Grafica Computazionale

La Pipeline Grafica

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

noi siamo qui

Informazione

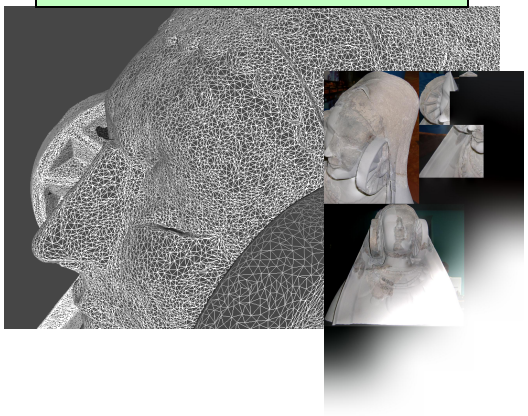
mondo reale (es: 3D scans)

creazione (es: videogames)

Calcolo (es: Sci-Vis)

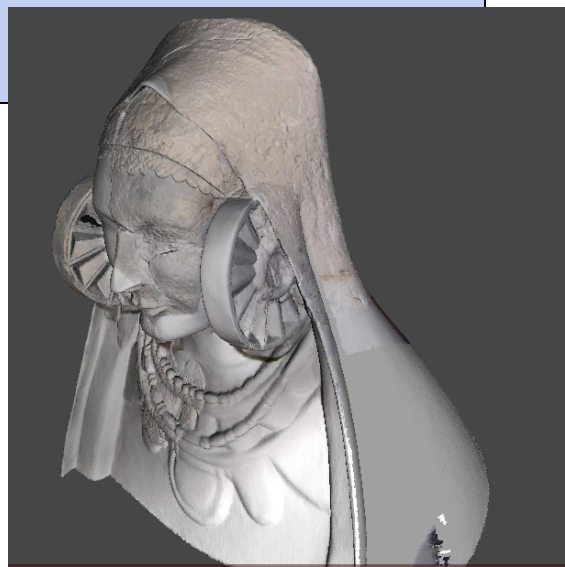
Modellazione

Trovare la rappresentazione più adeguata per...



Visualizzazione (rendering)

Trasformare la rappresentazione in immagine(i) sullo schermo

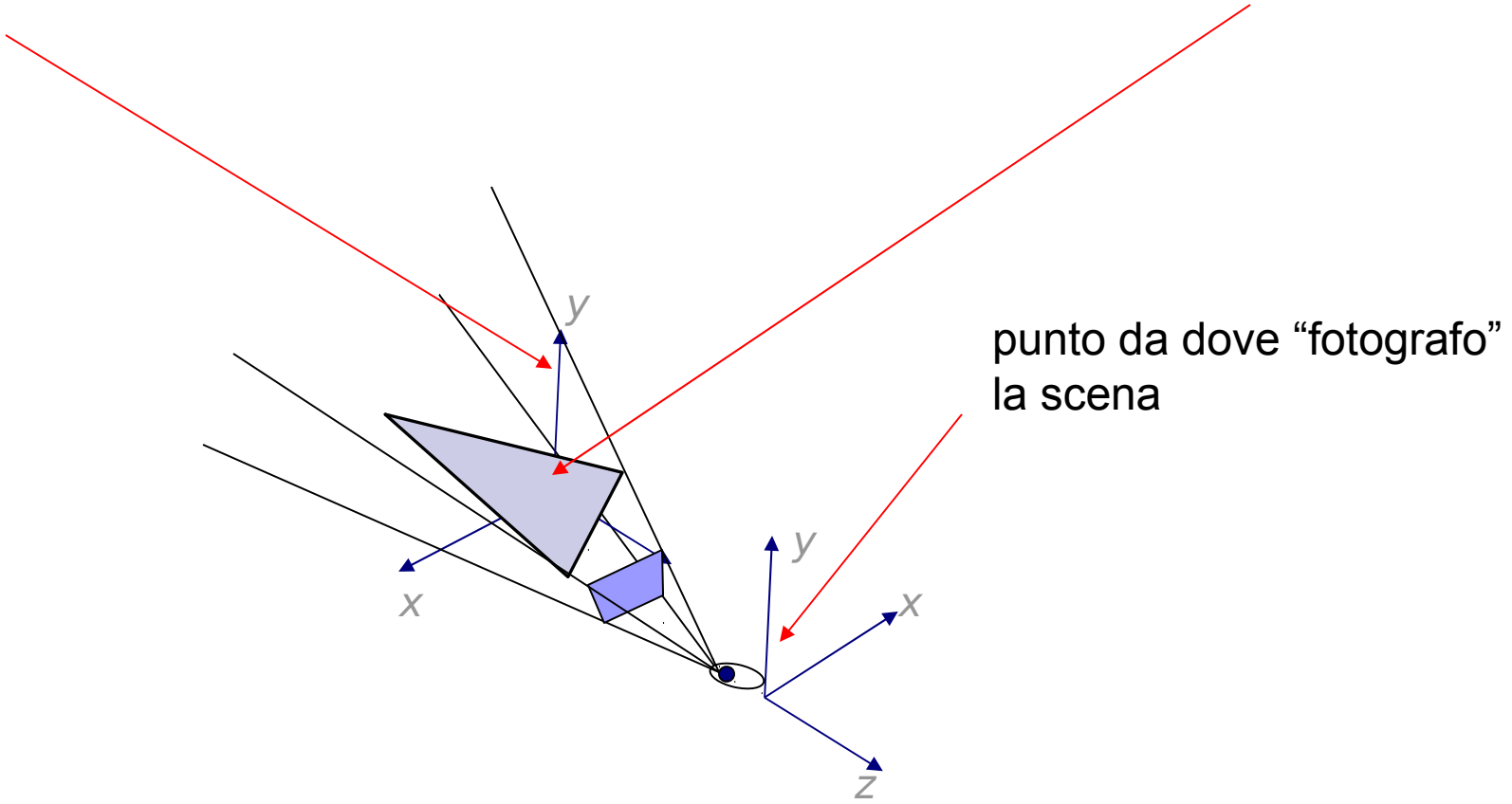


La Pipeline di Rendering

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

Esempio di pipeline di rendering

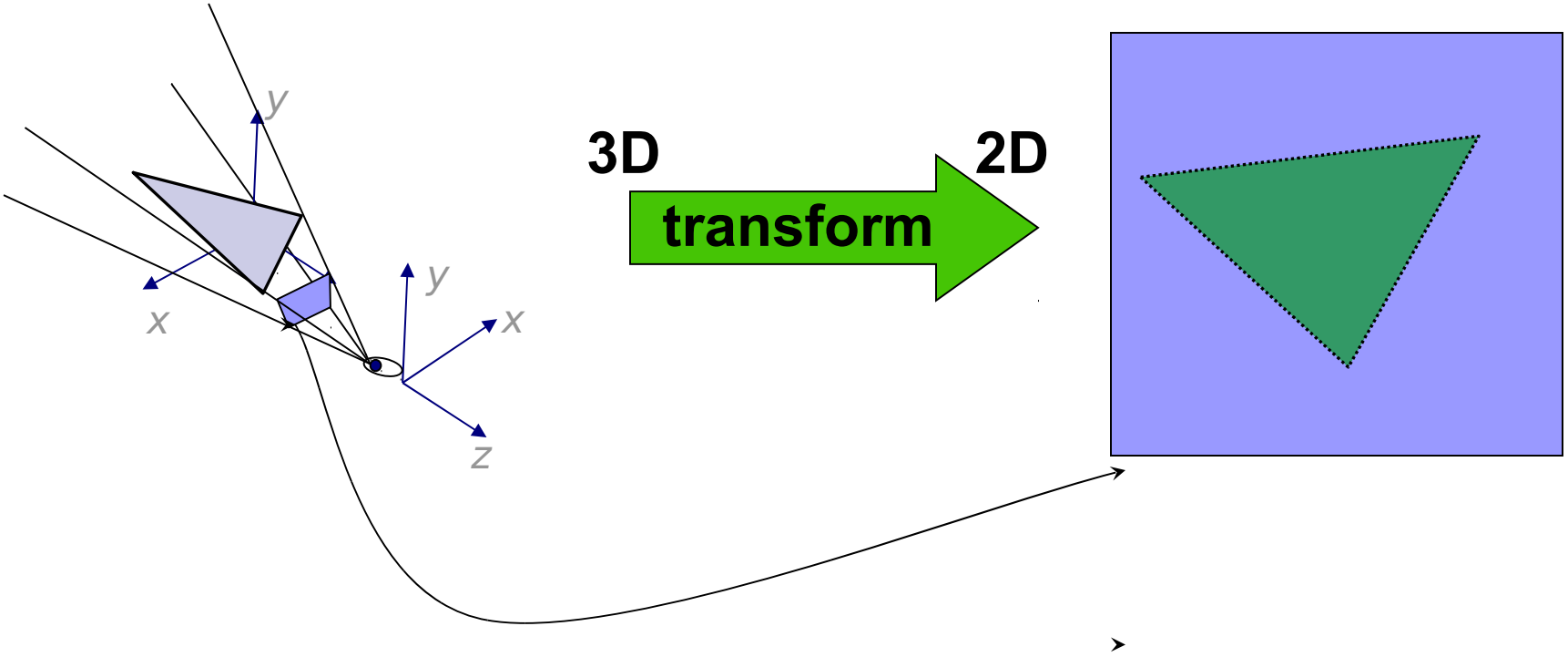
- Esempio su una scena fatta da un **triangolo** nello spazio 3D



- Il triangolo è descritto dalle posizioni dei suoi 3 vertici V0,V1 e V2: 3 punti in 3 dimensioni
- Ma lo schermo è piatto!...

Primo passo: **Transform**

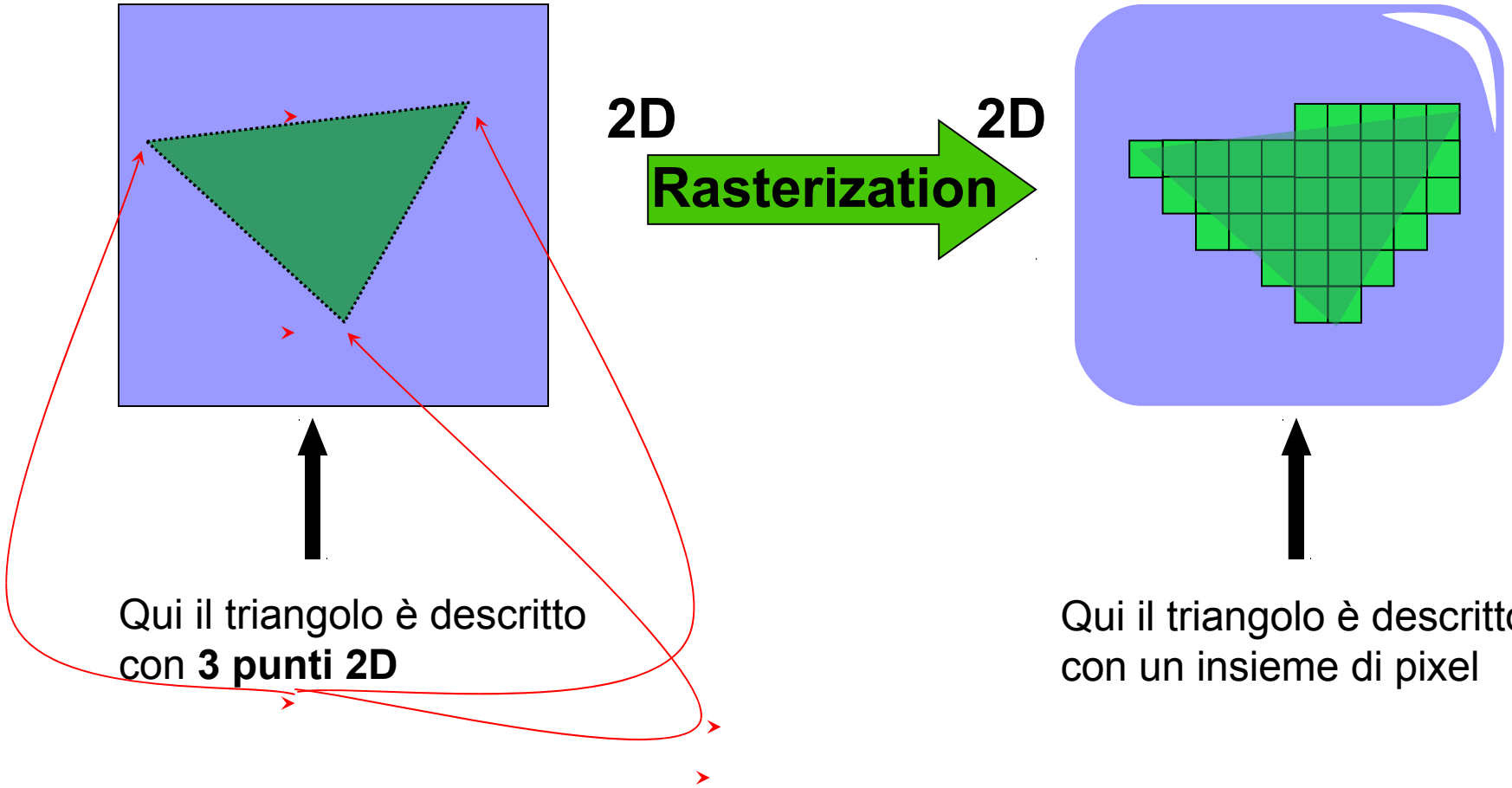
- Trasformare le coordinate tridimensionali in coordinate bidimensionali



- Il risultato dipende **da dove** fotografo la scena...
- ..e con cosa: uno zoom? un grandangolo?

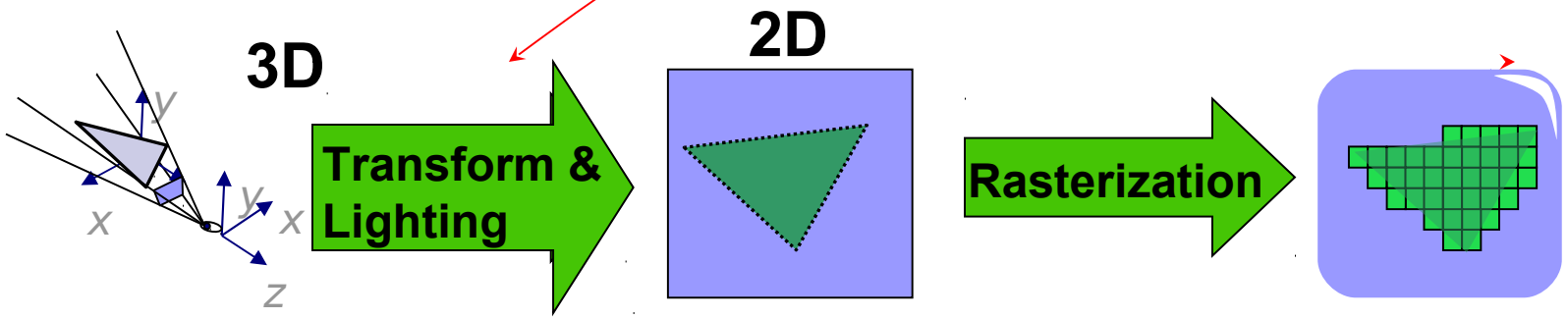
Secondo passo: **Rasterization**

- Trasformare la descrizione per vertici 2D del triangolo nell'insieme di pixel "da accendere"

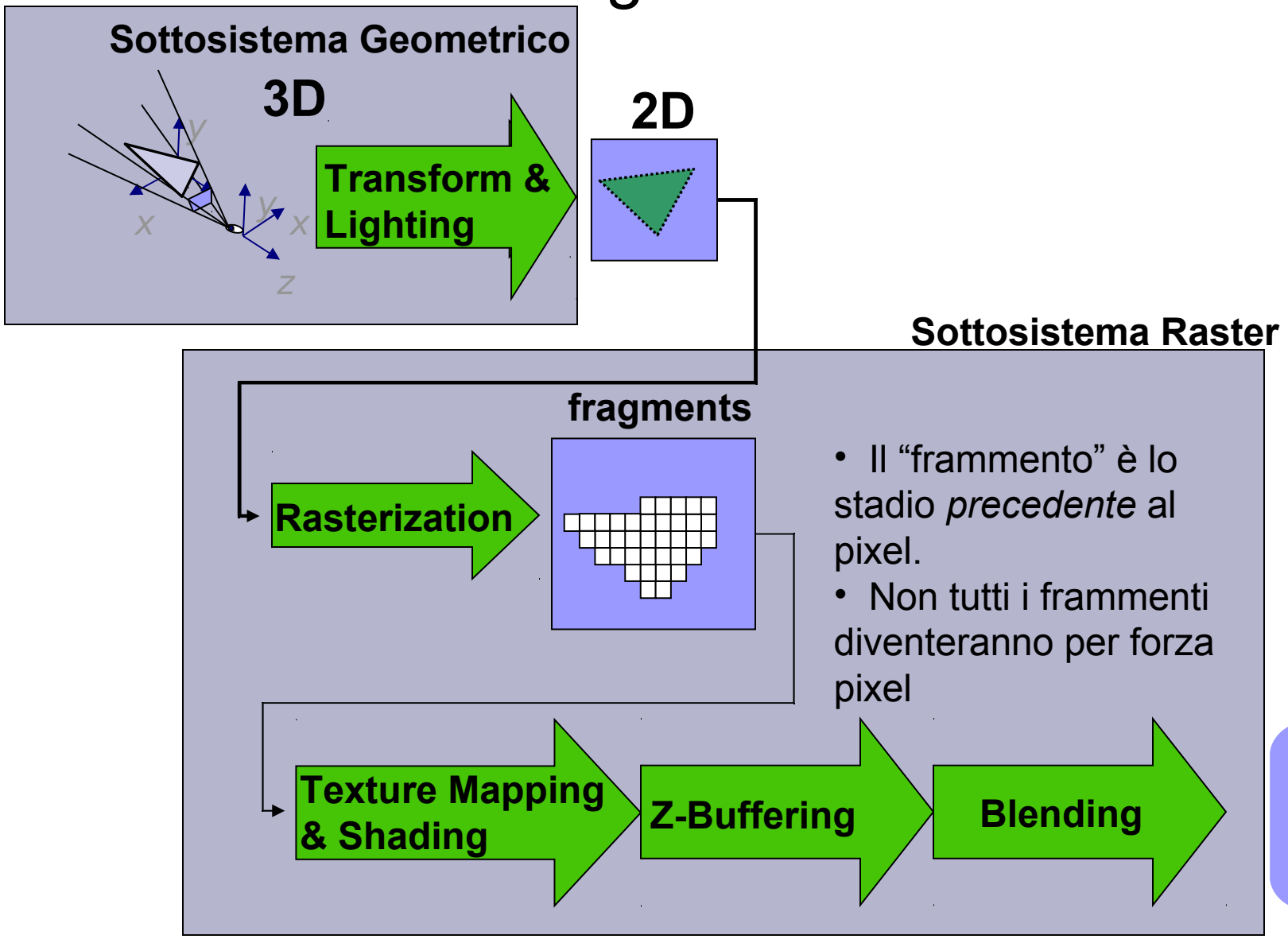


Lighting

- E la luce? Di che colore sono i pixel?
- La scena comprende anche:
  - descrizione delle fonti di luce
  - descrizione delle proprietà ottiche dei materiali
- L'interazione luce-materia viene modellata qui (in questo esempio di pipeline):



Qualche altro dettaglio....



Texture mapping, Z-Buffering, Blending

- **Texture mapping**: un'immagine è associata ad un poligono. Il colore dei frammenti generati dalla rasterizzazione del poligono dipendono dall'immagine
- **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

....vedremo tutto in dettaglio..

Caratteristiche

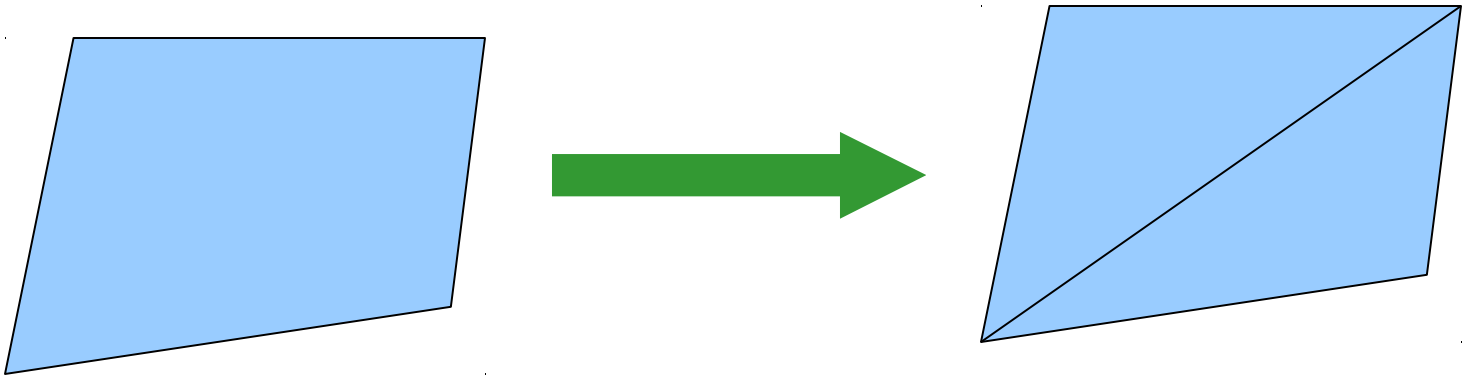
- La pipeline Rasterization-Based si basa su un'assunzione:

  "**Il mondo è fatto di punti, linee e poligoni**"

- L'implementazione hardware della pipeline (la scheda grafica) si basa su un'assunzione più stretta:

"**Il mondo è fatto di triangoli, segmenti e punti**"

Tutto sia composto da triangoli (3D)



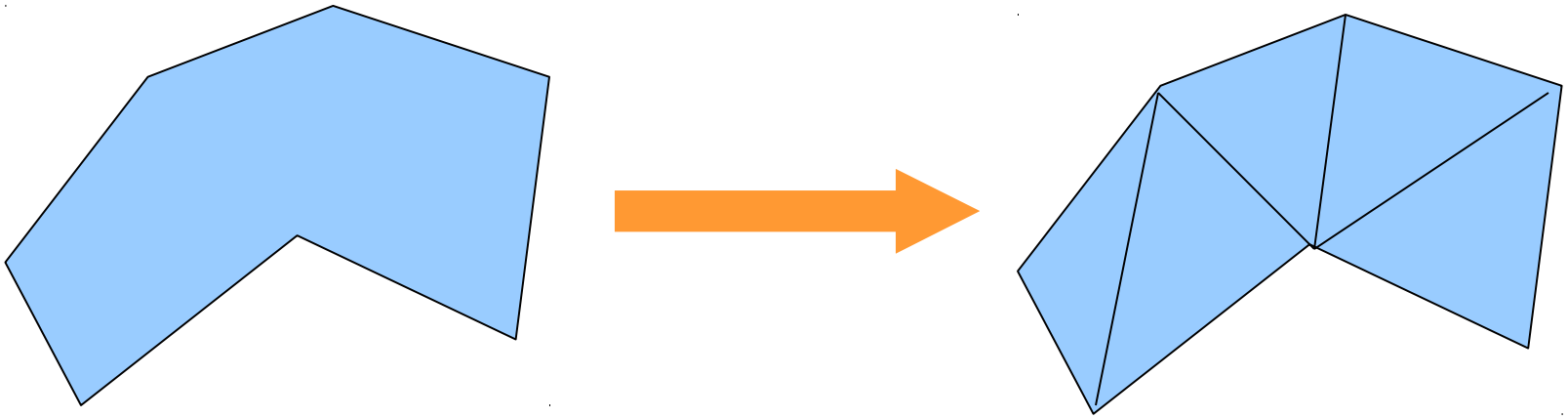
un quadrilatero?

due triangoli!

"quad"

"diagonal split"

Tutto sia composto da triangoli (3D)

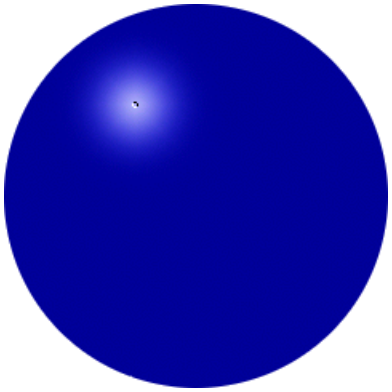


un poligono a *n* lati?

(n-2) triangoli!

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

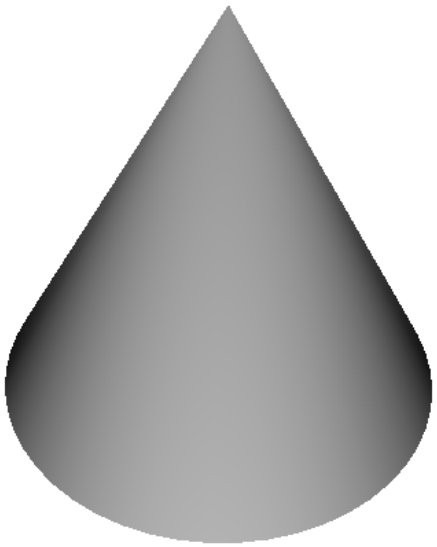
Tutto sia composto da triangoli (3D)

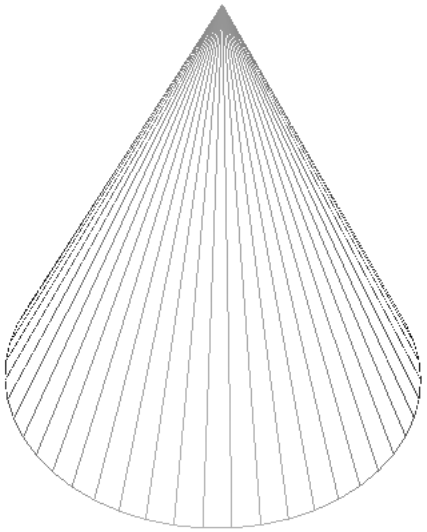


la superficie di
un solido geometrico,
per es. una sfera?

triangoli!

Tutto sia composto da triangoli (3D)

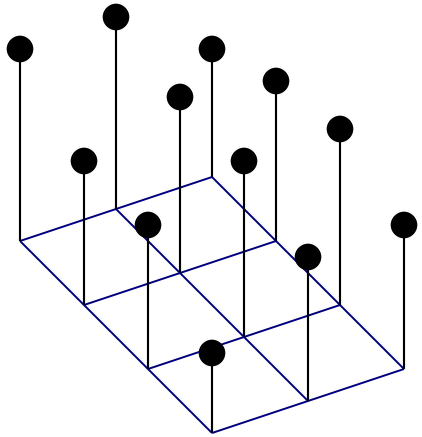


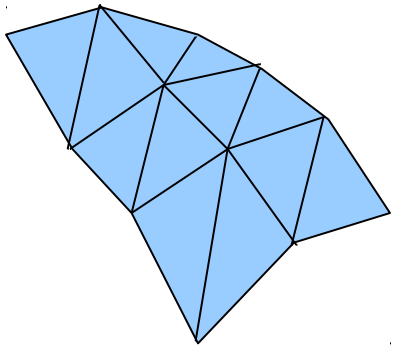


la superficie di
un solido geometrico,
per es. una cono?

triangoli!

Tutto sia composto da triangoli (3D)



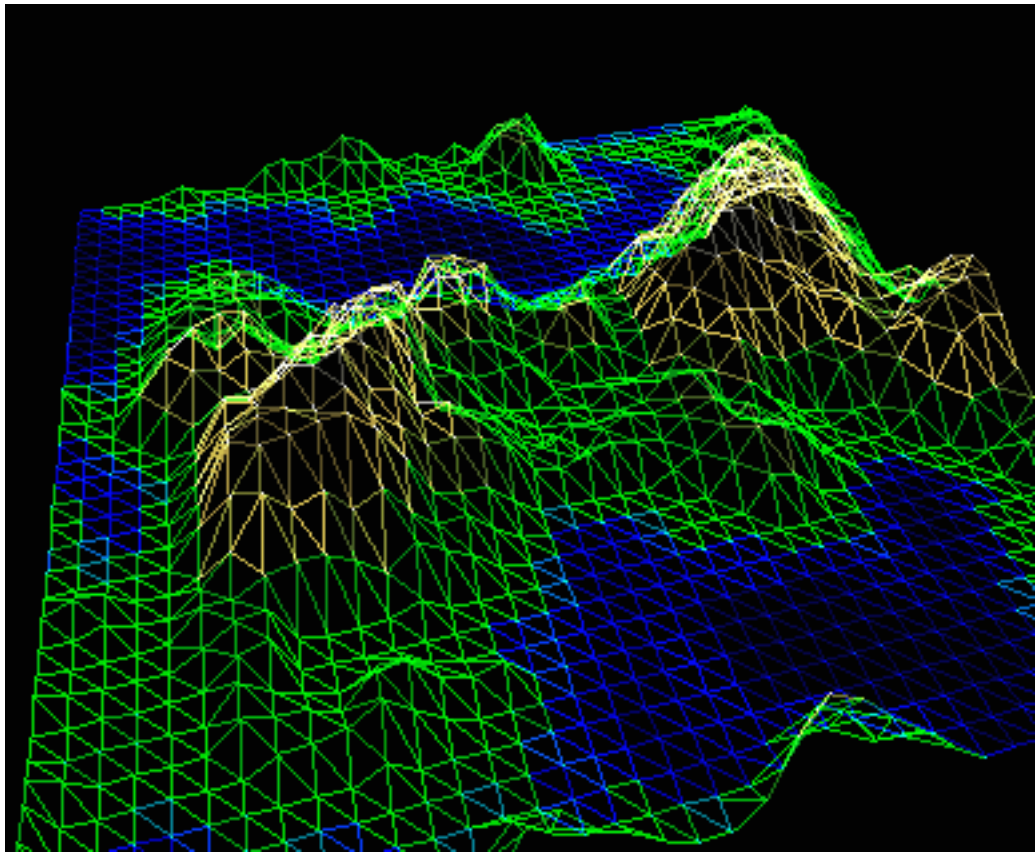


un campo d'altezza?
(array 2D di altezze,
e.g. per modellare un terreno?)

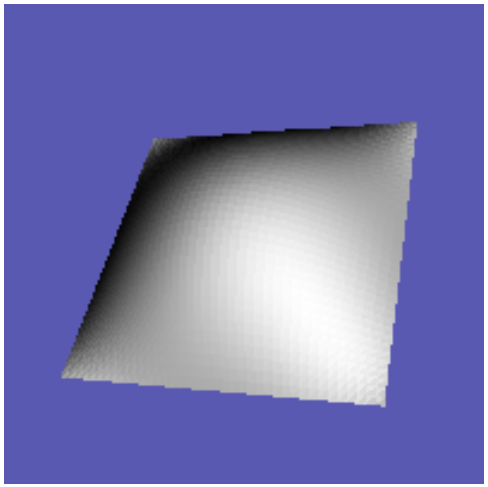
triangoli!

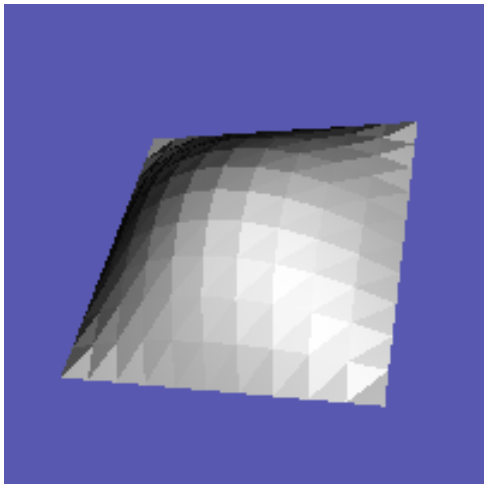
Esempio di campo di altezza triangolato

- "height field"
- un esempio tipico:
  - campo d'altezza per modellare un terreno...



Tutto sia composto da triangoli (3D)



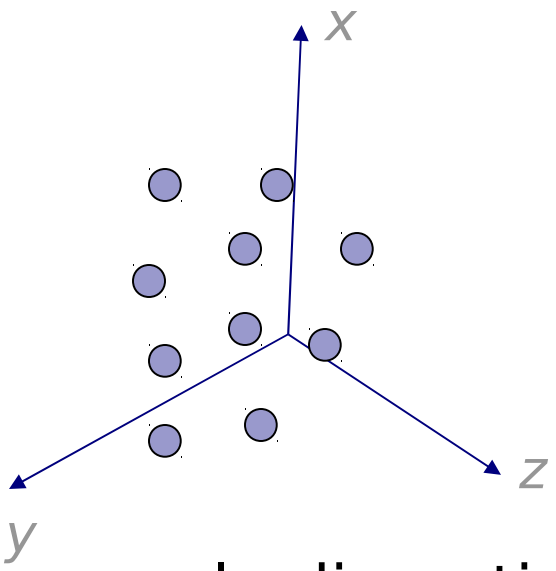


una superficie curva parametrica?

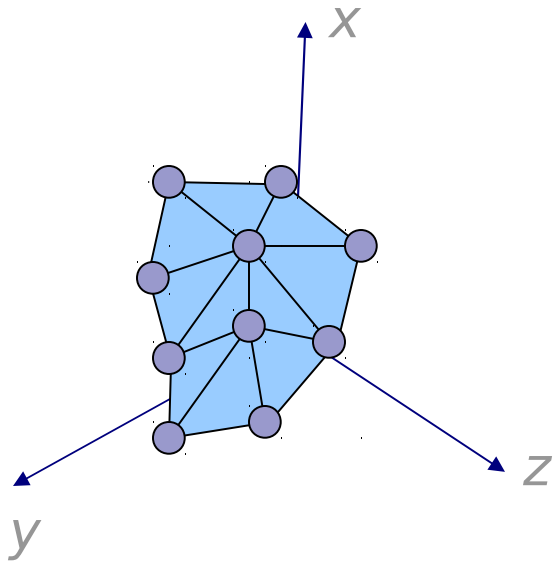
triangoli!

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

Tutto sia composto da triangoli (3D)



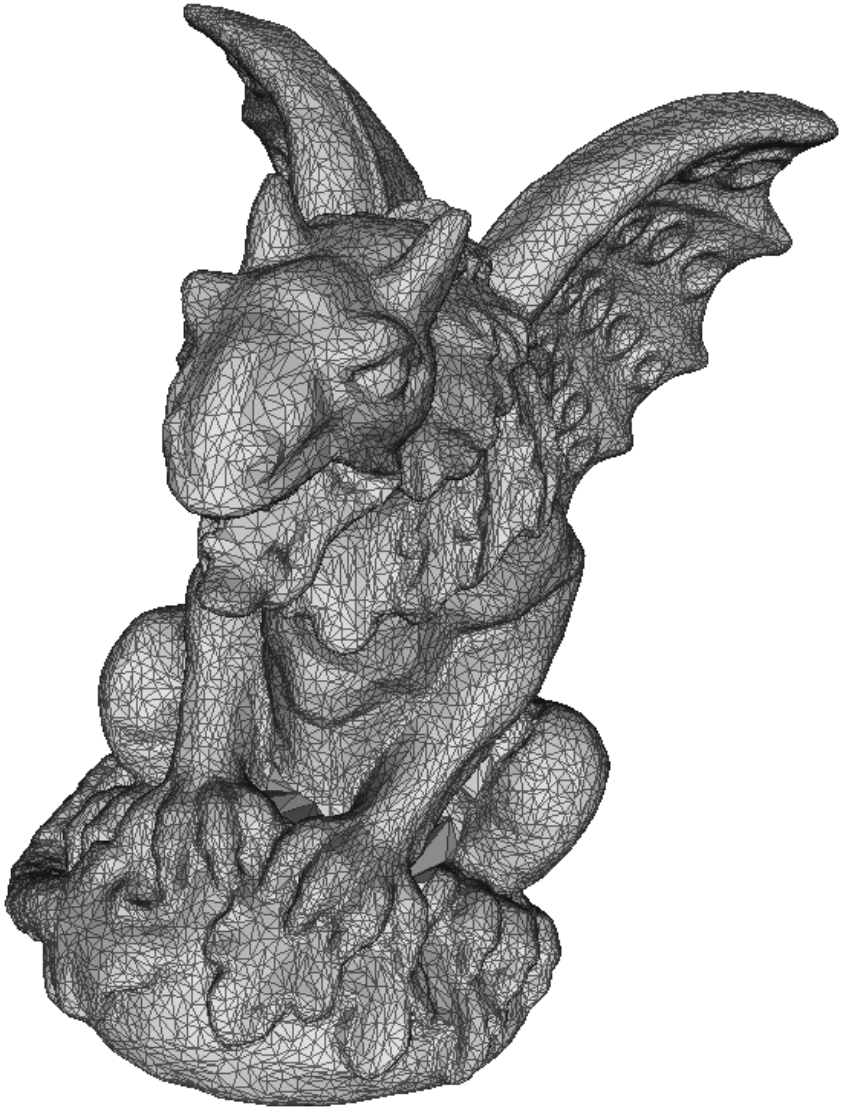
nuvola di punti ?
(*point clouds*)



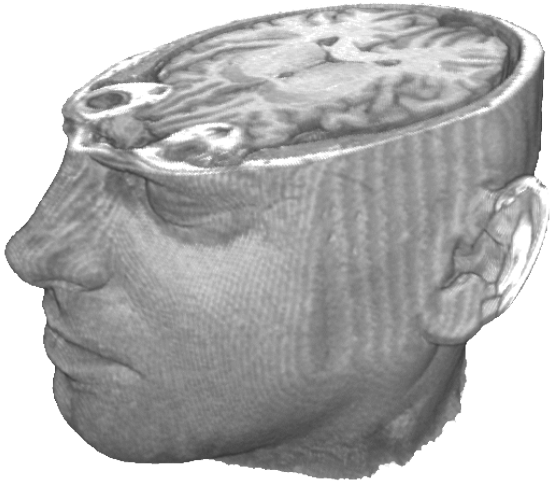
triangoli!

problema molto studiato,
e (nel caso generale) difficile

da nuvola di punti a triangoli: esempio



Tutto sia composto da triangoli (3D)

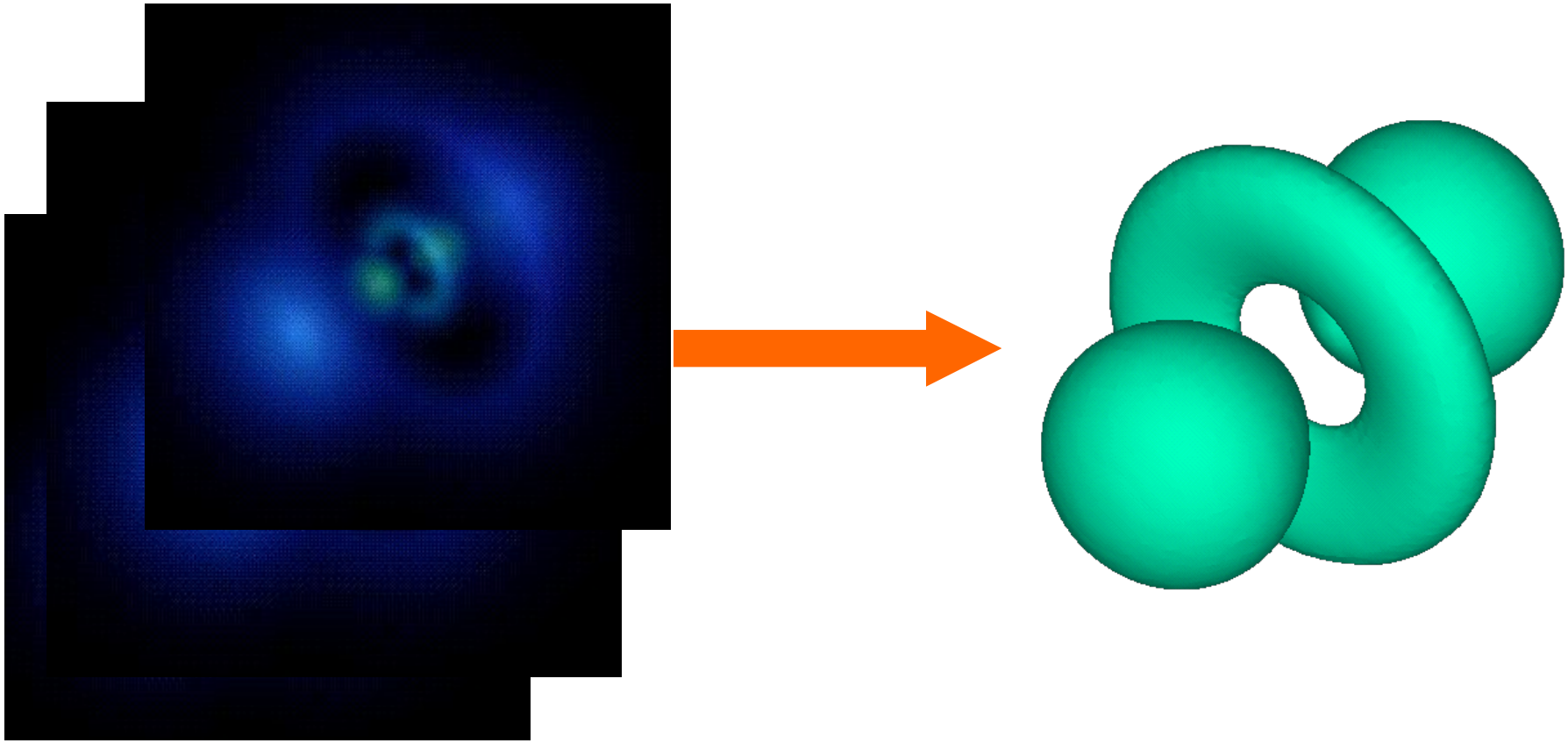


volume?

triangoli che definiscono una iso-superficie

triangoli!

Da dataset volumetrico a triangoli: esempio



Tutto sia composto da triangoli (3D)

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

triangoli che definiscono la superficie esplicitamente

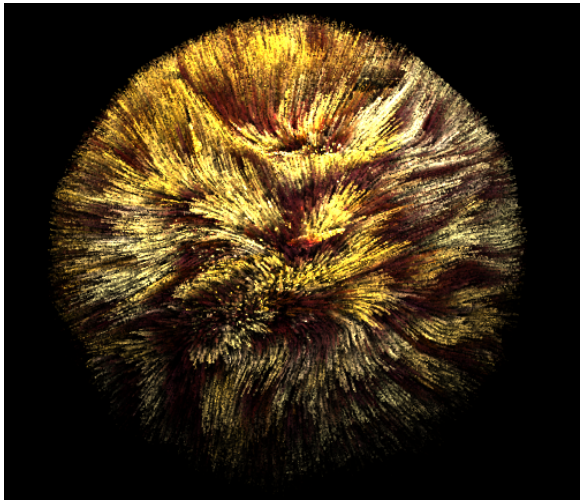
triangoli!

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

utili ad esempio
per particle systems

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)

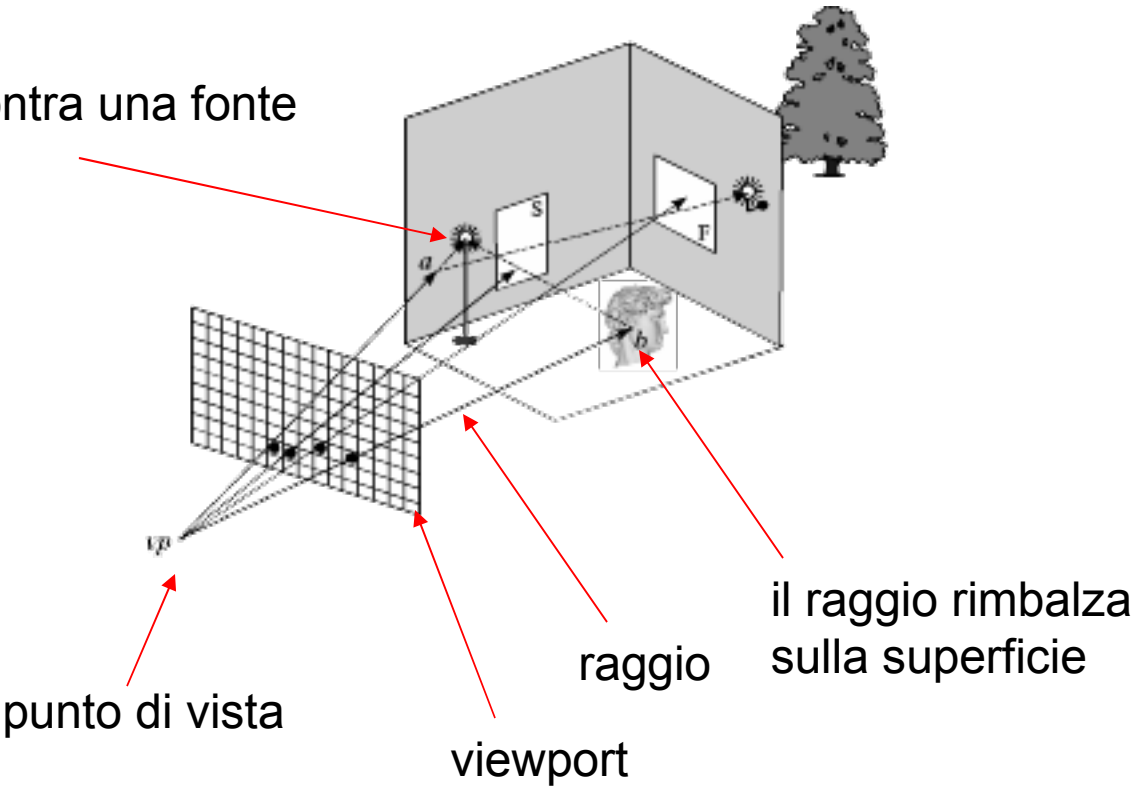
Caratterisiche

- 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

Altri paradigmi di rendering: Ray-Tracing

In un certo senso l'opposto del raster-based

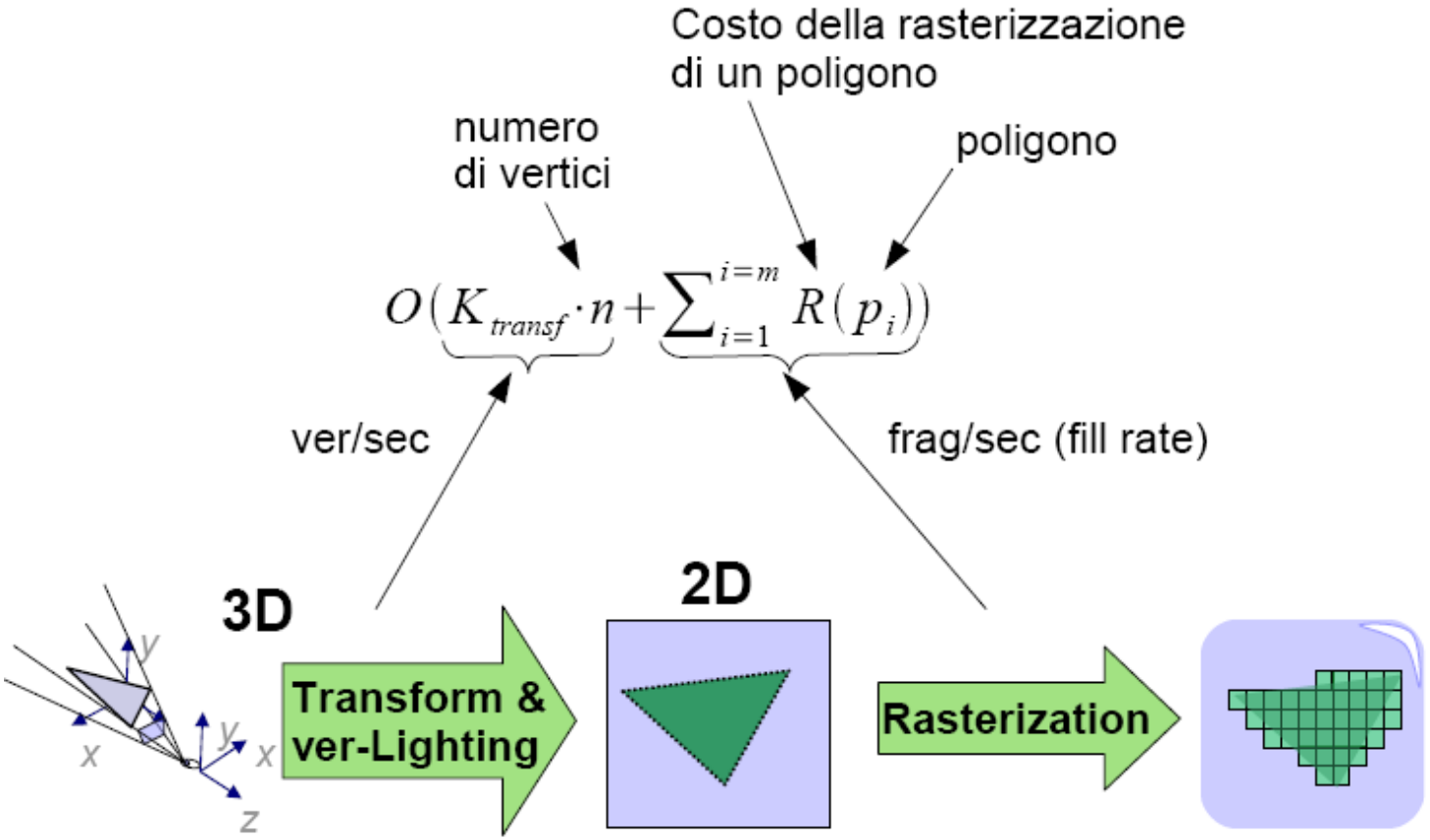
- Il principio: si seguono a ritroso i raggi di luce che giungono all'osservatore



Caratteristiche

- È un paradigma di rendering globale
- Modella facilmente le ombre portate
- Modella la inter-reflection “speculare” (luce su una superficie liscia)
- Anche la inter-reflection diffusa, ma meno bene (luce su una superficie non speculare: un muro)

Costi: rasterization based



Costi: Ray-Tracing

- *Int*(..) ha generalmente un costo sub-lineare
- Molte ottimizzazioni possibili

Altri paradigmi (Illuminazione globale)

- 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

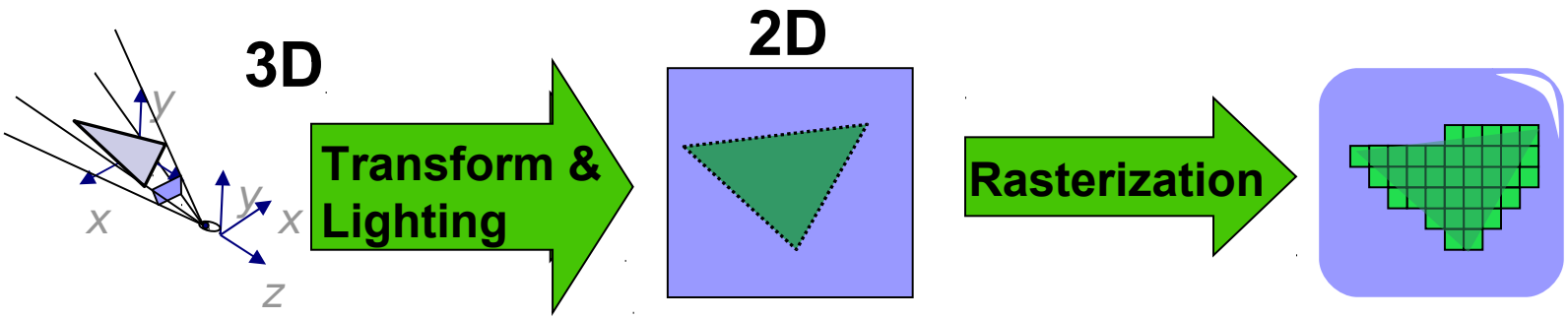
- Photon mapping
  - Si "simula" la luce con uno sciame di fotoni che arte dalle sorgenti e interagisce con le superfici
  - Computazionalmente proibitivo fino a qualche tempo fa

Paradigmi e hardware grafico

- Le schede grafiche sono pensate per il il paradigma Transform&Lighting
- Oggi la pipeline di rendering è interamente a carico della scheda grafica
- La CPU si limita a descrivere la scena

Cenni storici sull'hardware grafico

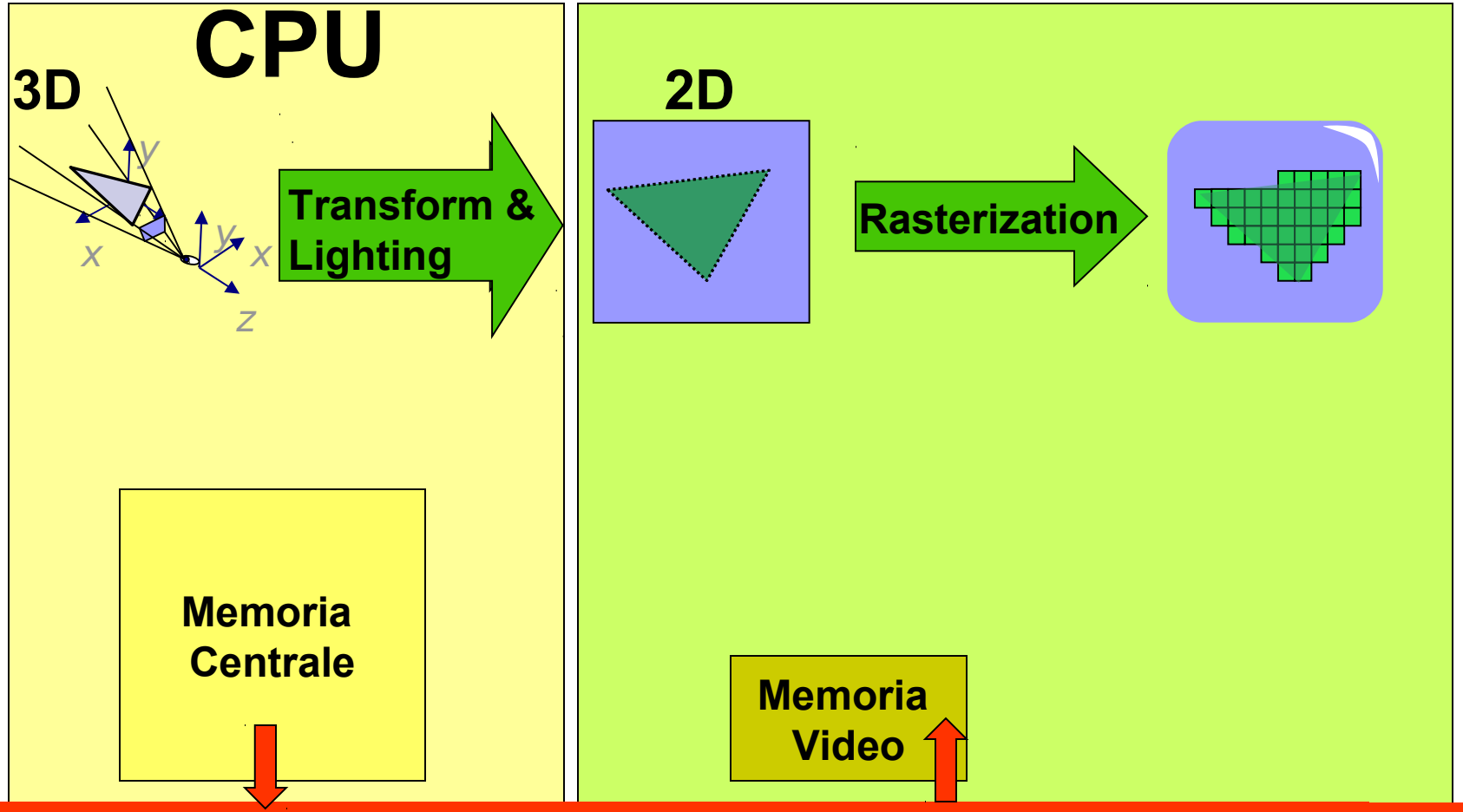
- Riprendiamo lo schema della pipeline di rendering



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

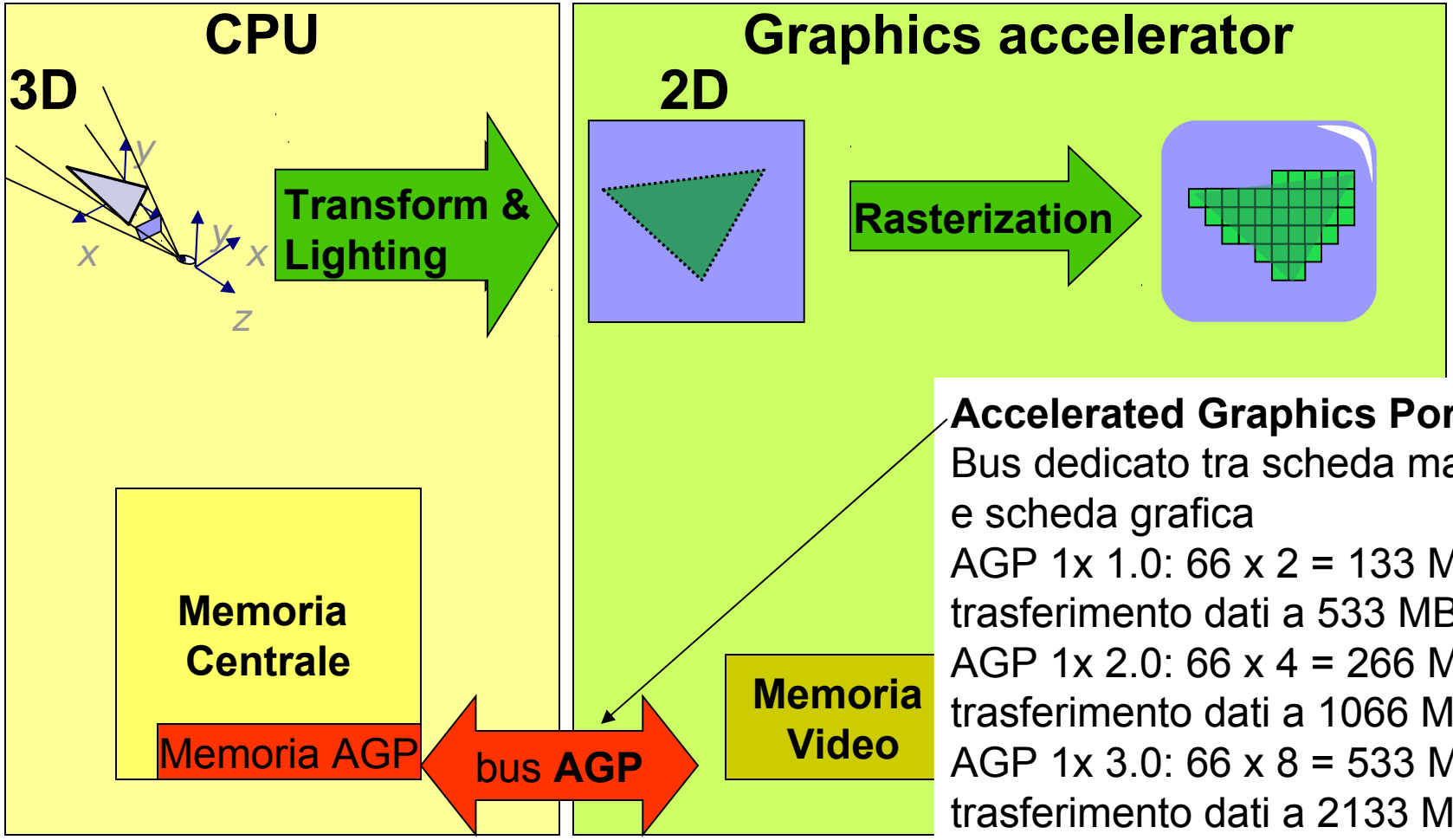
Cenni storici sull'hardware grafico

- 1995-1997: 3DFX Voodoo



Cenni storici sull'hardware grafico

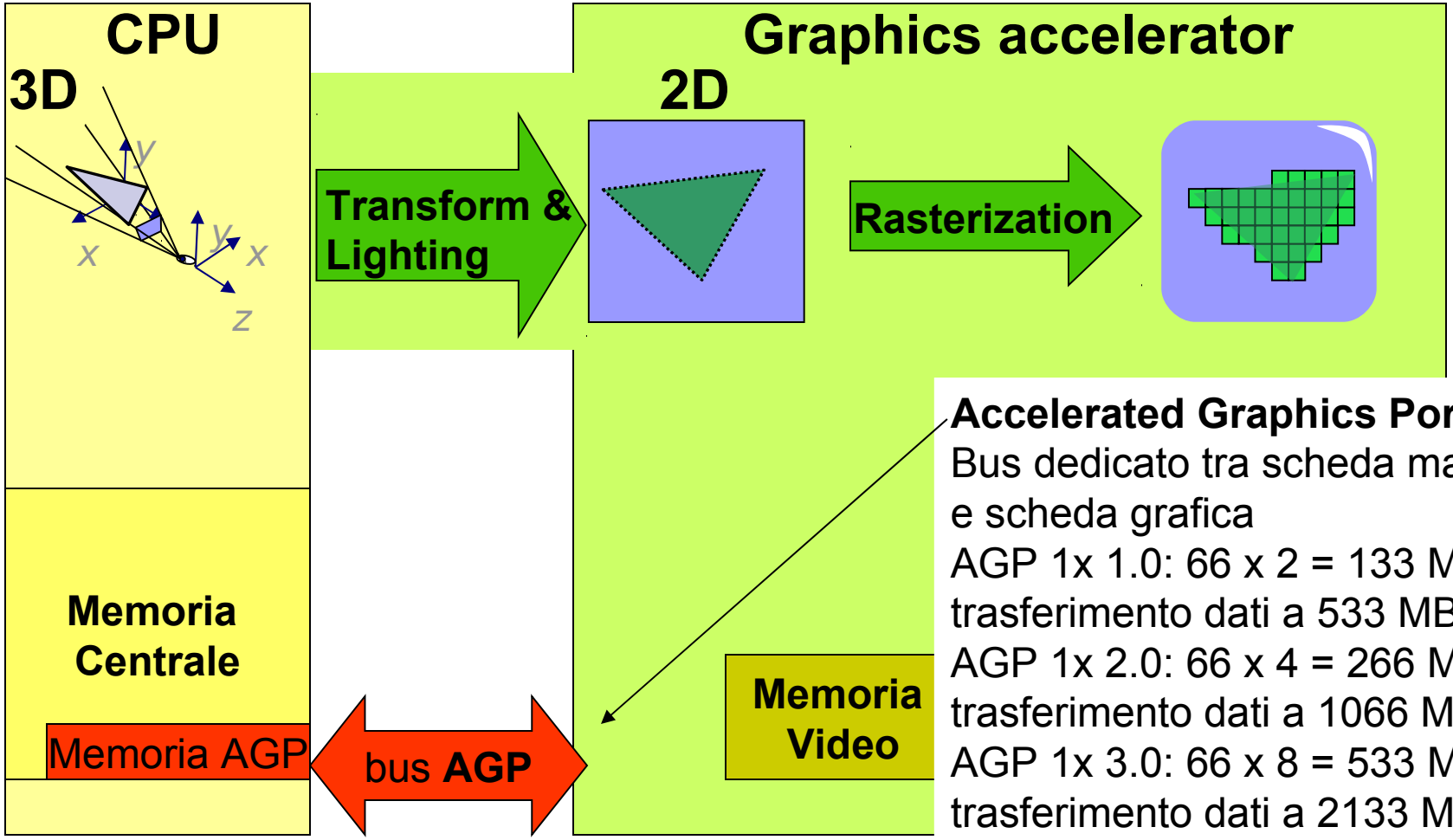
- 1998: NVidia TNT, ATI Rage



Accelerated Graphics Port
Bus dedicato tra scheda madre e scheda grafica
AGP 1x 1.0: 66 x 2 = 133 MHz, trasferimento dati a 533 MB/s
AGP 1x 2.0: 66 x 4 = 266 MHz, trasferimento dati a 1066 MB/s
AGP 1x 3.0: 66 x 8 = 533 MHz, trasferimento dati a 2133 MB/s

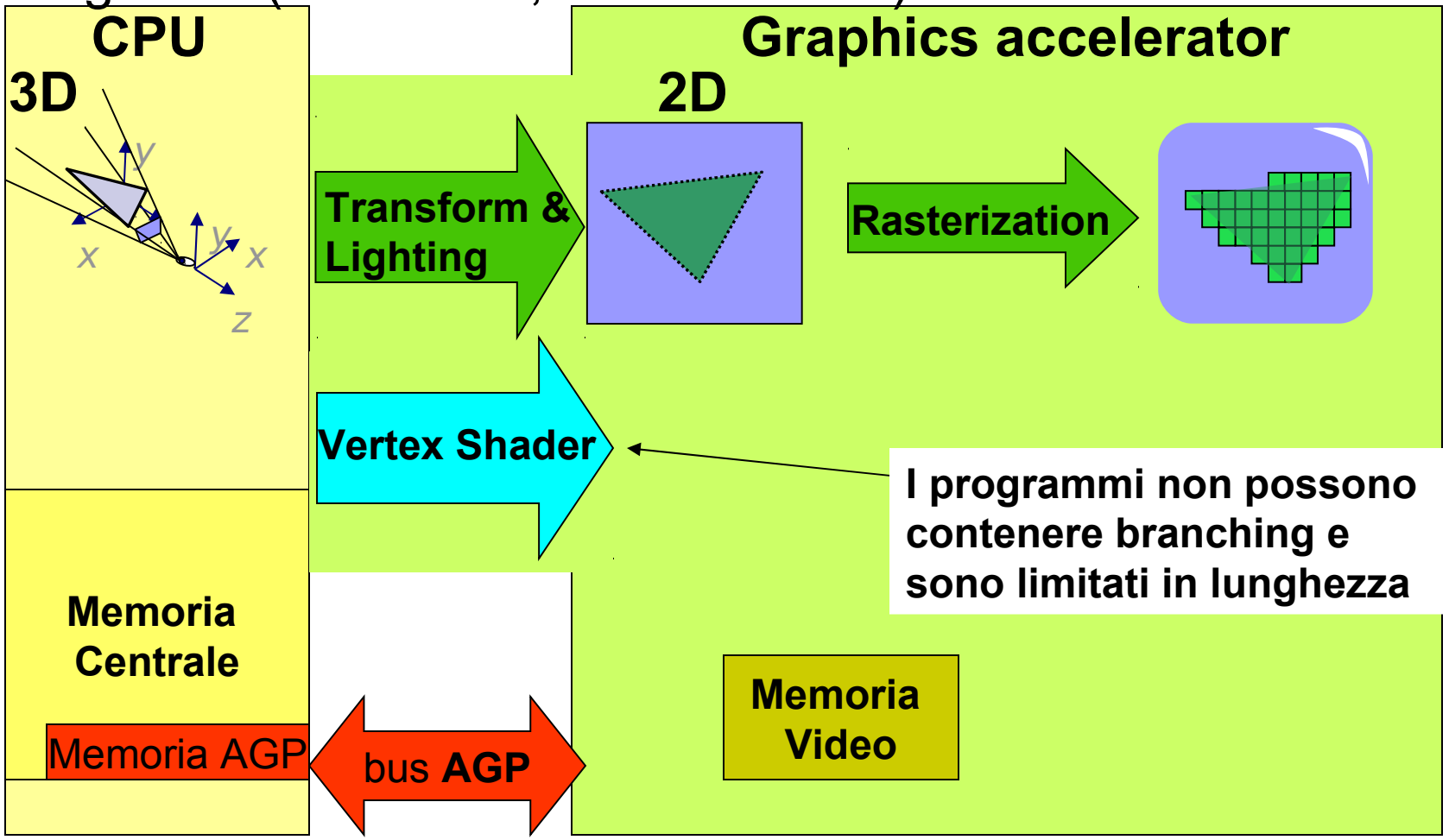
Cenni storici sull'hardware grafico

- 1998: NVidia TNT, ATI Rage



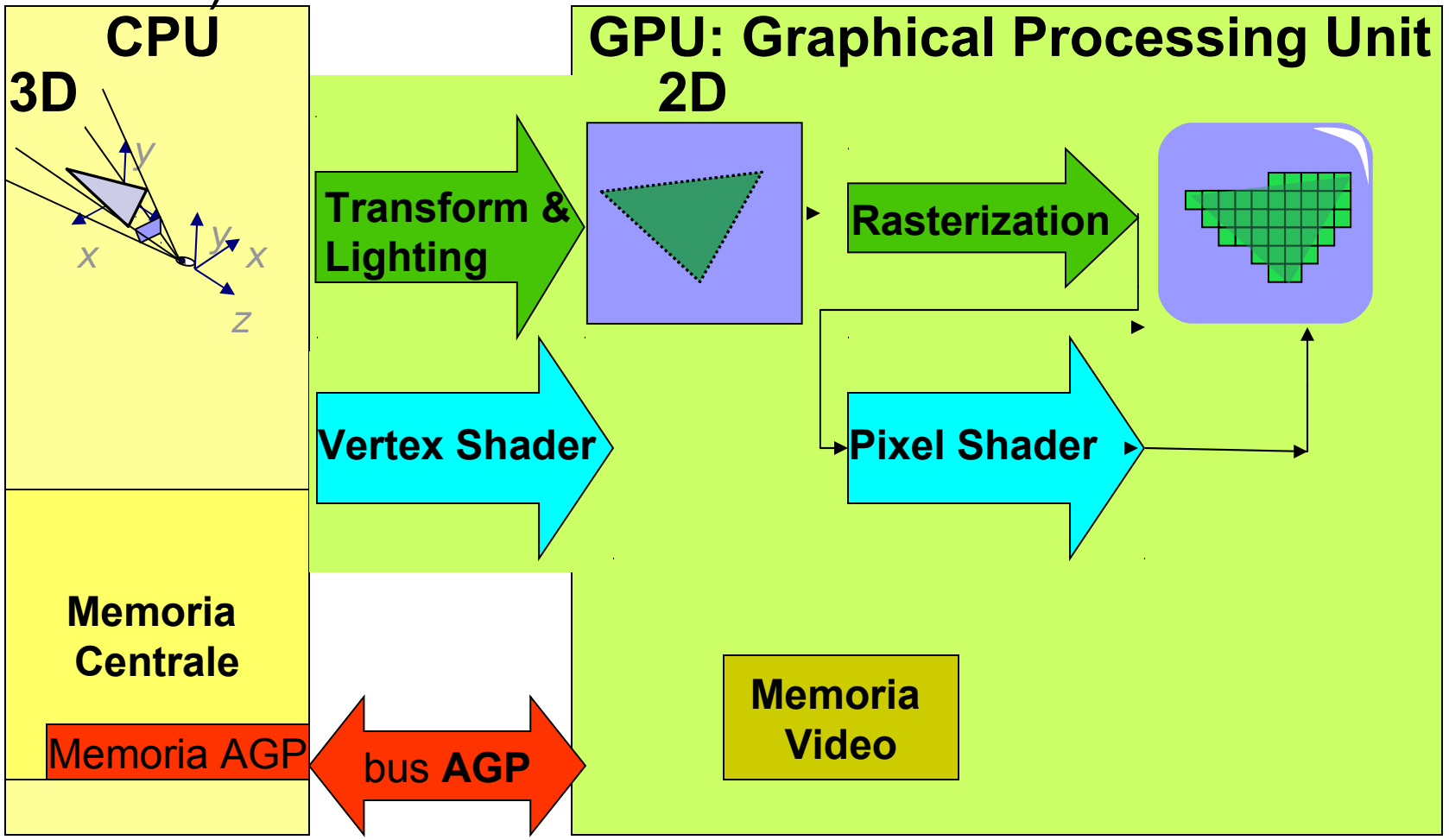
Cenni storici sull'hardware grafico

- 2001: Vertex Shader. Programmare il chip della scheda grafica (GeForce3, Radeon 8500)



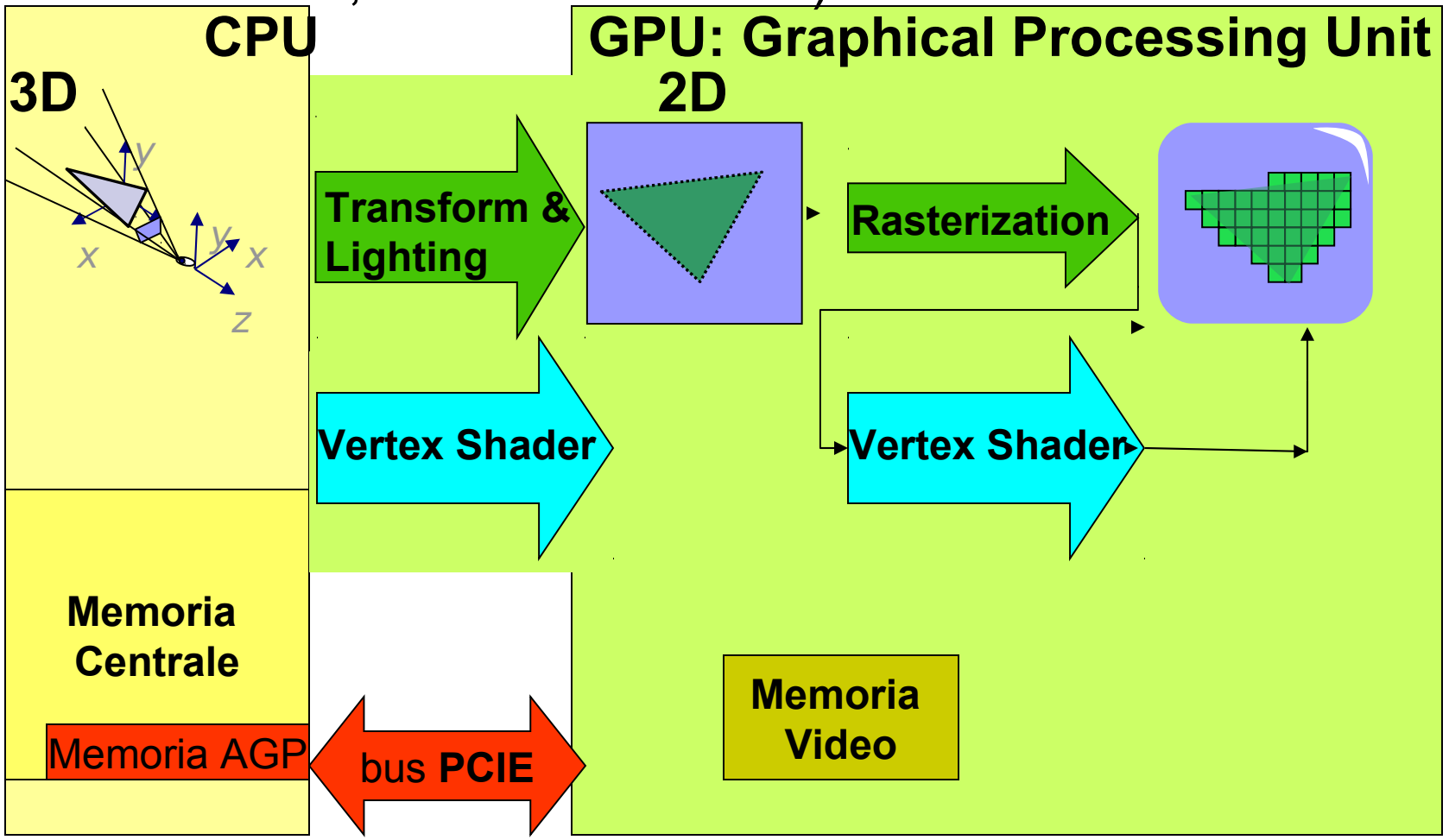
Cenni storici sull'hardware grafico

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



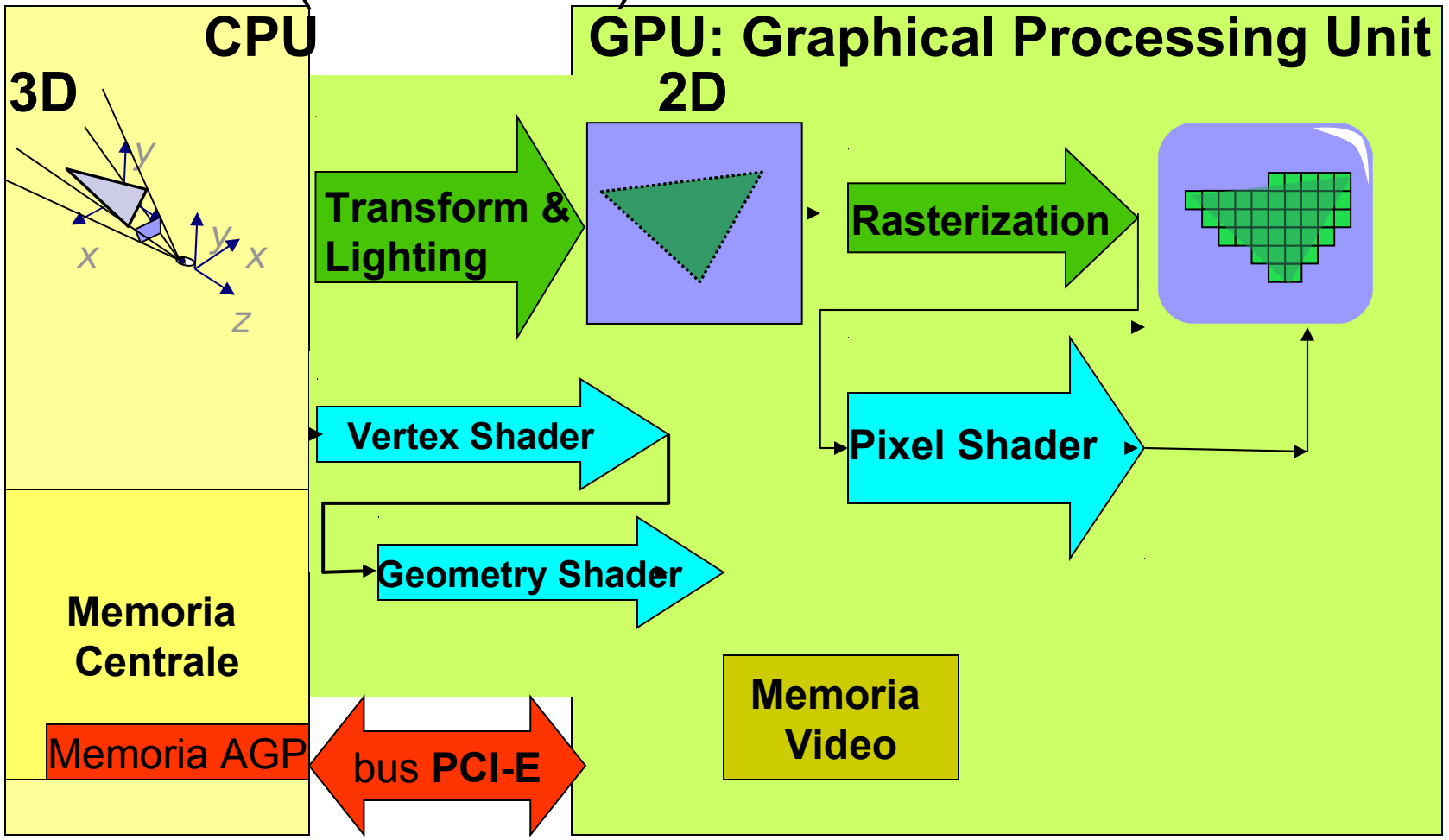
Cenni storici sull'hardware grafico

- 2004-5: PCI-Express, branching negli shader (GeForce 6800-7800, ATI Radeon 9200)



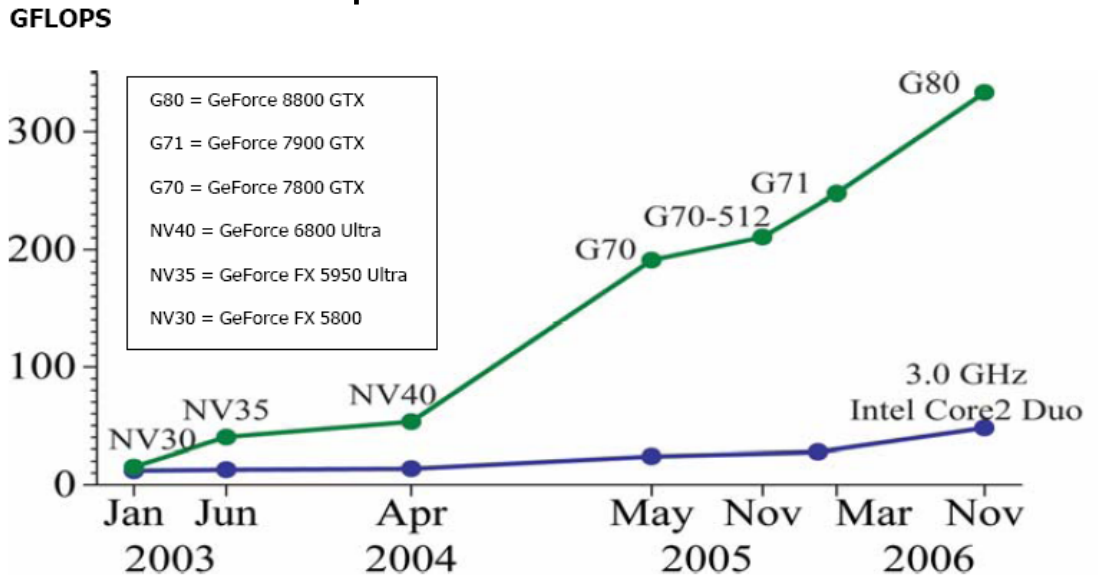
Cenni storici sull'hardware grafico

- 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*
- Potenza di calcolo: performance delle GPU NVidia



- Modello di memoria:
  - la memoria è progettata per contenere *textures*, NON per accesso random generico, per quanto sia possibile farlo. (*questo sta cambiando*)
- Ecco il perchè del General Purpose Computation on GPU.

Note sulla GPU odierne

- Stima delle prestazioni: I produttori scrivono tanti numeri, tra i quali:
  - Trasformazioni per secondo (sottosistema geometrico)
  - Fill-rate (frammenti al secondo)
  - Dimensioni - Memoria Video
- I primi due sono da prendersi con le molle
  - Sono prestazioni picco, nella pratica valgono solo su apposito esempio

Esempio esercizio da compitino

- Esercizio

  State effettuando il rendering di un modello composto da un milione di triangoli su un display con risoluzione 1280 x1024 pixel. Il rendering finale occupa l'80% del display. Se la scheda che state utilizzando ha una *performance* di 20 milioni di triangoli al secondo ed un fill-rate di 20 Mpixel al secondo, qual'è il numero massimo di fotogrammi al secondo che potete ottenere?

  Soluzione:

  - 1280x1024x0.8 = 838.860,8 numero di pixels da accendere per un rendering
  - 20*2^20=20.571.920 numero di pixels che la scheda può rasterizzare al secondo

  - Il sottosistema raster può produrre:

    20.571.920 / 838.860,8 = 25 fotogrammi al secondo

  - Il sottositema geometrico può produrre

    20 * 10^6 / 10^6 = 20

  Quindi la risposta è 20