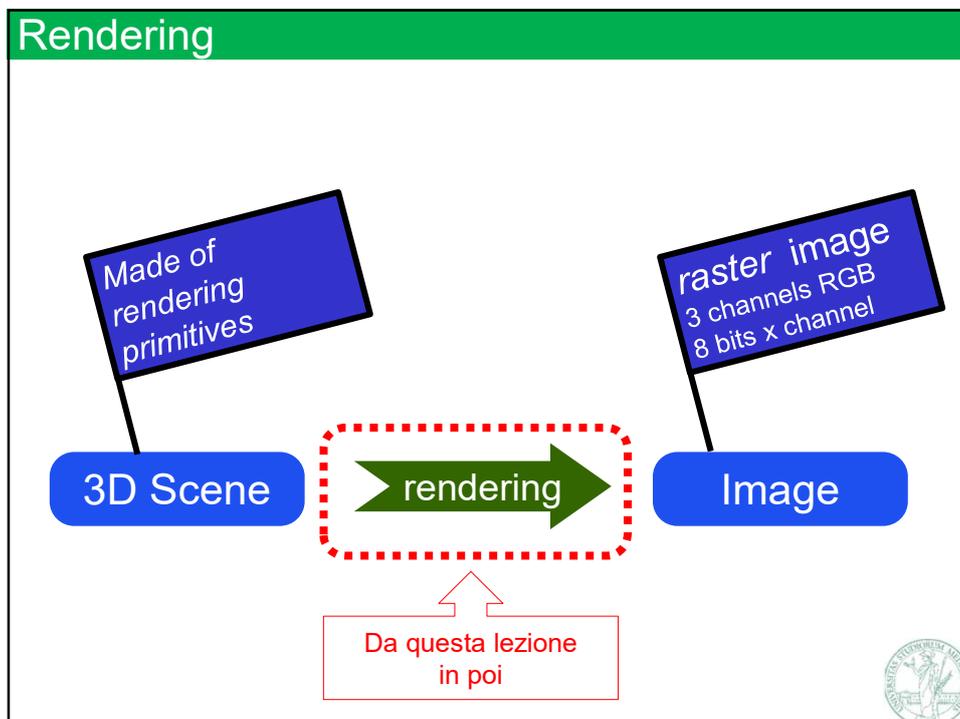


Marco Tarini - Computer Graphics 2020/2021
Università degli Studi di Milano

Rendering: ray-tracing

The slide features a green header with the author's name and university. Below it, a white rounded rectangle contains the title 'Rendering: ray-tracing'. The background shows a faint wireframe teapot and a classical figure. An inset shows a cartoon artist painting a teapot on an easel. At the bottom right, there is a logo with green virus-like shapes and the text 'TELEDIDATTICA!'.

1



2

Rendering

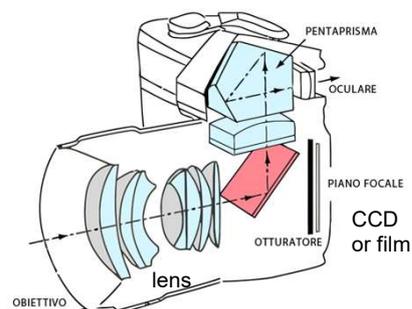
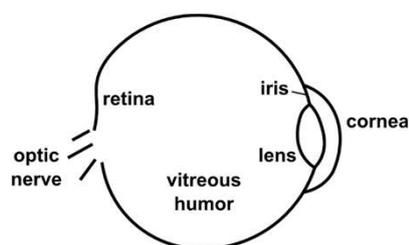
- ✓ Molti algoritmi di rendering prendono diretta ispirazione dal modo in cui un dispositivo reale di sintesi di immagini produce un'immagine
 - ⇒ Dispositivi come: macchina fotografica, una telecamera, un occhio umano
- ✓ A questo fine, possiamo rimuovere tutti gli elementi non necessari di tali dispositivi, per ridurli ad un modello più semplice possibile
 - ⇒ Questo modello è detto Pin-hole camera.



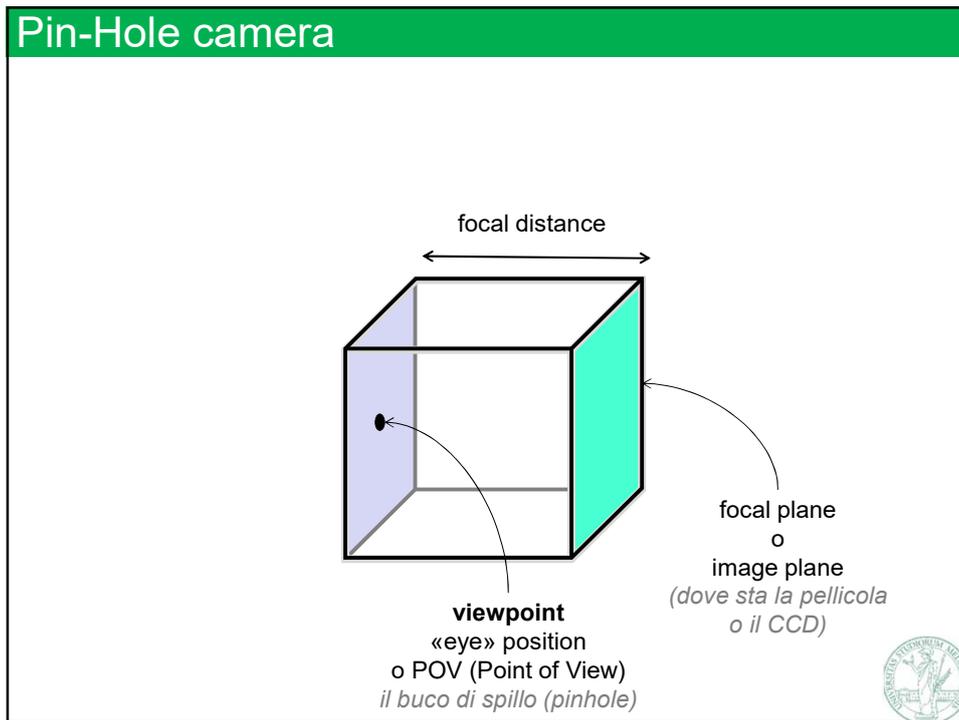
4

Modelling the picture making apparatus

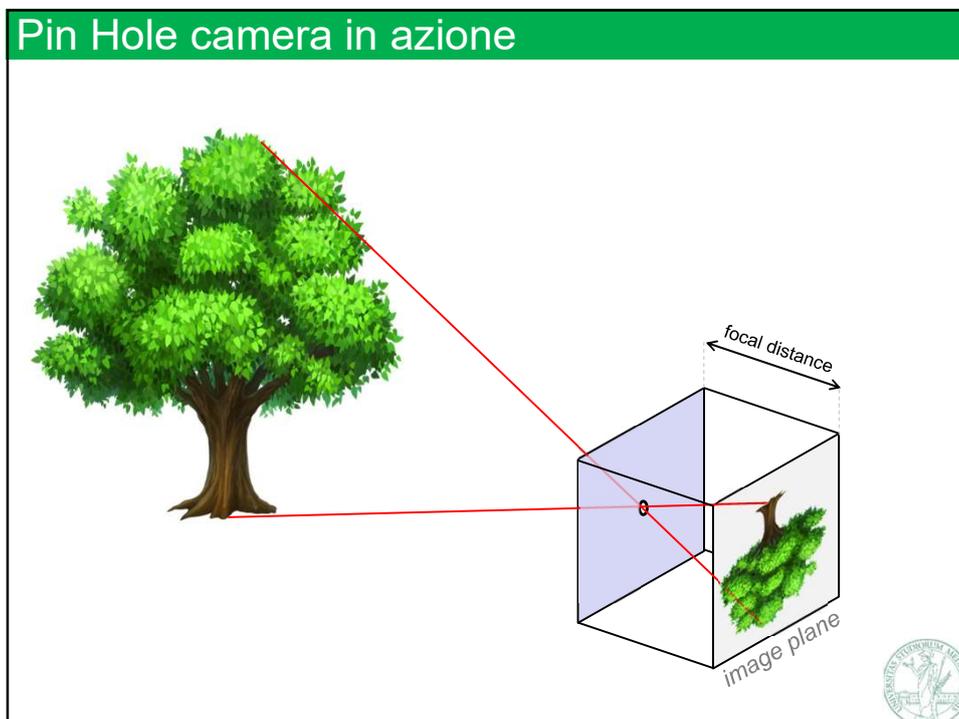
- ✓ Human Visual System
- ✓ Camera



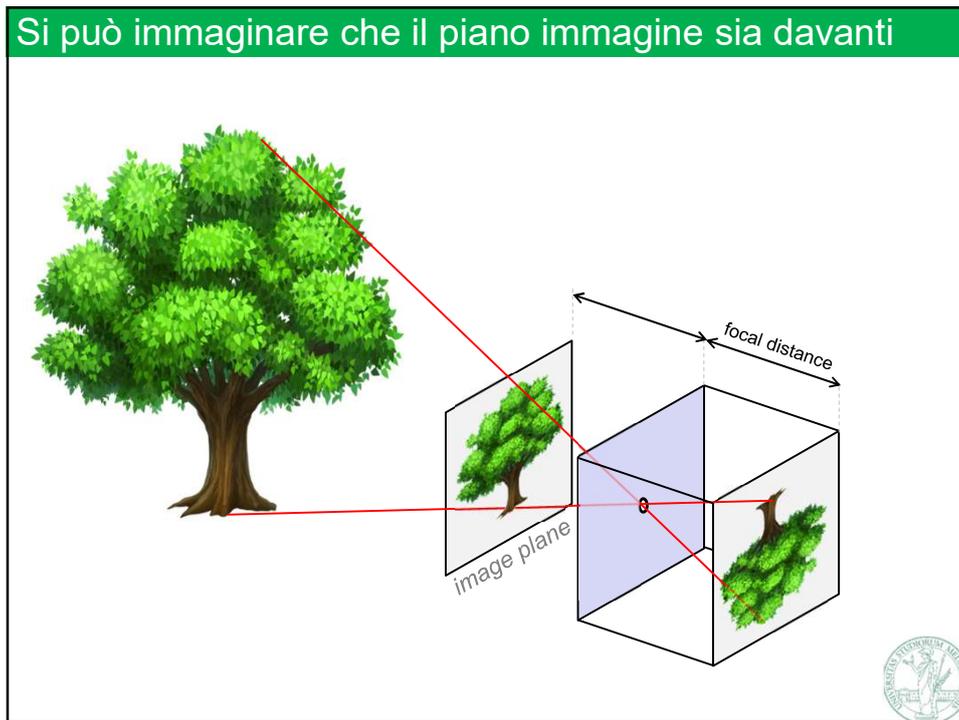
5



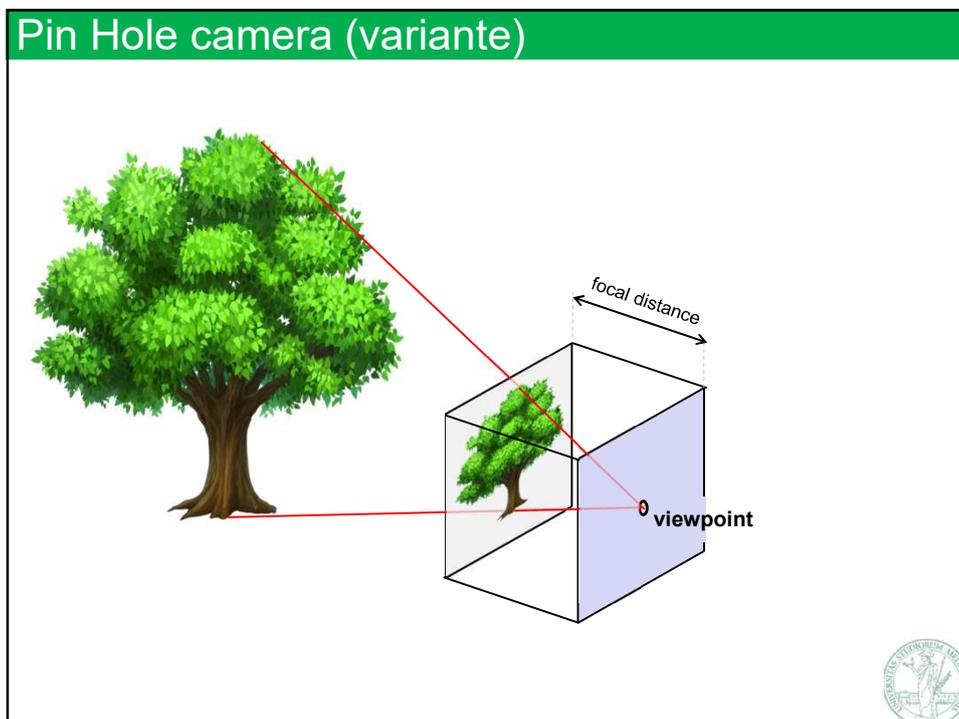
8



9



10



11

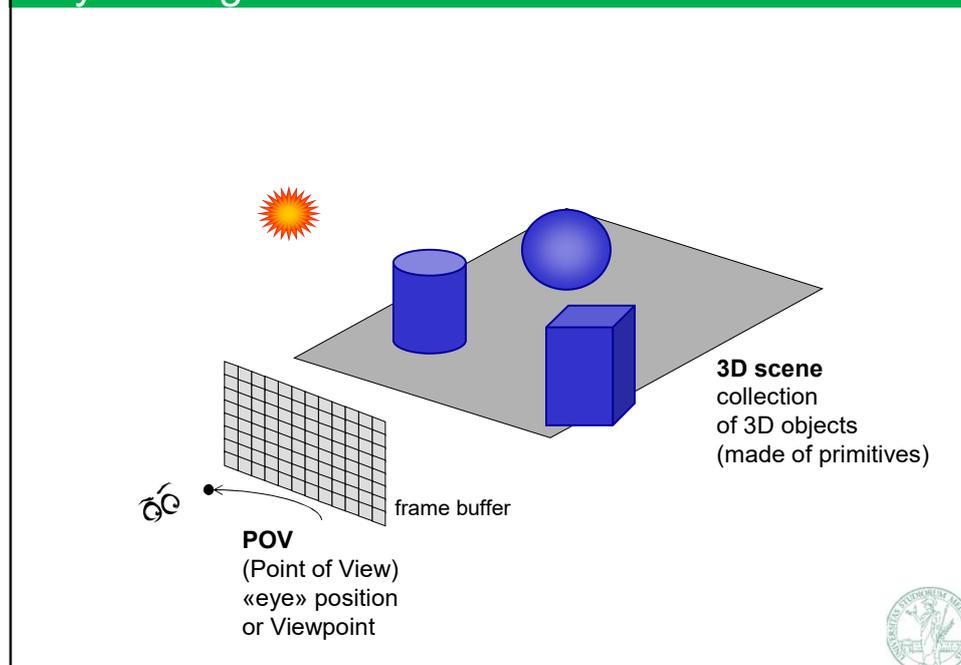
Pin-hole camera

- ✓ La pin-hole camera cattura tutta la luce che passa per il **POV**
 - ⇒ la luce = i fotoni
- ✓ La pin-hole camera misura quanta di questa luce arriva da ogni **direzione**
 - ⇒ ogni pixel rilevato $p[i,j]$ = quantità di luce che è passata dal POV in direzione (simile a) ($POV - P(i,j)$) durante il tempo di esposizione
- ✓ La luce è partita da degli emettitori (fonti di luce)
 - ⇒ prima di arrivare al POV, ha interagito con la scena!

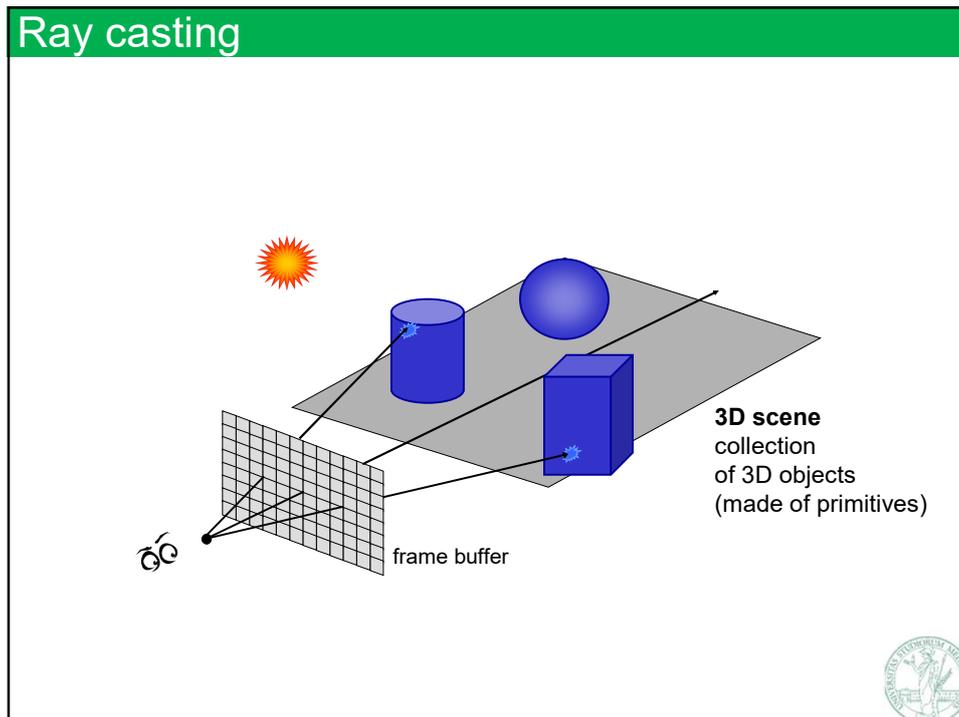


12

Ray casting



16



17

Ray-Casting

- ✓ Idea: seguire *a ritroso* i fotoni che raggiungono il POV
 - ⇒ per questo, detto anche : "*backwards*" ray tracing
- ✓ per ogni pixel sullo schermo:
 - ⇒ costruisco un raggio (detto il "*raggio primario*" di quel pixel)
 - ⇒ trovo le sue intersezioni con gli oggetti della scena
 - ⇒ scelgo l'intersezione più vicina al POV
- ✓ Implementazione:
 - ⇒ basata sul computo dell'*intersezione raggio-primitive* (che quindi deve essere ottimizzata)
 - ⇒ Numero di intersezioni da calcolare = numero di pixel

18

Ray Casting pseudo-code

```
For each pixel  $p$ :  
  make a ray  $r$  (viewpoint to  $p$ )  
  for each primitive  $o$  in scene:  
    find intersect( $r, o$ )  
  keep closest intersection  $o_j$   
  find color of  $o_j$  at  $p$ 
```



19

Cosa possiamo renderizzare

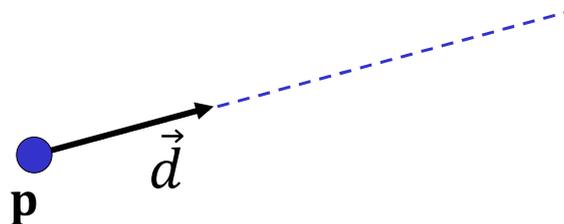
- ✓ Con un algoritmo di Ray-casting possiamo renderizzare qualsiasi cosa per la quale siamo in grado di computare l'intersezione con un raggio
 - ⇒ Vale anche per qualsiasi algoritmo di Ray-tracing – vedi sotto
- ✓ Esistono quindi ray-caster per il rendering di :
 - ⇒ Mesh poligonali
 - ⇒ Campi d'altezza
 - ⇒ Modelli impliciti
 - ⇒ Superfici parametriche, come patch di Bezier
 - ⇒ Etc
- ✓ In ciascun caso, serve di sapere implementare la procedura di intersezione raggio-oggetto
 - ⇒ Vediamo alcuni casi per semplici modelli impliciti



20

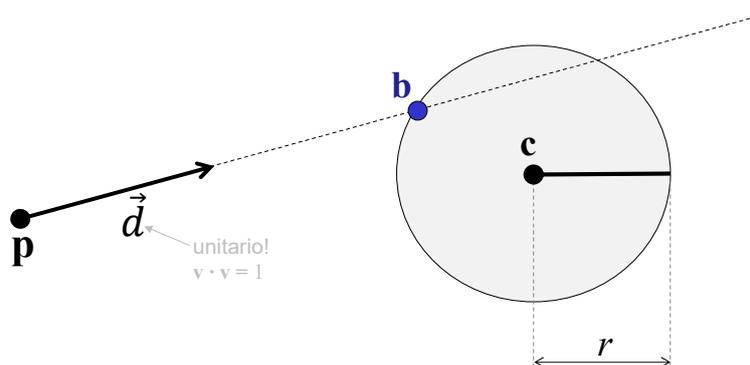
Rappresentazioni di un «raggio»

- ✓ Un raggio (ray) è una semiretta
- ✓ La possiamo modellare come
 - ⇒ Punto \mathbf{p} di partenza del raggio
 - ⇒ Vettore \vec{d} direzione (unitario o no, è una scelta)
- ✓ I punti sul raggio sono i punti
 - ⇒ $\mathbf{p} + k\vec{d}$ per un qualche scalare $k \geq 0$
 - ⇒ cioè «i punti raggiungibili a partire da \mathbf{p} facendo k passi in direzione \vec{d} »



21

Esempio: Intersezione raggio sfera 1/3



sfera (come superficie implicita):

$$f(\mathbf{q}) = 0 \quad \text{con} \quad f(\mathbf{q}) = \|\mathbf{q} - \mathbf{c}\|^2 - r^2$$

intersecare raggio con sfera = trovare un $k \geq 0$ (se \exists) t.c.

$$f(\mathbf{p} + k\vec{d}) = 0$$



22

Esempio: Intersezione raggio sfera 2/3

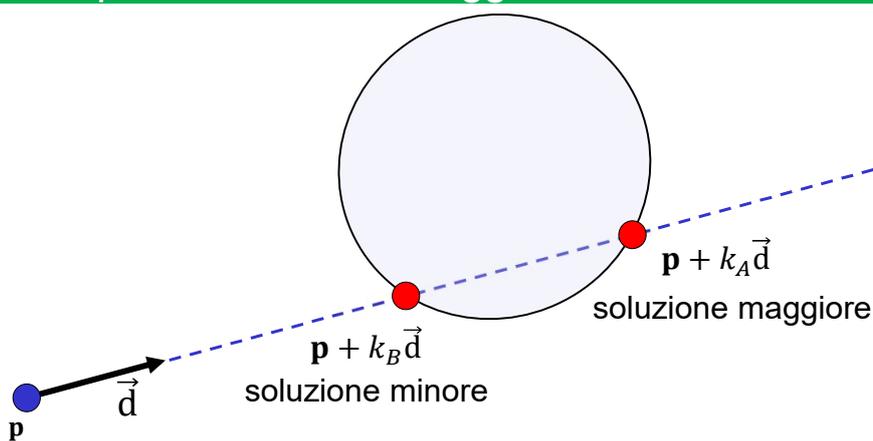
$$\begin{aligned}
 f(\mathbf{p} + k\vec{d}) &= 0 \\
 &\Leftrightarrow \\
 \|\mathbf{p} + k\vec{d} - \mathbf{c}\|^2 - r^2 &= 0 \\
 &\Leftrightarrow \|\vec{v}\|^2 = \vec{v} \cdot \vec{v} \\
 (\mathbf{p} + k\vec{d} - \mathbf{c}) \cdot (\mathbf{p} + k\vec{d} - \mathbf{c}) - r^2 &= 0 \\
 &\Leftrightarrow \\
 ((\mathbf{p} - \mathbf{c}) + k\vec{d}) \cdot ((\mathbf{p} - \mathbf{c}) + k\vec{d}) - r^2 &= 0 \\
 &\Leftrightarrow \\
 k^2 \|\vec{d}\|^2 + k 2 (\mathbf{p} - \mathbf{c}) \cdot \vec{d} + \|\mathbf{p} - \mathbf{c}\|^2 - r^2 &= 0
 \end{aligned}$$

Equazione 2° grado in k Prendo la soluzione minore. Se è > 0 abbiamo trovato k e quindi anche il punto di intersezione



23

Esempio: Intersezione raggio sfera 3/3



$$k_{A,B} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

devo ovviamente considerare la soluzione minore (quella col segno -)



24

Esempio: Intersezione Raggio / Piano

- ✓ piano definito da: un punto sul piano \mathbf{s} e la sua normale $\hat{\mathbf{n}}$
- ✓ punti su piano: { tutti i punti \mathbf{q} tali che $(\mathbf{q} - \mathbf{s}) \cdot \hat{\mathbf{n}} = 0$ }
- ✓ intersecare quel piano con un raggio dato $(\mathbf{p}, \vec{\mathbf{d}})$ = trovare un k (se \exists) t.c. $(\mathbf{p} + k\vec{\mathbf{d}})$ è un punto sul piano

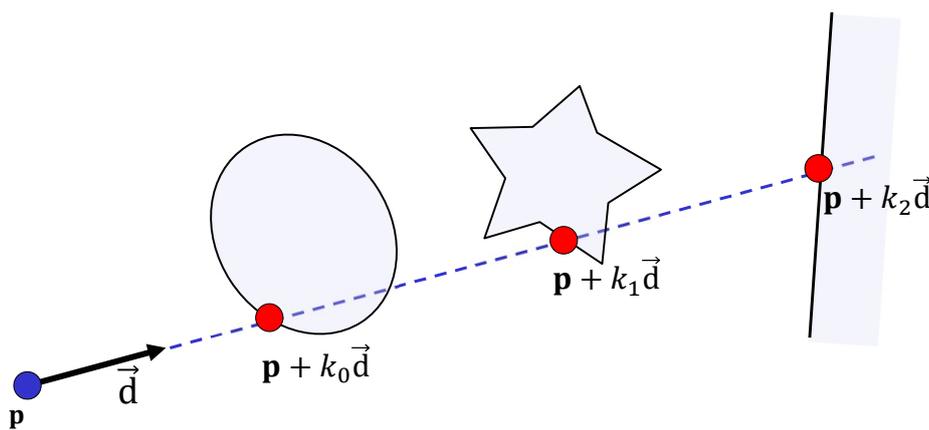
$$\begin{aligned}
 &(\mathbf{p} + k\vec{\mathbf{d}} - \mathbf{s}) \cdot \hat{\mathbf{n}} = 0 \\
 &\Leftrightarrow \\
 &(\mathbf{p} - \mathbf{s}) \cdot \hat{\mathbf{n}} + k\vec{\mathbf{d}} \cdot \hat{\mathbf{n}} = 0 \\
 &\Leftrightarrow \\
 &k(\vec{\mathbf{d}} \cdot \hat{\mathbf{n}}) = (\mathbf{s} - \mathbf{p}) \cdot \hat{\mathbf{n}} \\
 &\Leftrightarrow \\
 &k = \frac{(\mathbf{s} - \mathbf{p}) \cdot \hat{\mathbf{n}}}{\vec{\mathbf{d}} \cdot \hat{\mathbf{n}}}
 \end{aligned}$$

Eq. 1mo grado in k
 Se ha soluzione $k > 0$,
 allora intersezione esiste.

Se $\vec{\mathbf{d}} \cdot \hat{\mathbf{n}} = 0$, div per zero.
 Ma se è 0, allora
 il raggio è ortogonale a $\hat{\mathbf{n}}$
 cioè corre parallelo
 al piano: no intersez.

25

Intersezione raggio con più primitive



Basta prendere la primitiva che
 interseca col k **minore** fra $k_{0,1,2}$



26

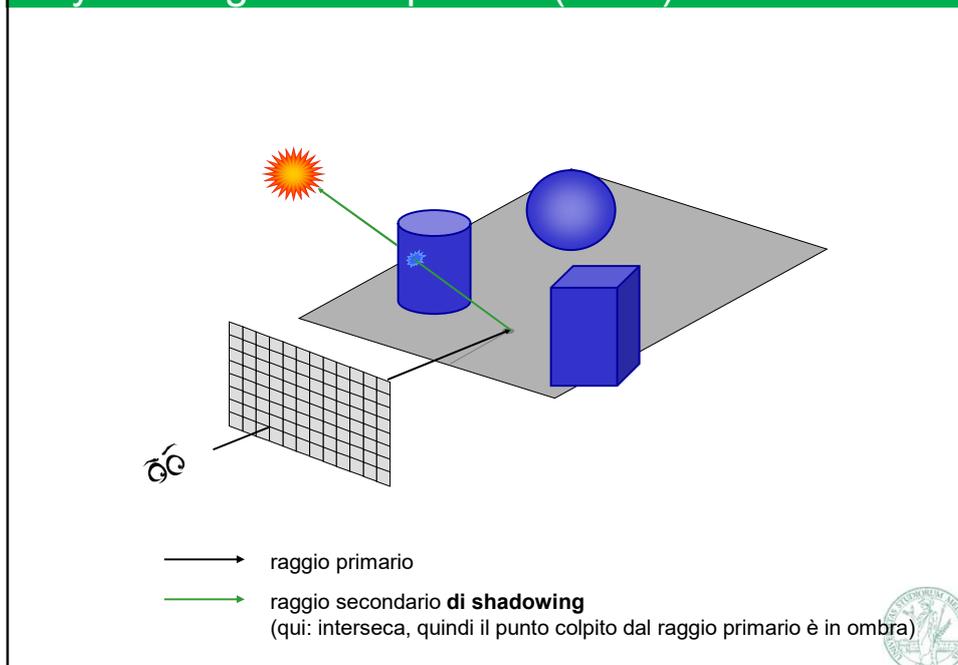
Ray Tracing (classe di algoritmi)

- ✓ Seguendo la stessa idea per un altro passo...
- ✓ **Tracciamo** a ritroso il percorso dei fotoni
 - ⇒ non solo dal POV ad un punto su un oggetto 3D,
 - ⇒ ma dall'oggetto 3D all'emettitore della luce
 - ⇒ seguendo vari passaggi
- ✓ Questo richiede di computare le intersezioni su altri raggi
 - ⇒ Detti raggi «secondari»
 - ⇒ Vediamo esempi per calcolo di ombre, di riflessioni, di rifrazioni (oggetti trasparenti)
 - ⇒ Nota: richiede solo di eseguire la stessa procedura, intersezione raggio primitiva, ancora altre volte per pixel.
- ✓ Algoritmi basati su tracciamento di raggi sono detti di Ray-Tracing (compreso il semplice ray-casting)

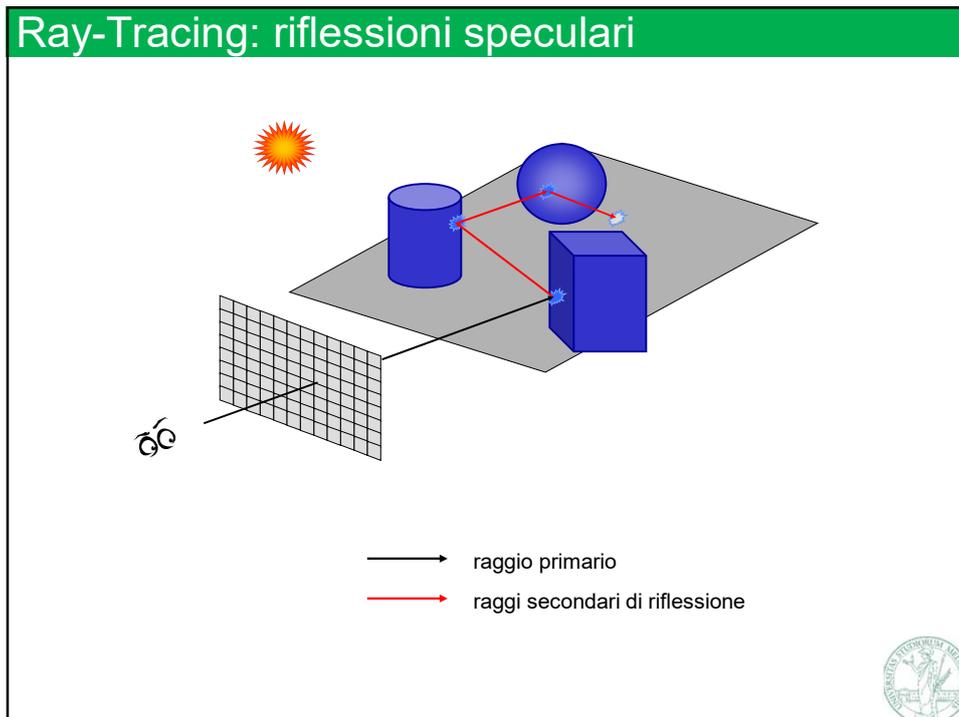


27

Ray-Tracing: ombre portate (nette)



28



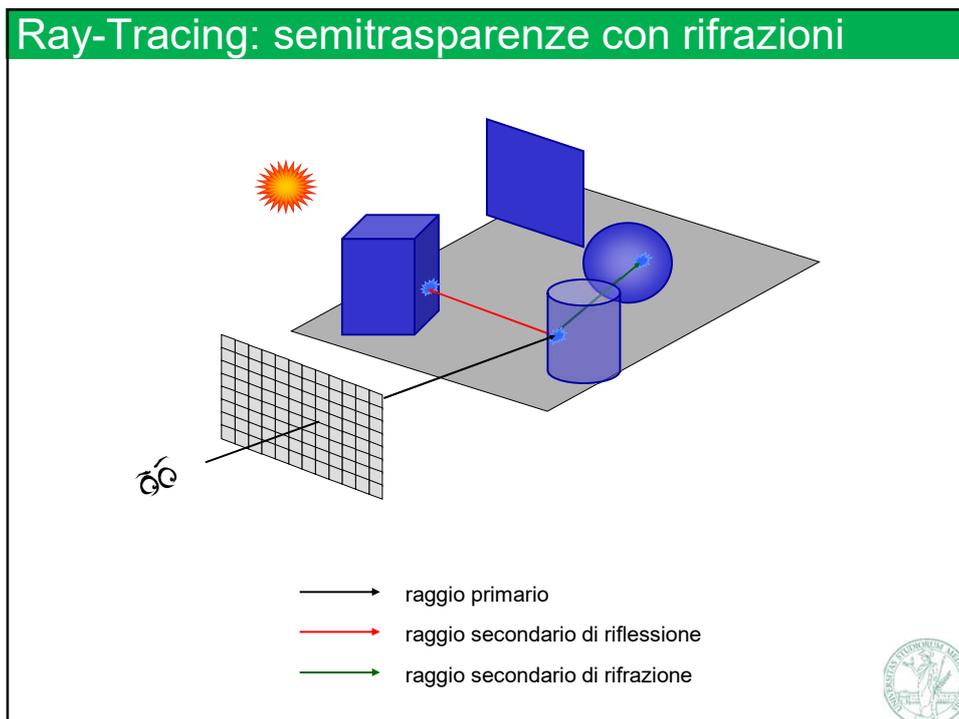
35

Come computare la direzione raggio riflesso?

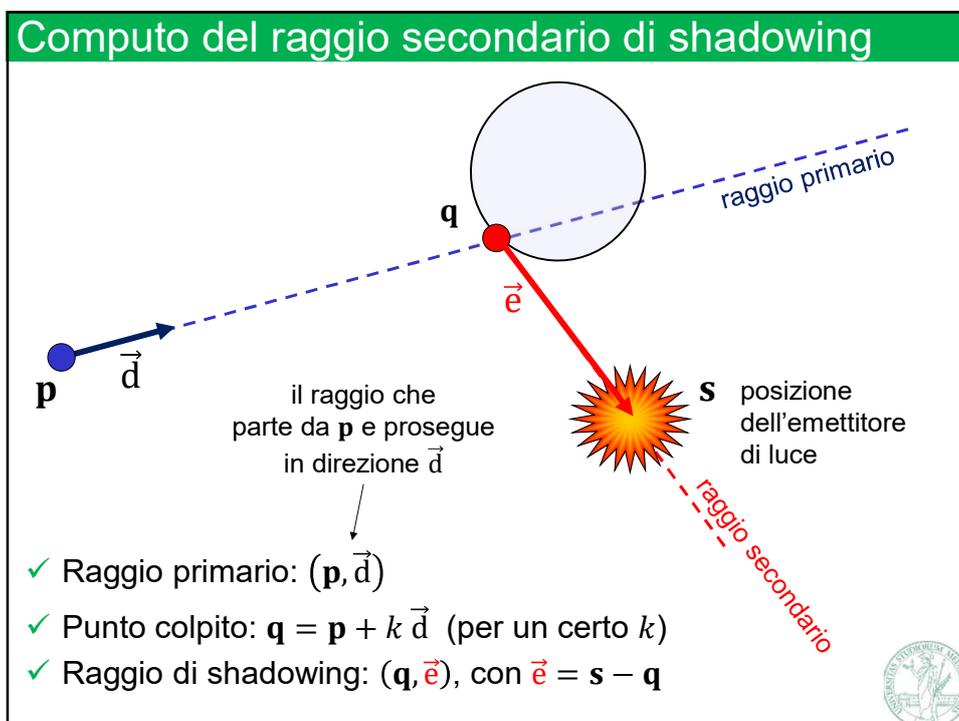
- ✓ Input:
 - ⇒ Direzione del raggio primario
 - ⇒ Normale della superficie nel punto colpito dal raggio primario
- ✓ Output:
 - ⇒ Direzione del raggio secondario
 - ⇒ (insieme alla posizione colpita dal raggio primario, questo modella il raggio secondario di riflessione)



36



37



38

Computo del raggio secondario di shadowing

\checkmark Raggio primario: (\mathbf{p}, \vec{d})
 \checkmark Punto colpito: $\mathbf{q} = \mathbf{p} + k \vec{d}$ (per un certo k)
 \checkmark Raggio di shadowing: (\mathbf{q}, \vec{d}_R) ,
 con \vec{d}_R il vettore \vec{d} riflesso da un piano di normale \hat{n}

Normale della superficie in \mathbf{q}

39

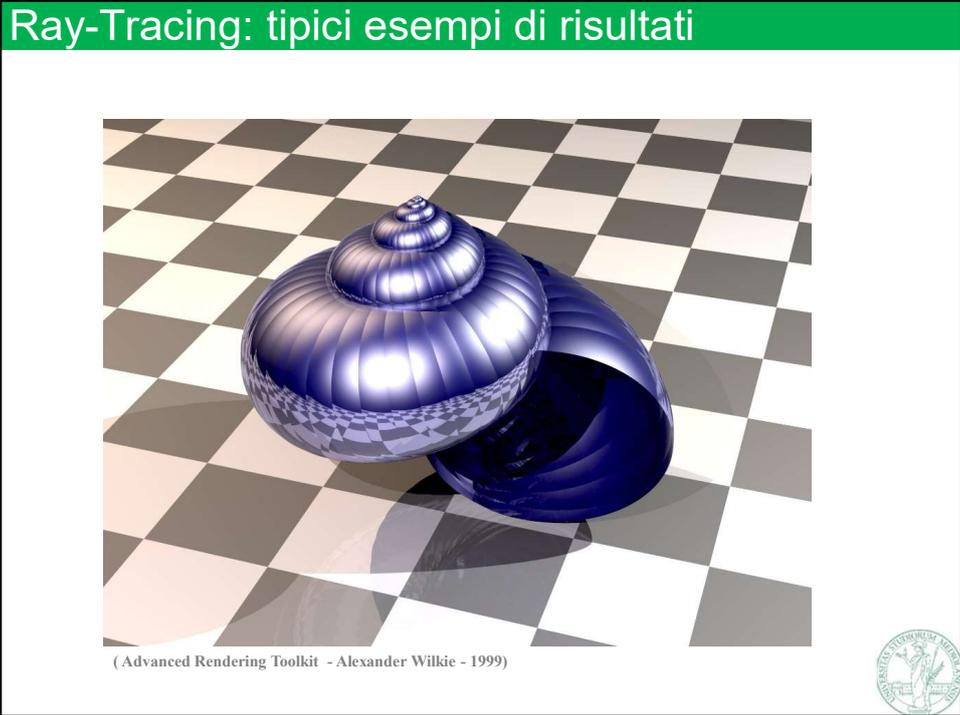
Computo di una direzione riflessa da un piano

$$\vec{d}_R = \vec{d} + 2 h \hat{n}$$

dove

$$h = (-\vec{d} \cdot \hat{n})$$

40



42



43

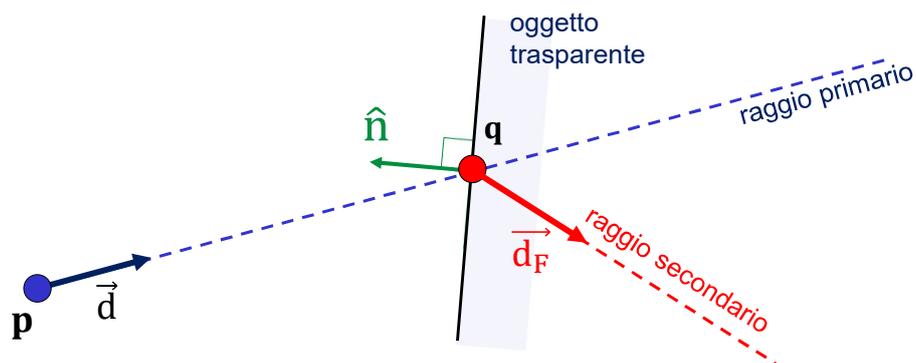
Algoritmi di Ray-tracing: efficienza

- ✓ Un'implementazione triviale per
 - ⇒ una scena con M oggetti («primitive»)
 - ⇒ immagini $N \times N$ pixel
 - ⇒ con K raggi per pixel (1 primario e $K-1$ secondari)richiede ben $N^2 M K$ intersezioni raggio-primitiva!
 - ⇒ È un numero molto molto elevato
 - ⇒ Complessità temporale dell'algoritmo: $O(N^2 M K)$
 - ⇒ Per questo motivo, questo tipo di algoritmi è usato soprattutto per il **rendering offline**
- ✓ Sono possibili però ottimizzazioni...
 - ⇒ per testare solo un sottoinsieme delle primitive
 - ⇒ **Implementazioni parallelizzate** attraverso GPU (infatti ogni pixel può essere computato indipendentemente dagli altri: elevato livello di **parallelismo implicito**)



45

Computo del raggio secondario di rifrazione



- ✓ Raggio primario: $(\mathbf{p}, \vec{\mathbf{d}})$
- ✓ Punto colpito: $\mathbf{q} = \mathbf{p} + k \vec{\mathbf{d}}$ (per un certo k)
- ✓ Raggio di shadowing: $(\mathbf{q}, \vec{\mathbf{d}}_F)$,
con $\vec{\mathbf{d}}_F$ il vettore $\vec{\mathbf{d}}$ rifranto da un piano di normale $\hat{\mathbf{n}}$
 - La formula della direzione rifranta $\vec{\mathbf{d}}_F$ (che non vediamo) dipende anche da l'«*indice di rifrazione*» del materiale



47