Texture Mapping

Modelli per i materiali

- Il realismo visivo può essere ottenuto mediante i modelli di illuminazione e l'utilizzo dei materiali

- Per modellare i materiali si possono utilizzare delle immagini dette immagini di texture da incollare sulla struttura geometrica dell'oggetto
  - Durante la fase di shading

Immagini di texture o mappe

- Sono esempi di texture:
  - color-map: ogni pixel è un colore (componenti: R- G-B, o R-G-B-A) – l'intensità di colore di un punto della superficie dipende dal colore del punto corrispondente della immagine di texture
  - alpha-map: ogni pixel è un valore di trasparenza alpha
  - normal-map" o "bump-map" ogni pixel è una normale (componenti: X-Y-Z): le variazioni dei livelli di grigio della mappa daranno luogo a variazioni di altezza sulla superficie

  - **shininess-map**: ogni pixel contiene un valore di specularità

•Le immagini texture possono essere di vario tipo, da immagini fotografiche digitalizzate a immagini generate in modo procedurale.

•Le proprietà dell'oggetto che possono essere modificate dal texture mapping, andando a modificare i relativi coefficienti nel modello di illuminazione locale, sono le seguenti:

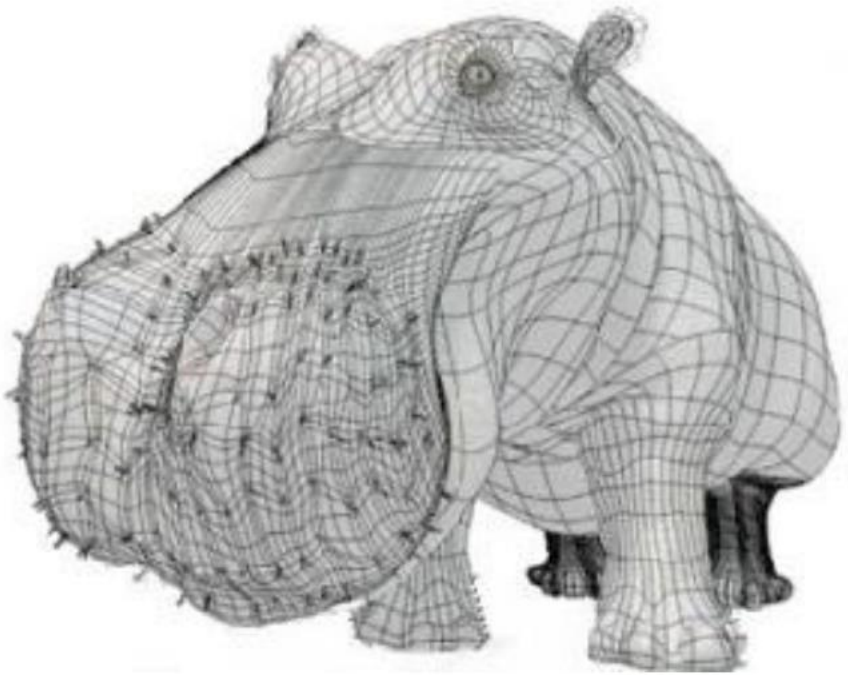
•**Colore**: viene modificato il colore (diffuso o speculare) nel modello di illuminazione di Phong con il corrispondente colore dalla texture map;

•**Colore riflesso** (environment mapping): simula la riflessione di un oggetto speculare che riflette l'ambiente circostante; permette di creare immagini che apparentemente sembrano generate da un ray tracer;

•**Perturbazione della normale** (bump mapping): perturbazione della normale alla superficie in funzione di un valore corrispondente in una funzione 2D;

•**Trasparenza**: controllare l'opacità di un oggetto trasparente.

modelling

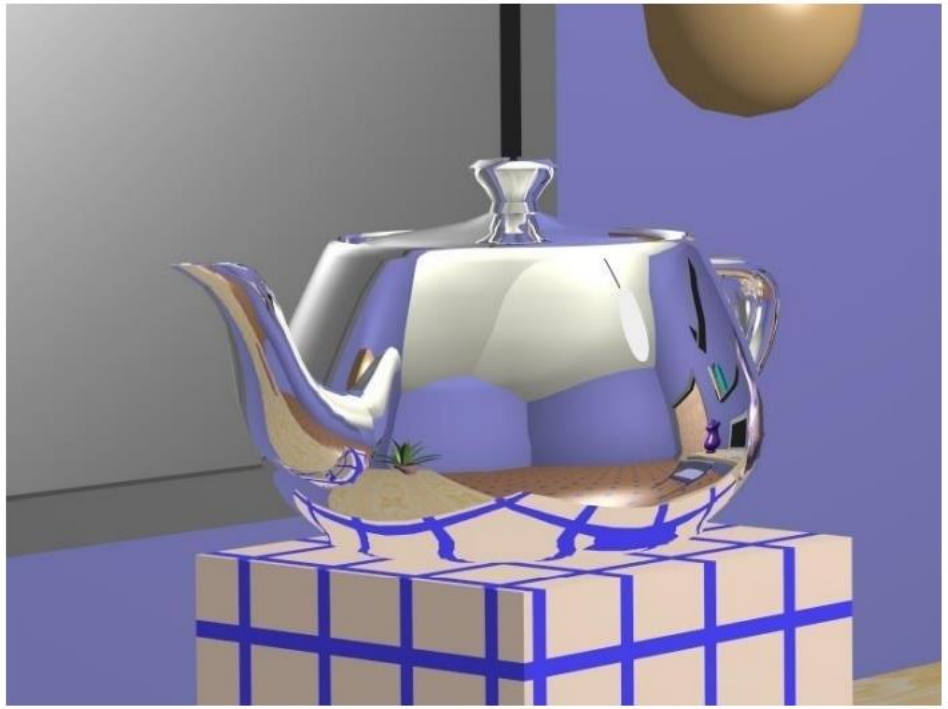


modelling + shading

Il colore della texture modifica il colore finale

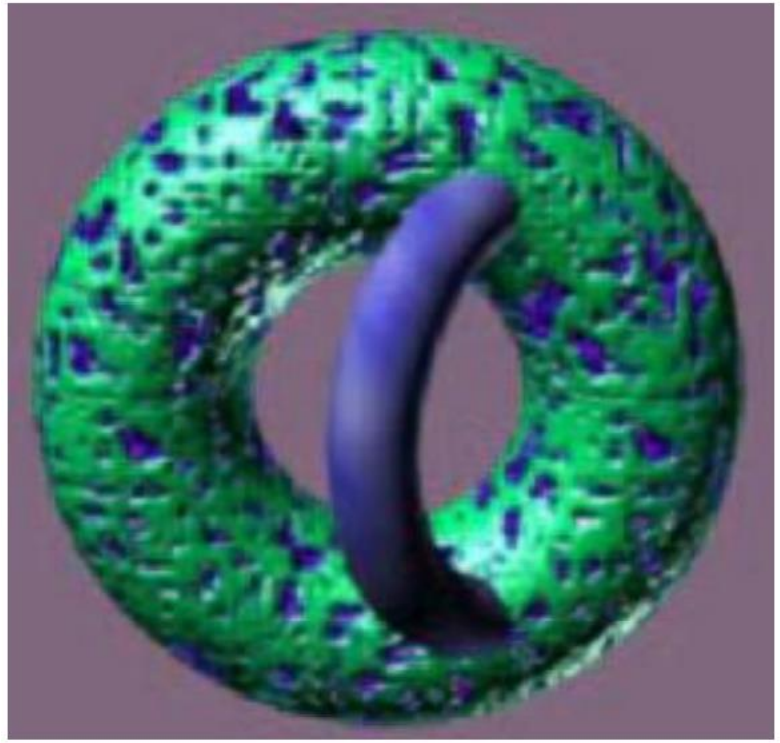
Environment Mapping

- Utilizza un'immagine dell'ambiente per le mappe delle texture
- Consente la simulazione di superfici altamente riflettenti



Modifica l’opacità di un oggetto trasparente

Il valore di intensità di un'immagine sorgente viene utilizzato per modificare la direzione della normale alla superficie dell'oggetto. Non modifica la geometria della superficie, ma il risultato visivo ci induce a pensare che la superficie non sia liscia

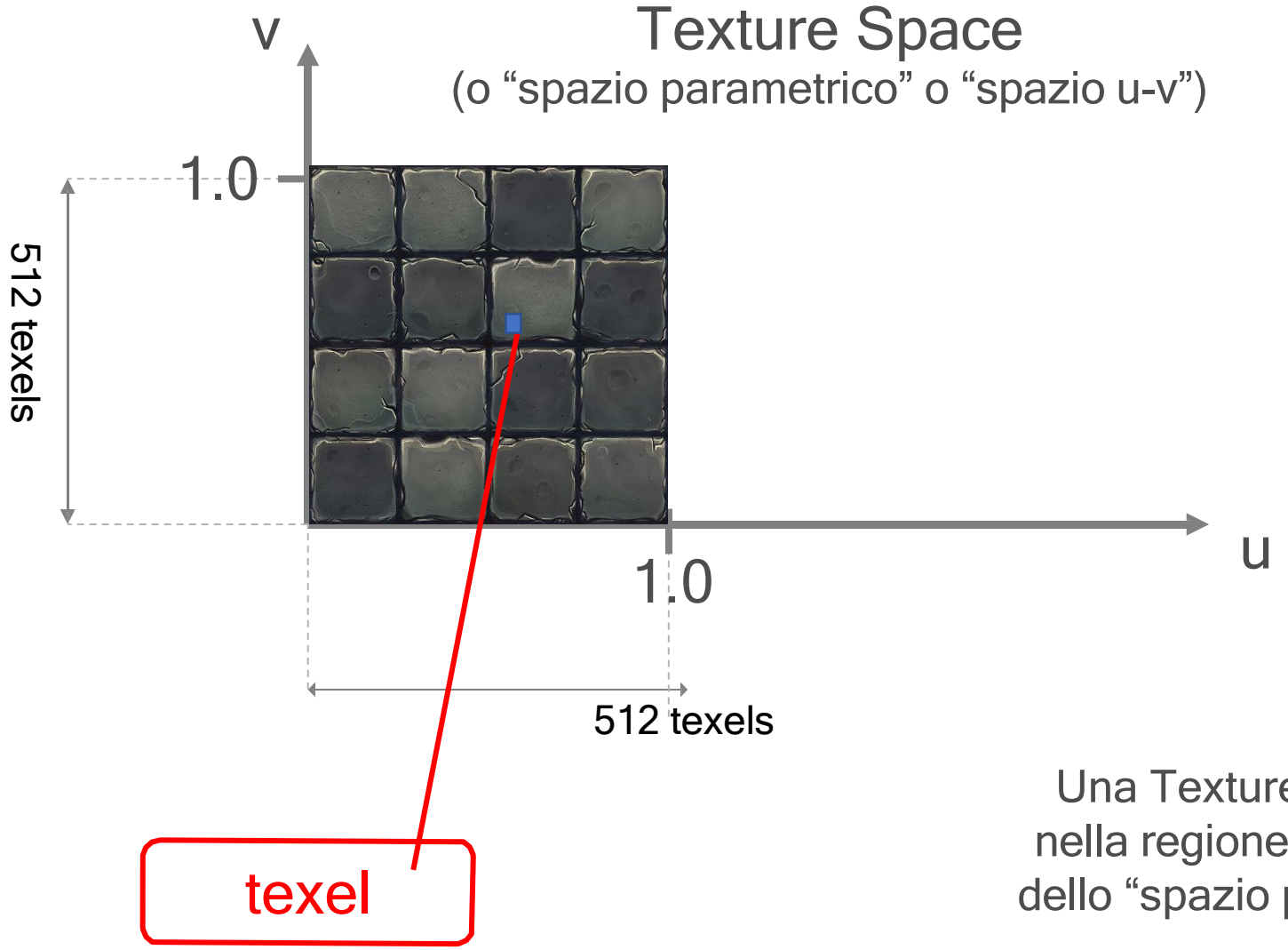


Storia

Ed Catmull

- 1974 introdotto da Ed Catmull nella sua Phd Thesis
- Solo nel 1992 si ha il texture mapping in hardware Silicon Graphics RealityEngine
- Dal '92 ad oggi ha avuto una rapidissima diffusione
- Oggi è una delle fondamentali tecniche di rendering
- La più utilizzata tecnica image based

Notazione



Una Texture è definita
nella regione [0,1] x [0,1]
dello “spazio parametrico”

Texture mapping

- Consideriamo una texture come una matrice di $n \times m$ elementi, il texture mapping è una funzione che
- Associa i texel ai punti della superficie di un oggetto geometrico.
- Proietta i punti sullo schermo

E' una funzione matematica che trasforma

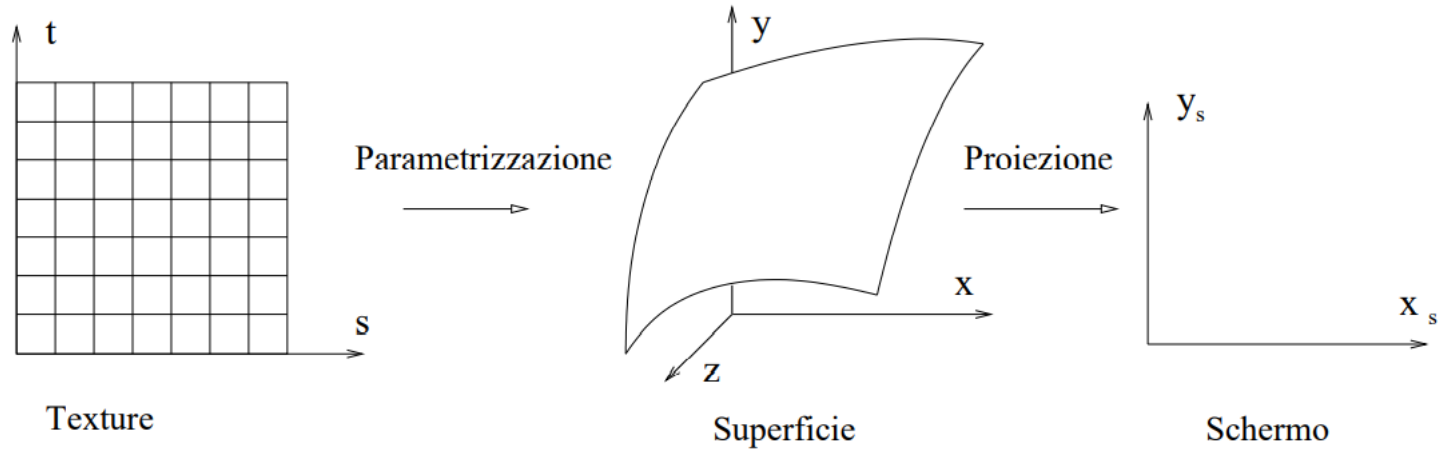
- Coordinate texture (u,v) in coordinate del mondo (X,Y,Z) in coordinate schermo (x_s, y_s)

Texture Mapping

- Texture mapping consiste di 3 passi
  - Definire la texture nello spazio (u,v)
  - Associare durante la fase di modellazione a ciascun vertice di ciascun triangolo (X_i, Y_i, Z_i) il corrispondente punto nello spazio (u,v)
  - Calcolare durante la fase di rendering per ogni pixel generato all'interno di una faccia il valore della texture per interpolazione

Occorre definire la funzione di parametrizzazione $W(\cdot)$ che associa un punto (s, t) della texture ad un punto P della superficie dell'oggetto 3D (è una funzione che "spalma " la texture sulla superficie).

Il punto P viene poi mappato dalla proiezione in un punto (x_s, y_s) dello schermo



- il texture rendering si occupa poi di stabilire il valore di texture da associare a ciascun pixel

Parametrizzazione

- Nella mappatura in un passo si definisce (in forma analitica) la **funzione W che definisce la corrispondenza tra i pixel della texture ed i punti della superficie.**
- Questo si può fare quando si ha la descrizione parametrica della superficie che ha generato la mesh poligonale.
- Altrimenti si specifica manualmente la corrispondenza W^{-1} tra vertici della mesh e punti della texture.
- Se la superficie è data in forma parametrica, ad ogni suo punto P sono associate due coordinate (parametri) (u, v).
- Per ottenere W basta specificare la mappa che va da (u, v) sulla superficie a (s, t) nella texture.
- è necessario che W sia invertibile.
- Spesso è l'identità (con qualche fattore di normalizzazione).

Parametrizzazione di una sfera

- Ad esempio si consideri una sfera di raggio r e centro l'origine:

$$S(\theta,\phi)\begin{cases} r\sin(\phi)\cos(\theta), \\ r\cos(\phi) \\ r\sin(\phi)\sin(\theta) \end{cases} \qquad \theta \in [0, 2\pi] \qquad \phi \in [0, \pi)$$

Si può scrivere come

$$S(u,v) = \begin{cases} r\sin(\pi v)\cos(2\pi u), \\ r\cos(\pi v) \\ r\sin(\pi v)\sin(2\pi u) \end{cases} \qquad u, v \in [0, 1]$$

$W^{-1} : (u,v) \rightarrow (s,t)$ è l'identità

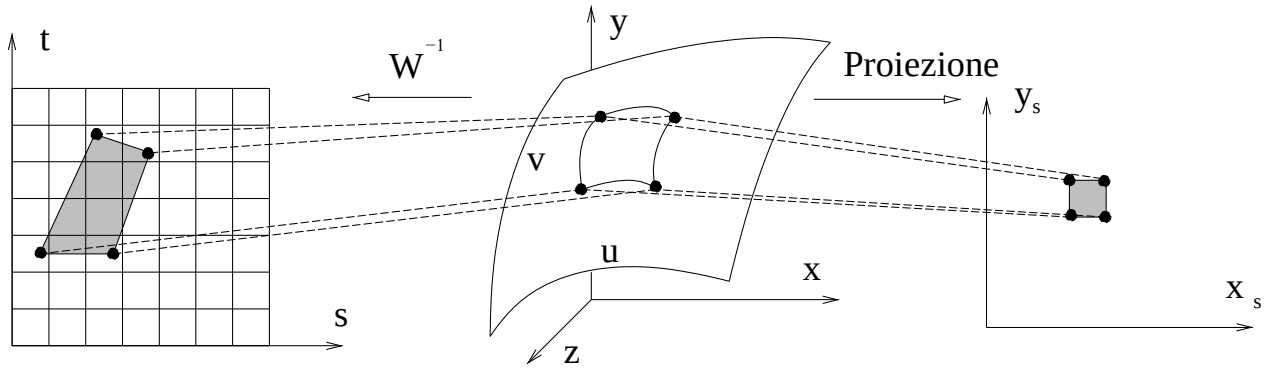
s=u, t=v

- Una tecnica più generale, che si può usare **senza conoscere l'equazione parametrica** della superficie, è la mappatura **in due passi**
- Si mappa la texture su una superficie intermedia semplice, in modo che la parametrizzazione (corrispondenza punti-superficie con pixel-texture) sia immediata; questa prende il nome di **S-mapping**
- Quindi si mappa ogni punto della superficie intermedia in un punto della superficie in esame; questa prende il nome di **O-mapping**
- La concatenazione dei due mapping genera la corrispondenza *W* trai pixel della texture ed i punti dell'oggetto
- Il primo passaggio (S-mapping) è in genere semplice; basta scegliere superfici facili da parametrizzare
- Ad esempio si può prendere come superficie intermedia un cilindro
- Oltre al cilindro è facile fare l'S-mapping con cubi, piani e sfere.
- In genere si considera la superficie intermedia come esterna all'oggetto da texturizzare

Rendering

- Quello che serve per il rendering è la corrispondenza tra texture (s, t) e pixel dell'immagine (x_s, y_s).
- Vi sono tipicamente due strategie per poter mappare una texture:
  1. Mappatura in avanti: (forward mapping) dato un pixel (s, t) sulla texture si trova il punto (x, y, z) dell'oggetto su cui tale pixel viene mappato da W e quindi, con la normale proiezione che abbiamo studiato, si trova il suo corrispondente (x_s,y_s).
  2. Mappatura all'indietro: (inverse mapping) Per ogni pixel (x_s, y_s) si trova sulla texture la corrispondente coordinata (s, t). Si usa la retroproiezione del pixel sulla superficie e la funzione W^{-1}.
- Per evitare fenomeni di aliasing, nell'assegnare un valore di texture ad un pixel dell'immagine bisogna tenere presente che la "pre-immagine" di un pixel nello spazio texture non `e un punto, ma un quadrilatero curvilineo (in generale), il quale può comprendere al suo interno molti pixel della texture.
- Si usa quasi sempre l'inverse mapping.

- Supponiamo che siano date la funzione W e l'equazione parametrica della superficie.
- Dato il pixel che vogliamo disegnare, lo si (retro)proietta sulla superficie (Si usano i quattro vertici). La sua "impronta" sulla superficie è un patch quadrangolare.



- Questo patch è descritto da due parametri (u, v) che, mappati con W^{-1} in (s, t), definiscono una quadrilatero nella texture.
- In realtà l'immagine di un pixel (quadrato) secondo il la mappatura all'indietro è un quadrilatero curvilineo nello spazio texture.
- Integrando sul quadrilatero si ottiene il valore per il pixel.
- Questa può essere presa anche come descrizione del processo "ideale" di texture mapping.

Cosa avviene in Opengl?

- La mappatura W^{-1}, associa una coordinata texture (s, t) ad ogni vertice della mesh;
- Quindi a ciascun vertice di un triangolo proiettato è associata una coordinata texture (s, t).
- Durante la scan conversion del triangolo si determinano le coordinate texture di ciascun pixel interno conl'interpolazione scan-line.
- Si determina quindi il valore di texture da associare al pixel arrotondando le coordinate texture all'intero più vicino. In questo modo ogni pixel riceve contributo da un solo texel (texture element).
- Se la risoluzione della texture è molto diversa da quella del display si incorre nell'aliasing.

Mapping inverso

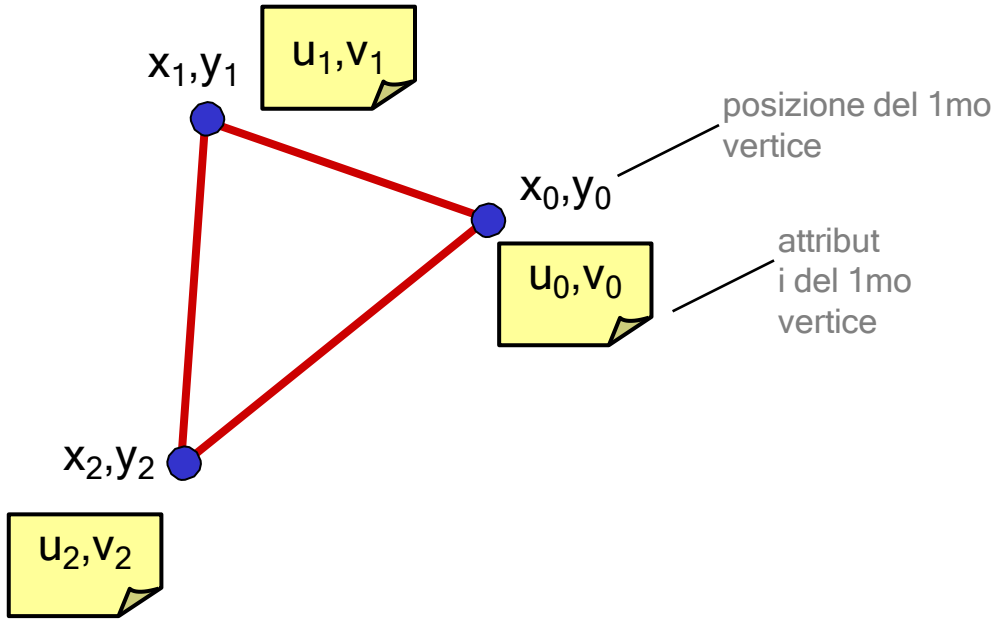
- Il calcolo è effettuato nelle coordinate dello schermo, per evitare il calcolo su frazioni di pixel
- E' necessario considerare la trasformazione inversa dalle coordinate dello schermo (x_s,y_s) alle coordinate della texture (u,v):
  - Per ogni pixel dello schermo si calcola mediante trasformazione inversa il punto corrispondente (X_i, Y_i, Z_i) sulla superficie
  - Il punto (X_i, Y_i, Z_i) viene mappato nello spazio della texture (u,v)

Texture mapping e modello di illuminazione

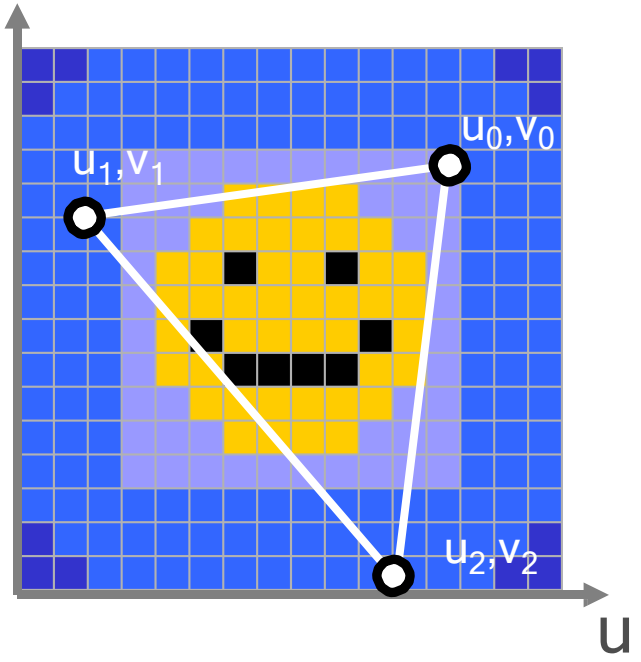
- Il valore di texture – ottenuto sia in maniera diretta che inversa - può essere utilizzato
  - Per attribuire direttamente il colore al pixel
  - Oppure per attribuire il colore al pixel dopo essere stato composto con il valore di riflessione diffusa di Phong
    - $I_f = (1 - \alpha_t)\, I_t + \alpha_t\, I$
    - I_f - valore finale
    - I_t - valore di texture
    - I - valore di illuminazione Phong
    - α_t - valore di trasparenza per la texture (0..1)

Parametrizzazione di un triangolo

- Ad ogni vertice (di ogni triangolo) si associa un punto
- di coordinate u,v nello spazio della texture



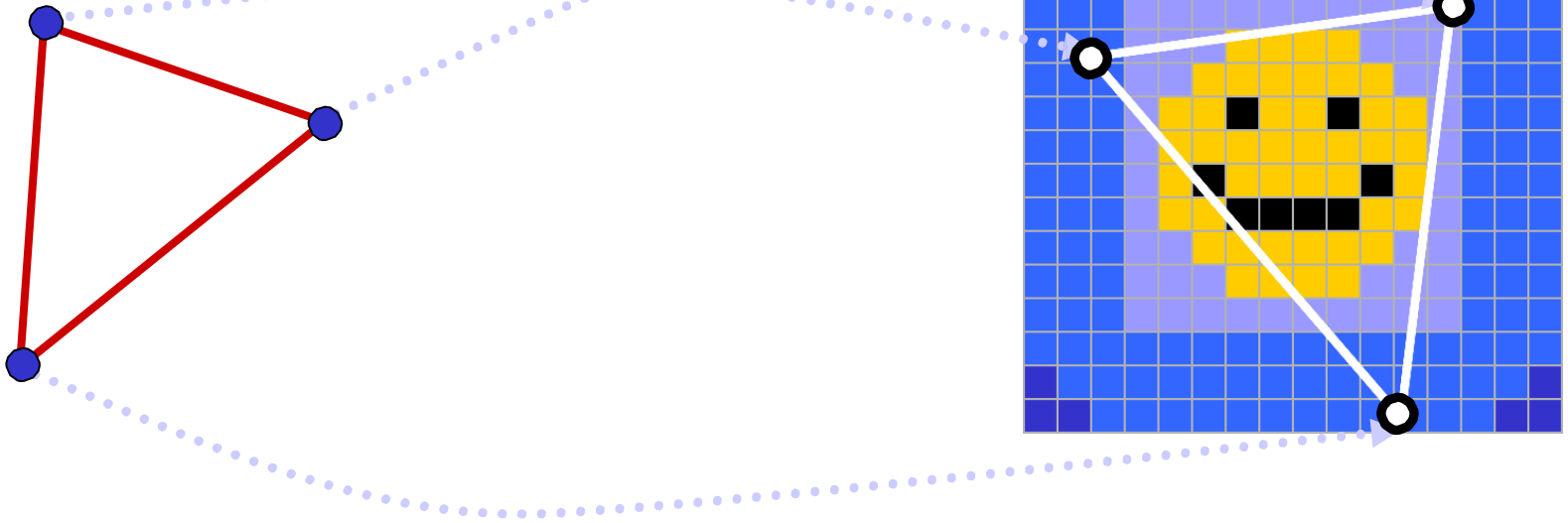
Triangolo in WCS



Texture Space

Texture Mapping

- Si definisce cioè un mapping fra il triangolo e un triangolo sulla texture

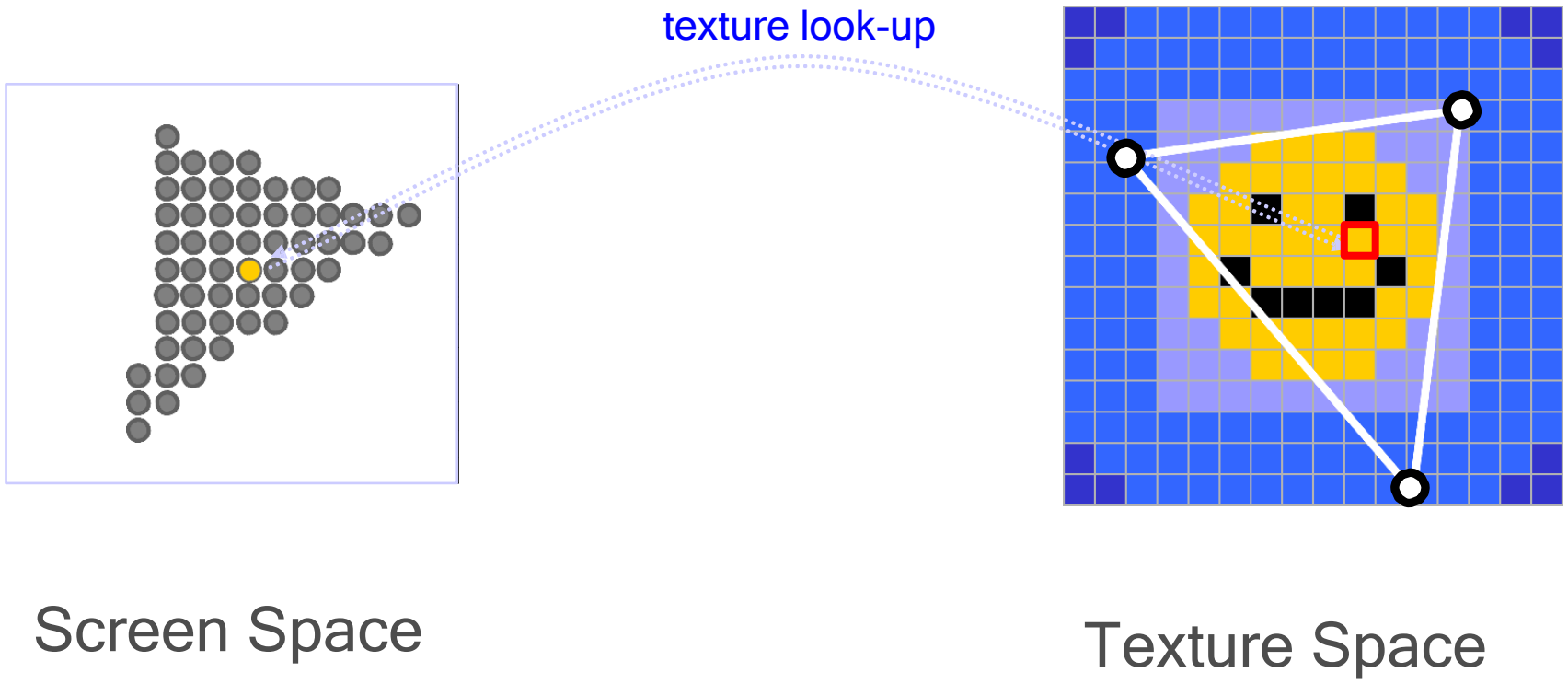


Screen Space

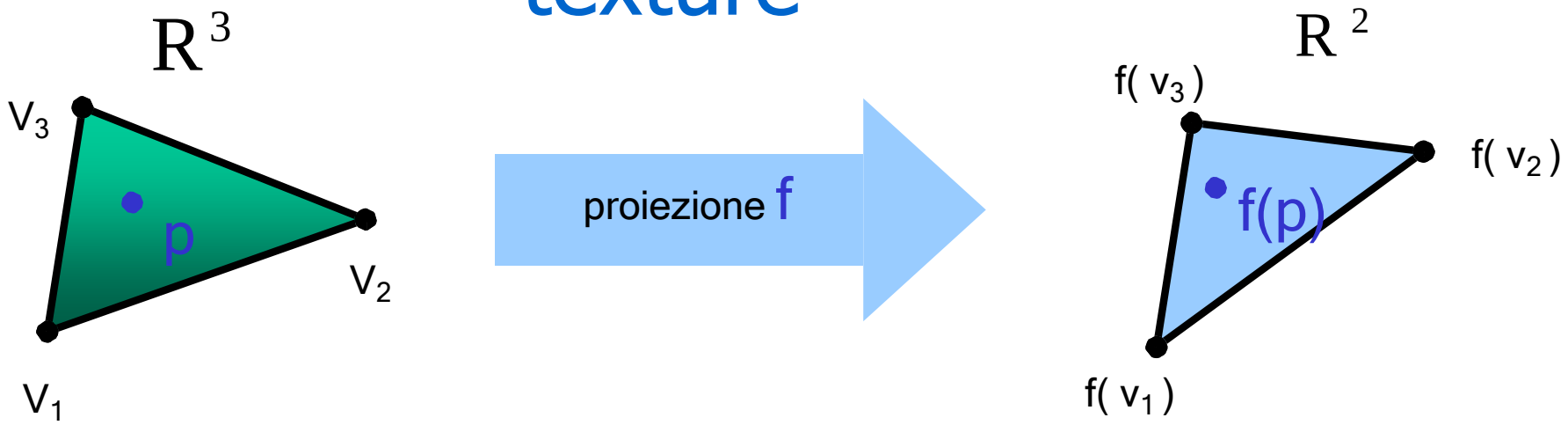
Texture Space

Texture Mapping

- Ad ogni vertice (di ogni triangolo) si associa un punto di coordinate u,v nello spazio della texture



Interpolazione delle coordinate texture



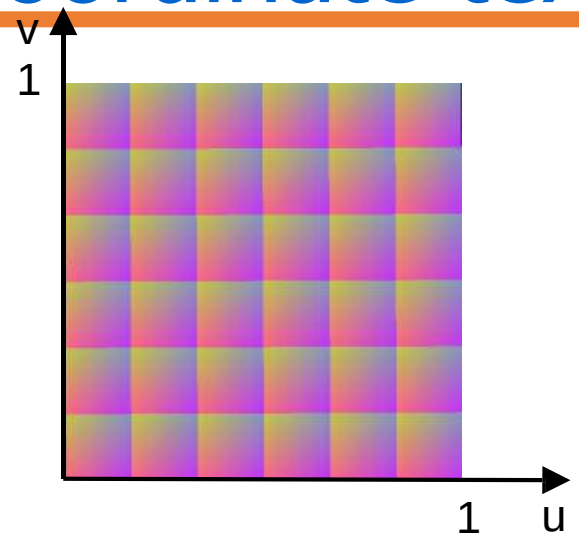
p ha coordinate baricentriche a,b,c nel triangolo v_1 v_2 v_3

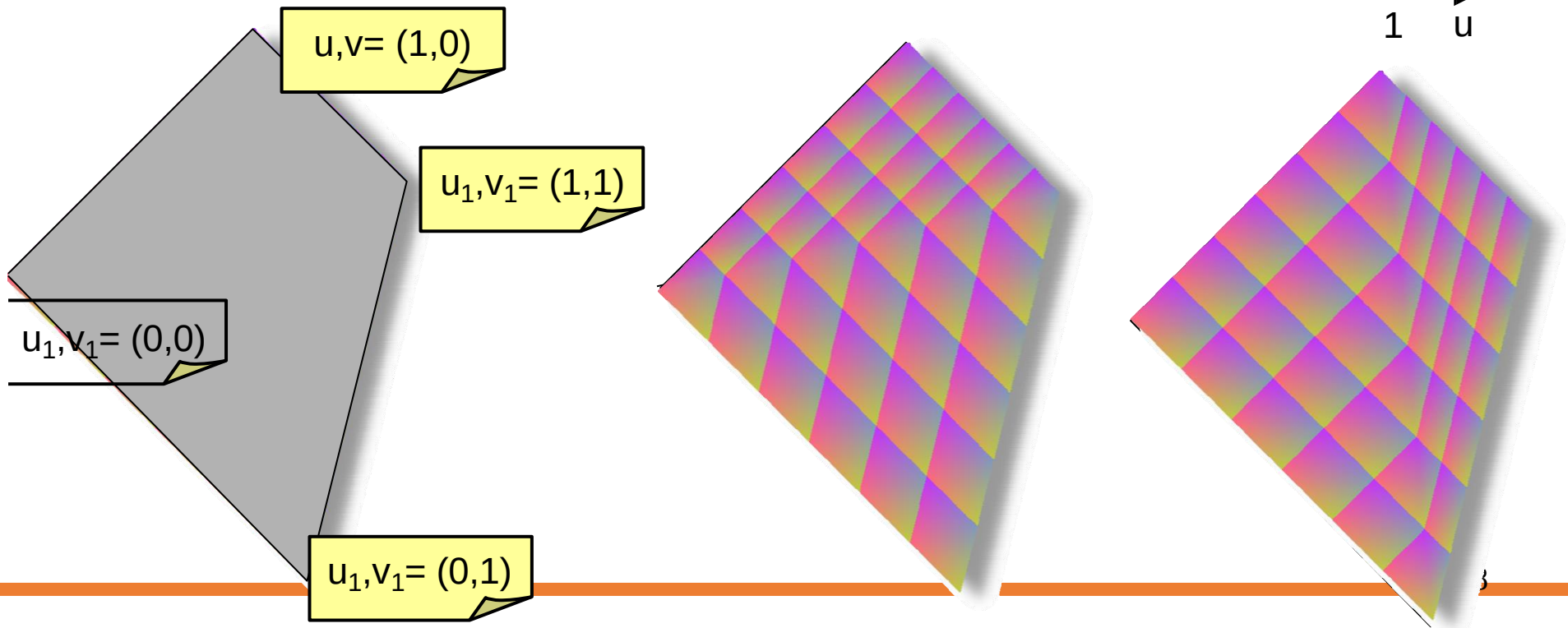
f(p) ha coordinate baricentriche a,b,c nel triangolo $f(v_1)$ $f(v_2)$ $f(v_3)$

- La proiezione è una approssimazione utile per colori e normali ma non funziona quando si interpolano le coordinate di texture...

Interpolazione delle coordinate texture

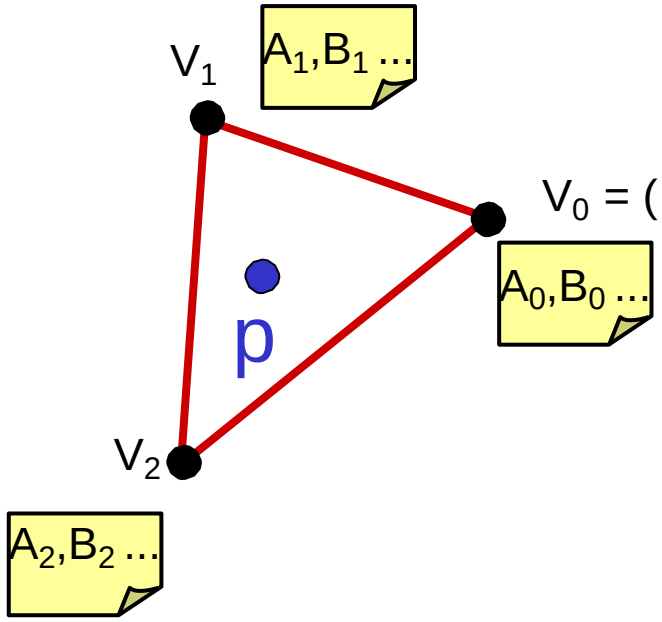
- Esempio:





Correzione Prospettica

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



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

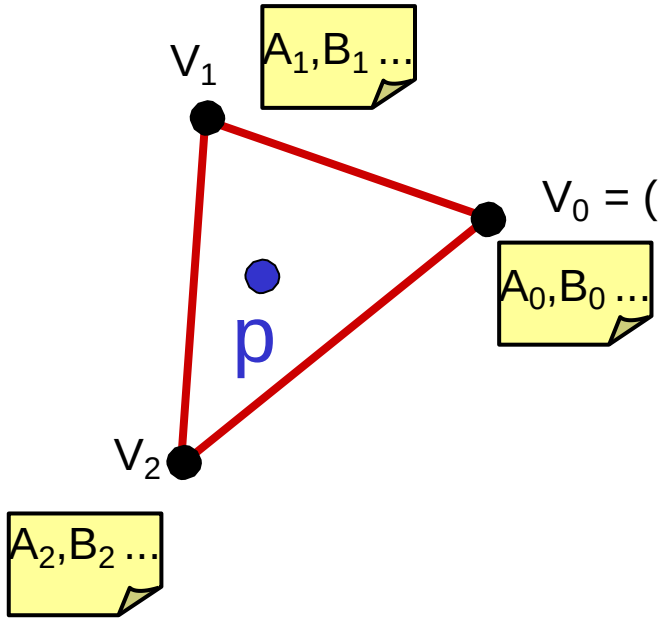
attributi di p:
(senza considerare
la correzione prospettica)

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

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

Correzione Prospettica

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



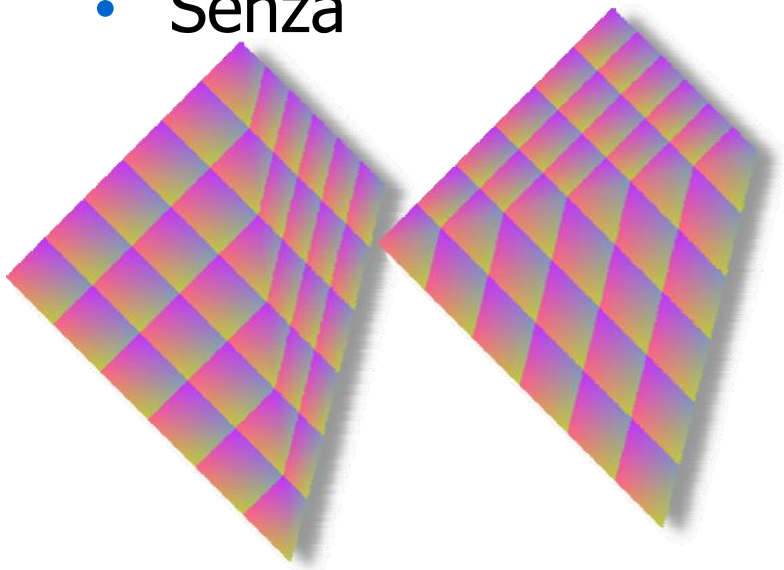
$$p = c_0 v_0 + c_1 v_1 + c_2 v_2$$

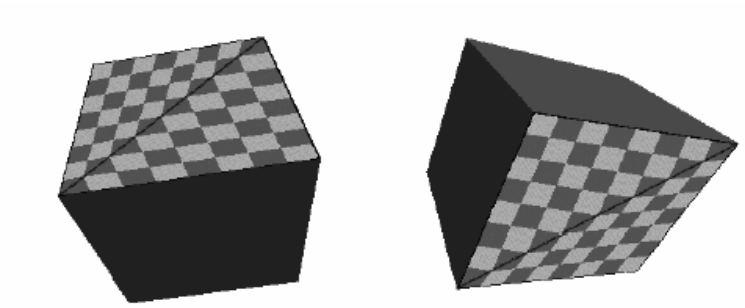
attributi di p:
(con la correzione prospettica)

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

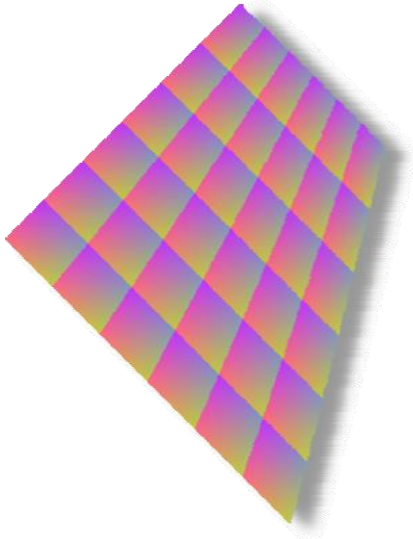
Correzione Prospettica

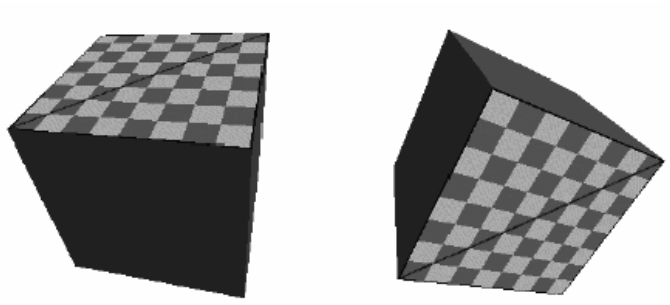
- Senza





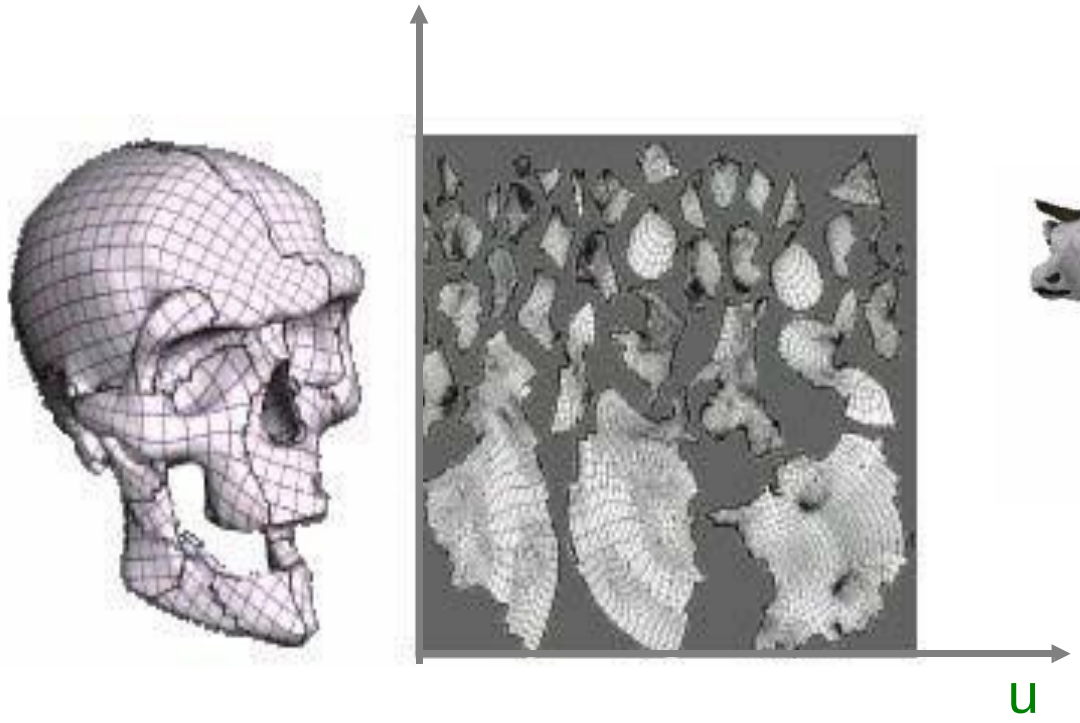
- Con

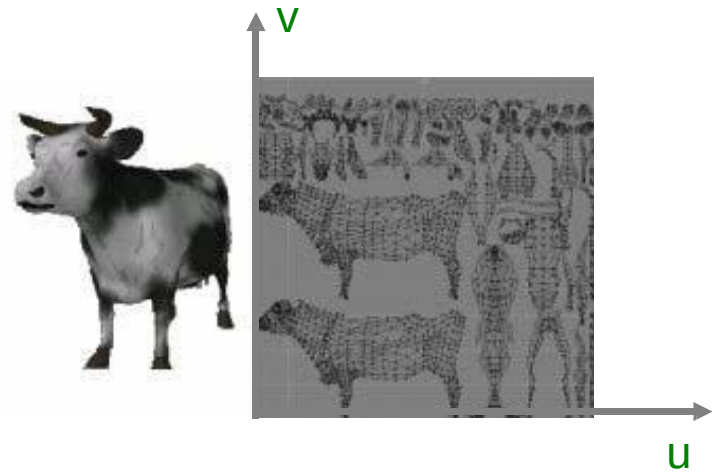




u-v mapping

- Assegnare una coppia di coordinate di texture ad ogni vertice della mesh

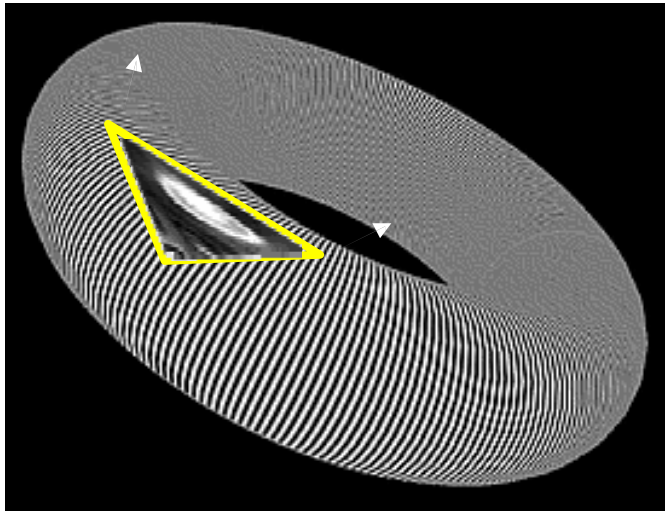




Texture atlas

Esempio environment mapping

Environment map: una texture
che memorizza il colore dell'ambiente "riflesso"
da ogni normale della semisfera.



Environment mapping: sferico

simula oggetto a specchio che riflette uno sfondo lontano

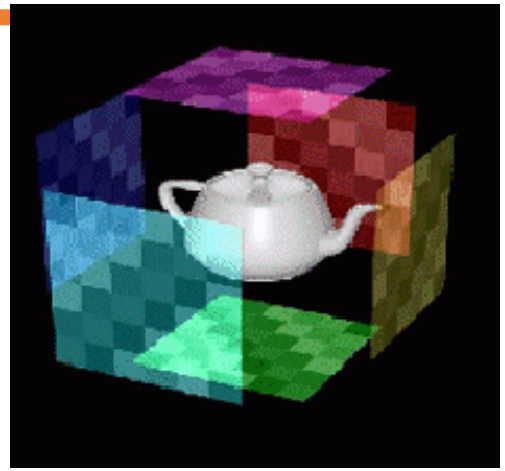


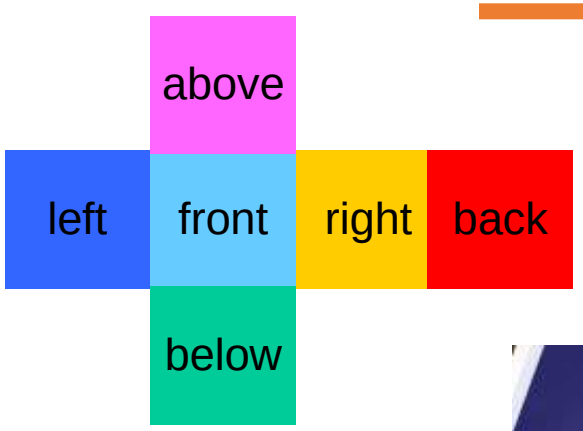





simula un materiale complesso
(condizioni di luce fisse)

Environment mapping: cubico



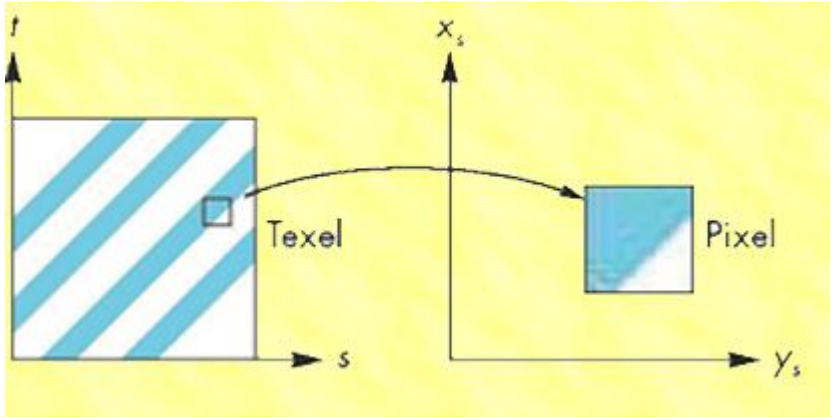


Aliasing

Si ha aliasing ogni qualvolta non ci sia corrispondenza uno-a-uno tra elementi del display (pixel) ed elementi della texture (texel).

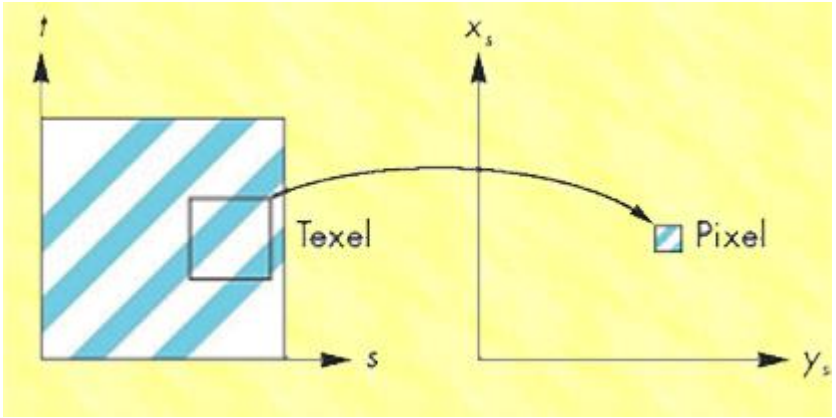
Distinguiamo due casi:

(a) Magnification: un texel corrisponde a molti pixel. In questo caso un singolo texel appare come un blocco di pixel nell'immagine. Quello che si vede è una versione ingrandita "a blocchi" della texture. E' come ingrandire una immagine digitale: se i pixel diventano troppo grossi si notano effetti "a blocchi".



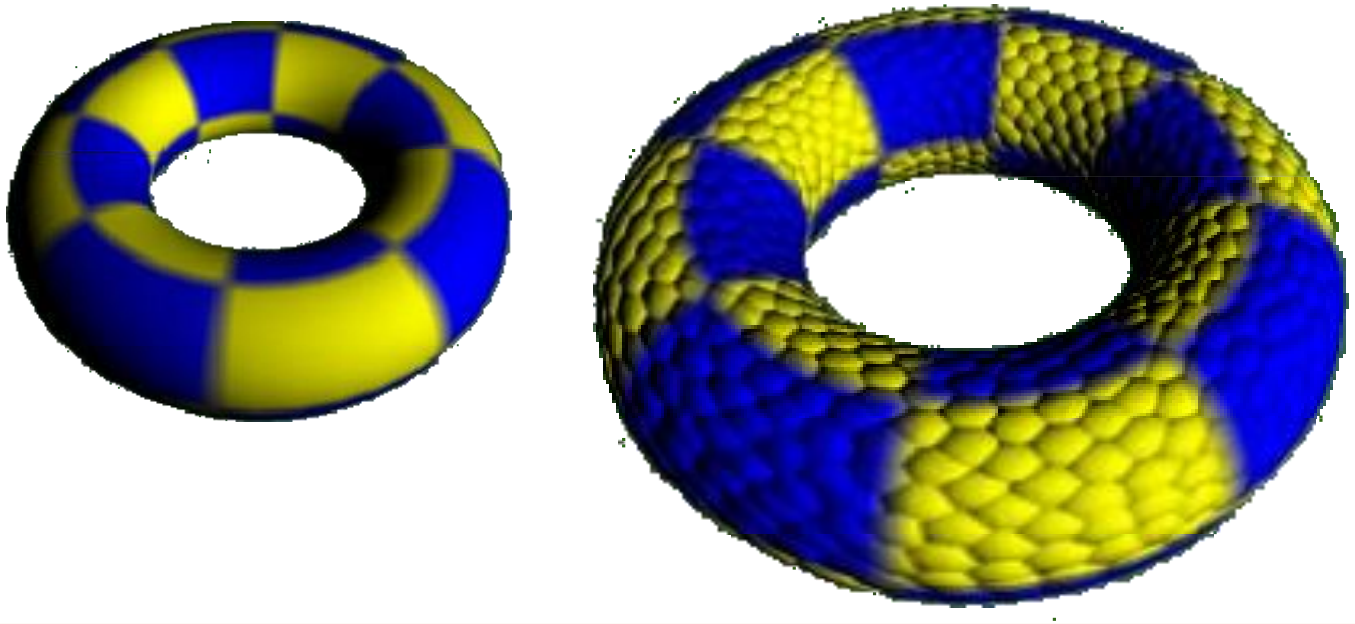
Minification: Cosa accade se un pixel copre molti texel? La texture deve essere rimpicciolita.

b) Minification: un pixel corrisponde a molti texel. Si dovrebbe vedere una versione rimpicciolita della texture. Ma se si usa il metodo delineato prima, in cui ad ogni pixel contribuisce un solo texel, vuol dire che ci sono texel che restano fuori.E' come quando si riduce un'immagine digitale: se si sottocampiona si possono perdere particolari



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 pixel per pixel utilizzando la formula:

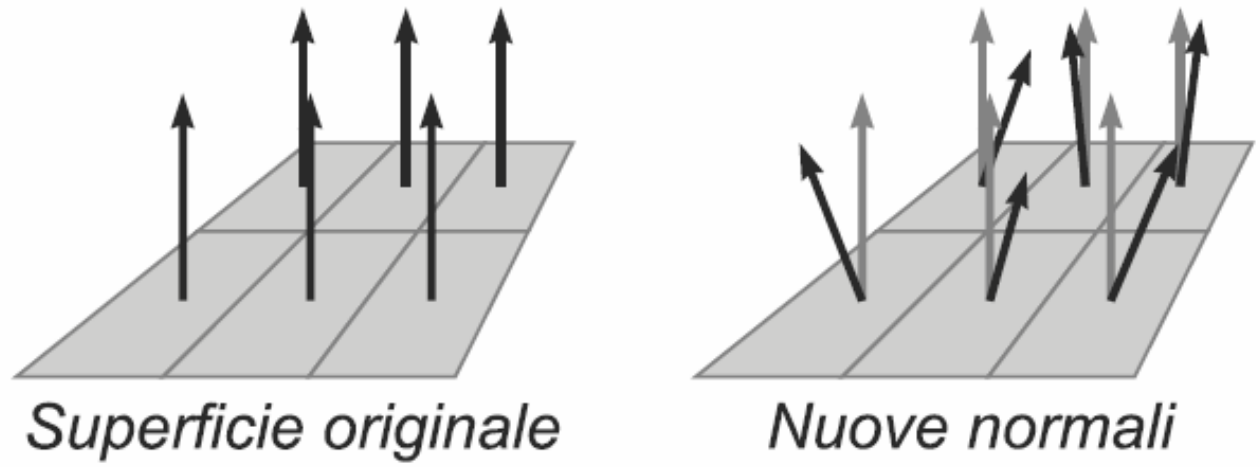
$$\vec{\mathbf{N}}_{new} = \vec{\mathbf{N}}_{old} + \vec{\mathbf{D}};$$

$$\vec{\mathbf{D}} = (\Delta x, \Delta y, \Delta z)$$

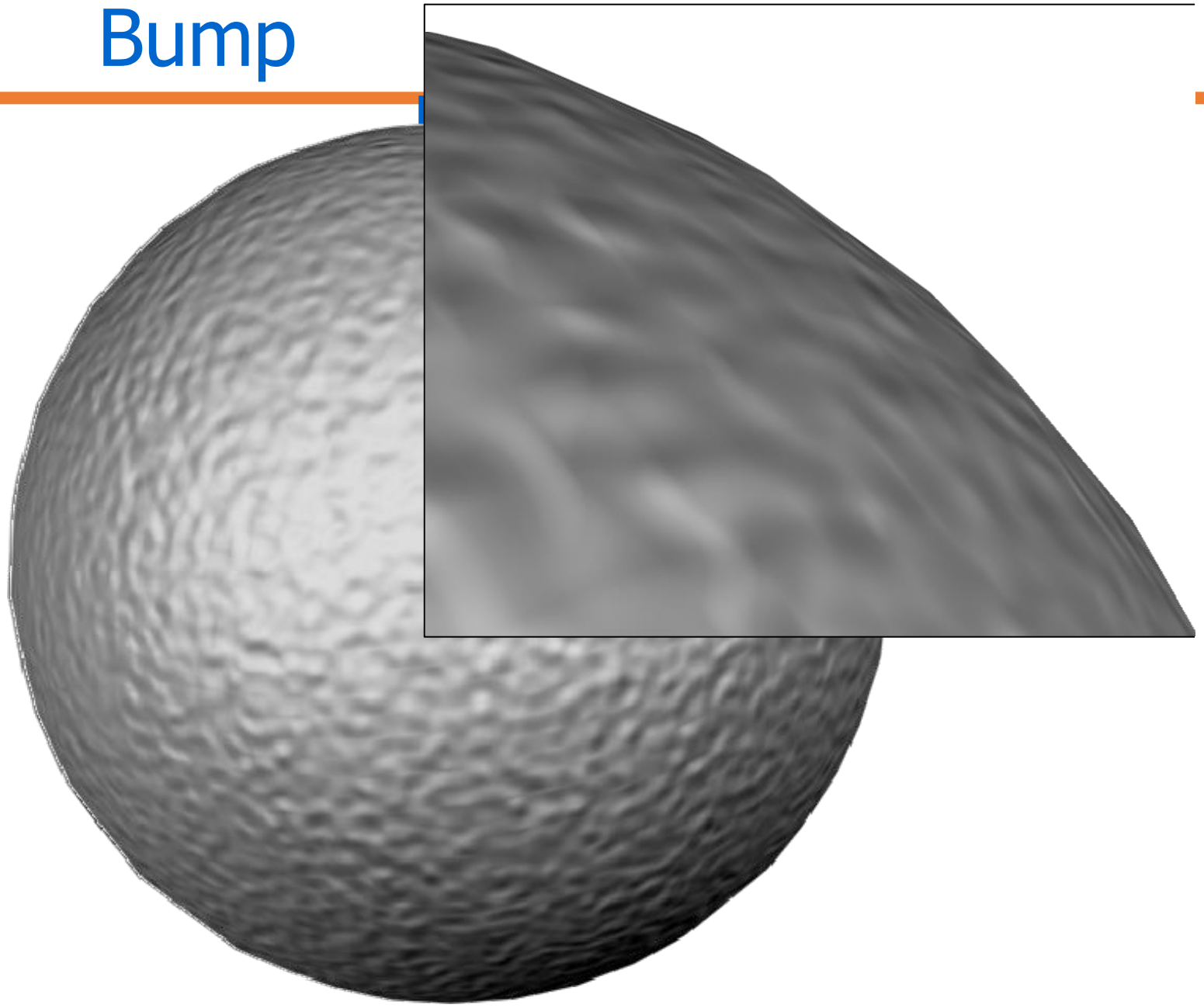
- I texel in questo caso sono utilizzati ad uno stadio diverso rispetto ai color texel, prima del calcolo dell'equazione di illuminazione

Bump mapping

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



Bump



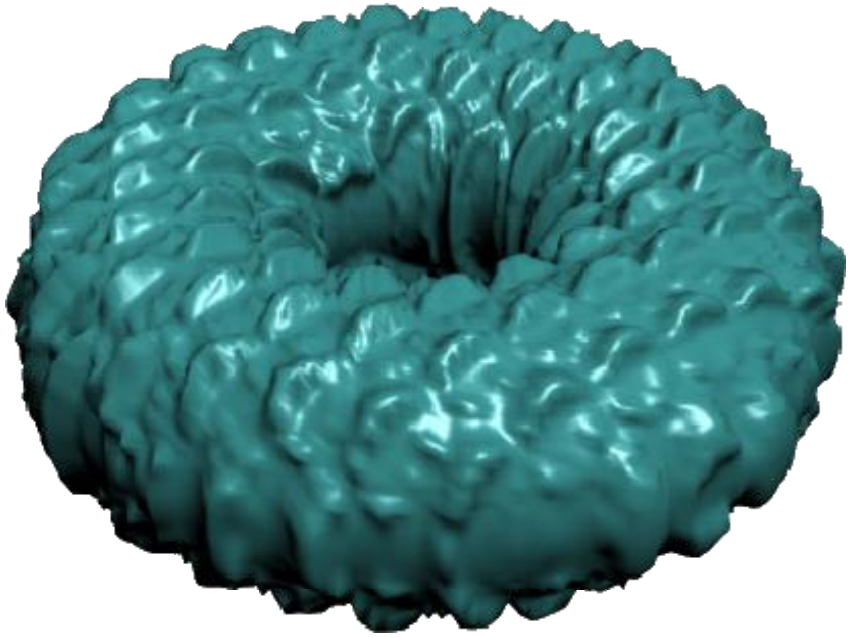
Displacement mapping

- 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 modica stabilmente la geometria della scena
- Rispetto al bump mapping anche la silhouette del modello mostra le corrette deformazioni

Displacement mapping



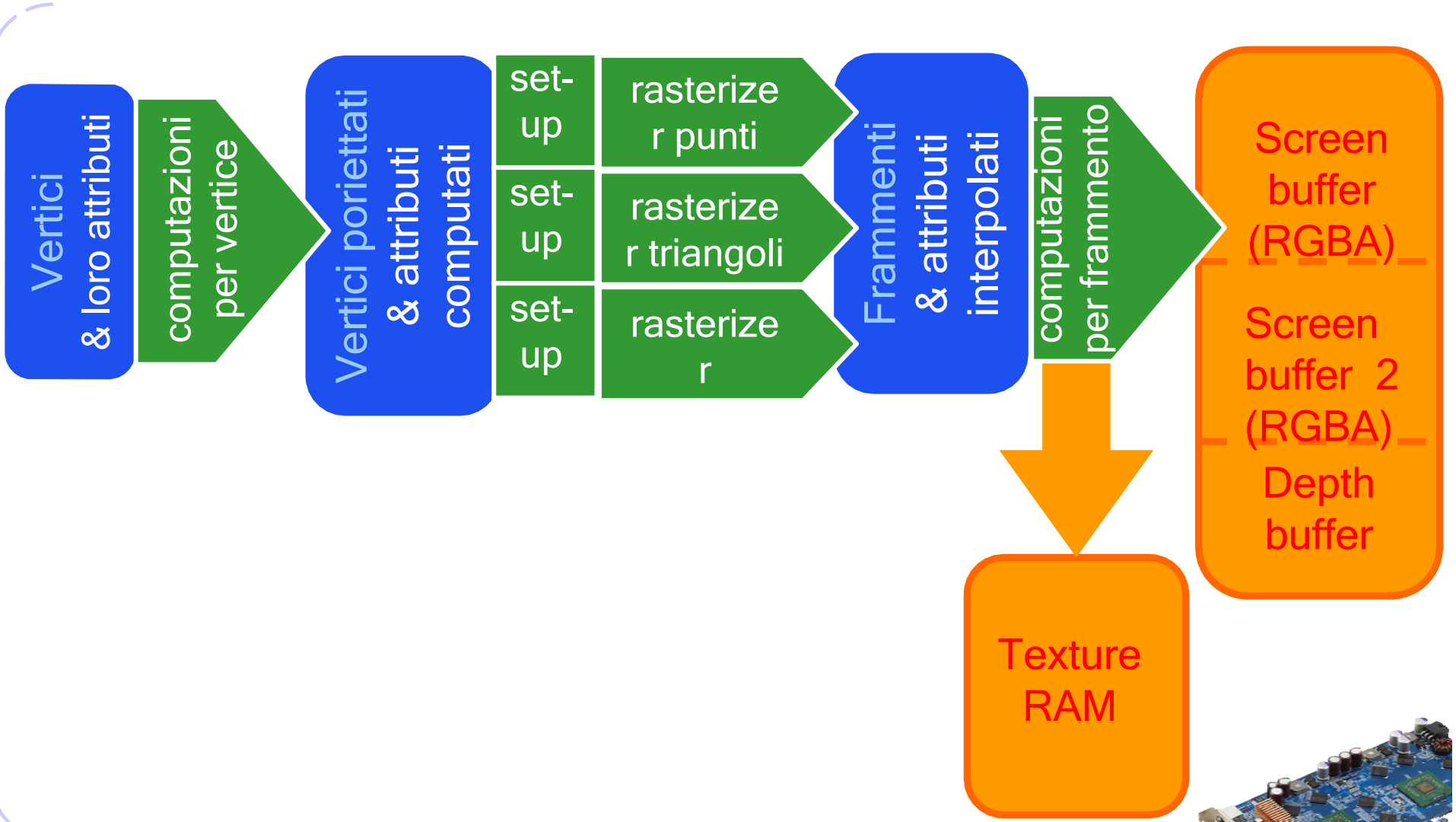
Texture mapping e schede grafiche

- La grande diffusione del texture mapping è dovuta anche al largo utilizzo nei videogiochi
- La struttura di un video gioco è caratterizzata da
  - Uso di geometrie semplici con livello di approssimazione delle forme basso
  - Utilizzo di sofisticate textures.

Architettura del processo di rendering

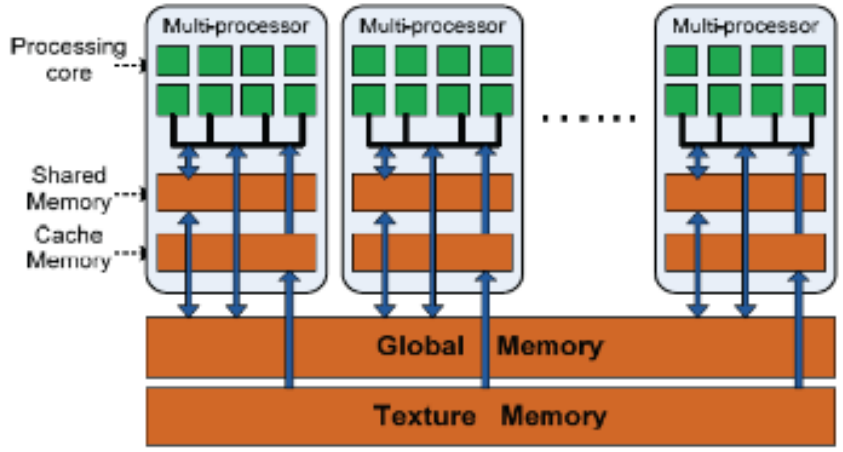
- Il sistema di rendering si può suddividere in
  - Sottosistema geometrico – si occupa del calcolo delle trasformazioni geometriche ed illuminazione dei vertici delle primitive geometriche
  - Sottosistema raster – si occupa della rasterizzazione e calcolo della texture delle primitive

Memoria RAM nelle schede grafiche



Texture Mapping

- Nelle operazioni per frammento in coordinate dello schermo, per la gestione delle textures si accede ad una RAM apposita**: la Texture RAM** strutturata in un insieme di Textures Units.



Texture Unit

- Una **texture unit**, chiamata anche **texture mapping unit (TMU) o texture processing unit (TPU)**, è un componente hardware in una GPU che esegue il campionamento.
- Il campionamento è il processo di calcolo di un colore a partire da una texture e dalla coordinate di texture.
- La mappatura di un'immagine di texture su una superficie è un'operazione abbastanza complessa, poiché richiede molto di più della semplice restituzione del colore del texel che contiene alcune coordinate di texture date.
- Richiede inoltre l'applicazione dell'appropriato filtro di minification o magnification, possibilmente utilizzando mipmap se disponibili.
- Il campionamento rapido delle texture è uno dei requisiti chiave per buone prestazioni della GPU.

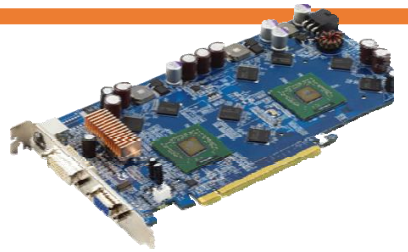
Un texture object è una struttura dati che contiene:

- i dati del colore dell'immagine texture
- possibilmente per un set di mipmap per la texture,
- i valori delle proprietà della texture come i filtri di minimizzazione e ingrandimento
- la modalità di ripetizione della texture.

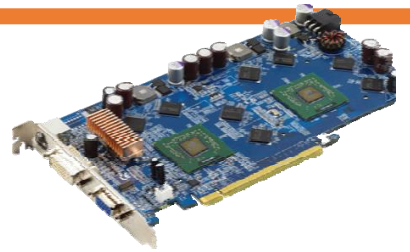
Una texture unit deve accedere a un texture object per svolgere il proprio lavoro.

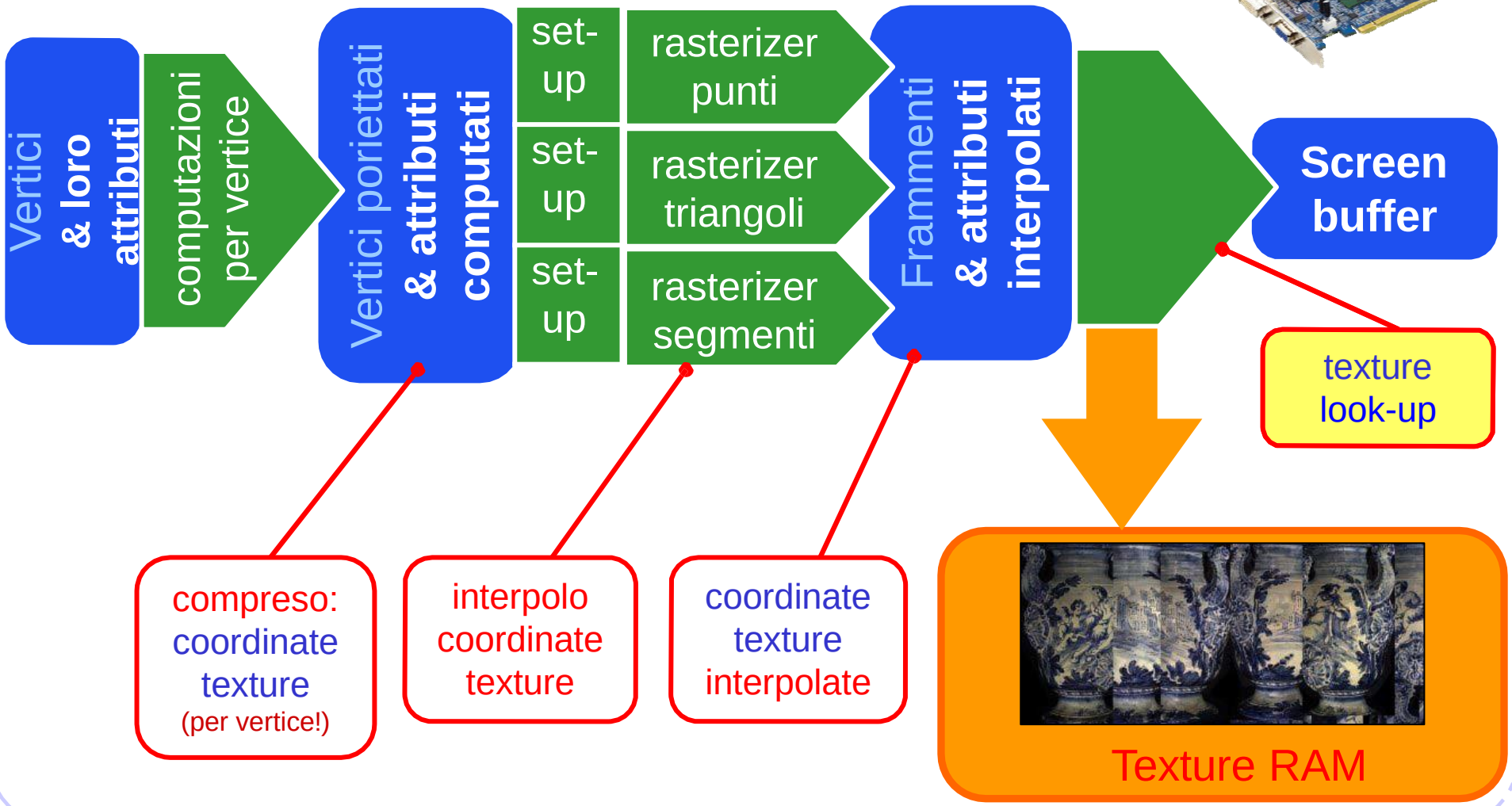
La texture unit è il processore; il texture object contiene i dati che elabora.

Texture Mapping



Texture Mapping

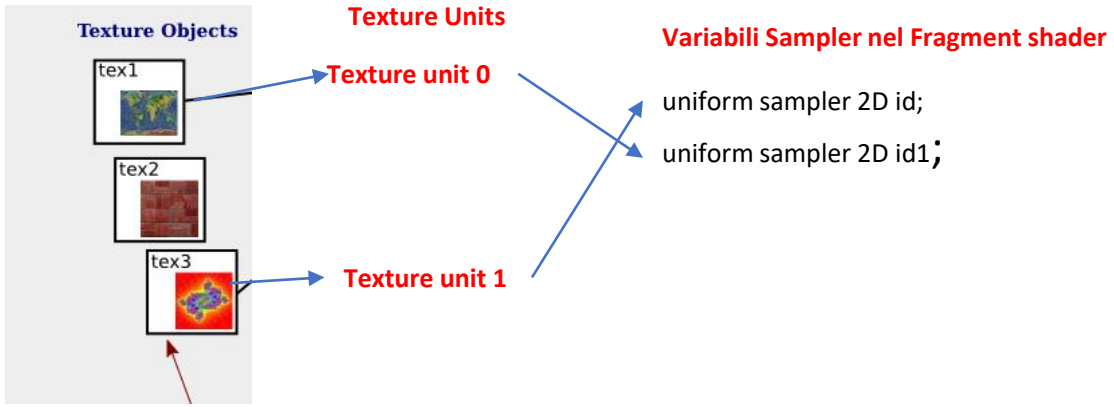




Supponiamo di avere **diverse immagini da mappare su primitive diverse**

Esistono tre modi diversi:

- **Usare un singolo oggetto texture e una singola Unit Texture**. L'oggetto texture associato, l'unità texture attiva e il valore della variabile sampler possono essere impostati una volta e mai modificati. Per passare a una nuova immagine, basta caricare l'immagine nell'oggetto texture, usando glTexImage2D per caricare l'immagine nell'oggetto texture. . Questo modo è inefficiente, tranne quando si utilizza ogni immagine solo una volta.
- Usare un **oggetto texture diverso per ogni immagine, ma utilizzare solo una singola Unit Texture .** La texture attiva e il valore della variabile sampler non dovranno mai essere cambiati. In questo caso si fa il glbindTexture per associare l'oggetto texture che contiene l'immagine desiderata.
- **Usare una texture unit diversa per ogni immagine**. In questo caso bisogna caricare ogni immagine nel proprio oggetto texture e associare quell'oggetto a una delle unità texture. In questo caso è necessario cambiare il valore della variabile *sampler.*



Una variabile sampler utilizza una texture unit, che utilizza un oggetto texture, che contiene un'immagine texture.

Per applicare un'immagine texture a una primitiva, bisogna impostare l'intera catena.

Vedremo come: creare un oggetto texture, associarlo ad una texture unit e passare al fragment shader la variabile uniforme che indica la textur unit da utilizzare.

Gestione delle textures in Opengl

Caricare un'immagine

Carica un'immagine
Le immagini di texture possono essere memorizzate in diversi formati di file, ognuno con la propria struttura e l'ordinamento dei dati .
OpenGL non legge i formati di immagine come JPEG, PNG, GIF, PPM, ...

stb_image.h: è una libreria per il caricamento delle immagini che supporta diversi formati di immagini

Per installarla, è sufficiente scaricare il file di intestazione, stb_image.h, ed includerlo nel progetto

```
#define STB_IMAGE_IMPLEMENTATION

#include stb_image.h

int larghezza, altezza, nrChannels;
 unsigned char * data = stbi_load ("pippo.jpg", & width, & height, & nrChannels, 0);
```

Applicare una Texture

Diverse fasi:

1. Carica e genera la texture –
   - Carica un'immagine da un file o generala in modo procedurale –
   - Genera ID texture –
   - Carica immagine nella texture memory
2. Mappatura della texture sull'oggetto –
   - Specifica le coordinate u, v per i vertici di ogni triangolo
3. Specifica i parametri della texture - parametri di texture, filtro, wrapping
4. Visualizza l'oggetto con la texture

Generare un oggetto texture

- generare un riferimento per la texture:

```
unsigned int texture;
glGenTextures(1, &texture);
```

glGenTextures :prende come input quante texture vogliamo generare e restituisce l'identificativo nel secondo argomento.

Per renderla attiva:

```
glBindTexture(GL_TEXTURE_2D, texture);
```

Caricare un' immagine nella texture memory

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);

Il primo argomento specifica il target della texture : GL_TEXTURE_1D , GL_TEXTURE_2D, GL_TEXTURE_3D

Il secondo argomento specifica il livello di mipmap per il quale vogliamo creare una texture nel caso in cui si desidera impostare manualmente ciascun livello di mipmap (noi specifichiamo alcun livello di mipamp e poniamo a 0 questo argomento).

Il terzo argomento specifica il tipo di formato in cui memorizzare la texture. Se l'immagine ha solo RGB valori, anche la texture viene memorizzata con valori RGB.

Il **quarto** e il **quinto** argomento impostano la larghezza e l'altezza della texture risultante.

L'argomento successivo dovrebbe essere sempre 0(per legacy).

Il **settimo e l'ottavo argomento specificano il formato e il tipo di dati dell'immagine sorgente**. Abbiamo caricato l'immagine con RGB valori e li abbiamo archiviati come chars (byte), quindi passeremo i valori corrispondenti.

L'ultimo argomento sono i dati dell'immagine.

Impostare il Texture Wrapping

Le coordinate della texture di solito vanno da (0,0)a (1,1), ma cosa succede se specifichiamo coordinate al di fuori di questo intervallo?

OpenGL offre la possibilità di scegliere tra:

GL_REPEAT : (defaul) . Ripete l'immagine della texture.

GL_MIRRORED_REPEAT : uguale a GL_REPEAT ma rispecchia l'immagine ad ogni ripetizione.

GL_CLAMP_TO_EDGE : blocca le coordinate tra 0 e 1.

GL_CLAMP_TO_BORDER : alle coordinate al di fuori dell'intervallo viene ora assegnato un colore specificato dall'utente.

GL_REPEAT

GL_MIRRORED_REPEAT

GL_CLAMP_TO_EDGE

GL_CLAMP_TO_BORDER

GL_REPEAT

GL_MIRRORED_REPEAT

GL_CLAMP_TO_EDGE

GL_CLAMP_TO_BORDER

glTexParameter: Funzione per impostare il wrapping

- glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);
- glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);
- Il primo argomento specifica il target della texture; stiamo lavorando con textures 2D, quindi l'obiettivo della texture è GL_TEXTURE_2D .
- Il secondo argomento richiede quale opzione vogliamo impostare e per quale asse di texture; vogliamo configurarlo sia per l'asse s che per l'asse t . L'ultimo argomento richiede di selezionare la modalità di wrapping delle textures desiderata (in questo caso GL_MIRRORED_REPEAT .)
- Se scegliamo l' opzione GL_CLAMP_TO_BORDER dovremmo anche specificare un colore del bordo. Questo viene fatto usando l'l'equivalente funzione glTexParameterfv con GL_TEXTURE_BORDER_COLOR come opzione in cui passiamo in una matrice float del valore di colore del bordo:
- float borderColor[] = { 1.0f, 1.0f, 0.0f, 1.0f };
- glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);

Impostare il texture filtering

Poiché le coordinate di texture sono indipendenti dalla risoluzione, non corrisponderanno sempre esattamente a un pixel.
Questo accade quando:

- un'immagine texture è piccola rispetto al poligono su cui deve essere mappata e quindi viene allungata oltre la sua dimensione originale
- quando è più grande rispetto al poligono su cui deve essere mappata e quindi va ridimensionata.

OpenGL offre vari metodi per decidere il colore campionato quando si verifica una di queste situazioni
Questo processo è chiamato filtraggio e sono disponibili i seguenti metodi

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

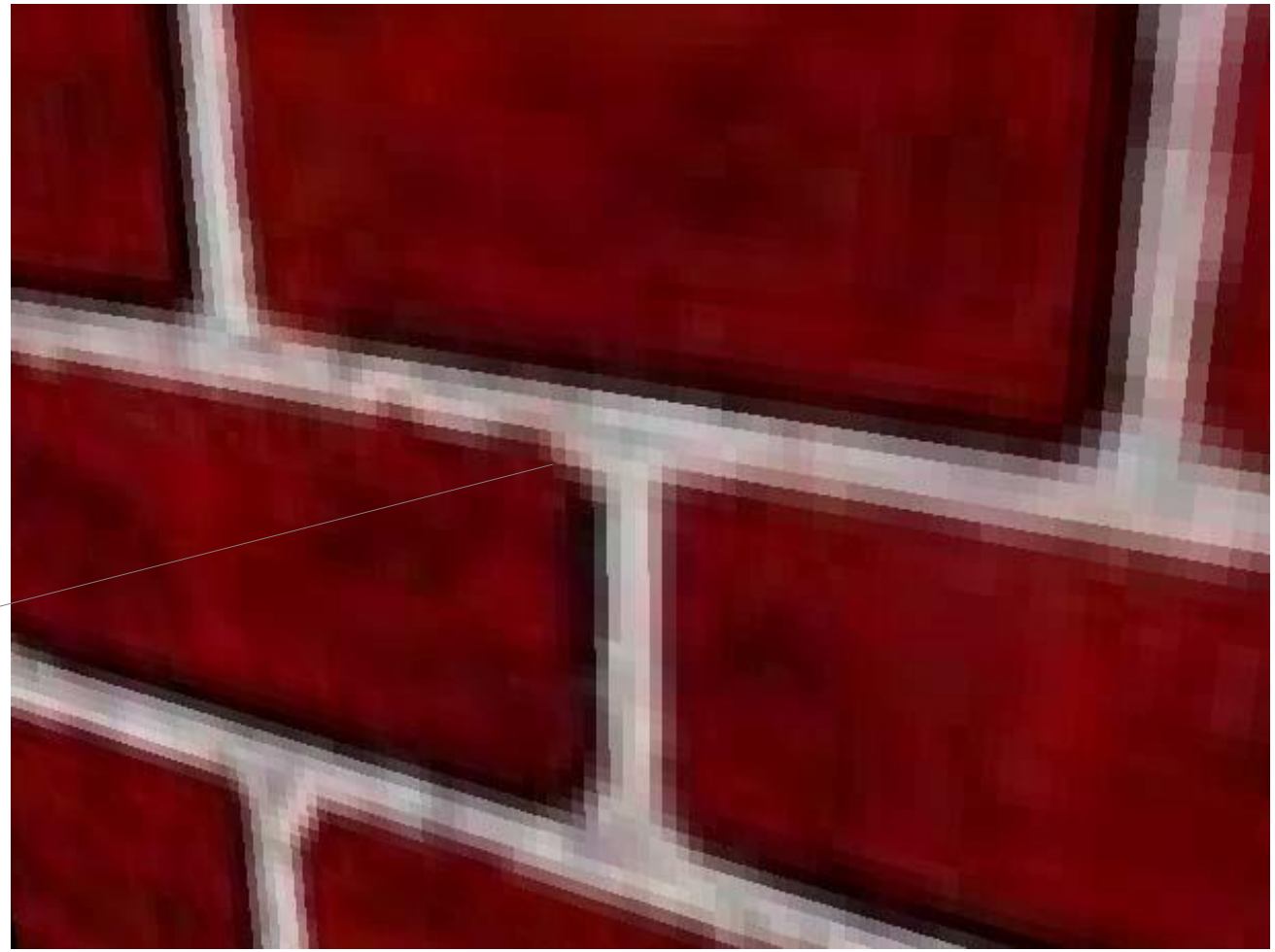
GL_LINEAR (noto anche come filtro (bi) lineare) prende un valore interpolato dai texel vicini della coordinata texture, approssimando un colore tra i texel

Caso Magnification

Nearest Neighbour : risultato visivo

texture 128x128

"si vedono i texel !"



Caso Magnification

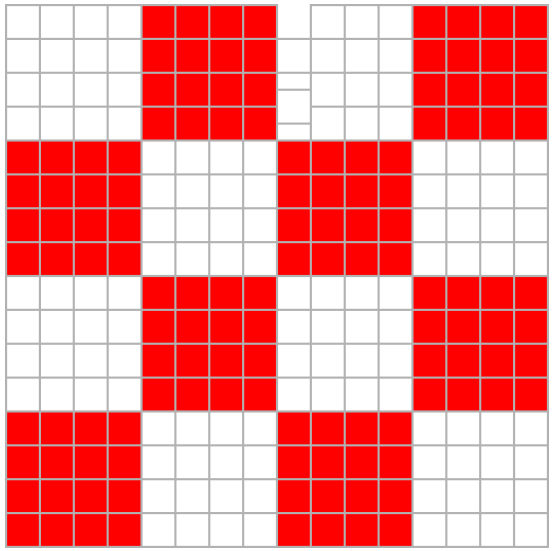
Bilinear Interpolation: risultato visivo

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 "sfuocato"

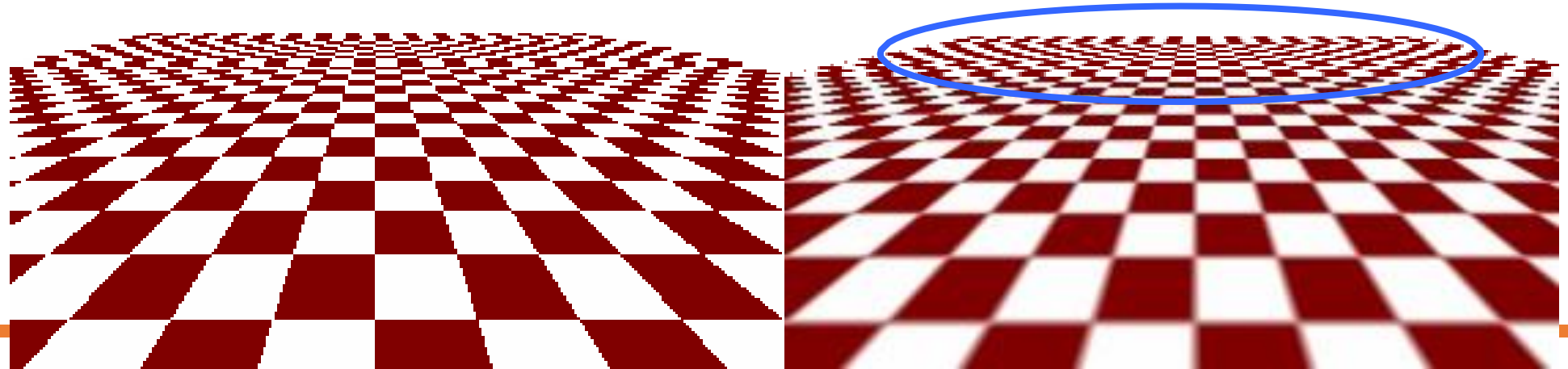
Caso Minification



Nearest Neighbour

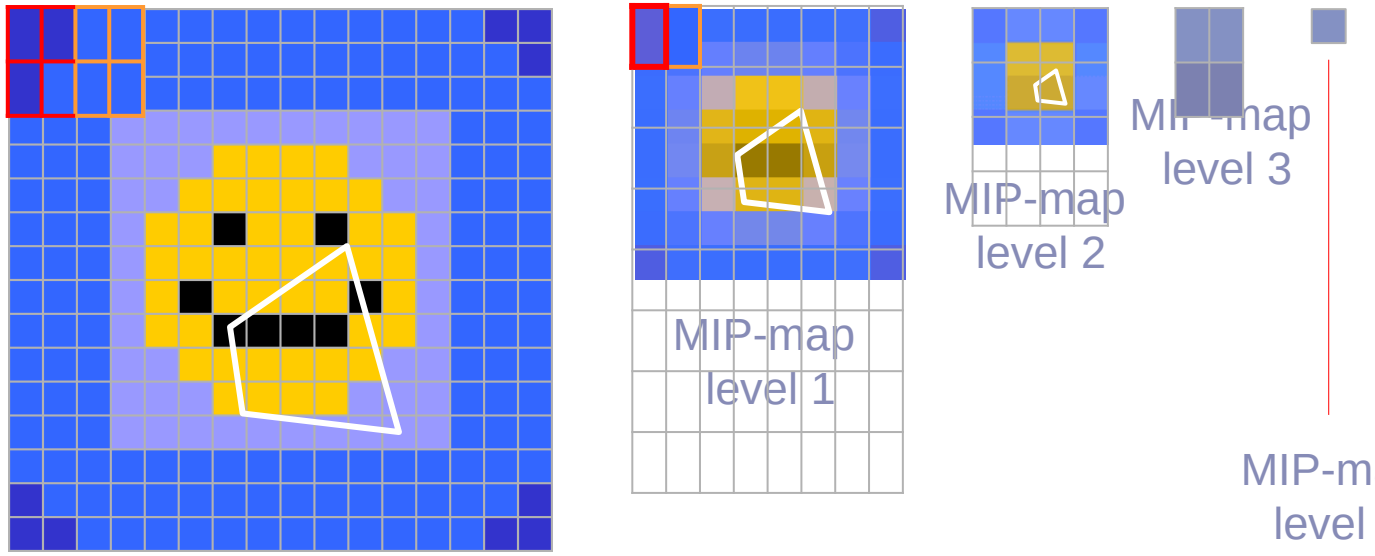
Bilinear interpolation

non risolve il problema



Caso Minification: MIP-mapping

MIP-mapping: multum in parvo



Si considera una gerarchia di mappe di texture a risoluzione diversa – piramide di immagini

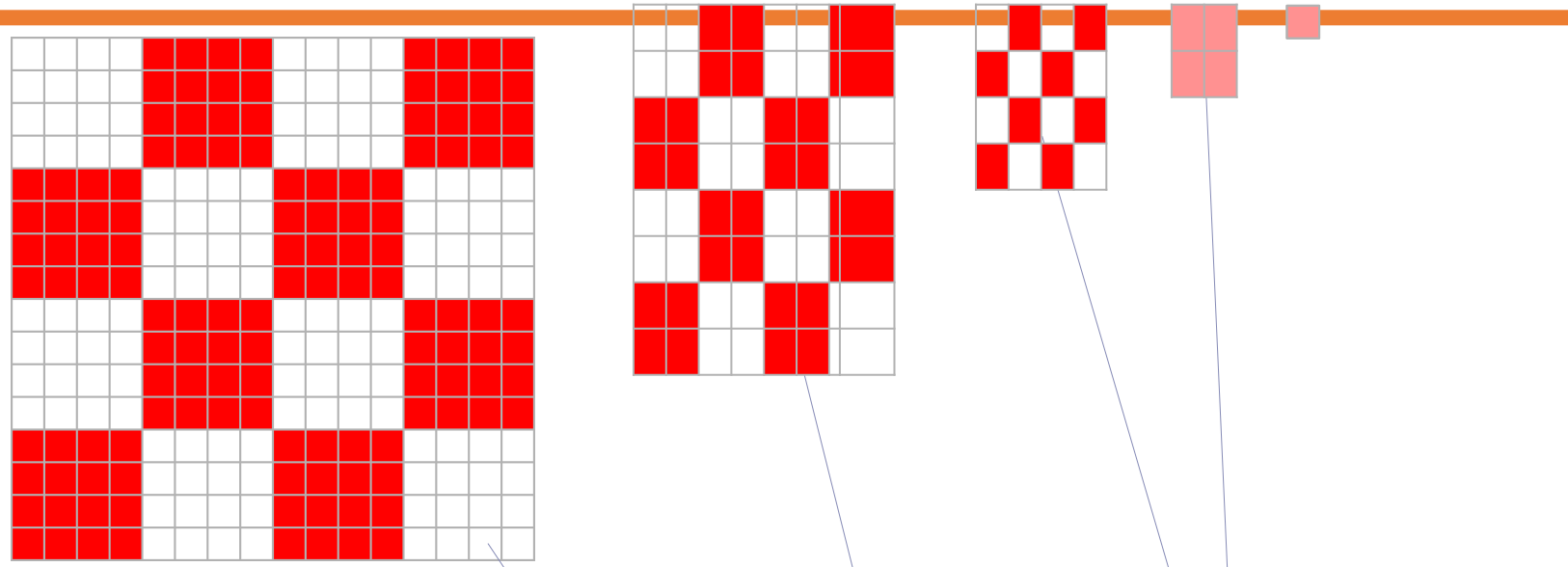
L'occupazione di memoria della piramide è circa il doppio della occupazione della singola immagine

Mipmap

- Si definisceun fattore di scala:
  - ρ= texel/pixel
  - come valore massimo fra ρ_x e ρ_y, che può variare sullo stesso triangolo

- 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

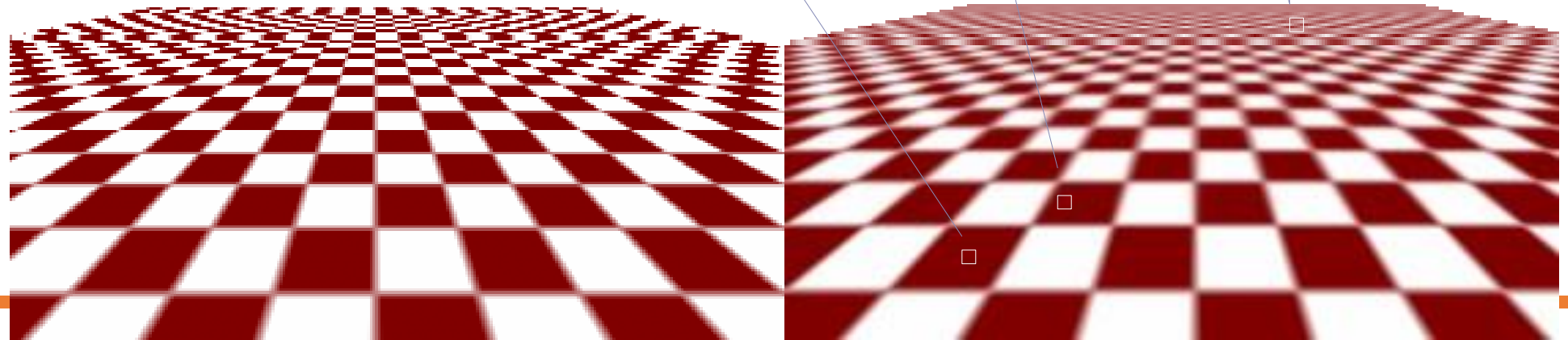
Caso Minification: MIP-mapping



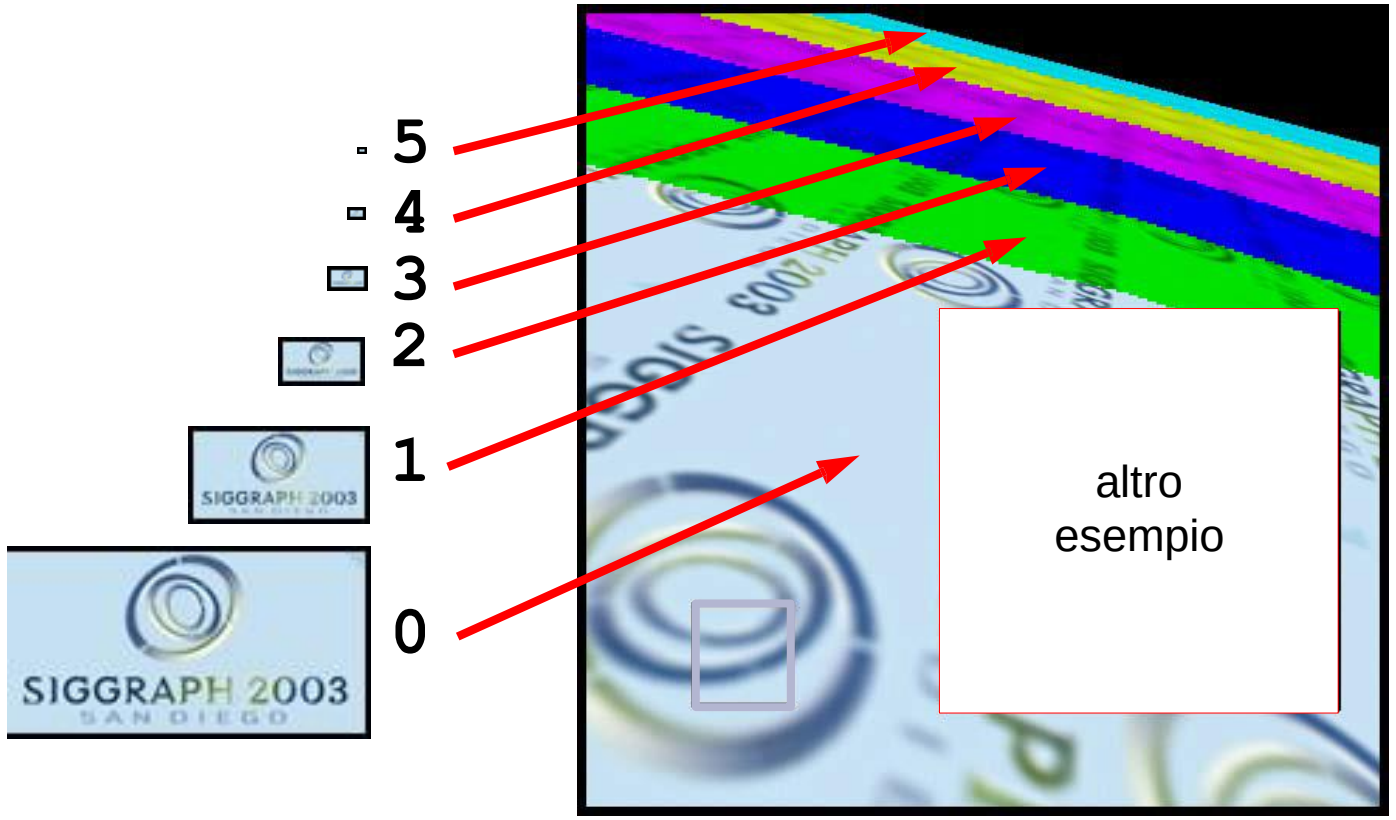
Bilinear interpolation

non risolve il problema

MIP-mapping



Caso Minification: MIP-mapping



La creazione di una raccolta di textures mipmapped per ogni immagine di texture è complicata da eseguire manualmente, ma per fortuna OpenGL è in grado di fare tutto il lavoro per noi con una sola chiamata a glGenerateMipmap() dopo che abbiamo creato una texture.

Mipmap

Supponiamo di avere una grande stanza con migliaia di oggetti, ognuno con una texture attaccata. Ci saranno oggetti lontani che hanno la stessa texture ad alta risoluzione, rispetto a quella che hanno gli oggetti vicini all'osservatore

Ciò produrrà artefatti visibili su piccoli oggetti, per non parlare dello spreco della larghezza di banda della memoria usando textures ad alta risoluzione su piccoli oggetti.

Per risolvere questo problema OpenGL utilizza un concetto chiamato mipmap:
Si tratta di una raccolta di immagini di texture in cui ogni texture successiva è due volte più piccola rispetto alla precedente.
L'idea alla base delle mipmap è questa: dopo una certa soglia di distanza dal visualizzatore, OpenGL utilizzerà una diversa texture mipmap che si adatta meglio alla distanza dall'oggetto.

Poiché l'oggetto è lontano, la risoluzione più piccola non sarà evidente per l'utente. OpenGL è quindi in grado di campionare i texel corretti e c'è meno memoria cache durante il campionamento di quella parte delle mipmap

Vertex shader

```
layout (location = 0) in vec3 aPos;   // the position variable has
attribute position 0
layout (location = 1) in vec4 aColor; // the color variable has attribute
position 1
layout (location = 2 ) in vec3 vertexNormal; // Attributo Normale 2
 layout (location =3 ) in vec2 coord_st; // Attributo texture
uniform mat4 Model;
uniform mat4 Projection;
uniform mat4 View;
out vec2  frag_coord_st;
void main()
{

      gl_Position = Projection*View*Model*vec4(aPos, 1.0);

      frag_coord_st=coord_st;


}
```

Fragment shader

```
uniform sampler2D id_tex;
in vec4 ourColor;
out FragColor;
 in vec2  frag_coord_st;
 uniform sampler2D id_tex;
void main()
{

    FragColor=mix(ourColor,texture(id_tex,vec2(frag_coord_st.x,frag_coord_st.y)),0.5);

}
```