Grafica al calcolatore

-

Computer Graphics

10 – Tecniche per fotorealismo

Introduzione

- La rasterization pipeline impiega modelli di illuminazione locali (Phong) ed è il metodo di rendering più usato in applicazioni real-time in quanto semplice ed oramai ottimizzato anche dal punto di vista hardware.
- La resa grafica delle rasterization può essere molto buona, soprattutto in congiunzione con il texture mapping.
- Però alcune proprietà tipiche dell'interazione globale della luce in una scena sono fondamentali per dare un senso di realtà (o fotorealismo)
  - Ombre, illuminazione indiretta (radiosity), riflessioni speculari (ray-tracing) trasperenza e rifrazione (ray-tracing)
- In questo capitolo passeremo in rassegna una serie di "trucchi" di natura locale (implementabili nella rasterization pipeline) che simulano gli effetti dell'interazione globale della luce menzionati sopra ed aumentano il fotorealismo del risultato.

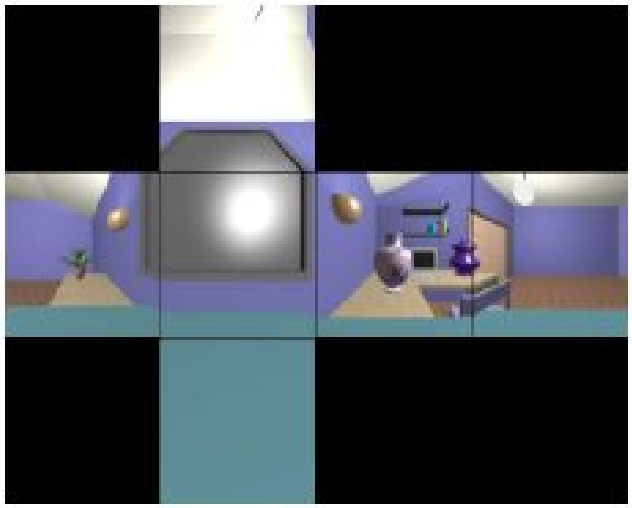
Environment map

- Gli algoritmi di illuminazione locali non possono creare effetti di riflessione, come fanno invece alcuni algoritmi globali (ad esempio il ray-tracing).
- Le reflection map (Blinn 1976) o environment map sono un trucco per rendere l'effetto di una superficie speculare che riflette la scena circostante, usando un modello di illuminazione locale e texture mapping.
- Dato un oggetto compatto (e relativamente piccolo) con una superficie lucida riflettente (come la teiera metallica), lo si racchiude in un cubo ideale e si ottengono sei immagini corrispondenti a sei telecamere poste nel centro dell'oggetto e con piano immagine coincidente con le facce dei cubi
- Le immagini così ottenute si compongono in una texture map che prende il nome di environment map
- Il rendering viene effettuato con O-mapping della environment map cubica usando il vettore di riflessione $\mathbf{r}_v$.

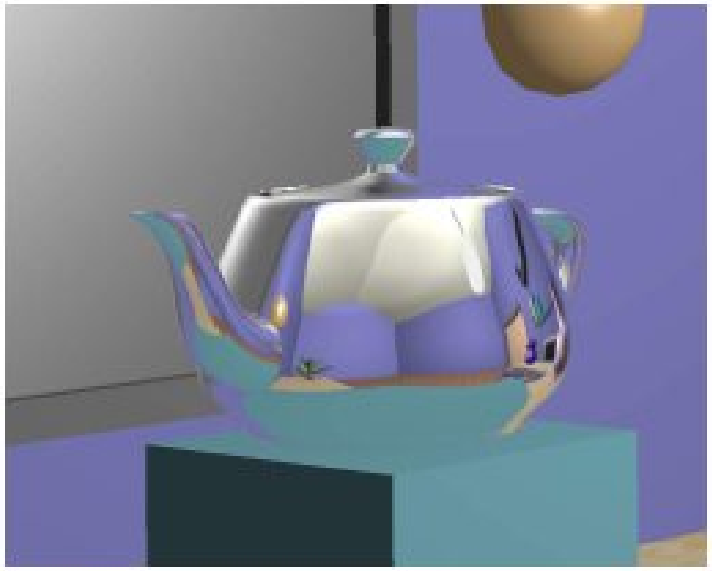
- $\mathbf{r}_v$ è il versore della direzione di vista v riflesso rispetto alla normale, e rappresenta la direzione con cui deve incidere un raggio di luce sulla superficie per essere riflesso specularmente lungo la direzione di vista:

$$\mathbf{r}_v = 2\mathbf{n}(\mathbf{n} \cdot \mathbf{v}) - \mathbf{v}$$

- Per assegnare la coordinata texture ad un punto della superficie si prosegue nella direzione di $\mathbf{r}_v$ fino ad incontrare un punto del cubo.
- L'oggetto sembrerà riflettere l'ambiente circostante



(1) Envmap



(2) Immagine

- La environment map pu`o essere generata togliendo l'oggetto e prendendo sei viste della scena, oppure può consistere di fotografie di una scena reale. In tal caso serve ad immergere realisticamente un oggetto sintetico riflettente in una scena reale.

(3) Environment (4) Immagine © Alan Watt

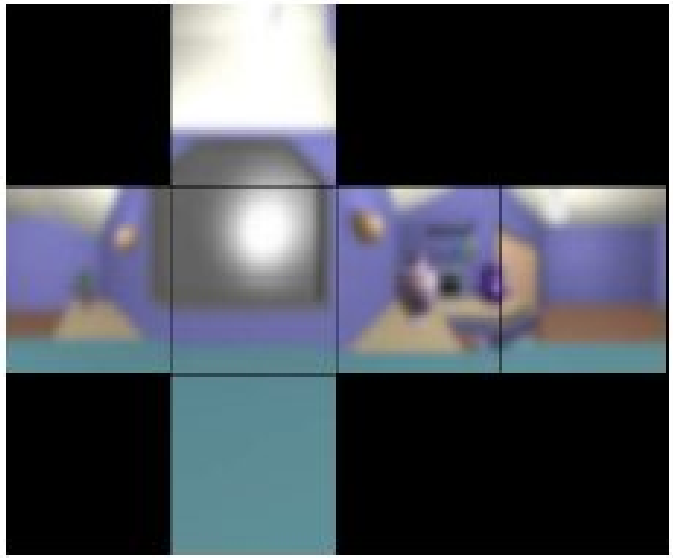
- Da notare che comunque questo tipo di tecnica è un trucco; per alcune tipologie di oggetti o situazioni particolari ci si può facilmente accorgere che la riflessione dell'oggetto non è realistica (es. non ci sono auto-riflessioni per oggetti non convessi).

- La tecnica qui delineata (mappatura cubica) è una delle possibili; un'altra fa uso di una mappatura sferica, ovvero l'oggetto viene racchiuso in una sfera e la mappa è una immagine di forma circolare che contiene una veduta deformata dell'ambiente.
- E' la stessa immagine che si ottiene dalla proiezione ortografica di una sfera perfettamente riflettente
- OpenGL supporta la mappatura sferica.

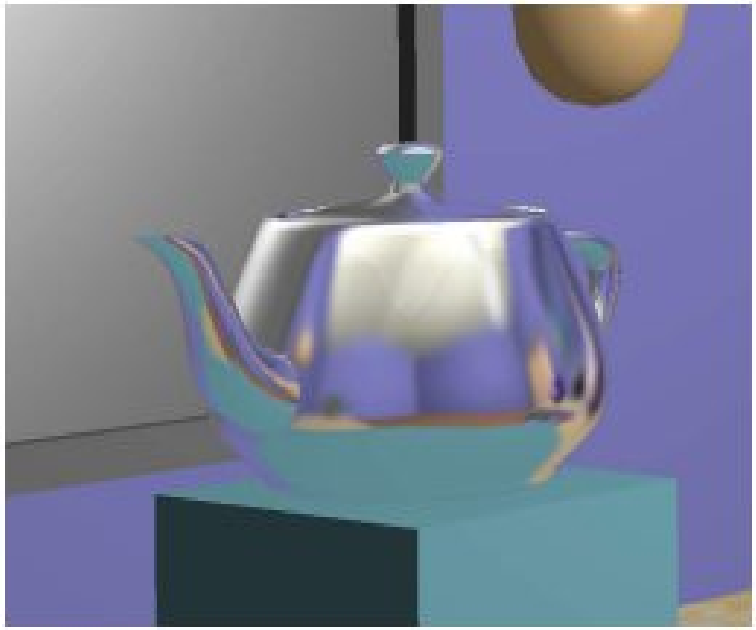
(5) Mappa sferica

(6) Immagine

- Si può creare un effetto lucido non perfettamente speculare sfocando (blurring) l'environment map.
- Accenniamo infine alla tecnica del chrome mapping, che prevede di usare una environment map fatta di chiazze di luce molto sfocate (che non c'entra nulla con la scena) per creare l'effetto di una superficie cromata.



(7) Blurred Envmap



(8) Immagine

Image-based lighting

- Si tratta di una estensione del reflection mapping.
- Serve ad illuminare in modo realistico un oggetto sintetico.
- Si memorizza in una immagine, chiamata light probe, il valore di illuminazione (radianza) lungo ogni direzione attorno ad un punto.
- Il light probe viene acquisito dal vero, tramite fotografie.
- Si usa poi questa nella soluzione della equazione della radianza. In particolare, si elimina la ricorsione nelle valutazione dell'integrale, poiché i valori di radianza lungo ogni direzione incidente sono predefiniti.

Image-based lightning

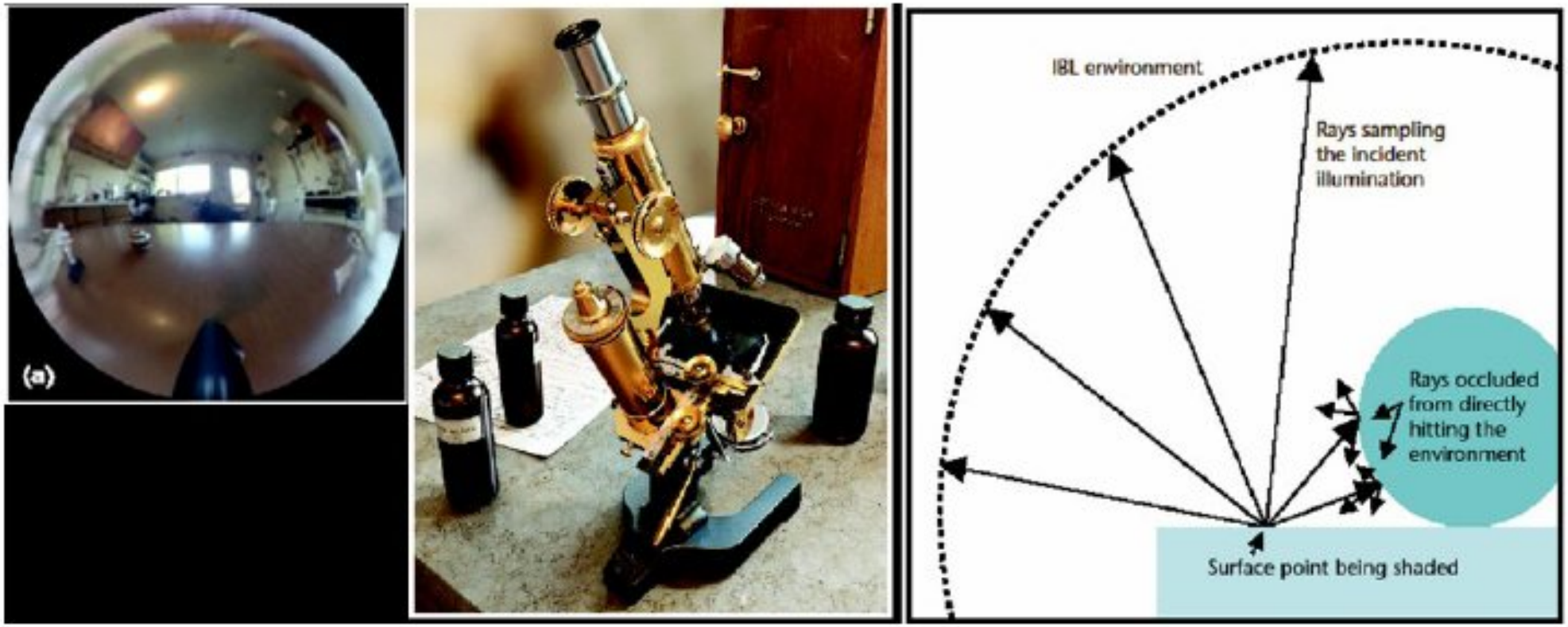


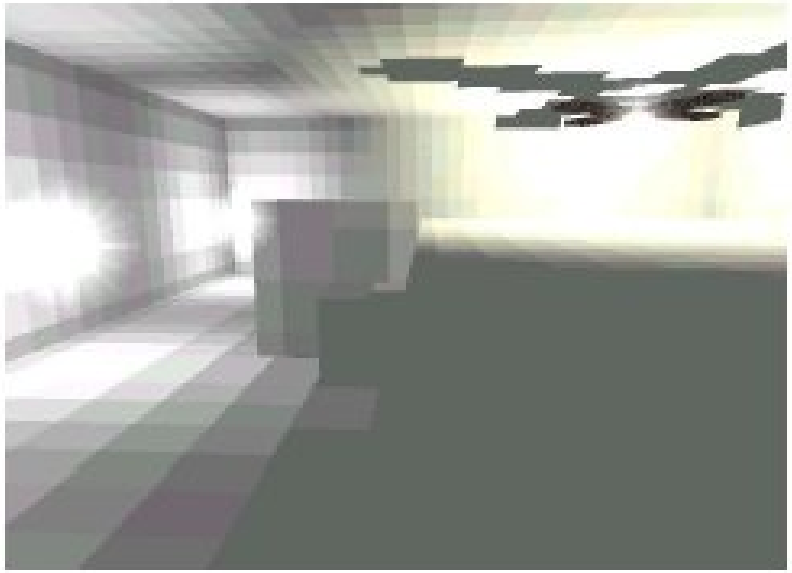
Figura 1: Light probe acquisito in una cucina, modello sintetico di un microscopio illuminato con il light probe, schema del IBL (da Debevec (2002))

Light map

- Si tratta, essenzialmente, di calcolare, off-line l'illuminazione in ogni punto delle superfici che compongono la scena (senza texture).
- Si usa solitamente una soluzione view independent della equazione della radianza, come radiosity.
- I valori di illuminazione vengono salvati in una immagine chiamata light map (tipicamente a livelli di grigio).
- In fase di rendering (on line) si aggiunge la texture con modulazione, ovvero moltiplicando lightmap e texture.

Light map

- Il vantaggio è che questa tecnica produce uno shading di migliore qualità rispetto a Gouraud ed è più veloce.
- Però funziona solo per elementi statici; elementi in movimento vanno trattati con altri metodi e le due cose vanno integrate in modo opportuno.
- La light map può avere anche una risoluzione molto inferiore a quella dell'immagine, poi viene filtrata prima dell'applicazione.
- La tecnica è largamente usata (specialmente nell'industria dei videogiochi).



(1) Solo Light map

(2) Con Light map filtrata

(3) Solo texture

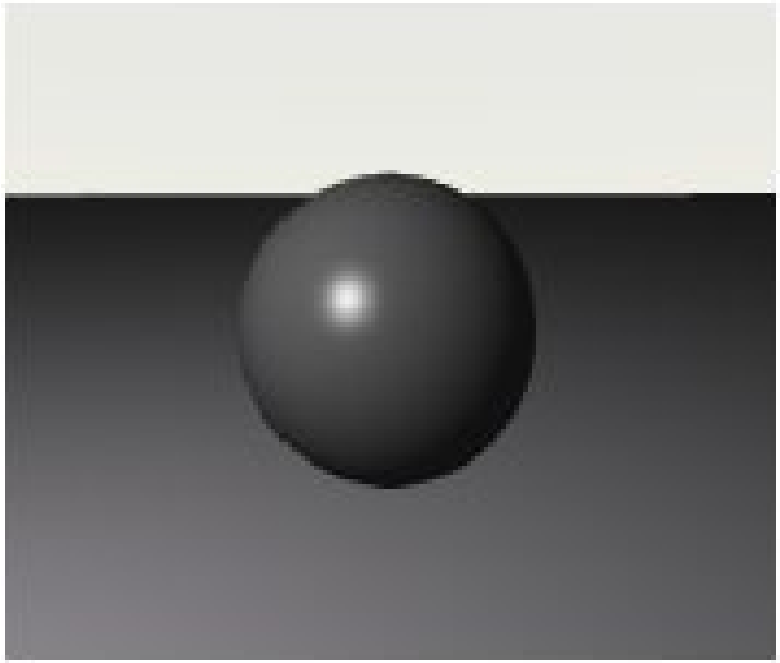
(4) Finale

Ombre geometriche

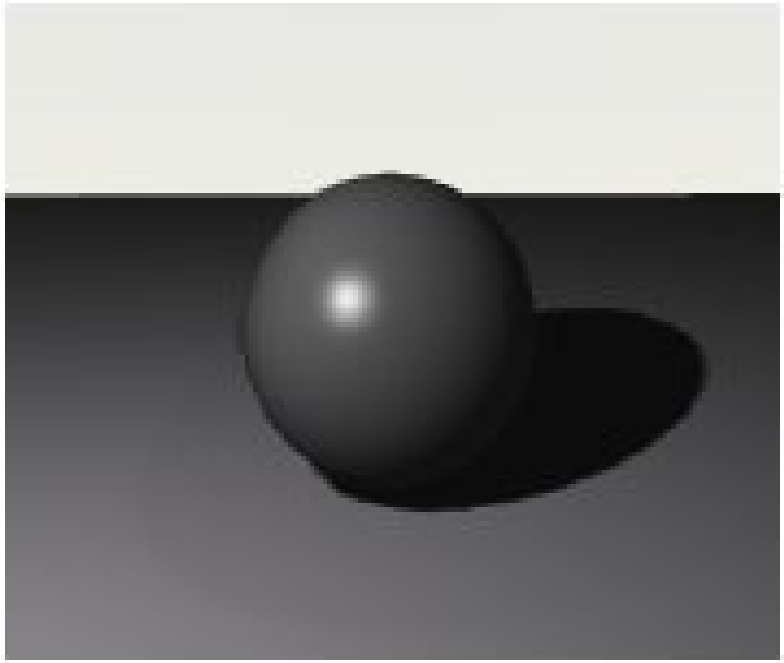
- Un particolare che si nota immediatemente nelle immagini ottenute con modelli di illuminazione locale, è l'assenza di ombre proiettate, che sono fondamentali per dare profondità e realismo all'immagine.
- Non si deve confondere il chiaroscuro (o shading), che può essere ottenuto con algoritmi locali, con l'ombra proiettata (cast shadow), che non è locale di natura.
- Vi sono svariate tecniche per introdurre le ombre in algoritmi locali; ne vediamo brevemente due (le più semplici, non le più usate)
- Queste due tecniche calcolano le ombre geometriche, perché calcolano la formadell'ombra ma non l'illuminazione al suo interno

Ombre geometriche

- Differenza tra shading e ombra



(5) Solo chiarosuro

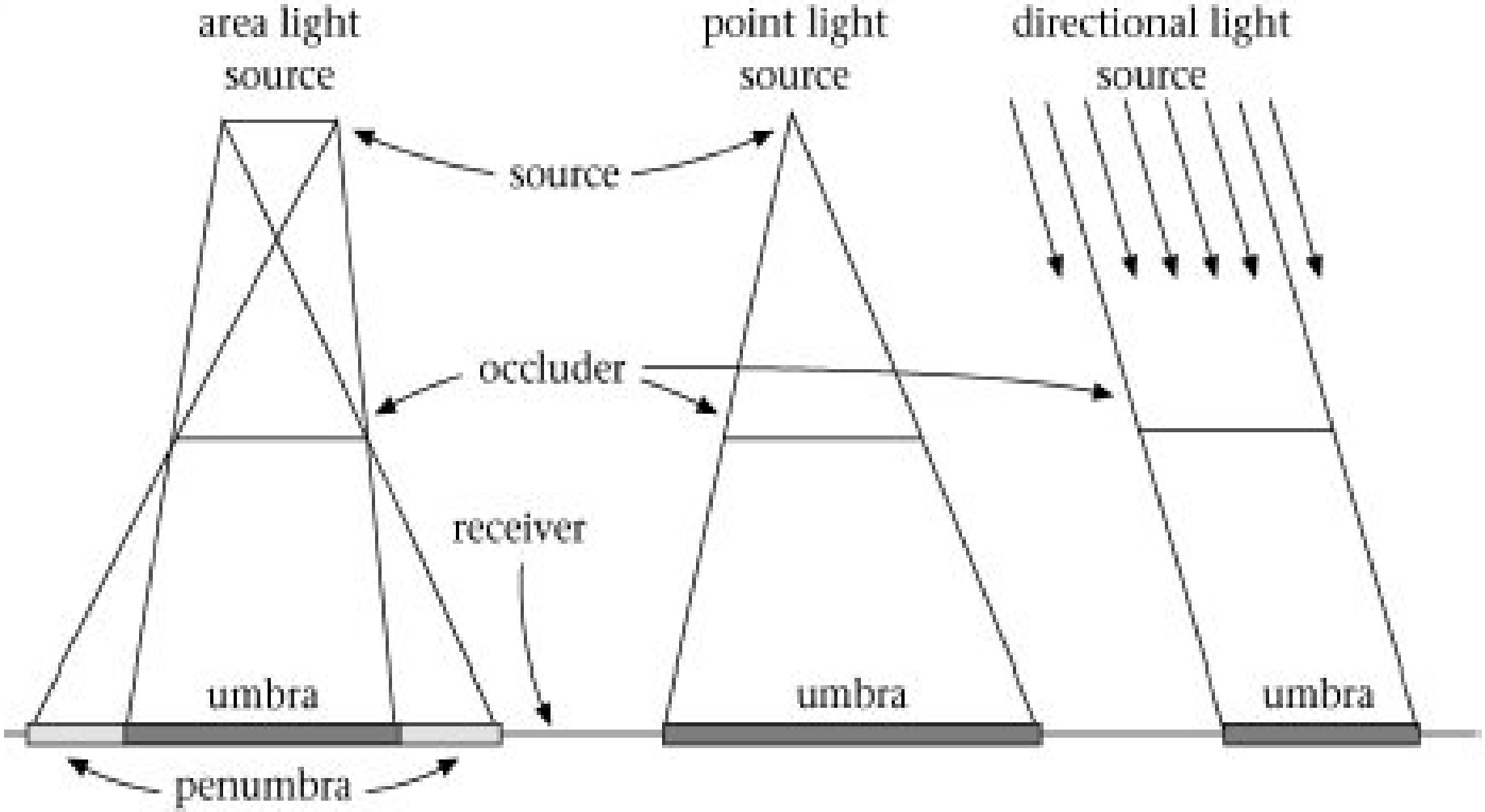


(6) Con ombra

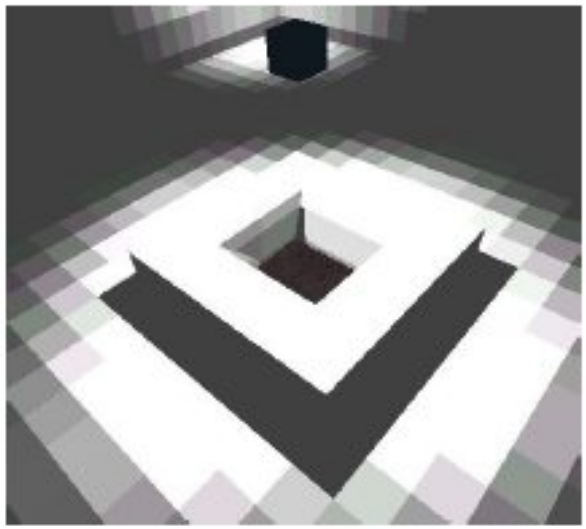
Proprietà delle ombre proiettate

- L'ombra che il poligono A getta sul poligono B a causa di una sorgente luminosa puntiforme si può calcolare proiettando il poligono A sul piano che contiene il poligono B con centro di proiezione fissato in coincidenza della sorgente.
- Non si vede alcuna ombra se la sorgente luminosa coincide con il punto di vista. Ovvero, le ombre sono zone nascoste alla luce. Questo implica che si possono usare tecniche (modificate) di rimozione di superfici nascoste per calcolare le ombre.
- Per scene statiche le ombre sono fisse. Non dipendono dalla posizione dell'osservatore
- Se la sorgente (o le sorgenti) è puntiforme l'ombra ha un bordo netto, ovvero non c'è penombra (come nello spazio).

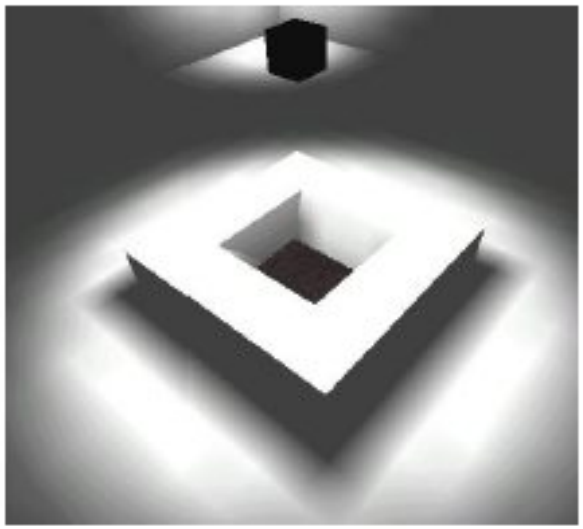
Proprietà delle ombre proiettate



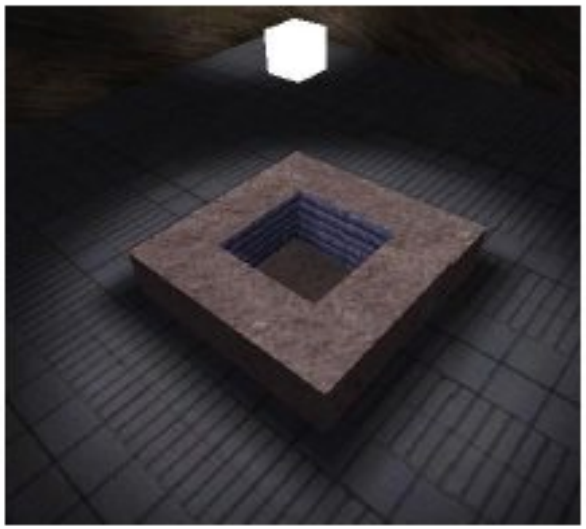
- Le ombre (geometriche) possono essere precalcolate per oggetti statici, ed aggiunte alla lightmap (molto usato nei videogiochi).



(7) Lightmap + geomeric shadows



(8) Filtrata

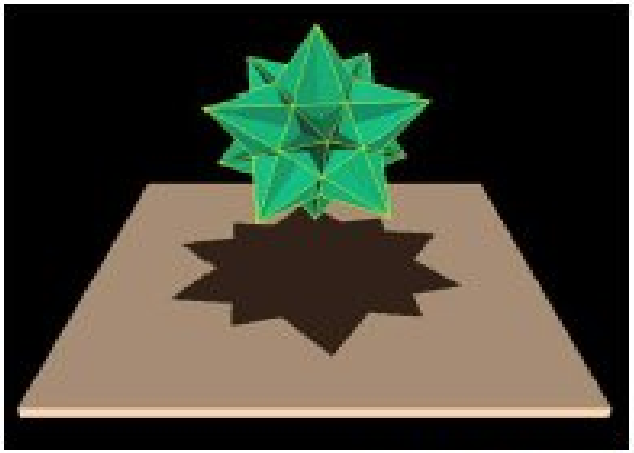


(9) Applicata all'oggetto con texture

© Alan Watt

Ombra sul piano (fake shadows)

- Si tratta di una tecnica per calcolare l'ombra portata da un oggetto su un piano π (di solito il terreno) a causa di una luce posizionata in $\mathbf{P}_l$.
- L'idea è di disegnare l'ombra come un oggetto piatto –(tipicamente nero) sul piano.
- Per disegnare l'ombra si effettua il rendering della scena, con colore nero, con una telecamera centrata in $\mathbf{P}_l$ e che ha il piano π come piano immagine.
- Se la luce è direzionale (distant light) si una una matrice di proiezione ortogonale
- La tecnica è semplice ed efficiente: si tratta semplicemente di fare il rendering della scena due volte.



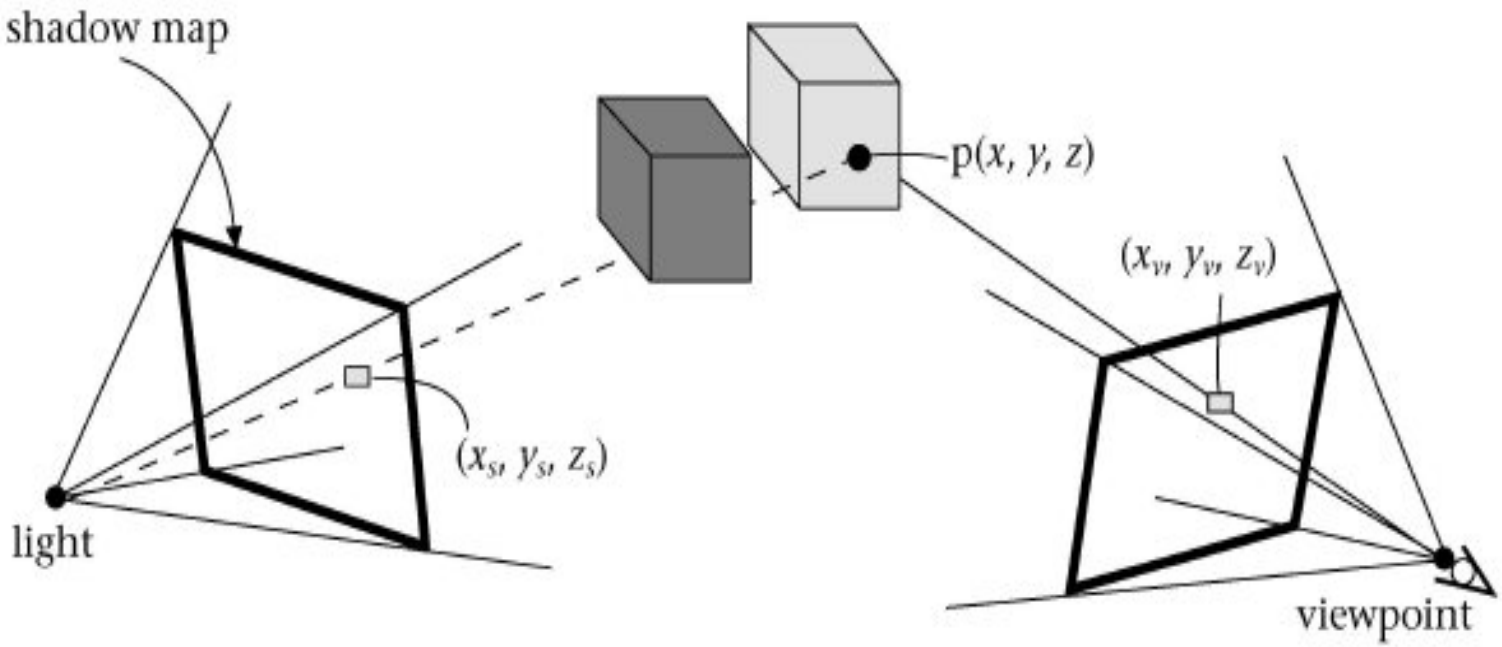
Ombra sul piano (fake shadows)

- Il metodo gestisce oggetti arbitrariamente complicati, ma la scena è vincolata ad essere molto semplice: un solo oggetto oppure più oggetti sufficientemente distanti da non farsi ombra.
- Calcolare l'ombra di un oggetto su un altro oggetto con questo metodo è fattibile, ma è più complicato
- I triangoli d'ombra giacciono sullo stesso piano del piano π visto sopra; ci possono essere problemi di risoluzione dello z-buffer per cui nel rendering alcuni di questi triangoli (o una loro parte) possono finire sotto il piano (z-fighting) .

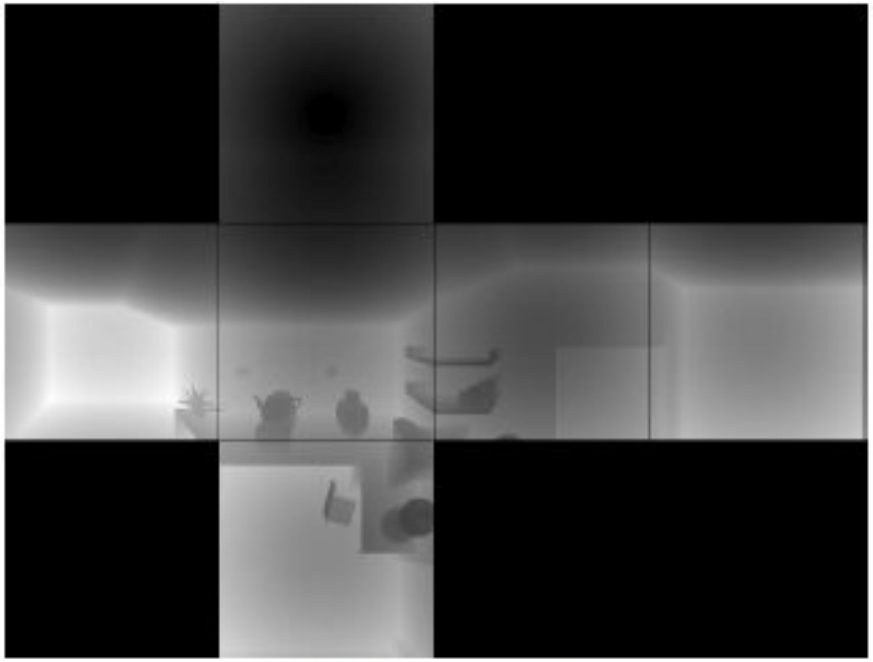
Shadow buffer

- Questa tecnica si rifà alla seconda osservazione sulle proprietà delle ombre, ovvero si basa su un algoritmo di rimozione delle superfici nascoste.
- Si calcola uno z-buffer dal punto di vista della luce (detto anche shadow buffer o shadow map):
  - Se la luce è spot-light allora basta calcolare uno z-buffer singolo
  - Se la luce è una point-light, la si immagina racchiusa in un cubo e si calcolano sei z-buffer, uno per ogni faccia del cubo.
- In fase di rendering, se un punto di un poligono deve essere disegnato, lo si trasforma nel sistema di riferimento della luce e si trova il valore z_l nello shadow buffer

- Se lo z del punto (nel riferimento della luce, non della camera) è più grande di z_l, significa che vi è un oggetto che blocca la luce per quel punto, quindi è in ombra e lo si può colorare di conseguenza. Altrimenti è colpito dalla luce e si opera lo shading normalmente
- Se si hanno più luci bisognerà avere uno shadow buffer per ognuna.
- La tecnica eredita le inefficienze dello z-buffer: richiede molta memoria e compie calcoli che poi possono venire sovrascritti.



Shadow buffer



(10) Shadow buffers

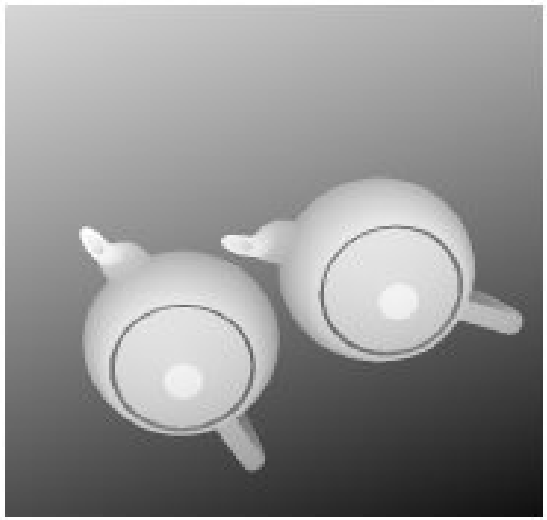
(11) Image

Esempio: environment map e shadow buffer

(12) Scena prima

(13) Scena dopo

(14) Envmap

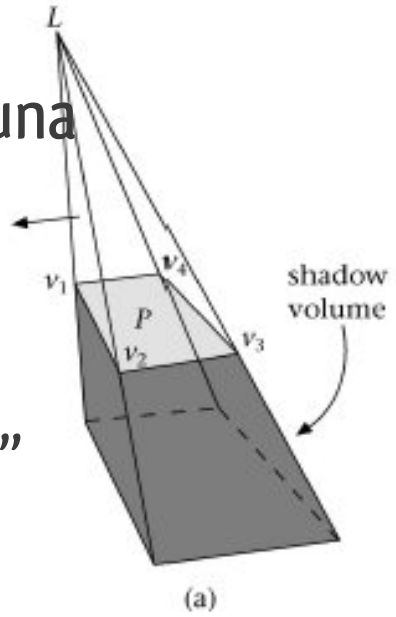


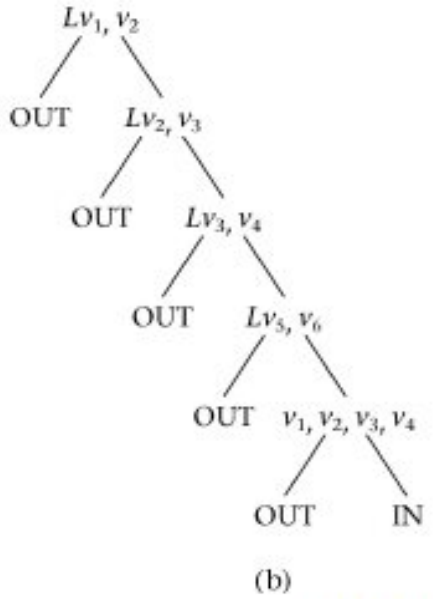
(15) Shadow buffer

Shadow volume

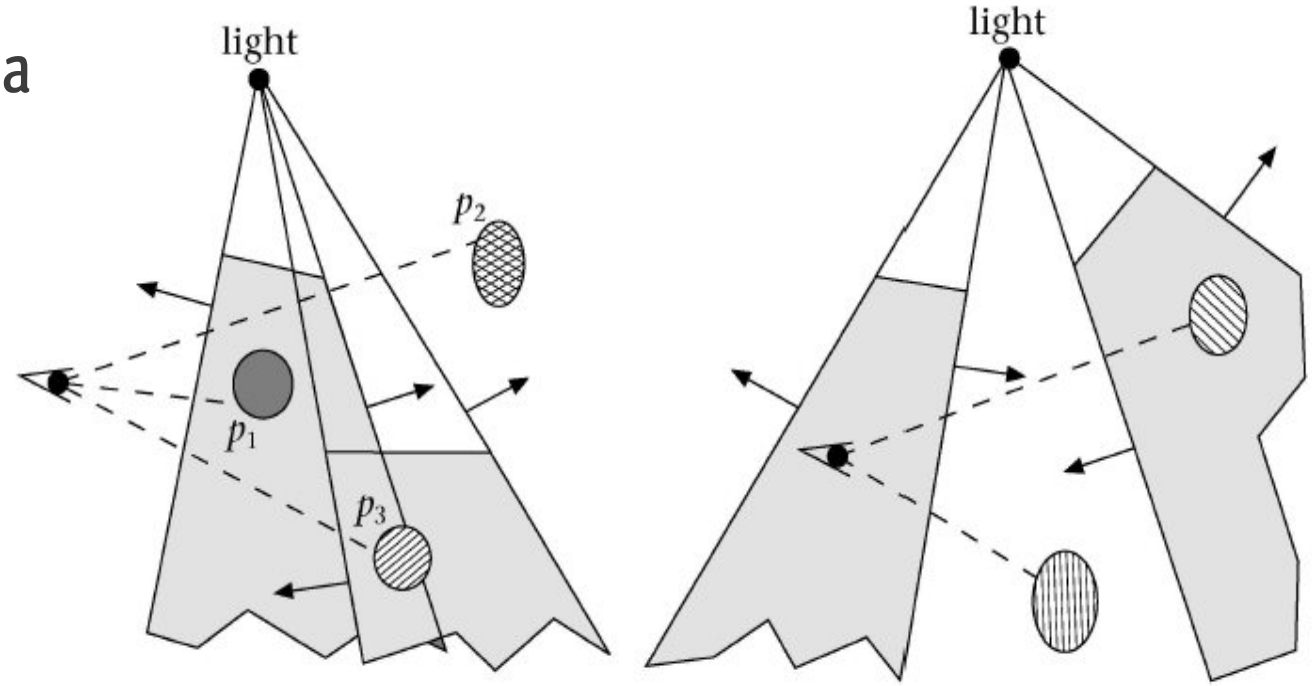
- Metodo introdotto da Crow (1977) e Bergeron (1986).
- Il volume d'ombra (shadow volume) di un poligono rispetto ad una sorgente luminosa puntiforme è la parte di spazio che il poligono occlude alla vista della luce

- E' un tronco di piramide semi-infinita (senza base) che ha il vertice nella sorgente luminosa ed è limitata da una parte dal poligono stesso.
- Le facce della piramide prendono il nome di shadow planes. Vengono rappresentate in modo che il "fronte" sia verso la luce ed il "dietro" verso l'ombra.



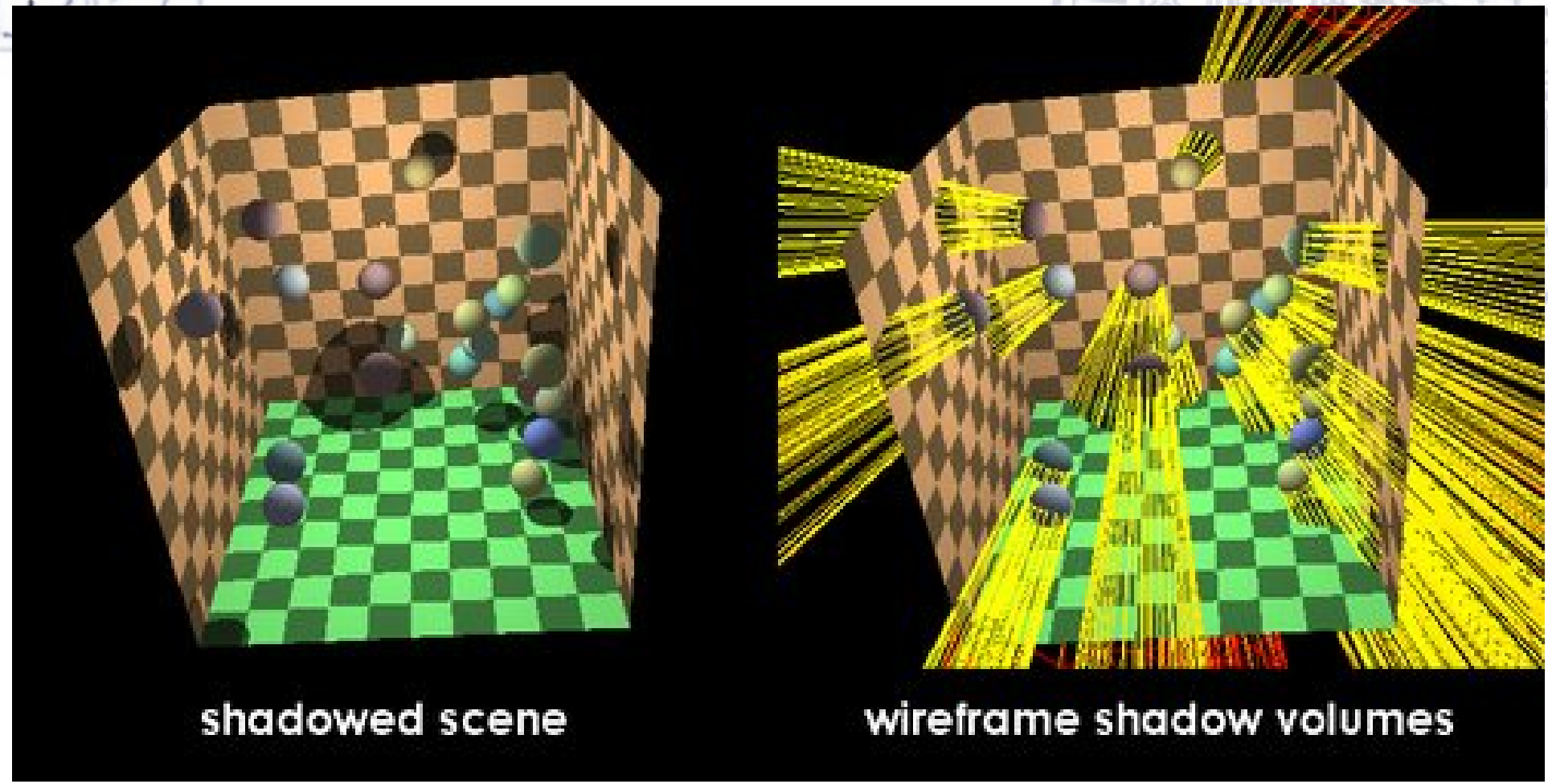


- Dato un punto di vista non in ombra, e dato un punto P della scena, tracciamo il segmento di retta che congiunge il punto di vista e P e contiamo gli attraversamenti degli shadow planes, incrementando un contatore quando l'attraversamento è in entrata (dal fronte) e decrementandolo quando è in uscita (dal retro). Se la differenza è zero il punto è nella luce, altrimenti è nell'ombra (di uno o più poligoni).
- Se il punto di vista è nell'ombra bisogna partire dal numero di shadow volumes nei quali è contenuto. Si calcola contando le intersezioni con gli shadow planes lungo una semiretta arbitraria con origine nel punto di vista.



Shadow volume

- In pratica: la generazione degli shadow planes avviene off-line, vengono aggiunti alla scena come poligoni (vengono tagliati ad una certa distanza dalla luce) e vengono processati come tutti gli altri poligoni, eccetto che sono invisibili. Durante la scan conversion (con un algoritmo scan-line), quando viene incontrato un tale poligono invece che colorare il pixel corrispondente, si incrementa/decrementa un contatore per pixel. Il risultato è una mappa che per ogni pixel dice se è in luce (= 0) oppure in ombra (> 0)
- In OpenGL si fa in tre passate, con un trucco che usa z-buffer e stencil buffer.
- Modifiche:
  - calcolare le silhouette delgli oggetti ed usarle per gli shadow volume (invece dei singoli poligoni).
  - creare gli shadow volumes solo degli oggetti “importanti”.



From wikipedia

Trasparenza

- Si deve inserire nelle rappresentazioni del colore. Queste sono notoriamente:
  - Indicizzato: viene costruita una tabella di colori predefiniti (la cosiddetta palette); ad ogni pixel si può quindi associare semplicemente un numero intero che indica una posizione nella palette. In genere la dimensione della palette è di 256 elementi a cui corrisponde un frame-buffer a 8-bit di profondità di colore.
  - True color: viene associato ad ogni pixel direttamente un colore parametrizzato dalla sue componenti RGB. In genere ciascuna componente può assumere 256 valori, da 0 a 255; si ha quindi un frame-buffer a 24-bit di profondità di colore.

Colore e trasparenza

- La rappresentazione con RGB non è l'unica parametrizzazione di un colore; è quella usata dalle OpenGL, quindi non ne vediamo altre.
- Nel caso indicizzato potrei associare un valore alla trasparenza.
- Quello che si fa nella rappresentazione RGB è associare una quarta componente ad ogni pixel, detta α. Si ha a che fare in tal caso con un frame-buffer RGBα a 32-bit.Questa può codificare opacità/trasparenza

Trasparenza

- La componente α della rappresentazione RGBα può essere usata per per specificare un valore di opacità k_o compreso tra zero ed uno.
- L'opacità viene specificata tramite una texture RGBα.
- L'opacità di una superficie è una misura di quanto la luce riesce a penetrarla.
- Se k_o è pari a uno, allora il punto è completamente opaco e si effettua il rendering normalmente.
- Per valori di k_o diversi da uno la superficie in quel punto è trasparente o traslucida e bisogna miscelare il colore calcolato dal modello di illuminazione con il colore del punto che sta otticamente "dietro"

Trasparenza

- In una scena con poligoni opachi e trasparenti, ogni poligono che sta dietro ad uno opaco non deve essere disegnato (principio su cui si basa la rimozione delle superfici nascoste), ma un poligono trasparente davanti ad un'altro poligono (trasparente o opaco) deve essere disegnato miscelandone il colore con quello dietro.
- Riassumendo: per decidere il colore di un punto trasparente devo aver calcolato il colore del punto che sta dietro.

Trasparenza

- Si supponga che per un dato pixel si sia stabilito un colore C_b dopo aver disegnato una serie di poligoni in ordinamento del pittore (back-to-front); si supponga che disegnando un nuovo poligono la formula di shading assegni intensità C_f per quel pixel dovuta al nuovo poligono e che tale pixel risulti di opacità k_o. Allora il dato pixel assumerà un colore pari a

$$C = k_o C_f + (1 - k_o) C_b .$$

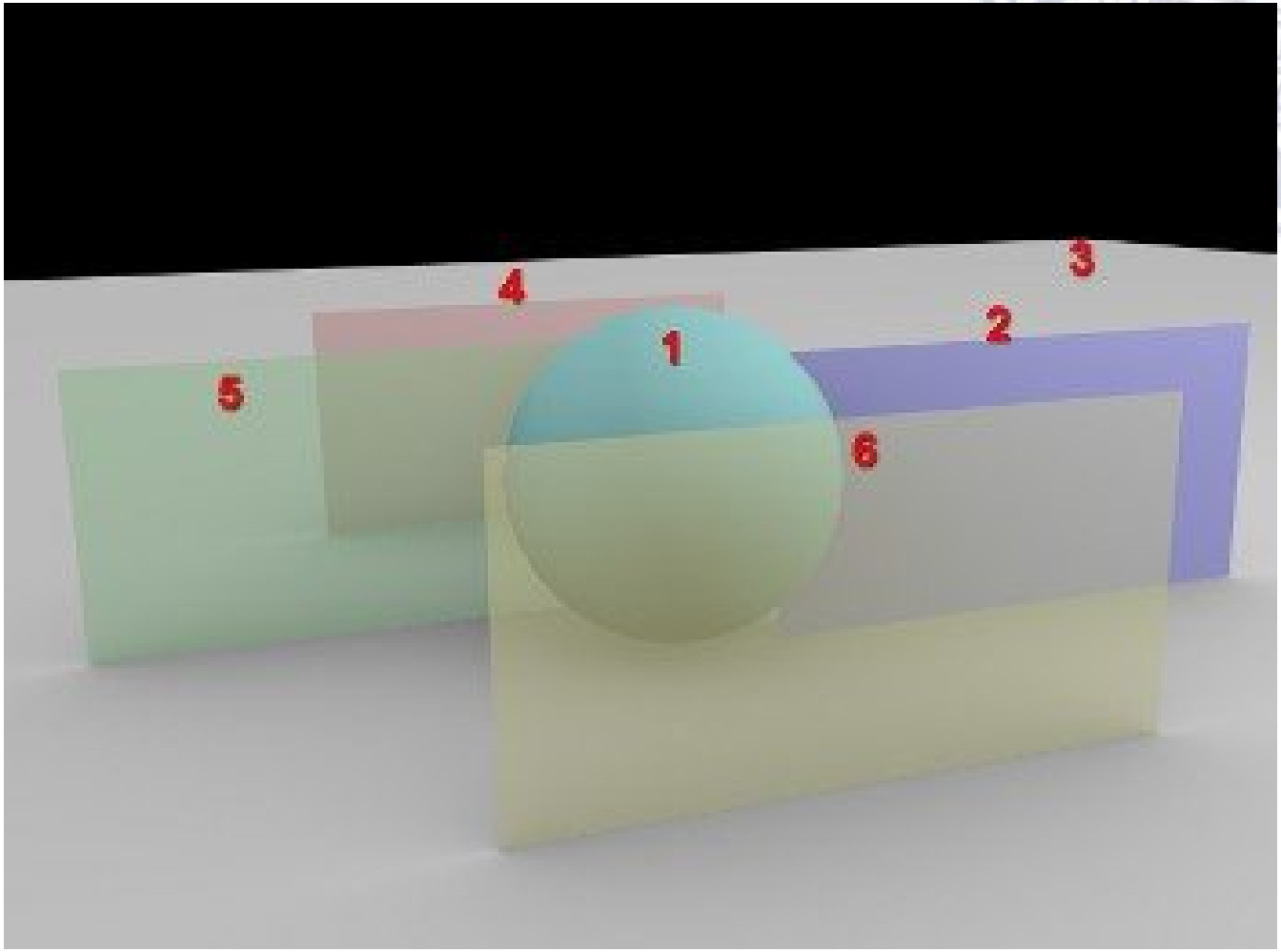
- Dunque il rendering di oggetti trasparenti è immediato da implementare in una pipeline con list-priority, che procede naturalmente back-to-front.

Trasparenza

- Con lo z-buffer è più macchinoso (lo z-buffer non procede in ordine back-to-front). Si risolve con un "trucco" in due passate: prima si fa il rendering dei poligoni opachi solamente (disabilitando quelli con $k_o < 1$), poi si rifà il rendering dei soli trasparenti, impedendo ai questi di aggiornare lo z-buffer.
- Nota: generalmente la luce che attraversa un materiale trasparente viene anche rifratta (cambia direzione). La tecnica illustrata qui non tiene conto del fenomeno, come invece fa ray tracing.

Esempio

- 1,2,3 opachi, 4,5,6 trasparenti



Riferimenti

- Angel 6th ed. p.366
- Scateni et al. (Fondamenti di grafica interattiva tridimensonale) p. 168
- Buss cap. 5
- http://en.wikipedia.org/wiki/Shadow_mapping
- http://en.wikipedia.org/wiki/Shadow_volume

Domande di verifica

- Come si può codificare la trasparenza nelle immagini digitali?
- Che cos'è una light map? A cosa serve? Quali sono i limiti nel suo uso nella grafica interattiva?