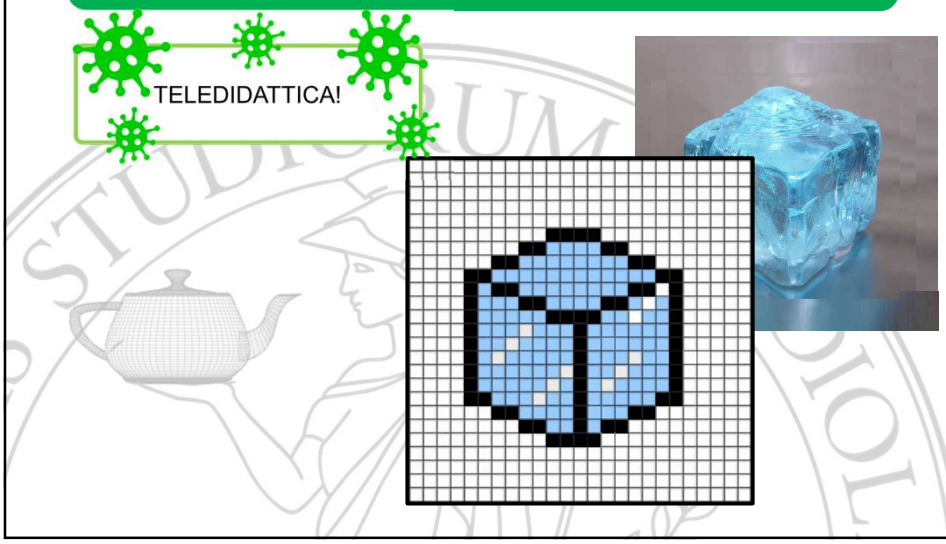
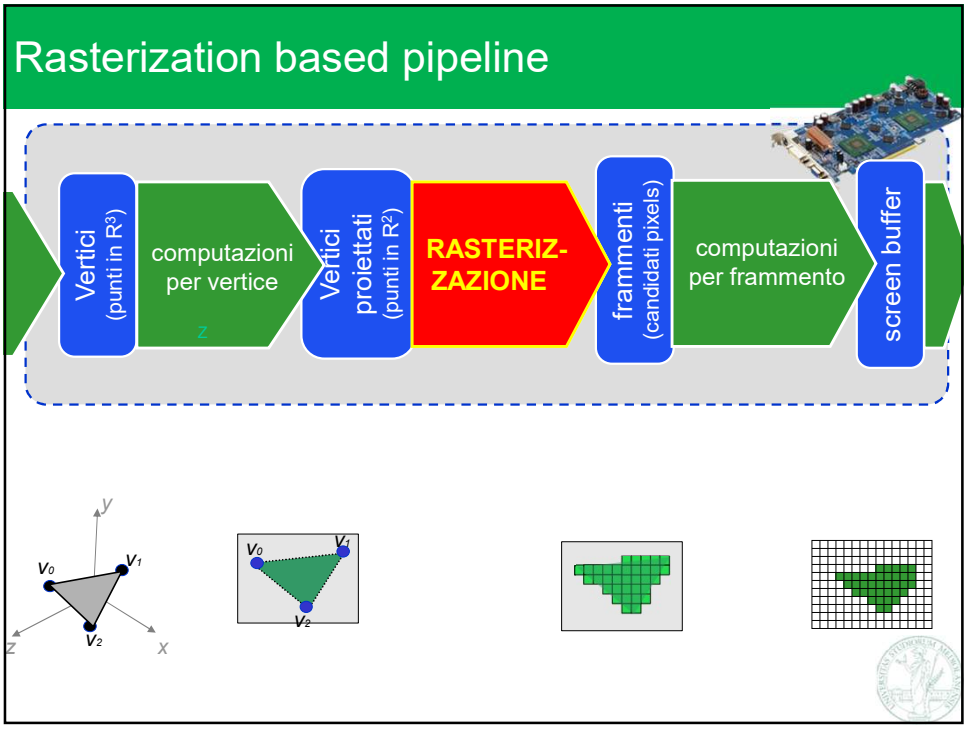
Marco Tarini - Computer Graphics 2020-2021
Università degli Studi di Milano

Rasterizing primitives



1

Rasterization based pipeline



2

Rasterizzazione

- ✓ Individuare i frammenti che compongono una primitiva
  - ⇒ Uno per ogni pixel coperto dalla primitva
  - ⇒ I frammenti prodotti vengono passata alla fase successiva (che dermina il colore schermi)
- ✓ E' un procedimento puramente 2D
- ✓ E' estremamente parallelizzabile
  - ⇒ Per sfruttare questo, si adotta un hardware parellelo
  - ⇒ Sulle GPU, è implementato in hardware (architettura specializzata per lo scopo)
  - ⇒ E' una fase dell'hardware molto efficiente e poco flessibile (non è «programmabile» dall'utente)
    in pratica: fa sempre la stessa cosa
  - ⇒ Viene implementato uno specifico algoritmo

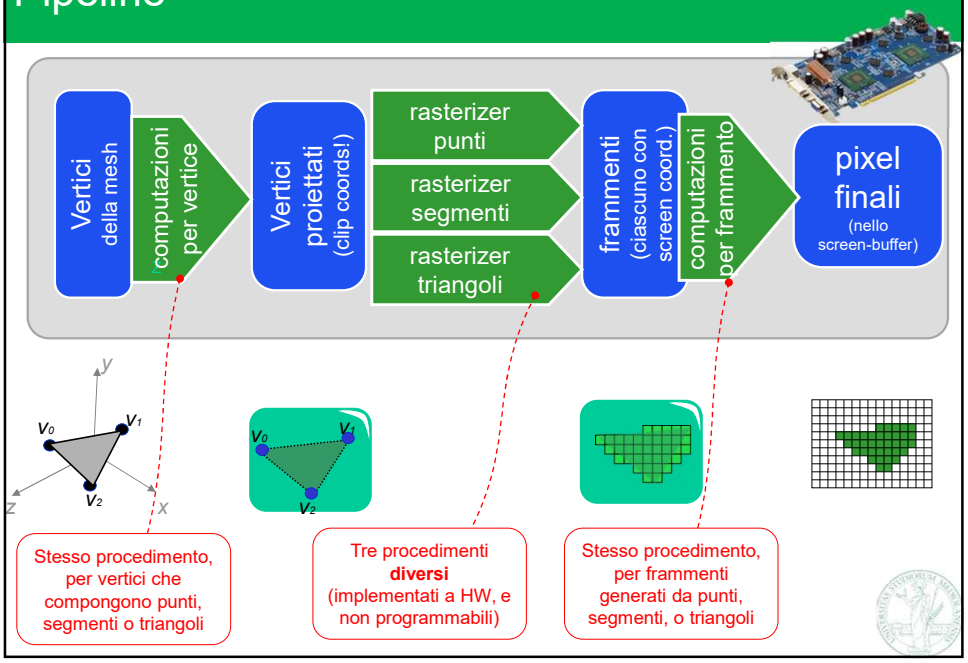
3

Rasterizzazione

- ✓ Tecnicamente, l'hardware GPU è pesato per rasterizzare solo tre primitive (2D, 1D, 0D)
  - ⇒ Triangoli – 3 vertici
  - ⇒ Segmenti («linee») – 2 vertici
  - ⇒ Punti («point splats») – 1 vertice
- ✓ Il primo caso è particolarmente utile ed ottimizzato
- ✓ I vertici sono passati al rastrizzatore in coordiante clip omogenee
- ✓ Per prima cosa, vengono convertite in coordinate schermo cartesiane
  - ⇒ come sappiamo già
- ✓ Il rasterizzatore usa solo le x e la y,

4

Pipeline



7

Rasterizzare triangoli

- ✓ Individuare i frammenti che compongono i triangoli
  - ⇒ *Input:* tre vertici (punti 2D in screen space)
  - ⇒ *Output:* i frammenti interni al triangolo
- ✓ Più precisamente: devono essere prodotti tutti e soli i frammenti il cui centro è interno al triangolo
- ✓ Il rasterizzatore si occupa anche di INTERPOLARE GLI ATTRIBUTI
  - ⇒ *Input ulteriore:* tre set di attributi (vettori, scalari…)
  - ⇒ *Output ulteriore:* l'interpolazione di ciascun attributo per ogni frammento prodotto (iterpolazione con coordinate baricentriche)

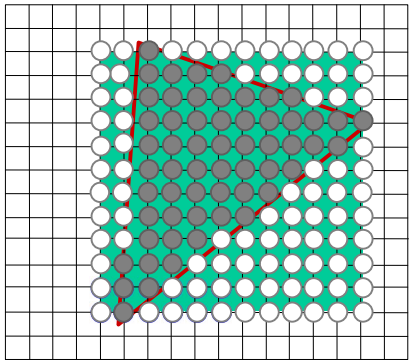
8

Rasterizzazione

- ✓ Il rasterizzatore usa solo le x e la y (in spazio schermo),
- ✓ la z (cioè il valore di depth del frammento) è considerato un attributo da interpolare

9

Un algoritmo di rastizzazione per triangoli parallelizzato

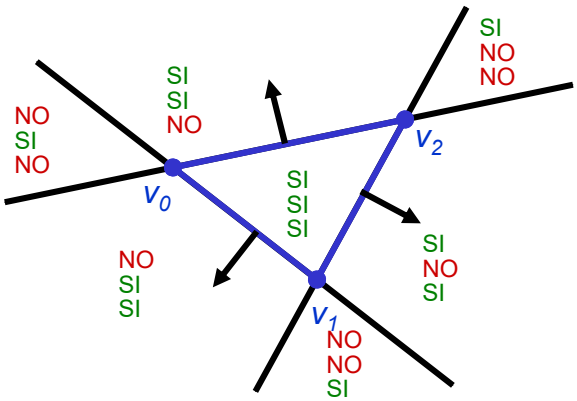


1. Trovo un rettangolo (coordianate intere) che contiene il triangolo
2. Processo tutti le posizioni interne (nota: PARALLELIZZABILE)
3. Processo ogni posizione interna: se è dentro al triangolo, produco un frammento

10

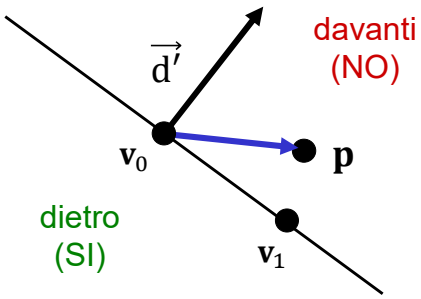
Test di appartenenza ad un triangolo

- ✓ Triangolo = interesezione di 3 semipiani
- ✓ Un punto è interno al triangolo *sse* appartiene a tutti e tre i semipiani



11

Test di appartenenza ad un semipiano (in 2D!)



Da quale parte della retta passante per $\mathbf{v}_0$ e $\mathbf{v}_1$ si trova il punto $\mathbf{p}$?

Soluzione:

$$\vec{d} = (\mathbf{v}_1 - \mathbf{v}_0) = \begin{pmatrix} d_x \\ d_y \end{pmatrix}$$

$$\vec{d}' = \begin{pmatrix} -d_y \\ d_x \end{pmatrix}$$

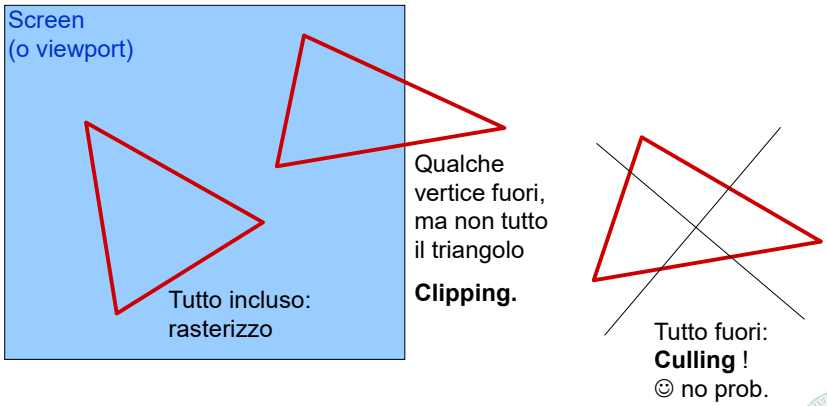
cioè $\vec{d}'$ è il vettore $\vec{d}$ ruotato di 90 gradi

Basta verificare se:

$$(\mathbf{q} - \mathbf{p}_0) \bullet \vec{d}' < 0$$

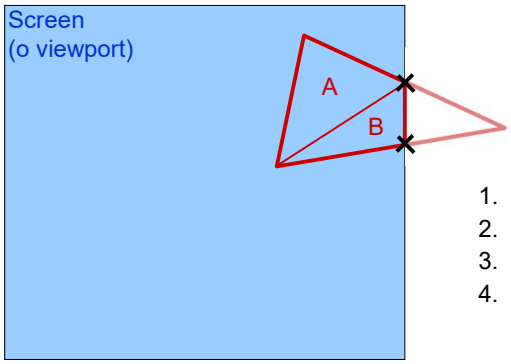
12

Clipping e culling



13

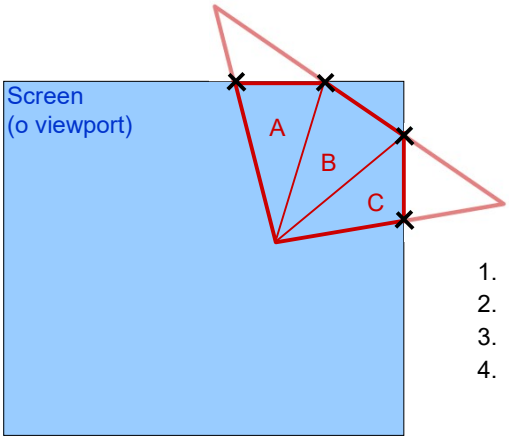
Clipping: la vecchia scuola



1. Trovo intersezioni
2. Unisco intersezioni
3. Divido poligono in triangoli
4. Rasterizzo ogni triangolo
   A e B

14

Clipping: la vecchia scuola



1. Trovo intersezioni
2. Unisco intersezioni
3. Divido poligono in triangoli
4. Rasterizzo ogni triangolo
   A, B, e C

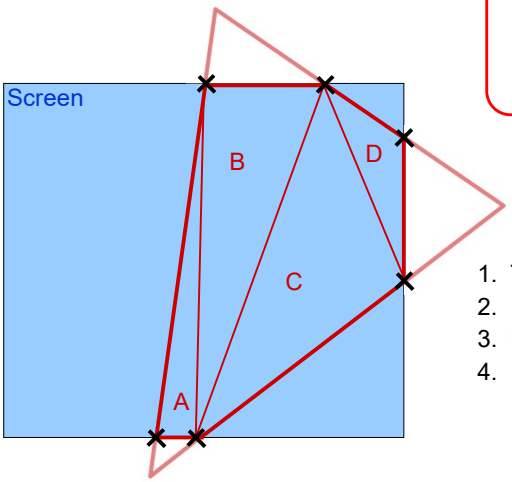
15

Clipping: la vecchia scuola

Caso pessimo molto complicato

Non adatto ad
una implementazione HW

L'HW deve prevedere il
caso pessimo, anche se è raro

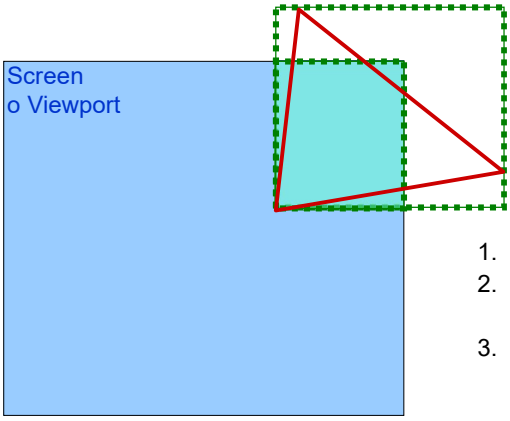


1. Trovo intersezioni
2. Unisco intersezioni
3. Divido poligono in triangoli
4. Rasterizzo ogni triangolo
   A,B,C,D

16

Clipping: versione più adatta ad una implementazione HW

✓Clipping!



Come si interseca un bounding box con lo schermo (cioe' col rettangolo definito dal ViewPort)?

1. Trovo bounding box
2. Interseco bounding box con schermo (o viewport)
3. Rasterizzo nel bounding box come normale

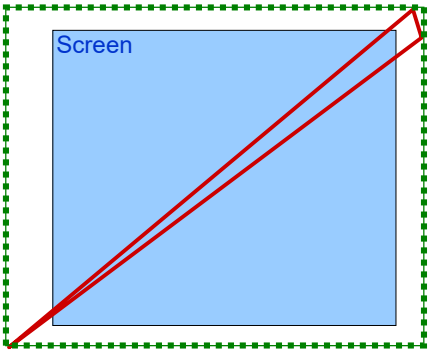
Rasterizzazione triangoli:

✓Il metodo basato su bounding box e test di appartenenza (edge functions):

⇒Vantaggi
- fortemente parallelizzabile
  - (es: gruppi 4x4)
- clipping diventa facile facile
  - per i planes UP, DOWN, LEFT e RIGHT

⇒Svantaggi
- Overhead grandini
  - Si testano molti frammenti inutilmente
  - Specialmente quando...

Un caso sfortunato

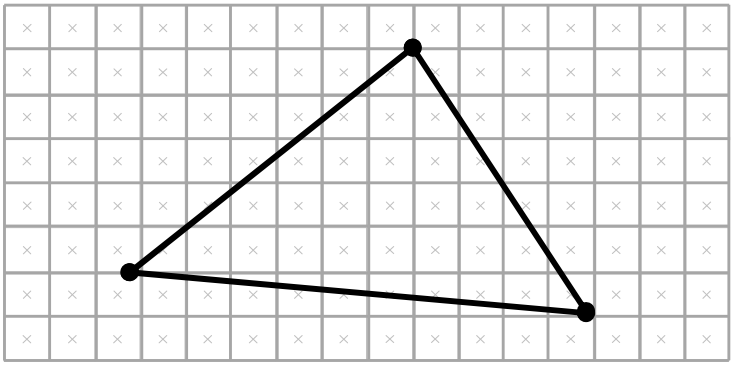


Per questo motive
(e molti altri)
i triangoli lunghi e stretti
non sono ideali

Triangoli:
- lunghi e stretti
- orientati lungo la direzione diagonale ☹

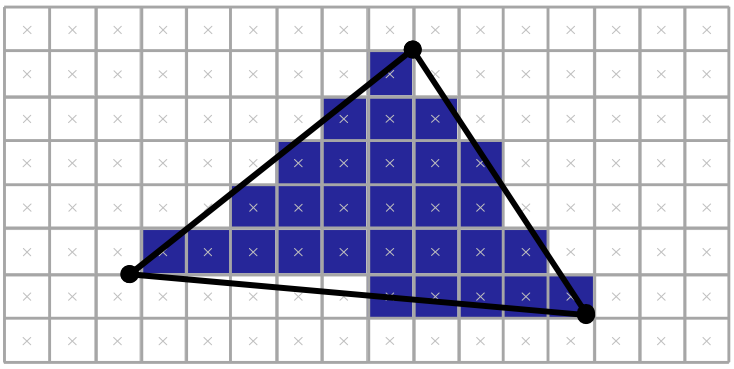
20

Rasterizzare Triangoli



21

Rasterizzare Triangoli

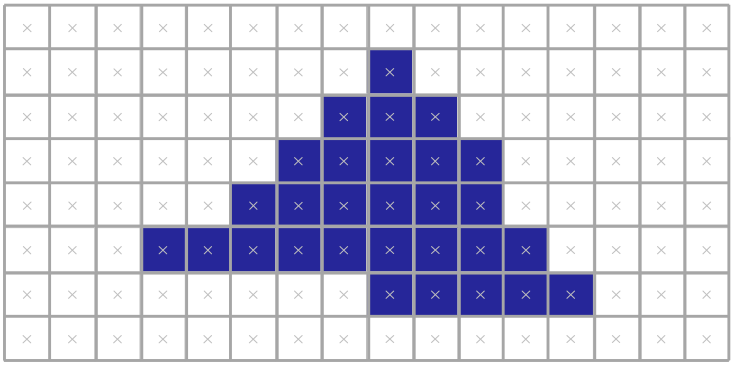


Pixel
(cross = center; x,y @ 0.5)

Triangle

22

Rasterizzare Triangoli

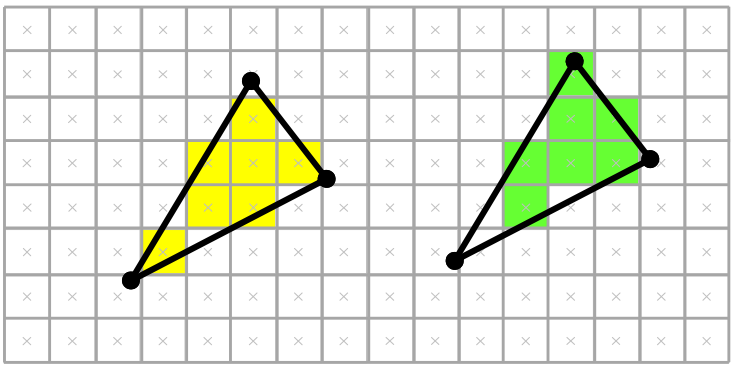


Pixel
(cross = center; x,y @ 0.5)

Triangle

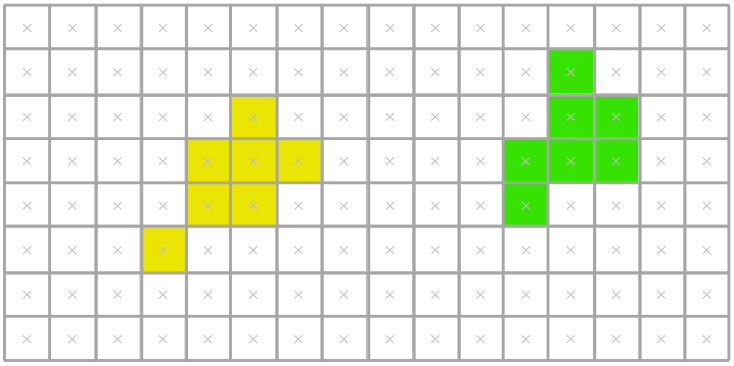
23

Rasterizzare Triangoli



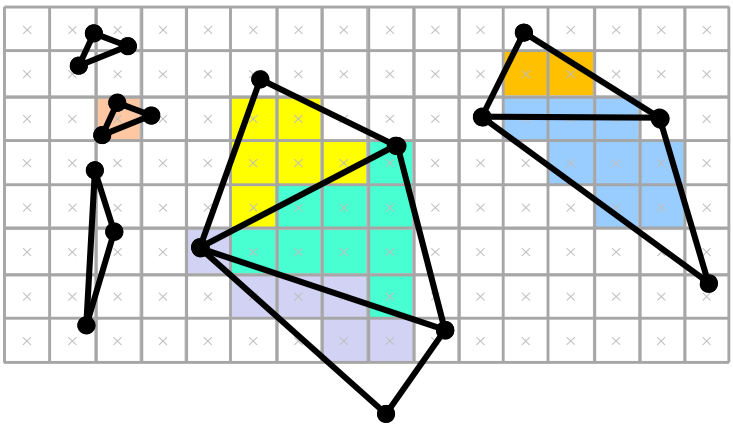
27

Rasterizzare Triangoli



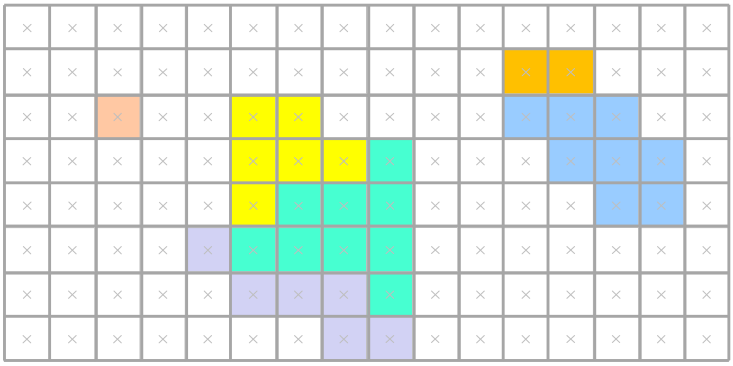
28

Rasterizzare Triangoli



38

Rasterizzare Triangoli



39

Note: rasterizzazione triangoli 1/2

- ✓ La rasterizzazione avviene in 2D
  - ⇒ input: tre punti in 2D
    - (coordinate X,Y non intere, in generale)
  - ⇒ output: frammenti
- ✓ Viene generato un frammento per ogni pixel il cui centro ricade dentro il triangolo 2D
- ✓ Un triangolo può generare qualsiasi numero di frammenti
  - ⇒ anche nessuno
  - ⇒ anche tutti
- ✓ Se un triangolo è parzialmente fuori al viewport, vengono generati solo i frammenti interni («clipping»)

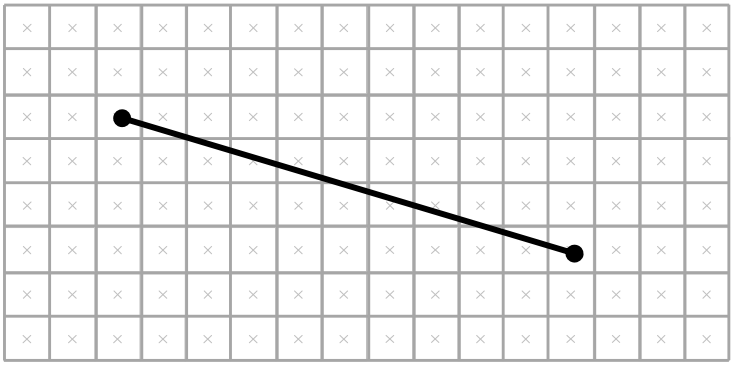
40

Note: rasterizzazione triangoli 2/2

- ✓ Se un triangolo è parzialmente fuori al viewport, vengono generati solo i frammenti interni
  - ⇒ «clipping» della primitiva triangolo
    - «clipping» = spezzare la primitive
  - ⇒ se è completamente fuori dal viewport: nessun frammento generato
- ✓ Test che il rasterizzatore compie:
  - ⇒ "il centroide di questo pixel casca dentro a questo triangolo 2D?"
  - ⇒ traccia: considerare il triangolo come intersezione di tre semipiani
- ✓ Garanzia: la rasterizz. di due triangoli che condividono esattamente un edge (un lato, due vertici), genera 1 frammento per ogni pixel nella loro unione
  - ⇒ no sovrapposizioni (pixel coperto da due frammenti, uno per ogni tri)
  - ⇒ no gaps (pixel appartenente all'unione non coperto da alcun frammento)

41

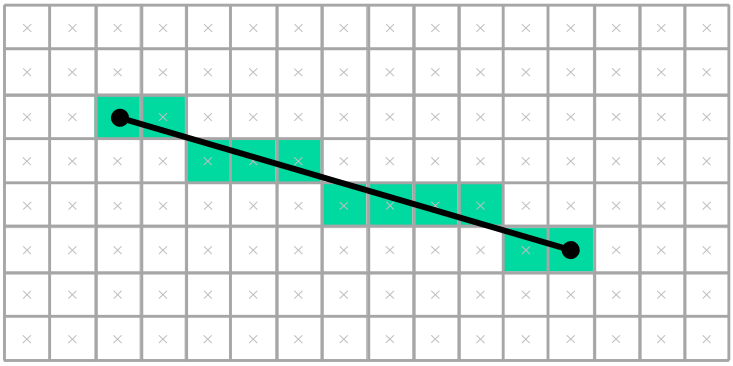
Rasterizzare Linee



Bresenham's line algorithm (1962)

42

Rasterizzare Linee



Bresenham's line algorithm (1962)

43

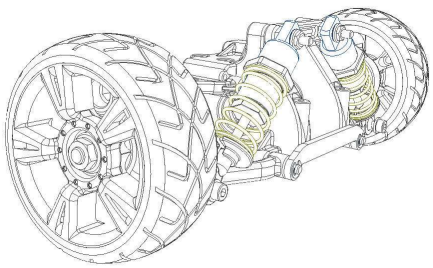
Nota storica: Bresenham's line algorithm

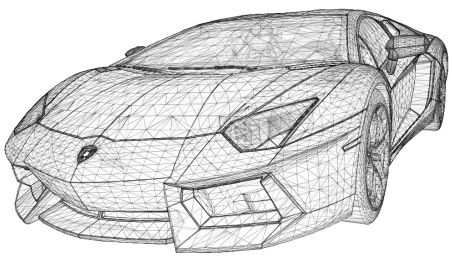
- ✓Un vecchio algoritmo per rasterizzare line
- ✓Vantaggi:
  - ⇒Richiede solo computazioni fra interi (no virgola mobile)
  - ⇒Efficienza
- ✓Svantaggi:
  - ⇒sequenziale (non parallelizzabile)
  - ⇒iterazione del ciclo dipende dalla precedente ☹
- ✓Molto usato in passato, non più oggi (almeno nelle GPU)

44

Rasterizzare Linee

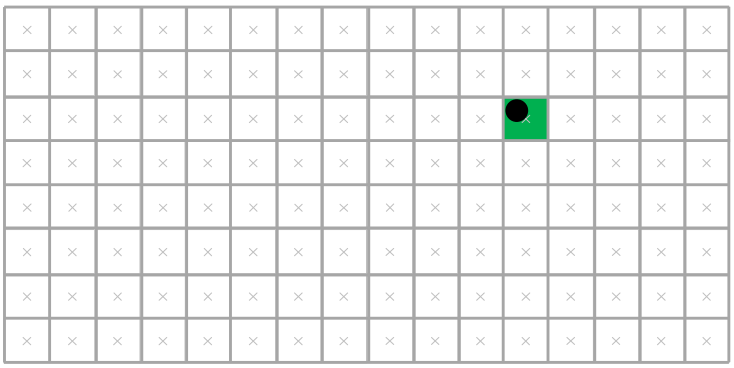
- ✓La rasterizzazione delle linee è usata nell modalità di rendering wireframe





45

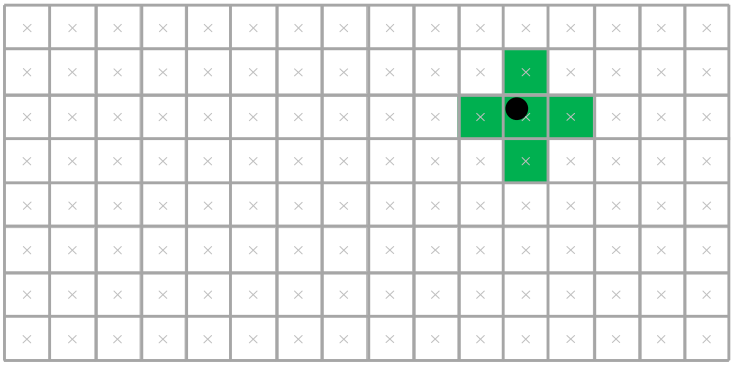
Rasterizzare punti



Point “splat”

47

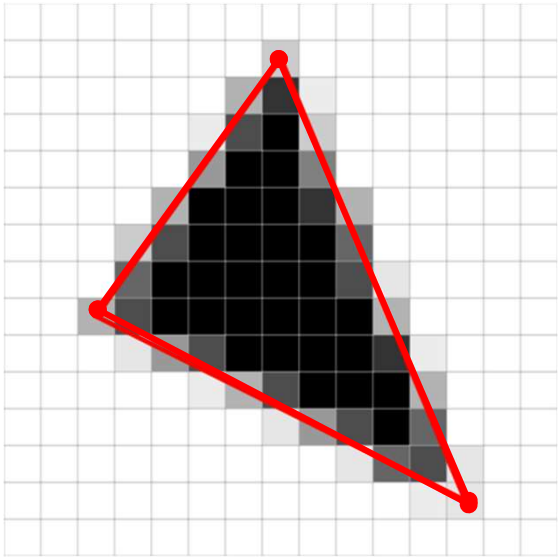
Rasterizzare punti



Point “splat”

48

Antialiasing: nel rasterizzare triangoli

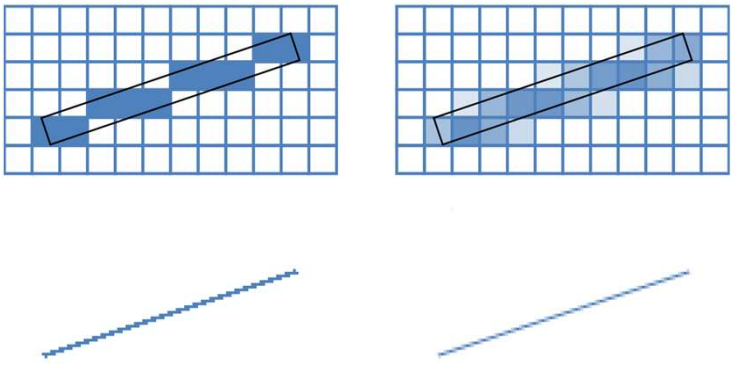


49

Antialiasing: nel rasterizzare segmenti

Senza:

Con:



50