Capitolo 8
Rendering Globale

Diapositive a corredo del libro: "Fondamenti di Grafica Tridimensionale Interattiva"
R. Scateni, P. Cignoni, C. Montani e R. Scopigno - McGrawHill Italia

Cap. 8 - Contenuti

- 8.1 Dal modello locale ai modelli globali
  - Limitazioni del modello locale, effetti globale e modi per approssimarli in un contesto locale
- 8.2 Ray-tracing
- 8.3 Radiosity
- 8.4 Rendering non-fotorealistico

Sezione 8.1

Dal modello locale
ai modelli globali

Rendering Locale

- Il modello di rendering che abbiamo finora visto è detto **locale:**
  - Ogni primitiva è trattata in maniera **indipendente** da tutte le altre
  - Il concetto di scena, la presenza di altri oggetti etc, interessano solo l'utente/programmatore e non il renderer
  - Il modo in cui è disegnato un triangolo dipende solo da:
    - Le caratteristiche del triangolo stesso
    - Lo stato dello z-buffer
    - Le luci presenti nella scena

Rendering locale

- Questo tipo di modello ha numerosi vantaggi:
  - Semplicità
  - Parallelismo a livello di primitiva
  - Costo costante per primitiva

Limitazioni del modello locale

- Il modello di rendering locale presenta una serie di **limitazioni** nella gestione di:
  - Rifrazione e semi-trasparenza
  - Riflessione speculare
  - Ombre portate
    - Se un oggetto blocca la luce proveniente da una sorgente, gli oggetti al di là di esso restano in ombra
  - Modellazione corretta della riflessione (BRDF)
    - Se un oggetto è riflettente, la luce che si riflette da esso illumina gli altri oggetti

Mod. locale vs. globale

Modello Globale:

- Migliore simulazione della realtà ➔ maggiore veridicità delle immagini
- Maggiore comprensibilità dei rapporti spaziali tra oggetti della scena

Un esempio: ombre portate OFF/ON -- quale testa è più vicina al piano?

Mod. locale vs. globale

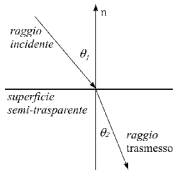
Un secondo esempio: simulazione dell'illuminazione diffusa in modo globale - milgiore percezione dei pieni e dei vuoti e della profondità relativa

Effetti globali con OpenGL?

- E' possibile sfruttare la pipeline di rendering di **OpenGL** (che adotta un modello di rendering locale) per ottenere effetti di rendering globale?
  - Si, ma non è facile (e spesso il risultato è approssimato)
  - Soprattutto occorre considerare tutta la nostra scena assieme e farla passare più volte nella pipeline di rendering
  - Vediamo alcune di queste soluzioni insieme alla descrizione delle limitazioni del modello **locale**

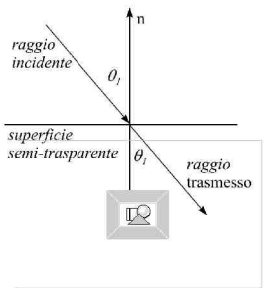
Semitrasparenza

- Tener conto solo di fenomeni locali ➔ non permette di modellare la **semitrasparenza**
- l'angolo di rifrazione caratterizza i materiali fisici (superficie di discontinuità tra materiali diversi)



Semitrasparenza in OpenGL

- Se ci si limita a situazioni in cui l'angolo di rifrazione sia zero, si può simulare tramite **alpha-compositing**



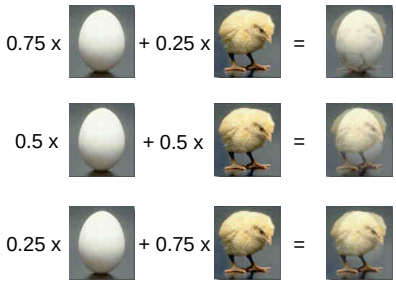
Semitrasparenza in OpenGL

Un vecchio trucco di imaging 2D:

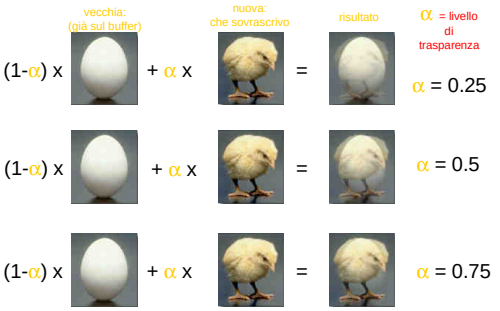
- Quando scrivo un colore nello screen buffer,
  invece di:
  - sovrascrivere il colore vecchio con quello nuovo

  eseguo:
  - comporre il colore vecchio con quello nuovo (sovrascrivo il colore vecchio con una interpolazione fra quello vecchio e quello nuovo)

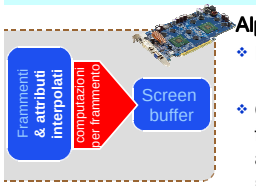
Semitrasparenza in OpenGL



Semitrasparenza in OpenGL



Semitrasparenza in OpenGL



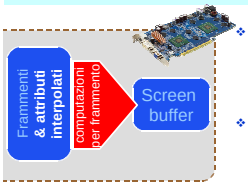
Alpha Blending

- I colori hanno 4 componenti:
  R,G,B, **α**
- Quando arriva un frammento (che sopravvive al depth test)
  invece di sovrascriverlo uso la formula

$$(r,g,b)_{finale} = (r,g,b)_{vecchio} \cdot (1-\alpha) + (r,g,b)_{nuovo} \cdot (\alpha)$$

"alpha blending"

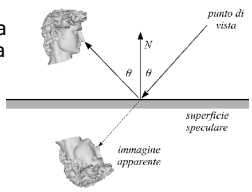
Semitrasparenza in OpenGL



- Comodo avere una la trasparenza come un canale del colore!
- Problema: Alpha blending e z-buffer
  - Per oggetti semi-trasparenti, l'ordine di rendering torna ad essere determinante
  - Necessario un passo di ordinamento in profondità degli elementi della scena (sorting da effettuare per ogni diverso viewpoint) ➔ overhead che riduce frame rate

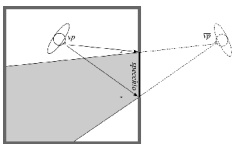
Riflessione speculare

- Tener conto solo di fenomeni locali ➔ non permette di modellare la **riflessione speculare** tra gli oggetti
- Ad esempio la simulazione della riflessione su uno **specchio**, cioè su una superficie la cui componente diffusiva è pressoché nulla



Riflessione speculare in OpenGL

- La presenza di una superficie a specchio in una scena può essere simulata in OpenGL applicando alla superficie speculare una tessitura opportunamente calcolata
- Va calcolato da quale direzione si debba riprendere l'immagine che va usata come texture map

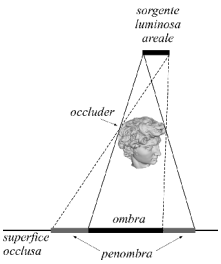


Calcolo delle ombre portate

- Tener conto solo di fenomeni locali ➔ non permette di modellare le **ombre** portate:
  - individuazione delle parti della superficie di oggetti della scena che non ricevono radiazione luminosa da una o più sorgenti
  - ombre portate: frutto della interrelazione tra tre entità:
    - *sorgente luminosa*
    - oggetto *occludente*
    - oggetto *ricevente*
  - necessario individuarle per una resa corretta dell'illuminazione

Calcolo delle ombre portate

- *Forma* delle ombre:
  - Sorgenti puntiformi ➔ ***hard shadows*** *(solo zona di ombra netta)*
  - Sorgenti areali ➔ ***soft shadows*** *(presenza di zone di penombra)*



Ombre portate in OpenGL

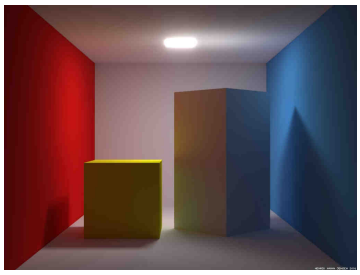
- Per ogni sorgente luminosa, si può effettuare un rendering per valutare quali siano gli oggetti della scena (o le porzioni degli stessi) visibili dalla sorgente ➔ porzioni non in ombra
- Se la scena è **statica** (posizione degli oggetti fissa nel tempo) le zone non in ombra possono essere calcolate in un passo di preprocessing
- Questa informazione può essere collegata alle geometrie 3D (es. via texture mapping)

Inter-riflessione diffusiva

- Tenendo conto solo di fenomeni locali ➔ non permette di modellare la **riflessione diffusiva** tra gli oggetti nella scena (inter-riflessione)
- Gli oggetti riflettono mutuamente radiazione luminosa, con spetro di colore dipendente dalle caratteristiche delle superfici
- Colore dipende dal:
  - Colore della radiazione emessa dalle sorgenti luminose nella scena (illuminazione diretta)
  - Colore della radiazione riflessa dagli altri oggetti (illuminazione indiretta)

Inter-riflessione diffusiva

- Un esempio:



Dal locale al globale

- Per risolvere i problemi di
  - punto di vista e luci nella scena
  - trasparenza
  - riflessioni speculari da altri oggetti
  - ombre
- si applica il metodo **Ray Tracing**

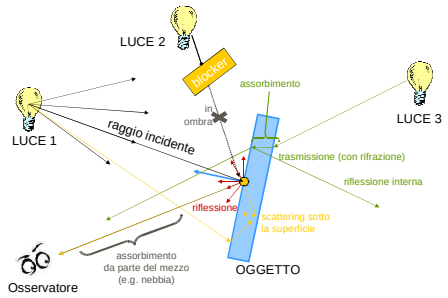
Dal locale al globale

- Il metodo **Radiosity**, invece, consente di risolvere i problemi relativi a
  - sorgenti luminose di forma qualsiasi
  - modello di illuminazione globale (che tenga conto delle inter-riflessioni diffuse tra gli oggetti)

Sezione 8.2

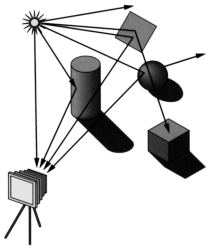
Ray-tracing

Lighting: alcuni fattori



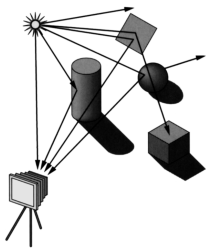
Ray Tracing

- Il metodo **Ray Tracing** è basato sull'osservazione che, di tutti i raggi luminosi che lasciano una sorgente, i soli che contribuiscono all'immagine sono quelli che raggiungono l'osservatore



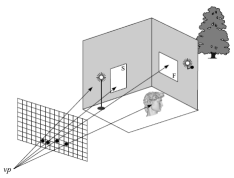
Ray Tracing

- I raggi luminosi possono raggiungere l'osservatore sia direttamente, sia per effetto delle interazioni con le altre superfici
- La maggior parte dei raggi che lasciano la sorgente non raggiungerà l'osservatore, e dunque non contribuirà all'immagine



Ray Tracing

- Mentre è molto costoso seguire la **traiettoria** di ciascun raggio, possiamo determinare i raggi che contribuiscono all'immagine se invertiamo la traiettoria dei raggi, e consideriamo solo quelli che partono dalla posizione dell'osservatore

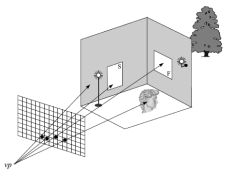


Ray Tracing

- Questa è esattamente l'idea alla base del metodo ray tracing, che simula **all'indietro** il cammino compiuto dalla radiazione luminosa per giungere all'osservatore
- Poiché si deve assegnare un colore a ciascun pixel, si deve considerare almeno un raggio luminoso per ogni pixel
- Questo raggio è detto **raggio primario**

Ray Tracing

- Ciascun raggio primario può intersecare:
  - una superficie
    - ➔ calcolo della gradazione di colore, *shading*, per il punto di intersezione
  - oppure può andare all'infinito senza intersezioni
    - ➔ assegnato un colore di sfondo
- Con i soli raggi primari ed usando un modello di shading locale ➔ si ottiene un risultato uguale al rendering in OpenGL



Ray Tracing

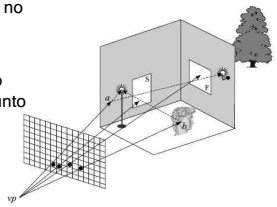
- Avendo a disposizione tutta la scena, ci possiamo *guardare intorno* per calcolare lo shading di un singolo punto intercettato da un raggio primario
- Con il ray tracing ciò si ottiene calcolando via raggi secondari:
  - Ombre portate
  - Riflessione diretta
  - Trasparenza e rifrazione

Ray Tracing e ombre portate

- Una volta determinato il punto della superficie visibile da un certo pixel,

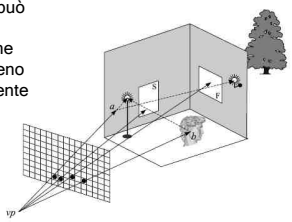
  per calcolare se il punto di intersezione è in ombra o no rispetto ad una sorgente luminosa

  si generano e si tracciano **raggi ombra**, diretti dal punto sulla superficie verso la sorgente luminosa



Ray Tracing e ombre portate

- Se un raggio ombra interseca una qualunque superficie prima di incontrare la sorgente, la luce è bloccata e non può raggiungere il punto considerato, che rimane dunque in ombra, almeno rispetto a questa sorgente

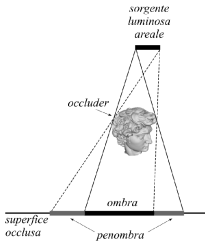


Ray Tracing e ombre portate

- Se le superfici sono tutte opache, e se non si considerano gli effetti della luce diffusa da superficie a superficie, abbiamo un'immagine che ha, oltre all'illuminazione, presenta anche delle ombre
- Il prezzo che si deve pagare per l'introduzione delle ombre è quello di dover ***tracciare,*** per ogni punto della superficie che abbiamo determinato essere visibile, **un raggio ombra per ogni sorgente luminosa** presente nella scena

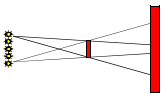
Ray Tracing e ombre portate

- Limiti
  - Ombre nette: il meccanismo del raggio ombra (come del resto anche l'equazione di shading di Phong) assume che la sorgente di luce sia puntiforme, cosa inesistente in natura (assenza di penombra)



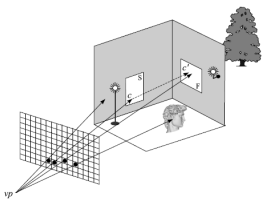
Ray Tracing e ombre portate

- Escamotage: si possono modellare sorgenti di luci ***non puntiformi*** (areali) utilizzando molte luci
- Svantaggio: il numero di raggi ombra aumenta, quindi aumenta anche il tempo di rendering



Ray Tracing e superfici riflesse

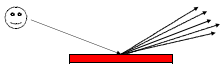
- Se alcune superfici sono invece altamente **riflessive (speculari)**, possiamo seguire il raggio primario riflesso come se fosse un nuovo raggio primario
- Il **raggio riflesso** determina il contributo dovuto alla riflessione di altri oggetti della scena sulla superficie stessa



Ray Tracing e superfici riflesse

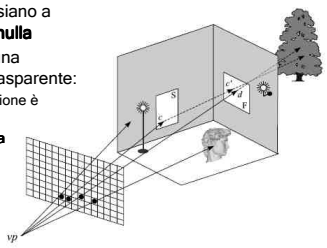
Il calcolo degli effetti di riflessione ha un costo (si traccia un altro raggio) e un limite:

- Modella accuratamene solo superfici perfettamente lisce parzialmente riflettenti. Nella realtà la maggior parte delle superfici NON sono perfettamente lisce ma riflettono in un insieme di direzioni (il riflesso è sfuocato)
- Si può modellare tracciando molti raggi riflessione ➔ overhead!



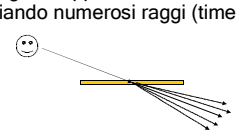
Ray Tracing e trasparenza

- Il metodo ray tracing consente inoltre di trattare in modo adeguato anche i casi in cui le superfici siano a **trasparenza non nulla**
- Al passaggio da una superficie semi-trasparente:
  - Parte della radiazione è **assorbita**
  - Parte è **trasmessa**



Ray Tracing e trasparenza

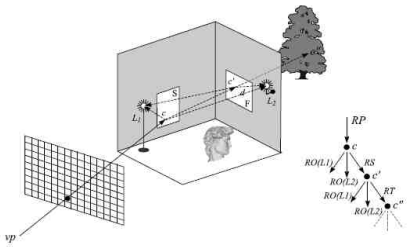
- Limiti
  - Al solito solo superfici perfettamente lisce.
  - Vetri smerigliati, appannati etc. sono simulabili solo tracciando numerosi raggi (time overhead)



  - Per modellare correttamente la rifrazione occorre costruire i nostri oggetti in maniera solida (non solo superfici giustapposte…)

Ray Tracing

- Processo ricorsivo di tracciamento dei raggi



Ray Tracing in pratica

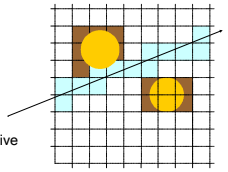
- Il nucleo di un algoritmo raytracing è il **test intersezione** segmento3D (una porzione di raggio) con una qualsiasi primitiva di modellazione (triangolo, sfera, etc.)
- Il **costo** del rendering di una scena in raytracing è soprattutto il tempo di esecuzione di un enorme numero di questi test. Dipende da:
  - **Risoluzione** dell'immagine da generare
  - Numero degli **oggetti** della scena
  - **Profondità massima** della ricorsione
- È facilmente **estendibile** al trattamento di primitive di modellazione qualsiasi (ad esempio primitive solide, sfere, coni, blocchi, etc.), basta specificare il codice per il calcolo delle intersezioni tra i raggi e la nuova primitiva

Ray Tracing in pratica

- Un'implementazione banale di un raytracer che gestisca modelli fatti solo di sfere perfette è molto semplice
- Un'implementazione molto efficiente di un raytracer è abbastanza complessa
  - La cosa più importante è l'implementazione efficiente del test intersezione raggio-primitiva
  - Numerose tecniche di speedup del calcolo dell'intersezione
  - Obiettivo: abbassare la complessità del test di intersezione a qualcosa che sia **sublineare** nella complessità della scena

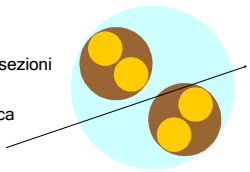
Ray Tracing - Ottimizzazioni

- Indicizzazione dello spazio della scena con una **griglia** (gerarchica o meno) che dica quali primitive sono contenute in una certa porzione dello spazio
  - Il test d'intersezione avviene controllando tutte le celle del grigliato attraversate da un dato raggio
  - Nell'ordine di tracciamento del raggio, si testano solo le primitive associate a celle intersecate dal raggio



Ray Tracing - Ottimizzazioni

- Indicizzazione delle primitive della scena mediante **gerarchie** (alberi) di **bounding volumes** (sfere o bounding box)
- Il test di intersezione viene fatto visitando l'albero dei bounding volumes, partendo dalla radice e testando i figli solo se il padre interseca il raggio
- Si evita di calcolare esplicitamente tutte le intrsezioni raggio-oggetto per cui il bounding box non interseca il raggio



Ray Tracing

- Gestisce correttamente
  - Hidden surface removal
  - Ombre portate da luci puntiformi
  - Superfici riflettenti perfette
  - Effetti rifrazione perfetta
  - Altamente estendibile
  - Facilmente parallelizzabile
  - Costo computazionale maggiore del modello locale

Ray Tracing

- Gestisce in modo approssimato (con un alto costo computazionale)
  - Sorgenti luminose non puntiformi
  - Superfici riflettenti o trasparenti non perfettamente lisce
- Non gestisce:
  - illuminazione **non diretta**, ossia quella quantità di luce che non arriva direttamente dalla sorgente luminosa ma raggiunge la superficie dopo aver rimbalzato su altre supefici

Sezione 8.3

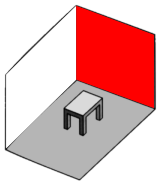
Radiosity

Radiosity

- Il metodo **radiosity** è invece concepito per la visualizzazione realistica delle superfici perfettamente diffusive
- Senza entrare nei dettagli, vediamo solo le idee di base di questa tecnica

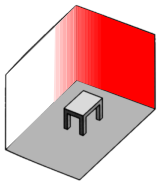
Radiosity

- Consideriamo una scena costituita semplicemente da due pareti perfettamente diffusive, una bianca ed una rossa
- Se visualizziamo la scena supponendo di avere una sorgente luminosa distante, ogni parete assumerà un colore costante



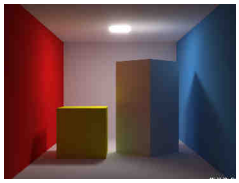
Radiosity

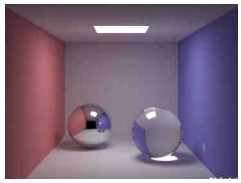
- Nella realtà, invece, la riflessione diffusa della parete rossa, colpisce la parete bianca, col risultato che della luce di colore rosso andrà ad aggiungersi alla luce bianca riflessa dalle parti di parete più vicine alla parete rossa



Radiosity

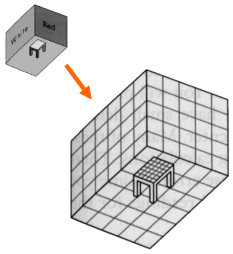
- Un paio di esempi





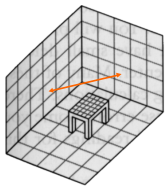
Radiosity base

- Nel metodo radiosity la scena viene suddivisa in *pezze* (*patches*), ovvero in molti poligoni piatti e di dimensioni limitate, ciascuno dei quali è considerato perfettamente diffusivo



Radiosity

- Il metodo prevede due passi per determinare le gradazioni di colore da assegnare alle varie pezze
- Il primo passo consiste nel determinare, per ogni coppia di pezze, i **fattori di forma (form factor)**, che descrivono come la luce che lascia una pezza possa influenzare l'altra



Form Factor

- In pratica i form factor definiscono quanta parte dell'energia che esce da una patch arriva su un'altra patch, tenendo in considerazione di occlusioni, orientamento delle patch, distanza etc.
- Il calcolo dei form factor delle patch di una scena sarebbe inerentemente quadratico, ma la maggior parte dei form factor sono praticamente nulli (patch lontane non si influenzano in modo percepibile)

Radiosity

- Una volta determinati i **fattori di forma**, sapendo quali **patch emettono luce (sorgenti luminose)**, la risoluzione di un **sistema di equazioni lineari** permette di capire come si distribuisce la luce all'interno della scena
- La somma delle quantità di luce (radianza) che arrivano su una patch deve essere uguale alla radianza che esce più la luce assorbita dalla patch stessa
- Il sistema di equazioni lineari e' in genere enorme
  - NxN, con N cardinalita' della scena (numero di patch)
  - ma sparso!
- Risultato finale: per ogni patch, la quantità di luce che la raggiunge (shading)

Radiosity

- Il risultato del calcolo della radiosity è relativo alla componente diffusa dello shading di una superficie, quindi è indipendente dalla posizione dell'osservatore
- Si può mostrare il risultato del calcolo interattivamente (in opengl) disabilitando il calcolo dell'illuminazione e colorando le superfici in proporzione alla radiosity calcolata ➔ rendering interattivo (se non si cambia l'illuminazione)

Radiosity

- In sintesi
  - Modella in maniera accurata la distribuzione dell'illuminazione in una scena composta solo da superfici perfettamente diffusive
  - L'accuratezza della soluzione trovata dipende da:
    - Accuratezza della suddivisione in patch della scena (complesso realizzare una suddivisione ottimale)
    - Accuratezza nel calcolo dei form factor
    - Accuratezza nella soluzione del sistema

Radiosity

- In pratica funziona molto bene, per calcolare l'illuminazione di scene architettoniche (le pareti sono diffusori quasi perfetti) e per mostrarle poi interattivamente
- Usato in molti giochi per calcolare l'illuminazione della scena una sola volta…
- La tecnica presentata è estendibile (con una differente e più complessa definizione di form factor) anche alla gestione superfici non perfettamente diffusive, non perfettamente opache ecc.

Comparazione

Solo ray-tracing

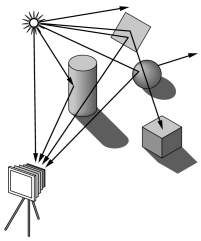
Ray-tracing & radiosity

Photon Mapping

- L'affermazione fatta durante l'introduzione del ray tracing e che è alla base del ray tracing stesso:

  non è possibile seguire il percorso di tutti i raggi di luce partendo dalla sorgente luminosa

- È in realtà stata recentemente smentita dalla tecnica detta **photon tracing (o photon mapping)**



Photon mapping

- L'idea è quella di sparare un enorme numero di particelle (**fotoni**) dalla sorgente di luce e registrare tutti i punti delle superfici che colpiscono prima di essere definitivamente assorbiti
- Si accumula per ogni porzione di superficie (ad es. patch triangolare) il numero di fotoni che la colpiscono
- La parte principale di questo approccio è la scelta delle direzioni in cui sparare le particelle che deve adattarsi alla scena in maniera tale da raccogliere informazioni accurate sull'illuminazione che raggiunge le varie superfici

Photon Mapping

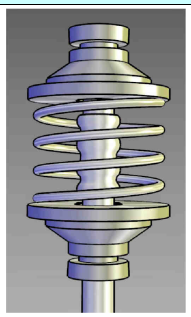
- Alla fine del processo su ogni superficie sono presenti un gran numero di fotoni (ognuno con una sua energia e direzione di provenienza) che sono utilizzati per calcolare lo shading (e non solo diffusivo) della superficie
- La qualità del risultato finale dipende da quanti fotoni, e soprattutto da come questi sono distribuiti nella scena

Sezione 8.4

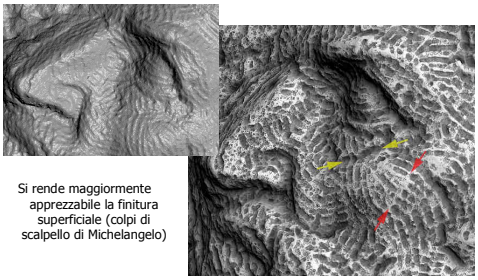
Grafica non-fotorealistica

Rendering non fotorealistico

- **NPR** = **Non Photorealistic Rendering**
- L'obiettivo non è il **realismo** ma:
  - la ***chiarezza*** della rappresentazione visiva
    - immagine che dà informazione nel modo più chiaro possibile
  - oppure l'**imitazione di uno stile** umano:
    - stile disegno a matita o chiaroscuro
    - stile pittorico
    - stile toon (cartone animato)
    - Etc.



Rendering non fotorealistico

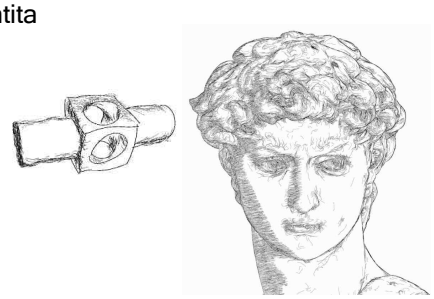


Si rende maggiormente apprezzabile la finitura superficiale (colpi di scalpello di Michelangelo)

[Images by M. Levoy]

Rendering non fotorealistico

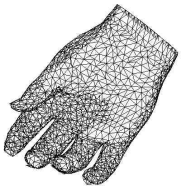
- Esempio: rendering in stile disegni a matita



Rendering non fotorealistico

- Esempio: tratteggio

modello 3D



rendering (in tempo reale)

Emil Praun, Hugues Hoppe, Matthew Webb, Adam Finkelstein 2001

Rendering non fotorealistico

- Esempio: puntinatura

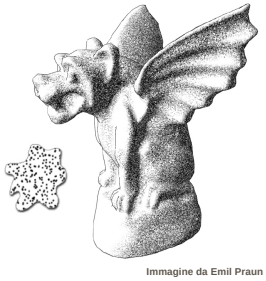
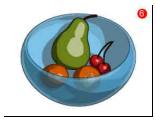


Immagine da Emil Praun

Rendering non fotorealistico

- Esempio: renderings stile toon



Esistono algoritmi via HW grafico per ottenere questo tipo di effetti in **tempo reale** !

bordi resi più marcati da linee scure

shading quantizzato (due sole gradazioni di verde, non sfumature via shading interpolato)