Corso di
Grafica Computazionale
Texturing

Docente:
Massimiliano Corsini

Laurea Specialistica in Ing. Informatica

Facoltà di Ingegneria
Università degli Studi di Siena

Texturing

- Il concetto di ***texturing*** è importante
- Si tratta di “modulare” un qualsiasi attributo del vertice in modo da ottenere l’effetto visivo desiderato
- Attributi modulabili: colore, normali, trasparenza, un parametro del modello di illuminazione, ecc.

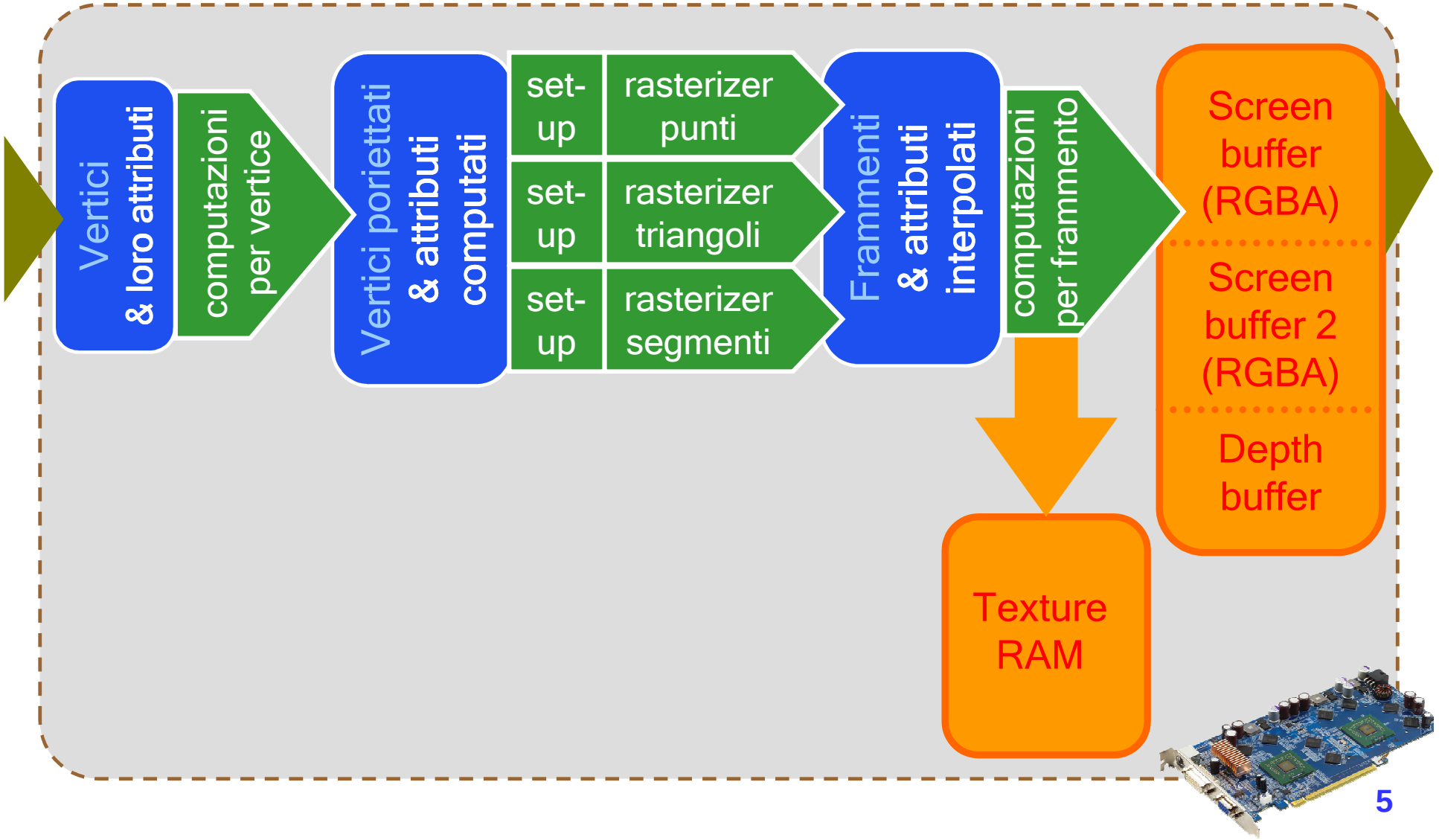
Texture Mapping

- L’attributo più immediato per conferire ulteriore dettaglio alla superfice rispetto a quello che abbiamo visto finora è il colore!
- La modulazione dell’attributo colore prende il nome di ***texture mapping***!
- Vedremo in dettaglio tra poco...

Texture RAM

- Nelle operazioni per frammento si può accedere ad una RAM apposita: la Texture RAM strutturata in un insieme di Textures (“tessiture”)
- Ogni tessitura è un array 1D, 2D o 3D di Texels (campioni di tessitura, prende il nome dai pixels) dello stesso tipo

Texture RAM sulla scheda grafica



Texels

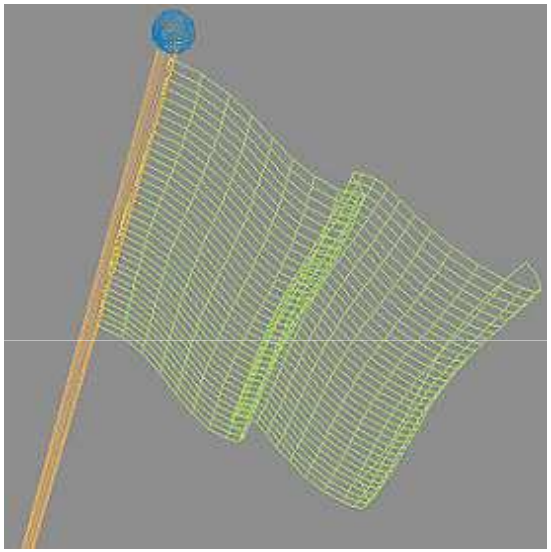
- Sono esempi di texels:
  - Ogni texel un colore (componenti: R-G-B, o R-G-B-A): la tessitura è una “color-map”
  - Ogni texel una componente alpha: la tessitura è una “alpha-map”
  - Ogni texel una normale (componenti: X-Y-Z): la tessitura è una “normal-map” o “bump-map”
  - Ogni texel contiene un valore di specularità: la tessitura è una “shininess-map”

Texture Mapping: Storia

Ed Catmull

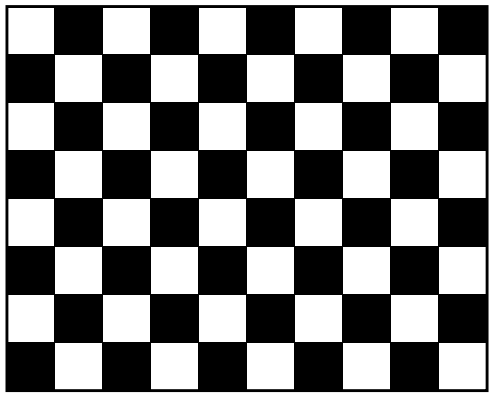
- 1974 introdotto da Ed Catmull
  - nella sua Phd Thesis
- Solo nel 1992 (!) si ha texture mapping in hardware
  - Silicon Graphics RealityEngine
- Dal 92 a oggi ha avuto aumento rapidissimo della diffusione
  - strada intrapresa soprattutto dall'hardware grafico
- Oggi è una delle più fondamentali tecniche di rendering

Mappare Immagini sulla Geometria



geometria 3D
(mesh di quadrilateri)

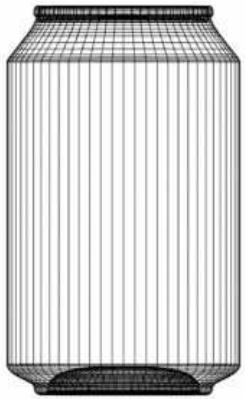
+



RGB texture 2D
(color-map)

=

Mappare Immagini sulla Geometria

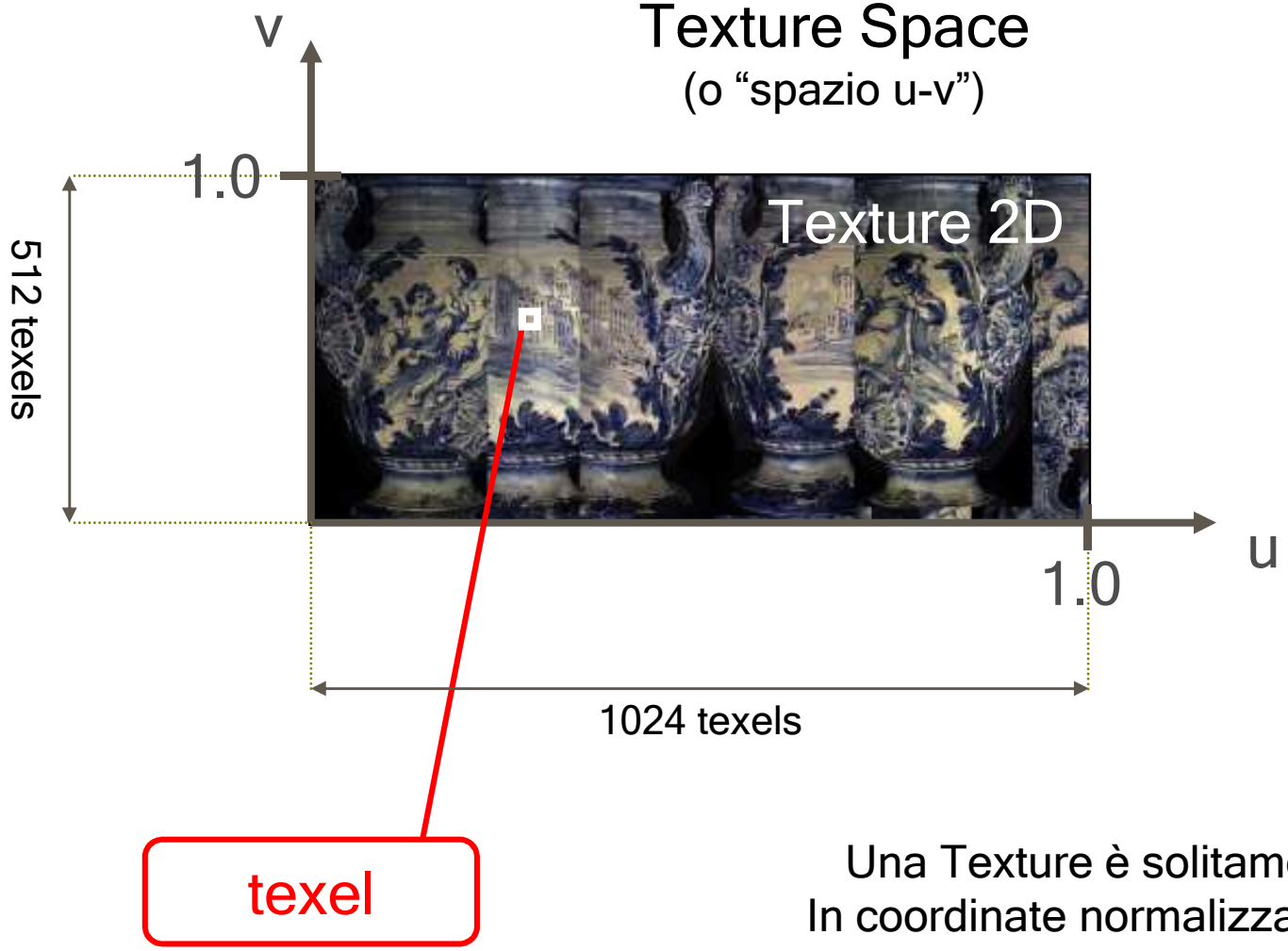


Mappare Immagini sulla Geometria

+

=

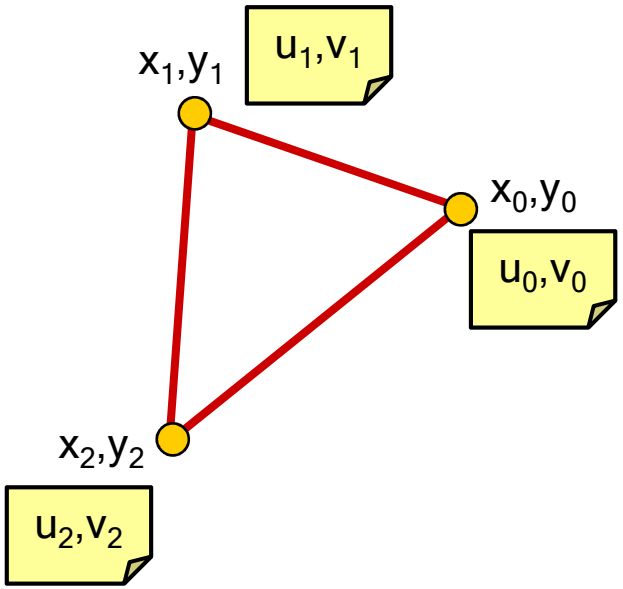
Spazio Texture (“spazio u-v”)



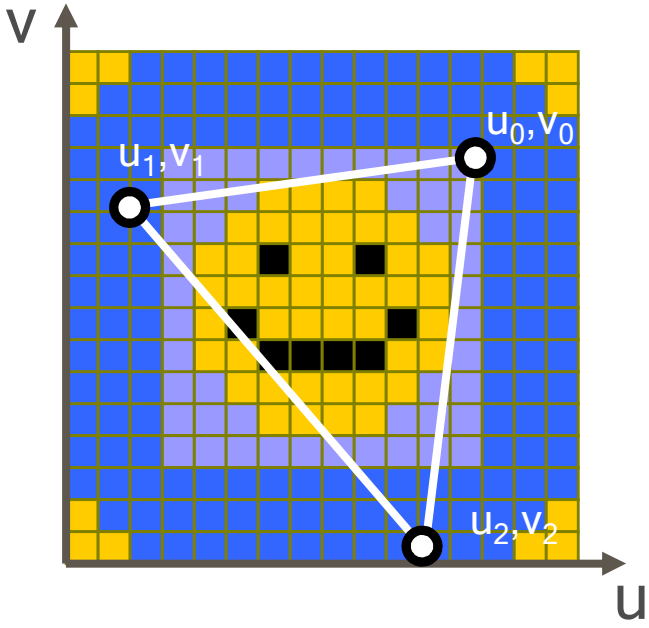
Una Texture è solitamente definita
In coordinate normalizzate [0,1] x [0,1]
nello spazio parametrico della texture

Texture Mapping

- Ad ogni vertice (di ogni triangolo) assegno le sue coordinate *u,v* nello spazio tessitura



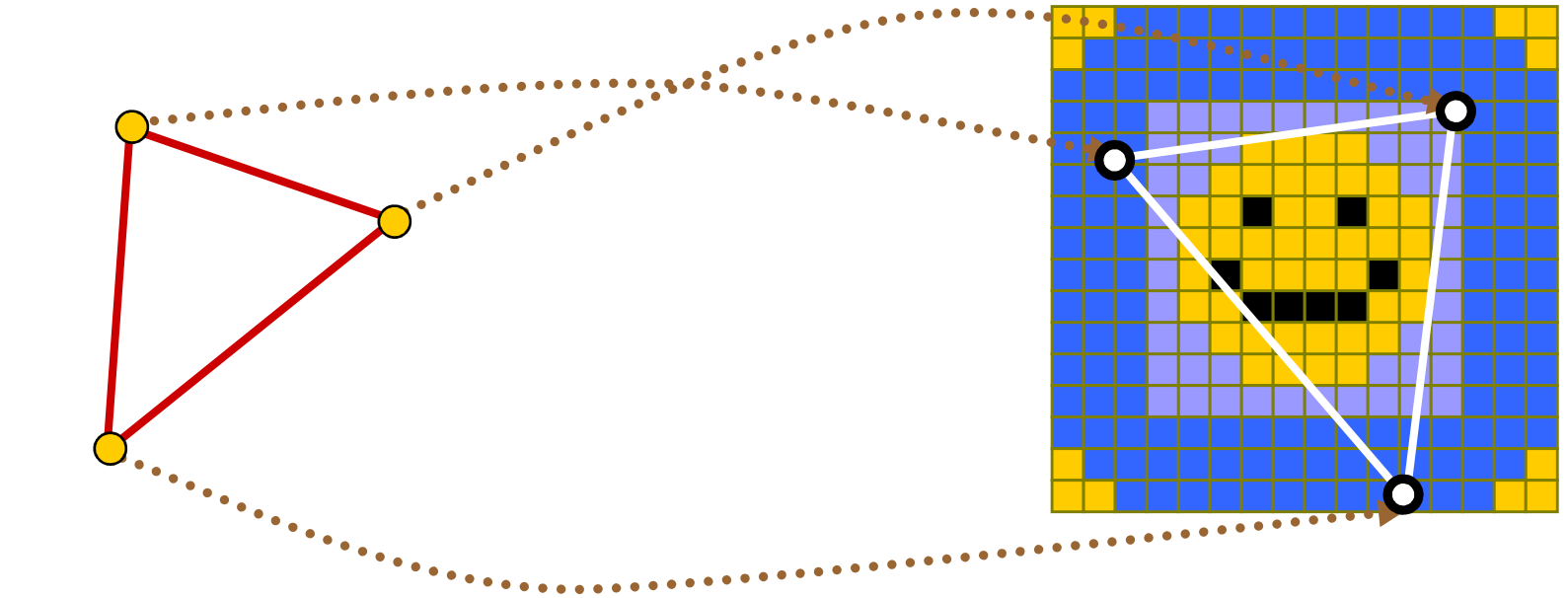
Screen Space



Texture Space

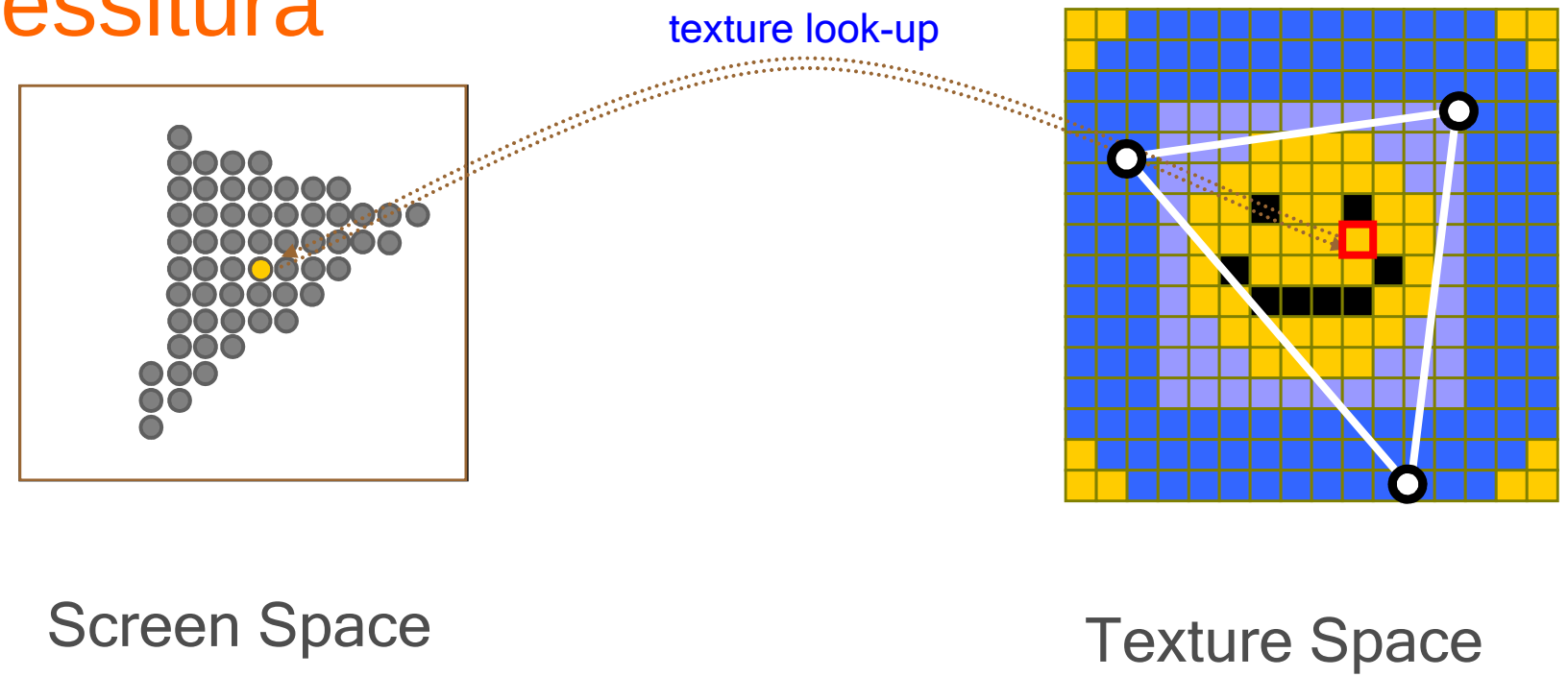
Texture Mapping

- Così in pratica definisco un mapping fra il triangolo e un triangolo di tessitura

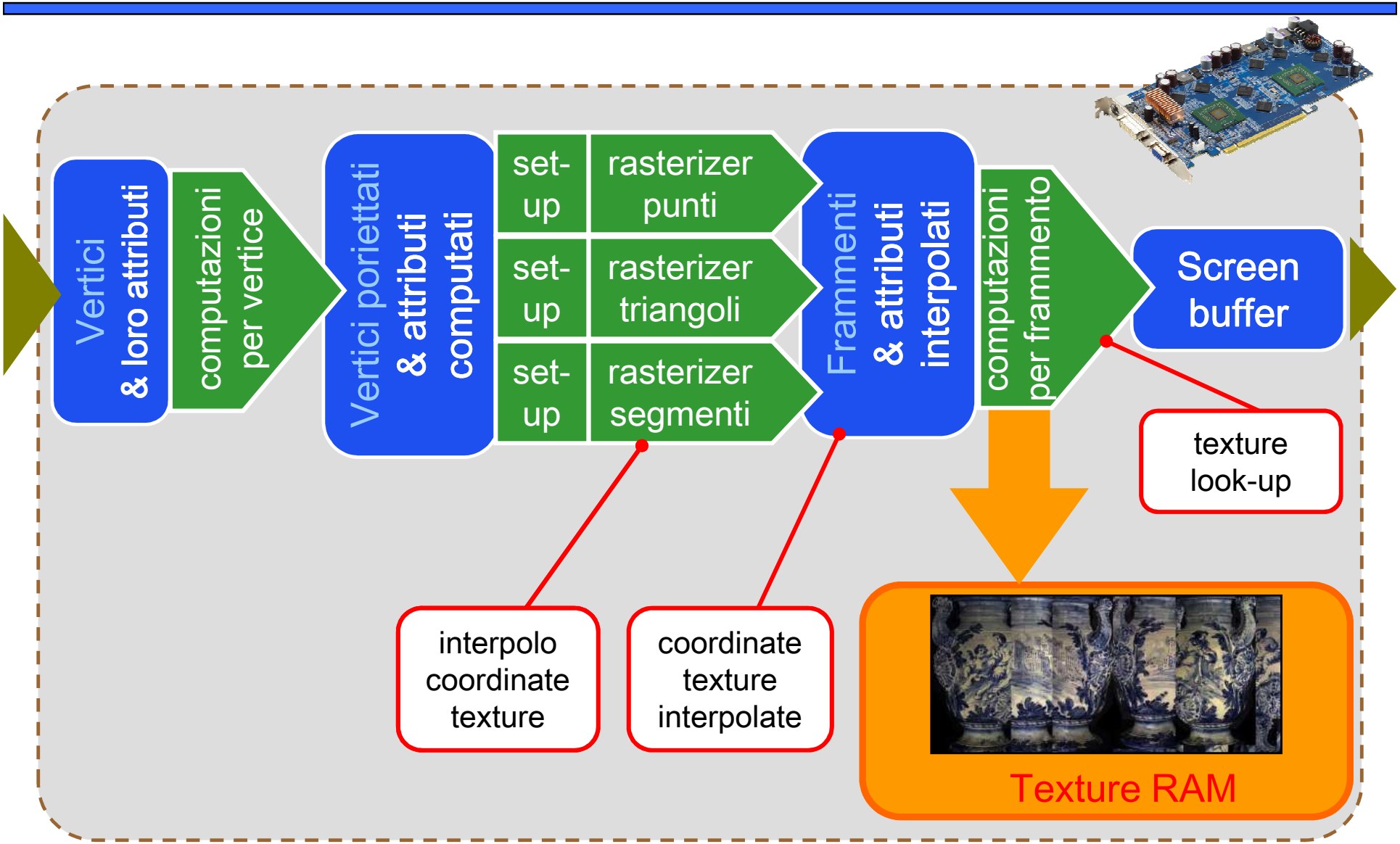


Texture Mapping

- Ogni vertice (di ogni triangolo) ha le sue coordinate *u*,*v* nello spazio tessitura

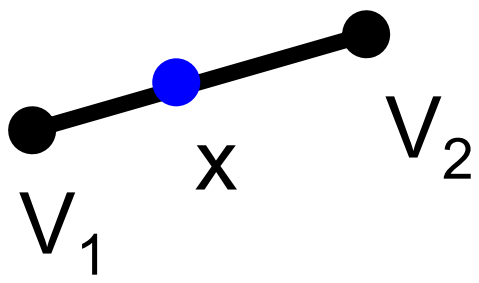


Texture Mapping



Interpolazione degli Attributi

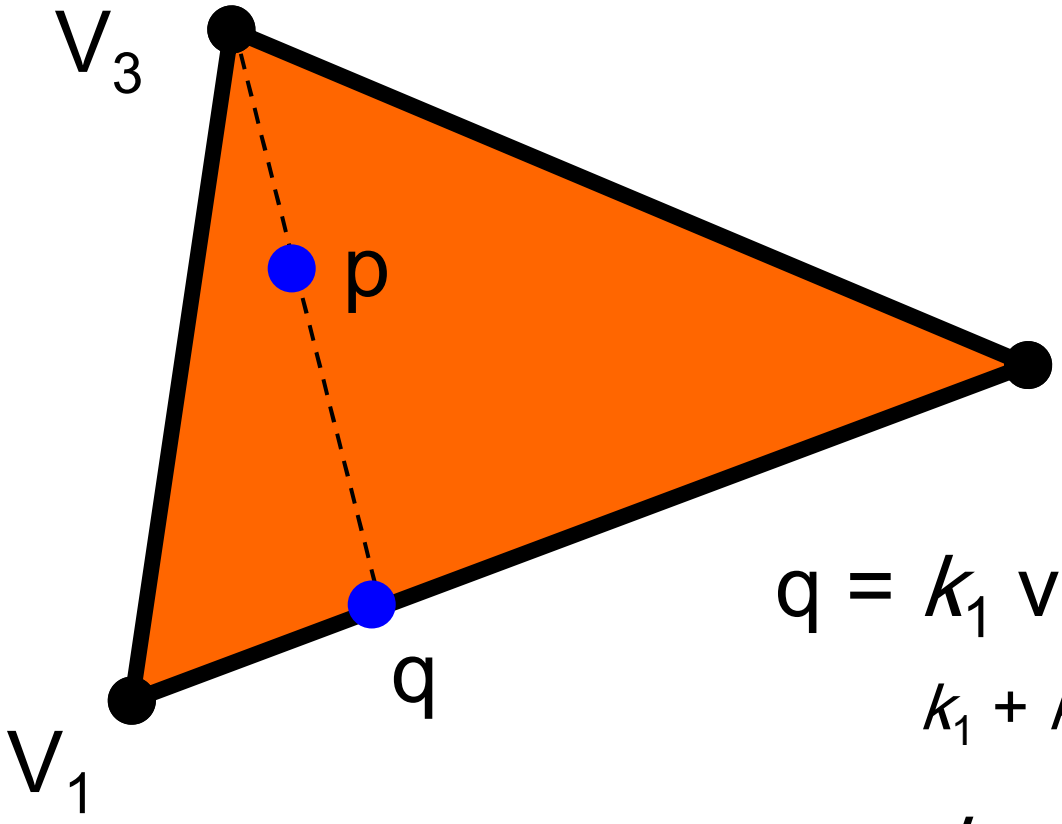
- Di norma si utilizzano le coordinate baricentriche...
- Ma cosa sono le coordinate baricentriche?
- Ricordiamo che un segmento si può scrivere come la combinazione lineare di due punti:



$$x = a\,v_1 + b\,v_2$$

a e b scalari positivi
con $a + b = 1$
(quindi $0 \le a \le 1$ e $0 \le b \le 1$)

Coordinate Baricentriche



$$q = k_1 v_1 + k_2 v_2$$

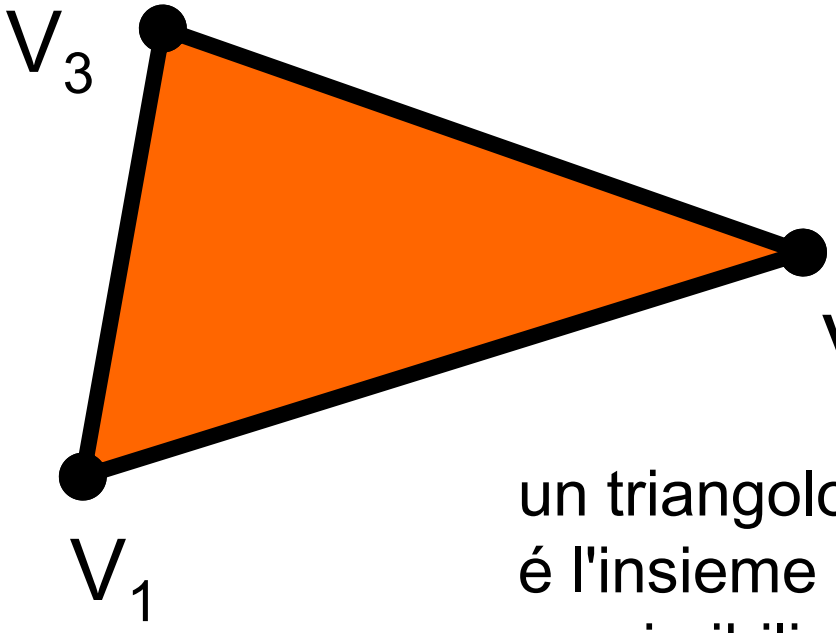
$$k_1 + k_2 = 1 \quad k_1, k_2 > 0$$

$$p = h_1 v_3 + h_2 q$$

$$h_1 + h_2 = 1 \quad h_1, h_2 > 0$$

esercizio: sostituiamo e...

Coordinate Baricentriche



un triangolo di vertici $v_1\ v_2\ v_3$
é l'insieme di tutti i punti x
esprimibili come

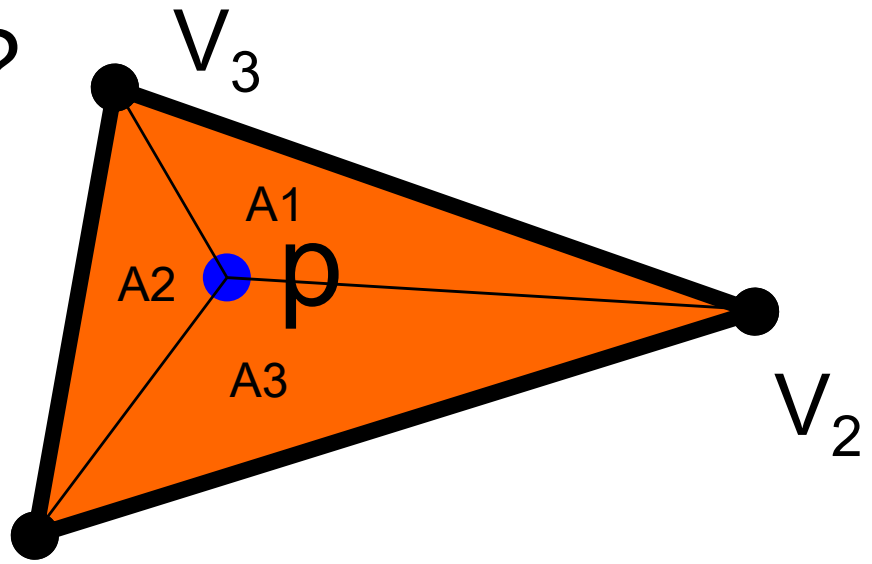
$$x = a_1\ v_1 + a_2\ v_2 + a_3\ v_3$$

a_1, , a_2 , a_3 scalari positivi

$$a_1 + a_2 + a_3 = 1$$

Coordinate Baricentriche

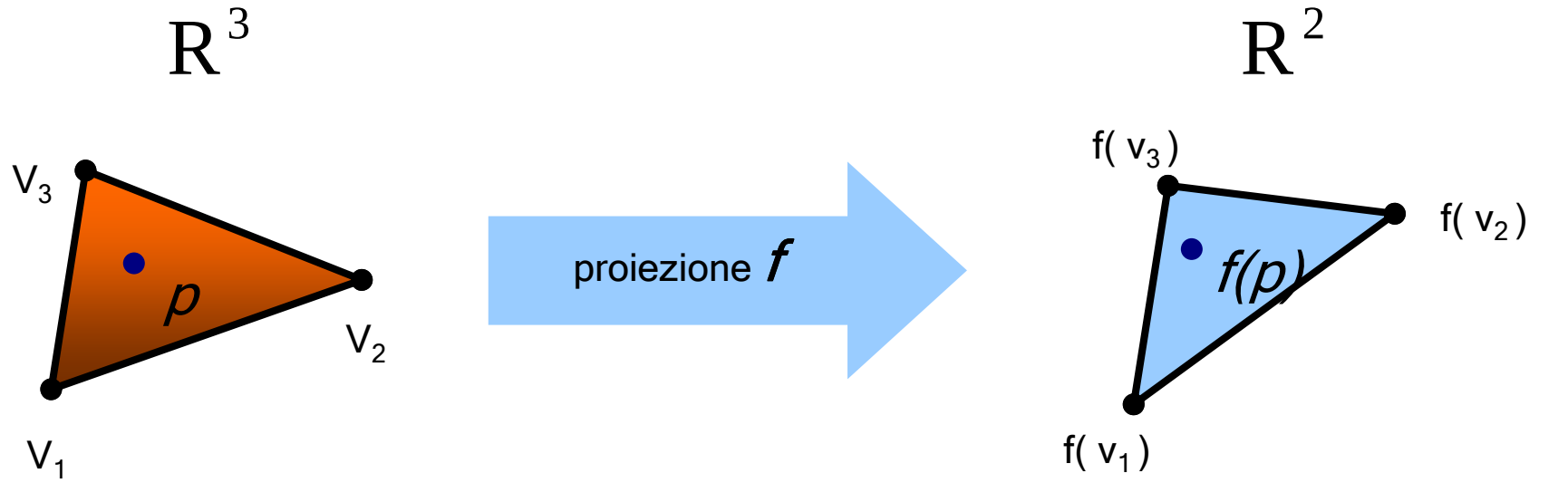
- Quali sono le coord. baricentriche di un punto p?



$$p = a_1 v_1 + a_2 v_2 + a_3 v_3$$

$$a_1 = \frac{A_1}{A_{tot}} \qquad a_2 = \frac{A_2}{A_{tot}} \qquad a_3 = \frac{A_3}{A_{tot}} \qquad A_{tot} = A_1 + A_2 + A_3$$

Interpolazione degli attributi



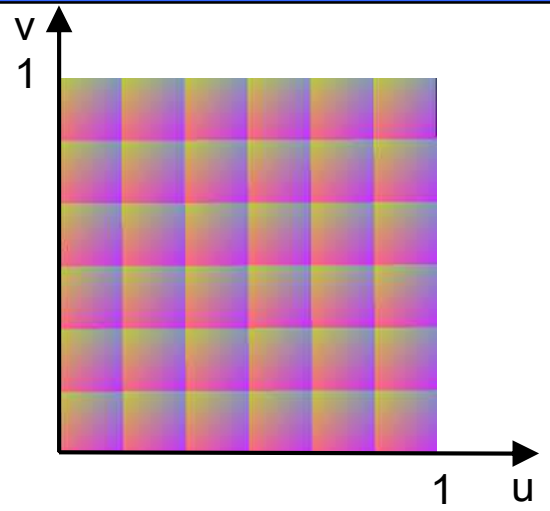
p ha coordinate baricentriche a,b,c nel triangolo $v_1\ v_2\ v_3$

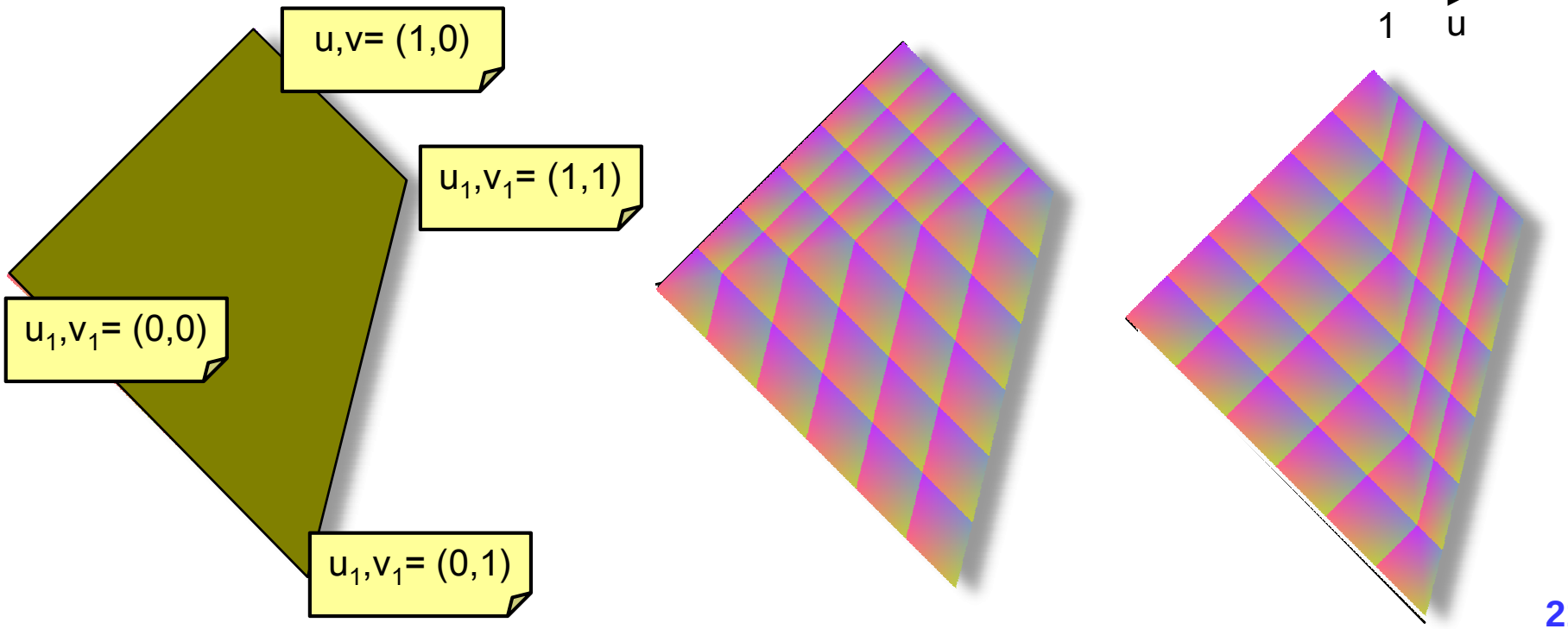
f(p) ha coordinate baricentriche a,b,c nel triangolo $f(v_1)\ f(v_2)\ f(v_3)$

- L'interpolazione secondo le coordinate baricentriche non funziona quando interpoliamo le coordinate texture a causa della proiezione prospettica...
- Funziona però per gli altri attributi (es. normali)

Interpolazione delle Coordinate Texture

- Esempio:

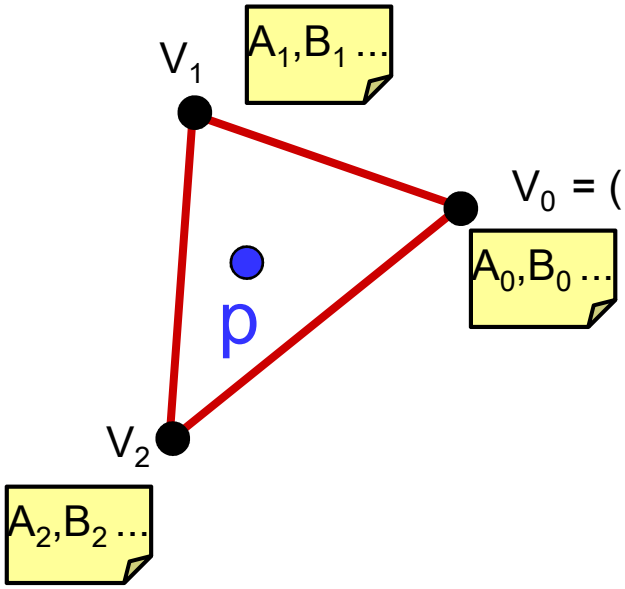




Correzione Prospettica

- p ha coordinate baricentriche $c_0\ c_1\ c_2$

$$p = c_0 v_0 + c_1 v_1 + c_2 v_2$$



attributi di p:
(senza considerare
la correzione prospettica)

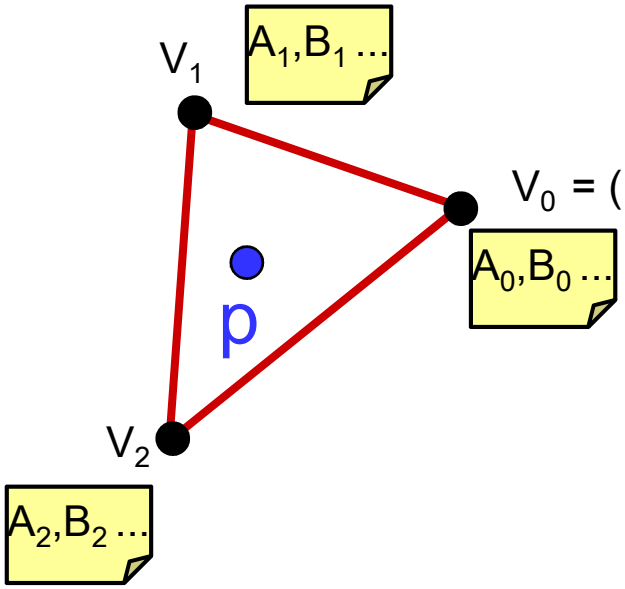
$$A_p = c_0 A_0 + c_1 A_1 + c_2 A_2$$

$$B_p = c_0 B_0 + c_1 B_1 + c_2 B_2$$

Correzione Prospettica

- p ha coordinate baricentriche $c_0\ c_1\ c_2$

$$p = c_0 v_0 + c_1 v_1 + c_2 v_2$$

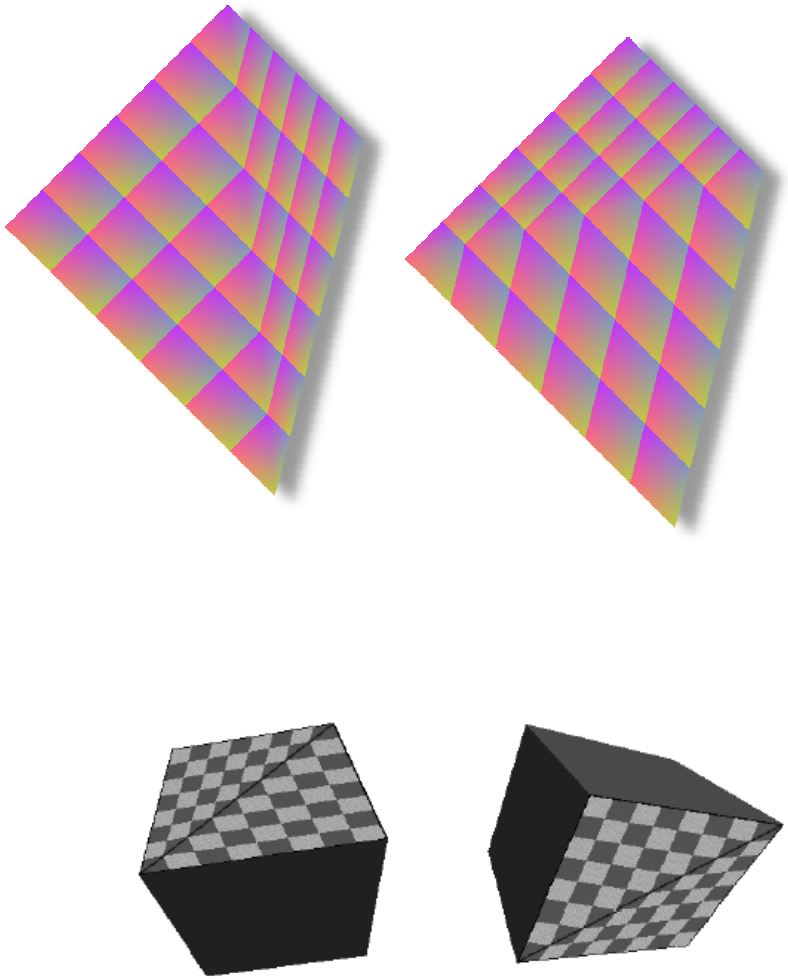


attributi di p:
(con la correzione prospettica)

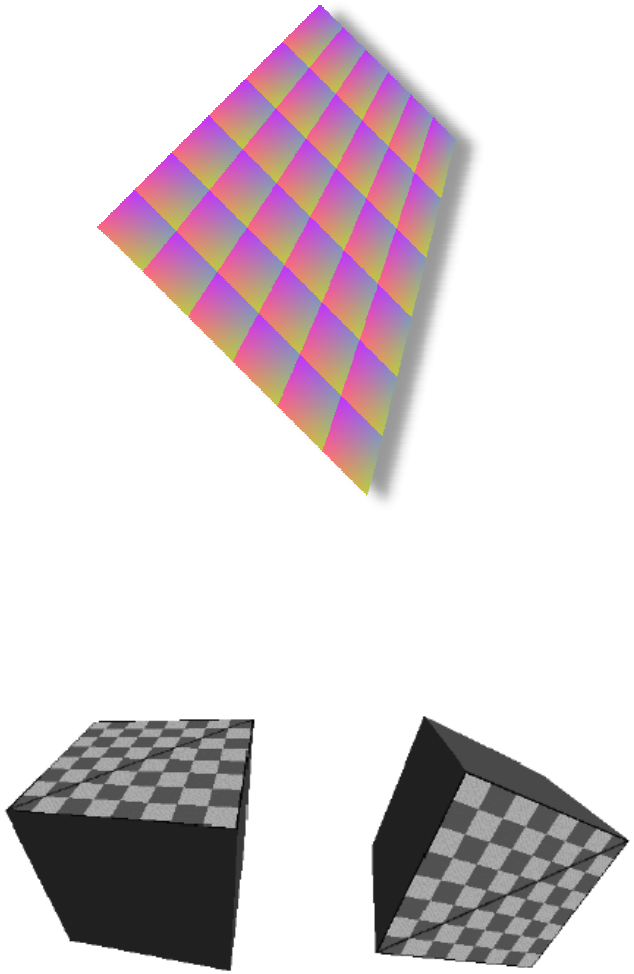
$$A_p = \frac{c_0 \frac{A_0}{w_0} + c_1 \frac{A_1}{w_1} + c_2 \frac{A_2}{w_2}}{c_0 \frac{1}{w_0} + c_1 \frac{1}{w_1} + c_2 \frac{1}{w_2}}$$

Correzione Prospettica

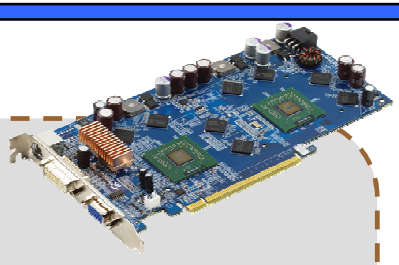
- Senza



- Con



La tessitura va copiata sulla texture RAM (!!)



La tessitura va copiata sulla texture RAM (!!)

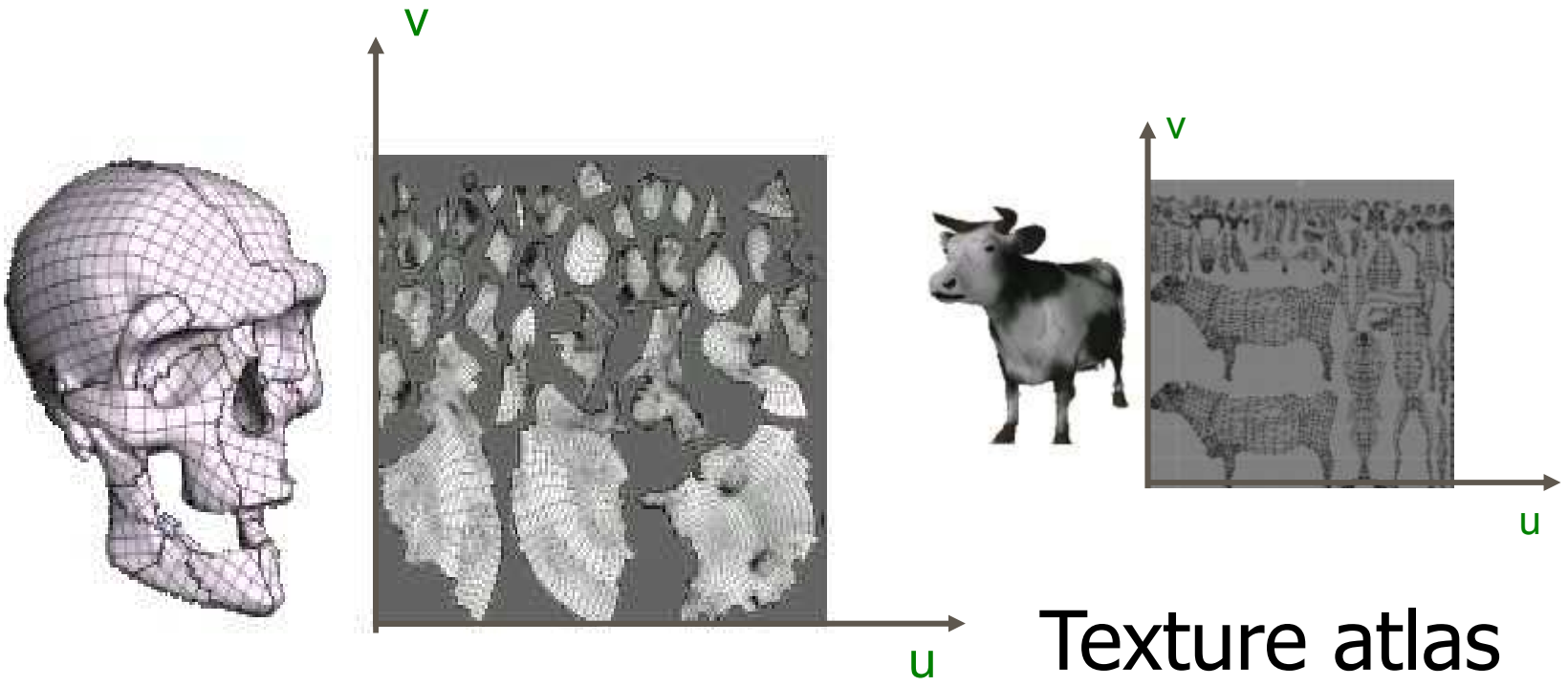
- Dalla memoria principale si deve copiare la texture nella *Texture RAM* (on board dell'HW grafico)
- Anche il passaggio inverso è possibile
- Entrambe le operazioni sono piuttosto lente si deve quindi tenere conto di come gestire le textures nella progettazione dell'applicazione

Come assegnare le coordinate texture?

- Soluzioni:
  - Calcolare le coordinate textures on-the-fly durante il rendering…
  - Precomputarle (e salvarle insieme alla mesh)
  - Spesso le assegna il modellatore...
- Non esiste una soluzione ideale, dipende dall'applicazione che stiamo progettando

UV Mapping: problema difficile

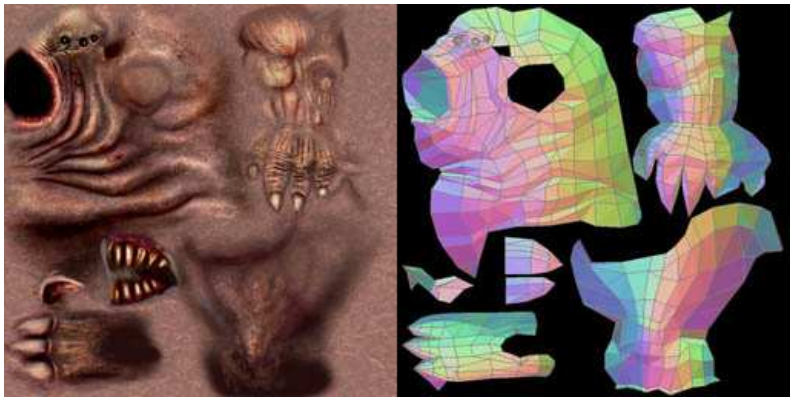
- Assegnare una coppia di coordinate textures ad ogni vertice della mesh in preprocessing



Texture atlas

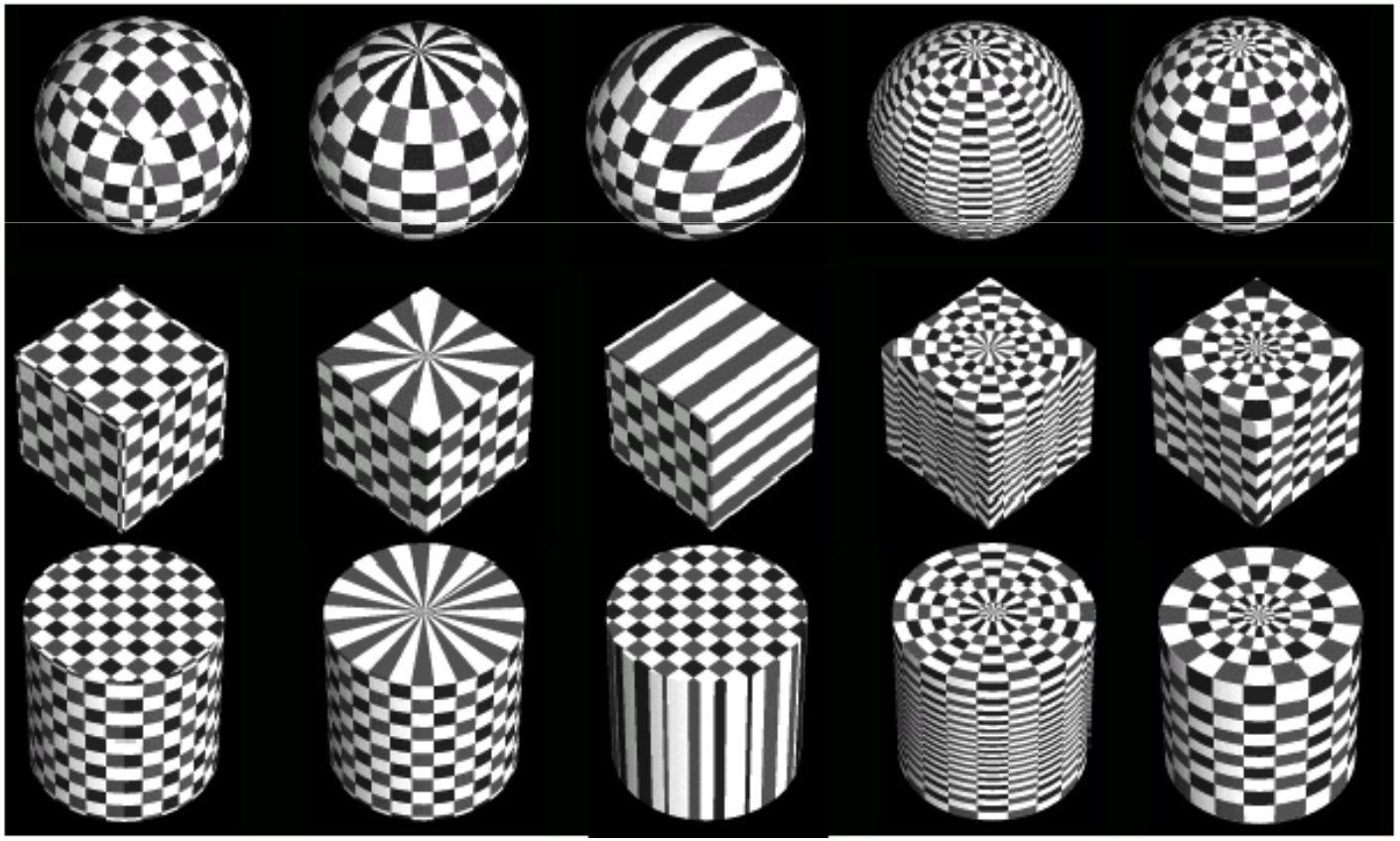
UV Mapping: problema difficile

fatto a mano,
o automatizzato



Creazione automatica delle coordinate texture

- Si utilizza un funzione di proiezione da (x,y,z) a (u,v) in coordinate oggetto o vista



Creazione automatica delle coordinate texture

- Proiezione planare (lungo asse x)

$$(x, y, z) \mapsto (u, v) : \begin{cases} u = z \\ v = y \end{cases}$$

- Proiezione cilindrica

$$(x, y, z) \mapsto (u, v) : \begin{cases} u = \text{atan2}(z, x) \\ v = y \end{cases}$$

Nota: le coordinate (u,v) devono essere normalizzate nell'intervallo [0, 1]

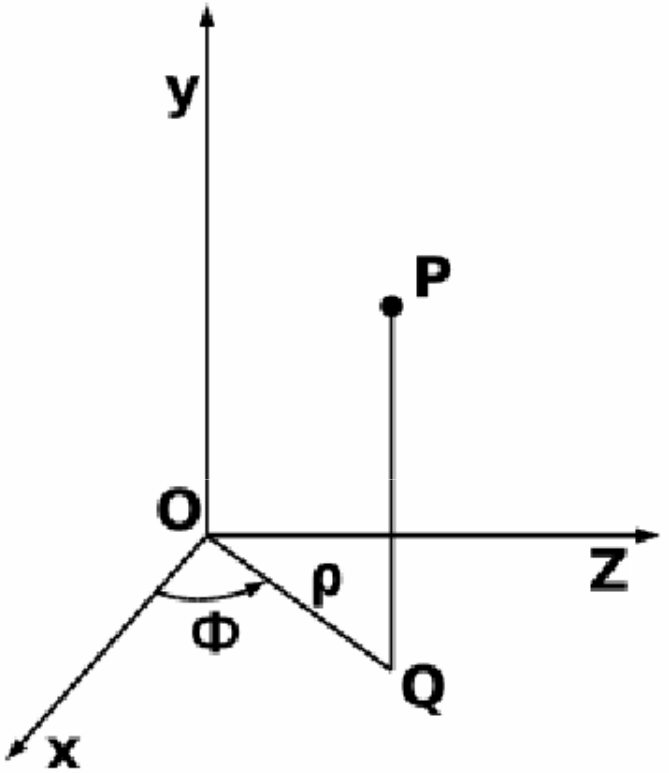
Approfondimento Proiezione Cilindrica

- Coordinate Cilindriche → Coordinate Cartesiane

$$x = \rho \cos \phi$$
$$y = y$$
$$z = \rho \sin \phi$$

- Coordinate Cartesiane → Coordinate Cilindriche

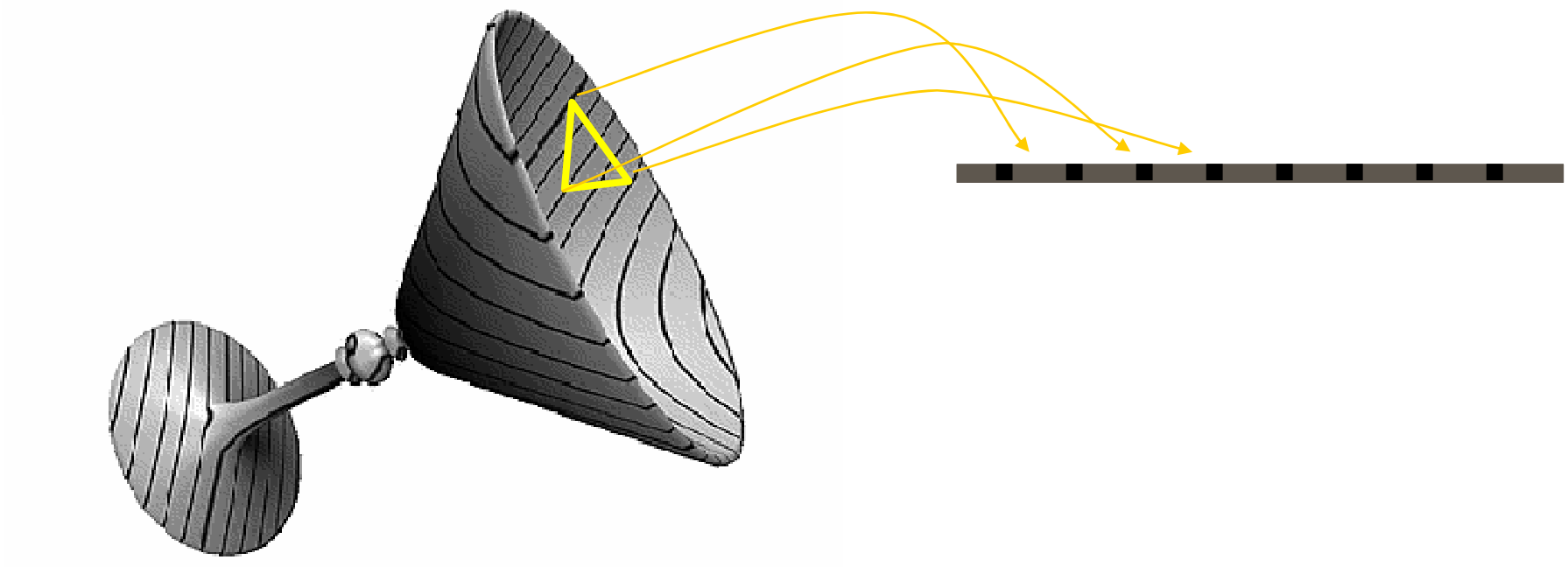
$$\rho = \sqrt{x^2 + y^2}$$
$$y = y$$
$$\phi = \arctan(z/x)$$



Atan2(z,x) corrisponde all'arcotangente di z/x. A differenza dell'arcotangente il valore dell'angolo restituito va da -π a π grazie al considerare i segni di *z* ed *x* nel calcolo.

Texture 1D

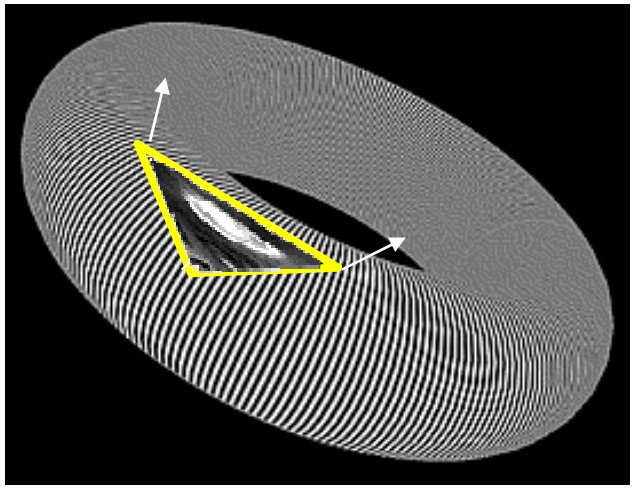
- La texture può anche essere una semplice immagine 1D



Environment Mapping: Sphere Mapping

Environment map: una tessitura che memorizza il colore dell'ambiente "riflesso" da ogni normale della semisfera. Come coordinate uv, basta utilizzare il ***vettore riflessione***:

$$u = \text{atan2}(r_y, r_x)$$
$$v = \arccos(r_z)$$



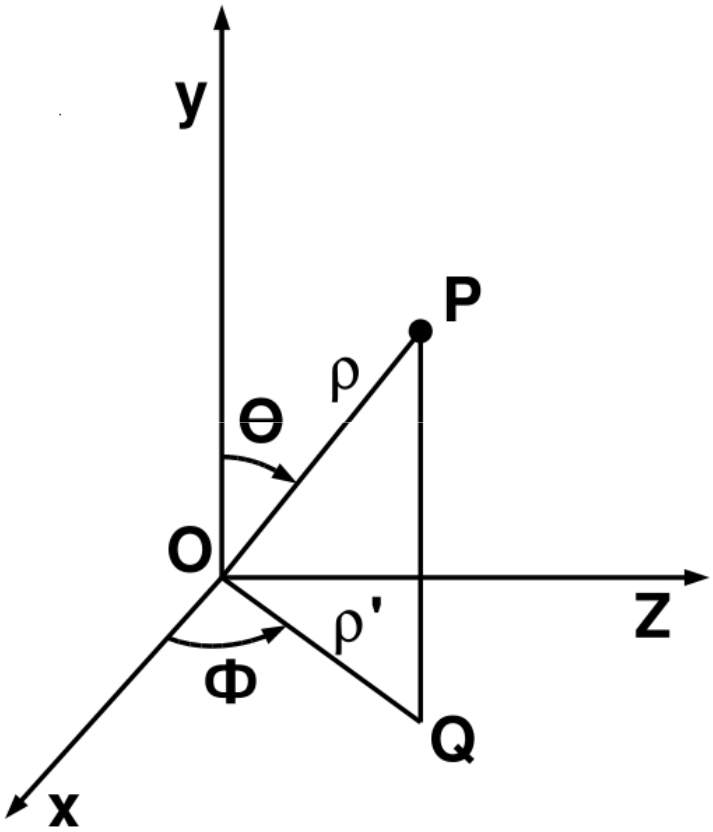
Approfondimento Sphere Mapping

- Coordinate Sferiche → Coordinate Cartesiane

$$x = \rho \sin\theta \cos\phi$$
$$y = \rho \cos\theta$$
$$z = \rho \sin\theta \sin\phi$$

- Coordinate Cartesiane → Coordinate Sferiche

$$\rho = \sqrt{x^2 + y^2 + z^2}$$
$$\phi = \arctan(z/x)$$
$$\theta = \arctan\left(\frac{y}{x^2+y^2+z^2}\right)$$



Environment Mapping: Sphere Mapping

simula oggetto a specchio che riflette uno sfondo lontano

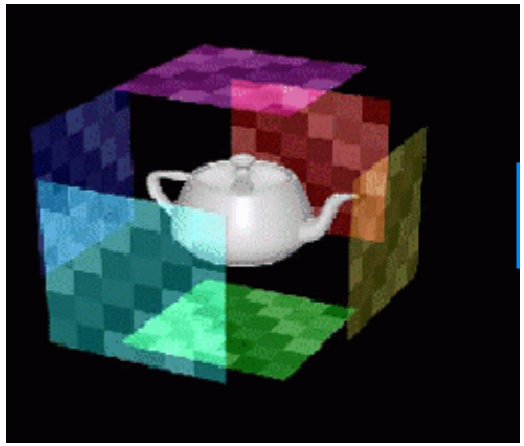


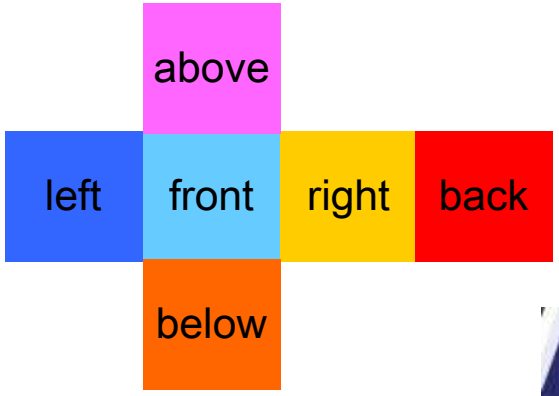


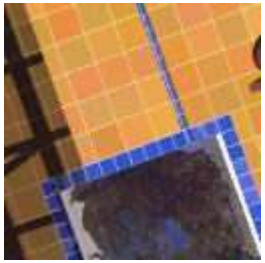



simula un materiale complesso
(condizioni di luce fisse)

Cube Mapping



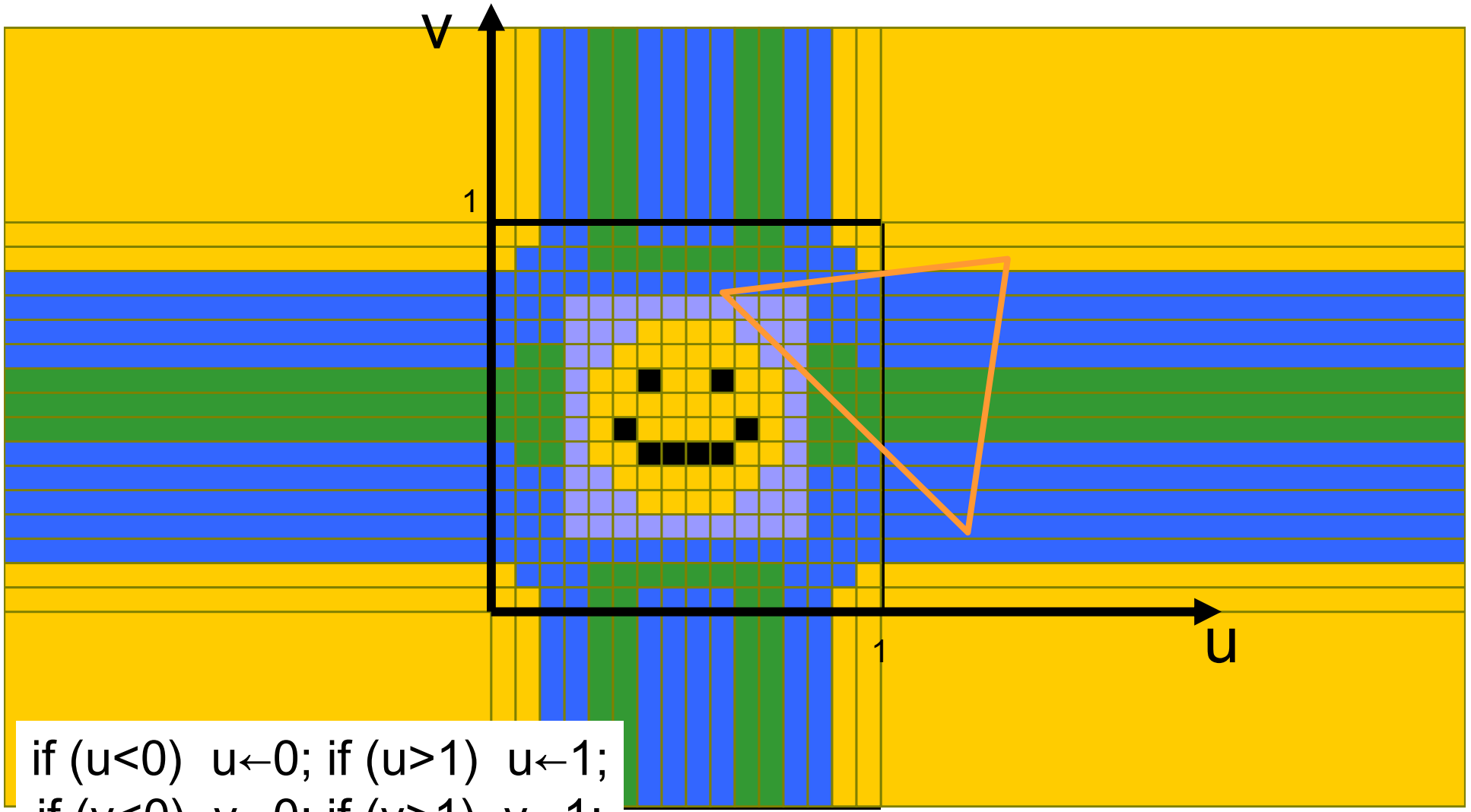




Cube Mapping

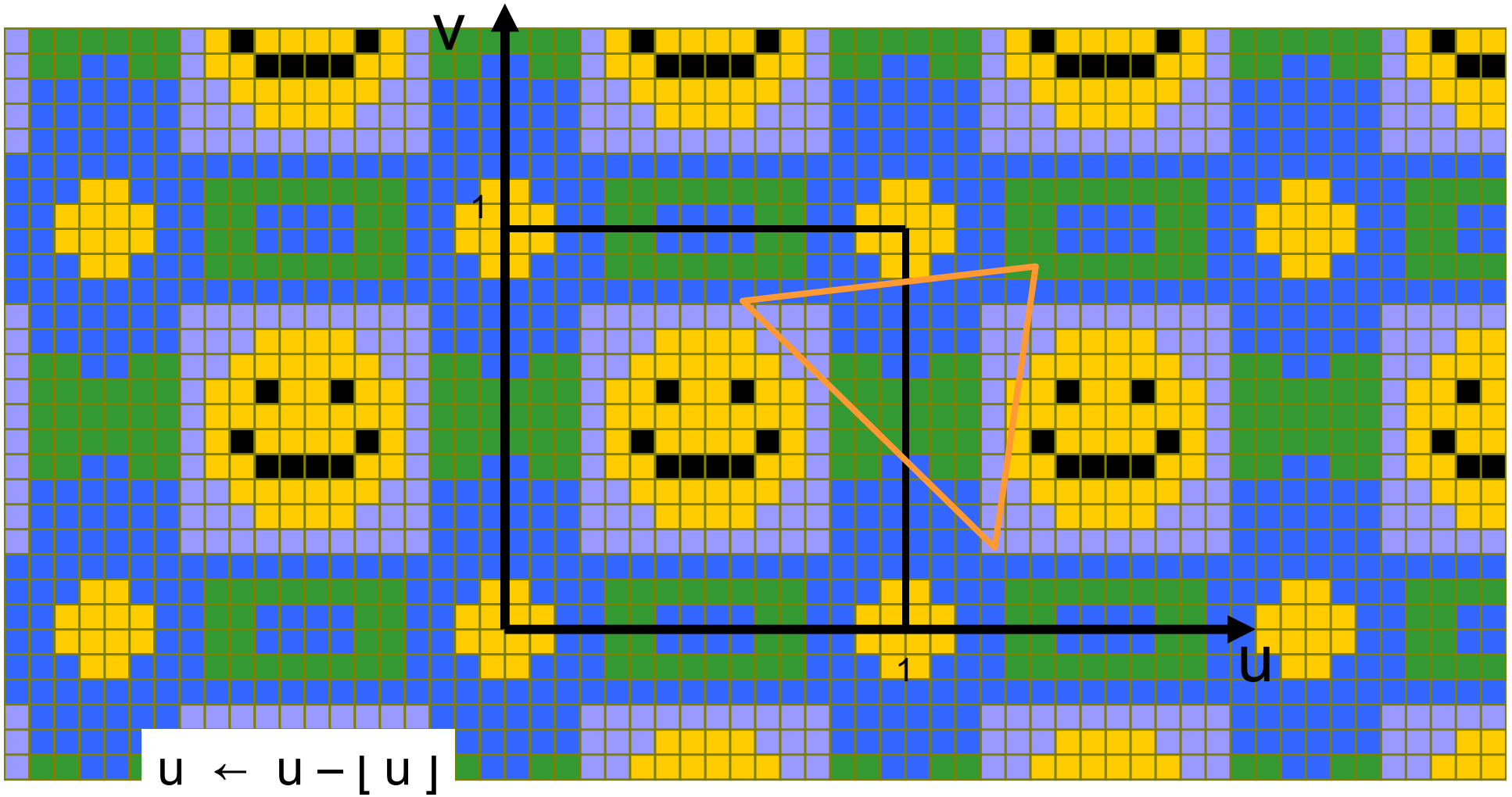
- Campionamento più uniforme dello sphere mapping
- Possibilità di generare real-time l'environment in modo semplice
- Generazione delle coordinate uv dato $r = (r_x , r_y , r_z)$:
  - Il valore massimo definisce la faccia su cui mappare
  - Esempio: (-3.2 , 5.1 , -8.4) → faccia –Z
  - Le coordinate sono ottenute dividendo le coordinate rimanenti per il valore del valore massimo (range [-1 , 1]) e normalizzando tra [0,1]
  - Esempio: (-3.2 / 8.4 , 5.1 / 8.4) → (-0.38, 0.61)
  - Per passare da un range di valori in [-1 , 1] ad uno in [0, 1] si aggiunge 1 e si divide per 2
  - Esempio: ((-0.38 + 1) / 2 , (0.61 + 1) / 2) → (0.31 , 0.80)

Gestione della Texture fuori dai bordi → modo *clamp*



if (u<0) u←0; if (u>1) u←1;
if (v<0) v←0; if (v>1) v←1;

Gestione della Texture fuori dai bordi → modo *repeat*



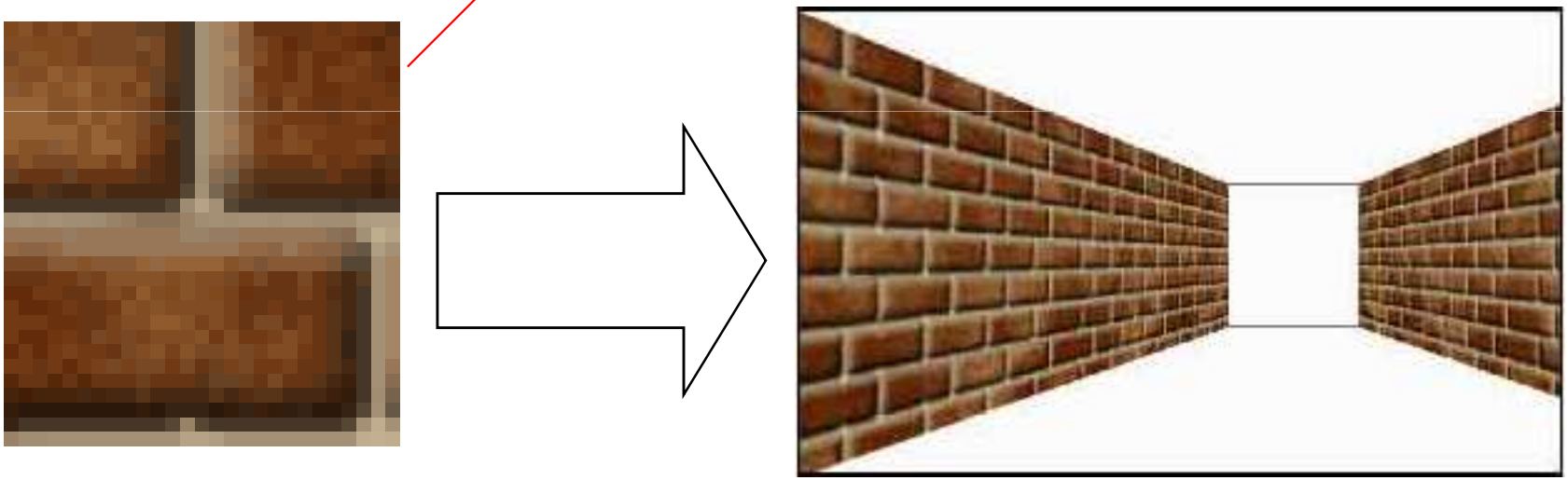
$u \leftarrow u - \lfloor u \rfloor$

$v \leftarrow v - \lfloor v \rfloor$

Texture Ripetute

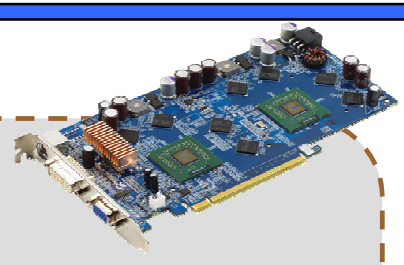
- Tipico utilizzo:

Nota: deve essere predisposta ad essere ripetuta in modo da non creare discontinuità visive (si parla di TILE e di TILING)



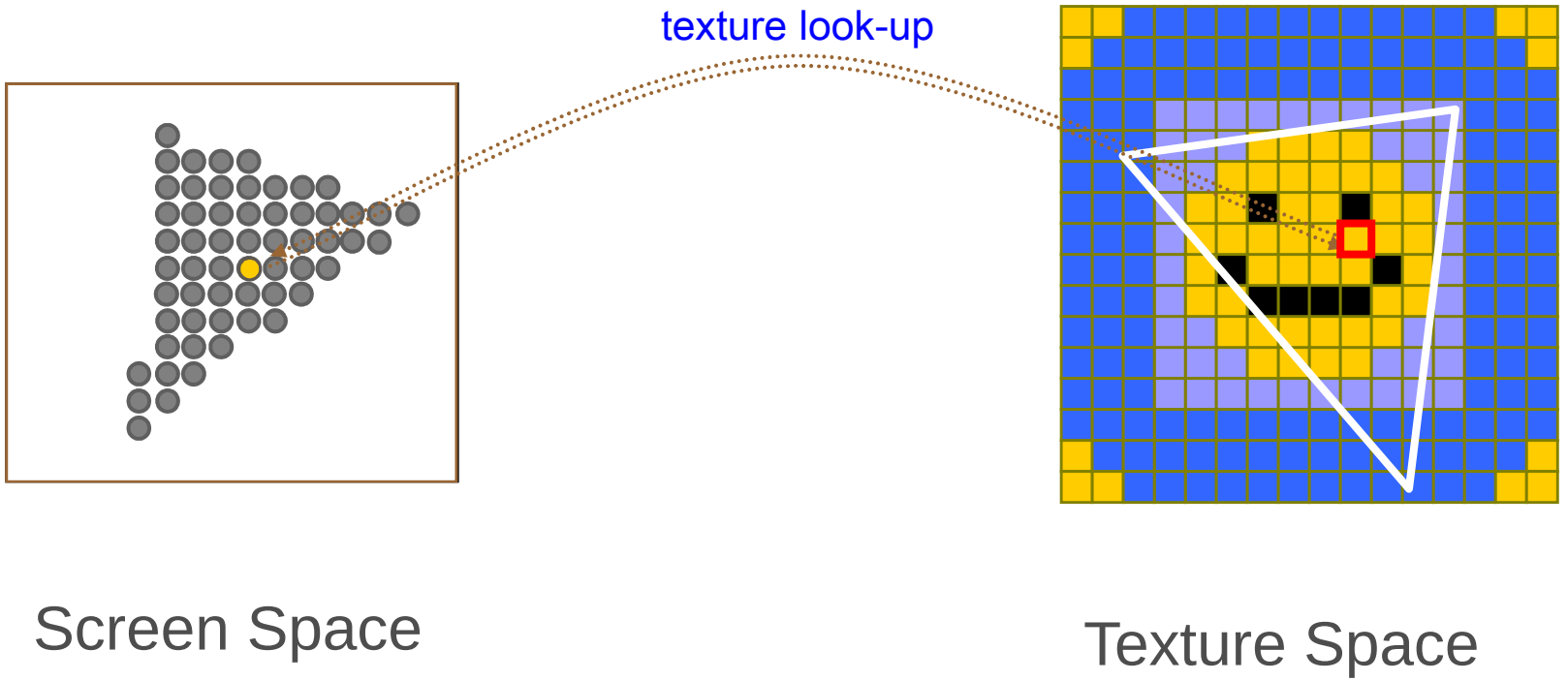
Molto efficiente in spazio: una sola texture mappa su molti triangoli

Texture Mapping: texture look-up

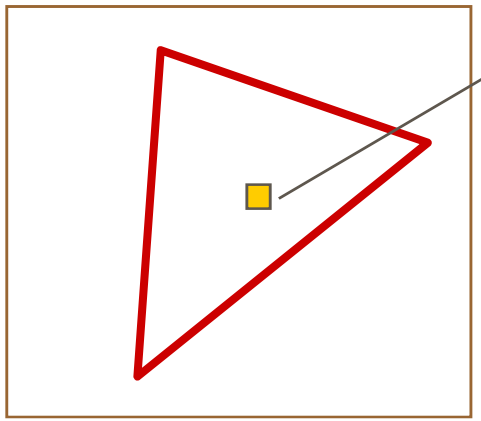


Texture Look-Up

- Un frammento ha coordinate non intere (in texels)

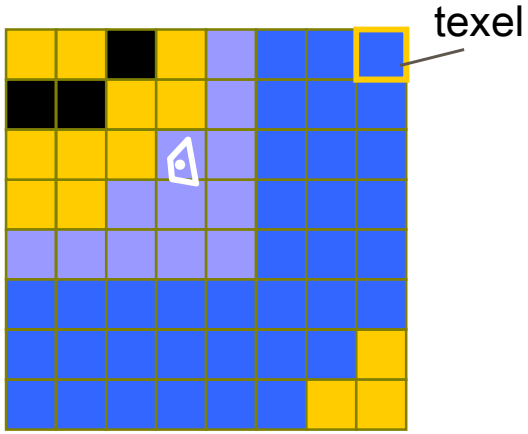


Texture Look-Up



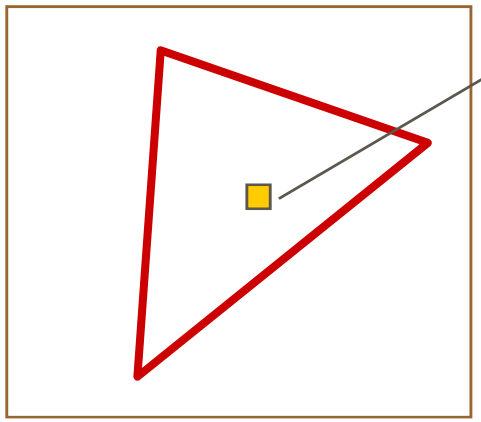
un pixel = meno di un texel

magnification



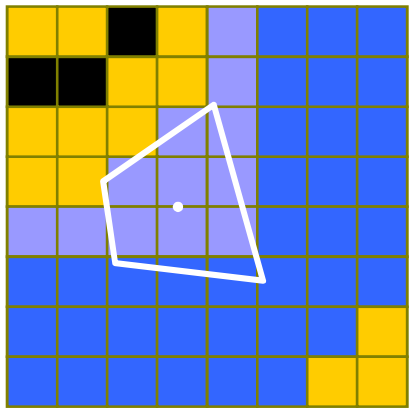
Screen Space

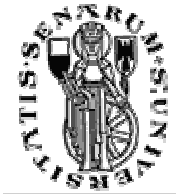
Texture Space



un pixel = più di un texel

minification





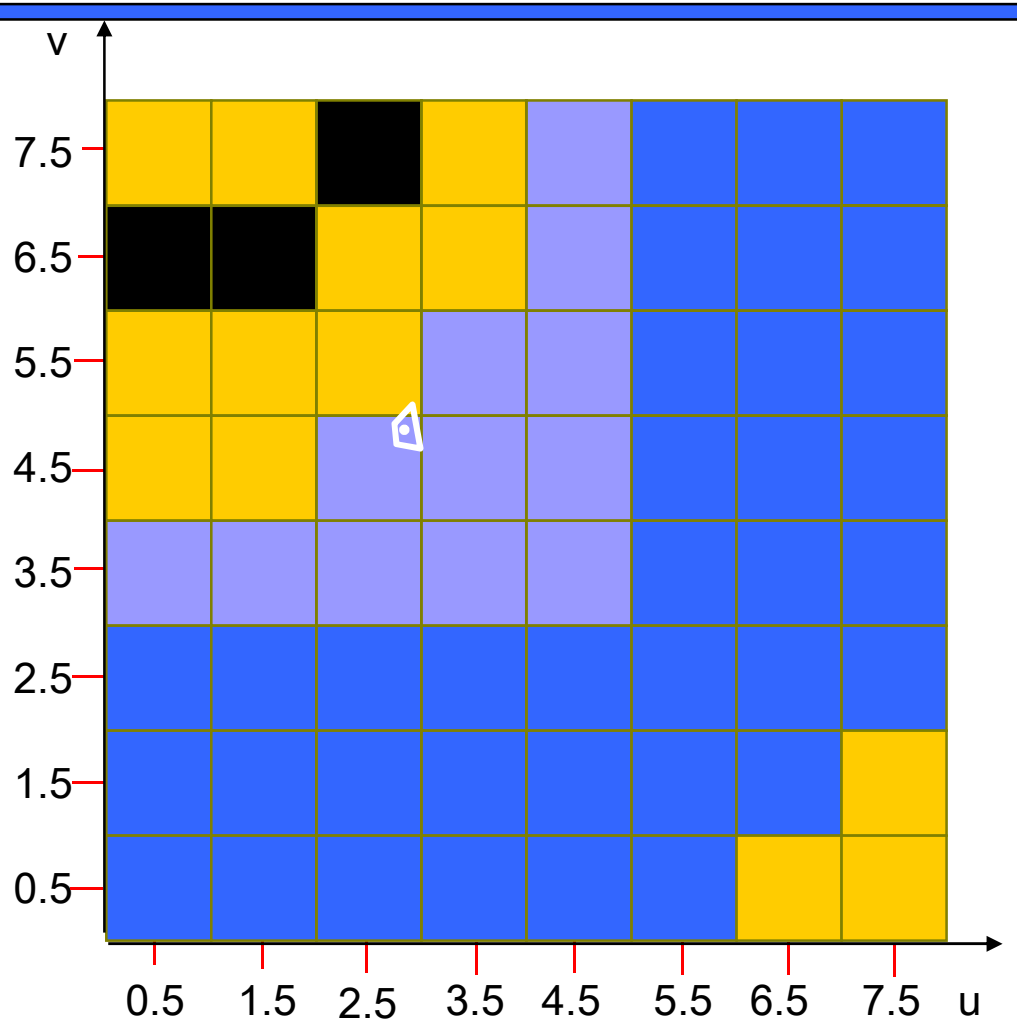
Magnification

Soluzione 1:
prendo il texel in cui sono

(equivale a prendere
il texel più vicino)

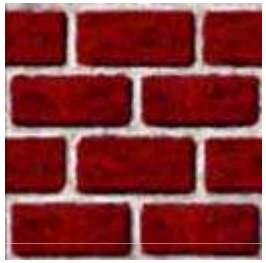
equivale ad arrotondare
alle coordinate texel
intere

"Nearest Filtering"



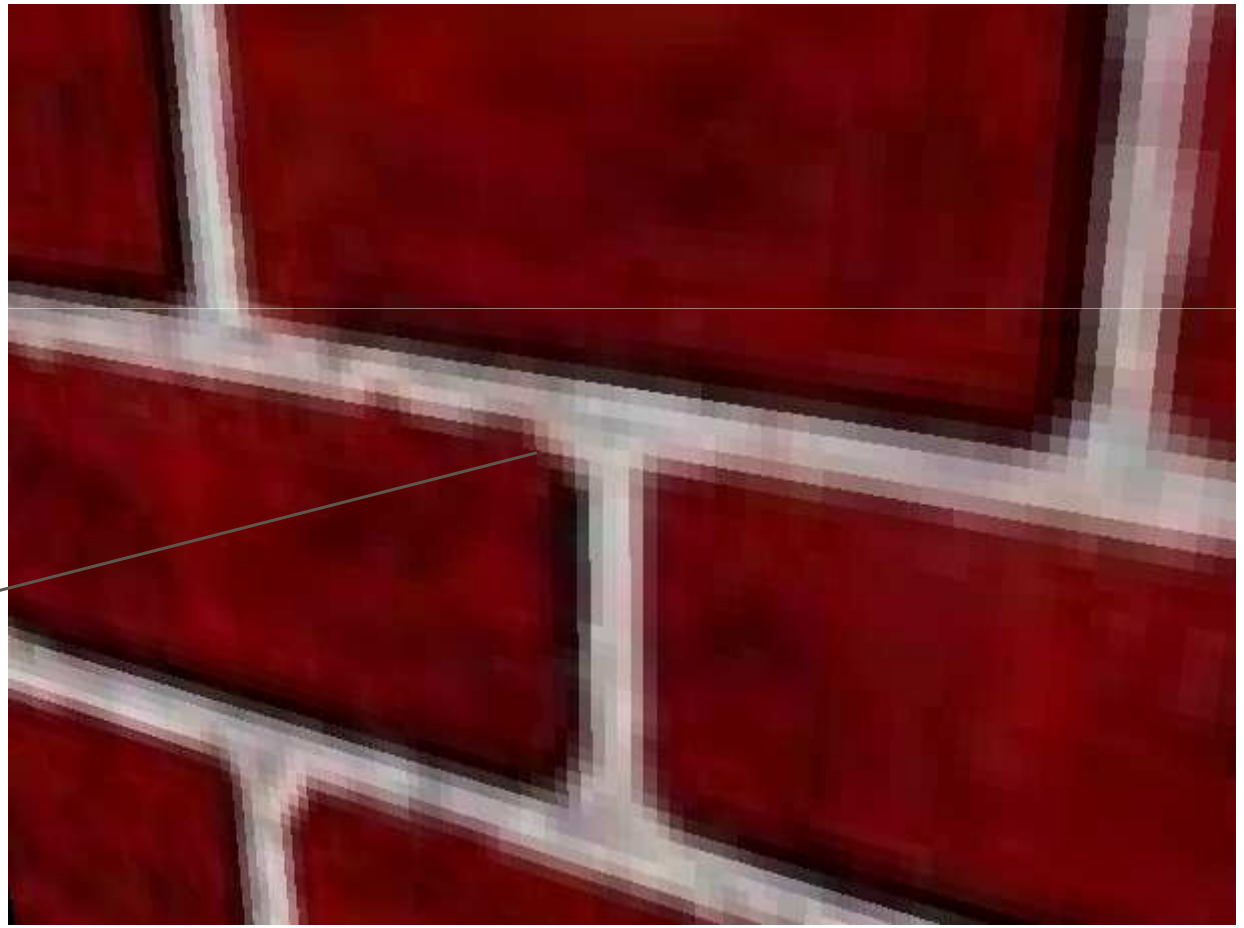
Magnification

Nearest Filtering: risultato visivo



texture 128x128

"si vedono i texel !"

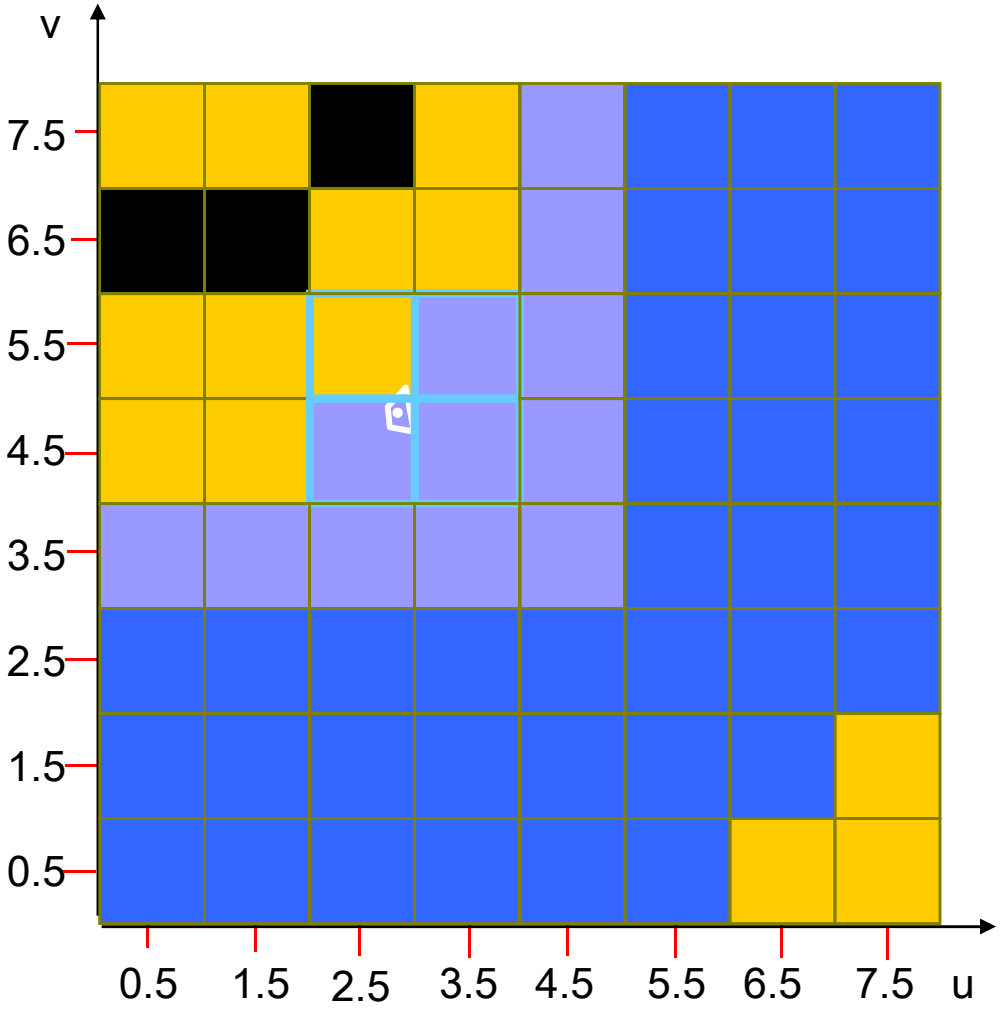


Magnification

Soluzione 2:
Medio il valore dei quattro texel più vicini

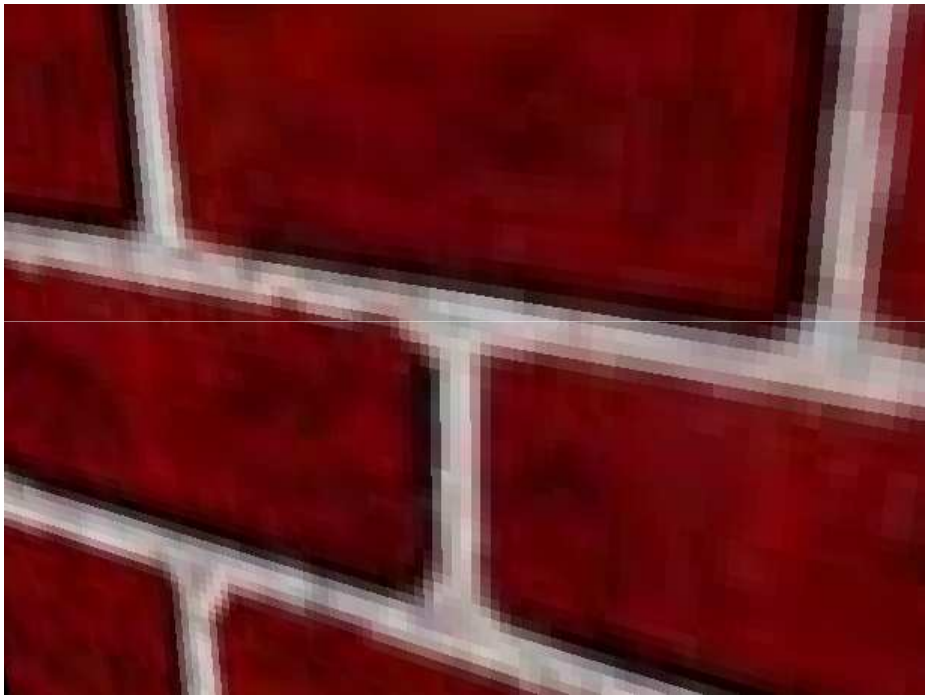
Interpolazione Bilineare

$$
\begin{aligned}
\alpha &= x - \lfloor x \rfloor \\
\beta &= y - \lfloor y \rfloor \\
p(x,y) &= (1-\alpha)(1-\beta)p_{11} + \\
&+ \alpha(1-\beta)p_{12} + \\
&+ (1-\alpha)\beta p_{21} + \alpha\beta p_{22}
\end{aligned}
$$



Magnification

Nearest Filtering

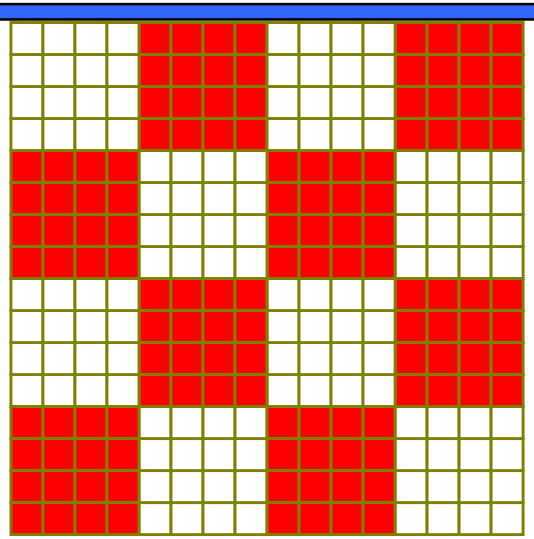


Bilinear Interpolation

Magnification

- **Modo Nearest:**
  - si vedono i texel
  - va bene se i bordi fra i texel sono utili
  - più veloce
- **Modo Interpolazione Bilineare:**
  - di solito qualità migliore
  - può essere più lento
  - rischia di avere un effetto "sfuocato"

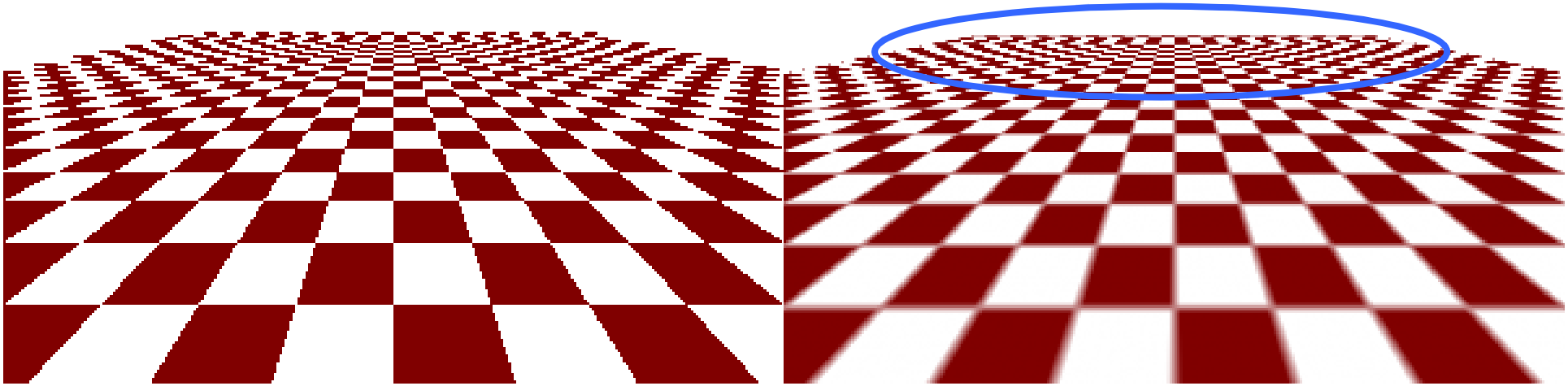
Minification



Nearest Filtering

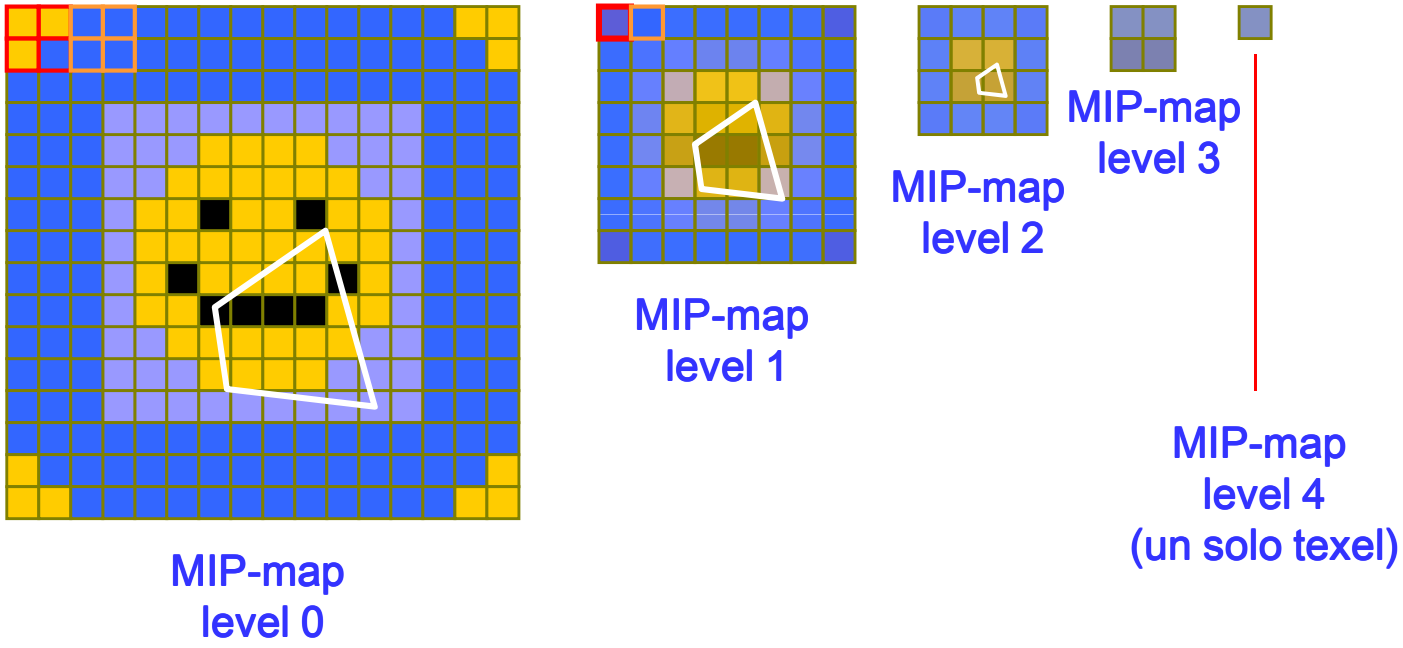
Bilinear interpolation

non risolve il problema



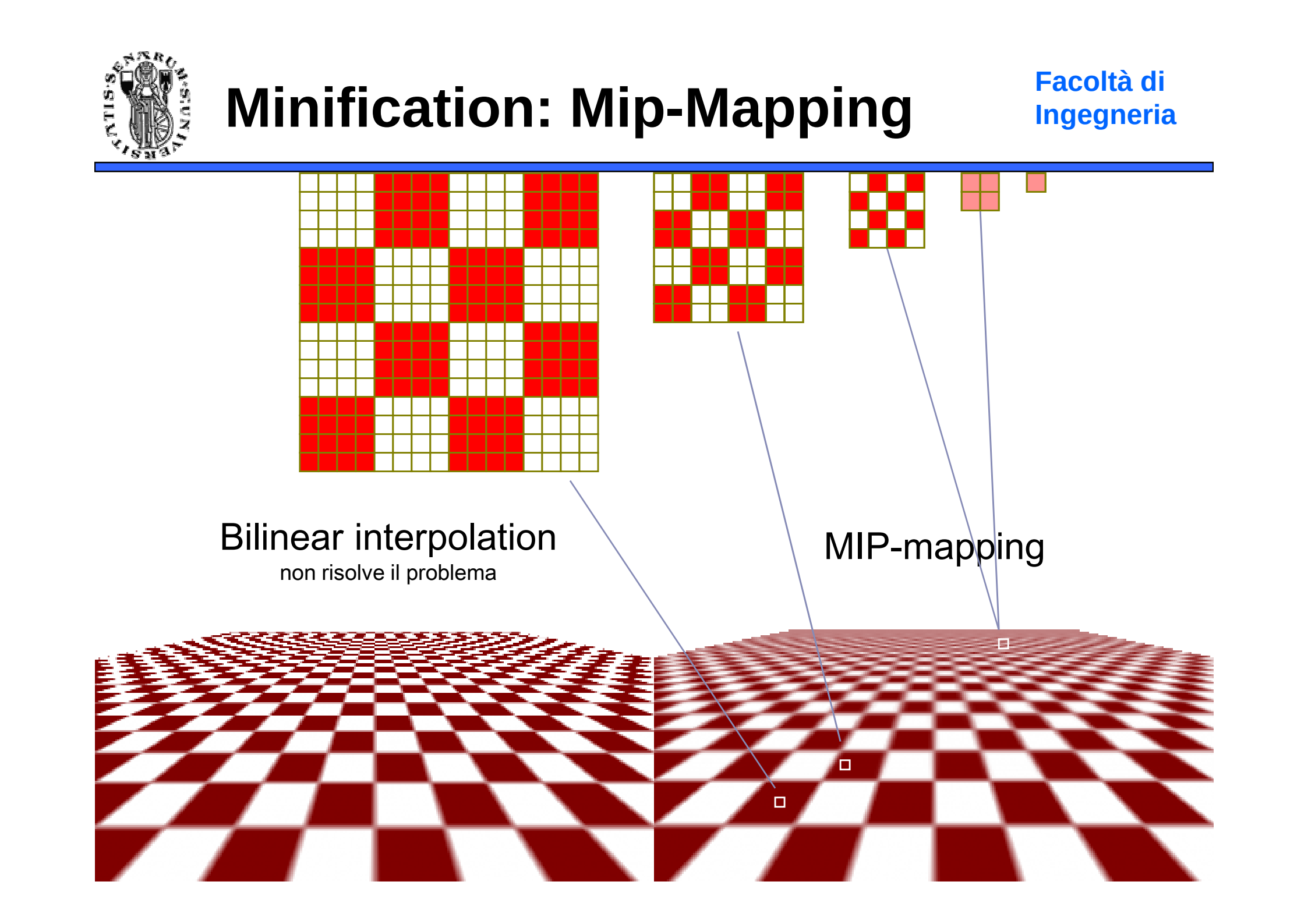
Minification: Mip-Mapping

MIP-mapping: "Multum In Parvo"

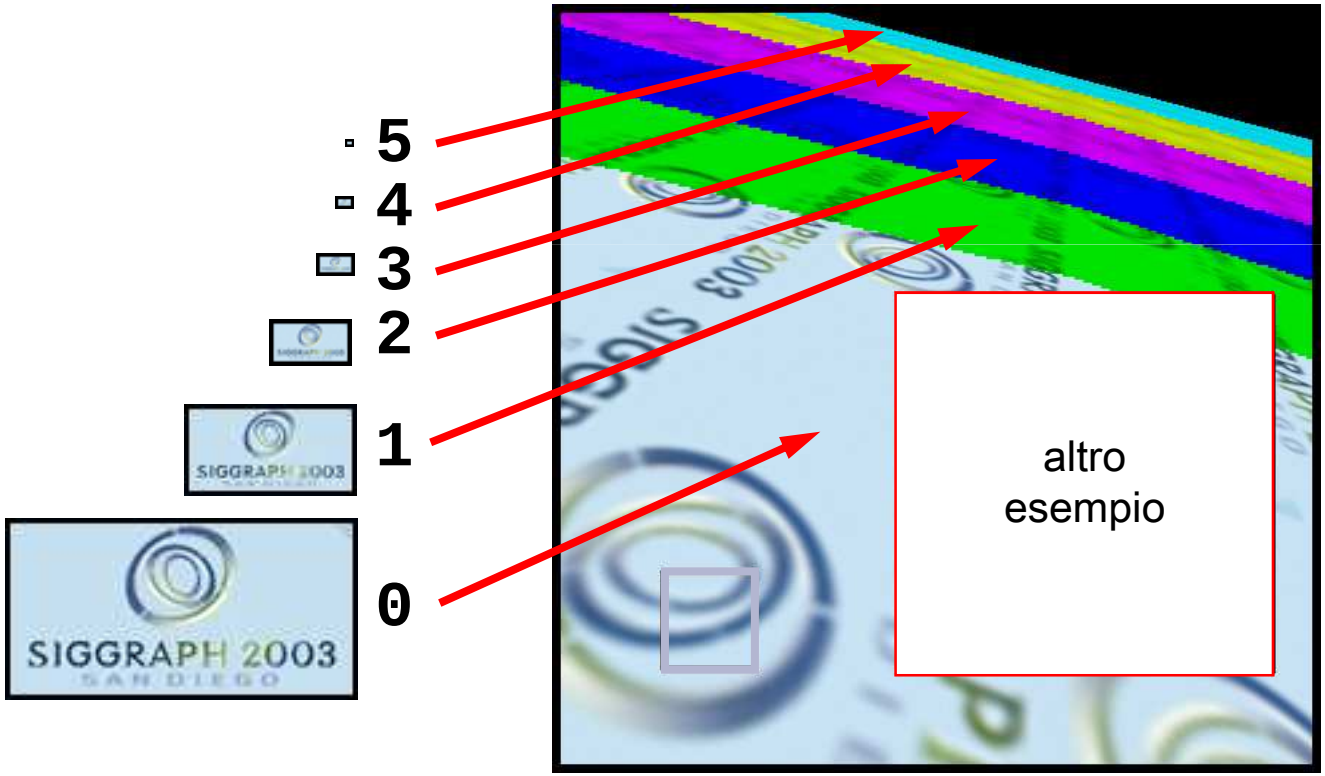


Mip-Mapping

- Definiamo un **fattore di scala**, ρ=texels/pixel come valore massimo fra ρ_x e ρ_y, che può variare sullo stesso triangolo, può essere derivato dalle matrici di trasformazione ed è calcolato nei **vertici**, interpolato nei **frammenti**
- Il **livello di mipmap** da utilizzare è: $\log_2 \rho$ dove il livello 0 indica la massima risoluzione
- Il livello non è necessariamente un numero intero e può quindi essere arrotondato

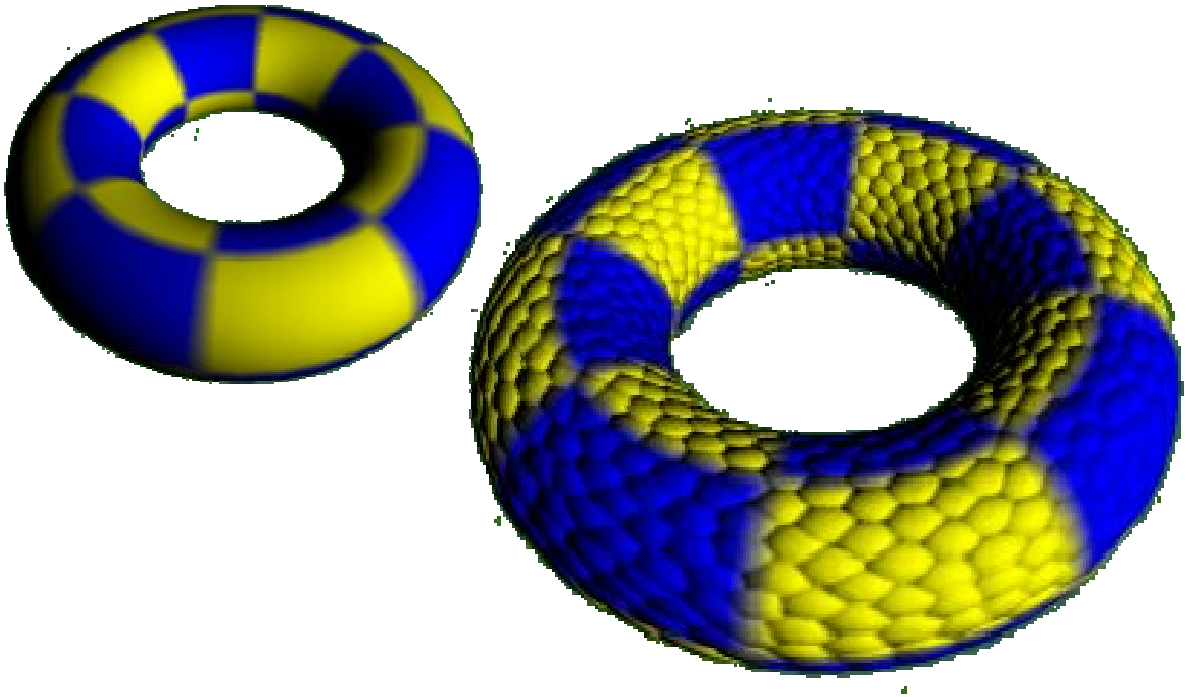

Minification: Mip-Mapping
Facoltà di Ingegneria
Bilinear interpolation
non risolve il problema
MIP-mapping

Minification: Mip-Mapping



Bump Mapping

- Un ulteriore modifica all'apparenza del rendering può essere effettuata usando il bump mapping



Bump Mapping

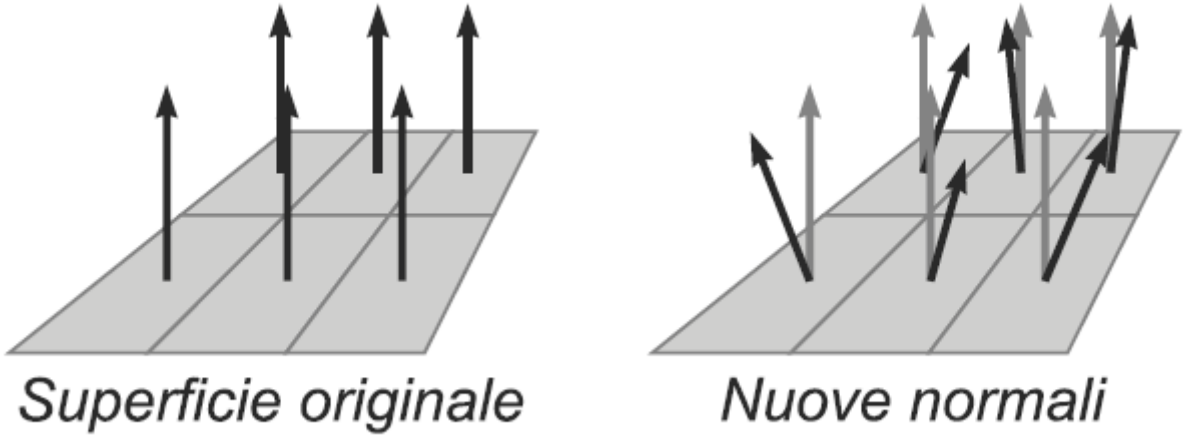
- Il metodo prevede di variare la normale alla superficie vertice per vertice utilizzando la formula:

$$\vec{\mathbf{N}}_{new} = \vec{\mathbf{N}}_{old} + \vec{\mathbf{D}};$$
$$\vec{\mathbf{D}} = (\Delta x, \Delta y, \Delta z)$$

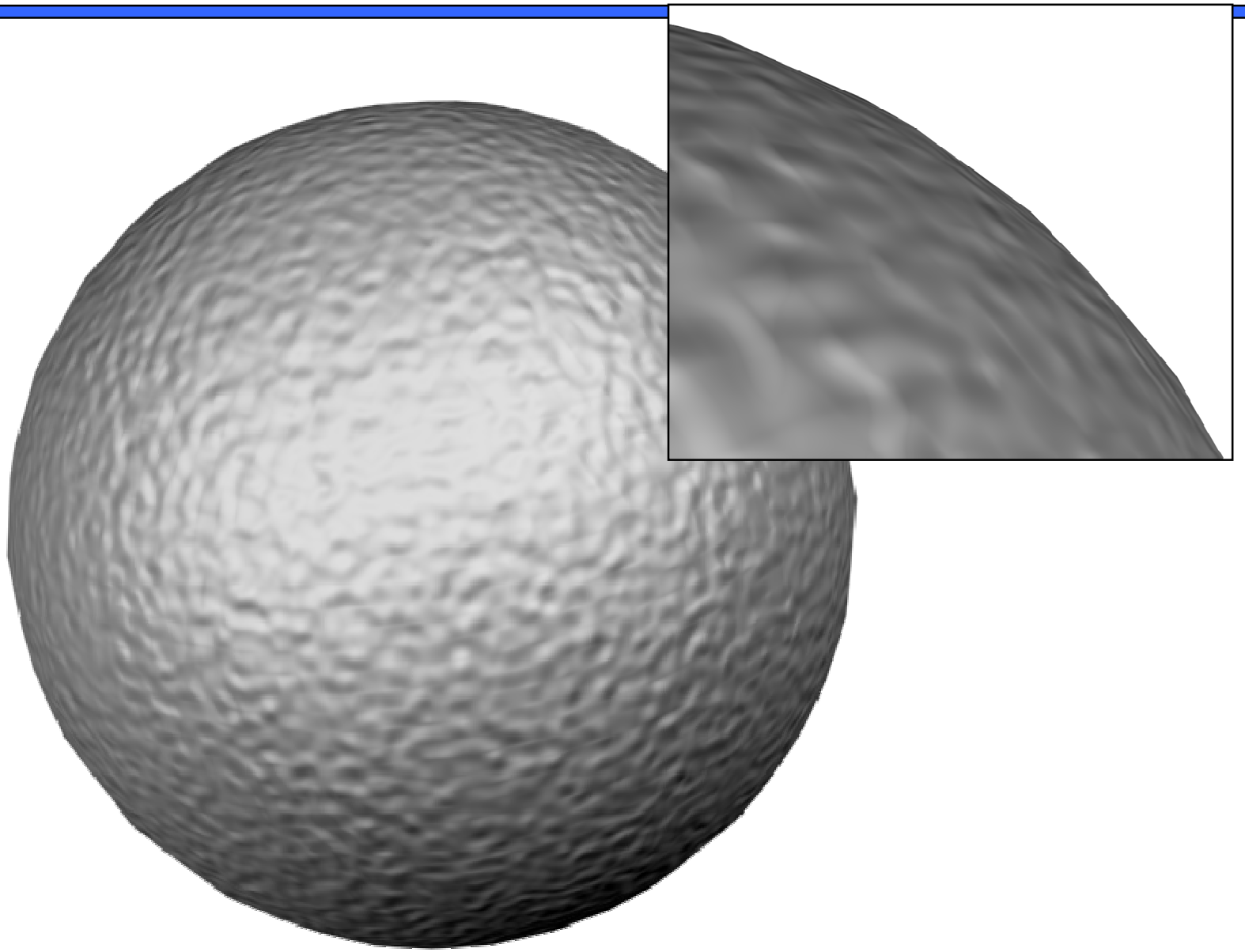
- I texel in questo caso sono utilizzati ad uno stadio diverso rispetto ai color texel, ossia prima del calcolo dell'equazione di illuminazione

Bump Mapping

- L’effetto che si ottiene è una perturbazione del valore delle normali che altera il rendering senza modificare la geometria



Bump Mapping

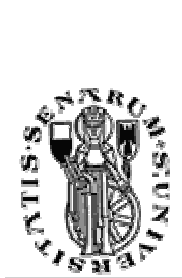


Displacement Mapping

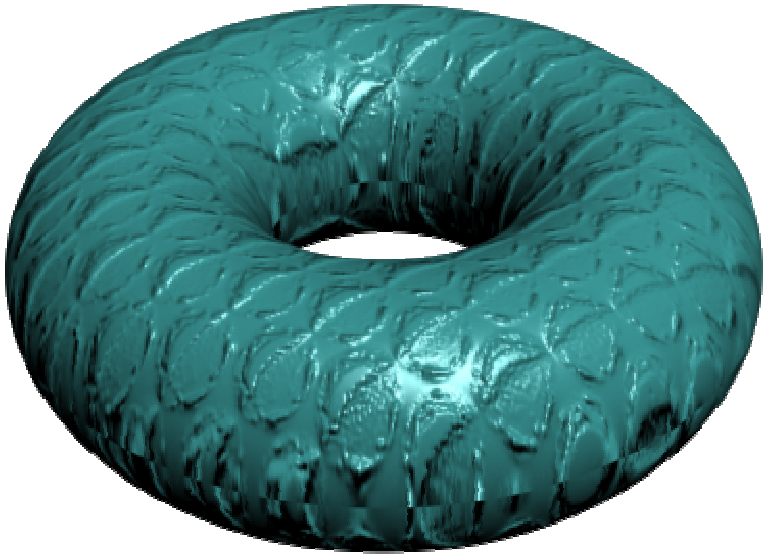
- Nel displacement mapping si modifica effettivamente la geometria dell'oggetto spostando i punti della superficie:

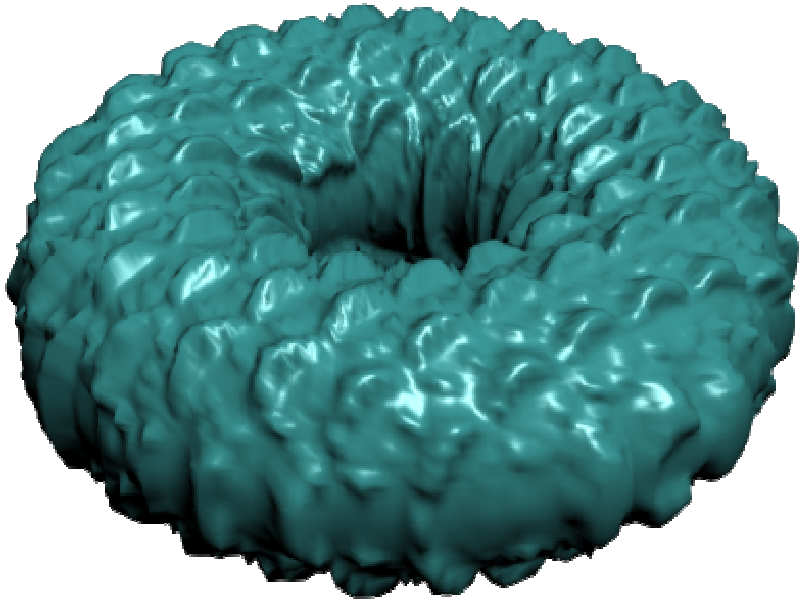
$$P_{new} = P_{old} + h \cdot \vec{\mathbf{N}}$$

- Il displacement mapping è eseguito in fase di rendering e non modifica stabilmente la geometria della scena
- Rispetto al bump mapping anche la silhouette del modello mostra le corrette deformazioni



Displacement Mapping (esempio)





Bump Mapping

Displacement Mapping

Domande?