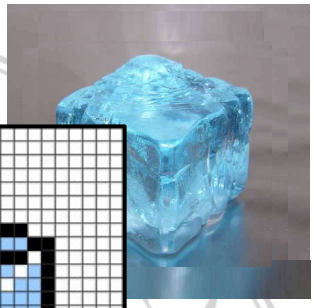
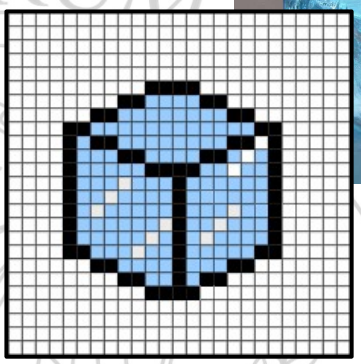
Marco Tarini - Computer Graphics 2022/2023
Università degli Studi di Milano

Rasterizing primitives

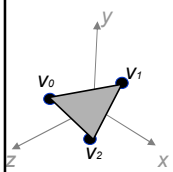


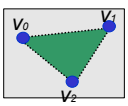


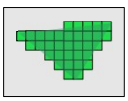
1

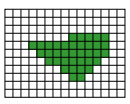
Rasterization based pipeline

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









2

Rasterizzazione

- ✓ Individuare i frammenti che compongono una primitiva
  - ⇒ Uno per ogni pixel coperto dalla primitiva
  - ⇒ I frammenti prodotti vengono passata alla fase successiva (che determina il colore RGB del pixel)
- ✓ 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) è «hard-wired» in pratica: fa sempre la stessa cosa,
  - ⇒ hardware diversi implementano algoritmi diversi (spesso)

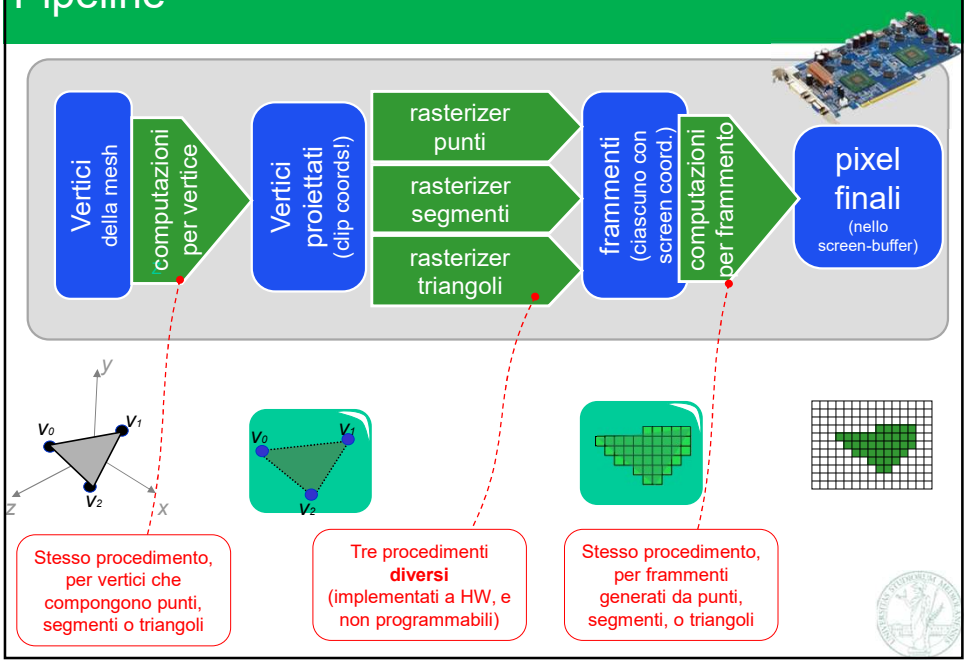
3

Rasterizzazione

- ✓ Gli hardware GPU attuali sono pesati per rasterizzare solo tre tipi di primitive:
  - ⇒ Triangoli 2D – 3 vertici
  - ⇒ Segmenti 1D («linee») – 2 vertici
  - ⇒ Punti 0D («point splats») – 1 vertice
- ✓ Ovviamente, il primo caso è particolarmente utile ed ottimizzato
- ✓ I vertici sono passati al rasterizzatore in coordinate clip omogenee
- ✓ Per prima cosa, vengono convertite in coordinate schermo cartesiane
  - ⇒ come sappiamo già
- ✓ Poi, il rasterizzatore usa solo le x e la y (in spazio schermo, cioè le coordinate pixel),
  - ⇒ la z (cioè la *depth* del pixel) viene ignorata in questa fase

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
- ✓ Ogni frammento prodotto ha coordinate schermo intere
- ✓ Devono essere prodotti tutti e soli i frammenti le cui coordinate cascano all'interno del triangolo 2D

8

Rasterizzare triangoli

✓ Il rasterizzatore si occupa anche di
INTERPOLARE GLI ATTRIBUTI
in ogni frammento prodotto

⇒ *Input ulteriore:* un set di attributi per ogni vertice (qualsiasi insieme di vettori, scalari…)

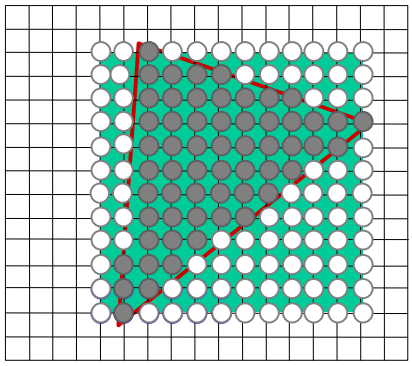
⇒ *Output ulteriore:* un set di valori interpolati, per ogni frammento prodotto

⇒ come sappiamo, l'output è dato da l'iterpolazione degli input avente come pesi le coordinate baricentriche del frammento nel triangolo renderizzato

⇒ Gli attributi interpolati sono passati dal rasterizzatore al processing per frammento

9

Esempio di algoritmo per rastizzazione di triangoli (parallelizzabile!)

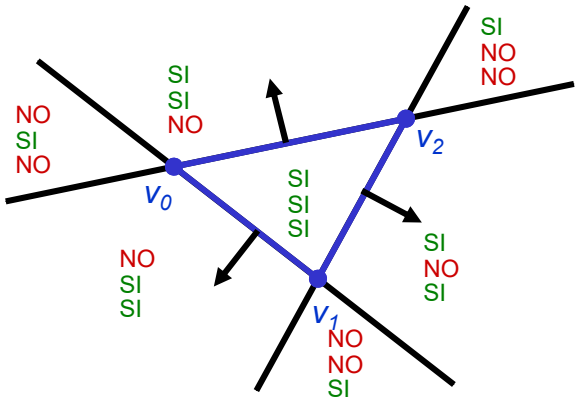


1. Trovo un rettangolo (di coordianate intere) che contiene il triangolo detto "bounding box" o "bounding rectangle"
2. Processo tutti le posizioni interne (nota: questo è parallelizzabile)
3. Per ogni posizione interna: se è dentro al triangolo, allora produco un frammento

10

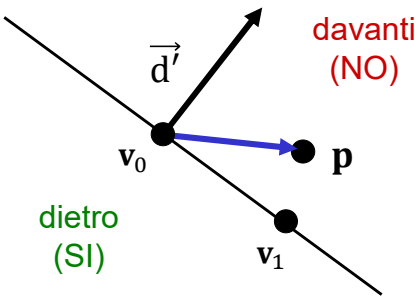
Test di appartenenza ad un triangolo

- ✓ Triangolo 2D = interesezione di 3 semipiani
- ✓ Un punto 2D è interno al triangolo 2D *sse* appartiene a tutti e tre i semipiani



11

Test di appartenenza ad un semipiano (ci serve solo in 2D)



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

Soluzione:

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

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

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

la cosiddetta *Edge function* dell'edge da $\mathbf{v}_0$ a $\mathbf{v}_1$

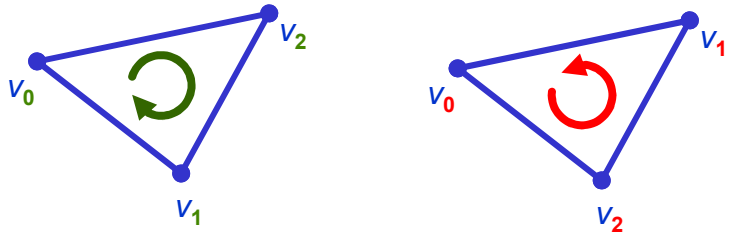
Basta verificare se:

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

12

Domande avanzate

1. Cosa succede a questo algoritmo se l'orientamento del triangolo è in verso opposto? Come puoi riconoscere i frammenti interni al tri in entrambi i casi?

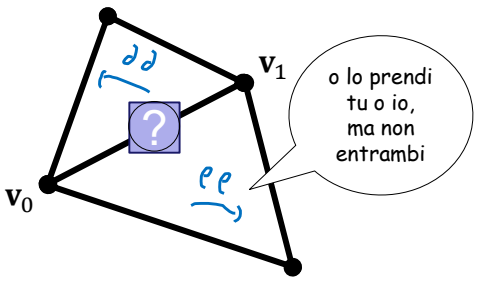


2. Date le coordinate 2D di $v_1\ v_2\ v_3$ sapresti calcolare le coordinate baricentriche del frammento in posizione **p**? (sono necessarie per interpolare gli attributi)

13

Dettaglio: cosa succede se il frammento è "proprio sulla linea"? (cioè se la edge function fa proprio 0)

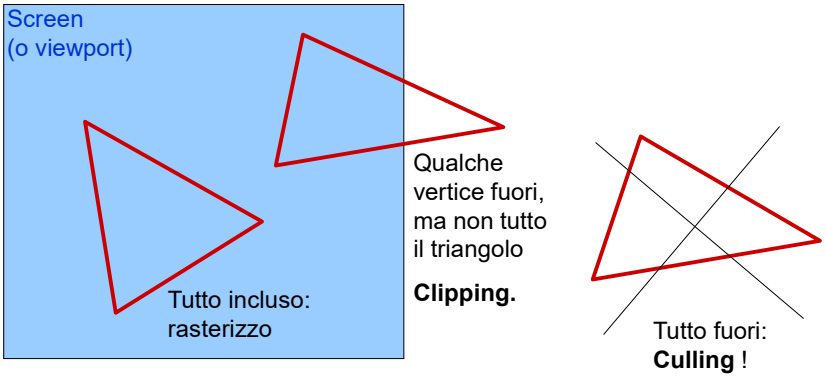
- ✓ Molte soluzioni sono possibili, ma una regola deve sempre valere (è compito dell'implementazione hardware farla valere):
- ✓ un frammento su una linea "a cavallo" fra due triangoli deve essere rasterizzato da *esattamente* uno dei due
  - ⇒ Non entrambi: perché vanno evitati gli "overdraw" inutili
  - ⇒ Non nessuno dei due: perché vanno evitati i gap (buchi) fra i i due



- ✓ Cioè: se il frammento è scartato dell'edge "da $\mathbf{v}_0$ a $\mathbf{v}_1$" allora NON deve essere scartato testando l'edge in senso opposto "da $\mathbf{v}_1$ a $\mathbf{v}_0$", e viceversa

15

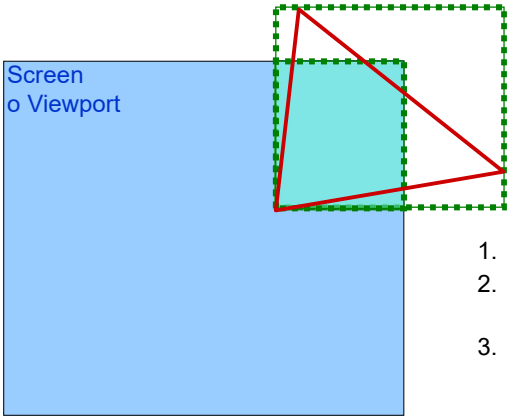
Clipping e culling



16

Clipping: versione più adatta ad una implementazione HW

✓Clipping



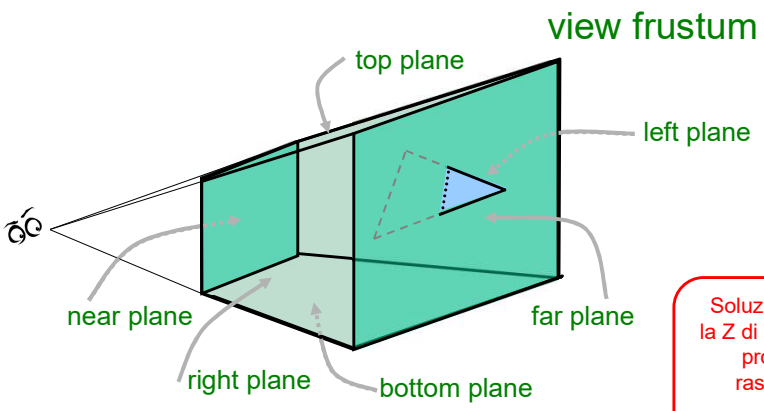
Come si interseca un bounding box con lo schermo (cioe' col rettangolo definito dal ViewPort)?

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

20

Clipping contro far e near clipping plane

✓ E il clipping contro il far e il near plane?



Soluz: si fa testando la Z di ogni frammento prodotto dalla rasteizzazione.

(TEST x FRAMMENTO)

21

Clipping contro far e near clipping plane

✓ Oltre agli attributi che definiamo sui vertici della mesh (come normale, colore, etc), il rasterizzatore interpola sempre un attributo «automatico» ulteriore: la **depth**

⇒ Questa è definito, per ciascuno dei tre vertici agli angoli del triangolo, come la z di quel vertice in spazio schermo

✓ Quindi, ogni frammento prodotto ha la sua depth.

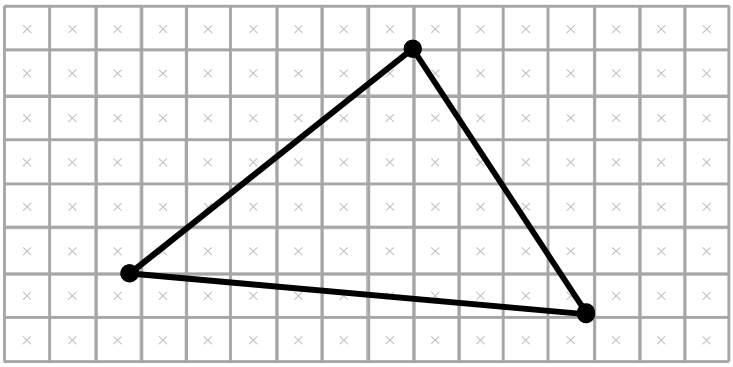
⇒ Interpolata, attraverso le coordinate baricentriche, dalle depth dei tre vertici

✓ I frammenti la cui depth non è inclusa fa 0 e 1 vegono scartati

⇒ È un esempio di test per frammento

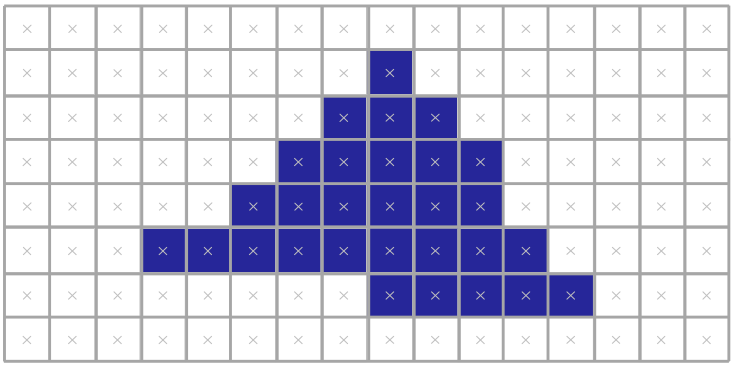
22

Rasterizzare Triangoli



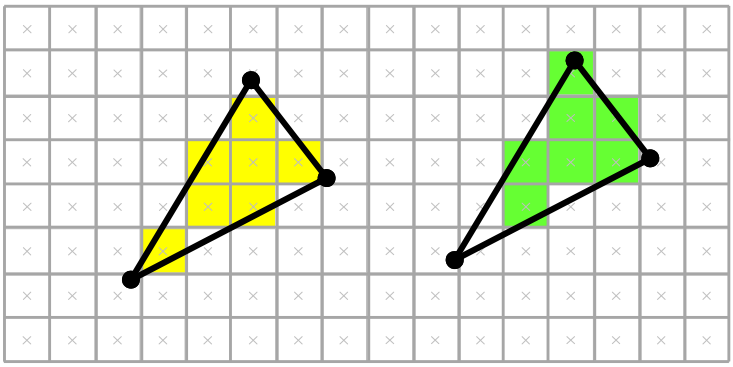
25

Rasterizzare Triangoli



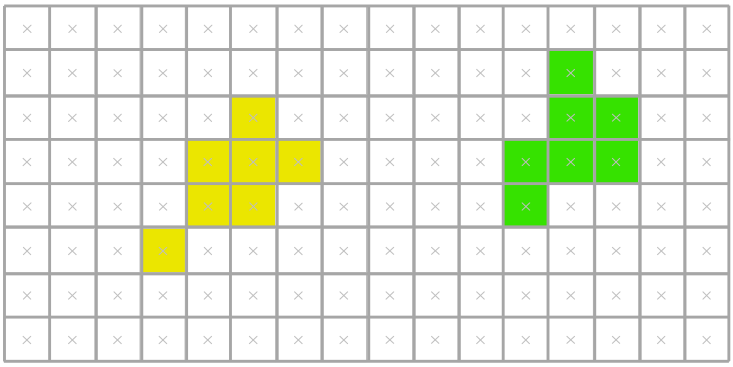
27

Rasterizzare Triangoli



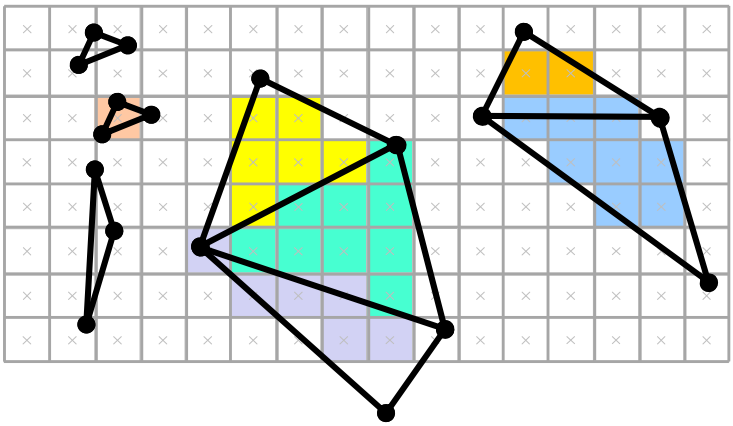
31

Rasterizzare Triangoli



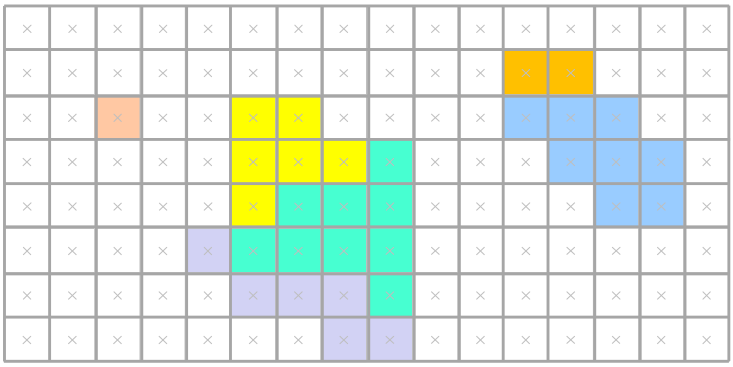
32

Rasterizzare Triangoli



42

Rasterizzare Triangoli



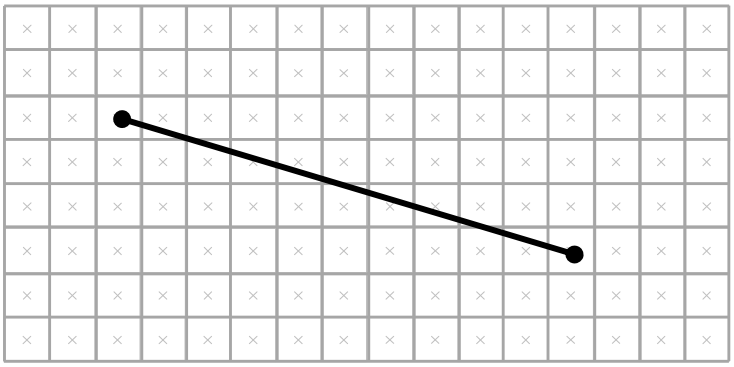
43

Note: rasterizzazione triangoli 1/2

- ✓ Triangoli della stessa forma ma posizioni diverse possono generare insieme di frammenti diversi
- ✓ Un triangolo può generare qualsiasi numero di frammenti
  - ⇒ anche nessuno!
  - ⇒ anche tutti quelli sullo schermo
- ✓ Se un triangolo è parzialmente fuori al viewport, vengono generati solo i frammenti interni al viewport («clipping»)

44

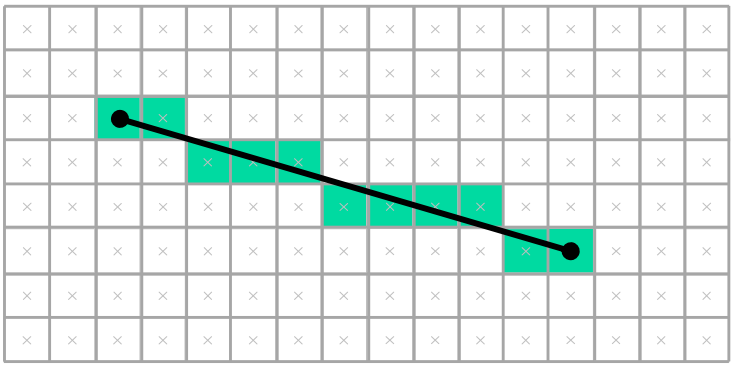
Rasterizzare Linee



Bresenham's line algorithm (1962)

46

Rasterizzare Linee



Bresenham's line algorithm (1962)

47

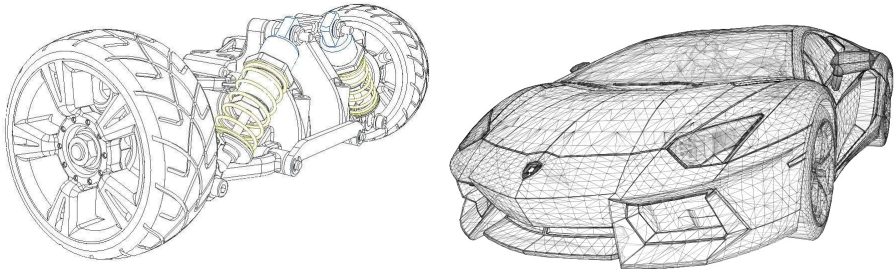
Nota storica: Bresenham's line algorithm

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

48

Rasterizzare Linee

- ✓ La rasterizzazione delle linee è usata nella modalità di rendering detta *wireframe*



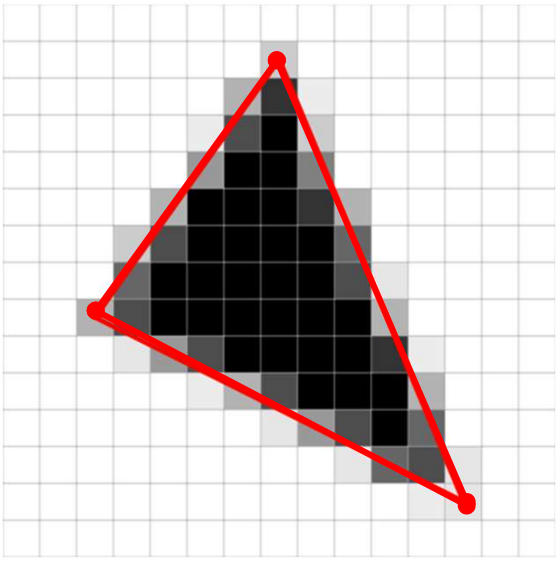
49

Antialiasing

- ✓ Negli esempi visti, il rasterizzatore produce o scarta frammenti "o tutto o niente"
- ✓ L'antialiasing (a volte: AA) è il nome di un insieme di tecniche per nascondere il difetto di scalettatura indotto dai pixel (scalettatura)
- ✓ Esistono degli algoritmi di rasterizzazione che producono frammenti solo parzialmente opachi nei bordi delle primitive, in modo da produrre questi effetti

51

Antialiasing: nel rasterizzare triangoli

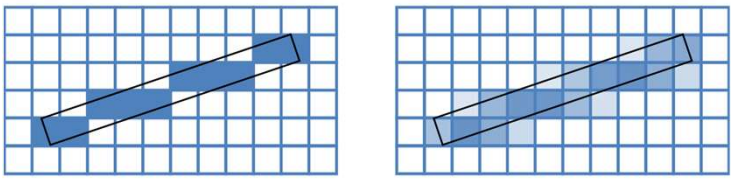


52

Antialiasing: nel rasterizzare segmenti

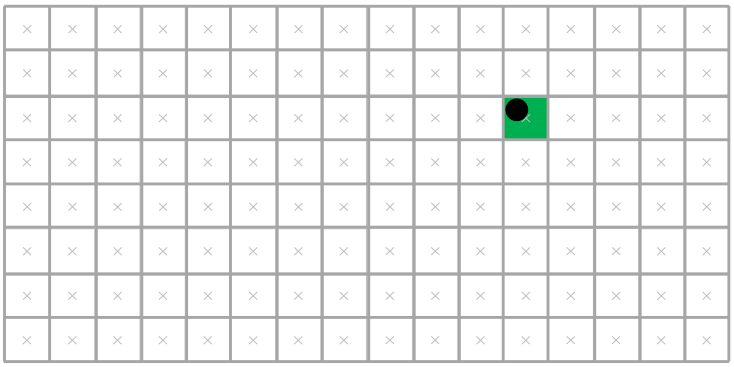
Senza:

Con:



53

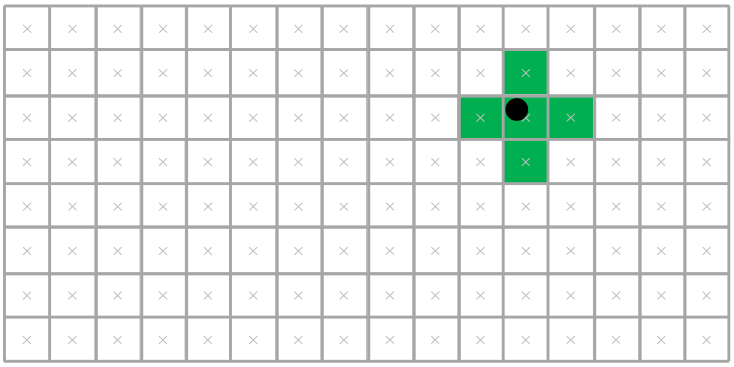
Rasterizzare punti



Point “splat”

54

Rasterizzare punti



Point “splat”
di dimensione maggiore

55