Grafica 3D per i beni culturali: 3D and rendering

Lezione 2: 24 Febbraio 2011

Daniele Bernabei

Modeling/Acquisition e Rendering

Modeling/Acquisition

- **creazione** di un **modello digitale** che rappresenti una scena/oggetto di interesse (reale o non)
- *Strumento di conoscenza*

Rendering

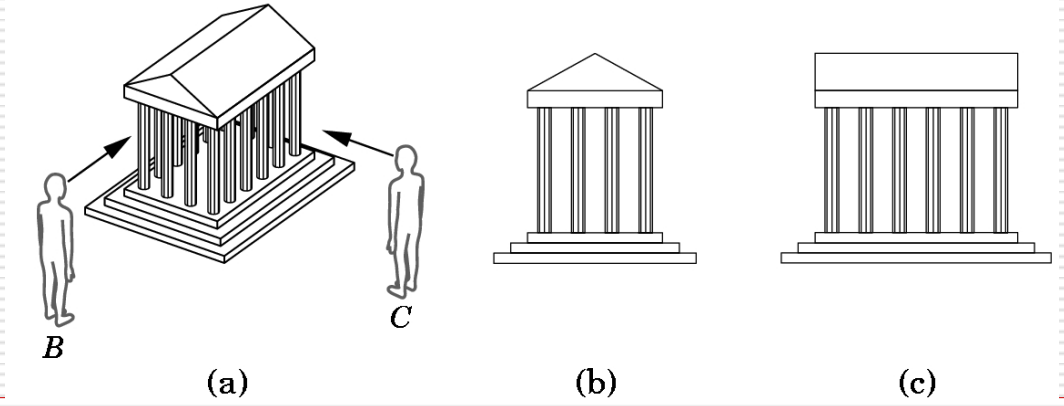
- tecniche che, a partire da un modello digitale, permettono di **produrre** delle **immagini** (statiche o animazioni dinamiche)
- *"Turning ideas into pictures"*
- *Strumento di comunicazione*

Abbiamo la rappresentazione, vogliamo vederla!

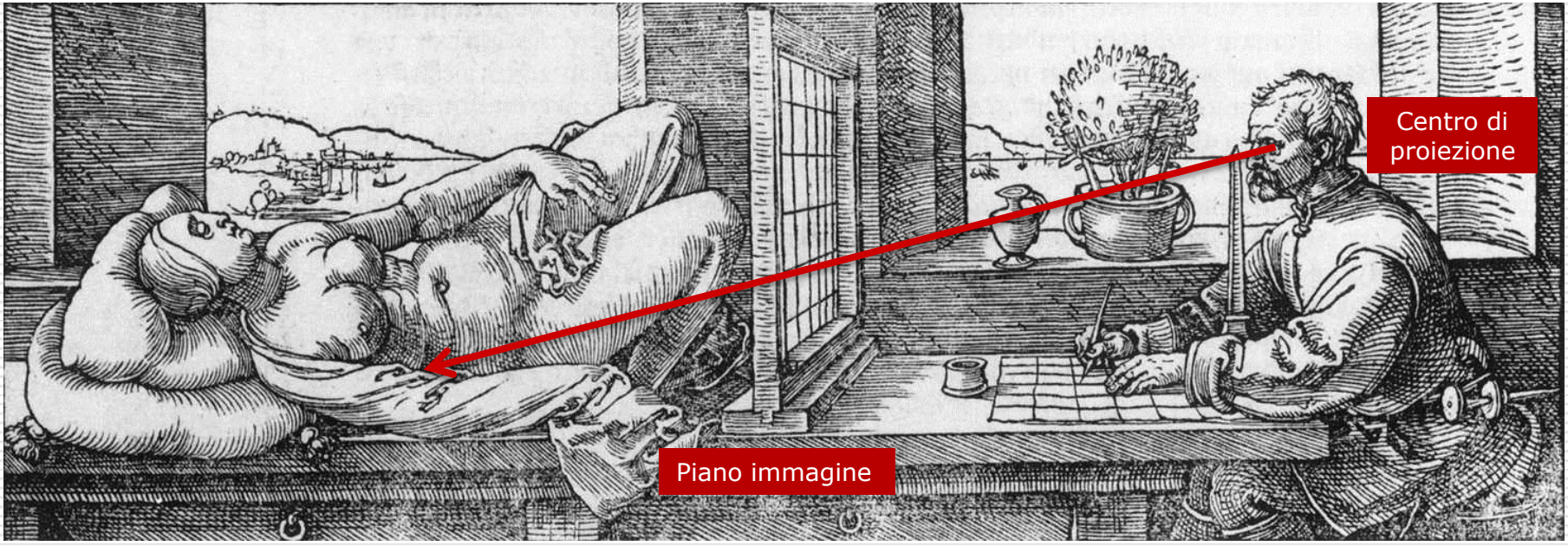
- Metafora fondamentale

Object vs viewer

- Object (*scene*): rappresentazione digitale (forma e caratteristiche) di un oggetto reale tridimensionale
- Viewer: *strumento* che permette di otternere da un object un immagine
- *Rendering* è il processo con cui un viewer genera un immagine a partire da una scene.



Insegnare al computer a disegnare



Griglia di Dührer: un raggio per ogni “pixel”

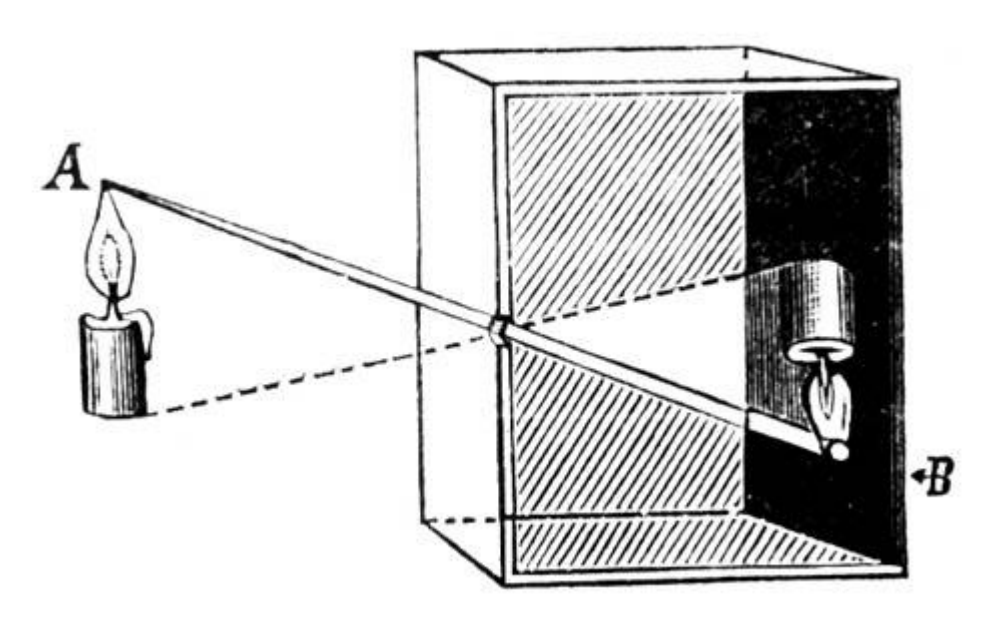
Un raggio e’:

- Un origine (corrisponde al **centro di proiezione**)
- Una direzione (determinata dall’elemento della griglia)

Geometric Optics: usare la metafora dei raggi per descrivere la luce.

Per approfondire: Image Plane

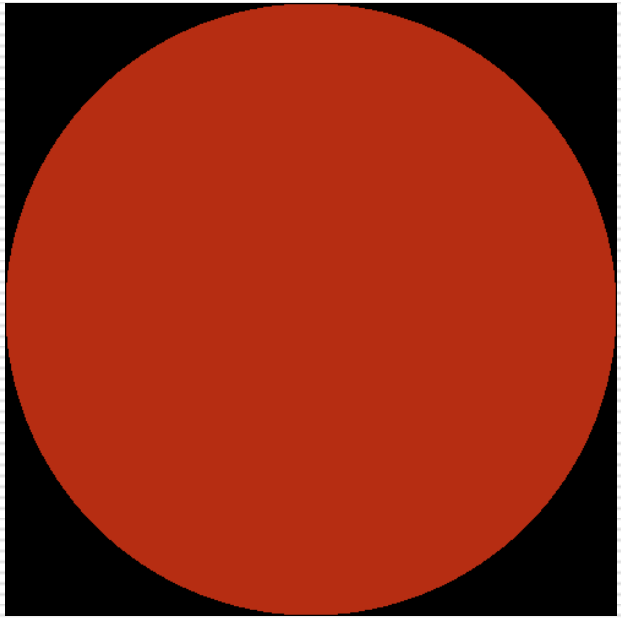
- Nelle macchine fotografiche il **piano immagine (image plane)** e' situato dietro il centro di proiezione.
  - Camera oscura: parete della scatola
  - Macchine fotografiche: pellicola o sensore CCD
  - Occhio umano: retina
- In questi casi l'immagine si inverte!

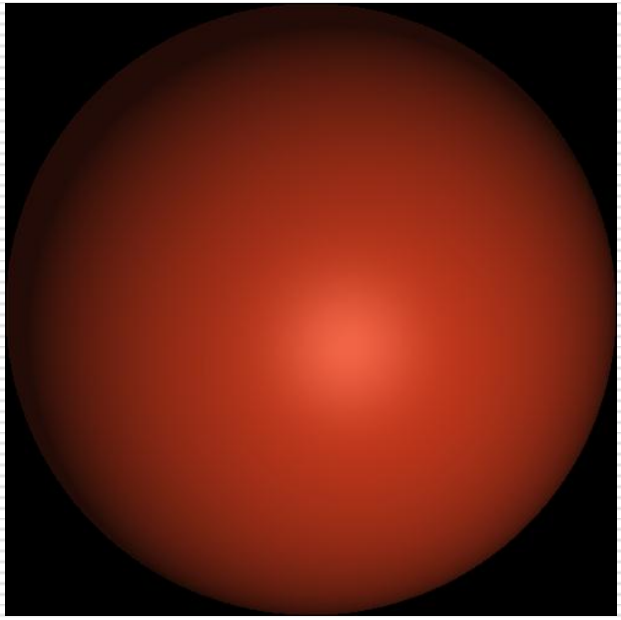


Intersezione raggio-oggetto

- Come sapere se un raggio vede un oggetto?
- Matematicamente, avviene se quel raggio (semi-retta) interseca l'oggetto!
- "Facile" per certe primitive geometriche:
  - Sfera
  - Cilindro
  - Cubo
- Risolvere "semplici" sistemi di equazioni.

E il colore?





Dipende da:

- proprietà ottiche del materiale dell'oggetto
- le fonti di luce che lo illuminano: le scene devono contere **luci**!
- (il punto da cui lo si guarda)

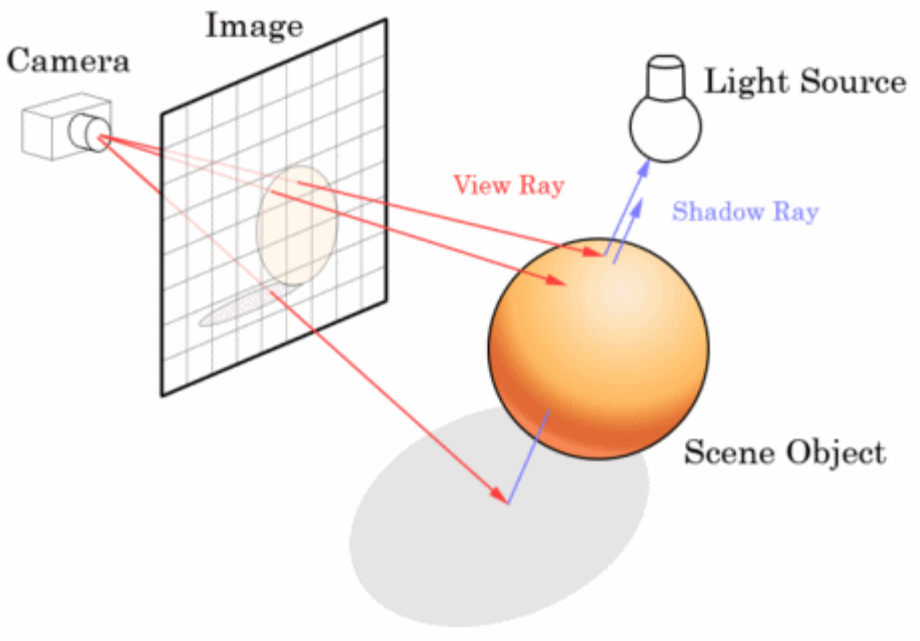
Lighting: calcolare il colore di un oggetto cosi' come appare ad un osservatore in una data scena.

Definire i materiali: Texturing

- **Texture mapping**: una o piu' immagini sono associate ad un oggetto digitale. Per ogni punto della superficie dell'oggetto sappiamo quale pixel della texture definisce il suo colore.
  - Mapping: superficie -> texture

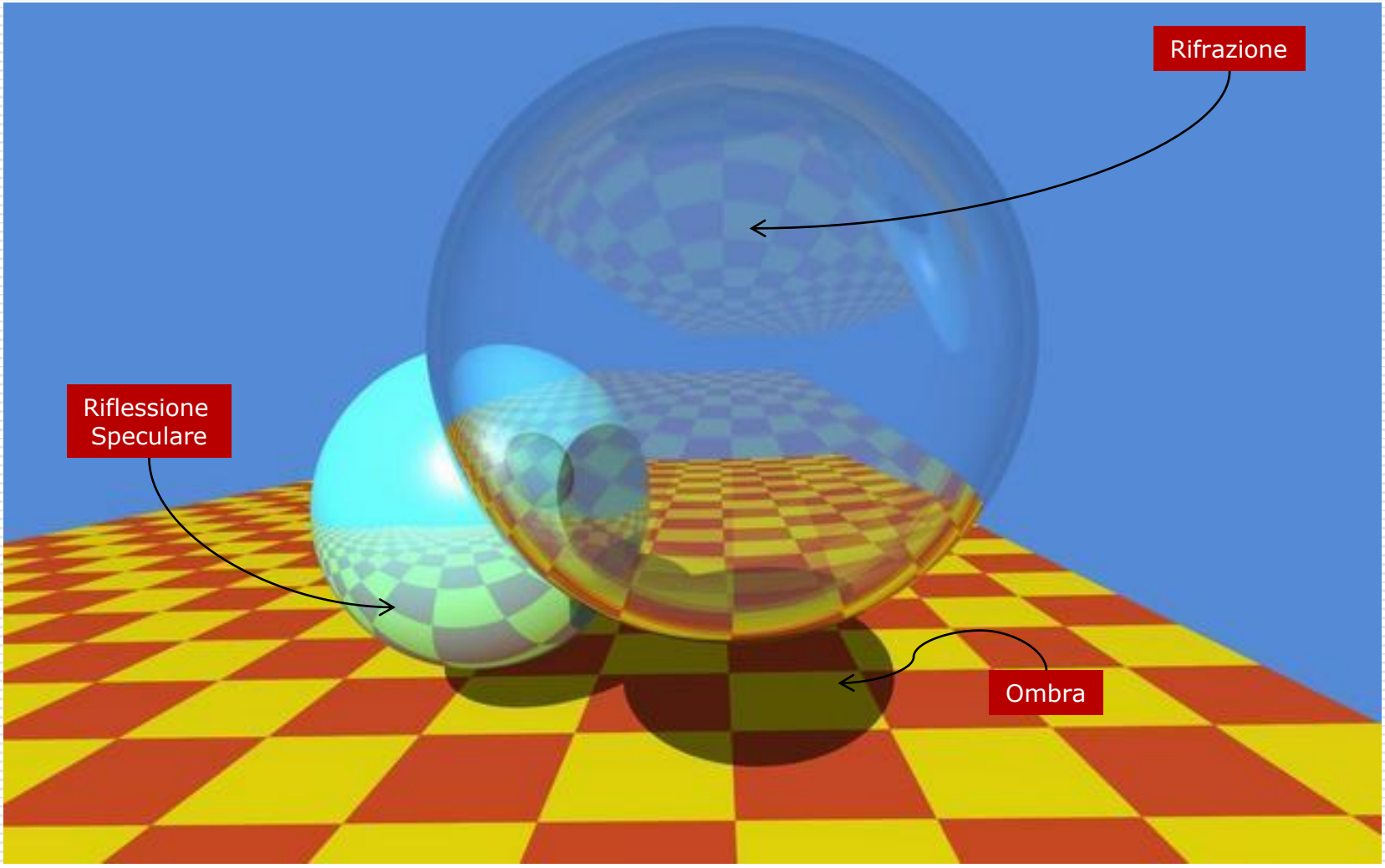
Il lighting e le fonti di luce

- Questione principale:
  - **Ombre**: Tra il punto di intersezione e la **fonte di luce** c'e' un altro oggetto? [facile: spara un'altro raggio!]



- Altri effetti possibili:
  - **Rifrazioni**
    (I raggi deviano)
  - **Riflessioni speculari**
    (I raggi rimbalzano)

Ray-tracing (**Turner Whitted**)



Ray Tracing: Caratteristiche

- È un paradigma di **rendering globale**
- Modella facilmente le ombre portate
- Modella bene le inter-reflection speculari e rifrattive
- Anche la inter-reflection diffusa (es. La riflessione di un muro), ma a costo di tempo elevato

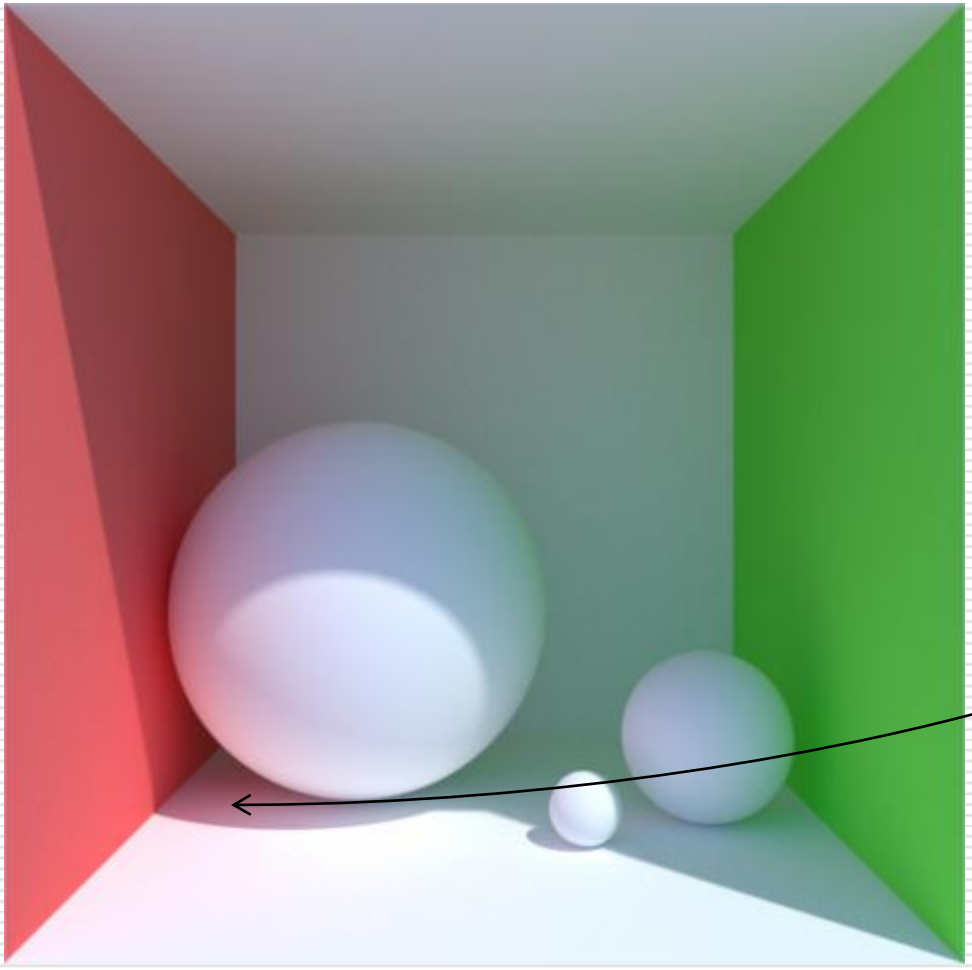
Problemi

- Per ogni pixel (HD: 2 milioni)
  - Calcola l'intersezione con **ogni** oggetto
- Per calcolare anche le ombre il costo raddoppia…
  - Per le inter-reflection piu' complicate il costo si moltiplica esponenzialmente…
- Eseguire il rendering di una scena puo' occupare da minuti a ore!

Altri paradigmi (**Illuminazione globale**)

- **Photon mapping**
  - Si “simula” la luce nella sua vera direzione, partendo dalle sorgenti e interagisce con le superfici.
  - Illumina anche zone non viste (non necessarie)
- **Radiosity**
  - Le superfici sono partizionate in pezzetti (*patch)*
  - Si pre-calcola quanta parte dell’energia che esce da una patch arriva ad ogni altra patch della scena
  - Ottime riflessioni diffuse, ma impossibile catturare riflessioni speculari!

Riflessione diffusa



Il colore del pavimento si tinge di rosso a causa della parete vicina

E le primitve geometriche?

- Il mondo non e' fatto solo di cubi e sfere…
- E' possibile modellare una scena sottraendo e sommando primitive semplici: **Constructive Solid Geometry**
  - Poca espressivita'. Esistono paradigmi migliori. (es. **NURBS**: piccoli elementi curvi)

Facce poligonali

- Faccia: piccolo poligono **planare** definito da **vertici**

- Facce con quattro o piu' vertici possono **non** essere planari…

- … ma per tre punti passa uno e un solo piano.

Cambiamo paradigma: la Proiezione

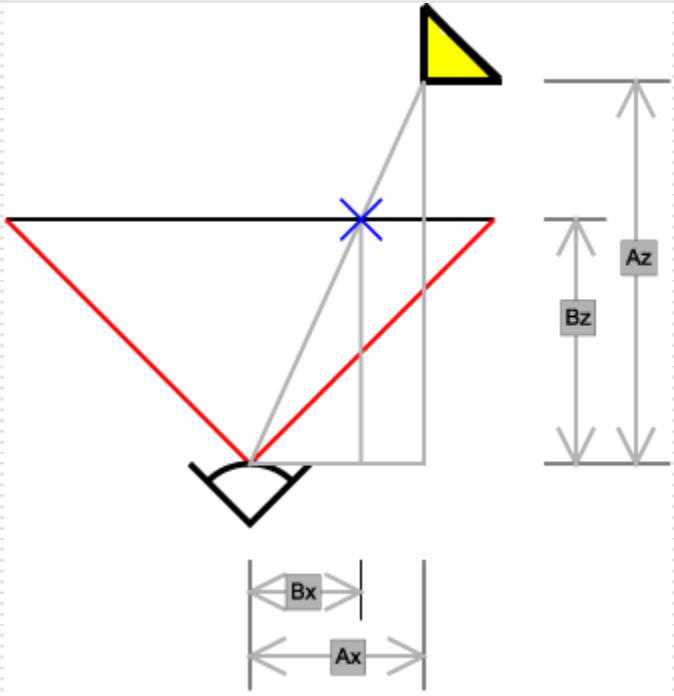
- Le facce sono primitive piu' versatili
- Se conosco la **proiezione** dei **vertici**, posso disegnare i **bordi**, riempire le **facce**

Ribaltare il paradigma di rendering

- Ray Tracing:
  - Ogni raggio deve “cercare” nella scena gli oggetti che vede
- Proiezione:
  - Ogni oggetto “si calcola” la zona del piano immagine che occupa

Proiettare vertici: un accenno

- Se il punto di proiezione e' nell'**origine** del mio sistema di coordinate….
- e il piano immagine si trova perpendicolare all'asse delle Z…



$$B_x = A_x \frac{B_z}{A_z}$$

Ovvero: la formula e' incredibilmente semplice e "veloce" da calcolare!

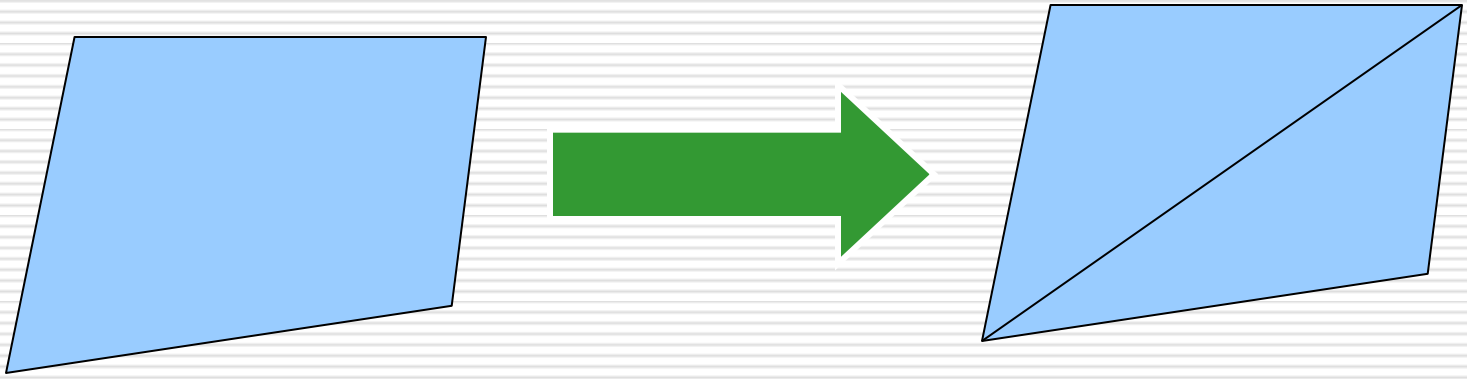
Proiezione

- Il costo ora e':
  - Per ogni vertice di ogni triangolo di ogni oggetto, calcola la sua proiezione sul piano immagine.
- I pixel che non vedono niente non vengono attivati.
- La proiezione e'
  - Computazionalmente **poco costosa**
  - Matematicamente **identica** per ogni vertice.
  - **Indipendente** per ogni vertice.
- Posso quindi progettare hardware per risolvere questo compito in **parallelo**.

Transforms: calcolare coordinate di vertici

- Proiezione sulla tela bidimensionale
  - Trasformazione 3D -> 2D
  - “schiacciare le coordinate sul piano immagine”
- Scena composta di oggetti multipli
  - Trasformazione 3D -> 3D
  - “spostare le coordinate”
- Le trasformazioni:
  - Ruota
  - Transla
  - Scala
  - Proietta

Posso davvero usare i triangoli come primitiva universale?



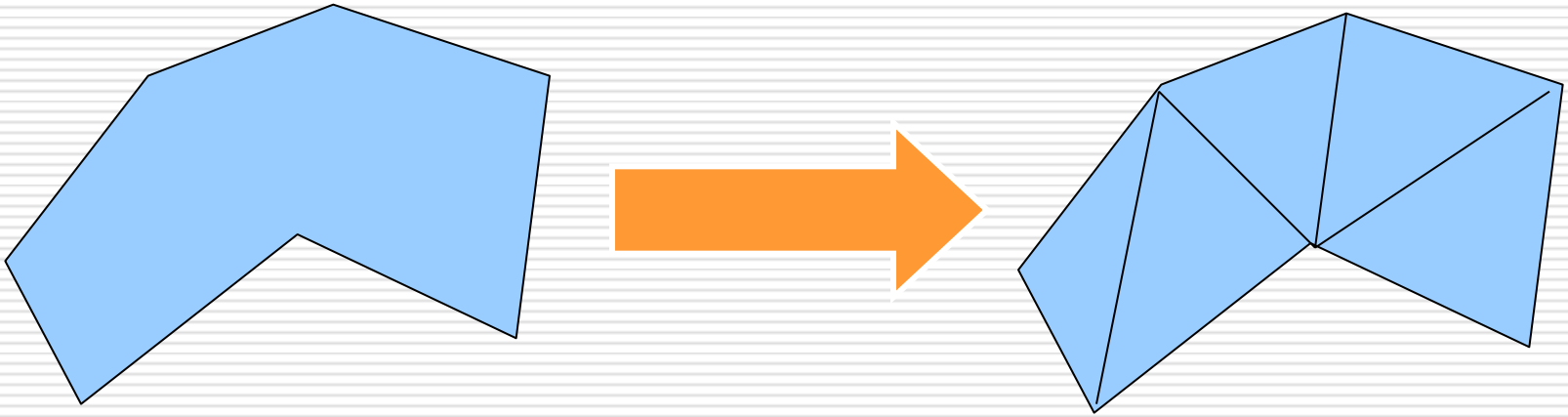
un quadrilatero?

due triangoli!

"quad"

"diagonal split"

Posso davvero usare i triangoli come primitiva universale?

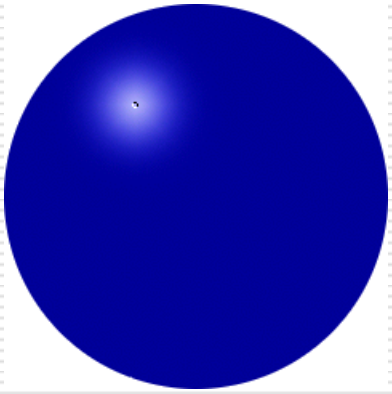


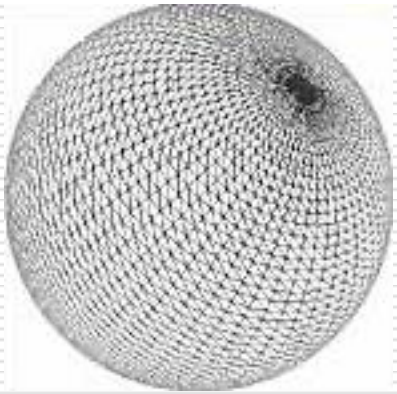
un poligono a *n* lati?

(n−2) triangoli!

triangolarizzazione di poligono: (non un problema del tutto banale...)

Posso davvero usare i triangoli come primitiva universale?

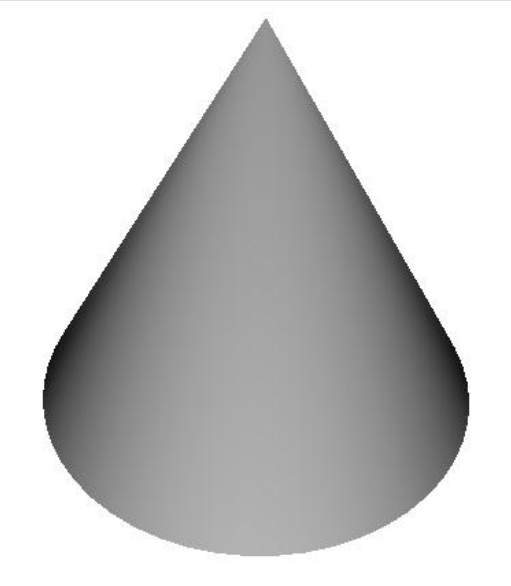


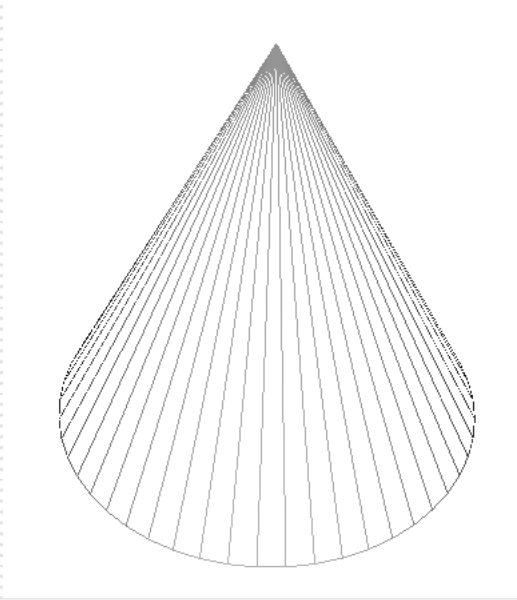


la superficie di
un solido geometrico,
per es. una sfera?

triangoli!

Posso davvero usare i triangoli come primitiva universale?

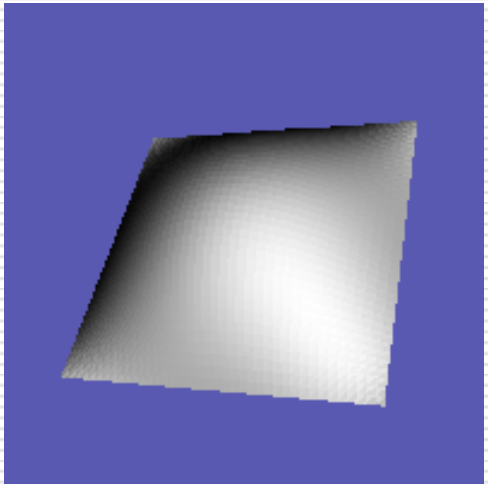


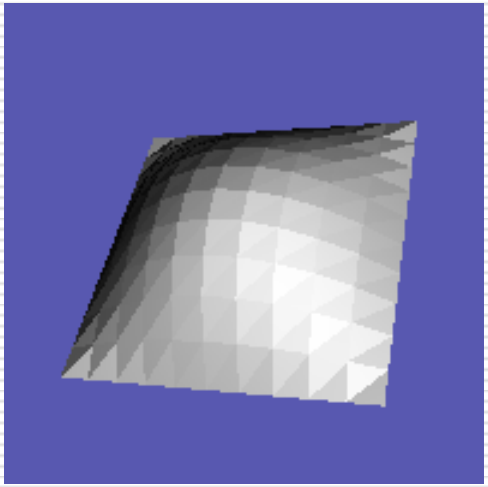


la superficie di
un solido geometrico,
per es. un cono?

triangoli!

Posso davvero usare i triangoli come primitiva universale?



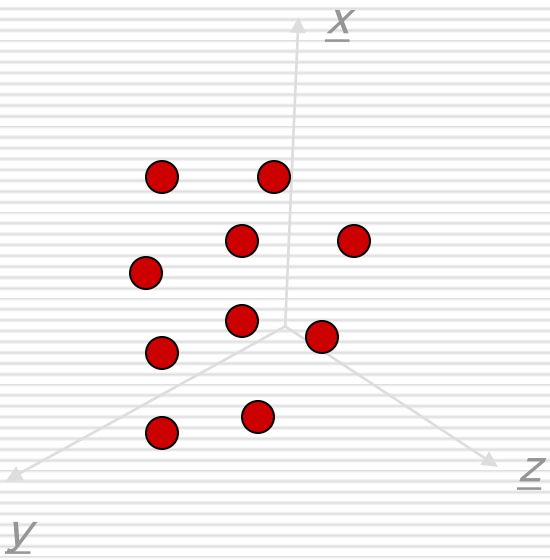


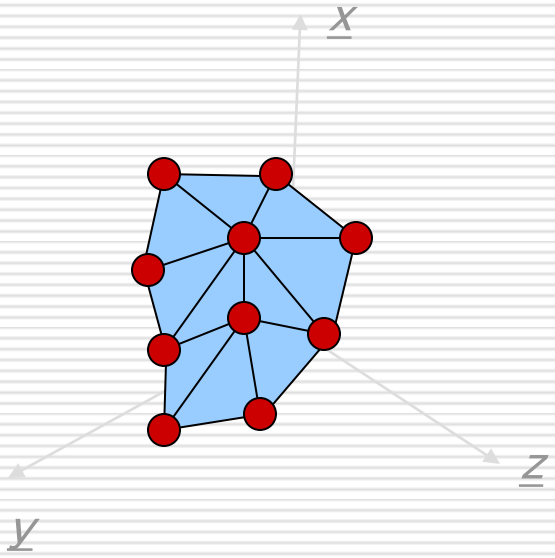
una superficie curva parametrica?

triangoli!

questo è facile. Il contrario, che qualche volta è utile, MOLTO meno

Posso davvero usare i triangoli come primitiva universale?





nuvola di punti ?
(*points cloud*)

triangoli!

problema molto studiato
e (nel caso generale) difficile

Posso davvero usare i triangoli come primitiva universale?

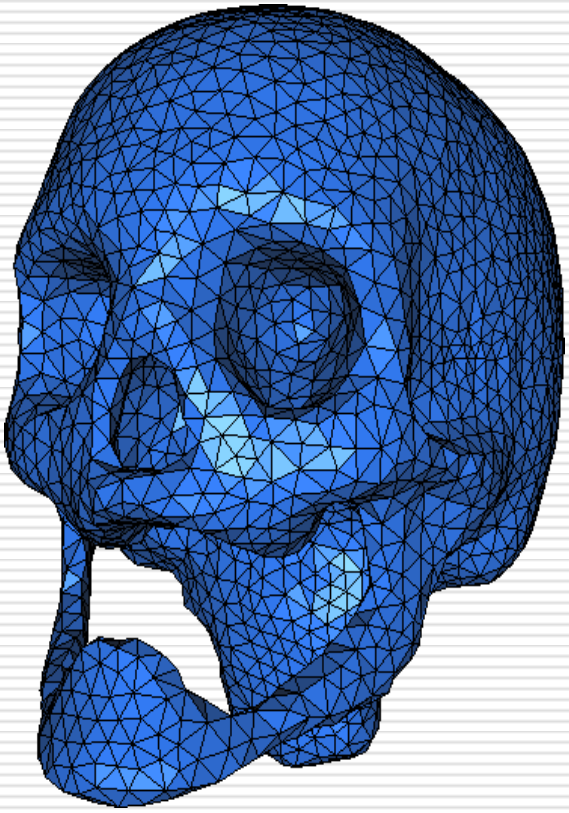
superfici implicite?
f(x,y,z)=0

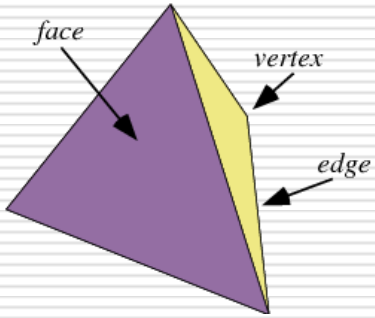
triangoli
che definiscono
la superficie
esplicitamente

triangoli!

Dai triangoli alla **Mesh**

- Chiameremo Mesh una struttura poligonale
- Le nostre saranno **Triangular Meshes**





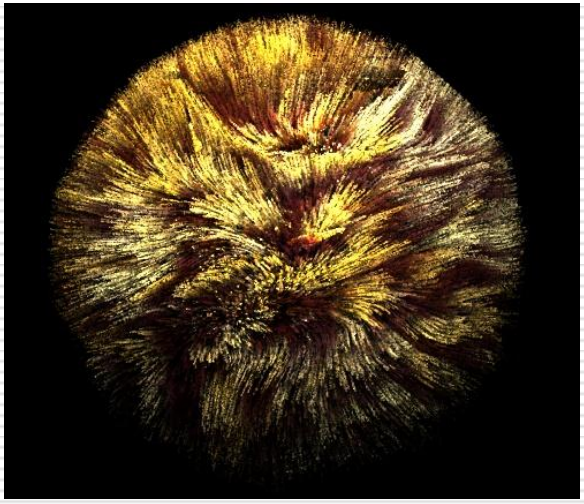
- La topologia:
  - Vertex
  - Face (Triangle)
  - Edge

Un limite di questo approccio

- Non sempre e' semplice modellare le entità da rappresentare con triangoli...
  - esempi:
    - nuvole
    - fuoco
    - pelliccia

by Niniane Wang (non real time)

by N. Adabala uni florida (non real time)



by M. Turitzin and J. Jacobs Stanford Uni (real time!)

Per dirla tutta sulle primitive di rendering

- Triangoli
  - ok, abbiamo capito
- Quads
  - in un certo senso, perchè diventano triangoli al volo
- Segmenti
- Punti

Tutto l'hardware è progettato e ottimizzato principalmente per questo caso

Per dirla tutta sulle primitive di rendering

- Triangoli
  - ok, abbiamo capito
- Quads
  - in un certo senso, perchè diventano triangoli al volo
- Segmenti
- Punti

utile ad esempio per fare rendering di capelli peli etc

(ma non è l'unico sistema e non è detto che sia il migliore)

nVIDIA tech demo at SIGGRAPH 2001
nVIDIA quadro

Per dirla tutta sulle primitive di rendering

- Triangoli
  - ok, abbiamo capito
- Quads
  - in un certo senso, perchè diventano triangoli al volo
- Segmenti
- Punti

utili ad esempio
per particle systems

La Pipeline di Rendering

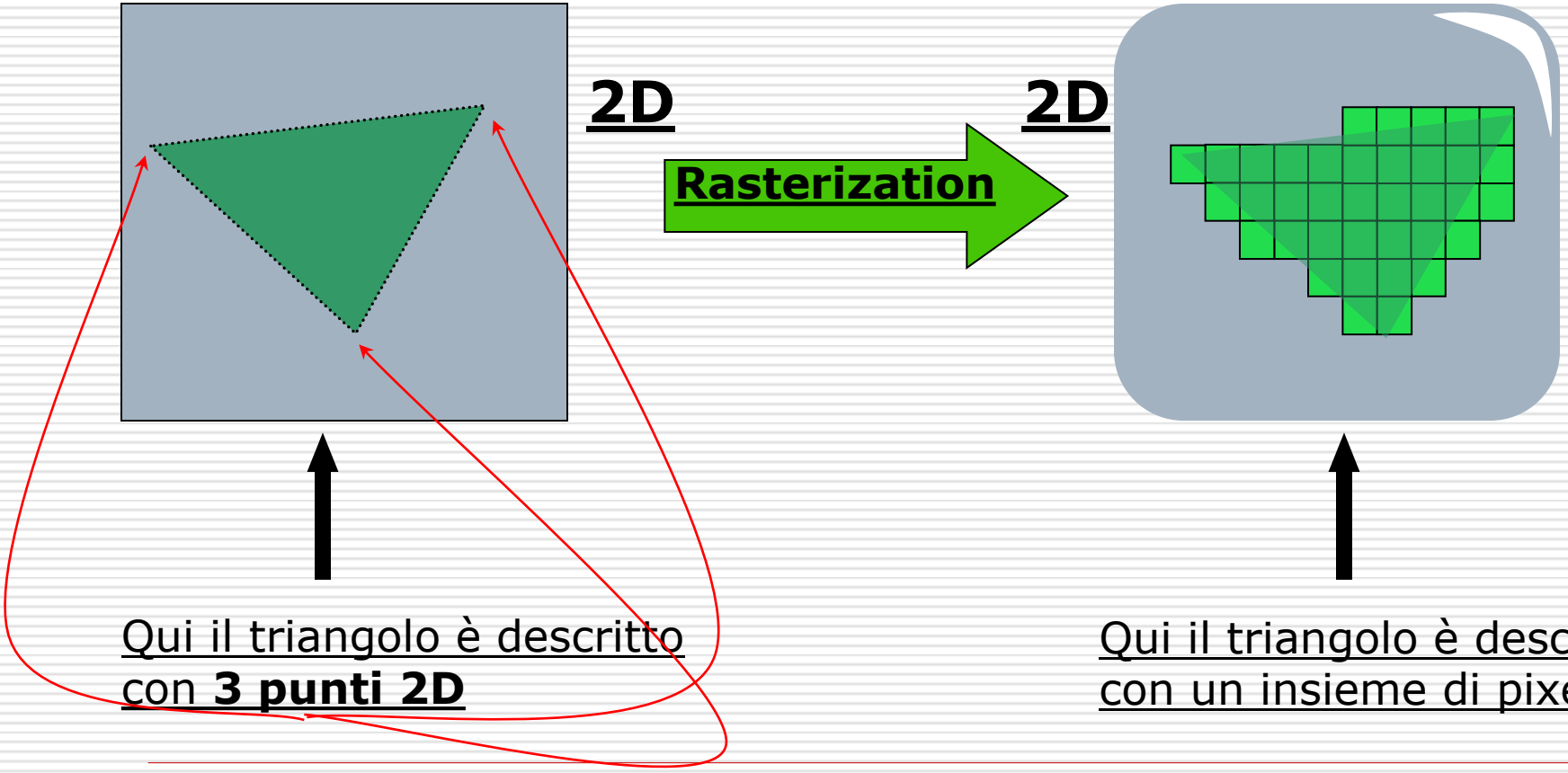
- Def: La serie di passi che trasformano la descrizione geometrica di una scena in un'immagine sullo schermo

- La pipeline che ci interessa e' basata sulla **proiezione** e sulla **rasterizzazione**. La chiamiamo **Transform&Lighting**.

- E' la pipeline presente nell'hardware grafico dei nostri PC (e smartphone!)

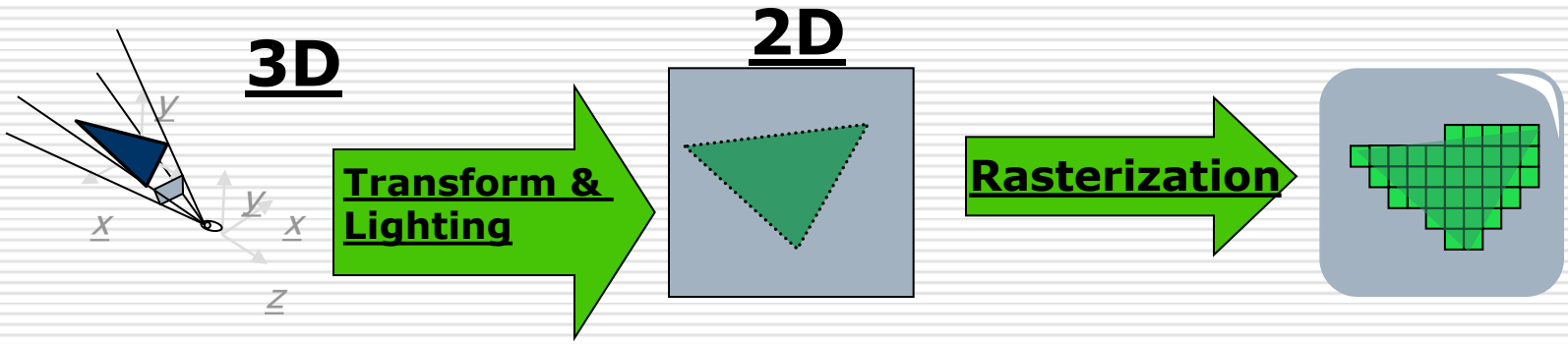
La **rasterizzazione**

- Una volta eseguite tutte le trasformazioni (inclusa la proiezione)



Cenni storici sull'hardware grafico (1)

- Lo schema base della pipeline di rendering

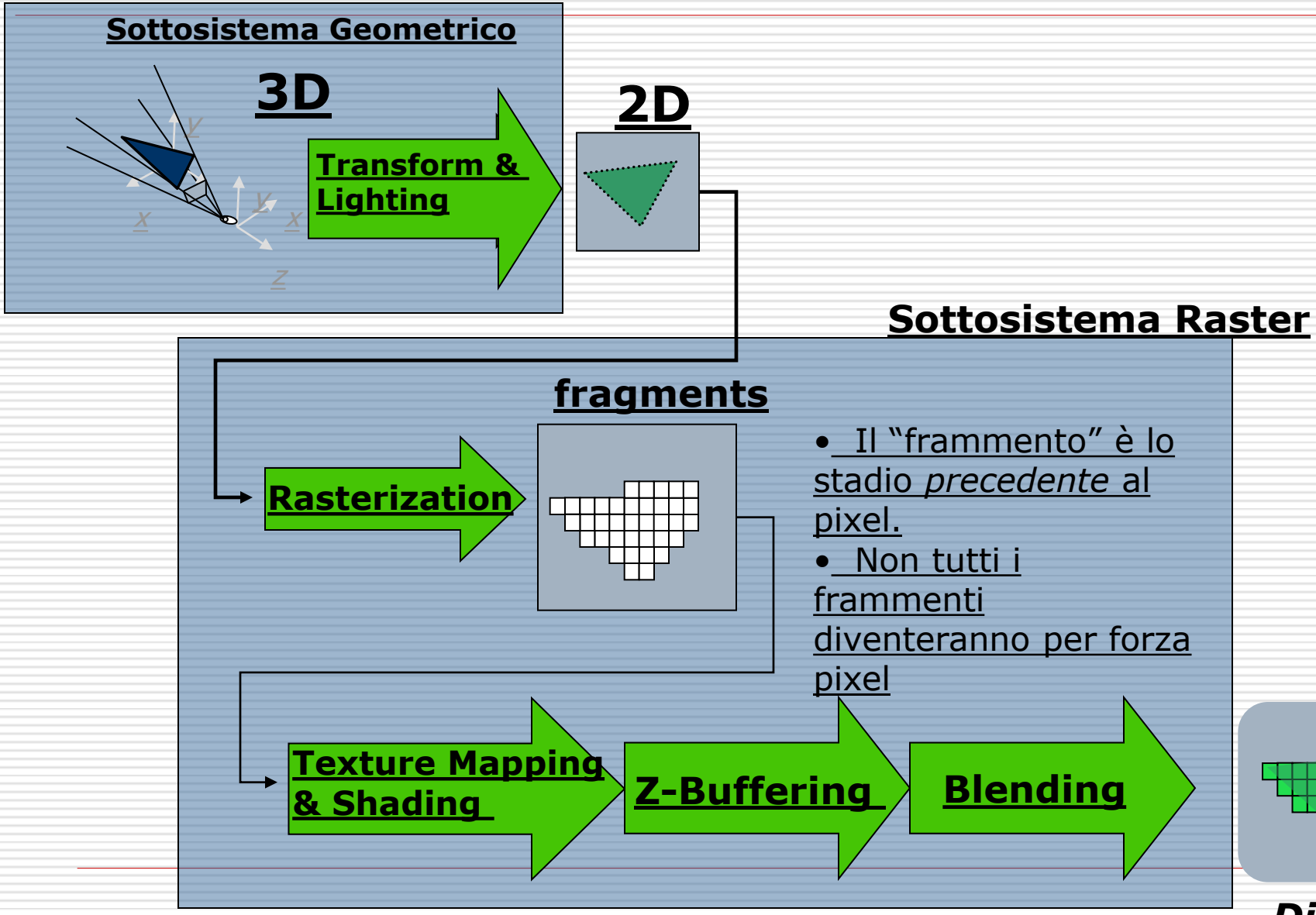


- Fino a metà degli anni 90 tutte le parti della pipeline erano eseguite dalla CPU

Il lighting nel paradigma T&L

- L'illuminazione è "locale" alla primitiva (triangolo, punto, linea) renderizzata
- L'illuminazione viene calcolata via via che si disegnano le primitive
- ...niente ombre portate
- ...niente inter-reflection (luce che rimbalza su un oggetto e ne illumina un altro)
- detto in altro modo: il paradigma di rendering raster-based **non** codifica il concetto di scena

Qualche altro dettaglio....



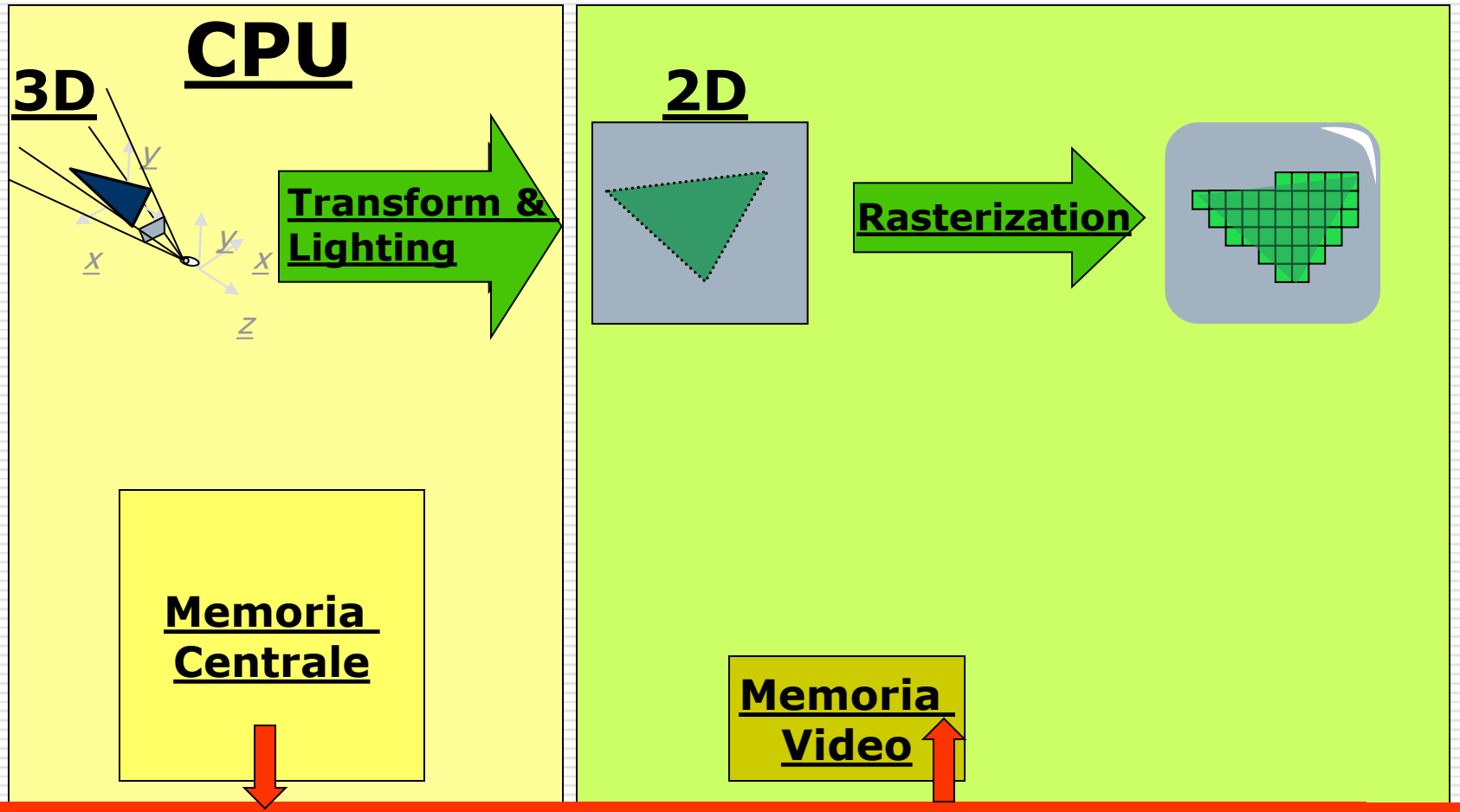
Z-Buffering, Blending

- **Z-buffering**: più "oggetti" possono essere proiettati sulle stesse coordinate, ed ognuno genera un frammento. Lo Z-buffering è un algoritmo che determina quale dei frammenti in conflitto diventerà il pixel

- **Blending** (blending = mischiare, fondere insieme): capita che il colore del pixel debba essere ottenuto mescolando il colore di più frammenti (relativi alla medesima posizione). Es: materiali semitrasparenti

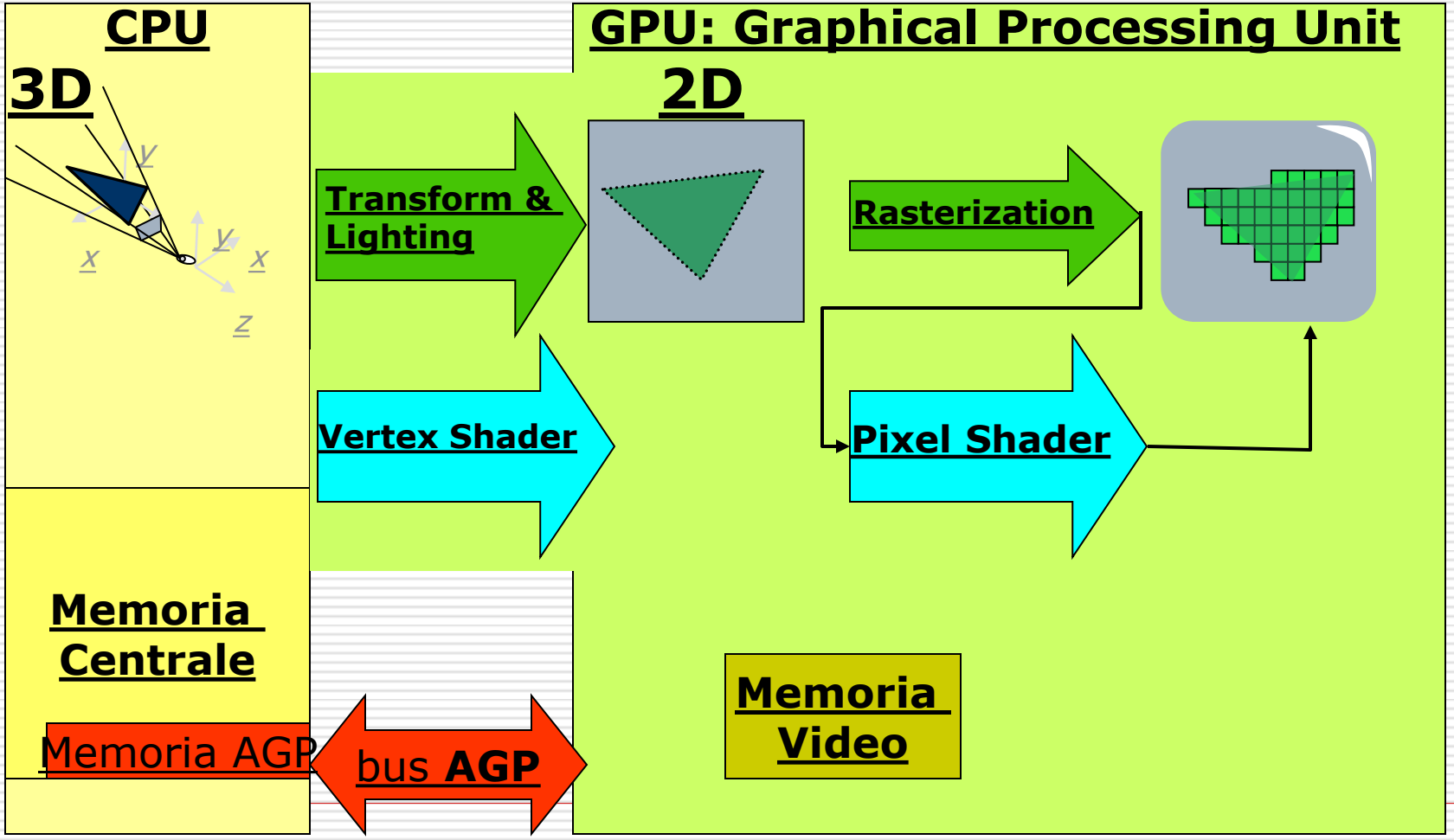
Cenni storici sull'hardware grafico (2)

- 1995-1997: 3DFX Voodoo



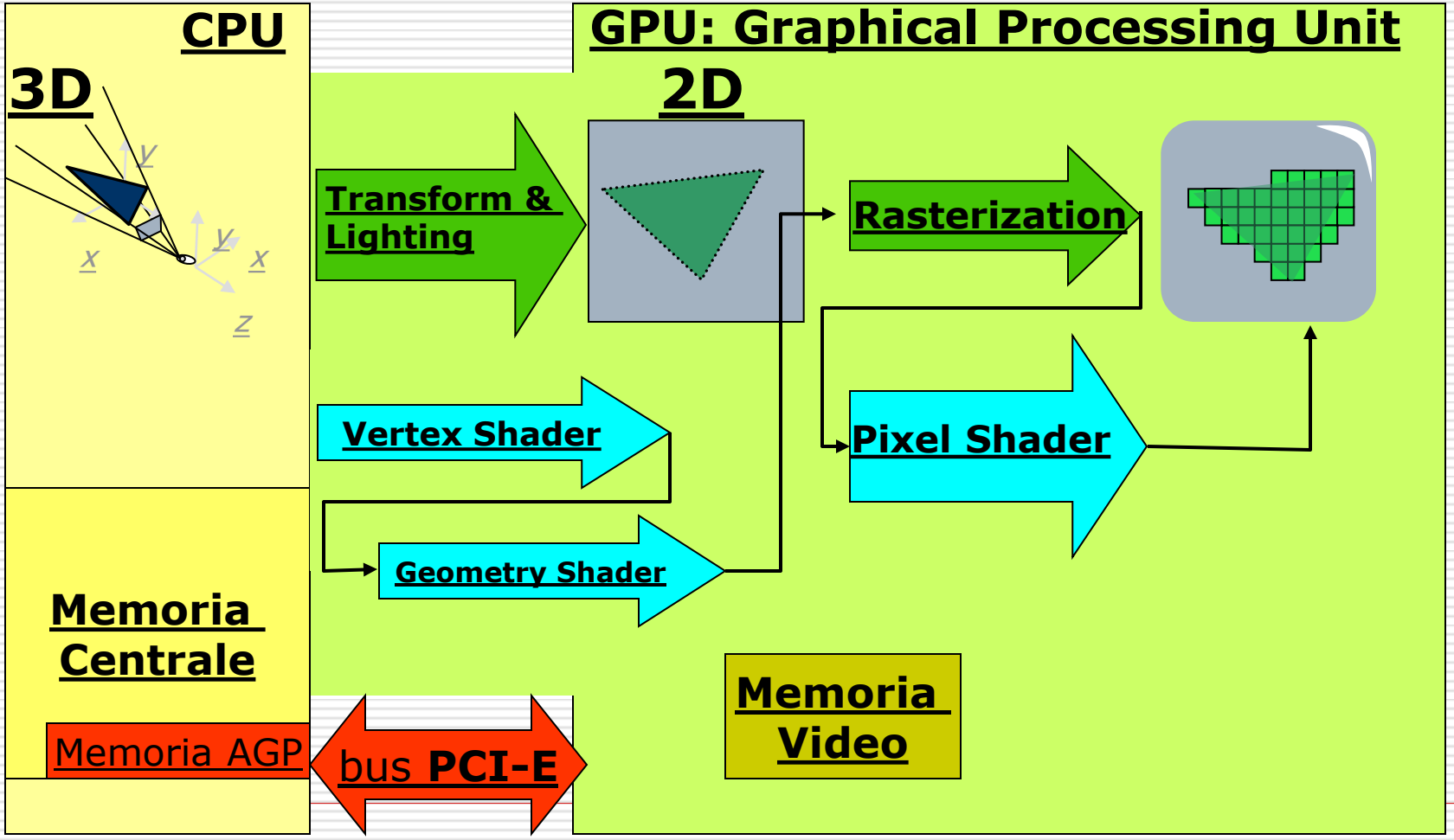
Cenni storici sull'hardware grafico (3)

- 2002-3: Pixel (o fragment) Shader (GeForceFX, Radeon 8500)



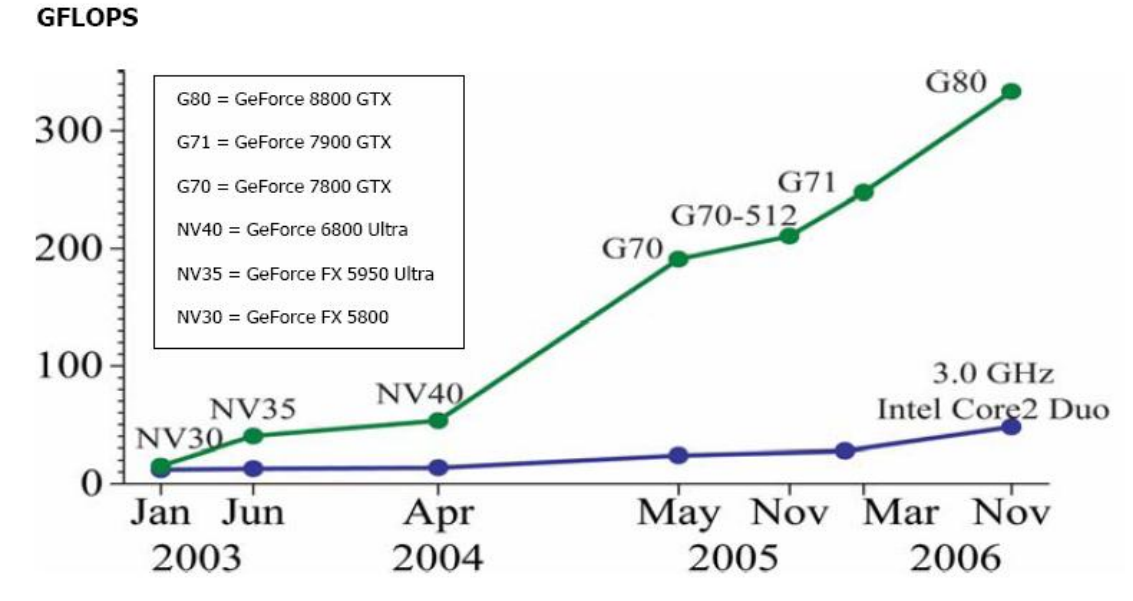
Cenni storici sull'hardware grafico (4)

- 2007: geometry shader, stesso hardware per tutti gli shaders (NVidia 8800)



Note sulla GPU odierne

- Modello di computazione
  - SIMD (single intruction multiple data): sfrutta l'alto grado di parallelismo *insito nella pipeline*
- Po Vidia



- Ecco il perchè del General Purpose Computation on GPU.

General Purpose Computing on GPUs

- Ogni algoritmo facilmente parallelizabile puo' diventare 10 o 100 volte piu' veloce se eseguito su GPU.
- Ray Tracing, Photon Mapping e altri paradigmi di illuminazione globale tornano ad essere possibili.
- Non siamo ancora pronti pero' ad abbandonare la pipeline T&L

Note finali

- Il realismo costa
  - Algoritmi di illuminazione globale richiedono ancora troppo tempo
- L'hardware grafico e' stato pensato per T&L su triangoli
  - Nell'ambito beni culuturali questo spesso e' una limitazione

Next in line…

Next lesson:

- 3D Scanning: first part

Contacts:

Matteo Dellepiane

c/o ISTI-CNR Via G. Moruzzi 1

56124 Pisa (PI)

Tel. 0503152925

E-Mail: dellepiane@isti.cnr.it

Personal website: http://vcg.isti.cnr.it/~dellepiane/

VCG website: http://vcg.isti.cnr.it