



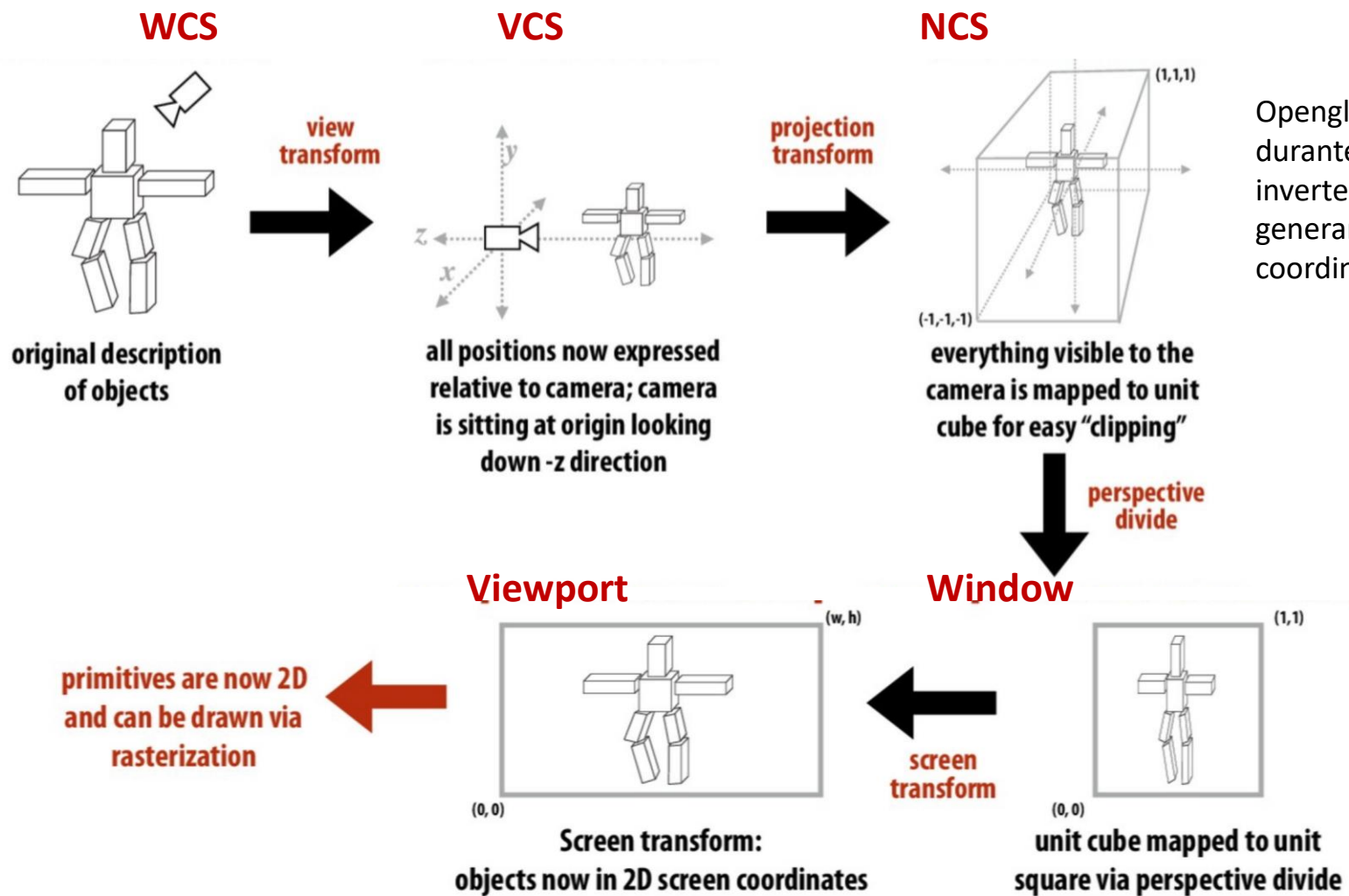
---

# La pipeline di rendering

## Il sottosistema raster



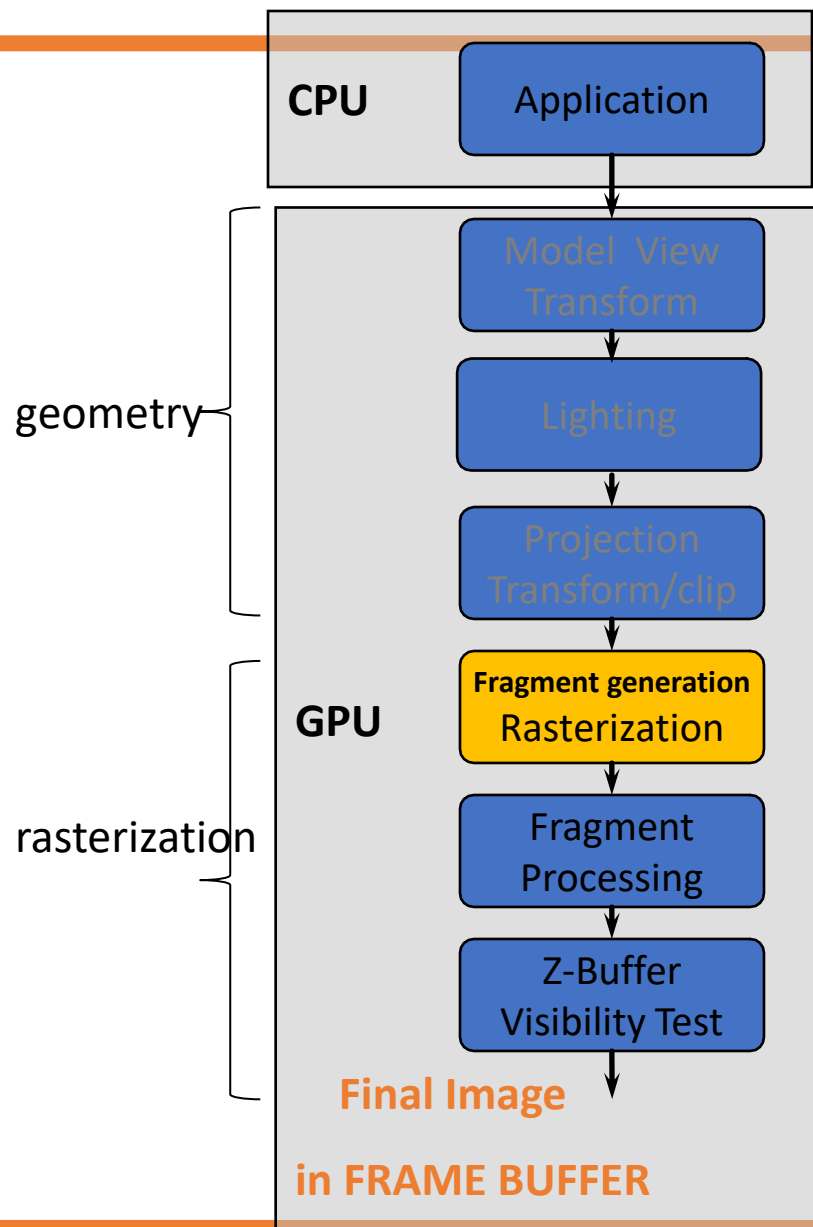
# Geometry stage



Opengl, per convenzione, durante la proiezione, inverte l'asse z del cuboide generando un left handed coordinate system

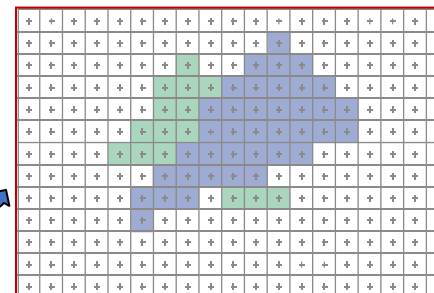
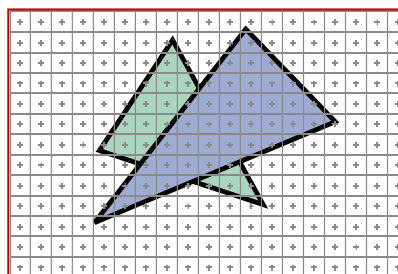


# Scan Conversion (Rasterization)



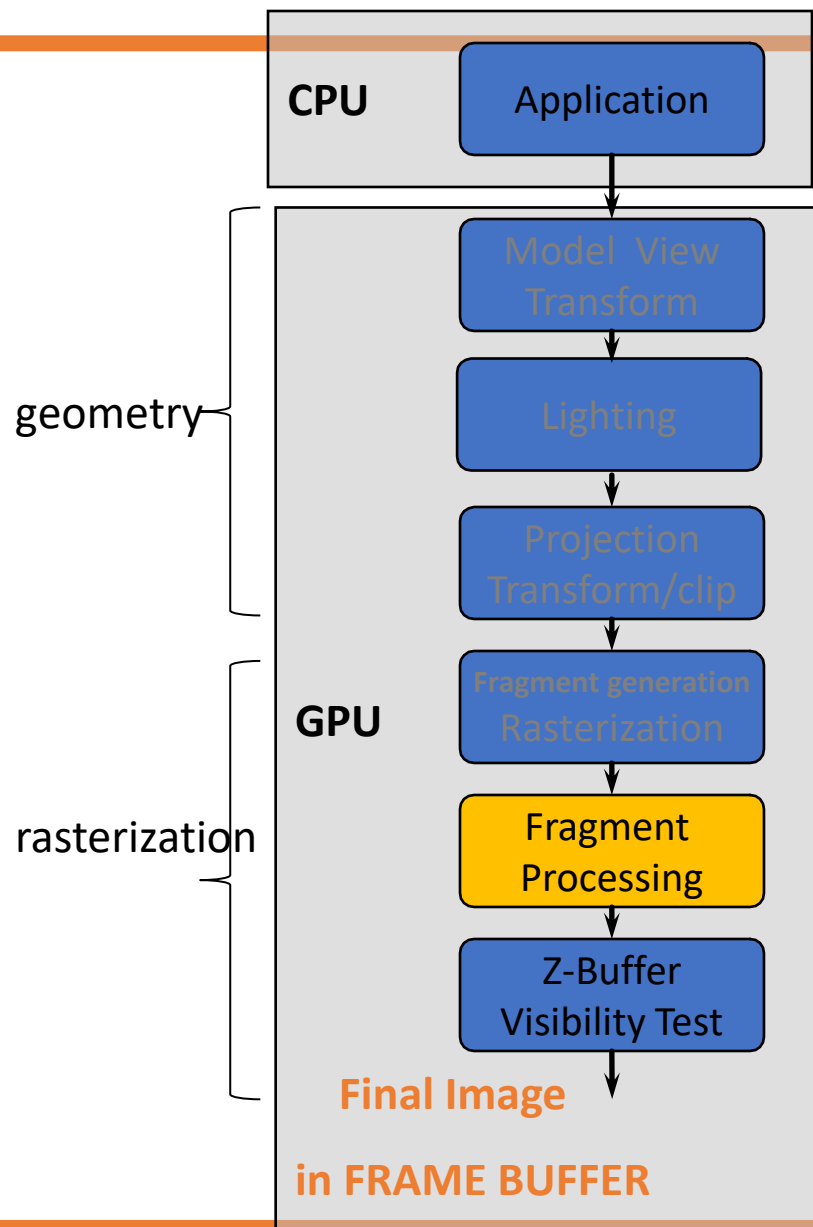
← *Vertici*

- Determina quali pixels sono interni alla primitiva specificata da un insieme di vertici
- Produce un insieme di frammenti.
- **I Frammenti** hanno una posizione (pixel location) e altri attributi, come colore e coordinate di texture che vengono determinati interpolando i valori sui vertici.



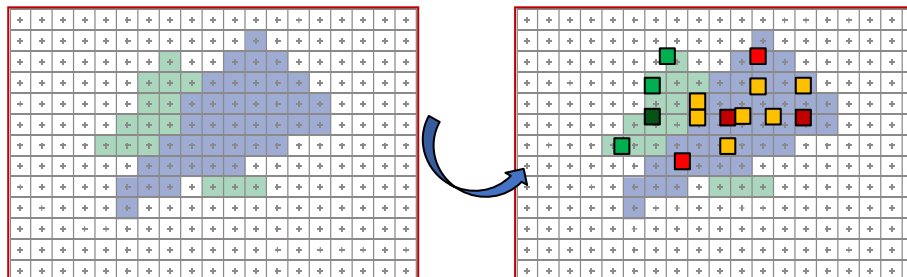


# Fragment processing



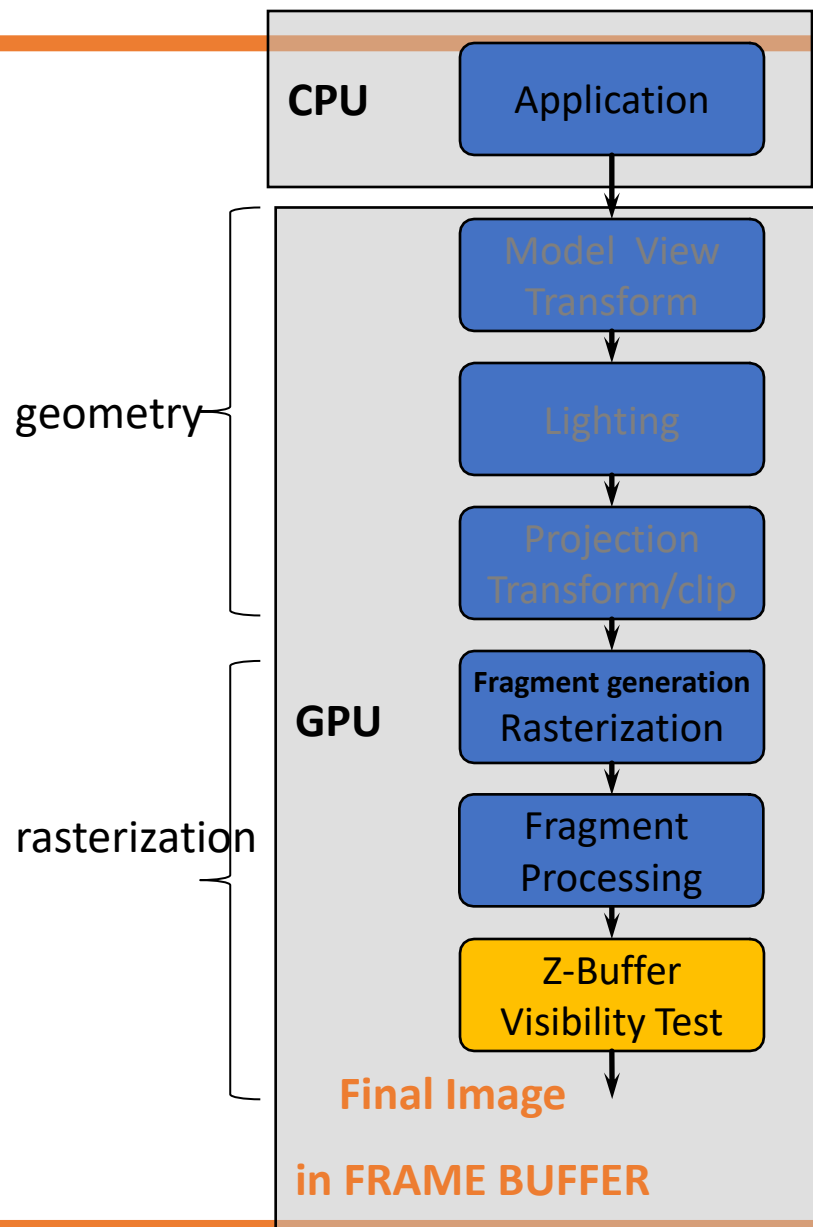
- **Elaborazione dei Frammenti**

I colori dei pixel vengono determinati successivamente usando colori, texture ed altre proprietà dei vertici. (Texture Mapping)





# Visibility / Display

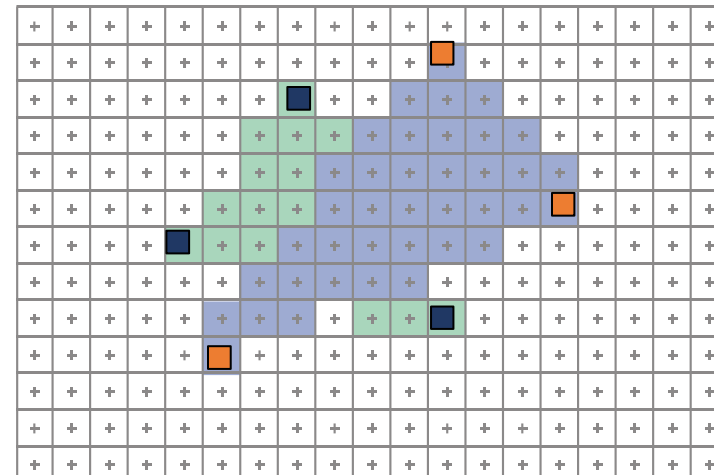
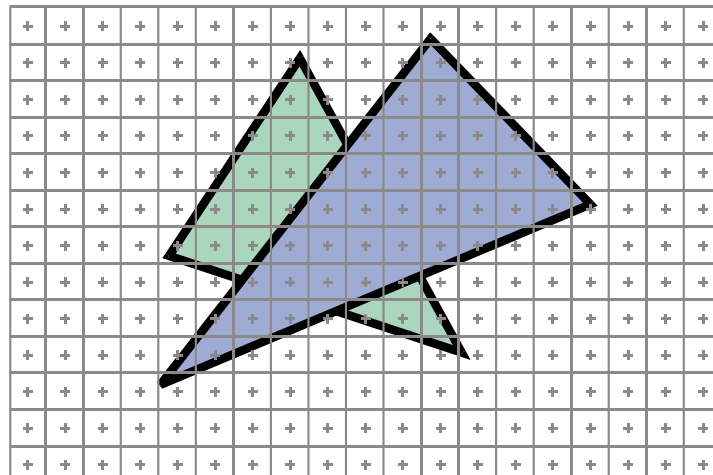


Disegna l'oggetto più vicino.  
(depth buffer)



# Rasterization

- Primitive geometriche  
(punti, line, poligoni, cerchi, etc,)
- Le primitive sono continue, lo schermo è discreto.
- Rasterization: algoritmi per la generazione efficiente dei frammenti occupati da una primitive geometrica.
  - Enumera i frammenti occupati dalla primitiva
    - Interpola i valori, detti attributi, lungo la primitiva.





# Rasterizzazione

---

- Con il termine rasterizzazione si intende propriamente il processo di **discretizzazione** che consente di **trasformare una primitiva geometrica definita in uno spazio continuo 2D nella sua rappresentazione discreta, composta da un insieme di pixel di un dispositivo di output.**
-



# Drawing

---

- Tracciare un segmento di retta all'interno di una griglia regolare di pixel che hanno distanza unitaria in ascissa e in ordinata (coordinate schermo)
-





# Casi Speciali

---

- **Linea orizzontale:**

Disegna il pixel P e incrementa il valore della coordinata x di 1 per ottenere il pixel successivo.

- **Linea verticale:**

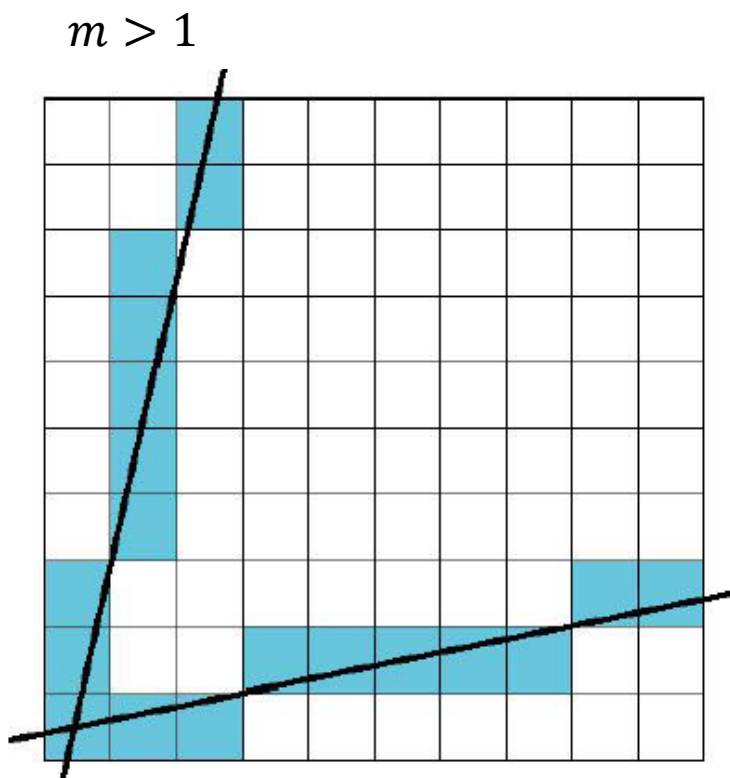
Disegna il pixel P e incrementa il valore della coordinata y di 1 per ottenere il pixel successivo.

- **Linea diagonale:**

Disegna il pixel P e incrementa entrambe le coordinate x ed y di 1 per ottenere il pixel successivo.

---

Per coefficienti angolari  $|m| > 1$   
la rasterizzazione presenta un pixel per riga.



- Per coefficienti angolari  $|m| \leq 1$  la rasterizzazione presenta un pixel per colonna.



## Algoritmo DDA (Digital Differential Analyzer)

$y = mx + q, \quad |m| \leq 1 \rightarrow$  il prossimo pixel si accende per  $x+1$

$$y_{new} = m(x + 1) + q = mx + q + m \equiv y + m$$

$|m| \geq 1 \rightarrow$  il prossimo pixel si accende per  $y+1$

$$x_{new} = \frac{1}{m}(y + 1 - q) = \frac{1}{m}(y - q) + \frac{1}{m} \equiv x + \frac{1}{m}$$

*Calcola  $m = (y1 - y0) / (x1 - x0)$*

*Se  $|m| \leq 1$  allora*

*Poni  $y = y0$*

*Ripeti per  $x = x0; x \leq x1; x++$*

*Accendi il punto  $(x, \text{round}(y))$*

*$y += m$*

*Altrimenti*

*Poni  $x = x0$*

*Ripeti per  $y = y0; y \leq y1; y++$*

*Accendi il punto  $(\text{round}(x), y)$*

*$x += (1/m)$*



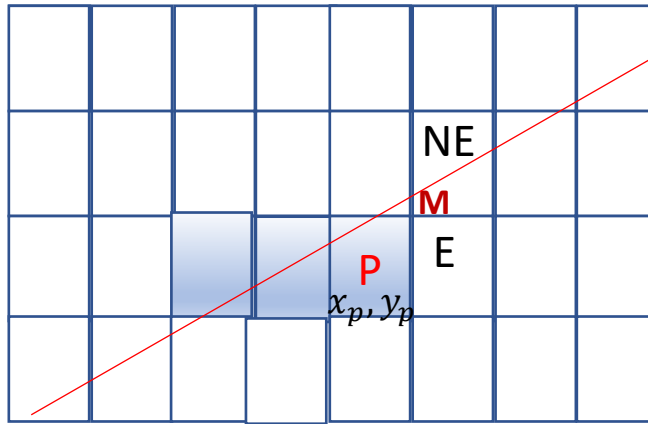
# Algoritmo di Bresenham o Mid-Point

---

- ❖ L'algoritmo di Bresenham (detto anche *algoritmo del punto di mezzo*) risolve il problema dell'errore introdotto dall'uso di aritmetica floating point nell'algoritmo DDA;
  - ❖ L'algoritmo di Bresenham fa uso solo di operazioni in aritmetica intera;
  - ❖ E' ancora un algoritmo di tipo differenziale; fa uso delle informazioni calcolate per individuare il pixel al passo  $i$  per individuare il pixel al passo  $i+1$ .
  - ❖ Nel seguito, ancora l'ipotesi non restrittiva  $m < 1$ .
-



# Algoritmo di Bresenham



$$NE(x_p + 1, y_p + 1)$$

$$E(x_p + 1, y_p)$$

Si assumerà inizialmente di considerare  $m > 0$  e, in particolare,  $m \in (0, 1]$ .

La soluzione per le altre pendenze si può ottenere semplicemente per riflessione della geometria illustrata in figura rispetto agli assi principali.

Si consideri di aver già acceso i pixel della primitiva fino al generico punto P di coordinate  $(x_p, y_p)$ .



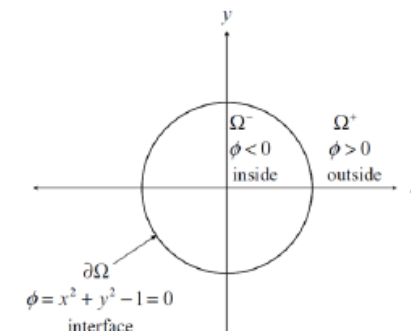
## Piccolo inciso

La forma implicita di una curva, valutata in un punto P di coordinate (x,y) , ci dà informazioni sulla posizione del punto rispetto alla curva, cioè se il punto appartiene alla curva, è situato sopra la curva o è situato sotto la curva.

Se  $F(x,y)=0$  il punto è sulla curva

Se  $F(x,y)<0$  il punto è dentro

Se  $F(x,y)>0$  il punto è fuori



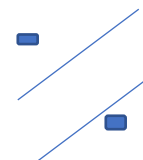
Nel caso di una retta, la forma implicita è  $F(x,y)=ax+by+c$ ;

Se valutiamo F in un punto P (x,y), allora

Se  $F(x,y)=0$  il punto è sulla retta

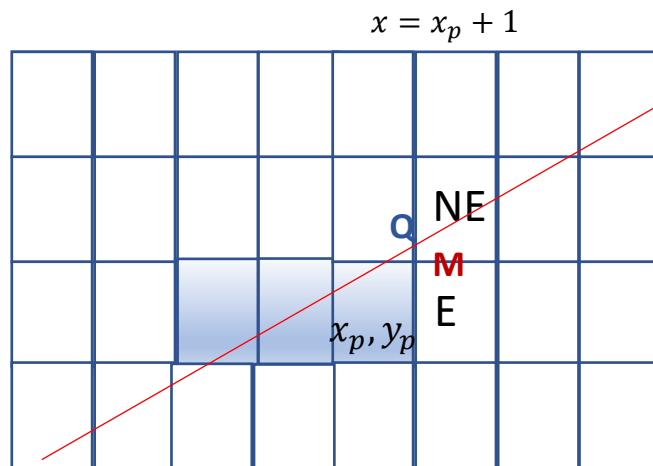
Se  $F(x,y)<0$  il punto è sopra la retta (la retta passa sotto il punto P(x,y))

Se  $F(x,y)>0$  il punto è sotto la retta. (la retta passa sopra il punto P(x,y))





# Algoritmo Midpoint



$$NE(x_p + 1, y_p + 1)$$

$$E(x_p + 1, y_p)$$

L'idea di base di questo approccio è quella di individuare, passo dopo passo, il pixel più vicino all'effettivo punto di intersezione  $Q$  tra la retta da disegnare e la retta  $x = x_p + 1$  corrispondente al successivo punto da accendere.

I candidati, ad ogni passo sono solo i due pixel  $E(x_p + 1, y_p)$  ovvero  $NE(x_p + 1, y_p + 1)$ .

Per far questo si valuterà il passaggio della retta, espressa in forma implicita

$$F(x, y) = ax + by + c = 0, \text{ per il punto medio } M(x_p + 1, y_p + 1/2)$$



# Algoritmo Midpoint

---

L'implementazione è, anche in questo caso, incrementale.

Si consideri l'equazione della retta in forma esplicita:

$$y=mx+q=(y_1 - y_0)/(x_1 - x_0)x+q=(dy/dx)x+q.$$

Allora la forma implicita si ottiene come:  $F(x, y) = dy \cdot x - dx \cdot y + dx \cdot q = ax + by + c = 0$

Si definisce una variabile di decisione  $d$

$$d = F(x_M, y_M) = F\left(x_p + 1, y_p + \frac{1}{2}\right) = a(x_p + 1) + b\left(y_p + \frac{1}{2}\right) + c$$

Per  $d > 0$  si accenderà NE, mentre per  $d \leq 0$  si accenderà E.

L'algoritmo consiste nel calcolare incrementalmente la sola variabile di decisione.

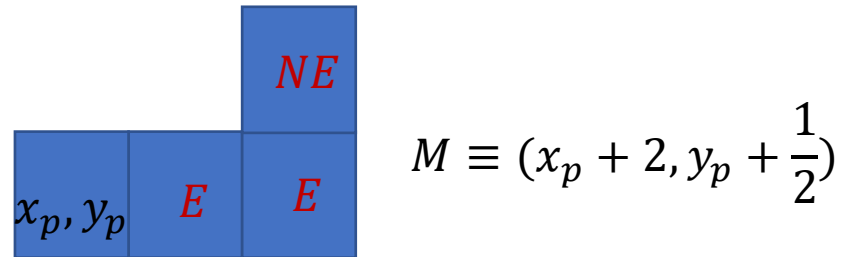
---





# Algoritmo Midpoint

Se ad un certo passo  $p+1$  si accende il pixel  $E$ , allora il nuovo valore della variabile  $d_{new}$ , noto  $d_{old}$  al passo precedente  $p$ , si ottiene come:



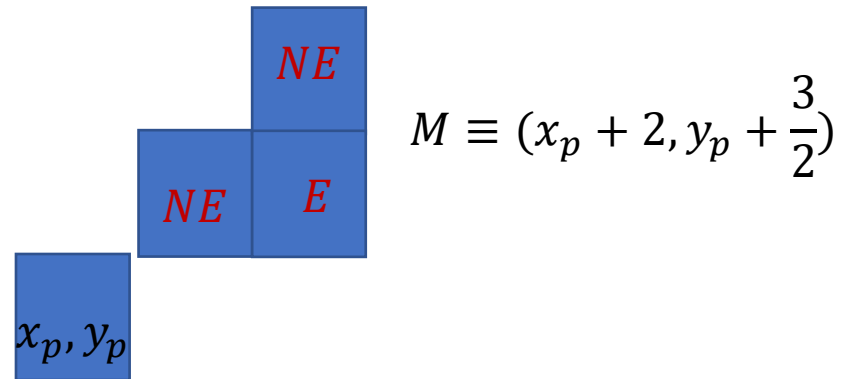
$$d = F(x_M, y_M) = F\left(x_p + 1, y_p + \frac{1}{2}\right) = a(x_p + 1) + b\left(y_p + \frac{1}{2}\right) + c$$

$$d_{new} = a(x_p + 2) + b\left(y_p + \frac{1}{2}\right) + c = a(x_p + 1) + b\left(y_p + \frac{1}{2}\right) + c + a = d_{old} + a = d_{old} + a = d_{old} + dy$$



# Algoritmo Midpoint

Alternativamente, se al passo  $p+1$  si accende il pixel  $NE$ , allora:



$$d_{new} = a(x_p + 2) + b\left(y_p + \frac{3}{2}\right) + c = a(x_p + 1) + b\left(y_p + \frac{1}{2}\right) + c + a + b = d_{old} + a + b = d_{old} + (dy - dx)$$



# Algoritmo Midpoint

---

Il processo si inizializza calcolando il valore di partenza. Partendo da  $(x_0, y_0)$  calcoliamo  $d_{start}$ :

$$\begin{aligned}d_{start} &= F\left(x_0 + 1, y_0 + \frac{1}{2}\right) = a(x_0 + 1) + b\left(y_0 + \frac{1}{2}\right) + c = ax_0 + by_0 + c + a + \frac{b}{2} = \\&= F(x_0, y_0) + a + \frac{b}{2} \equiv a + \frac{b}{2} = dy - \frac{dx}{2}\end{aligned}$$

Per eliminare la divisione ed utilizzare solo l'aritmetica intera si può far riferimento alla nuova variabile di decisione  $2d$  che deve solamente essere confrontata con il valore 0 ad ogni passo. Quindi:

$$d_{start} = 2dy - dx$$

$$\text{deltaE} = 2dy$$

$$\text{deltaNE} = 2(dy - dx)$$



# Algoritmo Midpoint

---

```
Calcola dx=x1-x0;  
Calcola dy=y1-y0;  
Calcola dstart=2dy-dx;  
Calcola deltaNE=2(dy-dx);  
Calcola deltaE=2dy;  
x=x0; y=y0; d=dstart;  
Ripeti finché x<x1  
    Se d<=0 allora  
        d+=deltaE;  
        x++;  
    Altrimenti  
        d+=deltaNE;  
        x++;  
        y++;  
    Accendi il pixel (x,y)
```

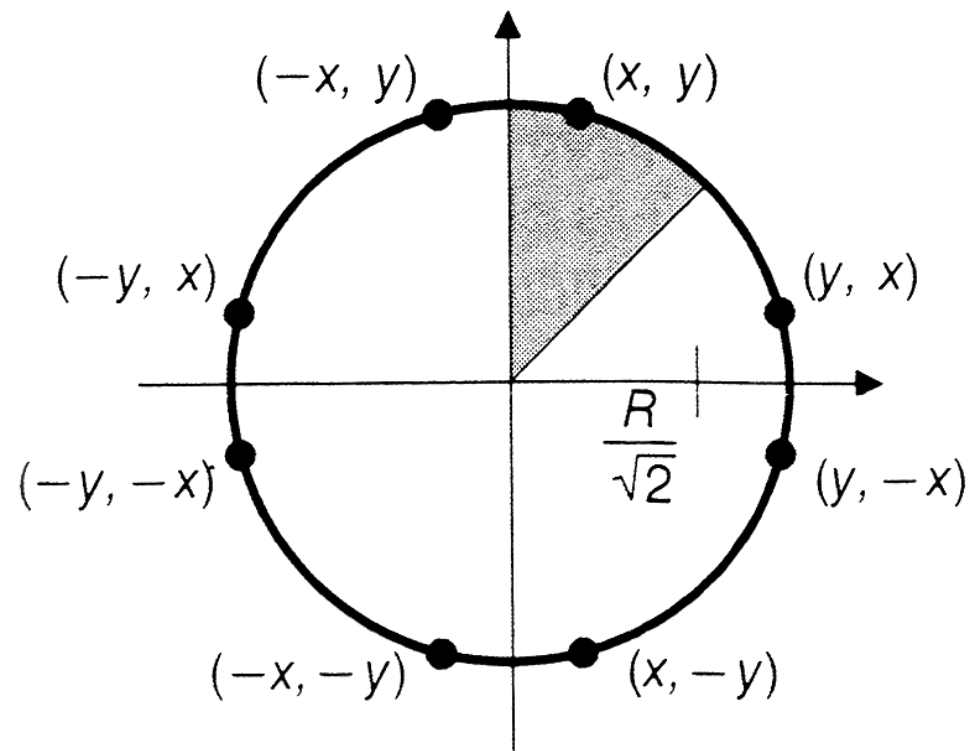
---



# Midpoint per circonferenze

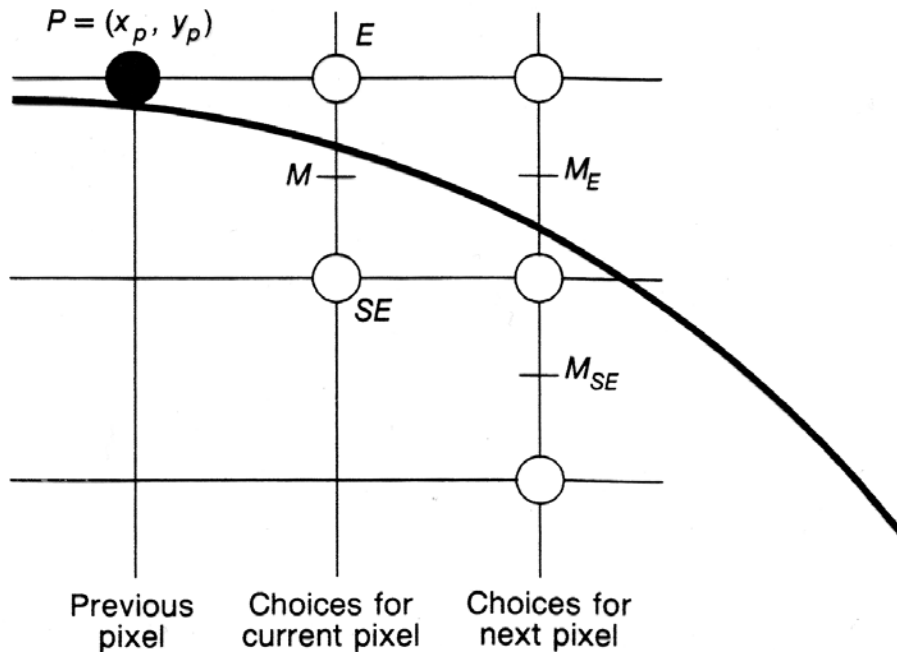
- L'equazione implicita di una circonferenza  
 $F(x,y)=x^2+y^2-R^2=0$   
fornisce 2 valori di  $y$  per dato  $x$ .

- Si calcola il midpoint per il primo ottante  
 $x \in [0, R/\sqrt{2}]$  e poi si replica per gli altri





# Midpoint per circonferenze



Partendo dal punto di valutazione della circonferenza  $P=(x_p, y_p)$  si sceglie il successivo pixel da accendere sulla base dell'analisi del punto medio

$M (x_p+1, y_p - 1/2)$  tra i pixel  $E$  ed  $SE$  di  $P$ .

La variabile di decisione sarà definita come l'equazione implicita della circonferenza valutata in  $M$ .

$$d = F(x_M, y_M) = (x_p + 1)^2 + \left(y_p - \frac{1}{2}\right)^2 - R^2$$

*Se  $F(M) \geq 0$  //  $M$  giace o è esterno alla circonferenza*

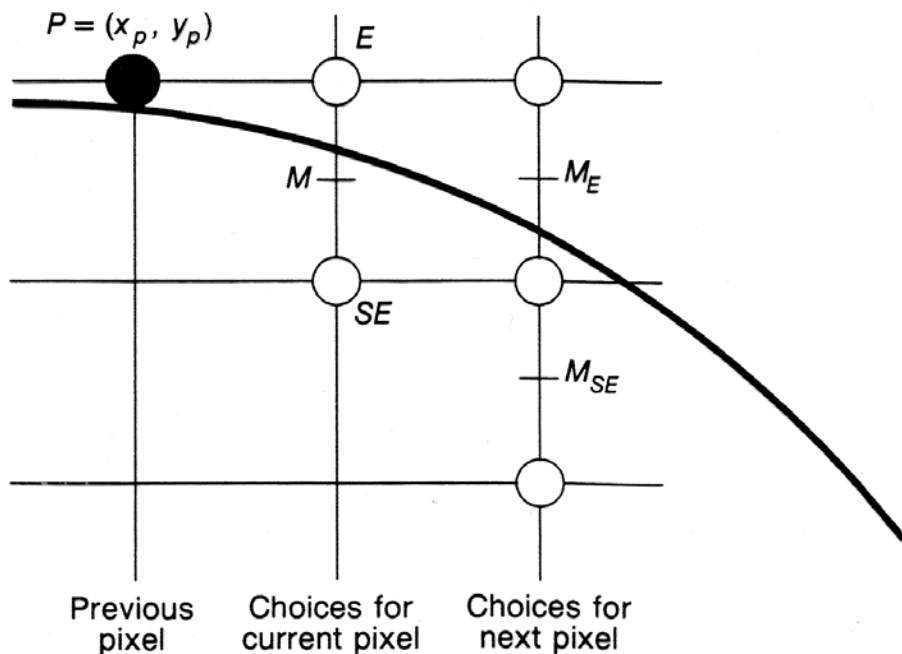
*Accendi  $SE$*

*Altrimenti //  $F(M) < 0$  cioè  $M$  è interno alla circonferenza*

*Accendi  $E$*



# Midpoint per circonferenze



mentre, nel caso di accensione di SE:

Procedendo in maniera analoga a quanto fatto per il tracciamento delle rette, si possono calcolare i due incrementi della variabile di decisione al generico passo  $p+1$ , noto il valore  $d_{old}$  della variabile di decisione al passo  $p$  nei due casi in cui venga acceso il pixel  $E$  o  $SE$ .

Nel caso di accensione, al passo  $p$ , di  $E$ :

$$d_{new} = F(x_p + 2, y_p - 1/2) = d_{old} + (2x_p + 3)$$

$$d_{new} = F(x_p + 2, y_p - 3/2) = d_{old} + 2(x_p - y_p) + 5$$

$d_{start}$  si valuta nel punto  $(0, R)$  che è la sommità dell'ottante.

Per mantenere l'uso dell'aritmetica intera, si considera  $h = d - 1/4 \rightarrow h_{start} = 1 - R$   
 $h$  si confronta sempre con 0

$$\begin{aligned} d_{start} &= F\left(0 + 1, R - \frac{1}{2}\right) = F\left(1, R - \frac{1}{2}\right) = \\ &= 1 + \left(R - \frac{1}{2}\right)^2 - R^2 = 1 + R^2 - R + \frac{1}{4} - R^2 = \frac{5}{4} - R \end{aligned}$$



# Midpoint per circonferenze

---

```
void MidpointCircle (int radius, int value)
/* Assumes center of circle is at origin. Integer arithmetic only */
{
    int x = 0;
    int y = radius;
    int d = 1 - radius;
    CirclePoints (x, y, value);

    while (y > x) {
        if (d < 0)          /* Select E */
            d += 2 * x + 3;
        else {              /* Select SE */
            d += 2 * (x - y) + 5;
            y--;
        }
        x++;
        CirclePoints (x, y, value);
    } /* while */
} /* MidpointCircle */
```

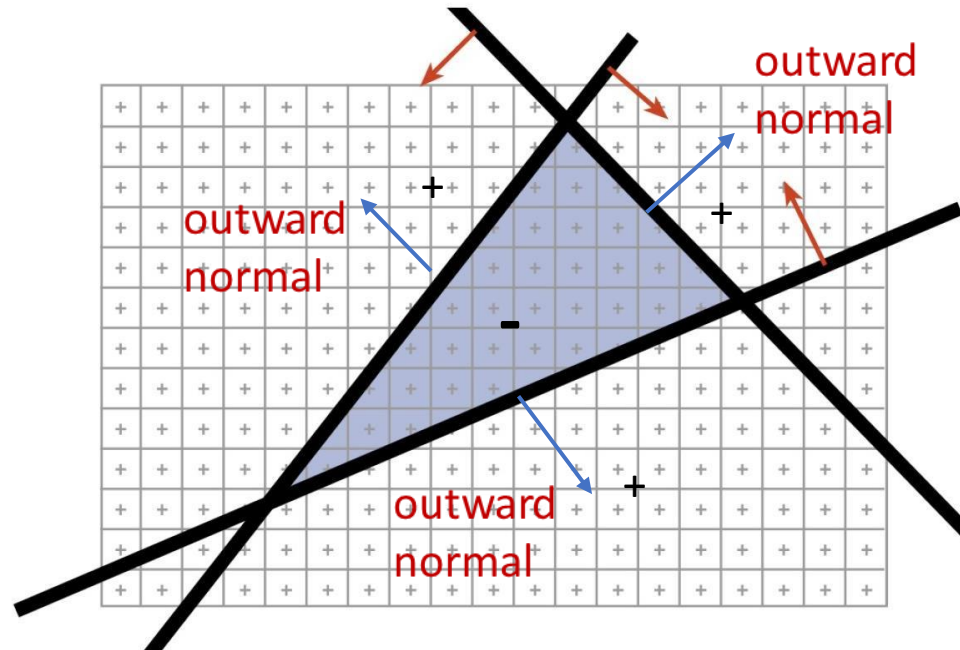
---





# Rasterizzazione di un triangolo

- Gli edge 3D del triangolo vengono proiettati su un piano.
- L'interno del triangolo è l'insieme di punti che si trova all'interno di tutti e tre i semispazi negativi definiti da queste linee





# Inside Triangle Test

- Un punto è all'interno di un triangolo se si trova nel semispazio negativo di tutte e tre gli edge di confine, ciò implica che siano soddisfatte le seguenti condizioni:
- I vertici del triangolo sono ordinati in senso antiorario
- Il punto deve trovarsi a sinistra di ogni edge di confine

1) Ricavare la forma implicita dell'equazione dell'edge tra i vertici

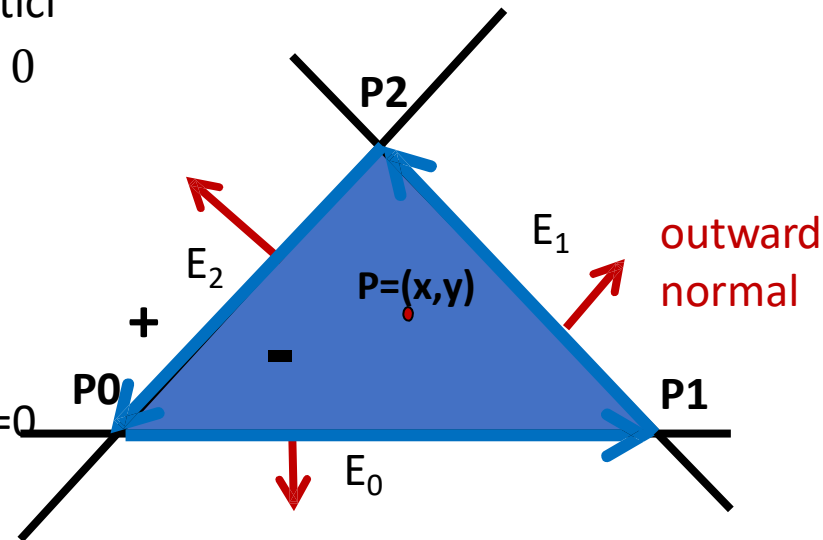
$$P_0 \equiv (x_0, y_0) \text{ e } P_1 \equiv (x_1, y_1) \quad E_0(x, y) = ax + by + c = 0$$

a partire dalla forma esplicita  $\frac{(x-x_0)}{(x_1-x_0)} = \frac{(y-y_0)}{(y_1-y_0)}$

$$(x - x_0)(y_1 - y_0) - (x_1 - x_0)(y - y_0) = 0$$

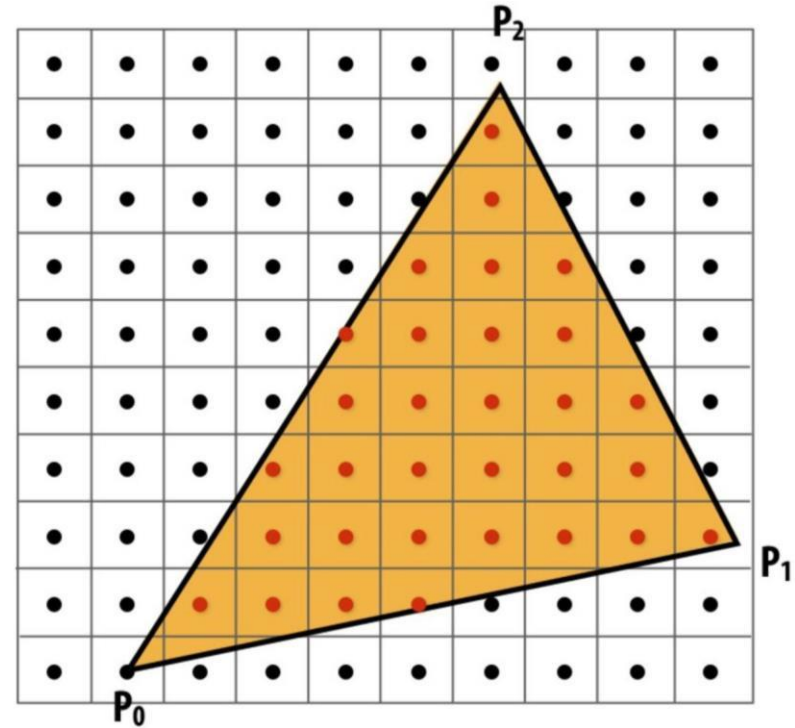
$$\underbrace{x(y_1 - y_0)}_a + \underbrace{y(x_0 - x_1)}_b - \underbrace{x_0(y_1 - y_0) + y_0(x_1 - x_0)}_c = 0$$

Il metodo triangolo funziona solo per i poligoni convessi!!



$$(x, y) \text{ interno al triangolo} \Leftrightarrow E_i(x, y) \leq 0, \forall i = 0, 1, 2$$

- I punti campionati interni al triangolo sono rossi



- Tutte le GPU sono fornite di hardware special-purpose hardware per realizzare in maniera efficiente **inside triangle test**.



# Rasterization Pseudocode

Per ogni triangolo

Calcola la proiezione dei vertici, calcola gli  $E_i$  ( $a_i, b_i, c_i$ )

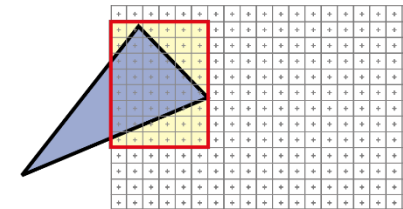
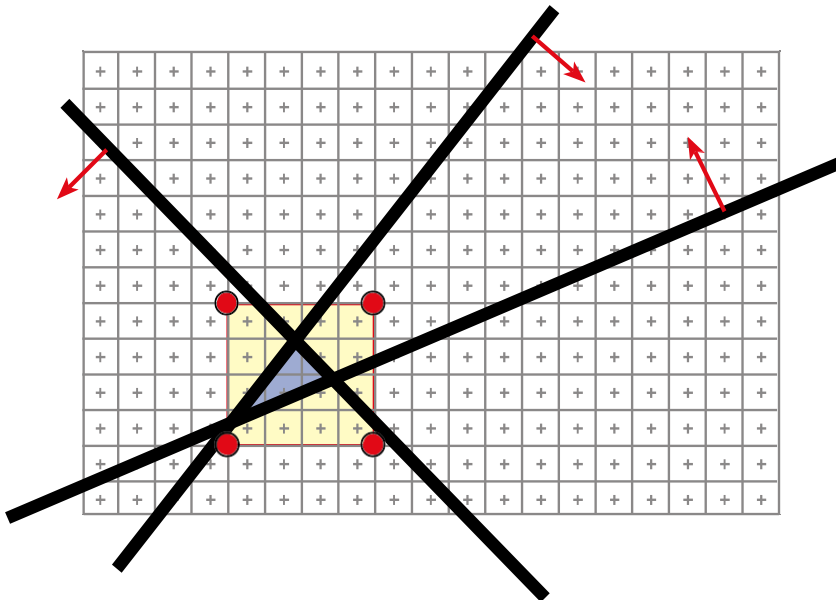
Calcola il bounding box (bbox) sullo schermo, taglia il bbox rispetto ai limiti dello schermo

Per ogni pixel  $(x, y)$  nel bbox

Valuta le funzioni edge  $E_i$ ,  $i=1,2,3$  nel centro del pixel

If all  $E_i < 0$     % il pixel è interno

Framebuffer[ $x, y$ ] = triangleColor



Come selezionare il bounding box?

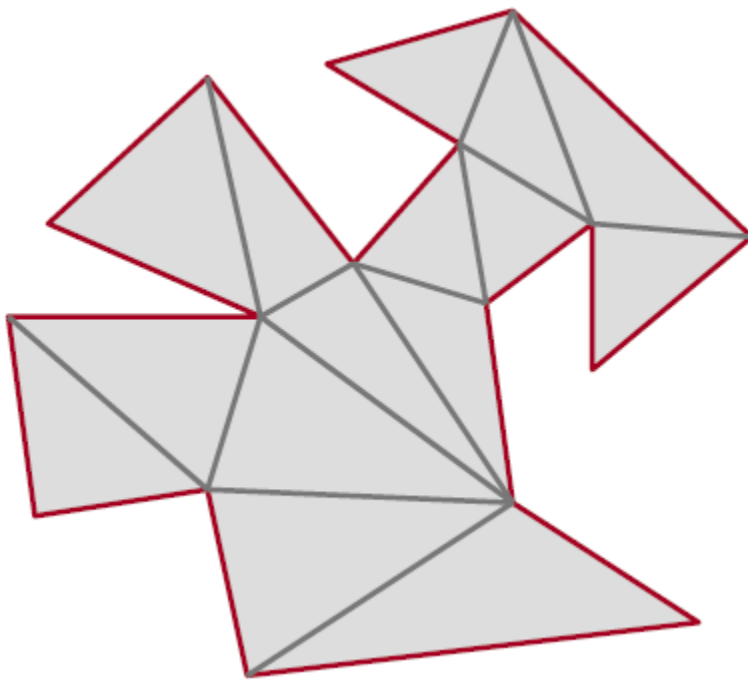
Xmin, Xmax, Ymin, Ymax dei vertici del triangolo proiettato.



# Rasterizzazione per poligoni concavi: tessellator

---

Converte ogni cosa in triangoli e poi  
fa la scan-conversion dei triangoli





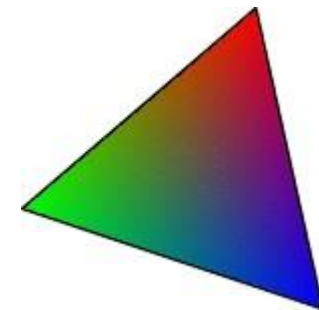
# Framebuffer[x,y] = Color(?)

---

- Memorizziamo i dati (come il colore, ecc.) sui vertici dei triangoli e successivamente utilizziamo l'interpolazione per calcolare i valori di questi dati all'interno del triangolo

Per esempio:

- Specifichiamo un colore (R,G,B) su ognuno dei vertici di un triangolo
- Per ogni pixel interno al triangolo, calcoliamo le coordinate di interpolazione per quel pixel.
- Usiamo queste coordinate interpolate, per calcolare un colore interpolato (R,G,B) per quel pixel.





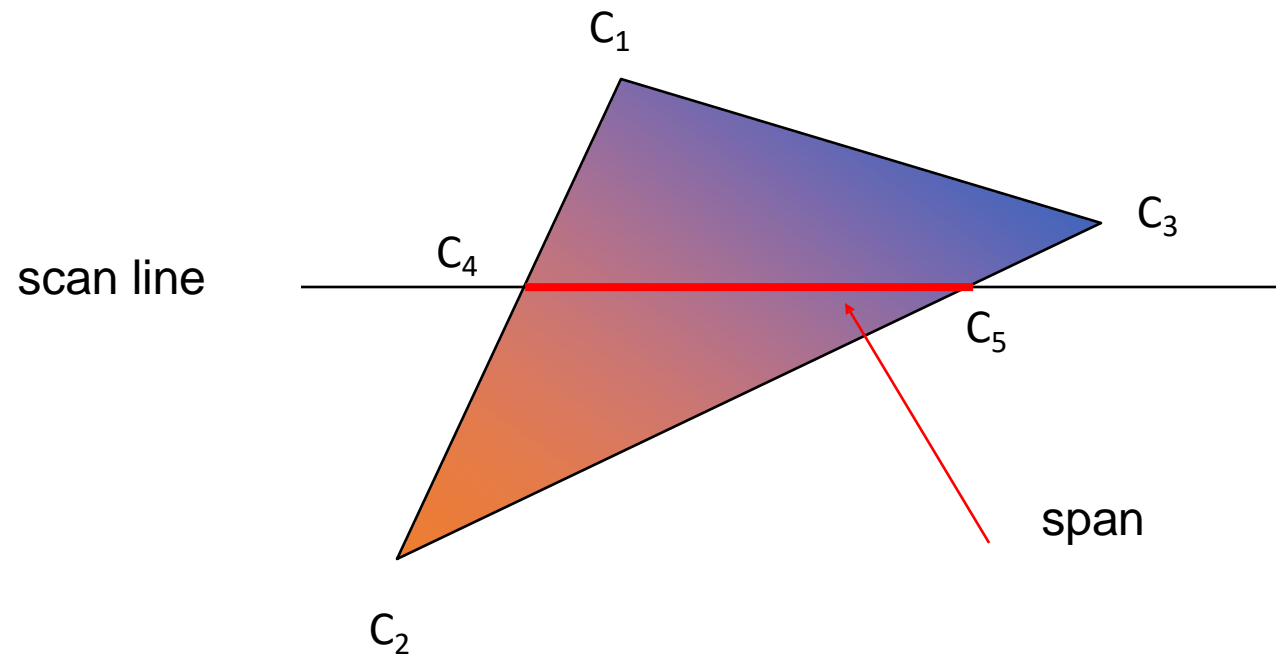
# Per-pixel color: linear Interpolation

$C_1 C_2 C_3$  specificano il colore per vertice

$C_4$  viene determinato per interpolazione tra  $C_1$  e  $C_2$

$C_5$  viene determinato per interpolazione tra  $C_2$  e  $C_3$

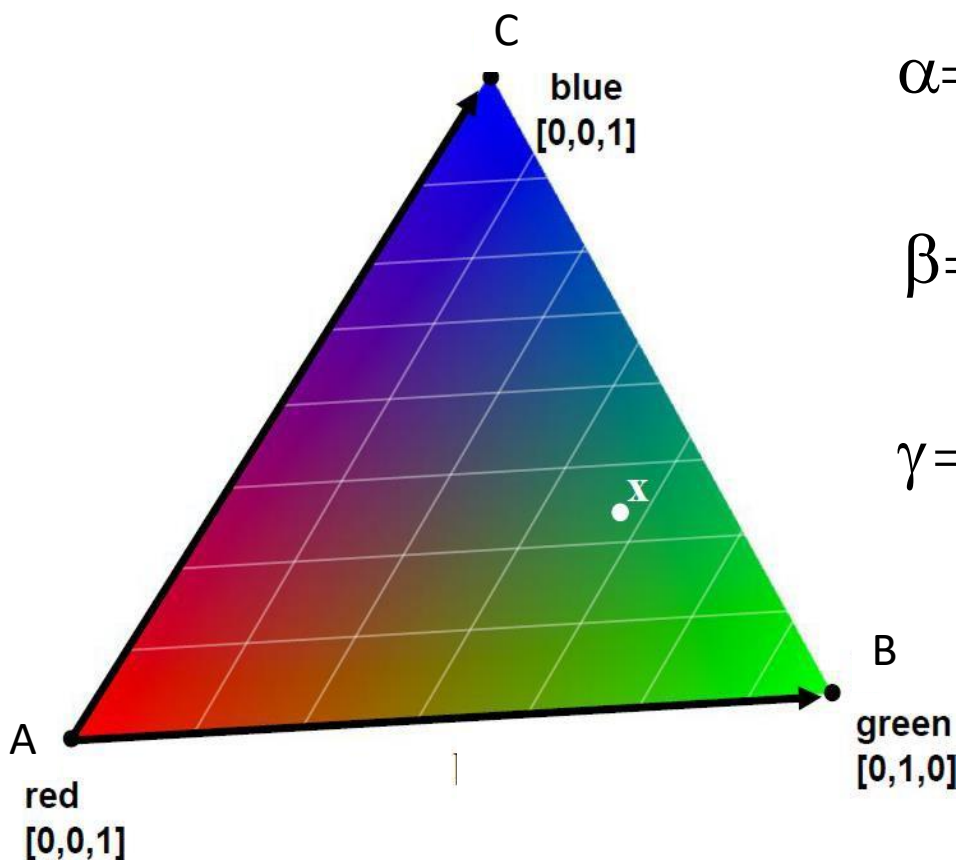
Interpoliamo i valori tra  $C_4$  e  $C_5$  lungo una span,





# Per-Pixel color: barycentric interpolation

Il colore di un triangolo in un punto?



$$\alpha = \frac{\text{area}_{xBC}}{\text{area}_{ABC}} = \frac{\frac{1}{2} E_{BC}(x)}{\frac{1}{2} E_{BC}(A)} = \frac{E_{BC}(x)}{E_{BC}(A)}$$

$$\beta = \frac{E_{CA}(x)}{E_{CA}(B)}$$

$$\gamma = \frac{E_{AB}(x)}{E_{AB}(C)}$$

Il colore in  $x$  è la combinazione affine dei colori sui tre vertici dei triangoli

$$x_{color} = \alpha A_{color} + \beta B_{color} + \gamma C_{color}$$





# Per-pixel attributes

---

Interpoliamo i colori

$$\blacksquare R = \alpha_0 R_0 + \alpha_1 R_1 + \alpha_2 R_2$$

$$\blacksquare G = \alpha_0 G_0 + \alpha_1 G_1 + \alpha_2 G_2$$

$$\blacksquare B = \alpha_0 B_0 + \alpha_1 B_1 + \alpha_2 B_2$$

Interpoliamo i vettori normali

$$\blacksquare N = \alpha_0 N_0 + \alpha_1 N_1 + \alpha_2 N_2$$

Interpoliamo i valori di profondità

$$\blacksquare z = \alpha_0 z_0 + \alpha_1 z_1 + \alpha_2 z_2$$

Interpoliamo le coordinate di texture.

$$\blacksquare u = \alpha_0 u_0 + \alpha_1 u_1 + \alpha_2 u_2$$

$$\blacksquare v = \alpha_0 v_0 + \alpha_1 v_1 + \alpha_2 v_2$$

---



## Coordinate baricentriche di un punto nello spazio affine

---

- Una combinazione affine è una combinazione lineare di punti con coefficienti che hanno somma 1.

$$P = \alpha_0 P_0 + \alpha_1 P_1 + \dots + \alpha_N P_N \quad \text{con } \alpha_0 + \alpha_1 + \dots + \alpha_N = 1.$$

*I coefficienti  $\alpha_0, \alpha_1, \dots, \alpha_N$ , sono le coordinate baricentriche (affini) di  $P$  nello spazio affine.*

---

- La combinazione affine di due punti distinti descrive la retta passante per i due punti.
- Siano  $Q$  e  $R$  due punti dello spazio affine reale e sia  $v$  il vettore da essi individuato:

$$v = R - Q$$

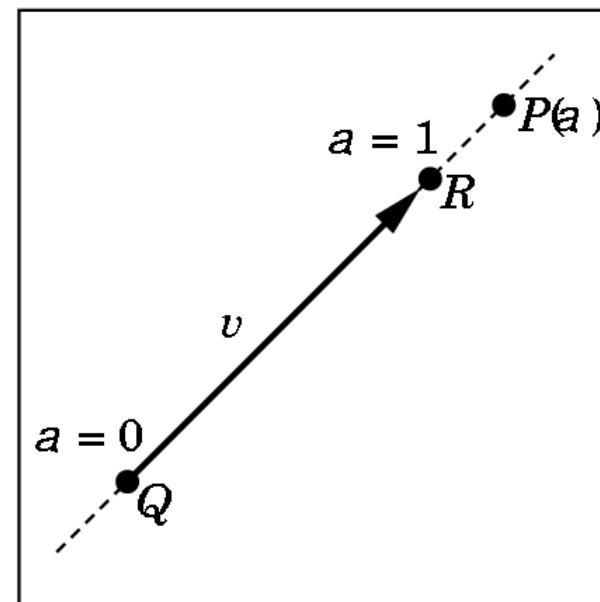
- Consideriamo la loro combinazione affine,  $\alpha + \beta = 1$ ,

- $P = \alpha R + \beta Q, \quad \beta = 1 - \alpha$

$$P(\alpha) = \alpha R + (1 - \alpha)Q$$

$$P(\alpha) = Q + \alpha(R - Q)$$

$$P = Q + \alpha v$$



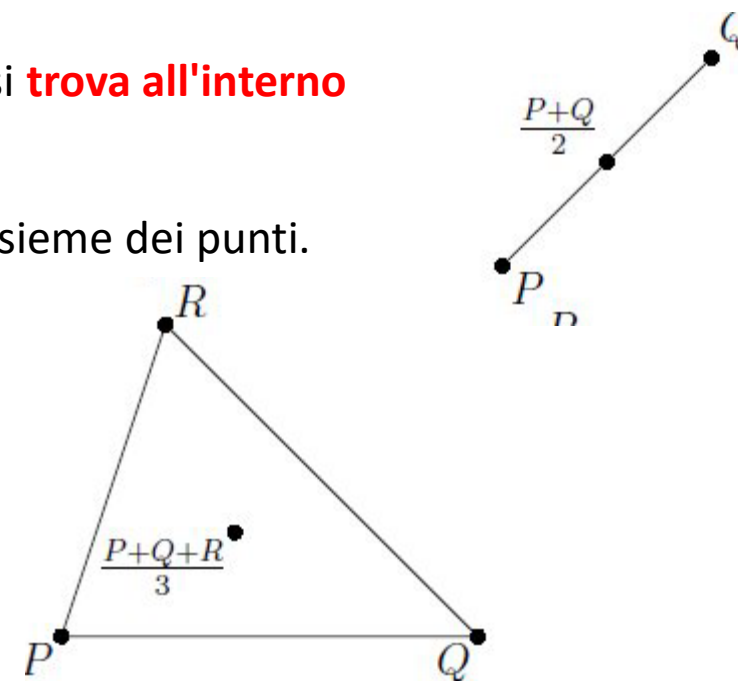


# Combinazione convessa

- La combinazione convessa è una combinazione **affine con pesi positivi**.
- Nel caso della combinazione convessa di due punti, il punto risultante giace sul segmento che congiunge i due punti. Se i pesi sono entrambi pari a 0.5, il punto risultante si trova a metà tra i due. (In figura la combinazione lineare convessa tra i punti P e Q con pesi  $\alpha_1 = \alpha_2 = 0.5$ , dà un punto che giace nel punto medio del segmento che congiunge P e Q.)
- Nel caso di n punti che formano un poligono convesso, il punto risultante si **trova all'interno del poligono**.

Se tutti i pesi sono uguali a  $1/n$ , il punto risultante si chiama **centroide** dell'insieme dei punti.

La combinazione lineare convessa dei punti P, Q ed R con pesi  $\alpha_1 = \alpha_2 = \alpha_3 = 1/3$  costituisce il centroide del triangolo.





## Coordinate baricentriche di un punto appartenente ad un triangolo

- $P = \alpha_0 A + \alpha_1 B + \alpha_2 C$
- $\alpha_0 + \alpha_1 + \alpha_2 = 1, \alpha_i > 0, i = 0, 1, 2$

$$\alpha_0 = \frac{\text{area}(PBC)}{\text{area}(ABC)}$$

$$\alpha_1 = \frac{\text{area}(PCA)}{\text{area}(ABC)}$$

$$\alpha_2 = \frac{\text{area}(PAB)}{\text{area}(ABC)}$$

