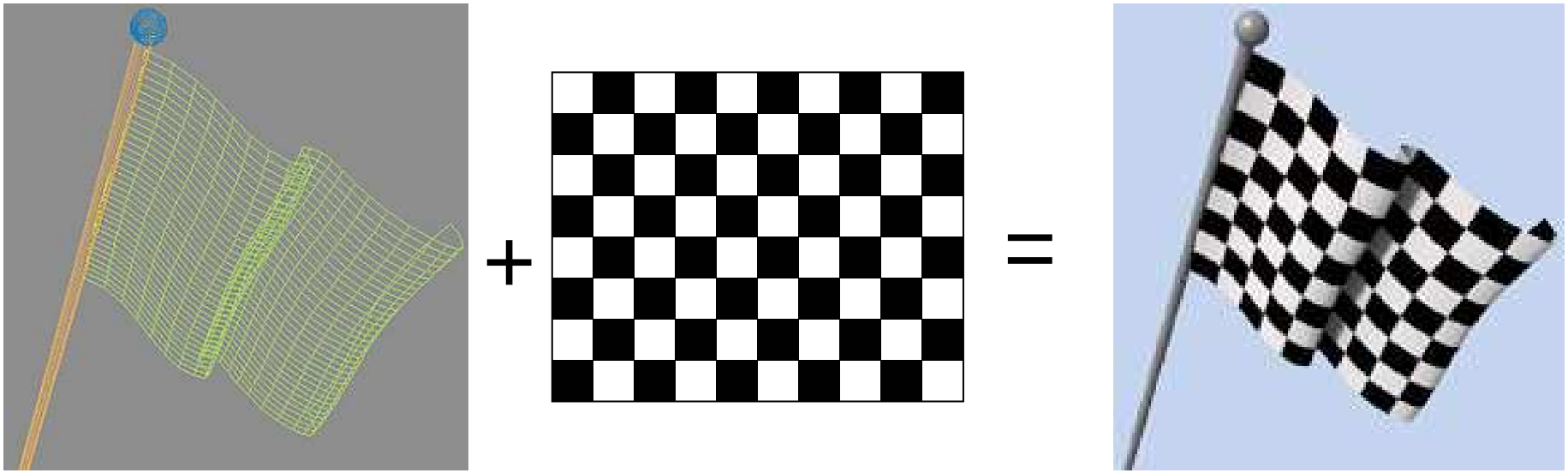
Grafica Computazionale

Texturing

Fabio Ganovelli
fabio.ganovelli@gmail.com
a.a. 2006-2007

Tipica applicazione: rimappare immagini sulla geometria



geometria 3D
(mesh di quadrilateri)

RGB texture 2D

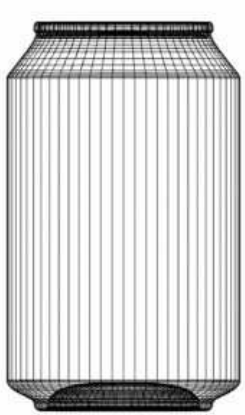
(color-map)

- Questo specifico risultato potremmo ottenerlo anche senza texture, con i giusti poligoni di colore bianco e nero

Altri esempi

immagine

Geometria + immagine



Altro esempio

Geometria

+

Immagine

=

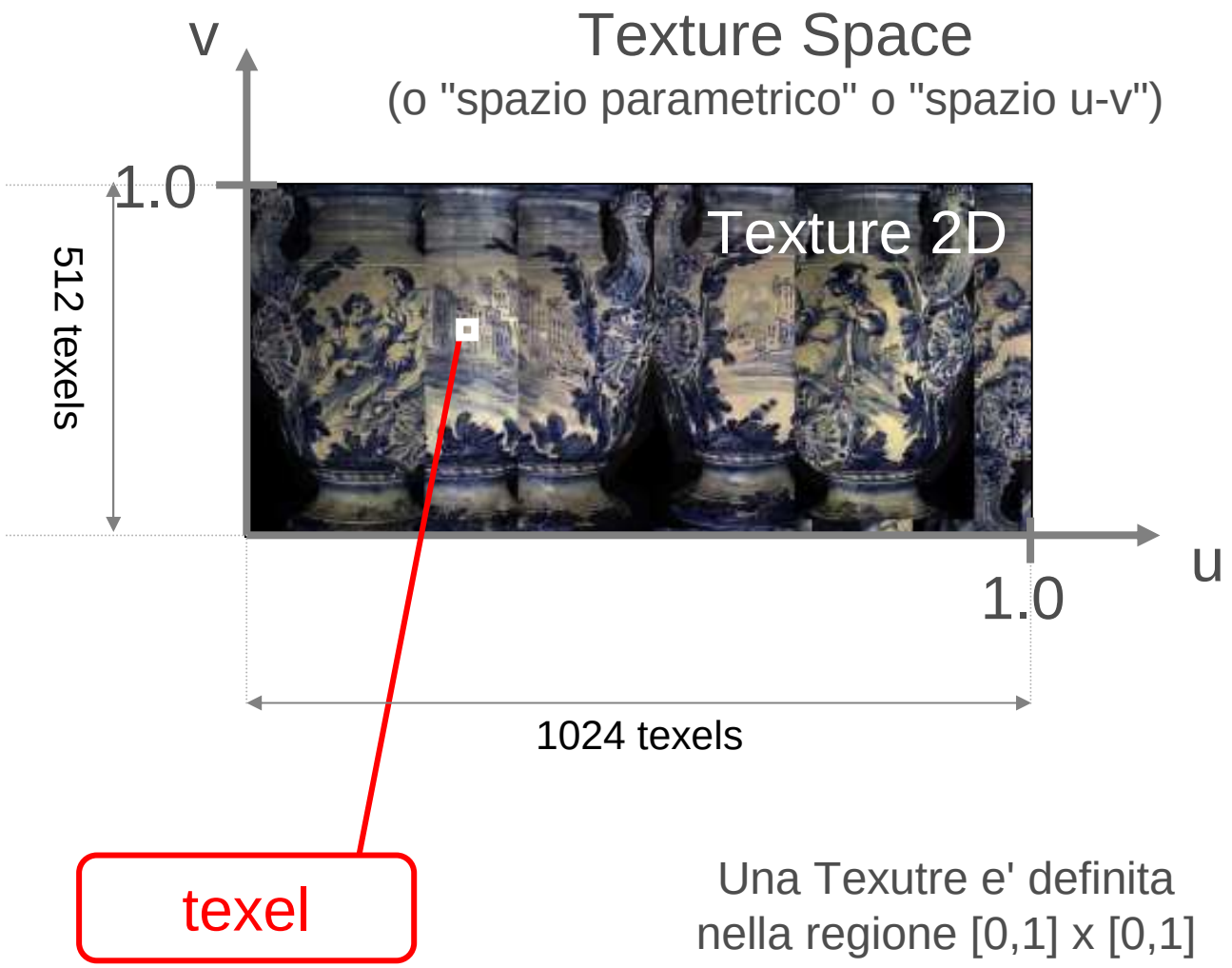
Geometria+
immagine

Notazione: texture, texel

Texture: un'immagine raster da "applicare" sulla geometria

Texel: **Tex**ture **El**ement, un pixel della texture

Texture Space: lo spazio parametrico della texture



Una Texutre e' definita nella regione [0,1] x [0,1] dello "spazio parametrico"

Texture Mapping: Storia

- 1974 introdotto da Ed Catmull
  - nella sua Phd Thesis

Ed Catmull

(MEGA-MEGA-GURU)

- Solo nel 1992 (!) si ha text. mapping hardware
  - Silicon Graphics RealityEngine
- Dal 92 a oggi:
  rapido aumento della diffusione
  - strada intrapresa soprattutto da low end graphic boards
- Oggi:
  una delle fondamentali primitive di rendering
  - la principale tecnica image based

Texture Mapping

- 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

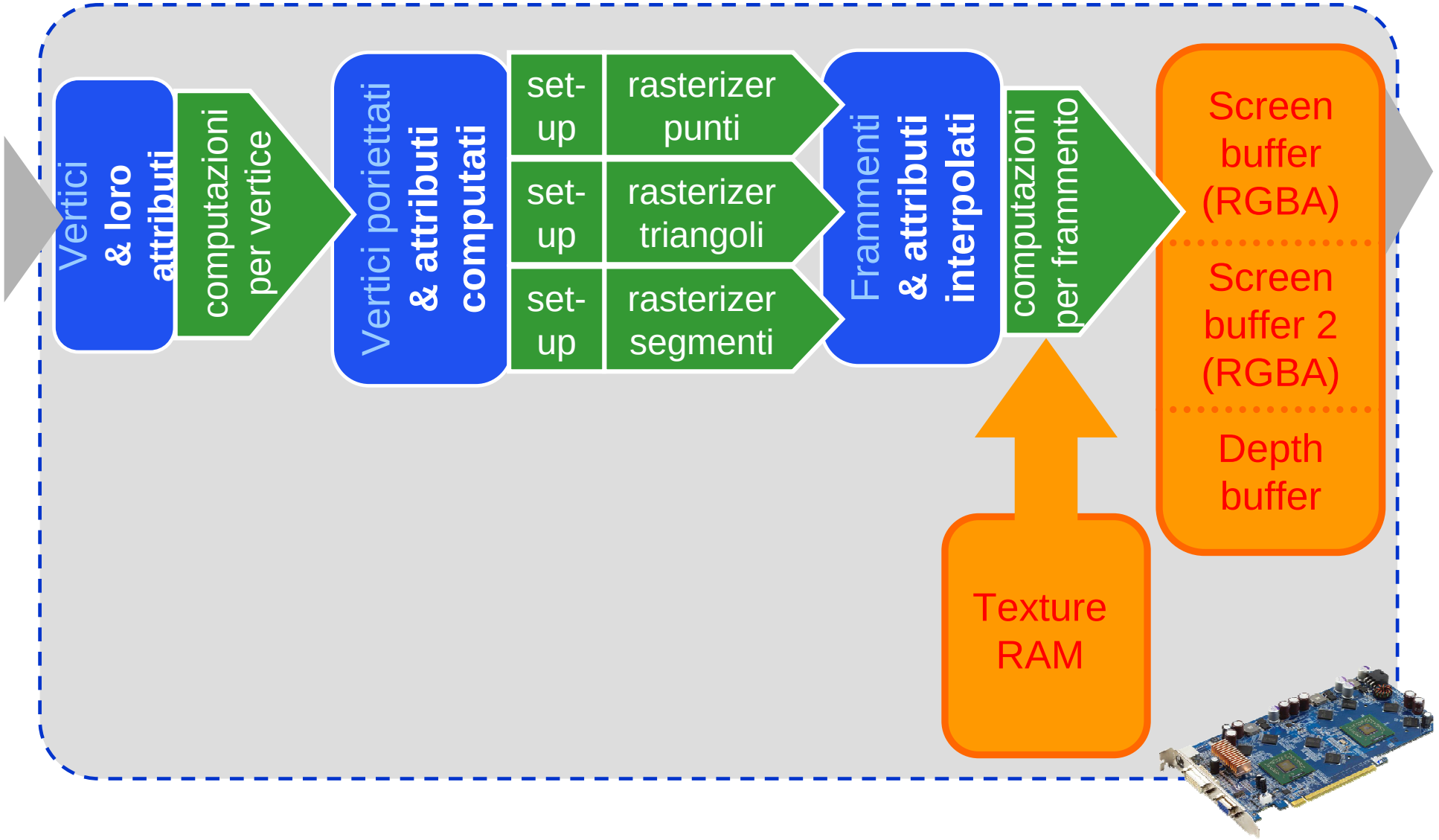
  di Texels (campioni di tessitura)

  dello stesso tipo

Texels

- Tipici 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"
  - ...

Memoria RAM nelle schede grafiche



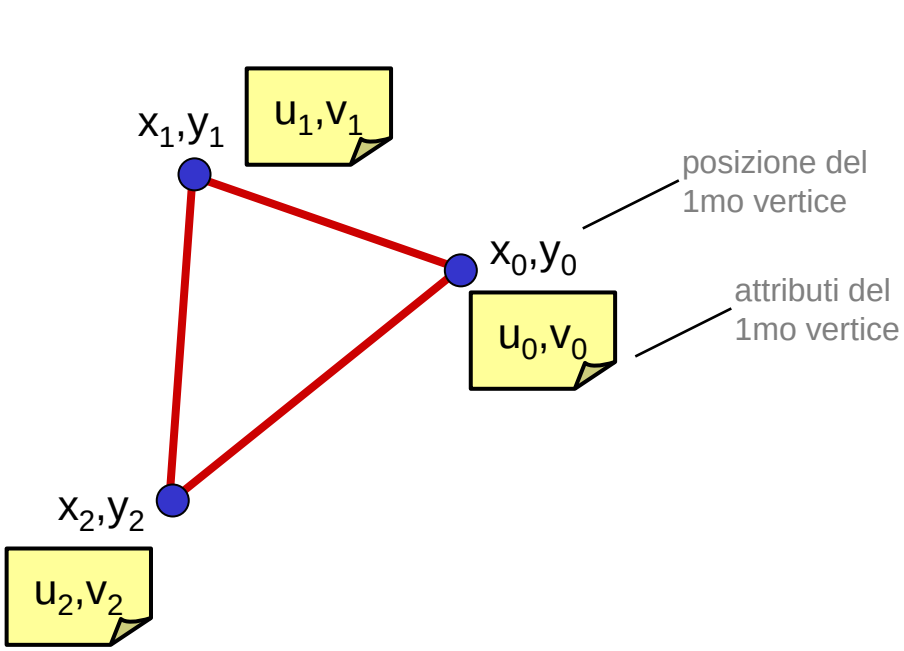
In OpenGL

- Ad esempio:

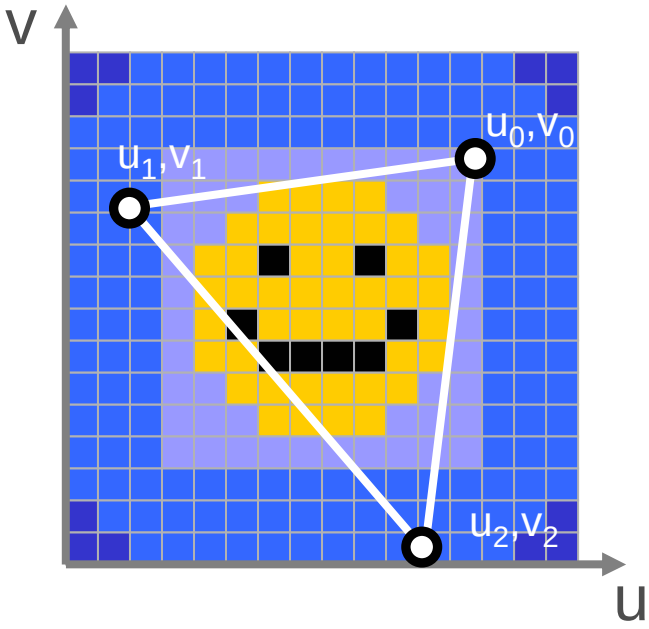
```
glEnable(GL_TEXTURE_2D);
glBindTexture (GL_TEXTURE_2D, ID);

glTexImage2D (
      GL_TEXTURE_2D,
      0,          // mipmapping
      GL_RGB,   // formato interno
      imageWidth, imageHeight,
      0,        // bordo
      GL_RGB, // formato nella RAM
      GL_UNSIGNED_BYTE,
      imageData);
```

Texture Mapping

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



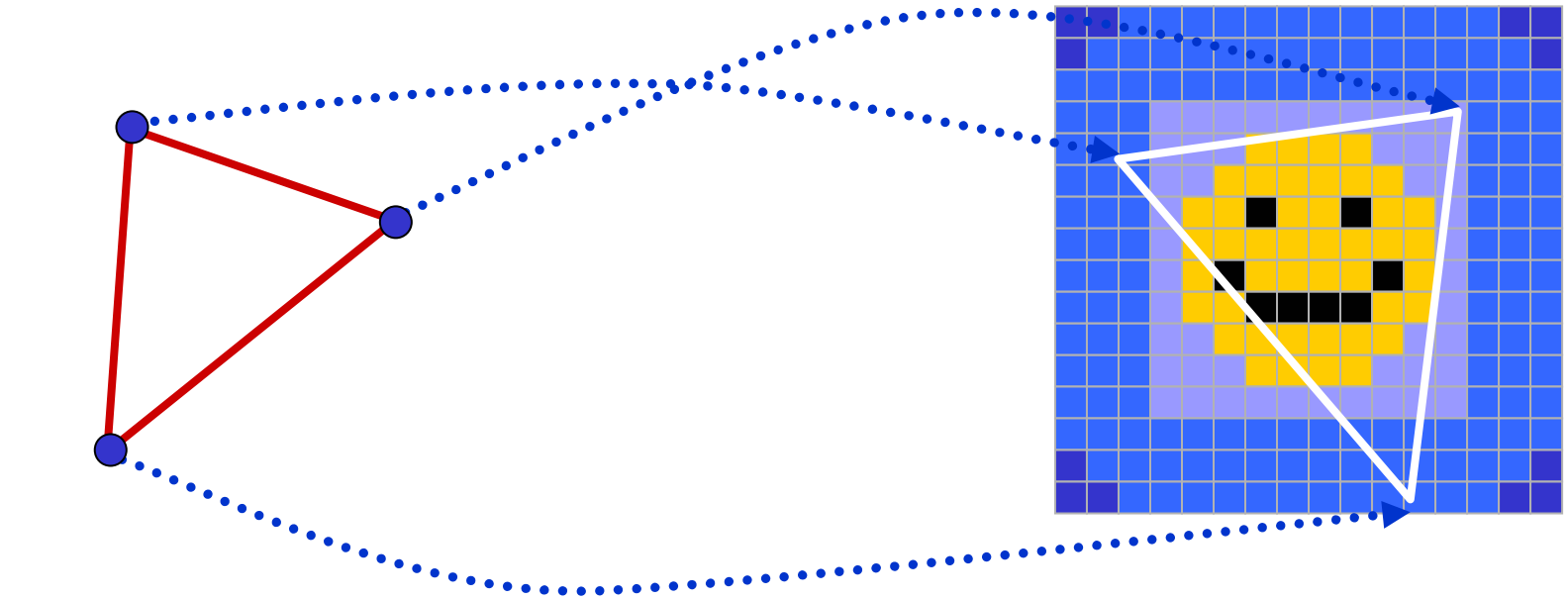
Screen Space



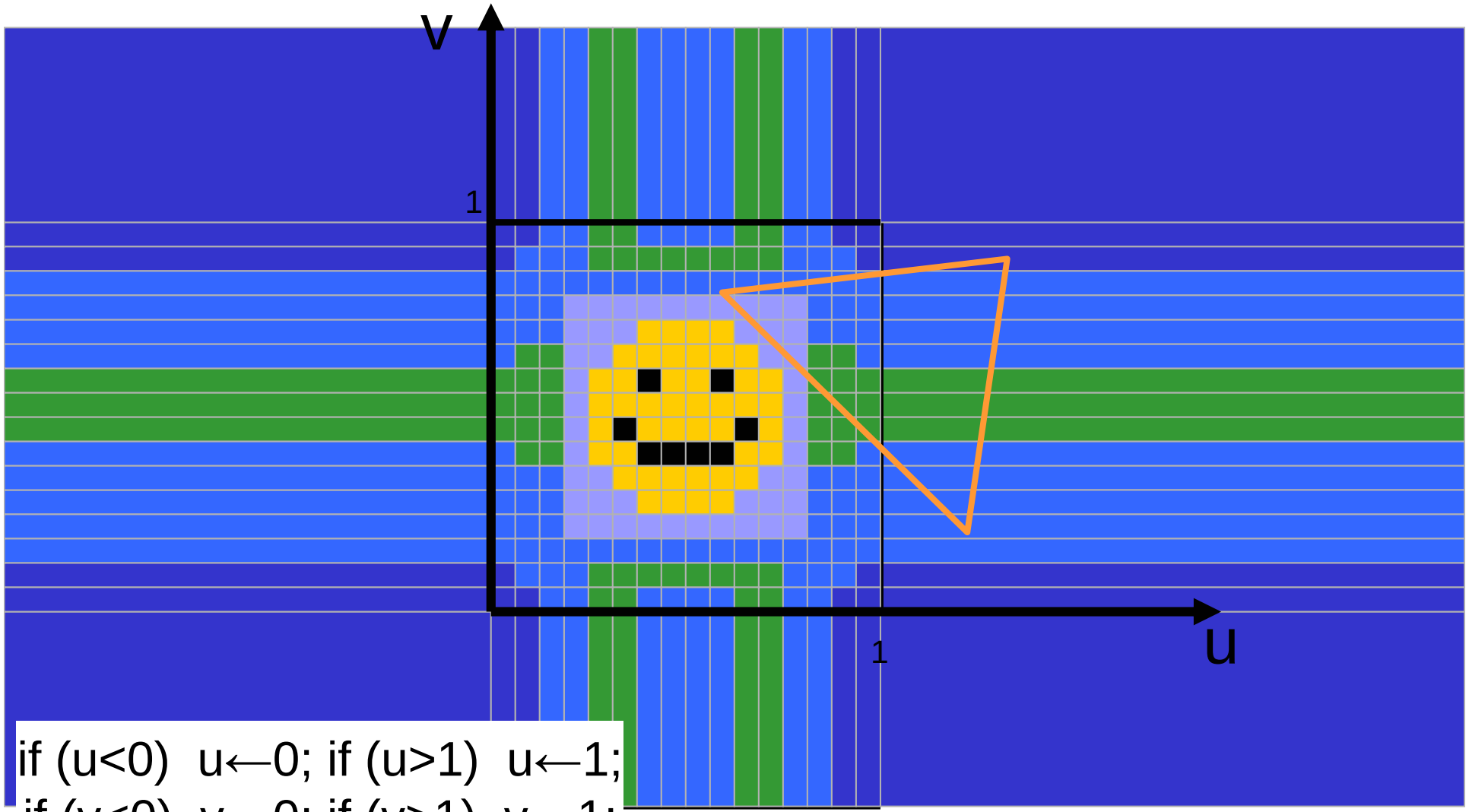
Texture Space

Texture Mapping

- Così in pratica definisco un mapping fra il triangolo 3D e un triangolo di tessitura

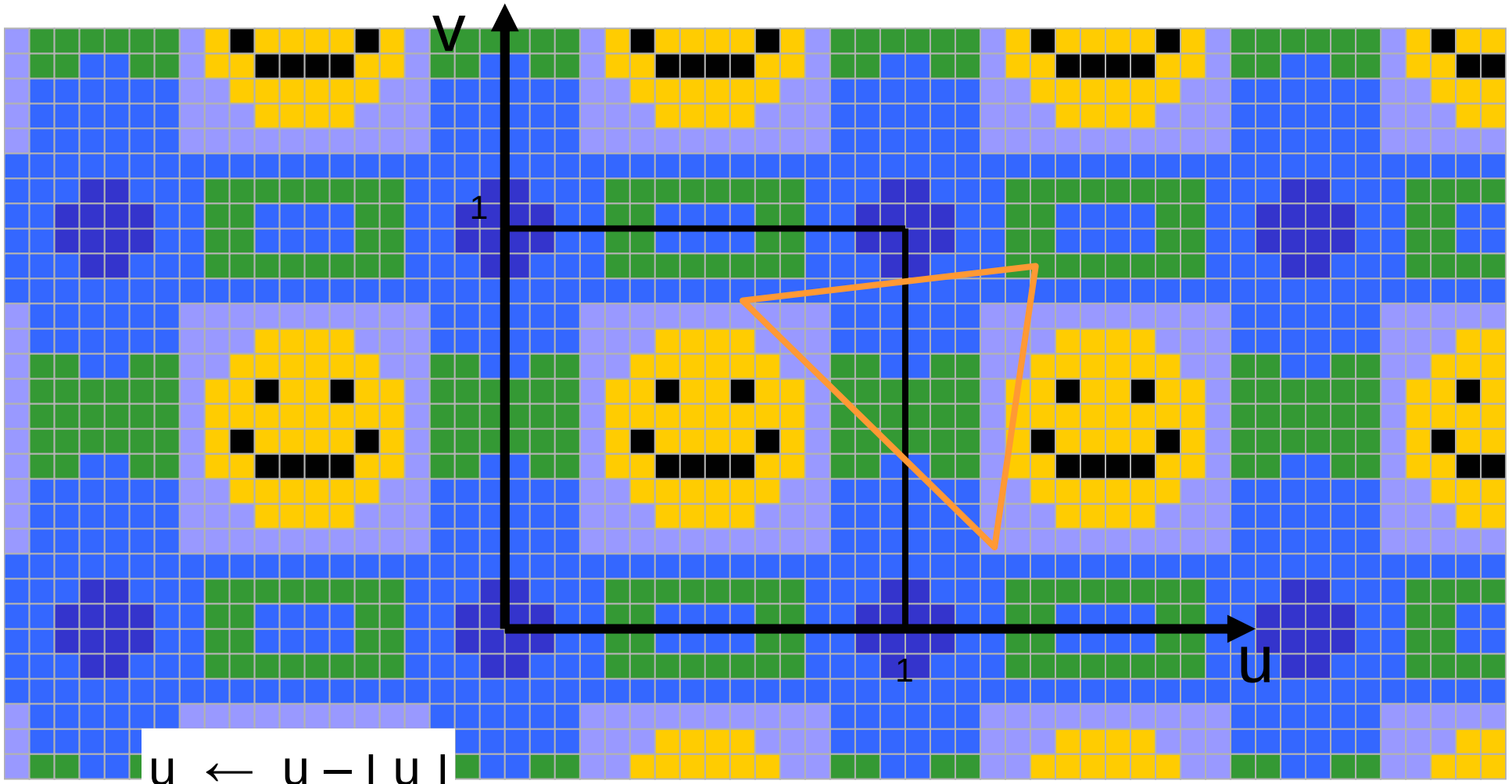


Texture Look-up:
fuori dai bordi: modo "clamp"



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

Texture Look-up:
fuori dai bordi: modo "repeat"



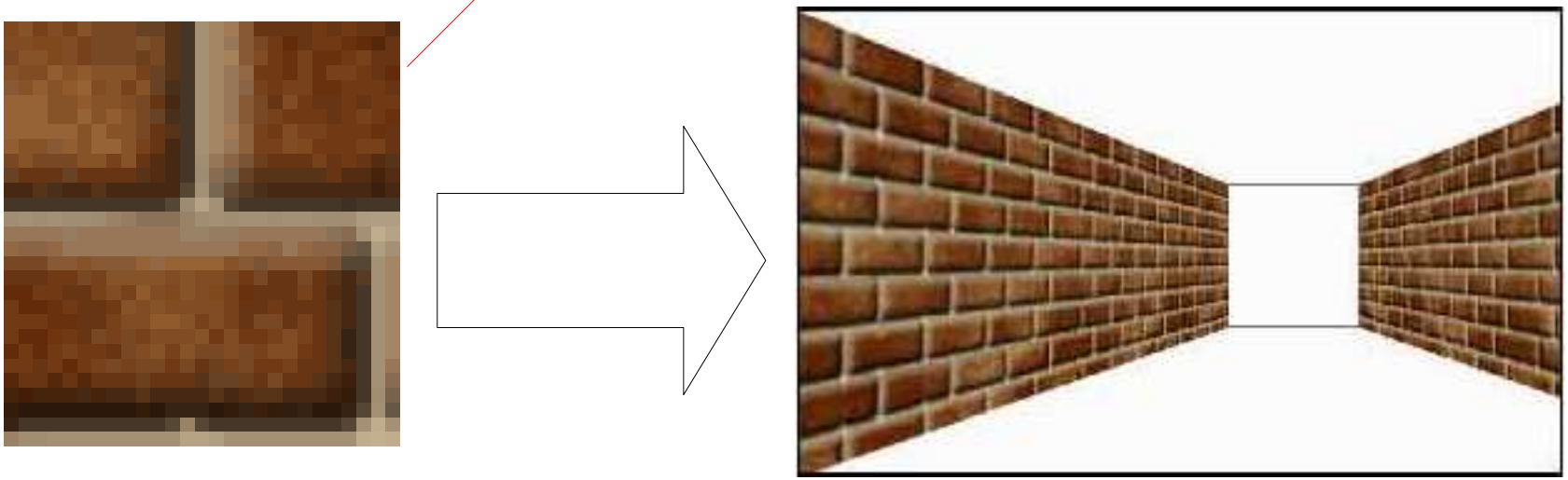
$u \leftarrow u - \lfloor u \rfloor$

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

Tessiture **ripetute**

- Tipico utilizzo:

Nota: deve essere TILEABLE



Molto efficiente in spazio!

In OpenGL

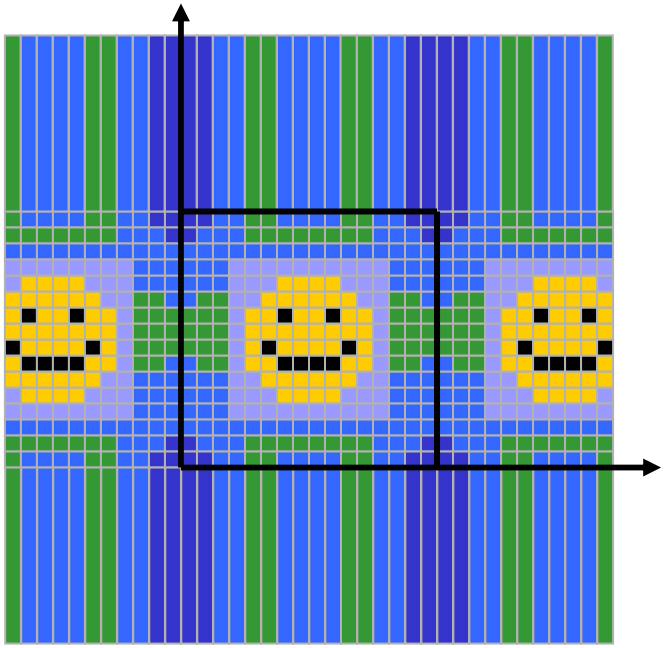
sono parametri della tessitura. Ogni tessitura caricata in memoria ha i propri parametri.

```
glTexParameteri(
   GL_TEXTURE_2D,
   GL_TEXTURE_WRAP_S,
   GL_CLAMP );
```

nota:
setto
per u e v
separatamente

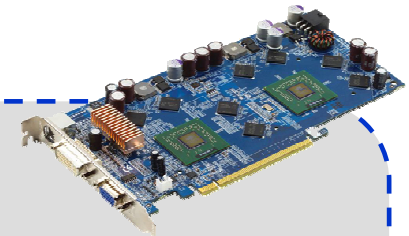
o

```
glTexParameteri(
   GL_TEXTURE_2D,
   GL_TEXTURE_WRAP_S,
   GL_REPEAT );
```



esempio: repeat sulla **u** e clamp sulla **v**

Texture Mapping



Vertici & loro attributi

computazioni per vertice

Vertici poriettati & attributi computati

set-up

rasterizer punti

set-up

rasterizer triangoli

set-up

rasterizer segmenti

Frammenti & attributi interpolati

computazioni per frammento

Screen buffer

texture look-up

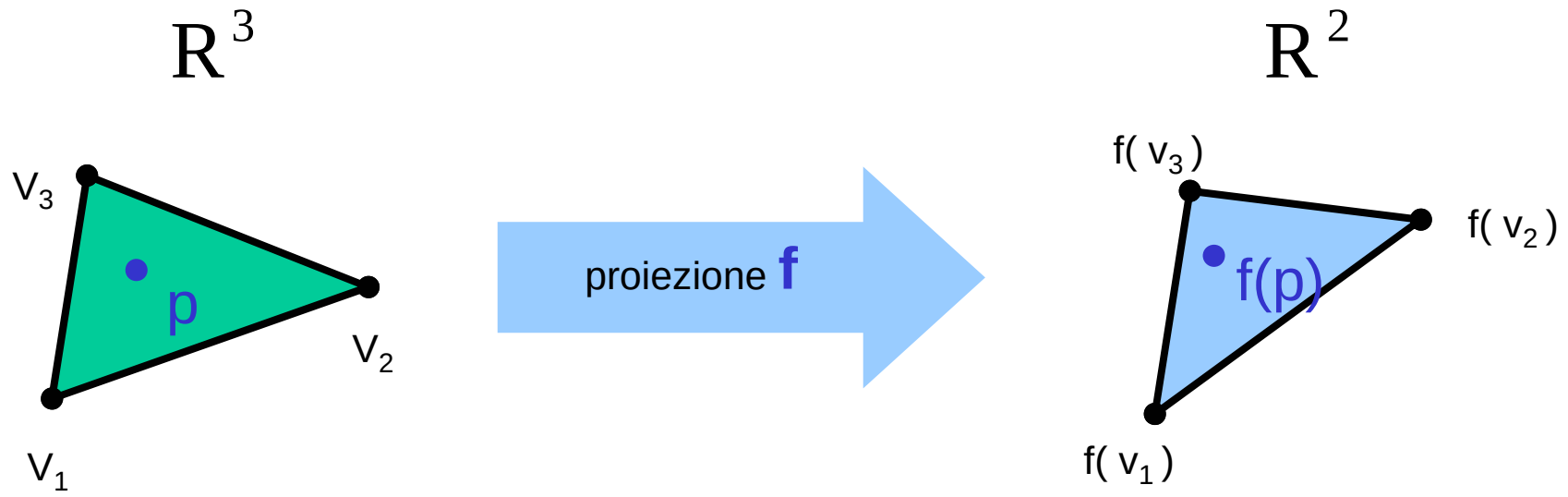
compreso: coordinate texture (per vertice!)

interpolo coordinate texture

coordinate texture interpolate

Texture RAM

Problema: interpolazione lineare coordinate texture



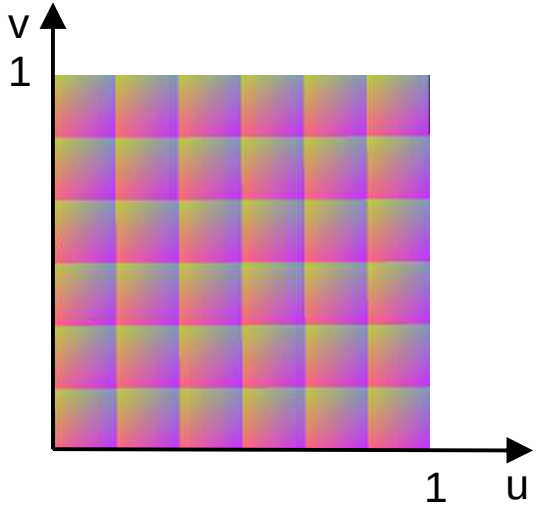
p ha coord. baricentriche a,b,c nel triangolo v_1 v_2 v_3

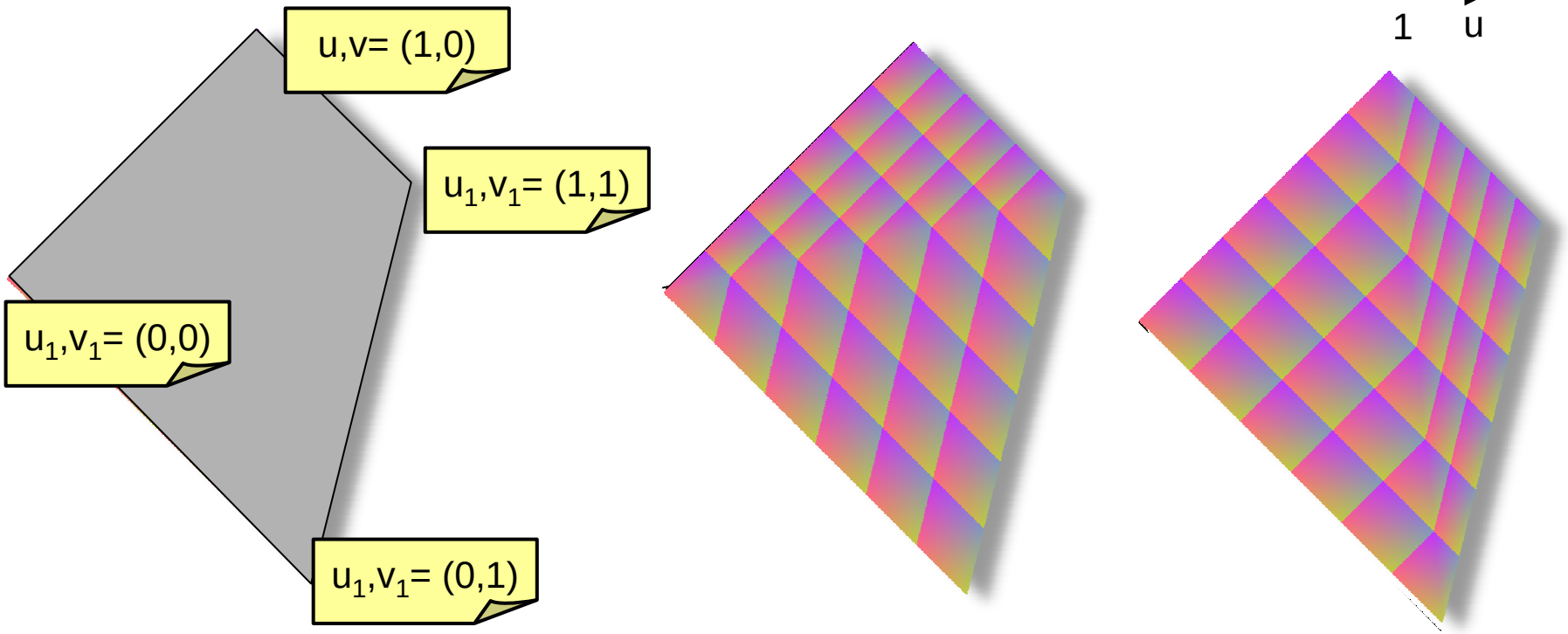
f(p) ha coord. baricentriche a,b,c nel triangolo $f(v_1)$ $f(v_2)$ $f(v_3)$

- Non vale per la proiezione prospettica!
  - era solo una approssimazione
  - andava bene quando interpolavamo colori, normali
  - non va bene quando interpoliamo coordinate texture...

Problema: interpolazione lineare coordinate texture

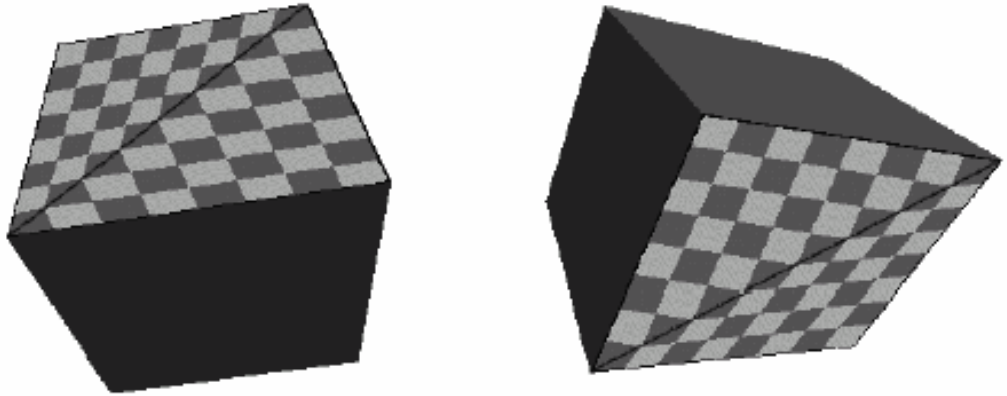
- Esempio:





Problema:
interpolazione lineare coordinate texture

- Esempio:

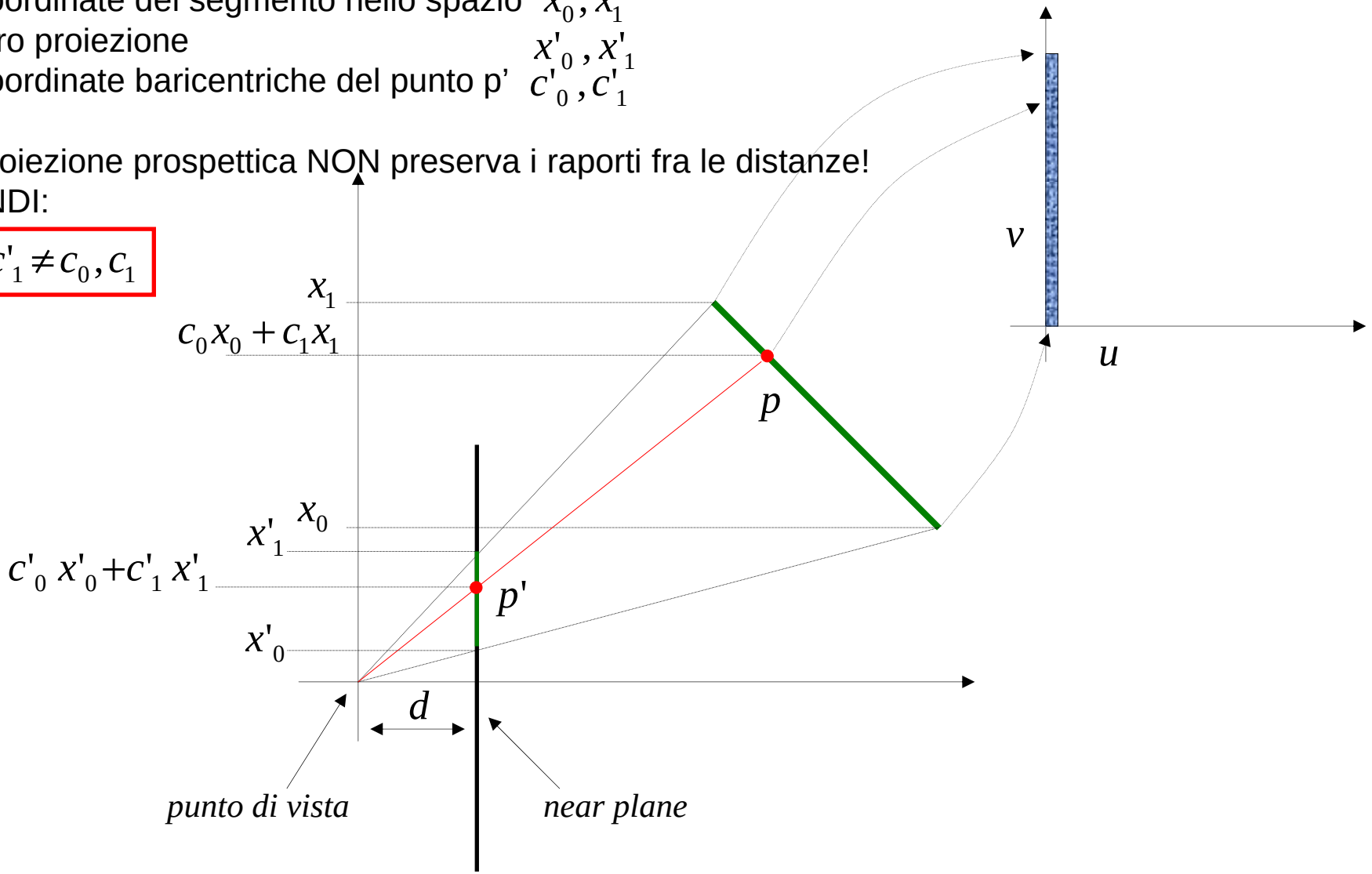


La correzione prospettica

Le coordinate del segmento nello spazio x_0, x_1
La loro proiezione x'_0, x'_1
Le coordinate baricentriche del punto p' c'_0, c'_1

La proiezione prospettica NON preserva i raporti fra le distanze!
QUINDI:

$$c'_0, c'_1 \neq c_0, c_1$$



Qualche passaggio

$$\begin{pmatrix} c'_0\, x'_0 + c'_1\, x'_1 \\ 1 \end{pmatrix} = \begin{pmatrix} c'_0 \dfrac{x_0}{(x_{0_z}/d)} + c'_1 \dfrac{x_1}{(x_{1_z}/d)} \\ 1 \end{pmatrix} = \begin{pmatrix} \dfrac{c'_0}{W_0} x_0 + \dfrac{c'_1}{W_1} x_1 \\ 1 \end{pmatrix} =$$

$$W_0 = x_{0_z}/d, W_1 = x_{1_z}/d$$

$$\begin{pmatrix} a \\ 1 \end{pmatrix} = \begin{pmatrix} aw \\ w \end{pmatrix}, \quad w \neq 0$$

Abbiamo espresso le coordinate c_0, c_1 come funzione delle coordinate c'_0, c'_1

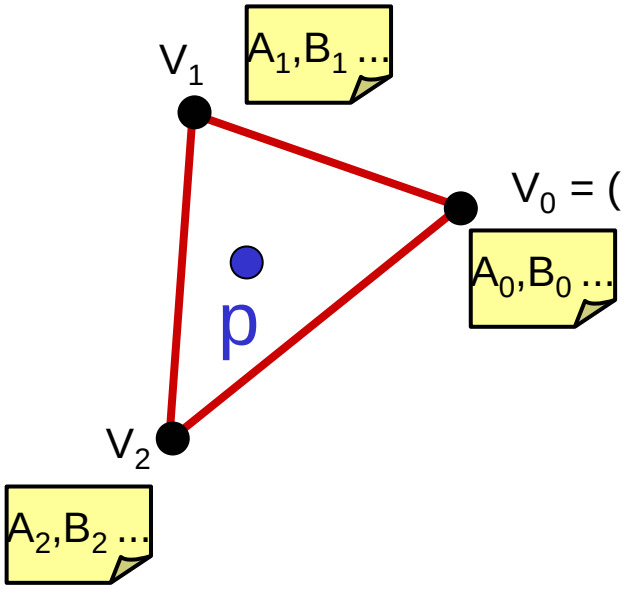
$$= \begin{pmatrix} \dfrac{c'_0}{W_0} x_0 + \dfrac{c'_1}{W_1} x_1 \\ 1 \end{pmatrix} \cdot \frac{1}{\dfrac{c'_0}{W_0} + \dfrac{c'_1}{W_1}} = \begin{pmatrix} \dfrac{\dfrac{c'_0}{W_0} x_0 + \dfrac{c'_1}{W_1} x_1}{\dfrac{c'_0}{W_0} + \dfrac{c'_1}{W_1}} \\ \dfrac{1}{\dfrac{c'_0}{W_0} + \dfrac{c'_1}{W_1}} \end{pmatrix} = \begin{pmatrix} \dfrac{\dfrac{c'_0}{W_0}}{\dfrac{c'_0}{W_0} + \dfrac{c'_1}{W_1}} x_0 + \dfrac{\dfrac{c'_1}{W_1}}{\dfrac{c'_0}{W_0} + \dfrac{c'_1}{W_1}} x_1 \\ \dfrac{1}{\dfrac{c'_0}{W_0} + \dfrac{c'_1}{W_1}} \end{pmatrix}$$

c_0 c_1

Soluzione:
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")

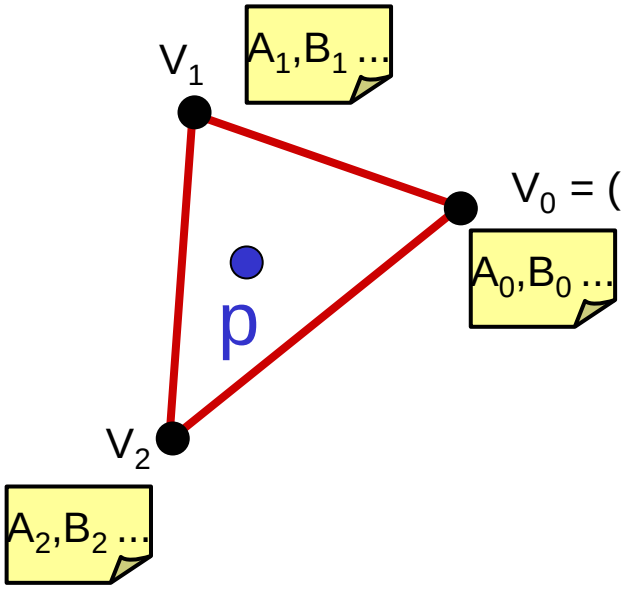
$$A_p = c_0 A_0 + c_1 A_1 + c_2 A_2$$

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

Soluzione: Correzione Prospettica

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

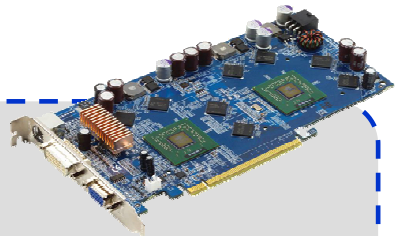
$$p = c_0 v_0 + c_1 v_1 + c_2 v_2$$



attributi di p:
(considerando
la "correzione prospettica")

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

Soluzione:
Correzione Prospettica



attributo originale A

applico transofrm.

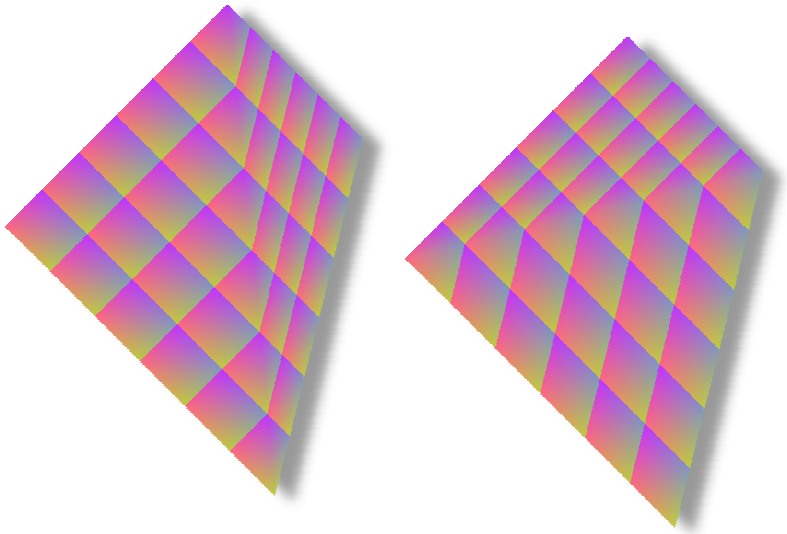
poi calcolo:
A' = A / w
e
w' = 1 / w

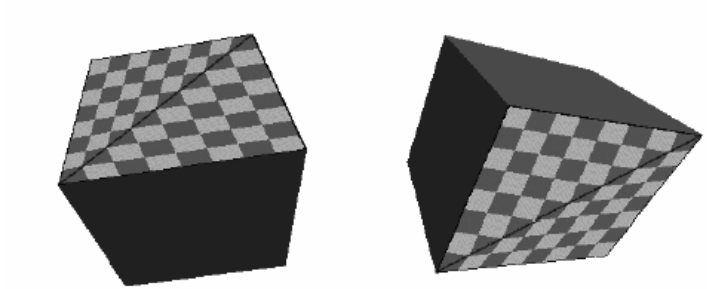
interpolo
A' e w'

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

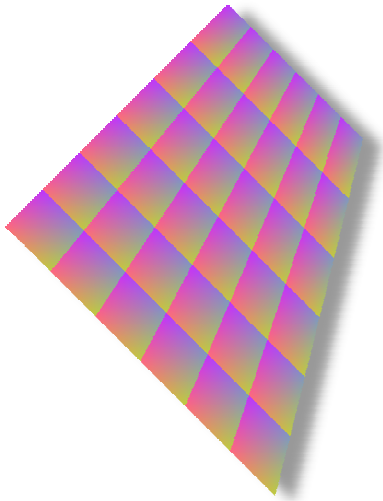
Correzione Prospettica

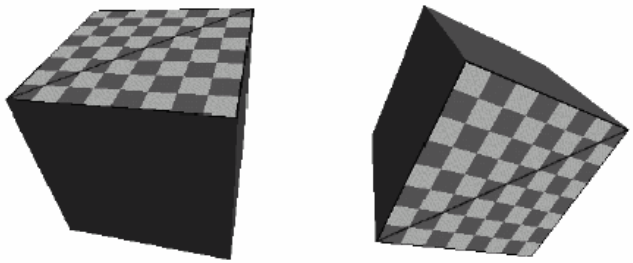
- Senza





- Con

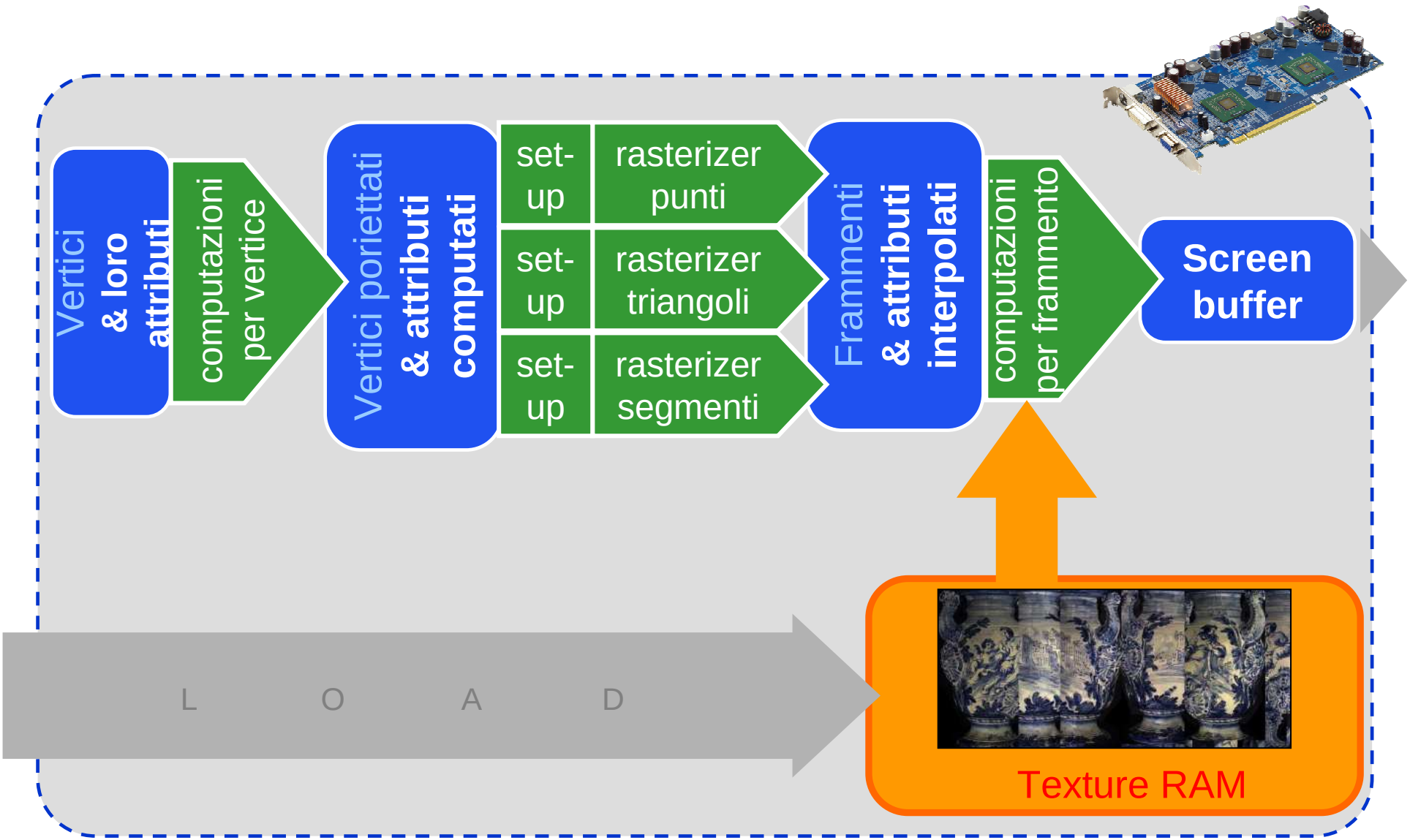




Correzione Prospettica

- Texture mapping con correzione prospettica
  - anche conosciuto come:
    - texture mapping perfetto
    - metodo dei 3 vettori magici *(desueto)*
    - metodo dei 9 numeri magici *(desueto)*

Nota: la tessitura va caricata



Nota: la tessitura va caricata

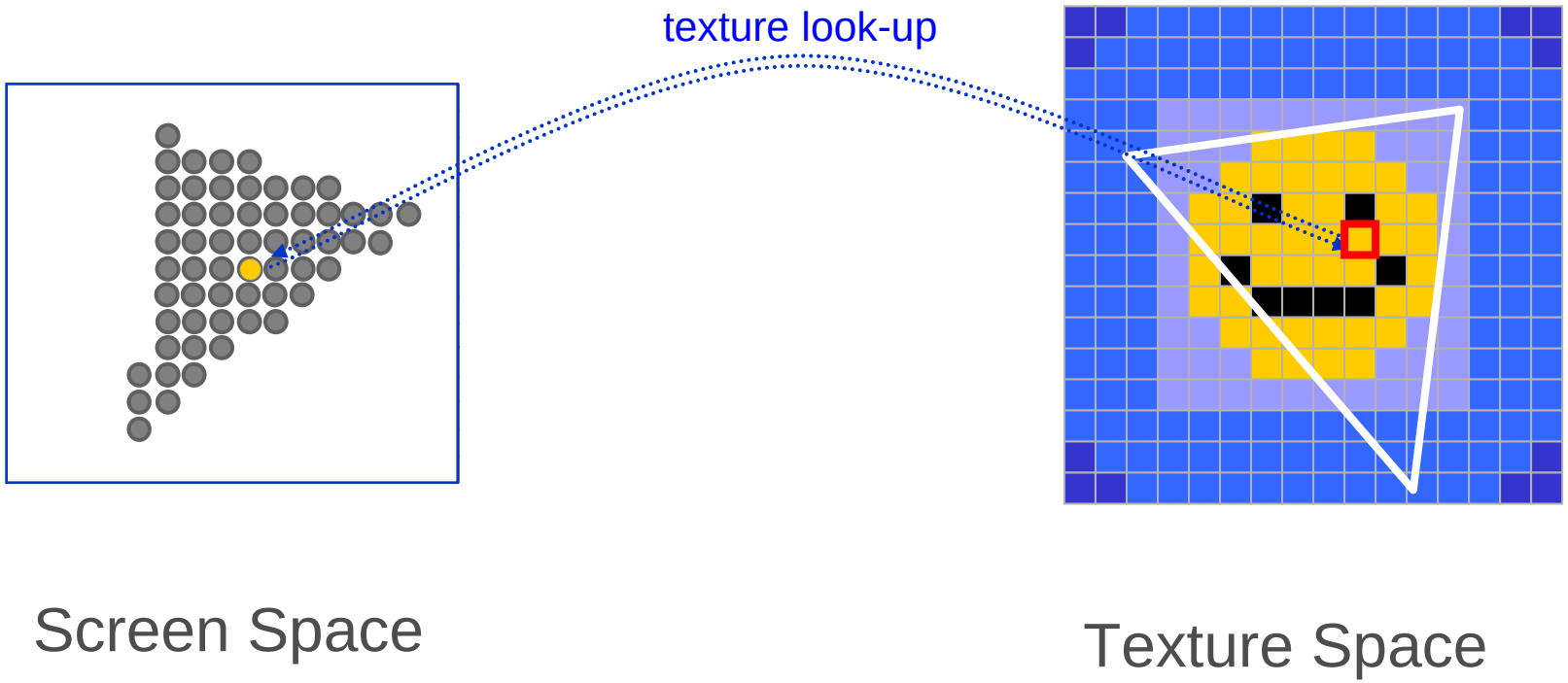
1. Da disco a memoria RAM main
   - (sulla scheda madre)
2. Da memoria RAM main a Texture RAM
   - (on board dell'HW grafico)

Entrambe operazioni piuttosto lente.

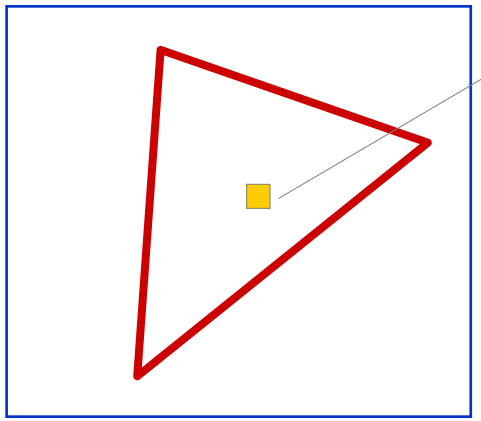
Per es, impossibili da fare una volta per frame!

Texture Look-up

- Un frammento ha coordinate non intere (in texels)

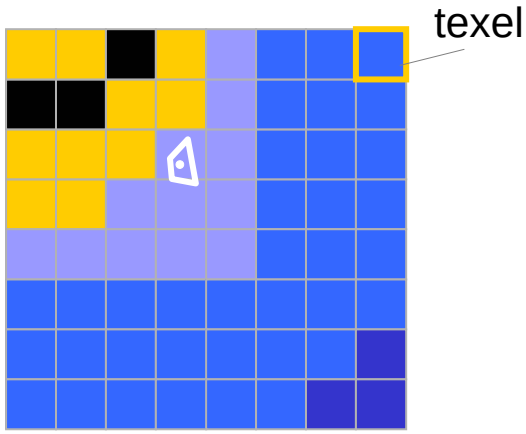


Texture Look-up



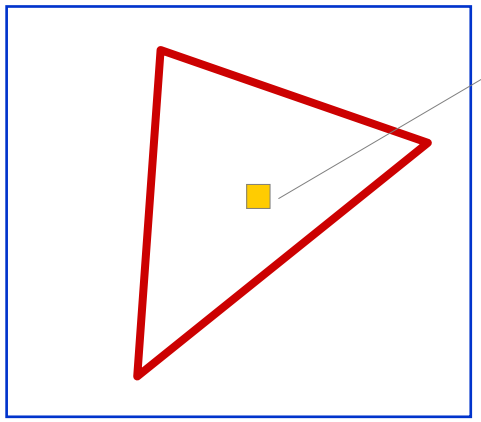
un pixel = meno di un texel

magnification



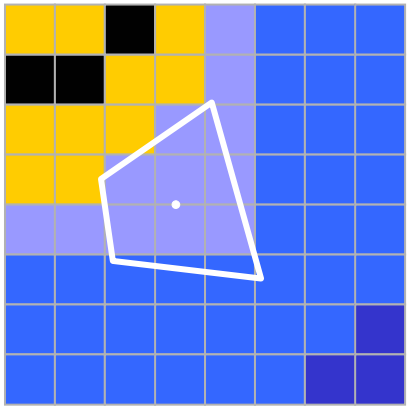
Screen Space

Texture Space



un pixel = più di un texel

minification



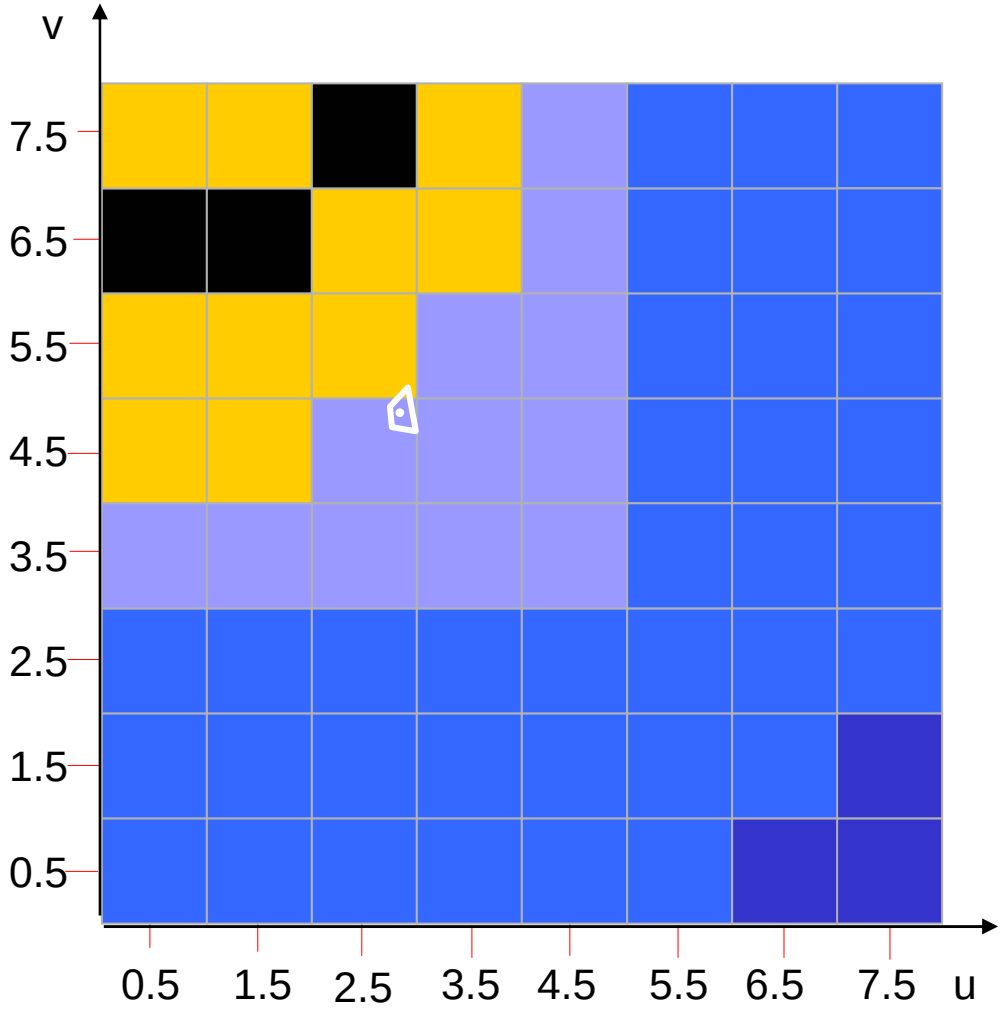
Caso Magnification

Soluzione 1:
prendo il texel in cui casco

(cioè il texel il cui centro
è più vicino alle coordinate u,v
del frammento)

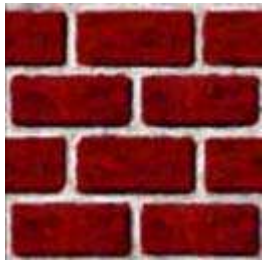
equivale ad arrotondare
le coordinate texel
ad interi

"Nearest Filtering"



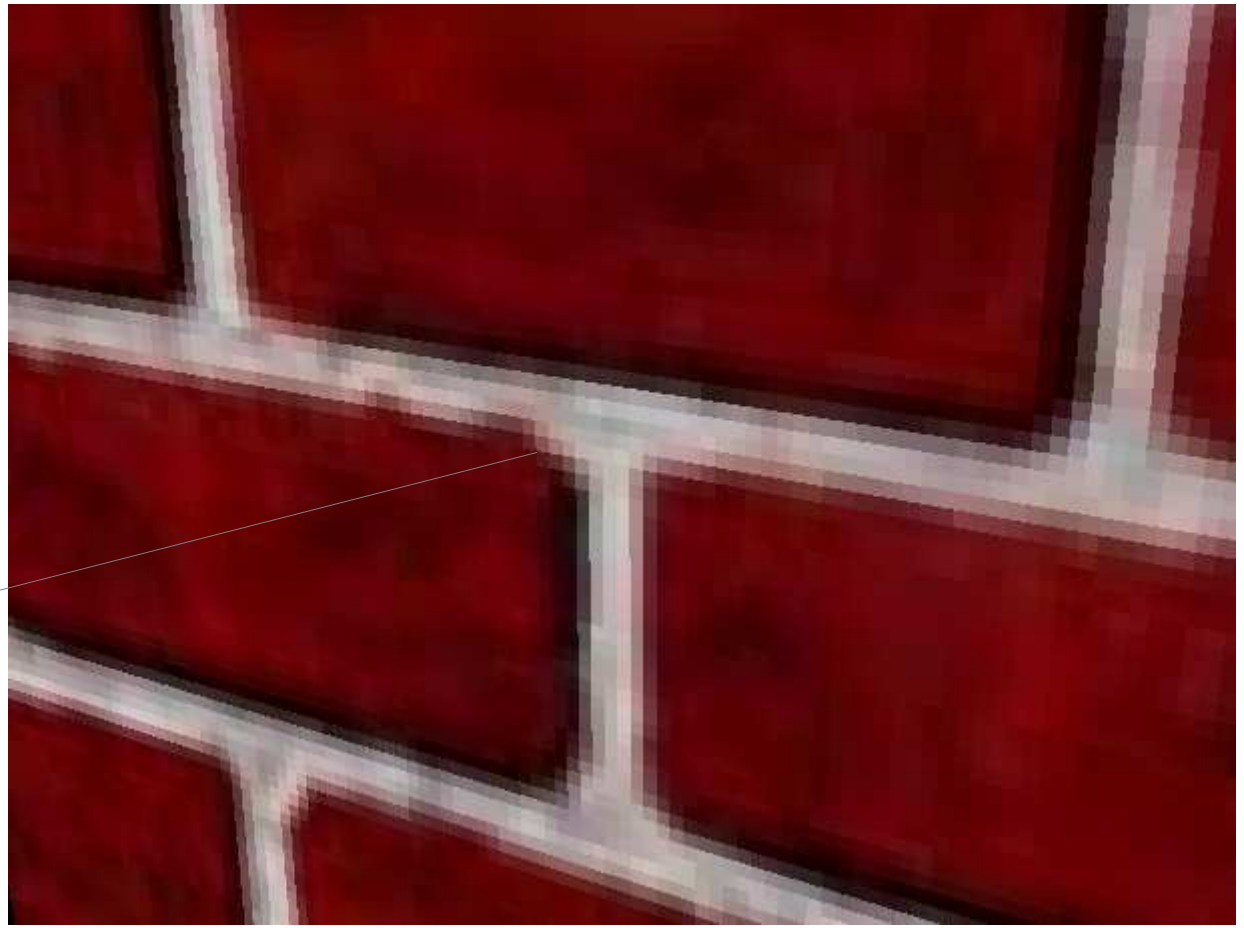
Caso Magnification

Nearest Filtering: risultato visivo



texture 128x128

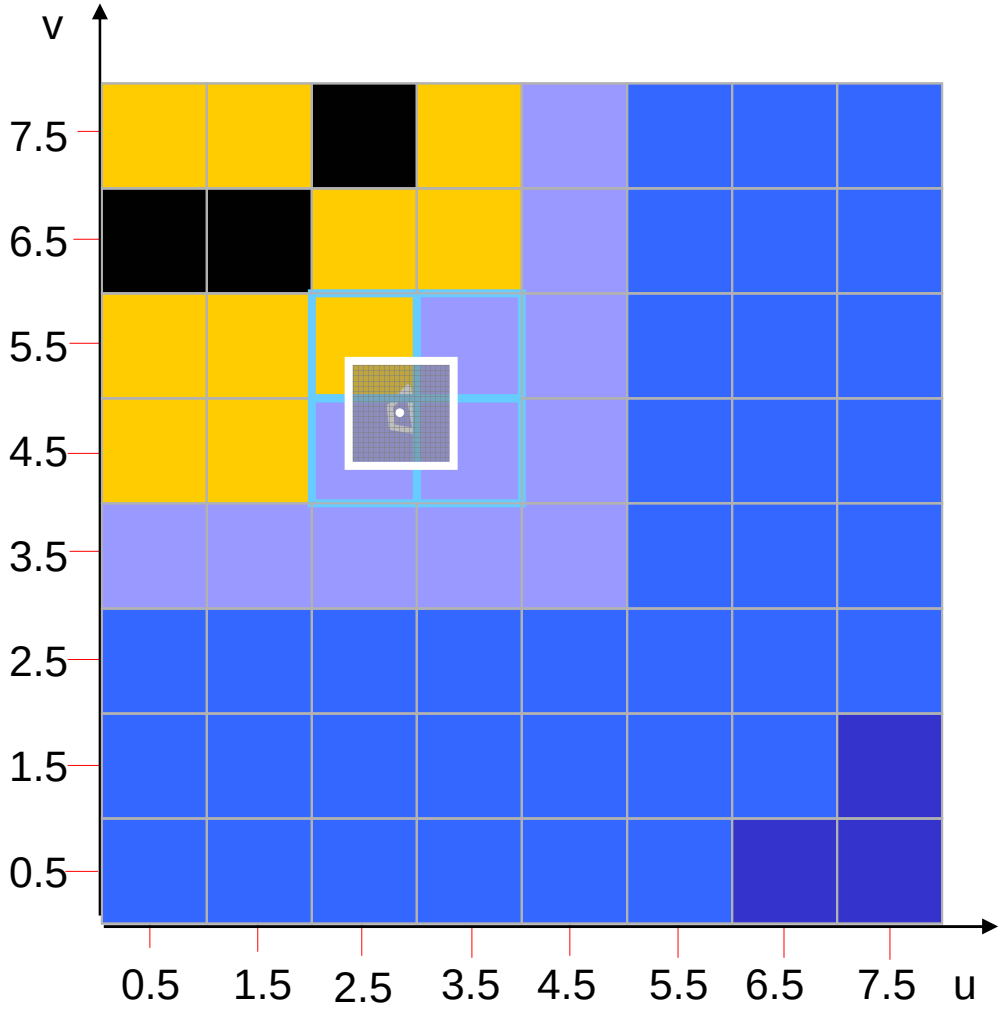
"si vedono i texel !"



Caso Magnification

Soluzione 2:
Medio il valore dei quattro texel più vicini

Ripasso:
interpolazione lineare (1D)
interpolazione bi-lineare (2D)



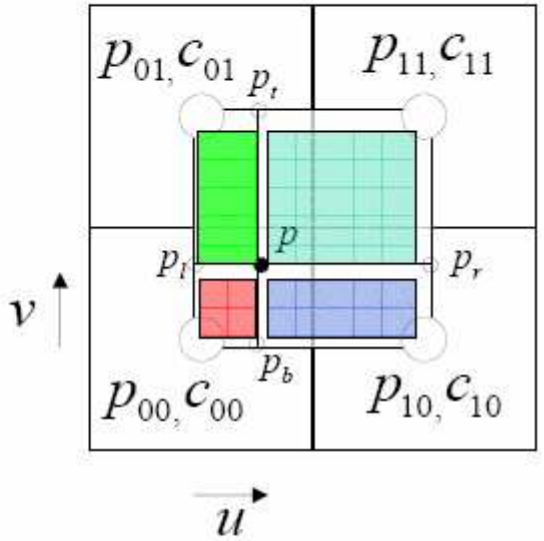
$$p_b = (1-u)\,p_{00} + u\,p_{10}$$

$$p_t = (1-u)\,p_{01} + u\,p_{11}$$

$$p_l = (1-v)\,p_{00} + v\,p_{01}$$

$$p_r = (1-v)\,p_{10} + v\,p_{11}$$

$$p = (1-u)\,p_l + u\,p_r$$



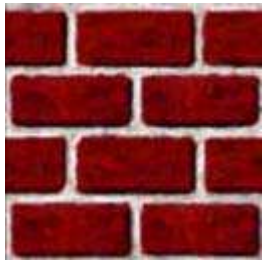
oppure $p = (1-v)\,p_b + v\,p_t$

In entrambi i casi:

$$p = (1-u)(1-v)\,p_{00} + (1-u)v\,p_{01} + (1-v)u\,p_{10} + uv\,p_{11}$$

Caso Magnification

Bilinear Interpolation: risultato visivo



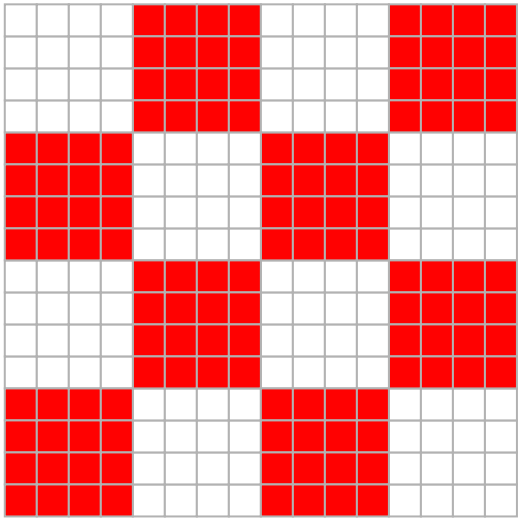
texture 128x128

Caso 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 "sfocato"

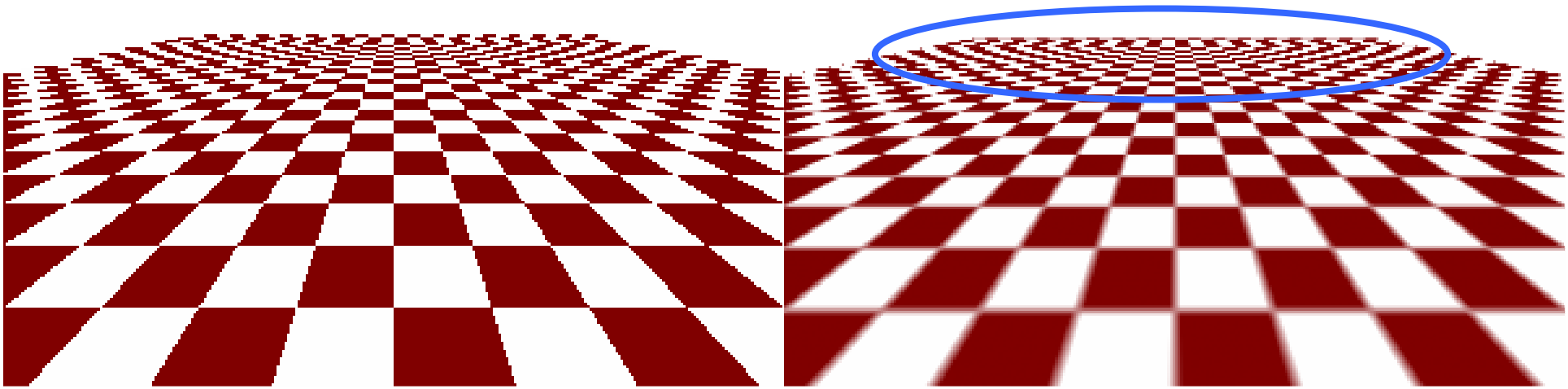
Caso Minification



Nearest Filtering

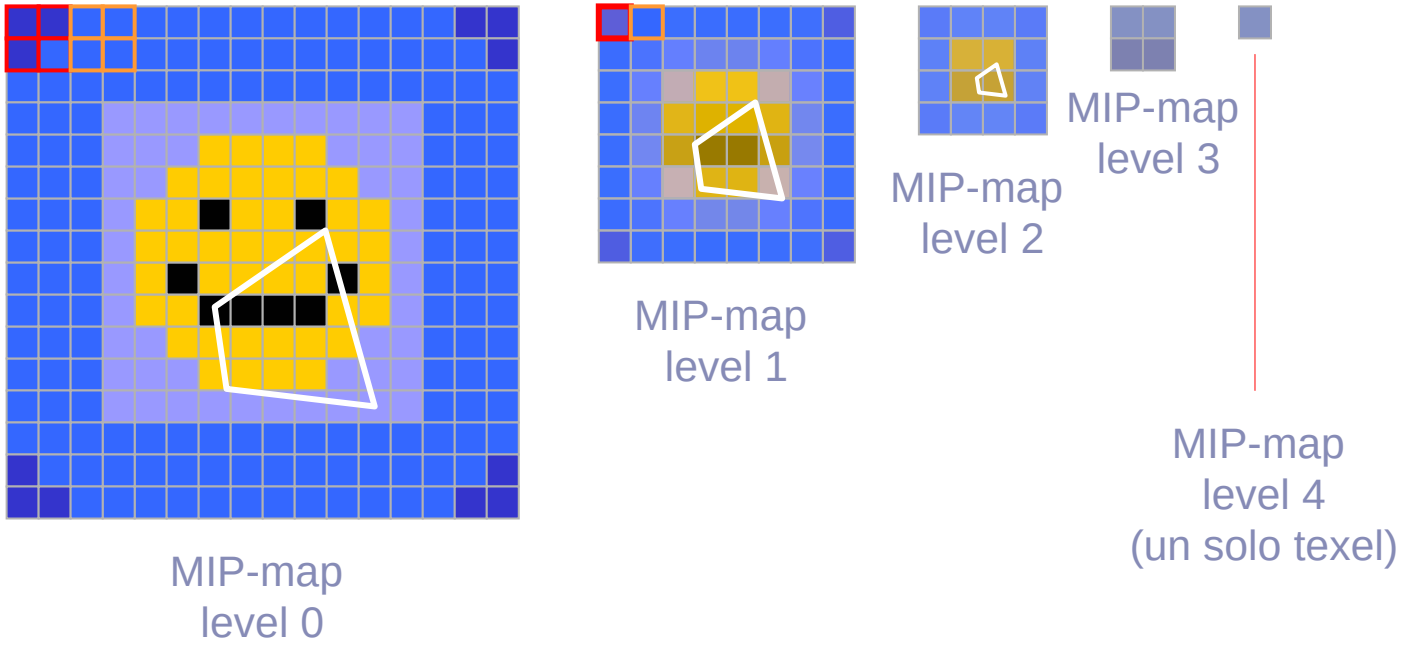
Bilinear interpolation

non risolve il problema



Caso Minification: MIP-mapping

MIP-mapping: "Multum In Parvo"

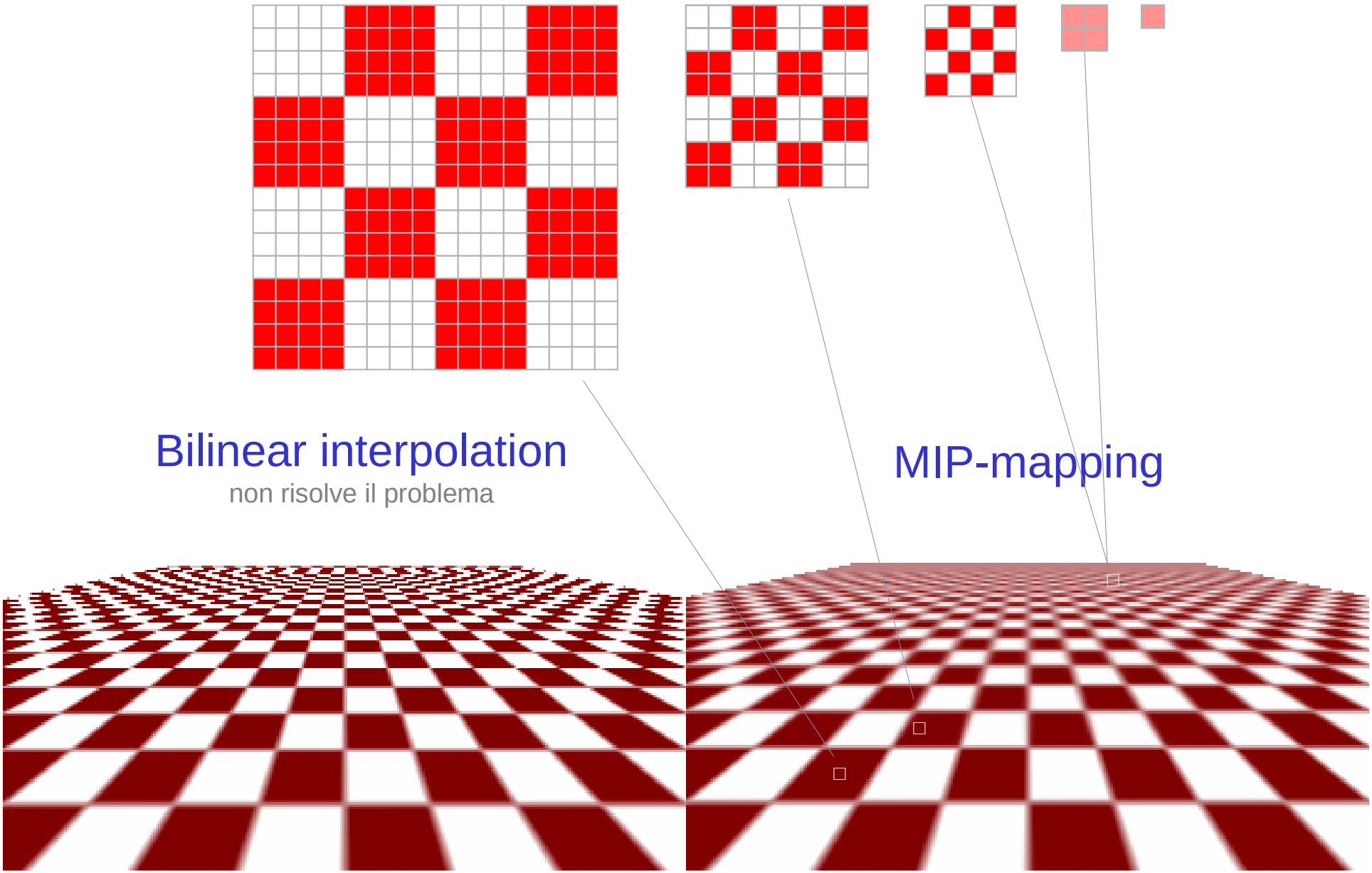


Mipmap Math

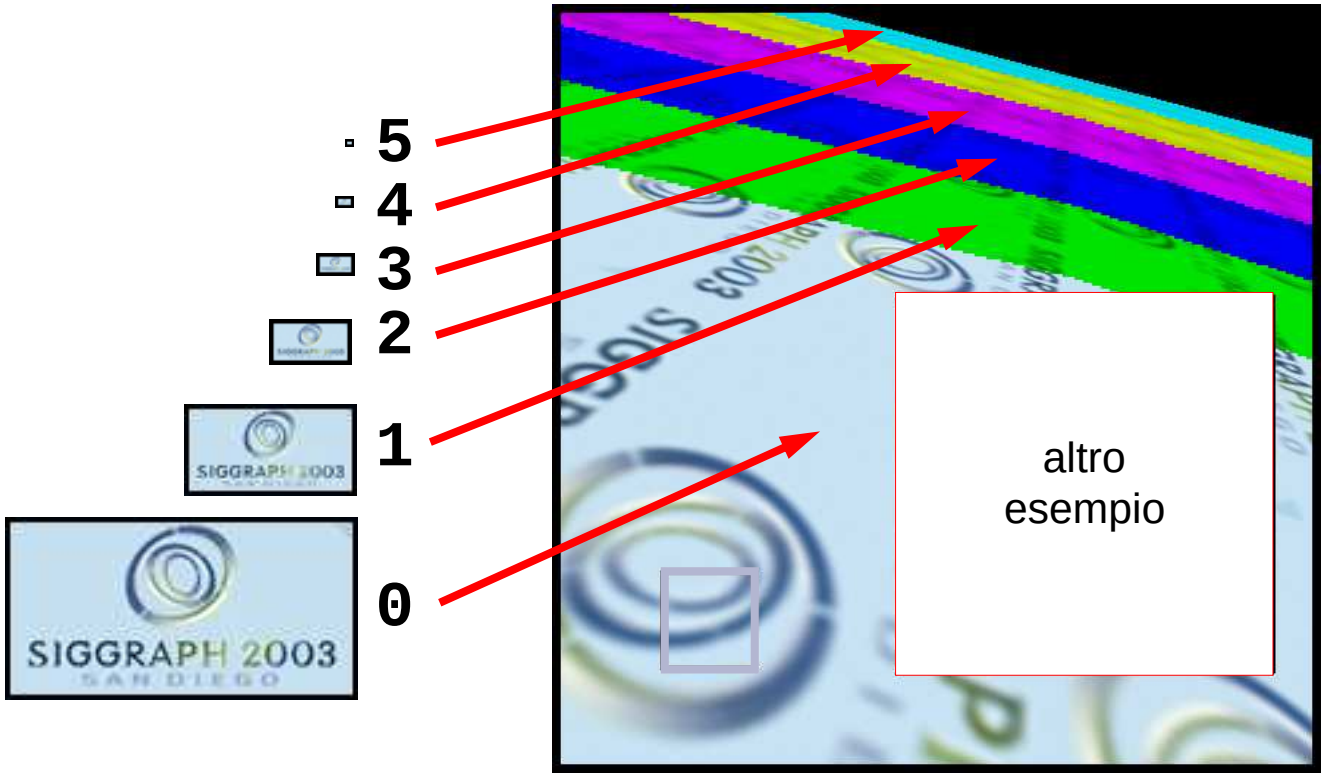
- Definiamo un fattore di scala, ρ=texels/pixel
  - ρ è il massimo fra ρ_x e ρ_y
  - può variare entro lo stesso triangolo
  - può essere derivato dalle matrici di trasformazione
  - e calcolato nei vertici, interpolato nei frammenti

- il livello di mipmap da utilizzare è: $\log_2 \rho$
  - livello 0 = massima risoluzione
  - se livello<0 cosa significa?
  - nota: il livello non è necess. un numero intero

Caso Minification: MIP-mapping



Caso Minification: MIP-mapping



In OpenGL

decidere che tipo di magnificazione usare:

```
glTexParameteri(
  GL_TEXTURE_2D,
  GL_TEXTURE_MAG_FILTER,
  GL_NEAREST);
```

oppure

```
glTexParameteri(
  GL_TEXTURE_2D,
  GL_TEXTURE_MAG_FILTER,
  GL_LINEAR );
```

In OpenGL

decidere che tipo di minifacazione usare:

```
glTexParameteri(
  GL_TEXTURE_2D,
  GL_TEXTURE_MIN_FILTER,
  modo );
```

dove

```
modo = GL_NEAREST
       GL_LINEAR
       GL_NEAREST_MIPMAP_NEAREST
       GL_LINEAR_MIPMAP_NEAREST
       GL_NEAREST_MIPMAP_LINEAR
       GL_LINEAR_MIPMAP_LINEAR
```

interpolazione trilineare

In OpenGL

- Caricare sulla scheda
  i vari livelli di mipmapping.
  - uno per uno:

```
glTexImage2D (
      GL_TEXTURE_2D,
      i,          // MIP-map level
      GL_RGB,  // formato interno
      imageWidth, imageHeight,
      0,        // bordo
      GL_RGB, // formato nella RAM
      GL_UNSIGNED_BYTE,
      imageData);
```

In OpenGL

- Caricare sulla scheda
  i vari livelli di mipmapping.
  - tutti insieme (con la liberira glu):

```
gluBuild2DMipmaps (
      GL_TEXTURE_2D,
      GL_RGB,  // formato interno
      imageWidth, imageHeight,
      0,       // bordo
      GL_RGB, // formato nella RAM
      GL_UNSIGNED_BYTE,
      imageData);
```

Come si assegnano le coordinate texture ai vertici?

- Ovvero: come si definisce la corrispondenza tra vertice e texture space?
- Due casi:
  - La coordinata texture è una proprietà “costante” “statica” del vertice.
    - Si calcola una volta e poi è quella
  - La coordianata texture viene assegnata al vertice ad ogni frame, o comunque non è costante
- Fino ad ora abbiamo implicitamente assunto il primo caso

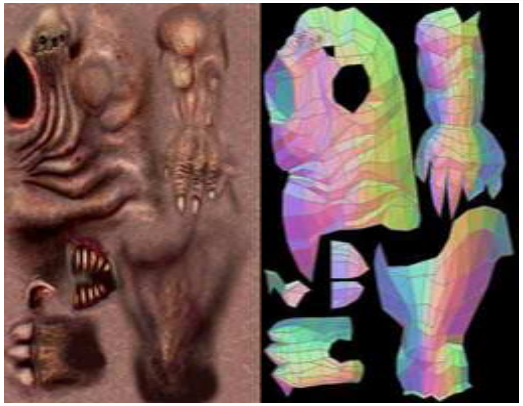
In OpenGL

- Come ogni altro attributo

```
glTexCoord2d( u,v )
```

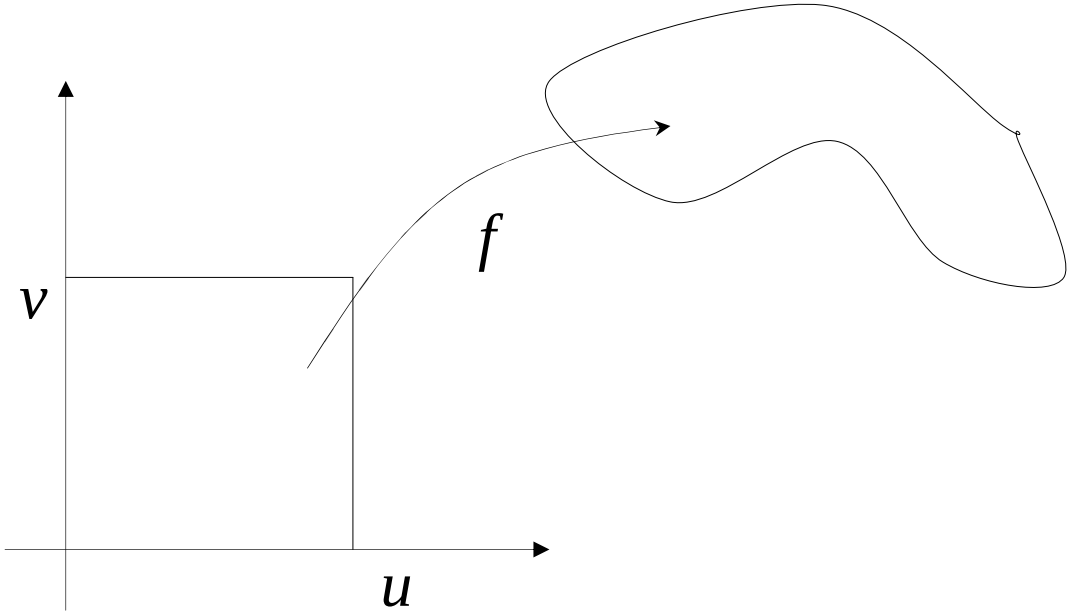
Coordinata texture come attributo statico

- Nella modellazione CAM sono calcolate a mano o comunque in maniera semiautomatica
- Si colora il modello in 3D e poi si costruiscono le texture prendendo la superficie e “stirandola” su un quadrato [0,1]x[0,1]
- Ogni vertice finisce nella sua coordinata texture
- Quando è difficile fare lo “stiramento” ?
- abbastanza….



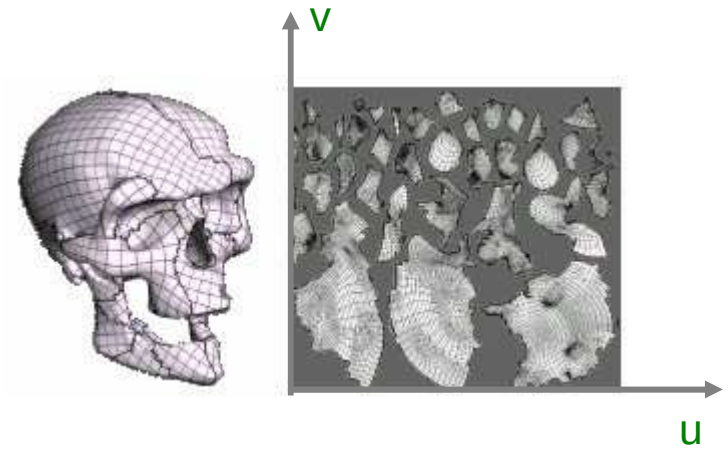
UV Mapping

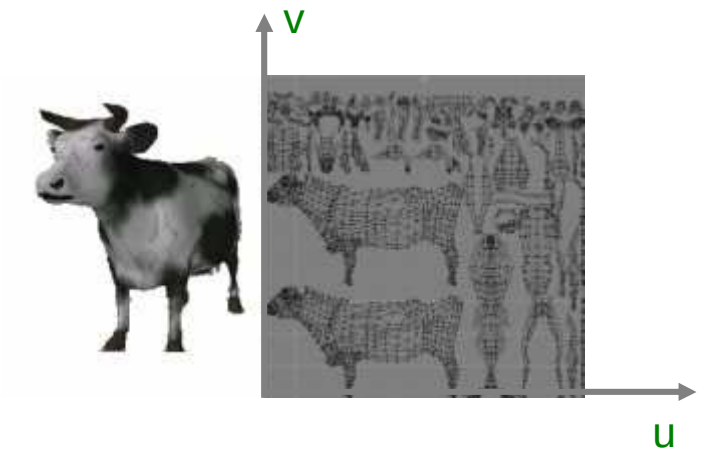
- UV Mapping, o **parametrization:** trovare una funzione cha mappa [0,1]x[0,1] nella superficie
- Esiste sempre?..NO
- Se esiste è unica? ..NO
- Ce ne è una migliore di tutte?..NO, dipende dalla metrica
- e una volta stabilita la metrica?..In generale NO, ci sono minimi locali



ATLASing

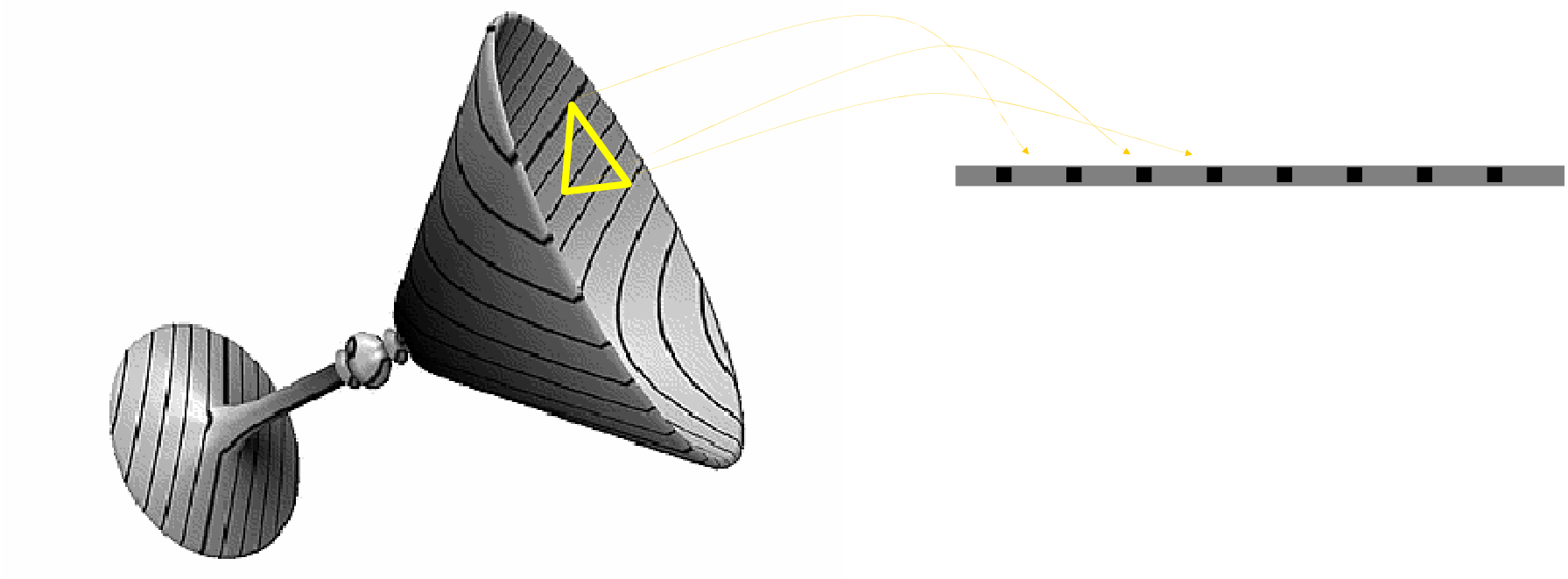
- Costruire un atlas: partizionare la superficie in modo che le singole parti siano facilmente parametrizzabili





Coordinate texture generate automaticamente

- Si possono generare le coordinate texture come funzione delle coordinate dei vertici e delle matrici di trasformazione



Enviroment mapping

- Idea di base: simulare la riflessione dell’ambiente su un oggetto per mezzo di una texture
- Faccio in modo che il colore risultante su ogni punto dell’oggetto sia quello dell’ambiente riflesso su quel punto

Sphere Environment Mapping 1/3

- Il problema: guardando un oggetto di superficie speculare ci voglio vedere il mondo (environment) riflesso
- L'idea: memorizzo in una texture come sarebbe il mondo riflesso su una semisfera perfettamente speculare
- La texture sarà tipo questa:

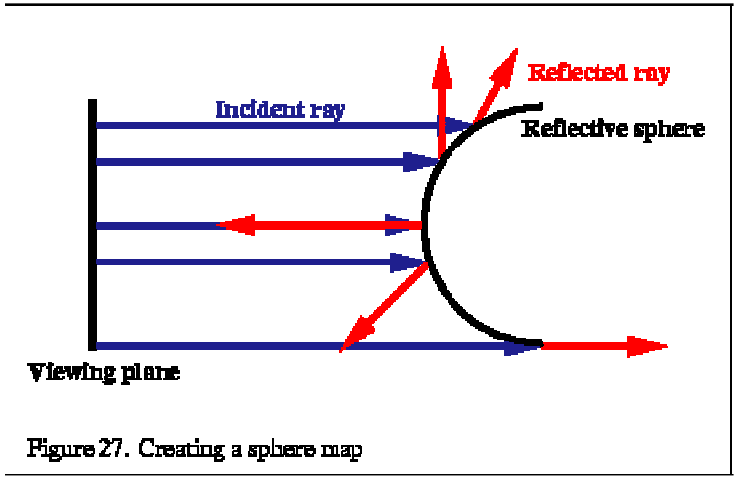
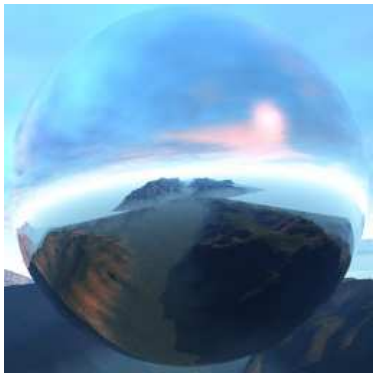


Figure 27. Creating a sphere map

Environment mapping: sferico

simula oggetto a specchio che riflette uno sfondo lontano




simula un materiale complesso
(condizioni di luce fisse)

Sphere Environment Mapping 2/3

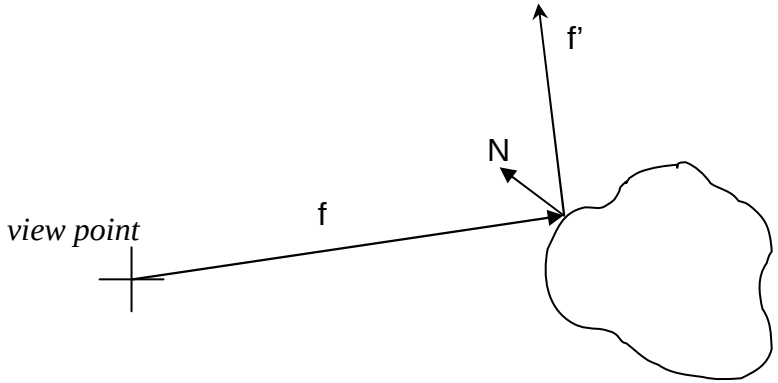
- **a tempo di rendering, le coordinate texture vengono generate in base alla formula:**

$$f' = -f + 2(N \cdot f)N$$

$$p = 2\sqrt{f'_x{}^2 + f'_y{}^2 + (f'_z + 1)^2}$$

$$t = f'_x / p + 0.5$$

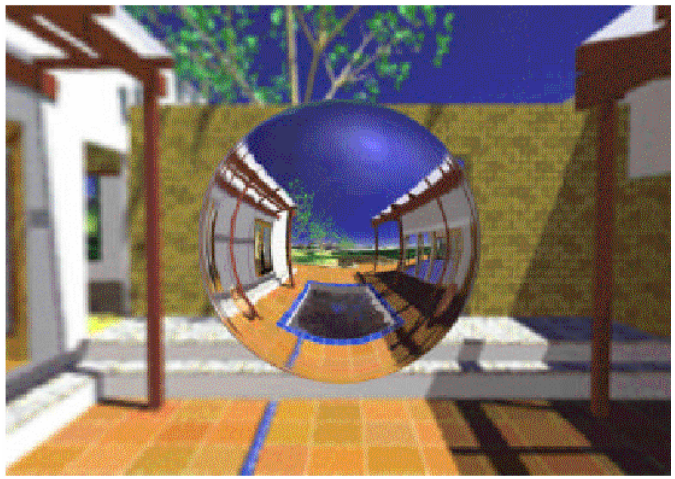
$$s = f'_y / p + 0.5$$



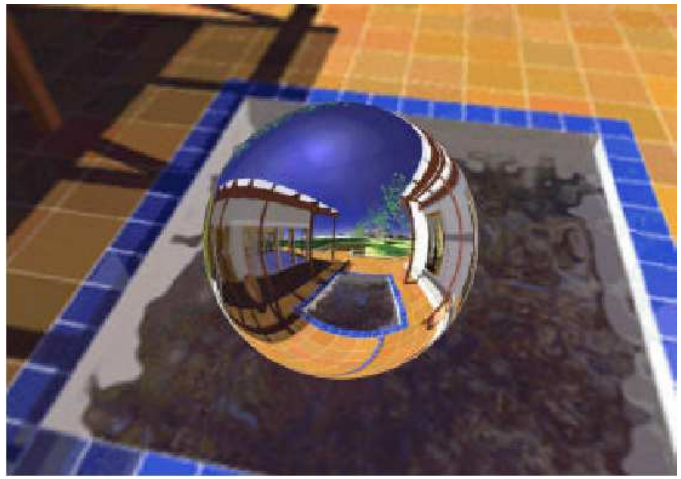
Cosa fa?
Mappa tutti le direzioni “riflesse” nel cerchio di texture

Sphere Env.Mapping.: problemi

- È view dependent: la mappa viene costruita da un punto di vista preciso ed è corretta solo per quello



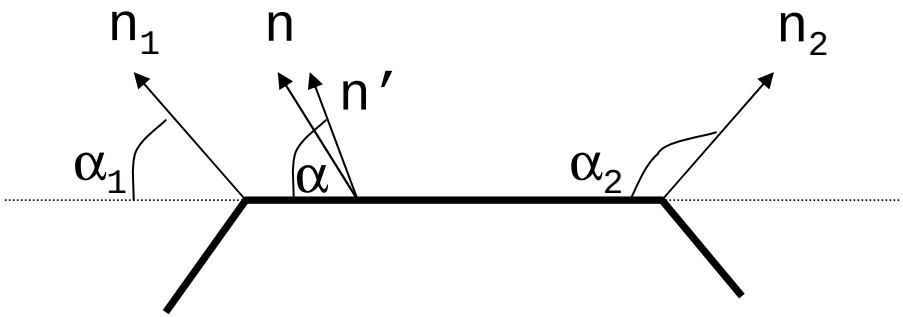
Corretto



Errato: vedo la piscina e il riflesso della piscina

Sphere Env.Mapping.: problemi

- interpolazione:
  - l'interpolazione delle normali è lineare ma la sphere map non è uno spazio lineare

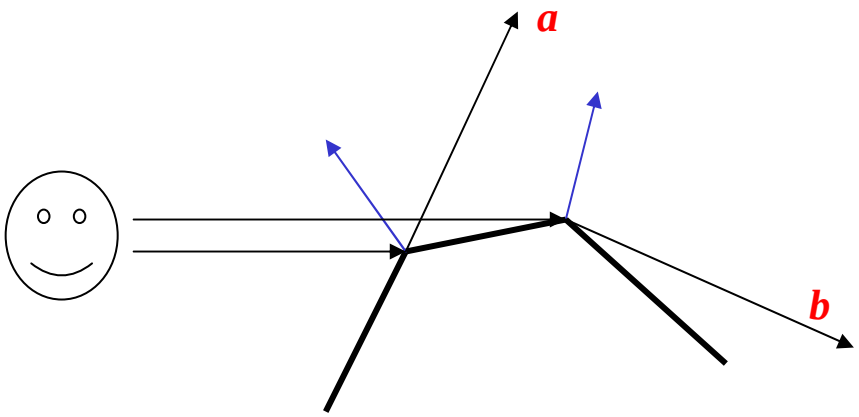


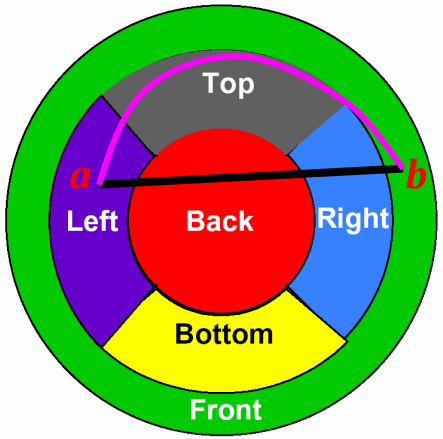
$n = n_1 (1-\lambda)+n_2 \lambda$:come è

$\alpha = \alpha_1 (1-\lambda)+\alpha_2 \lambda$:come dovrebbe essere

Sphere Env.Mapping.: problemi

- Sul bordo di un oggetto puo’ capitare che coordinate calcolate per I vertici di un triangolo corrispondano a punti molto distanti nella texture

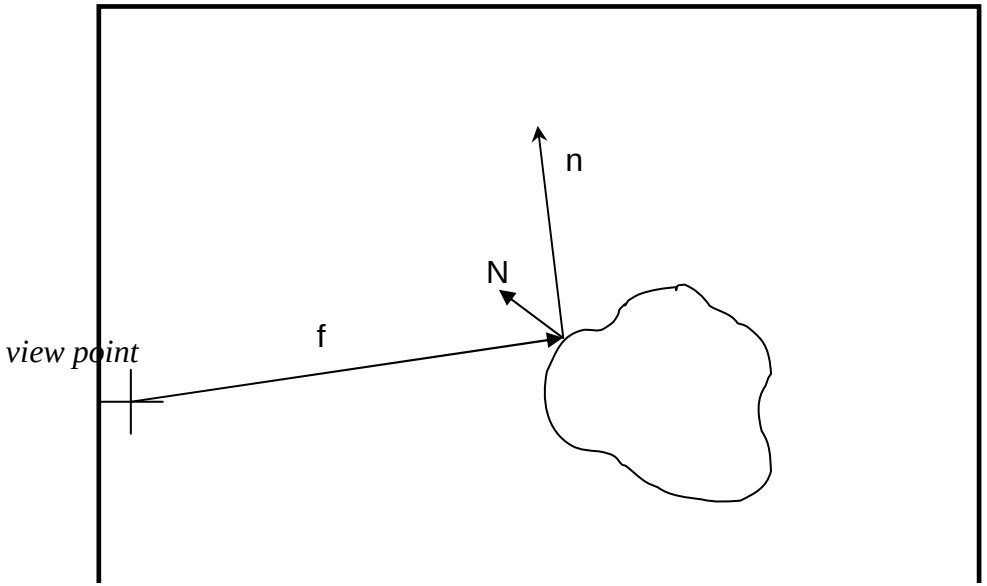




Cube Maps

- Stesso principio, però
  - più semplice
  - più robusta
  - view independent
  - meno distorsione
- L'idea: metto un cubo di grandezza infinita intorno alla scena. Ogni faccia è l'environment in quella direzione. Per il resto procedo come per lo sphere mapping

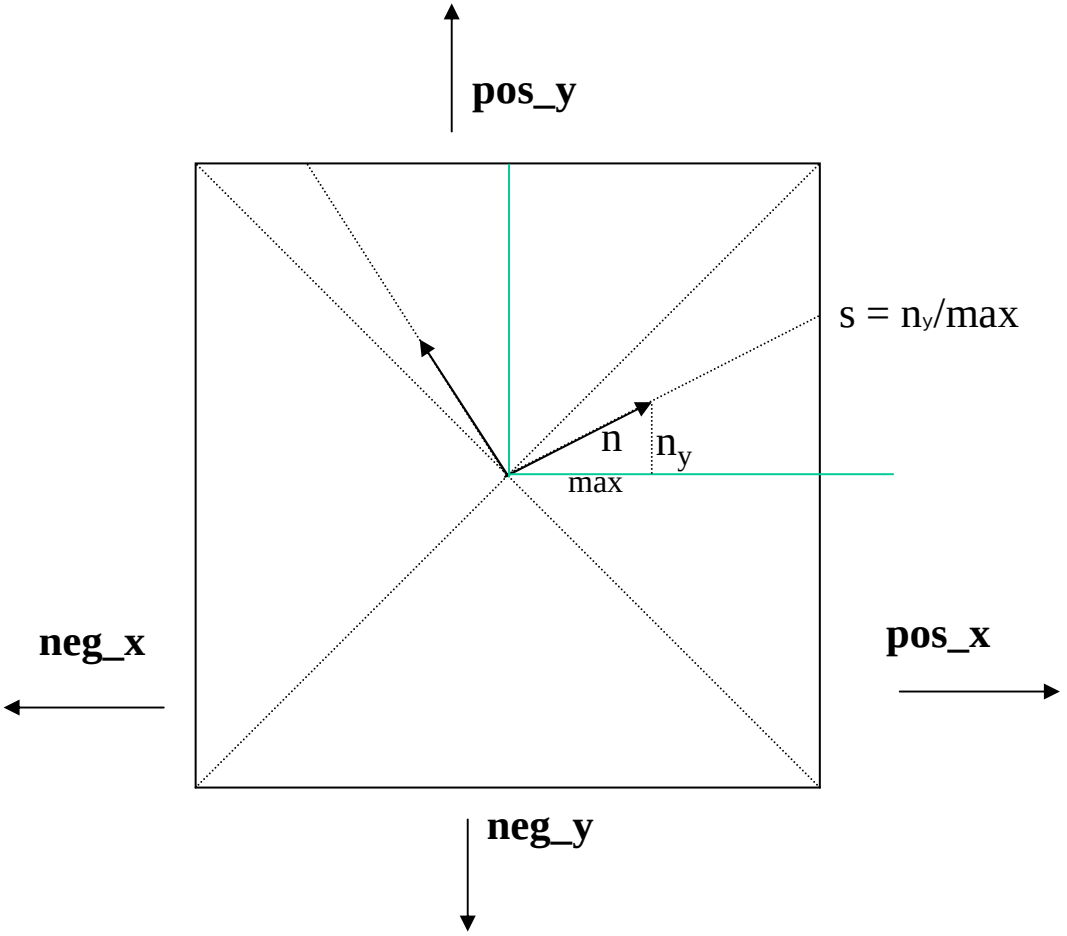
Come si fa?



Cube Maps

- Si usano 6 textures, una per ogni faccia del cubo (+X,-X,+Y,-Y,+Z,-Z):
- Ogni texture è il mondo fotografato dal centro del cubo nella direzione corrispondente

Cube Maps: accesso alle textures



La texture viene scelta in base alla componente di massimo valore assoluto *max*

le coordinate texure sono:
s = (n1 / max+1)/2
t = (n2 / max +1)/2
dove n1 e n2 sono le altre due componenti di n

.....e cosa è *n*?

Cube Maps: accesso alle textures

- n è una tripla di coordinate texture (S,T,R) associate ad ogni vertice
- potreste specificarle voi con `glCoord3f(nx,ny,nz)` e la cube map funzionerebbe
- Noi però le vogliamo **dipendenti dalla normale** alle superficie in *eye space*
- Occorre farle generare automaticamente

Creazione Automatica Coordinate Texture

1- abilitarla:

S, T, R, Q

```
glEnable(GL_TEXTURE_GEN_S);
```

2- scegliere la modalita':

```
glTexGeni(GL_S , GL_TEXTURE_GEN_MODE , mode )
```

mode =

GL_OBJECT_LINEAR	computa le coord texture dalle pos nel sist di rif oggetto
GL_EYE_LINEAR	computa le coord texture dalle pos nel sist di rif camera
GL_SPHERE_MAP	computa come coord texture il raggio di vista riflesso e mappato sulle sphere map
GL_NORMAL_MAP	computa come coord texture la normale
GL_REFLECTION_MAP	computa come coord texture la direzione riflessa normale

Creazione Automatica Coordinate Texture

3- scegliere il piano

S, T, R, Q

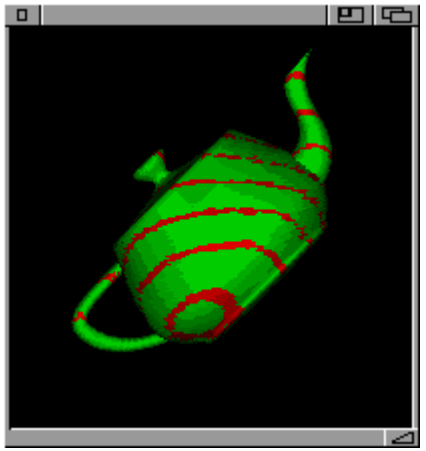
```
glTexGenfv(GL_S, GL_EYE_PLANE , v);
```

EYE o **OBJECT**

vettore di 4 elementi

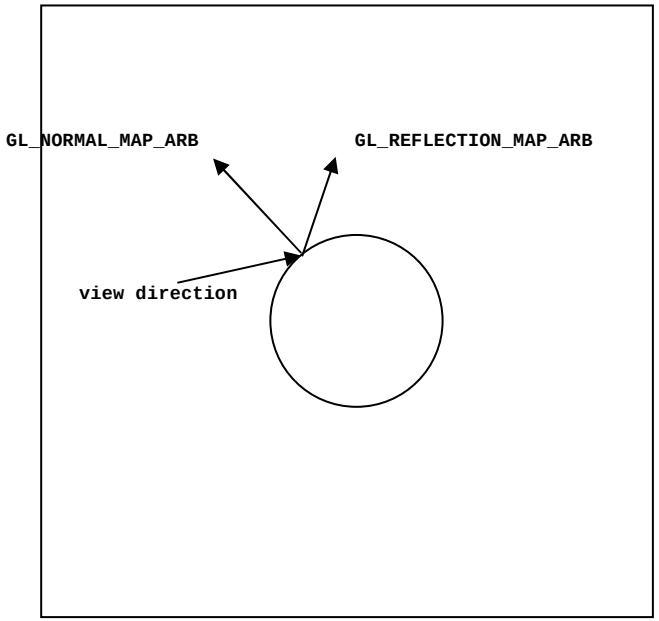
coordinata texture prodotta = v^T • pos_vertice

(e' distanza da un piano!)



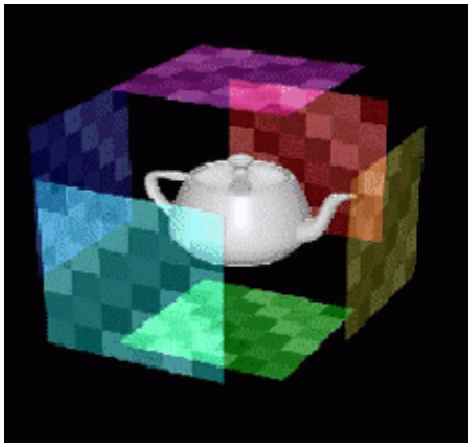
Normal Map e Reflection map

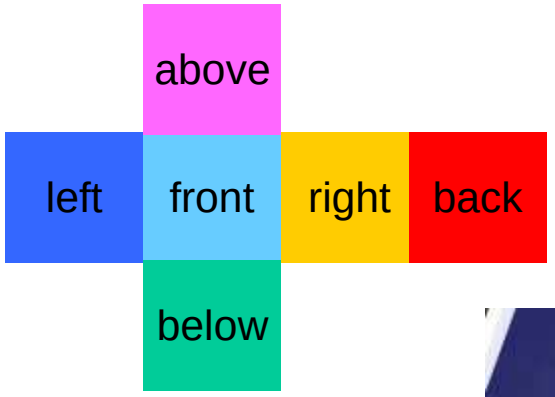
- `glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);`
- `glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_NORMAL_MAP);`

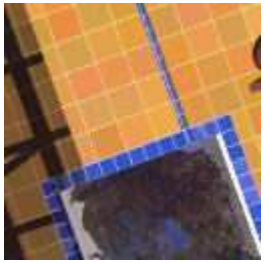


NB: utili per il cube mapping, si usano congiuntamente al cube mapping ma sono due concetti ortogonali

Environment mapping: cube maps







Environment mapping: cubico