

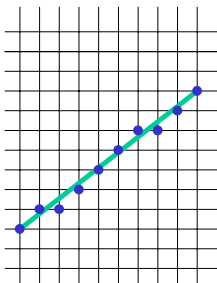
Costruzione di Interfacce Lezione 8 Rasterizzazione

cignoni@isti.cnr.it
<http://vcg.isti.cnr.it/~cignoni>

Oggi parleremo di...

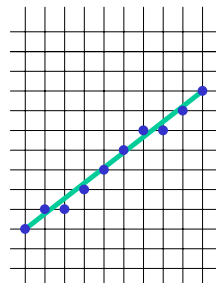
- ❖ Algoritmi raster 2D
 - ❖ La scan-conversion di linee
 - ❖ Algoritmo di Bresenham per segmenti
 - ❖ Rasterizzazione di poligoni

Rasterizzazione di segmenti



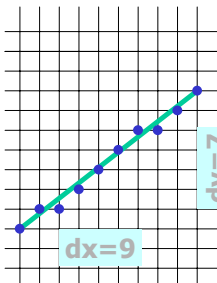
- ❖ L'algoritmo di rasterizzazione di un segmento di retta deve calcolare le coordinate dei pixel che giacciono sulla linea ideale o che sono il più vicino possibile ad essa

Rasterizzazione di segmenti



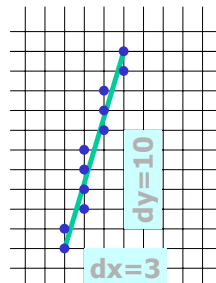
- ❖ Vogliamo avere la sequenza di pixel che
 - ❖ approssima al meglio il segmento
- ❖ e quindi
 - ❖ sia il più in linea retta possibile

Rasterizzazione di segmenti



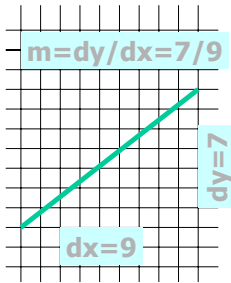
- ❖ Consideriamo un'approssimazione della retta larga un pixel
- ❖ Per coefficienti angolari $|m| \leq 1$ la rasterizzazione conterrà un pixel per ogni colonna

Rasterizzazione di segmenti



- ❖ Consideriamo un'approssimazione della retta larga un pixel
- ❖ Per coefficienti $|m| > 1$ conterrà un pixel per ogni riga
- ❖ Noi considereremo solo il caso di $m \leq 1$: gli algoritmi sviluppati in questo caso possono essere facilmente estesi agli altri

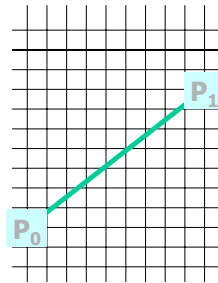
Algoritmo analitico



- ❖ La funzione analitica che rappresenta la retta è

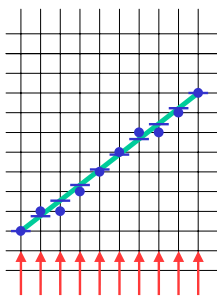
$$y = mx + B$$

Algoritmo analitico



- ❖ Vogliamo rasterizzare il segmento che va dal punto P_0 di coordinate (x_0, y_0) al punto P_1 di coordinate (x_1, y_1)
- ❖ Entrambi i punti hanno coordinate intere

Algoritmo analitico



1. Partendo dal pixel con coordinata x minima x_0 :
 - ❖ 2.1 Incrementare x a passo 1
 - ❖ 2.2 $\forall x_i$ calcolare y_i come $mx_i + B$
 - ❖ 2.3 Arrotondare quindi y_i

Algoritmo analitico

- ❖ In questa maniera si seleziona sempre il pixel che è più vicino alla linea ideale, quello cioè che ha distanza minima dalla linea
- ❖ Per identificare ogni pixel si devono fare 3 operazioni: un'addizione, una moltiplicazione ed un arrotondamento

Algoritmo DDA

- ❖ Si può eliminare la moltiplicazione usando una tecnica incrementale, che consiste nel calcolare un punto della retta sulla base del punto precedente
- ❖ L'algoritmo che si ottiene prende il nome di **algoritmo DDA** (*digital differential analyzer*)

Algoritmo DDA

- ❖ Notando che
$$y_{i+1} = mx_{i+1} + B$$
$$y_{i+1} = m(x_i + \Delta x) + B$$
$$y_{i+1} = y_i + m\Delta x$$
$$y_{i+1} = y_i + m$$

Algoritmo DDA

- ❖ Questo vale per tutti i punti della linea:

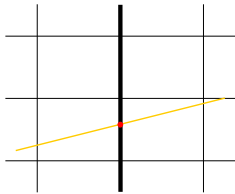
$$x_{i+1} = x_i + 1 \Rightarrow y_{i+1} = y_i + m$$
- ❖ Ad ogni passo si deve fare una operazione di arrotondamento e le variabili utilizzate (e quindi l'aritmetica) sono reali
- ❖ Usare aritmetica reale vuol dire introdurre errori di arrotondamento

Algoritmo DDA

```

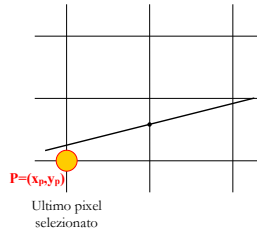
Line(int x0,
    int y0,
    int x1,
    int y1,
    int value)
{
    int x;
    float dy, dx, y, m; Variabili reali
    dy = y1-y0;
    dx = x1-x0;
    m = dy/dx;
    y = y0; Inizializzazione
    for (x=x0; x<=x1; x++) { Arrotondamento
        WritePixel(x, floor(0.5+y), value);
        y = y+m;
  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

Algoritmo di Bresenham



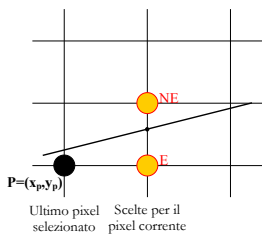
- ❖ Alla base dell'algoritmo di Bresenham (detto anche algoritmo del punto di mezzo) c'è il tentativo di usare solo operazioni in aritmetica intera

Algoritmo di Bresenham



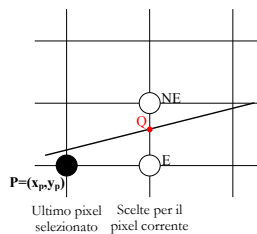
- ❖ Alla base dell'algoritmo di Bresenham c'è l'idea di usare solo aritmetica intera
- ❖ L'ultimo pixel facente parte della nostra rasterizzazione è il pixel P di coordinate (x_p, y_p)

Algoritmo di Bresenham



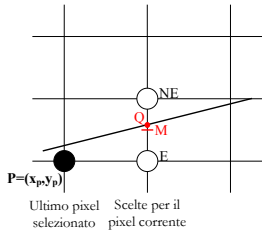
- ❖ Il prossimo pixel della rasterizzazione sarà o quello immediatamente a destra di P (E, per east pixel) o quello in alto a destra (NE, per north-east pixel)

Algoritmo di Bresenham



- ❖ Chiamiamo Q il punto in cui la linea da convertire interseca la colonna $X = X_p + 1$: sceglieremo come prossimo pixel quello, tra E e NE, che minimizza la distanza da Q

Algoritmo di Bresenham



- ❖ Detto M il punto di mezzo del segmento E-NE, dobbiamo scegliere il punto che sta dalla stessa parte di Q rispetto ad M
- ❖ Dobbiamo quindi calcolare da che parte di M sta Q

Algoritmo di Bresenham

- ❖ Dobbiamo calcolare da che parte di M sta Q
- ❖ Come facciamo?
- ❖ Conviene utilizzare la forma implicita dell'equazione della retta:

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

Algoritmo di Bresenham

❖ Poiché

- ❖ $m = dy/dx$
- ❖ $dx = x_1 - x_0$
- ❖ $dy = y_1 - y_0$

la forma esplicita si può riscrivere

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

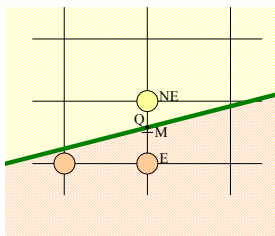
Algoritmo di Bresenham

❖ Quindi

$$\text{con } F(dx, y) = \frac{dy}{dx}x + B \cdot dx + B \cdot dx = 0$$

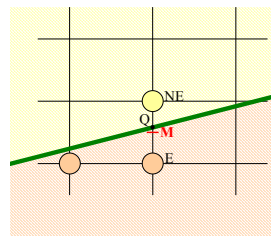
$$a = dy; \quad b = -dx; \quad c = B \cdot dx$$

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
- ❖ E' chiaro che $F(Q)=0$

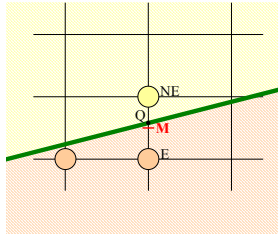
Algoritmo di Bresenham



- ❖ A questo punto una maniera semplice per decidere se scegliere E o NE, consiste nel calcolare

$$\text{e vederne il segno } F(M) = F(x_p + 1, y_p + \frac{1}{2})$$

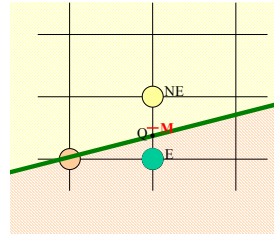
Algoritmo di Bresenham



- ❖ Poiché la nostra decisione si basa sul segno di $F(M)$, chiamiamo questa variabile *variabile di decisione* d
- ❖ Quindi

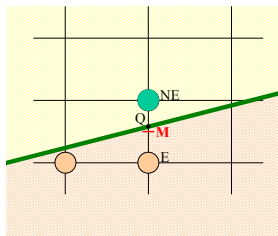
$$d = F(M) + 1 \cdot \Delta y \cdot b \left(\frac{1}{2} + \frac{1}{2} \right) + c$$

Algoritmo di Bresenham



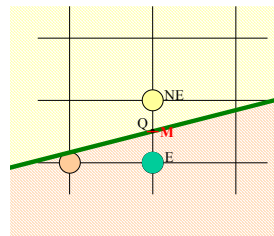
- ❖ $d < 0$
- ❖ M sta sopra la retta
- ❖ Scegliamo E come prossimo pixel della rasterizzazione

Algoritmo di Bresenham



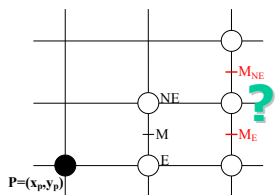
- ❖ $d > 0$
- ❖ M sta sotto la retta
- ❖ Scegliamo NE come prossimo pixel della rasterizzazione

Algoritmo di Bresenham



- ❖ $d = 0$
- ❖ M sta sulla retta ($Q \equiv M$)
- ❖ Scegliamo come prossimo pixel della rasterizzazione uno qualsiasi dei due
- ❖ Diciamo che scegliamo E

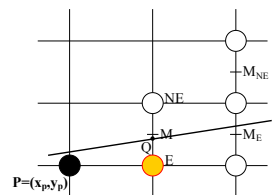
Algoritmo di Bresenham



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

- ❖ Se voglio che anche il valore di d sia costruito in maniera incrementale mi devo chiedere qual è il prossimo M, e quindi quanto vale d , al prossimo passo (sulla prossima colonna) sulla base della scelta fatta a questo passo

Algoritmo di Bresenham



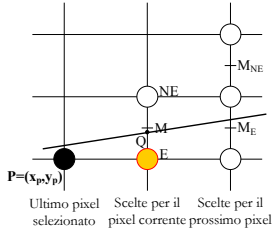
Ultimo pixel selezionato Scelte per il pixel corrente Scelte per il prossimo pixel

- ❖ Se l'ultimo pixel selezionato è stato E

$$d_{\text{new}} = F(x_p + 2) \cdot \Delta y \cdot b \left(\frac{1}{2} + \frac{1}{2} \right) + c$$

Algoritmo di Bresenham

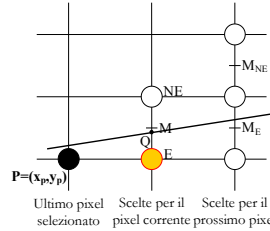
- ❖ Se l'ultimo pixel selezionato è stato E



- ❖ poiché
- $$d_{\text{new}} = a(x_p + 2) + b(y_p + \frac{1}{2}) + c$$
- $$d = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$$

Algoritmo di Bresenham

- ❖ Se l'ultimo pixel selezionato è stato E



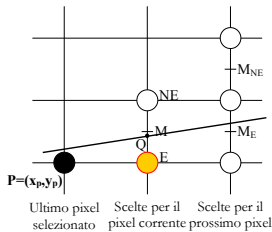
- ❖ poiché
- $$d_{\text{new}} = a(x_p + 2) + b(y_p + \frac{1}{2}) + c$$
- ❖ sottraendo si ha
- $$d = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$$
- $$d_{\text{new}} = d_{\text{old}} + a$$

Algoritmo di Bresenham

- ❖ L'incremento da aggiungere a d dopo aver scelto E lo chiamiamo Δ_E

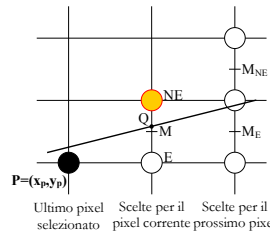
$$\Delta_E = a = dy$$

- ❖ Questo è un risultato generico e vale per ogni passo della rasterizzazione



Algoritmo di Bresenham

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



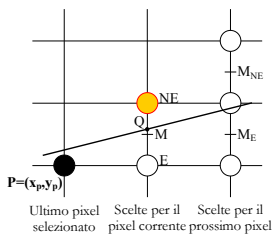
$$d_{\text{new}} = a(x_p + 2) + b(y_p + \frac{3}{2}) + c$$

Algoritmo di Bresenham

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

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

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



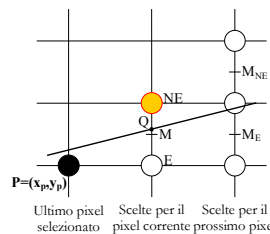
Algoritmo di Bresenham

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

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

- ❖ da cui
- $$d_{\text{new}} = d_{\text{old}} + a + b$$

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



Algoritmo di Bresenham

- ❖ Cosa abbiamo quindi costruito?
- ❖ Un algoritmo che
 - ❖ ad ogni passo sceglie il prossimo pixel tra due possibili candidati basandosi sul valore corrente di una variabile (detta di decisione)
 - ❖ ricalcola il valore della variabile di decisione incrementalmente aggiungendo al suo valore corrente una quantità fissa predefinita (Δ_E o Δ_{NE})

Algoritmo di Bresenham

- ❖ A questo punto ci serve solo un valore di inizializzazione della variabile d
- ❖ Il valore iniziale è

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

Algoritmo di Bresenham

- ❖ A questo punto ci serve solo un valore di inizializzazione della variabile d
 - ❖ Il valore iniziale è
- $$F(x_0 + 1, y_0 + \frac{1}{2}) = F(x_0, y_0) + a + \frac{b}{2}$$
- ❖ (x_0, y_0) appartiene alla retta e $F(x_0, y_0) = 0$

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

Algoritmo di Bresenham

```

MidpointLine(int x0,
             int y0,
             int x1,
             int y1,
             int value)
{
    int dx, dy, incRE, incNE, d, x, y;
    while (x < x1) {
        if (d <= 0) {
            d = d + incRE;
            x++;
        } else {
            d = d + incNE;
            x++;
            y++;
        }
        writePixel(x, y, value);
    }
}
    
```

Inizializzazione

Variabili intere

Scelta di E

Scelta di NE

Moltiplico per 2 per avere solo interi

Rasterizzazione di poligoni



Convesso

- ❖ Vogliamo un algoritmo generico capace di rasterizzare poligoni di qualunque tipo:
 - ❖ Convesso

Rasterizzazione di poligoni



Convesso



Concavo

- ❖ Vogliamo un algoritmo generico capace di rasterizzare poligoni di qualunque tipo:
 - ❖ Convesso
 - ❖ Concavo

Rasterizzazione di poligoni



Convesso



Concavo



Intrecciato

❖ Vogliamo un algoritmo generico capace di rasterizzare poligoni di qualunque tipo:

- ❖ Convesso
- ❖ Concavo
- ❖ Intrecciato

Rasterizzazione di poligoni



Convesso



Concavo



Intrecciato



Contorni multipli

❖ Vogliamo un algoritmo generico capace di rasterizzare poligoni di qualunque tipo:

- ❖ Convesso
- ❖ Concavo
- ❖ Intrecciato
- ❖ Contorni multipli

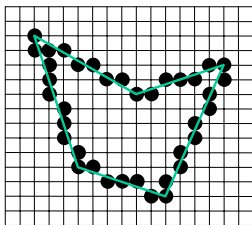
Rasterizzazione di poligoni

- ❖ L'algoritmo che vedremo ha queste caratteristiche
- ❖ Per ottenere questo risultato si ricavano una dopo l'altra le *span* di pixel che stanno dentro il poligono
- ❖ I punti estremi delle *span* sono calcolati per mezzo di un algoritmo *incrementale*, simile a quello visto per i segmenti

Rasterizzazione di poligoni

- ❖ L'algoritmo di filling consiste nella soluzione di due problemi successivi:
 - ❖ Rasterizzare i contorni del poligono
 - ❖ Rasterizzare l'interno basandosi sulla rasterizzazione dei contorni
- ❖ I due passi possono essere eseguiti anche non in successione stretta

I contorni



❖ La maniera più immediata di calcolare le intersezioni è utilizzare l'algoritmo di scan-conversion delle linee su ogni spigolo del poligono

I contorni

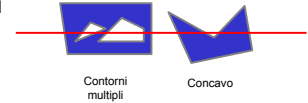
- ❖ L'algoritmo che consideriamo lavora invece incrementalmente sulle scan-line
- ❖ Una volta operato filling del poligono su una scan-line (identificati i pixel della scan-line che appartengono al poligono) adopera le informazioni trovate per aggiornare incrementalmente le intersezioni e fare filling sulla scan-line successiva

I contorni

- ❖ L'operazione di filling di una singola scan-line si svolge in 3 passi:
 - ❖ Trovare 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, usando per la determinazione di quali pixel sono interni, la **regola odd-parity**

Regola odd-parity

- ❖ Si attiva un bit detto di parità che può assumere valore **pari** o **dispari**
- ❖ La parità è inizialmente pari, ogni intersezione cambia il bit di parità, si disegnano i pixel quando la parità è dispari, non si disegnano quando la parità è pari



Interno/esterno

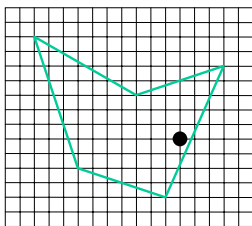
- ❖ Prima di passare ad analizzare i problemi di intersezione e sorting, vediamo come si definisce, in tutti i casi particolari che possono sorgere, se un pixel è interno o meno al poligono
- ❖ I casi da prendere in considerazione sono 4

Caso 1

Data un'intersezione con un valore generico della x razionale, come determino quale dei due pixel sui lati dell'intersezione è quello cercato?

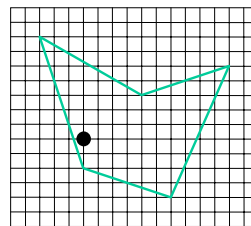
Se stiamo incontrando un'intersezione provenendo da *dentro* il poligono (il parity bit è dispari) arrotondiamo all'intero minore per rimanere dentro
Se siamo *fuori* dal poligono (il parity bit è pari) arrotondiamo all'intero maggiore per entrare dentro

Caso 1



Se stiamo incontrando un'intersezione provenendo da *dentro* il poligono (il parity bit è dispari) arrotondiamo all'intero minore per rimanere dentro

Caso 1



Se stiamo incontrando un'intersezione provenendo da *dentro* il poligono (il parity bit è dispari) arrotondiamo all'intero minore per rimanere dentro
Se siamo *fuori* dal poligono (il parity bit è pari) arrotondiamo all'intero maggiore per entrare dentro

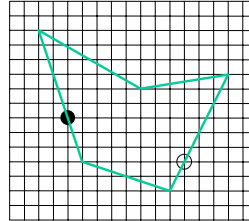
Caso 2

Come tratto 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

Caso 2

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



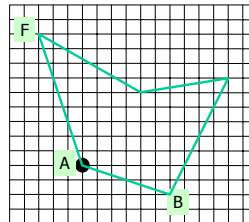
Caso 3

E se l'intersezione a coordinate intere riguarda un vertice?

Nel calcolo del parity bit, si considera solo il vertice y_{\min} e non il vertice y_{\max}

Caso 3

Nel calcolo del parity bit, si considera solo il vertice y_{\min} e non il vertice y_{\max} . Nella figura il vertice A è considerato solo come vertice y_{\min} dello spigolo FA e non come vertice y_{\max} dello spigolo AB



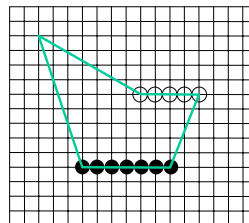
Caso 4

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

Basta considerare i vertici di una linea orizzontale come non influenti nel calcolo del parity bit e si ottiene automaticamente che i lati orizzontali in basso vengano disegnati e quelli in alto no

Caso 4

Basta considerare i vertici di una linea orizzontale come non influenti nel calcolo del parity bit e si ottiene automaticamente che i lati orizzontali in basso vengano disegnati e quelli in alto no



Le intersezioni

- ❖ Per calcolare le intersezioni vogliamo evitare di fare un test che verifichi l'intersezione tra la scan-line ed ogni spigolo del poligono
- ❖ Ci piacerebbe avere un metodo più *furbo* (efficiente)

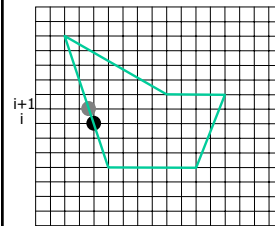
Le intersezioni

- ❖ Notando che molti spigoli che intersecano la scan-line i intersecano anche la scan-line $i+1$, si può utilizzare un approccio incrementale molto simile a quello dell'algoritmo di scan-conversion per le linee
- ❖ Il valore dell'intersezione con la scan-line i mi serve per calcolarla al passo $i+1$

Le intersezioni

- ❖ La differenza tra questo algoritmo e quello di rasterizzazione di segmenti consiste nel fatto che nel caso della rasterizzazione di linee devo selezionare il pixel più *vicino* alla linea ideale, mentre in questo caso devo tenere conto del *dentro* e del *fuori* e arrotondare per restare *dentro* il poligono

Le intersezioni



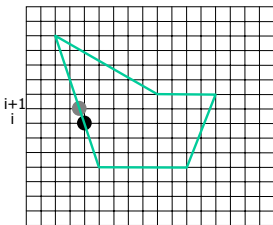
- ❖ Conoscendo le intersezioni per una scan-line, quando passo alla scan-line successiva le intersezioni si ricalcoleranno con la formula

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

- ❖ Dove $m = \frac{(y_{\max} - y_{\min})}{(x_{\max} - x_{\min})}$ è il coefficiente angolare della linea-spigolo

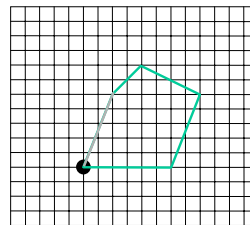
Le intersezioni

- ❖ Anziché utilizzare aritmetica reale per calcolare gli incrementi $1/m$ considero l'incremento come numero razionale, e uso il suo numeratore e il suo denominatore



Esempio

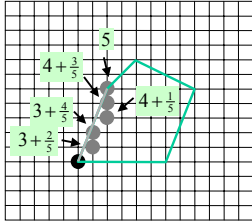
- ❖ Linea-spigolo sinistra
- ❖ Coefficiente angolare $m > 1$
- ❖ $x_{\min} = 3$
- ❖ $m = 5/2$



3

Esempio

- ❖ La sequenza di valori della x sarà:

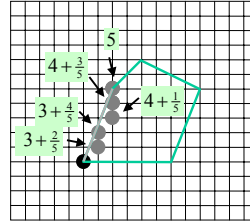


$$\begin{aligned}
 &3 \\
 &3 + \frac{2}{5} \\
 &3 + \frac{4}{5} \\
 &3 + \frac{6}{5} = 4 + \frac{1}{5} \\
 &4 + \frac{3}{5} \\
 &4 + \frac{5}{5} = 5
 \end{aligned}$$

3 4 5

Esempio

- ❖ 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



3 4 5

Spigolo sinistro

```

LeftEdgeScan (int xmin,
              int ymin,
              int xmax,
              int ymax,
              int value)
{
    int x, y, numerator, denominator, increment;
    x = xmin;
    numerator = xmax-xmin;
    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;
        }
    }
}
    
```

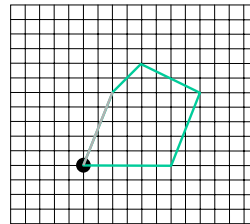
Variabili intere

Inizializzazione

Esempio

- ❖ Le sequenze di valori delle variabili sono:

increment	x
5	3

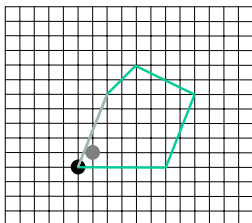


3 4 5

Esempio

- ❖ Le sequenze di valori delle variabili sono:

increment	x
5	3
7→2	4

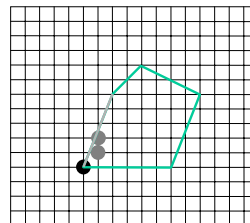


3 4 5

Esempio

- ❖ Le sequenze di valori delle variabili sono:

increment	x
5	3
7→2	4
4	4

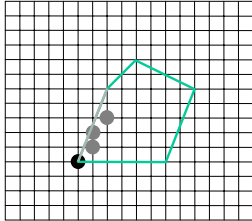


3 4 5

Esempio

❖ Le sequenze di valori delle variabili sono:

increment	x
5	3
7→2	4
4	4
6→1	5



3 4 5

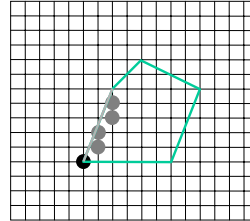
Costruzione di Interfacce - Paolo Cignoni

73

Esempio

❖ Le sequenze di valori delle variabili sono:

increment	x
5	3
7→2	4
4	4
6→1	5
3	5



3 4 5

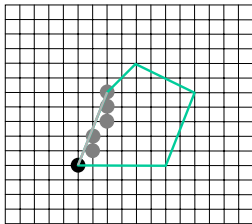
Costruzione di Interfacce - Paolo Cignoni

74

Esempio

❖ Le sequenze di valori delle variabili sono:

increment	x
5	3
7→2	4
4	4
6→1	5
3	5
5	5



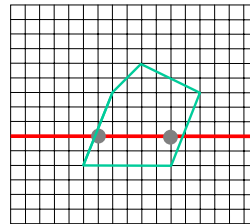
3 4 5

Costruzione di Interfacce - Paolo Cignoni

75

Esempio

❖ Il calcolo incrementale dei valori delle linee spigolo non avviene in un'unica soluzione ma deve essere interrotto salvando i valori calcolati al passo precedente



Costruzione di Interfacce - Paolo Cignoni

76

La struttura dati

❖ Per far questo abbiamo necessità di una struttura dati adeguata che consenta di:

- ❖ Trovare 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

Costruzione di Interfacce - Paolo Cignoni

77

La struttura dati

❖ La struttura dati che utilizzeremo a questo scopo sarà una lista che chiamiamo

Active Edge Table (tabella degli spigoli attivi) nella quale inseriamo le informazioni a partire da un'altra struttura dati, la **Edge Table** (tabella degli spigoli)

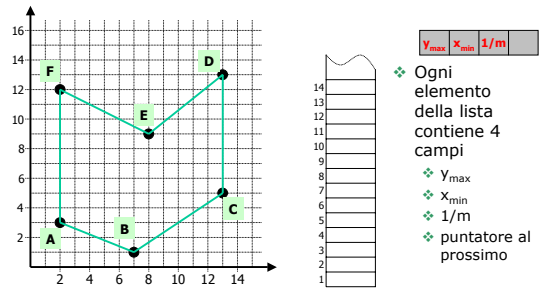
Costruzione di Interfacce - Paolo Cignoni

78

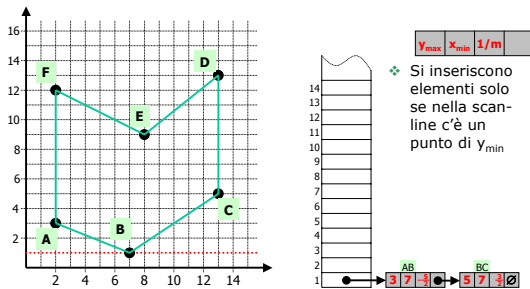
Edge Table

- ❖ La Edge Table viene costruita all'inizio dell'esecuzione dell'algoritmo e contiene tutte le informazioni necessarie per la rasterizzazione degli spigoli
- ❖ E' un array di liste, con tante celle per quante sono le scan-line dello schermo

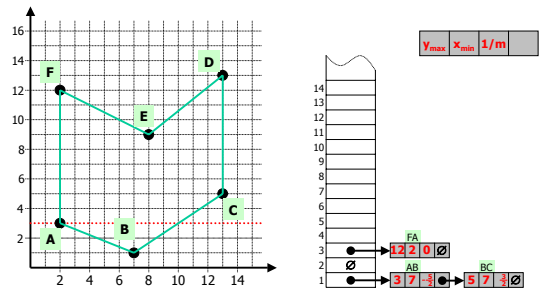
Edge Table



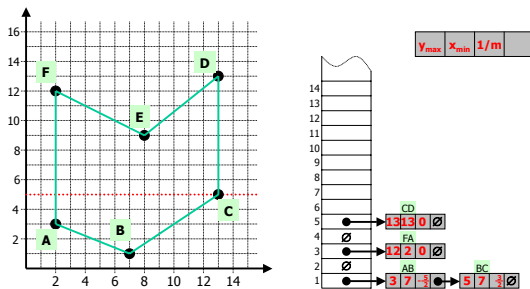
Edge Table



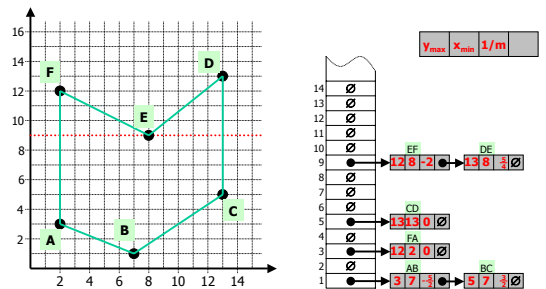
Edge Table



Edge Table

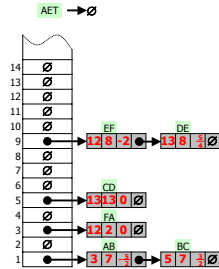


Edge Table



Active Edge Table

- ❖ La Active Edge Table viene costruita e modificata copiando elementi della lista dalla Edge Table
- ❖ Inizialmente sarà vuota

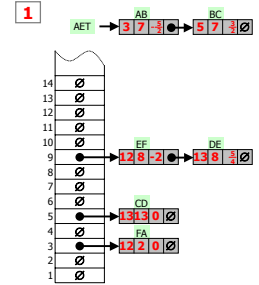


Costruzione di Interfacce - Paolo Cignoni

85

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

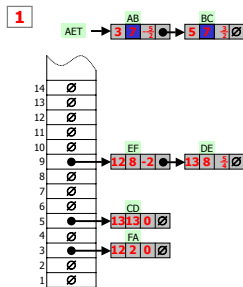


Costruzione di Interfacce - Paolo Cignoni

86

Active Edge Table

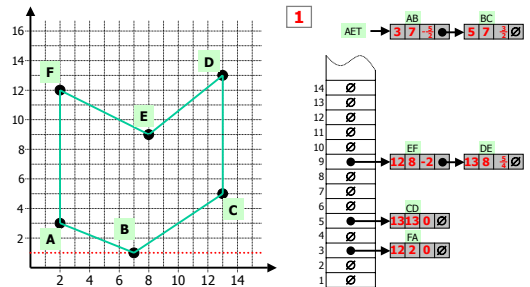
- ❖ Nella AET il secondo valore non rappresenta x_{min} bensì il valore della x corrente da usare per la rasterizzazione



Costruzione di Interfacce - Paolo Cignoni

87

Active Edge Table



Costruzione di Interfacce - Paolo Cignoni

88

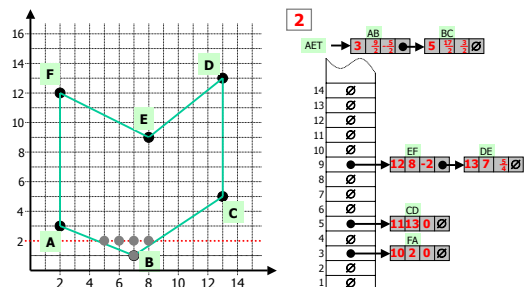
Algoritmo

- ❖ Settare y al minimo y non vuoto della ET
- ❖ Inizializzare la 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 fare sorting su AET per x
 - ❖ Disegnare i pixel della scan-line pescando 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

Costruzione di Interfacce - Paolo Cignoni

89

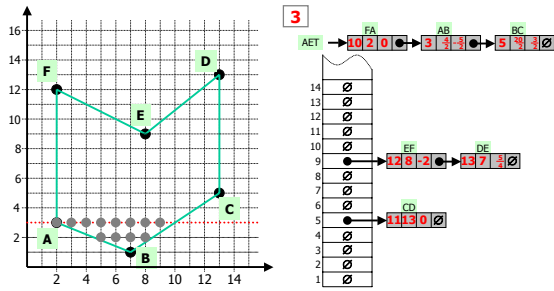
Active Edge Table



Costruzione di Interfacce - Paolo Cignoni

90

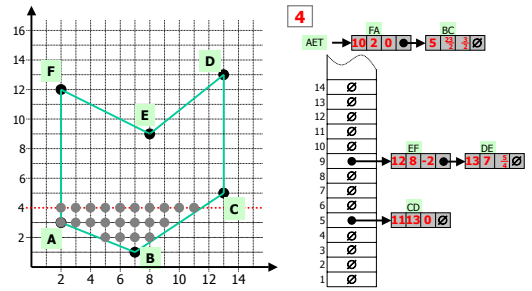
Active Edge Table



Costruzione di Interfacce - Paolo Cignoni

91

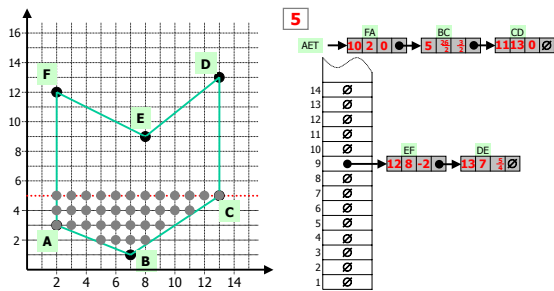
Active Edge Table



Costruzione di Interfacce - Paolo Cignoni

92

Active Edge Table



Costruzione di Interfacce - Paolo Cignoni

93