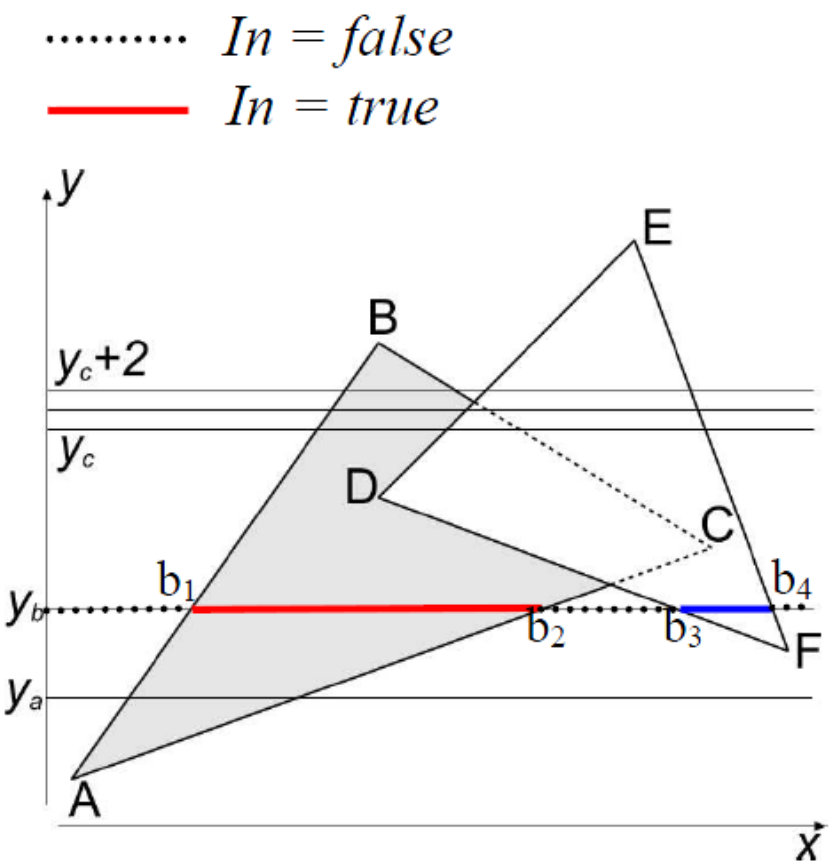
Algoritmo:

Scan-line $y=y_b$;

Spigoli attivi: AB, AC, DF ed EF;

In=true tra b_1 e b_2 nella primitiva ABC e tra b_3 e b_4 in DEF;

Nessun conflitto in questi intervalli; i pixel di ordinata $y=y_a$ e ascissa compresa tra b_1 e b_2, b_3 e b_4 sono visibili.



Algoritmo di Scan Line

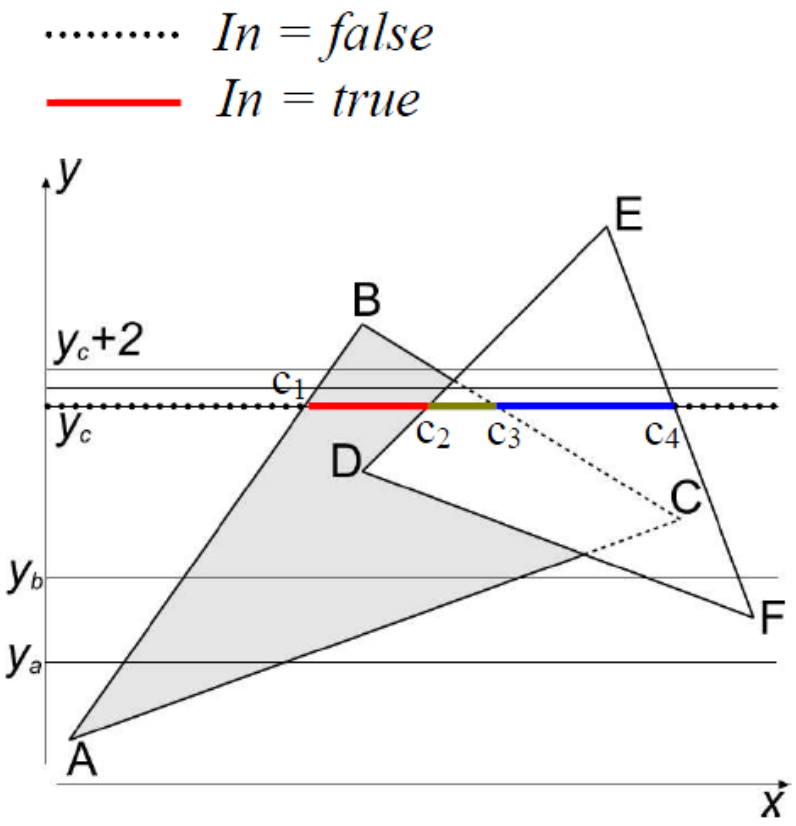
Algoritmo:

Scan-line $y=y_c$;

Spigoli attivi: AB, DE, BC ed EF;

Un solo flag *true* tra c_1 e c_2 e tra c_3 e c_4;

Tra c_2 e c_3 due flag *In* risultano *true*. Occorre valutare la profondità di entrambe le primitive per decidere chi è visibile. E' sufficiente il test in un solo punto (c_2 nell'esempio) .

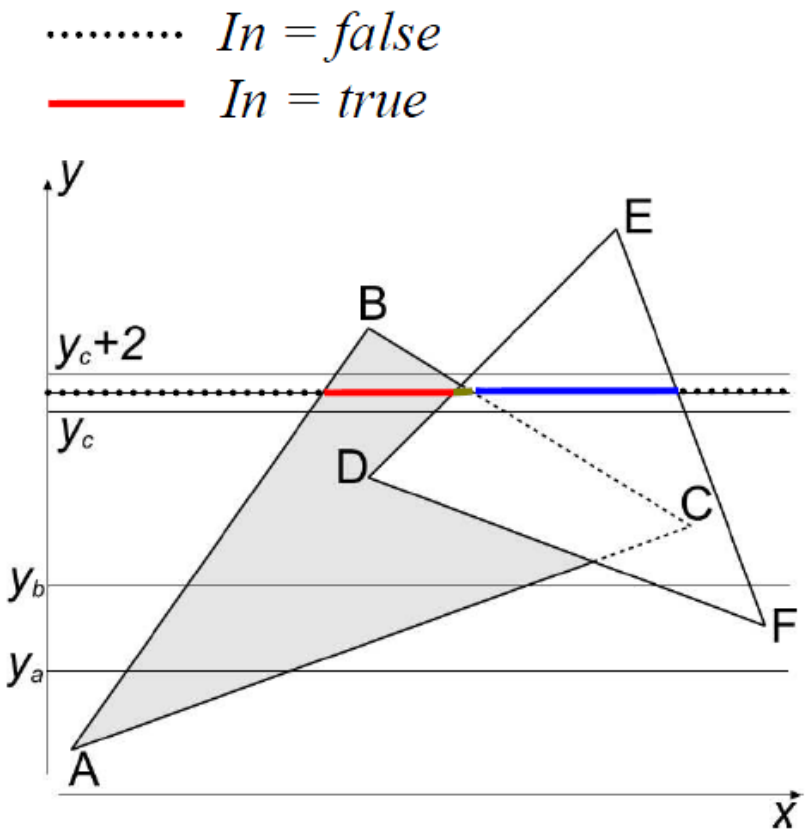


Algoritmo di Scan Line

Algoritmo:

Scan-line $y=y_{C;+1}$

Gli spigoli attivi sono gli stessi dei precedenti (e nello stesso ordine)
Coerenza di oggetto: nessun cambiamento topologico rispetto alla scan-line precedente, è quindi sufficiente aggiornare le ascisse delle intercette



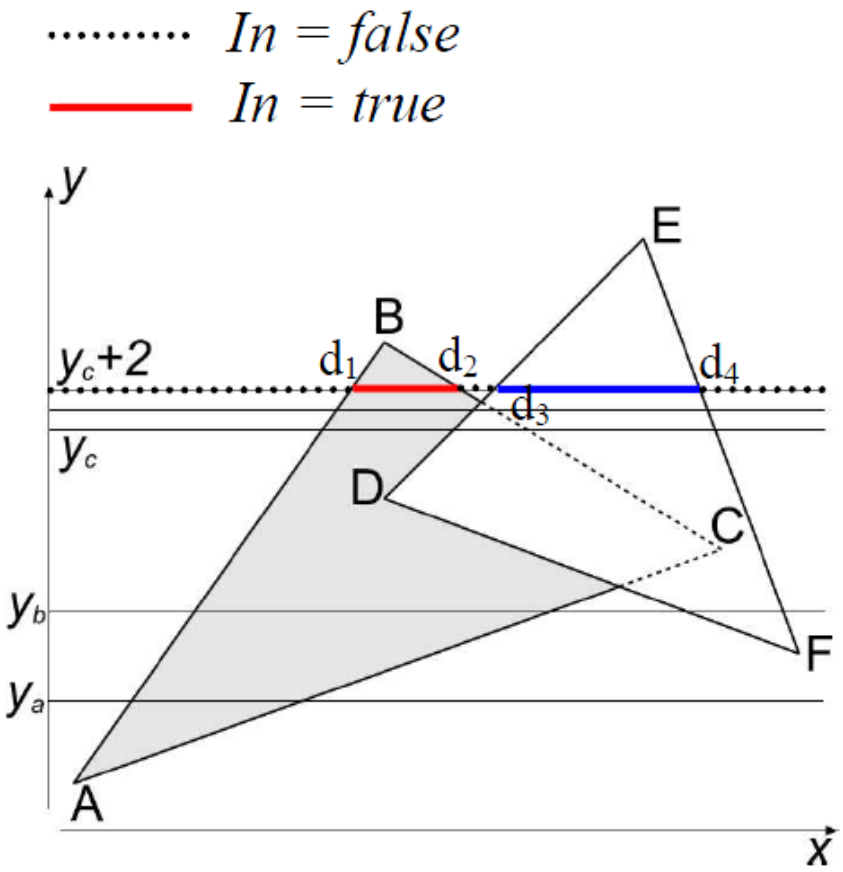
Algoritmo di Scan Line

Algoritmo:

Scan-line $y=y_{c+2}$;

Gli spigoli attivi (AB, BC, DE ed EF) sono cambiati (nell'ordine) rispetto ai precedenti;

Non è sfruttabile la coerenza rispetto alla scan-line precedente.

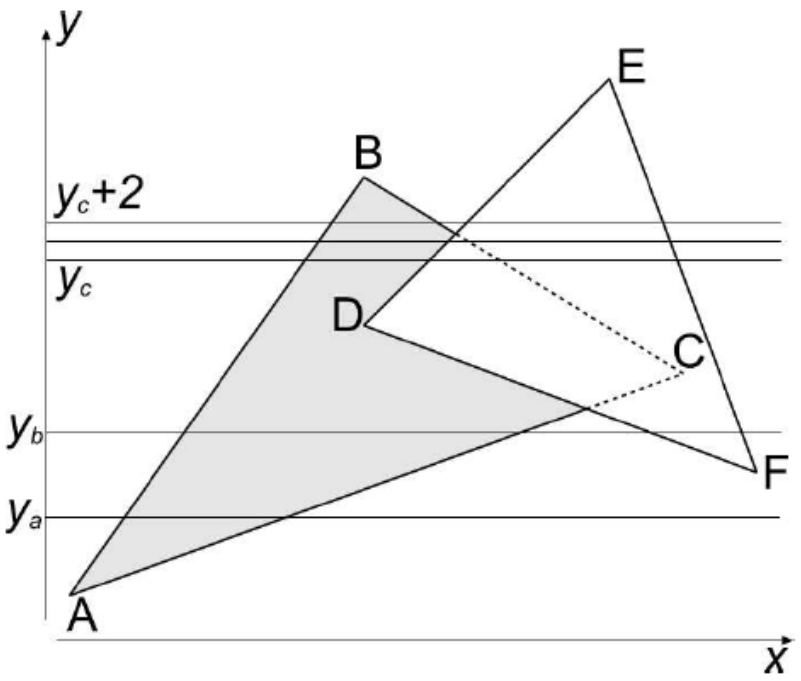


Algoritmo di Scan Line

Algoritmo:

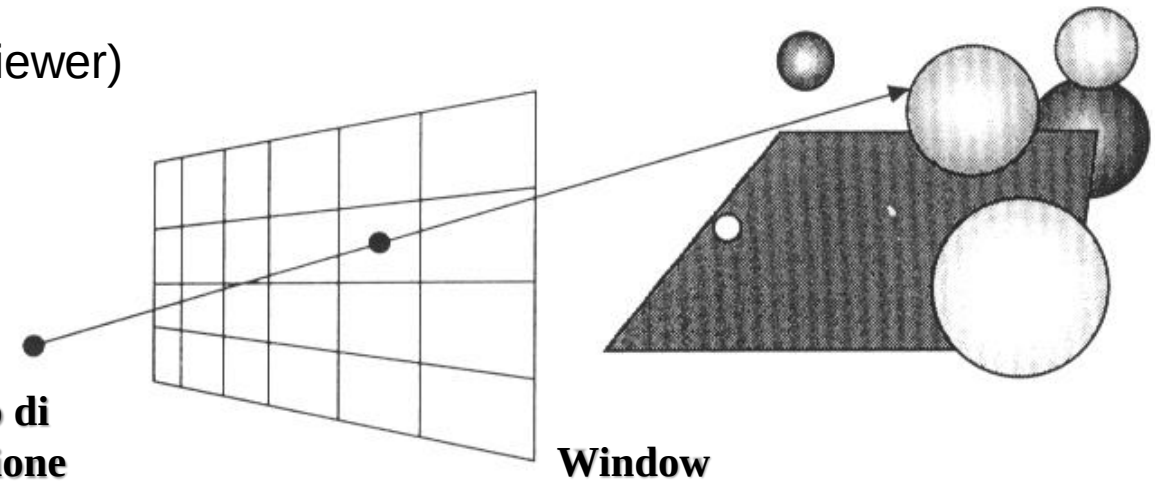
Rispetto alla tecnica *z-buffer* il numero di controlli sulla profondità è molto inferiore;

L'algoritmo *z-buffer* non necessita di strutture dati la cui dimensione dipende dalla complessità della scena.



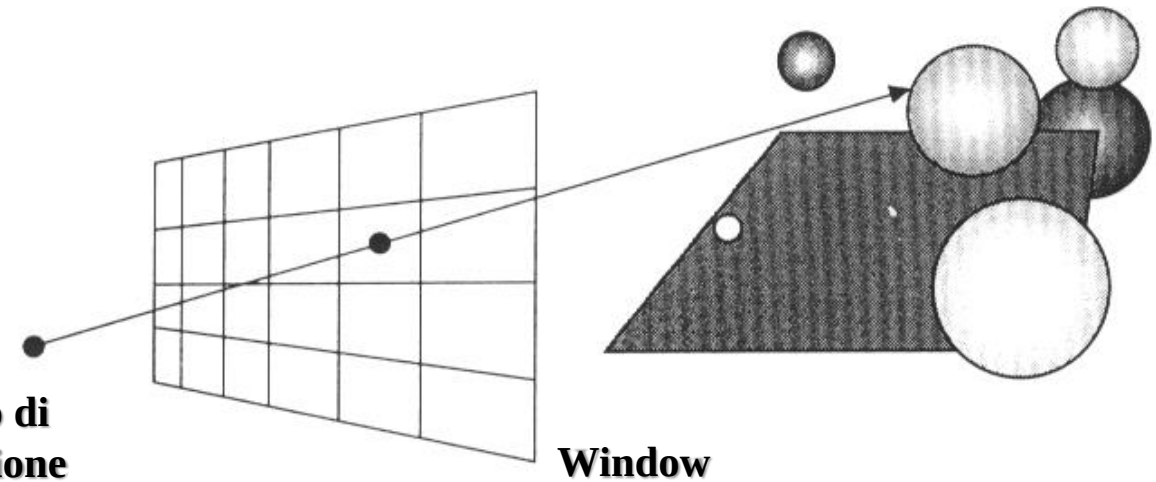
Ray Casting

- Determina la visibilità delle superfici tracciando raggi di luce immaginari dall'osservatore agli oggetti nella scena
- Algoritmo image-space
- Per ogni pixel → (x,y) window
  - viene tracciata retta di proiezione (passante per il centro)
  - intersezione retta – oggetti → dà le distanze
  - oggetto a distanza minimia → dà colore per il pixel
- Usa:
  - centro di proiezione (viewer)
  - view plane → window
- Basato su:
  - intersezioni



Ray Casting

```
Select the center of projection and viewplane window
for (each scan line) {
    for (each pixel on the scan line) {
        Find ray from the center of projections through the pixel
        for (each object in the scene) {
            if (object is intersected and intersection is closest
                    considered so far) {
                Record the intersection and the object name
            }
        }
        Set pixel color to that of the closest intersected object
    }
}
```



Ray Casting

Calcolo delle intersezioni raggio-oggetto (non banale), vediamo due casi semplici
Si scrive l'equazione del raggio in forma parametrica

$$x = x_0 + t(x_1 - x_0), y = y_0 + t(y_1 - y_0), z = z_0 + t(z_1 - z_0)$$

$$\Delta x = (x_1 - x_0), \Delta y = (y_1 - y_0), \Delta z = (z_1 - z_0)$$

$$x = x_0 + t\Delta x, y = y_0 + t\Delta y, z = z_0 + t\Delta z$$

- Intersezione **raggio-sfera**

sfera di raggio *r* centrata in *(a,b,c),* in forma implicita

$$(x-a)^2 + (y-b)^2 + (z-c)^2 = r^2$$

Sostituendo

$$(x_0 + t\Delta x - a)^2 + (y_0 + t\Delta y - b)^2 + (z_0 + t\Delta z - c)^2 = r^2$$

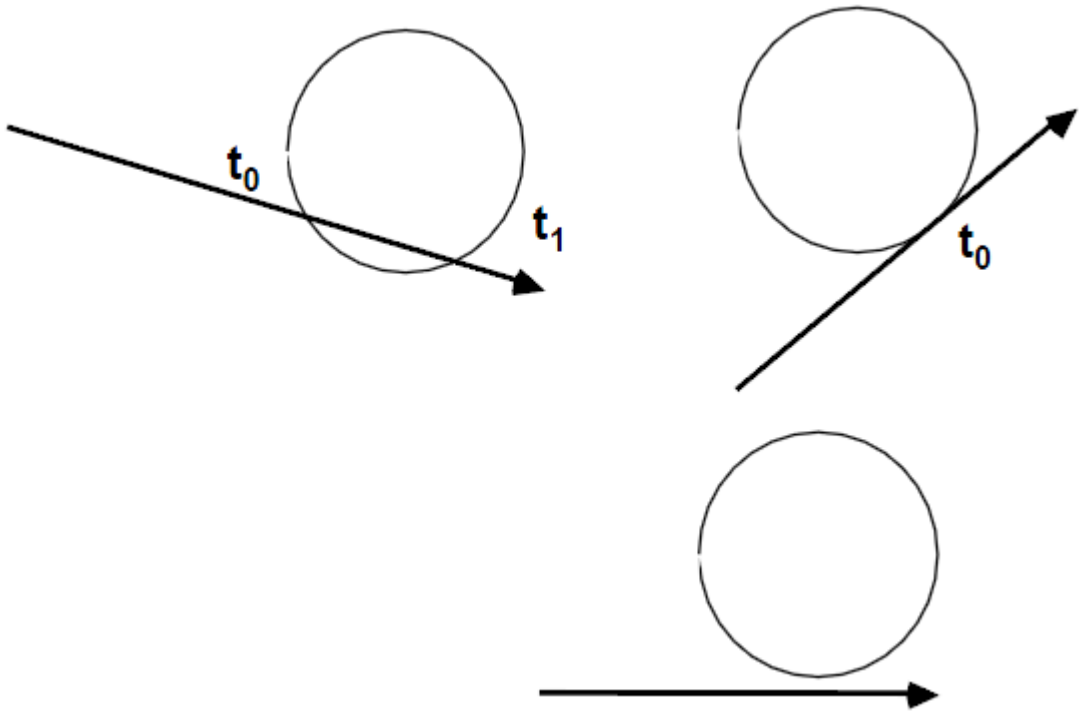
e quindi

$$(\Delta x^2 + \Delta y^2 + \Delta x^2)t^2 + 2t[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] +$$
$$+ (x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2 = 0$$

equazione di secondo grado in *t*

Ray Casting

- Possibili soluzioni



Ray Casting

- Intersezione **raggio-poligono**

Si scrivono le equazioni parametriche del piano a cui appartiene il poligono e del raggio (retta passante per un punto O con direzione D) e poi si calcola l'intersezione

$$N \cdot P + d = 0$$

$$P = O + Dt$$

$$t = -\frac{N \cdot O + d}{N \cdot D}$$

Per capire se il punto di intersezione giace sul poligono si proiettano il poligono e il punto ortogonalmente su un piano coordinato e si esegue un test di appartenenza in 2D