

## Grafica Computazionale

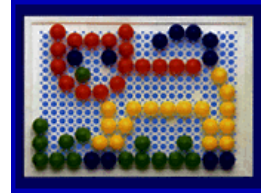
### Rasterizzazione di linee e poligoni

Fabio Ganovelli  
[fabio.ganovelli@isti.cnr.it](mailto:fabio.ganovelli@isti.cnr.it)  
a.a. 2005-2006

Diapositive a corredo del libro: "Fondamenti di Grafica Tridimensionale Interattiva"  
R. Scateni, P. Cianoni, C. Montani e R. Scopigno - McGrawHill Italia

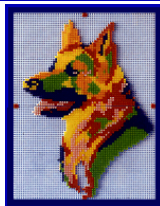
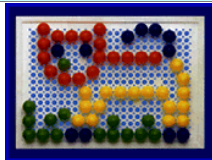
### Rasterizzazione (RST)

- ❖ Con il termine *rasterizzazione* si intende 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



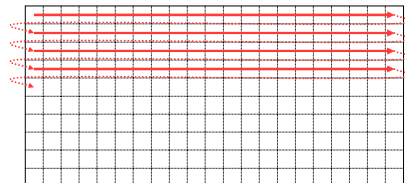
### Rasterizzazione (RST)

- ❖ Concetto molto simile al gioco che prevede un piano rettangolare con tanti fori disposti in maniera regolare e chiodini colorati;
- ❖ Se il piano è sufficientemente grande e si hanno abbastanza tonalità di colore a disposizione, l'immagine finale che ne risulta non è troppo diversa dall'originale.

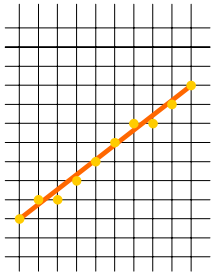


### Rasterizzazione (RST)

- ❖ Gli algoritmi di rasterizzazione si dicono anche di *scan-conversion* dal nome delle linee (scan-line) di pixel che compongono l'immagine raster sul dispositivo di output.

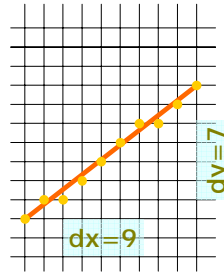


## RST segmenti



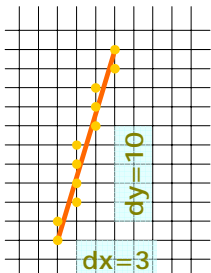
- ❖ L'algoritmo di rasterizzazione di un segmento di retta deve individuare le coordinate dei pixel che giacciono sulla linea ideale o che sono il più vicino possibile ad essa
- ❖ la sequenza di pixel deve approssimare al meglio il segmento.

## RST segmenti



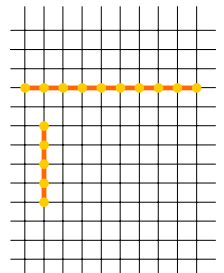
- ❖ Lo spessore minimo del segmento rasterizzato (idealmente nullo) risulterà di un pixel;
- ❖ Per coefficienti angolari  $|m| \leq 1$  la rasterizzazione presenta un pixel per ogni colonna.

## RST segmenti



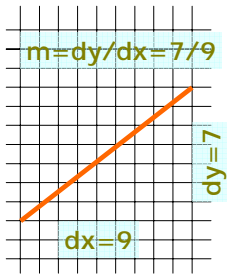
- ❖ Per coefficienti angolari  $|m| > 1$  la rasterizzazione presenta un pixel per ogni riga.

## RST segmenti



- ❖ Banale la rasterizzazione di segmenti orizzontali o verticali (sequenze di pixel su una riga o una colonna).

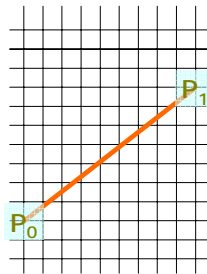
## RST segmenti - Soluzione analitica



- ❖ Senza perdere di generalità ci limiteremo nel seguito al caso di segmenti con coefficienti angolari  $m \leq 1$ ;
- ❖ L'espressione analitica della retta su cui giace il segmento è:

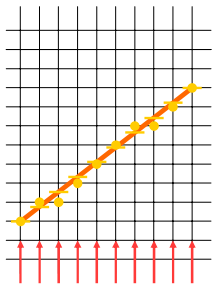
$$y = mx + B$$

## RST segmenti - Soluzione analitica



- ❖ Si intende rasterizzare il segmento di estremi  $P_0=(x_0, y_0)$  e  $P_1=(x_1, y_1)$ ;

## RST segmenti - Soluzione analitica



1. A partire dal pixel con coordinata  $x$  minima  $x_0$ :
  - 2.1 Incrementare  $x$  con passo costante uguale a 1;
  - 2.2  $\forall$  valore assunto dall'ascissa ( $x_i$ ), calcolare  $y_i$  come  $y_i = mx_i + B$ ;
  - 2.3 Arrotondare  $y_i$  all'ordinata intera più vicina.

## RST segmenti - Soluzione analitica

- ❖ L'algoritmo analitico seleziona il pixel più vicino alla linea ideale, il pixel cioè che ha distanza minima dalla linea;
- ❖ L'individuazione di un pixel implica 3 operazioni: una moltiplicazione ( $mx$ ), un'addizione ( $mx+B$ ), ed un arrotondamento ( $y_i$ ).
- ❖ La moltiplicazione può essere eliminata utilizzando una tecnica incrementale: il punto sulla retta può essere individuato sulla base del punto precedente
- ❖ L'algoritmo che ne deriva prende il nome di *algoritmo DDA* (*digital differential analyzer*)

## RST segmenti - Algoritmo DDA

- Notando che:

$$y_{i+1} = mx_{i+1} + B$$

$$y_{i+1} = m(x_i + \Delta x) + B$$

$$y_{i+1} = mx_i + B + m\Delta x$$

$$y_{i+1} = y_i + m\Delta x$$

- e che  $\Delta x = 1$

- si ha  $y_{i+1} = y_i + m$

## RST segmenti - Algoritmo DDA

- Quindi, per ogni punto della linea, abbiamo:

$$x_{i+1} = x_i + 1 \Rightarrow y_{i+1} = y_i + m$$

- Ad ogni passo è necessaria una operazione di arrotondamento con variabili (e l'aritmetica) in virgola mobile;
- L'impiego di aritmetica floating point implica introduzione e propagazione di errore.

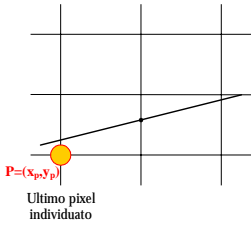
## RST segmenti - Algoritmo DDA

```
Line(int x0,
     int y0,
     int x1,
     int y1,
     int val ue)
{
    int x;
    float dy, dx, y, m; Variabili reali
    dy = y1-y0;
    dx = x1-x0;
    m = dy/dx; Inizializzazione
    y = y0;
    for (x=x0; x<=x1; x++) { Arrotondamento
        WritePixel(x, floor(0.5+y), val ue);
        y = y+m; Scrive nel frame buffer
    }
}
```

## RST segmenti - Algoritmo di Bresenham

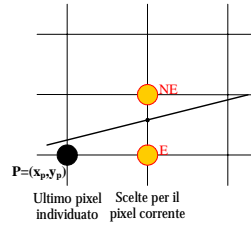
- L'algoritmo di Bresenham (detto anche *algoritmo del punto di mezzo*) risolve il problema dell'errore introdotto dall'uso di aritmetica floating point nell'algoritmo DDA;
- L'algoritmo di Bresenham fa uso solo di operazioni in aritmetica intera;
- E' ancora un algoritmo di tipo differenziale; fa uso 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  $m < 1$ .

## RST segmenti - Algoritmo di Bresenham



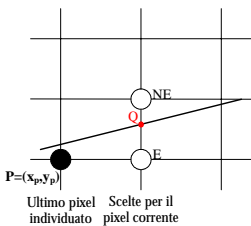
- ❖ Supponiamo che l'ultimo pixel individuato dal processo di rasterizzazione sia il pixel  $P$  di coordinate  $P=(x_p, y_p)$

## RST segmenti - Algoritmo di Bresenham



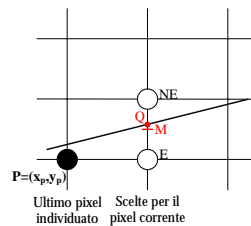
- ❖ Il prossimo pixel della rasterizzazione sarà il pixel immediatamente a destra di  $P$  ( $E$ , per *east pixel*) oppure quello in alto a destra ( $NE$ , per *north-east pixel*).
- ❖ La scelta del prossimo pixel è limitata a due sole possibilità

## RST segmenti - Algoritmo di Bresenham



- ❖ Indichiamo con  $Q$  il punto in cui il segmento interseca la retta  $x = x_p + 1$ . Il prossimo pixel è quello, tra  $E$  e  $NE$ , con distanza minima da  $Q$ .
- ❖ La scelta di un pixel è ricondotta alla misura di una distanza.

## RST segmenti - Algoritmo di Bresenham



- ❖ Detto  $M$  il punto di mezzo del segmento  $(E, NE)$ , si deve scegliere il punto che sta dalla stessa parte di  $Q$  rispetto ad  $M$ ;
- ❖ Dobbiamo quindi definire da che parte è  $Q$  rispetto ad  $M$ .
- ❖ La scelta di un pixel è ricondotta all'analisi della relazione geometrica tra due punti.

## RST segmenti - Algoritmo di Bresenham

- ❖ Il problema è quindi definire da che parte si trova Q (intersezione del segmento con la retta  $x = x_p + 1$ ) rispetto a M (punto medio tra i centri dei pixel E ed NE);
- ❖ Conviene utilizzare la forma implicita dell'equazione della retta:

$$F(x, y) = ax + by + c = 0$$

## RST segmenti - Algoritmo di Bresenham

- ❖ Poiché  $m = dy/dx$ ;  $dx = x_1 - x_0$ ;  $dy = y_1 - y_0$ , la forma esplicita può essere riscritta come:

$$y = mx + B$$

$$y = \frac{dy}{dx} x + B$$

$$y = \frac{y_1 - y_0}{x_1 - x_0} x + B$$

## RST segmenti - Algoritmo di Bresenham

- ❖ Quindi

$$y = \frac{dy}{dx} x + B$$

$$dx \cdot y = dy \cdot x + B \cdot dx$$

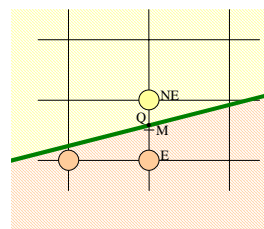
$$dy \cdot x - dx \cdot y + B \cdot dx = 0$$

$$F(x, y) = dy \cdot x - dx \cdot y + B \cdot dx = 0$$

- ❖ con

$$a = dy; \quad b = -dx; \quad c = B \cdot dx$$

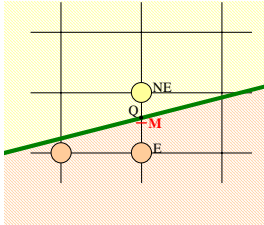
## RST segmenti - Algoritmo di Bresenham



- ❖ La funzione F:

- ❖ vale 0 per tutti i punti della retta;
- ❖ assume valori positivi sotto la retta;
- ❖ assume valori negativi sopra la retta.
- ❖  $F(Q) = 0$

## RST segmenti - Algoritmo di Bresenham

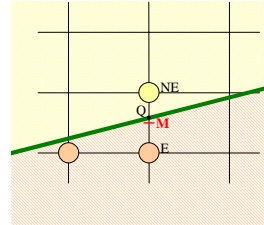


- ❖ La scelta tra E e NE si riduce alla valutazione del segno della funzione F nel punto M.

$$F(M) = F(x_p + 1, y_p + \frac{1}{2})$$

- ❖ L'analisi della relazione geometrica tra due punti si riduce quindi a valutare il segno di una funzione.

## RST segmenti - Algoritmo di Bresenham



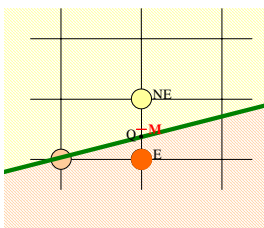
- ❖ La decisione si basa sul segno di  $F(M)$ ; indichiamo  $F(M)$  come *variabile di decisione*  $d$ .

$$d = F(M)$$

$$d = F(x_p + 1, y_p + \frac{1}{2})$$

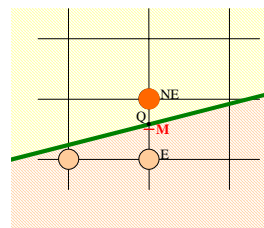
$$d = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$$

## RST segmenti - Algoritmo di Bresenham



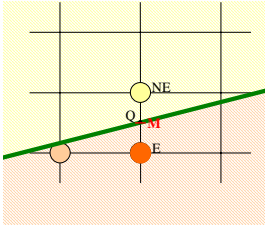
- ❖ Se  $d < 0$ 
  - ❖ M giace sopra la retta;
  - ❖ Scegliamo E come prossimo pixel della rasterizzazione.

## RST segmenti - Algoritmo di Bresenham



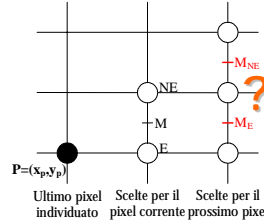
- ❖ Se  $d > 0$ 
  - ❖ M giace sotto la retta;
  - ❖ Scegliamo NE come prossimo pixel della rasterizzazione.

## RST segmenti - Algoritmo di Bresenham



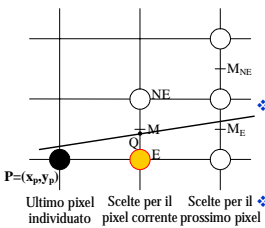
- ❖ Se  $d = 0$ 
  - ❖ M appartiene alla retta ( $Q \equiv M$ );
  - ❖ Scegliamo come prossimo pixel della rasterizzazione uno qualsiasi tra E ed NE
- ❖ Supponiamo che E sia scelto.

## RST segmenti - Algoritmo di Bresenham



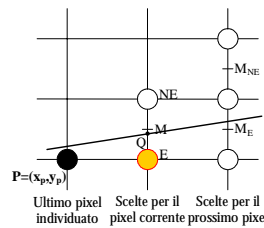
- ❖ L'algoritmo di Bresenham costruisce anche  $d$  in modo incrementale.
- ❖ A tal fine è necessario individuare il punto M al prossimo passo ( $x = x_p + 2$ ) sulla base della scelta fatta al passo corrente.

## RST segmenti - Algoritmo di Bresenham



- ❖ Se l'ultimo pixel selezionato è stato E
  - $d_{\text{new}} = F(x_p + 2, y_p + \frac{1}{2})$
  - $d_{\text{new}} = a(x_p + 2) + b(y_p + \frac{1}{2}) + c$
- ❖ poiché
  - $d = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$
  - ❖ sottraendo si ha
  - $d_{\text{new}} = d_{\text{old}} + a$

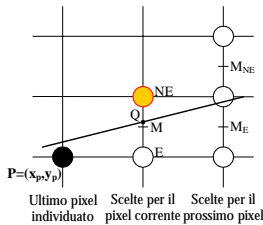
## RST segmenti - Algoritmo di Bresenham



- ❖ L'incremento da aggiungere a  $d$  dopo aver scelto E è quindi:
  - $\Delta_E = a = dy$
- ❖ Questo risultato vale ad ogni passo della rasterizzazione.



## RST segmenti - Algoritmo di Bresenham



- Se invece l'ultimo pixel selezionato è stato NE:

$$d_{\text{new}} = F(x_p + 2, y_p + \frac{3}{2})$$

$$d_{\text{new}} = a(x_p + 2) + b(y_p + \frac{3}{2}) + c$$

- quindi

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

- da cui

$$\Delta_{\text{NE}} = a + b = dy - dx$$

## RST segmenti - Algoritmo di Bresenham

- Ad ogni passo l'algoritmo sceglie il prossimo pixel tra due possibili candidati basandosi sul valore corrente di una variabile  $d$  di decisione;
- ricalcola il valore della variabile di decisione incrementalmente aggiungendo al suo valore corrente una quantità fissa predefinita ( $\Delta_E$  o  $\Delta_{\text{NE}}$ );
- Il valore iniziale per  $d$  risulta:

$$F(x_0 + 1, y_0 + \frac{1}{2}) = a(x_0 + 1) + b(y_0 + \frac{1}{2}) + c$$

$$F(x_0 + 1, y_0 + \frac{1}{2}) = ax_0 + by_0 + c + a + \frac{b}{2}$$

$$F(x_0 + 1, y_0 + \frac{1}{2}) = F(x_0, y_0) + a + \frac{b}{2}$$

## RST segmenti - Algoritmo di Bresenham

- Poiché  $(x_0, y_0)$  appartiene al segmento  $F(x_0, y_0) = 0$  quindi:

$$F(x_0 + 1, y_0 + \frac{1}{2}) = F(x_0, y_0) + a + \frac{b}{2}$$

$$d_{\text{start}} = a + \frac{b}{2} = dy - \frac{dx}{2}$$

- La frazione può essere eliminata utilizzando come funzione decisionale la funzione  $2F$ .

## RST segmenti - Algoritmo di Bresenham

- L'algoritmo di Bresenham si dice **algoritmo differenziale del primo ordine**;
- Il primo ordine si riferisce ai passi "in avanti" (in effetti sono derivazioni della funzione) utilizzati per calcolare la differenza tra i valori della funzione;
- Il suo parente matematico più stretto è un'equazione differenziale del primo ordine.

## RST segmenti - Algoritmo di Bresenham

```

Mi dpoi nTLI ne(int x0,
                int y0,
                int x1,
                int y1,
                int val ue)
{
    int dx, dy, incE, incNE, d, x, y;
    while (x < x1) {
        if (d <= 0) {
            d = d + incE;
            x++;
        } else {
            d = d + incNE;
            x++;
            y++;
        }
        writePixel(x, y, val ue);
    }
}
    
```

**Variabili intere**

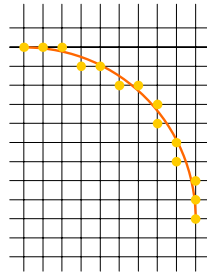
**Inizializzazione**

**Scelta di E**

**Scelta di NE per usare solo interi**

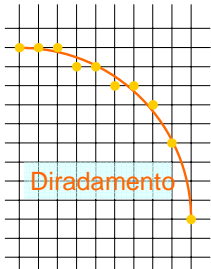
**Moltiplico per 2**

## Rasterizzazione di circonferenze



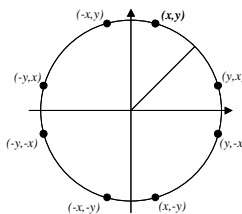
- ❖ Così come per i segmenti l'algoritmo di rasterizzazione di una circonferenza o di un arco deve calcolare le coordinate dei pixel che giacciono il più possibile vicino ad essa

## Rasterizzazione di circonferenze



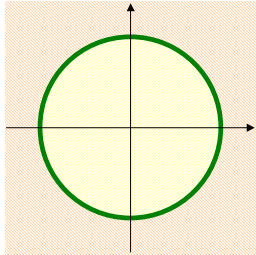
- ❖ L'equazione, implicita, di una circonferenza (centrata nell'origine, di raggio R) è
 
$$x^2 + y^2 = R^2$$
- ❖ In forma esplicita
 
$$y = \pm\sqrt{R^2 - x^2}$$
- ❖ Problemi legati al costo delle operazioni di valutazione della funzione e al diradamento dei pixel.

## Rasterizzazione di circonferenze



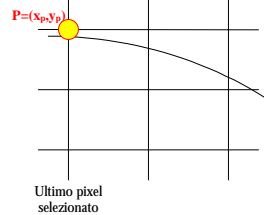
- ❖ La rasterizzazione di una circonferenza può essere effettuata per ottanti anziché per quadranti;
- ❖ Se un pixel  $(x, y)$  rappresenta la circonferenza, anche gli altri 7 pixel da esso derivati per simmetria radiale cambiando di segno e scambiando tra loro  $x$  e  $y$  la rappresentano.

## RST circonferenze - Algoritmo di Bresenham



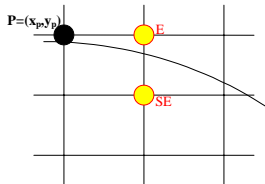
- ❖ Molto simile all'algoritmo di Bresenham per segmenti;
- ❖ Si considera la funzione implicita:
 
$$F(x, y) = x^2 + y^2 - R^2$$
- ❖ che vale:
  - ❖  $F=0$  sulla circonferenza;
  - ❖  $F<0$  al suo interno;
  - ❖  $F>0$  al suo esterno.

## RST circonferenze - Algoritmo di Bresenham



- ❖ Supponiamo di operare nel secondo ottante, (da  $x = 0$  a  $x = R/\sqrt{2}$ );
- ❖ L'ultimo pixel individuato dall'algoritmo sia il pixel P di coordinate  $(x_p, y_p)$ .

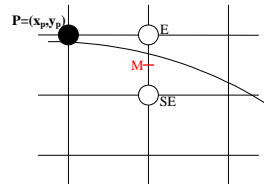
## RST circonferenze - Algoritmo di Bresenham



Ultimo pixel selezionato    Scelte per il pixel corrente

- ❖ Il prossimo pixel della rasterizzazione sarà quello immediatamente a destra di P (E, per *east pixel*) o quello in basso a destra (SE, per *south-east pixel*)

## RST circonferenze - Algoritmo di Bresenham



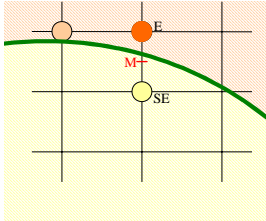
Ultimo pixel selezionato    Scelte per il pixel corrente

- ❖ Come nel caso della retta definiamo la funzione  $F(M)=d$  che servirà da variabile decisionale:
 
$$d = F(x_p + 1, y_p - \frac{1}{2})$$

$$d = (x_p + 1)^2 + (y_p - \frac{1}{2})^2 - R^2$$

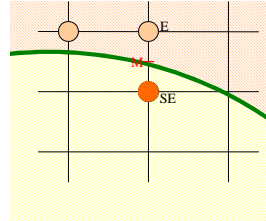
$$d = x_p^2 + y_p^2 + 2x_p - y_p - R^2 + \frac{5}{4}$$
- ❖  $d$  è un polinomio del secondo ordine.

### RST circonferenze - Algoritmo di Bresenham



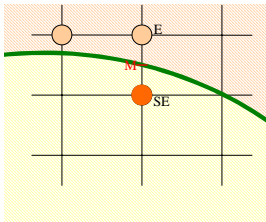
- ❖ Se  $d < 0$ 
  - ❖ M è interno alla circonferenza;
  - ❖ E sarà il prossimo pixel della rasterizzazione.

### RST circonferenze - Algoritmo di Bresenham



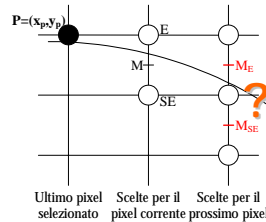
- ❖ Se  $d > 0$ 
  - ❖ M è esterno alla circonferenza;
  - ❖ SE sarà il prossimo pixel della rasterizzazione.

### RST circonferenze - Algoritmo di Bresenham



- ❖ Se  $d = 0$ 
  - ❖ M è sulla circonferenza;
  - ❖ E oppure SE possono diventare il prossimo pixel della rasterizzazione;
  - ❖ Se adotta SE per convenzione.

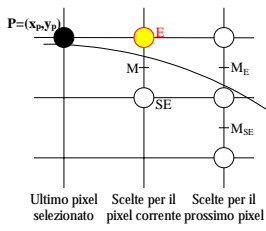
### RST circonferenze - Algoritmo di Bresenham



- ❖ Anche in questo caso la posizione di M al prossimo passo (e quindi il prossimo valore di  $d$ ) dipenderà dalla scelta operata in questo passo.

Ultimo pixel selezionato    Scelte per il pixel corrente    Scelte per il prossimo pixel

### RST circonferenze - Algoritmo di Bresenham

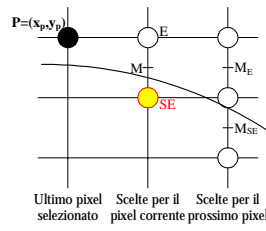


- Se l'ultimo pixel selezionato è stato E
 
$$d_{\text{new}} = F(x_p + 2, y_p - \frac{1}{2})$$

$$d_{\text{new}} = (x_p + 2)^2 + (y_p - \frac{1}{2})^2 - R^2$$
- poiché
 
$$d_{\text{new}} = x_p^2 + y_p^2 + 4x_p - y_p - R^2 + \frac{17}{4}$$
- sottraendo si ha
 
$$d = x_p^2 + y_p^2 + 2x_p - y_p - R^2 + \frac{5}{4}$$

$$\Delta_E = 2x_p + 3$$
- funzione di x<sub>p</sub>.

### RST circonferenze - Algoritmo di Bresenham



- Se l'ultimo pixel selezionato è stato SE
 
$$d_{\text{new}} = F(x_p + 2, y_p - \frac{3}{2})$$

$$d_{\text{new}} = (x_p + 2)^2 + (y_p - \frac{3}{2})^2 - R^2$$
- poiché
 
$$d_{\text{new}} = x_p^2 + y_p^2 + 4x_p - 3y_p - R^2 + \frac{25}{4}$$
- sottraendo si ha
 
$$d = x_p^2 + y_p^2 + 2x_p - y_p - R^2 + \frac{5}{4}$$

$$\Delta_{SE} = 2x_p - 2y_p + 5$$
- funzione di x<sub>p</sub> e y<sub>p</sub>

### RST circonferenze - Algoritmo di Bresenham



- Il valore iniziale di d è:
 
$$F(1, R - \frac{1}{2}) = 1^2 + (R - \frac{1}{2})^2 - R^2$$

$$F(1, R - \frac{1}{2}) = 1 + (R^2 - R + \frac{1}{4}) - R^2$$

$$F(1, R - \frac{1}{2}) = \frac{5}{4} - R$$

### RST circonferenze - Algoritmo di Bresenham

```

Mi dpoi nti Cl rcl e (Int radi us,
                    Int val ue)
{
    Int x = y;
    float d;
    x = 0;
    y = radi us;
    d = 5/4 - radi us;
    Cl rcl e Poi nts(x, y, val ue);
    whi l e ( y > x ) {
        i f ( d < 0 ) {
            d = d + 2 * x + 3;
            x++;
        }
        el se {
            d = d + 2 * (x - y) + 5;
            x++;
            y--;
        }
        Cl rcl e Poi nts(x, y, val ue);
    }
}
    
```

Variabili reali  
 Inizializzazione  
 Moltiplicazioni  
 Replaga pixel  
 Scelta di E

### RST circonferenze - Algoritmo di Bresenham

- ❖ Sono ancora presenti moltiplicazioni, dovute al calcolo di  $d$ ;
- ❖  $\Delta_E$  e  $\Delta_{SE}$  non sono costanti ma funzioni polinomiali di primo grado che dipendono da  $P$  (*punto di valutazione*);
- ❖  $\Delta_E$  e  $\Delta_{SE}$  possono essere calcolate in modo incrementale.

### RST circonferenze - Algoritmo di Bresenham

- ❖ Se E è il prossimo pixel, il punto di valutazione si muove da  $P=(x_p, y_p)$  a  $P_{new}=(x_p+1, y_p)$
- ❖ In  $P$  la funzione  $\Delta_E$  vale:
  - $\Delta_{E_{old}} = 2x_p + 3$
  - ❖ in  $P_{new}$  sarà
    - $\Delta_{E_{new}} = 2(x_p + 1) + 3$
    - $\Delta_{E_{new}} = 2x_p + 5$
    - ❖ e
      - $\Delta_{E_{new}} - \Delta_{E_{old}} = 2$
- ❖ In  $P$  la funzione  $\Delta_{SE}$  vale:
  - $\Delta_{SE_{old}} = 2x_p - 2y_p + 5$
  - ❖ in  $P_{new}$  sarà
    - $\Delta_{SE_{new}} = 2(x_p + 1) - 2y_p + 5$
    - $\Delta_{SE_{new}} = 2x_p - 2y_p + 7$
    - ❖ e
      - $\Delta_{SE_{new}} - \Delta_{SE_{old}} = 2$

### RST circonferenze - Algoritmo di Bresenham

- ❖ Se SE è il prossimo pixel, il punto di valutazione si muove da  $P=(x_p, y_p)$  a  $P_{new}=(x_p+1, y_p-1)$
- ❖ In  $P$  la funzione  $\Delta_E$  vale:
  - $\Delta_{E_{old}} = 2x_p + 3$
  - ❖ in  $P_{new}$  sarà
    - $\Delta_{E_{new}} = 2(x_p + 1) + 3$
    - $\Delta_{E_{new}} = 2x_p + 5$
    - ❖ e
      - $\Delta_{E_{new}} - \Delta_{E_{old}} = 2$
- ❖ In  $P$  la funzione  $\Delta_{SE}$  vale:
  - $\Delta_{SE_{old}} = 2x_p - 2y_p + 5$
  - ❖ in  $P_{new}$  sarà
    - $\Delta_{SE_{new}} = 2(x_p + 1) - 2(y_p - 1) + 5$
    - $\Delta_{SE_{new}} = 2x_p - 2y_p + 9$
    - ❖ e
      - $\Delta_{SE_{new}} - \Delta_{SE_{old}} = 4$

### RST circonferenze - Algoritmo di Bresenham

- ❖ Ricapitolando:
  - ❖ Scelta del prossimo pixel in base al valore di  $d$ ;
  - ❖ Aggiornamento di  $d$  in base al valore calcolato al passo precedente di  $\Delta_E$  o  $\Delta_{SE}$
  - ❖ Aggiornamento di  $\Delta_E$  e  $\Delta_{SE}$  usando gli incrementi fissi, tenendo di conto di quello che è il nuovo punto di valutazione
  - ❖ Spostamento.

## RST circonferenze - Algoritmo di Bresenham

```
MidpointCircle(int radius,
               int value)
{
    int x, y, d, del taE, del taSE;
    CirclePoints(x, y, value);
    while (y > x) {
        if (d < 0) {
            d = d + del taE;
            del taE = del taE + 2;
            del taSE = del taSE + 2;
            x++;
        } else {
            d = d + del taSE;
            del taE = del taE + 2;
            del taSE = del taSE + 4;
            x++;
            y--;
        }
        CirclePoints(x, y, value);
    }
}
```

**Variabili intere**

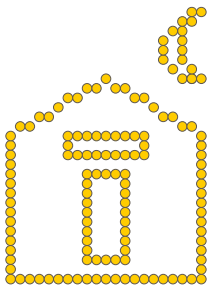
**Inizializzazione**

```
x = 0;
y = radius;
d = 1 - radius;
del taE = 3;
del taSE = -2 * radius + 5;
```

**Scelta di SE**

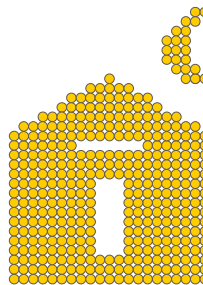
**Scelta di E**

## Rasterizzazione (filling) di poligoni



- ❖ La tecnologia raster permette di "disegnare" primitive geometriche rappresentate dal solo contorno e primitive "piene".

## Rasterizzazione (filling) di poligoni



- ❖ La tecnologia raster permette di "disegnare" primitive geometriche rappresentate dal solo contorno e primitive "piene".

## Rasterizzazione (filling) di poligoni

- ❖ Per la formazione di un'immagine "piena" la tecnologia raster si basa su una *carezza* del sistema visivo umano: l'occhio (la retina) esegue un'integrazione spaziale tra oggetti che non è in grado di distinguere per mancanza di risoluzione;
- ❖ Poiché il numero di recettori visivi è limitato, l'occhio percepisce elementi visuali distinti ma sufficientemente vicini tra loro (come due pixel vicini in un monitor) come un singolo oggetto;
- ❖ Un insieme di pixel contigui dello stesso colore, per l'occhio umano, risultano una singola figura con colore uniforme.

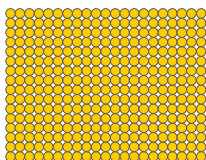
## Rasterizzazione (filling) di rettangoli

- ❖ Disegnare un rettangolo *vuoto* consiste nell'applicare 4 volte l'algoritmo di rasterizzazione dei segmenti che ne rappresentano i lati;



## Rasterizzazione (filling) di rettangoli

```
for y from  $y_{min}$  to  $y_{max}$  of the rectangle  
  for x from  $x_{min}$  to  $x_{max}$   
    WritePixel(x, y)
```



- ❖ Per disegnare un rettangolo *pieno*, con lati paralleli agli assi cartesiani, è sufficiente innescare un doppio ciclo di "accensione" dei pixel interni al rettangolo

## Rasterizzazione (filling) di poligoni



Convesso



Concavo



non  
semplice



Contorni  
multipli

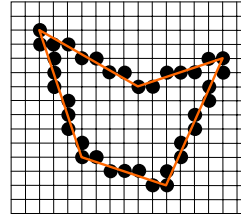
- ❖ L'algoritmo di rasterizzazione deve operare correttamente sulle diverse tipologie di primitive geometriche:
  - ❖ Polig. Convesso
  - ❖ Polig. Concavo
  - ❖ Polig. Intrecciato
  - ❖ Polig. A Contorni multipli



## Rasterizzazione (filling) di poligoni

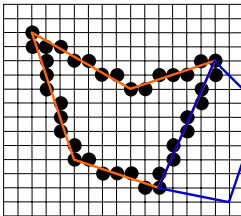
- ❖ L'algoritmo analizzato individua in modo sequenziale, dal basso verso l'alto, le *span* (righe orizzontali) di pixel che rappresentano il poligono;
- ❖ Per ogni riga dello schermo, i punti estremi delle span sono calcolati in modo *incrementale* (simile al modo visto per i segmenti);
- ❖ Approccio *brute force*:
  - ❖ Rasterizzare i contorni del poligono;
  - ❖ Rasterizzare l'interno basandosi sulla rasterizzazione dei contorni.

## Filling di poligoni (a partire dal contorno)



- ❖ La maniera più immediata di calcolare le intersezioni del poligono con le scan-line è quella di utilizzare l'algoritmo di scan-conversion per segmenti su ogni spigolo del poligono;
- ❖ Purtroppo, l'algoritmo di scan-conversion di segmenti non ha alcuna nozione del concetto di interno o di esterno (al poligono) e quindi può selezionare pixel che sono **esterni** al poligono.

## Filling di poligoni (a partire dal contorno)



- ❖ Certamente un problema quando si rasterizzano poligoni di colore diverso con spigoli condivisi;
- ❖ Possibilità di pixel di un poligono che *invadono* l'altro poligono generando un effetto visivo scorretto;
- ❖ La rasterizzazione complessiva dipende dall'ordine in cui i singoli poligoni sono rasterizzati.

## Filling di poligoni - Generalità

- ❖ L'algoritmo analizzato opera incrementalmente sulle scan-line;
- ❖ Una volta effettuato il filling del poligono su una scan-line (cioè identificati i pixel della scan-line che appartengono al poligono, l'algoritmo sfrutta le informazioni trovate per aggiornare incrementalmente le intersezioni e fare il filling sulla scan-line successiva);
- ❖ Per ogni scan-line:
  - ❖ Individuare le intersezioni della scan-line con tutti gli spigoli del poligono;
  - ❖ Ordinare le intersezioni sulla coordinata x;
  - ❖ Selezionare tutti i pixel, tra coppie di intersezioni, che sono interni al poligono.
- ❖ Per la determinazione dei pixel interni si usa la **regola odd-parity**.

## Filling di poligoni - Regola odd-parity

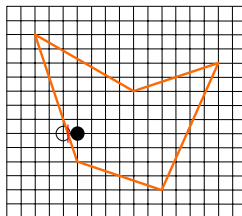
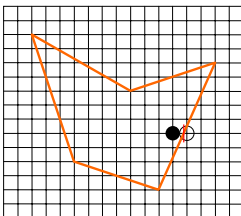
- ❖ Si utilizza un bit (flag) di parità che può assumere valore **pari** o **dispari**;
- ❖ La parità è inizialmente pari, ogni intersezione cambia il bit di parità, i pixel fanno parte del poligono quando la parità è dispari, non ne fanno parte quando la parità è pari.
  
- ❖ La regola odd-parity è utilizzata per verificare l'appartenenza di un pixel ad un poligono;
- ❖ 4 casi distinti sono da considerare

## Filling di poligoni - Appartenenza caso 1

- ❖ **Data un'intersezione con un valore generico  $x$  razionale, determinare quale dei due pixel ai lati dell'intersezione è quello cercato?**
- ❖ Se si incontra l'intersezione provenendo da *dentro* il poligono (*parity bit* dispari) si arrotonda all'intero inferiore per rimanere *dentro*.
- ❖ Se si incontra l'intersezione provenendo da *fuori* il poligono (*parity bit* pari) si arrotonda all'intero superiore per entrare dentro.

## Filling di poligoni - Appartenenza caso 1

- ❖ ... provenendo da dentro ...      ... provenendo da fuori ...



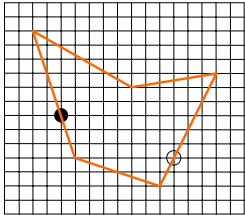
## Filling di poligoni - Appartenenza caso 1

- ❖ In questa maniera i pixel (che hanno sempre coordinata intera) a **sinistra** dell'intersezione **sinistra** ed a **destra** dell'intersezione **destra** non vengono mai attribuiti al poligono;
- ❖ I pixel a **destra** dell'intersezione **sinistra** ed a **sinistra** dell'intersezione **destra** vengono attribuiti al poligono.

## Filling di poligoni - Appartenenza caso 2

❖ Come si tratta il caso speciale dell'intersezione a coordinate intere?

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

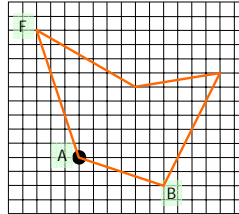


## Filling di poligoni - Appartenenza caso 3

❖ Come si tratta il caso speciale in cui l'intersezione riguarda un vertice (che ha sempre coordinate intere)?

❖ Nel calcolo del *parity bit*, si considera solo il vertice  $y_{min}$  e non il vertice  $y_{max}$ .

❖ Nell'esempio il vertice A è considerato solo come vertice  $y_{min}$  dello spigolo FA e non come vertice  $y_{max}$  dello spigolo AB.

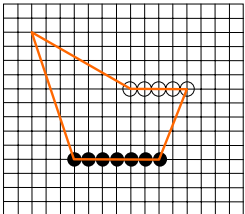


## Filling di poligoni - Appartenenza caso 4

❖ Come si tratta il caso speciale di vertici che definiscono uno spigolo orizzontale?

❖ I vertici di una linea orizzontale non influiscono sul calcolo del *parity bit*;

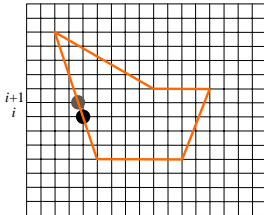
❖ In modo automatico (in virtù del trattamento del caso 3) i lati orizzontali "bassi" del poligono sono disegnati mentre quelli "alti" sono omessi.



## Filling di poligoni - Le intersezioni

- ❖ Si vuole evitare il calcolo delle intersezioni tra tutte le scan-line e tutti gli spigoli del poligono;
- ❖ Gli spigoli che intersecano la scan-line  $i$  intersecano anche la scan-line  $i+1$ , a meno che lo spigolo non abbia il suo vertice nella scan-line  $i$ ;
- ❖ Può essere utilizzato un approccio incrementale molto simile a quello dell'algoritmo di scan-conversion per le linee;
- ❖ La differenza tra scan conversion di segmenti e filling di poligoni è che nel primo caso si deve selezionare il pixel più vicino alla linea ideale, nel secondo caso si deve tenere conto del *dentro* e del *fuori* ed applicare le regole di arrotondamento che permettono di rimanere dentro al poligono;
- ❖ Occorre distinguere tra spigoli a sinistra (entrata) e spigoli a destra (uscita).

## Filling di poligoni - Le intersezioni



- Note le intersezioni per una scan-line, le intersezioni per la scan-line successiva possono essere derivate come:

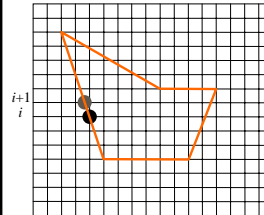
$$x_{i+1} = x_i + \frac{1}{m}$$

- dove

$$m = \frac{(y_{\max} - y_{\min})}{(x_{\max} - x_{\min})}$$

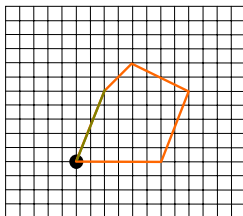
è il coefficiente angolare della linea-spigolo.

## Filling di poligoni - Le intersezioni



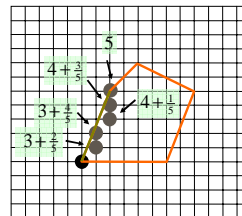
- Anziché utilizzare aritmetica reale per calcolare gli incrementi  $1/m$  si considera l'incremento come numero razionale espresso come numeratore e denominatore.

## Filling di poligoni - Le intersezioni



3

## Filling di poligoni - Le intersezioni



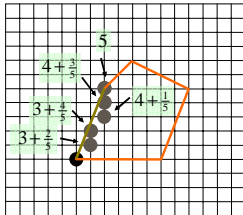
3 4 5

- Linea-spigolo *sinistro*
- Coefficiente angolare  $m > 1$
- $x_{\min} = 3; m = 5/2$

- La sequenza di ascisse:

- 3
- $3 + 2/5$
- $3 + 4/5$
- $3 + 6/5 = 4 + 1/5$
- $4 + 3/5$
- $4 + 5/5 = 5$

## Filling di poligoni - Le intersezioni



- Ad ogni iterazione, quando la parte frazionaria eccede 1 si incrementa  $x$  (la parte intera) di 1 e si sottrae 1 dalla parte frazionaria muovendosi quindi di 1 pixel verso destra.

## Filling di poligoni - Gestione spigolo sinistro

```

LeftEdgeScan (int xmin,
              int ymin,
              int xmax,
              int ymax,
              int value)
{
    int x, y, numerator, denominator, increment; Variabili intere
    x = xmin;
    numerator = xmax-xmin; Inizializzazione
    denominator = ymax-ymin;
    increment = denominator;
    for (y=ymin; y<ymax; y++) {
        WritePixel(x, y, value);
        increment = increment + numerator;
        if (increment > denominator) {
            x++;
            increment = increment - denominator;
        }
    }
}
    
```

## Filling di poligoni - Le strutture dati

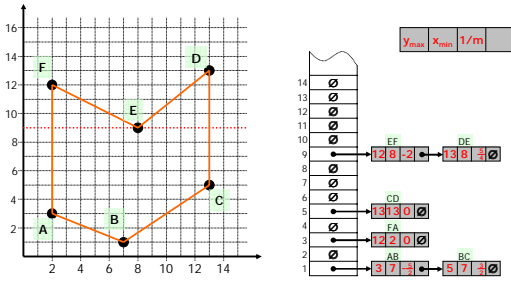
- Per migliorare l'efficienza complessiva ad ogni passo si considera una scan-line e tutti i lati ed eventualmente poligoni attraversati;
- Il calcolo incrementale dei valori delle linee spigolo non avviene quindi in un'unica soluzione ma deve essere interrotto salvando i valori calcolati al passo precedente;
- La struttura dati deve consentire di:
  - Trovare le intersezioni della scan-line con tutti gli spigoli del(i) poligono(i);
  - Ordinare le intersezioni sulla coordinata  $x$ ;
  - Selezionare tutti i pixel, tra coppie di intersezioni, che sono interni al(i) poligono(i).

## Filling di poligoni - Le strutture dati

- La struttura dati utilizzata è la lista **Active Edge Table** (tabella degli spigoli attivi) che, ad ogni passo dell'algoritmo, contiene le informazioni "attive" della lista **Edge Table** (tabella degli spigoli);
- La Edge Table viene costruita in una fase iniziale e contiene tutte le informazioni necessarie alla rasterizzazione degli spigoli;
- La Edge Table è un **bucket** (array di liste), con tante celle per quante sono le scan-line dello schermo;
- La Edge Table non contiene spigoli orizzontali.

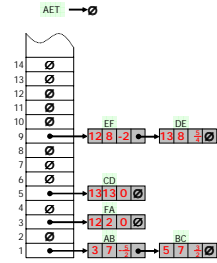


## Filling di poligoni - Edge Table



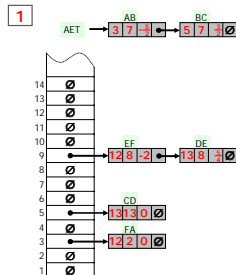
## Filling di poligoni - Active Edge Table

- La Active Edge Table viene costruita e modificata copiando elementi della lista dalla Edge Table;
- Inizialmente è vuota.



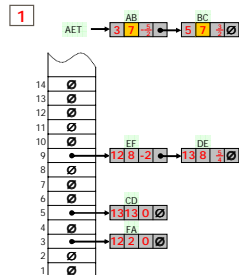
## Filling di poligoni - Active Edge Table

- Non appena, scendendo la ET, si trova una cella non vuota la AET viene inizializzata e la procedura di rasterizzazione ha effettivo inizio.

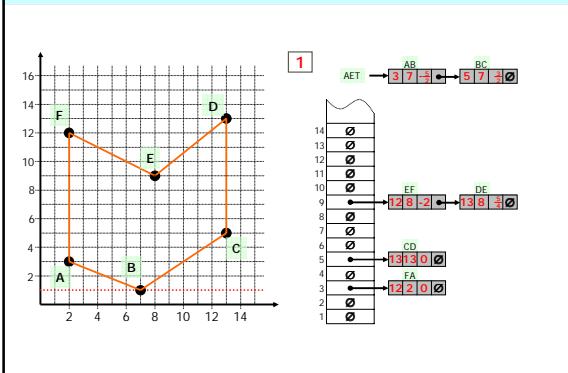


## Filling di poligoni - Active Edge Table

- Nella AET il secondo campo non rappresenta  $x_{min}$  bensì il valore della  $x$  corrente da usare nella rasterizzazione.



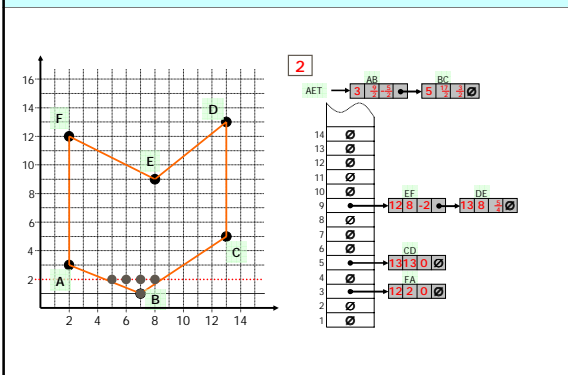
## Filling di poligoni - Active Edge Table



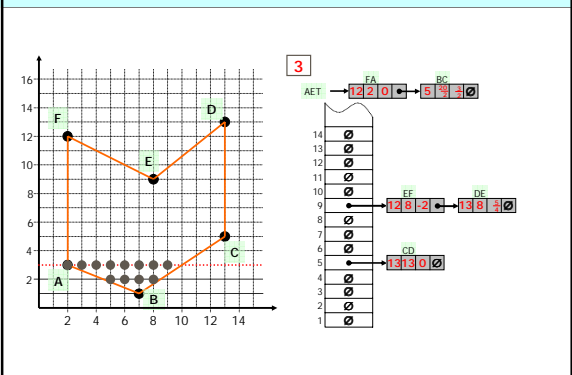
## Filling di poligoni - L'algoritmo

- ❖ Impostare  $y$  alla minima ordinata non vuota in ET;
- ❖ Inizializzare AET (vuota);
- ❖ Ripetere, fino allo svuotamento di AET e ET:
  - ❖ Muovere dal bucket di ET al corrispondente di AET gli edge per cui  $y_{\min}=y$ , quindi ordinare su AET per  $x$ ;
  - ❖ Disegnare i pixel della scan-line per coppie di coordinate  $x$  dalla AET;
  - ❖ Rimuovere dalla AET gli edge per cui  $y_{\max}=y$  (quelli che non intersecano la prossima scan-line);
  - ❖ Incrementare  $y$  di 1;
  - ❖ Per ogni edge non verticale nella AET, aggiornare  $x$  per il nuovo  $y$ .

## Filling di poligoni - L'algoritmo

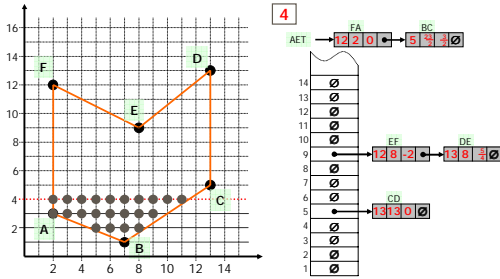


## Filling di poligoni - L'algoritmo

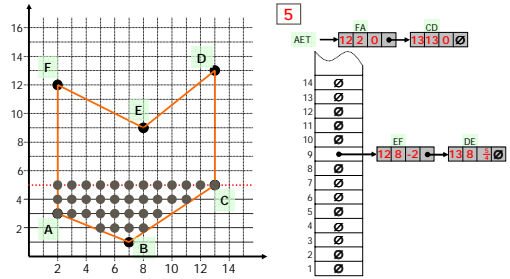




## Filling di poligoni - L'algoritmo

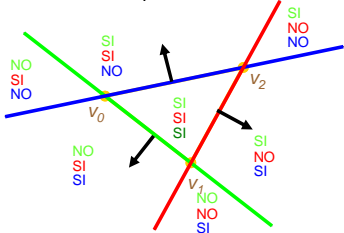


## Filling di poligoni - L'algoritmo

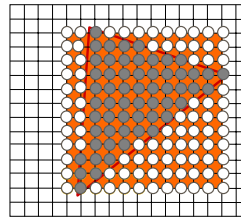


## Triangle filling: com'è oggi...

- ❖ Triangolo=intersezione di 3 semipiani
- ❖ un punto è interno al triangolo sse è dalla stessa parte di tutti e tre i piani



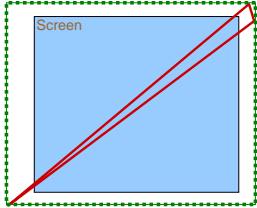
## Triangle filling: com'è oggi...



1. Trovo il bounding box del triangolo
2. Per ogni frammento, verifico se è interno al triangolo

- È una soluzione coerente con il modello di calcolo "parallelo" della GPU
- Infatti il passo 2 non è fatto sequenzialmente ma in parallelo su blocchi (es: 2,4) di frammenti

## Un caso sfortunato



**Sinonimo di MALE**  
in computer graphics:  
(anche per questo, ma non solo)

**lunghi e stretti = male**  
molti algoritmi portano ad artefatti  
(come vedremo)

**circa equilateri = bene**  
robusti con praticamente tutti gli algoritmi

Triangoli:

- lunghi e stretti
  - messi lungo la direzione diagonale
- ☹