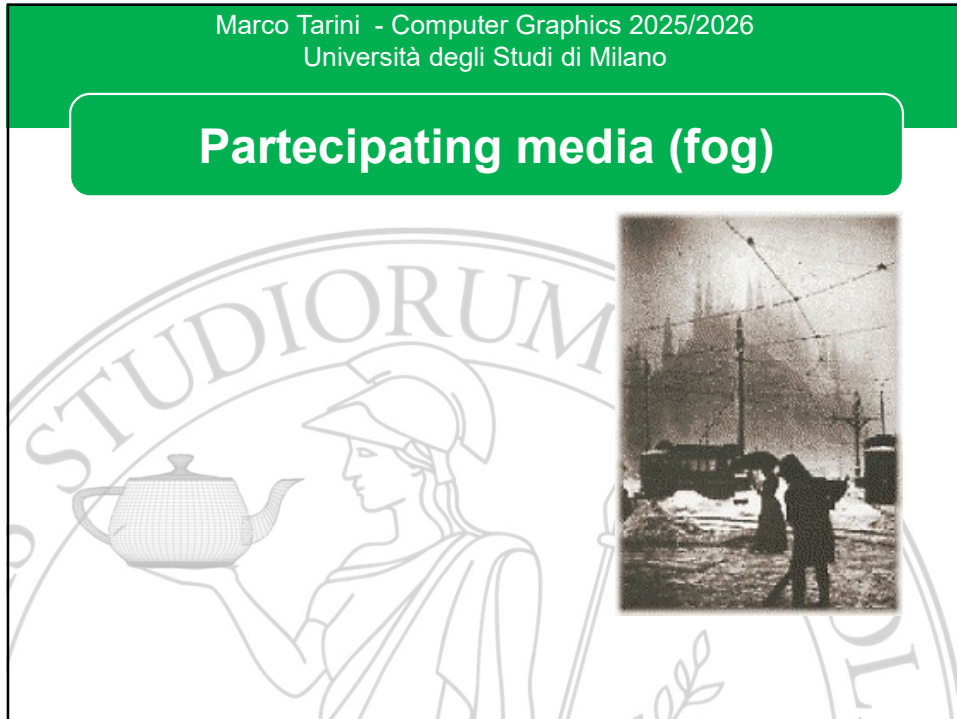


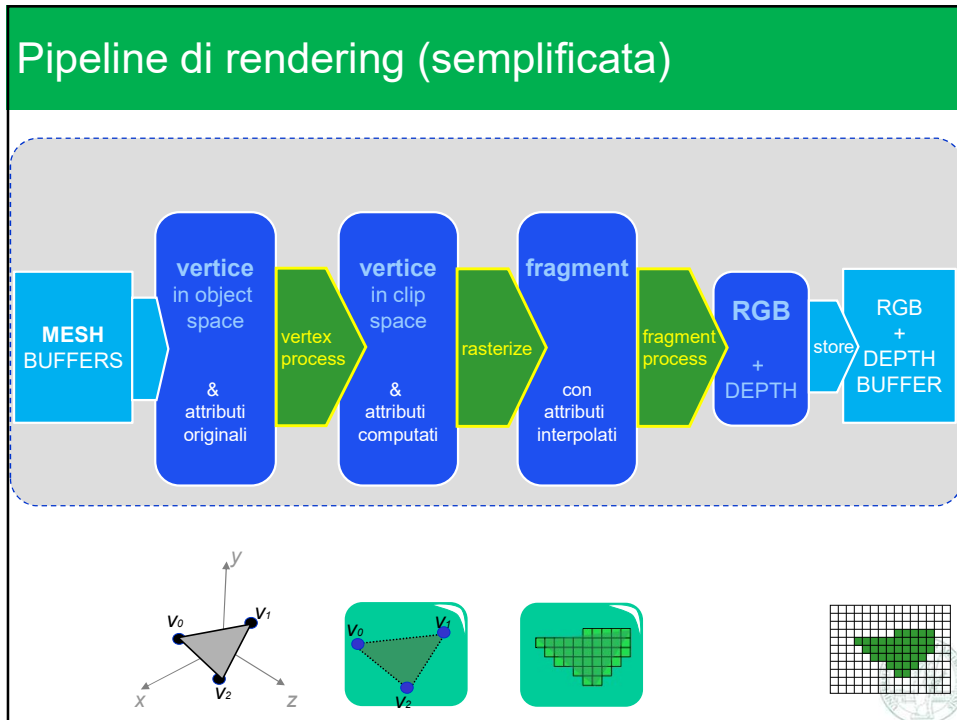
Marco Tarini - Computer Graphics 2025/2026  
Università degli Studi di Milano

## Participating media (fog)



The slide features a green header with the course name and university. Below it, a white rounded rectangle contains the title 'Participating media (fog)'. The background is a light gray watermark of the University of Milan logo, which includes a woman holding a teapot and the word 'STUDIORUM'. On the right side, there is a small, square, grainy image showing a person in a dark coat standing in a foggy, outdoor environment with some structures in the background.

1



2

## Pipeline di rendering: un sommario

- ✓ Le varie tecniche algoritmiche di rendering su GPU basati su rasterizzazione sono pensate per essere implementate sulla pipeline di rendering vista
  - ⇒ Determinando quale computazione effettuare in ogni fase
- ✓ Domande da chiedersi, di caso in caso:
  - ⇒ Quale attributo mi è necessario per vertice?
  - ⇒ Quale quantità è calcolata in quale fase? (per vertice, o per frammento)
  - ⇒ Le variabili in uscita dal fase per vertice (attributi) (in qualsiasi numero e di qualsiasi tipo siano) sono interpolate dal rasterizzatore e i valori risultanti sono forniti alla fase per frammento, per ogni frammento prodotto
- ✓ Vediamo un esempio: l'effetto nebbia (fog)



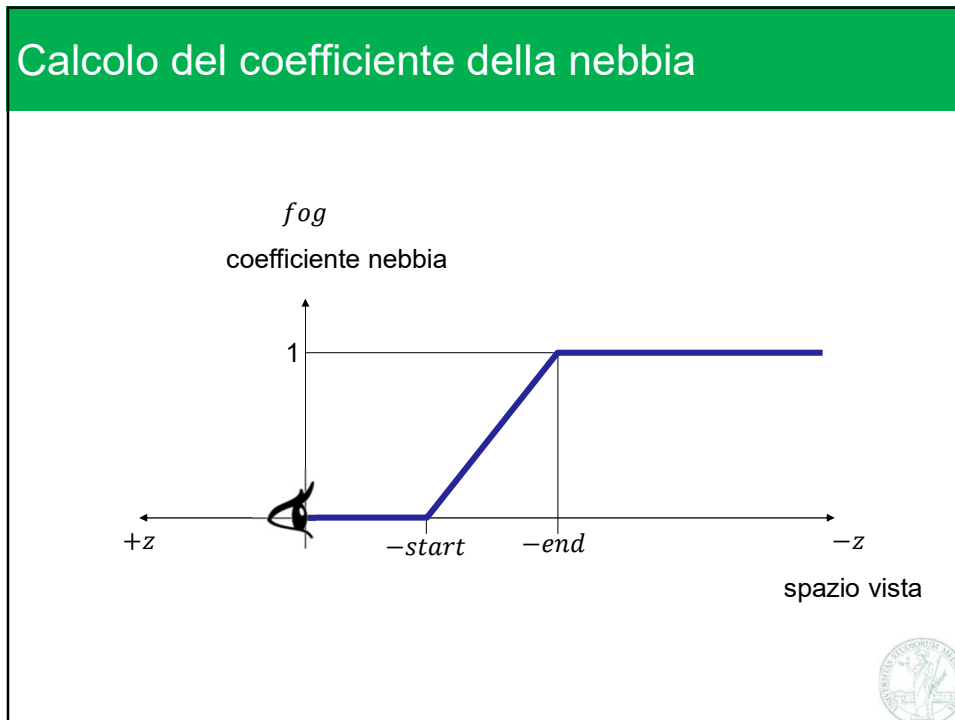
3

## Fog (cioè participating media, cioè depth cueing)

- ✓ L'effetto reale che vogliamo simulare: "participating medium":
  - ⇒ il mezzo (il «medium») attraverso il quale viaggia la luce influenza la luce stessa, assorbendone una parte (su alcune frequenze)
  - ⇒ Tanto maggiore è la lunghezza del percorso fino alla camera, tanto più accentuato è l'effetto
- ✓ E' facile simulare questo effetto nel rendering basato su rasterizzazione
- ✓ Idea Base:
  - ⇒ tanto più un frammento dista dall'osservatore (maggiore  $z$  in sp. vista) quanto più il suo colore è impattato da nebbia / foschia / etc
- ✓ Quindi: la  $Z$  del vertice in *spazio vista* determina il «coefficiente di nebbia»
  - ⇒ in alternativa, potremmo calcolare l'effetto anche in spazio clip oppure spazio schermo



4



5

### Calcolo del coefficiente della nebbia

- ✓ A parole: «il coefficiente nebbia è una funzione lineare «clampata» fra 0 e 1 della z in spazio vista»
- ✓ In formula, dato un punto di coordinata z in spazio vista (negativa):
$$fog = clamp\left(\frac{(-z) - start}{end - start}, 0, 1\right)$$
  - coefficiente nebbia  
0 = niente nebbia  
1 = si vede solo nebbia
  - funzione che porta a 0 se <0 e a 1 se >1
  - la formula lineare che restituisce...  
0 quando z = -start  
1 quando z = -end  
(verifica!)
- ✓ start e end sono parametri dell'effetto:
  - ⇒ start: la distanza (positiva) dall'osservatore prima della quale non si percepisce alcuna nebbia
  - ⇒ end: la distanza (positiva) dall'osservatore, dopo la quale si ha 100% nebbia (l'oggetto sparisce del tutto)
- ✓ Anche altre formule (più realistiche di questa) sono possibili

6

## Usare il fattore nebbia per definire il colore del pixel

✓ Dopo aver calcolato il fattore nebbia, lo applico al colore finale del pixel associato al frammenti  
⇒ ricordando che colore = tripletta di valori r,g,b

il colore che avremmo normalmente assegnato a questo pixel

il colore della nebbia (una costante, un parametro)


$$\text{colore}_{\text{finale}} = \text{mix}(\text{colore}_{\text{original}}, \text{colore}_{\text{nebbia}}, \text{fog})$$

cioè

$$\text{colore}_{\text{finale}} = (1 - \text{fog}) \cdot \text{colore}_{\text{originale}_{\text{frammento}}} + \text{fog} \cdot \text{colore}_{\text{nebbia}}$$

detto a parole: «il colore finale è un'interpolazione lineare fra i colori nebbia e originale»  
(l'ennesimo l'uso di interpolazione lineare che incontriamo, questa volta sui colori)

Osserva che questa formula restituisce...  $\text{colore}_{\text{nebbia}}$  quando  $\text{fog} = 1$   
 $\text{colore}_{\text{originale}}$  quando  $\text{fog} = 0$



7

## Possibili usi dell'effetto fog 1/2

1. Simulare (in modo approssimato) il participating media  
⇒ Nebbia, foschia, acqua sporca (rendering subacquei) etc;  
⇒ Se color fog (e sfondo) = nero: effetto oscurità (cruda approssimazione)
2. Mascherare gli artefatti di *popping* dovuto al *clipping* contro il *far-clipping-plane*  
⇒ Senza fog: gli oggetti spariscono o riappaiono di colpo quando attraversano il *far-clipping plane*  
⇒ Con fog: gli oggetti subiscono un *fade-out*, cioè si dissolvono progressivamente nella nebbia, con continuità, diventando color-sfondo, man mano che si avvicinano al *far clipping plane*.  
⇒ Per ottenere questo effetto dobbiamo scegliere:  
(a) il colore della nebbia identico al colore sfondo  
(b) la distanza *far* coincidente con la distanza *z\_far* usata nella matrice di proiezione  
⇒ Usato nei videogames (che spesso hanno bisogno di *far clipping plane* più vicini di quanto si desidera, per motivi di efficienza)



8

## Possibili usi dell'effetto fog 2/2

### 3. Sugerire visivamente la profondità degli oggetti:

- ⇒ Nel contesto della visualizzazione scientifica (e altri)
- ⇒ Il fog consente di leggere la profondità relativa degli oggetti nei rendering 3D, e produrre immagini intuitive e facili da interpretare
- ⇒ Particolarmente utile in assenza di altri indizi, come la prospettiva (cioè quando è usata proiezione ortografica)
- ⇒ Particolarmente utile in presenza di forme 3D complesse e non intuitive
- ⇒ Il fog usato per questo scopo è detto «**depth cueing**» («dar spunti di profondità»)



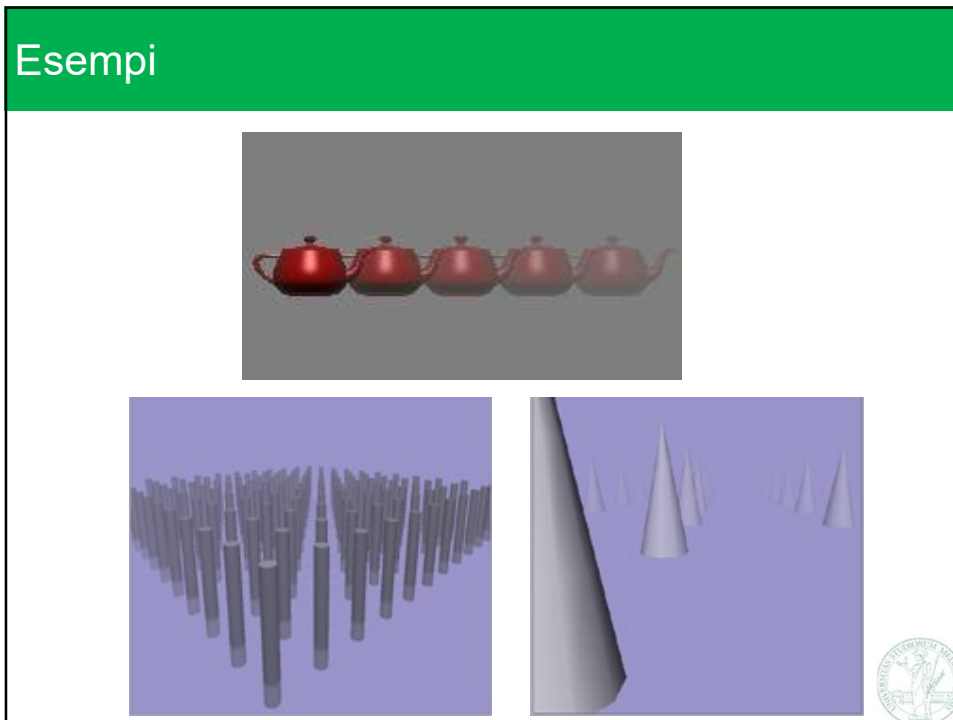
9

## Esempi

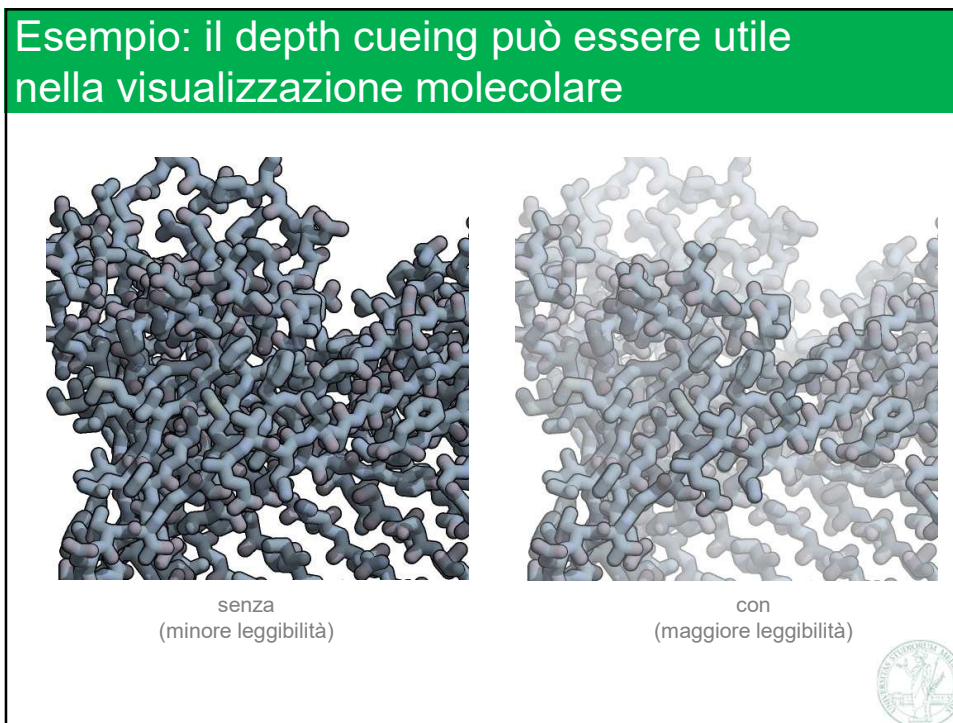
spesso usato nei videogames per mascherare il **popping** dovuto al **clipping** contro il **far-clipping-plane**



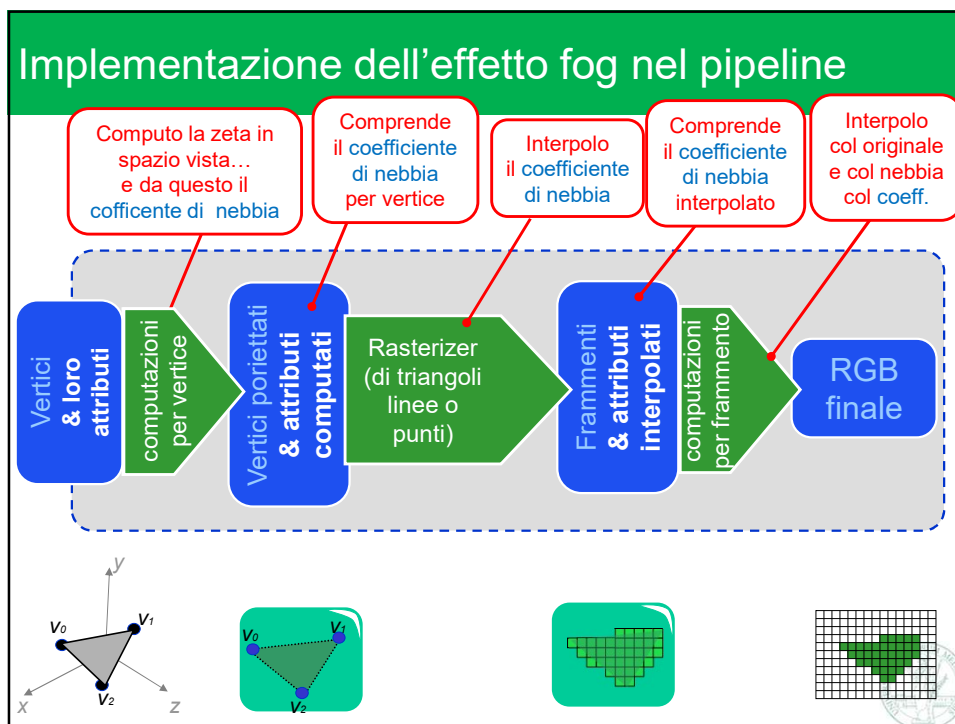
10



11



12



13

### Implementare l'effetto fog su API a basso livello

- ✓ Il computo del fog è implementato nelle computazioni per vertice e per frammento
  - ⇒ Nessun particolare supporto HW è necessario
  - ⇒ (a differenza di meccanismi come depth-test o back-face culling che sono implementati in HW, automaticamente)
- ✓ A basso livello (in API come OpenGL o WebGL):
  - ⇒ l'effetto fog è implementato programmandolo nel vertex shader (il programma utente eseguito su ogni vertice) e nel fragment shader (il programma utente eseguito su ogni frammento)
- ✓ Nelle librerie ad alto livello (come three.js o Unity)
  - ⇒ l'effetto fog è già implementato dalla libreria, e deve solo essere abilitato
- ✓ (le stesse considerazioni varranno anche per il lighting)

14

## Implementare l'effetto fog su librerie ad alto livello (vedi `cgLab04.html`)

- ✓ Le librerie ad alto livello, come `three.js`, implementano per noi l'effetto fog.
- ✓ Dobbiamo solo settarne i parametri, e abilitarlo

```
var colorNebbia = 0x00FFFF; // qui: un azzurro cielo
var near = 0.5; // espresso in spazio vista
var far = 4.0; // espresso in spazio vista
var nebbia = new THREE.Fog(colorNebbia, near, far);

scena = new THREE.Scene();
// ...
scena.fog = nebbia;
```

- ⇒ Quali parametri usiamo? Una scelta naturale è:  
usare `colorNebbia` anche come colore di sfondo (il clear color),  
e `far` anche come distanza del far clipping plane  
(che è uno dei parametri intrinseci della macchina fotografica)

