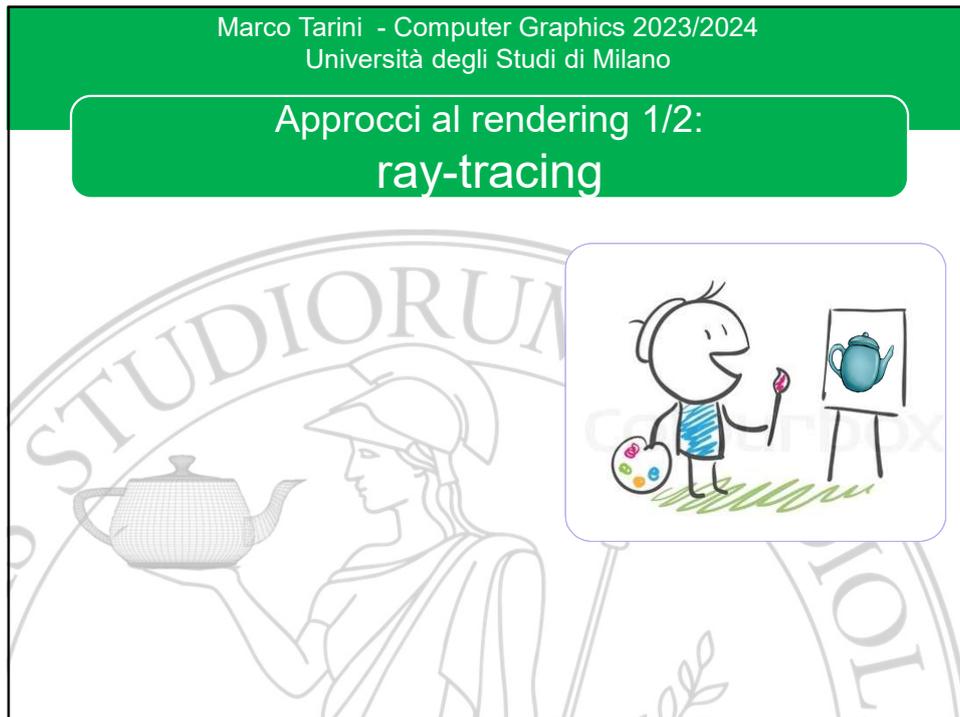


Marco Tarini - Computer Graphics 2023/2024
Università degli Studi di Milano

Approcci al rendering 1/2: ray-tracing



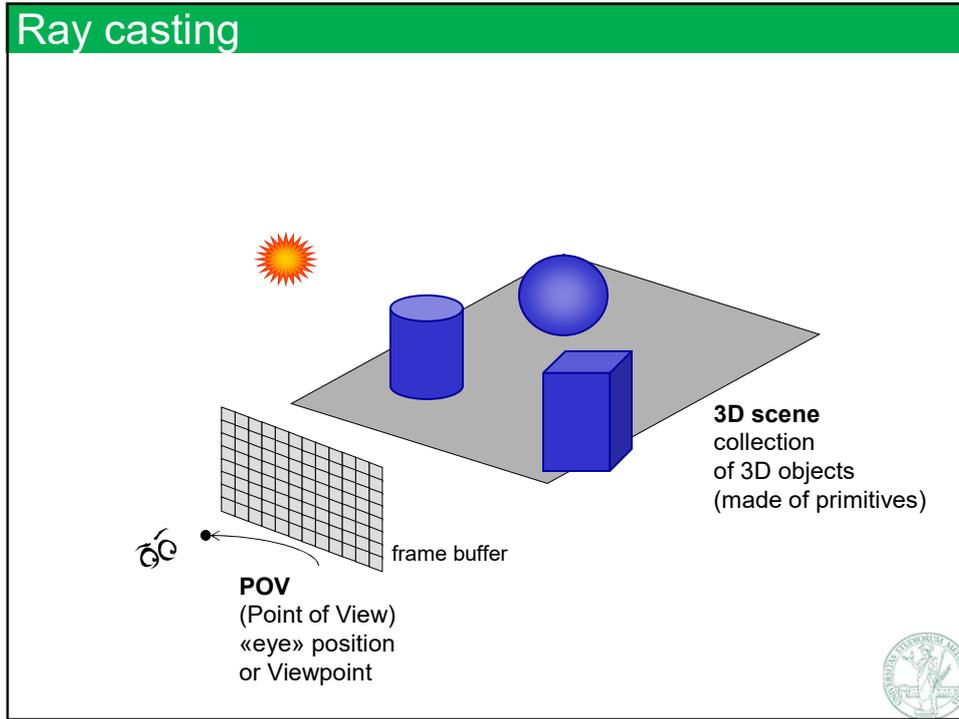
14

Ray-Casting

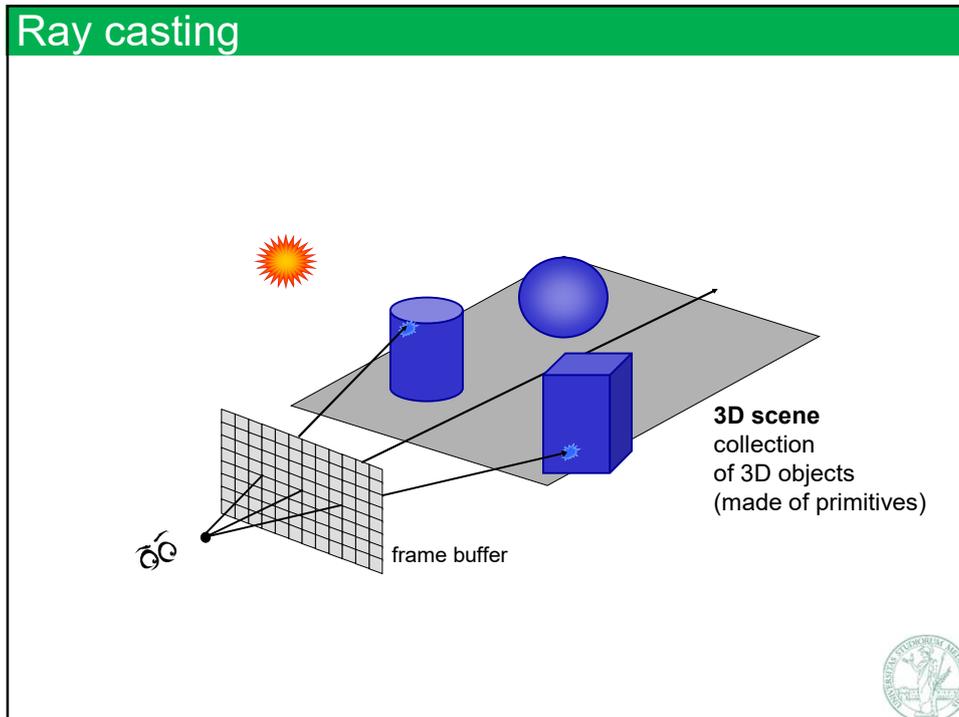
- ✓ Idea: seguire *a ritroso* i fotoni che raggiungono il POV
 - ⇒ per questo, detto anche : "*backward*" ray tracing
- ✓ per ogni pixel sull'immagine da produrre:
 - ⇒ produco un raggio che parte dal POV e attraversa quell pixel (detto il "*raggio primario*" di quel pixel)
 - ⇒ trovo le sue intersezioni con gli oggetti della scena
 - ⇒ scelgo l'intersezione più vicina al POV
- ✓ Implementazione:
 - ⇒ Numero di raggi da processare = numero di pixel dell'immagine da produrre
 - ⇒ Per ogni raggio, è necessario computare la sua *intersezione con gli oggetti che costituiscono la scena (cioè le "primitive")* (quest'operazione deve quindi deve essere ottimizzata)



15



16



17

Ray Casting pseudo-code

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



18

Primitive di rendering (precisazione)

- ✓ Per “primitiva di rendering” si intende una descrizione di un'entità (3D o 2D) che un dato algoritmo di rendering può processare direttamente
 - ⇒ Diversi algoritmi di rendering prevedono diverse primitive
- ✓ Per esempio...
 - ⇒ ...se disponiamo di un algoritmo in grado di processare triangoli 3D, ma non quadrilateri 3D, (primitiva di rendering = triangolo) allora possiamo renderizzare direttamente una tri-mesh, ma una quad-mesh deve prima essere convertita in una tri-mesh (attraverso diagonal split)
 - ⇒ ...se disponiamo di un algoritmo di rendering in grado di processare campi di altezza, allora non è necessario convertire il campo di altezza in una rappresentazione poligonale
- ✓ Come vedremo, la primitiva prevista dagli algoritmi di rendering più diffusi è il triangolo (ma non è l'unica!)
 - ⇒ A questa considerazione si deve la predominanza delle tri-mesh come modelli 3D



19

Primitive di rendering, nel caso del Ray-casting

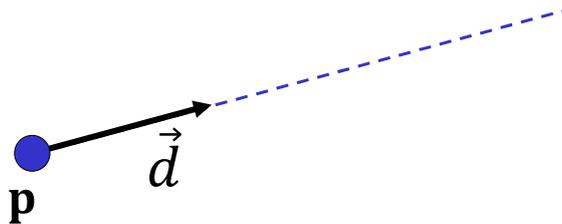
- ✓ Con un algoritmo di Ray-casting possiamo renderizzare qualsiasi cosa per la quale siamo in grado di computare l'intersezione con un raggio
- ✓ Esistono quindi ray-caster per il rendering di :
 - ⇒ Mesh triangolari
(di gran lunga il caso più rilevante, industrialmente)
 - ⇒ Mesh poligonali generiche (incluso quad mesh)
 - ⇒ Campi d'Altezza (che non abbiamo visto)
 - ⇒ Modelli impliciti (che non abbiamo visto)
 - ⇒ Superfici parametriche, come patch di Bezier
 - ⇒ Etc
- ✓ In ciascun primitiva, è necessario implementare la procedura di intersezione raggio-primitiva
 - ⇒ Vediamo alcuni casi per semplici modelli impliciti



20

Rappresentazione di un «raggio»

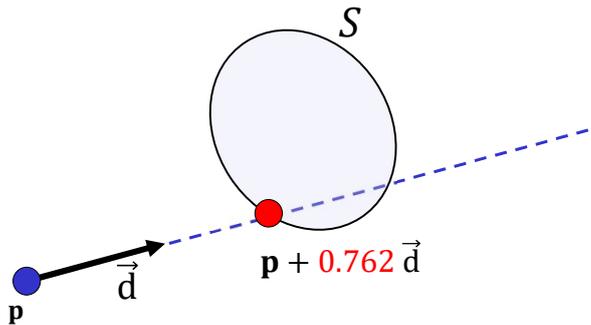
- ✓ Un raggio = una semiretta (nota, in inglese: «ray» non «radius»)
- ✓ Rappresentabile come
 - ⇒ Un dato punto \mathbf{p} di partenza del raggio
 - ⇒ Un dato vettore \vec{d} direzione (unitario o no, è una scelta implementativa)
- ✓ Il raggio è dato da tutti i punti esprimibili come $\mathbf{p} + k \vec{d}$
 - ⇒ con k scalare non-negativo, $k \geq 0$
(non-negativo, perché il raggio è una **semi**-retta)
 - ⇒ cioè «i punti raggiungibili a partire da \mathbf{p} facendo k passi in direzione \vec{d} »



22

Intersezione raggio – primitiva

✓ Trovare l'intersezione con una data superficie S significa trovare uno scalare k tale che $\mathbf{p} + k\vec{\mathbf{d}}$ stia su S

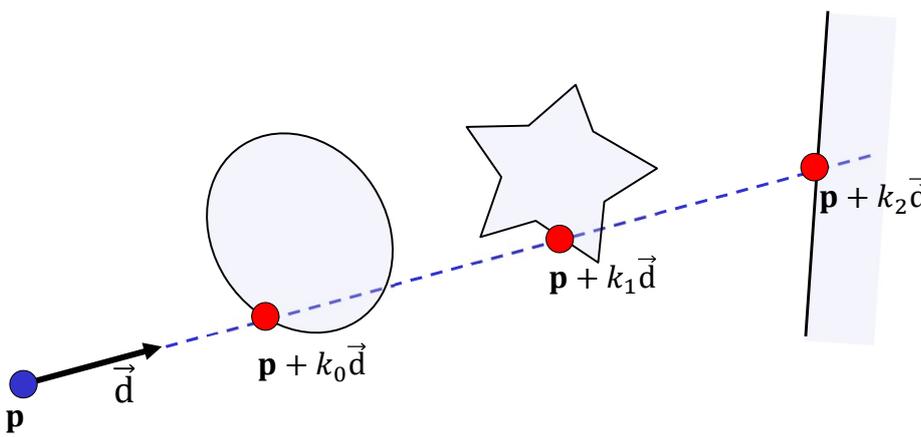


The diagram illustrates a ray starting at point \mathbf{p} (blue dot) and traveling in direction $\vec{\mathbf{d}}$ (black arrow). A dashed blue line represents the ray's path. It intersects a light blue oval-shaped surface S at a red dot labeled $\mathbf{p} + 0.762\vec{\mathbf{d}}$.



23

Intersezione raggio-scena (cioè tante primitive)



The diagram shows a ray starting at point \mathbf{p} (blue dot) and traveling in direction $\vec{\mathbf{d}}$ (black arrow). A dashed blue line represents the ray's path. It intersects three primitives: a light blue oval at point $\mathbf{p} + k_0\vec{\mathbf{d}}$, a light blue star at point $\mathbf{p} + k_1\vec{\mathbf{d}}$, and a light blue rectangle at point $\mathbf{p} + k_2\vec{\mathbf{d}}$.

Basta prendere la primitiva che interseca col k **minore** fra $k_{0,1,2}$
Questo implementa il fenomeno dell'**occlusione** ("le cose davanti coprono quelle dietro")



24

Intersezione raggio – oggetto singolo

- ✓ Vedremo ora alcuni casi di computo dell'intersezione di un raggio con alcune semplici primitive
- ✓ (accertarsi di aver capito bene il prodotto dot)

p \vec{d} $p + k \vec{d}$ S



25

Costruzione del raggio primario

- ✓ Dato un pixel Q , quale è il suo raggio primario?
- ✓ Risposta:
 - $\Rightarrow p$ (punto di partenza del raggio): il POV
 - $\Rightarrow \vec{d}$ (direzione del raggio): $(q - p)$
(eventualmente, normalizzato, cioè scalato di $\frac{1}{\|q-p\|}$)

dove q è la posizione 3D del pixel Q sul piano immagine

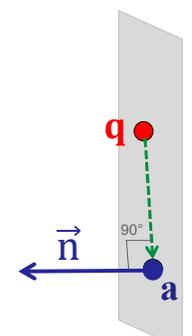
$\odot p$ q



28

Esempio: Intersezione Raggio – Piano 1/3

- ✓ piano definito da: un punto su di esso \mathbf{a} e il suo vettore normale $\vec{\mathbf{n}}$ (l'orientamento del piano)
- ✓ punti su piano: cioè tutti i punti \mathbf{q} tali che il vettore $(\mathbf{a} - \mathbf{q})$ è ortogonale ad $\vec{\mathbf{n}}$, cioè $(\mathbf{a} - \mathbf{q}) \cdot \vec{\mathbf{n}} = 0$
- ✓ intersecare il piano con un raggio dato $(\mathbf{p}, \vec{\mathbf{d}}) =$ trovare un k (se \exists) t.c. $(\mathbf{p} + k\vec{\mathbf{d}})$ sia un punto sul piano

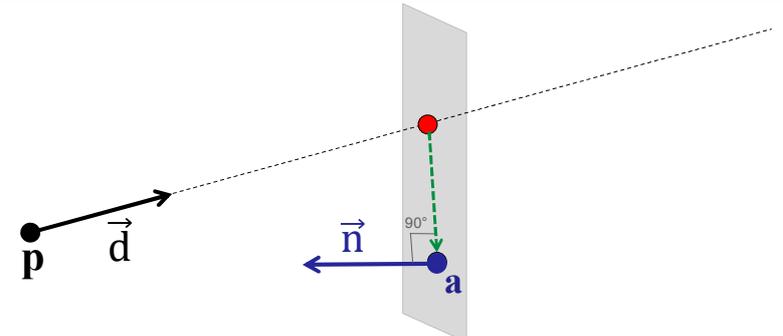


Piano passante per il punto \mathbf{a} e avente normale $\vec{\mathbf{n}}$



30

Esempio: Intersezione raggio piano 2/3



intersecare raggio con sfera = trovare un $k \geq 0$ (se \exists) tale che

Il vettore verde

$$\underbrace{(\mathbf{a} - (\mathbf{p} + k\vec{\mathbf{d}}))}_{\text{Il punto rosso}} \cdot \vec{\mathbf{n}} = 0$$

Il punto rosso

perché so che il vettore verde tratteggiato deve essere ortogonale ad $\vec{\mathbf{n}}$



31

Esempio: Intersezione raggio piano 3/3

Trovo k (che è l'unica incognita) manipolando l'equazione

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

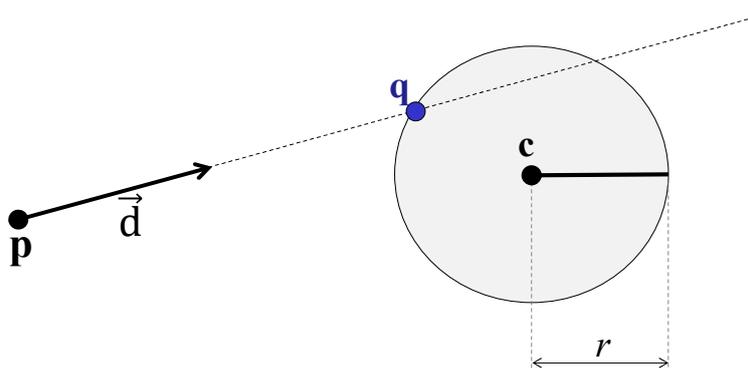
Se $\vec{\mathbf{d}} \cdot \hat{\mathbf{n}} = 0$, div per zero.
 Ma se è così, allora
 il raggio è ortogonale a $\hat{\mathbf{n}}$
 cioè corre parallelo
 al piano: no intersez.

Se $k > 0$, allora il raggio interseca il piano, e l'intersezione è alla posizione $\mathbf{a} + k\vec{\mathbf{d}}$
 altrimenti, il raggio non interseca il piano



32

Esempio: Intersezione raggio sfera 1/3



Sfera (di centro nel punto \mathbf{c} e radius r):
 luogo di punti distanti esattamente r da \mathbf{c} .

Cioè: \mathbf{q} appartiene alla sfera sse

$$\|\mathbf{q} - \mathbf{c}\| = r \quad \text{cioè} \quad \|\mathbf{q} - \mathbf{c}\|^2 = r^2 \quad \text{cioè} \quad \|\mathbf{q} - \mathbf{c}\|^2 - r^2 = 0$$

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

il punto $\mathbf{q} = \mathbf{p} + k\vec{\mathbf{d}}$ soddisfa l'equazione qui sopra



35

Esempio: Intersezione raggio sfera 2/3

Punto sul Raggio
(per un valore ignoto k)

$$\| \mathbf{p} + k\vec{d} - \mathbf{c} \|^2 - r^2 = 0$$

$$\Leftrightarrow \| (\mathbf{p} - \mathbf{c}) + k\vec{d} \|^2 - r^2 = 0$$

$$\Leftrightarrow ((\mathbf{p} - \mathbf{c}) + k\vec{d}) \cdot ((\mathbf{p} - \mathbf{c}) + k\vec{d}) - r^2 = 0$$

$\|\vec{v}\|^2 = \vec{v} \cdot \vec{v}$

$$k^2 \|\vec{d}\|^2 + 2k(\mathbf{p} - \mathbf{c}) \cdot \vec{d} + \|\mathbf{p} - \mathbf{c}\|^2 - r^2 = 0$$

SCALARE SCALARE SCALARE

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



36

Esempio: Intersezione raggio sfera 3/3

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

devo ovviamente considerare la soluzione minore
(quella scegliendo il segno meno nel \pm)



37

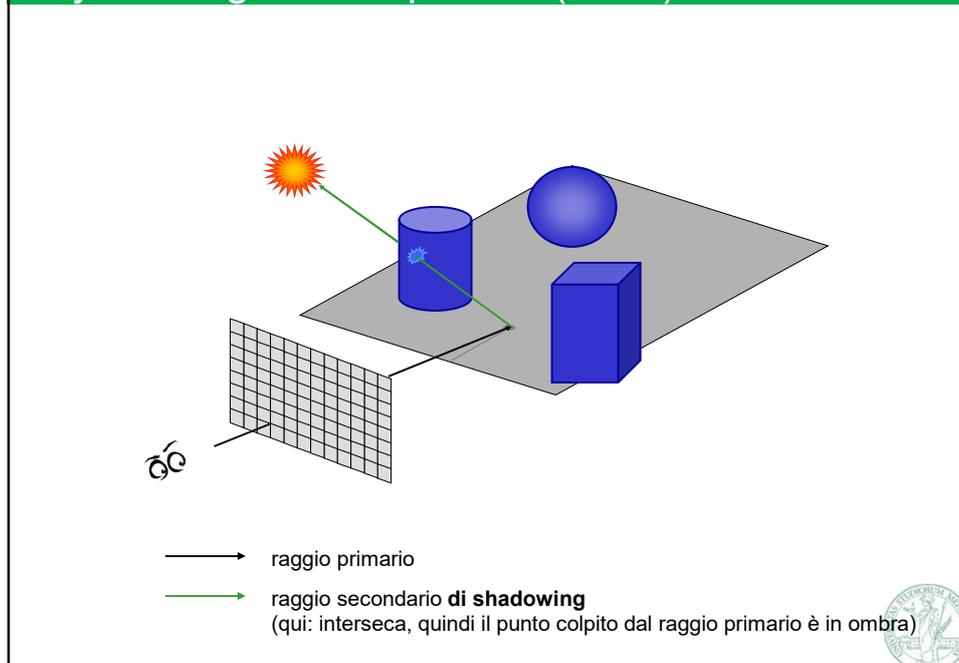
Ray Tracing (classe di algoritmi)

- ✓ 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 che ha emesso quel fotone
- ✓ Questo richiede di computare le intersezioni su un ulteriore raggio
 - ⇒ Detto di shadowing
 - ⇒ E' un esempio di raggio «secondario» (ne vedremo altri)
 - ⇒ Nota: è la *stessa* procedura, «intersezione raggio-primitiva», eseguita più volte per uno stesso pixel.
- ✓ Algoritmi basati su tracciamento di raggi sono detti di Ray-Tracing (compreso il semplice ray-casting)

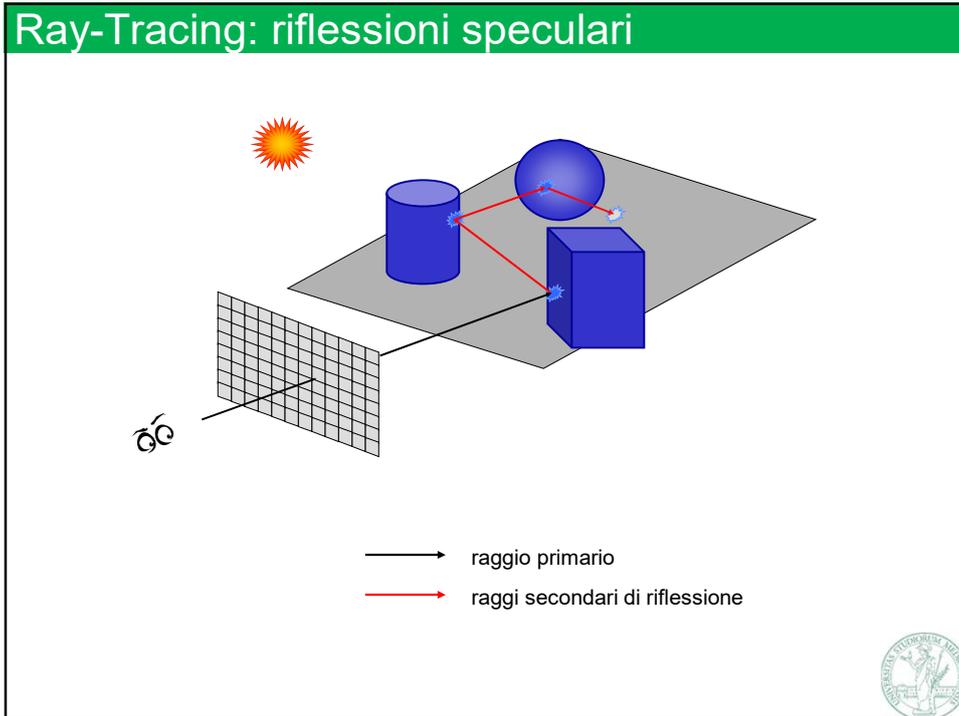


38

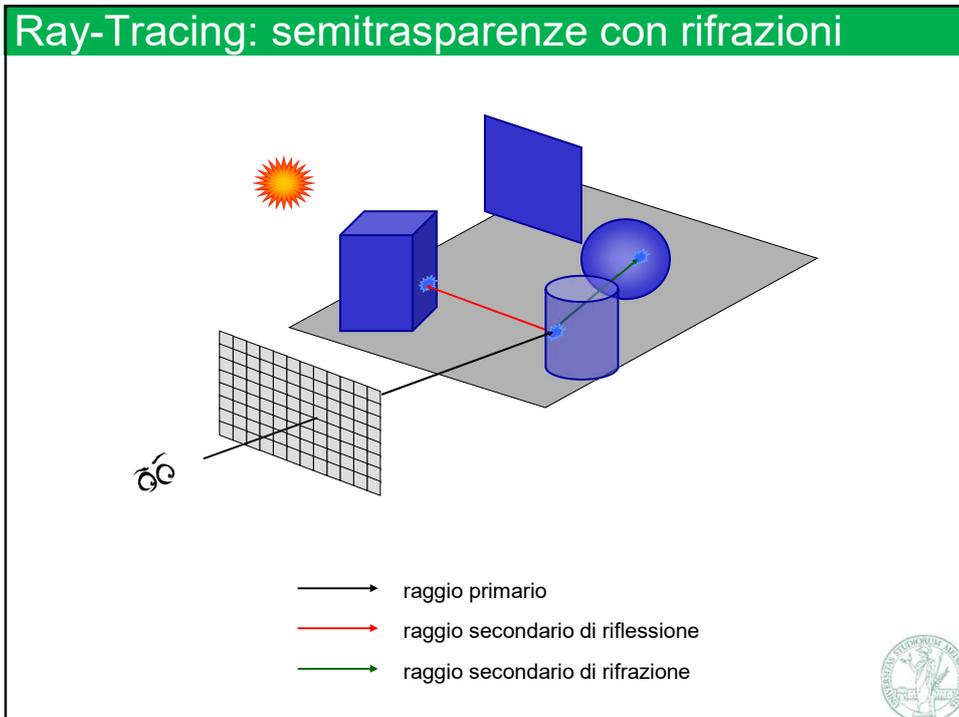
Ray-Tracing: ombre portate (nette)



39



40



41

Computo del raggio secondario di shadowing

il raggio che parte da \mathbf{p} e prosegue in direzione $\vec{\mathbf{d}}$

\mathbf{s} { posizione dell'emettitore di luce

- ✓ Raggio primario: $(\mathbf{p}, \vec{\mathbf{d}})$
- ✓ Punto colpito: $\mathbf{q} = \mathbf{p} + k \vec{\mathbf{d}}$ (per un certo k)
- ✓ Raggio di shadowing: $(\mathbf{q}, \vec{\mathbf{e}})$, con $\vec{\mathbf{e}} = \frac{(\mathbf{s} - \mathbf{q})}{\|\mathbf{s} - \mathbf{q}\|}$

42

Computo del raggio secondario di riflessione

Normale della superficie in \mathbf{q}

- ✓ Raggio primario: $(\mathbf{p}, \vec{\mathbf{d}})$
- ✓ Punto colpito: $\mathbf{q} = \mathbf{p} + k \vec{\mathbf{d}}$ (per un certo k)
- ✓ Raggio di riflessione: $(\mathbf{q}, \vec{\mathbf{d}}_R)$, dove $\vec{\mathbf{d}}_R$ è il vettore $\vec{\mathbf{d}}$ riflesso da un piano di normale $\hat{\mathbf{n}}$
- ✓ Vediamo come calcolarlo (semplice "ottica meccanica")

43

Calcolo della direzione riflessa \vec{d}_R (1/2)

Dal disegno si capisce che...

(1) Il vettore $\vec{d} + \vec{d}_R$ è ortogonale ad \hat{n}
 Cioè:
 $(\vec{d} + \vec{d}_R) \cdot \hat{n} = 0$

(2) Il vettore \vec{d}_R è ottenibile sommando a \vec{d} un vettore allineato ad \hat{n}
 Cioè:
 $\vec{d}_R = \vec{d} + h \hat{n}$
 (per un dato scalare h , da trovare)

45

Calcolo della direzione riflessa \vec{d}_R (2/2)

Sostituedo, trovo
 $\vec{d}_R = \vec{d} + h \hat{n}$
 con $h = -2 \vec{d} \cdot \hat{n}$

46

Computo del raggio secondario di rifrazione

oggetto trasparente

raggio primario

raggio secondario

\hat{n}

\mathbf{q}

\mathbf{p}

\vec{d}

\vec{d}_R

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

Una formula di meccanica ottica, (che non vediamo) descrive come trovare la direzione rifratta \vec{d}_R dato l'«indice di rifrazione» del materiale

47

Ray-Tracing : tipici esempi di risultati

(Advanced Rendering Toolkit - Alexander Wilkie - 1999)

(with CUDA)

52

Algoritmi di Ray-tracing: efficienza

- ✓ Un'implementazione **triviale** per
 - ⇒ una scena con M «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 troppo 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 molte 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**)



54

Ray tracing: alcune varianti

- ✓ Molti algoritmi e varianti si basano su principi simili.
- ✓ Alcuni di questi:
 - ⇒ **Ray-casting**:
 - un solo raggio (primario) viene "mandato" (*cast*) per pixel
 - ⇒ **Ray-tracing**:
 - Ogni raggio viene "tracciato" (*traced*) attraverso vari fenomeni successivi riflessione o rifrazione, attraverso raggi secondari
 - Nota: in ordine inverso rispetto alla realtà, come al solito
 - (è anche il usato come nome della classe generale di questi algoritmi)
 - ⇒ **Path-tracing**:
 - Per ogni pixel, lanciare molti raggi, che seguono i path pseudo-casuali ("monte carlo sampling") con una distribuzione che riflette quella reale
 - Mediare i risultati per ottenere il pixel finale
 - Più raggi vengono mandati, maggiore il realismo
 - ⇒ **Ray-marching**:
 - Una classe di algoritmi per accelerare l'intersezione raggio-primitiva
 - Principio: raggio viene spezzato in una serie di passi (segmenti) successivi, per ciascuno dei quali si devono testare solo le primitive a lui vicine (invece di tutte)

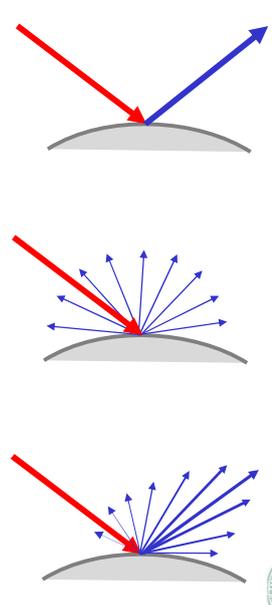
Nota:
questa
terminologia
può variare
leggermente
da una fonte
all'altra



55

Modelli *semplificati* di riflessione della luce

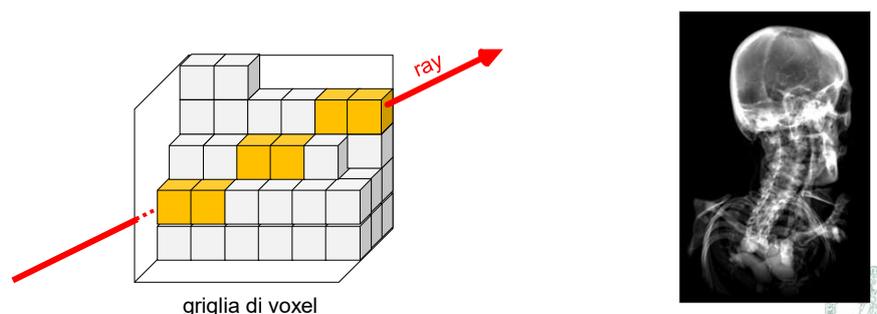
- ✓ Riflessioni speculari
⇒ luce (fotoni) rimbalza proprio come una pallina da ping-pong su di un tavolo
- ✓ Riflessioni diffuse
⇒ luce riflessa in ogni direzione possibile in egual misura (i fotoni vengono riflessi in tutte le direzioni)
- ✓ Miste (“glossy”)
⇒ luce riflessa in ogni direzione ma maggiormente in quelle simili alla direzione speculare



56

Esempi: Ray tracing su alcuni tipi di modelli visti

- ✓ Ray-tracing di volume di voxel:
(con ciascun voxel = valore di densità)
 - ⇒ È una forma comune di **direct volume rendering**
 - ⇒ Per ogni raggio primario, identifico i voxel attraversati
 - ⇒ Assegno il pixel corrispondente ad una funzione di tutti i valori di intensità dei voxel attraversati
 - ⇒ per esempio, banalmente, il tono di grigio corrispondente al valore max



griglia di voxel

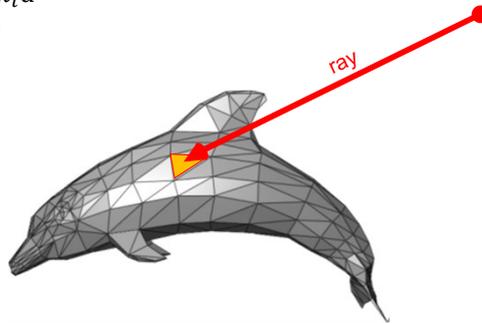


59

Esempi: Ray tracing su alcuni tipi di modelli visti

✓ Raytracing di tri-mesh

- ⇒ Per ogni triangolo:
trovare l'intersezione $\mathbf{p} + k_i \vec{d}$
raggio-triangolo, se esiste
(come implementare
questo calcolo?)
- ⇒ Prendere l'intersezione
con il k_i minore



E' una soluzione molto generale, perché è sempre possibile tradurre i nostri modelli 3D in una mesh.
Ma: può essere una soluzione molto onerosa,
(molte primitive sono necessarie per approssimare la forma),
e la discretizzazione introduce errori



62

Nota: esempio di codice di ray-tracing

- ✓ L'algoritmo di ray-tracing, è concettualmente semplice
 - ⇒ Una versione di base (non ottimizzata, e su poche primitive semplici) può essere implementata in poche centinaia di righe di codice
- ✓ Nota «storica»: nelle edizioni precedenti di questo corso (di 9 CFU), dedicando 4-5 lezioni a questo, era stato possibile implementare in classe, da zero, un ray-tracer giocattolo
 - ⇒ In un linguaggio chiamato GLSL
- ✓ Il codice, disponibile in rete, e può essere un esempio utile:
 - ⇒ https://tarini.di.unimi.it/glsitoj/?raytracer_riflessi_materiale.glsl
 - ⇒ Il link è anche sul sito del corso
- ✓ Questo codice (che viene eseguito su ogni pixel) contiene:
 - ⇒ Computo del raggio primario per il pixel processato
 - ⇒ Computo intersezione raggio-piano
 - ⇒ Computo intersezione raggio-sfera
 - ⇒ Computo della PRIMA intersezione con le primitive (occlusione)
 - ⇒ Computo del raggio secondario di shadowing
 - ⇒ Computo del raggio secondario di riflessione



63