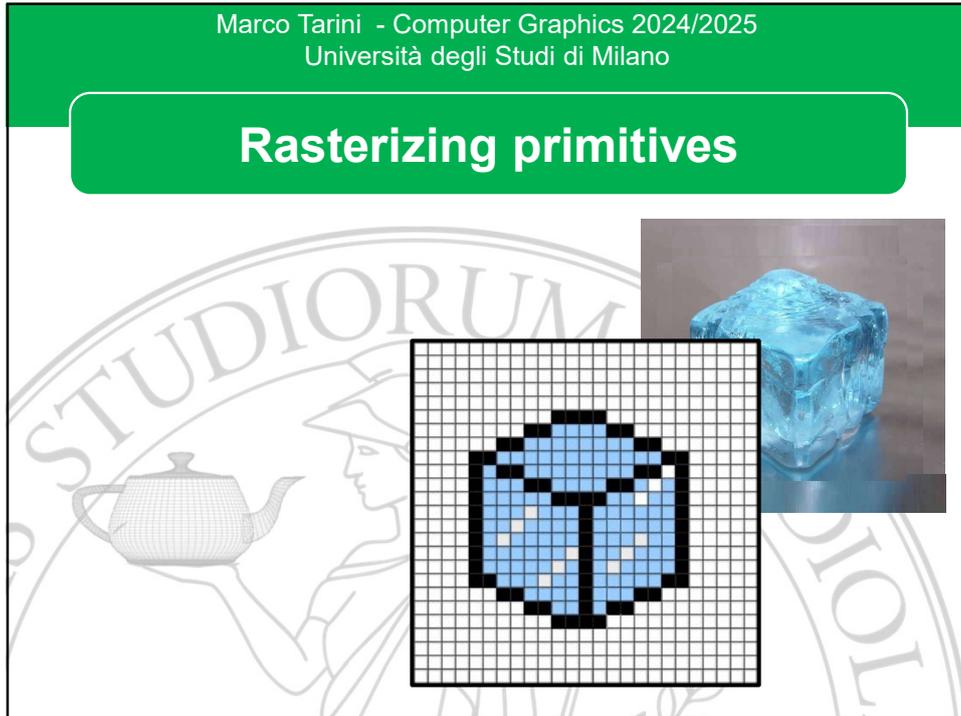


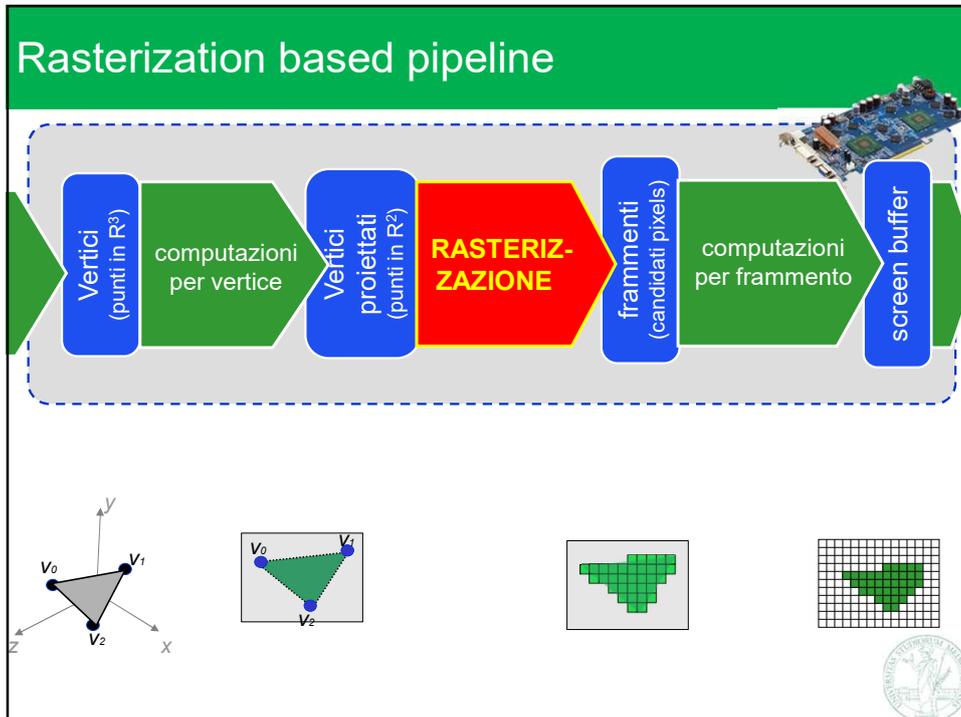
Marco Tarini - Computer Graphics 2024/2025
Università degli Studi di Milano

Rasterizing primitives



1

Rasterization based pipeline



Vertici (punti in R^3)

computazioni per vertice

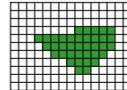
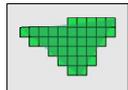
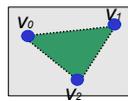
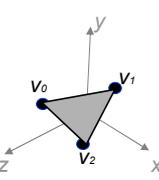
Vertici proiettati (punti in R^2)

RASTERIZZAZIONE

frammenti (candidati pixels)

computazioni per frammento

screen buffer



2

Rasterizzazione di triangoli

- ✓ Individuare i frammenti che compongono una primitiva
 - ⇒ Uno per ogni pixel coperto dalla primitiva
 - ⇒ I frammenti prodotti vengono passati alla fase successiva (che determina il colore RGB del pixel)
- ✓ E' un procedimento del tutto 2D
- ✓ E' estremamente parallelizzabile
 - ⇒ Per sfruttare questo, si adotta un hardware parallelo
 - ⇒ 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 (di qualsiasi primitiva)

- ✓ I vertici sono passati al rasterizzatore in **coordinate clip omogenee**
- ✓ Per prima cosa, vengono convertite in **coordinate schermo cartesiane**
 - ⇒ come sappiamo già
 - ⇒ è uno spazio in cui i pixel hanno coordinate intere
- ✓ 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
- ✓ Le coordinate in spazio schermo dei vertici
 - ⇒ non sono (se non per caso) intere
 - ⇒ non sono necessariamente dentro al viewport
 - ⇒ (tuttavia, devono solo essere prodotti frammenti dentro al viewport)



4

Algoritmo di rasterizzazione di triangoli

- ✓ Vanno individuati i frammenti che compongono i triangoli
 - ⇒ *Input*: tre punti 2D (le posizioni x,y dei vertici in screen space)
 - ⇒ *Output*: i frammenti interni al triangolo
- ✓ Un frammento per ogni pixel dello screen buffer interno 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
- ✓ Nota:
 - i tre vertici di input hanno coordinate (in generale) non intere
 - ⇒ Sono le coordinate x e y risultanti dalle trasformazioni spaziali



7

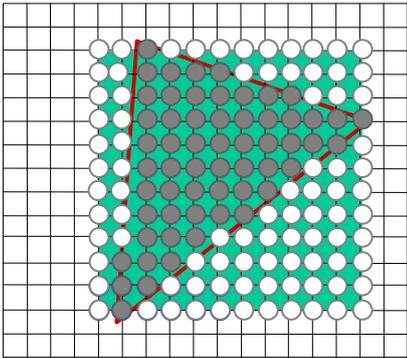
Algoritmo di rasterizzazione di triangoli

- ✓ 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'**interpolazione** degli input avente come pesi le **coordinate baricentriche** del frammento nel triangolo renderizzato
 - ⇒ Gli attributi interpolati sono passati dal rasterizzatore al processing per frammento



8

Esempio di algoritmo per rasterizzazione di triangoli (parallelizzabile!)



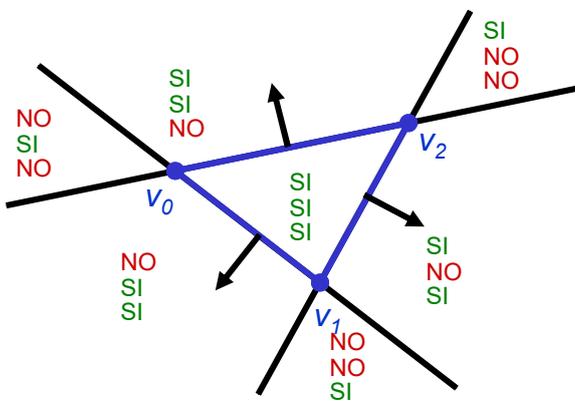
1. Trovo un rettangolo (di coordinate intere) che contiene il triangolo detto "bounding box" o "bounding rectangle"
2. Processo tutti le posizioni interne (nota: questo è parallelizzabile)
3. Per ogni posizione interna: se è dentro al triangolo, allora produco un frammento



9

Test di appartenenza ad un triangolo

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



10

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

davanti
(NO)

dietro
(SI)

la cosiddetta *Edge function* dell'edge da v_0 a v_1

Da quale parte della retta passante per v_0 e v_1 si trova il punto p ?

Soluzione:

$$\vec{d} = (v_1 - v_0) = \begin{pmatrix} d_x \\ d_y \end{pmatrix}$$

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

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

Basta verificare se:
 $(p - v_0) \cdot \vec{d}' < 0$

11

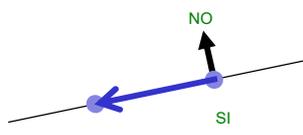
Domande avanzate

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?

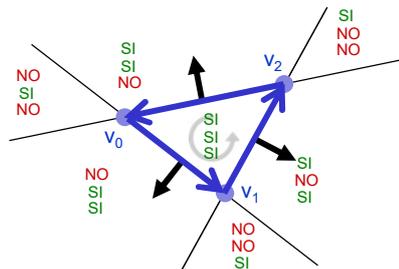
2. Date le coordinate 2D di v_1, v_2, v_3 sapresti calcolare le coordinate baricentriche del frammento in posizione p ? (sono necessarie per interpolare gli attributi)

12

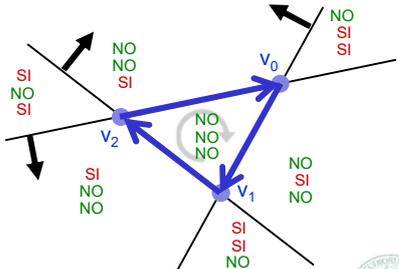
Test di appartenenza di frammento a un triangolo in senso orario e antiorario



SCREEN SPACE:



v0 v1 v2 in senso antiorario

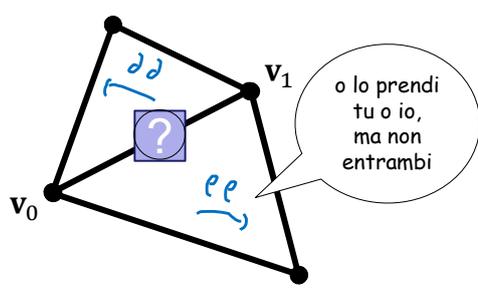


v0 v1 v2 in senso orario

16

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 due



- ✓ Cioè: se il frammento è scartato dell'edge “da v_0 a v_1 ” allora NON deve essere scartato testando l'edge in senso opposto “da v_1 a v_0 ”, e viceversa

19

Clipping e culling

Screen
(o viewport)

Tutto incluso: rasterizzo

Qualche vertice fuori, ma non tutto il triangolo
Clipping.

Tutto fuori:
Culling!
© scartato.

20

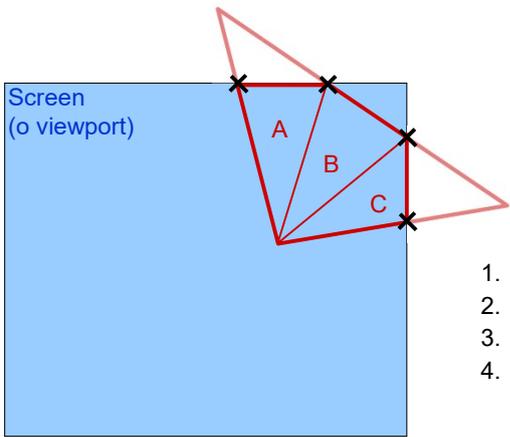
Clipping: la vecchia scuola

Screen
(o viewport)

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

21

Clipping: la vecchia scuola



Screen
(o viewport)

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



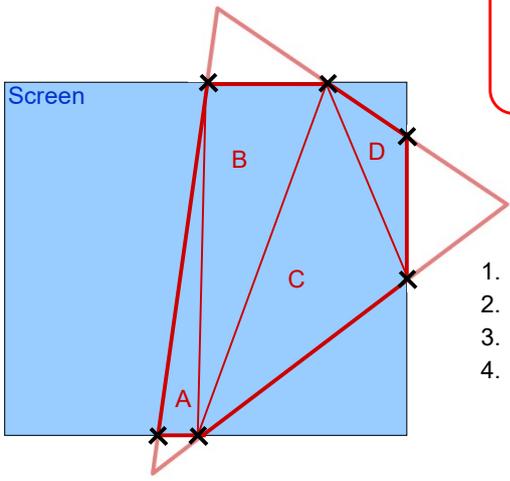
22

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



Screen

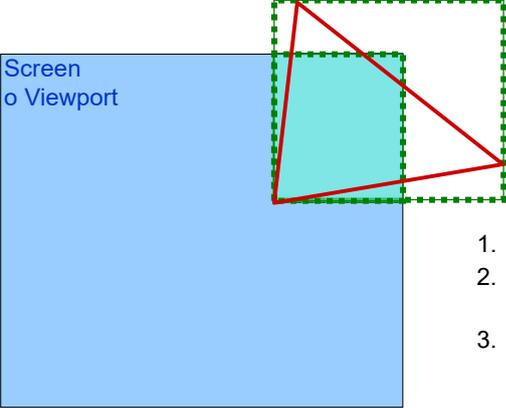
1. Trovo intersezioni
2. Unisco intersezioni
3. Divido poligono in triangoli
4. Rasterizzo ogni triangolo
A,B,C,D



23

Clipping: versione più adatta ad una implementazione HW

✓ Clipping



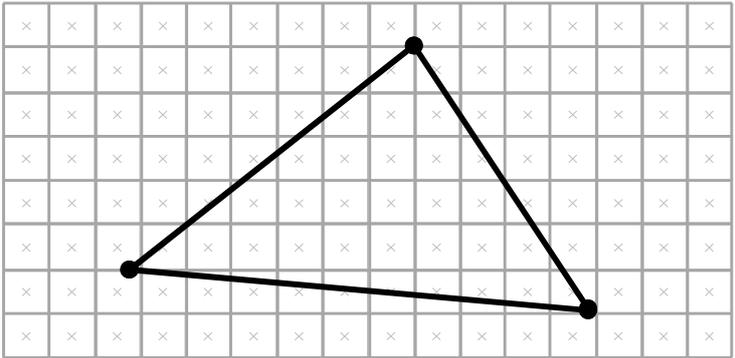
Come si interseca un bounding box con lo schermo (cioe' col rettangolo definito dal ViewPort)?

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



24

Rasterizzare Triangoli



29

Rasterizzare Triangoli

A 15x8 grid of pixels is shown. A triangle is drawn with vertices at (1, 8), (7, 3), and (7, 14). The pixels inside the triangle are shaded blue. The grid is labeled with 'x' marks in each cell.



30

Rasterizzare Triangoli

A 15x8 grid of pixels is shown. A triangle is drawn with vertices at (1, 8), (7, 3), and (7, 14). The pixels inside the triangle are shaded blue. The grid is labeled with 'x' marks in each cell.



31

Rasterizzare Triangoli

The diagram shows a 14x14 grid of pixels, each marked with a small 'x'. Two triangles are overlaid on the grid. The left triangle is yellow and has vertices at approximately (row, col) coordinates (2, 4), (4, 6), and (6, 3). The right triangle is green and has vertices at (2, 11), (4, 13), and (6, 10). The pixels within the bounding boxes of these triangles are filled with their respective colors.



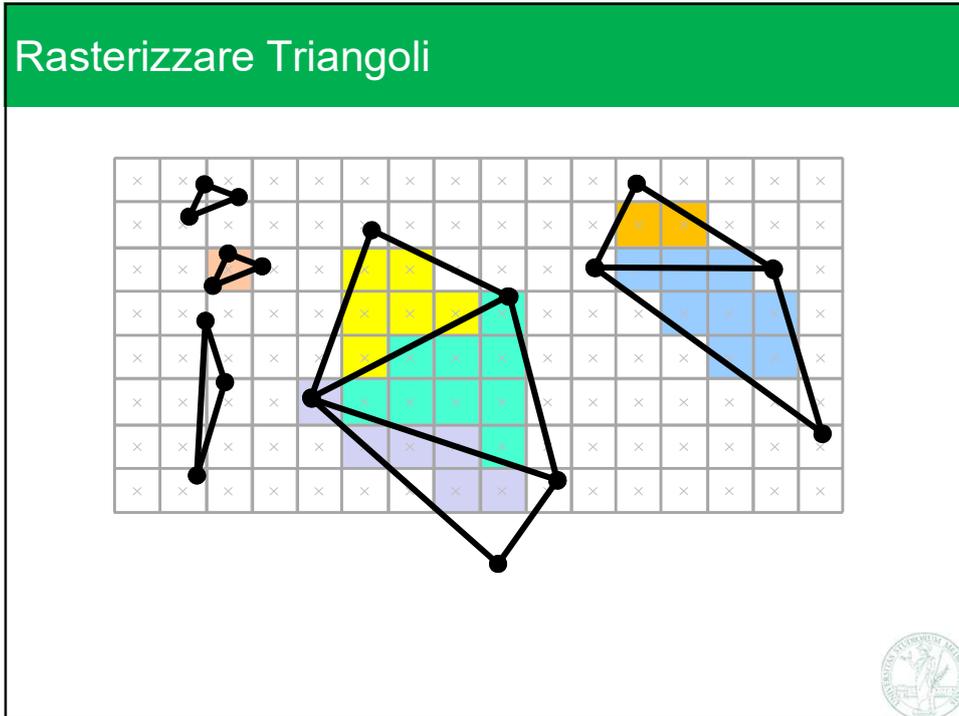
35

Rasterizzare Triangoli

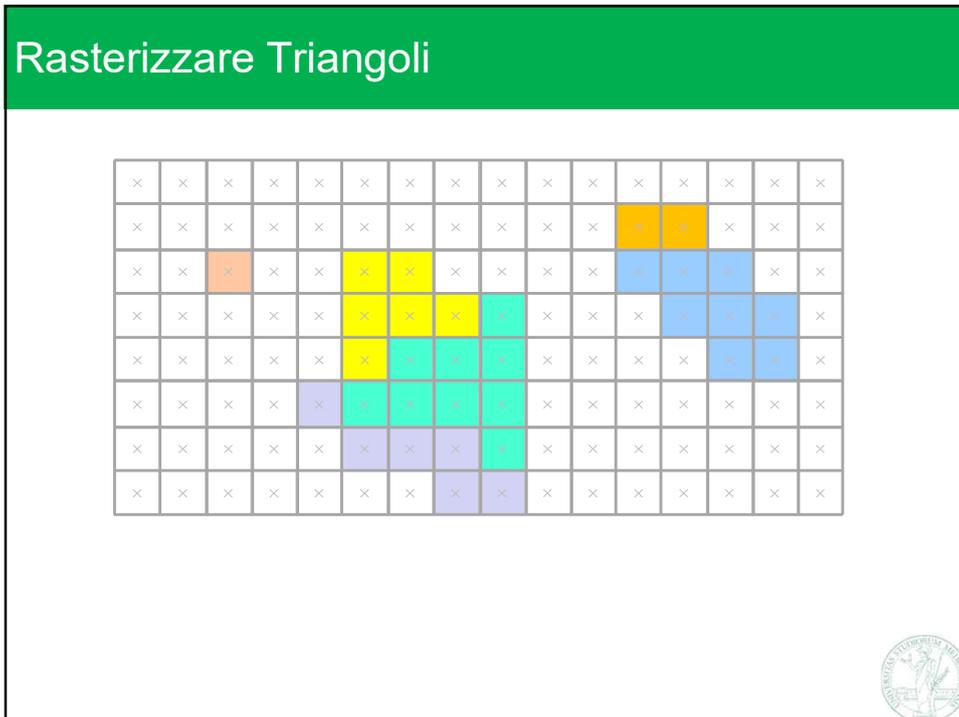
The diagram shows a 14x14 grid of pixels, each marked with a small 'x'. Two triangles are overlaid on the grid. The left triangle is yellow and has vertices at approximately (row, col) coordinates (2, 4), (4, 6), and (6, 3). The right triangle is green and has vertices at (2, 11), (4, 13), and (6, 10). The pixels within the bounding boxes of these triangles are filled with their respective colors.



36



46



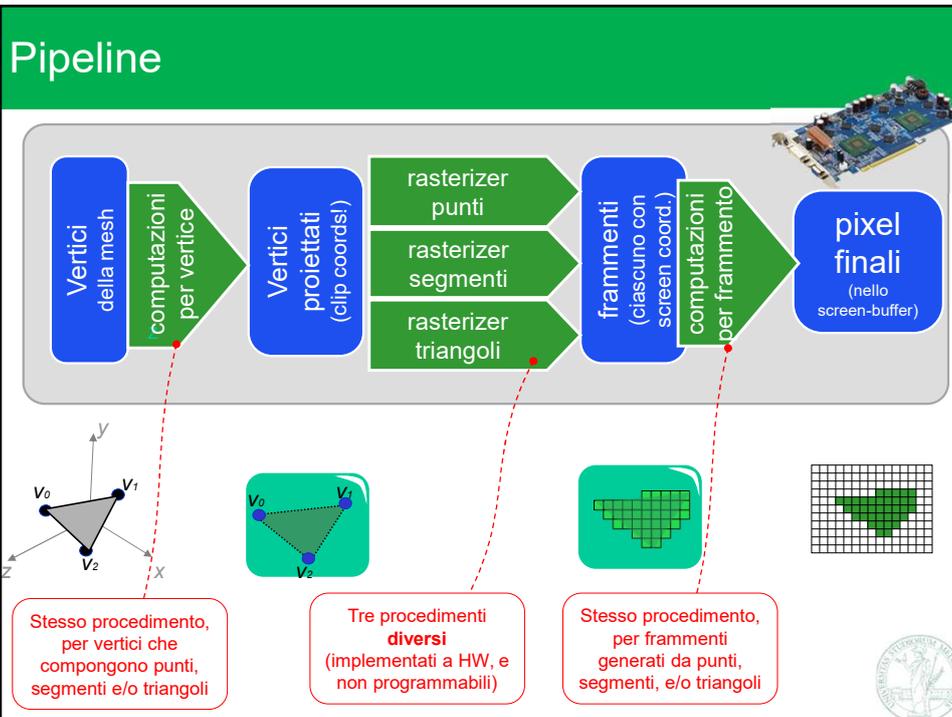
47

Note: rasterizzazione triangoli

- ✓ 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»)



48



50

Rasterizzazione di altre primitive

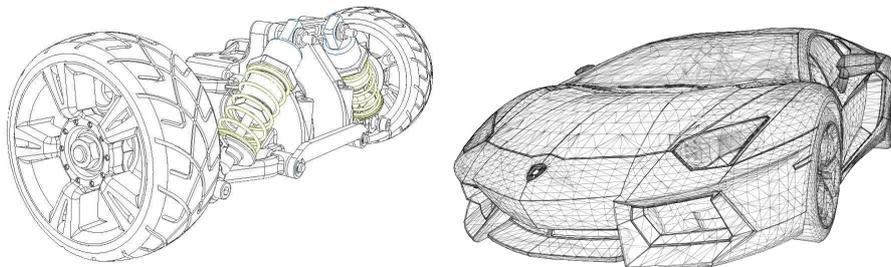
- ✓ Gli hardware GPU attuali sono pesati per rasterizzare solo tre tipi di primitive:
 - ⇒ Triangoli 2D – 3 vertici
 - ⇒ Segmenti 1D («linee») – 2 vertici
 - ⇒ Punti 0D («point splats») – 1 vertice
- ✓ Ovviamente, il primo caso è particolarmente utile ed ottimizzato
- ✓ La fase precente (per vertice) e successiva (per frammento) prescinde dal tipo di primitiva
 - ⇒ I vertici sono trasformati nello stesso modo, indipendentemente da che siano usati per definire punti, linee, o triangoli
 - ⇒ Il frammenti sono trattati nello stesso modo, indipendentemente da che siano stati generati rasterizzando punti, linee, o triangoli



51

Rasterizzare Linee

- ✓ La rasterizzazione delle linee è usata nella modalità di rendering detta *wireframe* (o «hidden lines»)



53

Rasterizzare Linee

Bresenham's line algorithm (1962)



54

Rasterizzare Linee (a spessori diversi)

Bresenham's line algorithm (1962)



55

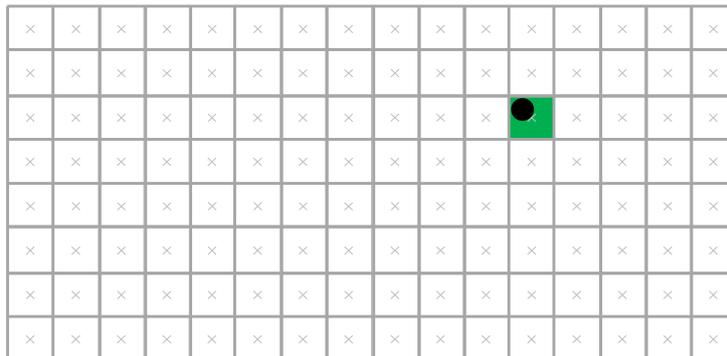
Nota storica: Bresenham's line algorithm

- ✓ Un algoritmo iterativo per rasterizzare linee
- ✓ 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, non nelle GPU)



56

Rasterizzare punti

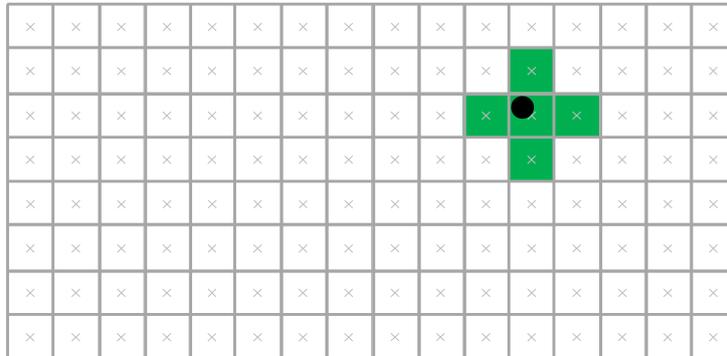


Point "splat"



58

Rasterizzare punti



Point “splat”
di dimensione maggiore



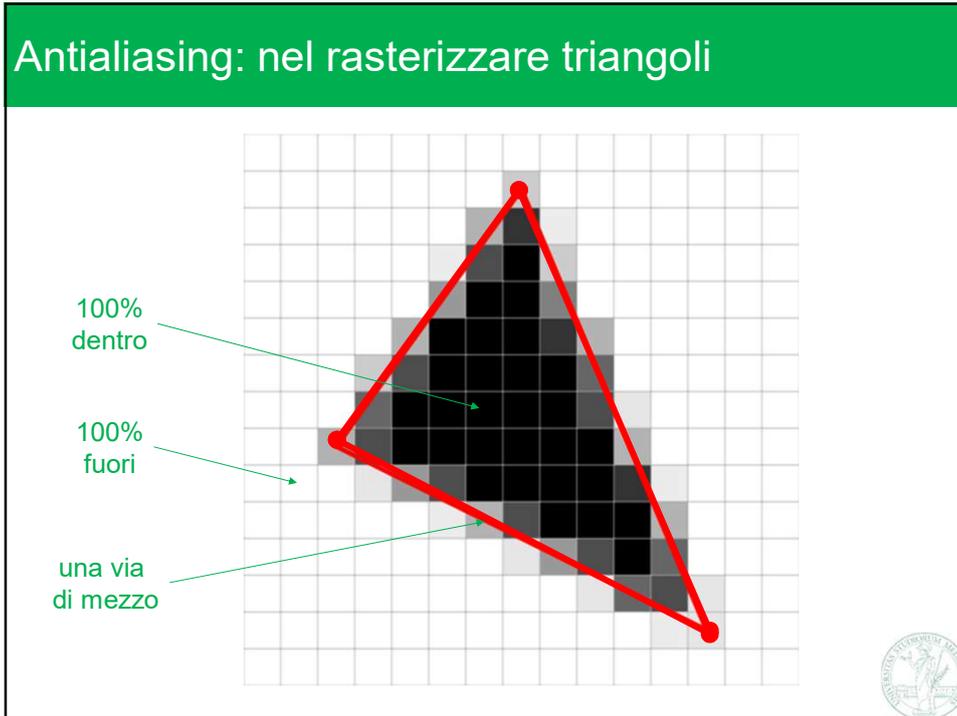
59

Antialiasing

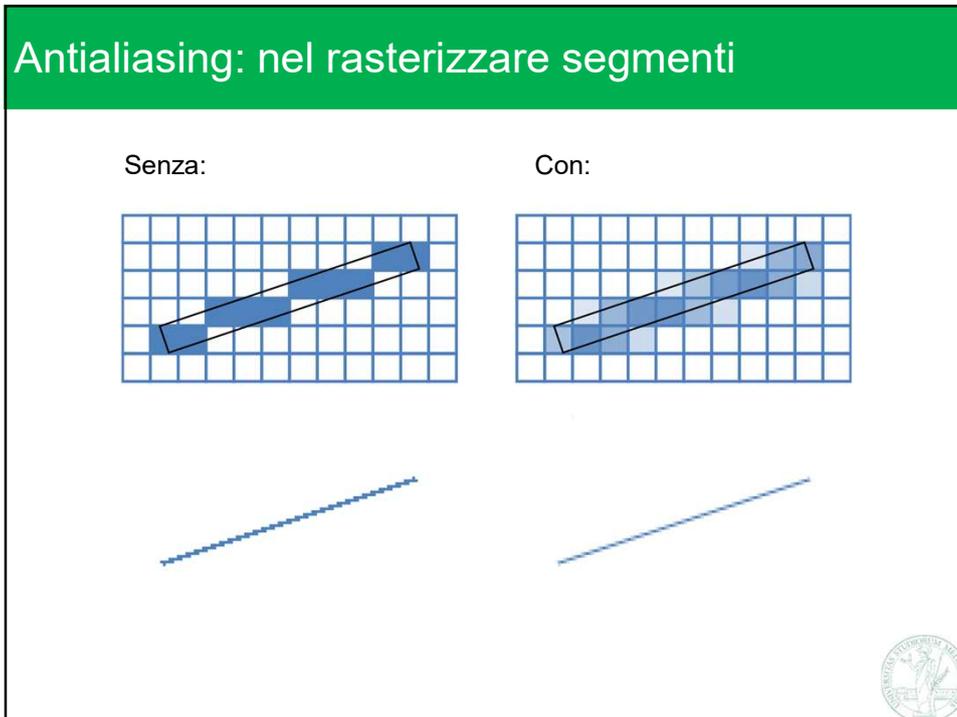
- ✓ Negli esempi visti, il rasterizzatore produce o scarta frammenti “o tutto o niente”
- ✓ L'antialiasing (a volte riferito con la sigla AA) è il nome di un insieme di tecniche per nascondere il difetto di scalettatura indotto dai pixel (aliasing)
- ✓ Esistono degli algoritmi di rasterizzazione che producono frammenti «parziali», cioè semitrasparenti, sui bordi delle primitive



60



61



62