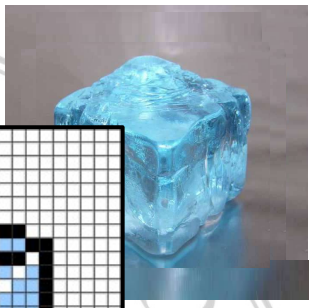
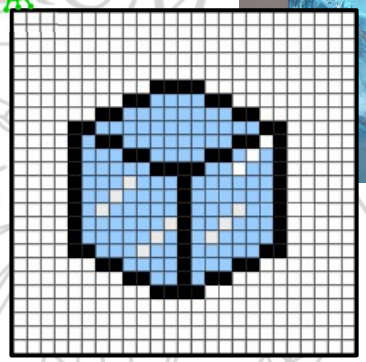
Marco Tarini - Computer Graphics 2019-2020
Università degli Studi di Milano

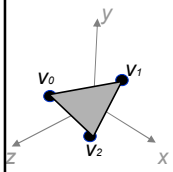
Rasterization

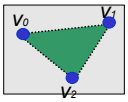


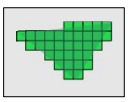


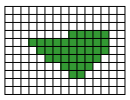
12

Rasterization based pipeline









Vertici (punti in R^3) → computazioni per vertice → Vertici proiettati (punti in R^2) → RASTERIZ-ZAZIONE → frammenti (candidati pixels) → computazioni per frammento → screen buffer

13

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

14

Rasterizzazione

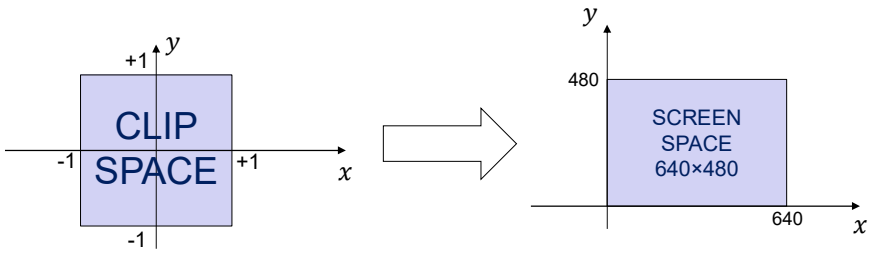
- ✓ Tecnicamente, l'hardware GPU è può tipicamente rasterizzare solo tre primitive (2D, 1D, 0D)
  - ⇒ Triangoli – 3 vertici
  - ⇒ Segmenti («linee») – 2 vertici
  - ⇒ Punti («point splats») – 1 vertice
- ✓ Il primo caso è particolarmente utile ed ottimizzato
- ✓ I vertici sono passati al rastrizzatore in coordiante clip
  - ⇒ in 2D: il rasterizzatore ignora la z!
- ✓ Per prima cosa, vanno convertite in «spazio schermo»…

15

Da Clip Space a Screen Space (o "Viewport")

- ✓ Spazio Clip 2D: [-1 , +1] × [-1 , +1] (la z viene ignorata)
- ✓ Spazio Screen: [0 , resX-1] × [0, resY-1]
  - ⇒ Coordinate schermo (screen)
    o coordiante pixel
    o coordiante viewport

il rettangolo di schermo che contiene il rendering (nelle applicazioni non a schermo intero)



- ✓ In coordinate screen, i pixel (e i frammenti) hanno coordinate intere!

16

Da Clip Space a Screen Space (o Viewport)

$$f_{VIEWPORT}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} (x+1)/2 \cdot RES_X \\ (y+1)/2 \cdot RES_Y \end{pmatrix}$$

in $[-1,+1] \times [-1,+1]$

in $[0,1] \times [0,1]$

(nota: sono spazi 2D)

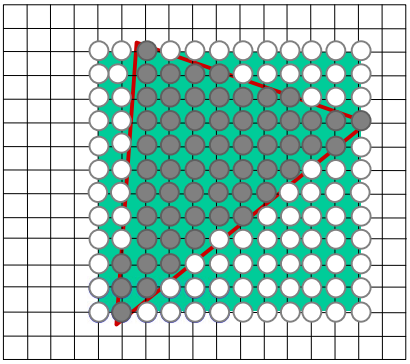
17

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)

18

Un algoritmo di rastizzazione per triangoli parallelizzato

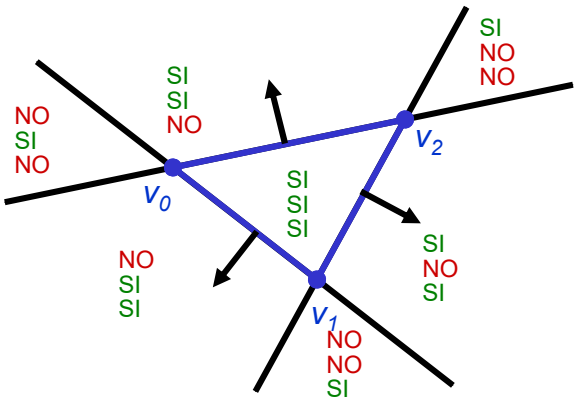


1. Trovo un rettangolo (coordianate intere) che contiene il triangolo
2. Processo tutti le posizioni interne (nota: PARALLELIZZABILE)
3. Processo ogni posizione interna: se è dentro al triangolo, produco un frammento

19

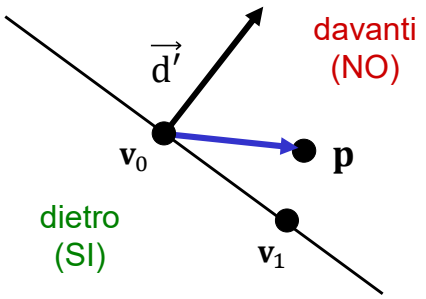
Test di appartenenza ad un triangolo

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



20

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:

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

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

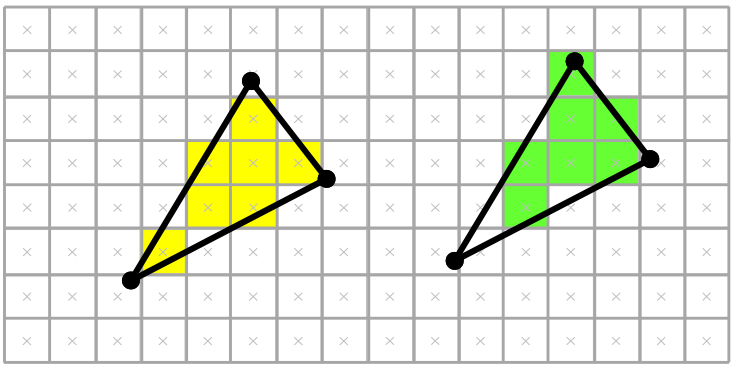
cioè $\vec{d}'$ è il vettore $\vec{d}$ ruotato di 90 gradi

Basta verificare se:

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

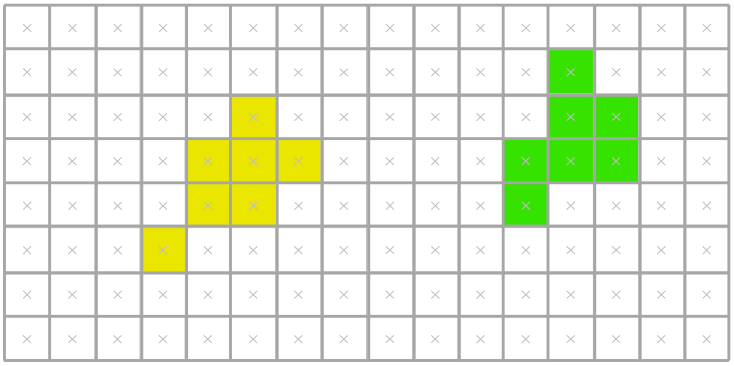
21

Rasterizzare Triangoli



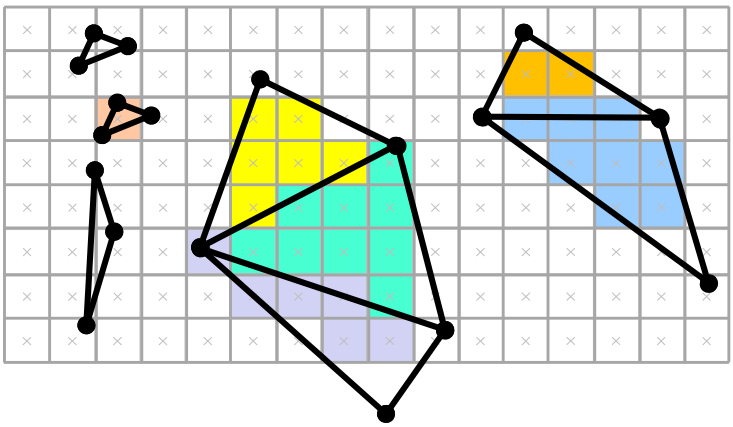
24

Rasterizzare Triangoli



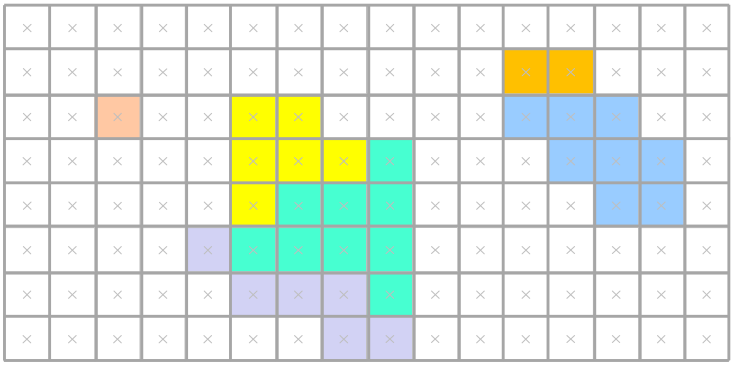
25

Rasterizzare Triangoli



35

Rasterizzare Triangoli



36

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»)

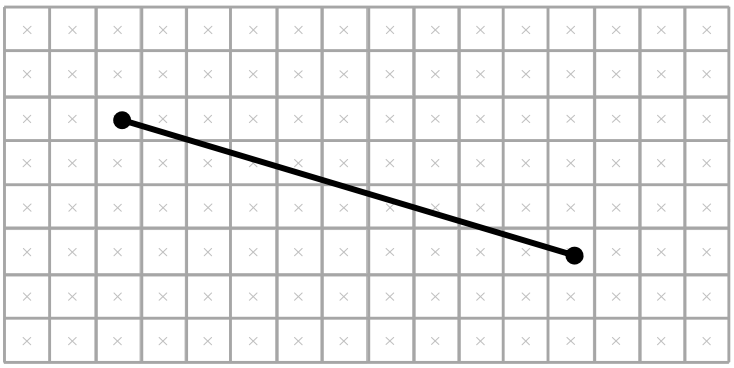
37

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)

38

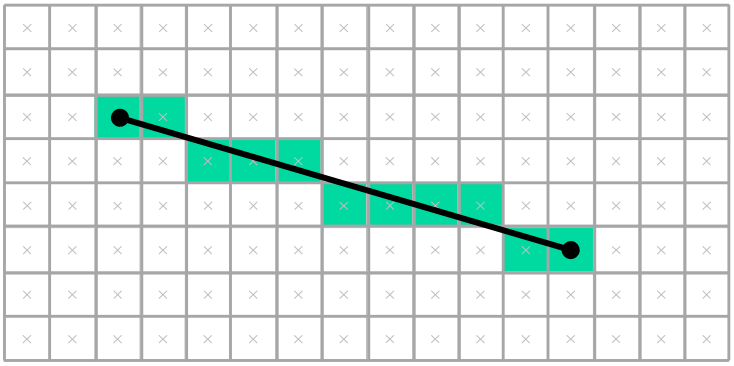
Rasterizzare Linee



Bresenham's line algorithm (1962)

39

Rasterizzare Linee



Bresenham's line algorithm (1962)

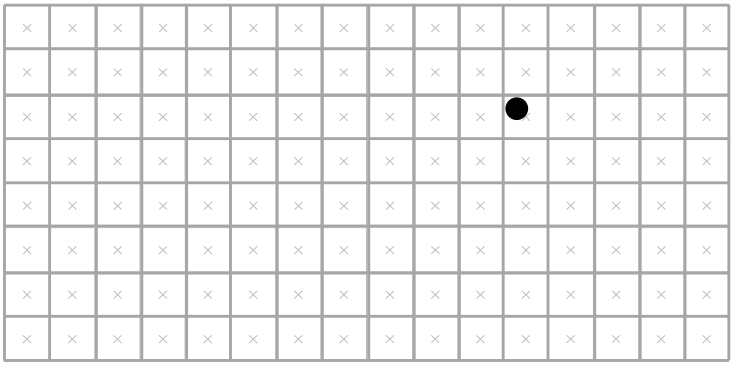
40

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)

41

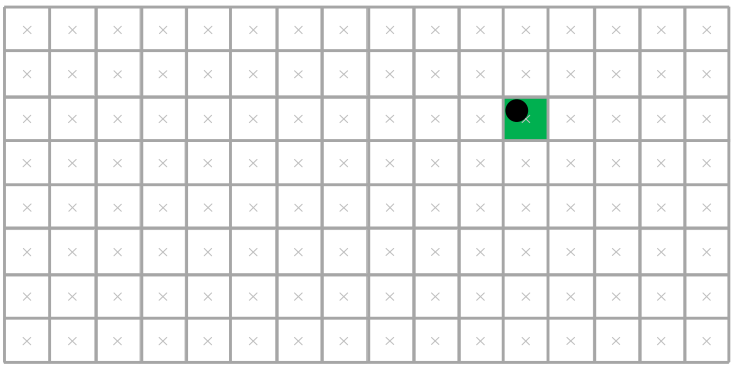
Rasterizzare punti



Point “splat”

42

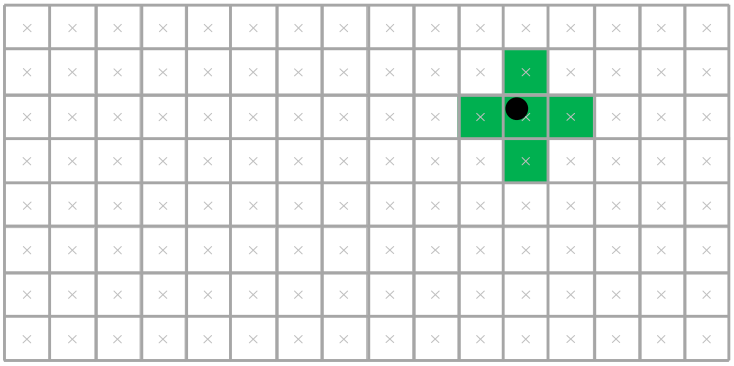
Rasterizzare punti



Point “splat”

43

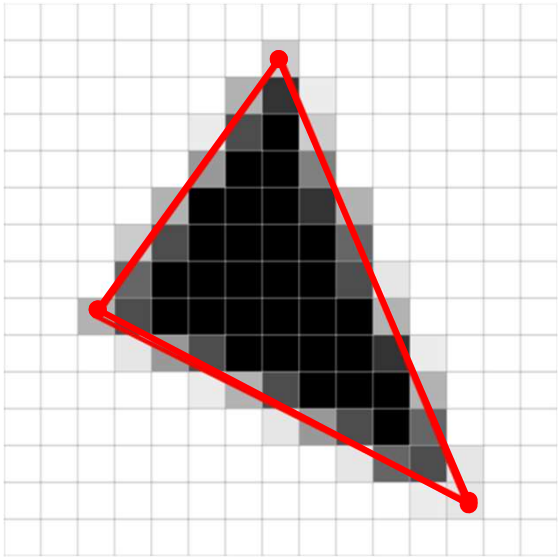
Rasterizzare punti



Point “splat”

44

Antialiasing: nel rasterizzare triangoli

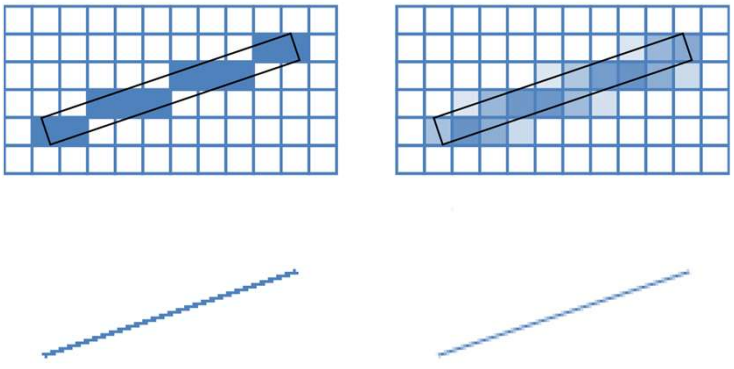


45

Antialiasing: nel rasterizzare segmenti

Senza:

Con:



46