Università degli studi di Bologna
Dipartimento di Matematica

Fondamenti di Computer Graphics

Docente:
Prof. Serena Morigi

E-mail:
E-mail serena.morigi@unibo.it

A.A. 2014/2015

Indice

Chapter 1

Il sistema grafico: overview Hardware e Software

Con sistema grafico si intende un insieme di dispositivi hardware e programmi software che interagiscono per produrre immagini grafiche.

I dispositivi hardware rappresentano la potenzialità del sistema grafico: le risorse che rendono possibile produrre immagini.

Il software serve ad organizzare le risorse hardware: utilizza l'hardware per creare e manipolare immagini grafiche. Ci sono diversi livelli ai quali l'utente può interagire con un software di un sistema grafico, come mostrato in figura 1.1.

Le **schede grafiche** sono state progettate per estendere le potenzialità di elaborazione della CPU. I **sistemi operativi** come Linux, Windows, interagiscono con l'hardware grafico attraverso i **driver** (device dependent) che sono preposti a sfruttare l'accelerazione disponibile sulla scheda grafica. Un' applicazione o un 3D game farà uso di chiamate alle librerie grafiche o API le quali interagiscono con i driver per visualizzare poligoni, texture, scene 3D.

Esempi di Application Programmers Interface (API) sono OpenGL, PHIGS, DirectX, VRML, JAVA-3D, WebGL.

Lo sviluppo della Computer Graphics è parallelo a quello della tecnologia hardware; quindi per poter conoscere e sfruttare fino in fondo le sue potenzialità è necessario avere una visione chiara di come si è evoluto nel

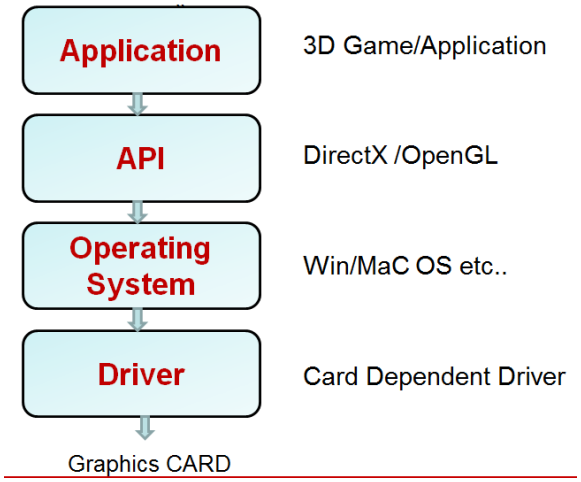


Figure 1.1: Software overview

Figure 1.2: Alcuni dispositivi di input 3D.

tempo l'hardware grafico e delle conseguenze di quest'evoluzione nella pipeline grafica.

I due aspetti peculiari dell' hardware di un sistema grafico sono i dispositivi periferici di input/output e le schede grafiche.

I principali dispositivi periferici (detti semplicemente periferiche) impiegati in computer grafica sono

- dispositivi di input grafico interattivo
  - data gloves
  - mouse
  - joystick, trackball
  - scanner 2D/3D
- Raster scan display system
- Force feedback device, VR glasses, kinect, leap

- Fotocamera/video digitale

Alcuni dispositivi periferici sono illustrati in figura 1.2.

Display, stampanti e fotocamere digitali sono di tipo raster. Esistono anche analoghe periferiche non raster (es. plotter), ma sono specializzate per particolari applicazioni.

Nelle periferiche raster la parte di visualizzazione (nel monitor e nella stampante) o di lettura (nello scanner 2D e nella camera digitale) è una matrice rettangolare, detta raster, di punti quadrati ognuno dei quali si chiama pixel (picture element). Ogni pixel può assumere uno dei diversi colori producibili o leggibili dalla periferica stessa.

La **risoluzione** di un dispositivo raster è il numero di pixel per unità di misura (pollice o centimetro). Per esempio a risoluzione 370 pixel per pollice (ppi, pixel per inch), in un pollice lineare ci sono 370 pixel. (Pollice - 1 pollice = 2.54 cm)

La **risoluzione display** è il numero totale di pixel (es.1024×768).

Le **dimensioni fisiche** di un dispositivo raster (larghezza e altezza in centimetri o pollici) dipendono dalla risoluzione:
dimensione fisica = risoluzione display/ risoluzione.

Mantenendo invariato il numero totale di pixel si possono modificare le dimensioni fisiche (vengono modificate la risoluzione e il lato del pixel, si veda figura 1.3). Modificando il numero dei pixel (downsamplig e upsampling) a parità di dimensioni fisiche vengono modificate la risoluzione e il lato del pixel.

Maggiore è il numero di pixel nello schermo e migliore è la risoluzione, l'immagine appare di conseguenza più nitida.

Prima di prendere in esame le varie periferiche esaminiamo due concetti che stanno alla base del loro funzionamento: la gestione del colore e la grafica raster/vettoriale.

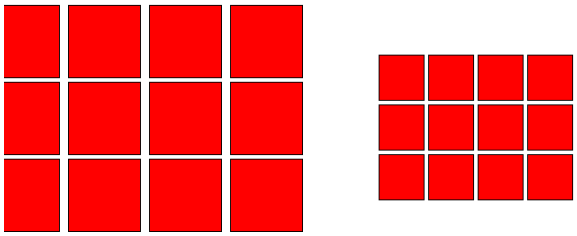


Figure 1.3: Cambiano le dimensioni fisiche (e la risoluzione) ma non la risoluzione display.

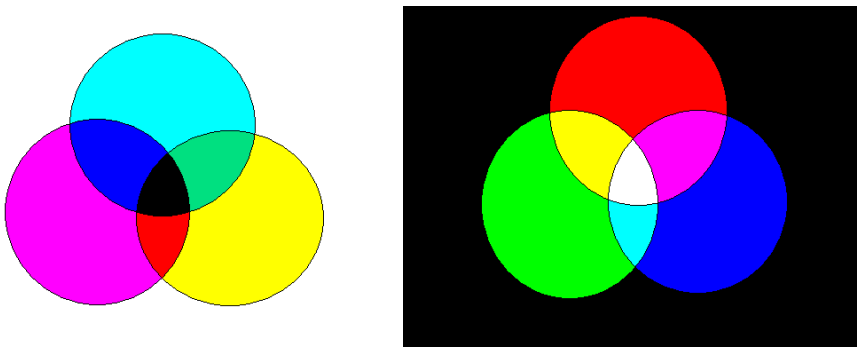


Figure 1.4: Sintesi sottrattiva (sinistra); sintesi additiva (destra).

1.1 Spazi colorimetrici

I principali modelli per la rappresentazione del colore in un'immagine digitalizzata, visualizzati in Fig. 1.4, sono i seguenti:

1. **Modello RGB (red, green, blu)**
   Sintesi additiva: il colore risultante è ottenuto sommando i contributi luminosi dei tre colori primari. (video)
2. **Modello CMY (cyan, magenta, yellow)**
   Sintesi sottrattiva: il colore risultante è ottenuto rimuovendo dalla luce bianca i contributi dei 3 colori primari (stampanti a colori). Gli oggetti appaiono colorati in un certo modo soltanto perchè le sostanze che li costituiscono assorbono dalla radiazione incidente l'energia associata a certe lunghezze d'onda.

$$\begin{pmatrix} C & M & Y \end{pmatrix}^T = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}^T - \begin{pmatrix} R & G & B \end{pmatrix}^T$$

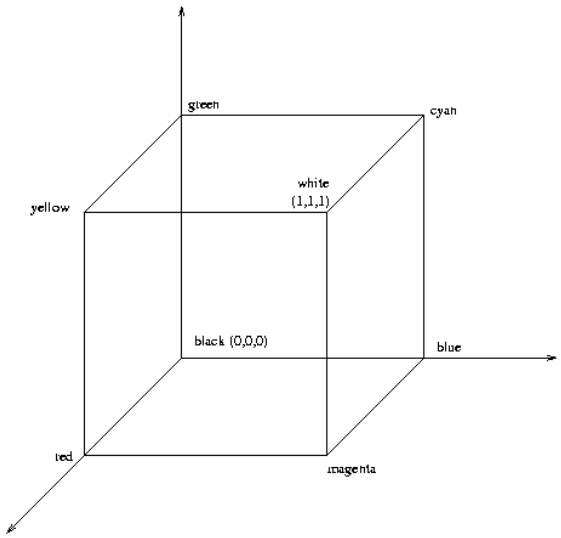


Figure 1.5: Spazio RGB

3. **Modello YIQ**
   (luminosità , Inphase, Quadrature) (TV).

Nella rappresentazione 3D del modello colorimetrico possiamo associare ad ogni asse un colore primario (red,green,blue). La rappresentazione dello spazio colorimetrico RGB è illustrata in Fig. 1.5; ogni punto (x, y, z) all'interno del cubo unitario rappresenta un determinato colore di intensità le componenti (x, y, z) rispettivamente RGB. Analogamente si può rappresentare lo spazio CMY associando alle coordinate le componenti CMY. Punti sulla diagonale principale $(x = y = z)$ rappresentano i diversi livelli di grigio.

Indipendentemente dai tre modelli considerati, il colore può essere specificato mediante il **Modello HSI (Hue, saturation, Intensity)**, dove:

- **Hue (tinta)**: tonalità , colore dominante così come viene percepito dall'osservatore;
- **Saturation:** purezza del colore o quantità di luce bianca miscelata con una certa tinta;
- **Intensity:** luminosità

Si definisce cromaticità: $\boxed{\text{Hue + saturation = cromaticità}}$.

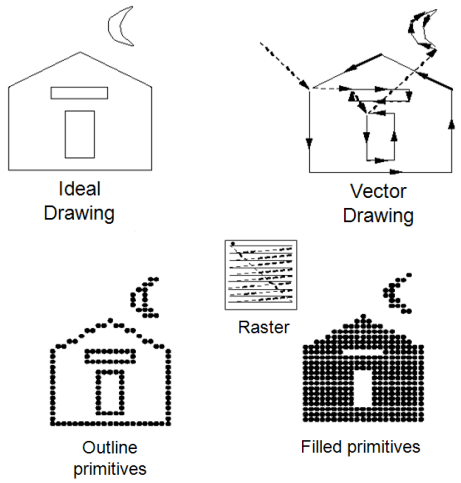


Figure 1.6: Grafica vettoriale (sopra); Grafica raster (sotto)

Se un colore viene definito in modalità HSI, esso dovrà poi essere convertito in uno dei tre modelli sopra.

1.2 Grafica raster e grafica vettoriale

Nella grafica raster, l'immagine è una griglia rettangolare (raster) di pixel colorati. Nella memoria del computer vengono conservati i singoli pixel dell'immagine bitmapped (vedi Fig.1.6,sotto).

Nella grafica ad oggetti, detta anche grafica vettoriale, un' immagine consiste di oggetti grafici (punti, linee, rettangoli, curve, e così via) ognuno dei quali è definito, nella memoria del computer, da un'equazione matematica. Ogni oggetto è indipendente e può essere spostato, modificato od eliminato senza influenzare gli altri oggetti dell'immagine (vedi Fig.1.6,sopra).

Poichè ogni oggetto è rappresentato (in memoria) da un'equazione matematica, per riprodurre l'immagine su un dispositivo raster questa va trasformata in pixel, operazione che si dice rasterizzazione. Per le immagini bitmap ovviamente non esiste il concetto di rasterizzazione, in quanto sono già per definizione un rettangolo di pixel colorati.

1.2.1 Grafica raster

La grafica a punti (grafica bitmap o raster) trova applicazione soprattutto nell'elaborazione di immagini fotografiche e nelle illustrazioni pittoriche. La loro natura a mosaico costituisce il punto di forza e anche di debolezza delle immagini bitmap.

I vantaggi di questo tipo di grafica consistono nel fatto che lavorando con i singoli pixel si possono ottenere effetti simili a quelli della pittura e grafica tradizionale (pennello, aerografo, matita, carboncino). I programmi di fotoritocco, per esempio, funzionano con immagini raster, e i ritocchi sono appunto possibili punto per punto.

D'altra parte l'immagine si può ingrandire (su monitor o in stampa) solo ingrandendo la dimensione del pixel, che può diventare visibile, fino a creare effetti sgradevoli (pixelizzazione).

Inoltre per elaborare (spostare, modificare, cancellare) una parte dell'immagine occorre letteralmente selezionare alcuni pixel e spostarli, indipendentemente da quello che rappresentano. La zona in cui erano rimane vuota, e i pixel vanno a sovrascrivere permanentemente quelli che si trovavano nella posizione di arrivo.

Nel ruotare un'immagine bitmap, i pixel vengono risistemati in modo che l'immagine appaia ruotata; ma, a meno che la rotazione non sia stata di 90° o multipli, la rotazione è solo un'approssimazione dell'immagine originale.

Nello stampare un'immagine bitmap, la stampante riproduce l'immagine punto per punto esattamente come i punti sono. Ciò indipendentemente dalla risoluzione della stampante.

1.2.2 Grafica vettoriale

La grafica a oggetti è un'evoluzione della grafica a punti. I programmi che consentono di creare immagini ad oggetti non memorizzano il disegno come insieme di punti, ma con formule matematiche che descrivono i singoli oggetti; le formule matematiche sono scritte in qualche linguaggio (PostScript è il più noto). Ogni oggetto del disegno viene memorizzato in un database interno di oggetti grafici descritti matematicamente.

Grazie a questa tecnica gli oggetti si possono ingrandire, rimpicciolire, ruotare, ridimensionare, colorare (bordi e contenuto) senza nessuna perdita di qualità. Inoltre, gli oggetti si possono trattare in modo indipendente, come se ognuno fosse tracciato su un foglio trasparente. Gli oggetti possono essere messi uno sull'altro, eventualmente quello sopra può nascondere quello sotto, che comunque non viene permanentemente cancellato.

In fase di stampa, invece di indicare alla stampante dove vanno stampati i singoli pixel, alla stampante arriva la descrizione (matematica, descritta in qualche linguaggio, per esempio PostScript) e la stampante (che può contenere un interprete PostScript) stampa l'immagine alla sua migliore risoluzione.

Quindi, contrariamente alla grafica a punti, la qualità di una illustrazione ad oggetti è device independent: l'output uscirà alla miglior risoluzione della stampante. Generalmente poi la descrizione di un'immagine ad oggetti occupa meno spazio della descrizione di un'immagine a punti.

I formati ad oggetti possono naturalmente contenere delle parti a punti, poichè il punto è anch'esso un oggetto grafico.

In modalità bitmap la rappresentazione di un cerchio averrà con pixel neri su sfondo bianco. In modalità vettoriale il cerchio verrà rappresentato con una formula, per esempio con le coordinate del centro e la lunghezza del raggio.

Ogni programma utilizza un proprio metodo privato per memorizzare le formule dei diversi oggetti dell'immagine.

Gli oggetti grafici rappresentabili sono punti, linee, archi circolari e curve. Poichè esistono moltissime curve diverse, e un programma non può gestirle tutte, in computer grafica vettoriale viene normalmente utilizzato un tipo di curva, studiato per la prima volta negli anni 60 dal matematico Pierre Bèzier.

Pierre Bèzier *era un ingegnere francese nato nel 1910 e morto nel 1999. Nel 1933 iniziò a lavorare per la Renault, inventando tecniche di modellazione che la Renault iniziò ad utilizzare nel dopoguerra.*

Nel 1965 Bèzier scrisse un articolo riguardante la possibilità di trasferire le curve matematiche del progetto della carrozzeria di un'automobile alle presse che l'avrebbero costruita. Il sistema si chiamava UNISURF e a partire dal 1972 è

stato utilizzato per definire le carrozzerie di diverse auto prodotte dalla Renault. In UNISURF Bèzier utilizza le curve che hanno poi preso il suo nome.

Nel 1976 nasce la prima versione di PostScript (ma allora si chiamava Design System). Si tratta di un linguaggio per la computer grafica vettoriale che utilizza le curve di Bèzier e venne inizialmente utilizzato per applicazioni di computer aided design (progettazione assistita da calcolatore). Adobe, la società creata dai progettisti di PostScript, nasce nel 1982. Nel 1985 il linguaggio PostScript viene utilizzato nella LaserWriter, la stampante laser della Apple, ora è estremamente diffuso.

1.3 Formati grafici di un'immagine

Quando l'immagine viene registrata su un supporto di massa (tipicamente su disco rigido) è necessario scrivere, oltre ai dati dell'immagine, anche altri dati che consentono di ricostruirla. Il modo in cui una immagine viene scritta su disco viene detto formato grafico. Naturalmente i formati grafici si raggruppano in due categorie: per le immagini bitmap e per le immagini vettoriali.

Per memorizzare una immagine bitmap si scrivono due tipi di informazioni:

- i valori dei pixel
  - grigio: un valore per ogni pixel
  - RGB: tre valori
  - CMYK: quattro valori
- le metainformazioni

Per le immagini a 8 bit, i valori di ogni pixel vanno da 0 a 255, per le immagini a 16 bit da 0 a 65535. Secondo il modo in cui sono organizzate le metainformazioni si hanno diversi formati grafici bitmapped tra i quali ricordiamo GIF, TIFF, BMP, JPG, PNG,...

JPEG

JPEG è un formato standard di compressione dei file grafici bitmapped. JPEG è l'acronimo di Joint Photographic Experts Group, il nome del comitato che ha scritto le specifiche. JPEG è stato progettato per memorizzare immagini a colori o a grigi di scene fotografiche naturali in modo com-

presso. Funziona bene sulle fotografie, sui quadri naturalistici e simili; non è adatto per immagini rapppresentanti fumetti, disegni al tratto, logo, lettering. JPEG tratta solo immagini statiche, ma esiste un altro standard correlato, **MPEG**, per i filmati (immagini in movimento).

PostScript è un linguaggio di programmazione specializzato per la computer grafica vettoriale (ma può trattare anche grafica bitmap). Essendo PostScript un linguaggio, un file PostScript (suffisso .ps) è un file di testo che contiene un programma (dati e istruzioni) che viene eseguito su un processore collegato con (o incorporato in) una stampante. Questo processore è detto RIP (Raster Image Processor) ed è composto di tre parti: un interprete, una parte che realizza la rasterizzazione e una terza parte che realizza la retinatura.

Il risultato dell'interpretazione del programma PostScript è un file (display list) che contiene, pagina per pagina, l'elenco degli oggetti che verranno infine stampati sulla stampante.

Per esempio, questo è un frammento di programma PostScript che traccia 30 segmenti uno sotto l'altro:

```
35 600 moveto
1 1 30 {400 0 rlineto -400 10 rmoveto} for
stroke
```

Da notare l'istruzione *for*, tipica di un linguaggio di programmazione. L'interpretazione di questo programma PostScript dà luogo alla lista dei 30 segmenti da tracciare, e al loro tracciamento. Le stampanti PostScript sono quelle che accettano istruzioni in linguaggio PostScript.

EPS

Encapsulated PostScript (EPS) è un programma PostScript formattato in modo particolare e soggetto ad alcuni vincoli (deve rispettare la Adobe Document Structuring Convention (DSC)). Può contenere una singola immagine in una singola pagina (mentre un normale programma PostScript può contenere numerose immagini in numerose pagine) e di questa immagine viene dato il bounding box (posizionamento nella pagina) in un commento.

EPS è uno standard pensato per l'esportazione e l'importazione di file PostScript in qualunque ambiente. Può contenere ogni combinazione di

testo, grafica vettoriale e grafica bitmap, il tutto descritto in PostScript.

PDF

PDF è un formato grafico derivato dal PostScript con il quale condivide il modo di descrivere gli oggetti grafici: le pagine, i colori, le coordinate, il testo, i bitmap. Un file PDF è molto simile alla display list di un rip. Non è un programma come un file PostScript, ma un elenco di oggetti grafici ottenuti interpretando (eseguendo) un file PostScript (l'interprete può essere Acrobat Distiller).

SVG

Il formato SVG, acronimo di Scalable Vector Graphics, è stato creato per fornire uno standard per la rappresentazione di immagini vettoriali, non solo per l'uso destinato al web. La prima versione raccomandata dal W3C è del 2001. SVG può essere inserito in unfile HTML 5.

Creiamo un file con estensione .svg

```
<svg width="100" height="100">
<circle cx="50" cy="50" r="40"
stroke="green" stroke-width="4" fill="yellow" / >
< /svg>
```

Dall'esempio possiamo subito notare alcune caratteristiche:

Il file con suffisso .svg è basato sulla sintassi XML e ne osserva le regole. La root di un documento è il tag svg che determina l'area dell'immagine attraverso gli attributi width e height. Contiene gli elementi di disegno e di testo che rappresentano l'immagine. I tag, in modo molto simile all'HTML, sono auto descrittivi e rendono comprensibile il markup, anche aprendo il file con un editor di testo.

1.4 Dispositivi input di scansione 3D

Gli scanner 3D sono strumenti per l'acquisizione di forma (e colore) di oggetti reali tridimensionali. Il punto di partenza nel processo di costruzione di un modello geometrico virtuale di un oggetto reale è costituito dal campionamento della superficie che racchiude l'oggetto stesso. Le tecnologie per ottenere tale risultato sono numerose, come risulta evi-

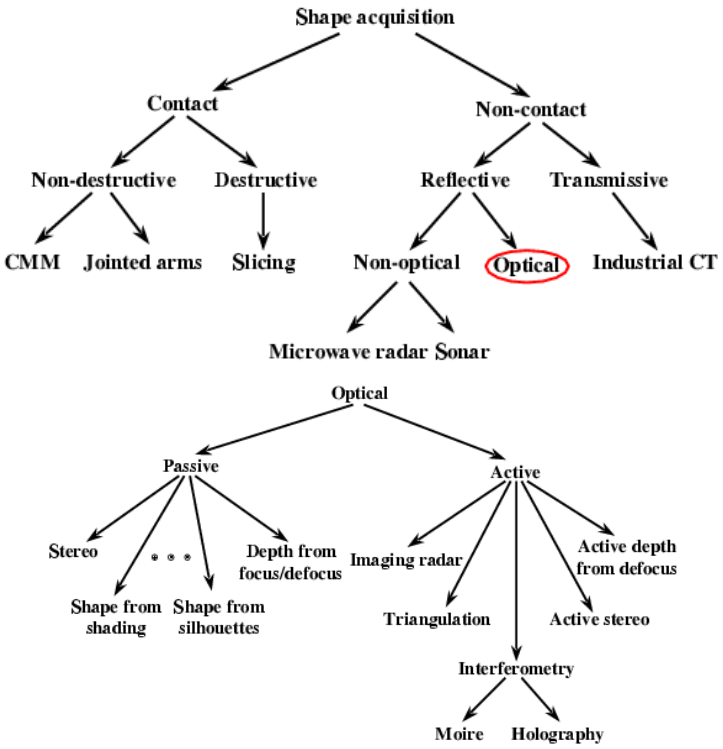


Figure 1.7: Tassonomia dei sistemi automatici e semiautomatici per l'acquisizione di oggetti reali tridimensionali.

dente dalla tassonomia riportata in figura 1.7.

Sfortunatamente, non sembra esistere al momento una tecnologia adeguata per tutte le applicazioni. I diversi modelli di scanner 3D si differenziano per principi di funzionamento, prezzo, prestazioni e applicazione. In generale i parametri tecnici da prendere in considerazione nella valutazione di questo tipo di strumenti sono fondamentalmente quattro: **accuratezza** (legata all'errore di misura), **densità massima di campionamento** (**risoluzione**), **velocità** di acquisizione e **affidabilità** .

Una quinta proprietà, apprezzata in molte applicazioni, è la possibilità di acquisire anche il colore (**texture**). Un altro aspetto da tenere in considerazione è il prezzo, che varia da poche migliaia di euro fino ad arrivare alle centinaia di migliaia.

Si assumerà nel seguito che l'output di una singola acquisizione sia costi-

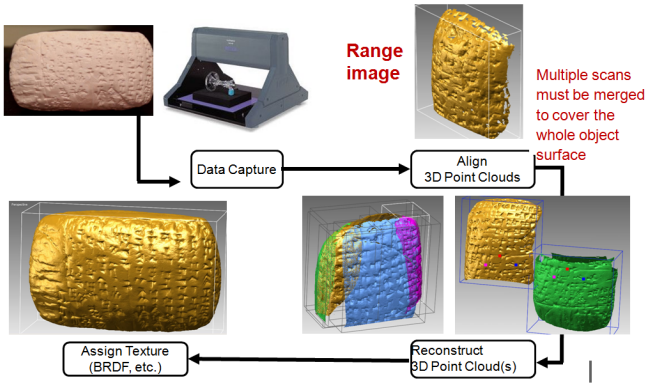


Figure 1.8: Ricostruzione di una tavoletta con incisioni cuneiformi acquisita da scanner 3D mediante allineamento e merging delle diverse range image.

tuito da una funzione ad un sol valore definita su una griglia $m \times n$, corrispondente al campionamento della porzione di superficie "vista" dallo scanner.

Chiameremo la funzione ad un sol valore in vari modi: mappa di profondità (o il suo corrispondente inglese, **range image**) quando ci interesserà sottolineare la particolare struttura (a griglia) dei dati; **scansione** come termine generico.

In Fig.1.8 sono mostrate le fasi di allineamento e merging delle diverse range image per la ricostruzione di una tavoletta acquisita con scanner 3D.

1.4.1 Breve descrizione delle principali tecnologie di acquisizione

Nella tassonomia mostrata in figura 1.7 osserviamo che i dispositivi vengono classificati nelle due principali categorie di scanner a contatto o non-contatto. Nella prima ricadono i digitalizzatori (manuali) e scanner touch probe (automativi). Nella seconda gli scanner ottici passivi e attivi.

Figure 1.9: Dimension 3D ScanBook: scanner a profilo

Scanner ottici passivi

Della grande varietà di scanner che ricadono in questa categoria, detti passivi in quanto acquisiscono informazioni dalla sola luce riflessa dall'oggetto da acquisire, saranno descritti brevemente solamente i profiler ("shape from silhouette") e quelli fotogrammetrici (denominati "stereo" nella tassonomia di figura 1.7).

Profilers

Il processo di acquisizione per questo tipo di dispositivi è concettualmente suddivisibile in due fasi distinte: rilevamento di un singolo profilo (o silouette) e ricostruzione dell'oggetto a partire dai profili ottenuti, facendolo girare attorno ad un asse fisso, tipicamente verticale. Nella prima fase si fotografa l'oggetto davanti ad uno sfondo di colore opaco e uniforme (spesso verde o blu) eliminando poi dall'immagine risultante il colore di sfondo. Il secondo passo semplicemente pone l'oggetto su di una piattaforma girevole controllata dal calcolatore, che conosce esattamente a quale angolo di rotazione corrisponde ciascun profilo. Un esempio di questo tipo di scanner è dato in figura 1.9.

Il maggiore svantaggio di un approccio di questo tipo è dato dal fatto che non si possono acquisire oggetti concavi. Alcune case produttrici colmano questa lacuna utilizzando tecnologie diverse per le parti concave.

Scanner fotogrammetrici

La fotogrammetria consente, attraverso l'elaborazione di due o più immagini di uno stesso oggetto prese da punti di vista differenti, di ricostruire l'oggetto in questione. Noti tre punti appartenenti all'oggetto, una dimensione qualunque, e le caratteristiche fisiche della fotocamera, è possibile risalire alla posizione di altri punti dell'oggetto stesso.

Il grande vantaggio di questo approccio è l'estrema velocità di acquisizione delle immagini, rendendo possibile la costruzione di modelli tridimensionali di esseri viventi, per esempio. Il prezzo da pagare è il tempo necessario ad elaborare le immagini, in quanto il procedimento è piuttosto complesso. Questo è la causa di un altro inconveniente: il costo piuttosto alto di queste apparecchiature.

Scanner ottici a luce strutturata (attivi)

Gli scanner a luce strutturata funzionano proiettando un pattern luminoso sull'oggetto (da cui il nome "attivi") e riprendendo con una fotocamera digitale l'immagine risultante. Usando la triangolazione è poi possibile risalire dal pattern riflesso (e quindi distorto dalle caratteristiche superficiali dell'oggetto da acquisire) alla posizione dei punti nello spazio. La fonte luminosa che genera il pattern può essere una normale lampadina alogena (scanner a luce non coerente) o un laser (scanner a luce coerente).

Gli scanner laser a triangolazione sono quelli di gran lunga più utilizzati, grazie ai notevoli vantaggi che offrono rispetto ai sistemi concorrenti; fra questi i principali sono sicuramente la velocità di acquisizione e l'accuratezza del campionamento.

Il principio di funzionamento si basa sulla triangolazione, come mostrato in figura 1.10. Una sorgente laser proietta il pattern sulla superficie da acquisire. Ad una distanza nota dalla sorgente vi è una telecamera (CCD array) che "cattura" l'immagine riflessa dalla superficie dell'oggetto.

In linea di principio, nota la lunghezza focale della lente e le caratteristiche fisiche della telecamera è possibile ottenere le tre coordinate spaziali di ciascun punto della striscia proiettata sull'oggetto. Spesso l'estrazione dell'informazione spaziale dalla traccia video avviene nelle tre fasi rapp-

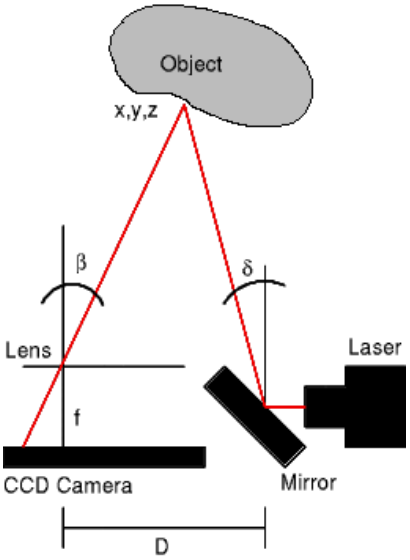


Figure 1.10: Principio di funzionamento degli scanner laser a triangolazione: f e D sono noti perché fissati in fase di progetto, δ si ottiene con un semplice encoder angolare, mentre β, il parametro più delicato, viene ottenuto elaborando l'immagine acquisita dalla telecamera ed è funzione di f e della dimensione dei pixel di quest'ultima.

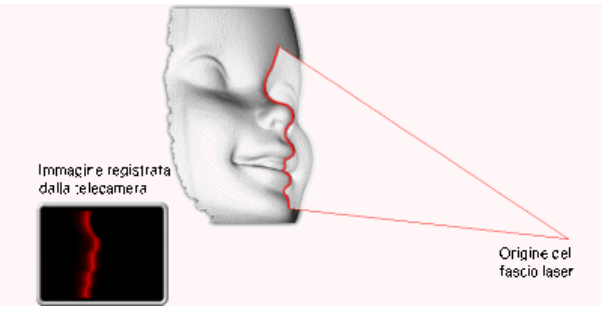
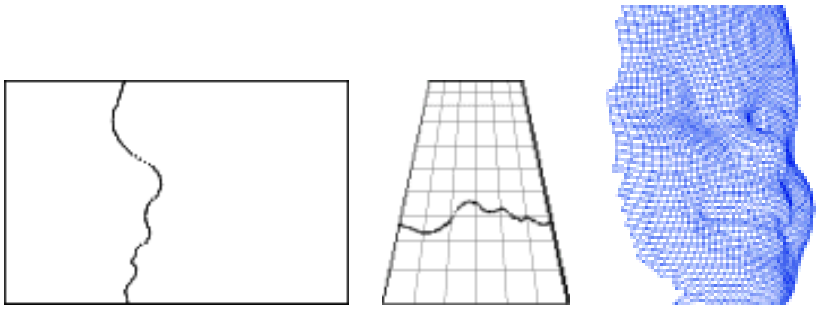


Figure 1.11: Una rappresentazione pittorica del processo di scansione

resentate rispettivamente nelle figure 1.11 e 1.12.

I due problemi principali di questo tipo di sistemi sono il costo piuttosto elevato e il fatto che, come per tutti gli scanner ottici, risulta problematica l'acquisizione di superfici con proprietà ottiche particolari (riflettenti,



(a) (b) (c)

Figure 1.12: (a) L'immagine viene elaborata per estrarre un unico punto per riga; (b) Si passa da un sistema di riferimento "locale" ad uno relativo al mondo reale attraverso un processo di calibrazione; (c) Tramite triangolazione si ottengono le tre coordinate x, y, z di ciascun punto.

trasparenti, traslucide).

Lo scanner 3D **Minolta VIVID 900**, (ENEA, Bologna, DAPT, Facoltà di Ing. Bologna), è uno scanner ottico a luce strutturata coerente (laser) dalle ottime prestazioni (figura 1.14(d)).

A seconda delle ottiche utilizzate, il volume di scansione varia da $110 \times 80 \times 40$ mm a $1200 \times 900 \times 750$ mm. La risoluzione è 0.17 mm, indipendentemente dalla direzione. Il pattern proiettato è una linea che si sposta in virtù del fatto che il piano di luce laser che la genera viene deviato da uno specchio rotante azionato da un galvanometro di precisione. L'immagine riflessa del pattern viene raccolta da una matrice CCD 640×480.

La messa a fuoco è automatica e assieme all'informazione geometrica viene acquisito anche il colore con una risoluzione di $640 \times 480 \times 24$.

Lo scanner NEXT ENGINE (Laboratorio High Performance Computer Graphics and Vision, Università di Bologna), in Fig. 1.13, è dotato di un software proprietario (ScanStudio) per la ricostruzione della mesh a partire dalle nuvole di punti acquisiti ed è fornito di due piattaforme per effettuare le scansioni, la prima, denominata **Autodrive** (vedi figura 1.13(sinistra)) è una piastra che ruota sull'asse z e permette di fare delle acquisizioni di oggetti a 360° ed ha un solo grado di libertà, per questo è indicata nei casi in cui non siano richiesta scansioni della parte superiore degli oggetti. Permette di fare acquisizioni di oggetti che variano per

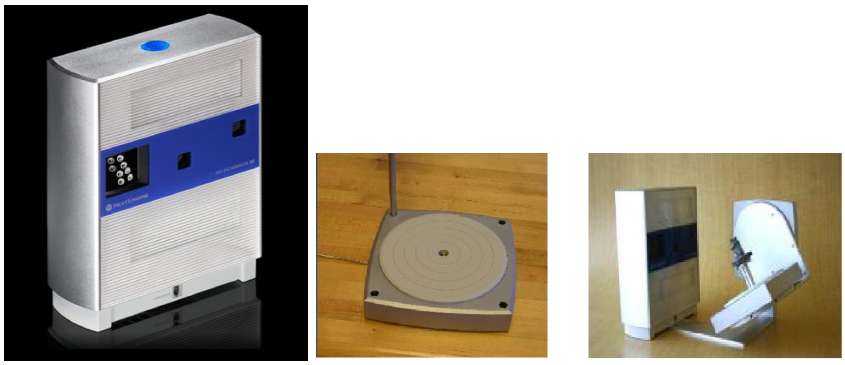


Figure 1.13: Scanner Laser 3D - NextEngine; Autodrive e MultiDrive.

dimensione tra $7,62 \times 12,7$ cm e $25,4 \times 33,2$ cm (modalità macro e wide). In caso contrario è assolutamente consigliato utilizzare il secondo device, il **MultiDrive** (figura **??**), che permette di fare acquisizioni della parte superiore e inferiore degli oggetti: infatti il supporto per le acquisizioni può ruotare su due assi.

Le proprietà ottiche dei materiali influenzano non poco la qualità delle scansioni. Le zone di colore molto scuro (nero in particolare), non vengono registrate. Materiali trasparenti, altamente riflettenti (riflessioni speculari) sono causa di rumore, spesso inaccettabile. Anche materiali che causano lo scattering del fascio laser sortiscono effetti analoghi. Una soluzione parziale e non sempre applicabile consiste nel ricoprire (alcune parti) l'oggetto con una sostanza adatta. Anche quando questo risulta possibile, l'informazione sul colore viene comunque persa.

Riassumiamo i pregi e i difetti dello scanner laser in generale. Cominciamo dai pregi:

- Accurato
- Portatile
- Alta velocità di acquisizione
- Acquisisce anche il colore

e finiamo con i difetti:

- Costoso
- Sensibile alle proprietà ottiche dei materiali

(a)

(b)

(c)

(d)

Figure 1.14: (a) Immersion MicroScribe 3D a contatto, manuale; (b) Roland PIX-30 scanner a contatto automatico; (c) Cyberware Whole body laser scanner; (d) Scanner laser VIVID 900, Minolta.

Scanner a contatto

Gli scanner di questo tipo sono composti da un sensore, il cui unico scopo è quello di rilevare il contatto con un oggetto solido, collegato attraverso un sistema meccanico a una base fissa. Toccando l'oggetto da acquisire in un punto è possibile risalire alla posizione di quest'ultimo rispetto ad un sistema di riferimento solidale con la base dello scanner. Il sistema meccanico che collega il sensore alla base può essere sia passivo (vedi figura 1.14(a)), ovvero azionato manualmente da un operatore, oppure automatico (figura 1.14(b)).

I principali vantaggi di questi tipi di scanner sono la semplicità d'uso e il costo contenuto. L'estrema lentezza del processo di acquisizione costituisce, d'altra parte, il maggior deterrente nell'adozione di dispositivi di questo tipo.

Lo scanner 3D **Roland PIX-30**, (Dipartimento di Matematica, Università di Bologna), è uno scanner a contatto di tipo automatico, ovvero il sensore è parte di un sistema mobile a tre gradi di libertà, che gli consente di raggiungere tutti i punti del volume di campionamento. Il sensore, di tipo piezoelettrico, è costituito da un ago metallico che può muoversi verticalmente (spostamento lungo l'asse z).

Questo è poi solidale ad un cursore che si sposta longitudinalmente (spostamento lungo l'asse x) lungo il ponte. Il pianale di appoggio è anch'esso mobile, consentendo lo spostamento nella direzione y. Il volume massimo di scansione è di 304.8mm lungo l'asse x, 203.2mm lungo l'asse y e 60.5mm lungo l'asse z. La distanza minima inter-campione è di 0.05mm in x, 0.05mm in y e 0.025mm in z.

Questi ultimi sono valori di tutto rispetto e consentono al PIX-30 di campionare oggetti molto dettagliati. Il PIX-30 è per sua natura molto lento nel processo di acquisizione; quando poi la risoluzione è alta, i tempi diventano veramente lunghi, fino a diverse ore per i pezzi più grandi.

Riassumendo, i pregi di questo dispositivo sono:

- Risoluzione elevata e variabile con granularità fine
- Costo contenuto
- Indipendenza dalle proprietà ottiche del materiale da acquisire

Non è comunque privo di difetti:

- Bassa velocità di acquisizione
- Volume di scansione limitato
- Delicatezza del sensore

1.5 Stampante

Anche la stampante è un dispositivo raster, in quanto può depositare sulla carta un raster (una griglia rettangolare) di minuscoli punti colorati

(o neri).

Nelle stampanti da scrivania il colore è normalmente ottenuto in sintesi sottrattiva a partire dai tre primari ciano, magenta e giallo (C, M, Y, cyan, magenta, yellow), oppure dai quattro primari CMYK (K sta per nero, black) o da sei primari (CcMmYK) dove c e m stanno per varianti di C e M.

Ogni stampante raster ha una risoluzione massima di stampa. Per esempio la Epson Stylus Color 1520 ha una risoluzione massima di 1440 pixel per pollice (dots per inch -**dpi**-).

Le stampanti desktop oggi in produzione utilizzano queste tecnologie di stampa:

- laser a toner (laser)
- getto d'inchiostro (inkjet)
- inchiostro solido (solid-ink)
- sublimazione (dye-sublimation)
- cera termica (thermal-wax)

1.6 Raster Scan Display System

Un monitor/video (display) è un dispositivo di output che fa parte di un più complesso computer vision system detto Raster Scan Display System.

I sistemi grafici a display raster consistono essenzialmente di tre componenti:

1. Scheda grafica (detta anche video card, graphics card) è una scheda hardware dedicata ad elaborare dati grafici e a pilotare il Display raster. Al suo interno si trova la GPU o PROCESSORE GRAFICO e una speciale memoria RAM detta Frame Buffer
2. MONITOR (display raster) connesso via cavo alla scheda grafica
3. Un device driver mediante il quale il sistema operativo controlla la scheda video.

Uno schema a blocchi di un raster scan display system è visualizzato in Fig. 1.15. I driver grafici e il sistema operativo inviano istruzioni e dati attraverso il PCI-E alla scheda grafica. Il (Peripheral Component Interconnect Express) PCI E è l'evoluzione del bus di espansione PCI,

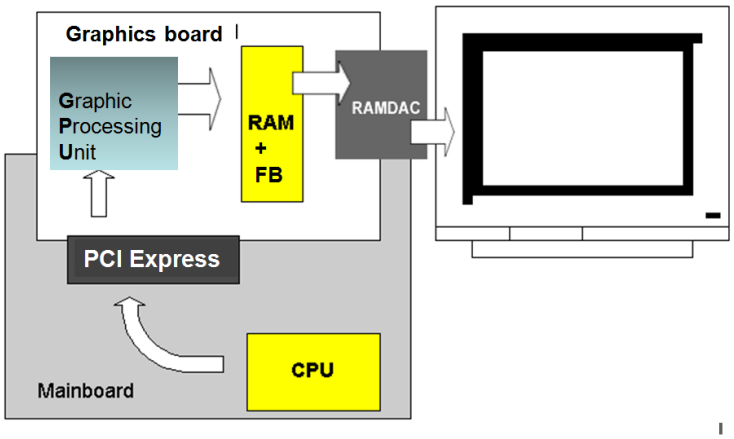


Figure 1.15: Raster scan display system

introdotto con i primi Pentium che ha preso il posto della vecchia interfaccia per schede grafiche, l'AGP. Le istruzioni sono elaborate dalla GPU. La GPU memorizza e ottiene dati dalla graphics memory o RAM durante l'esecuzione delle istruzioni. Per poter risparmiare memoria alcune schede grafiche utilizzano unità di compressione e decompressione all'interno della CPU. La GPU scrive su una memoria RAM speciale detta Frame Buffer l'intensità di ogni pixel (colore) che sarà poi inviata al monitor per essere visualizzata a video. A seconda della tecnologia del monitor sarà necessario utilizzare un convertitore digitale/analogico nella trasmissione da scheda grafica a monitor.

Il display (monitor) utilizzato con il personal computer è un dispositivo raster: lo schermo cioè consiste di una matrice rettangolare di pixel (picture element). Ogni pixel del monitor può assumere un colore tra quelli disponibili. L'intensità (colore) di ogni pixel viene letta dalla memoria frame buffer.

Le principali tecnologie con le quali sono realizzati i monitor sono:

- tubo a raggi catodici (CRT, cathode ray tube);
- cristalli liquidi (LCD, liquid crystal display);
- al plasma (PDP, plasma display peripheral).

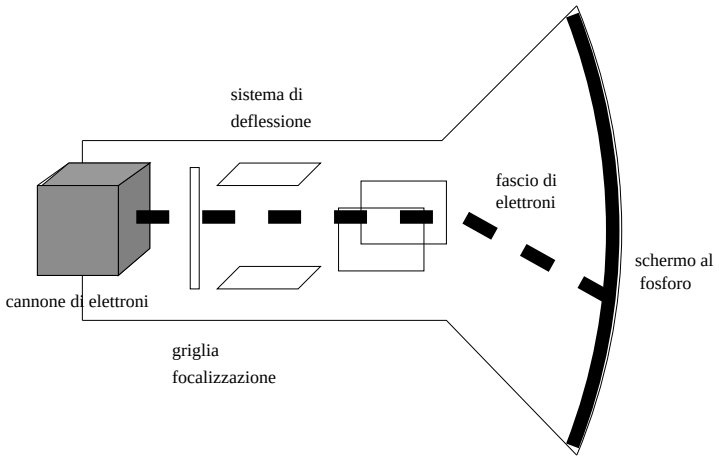


Figure 1.16: Monitor CRT

1.6.1 Monitor con tubo a raggi catodici (CRT)

I monitor del tipo CRT (Cathode Ray Tube) sono costituiti da un tubo a raggi catodici.

La tecnologia del tubo a raggi catodici risale ai primi anni del 1900. L'architettura di un monitor CRT è illustrata in figura 1.16. Un monitor CRT è semplicemente costituito da un tubo di vetro nel quale si realizza il vuoto, cioè si aspira tutta l'aria in esso contenuta, con una parte anteriore rivestita da fosfori, sostanze in grado di emettere luce se colpite da cariche elettriche in atmosfera rarefatta. Un fascio di elettroni viene generato dal catodo per emissione termoionica, ossia dovuta al riscaldamento sotto vuoto di un elemento metallico o di un ossido metallico, e viene incanalato mediante un campo elettrico verso lo schermo. La quantità di elettroni che fluiscono verso lo schermo è controllata da un potenziale di una griglia posta immediatamente dietro il catodo. A causa delle forze di repulsione che esercitano l'uno sull'altro, gli elettroni tenderebbero a disperdersi, pertanto è necessario disporre di un sistema che li faccia collimare (sistema di focalizzazione). La deflessione viene solitamente operata da due distinti apparati che operano nelle due direzioni geometriche dell'immagine. Il sistema di deviazione, formato solitamente da bobine elettromagnetiche, produce campi elettrici e magnetici che permette di deviare il fascio di elettroni nei differenti punti della superficie

in funzione dell'intensità di corrente che attraversa le bobine.

Il raggio elettronico focalizzato e deflesso, urtando la parete del tubo catodico, ricoperta di fosfori, trasferisce la sua energia agli elettroni dei fosfori portandoli a livelli quantici più elevati rispetto allo stato normale. Tornando successivamente a tale stato gli elettroni emettono fotoni che provocano la luminescenza dello schermo. La luminosità del fosforo del CRT dura solo una piccola frazione di secondo e pertanto, per poter mantenere stabile un'immagine, è necessaria che essa sia rigenerata rapidamente e ripetutamente.

In generale nei CRT a colori abbiamo tre cannoni di elettroni. Come sappiamo il nostro occhio reagisce ai tre colori primari verde, rosso e blu, e alle loro combinazioni che danno luogo ad un numero infinito di colori. I fosfori presenti sulla superficie frontale del tubo catodico sono particelle molto piccole che il nostro occhio non è in grado di percepire e che riproducono questi tre colori primari; questi fosfori, come detto, vengono accesi se colpiti dagli elettroni che vengono generati dal catodo e sparati dai tre cannoni presenti nel CRT; ogni cannone viene abbinato ad uno dei tre colori primari. I tre cannoni inviano elettroni sui vari fosfori, questi si accendono combinando i tre colori primari e formando le immagini con i colori voluti.

Come detto per ogni cannone abbiamo un fascio di elettroni che deve andare a colpire i fosfori di un ben determinato colore (verde, rosso o blu); è chiaro che bisogna evitare che il fascio di elettroni che deve colpire il fosforo rosso vada a colpire il verde o il blu e a questo scopo ci vengono in aiuto le cosiddette maschere.

Le maschere più diffuse sono sostanzialmente tre e vanno sotto il nome di "Shadow Mask", "Aperture Grille" e "Slot Mask".

SHADOW MASK

E' la maschera e quindi la tecnologia più diffusa nei monitor CRT (vedere figura 1.17); il principio su cui si basa è quello di anteporre allo schermo con i fosfori una griglia metallica che funziona come un mirino, cioè impedisce che un fascio di elettroni possa colpire il fosforo sbagliato e in un area differente da quella voluta. La maschera forma una griglia omogenea di punti, dove ogni punto è costituito da tre fosfori di colori primari verde, rosso e blu che possono essere eccitati dal bombardamento

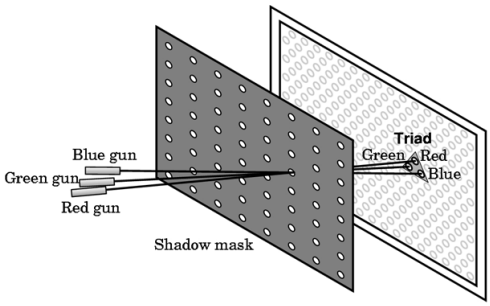


Figure 1.17: Shadow mask

elettronico operato dai tre cannoni. Eccitando in modo differente i fosfori e quindi combinando in modo opportuno i tre colori primari si può ottenere nel punto desiderato il colore voluto. La minima distanza tra fosfori di ugual colore prende il nome di dot pitch che rappresenta un indice della qualità dell'immagine; ad un dpi più basso corrisponde una qualità dell'immagine più alta.

Se l'immagine deve essere mantenuta il fosforo necessita di essere continuamente alimentato con elettroni perchè la persistenza del fosforo (il tempo in cui la luce viene emessa) è breve: da 40 microsecondi a 3 secondi. Per immagini in movimento è meglio una persistenza breve, ma la persistenza lunga riduce lo sfarfallio.

1.6.2 Monitor a Cristalli liquidi (LCD)

I monitor LCD (Liquid Crystal Display) illustrati in figura 1.18 sono monitor a cristalli liquidi, cioè costituiti da una particolare sostanza che si trova a metà strada tra un solido ed un fluido; questa sostanza possiamo ritenerla composta da tante piccole bacchette orientabili. La scoperta dei cristalli liquidi risale a più di cento anni fa, ma le prime applicazioni ebbero inizio nel momento in cui si scoprì che le piccole bacchette che li componevano, sotto lo stimolo di una carica elettrica, cambiavano il proprio orientamento, in modo tale da modificare le caratteristiche del fascio di luce che li attraversava. Sulla base di questa scoperta e con gli studi successivi che vennero effettuati, si riuscì a trovare un legame tra stimolo elettrico ed orientamento dei cristalli, tale da consentire la visualizzazione

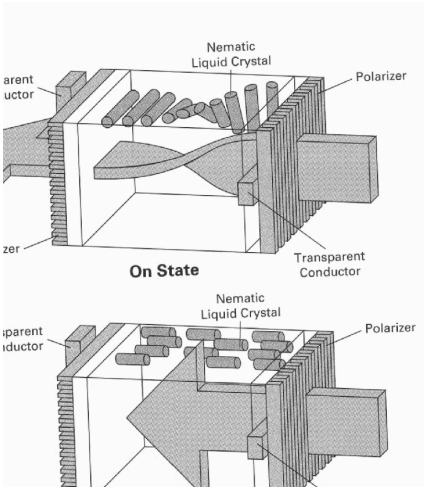


Figure 1.18: Monitor LCD

di immagini. Le prime applicazioni furono nel campo dei display per le calcolatrici e successivamente nei monitor dei PC portatili.

I cristalli liquidi sono composti organici che possono avere sia le proprietà dei liquidi sia le proprietà dei cristalli: come i liquidi possono essere versati ma come i cristalli mantengono una struttura molecolare ordinata.

Le molecole nei cristalli liquidi si possono considerare disposte su livelli; i cristalli liquidi nematici ritorti (twisted nematic, "nematico" significa filiforme) hanno le molecole del livello superiore orientate ad angolo retto con le molecole del livello inferiore. Quando viene applicato un campo elettrico, le molecole si allineano parallelamente tra loro. Quindi il cristallo liquido può essere in uno di due stati: twisted on oppure twisted off. Quando il cristallo liquido è twisted on ha l'utile proprietà di cambiare la polarizzazione della luce polarizzata di 90°, quando twisted off riflette la luce incidente.

I monitor LCD sono costituiti da ben sette strati fondamentali e partendo dalla zona centrale abbiamo due pannelli di vetro tra i quali vengono messi i cristalli liquidi.

Monitor LCD a **matrice passiva**. E' la tecnologia più economica adottata nella realizzazione di schermi LCD e la troviamo nei classici display

denominati DSTN.

Il termine matrice viene in pratica introdotto per la caratteristica suddivisione dello schermo in punti ognuno dei quali viene indirizzato separatamente dagli altri e quindi separatamente dagli altri può essere acceso o spento a piacere per la formazione di immagini a video attraverso la presenza di elettrodi. Si denomina passiva perchè questa tecnologia non è in grado di mantenere per lungo tempo le informazioni a video ed inoltre per creare delle immagini necessita sempre della presenza di un campo elettrico; le immagini si formano riga per riga, con il campo elettrico che passa da riga in riga provocandone l'accensione e la successiva e progressiva dissolvenza appena il campo stesso viene disattivato in quel punto - l'immagine non scompare subito dalle prime righe tracciate anche dopo che è stato tolto il campo elettrico per il semplice fatto che i cristalli rimangono in posizione orientata ancora per qualche secondo. I display ora descritti del tipo a matrice passiva hanno dei grossi limiti per quanto concerne l'aspetto qualitativo ed applicativo infatti hanno un'immagine non perfettamente nitida e tendenzialmente sfarfallante, ed inoltre l'uso di cristalli lenti, dato che rimangono in posizione anche dopo aver tolto il campo elettrico, non permette una corretta visualizzazione di immagini in movimento. Non meno trascurabile è il problema che deriva dall'interferenza che nasce tra i vari elettrodi che si manifesta sotto forma di aloni sul display.

Il monitor a **matrice attiva** (TFT, thin film transistor) è il più moderno. In questo monitor l'indirizzamento di ogni singolo pixel avviene tutto alle spalle del display stesso e i pixel sono attivati da un apposito transistor. Quindi non è più necessario porre davanti al video una serie di elettrodi: è sufficiente la presenza di un'unica lastra trasparente con funzioni di "terra". Il contrasto è quattro volte maggiore rispetto agli STN e l'angolo di visuale è più ampio.

Con la matrice passiva i vari elettrodi ricevevano ciclicamente della tensione man mano che il display veniva rigenerato per linee e l'eliminazione del campo elettrico comportava il progressivo dissolvimento dell'immagine relativa a quella determinata cella, dato che i cristalli ritornavano progressivamente nella loro configurazione originaria, nella matrice attiva invece si è abbinato ad ogni elettrodo un transistor di memoria in grado di memorizzare un'informazione digitale (numero binario 0 o 1) e quindi

di mantenere quella determinata immagine fino al ricevimento di un altro segnale. Dato che il tentativo di ritardare il dissolvimento dell'immagine nei display a matrice passiva veniva parzialmente raggiunto con strati maggiori di cristalli liquidi per aumentarne l'inerzia e quindi rallentarne i movimenti, ora abbiamo la possibilità di ridurre fortemente lo strato di cristalli liquidi presente. I transistors, come gli elettrodi, devono essere fabbricati con materiali trasparenti per evitare di bloccare il passaggio del fascio luminoso e vengono poi posizionati sul retro del display su uno dei pannelli di vetro che contengono i cristalli liquidi; a tal scopo si impiegano delle pellicole di materiale plastico "Thin Film Transistor", transistor a film sottile, da cui la nota sigla TFT, caratterizzate da uno spessore sottilissimo tra 1/10 e 1/100 di micron.

Il colore viene ottenuto tramite l'impiego di tre filtri riproducenti i tre colori fondamentali rosso, verde e blu e facendo passare il fascio di luce attraverso i filtri stessi. Combinando per ogni singolo punto o pixel dello schermo, questi tre colori fondamentali, siamo in grado di riprodurre ogni colore; la combinazione avviene gestendo in modo opportuno gli elettrodi che generano puntualmente la differenza di potenziale che orienta, generando un piccolo campo elettrico, i cristalli.

I monitor a cristalli liquidi hanno numerosi **vantaggi**: alta risoluzione, uniformità nello spazio e nel tempo (perchè ogni singolo pixel può essere indirizzato separatamente e non viene influenzato dai pixel adiacenti). Inoltre sono sottili (1-2 centimetri) e leggeri, necessitano di una potenza elettrica molto bassa, non espongono l'utente ai pericoli dei raggi catodici. Presentano tuttavia anche alcuni **difetti**: una risoluzione temporale molto bassa (problemi con le immagini dinamiche); bassa luminanza e basso contrasto cromatico; gamut di colore ridotto rispetto ai monitor CRT (soprattutto a causa del primario blu).

Vi sono comunque altre tecnologie che i vari costruttori stanno da tempo sviluppando ed alcune di queste vanno sotto il nome di "plasma" PDP (Plasma Display Panels) e FED (Field Emission Display). Monitor al plasma sono adatti ai grandi schermi con dimensioni di 40" e più.

1.6.3 Monitor al plasma

La tecnologia al plasma si basa sulla luce fluorescente. In ogni cella del display si trova il gas, normalmente una miscela di Neon e Xenon, che in un campo elettrico modifica le proprie caratteristiche. Applicando una tensione il gas ionizza e diventa un plasma che cede luce ultravioletta non visibile. La parte esterna del pannello è ricoperta con fosfori RGB che rendono visibile la luce. Tre celle adiacenti costituiscono un pixel.

Un clock di 30 kHz è sufficiente a visualizzare una immagine senza sfarfallio. Questa caratteristica ha tuttavia effetto anche sulla visualizzazione di immagini mobili. Mentre un colore viene già acceso, l'altro non è ancora illuminato. Per minimizzare questo problema i diversi costruttori utilizzano metodi diversi.

I display al plasma hanno un principio di funzionamento molto simile a quello delle lampade al neon, sono essenzialmente costituiti da un tubo nel quale viene creato il vuoto e nel quale sono presenti delle coppie di elettrodi strutturati in modo da sviluppare delle scariche elettriche attraverso un gas inerte, creando in questo modo della luce. Il display viene creato accoppiando due superfici in vetro e inserendo nell'intercapedine sigillata che si viene a creare del gas inerte, quale argon o neon; si dispongono poi sulle superfici di vetro, dei piccoli elettrodi trasparenti che, alimentati da una corrente alternata, producono una scarica elettrica puntiforme che va ad illuminare il vetro; in pratica il gas caricato elettricamente che prende il nome più tradizionale di plasma, emette una luce nel campo dell'ultravioletto che va a colpire ed eccitare dei fosfori che a loro volta emettono luce nel campo del visibile. Ogni pixel dello schermo funziona più o meno come una tradizionale lampada fluorescente. Alta luminosità e contrasto, unita all'assenza totale di sfarfallio rappresentano un ottimo biglietto da visita per questi display; si ricordi anche che l'angolo di visione aumenta rispetto ai soli 45° dei monitor LCD e quindi anche persone che si trovano ad osservare il monitor con angolo di osservazione maggiore di 45° non avranno problemi. Gli aspetti negativi sono rappresentati dal consumo di elettricità che aumenta con la dimensione e dal fatto che la risoluzione è tendenzialmente bassa; le proprietà dei fosfori decadono nel tempo rendendo progressivamente lo schermo meno luminoso e questo limita la durata dei monitor al plasma a circa 10.000 ore di

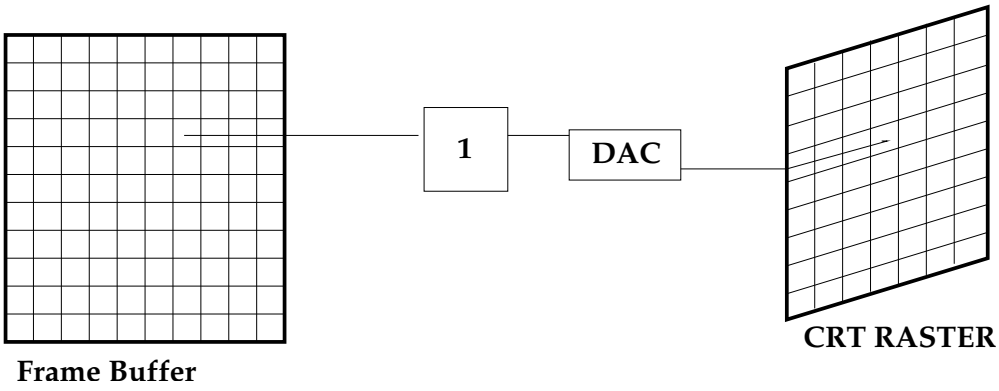


Figure 1.19: Color buffer a 1 piano di bit per video monocromatico

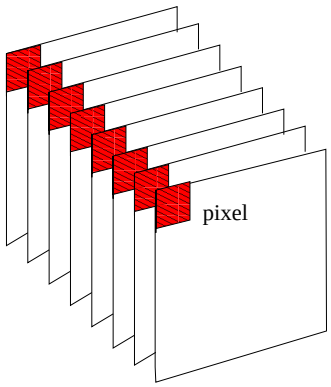


Figure 1.20: Color buffer a 8 piani di bit: 256 colori possibili per il pixel

funzionamento (circa 5 anni ipotizzandone l'uso in un ufficio).

1.6.4 Frame Buffer (FB)

Il frame buffer è una memoria RAM riservata al video. Il frame buffer è una collezione di diversi buffer (piani di bit) dedicati. Il principale tra questi buffer è il **color buffer**. Altri buffer disponibili sono lo **Z-buffer**, anche chiamato depth buffer, che gestisce la visibilità. Questo buffer ha tipicamente 24 bit per pixel. Il **double buffer** (front buffer + back buffer) è comunemente usato per velocizzare le operazioni di caricamento del FB con l'immagine resa e visualizzazione del suo contenuto a video. Se

l'immagine finale mostrata a video è contenuta nel front buffer, un off-screen back buffer contiene la scena che sta per essere mostrata. I due buffer vengono poi scambiati dal processore grafico tipicamente durante la scansione verticale. Dopo lo scambio i ruoli di back e front buffer si scambiano, e così succede dopo ogni frame visualizzato.

Per hardware che supportano stereo rendering, nel quale due immagini sono usate per dare agli oggetti un'apparenza tridimensionale, tipicamente si possono utilizzare due separati display buffer, o **stereo buffer**.

Altri buffer del cui utilizzo si parlerà più avanti nel corso, sono l'**accumulation buffer** e lo **stencil buffer**.

Color Buffer

Nel color frame buffer ogni posizione di memoria indica il colore di un pixel. Poichè il colore del pixel di un monitor si forma in sintesi additiva a partire da tre primari R, G e B, ogni posizione sarà divisa in tre zone per i tre valori R, G e B.

Se per ogni colore primario sono riservati otto bit (1 byte), ogni pixel richiede 24 bit (3 byte) e, per esempio, un monitor di 1024 x 768 pixel richiede 1024 x 768 x 3 byte = 768 x 3 Kbyte = 2304 Kbyte = 2.25 Mbyte di video RAM.

Metodi per codificare immagini a colori nel frame buffer:

- **Video monocromatico**
  il color buffer ha 1 bit per locazione, quindi basta 1 bit per pixel e quindi il color buffer ha un solo piano di bit (figura 1.19).
- **Video a colori**
  il color buffer ha N bit per locazione, cioè N bit per pixel e quindi il color buffer ha N piani di bit (figura 1.20).

Il color buffer per un monitor a colori può essere gestito in modalità true color o pseudocolor.

Modalità True Color I valori memorizzati nel color buffer rappresentano i valori effettivi delle componenti colore da visualizzare. Quindi un color buffer a N piani di bit è in grado di gestire 2^N valori di colore differenti. In figura 1.21 è mostrato un esempio di modalità true color per un monitor

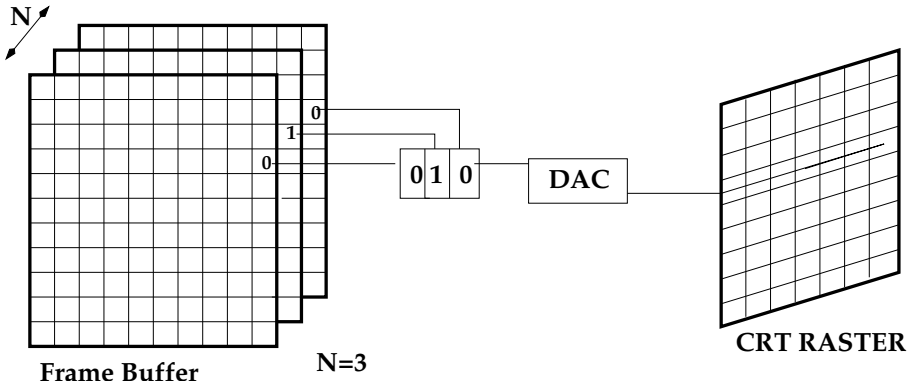


Figure 1.21: Color buffer in modalità true color: 3 piani di bit per visualizzare 8 intensità differenti (es. 8 livelli di grigio differentii)

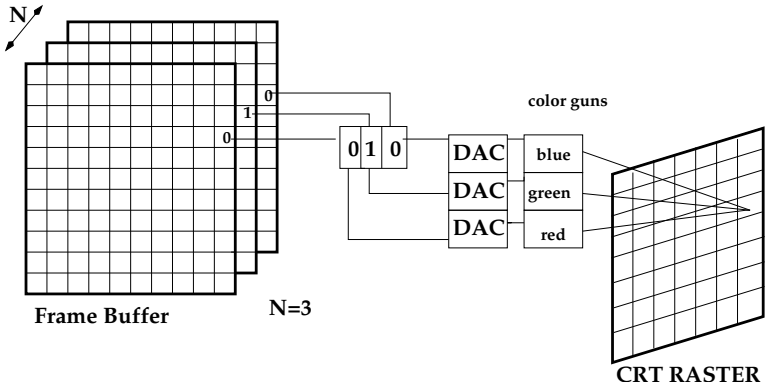


Figure 1.22: Color buffer in modalità true color: 3 piani di bit per visualizzare 8 colori differenti

che gestisce 8 livelli di grigio. Un esempio di true color per un monitor a colori è invece illustrato in figura 1.22. Con tre piani di bit utilizzati ciascuno per una componente colore diversa.

Tipicamente si fa uso di 24 piani di bit per le tre componenti colore (1 byte R, 1 byte G e 1 byte B) e un quarto byte per il canale alfa (che gestisce la trasparenza). Quindi 32 bit per pixel. In questo caso il valore di luminosità di una componente primaria può variare da 0 a 255. Quindi ci sono 256 possibili livelli di rosso, 256 possibili livelli di verde e 256 possibili livelli di blu, e quindi in totale 256 x 256 x 256 = 16.777.216 possibili colori (chiamati per brevità "16 milioni di colori") che un pixel può assumere.

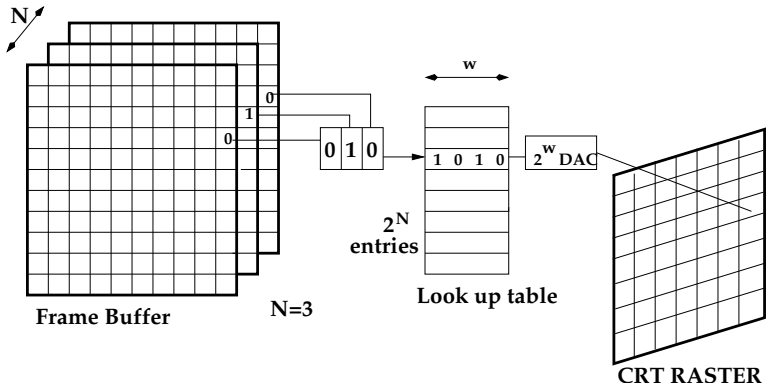


Figure 1.23: Color buffer in modalità pseudocolor: 3 piani di bit per indirizzare 2^N colori differenti

Modalità pseudocolor I valori memorizzati nel frame buffer sono trattati come indirizzi ad una tabella dei colori, chiamata *look up table* o colormap che contiene 2^N elementi ciascuno dei quali composto da w bit. Sono quindi disponibili 2^w intensità di colore differenti, contemporaneamente, sono però utilizzabili solo 2^N colori diversi. Per avere colori diversi si può via software caricare differentemente la look up table. In figura 1.23 è mostrato un esempio di modalità pseudocolor che gestisce 8 colori contemporaneamente visualizzabili, scelti da una gamma di 16 colori disponibili (2^4).

1.6.5 Convertitore Digitale/Analogico

I monitor CRT lavorano con segnali analogici. Il PC con dati digitali inviati alla scheda grafica. Prima di inviare questi segnali al monitor essi devono essere convertiti in output analogico e questo è compito del RAMDAC.

Se il monitor è digitale (monitor flat) non necessita delle funzionalità del RAMDAC. Ma se è analogico deve essere controllato da livelli di potenziale. Consideriamo per esempio un monitor CRT. Se il potenziale è 0 mV, il fosforo è spento (nero), se il potenziale è al massimo voltaggio, per esempio 700 mV, il fosforo è acceso al massimo della luminosità (per esempio il rosso più intenso).

Quando si deve visualizzare un pixel con determinate componenti colore RGB, per esempio 20, 220, 130 (se i valori vanno da 0 a 255), questi valori sono passati nella scheda grafica al FB e mediante un convertitore digitale analogico (DAC) vengono convertiti in tre segnali di ampiezza, per esempio 55, 604, 358 mV. A questo punto, se il monitor è CRT, al cannoncino che pilota il rosso viene spedito un segnale di 55 mV, al cannoncino che pilota il fosforo verde un segnale di 604 mV e infine al cannoncino che pilota il fosforo blu un segnale di 358 mV.

1.6.6 Sistema di refresh video

Il **refresh rate** è il numero di volte al secondo che il display hardware disegna i dati presenti nel FB. Questo si distingue dalla misura del **frame rate** poichè il refresh rate include il disegno ripetuto di frame identici, mentre il frame rate misura quanto spesso si può visualizzare un intero frame di nuovi dati sul display. Per esempio, un tipico valore di frame rate di un proiettore è 24 volte al secondo. Ma ogni frame è illuminato due o tre volte prima che il frame successivo venga proiettato. Quindi il proiettore gira a 24 frame al secondo, ma ha un refresh rate di 48 o 72 Hz.

In un monitor CRT il processore grafico scandisce ripetutamente il frame buffer dalla riga più alta alla più bassa e ogni riga da sinistra a destra. Questa operazione di rinfresco dell'immagine, denominata **progressive scan**, viene realizzata almeno 60/120 volte in un secondo per monitor CRT così da evitare l'effetto di tremolio (flickering) dell'immagine. In alternativa al progressive scan, la modalità **interlacing scan**, consente di formare l'immagine a video in due differenti scansioni, visualizzando prima le righe pari e poi le righe dispari. Dato che l'immagine viene formata nel suo complesso in tempi minori, i monitor non-interlacciati risentono meno del problema dello sfarfallio, ma richiedono una maggior velocità di scansione.

Il refresh rate è diventato meno importante con i monitor di tecnologia LCD perchè l'aggiornamento dell'intero schermo avviene simultaneamente invece che linea per linea. Tuttavia, i monitor LCD normalmente hanno un refresh di 60 fps.

Nei primi sistemi grafici (circa 1960) basati su un computer general-

purpose collegato mediante un convertitore digitale-analogico ad un monitor, la CPU inviava le informazioni necessarie per rigenerare l'immagine su display ad una velocità sufficiente per evitare 'sfarfallio'. Poi vennero introdotti gli special-purpose processor chiamati Display Processor Unit (DPU) collegati a CPU e video. L'unità DPU si occupava del refresh del video e gestiva anche istruzioni per visualizzare primitive a video. Attualmente l'intera gestione del display e del suo refresh è gestita dalla scheda grafica.

In seguito si descrivono le caratteristiche architetturali delle schede grafiche, mentre in sezione 1.8 si dettagliano le funzionalità delle schede grafiche, ovvero la gestione della rendering pipeline.

1.7 La scheda grafica

1.7.1 Evoluzione delle schede grafiche

Nel campo del computer graphics si sono compiuti sostanziali sviluppi nel corso degli ultimi anni, e al giorno d'oggi è possibile ottenere risultati che solo qualche tempo fa erano impensabili. Le protagoniste di questo notevole progresso sono senza dubbio le schede grafiche, che hanno permesso a tutti gli utenti di un Personal Computer, di ottenere eccellenti prestazioni grafiche a basso costo. E' chiaro che col crescere delle potenzialità di calcolo delle schede grafiche, anche il software si è evoluto, per potersi adattare alle nuove architetture e per poter sfruttare le nuove tecnologie che ne derivano. Una delle principali novità nel campo del computer graphics è senza dubbio la programmabilità, la possibilità cioè, di poter bypassare alcune delle fasi della pipeline di rendering, programmandone il comportamento. Ma facciamo un passo indietro e vediamo come si è arrivati a questo.

La CPU è la principale responsabile del processo di modelling il cui output si può per il momento pensare come definito da un insieme di primitive grafiche 3D molto semplici di tipo vertici, linee e triangoli. Queste primitive passano alla fase di rendering che le trasforma in un'immagine finale 2D che viene memorizzata nel Frame Buffer per poi essere visualizzata sullo schermo.

Nei primi general purpose computer (anni '60) la CPU era responsabile anche dell'intera fase di rendering compreso il refresh dello schermo.

Verso la fine degli anni 70 alle CPU vennero affiancati dei dispositivi special-purpose chiamati Display Processor Unit (DPU), che si occupavano del refresh del monitor ed erano collegati a CPU e video. Nel 1987 IBM introdusse il VGA (Video Graphics Array), un nuovo controller hardware che oltre alle funzioni dei precedenti DPU , offriva alcune istruzioni per visualizzare primitive a video. Era il primo passo verso la prima generazione di GPU (Graphics Processor Unit).

Nel tempo gli ingegneri hardware hanno trasferito gli algoritmi per le trasformazioni delle primitive e gli algoritmi di rasterizzazione per produrre le immagini 2D su specializzato hardware grafico 3D ad alte prestazioni. Questi multichip specializzati per la grafica 3D erano molto costosi e quindi integrati solo in costose workstation UNIX, come quelle prodotte nei primi anni 90 dalle compagnie Silicon Graphics International (SGI), e Evans Sutherland.

Con la diffusione dei personal computer, però, il peso economico del mercato dei videogiochi spingeva affinchè l'interactive rendering arrivasse sui PC a prezzi contenuti.

Questa spinta si concretizzò nella seconda metà degli anni 90 con la configurazione di un unico chip grafico a basso prezzo, chiamato **Graphics Processor Unit** (GPU), facilmente integrabile nei comuni PC e nelle console dei video game.

La pipeline grafica, che dopo anni di realizzazione software si ritrovava ad essere in buona parte realizzata in hardware, allo scopo di velocizzare quanto più possibile la resa grafica. In questa direzione si sono mosse la prima e la seconda generazione di GPU, come si può vedere in Tabella 1.1.

I programmatori potevano accedere alle funzionalità fornite dall'hardware grafico della GPU tramite opportune interfacce 3D di programmazione (API). Le due maggiori interfacce di programmazione grafica sono OpenGL (sviluppata dalla Silicon Graphics) e DirectX (sviluppata dalla Microsoft).

La realizzazione della pipeline in hardware, da un lato ha permesso di migliorare notevolmente le performance delle applicazioni, e le interfacce API hanno semplificato lo sviluppo di applicazioni, fornendo ai program-

GEN.	PROG.	ANNI	CHIPSET	CARATTERISTICHE
I	NO	'90s	Voodoo di 3dfx TNT 2 nVidia Rage di ATI	single chip GPU. Completa gestione della fase di rasterizzazione da parte delle GPU, trasformazioni dei vertici ancora in CPU, set operazioni su pixel limitato.
II	NO	'99-'00	nVidia GeForce 256 GeForce2, Ati Radeon 7500, S3 Savage3D	La GPU ha completa gestione delle trasformazioni vertici 3D, estese operazioni matematiche per combinare texture, miglior configurabilità, ma non è programmabile.
III	SI'	'01-'02	nVidia GeForce 3 Ge-Force 4, Microsoft Xbox, ATI Radeon 8500	Piena vertex programmability, E' possibile specificare le computazioni effettuate sul singolo vertice. Non si ha programmabilità pixel.
IV	SI'	'02-'03	nVidia GeForce FX, (architettura CineFX), ATI Radeon 9700/9800	Piena vertex e pixel programmability.
V	SI'	>2006	nVidia GeForce 8800, ATI RV770 Fermi INTEL Larrabee Maxwell	Miscrosoft lancia DirectX 10 e nello Shader Model 4.0 introduce il Geometry Shader con il quale è possibile gestire intere geometrie. GPGPU General-Purpose GPU: 32 bit floating point throughout the pipeline,

Table 1.1: Generazioni schede grafiche

matori un livello di astrazione rispetto alle specifiche del sistema, e garantendo una maggiore portabilità del software.

Da un altro lato, però, la standardizzazione degli algoritmi di rendering non permetteva flessibilità negli effetti di resa grafica, limitandone qualità e realismo.

Che cosa ha portato gli ingegneri di CG ad introdurre la programmabilità nelle schede grafiche?

Sebbene le performance guadagnate impiegando hardware grafico dedicato per eseguire vari compiti su vertici e pixel erano eccellenti a confronto della sola gestione della CPU, i programmatori 3D persero un notevole controllo sull'immagine prodotta.

Mentre gli ingegneri di grafica hardware erano impegnati nel migliorare le performance grafiche, pacchetti software per la CG, quali Renderman (Pixar) stavano cambiando il look di film e spot televisivi con strabilianti effetti speciali generati al computer.

L'**off-line rendering**, a differenza del **real-time rendering** si avvaleva di standard general purpose CPU che costruivano frame by frame le animazioni impiegando giorni, settimane. Il vantaggio di usare general purpose CPU era la flessibilità del programmatore di poter usare la CPU per ogni effetto che si poteva immaginare.

L'off-line rendering system peccava di scarsa velocità, ma riusciva ad ottenere rendering di alta qualità e realismo. La flessibilità e generalità di offline rendering systems sono le caratteristiche chiave che erano state abbandonate nelle precedenti generazioni di hardware grafico 3D. In altre parole quello che era stato perso era la programmabilità.

Per poter realizzare real-time rendering con effetti paragonabili a quelli ottenibili già qualche anno prima tramite tecniche di off-line rendering, era necessario realizzare GPU in cui le diverse fasi della pipeline fossero programmabili. In questa direzione si sono mosse le successive tre generazioni di GPU , come si può vedere sempre in Tabella 1.1.

Si sono cominciate a inserire nelle GPU delle unità computazionali, dette Shader Units, che possono essere riprogrammate, dando così al programmatore la possibilità di modificare nel dettaglio l'algoritmo di resa impiegato. Le interfacce di programmazione API si sono di conseguenza evolute, definendo dei linguaggi per costruire Shader Programs, ovvero dei programmi che possono essere compilati a tempo di esecuzione per ottenere codice assembler specifico eseguibile sulle Shader Units.

I tre più diffusi linguaggi di shading di alto livello sono Cg di NVIDIA(per Windows e Linux) che supportano calcoli a 32bit floating point (no double), High Level Shader Language HLSL incluso nella libreria DirectX di

Microsoft, e GLSL, lo shading language proposto da SGI e supportato pienamente dalla versione 2.0 di OpenGL in su.

In questo modo dalla proria applicazione i programmatori di grafica possono accedere e controllare direttamente le funzionalità della scheda grafica.

Le CPU sono di loro natura general purpose, mentre le GPU sono progettate specificatamente per la grafica. Quindi, le GPU risultano molto più veloci nell'elaborazione di graphic task rispetto alle CPU. Le nuove GPU elaborano infatti decine di milioni di vertici per secondo e rasterizzano centinaia di milioni di frammenti per secondo. Nelle GPU i dati vengono memorizzati nel Vertex Buffer, nella Texture Memory o nel Frame Buffer.

L'ultimo passo nell'evoluzione dalla fixed-function pipeline, al rendering programmabile è l'introduzione di shaders unificati.

Nelle ultime schede grafiche della quinta generazione invece di utilizzare processori separati specifici per gestire esclusivamente vertex shaders, geometry shaders, e pixel shaders, si è introdotta un'architettura "unified shader" che fornisce una grande griglia di processori paralleli floating-point sufficientemente generali da gestire i carichi computazionali di tutti gli shader.

Queste nuove configurazioni hardware hanno proiettato le GPU oltre al loro ruolo nativo di elaborazione di graphic task trasformandole in vere e proprie entità "parallele" alla CPU, utilizzate come general purpose high performance computing, GPGPU.

1.7.2 GPGPU (General Purpose GPU)

L'evoluzione tecnologica dell'hardware grafico è giunto oggi alla realizzazione di dispositivi multithread, multicore, altamente parallelizzati, performanti e largamente programmabili; ne sono un esempio la GeForce GT X 280 della nVidia, la Radeon 4870 dell'ATI ed il progetto Larrabee di Intel. Questi prodotti sono stati progettati in modo tale da offrire la massima flessibilità possibile, a tal punto che il loro utilizzo si è svincolato anche dal contesto applicativo per cui inizialmente le GPU erano state create: la Computer Graphics.

Da qualche anno si è iniziato a parlare di GPGPU (General- Purpose

GPU), termine che in generale indica tutte quelle tecnologie utilizzate per accedere alla potenza di calcolo delle moderne periferiche grafiche, e utilizzarla per effettuare pesanti operazioni su dati matematici/scientifici. Queste tecnologie hanno proiettato le moderne GPU oltre al loro ruolo nativo di supporto grafico, trasformandole in vere e proprie entità "parallele" alla CPU , tanto da renderle loro concorrenti in diversi settori. L'adozione di una soluzione GPGPU , in generale, offre un grande vantaggio in termini di tempo di calcolo in qualsiasi ambito applicativo che richiede l'analisi e/o la trasformazione di grandi quantità di dati; si citano tra i vari settori ad esempio, la previsione della struttura delle proteine, previsioni meteo, analisi delle immagini medicali, astrofisica, reti neurali, ricerche oceanografiche, fisica delle particelle, chimica quantistica, astronomia, acustica, simulazioni elettromagnetiche, elaborazione di video in alta definizione, criptografia, rendering audio posizionale e in decine di altri settori.

Riprendendo il nostro parallelismo tra evoluzione tecnologica e processo grafico, ci potremmo chiedere che effetto ha e sta avendo la GPGPU sulla pipeline grafica. L'estrema flessibilità delle GPU moderne, accompagnata dalla GPGPU , permette oggi una totale libertà nel personalizzare qualsiasi fase della pipeline grafica "classica", riportando potenzialmente l'intero processo grafico ad essere realizzato via software.

Purtroppo il semplice utilizzo di GPU per high performance computing richiedeva al programmatore un notevole sforzo per organizzare i dati del problema come immagini, ("texture maps") ed implementare gli algoritmi con la filosofia della sintesi di immagini ("rendering passes"). Questo è stato superato dall'introduzione di nuove tecnologie di programmazione GPGPU , tra le quali le principali sono:

- OpenCL
  Libreria basata sul linguaggio di programmazione C99 che può esser eseguito su una molteplicità di piattaforme CPU e GPU. Lo standard è stato originariamente proposto dalla Apple, successivamente ratificato dalla stessa assieme le principali aziende del settore (Intel, NVIDIA , AMD), ed infine portato a compimento dal consorzio noprofit Khronos Group. Il nome OpenCL è stato pensato in analogia a OpenGL ed OpenAL , in quanto tutti standard industriali aperti, pur con diverse finalità.

- nVidia CUDA
  NVIDIA, rilasciò nel 2006 una prima release di CUDA, un SDK che consentiva di programmare le GPU in modo da eseguire algoritmi di calcolo general purpose in parallelo. CUDA è una tecnologia sviluppata per consentire lo sviluppo di applicazioni che possano sfruttare la potenza elaborativa delle moderne GPU nVidia. CUDA è un'architettura parallela general purpose che sfrutta il motore di elaborazione parallela delle NVIDIA GPU per risolvere problemi computazionali anche molto complessi in una frazione del tempo richiesto dalla CPU. CUDA include il CUDA Instruction Set Architecture (ISA) e il motore parallelo di calcolo nella GPU. Per programmare l'archietttura CUDATM gli sviluppatori possono usare il linguaggio C, per esempio, che viene ottimizzato per ottenere ottime performance sul processore CUDATM. Sono supportate le librerie numeriche standard per algoritmi come FFT (Fast Fourier Transform) e BLAS (Basic Linear Algebra Subroutines). Inoltre il driver di CUDA interagisce con API come OpenGL e DirectX e sistemi operativi sia Linux 32/64-bit sia Windows XP 32/64-bit.
- ATI's CTM
  Tecnologia analoga a CUDA per ATI.

Queste soluzioni nascono per gestire in modo più o meno trasparente il parallelismo a livello di dati, ma non offrono alcuno strumento che guidi la realizzazione di sistemi grafici basati sull'esecuzione parallela di task, distribuiti tra CPU e GPU. Per sfruttare a pieno le tecniche GPGPU nella realizzazione dei futuri sistemi grafici, sarà necessario avere dei tools che guidino nella gestione del parallelismo a tutti i suoi livelli. La sfida sarà dunque quella di creare motori grafici basati su task, in grado di usare in maniera efficiente l'eterogeneità delle risorse computazionali sottostanti (CPU e/o GPU). Un'ultima previsione riguarda lo sviluppo tecnologico delle schede video. L'andamento del mercato e l'evoluzione tecnologica sta avvicinando sempre di più l'architettura delle GPU a quella delle CPU , e viceversa. Le aziende produttrici di processori, infatti, stanno investendo molto nell'incrementare il numero dei core all'interno dei loro prodotti, mentre le aziende produttrici di dispositivi grafici, proprio sulla spinta della GPGPU , stanno aumentando la logica dedicata al controllo. Questi due fattori stanno determinando una sorta di convergenza, omogeneizzazione, tra CPU e GPU. Questo spingerà sempre

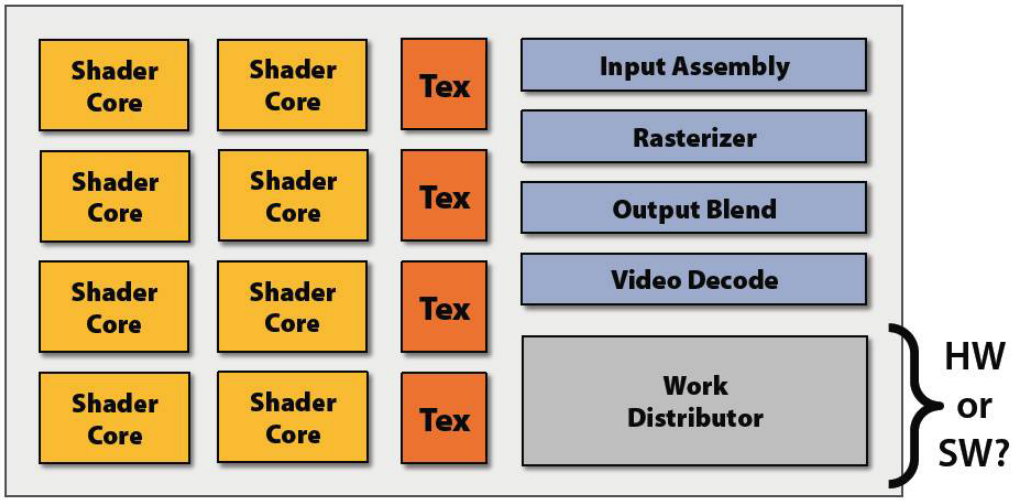


Figure 1.24: Elementi architetturali base delle GPU

di più a realizzare il processo di resa grafica attraverso tasks di natura software. I tasks saranno assegnati alle sottostanti risorse computazionali per mezzo di un sistema di supporto, che sarà facilitato nel suo compito proprio dall'omogeneizzazione della natura delle risorse computazionali sottostanti.

1.7.3 Architettura delle schede grafiche

Sulla base del lavoro di Fatahalian K. (SIGGRAPH 2009), si cercherà di spiegare brevemente qual è in sostanza l'architettura che c'è dietro alle moderne GPU, e come sia stato possibile realizzare dispositivi i cui livelli di performances hanno raggiunto ormai l'ordine dei TeraFlops (Floating Point Operations Per Second), grazie al loro elevato grado di parallelismo. Questa analisi aiuterà anche a capire come sfruttare le caratteristiche delle schede grafiche, per sviluppare applicazioni GPGPU. Innanzitutto è necessario spiegare quali sono gli elementi base presenti nelle moderne GPU. Come mostrato in Figura 1.24, oggi le GPU contengono un pool di processing cores programmabili, che servono ad elaborare i Shader programs. Tali cores sono assistiti da diverse unità di texturing, che si occupano dell'esecuzione di istruzioni per il texture sampling. Sono inoltre presenti diverse unità di elaborazione che forniscono una collezione

di funzioni hardware per la realizzazione di specifici tasks grafici (es. rasterizzazione). Infine l'ultimo componente che troviamo è il "Work Distributor", che gestisce il flusso di lavoro e lo scheduling dei diversi tasks durante l'esecuzione di un'applicazione, sia essa una pipeline grafica o un programma GPGPU. Proprio la riduzione in termini relativi dello spazio occupato da quest'ultimo componente, a favore di un maggior numero di processing cores programmabili, è alla base della GPGPU. La tendenza tecnologica che si osserva in questi anni, come già spiegato, è quella di preferire una maggiore flessibilità nell'utilizzo delle capacità di calcolo delle GPU , rispetto ad una realizzazione hardware altamente ottimizzata di un determinato numero di funzionalità fisse.

Ci si potrebbe chiedere come mai il numero dei processing cores presenti nelle GPU è oggi molto più elevato rispetto a quelli nelle CPU. La risposta è semplice: i primi svolgono algoritmi meno complicati rispetto a quelli svolti dai secondi e non cercano di prevenire situazioni di stallo, come ad esempio i branch. Generalizzando, gli Shader Programs sono algoritmi abbastanza semplici e lineari, che non richiedono una parte logica di controllo per ottimizzare la loro esecuzione, in quanto questa solitamente è sequenziale. Il vero problema degli Shader Programs è la quantità di dati su cui essi devono lavorare.

Partendo da una generica architettura "CPU -style", per abbassare i tempi d'esecuzione di questo tipo di applicazioni, una prima idea è quella di eliminare dal dispositivo tutta la logica superflua, che non comporta sufficienti benefici durante l'elaborazione degli Shader Programs. In questo modo lo spazio liberato può essere utilizzato per replicare lo "slim-core" così ottenuto.

Un'ulteriore ottimizzazione può essere realizzata osservando semplicemente che la sequenza di operazioni svolte per ciascun fragments è la medesima. Possiamo quindi condividere le fasi di Fetch e Decode adottando un'architettura di tipo SIMD (Single Instruction Multiple Data); in questo modo il costo e la complessità della logica per la gestione di ogni istruzione sono ammortizzate su più ALUs con il beneficio ulteriore di risparmiare altro spazio. Ovviamente le moderne CPU eseguono contemporaneamente diversi tipi di programmi; questo perchè ogni singolo core ha un'unità Fetch/Decode dedicata, che gli permette di elaborare un'istruzione indipendentemente da quali sono elaborate nello stesso is-

tante negli altri cores.

Avendo eliminato dai cores la logica che aiutava ad evitare gli stalli, ogni instruction stream può subire un blocco a seguito di istruzioni caratterizzate da un'elevata latenza, o la cui esecuzione deve essere posticipata a causa della dipendenza da risultati di altre operazioni, che devono ancora essere svolte.

Per nascondere queste situazioni, all'interno delle GPU in ogni cores sono processati diversi gruppi di fragments, in modo tale che se un gruppo subisce uno stallo si esegue un cambio di contesto e si prosegue con un altro. Per realizzare questa tecnica, che è chiamata "interleave processing", è necessario fornire ogni cores di un maggior numero di registri per mantenere registrati diversi contesti d'esecuzione.

Se il numero di contesti è sufficientemente elevato, si riescono ad ottenere sistemi in cui le ALUs di ogni singolo cores non sono mai idle. Questo segna un'ulteriore differenza tra GPU e CPU ; le prime usano la memoria sul chip principalmente per i contesti d'esecuzione, mentre le seconde la utilizzano per la memorizzazione dei dati (data cache). Per quanto riguarda la memorizzazione dei dati analogamente alle CPU all'interno della scheda grafica sono presenti diversi livelli di memoria.

Solitamente esiste una global memory accessibile da tutti i cores della GPU, che a loro volta contengono una local memory visibile solo dal singolo core e utilizzata per il caching dei dati onchip. In realtà, normalmente la memoria nelle schede grafiche è suddivisa a livello logico-fisico in diversi moduli, sulla base dell'uso che ne viene fatto all'interno della pipeline grafica, e si possono individuare tre tipi di memorie possibili: texture memory, frame buffer , vertex buffer.

Questa suddivisione permette un'ottimizzazione nell'esecuzione della pipeline grafica, poichè ciascun tipo di memoria è realizzata sulla base dell'utilizzo delle informazioni che contiene.

Riassumendo, l'elevata capacità di calcolo parallelo delle GPU è dovuta essenzialmente a tre caratteristiche dell'architettura:

- l'elevato numero di cores presenti al loro interno, reso possibile dall'eliminazione di una parte della logica di controllo presente invece nei cores delle CPU ;
- ogni cores condivide le fasi di Fetch e Decode tra più ALUs, che

li rende in grado di eseguire un'istruzione su più dati contemporaneamente;
- l'interleave processing permette di nascondere situazioni di stallo ed ottimizza l'uso delle diverse unità computazionali.

Sfruttando queste caratteristiche e beneficiando dello sviluppo tecnologico dell'elettronica, si hanno dispositivi che hanno raggiunto prestazioni nell'ordine del TeraFlops.

1.8 Rendering Graphics pipeline

La GPU non ha architettura Von Neumann bensì a pipeline poichè progettata per soddisfare la graphics rendering pipeline.

La pipeline grafica gestisce il rendering per una resa più efficiente di modelli tridimensionali ed una loro visualizzazione a video. La principale funzione della pipeline è di generare (rendere) un'immagine bidimensionale da memorizzare nel frame buffer. In input alla pipeline vi è un insieme di primitive geometriche 3D, camera virtuale, risorse luminose, texture ed altro.

I vari stadi della pipeline possono essere globalmente accorpati in tre principali stadi: application, geometry e rasterizer. Nello **stadio application**, lo sviluppatore ha pieno controllo sul suo programma grafico, poichè è sempre eseguito in software. Alla fine di questo stadio, la geometria che deve essere resa diventa l'input del prossimo stadio della rendering pipeline.

Lo **stadio geometry**, è responsabile per le operazioni Per-vertex che accettano primitive geometriche (descritte da punti, segmenti, poligoni) come input e forniscono vertici trasformati (rispetto al volume di vista) con associati valori di colore, profondità, e a volte valori di texture.

Nello stadio successivo, **stadio rasterizzazione** o Per-pixel, i dati geometrici sono resi per produrre un array di frammenti corrispondenti a descrizioni 2D della geometria. L'output verrà salvato nel frame buffer. Operazioni di copia pixel da e nel frame buffer e memoria texture. Quindi sono elaborate operazioni su singoli frammenti (Per-Fragment Operations) prima che questi alterino definitivamente il frame buffer.

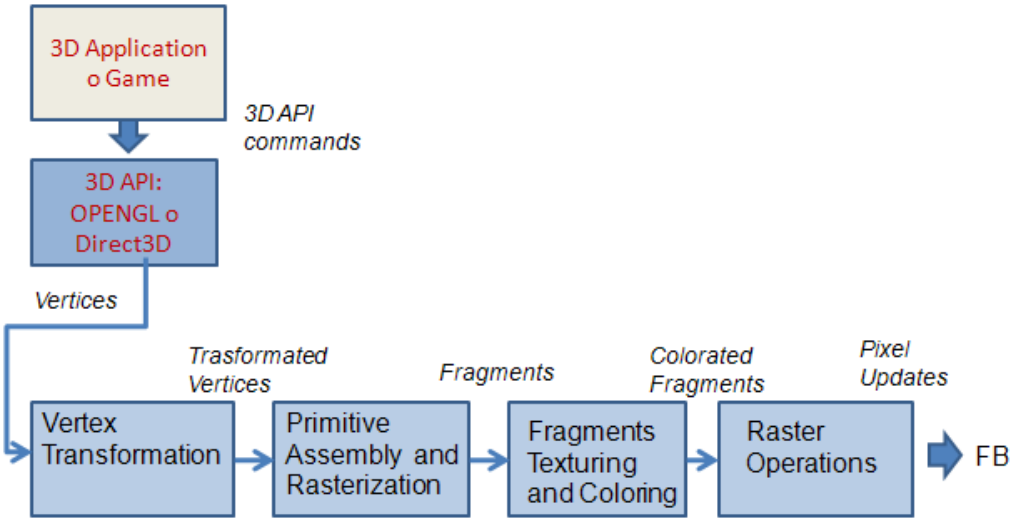


Figure 1.25: Fixed-function graphics pipeline

Gli stadi geometry e rasterizer sono realizzati sulla scheda grafica, e implementati in parte in hardware.

In una fixed-function pipeline, ovvero in una pipeline non programmabile, il programmatore interagisce con la pipeline via interfaccia software standard quale per esempio OpenGL o DirectX.

Una fixed-function pipeline della 3D API (OpenGL o DirectX) implementa gli algoritmi di rendering direttamente nella scheda grafica, memorizza i dati dei vertici nella onboard video memory, evitando il passaggio dei dati attraverso il system bus. Una convenzionale pipeline grafica elabora una moltitudine di vertici, primitive geometriche, e fragment in modo pipeline operando in modalità SIMD (Single Instruction Multiple Data). Il rendering è un processo ripetitivo che si presta ad essere elaborato in parallelo, per questo su una scheda grafica si possono trovare diverse rendering pipeline in parallelo. Questo permette di elaborare diversi pixel per ciclo di clock anzichè un solo pixel.

In figura 1.25 è mostrato lo schema di una fixed-function rendering pipeline. Mentre la figura 1.26 rappresenta una rendering pipeline programmabile. Questo significa libertà da parte del programmatore di implementare vari tipi di algoritmi come differenti modelli di luminosità, o generazione di

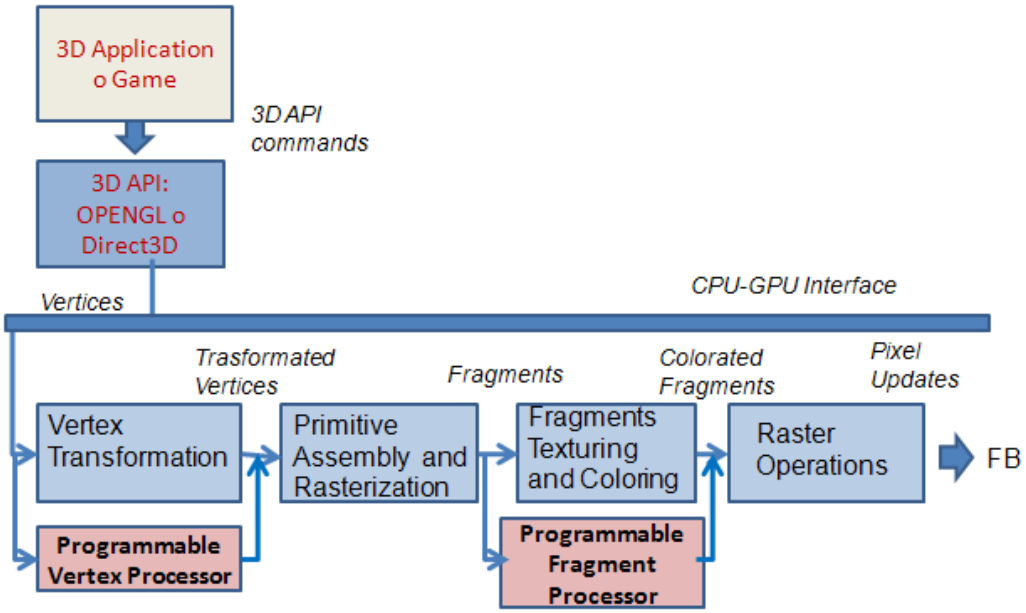


Figure 1.26: Graphics pipeline programmabile

coordinate texture, trasformazioni non lineari. Si hanno due distinte unità programmabili: vertex processor e fragment processor. Queste rappresentano rispettivamente la manipolazione della geometria e la rasterizzazione pixel. Dall'ultima generazione di unified shader, tra il vertex processor e l'unità hardware di Primitive Assembly and Rasterization si può trovare la Tessellation Control Shader e Geometry Shader.

1.9 Introduzione agli Shaders Program

Nelle moderne GPU ciascuno dei due macro–stadi logici della pipeline può essere interamente ridefinito fornendo uno shader compilato; si parla di Vertex Shader quando questo va a specificare le operazioni a livello di vertici da eseguire durante il Geometry Stage, viceversa di Fragment Shader (o Pixel Shader) se va a definire il Rasterization Stage. Nei paragrafi 1.9.1 e 1.9.2 si approfondisce questo argomento.

Di recente sono state proposte soluzioni hardware che comprendono un' ulteriore tipologia di shader, quella dei Geometry Shader, eseguiti du-

rante lo stadio Geometry Stage; questi permettono di specificare le modalità con cui i poligoni vengono costruiti a partire dai vertici, ad esempio per duplicare automaticamente le primitive o definirne di nuove. Tra le possibili applicazioni di questa nuova tecnologia ci sono l'esecuzione su Graphics Processing Unit di algoritmi di Shadow Volume Generation, per disegnare le ombre tramite opportuni volumi d'ombra proiettati da ciascun poligono illuminato da una fonte di luce, e di tassellazione, per approssimare primitive geometriche curve attraverso mesh poligonali.

L'utilità pratica dei Geometry Shader non sta tanto nella personalizzazione della pipeline, dato che le stesse operazioni possono essere eseguite nell' Application Stage, quanto piuttosto nel fatto di spostare una parte dei calcoli richiesti per generare la geometria, dall'unità centrale al dispositivo di rendering, permettendo un maggiore bilanciamento del carico computazionale sulla pipeline, al fine di massimizzare lo throughput.

1.9.1 Vertex Shaders

Un Vertex Shader è un programma che prende in ingresso una serie di informazioni su un vertice della scena, e da in uscita la posizione di questo nel sistema di coordinate di vista, e, opzionalmente, il colore, il vettore normale alla superficie e le coordinate texture. La Shader Unit quindi può modificare proceduralmente i dati in uscita variando le matrici di trasformazione, gli attributi relativi alle fonti di luce che illuminano la scena, o altri parametri necessari a specificare le operazioni di trasformazione e illuminazione del Geometry Stage.

I primi Shading Language, presenti in DirectX 8.1 e OpenGL 1.5 tramite estensioni, prevedevano un massimo di 128 istruzioni di tipo matematico–vettoriale; non era possibile inserire istruzioni di controllo come salti condizionali e cicli, dunque era garantito che ogni vertice impegnasse per un tempo fisso la Shader Unit.

Col passare degli anni le specifiche si sono evolute, parallelamente allo svilupparsi delle soluzioni hardware, supportando via via nuove istruzioni matematiche e di controllo e incrementando il numero di registri disponibili. Qualora si renda necessario eseguire un Vertex Shader su un dispositivo grafico che non supporta alcune delle funzionalità richieste, le API

di programmazione 3D si occupano di emulare lo shader eseguendo sulla CPU le operazioni di trasformazione specificate, ed eseguendo poi il solo passo di Rasterization sulla GPU.

I Vertex Shader aprono la strada per una vasta gamma di effetti, che prima dovevano essere elaborati nel livello applicativo della pipeline grafica, come deformazioni della geometria (oggetti semi–rigidi spostati dal vento, movimento dei fluidi, oggetti flessibili) o distorsioni che simulano effetti ottici come la rifrazione della luce attraverso diversi materiali. Inoltre è possibile implementare un modello di illuminazione diverso da quello di default del Geometry Stage, andando a calcolare opportunamente il colore del vertice in funzione degli attributi della superficie e della sorgente di luce.

1.9.2 Fragment Shaders

Lo stadio Rasterization Stage della pipeline grafica ha il compito di disegnare uno ad uno, sul Frame Buffer o meglio sul Color Buffer che costituisce l'immagine renderizzata della scena, i poligoni proiettati nel passo precedente. Per fare questo ciascuno viene scomposto in una serie di frammenti che ne compongono l'immagine, uno per ogni pixel occupato dal poligono, e ne viene calcolato il colore in funzione dei parametri di illuminazione calcolati durante il Geometry Stage ed, eventualmente, del valore reperito da una texture map.

Nelle moderne GPU invece è possibile che ad uno stesso oggetto siano associate più texture, ad esempio nel caso dell'environment mapping una texture è quella che simula la riflessione dell'ambiente circostante sull'oggetto, mentre un' altra potrebbe memorizzare il colore della superficie. Dunque questo stadio è suddiviso in una serie di Texture Stages, ciascuno dei quali legge un valore da una texture map, e i risultati vengono mescolati attraverso opportune funzioni di Texture Blending, che è possibile configurare a seconda dell'effetto desiderato.

I Fragment Shader rendono il procedimento molto più flessibile, dando la possibilità di definire dei veri e propri algoritmi per il calcolo del colore di ciascun frammento che viene renderizzato. Questi programmi hanno in ingresso dallo stadio precedente i parametri di illuminazione e le coordi-

nate texture opportunamente interpolati sui vari frammenti del poligono oltre ad una serie di costanti globali, e devono dare in uscita il colore finale[1] che verrà inserito nel Color Buffer.

I primi modelli di Fragment Shader proposti supportavano solo un set limitato di istruzioni che comprendeva operazioni di lettura da texture maps di vario tipo ed alcune operazioni aritmetiche di base. Esattamente come è accaduto per i Vertex Shader, anche questi si sono sviluppati rapidamente negli ultimi anni, andando ad includere molte istruzioni più articolate e parametri di ingresso aggiuntivi, come la posizione del frammento nell'immagine e il valore di fog distance[2].

1.9.3 OpenGL Shading Language

Lo standard di specifiche OpenGL ha cominciato a supportare la programmazione di shader a partire dalla versione 1.3, tramite le estensioni ARB_vertex_program e ARB_fragment_program, che permettono di esporre le funzionalità di vertex e fragment shading messe a disposizione dal dispositivo di rendering, e di programmarle tramite un linguaggio assembler di basso livello.

Nella versione 1.5 è stata introdotta un' ulteriore estensione [3] nella quale si definisce un linguaggio di shading di alto livello denominato GL Shading Language (d'ora in avanti GLSL). I programmi scritti in questo linguaggio vengono inviati alla Graphics Processing Unit, dove un compilatore GLSL li trasforma in codice hardware–specific per le proprie Shader Units. Questo approccio presenta diversi vantaggi, sia in termini di portabilità degli shader, che possono essere eseguiti su qualunque dispositivo che soddisfi le specifiche OpenGL, sia di semplicità di scrittura, visto che è il compilatore ad occuparsi di interpretare il codice di alto livello, e di produrre le opportune ottimizzazioni.

GLSL è un linguaggio simile a C, con la possibilità di definire funzioni,

[1] E' possibile influire su altri parametri di uscita, come la profondità, usata più avanti nella pipeline per stabilire quali frammenti sono visibili, e quali invece sono coperti da altri poligoni che si trovano più vicini all'osservatore.

[2] Questo parametro è usato negli algoritmi di shading che attenuano il colore degli oggetti più lontani dall'osservatore, per migliorare la percezione della profondità.

[3] Inclusa poi nativamente nella versione OpenGL 2.0.

variabili, salti condizionali e cicli.

I tipi base fondamentali sono:

float, int, bool
: Per numeri floating point, interi e boolean.

vec2, vec3, vec4
: Per vettori fino a quattro componenti.

mat2, mat3, mat4
: Per matrici fino a quattro colonne.

sampler1D, sampler2D, sampler3D
: Per identificativi di texture maps fino a tre dimensioni.

Per permettere il passaggio di valori dal livello applicativo agli shader sotto forma di costanti, e dai Vertex Shader ai Fragment Shader, sono disponibili quattro qualificatori di tipo, che premessi alla definizione di una variabile globale, le danno uno specifico ruolo

uniform
: La variabile viene usata per passare un valore dall'applicazione allo shader ed assume un valore costante su tutti i vertici e i frammenti di ciascun poligono[4].

attribute
: La variabile rappresenta un attributo definito a livello di vertice.

varying
: La variabile è usata per passare al Fragment Shader un attributo interpolato tra i vertici del poligono[5].

const
: Si tratta di una costante definita a tempo di compilazione.

Sono poi disponibili alcune funzioni built–in ed operatori, di tipo matematico, per operazioni su scalari, vettori o matrici, istruzioni di lettura dalle texture map, ed una serie di variabili globali che rappresentano i parametri di input e di output del programma.

Per uniformità lo stesso linguaggio è usato per scrivere Vertex Shader e Fragment Shader; l'unica differenza sta nel tipo di operazioni che è

[4] Si possono passare in questo modo sia valori costanti per tutta la scena, sia variabili definite a livello di poligono.

[5] Come avviene per esempio per le coordinate texture ed i parametri di illuminazione passati al Rasterization Stage.

possibile fare nei due casi, e quali di queste sono obbligatorie. Ad esempio gl_FragColor, una variabile di tipo vec4 che rappresenta il colore su schermo di un frammento, non è disponibile nei Vertex Shader, mentre è indispensabile che un Fragment Shader scriva all'interno di questa variabile nel corso della propria esecuzione; analogamente la attribute vec4 gl_Position, che contiene la posizione in spazio oggetto di un vertice, è disponibile solo per i Vertex Shader.

Chapter 2

Geometria per la Computer Graphics

In questo capitolo presentiamo alcuni elementi di algebra lineare e geometria che sono alla base di molte elaborazioni in computer graphics. Le trasformazioni geometriche che rappresentano un fondamentale tool in CG sono utilizzate nella fase di modellazione, in animazione e in tutta la fase di rendering della pipeline grafica.

2.1 Sistemi di coordinate

Il sistema di coordinate cartesiano può assumere due forme: destrorso (right-handed system), e sinistrorso (left-handed system), come mostrato

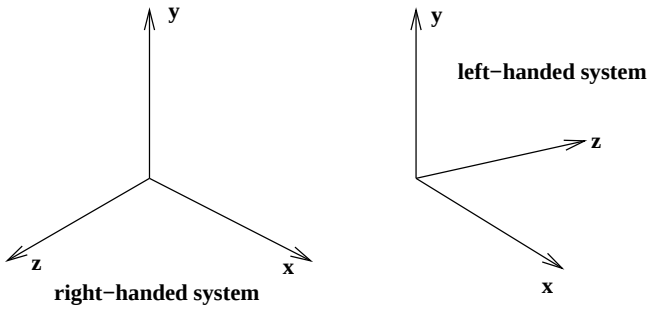


Figure 2.1: Sistema di coordinate cartesiano in CG: destrorso e sinistrorso

in Fig. 2.1. In generale, ci baseremo su di un sistema destrorso nella modellazione e sinistrorso nel rendering.

La pipeline grafica rappresenta i passi attraverso i quali una scena 3D è modellata e poi resa, ovvero trasformata in immagine memorizzabile nel frame buffer. Questi passaggi che il modello 3D subisce possono essere visti come successive trasformazioni sistematiche tra sistemi di coordinate: a partire dallo Spazio Mondo fino allo Spazio Dispositivo.

- **Object Space** - Sistema di coordinate locale in cui un singolo oggetto è stato creato. Dopo la sua modellazione l'oggetto sarà poi trasformato, duplicato,..etc. e spostato nel sistema di coordinate globale, che è comunemente chiamato world space.
- **World Space** - Il sistema di coordinate in cui è modellata la scena
- **View Space** - Il sistema di coordinate in cui è definita la camera virtuale
- **Normalized Coord. System (o Image Space)** - Sistema di coordinate contenente il Canonical View Volume. Per permettere calcoli di clipping questo spazio è usualmente definito da coordinate $x, y, z \in [-1, 1]$. Punti nello spazio immagine sono poi proiettati nello screen space attraverso la loro proiezione ortogonale nel piano immagine
- **Screen Space** - definito da una finestra $-1 \leq x, y \leq 1, x, y \in R$ (quadrato) nel piano xy.
- **Device space** - sistema di coordinate 2D nel quale i punti sono contraddistinti da coordinate (x, y) intere che rappresentano la loro posizione pixel all'interno della finestra grafica sul monitor. L' origine è in alto a sinistra (0,0), con valori x che si incrementano verso destra e valori y che si incrementano verso il basso.

2.2 Scalari, punti e vettori

Iniziamo con il considerare le entità base utilizzate nella CG: **scalari, punti e vettori**. I punti definiscono gli oggetti, i vettori ci servono per definire gli spostamenti, mentre gli scalari definiscono l'entità degli spostamenti.

Uno **scalare** è un valore numerico $(\in \mathbb{R})$ che specifica una quantità.

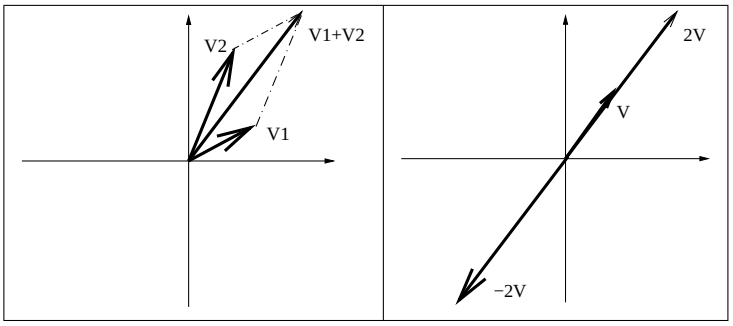


Figure 2.2: (sinistra) Somma di vettori; (destra) moltiplicazione per uno scalare negativo

Un **punto** rappresenta una posizione nello spazio. Un punto 'matematico' non ha nè dimensione nè forma;

Un **vettore** è definito da un *orientamento* (direzione e verso) e un *modulo* . Un vettore non ha una posizione fissa nello spazio, e si può definire come differenza tra due punti.

Punti, scalari e vettori sono membri di spazi matematici.

2.3 Spazi vettoriali

Uno **spazio vettoriale (o lineare)** contiene due tipi di entità: vettori e scalari. Oltre a combinare fra loro scalari sono definite le operazioni di moltiplicazione scalare-vettore, e somma vettore-vettore. Esempi di spazi lineari sono lo spazio dei vettori in $\mathbb{R}^2/\mathbb{R}^3$, lo spazio dei polinomi, e lo spazio delle matrici.

Esempio. Nello spazio lineare dei vettori in $\mathbb{R}^2$, un vettore è rappresentato da una coppia (x, y) di scalari. Due vettori $v_1 = (x_1, y_1)$ e $v_2 = (x_2, y_2)$ possono essere sommati utilizzando la regola del parallelogramma illustrata in Fig 2.2(sinistra) dando come risultato un vettore di componenti $v = (x_1 + x_2, y_1 + y_2)$. Un vettore v può essere moltiplicato per uno scalare α e il risultato è un vettore con direzione quella del vettore v e componenti $(\alpha x_1, \alpha x_2)$ (Fig.2.2(destra)).

Definizione: Combinazione lineare *Dato un insieme di n vettori* ***linear-***

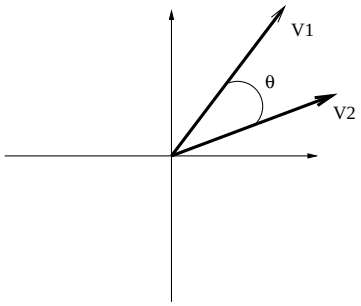


Figure 2.3: Prodotto scalare in $\mathbb{R}^2$ tra i vettori v_1 e v_2.

mente indipendenti *$v_1, v_2, .., v_n$ di uno spazio lineare V, $dim(V) = n$, ogni vettore $v \in V$ è unicamente determinato dalla combinazione lineare di questi n vettori*

$$v = \alpha_1 v_1 + \alpha_2 v_2 + ... + \alpha_n v_n$$

per certi scalari $\alpha_1, \alpha_2, ..., \alpha_n$ coefficienti di v rispetto alla base $v_1, v_2, .., v_n$.

Nello spazio dei vettori in $\mathbb{R}^3$, un vettore $v \in \mathbb{R}^3$ può essere rappresentato come combinazione lineare di tre vettori base linearmente indipendenti $v_1 = (1, 0, 0), v_2 = (0, 1, 0), v_3 = (0, 0, 1)$:

$$v = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3,$$

dove gli scalari $\alpha_1, \alpha_2, \alpha_3$ sono le coordinate di v.

Ricordiamo due semplici ed utili operazioni sui vettori in $\mathbb{R}^2, \mathbb{R}^3$: il prodotto scalare o dot product e il prodotto vettoriale o cross product.

Definizione: spazio lineare con prodotto interno (o prodotto scalare) *In uno spazio lineare si può definire una funzione prodotto scalare $< \cdot, \cdot >: \mathbb{R}^n \rightarrow \mathbb{R}$, che associa ad ogni coppia ordinata di vettori $(x, y), x, y \in \mathbb{R}^n$, di componenti $x = (x_1, x_2, .., x_n)$, $y = (y_1, y_2, .., y_n)$, lo scalare (positivo o nullo):*

$$x \cdot y =< x, y >= x^T y = \sum_{i=1}^{n} x_i y_i.$$

Nello spazio vettoriale dei vettori in $\mathbb{R}^3$, questo ci consente di introdurre una misura della lunghezza di un vettore $x = (x_1, x_2, x_3)$, o **NORMA**

EUCLIDEA:

$$\|x\|_2 = \sqrt{< x, x >} = \sqrt{x^T x} = \sqrt{x_1^2 + x_2^2 + x_3^2}.$$

Un vettore di lunghezza euclidea 1 è chiamato vettore unitario, normalizzare un vettore significa renderlo unitario: $\frac{x}{\|x\|_2}$.

Siano $x = (x_1, x_2, x_3)$ e $y = (y_1, y_2, y_3)$ due vettori in $\mathbb{R}^3$. Il prodotto scalare è definito da

$$x \cdot y = \sum_{i=1}^{3} x_i * y_i$$

Se $x \cdot y = 0$ allora x ed y sono tra loro ortogonali.
Il calcolo dell'angolo θ compreso tra i due vettori:

$$\cos(\theta) = \frac{x \cdot y}{\|x\| \|y\|}.$$

Il vettore w proiezione ortogonale di x su y é dato da:

$$w = \|x\| \cos\theta \frac{y}{\|y\|} = (x \cdot y) \frac{y}{\|y\|},$$

e ha lunghezza

$$\|w\| = \frac{x \cdot y}{\|y\|}.$$

Dati due vettori non paralleli $x, y \in \mathbb{R}^3$, il loro **prodotto vettoriale** (o cross product) $x \times y$ è un vettore a loro ortogonale dato da:

$$n = x \times y = (x_2 y_3 - x_3 y_2, x_3 y_1 - x_1 y_3, x_1 y_2 - x_2 y_1)$$

orientato nella direzione stabilita dalla regola della mano destra. Inoltre la terna di vettori (x, y, n) forma un sistema di coordinate destrorso. Per determinare tre vettori mutuamente ortogonali (u, v, w) dati u e v, si calcola $n = u \times v$, quindi $w = u \times n$.

Dati due vettori $x, y \in \mathbb{R}^3$, sia θ l'angolo tra essi, allora:

$$\|\sin\theta\| = \frac{x \times y}{\|x\| \|y\|}.$$

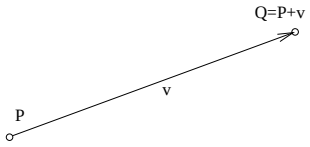


Figure 2.4: Operazione somma punto vettore

Il prodotto vettoriale si utilizza inoltre nel calcolo dell'area del parallelogramma formato dai vettori x e y, che è data da

$$\|x \times y\|.$$

Quindi il triangolo di lati x e y avrà area $\frac{1}{2}\|x \times y\|$.

2.4 Spazi Affini

In uno spazio lineare manca il concetto di posizione e distanza. Questa difficoltà viene superata introducendo gli spazi affini ovvero un'estensione dello spazio lineare che include un'ulteriore tipo di entità: i punti. Oltre a somma e moltiplicazione tra scalari, somma tra vettori, e moltiplicazione scalare-vettore, si introduce una nuova operazione: la **somma punto-vettore**

$$Q = P + v$$

che definisce un nuovo punto, (o meglio) la **sottrazione punto-punto** che genera un vettore a partire da due punti (Fig. 2.4).

In uno spazio affine non sono definite le operazioni di somma tra punti e moltiplicazione tra scalare e punto, esiste però la somma o combinazione affine.

Una combinazione affine è una combinazione lineare con i coefficienti che sommano a 1.

Definizione: Combinazioni affini e coordinate baricentriche *Dato un insieme di punti $P_0, P_1, .., P_n$, consideriamo tutte le combinazioni affini su questi*

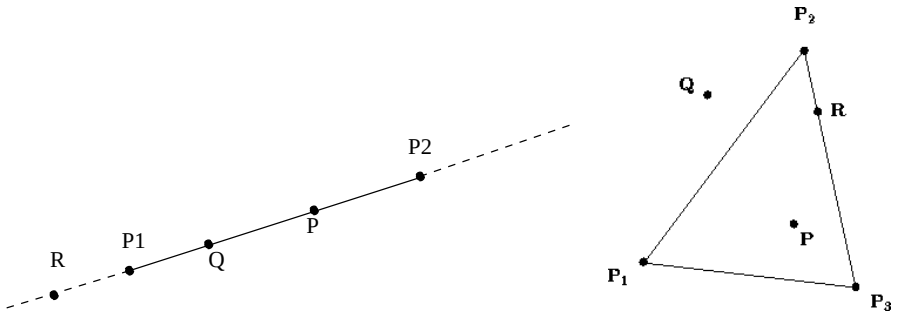


Figure 2.5: Punti su una retta definita da P_1 e P_2 (sinistra); Punti sul piano definito da P_1, P_2 e P_3 (destra).

punti, cioè tutti i punti P ottenuti da:

$$P = \alpha_0 P_0 + \alpha_1 P_1 + ... + \alpha_n P_n$$

per certi scalari $\alpha_0 + \alpha_1 + ... + \alpha_n = 1$. *Questo insieme di punti forma uno spazio affine. Le coordinate* $(\alpha_0, \alpha_1, ..., \alpha_n)$ *sono dette* ***coordinate baricentriche*** *del punto P nello spazio affine.*

Le coordinate baricentriche consentono di definire quindi un sistema di coordinate locali rispetto ad un insieme di punti.

Combinazioni affini con i coefficienti limitati nell'intervallo $[0, 1]$ si dicono combinazioni convesse.

Definizione: Combinazioni convesse *Dato un insieme di punti* $P_0, P_1, .., P_n$, *una combinazione convessa di questi punti con coefficienti* $\alpha_0, \alpha_1, ..., \alpha_n$ *è definita da:*

$$P = \alpha_0 P_0 + \alpha_1 P_1 + ... + \alpha_n P_n$$

dove

$$\alpha_0 + \alpha_1 + ... + \alpha_n = 1 \quad e \quad 0 \leq \alpha_i \leq 1.$$

Il punto P è chiamato combinazione convessa dei punti $P_0, P_1, .., P_n$.

Esempio Consideriamo le coordinate baricentriche α_1, α_2, di un punto P appartenente ad una retta passante per i punti P_1 e P_2:

$$P = \alpha_1 P_1 + \alpha_2 P_2.$$

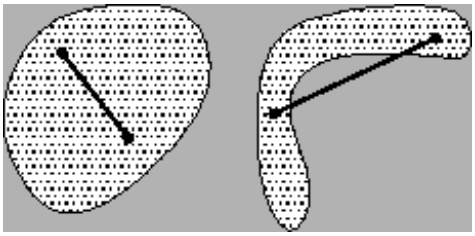


Figure 2.6: Insieme convesso (sinistra), Insieme non convesso (destra)

Se $0 \leq \alpha_1, \alpha_2 \leq 1$ allora P appartiene al segmento che unisce P_1 e P_2.

Nell'esempio in Fig. 2.5(sinistra) si ha: $P : \alpha_1 = 1/3, \alpha_2 = 2/3$, $Q : \alpha_1 = 3/4, \alpha_2 = 1/4$ e $R : \alpha_1 = 4/3, \alpha_2 = -1/3$.

Sia il punto $Q = \frac{1}{3}P_0 + \frac{2}{3}P_1$ che il punto $R = -\frac{2}{5}P_0 + \frac{7}{5}P_1$ sono combinazioni affini di P_0 e P_1, ma solo Q è combinazione convessa.

Esempio Consideriamo le coordinate baricentriche $(\alpha_1, \alpha_2, \alpha_3)$, tali che $\alpha_1 + \alpha_2 + \alpha_3 = 1$, di un punto P su un piano π definito dai punti P_1,P_2 e P_3

$$P = \alpha_1 P_1 + \alpha_2 P_2 + \alpha_3 P_3.$$

Se $0 \leq \alpha_1, \alpha_2, \alpha_3 \leq 1$ allora $P \in \pi$ appartiene al triangolo di vertici P_1, P_2, P_3.

Nell'esempio in Fig. 2.5(destra) si ha :
$P : \alpha_1 = \alpha_2 = 1/4, \alpha_3 = 1/2$,
$Q : \alpha_1 = 1/2, \alpha_2 = 3/4, \alpha_3 = -1/4$,
$R : \alpha_1 = 0, \alpha_2 = 3/4, \alpha_3 = 1/4$.

Un insieme di punti $P_0, P_1, .., P_n$ è detto **insieme convesso** se ogni possibile combinazione affine di questi punti è anch'essa nell'insieme.

Semplicemente, osserviamo la Fig.2.6. Se si può tracciare un segmento di linea tra due punti dell'insieme che non è contenuto completamente dentro l'insieme, l'insieme non è convesso.

Definizione: Convex hull (guscio convesso) *Dato un insieme di punti $P_0, P_1, .., P_n$. L'insieme dei punti P che possono essere rappresentati dalla combinazione convessa di $P_0, P_1, .., P_n$ forma il guscio convesso (convex hull) dell'insieme*

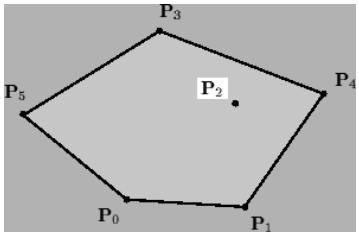


Figure 2.7: Convex hull

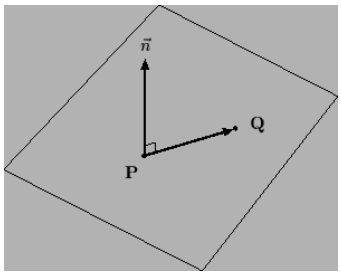


Figure 2.8: Rappresentazione del piano

stesso.

Il convex hull è necessariamente un insieme convesso, è infatti il più piccolo insieme convesso che contiene questi punti.

Il punto P_2 nell'esempio in Fig. 2.7 può essere scritto come combinazione convessa degli altri punti.

2.5 Rappresentazione della retta e del piano

In uno spazio affine possiamo definire una **retta** a partire da P_0, punto arbitrario, d vettore arbitrario, il parametro α, valore scalare definito nell'intervallo parametrico $\alpha \in]-\infty, +\infty[$,

$$P(\alpha) = P_0 + \alpha d.$$

Al variare del parametro α si ottengono tutti i punti della retta parallela a d e passante per P_0. Chiameremo questa rappresentazione della retta forma parametrica della retta.

La rappresentazione parametrica di un piano π passante per il punto $P_0 = (x_0, y_0, z_0)$, con vettori giacitura $v_1 = (v_{1x}, v_{1y}, v_{1z})$, $v_2 = (v_{2x}, v_{2y}, v_{2z})$ è data da

$$\pi(u,v) = P_0 + v_1 u + v_2 v = \begin{pmatrix} x_0 + v_{1x}u + v_{2x}v \\ y_0 + v_{1y}u + v_{2y}v \\ z_0 + v_{1z}u + v_{2z}v \end{pmatrix}$$

Otteniamo ora un'altra rappresentazione del piano.

Un piano in $\mathbb{R}^3$ che passa per un punto P è il luogo dei punti Q, tali che il vettore $Q - P$ risulti perpendicolare ad un vettore n (vettore normale). Ovvero tutti i punti Q del piano soddisfano l'equazione:

$$n \cdot < Q - P >= 0. \tag{2.1}$$

Si veda Fig.2.8.

Sia $n = (x_n, y_n, z_n)$ vettore, e $P = (x_p, y_p, z_p), Q = (x_q, y_q, z_q)$ punti. Imponendo la condizione (2.1) si ottiene l'equazione cartesiana del piano. Infatti:

$$\begin{aligned} 0 = \; & n \cdot < Q - P >= \\ = \; & (x_n, y_n, z_n) \cdot < x_q - x_p, y_q - y_p, z_q - z_p > \\ = \; & x_n(x_q - x_p) + y_n(y_q - y_p) + z_n(z_q - z_p) \\ = \; & x_n x_q + y_n y_q + z_n z_q - (x_n x_p + y_n y_p + z_n z_p) = 0 \\ = \; & \underbrace{x_n}_{A} x + \underbrace{y_n}_{B} y + \underbrace{z_n}_{C} z = \underbrace{x_n x_p + y_n y_p + z_n z_p}_{D}. \end{aligned}$$

L'equazione cartesiana del piano $Ax + By + Cz + D = 0$ può essere utilizzata per determinare

se un punto è posto $\begin{matrix} \text{sopra} \\ \text{sul} \\ \text{sotto} \end{matrix}$ piano,

verificando che

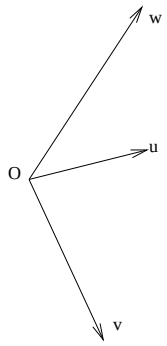


Figure 2.9: Frame

$$Ax + By + Cz - (Ax_0 + By_0 + Cz_0) \begin{matrix} > \\ = \\ < \end{matrix} 0.$$

2.6 Sistemi di riferimento (Frame)

Un sistema di riferimento in uno spazio affine è definito in computer graphics con il termine *frame*. Risorse luminose, camere, e modelli in scena sono definiti ciascuno nel proprio sistema di coordinate locali (frame) che poi viene posizionato all'interno della scena. Dobbiamo essere in grado di relazionare questi sistemi locali sia con un sistema di coordinate globale, sia tra di loro.

Tre vettori linearmente indipendenti forniscono una base per uno spazio vettoriale in $\mathbb{R}^3$. Di solito, ma non necessariamente, si scelgono tre vettori di modulo unitario ortogonali fra loro.

In $\mathbb{R}^3$ un **frame** $\Im = (\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{O})$ è un sistema di riferimento definito da un punto O detto *origine* e una base (tre vettori linearmente indipendenti) (vedi Fig. 2.9).

Sia V uno spazio affine di dimensione n nel quale è definito il frame con origine il punto O e $v_1, v_2, .., v_n$ una base di V.

- Ogni vettore v può essere rappresentato in $\Im$ in modo univoco :

$$v = c_1 v_1 + c_2 v_2 + ... + c_n v_n. \tag{2.2}$$

- Ogni punto P può essere rappresentato in $\Im$ in modo univoco:

$$P = v + O = c_1 v_1 + c_2 v_2 + ... + c_n v_n + O. \tag{2.3}$$

L'insieme di coordinate $(c_1, c_2, .., c_n)$ sono le coordinate di P relative al frame $\Im = (v_1, v_2, .., v_n, O)$.

Esempi di Frame

1. Sistema Cartesiano in $\mathbb{R}^2$: $(u, v, O), u =< 1, 0 >, v =< 0, 1 >, O = (0, 0)$. I punti P in questo riferimento hanno coordinate (x, y) tali che
$$P = xu + yv + O.$$
2. $(u, v, O), u =< 1, 0 >, v =< 1, 1 >, O = (0, 0)$. Il punto $P = (5, 3)$ in questo riferimento, può essere scritto come
$$P = 5u + 3v + O = 5 < 1, 0 > +3 < 1, 1 > +(0, 0)$$
in coordinate cartesiane $(8, 3)$.
3. $(u, v, O), u =< 1, 0 >, v =< 0, 2 >, O = (2, 2)$. Il punto $P = (5, 3)$ in questo riferimento, può essere scritto come
$$P = 5u + 3v + O = 5 < 1, 0 > +3 < 0, 2 > +(2, 2)$$
in coordinate cartesiane $(7, 8)$.

2.7 Rappresentazione di punti e vettori in coordinate omogenee

Osserviamo che bastano tre scalari per rappresentare un punto, come per un vettore, ma sono due entità diverse. Associare la tupla (x, y, z) ad un vettore e identificare quindi un vettore come il segmento dall'origine al punto $P = (x, y, z)$ può essere ambiguo. Si consideri ad esempio il vettore

da $(1,1,1)$ a $(2,3,4)$, è lo stesso del vettore da $(0,0,0)$ a $(1,2,3)$ poichè hanno stessa grandezza e direzione, ma non associeremo il primo vettore a $(1,2,3)$.

Punti e vettori possono essere univocamente rappresentati mediante le coordinate relative ad un frame specifico.

Dato il frame $\Im = (v_1, v_2, v_3, P_0)$ in uno spazio affine possiamo rappresentare un punto P utilizzando (3.1) e assumendo che $1 \cdot P = P$; si ottiene quindi

$$P = c_1 v_1 + c_2 v_2 + c_3 v_3 + 1 \cdot P_0.$$

In modo analogo, poichè i vettori dello spazio affine generano uno spazio vettoriale, si può rappresentare il vettore v mediante la (2.2) assumendo che $0 \cdot P = 0$ come combinazione:

$$v = c_1 v_1 + c_2 v_2 + c_3 v_3 + 0 \cdot P_0.$$

Quindi punti e vettori sono rappresentati nello spazio omogeneo da quattro coordinate, dette *coordinate omogenee*: i punti sono vettori riga con ultima componente 1 e i vettori sono rappresentati come vettori riga con ultima componente 0 (zero).

La rappresentazione di un punto/vettore in un frame $\Im$, si ottiene moltiplicando il vettore di coordinate omogenee del punto per la matrice 4×4 che rappresenta il frame:

$$P = \begin{bmatrix} c_1 & c_2 & c_3 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_O \end{bmatrix},$$

$$v = \begin{bmatrix} c_1 & c_2 & c_3 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_O \end{bmatrix},$$

dove

$$M = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_O \end{bmatrix} = \begin{bmatrix} v_{1x} & v_{1y} & v_{1z} & 0 \\ v_{2x} & v_{2y} & v_{2z} & 0 \\ v_{3x} & v_{3y} & v_{3z} & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix}. \tag{2.4}$$

2.7.1 Cambio del sistema di coordinate

Siano $\Im_1 = (v_1, v_2, v_3, P_0)$ e $\Im_2 = (u_1, u_2, u_3, Q_0)$ due frame (due basi di uno stesso spazio). Esprimiamo vettori e origine dell'uno in termini dei vettori e origine dell'altro:

$$\begin{aligned} u_1 &= v_{1x}\, v_1 + v_{1y}\, v_2 + v_{1z}\, v_3, \\ u_2 &= v_{2x}\, v_1 + v_{2y}\, v_2 + v_{2z}\, v_3, \\ u_3 &= v_{3x}\, v_1 + v_{3y}\, v_2 + v_{3z}\, v_3, \\ Q_0 &= x_0\, v_1 + y_0\, v_2 + z_0\, v_3 + P_0, \end{aligned}$$

In forma matriciale:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_O \end{bmatrix} = \begin{bmatrix} v_{1x} & v_{1y} & v_{1z} & 0 \\ v_{2x} & v_{2y} & v_{2z} & 0 \\ v_{3x} & v_{3y} & v_{3z} & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix}$$

Questo definisce la matrice M di dimensione 4×4 che rappresenta il cambiamento tra il sistema di riferimento $\Im_1$ e $\Im_2$, definita in (2.4).

Mentre M^{-1} fornisce i coefficienti per la conversione da $\Im_1$ a $\Im_2$.

La conversione di punti e vettori di uno stesso spazio affine da un frame ad un altro si ottiene facilmente nel modo seguente.

Dato un generico punto o vettore w, siano $a = (a_1, a_2, a_3, 1)^T$ e $b = (b_1, b_2, b_3, 1)^T$ le sue rappresentazioni in coordinate omogenee rispettivamente nei due frame $\Im_1$ e $\Im_2$ con l'ultima coordinata 1 per i punti e 0 per i vettori, e sia M matrice di conversione (2.4), allora

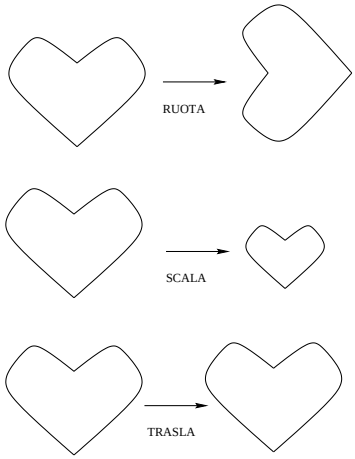


Figure 2.10: Trasformazioni Affini

$w = a^T \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix}$ rispetto ad $\Im_1$, $w = b^T \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{bmatrix}$ rispetto ad $\Im_2$,

quindi

$$w = b^T \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{bmatrix} = b^T M \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} = a^T \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix}$$

da cui si deduce

$$a = M^T b, \quad b = (M^T)^{-1} a.$$

2.8 Trasformazioni geometriche 2D

Le trasformazioni geometriche considerate nel seguito modificano le coordinate di un oggetto per ottenerne un altro simile, ma differente per posizione, orientamento e dimensione. Le trasformazioni considerate alterano la geometria lasciando invariata la topologia.

Ogni trasformazione geometrica complessa può essere decomposta in una concatenazione di trasformazioni geometriche elementari quali traslazione, scala e rotazione mostrate in Fig.4.6.

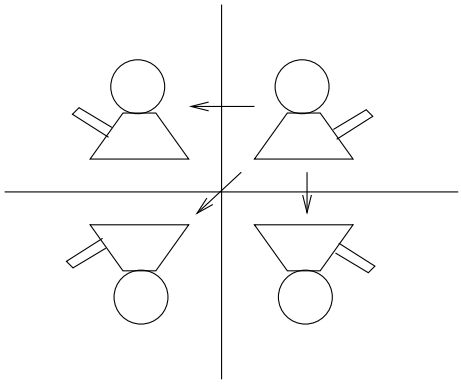


Figure 2.11: Scaling per riflessione: (in senso antiorario) originale; Sx=-1,Sy=1; Sx=-1,Sy=-1;Sx=1,Sy=-1;

Ogni trasformazione lineare è rappresenta da una certa matrice A non singolare tale che, in forma matriciale:

$$P' = AP = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = [(ax + by), (cx + dy)].$$

D'altra parte ogni matrice non singolare A rappresenta una trasformazione lineare.

L'importanza della proprietà di linearità sta nel fatto che, note le trasformazioni dei vertici, si possono ottenere le trasformazioni di combinazioni lineari dei vertici combinando linearmente le trasformazioni dei vertici. Non dobbiamo ricalcolare le trasformazioni per ogni combinazione lineare.

Se infatti $P \in P_1P_2 \Rightarrow P' \in P_1'P_2'$, con $P_1'P_2'$ estremi trasformati di P_1 e P_2.

Vediamo come rappresentare le trasformazioni lineari di scala e rotazione in forma matriciale.

- Scala

  $$P' = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} P$$

  $Sx, Sy > 1$ espansione $Sx, Sy < 1$ compressione $Sx = Sy$ trasformazione uniforme, $Sx < 0$ o $Sy < 0$ (riflessione, si veda Fig. 2.11)

- Rotazione 2D

$$P' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} P$$

La traslazione di un punto P in P' di un vettore $(\Delta x, \Delta y)$, esprimibile nella forma

$$P' = P + \begin{bmatrix} \Delta x & \Delta y \end{bmatrix}$$

non è una trasformazione lineare.

Queste trasformazioni si dicono *affini* ovvero composizioni di trasformazioni lineari e traslazioni esprimibili nella forma:

$$P' = P_0 + PA,$$

dove A rappresenta una trasformazione lineare, P_0 è un punto e P' è il trasformato del punto P.

Le trasformazioni affini sono le sole che conservano le linee rette (cioè la trasformazione di una retta è ancora una retta).

Se passiamo ad un sistema di coordinate omogenee possiamo trattare tutte le trasformazioni affini nella forma di moltiplicazioni matrice-vettore. Definiamo quindi la matrice di dimensione 3×3, che lascia invariata la terza componente della rappresentazione (0 per i vettori, 1 per i punti):

$$A = \begin{bmatrix} a & b & x_0 \\ c & d & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Trasformazioni affini 2D in coordinate omogenee sono definite da:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} a & b & x_0 \\ c & d & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

- Traslazione

$$P' = TP = \begin{bmatrix} 1 & 0 & \Delta_x \\ 0 & 1 & \Delta_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Scala

$$P' = SP = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Rotazione 2D

$$P' = RP = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Invertibilità delle trasformazioni

Poiché M é non singolare, allora esiste la matrice inversa M^{-1} tale che

$$MM^{-1} = I,$$

dove I rappresenta la matrice identità. Allora applicare ad un punto P', trasformato di P mediante M, la matrice inversa della matrice di trasformazione M equivale a riottenere P. Se $P' = MP$, allora moltiplicando a destra e sinistra per M^{-1} otteniamo $P = M^{-1}P'$.

Fortunatamente per le trasformazioni viste le matrici inverse sono particolarmente semplici da calcolare. Infatti, l'inversa della matrice di trasformazione di rotazione R di un angolo θ è data dalla matrice di rotazione di un angolo $-\theta$:

$$R(-\theta) = R^T(\theta) = R^{-1}(\theta)$$

(R è infatti una matrice ortogonale).

L'inversa della trasformazione di traslazione T di un vettore t:

$$T^{-1}(t) = T(-t).$$

L'inversa della trasformazione di scala S:

$$S^{-1}(s) = S(1/s_x, 1/s_y).$$

2.8.1 Concatenazione di trasformazioni

Se il punto P è trasformato in P' mediante la sequenza di trasformazioni $A_1, A_2, A_3, .., A_n$

$$P' = (A_n \ldots \cdot (A_3 \cdot (A_2 \cdot (A_1 \cdot P))))$$

allora per la proprietà associativa della moltiplicazione tra matrici si può considerare un'unica matrice di trasformazione $A = A_n \ldots \cdot A_3 \cdot A_2 \cdot A_1$ ed applicare quest'ultima a P:

$$P' = AP.$$

Esempio La concatenazione della trasformazione di scala (Sx=Sy=2) con una traslazione ($\Delta x = 10, \Delta y = 0$) è ottenuta mediante le seguenti operazioni matriciali:

$$P' = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 10 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 2 & 0 & 10 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = AP$$

dove, per la proprietà associativa di cui gode il prodotto tra matrici, si può considerare A la matrice di trasformazione della sequenza.

E' utile ricordare le **proprietà del prodotto di matrici**:

- associativa
$$(A_1 \cdot A_2) \cdot A_3 = A_1 \cdot (A_2 \cdot A_3)$$
- non vale in generale la commutativa
$$P(A_1 \cdot A_2) \neq P \cdot (A_2 \cdot A_1)$$

In figura 2.12 è illustrato un esempio di concatenazione di due trasformazioni di rotazione poi traslazione e traslazione poi rotazione che producono risultati diversi.

2.8.2 Trasformazioni rispetto ad un punto

Consideriamo un quadrato centrato nell'origine del sistema Cartesiano. Applichiamo ad esso la trasformazione di scala ad ogni vertice mediante

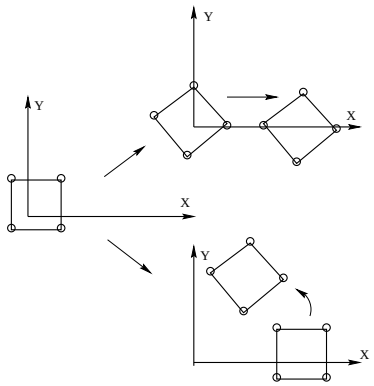


Figure 2.12: Esempio di composizione di due trasformazioni di rotazione e traslazione (alto) e traslazione e rotazione (basso).

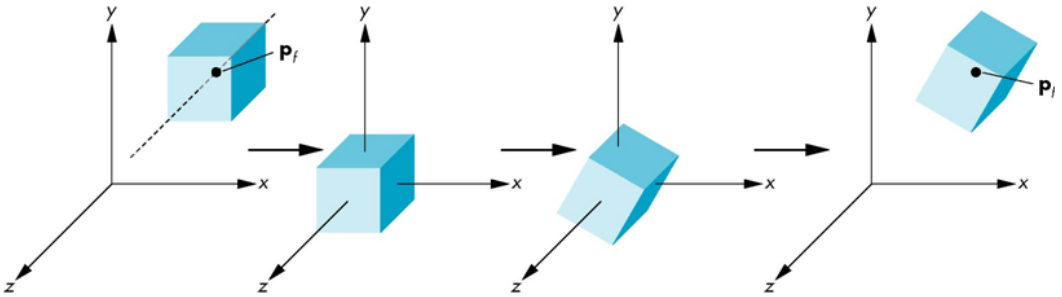


Figure 2.13: Esempio di rotazione rispetto ad un punto diverso dall'origine.

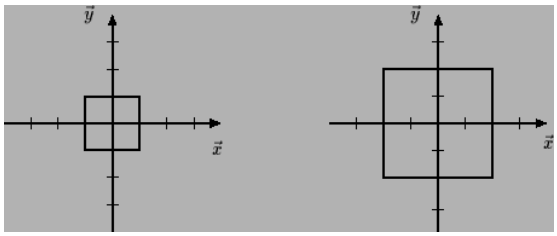
la matrice

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

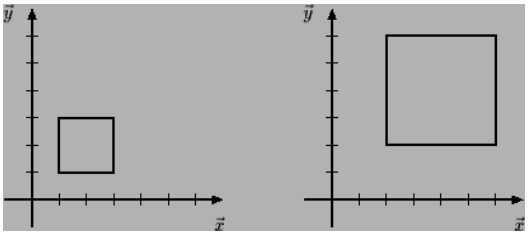
come illustrato in Fig.2.14(a).

Applicando ora la stessa scala ad un quadrato con centro in $(2,2)$; il quadrato scalato risultante ha centro in $(4,4)$, come illustrato in Fig.2.14(b).

Per operare quindi una scala o una rotazione con centro diverso dall'origine, si deve prima riportare con una traslazione il punto nell'origine, operare la trasformazione richiesta e poi applicare la traslazione inversa nel punto originale. Il concetto è illustrato in Fig.2.13.



(a)



(b)

Figure 2.14: Esempio di scala: quadrato centrato nell'origine (sopra); con origine diversa (sotto)

Ad esempio la rotazione di un angolo α rispetto ad un punto diverso dall'origine prevede una matrice di trasformazione composita

$$A = \begin{bmatrix} 1 & 0 & -u \\ 0 & 1 & -v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & u \\ 0 & 1 & v \\ 0 & 0 & 1 \end{bmatrix}$$

Mentre la trasformazione di scala rispetto ad un punto diverso dall'origine utilizza la matrice

$$A = \begin{bmatrix} 1 & 0 & -u \\ 0 & 1 & -v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & u \\ 0 & 1 & v \\ 0 & 0 & 1 \end{bmatrix}$$

2.9 Trasformazioni geometriche 3D

Analogamente alle trasformazioni affini in $\mathbb{R}^2$, anche in $\mathbb{R}^3$ utilizzando le coordinate omogenee, le trasformazioni affini assumono la forma gen-

erale di moltiplicazione matrice per vettore:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} a & b & c & 0 \\ d & e & f & 0 \\ g & h & i & 0 \\ l & m & n & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

In particolare:

- Traslazione

$$P' = TP = \begin{bmatrix} 1 & 0 & 0 & \Delta_x \\ 0 & 1 & 0 & \Delta_y \\ 0 & 0 & 1 & \Delta_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Scala

$$P' = SP = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Rotazioni 3D
  - Rotazione 3D attorno asse x

$$P' = R_x P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

  - Rotazione 3D attorno asse y

$$P' = R_y P = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

  - Rotazione 3D attorno asse z

$$P' = R_z P = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

La composizione di rotazioni intorno ad uno stesso asse è commutativa, mentre la composizione di rotazioni differenti non è commutativa.

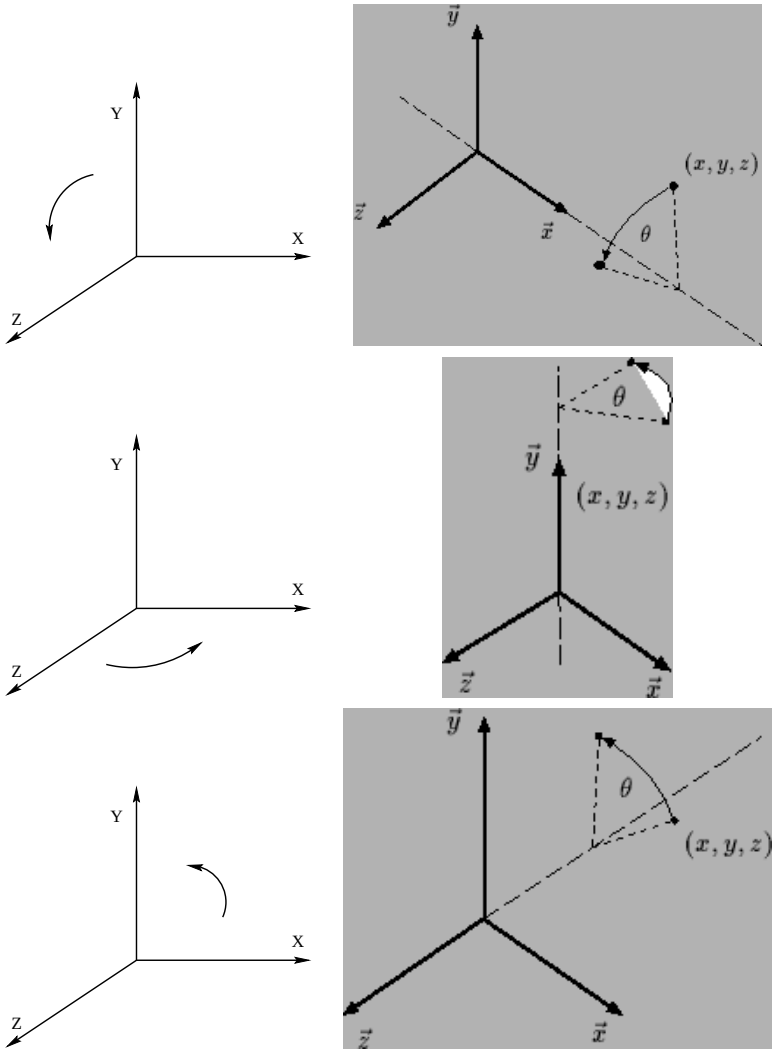


Figure 2.15: Rotazioni 3D: rispetto rispettivamente a x (sopra), ad y (centro) e a z (sotto).

Le rotazioni 3D relative ai tre assi sono mostrate in Fig.2.15.

Una trasformazione geometrica si definisce **rigida** o **Rigid-Body** se riposiziona gli oggetti lasciando invariata la loro forma e la loro dimensione. Quindi orientamento e posizione cambiano ma non la forma di un oggetto. Queste trasformazioni si ottengono come concatenazione di sole rotazioni e traslazioni.

Trasformazioni rigid-body hanno la caratteristica di preservare lunghezze (distanze fra punti) ed angoli tra linee.

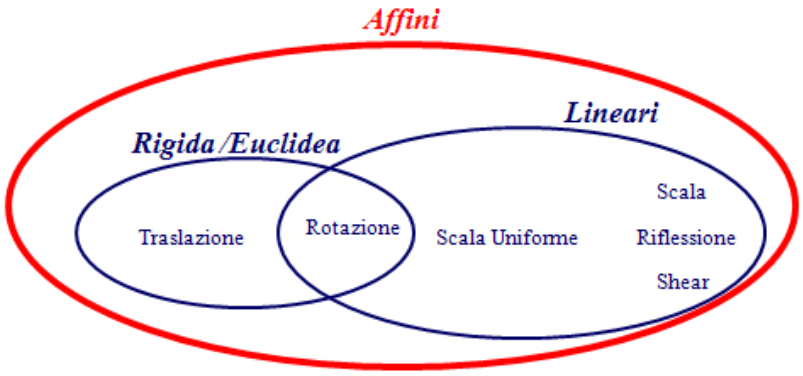


Figure 2.16: Trasformazioni Affini, Lineari, Rigide

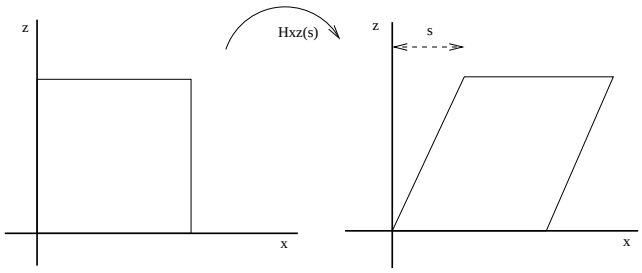


Figure 2.17: Trasformazione di shear.

Relazioni tra trasformazioni affini, lineari e rigide sono illustrate in Fig.2.16.

2.9.1 Trasformazione di Shear

La trasformazione affine di shear altera 2 o 3 valori di coordinate proporzionalmente al valore delle altre coordinate.

Matrici di shear:

$$H_{xy}(s), H_{xz}(s), H_{yx}(s), H_{yz}(s), H_{zx}(s), H_{zy}(s).$$

Con la notazione H_{ij} si indica con il primo pedice la coordinata da cambiare e con il secondo pedice la coordinata che deforma.

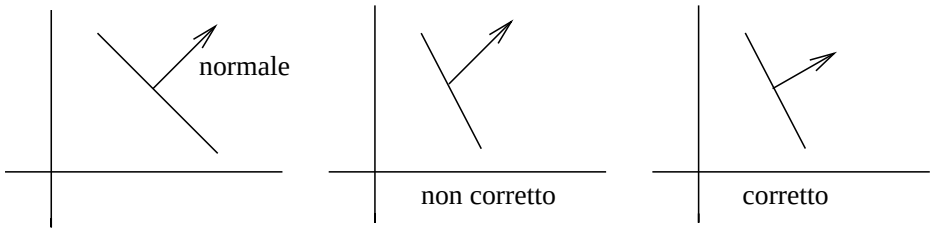


Figure 2.18: Trasformazione dei vettori normali

Nell'esempio illustrato in Fig. 2.17, si utilizza la matrice

$$H_{xz}(s) = \begin{bmatrix} 1 & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Quindi si ha

$$\begin{bmatrix} 1 & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} P = \begin{bmatrix} p_x + sp_z & p_y & p_z & 1 \end{bmatrix}$$

La trasformazione di shearing inversa applica shearing nella direzione opposta:

$$H_{ij}^{-1}(s) = H_{ij}(-s).$$

2.9.2 Trasformazione dei vettori normali

In Fig.2.18 applichiamo la trasformazione di scala S di (0.5,1) ai vertici del segmento. Se applichiamo la medesima trasformazione al vettore normale n al segmento otteniamo un risultato sbagliato illustrato in Fig.2.18 (centro). Per ottenere una corretta trasformazione della normale (Fig.2.18 (destra)), dobbiamo trasformare n mediante la matrice

$$(M^{-1})^T,$$

ovvero $n' = (M^{-1})^T n$.

Osserviamo che solo le trasformazioni di traslazione (o solo le trasformazioni di scaling) non modificano il vettore normale.

Chapter 3

Modellazione geometrica con curve e superfici

La *modellazione geometrica* si occupa di metodi matematici per descrivere la forma di un oggetto o per esprimere un processo fisico in termini di un'opportuna rappresentazione geometrica.

Si costruisce un modello, fisico o astratto, poichè è conveniente ed economico sostituirlo all'oggetto o processo reale e perchè è più facile e più pratico analizzare un modello virtuale piuttosto che validare, sperimentare, misurare un oggetto o un fenomeno reale.

Costruire il modello di un oggetto significa definire una rappresentazione matematica di questo, mediante opportune primitive geometriche. Il problema della rappresentazione di una forma qualsiasi, fu affrontato a partire dagli anni sessanta con l'avvento dei primi sistemi di Computer Aided Design (CAD) e portò alla nascita di quella disciplina che viene definita Computer Aided Geometric Design (CAGD).

La modellazione geometrica è alla base dello sviluppo dei sistemi CAD-CAM che si occupano di produzione- progettazione a livello industriale di automobili, carene di navi, fusoliere di aeroplani, utensili vari, oggettistica, giocattoli,..

Per passare dalla fase di progettazione a quella di produzione è indispensabile una descrizione matematica della forma dell'oggetto mediante curve e superfici matematiche.

La modellazione geometrica è una disciplina recente. Le prime proposte affiancano lo sviluppo dei primi sistemi grafici di computer realizzati per CAD-CAM e per controllare i percorsi delle prime macchine da taglio a controllo numerico. Tra i pionieri di questa disciplina ricordiamo Coons dell' MIT (1963), Ferguson della Boeing (1964), de Casteljau alla Citroen (1966), Bézier alla Renault (1966)....

3.1 Rappresentazione esplicita ed implicita di punti e curve

Le primitive geometriche punti, curve, superfici e volumi possono essere rappresentate matematicamente in forma **esplicita** o **implicita**.

Rappresentare i punti in forma esplicita significa definire le coordinate dei punti. In alternativa, la rappresentazione implicita definisce i punti come i valori per cui una certa funzione si annulla.

Per esempio, in una dimensione spaziale, i punti $x = -1$ e $x = 1$ sulla retta reale, possono essere rappresentati implicitamente come isocontorno zero della funzione $\phi(x) = x^2 - 1$, cioè tutti quei punti x tali che $\phi(x) = 0$. La funzione implicita $\phi()$ divide il dominio Ω (la retta reale) in due regioni una esterna Ω^- ed una interna Ω^+. I punti di bordo $\partial\Omega = \{-1, 1\}$ si chiamano interfaccia. Le regioni interne ed esterne sono primitive uno-dimensionali, mentre l'interfaccia è zero-dimensionale. La rappresentazione implicita dei punti $x = -1$ e $x = 1$ è illustrata in Fig.3.1(a).

Più in generale in $\mathbb{R}^n$, la funzione implicita $\phi(\overrightarrow{x})$ è definita per tutti gli $\overrightarrow{x} \in \mathbb{R}^n$, e il suo isocontorno (interfaccia) ha dimensione $n - 1$.

In due dimensioni, per rappresentare in forma implicita una curva, l'interfaccia uno-dimensionale è una curva che separa $\mathbb{R}^2$ in due sottodomini di area non nulla. Limitiamo la trattazione a curve chiuse, che stabiliscono una chiara definizione di interno ed esterno. Per esempio: $\phi(\overrightarrow{x}) = x^2 + y^2 - 1$ avrà l'interfaccia definita da $\phi(\overrightarrow{x}) = 0$ ovvero il cerchio unitario definito da $\partial\Omega = \{\overrightarrow{x} : \|\overrightarrow{x}\| = 1\}$. La rappresentazione implicita del cerchio unitario è illustrata in Fig.3.1(b).

Si chiama rappresentazione implicita poichè i punti (x, y) sulla curva devono essere determinati risolvendo l'equazione e non sono quindi definiti

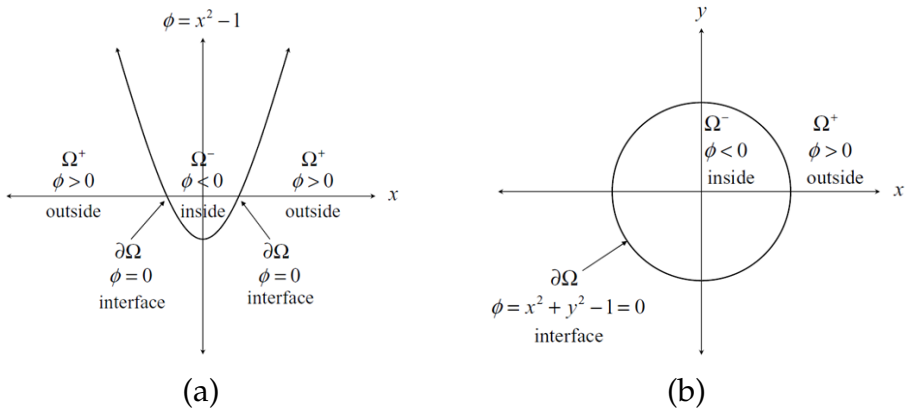


Figure 3.1: Rappresentazione implicita dei punti (a) e curve (b)

esplicitamente.

Una rappresentazione esplicita di una curva si ottiene come grafico di una funzione in una variabile. Sia $D \subset \mathbb{R}$, una funzione f in una variabile x, allora $f : D \to \mathbb{R}$ associa ad ogni $x \in D$ un unico numero reale $f(x)$. D è il dominio di f e il range è l'insieme dei valori che f assume:

$$\{y = f(x) / x \in D\}$$

Il grafico di f è l'insieme: $S = \{(x, y) \in \mathbb{R}^2 : y = f(x), x \in D\}$ è una curva di equazione esplicita $y = f(x)$.

La curva in tale forma esplicita è ad un sol valore, cioè definisce un unico valore di y per ogni $x \in D$. Perciò queste curve non possono auto-intersecarsi, ma soprattutto non si può con tale forma rappresentare una qualsiasi curva. Cioè non tutte le curve sono grafici di funzione, si pensi ad esempio al cerchio unitario dell'esempio definito in forma implicita. In forma esplicita potremmo rappresentare solo il semicerchio $y = \pm\sqrt{(1 - x.^2)}$. Inoltre non si possono in generale applicare ad esse trasformazioni affini (traslazioni, rotazioni,..) mantenendo l'integrità della rappresentazione. La formulazione esplicita vista è inadeguata per la modellazione geometrica o, più in generale, per l'utilizzo in problemi di computer graphics.

Per superare tali difficoltà si utilizza una forma esplicita vettoriale, anche detta forma **parametrica**.

Una curva parametrica in $\mathbb{R}^2$, è definita da una funzione vettoriale

$$C(t) = (x(t), y(t)),$$

dove $x(t)$, $y(t)$, sono funzioni del parametro $t \in I \subset \mathbb{R}$ dette componenti cartesiane del vettore posizione (punto sulla curva). $C(t)$ fa corrispondere a valori dell'intervallo I punti di $\mathbb{R}^2$: $C : I \to \mathbb{R}^2$. Al variare di t, le coordinate $(x(t), y(t))$ individuano un punto che si sposta sulla curva. $x(t), y(t)$ sono funzioni reali di variabile reale, o semplicemente funzioni scalari (es. polinomi). L'intervallo $I = [a, b]$ definisce un insieme di punti in cui si sceglie di visualizzare la curva anche se essa vive anche per valori al di fuori da questo intervallo: $(-\infty, +\infty)$. La curva ristretta ad I si definisce propriamente segmento di curva.

L'operazione di calcolo di un punto sulla curva è immediato. Trasformazioni geometriche come traslazioni o ratazioni possono essere semplicemente effettuate mediante moltiplicazioni con una matrice.

L'equazione parametrica della curva (o curva parametrica) in $\mathbb{R}^3$ è funzione vettoriale

$$C(t) = (x(t), y(t), z(t)),$$

dove il parametro $t \in I = [a, b]$ e $x(t), y(t), z(t)$ sono le componenti cartesiane del vettore posizione e sono funzioni scalari.

3.1.1 Lunghezza e curvatura di una curva parametrica

Si deve distinguere tra il termine parametrizzazione e immagine (o supporto della curva).

Vediamo di chiarirlo interpretando la curva come la traettoria di un oggetto puntiforme (una particella) che si muove nel tempo; in questo caso t è la variabile che rappresenta il tempo.

Consideriamo una curva piana in forma parametrica $C(t)$ descritta dall' equazione parametrica $C(t) = (x(t), y(t))$, $t \in [a, b]$. Se una particella si muovesse nel tempo lungo la curva parametrica, ad ogni istante t, la posizione della particella sarebbe $(x(t), y(t))$.

La stessa curva (supporto di curva) può essere tracciata a velocità differenti.

Per esempio sia data l'equazione parametrica della circonferenza

$$C(t) = (sin(2t), cos(2t)), t \in [0, 2\pi].$$

La curva inizia in $(0, 1)$ e la particella si muove lungo la curva percorrendo due volte la circonferenza. La stessa circonferenza si ottiene anche con le funzioni coordinate $C_1(t) = (sin(t), cos(t)), t \in [0, 2\pi]$. Le due rappresentazioni parametriche rappresentano il medesimo percorso (cerchio) ma valutandole per medesimi valori di t, rappresentano punti diversi sulla curva.

I punti sulla circonferenza rappresentano l'immagine o supporto.

Il moto della particella è differente anche se il percorso tracciato (curva) è lo stesso.

La stessa curva (supporto di curva, immagine) può avere perciò parametrizzazioni (cioè funzioni coordinate) differenti. Le equazioni

$$\begin{cases} x = x(t) \\ y = y(t) \\ z = z(t) \end{cases},$$

o equivalentemente l'applicazione C si dicono parametrizzazione della curva.

Per esempio la circonferenza può essere rappresentata dalle due diverse coppie di funzioni coordinate:

a) $C(t) = (sin(t), cos(t)), t \in [0, \frac{\pi}{2}]$;

b) $C(t) = (\frac{2t}{(1+t^2)}, \frac{(1-t^2)}{(1+t^2)}), t \in [0, 1]$.

Al variare di t le coordinate $(x(t), y(t))$ rappresentano un punto che si sposta sul cerchio unitario. Per che cosa differiscono a) e b)? Stessa immagine (supporto) ma differente parametrizzazione.

Tornando al nostro modello fisico del moto della particella, ad ogni istante t_0, le coordinate $(x(t_0), y(t_0))$ individuano un punto che si sposta sulla curva con velocità data dalla tangente alla curva parametrica in t_0:

$$v(t_0) = \frac{d}{dt}C(t_0) = C'(t_0) = (x'(t_0), y'(t_0)).$$

Si possono determinare due importanti informazioni:

- Il *modulo del vettore velocità* rappresenta la velocità istantanea in unità di distanza per unità di tempo) con la quale la particella si muove lungo la curva, ed e' dato da:

$$\|C(t_0)\|_2 = \sqrt{x'(t_0)^2 + y'(t_0)^2}.$$

- La *direzione del vettore velocità* è data dal vettore tangente $C'(t_0)$ e punta nel verso delle t crescenti.

La parametrizzazione di una curva non è unica.

Data una funzione strettamente monotona crescente $\tau(s) : [0, a] \to [0, b]$, la curva $C(t)$ può essere riparametrizzata in $C(\tau(s))$, in modo tale che muovendosi lungo $s \in [0, b]$, si generano gli stessi punti lungo la curva. La forma della curva rimane la stessa, ma la velocità di percorrimento diventa

$$\overrightarrow{v}(s) = \frac{d}{ds}C(\tau(s)) = \frac{d}{dt}C(\tau(s))\frac{d}{ds}\tau(s)$$

Ad ogni punto la velocità di $C(\tau(s))$ è scalata di un fattore $\frac{d}{ds}\tau(s)$. Data una curva parametrica esiste un'unica sua riparametrizzazione $C(\tau(s))$ con la proprietà che $\|C'(t)\|_2 = 1$, $s \in [0, b]$, questa si chiama *parametrizzazione alla lunghezza ad arco*. In una curva parametrizzata alla lunghezza ad arco, a spostamenti unitari lungo il parametro s, corrispondono spostamenti lungo la curva di lunghezza unitaria. In questo caso speciale di parametrizzazione alla lunghezza ad arco la curvatura in un punto $C(s)$, corrisponde alla derivata seconda $\frac{d^2}{ds^2}C(s)$. In generale, per una arbitraria curva in forma parametrica è un grave errore identificare la derivata seconda con la curvatura.

Lunghezza della curva Siano x' e y' derivate continue in $[0, a]$ di una curva $C(t) = (x(t), y(t))$, allora la lunghezza di C tra $C(0)$ e $C(a)$ è definita dalla seguente formula:

$$L = \int_0^a \sqrt{([x'(t)]^2 + [y'(t)]^2}dt = \int_0^a \|C'(t)\|dt.$$

La lunghezza dell'arco di curva da $C(0)$ a $C(a)$ si calcolata quindi come limite della lunghezza della corda (somma lunghezze di segmenti poligonali). La lunghezza di una curva in $\mathbb{R}^3$, $C(t) = (x(t), y(t), z(t))$ è esatta-

mente definita nello stesso modo:

$$L = \int_0^a \sqrt{([x'(t)]^2 + [y'(t)]^2 + [z'(t)]^2} dt.$$

Poichè $\|C'(t)\|_2$ rappresenta la velocità istantanea della curva C, allora se $\|C'(t)\|_2 = 1$ per ogni t , la velocità lungo la curva è costante e uguale a 1. In una curva parametrizzata alla lunghezza ad arco, a spostamenti unitari lungo il parametro s, corrispondono spostamenti lungo la curva di lunghezza unitaria.

Curvatura Ciò che vogliamo definire e calcolare è la misura matematica di quanto una curva devii dall'essere retta nell'intorno di un punto. Poichè la curvatura di una curva in generale cambia da punto a punto.

Sia $C(t), C : I \to \mathbb{R}^n$, una curva regolare di classe C^k, con $k \geq 2$. La curvatura di $C(t)$ è la funzione $k : I \to \mathbb{R}^+$, di classe C^{k-2}, data da:

$$k(t) = \frac{\|T'(t)\|}{\|C'(t)\|}. \tag{3.1}$$

La relazione (3.1) esprime la variazione del versore tangente rispetto ad una variazione nella posizione. Nel punto t_0, la curvatura è lo scalare $k(t_0)$.

La curvatura di una linea dritta è sempre zero poichè il vettore tangente è costante. Sia $C(t)$ una circonferenza di raggio R, $C(t) = (R\cos(t), R\sin(t))$. Allora $C'(t) = (-R\sin(t), R\cos(t))$, $\|C'(t)\|_2 = R$.

$$T(t) = \frac{C'(t)}{\|C'(t)\|} = (-\sin(t), \cos(t)).$$

$$k(t) = \frac{\|T'(t)\|}{R} = \frac{\|(-\cos(t), -\sin(t))\|_2}{R} = \frac{1}{R}.$$

Quindi cerchi di grande raggio hanno piccola curvatura mentre cerchi di piccolo raggio avranno grande curvatura, come intuibile.

Sebbene la formula (3.1) può essere usata in tutti i casi per ottenere la curvatura di una curva in forma parametrica, spesso risulta più conveniente

da applicare la seguente

$$k(t) = \frac{|C'(t) \times C''(t)|}{|C'(t)|^3}. \tag{3.2}$$

Per una curva parametrica piana $C(t) = (x(t), y(t))$, la (3.2) si riconduce a

$$k = \frac{|x'y'' - x''y'|}{[x'^2 + y'^2]^{\frac{3}{2}}} \tag{3.3}$$

dove il simbolo $'$ indica derivata rispetto a t quindi $x' = \frac{dx}{dt}$.

Per una curva in forma esplicita $y = f(x)$, si ha

$$k = \frac{|\frac{d^2y}{dx^2}|}{[1 + (\frac{dy}{dx})^2]^{\frac{3}{2}}}$$

Tale formula si verifica facilmente dalla (3.3) considerando la curva esplicita come caso particolare della parametrica. Infatti una curva funzionale $y = f(x)$ può essere rappresentata in forma esplicita parametrica $C(t) = (x(t), y(t))$ di parametro t, ponendo $x(t) = t, y(t) = f(t)$.

3.1.2 Continuità parametrica e geometrica di una curva

Consideriamo la possibilità di unire diversi segmenti curvi al fine di ottenere una curva più complessa. Ciò implica, nell'applicazione pratica, una maggiore possibilità di controllo locale della forma della curva.

Se due segmenti di curva si uniscono ad un estremo, si dice che tra i due segmenti di curva c'è un raccordo C^0 che assicura l'assenza di salti. Se due tratti di curva si uniscono in un punto P_0 e, inoltre, detti v_1 e v_2 i vettori velocità in P_0 del primo e secondo tratto di curva rispettivamente, si definisce

- *Continuità parametrica* C^1
  Le direzioni e i moduli dei vettori tangenti v_1, v_2 dei due segmenti curvi nel punto di contatto P_0 sono uguali
- *Continuità geometrica* G^1
  Le direzioni dei vettori tangenti v_1, v_2 dei due segmenti curvi nel punto di contatto sono uguali, i moduli possono essere diversi.

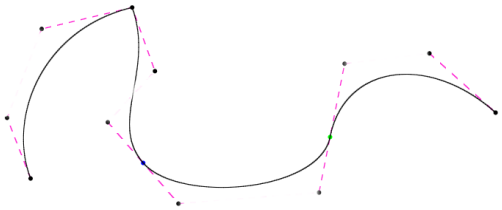


Figure 3.2: Quattro tratti di curve cubiche.

In generale C^1 implica G^1, il contrario in generale non è vero. Se la derivata di una curva continua è continua allora la curva è anche C^1. La continuità parametrica garantisce solo che il moto della particella che si muove lungo la curva sia continuo, cioè non ci siano salti improvvisi nella velocità, non è indicativa della regolarità (smooth)della forma della curva (percorso). Per esempio, una linea dritta con con un salto di velocità non è C^1, ma la curva è certamente regolare. Se la direzione tangente ad una curva parametrica varia in modo continuo allora è continua G^1. La sua grandezza può avere anche salti discontinui, ma la curva è ancora G^1. Quindi la particella che si muove lungo la curva varia la sua velocità a sbalzi ma ancora percorre una curva continua se la sua direzione cambia in modo continuo.

In Fig.3.2, sono rappresentati quattro tratti di curve congiunti fra loro in tre punti, rispettivamente C^0, C^1 e G^1.

3.2 Rappresentazione esplicita ed implicita di superfici

Il concetto di superficie si forma in modo intuitivo nell'esperienza quotidiana, considerando ad esempio il bordo di oggetti concreti. Una superficie è un ente geometrico che si può pensare generato in vari modi, come dal movimento continuo di una curva oppure dal contorno di un corpo solido. Una superficie in uno spazio euclideo tridimensionale (dotato di un sistema di assi cartesiani x,y,z), viene rappresentata in forma esplicita o implicita.

- Forma esplicita: la superficie viene definita come grafico di una funzione reale in due variabili.
  Funzione in due variabili Sia $D \subset \mathbb{R}^2$. Una funzione f in due variabili $f : D \to \mathbb{R}$ associa ad ogni coppia $(x, y) \in D$ un unico numero reale $f(x, y)$. D è il dominio di f e il range è l'insieme dei valori che f assume:

  $$\{z = f(x, y)/(x, y) \in D\}$$

  Il grafico di f funzione in due variabili di dominio D è l'insieme:

  $$S = \{(x, y, z) \in \mathbb{R}^3/z = f(x, y), (x, y) \in D\}$$

  Così come il grafico di una funzione in una variabile f è una curva di equazione $y = f(x)$, il grafico di una funzione in due variabili è una superficie S di equazione $z = f(x, y)$. Le **curve di livello** di una funzione f di due variabili sono le curve di equazione $f(x, y) = k$ dove k è una costante (nel range di f). Il **piano tangente ad una superficie** $z = f(x, y)$ **in due variabili** nel punto $P(x_0, y_0, z_0)$ è

  $$z - z_0 = f_x(x_0, y_0)(x - x_0) + f_y(x_0, y_0)(y - y_0).$$

- Forma implicita: la superficie è l'interfaccia che separa $\mathbb{R}^3$ in sottodomini di volume non nullo.
  Considerando solo superfici chiuse, si può quindi definire una regione esterna ed una interna. L'equazione implicita definisce implicitamente l'insieme dei punti che sono sulla superficie, come quei punti (x, y, z) che soddisfano $\phi(x, y, z) = 0$.
  Ad esempio sia data la funzione implicita

  $$\phi(x, y, z) = x^2 + y^2 + z^2 - 1,$$

  l'interfaccia è definita dai punti (x, y, z) tali che $\phi(x, y, z) = 0$ che è il contorno della sfera unitaria.

La classe delle superfici che possono essere rappresentate in forma esplicita è più ristretta rispetto a quelle rappresentabili in forma implicita.

Le limitazioni che si hanno con la rappresentazione esplicita di una superficie come grafico di una funzione bivariata possono essere superate

utilizzando la forma esplicita parametrica, ovvero la superficie è definita mediante una funzione vettoriale bivariata.

3.2.1 Superfici in forma parametrica

Superfici bidimensionali possono essere immerse in spazi di dimensioni arbitrarie, ma poichè noi vogliamo rappresentare oggetti reali, ci limitiamo a superfici bidimensionali immerse nello spazio Euclideo $\mathbb{R}^3$.

Definizione: Superficie parametrica *Una superficie immersa (o parametrizzata) nello spazio $\mathbb{R}^3$ è un'applicazione*

$$S : U \to \mathbb{R}^3$$

di classe C^∞, dove $U \subseteq \mathbb{R}^2$ è un aperto, S è funzione vettoriale nei due parametri $(u, v) \in U$:

$$S(u, v) = (x(u, v), y(u, v), z(u, v))$$

Le funzioni componenti o coordinate del vettore posizione sono $x(u, v)$,$y(u, v)$,$z(u, v)$.

Al variare delle coppie (u, v) in U, il luogo dei punti descritti dal vettore posizione è la superficie parametrica.

Mantenendo u costante allora $S(u_0, v)$ diventa una funzione vettoriale ad un unico parametro che definisce una curva che giace su S ed ha un vettore tangente:

$$S_v =< \frac{\partial x}{\partial v}(u_0, v), \frac{\partial y}{\partial v}(u_0, v), \frac{\partial z}{\partial v}(u_0, v) >=< x_v, y_v, z_v > .$$

Analogamente se manteniamo costante il parametro $v = v_0$, otteniamo una curva che giace su S di vettore tangente:

$$S_u =< \frac{\partial x}{\partial u}(u, v_0), \frac{\partial y}{\partial u}(u, v_0), \frac{\partial z}{\partial u}(u, v_0) >=< x_u, y_u, z_u > .$$

Vettore normale Poichè i vettori S_u e S_v giacciono nel piano tangente il loro prodotto esterno sarà un vettore normale al piano. Il vettore normale

di una superficie in forma parametrica S nel punto $P(u_0, v_0)$ è dato da

$$N = S_u \times S_v.$$

Per avere la normale unitaria:

$$N = \frac{S_u \times S_v}{\|S_u \times S_v\|}$$

Piano tangente $T_p(S)$ I due vettori tangenti S_v ed S_u in P_0 individuano, insieme al punto P_0, il piano tangente alla superficie nel punto P_0. Una parametrizzazione di tale piano tangente è quindi data da

$$T_{P_0}(u, v) = P_0 + S_u(u_0)u + S_v(v_0)v.$$

Una superficie in due variabili in forma esplicita $z = f(x, y)$ si può scrivere in forma parametrica (è un caso particolare della parametrica) considerando x ed y come parametri:

$$\begin{aligned} x &= u \\ y &= v \\ z &= f(u, v). \end{aligned}$$

Rappresentazione Esplicita vs. Implicita: pro e contro

La visualizzazione di una curva/superficie esplicita parametrica si ottiene discretizzando il dominio parametrico e valutando la curva/superficie direttamente dalla sua espressione parametrica per ogni punto del dominio 1D/2D.

La visualizzazione di una curva/superficie implicita richiede invece di discretizzare il dominio 2D/3D in una griglia uniforme e valutare per tutti i punti della griglia l'espressione implicita della curva/superficie. Infine, si deve interpolare tra i valori ottenuti nei punti griglia per approssimare i punti in cui l'espressione implicita della curva/superficie si annulla. Quindi la visualizzazione delle curve/superfici implicite risulta più onerosa computazionalmente rispetto alla visualizzazione di curve/-superfici parametriche e anche meno accurata.

La rappresentazione implicita invece offre i seguenti vantaggi:

- Classificazione di punti come esterni/interni all'interfaccia (curva/-superficie): è sufficiente un controllo del segno di $\phi(x)$ per conoscere da che parte dell'interfaccia si trovi x, mentre tale operazione risulta difficile con una rappresentazione esplicita;
- Le operazioni booleane di composizione di unione, intersezione e differenza risultano molto semplici con interfacce implicite poichè si riconducono all'uso di semplici operazioni di min/max sulle funzioni implicite;
- Rappresentazioni implicite sono in grado di gestire facilmente entità geometriche di topologia arbitraria (forme complesse con buchi) e cambi nella topologia.

3.3 Curve di Bézier

Il problema fondamentale della modellazione geometrica può essere espresso nei termini di ricerca di una formulazione matematica tale da consentire la costruzione di una curva a partire da una serie di punti assegnati.

La rappresentazione di curve nella forma di Bézier è una delle più utilizzate nella computer graphics e modellazione geometrica. La curva viene definita geometricamente, questo significa che i parametri hanno un significato geometrico, sono i punti nello spazio bi-tridimensionale. Le curve di Bézier vennero introdotte da due ingegneri francesi de Casteljau (Citroen) e Bézier (Renault) alla fine degli anni '60 con l'obiettivo di progettazione automatica di componenti per case automobilistiche. Bézier ideò uno dei primi sistemi CAD, UNISURF, usato dalla casa automobilistica francese Renault.

Una curva piana di Bézier di grado n (ordine $m = n + 1$) in forma parametrica può essere definita a partire da un insieme di punti $P_i = (x_i, y_i), i = 0, .., n$, con parametro $t \in [0, 1]$, ed è rappresentata nella forma

$$C(t) = \sum_{i=0}^{n} P_i B_i^n(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n} x_i B_i^n(t) \\ \sum_{i=0}^{n} y_i B_i^n(t) \end{pmatrix} \tag{3.4}$$

dove $B_i^n(t), i = 0, ..., n$ sono i *polinomi di Bernstein* di grado n definiti nell'intervallo $[0, 1]$ dalla formula (3.5). I punti $P_i = (x_i, y_i), i = 0, .., n$, sono chiamati **punti di controllo** e formano, uniti da segmenti, il **poligono**

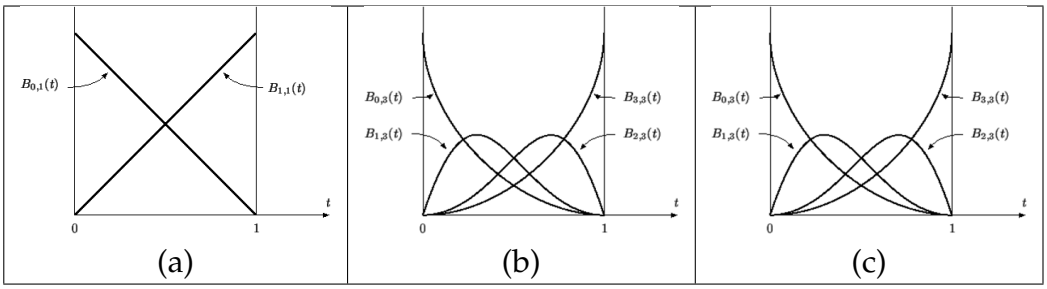


Figure 3.3: Base dei polinomi di Bernstein (a) di grado 1, (b) di grado 2, (c) di grado 3.

di controllo che approssima la forma della curva. $x(t)$ e $y(t)$ sono polinomi nella base di Bernstein.

Analogamente una curva di Bézier $C(t)$ nello spazio Euclideo $\mathbb{R}^3$ si definisce a partire da punti di controllo $P_i = (x_i, y_i, z_i), i = 0, .., n$.

I *polinomi di Bernstein* di grado n sono definiti nell'intervallo $[0, 1]$ dalla formula

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0, \cdots, n, \quad \binom{n}{i} = \frac{n!}{i!(n-i)!}. \tag{3.5}$$

I punti $P_i = (x_i, y_i), i = 0, .., n$, sono chiamati **punti di controllo** e formano, uniti da segmenti, il **poligono di controllo** che approssima la forma della curva.

I polinomi di Bernstein di grado n $\{B_i^n(t)\}_{i=0}^n$ formano una base dello spazio lineare $\mathbb{P}_n$ dei polinomi di grado al più n. Questa base, di dimensione $n+1$, è solo una delle infinite basi dello spazio dei polinomi. Un'altra base ben nota dello spazio lineare $\mathbb{P}_n$ è la base delle potenze $\{x^i\}_{i=0}^n$, mediante la quale un polinomio $p(x) \in \mathbb{P}_n$ si rappresenta come $p(x) = \sum_{i=0}^n c_i x^i$.

Vediamo alcuni esempi di polinomi di Bernstein.

Base dello spazio $\mathbb{P}_1$ dei polinomi di grado al più 1 è definita dalle funzioni:

$$B_0^1(t) = 1 - t, \quad B_1^1(t) = t,$$

ed illustrata in Fig.3.3(a).

Base dello spazio $\mathbb{P}_2$ dei polinomi di grado al più 2 è definita dalle funzioni:

$$B_0^2(t) = (1-t)^2, \quad B_1^2(t) = 2t(1-t), \quad B_2^2(t) = t^2,$$

ed illustrata in Fig.3.3(b).

Base dello spazio $\mathbb{P}_3$ dei polinomi di grado al più 3 è definita dalle funzioni:

$$B_0^3(t) = (1-t)^3, \quad B_1^3(t) = 3t(1-t)^2, \quad B_2^3(t) = 3t^2(1-t), \quad B_3^3(t) = t^3,$$

ed illustrata in Fig.3.3(c).

In un generico intervallo $[a,b]$ della retta reale la rappresentazione (3.5) dei polinomi di Bernstein assume la forma

$$B_i^n(x) = \binom{n}{i} \frac{(x-a)^i (b-x)^{n-i}}{(b-a)^n}, \quad i = 0, \cdots, n. \tag{3.6}$$

Poichè i polinomi sono invarianti per traslazione e scala dell'intervallo di definizione, sia $x \in [a,b]$ e $t \in [0,1]$, allora $x = a + t(b-a)$ e si avrà

$$B_i^n(x) = B_i^n(t), \quad i = 0, ..., n.$$

Verifica. Sia $x \in [a,b], t \in [0,1]$ perciò

$$\begin{aligned} B_i^n(x) = \quad & B_i^n(a + t(b-a)) = \\ & = \binom{n}{i} \frac{(b-a-t(b-a))^{n-i}(a+t(b-a)-a)^i}{(b-a)^n} = \\ & = \binom{n}{i} (1-t)^{n-i}(t)^i \\ & = B_i^n(t). \end{aligned} \tag{3.7}$$

Il cambio di variabile non altera i coefficienti. Se si vuole valutare $f(x)$ in un determinato valore $\underline{x}$, si determina il corrispondente valore di $\underline{t} = (\underline{x} - a)/(b-a)$ poi si valuta $f(\underline{t})$, quindi sarà $f(\underline{x}) = f(\underline{t})$.

3.3.1 Proprietà dei polinomi di Bernstein

- I polinomi di Bernstein sono tutti positivi nell' intervallo stretto $[0,1]$, $B_i^n(t) \geq 0$,
- Formano una partizione dell'unità, cioè

$$\sum_{i=0}^{n} B_i^n(t) = 1, \forall t \in [0,1],$$

- La partizione dell'unità è una proprietà molto importante in modellazione geometrica; in particolare, se vale tale proprietà di partizione dell'unità per ogni insieme di punti $P_0, P_1, .., P_n$, e per ogni valore di t, l' espressione

$$p(t) = P_0 B_0^n(t) + P_1 B_1^n(t) + ... + P_n B_n^n(t),$$

è una combinazione affine dell' insieme dei punti. Se, inoltre, per t è in $[0,1]$, i coefficienti sono tutti non negativi ($0 \geq B_i^n(t) \geq 1$), allora $p(t)$ è anche combinazione convessa dei punti.

I polinomi nella base di Bernstein possono essere rappresentati in forma matriciale se si considera la combinazione lineare

$$p(t) = c_0 B_0^n(t) + c_1 B_1^n(t) + ... + c_n B_n^n(t)$$

in termini di prodotto scalare

$$p(t) = \begin{pmatrix} c_0 & ... & c_n \end{pmatrix} \begin{pmatrix} B_0^n(t) \\ ... \\ B_n^n(t) \end{pmatrix}.$$

Un polinomio $p(t) \in \mathbb{P}_n$ rappresentato nella base delle potenze $\{1, t, t^2, ..., t^n\}$ può essere rappresentato nella base dei polinomi di Bernstein $\{B_0^n, ..., B_n^n\}$ e vice versa

$$p(t) = \sum_{i=0}^{n} c_i B_i^n(t) = \sum_{i=0}^{n} a_i t^i,$$

tramite le conversioni delle funzioni base:

$$B_i^n(t) = \sum_{k=i}^{n} (-1)^{k-i} \binom{n}{k} \binom{k}{i} t^k,$$

$$t^i = \sum_{k=i-1}^{n} \binom{k}{i} \binom{n}{i}^{-1} B_i^n(t).$$

Esempio. Sia $n = 2$. Allora

$$\begin{aligned} p(t) = \quad & c_0 B_0^2(t) + c_1 B_1^2(t) + c_2 B_2^2(t) = \\ & = c_0(1-t)^2 + c_1 2(1-t)t + c_2 t^2 = \\ & = c_0(1 - 2t + t^2) + c_1 2t - c_1 2t^2 + c_2 t^2 = \\ & = (c_0 - 2c_1 + c_2)t^2 + (2c_1 - 2c_0)t + c_0 \cdot 1 \end{aligned} \tag{3.8}$$

La forma matriciale del polinomio (3.8) in forma di Bernstein è data da:

$$p(t) = \begin{pmatrix} c_0 & c_1 & c_2 \end{pmatrix} \begin{pmatrix} 1 & -2 & 1 \\ 0 & 2 & -2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \end{pmatrix}$$

Il polinomio nella base delle potenze in forma matriciale si scrive:

$$p(t) = \begin{pmatrix} a_0 & a_1 & a_2 \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \end{pmatrix}$$

Allora

$$\begin{pmatrix} a_0 & a_1 & a_2 \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 \end{pmatrix} \begin{pmatrix} 1 & -2 & 1 \\ 0 & 2 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

Se invece dalla forma nella base delle potenze si vuole passare a rappresentare il polinomio in forma di Bernstein, si possono utilizzare i coefficienti dati da

$$\begin{pmatrix} c_0 & c_1 & c_2 \end{pmatrix} = \begin{pmatrix} a_0 & a_1 & a_2 \end{pmatrix} \begin{pmatrix} 1 & -2 & 1 \\ 0 & 2 & -2 \\ 0 & 0 & 1 \end{pmatrix}^{-1}$$

La curva derivata prima di una curva di Bézier di grado n è una curva di grado $n-1$:

$$C'(t) = \sum_{j=0}^{n-1} n(P_{j+1} - P_j) B_j^{n-1}(t). \tag{3.9}$$

Esempio.

Vogliamo determinare le derivate di una curva di Bézier di grado $n = 3$:

$$C(t) = (1-t)^3 P_0 + 3t^2(1-t)P_1 + 3t(1-t)^2 P_2 + t^3 P_3$$

Questa curva consiste di due o tre polinomi cubici in t. Deriviamo la curva $C(t)$ applicando la derivazione dei polinomi:

$$C'(t) = -3(1-t)^2 P_0 - 6(1-t)tP_1 + 3(1-t)^2 P_1 - 3t^2 P_2 + 6t(1-t)P_2 + 3t^2 P_3$$

riscrivendola raccogliendo

$$C'(t) = 3(1-t)^2(P_1 - P_0) + 3 \cdot 2(1-t)t(P_2 - P_1) + 3t^2(P_3 - P_2)$$

L'espressione ottenuta per la curva derivata prima è equivalente ad aver applicato direttamente la formula (3.9).

Osserviamo che per $t = 0$ si ha $C(0) = 3(P_1 - P_0)$, mentre per $t = 1$, si ha $C(1) = 3(P_3 - P_2)$.

Il poligono di controllo è tangente alla curva nei punti estremi della curva stessa. La direzione della tangente è rappresentata dal primo ed ultimo segmento del poligono di controllo.

3.3.2 Proprietà di una curva di Bézier

Una curva di Bézier di grado n in $[0,1]$ gode delle seguenti proprietà:

- La curva interpola il primo e l'ultimo punto di controllo:

$$C(0) = P_0, \quad C(1) = P_n;$$

- La curva è continua ed ha derivate continue di tutti gli ordini (questo

è motivato dal fatto che è costituita da componenti polinomiali);
- La curva è tangente agli estremi al primo ed ultimo segmento del poligono di controllo;
- Invariante rispetto a trasformazioni affini: operando una trasformazione affine al poligono di controllo si ottiene la medesima trasformazione sulla curva; sia T una trasformazione affine, allora

$$T(C(t)) = \sum_{i=0}^{n} T(P_i) B_i^n(t);$$

- La curva è contenuta nel *guscio convesso* formato dai suoi punti di controllo;
- Precisione lineare: se tutti i punti di controllo sono allineati lungo una linea, la curva stessa è una retta passante per i punti stessi (segue dalla proprietà del guscio convesso);
- E' variation diminishing : la curva di Bézier non ha più intersezioni con una qualsiasi retta di quante non ne abbia il suo poligono di controllo. Questo garantisce che le oscillazioni della curva siano al più quelle rappresentate dal suo poligono di controllo.

3.3.3 Valutazione di una curva di Bézier

Per il disegno della curva sarà necessario valutare la curva in un certo numero di valori del parametro t compresi in $[0,1]$. Dato un valore del parametro t, per valutare una curva di Bézier di grado n in corrispondenza di quel determinato valore del parametro t, si possono seguire tre differenti metodi:

- valutazione tramite formula (3.4)
- forma matriciale;
- algoritmo di de Casteljau.

Nel primo metodo si valutano in t i polinomi in forma di Bernstein che rappresentano le componenti della curva $C(t) = (x(t), y(t))$. Ovvero si valutano prima i polinomi di Bernstein di grado n in t, quindi si moltiplicano per i corrispondenti punti di controllo seguendo la combinazione lineare della formula (3.4).

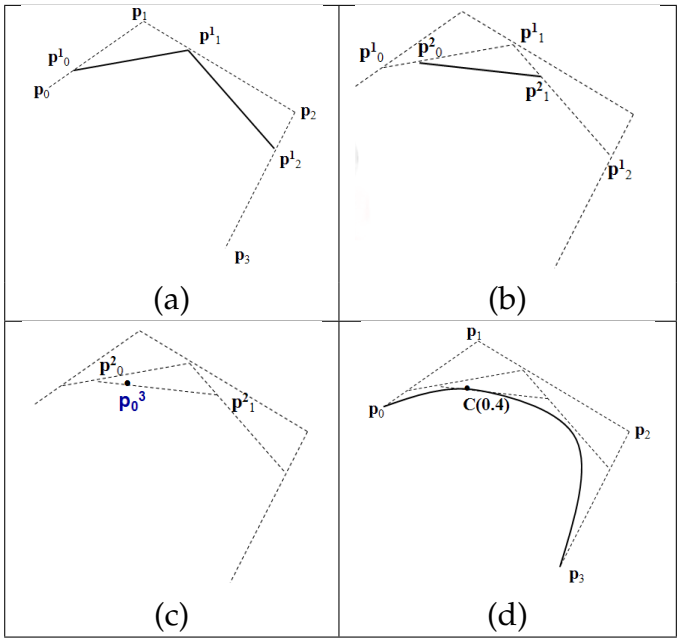


Figure 3.4: Passi dell'algoritmo di de Casteljau (a) passo 1, (b) passo 2, (c) passo 3, (d) punto sulla curva $C(0.4)$.

Algoritmo di de Casteljau

L'algoritmo introdotto dal francese de Casteljau descrive la curva mediante una serie ricorsiva di interpolazioni lineari.

L'interpolazione lineare (*Lerp*) è una tecnica per generare un nuovo valore compreso tra due valori. Un valore potrebbe essere un numero, un vettore, un colore.. Consideriamo l'interpolante tra due punti a e b con parametro t

$$Lerp(t, a, b) = (1 - t)a + tb. \tag{3.10}$$

Descriviamo i passi dell'algoritmo per la valutazione di una curva cubica $n = 3$ di Bézier, $C(t) = \sum_{i=0}^{3} P_i B_i^3(t)$ per il valore del parametro t.

Si illustrano i passi in Fig.3.4 per la valutazione in $t = 0.4$. A partire dai punti di controllo della curva: P_0, P_1, P_2, P_3, si eseguono i seguenti n passi:

- PASSO 1: Calcolo $P_0^1 = Lerp(t, P_0, P_1)$
  $P_1^1 = Lerp(t, P_1, P_2)$

$P_2^1 = Lerp(t, P_2, P_3)$,

- PASSO 2: Calcolo $P_0^2 = Lerp(t, P_0^1, P_1^1)$
  $P_1^2 = Lerp(t, P_1^1, P_2^2)$,
- PASSO 3: Calcolo $P_0^3 = Lerp(t, P_0^2, P_1^2)$,

Il valore della curva in t è dato da $C(t) = P_0^3$.

Come verifica vogliamo ritrovare la forma analitica della curva (**??**)

$$\begin{aligned}
C(t) &= P_0^3(t) \\
&= (1-t)P_0^2(t) + tP_1^2(t) \\
&= (1-t)[(1-t)P_0^1(t) + tP_1^1(t)] + t[(1-t)P_1^1(t) + tP_2^1(t)] \\
&= (1-t)^2 P_0^1(t) + 2t(1-t)P_1^1(t) + t^2 P_2^1(t) \\
&= (1-t)^2[(1-t)P_0^0(t) + tP_0^1(t)] + \\
&\quad 2t(1-t)[(1-t)P_1^0(t) + tP_2^0(t)] + \\
&\quad t^2[(1-t)P_2^0(t) + tP_3^0(t)] \\
&= (1-t)^3 P_0 + 3t^2(1-t)P_1 + 3t(1-t)^2 P_2 + t^3 P_3 \\
&= \sum_{i=0}^{3} P_i B_i^3(t)
\end{aligned}$$

L'algoritmo si può generalizzare a curve di Bézier di ogni grado semplicemente applicando più passi.

Per visualizzare la curva si può discretizzare l'intervallo parametrico $[0, 1]$ in modo uniforme, valutare la curva in corrispondenza di tali valori parametrici t, ed unire con segmenti i punti sulla curva così ottenuti.

In alternativa alla visualizzazione uniforme si può utilizzare la visualizzazione adattiva che prevede di ottimizzare le valutazioni della curva in base ad un criterio di planarità. Si considera un tratto di curva planare se le distanze tra i punti di controllo interni e il segmento corda tra i due punti di controllo estremi del tratto di curva sono tutte inferiori ad una tolleranza prefissata. Un tratto di curva considerato planare viene visualizzato mediante il segmento di corda che congiunge i due punti di controllo estremi, mentre la non planarità comporta la suddivisione del tratto di curva in due tratti di curva e la ripetizione ricorsiva del test di planarità su ciascun tratto.

3.3.4 Suddivisione di una curva di Bézier

Dato un valore del parametro $\hat{t}$, si vuole suddividere la curva di Bézier nel punto $C(\hat{t})$, per ottenere due curve di Bézier $C_1(t), t \in [0, \hat{t}]$ e $C_2(t), t \in [\hat{t}, 1]$ dello stesso grado e forma.

Esempio Sia assegnata una curva di Bézier $C(t), t \in [0, 1]$, cubica, definita dai punti di controllo:

$$P_0 = (4, 4), \quad P_1 = (0, 8), \quad P_2 = (8, 8), \quad P_3 = (8, 0),$$

Vogliamo suddividere la curva in $t = 0.5$.

Applichiamo il primo passo dell'algoritmo di de Casteljau (ovvero un'interpolazione polinomiale tra ciascun segmento poligonale nel punto medio)

$$\begin{aligned} P_0^1 &= 0.5P_0 + 0.5P_1 = (2, 6) \\ P_1^1 &= 0.5P_1 + 0.5P_2 = (4, 8) \\ P_2^1 &= 0.5P_2 + 0.5P_3 = (8, 4) \end{aligned}$$

Applichiamo ora il secondo passo dell'algoritmo di de Casteljau:

$$\begin{aligned} P_0^2 &= 0.5P_0^1 + 0.5P_1^1 = (3, 7) \\ P_1^2 &= 0.5P_1^1 + 0.5P_2^1 = (6, 6) \end{aligned}$$

Infine il tezo ed ultimo passo

$$P_0^3 = 0.5P_0^2 + 0.5P_1^2 = (4.5, 6.5)$$

Quest'ultimo rappresenta il punto sulla curva corrispondente a $C(0.5)$. Il primo tratto di curva $C_1(t), t \in [0, 0.5]$ sarà definito dai punti di controllo:

$$P_0, P_0^1, P_0^2, P_0^3.$$

Il secondo tratto di curva $C_2(t), t \in [0.5, 1]$ sarà definito dai punti di controllo:

$$P_0^3, P_1^2, P_2^1, P_3.$$

Il problema in generale di suddividere una curva $C(t), t \in [0, 1]$ in $\hat{t}$, in due curve di Bézier $C_1(t), t \in [0, \hat{t}]$ e $C_2(t), t \in [\hat{t}, 1]$, rappresentate anch'esse nella base di Bernstein equivale a valutare $C(t)$ nel punto $\hat{t}$ me-

diante l'algoritmo di de Casteljau e nel considerare i coefficienti intermedi della valutazione come i coefficienti di C_1 e C_2. Allora

$$C_1(t) = \sum_{j=0}^{n} P_0^{(j)} B_j^n(t), \quad C_2(t) = \sum_{j=0}^{n} P_{n-j}^{(j)} B_j^n(t). \tag{3.11}$$

Il processo di suddivisione può essere ripetuto più volte. Dopo k livelli di suddivisione si ottengono 2^k poligoni di Bézier ciascuno rappresentante un piccolo arco della curva originale. Questi poligoni convergono alla curva se k aumenta.

La suddivisione può essere utile per molte applicazioni come il disegno adattivo delle curve e l'intersezione fra curve.

3.4 Funzioni B-Spline

Le componenti $(x(t), y(t))$ di una curva di Bézier sono polinomi definiti nell'intero intervallo parametrico $[a, b]$ di definizione della curva. Questo comporta un limite per la modellazione geometrica per vari motivi:

- per descrivere forme complesse occorrono molti punti di controllo e questo impone un alto grado n della curva;
- curve di grado elevato sono inefficienti nella valutazione e numericamente instabili;
- una modifica ad un punto di controllo influenza globalmente la forma dell'intera curva.

La soluzione consiste nell'utilizzare curve polinomiali a tratti, o meglio comporre più curve di Bézier.

Consideriamo N curve di Bézier $C_i(t), i = 1, \ldots, N$, definite ciascuna in un intervallo $[0, 1]$, ciascuna delle quali è tale che $C_i(1) = C_{i+1}(0), i = 1, \ldots, N-1$. Chiameremo $C(u)$, la curva di Bézier complessiva definita in $[a, b]$, formata dall'unione dei tratti di curva $C_i(t), i = 1, \ldots, N$, e definita sull'intervallo parametrico

$$[a \equiv u_0, u_1, u_2, \ldots, b \equiv u_N]$$

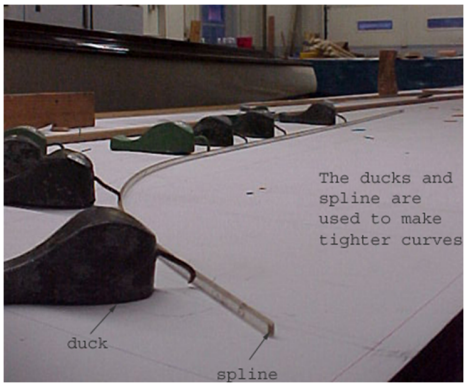


Figure 3.5: Dispositivo meccanico per il disegno: spline.

I valori u_i sono detti nodi e suddividono l'intervallo parametrico $[u_0, u_N]$ della curva $C(u)$ in N sottointervalli $I_i = [u_i, u_{i+1}], i = 1, \ldots, N$, ciascuno corrispondente ad un tratto di curva.

Come si uniscono fra loro più tratti di curva di Bézier rispettand una certa regolarità (continuità) nei punti di raccordo?

I segmenti possono essere raccordati nei nodi con un certo grado di continuità, non necessariamente lo stesso ad ogni nodo. $C(u)$ è detto essere C^k continuo nel nodo u_i se

$$C_i^{(j)}(u_i) = C_{i+1}^{(j)}(u_i), \forall j, \quad 0 \leq j \leq k,$$

indicando con $C_i^{(j)}$ la derivata di ordine j di $C_i(t)$.

Un limite evidente delle curve di Bézier a tratti è dato dal fatto che per poter congiungere più segmenti di curve di Bézier polinomiali occorre gestire 'manualmente' i raccordi.

Una spline è una funzione costituita da un insieme di polinomi raccordati in modo 'automatico' con una certa continuità tra loro.

Il termine *spline* deriva dal nome di un certo dispositivo meccanico che consisteva di una sottile asta flessibile, di un materiale elastico, fornita di scanalature ed un insieme di tiranti ("Ducks"), posizionabili lungo l'asta, preposti ad infilarsi nelle scanalature, utilizzati per tirare la spline nelle varie direzioni.

Il dispositivo, mostrato in Fig.3.5 era usato da architetti per disegnare curve regolari passanti per preassegnati punti. Basandosi su questo concetto fu introdotto da Schoenberg nel 1946, un modello matematico per definire una curva mediante dei punti che ne stabiliscano la forma.

Sia data una partizione dell'intervallo chiuso e limitato $[a, b]$,

$$(a \equiv x_0 < x_1 < x_2 < \cdots < x_{k+1} \equiv b), \tag{3.12}$$

una funzione *spline a nodi semplici* di grado n con nodi nei punti x_i con $i = 0, 1, 2, \cdots, n$ è una funzione su $[a, b]$ indicata con $s(x)$ tale che, nell'intervallo $[a, b]$ si abbia:

1. in ogni sottointervallo $[x_i, x_{i+1}[$ con $i = 0, ..., k$ la funzione $s(x)$ è un polinomio di grado n
2. la funzione $s(x)$ e le sue prime $n - 1$ derivate sono continue.

La funzione spline a nodi semplici è quindi un polinomio a tratti con la massima regolarità nei nodi tra un segmento polinomiale e il successivo. Se si vuole avere maggior flessibilità sulla regolarità in ogni nodo, si può introdurre il vettore di molteplicità associato ai nodi che permette di modificare in ogni nodo di raccordo la continuità da minima (solo C^0) a massima (C^{n-1}).

Definizione: Insieme delle spline a nodi multipli *Sia $[a, b]$ un intervallo chiuso e limitato, $\Delta = \{x_i\}_{i=1}^k$ un insieme di punti tale che*

$$a = x_0 < x_1 < \cdots < x_k < x_{k+1} = b;$$

assegnato un intero positivo m (ordine, dove $m = n + 1$, n grado), ed un vettore di interi positivi $M = \{m_i\}_{i=1}^k$, con $1 \leq m_i \leq m, \forall i = 1, .., k$. Si definisce l'insieme delle spline di ordine m con nodi Δ di molteplicità M, come

$$\begin{aligned} S(\mathbb{P}_m, M, \Delta) = \quad & \{s(x) | \exists s_0(x), ..., s_k(x) \in \mathbb{P}_m t.c. \\ & s(x) = s_i(x) \quad per \quad x \in [x_i, x_{i+1}], i = 0, .., k \\ & s_{i-1}^{(\ell)}(x_i) = s_i^{(\ell)}(x_i), \forall 0 \leq \ell \leq m - m_i - 1, i = 1, ..., k\} \end{aligned} \tag{3.13}$$

L'insieme delle spline a nodi multipli $S(\mathbb{P}_m, M, \Delta)$ è uno spazio di funzioni di dimensione $m + K$ dove

$$K = \sum_{i=1}^{k} m_i.$$

Se $m_i = 1, \forall i = 1, ..., k$ si ottiene la massima continuità, $s(x)$ di grado n risulta derivabile con continuità fino all'ordine $n - 1$ su ogni nodo x_i.

Se invece $m_i = m, \forall i = 1, ..., k$, allora poichè $m - m_i - 1 = m - m - 1 = -1$, la $s(x)$ risulta non continua nei nodi x_i, quindi siamo nel caso dei polinomi a tratti.

Si definisce inoltre vettore nodale *uniforme* se i nodi interni del vettore nodale (partizione) sono equispaziati, *non uniforme* altrimenti.

Ogni elemento dello spazio $S(\mathbb{P}_m, M, \Delta)$ cioè ogni funzione spline può essere rappresentata matematicamente come combinazione lineare di funzioni base $N_{i,m}$ dello spazio S nel seguente modo:

$$s(x) = \sum_{i=1}^{m+K} c_i N_{i,n}(x)$$

Introduciamo allo scopo il concetto di partizione estesa e le funzioni di base B-Spline $N_{i,n}(x)$.

Definizione : Partizione estesa. *L'insieme* $\Delta^* = \{t_i\}_{i=1}^{2m+K}$ *e* $K = \sum_{i=1}^{k} m_i$, *si chiama partizione estesa associata ad* $S(\mathbb{P}_m, M, \Delta)$ *se e solo se*

- $t_1 \leq t_2 \leq ... \leq t_{2m+K}$,
- $t_m = a$; $t_{m+K+1} = b$,
- $t_{m+1} \leq ... \leq t_{m+K} \equiv (\underbrace{x_1 = ... = x_1}_{m_1 volte} < ... < \underbrace{x_k = ... = x_k}_{m_k volte})$

Esempio. Sia $\Delta = \{\frac{1}{2}, \frac{3}{4}\}$ partizione in $[0, 1]$, $M = \{1, 2\}$, $m = 4$, quindi $K = 1 + 2 = 3$, e la partizione estesa è

$$\Delta^* = \{-3/4, -1/2, -1/4, 0, 1/2, 3/4, 3/4, 1, 5/4, 3/2, 7/4\}.$$

Una partizione estesa o vettore nodale si dice *aperto* (o non-periodico) quando il primo e l'ultimo nodo hanno molteplicità $n + 1$.

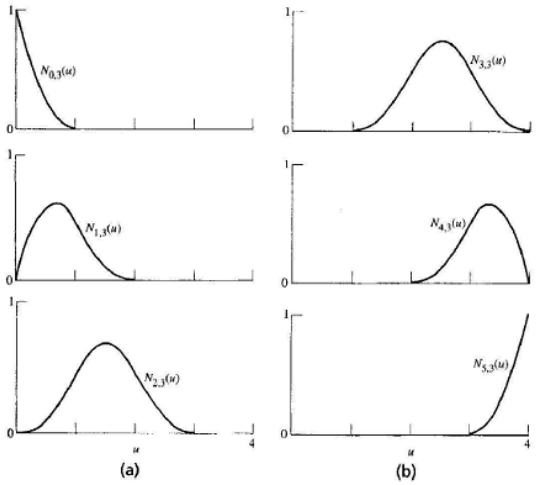


Figure 3.6: B-spline di ordine 3 con partizione nodale (0,0,0,1,2,3,4,4,4)

Il modo più efficiente computazionalmente per rappresentare le funzioni di base B-Spline è la seguente formula ricorrente.

Definizione: Funzioni base B-Spline Normalizzate. *Sia Δ^* la partizione estesa associata allo spazio $S(\mathbb{P}_m, M, \Delta)$, l'insieme delle funzioni B-Spline normalizzate $\{N_{i,n}\}_{i=1}^{n+1+K}$ è definito dalla formula ricorrente*

$$N_{i,n}(t) = \frac{t - t_i}{t_{i+n} - t_i} N_{i,n-1} + \frac{t_{i+n+1} - t}{t_{i+n+1} - t_{i+1}} N_{i+1,n-1}(t) \qquad (3.14)$$

dove

$$N_{i,0}(t) = \begin{cases} 1 & t_i \leq t < t_{i+1} \\ 0 & altrimenti \end{cases}$$

Si noti che :

- $N_{i,0}(t)$ è una funzione a gradino unitario nell'intervallo;
- per $n > 0$,$N_{i,n}(t)$ è una combinazione lineare di due funzioni base di grado $n - 1$;
- la funzione di grado n può valere il quoziente 0/0; per convenzione definiamo tale quoziente pari a 0.

In Fig.3.6 illustriamo un esempio di funzioni B-spline di grado $n = 2$.

Le funzioni B-spline normalizzate formano un base per lo spazio $S(\mathbb{P}_m, M, \Delta)$. L'intervallo $[t_i, t_{i+n+1})$ è chiamato *supporto* della funzione base.

Proprietà delle funzioni di base B-Spline:

Supporto Locale
$N_{i,n}(t) = 0, \quad \forall t$ fuori dal supporto $[t_i, t_{i+n+1})$ se $t_i < t_{i+n+1}$,

Non Negatività
$N_{i,n}(t) > 0, \quad \forall i, n, t \in (t_i, t_{i+n+1})$ se $t_i < t_{i+n+1}$,

Partizione dell'unità
$\sum_{i=1}^{n+1+K} N_{i,n}(t) = 1, \quad \forall x \in [a, b]$.

Definizione: Funzioni spline. *Ogni funzione spline $s(x) \in S(\mathbb{P}_m, M, \Delta)$ può essere rappresentata come combinazione lineare delle funzioni base B-Spline, cioè*

$$s(t) = \sum_{i=1}^{m+K} c_i N_{i,n}(t), \quad t \in [a, b] \tag{3.15}$$

con $c_i \in \mathbb{R}$ coefficienti scalari.

In particolare, per la proprietà di non negatività delle funzioni base solo nei propri supporti locali, si ha che per ogni $t \in [t_\ell, t_{\ell+1})$, $s(t)$ è data dalla somma di al più $n+1$ funzioni base B-spline, infatti la (3.15) diventa

$$s(t) = \sum_{i=\ell-n}^{\ell} c_i N_{i,n}(t), \quad t \in [t_\ell, t_{\ell+1}). \tag{3.16}$$

Quindi la spline presenta un comportamento di tipo locale: cambiando un qualsiasi coefficiente c_i la forma cambia solo per $n+1$ intervallini nodali.

3.5 Curve spline

Definizione: Curva spline in forma parametrica. *Siano $P_i \equiv (x_i, y_i)_{i=1,\dots,ncp}$, punti di controllo in $\mathbb{R}^2$. La curva spline di grado n e punti di controllo P_i è data da*

$$C(t) = \sum_{i=1}^{ncp} P_i N_{i,n}(t) = \begin{pmatrix} \sum_{i=1}^{ncp} x_i N_{i,n}(t) \\ \sum_{i=1}^{ncp} y_i N_{i,n}(t) \end{pmatrix}. \tag{3.17}$$

dove il numero dei nodi interni è dato da $K = ncp - m$.

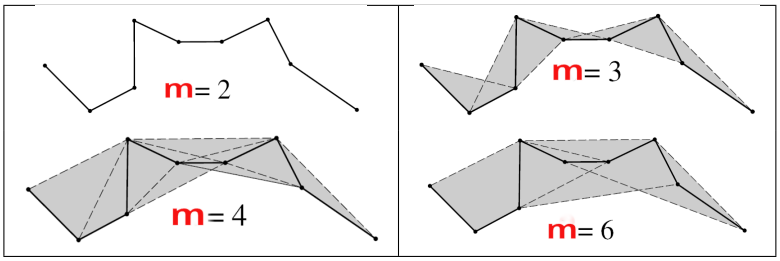


Figure 3.7: Guscio convesso per curve spline: di ordine $m = 2, 3, 4, 6$.

Dato l'ordine m, o il grado n, e il numero di punti di controllo ncp, il numero totale dei nodi si ricava da $ncp + m$, mentre il numero dei nodi interni è dato da $K = ncp - m$.

Per modificare la forma di una curva spline è possibile variare i seguenti parametri:

- Cambiare il grado
- Cambiare numero/posizione dei punti di controllo CP
- Usare CP multipli coincidenti
- Usare nodi interni multipli
- Modificare il vettore nodale

Se la partizione estesa non presenta nodi interni $(ncp = m)$ e i nodi aggiuntivi sono coincidenti, ovvero $\Delta^* = \{0, .., 0, 1, ..., 1\}$, allora la curva spline è una curva di Bézier. La proprietà di guscio convesso vista per le curve di Bézier è valida anche per le curve spline, in particolare, come illustrato in Fig.3.7 se l'ordine è $m = 2$ la spline coincide con la poligonale di controllo, mentre l'area che racchiude la curva spline, ovvero il guscio convesso si allarga all'aumentare dell'ordine della spline. Per una spline di ordine m viene calcolato come l'unione delle aree dei poligoni definiti da m vertici consecutivi, a partire dal primo.

3.5.1 Curve spline razionali (NURBS)

L'acronimo NURBS sta per "Not Uniform Rational B-Spline", traducibile in B-Spline razionali non uniformi. La parola *Rational* indica che le funzioni sono razionali, cioè polinomio fratto polinomio. Questo tipo di curve sono una generalizzazione delle curve spline e delle curve di Bézier.

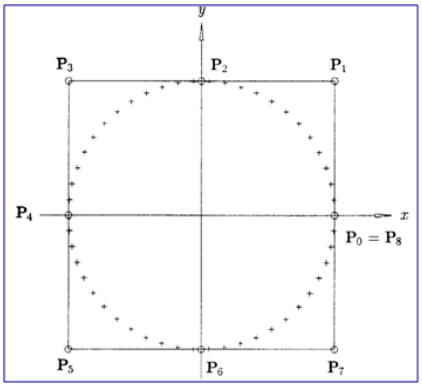


Figure 3.8: NURBS cerchio a 9 punti

Ken Versprille introdusse per primo le NURBS quando lavorava per la Computer Vision, al tempo una delle maggiori aziende CAD del mondo.

Una curva di Bézier o spline è razionale se i suoi punti di controllo sono specificati con coordinate omogenee.

Supponiamo di specificare i punti di controllo $P_i^w = (w_i x_i, w_i y_i, w_i)$. Utilizziamo la rappresentazione omogena $P_i^w = (w_i P_i, w_i)$, con $P_i = (x_i, y_i)$. I valori della curva $C^w(t)$ sono una combinazione pesata dei punti di controllo

$$C^w(t) = \sum_{i=1}^{ncp} P_i^w N_{i,n}(t),$$

e il punto sulla curva è anch'esso un punto in $\mathbb{R}^3$ definito in coordinate omogenee, ovvero una tripletta rappresentazione omogenea del punto proiettato in $\mathbb{R}^2$:

$$C(t) = \frac{\sum_{i=1}^{ncp} w_i P_i N_{i,n}(t)}{\sum_{i=1}^{ncp} w_i N_{i,n}(t)}.$$

La rappresentazione razionale presenta diversi vantaggi, quali:

- l'uso di coordinate omogenee permette di gestire un fattore peso che può essere utilizzato come ulteriore parametro di forma legato ai punti di controllo;
- l'uso dei pesi permette di rappresentare in modo esatto archi di circonferenza, ellissi, iperbole e sezioni coniche in generale;
- le curve razionali si preservano non solo per trasformazioni affini

ma anche prospettiche.

Una curva NURBS (Non-Uniform Rational B-Spline) di grado n e punti di controllo $P_i = (x_i, y_i)$ è definita

$$C(t) = \frac{\sum_{i=1}^{ncp} P_i w_i N_{i,n}(t)}{\sum_{i=1}^{ncp} w_i N_{i,n}(t)} = \left(\frac{\sum_{i=1}^{ncp} x_i w_i N_{i,n}(t)}{\sum_{i=1}^{ncp} w_i N_{i,n}(t)}, \frac{\sum_{i=1}^{ncp} y_i w_i N_{i,n}(t)}{\sum_{i=1}^{ncp} w_i N_{i,n}(t)} \right)^T, \tag{3.18}$$

dove $w_i > 0$ è il peso associato al punto di controllo P_i.

Per la proprietà di partizione dell'unità, se tutti i pesi valgono 1, $w_i = 1, i = 1, ..., ncp$, allora il denominatore vale 1 e la curva NURBS diventa una curva spline non razionale.

Le curve NURBS, modellano in maniera esatta, le curve coniche. Per esempio il cerchio a 9 punti si ottiene con la partizione estesa:

$$\Delta^* = \{0, 0, 0, 1/4, 1/4, 1/2, 1/2, 3/4, 3/4, 1, 1, 1\},$$

con pesi

$$w_i = \{1, \sqrt{2}/2, 1, \sqrt{2}/2, 1, \sqrt{2}/2, 1, \sqrt{2}/2, 1\},$$

e punti di controllo disposti come in Fig.3.8.

3.6 Patch di Bézier

Trascinando una curva di Bézier nello spazio 3D si forma una superficie. Se ogni punto di controllo si sposta lungo una curva di Bézier, allora la superficie è un patch di Bézier:

$$s(u, v) = \sum_{i=0}^{n} b_i(u) B_i^n(v) = \sum_{i=0}^{n} \sum_{j=0}^{m} b_{ij} B_j^m(u) B_i^n(v) \tag{3.19}$$

dove b_{ij} sono i punti di controllo della superficie di Bézier di grado n,m.

In Fig.3.9 è illustrato un patch di Bézier bi-cubico, con associato il suo poliedro di controllo, di vertici i punti di controllo. Un patch di Bézier interpola i quattro punti di controllo che sono gli angoli del poliedro di

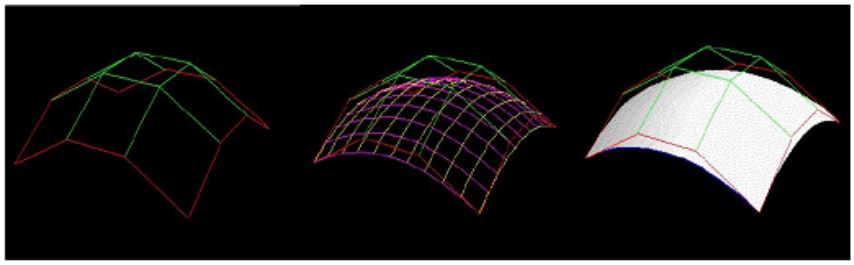


Figure 3.9: Patch di Bézier

controllo

$$s(0,0) = b_{00}, \quad s(0,1) = b_{0n}, \quad s(1,0) = b_{m0}, \quad s(1,1) = b_{mn}. \tag{3.20}$$

Per costruzione, le curve di bordo del patch sono curve di Bézier, per esempio fissando il parametro $v = 0$, si ottiene la curva

$$s(u,0) = \sum_{i=0}^{n} b_{i0} B_i(u). \tag{3.21}$$

Proprietà dei patch di Bézier:

- Guscio convesso: la superficie è contenuta nel guscio convesso dei suoi punti di controllo.
- Invarianza affine: ogni trasformazione affine dei punti di controllo definisce un nuovo patch che è la trasformazione affine del patch originale.
- Precisione lineare: se tutti i punti del poliedro di controllo sono su un piano, il patch sta sullo stesso piano.
- Approssimante in forma: la superficie di controllo suggerisce la forma del patch.

Le superfici di Bézier sono uno strumento molto potente per la progettazione di superfici, in quanto consentono di descrivere una superficie usando un numero limitato di punti. Hanno però una limitazione: per rappresentare delle forme complesse, usando una sola superficie, sono necessari molti punti di controllo. Questo significa che il grado dei polinomi usati nella rappresentazione di Bézier può diventare molto alto e

quindi difficile da trattare computazionalmente. La soluzione a questo problema è quella di utilizzare superfici di Bézier a tratti. Una superficie curva di forma complessa può essere creata affiancando e unendo tra loro diversi patch di Bézier. Il modo con cui questi patch vengono uniti può variare a seconda dell'effetto che si vuole ottenere. E' possibile generare degli spigoli sui lati di giunzione oppure imporre un passaggio dolce tra i patch. Questi risultati dipendono dal tipo di continuità con cui le superfici vengono raccordate. Modellare le superfici usando una collezione di patch di Bézier ha il vantaggio di avere un controllo locale sulla forma della superficie, ciò vuol dire che lo spostamento di un punto di controllo comporta la modifica della superficie solo in un intorno del punto, mentre le zone lontane non vengono influenzate da questa modifica. Gli stessi vantaggi delle superfici di Bézier a tratti si hanno con le superfici spline che, inoltre, evitano di dover imporre manualmente il controllo di raccordo tra ciascun patch.

3.7 Superfici spline

Creare una superficie spline implica l'ausilio del prodotto di funzioni B-spline. Siano $(\mathbb{P}_n, N, \Delta_x)$ e $(\mathbb{P}_m, M, \Delta_y)$ due spazi di funzioni spline polinomiali monovariate, di ordine n ed m con partizioni nodali $\Delta_x = \{x_i\}_{i=1}^{h}$ e $\Delta_y = \{y_i\}_{i=1}^{k}$ e vettori di molteplicità N ed M, di dimensioni rispettivamente $n + H$ e $m + K$, con $H = \sum_{i=1}^{h} n_i$, e $K = \sum_{i=1}^{k} m_i$.

Definizione. Superficie spline in forma parametrica *La superficie spline prodotto tensoriale in forma parametrica definita dai punti di controllo $P_{ij} \in \mathbb{R}^3$ e relativa alla partizione nodale $\Delta_x \times \Delta_y$, è definita da*

$$s(u, v) = \sum_{i=1}^{n+H} \sum_{j=1}^{m+K} P_{ij} N_{i,n}(u) N_{j,m}(v) \tag{3.22}$$

dove le $N_{i,n}(u)$,$N_{j,m}(v)$ sono le funzioni base B-spline di grado $n-1$ e $m-1$, con partizioni estese Δ_x^*, Δ_y^*.

I punti di controllo P_{ij} formano un array bidimensionale detto poliedero o mesh di controllo. La maggior parte delle proprietà delle curve spline valgono anche per superfici spline. Ad esempio:

- La superficie non interpola (passa) sempre il primo e l'ultimo punto.
- La superficie giace nel guscio convesso dei suoi punti di controllo.
- La continuità della superficie può essere controllata mediante i vettori di molteplicità e può essere resa indipendente in entrambe le direzioni.

3.7.1 Superfici spline razionali NURBS

Le superfici NURBS (Non-Uniform Rational B-Splines) sono una estensione delle superfici spline in grado di rappresentare accuratamente forme geometriche classiche come ad esempio un piano, una sfera, o quadrica in generale, oltre ad arbitrarie superfici a forma libera.

Grazie alla loro accuratezza e flessibilità, i modelli NURBS sono attualmente le rappresentazioni più comunemente utilizzate nella progettazione industriale. Le superfici NURBS sono superfici in forma parametrica definite a partire da punti di controllo in formato omogeneo $P_{ij}^w \in \mathbb{R}^4$ definite nello spazio omogeneo 4D:

$$s^w(u,v) = \sum_{i=1}^{H+n} \sum_{j=1}^{K+m} P_{ij}^w N_{i,n}(u) N_{j,m}(v), \tag{3.23}$$

Proiettando nello spazio 3D si ottiene:

$$s(u,v) = \begin{pmatrix} s_x(u,v) \\ s_y(u,v) \\ s_z(u,v) \end{pmatrix} = \frac{\sum_{i=1}^{n+H} \sum_{j=1}^{m+K} w_{ij} P_{ij} N_{i,n}(u) N_{j,m}(v)}{\sum_{i=1}^{n+H} \sum_{j=1}^{m+K} w_{ij} N_{i,n}(u) N_{j,m}(v)}, \tag{3.24}$$

con $s_x(u,v), s_y(u,v), s_z(u,v)$ funzioni spline scalari appartenenti ad uno stesso spazio spline prodotto tensoriale e punti di controllo $P_{ij} = (x_{ij}, y_{ij}, z_{ij})$.

Rispetto ad una superficie spline una superficie NURBS possiede dei gradi di libertà in più nella modellazione definiti dai pesi $w_{ij} > 0$. L'effetto del peso w_{ij} associato al punto di controllo P_{ij} è di attrarre i punti della superficie appartenenti all'intorno del corrispondente punto di controllo per valori crescenti del peso, viceversa, se decresce il peso $0 < w_{ij} < 1$. In Fig.3.10 mostriamo un esempio di tale effetto.

Il luogo geometrico dei punti di una superficie NURBS che hanno eguale

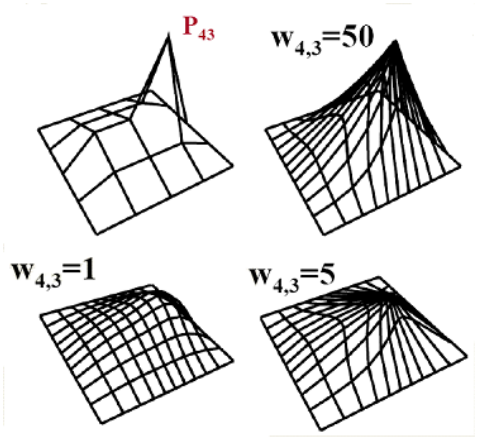


Figure 3.10: Effetto della variazione di un peso in una superficie NURBS

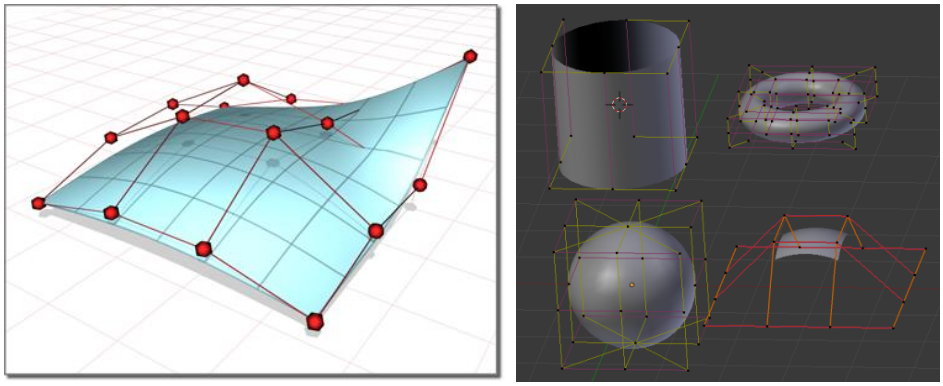


Figure 3.11: Superficie NURBS e mesh di controllo (sinistra); esempi di superfici NURBS in Blender (destra). spline.

valore del parametro u o eguale valore del parametro v è una curva NURBS che appartiene alla superficie e si dice isoparametrica. In Fig.3.11 è rappresentata una superficie NURBS con dieci curve isoparametriche in u e altrettante in v. I concetti relativi alle curve NURBS possono essere estesi a una superficie NURBS se si considerano applicati alle curve isoparametriche della superficie stessa.

Così come una curva spline può essere assimilata a una bacchetta flessibile, la superficie NURBS può essere vista come un tessuto elastico di forma rettangolare, teso tra i quattro punti di controllo che la controllano. Il

limite delle NURBS, benchè siano un ottimo strumento di modellazione geometrica, è quello di essere a "topologia rettangolare", infatti anche se è possibile rappresentare alcune forme a topologia arbitrarie collassando i punti di controllo, questo può provocare effetti indesiderati nei punti degeneri. Per far fronte a questo, si può ricorrere al trimming, o ritaglio, ossia possiamo ritagliare regioni del dominio parametrico, lasciando invariato il resto della superficie.

3.7.2 Valutazione di una superficie spline

La valutazione di una superficie spline in un punto $(\underline{u}, \underline{v})$ appartenente al dominio parametrico $U \times V$, comporta i seguenti passi:

- determinare l'intervallo nodale in U tale che $\underline{u} \in [u_l, u_{l+1})$;
- calcolare le n funzioni base non nulle in tale intervallo;
- determinare l'intervallo nodale in V tale che $\underline{v} \in [v_h, v_{h+1})$;
- calcolare le m funzioni base non nulle in tale intervallo;
- moltiplicare i valori delle funzioni base non nulle per i corrispondenti punti di controllo, mediante la formula

$$s(\underline{u}, \underline{v}) = \sum_{i=l-n+1}^{l} \sum_{j=h-m+1}^{h} P_{ij} N_{i,n}(\underline{u}) N_{j,m}(\underline{v}). \tag{3.25}$$

3.8 Superfici ottenute da curve

Dalla difficoltà di modellare la forma di superfici tridimensionali con un'interfaccia grafica e quindi una finestra in 2D, nasce l'esigenza di tecniche automatiche per costruire superfici a partire da curve. Di seguito ne elenchiamo alcune.

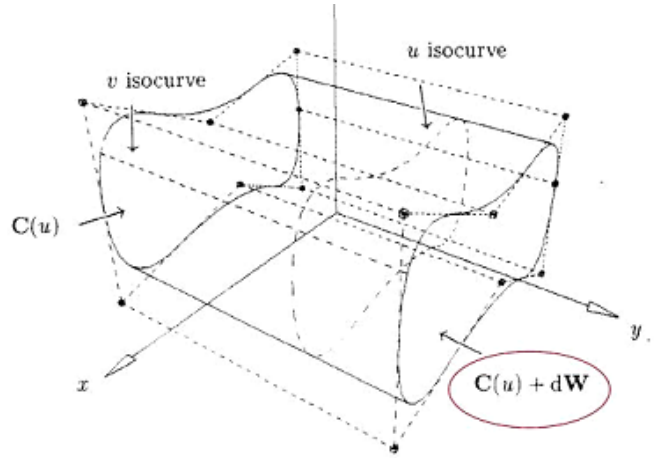


Figure 3.12: Superficie di estrusione.

3.8.1 Superfici di estrusione

A partire da un curva spline detta curva profilo,

$$C(u) = \sum_{i=1}^{n+H} P_i N_{i,n}(u) \tag{3.26}$$

di grado $n-1$, definita sul vettore nodale U, questa viene trascinata per una certa distanza d in direzione W, vettore unitario. Per un fissato $\underline{u}$ la superficie di estrusione $s(u,v)$ è un segmento retto da $C(\underline{u})$ a $C(\underline{u}) + dW$, mentre per un fissato $\underline{v}$, la superficie è data da

$$S(u,\underline{v}) = C(\underline{u}) + \underline{v}dW = \sum_{i=1}^{n+H} (P_i + \underline{v}dW)N_{i,n}(u), \tag{3.27}$$

che rappresenta la curva di profilo $C(u)$ traslata in un certo punto definito da $\underline{v}$. Come illustrato in Fig.3.12.

La funzione che rappresentazione la superficie di estrusione è data da

$$S(u,v) = \sum_{i=1}^{n+H} \sum_{j=1}^{2} P_{ij} N_{i,n}(u) N_{j,2}(v), \tag{3.28}$$

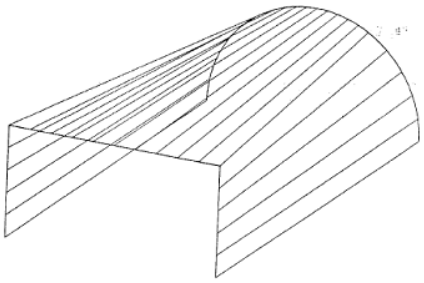


Figure 3.13: Superficie rigata

- Punti di controllo: $P_{i1} = P_i$; $\quad P_{i2} = P_i + dW$
- Vettori Nodali: $U \times V$,$V = [0, 0, 1, 1]$
- Se $C(u)$ è razionale di pesi w_i, allora anche $S(u, v)$ è razionale di pesi $w_{i1} = w_{i2} = w_i$.

3.8.2 Superficie rigata

In geometria una superficie si dice rigata se è ottenuta unendo due curve tramite segmenti di retta. Gli esempi più comuni e facili da visualizzare sono il piano, il cilindro ed il cono. Si veda un esempio illustrato in Fig.3.13. L'interesse per le superficie rigate è dovuto al fatto che la proprietà di una superficie di essere rigata è conservata dalle mappe proiettive. Anche per questo motivo, esse trovano diverse applicazioni nella geometria descrittiva ed in architettura.

Una superficie NURBS rigata è ottenuta mediante l'interpolazione lineare nella direzione v tra due curve $C_1(u)$ e $C_2(u)$ definite sulla parametrizzazione U. Per un fissato $\underline{u}$ la curva $s(\underline{u}, v)$ è un segmento che unisce le due curve. Quindi date le due curve di grado $n - 1$:

$$C_1(u) = \sum_{i=1}^{n+H} P_i N_{i,n}(u), \quad C_2(u) = \sum_{i=1}^{n+H} T_i N_{i,n}(u) \tag{3.29}$$

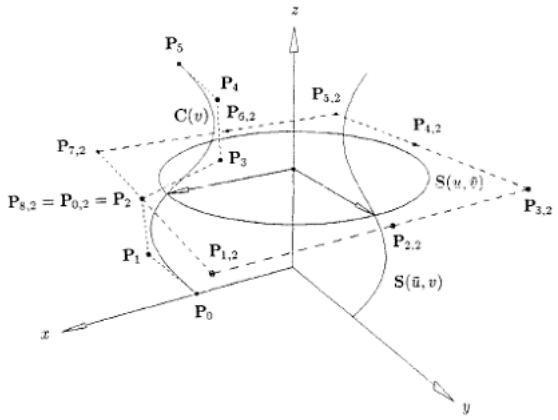


Figure 3.14: Superficie di rotazione.

L'espressione che rappresenta un superficie NURBS rigata è

$$s(u,v) = \sum_{i=1}^{n+H} \sum_{j=1}^{2} w_{ij} P_{ij} N_{i,n}(u) N_{j,2}(v), \tag{3.30}$$

- Punti di controllo: $P_{i,1} = P_i$; $\quad P_{i,2} = T_i$
- Vettori Nodali: $U \times V$,$V = [0,0,1,1]$
- Se $C_1(u)$ e $C_2(u)$ sono razionali di pesi w_i^1 e w_i^2, allora anche $s(u,v)$ è razionale di pesi $w_{i1} = w_i^1, w_{i2} = w_i^2$.

3.8.3 Superficie di rotazione

Se C è una curva nello spazio, la sua rotazione attorno ad un asse r descriverà una superficie di rotazione. La superficie di rotazione rispetto all'asse r è costituita da isocurve che sono circonferenze aventi il centro su r e giacenti su piani ortogonali ad r.

Per ottenere una superficie di rotazione, si definisce una curva profilo NURBS su un vettore nodale V

$$C(u) = \sum_{j=1}^{m+K} P_j N_{j,m}(v). \tag{3.31}$$

Per semplicità assumiamo che la curva giaccia nel piano XY e che debba essere ruotata intorno all'asse z. La superficie di rotazione avrà le seguenti caratteristiche (vedi Fig.3.14):

- Per un fissato $\underline{u}$, $s(\underline{u}, v)$ è la curva ruotata di un certo angolo rispetto all'asse z,
- Per un fissato $\underline{v}$, $s(u, \underline{v})$ è una circonferenza che giace in un piano perpendicolare all'asse z, ad un fissata quota $\underline{z}$, con centro sull'asse z.

Consideriamo la curva NURBS che rappresenta la circonferenza con nove punti di controllo su una partizione nodale

$$U = \{0, 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}, \frac{3}{4}, 1, 1, 1\}, \tag{3.32}$$

con pesi

$$W = \{1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1\frac{\sqrt{2}}{2}, 1\}, \tag{3.33}$$

La superficie NURBS di rotazione è

$$s(u, v) = \frac{\sum_{i=1}^{9} \sum_{j=1}^{m+K} w_{ij} P_{ij} N_{i,3}(u) N_{j,m}(v)}{\sum_{i=1}^{9} \sum_{j=1}^{m+K} w_{ij} N_{i,3}(u) N_{j,m}(v)}, \tag{3.34}$$

dove i punti di controllo si ottengono per rotazione dei punti di controllo della curva profilo:

$$P_{ij} = \begin{cases} P_j & i = 1 \\ \texttt{rotazione di } 45^\circ \texttt{ di } P_j & i = 2, 3, ..., 9 \end{cases} \tag{3.35}$$

I pesi della superfici sono definiti mediante il prodotto dei w_j con i pesi della circonferenza w_i definiti in (3.33).

3.9 Visualizzazione di superfici

La tassellazione è una tecnica di visualizzazione di una superficie parametrica mediante un insieme di faccette piane (triangoli o quadrilateri).

3.9.1 Tassellazione uniforme

La tassellazione uniforme si ottiene da una suddivisione a griglia uniforme del dominio parametrico con passo costante in u e in v. La mesh generata valutando la superficie $s(u, v)$ nei punti griglia è perciò uniforme ma potrebbe avere più faccette del necessario in aree più piane e meno in aree ad alta curvatura.

3.9.2 Tassellazione adattativa

Obiettivo della tassellazione adattativa è di fornire il minor numero di faccette per rappresentare accuratamente la superficie originale. Per una superficie curva questo significa avere più faccette in aree ad alta curvatura e meno in aree a bassa curvatura.

Prendiamo in considerazione un patch di Bézier e illustriamo come opera la tassellazione adattativa. Dato un patch $s(u, v)$, eseguiamo il test di planarità sui suoi punti di controllo. Se il patch non è sufficientemente piano lo suddividiamo in $v = 1/2$, quindi suddividiamo i patch risultanti in $u = 1/2$. Infine iteriamo nuovamente il test di planarità per i quattro patch ottenuti finchè ogni sotto patch non risulti approssimbile da un poligono per i suoi vertici.

Il test di planarità per un patch consiste nel costruire il piano per tre punti di controllo non allineati e misurare la distanza dei punti di controllo rimanenti dal piano. Se questa è minore di un certa tolleranza, il patch si ritiene piatto e viene approssimato dal piano stesso, altrimenti deve essere ulteriormente suddiviso.

Si potrebbe analogamente richiede un raffinamento adattivo rispetto alla camera virtuale, ovvero un raffinamento della mesh in aree più vicine alla camera e meno lontano da essa.

Chapter 4

Rendering: parte I

Il **Rendering** è un processo che genera un'immagine bidimensionale a partire da una scena di oggetti 3D e una camera virtuale o meglio, ci permette di visualizzare un mondo 3D su un dispositivo 2D.

Nell'esempio mostrato in Fig. 4.1, la fase di modellazione della scena ha prodotto il risultato di Fig. 4.1(a) e la resa della scena, ovvero la realizzazione dell'immagine da visualizzare sullo schermo è illustrata in Fig.4.1(b).

Iniziamo con l'identificare le risorse che definiscono una scena 3D illustrate nell'immagine di Fig.4.2:

- Camera virtuale, volume di vista, piano di vista
- Geometria: modelli 3D di oggetti in scena
- apparenza oggetti in scena (colore, materiale, ..)
- risorse luminose
- effetti speciali (es. ombre, effetti atmosferici)

Da questo, si può definire l'obiettivo della fase di rendering. Questa fase si occupa di localizzare i modelli geometrici 3D in scena (cioè posizionarli ed orientarli), curare un'apparenza degli oggetti in scena (colore, materiale,..), inquadrare la scena attraverso la camera virtuale, un volume di vista (frustum) e un piano di vista che conterrà l'immagine visibile su schermo, eliminare linee e superfici nascoste, gestire le luci ed eventuali effetti speciali (es. ombre, effetti atmosferici).

Ci si potrebbe chiedere perchè modellare nel 3D per poi dover proiettare

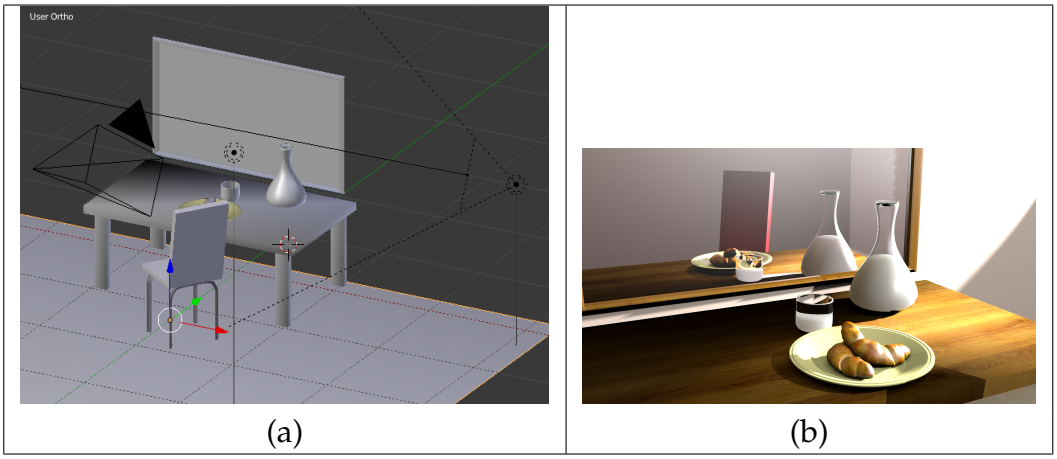


Figure 4.1: (a) modellazione 3D della scena; (b) immagine 2D della stessa scena da un certo punto di vista

Figure 4.2: Risorse che definiscono una scena 3D.

nel 2D e non modellare direttamente modelli 2D? Banalmente, il modello geometrico 3D dell'intera scena viene così definito solo una volta e si possono ottenere differenti viste solo cambiando la posizione della camera virtuale. Un buon modello rendering può a volte sostituire un complesso modello geometrico.

4.1 Forward e backward rendering

Si possono distinguere due approcci completamente opposti alla fase di rendering: il pipeline-based rendering, che potremmo definire forward rendering, e il ray-tracing o backward rendering.

Nel **pipeline Based Rendering (forward rendering)** gli oggetti in scena sono resi in una sequenza di passi che formano la rendering pipeline. Viene utilizzato il più possibile lo speciale hardware grafico per ottenere una visualizzazione real-time. Questo consente la gestione di primitive di rendering in input a questa pipeline molto semplici, come punti, segmenti e triangoli, eventualmente poligoni convessi.
Comuni modelli geometrici sono però definiti da primitive più articolate come le curve polinomiali a tratti, le superfici polinomiali a tratti o le superfici implicite (quadriche ,..). Curve e superfici sono quindi approssimate da un insieme di primitive elementari (segmenti e faccette) mediante la fase di ***tassellazione***.

Nel **ray-tracing (backward rendering)** invece, una serie di raggi sono proiettati attraverso il piano di vista e questo è colorato sulla base degli oggetti che i raggi intersecano.
Il colore di un punto sull'immagine è il risultato del contributo delle primitive che sono state proiettate in questo. Questo è un processo attualmente svolto interamente in software, quindi non certo competitivo con il forward rendering, in termini di resa non si arriva al real-time. D'altra parte però con questo approccio si riescono a gestire curve e superfici in modo esatto, senza dover procedere cioè ad una loro approssimazione in termini di segmenti o faccette e si ottengono inoltre molti effetti di illuminazione fotorealistici.

Ci occuperemo in seguito dell'approccio a pipeline di rendering o forward rendering, che viene gestito da GPU sulla scheda grafica parte in hardware e (solo dal 2003) anche mediante programmazione dei processori presenti su scheda grafica.

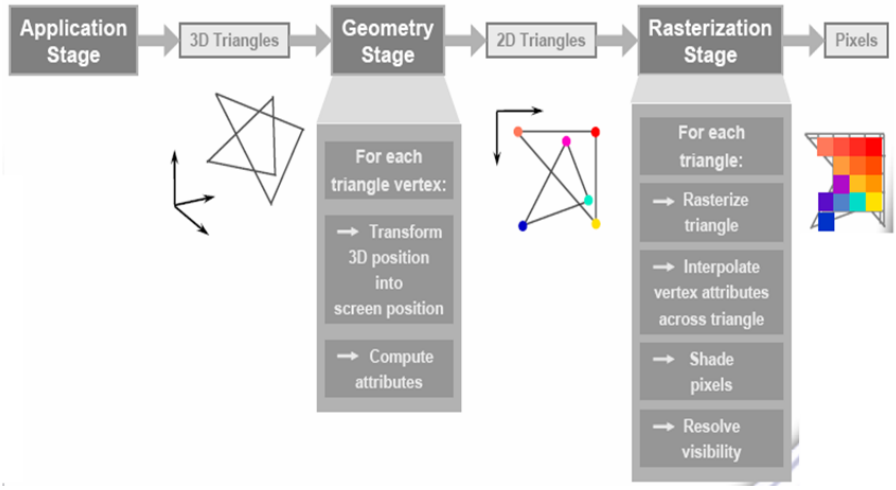


Figure 4.3: La pipeline di rendering.

4.2 Pipeline grafica di rendering

La fase di rendering (o resa) può essere decomposta in tre stadi concettuali: application, geometry e rasterizer, illustrati in Fig.4.3. Ogni stadio poi, può essere una pipeline o parallelizzato. In moderni sistemi grafici il primo stadio (Application) è interamente svolto via software (SW), mentre gli stadi geometry e rasterizer possono essere realizzati interamente in hardware o misti, hardware e software.

Il programmatore invia alla GPU le primitive geometriche che devono essere rese attraverso la pipeline grafica e le specifiche della scena, usando chiamate alle librerie grafiche API (OPENGL o Direct3D). Il geometry stage compie le operazioni sul flusso dei vertici della scena (**per-vertex**). Il rasterizer stage compie le operazioni sui pixel dell'immagine da visualizzare su schermo (**per-pixel**).

APPLICATION STAGE
: eseguito su CPU, modellazione di oggetti 3D e definizione delle caratteristiche della scena,

GEOMETRY STAGE
: su GPU, compie operazioni geometriche sui vertici dei segmenti o triangoli in input:
  - Trasformazioni di oggetti in scena

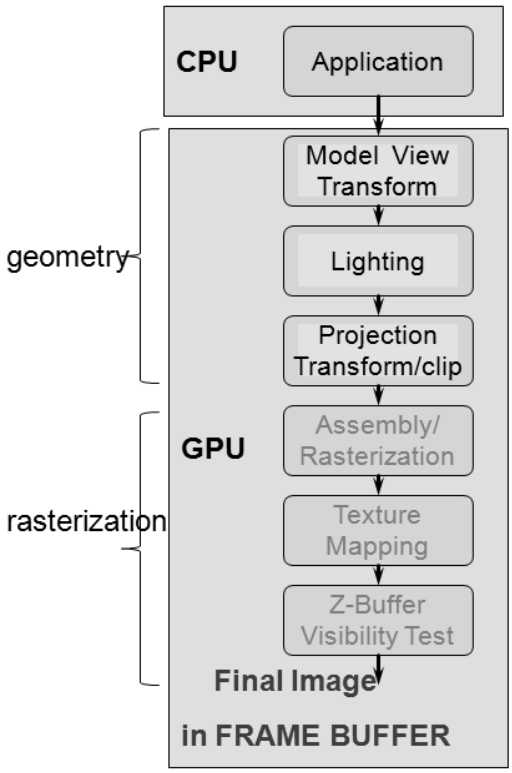


Figure 4.4: La pipeline di rendering con evidenziate le fasi di geometry e rasterization

- Trasformazioni di vista in base alla posizione della camera
- Calcolo della luminosità nei vertici dei triangoli
- Proiezione nello spazio immagine (prospettica o ortografica)
- Clipping (elimina primitive non visibili poichè fuori finestra)
- Proiezione nello spazio schermo (da 3D a 2D)
- Trasformazioni window-viewport

RASTERIZER STAGE

su GPU, calcola il colore di ciascun pixel sulla finestra schermo

- Convertire una primitiva in frammenti
- Determinare il colore dei pixel
- Gestione eventuali texture
- Determinare parti visibili
- altre operazioni sui pixel (alpha,..)

4.3 Geometry stage

Il geometry stage è il primo stadio della rendering pipeline, come mostrato in Fig.4.4. Definita una camera virtuale e una scena tridimensionale, la pipeline di rendering costruisce una serie di trasformazioni che proiettano la scena tridimensionale in un'immagine in una finestra contenuta nello schermo bidimensionale. Questo stadio della rendering pipeline può essere vista come una pipeline di trasformazioni geometriche o trasformazioni di sistemi di coordinate.

Un modello geometrico è infatti trasformato mediante trasformazioni di modellazione, vista, proiezione e viewport, ovvero trasformazioni tra vari sistemi di riferimento.

Inizialmente il modello geometrico viene creato nello spazio locale del modello. Viene quindi posizionato ed orientato in scena nel World Coordinate System (WCS) mediante **trasformazioni di modellazione** (affini) sui vertici del modello. Il WCS è unico e dopo che tutti i modelli geometrici necessari per creare una scena sono stati posizionati tutti i modelli sono rappresentati in questo spazio.

Poichè solo i modelli 'visti' dalla camera virtuale sono resi, tutti i modelli sono quindi trasformati mediante una **trasformazione di vista** nel sistema di riferimento della camera o camera frame (View Coordinate System - VCS).

Dal centro del sistema della camera solo una porzione di volume, detto volume di vista, contiene la scena che è visibile all'osservatore. Quindi il sistema di rendering elabora la **trasformazione di proiezione** per trasformare il volume di vista contenente la scena visibile dall'osservatore, in un cubo di lato 2 e diagonale di estremi $(-1,-1,-1)$ e $(1,1,1)$. Le proiezioni sono in generale ortografica o prospettica, entrambe possono essere costruite con matrici 4×4.

Queste proiezioni trasformano un volume (volume di vista) in un altro volume (cubo). La vera e propria proiezione dal volume di vista al piano di vista viene infine effettuata mediante una proiezione ortogonale su una faccia del cubo che conserva quindi le coordinate x ed y, memorizzando opportunamente la coordinata z, che rappresenta la profondità dei vertici, in una speciale memoria del frame buffer, detta Z-buffer.

Questo speciale passaggio intermedio da 3D volume di vista a cubo (sistema di riferimento normalizzato) serve essenzialmente per facilitare le operazioni di clipping (di cui si parlerà nel capitolo Rendering parte II). In breve, le primitive che risiedono interamente all'interno del volume di vista verranno disegnate sullo schermo, quelle completamente esterne verranno scartate, mentre quelle parzialmente contenute nel volume di vista verranno eliminate.

Solo le primitive interne al volume di vista sono passate all'ultima trasformazione del geometry stage, la **trasformazione di viewport** (o window-viewport o screen mapping). Quest'ultima converte le coordinate (x, y) reali di ogni vertice, in coordinate schermo espresse in pixel (device coordinate system (NDC)). Quest'ultime, insieme con la coordinata $-1 \leq z \leq 1$ forniranno l'input per lo stadio successivo della pipeline, la fase di rasterizzazione.

Riassumendo:

step 1. Composizione degli oggetti in scena in un sistema di riferimento desctrorso WCS (sistema di coordinate del mondo);

step 2. $WCS \Rightarrow VCS$ Trasformazione delle coordinate della scena da WCS al sistema di coordinate della camera (punto di vista dell'osservatore o camera (VCS)) VCS sinistrorso

step 3. Volume di vista: porzione (volume) dello spazio WCS che risulta visibile all'osservatore. Clipping: tutto quello che si trova esterno al volume risulta tagliato e quindi non visibile;

step 4. Trasformazione spazio immagine: dal volume di vista al cubo (sistema di riferimento normalizzato). Infine una proiezione ortogonale proietta $3D \Rightarrow 2D$ su un piano immagine 2D (piano di proiezione)

step 5. Trasformazione di viewport $window \Rightarrow viewport$ trasforma l'immagine contenuta nella window in coordinate pixel viewport (corrispondente finestra sullo schermo)

4.4 Trasformazioni del sistema di riferimento

Prima di addentrarci nei dettagli delle varie trasformazioni citate, rivediamo il semplice concetto di cambio del sistema di riferimento.

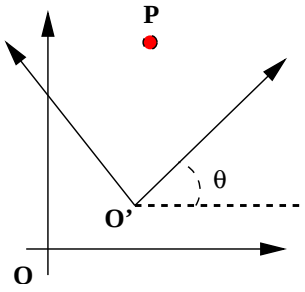


Figure 4.5: Cambio del sistema di riferimento, esempio in $\mathbb{R}^2$.

Consideriamo inizialmente il seguente esempio bidimensionale illustrato in Fig.4.5. Vogliamo definire una trasformazione da un sistema di riferimento S_1 ad un nuovo sistema S_2 che, applicata ad ogni punto P in S_1 fornisca le sue coordinate rispetto al nuovo sistema S_2.

Un primo modo per effettuare questa trasformazione è rappresentarla mediante una matrice ottenuta componendo nell'ordine, le matrici inverse delle trasformazioni elementari necessarie per portare il primo sistema nel secondo.

Il sistema S_2 ha origine $O' = (1, 2)$, è scalato di $1/2$ e ruotato di un angolo θ rispetto al sistema S_1 centrato in O. Quindi si compongono le matrici elementari

$$T = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \quad S = \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

in una matrice risultante di trasformazione:

$$T_{view} = T^{-1} S^{-1} R^T.$$

Per ottenere le coordinate $(x', y', 1)$ in S_2 di un generico punto P di coordinate $(x, y, 1)$ in S_1 basterà applicare:

$$P' = T_{view} * P.$$

Consideriamo ora il caso tridimensionale. Per convertire le coord. di un punto $P = (x_w, y_w, z_w, 1)$ espresse nel sistema WCS in termini di co-

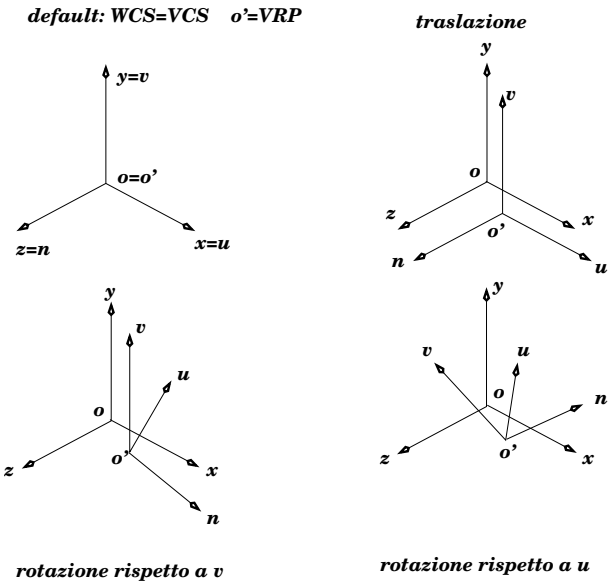


Figure 4.6: Cambio del sistema di riferimento in $\mathbb{R}^3$.

ord. $P'(x_v, y_v, z_v, 1)$ nel nuovo sistema VCS usiamo l'inversa della trasformazione che mappa l'origine del WCS in coord. VCS e mappa gli assi del WCS nei nuovi assi nel sistema VCS.

Possiamo costruire la matrice di trasformazione usando rotazioni e traslazioni, illustrate in Fig.4.6:

1. matrice T: trasla l'origine del sistema WCS nell'origine del sistema VCS o' $(T(xc, yc, zc))$
2. matrice R: gli assi x,y,z diventano u,v,n: Rotazione in v e Rotazione in u

La matrice risultante sarà quindi:

$$T_{view} = (RT)^{-1} = T(-xc, -yc, -zc)R^T$$

dove R matrice di rotazione che orienta un vettore nel sistema u,v,n rispetto al sistema originale WCS.

Ogni punto $P = (x_w, y_w, z_w, 1) \in WCS$ è trasformato in un punto $P' = (x_v, y_v, z_v, 1) \in VCS$ mediante:

$$P' = T_{view} * P.$$

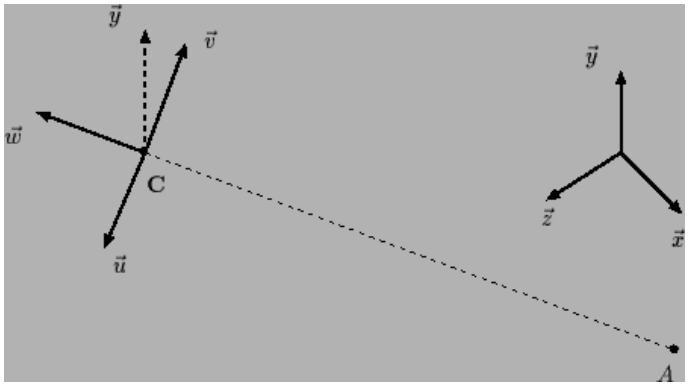


Figure 4.7: Sistema di riferimento della camera: direzione di vista e view-up vector

In alternativa, la conversione dei vertici tra due sistemi di riferimento WCS - VCS si può ottenere interpretandola come un cambio di base in $\mathbb{R}^3$ dal frame WCS al frame VCS. Tale conversione è stata trattata nella sezione 2.7.1.

4.5 Trasformazioni di vista (WCS $\Rightarrow$ VCS)

Per costruire una trasformazione di vista è necessario posizionare la camera nel WCS e orientarla opportunamente, questo permette di generare la matrice del cambiamento di sistema di riferimento. Questa matrice di trasformazione sarà quindi utilizzata per trasformare ogni vertice degli oggetti in scena dal WCS al VCS.

4.5.1 Definire un camera-frame "look at" nella posizione della camera

Definiamo un sistema di riferimento (Viewing Coordinate System (VCS)) associato all'osservatore (camera) di origine C, che stabilisce la posizione della camera (Fig.4.7):

$$\Im_{camera} = (\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{C})$$

Assegnato il punto di vista C, l' asse w del VCS è semplicemente la direzione di vista di lunghezza unitaria che punta verso la camera, o anche la normale al piano di proiezione:

$$w = \frac{C - A}{\|C - A\|},$$

dove A è il punto di attenzione , in genere il centro, della scena oseervata.

La camera (l'osservatore) dovrebbe guardare verso l'asse negativo di w. Assumiamo che la direzione verticale della camera (**view-up vector** VUP) debba essere parallela al vettore $y =< 0, 1, 0 >$ (Fig. 4.7).

L' asse unitario u che punta alla destra dell'osservatore, è perpendicolare sia all'asse w che al vettore up VUP:

$$u = \frac{y \times w}{\|y \times w\|}.$$

Un prodotto vettoriale con un vettore parallelo all'asse y può essere ottimizzato $\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \times w = \begin{bmatrix} w_z & 0 & -w_x \end{bmatrix}$.

L' asse v è ortogonale sia all' asse w che all'asse y, quindi

$$v = w \times u.$$

Notiamo che v non necessita di essere normalizzato poichè è già di lunghezza unitaria.

Si fa notare che questo è ovviamente un possibile modo di definire il sistema VCS, ne esistono altri che si vedranno nel seguito per simulare camere orbitanti, simulatori di volo,..

E quando vogliamo guardare in direzione $< 0, 1, 0 >$ o $< 0, -1, 0 >$?

In questi casi o $y = w$ o $y = -w$ e $y \times w = O$, quindi scegliamo un altro vettore da utilizzare come direzione verticale.

Vogliamo ora trovare una trasformazione che prende la rappresentazione di un punto P_w in coordinate omogenee WCS e ne fornisce la sua rappresentazione P_v in coordinate VCS :

$$P_v = T_v P_w.$$

Esprimiamo prima di tutto il sistema di riferimento della camera VCS (u, v, w, C) in termini del sistema WCS (x, y, z, O):

$$\begin{aligned} u = &\ u_x x + &\ u_y y + &\ u_z z \\ v = &\ v_x x + &\ v_y y + &\ v_z z \\ w = &\ w_x x + &\ w_y y + &\ w_z z \\ C = &\ C_x x + &\ C_y y + &\ C_z z + \ 1 \end{aligned}$$

In forma matriciale:

$$\begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ C_x & C_y & C_z & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Questo definisce una matrice 4×4 di cambiamanto del sistema di riferimento:

$$M = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ C_x & C_y & C_z & 1 \end{bmatrix}.$$

Se un punto nel frame WCS ha coordinate omogenee $P_w = (x_w, y_w, z_w, 1)$ e nel frame VCS ha coordinate omogenee $P_v = (x_v, y_v, z_v, 1)$, allora per quanto detto in sezione 2.7.1

$$P_w = M^T P_v.$$

La matrice M^T ci fornisce il cambio di coordinate da frame VCS a sistema WCS, a noi serve la matrice inversa $(M^T)^{-1}$ da WCS a VCS che chiameremo T_v:

$$P_v = (M^T)^{-1} P_w = T_v P_w$$

che determina la rappresentazione di un punto P_v in coordinate omogenee in VCS dalla sua rappresentazione in WCS.

Supponiamo di voler rendere una scena con una sedia da un certo punto di vista (camera). La sedia è posizionata in coordinate mondo con matrice T_M, la camera è posizionata in coordinate mondo con matrice T_v. La

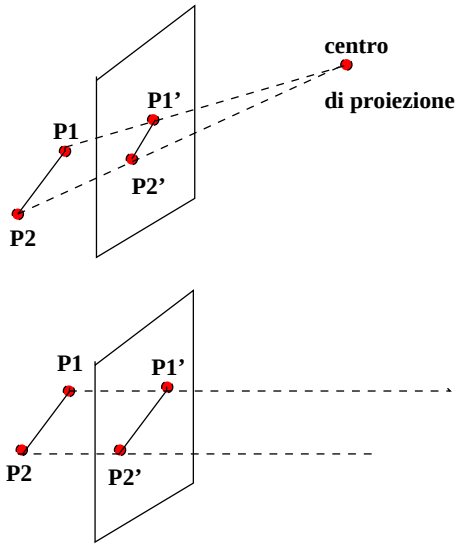


Figure 4.8: Proiezione prospettica (alto); proiezione ortogonale (basso)

seguente trasformazione prende ogni vertice dell' oggetto sedia dal sistema di coord. locali in coordinate mondo e da coordinate mondo WCS a coord. della camera VCS: $P_v = T_v T_M P_w$.

Stiamo "vedendo" gli oggetti in scena dall'origine del frame camera e la scena è stata trasformata per stare nell' asse negativo di z. Per poterli finalmente visualizzare i punti dovranno poi essere proiettati nello spazio 2D.

4.6 Trasformazione di Proiezione

Si dice **proiezione** una trasformazione geometrica con il dominio in uno spazio di dimensione n ed il codominio in uno spazio di dimensione n?1 (o minore): $f : \mathbb{R}^m \rightarrow \mathbb{R}^n, \quad m > n$. In computer graphics le trasformazioni di proiezione utilizzate sono *proiezioni geometriche piane* dallo spazio 3D (il mondo dell'applicazione) al 2D (la superficie del dispositivo di output).

L'osservatore posto in C può vedere l'immagine che si ottiene proiettando la porzione del mondo contenuta nel volume di vista (porzione di scena inquadrata) su di una window che giace sul cosiddetto piano immagine.

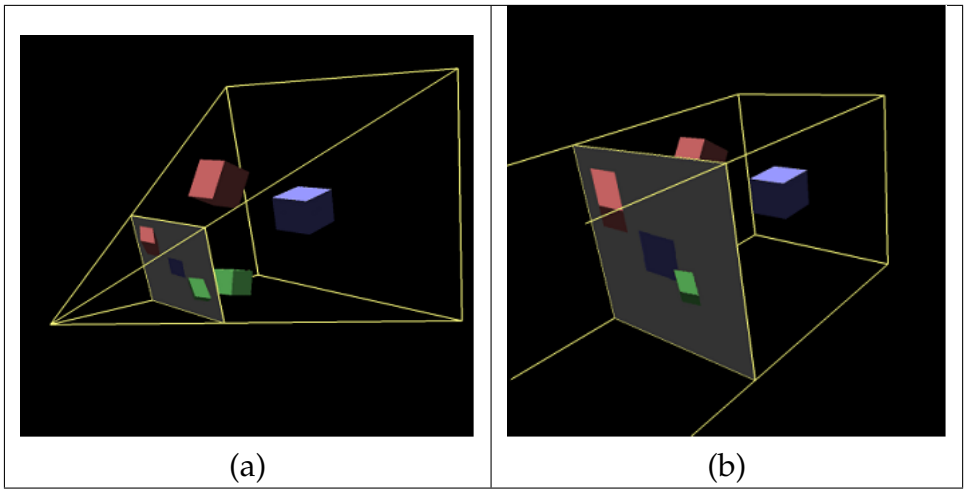


Figure 4.9: Proiezione prospettica (a); proiezione ortogonale (b)

Tutti gli oggetti in scena devono venire proiettati in questo piano. La vera e propria trasformazione da $3D$ a $2D$ avviene dopo una fase preliminare di proiezione del contenuto del volume di vista in un semplice volume, un cubo di coordinate da -1 a 1, detto sistema di riferimento normalizzato.

Iniziamo con il decidere la legge di proiezione da impostare, che di conseguenza definisce la forma del volume di vista. Da un punto di vista geometrico, la proiezione è definita per mezzo di un insieme di rette di proiezione (i proiettori) aventi origine comune in un centro di proiezione (COP), passanti per tutti i punti dell'oggetto da proiettare ed intersecanti un piano di proiezione.

Le proiezioni che si applicano sono in generale di due tipi:

1. Proiezione prospettica
   Pone l'osservatore (COP) a distanza finita. Apparentemente la dimensione degli oggetti diminuisce con la distanza riproducendo la percezione del sistema visivo umano della profondità. Il volume di vista in questo caso ha la forma di un **tronco di piramide**.
2. Proiezione ortogonale
   Pone l'osservatore (COP) a distanza infinita, i proiettori sono paralleli. Il volume di vista risulterà essere il **parallelepipedo** retto ed obliquo, delimitato dai piani near e far.

I due tipi di proiezione sono mostrati in Fig.4.9.

Una proiezione ortografica è una trasformazione affine e quindi preserva linee parallele. Nel caso di proiezione prospettica invece linee parallele non necessariamente rimangono parallele dopo la trasformazione, infatti si tratta di una trasformazione non affine bensì proiettiva.

La fase preliminare di trasformazione nel sistema di riferimento normalizzato (detto image space) porta la scena ad essere contenuta in un cubo $2 \times 2 \times 2$ centrato nell'origine che contiene punti di coordinate $-1 \leq x, y, z \leq 1$.

Punti con $z = 1$ sono vicini all'osservatore, punti con $z = -1$ sono più lontani dall'osservatore. La ragione per trasformare in questo semplice volume di vista anzichè direttamente in 2D, è che l'operazione di clipping (eliminazione dei vertici fuori dal volume di vista) è molto più efficiente in questa semplice geometria. Il clipping, una volta stabilito il volume di vista (o volume di clipping) scarta i poligoni esterni ad esso, e mantiene i poligoni interni, mentre i poligoni che intersecano il boundary del volume sono tagliati.

Le trasformazioni di vista e proiezione sono rappresentate da matrici 4×4 e quindi moltiplicate insieme e definite da un' unica matrice che è applicata poi a tutti i vertici della scena. Ogni punto $P_w = (x_w, y_w, z_w, 1) \in WCS$ è trasformato in un punto $P' = (X, Y, Z, W)$ appartenente all'image space:

$$P' = T_{proiez} * T_v * T_M * P_w.$$

Vediamo nel seguito a seconda del tipo di proiezione scelta come viene definita la matrice di proiezione T_{proiez}.

4.6.1 Proiezioni ortogonali o ortografiche

Le proiezioni parallele si classificano in base alla relazione esistente tra la direzione di proiezione e la normale al piano di proiezione. Se le due direzioni coincidono (i proiettori sono ortogonali al piano di vista: corrisponde a COP all'infinito) si parla di proiezioni ortografiche, altrimenti di proiezioni oblique.

Tali proiezioni non danno una vista realistica di una scena ma conservano il parallelismo e le distanze.

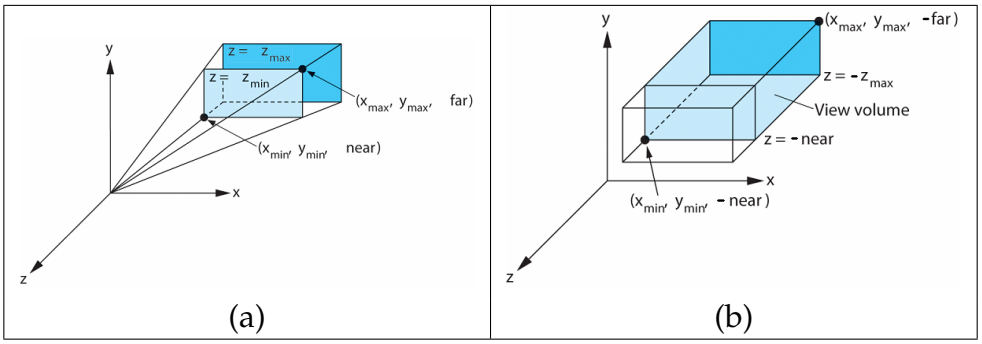


Figure 4.10: volume di vista in proiezione prospettica (a) e proiezione ortogonale (b)

Le proiezioni parallele sono pertanto molto utili quando si voglia far sí che linee parallele nel modello tridimensionale rimangano tali nella proiezione (ad esempio per effettuare misure sulla proiezione).

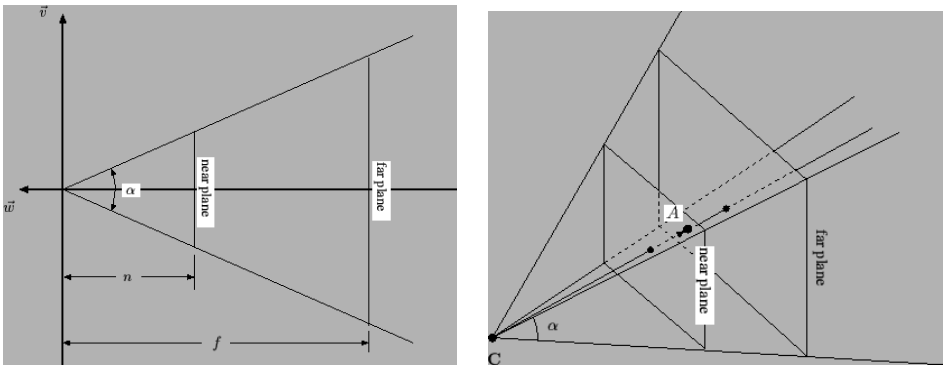
La **trasformazione preliminare** di proiezione comporta il mapping del volume di vista rappresentato da un generico parallelepipedo nell'image space (cubo centrato nell'origine).

Questo consiste in: (1) scalatura per ridimensionare il volume di vista nel cubo ($2 \times 2 \times 2$); (2) traslazione per centrare il cubo nell'origine.

Come illustrato in figura 4.10(b), il parallelepipedo che definisce il volume di vista è caratterizzato dai parametri piani near (n) e far (f) e dai vertici opposti del parallelepipedo di coordinate $(x_{min}, y_{min}, -n)$ e $(x_{max}, y_{max}, -f)$. Perciò la matrice di trasformazione sarà:

$$T_{proiez} = \begin{bmatrix} \frac{2}{x_{max}-x_{min}} & 0 & 0 & -\frac{x_{min}+x_{max}}{x_{max}-x_{min}} \\ 0 & \frac{2}{y_{max}-y_{min}} & 0 & -\frac{y_{min}+y_{max}}{y_{max}-y_{min}} \\ 0 & 0 & \frac{2}{n-f} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Dopo questa trasformazione gli oggetti in scena sono rappresentati da un *Canonical View Volume* nel sistema di riferimento normalizzato NCS.



(a) (b)

Figure 4.11: (a) Sezione del **volume di vista** in caso di proiezione prospettica; (b) Camera virtuale e volume di vista.

4.6.2 Proiezione prospettica

L'effetto di questo tipo di proiezione è di ridurre la dimensione degli oggetti lontani dal centro di proiezione rispetto ad oggetti vicini al centro di proiezione. Una vista prospettica mostra la scena da un particolare punto di vista, dando una percezione realistica tridimensionale della scena. Nella proiezione prospettica, ogni insieme di linee parallele (ma non parallele al piano di proiezione) converge in un punto detto punto di fuga.

La **trasformazione preliminare** di proiezione comporta in questo caso il mapping del volume di vista rappresentato dal tronco di piramide (view frustum) nell'image space (cubo centrato nell'origine). Vediamo ora come caratterizzare il volume di vista a partire da parametri stabiliti dall'applicazione.

In Fig. 4.11(a), e Fig.4.11(b) sono mostrati i parametri che definiscono la frontiera del volume di vista:

- **Field-of-View** α: angolo di apertura dall'apice della piramide di vista (valori tra 65 e 140 gradi);
- **Piani Near e Far**: piani perpendicolari alla direzione di vista ad una distanza n ed f dalla camera, rispettivamente.
- **aspect ratio** è il rapporto tra larghezza piano immagine e altezza.

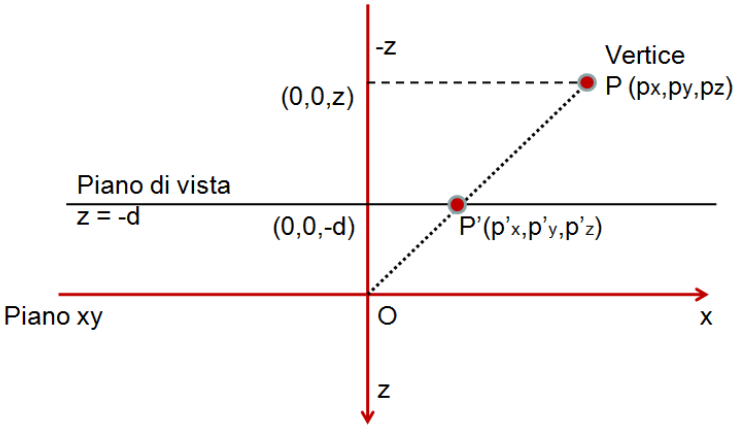


Figure 4.12: Proiezione prospettica

In relazione alle grandezze rappresentate in Fig.4.10, si hanno le seguenti corrispondenze:

$$y_{max} = n \cdot tan(\frac{\alpha}{2}); \quad y_{min} = -y_{max}; x_{max} = y_{max} \cdot aspect\ ratio; \quad x_{min} = -x_{max};$$

Il **volume di vista** è quindi definito da C, A, e α, e consiste di una piramide con la camera posizionata al vertice e il vettore (**view-plane normal**) $A - C$ che forma l'asse della piramide (vedi Fig.4.11(b)). I piani near e far formano una piramide troncata che contiene la regione dello spazio visibile all'osservatore.

La proiezione trasforma il view frustum (dai piani near $z = -n$ e far $z = -f$) nel cubo (spazio immagine)

$$-1 \leq u, v, w \leq 1.$$

Possiamo pensare ad un tronco di piramide (frustum) come ad un cubo distorto poichè ha comunque 6 facce di 4 lati ciascuna. Come rappresentare questa trasformazione matematicamente?

4.6.3 Costruzione della matrice di proiezione prospettica

Vediamo inizialmente come ottenere la matrice che rappresenta una semplice proiezione prospettica di un punto. Si osservi la Fig. 4.12. Il punto $P = (p_x, p_y, p_z)$ è proiettato sul piano $z = -d$ ad una distanza d dall'osservatore posto nell'origine. Per la proprietà di triangoli simili, si ha:

$$\frac{p'_x}{p_x} = \frac{-d}{p_z}$$

da cui $p'_x = \frac{-dp_x}{p_z}$. Il punto proiettato P' avrà quindi le coordinate

$$P' = (\frac{-dp_x}{p_z}, \frac{-dp_y}{p_z}, -d). \tag{4.1}$$

Questa è quindi una trasformazione irreversibile: tutti i punti lungo una linea di proiezione sono proiettati in un unico punto cartesiano. Una linea in coordinate omogenee infatti corrisponde ad un unico punto in coordinate cartesiane. Per definire tale trasformazione prospettica in termini di moltiplicazione matrice vettore, consideriamo la trasformazione:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix}.$$

Il punto $P' = Q\ P$ ha coordinate $(p_x, p_y, p_z, \frac{-p_z}{d})$ ovvero, per ottenere le sue coordinate omogenee dividiamo per l'ultima coordinata ottenendo: $(\frac{-dp_x}{p_z}, \frac{-dp_y}{p_z}, -d, 1)$, ovvero (4.1).

Costruiamo ora la matrice di proiezione prospettica che deforma il tronco di piramide in parallelepipedo, ovvero trasforma il volume di vista nel cubo $-1 \leq u, v, w \leq 1$, detto *volume canonico*.

Consideriamo la trasformazione:

$$T_{proiez} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

che proietta, scala i valori z (nella piramide $-n \leq z \leq -f \rightarrow -1 \leq z \leq 1$), e trasla il centro della piramide nell'origine.

I valori a e b sono scelti in modo tale che alla faccia della piramide troncata definita da $z = -n$, corrisponda la faccia $w = 1$ del volume canonico, mentre alla faccia definita da $z = -f$ corrisponda la faccia $w = -1$ del volume canonico. Questo implica che le seguenti equazioni matriciali devono essere soddisfatte:

$$T_{proiez} \begin{bmatrix} 0 & 0 & -n & 1 \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}^T$$

$$T_{proiez} \begin{bmatrix} 0 & 0 & -f & 1 \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & -1 & 1 \end{bmatrix}^T$$

Con alcuni passaggi otteniamo:

$$a = -\frac{f+n}{f-n} \quad b = -\frac{2fn}{f-n}$$

quindi la matrice di trasformazione T_{proiez} diventa:

$$T_{proiez} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Per ora abbiamo trasformato solo le facce near e far, per deformare anche le altre facce della piramide di vista (sopra, sotto, destra, sinistra) nel cubo unitario, applichiamo la nostra trasformazione trovata ai punti che delimitano la piramide di vista. In particolare, ai punti sul piano top della

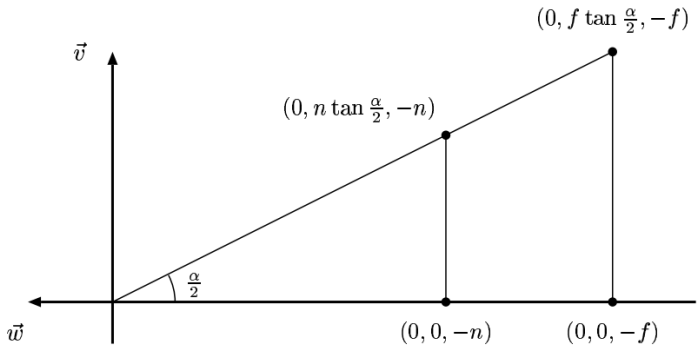


Figure 4.13: Costruzione della matrice di proiezione

piramide con la notazione illustrata in Fig.4.13, abbiamo:

$$T_{proiez}\begin{bmatrix} 0 & n\tan(\alpha/2) & -n & 1 \end{bmatrix}^T = \begin{bmatrix} 0 & n\tan(\alpha/2) & -n\frac{f+n}{f-n} + \frac{2fn}{f-n} & n \end{bmatrix}^T = \begin{bmatrix} 0 & n\tan(\alpha/2) & n & n \end{bmatrix}^T$$

$$T_{proiez}\begin{bmatrix} 0 & f\tan(\alpha/2) & -f & 1 \end{bmatrix}^T = \begin{bmatrix} 0 & f\tan(\alpha/2) & -f\frac{f+n}{f-n} + \frac{2fn}{f-n} & f \end{bmatrix}^T = \begin{bmatrix} 0 & f\tan(\alpha/2) & f & f \end{bmatrix}^T$$

Le coordinate omogenee si ottengono dividendo per la quarta coordinata.

Per avere poi le coordinate x , y pari a 1, moltiplichiamo la matrice T_{proiez} per una matrice di scalatura di un fattore $\frac{1}{\tan(\alpha/2)}$ applicata alle coordinate x ed y, come illustrato in Fig.4.13.

La trasformazione completa di proiezione è data da:

$$T_{proiez} = \begin{bmatrix} \cot\frac{\alpha}{2} & 0 & \frac{x_{max}+x_{min}}{x_{max}-x_{min}} & 0 \\ 0 & \cot\frac{\alpha}{2} & \frac{y_{max}+y_{min}}{y_{max}-y_{min}} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

La matrice T_{proiez} è applicata a punti nello spazio omogeneo. Nella matrice di proiezione prospettica, l'ultima colonna non ha ultima componente pari a 1. Questo significa che trasformando un punto 3D $\begin{bmatrix} v_x & v_y & v_z & 1 \end{bmatrix}$ non necessariamente riotteniamo un punto cioè un 1 nella 4 componente

del vettore risultante. Abbiamo invece un reale punto 4D di coordinate $\begin{bmatrix} v'_x & v'_y & v'_z & v'_w \end{bmatrix}$. Il passo finale della proiezione sarà quindi fornito dalla proiezione di punti nello spazio omogeneo (4D) in coordinate cartesiane 3D per ottenere punti di coordinate $\begin{bmatrix} v'_x/v'_w & v'_y/v'_w & v'_z/v'_w & 1 \end{bmatrix}$. Questo passo si chiama **divisione prospettica** e viene realizzato dopo aver applicato la matrice di trasformazione proiezione T_{proiez}.

4.7 Trasformazione window

Infine si deve applicare la trasformazione di window che proietta i punti dal cubo normalizzato (volume canonico) in una regione rettangolare (sistema di coordinate di window in $[-1,1]^2$). Questa è una proiezione ortografica da $3D$ a $2D$. Punti proiettati sul piano $w = 0$ mantengono le coordinate u e v ed avranno $w = 0$.

In coordinate omogenee:

$$u' = u \quad v' = v \quad w' = 0$$

in forma matriciale:

$$P_{ort} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

$$P_{ort} \begin{bmatrix} u & v & w & 1 \end{bmatrix}^T = \begin{bmatrix} u' & v' & 0 & 1 \end{bmatrix}^T.$$

Questa è quindi una trasformazione irreversibile, la coordinata z viene persa. Il valore z in realtà (profondità o depth value) è usualmente mantenuto in un valore intero a 32 bit (fixed point) da 0 (oggetti più vicini) to 0xffffffff (oggetti più lontani) nello Z-buffer contenuto nel frame buffer.

Riassumendo, ogni punto $P = (x_w, y_w, z_w, 1) \in WCS$ è trasformato in un punto $P' = (X, Y, Z, W)$:

$$P' = P_{ort} * T_{proiez} * T_v P.$$

con P_{ort} matrice di proiezione (4.2).

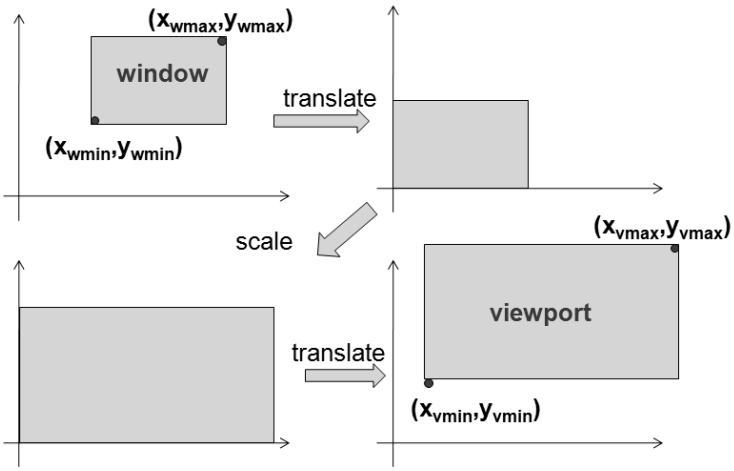


Figure 4.14: window (sopra-sinistra) e viewport (sotto-destra).

4.8 Trasformazione window-viewport

Una window ed una viewport sono aree rettangolari. La window è la finestra 2D attraverso la quale si guarda la scena 3D e che verrà visualizzata sullo schermo in un'area rettangolare detta viewport (sistema di coordinate schermo). La viewport quindi è un array di $(n_x \times n_y)$ pixel.

Alle coordinate reali di ogni vertice nella window dovranno corrispondere coppie di indici interi che individuano il corrispondente pixel nella viewport. Inoltre, anche il colore dei vertici ed eventuali coordinate texture dovranno essere approssimate in coordinate intere.

Considerando le notazioni in Fig. 4.14, per mantenere le proporzioni tra window e viewport ad un punto in window (x_w, y_w) deve corrispondere il punto in viewport (x_v, y_v) tale che:

$$\frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}} = \frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}}$$

$$\frac{y_w - y_{wmin}}{y_{wmax} - y_{wmin}} = \frac{y_v - y_{vmin}}{y_{vmax} - y_{vmin}}.$$

La trasformazione window-viewport (da coordinate reali ad intere) sarà quindi data da:

$$x_v = \lfloor S_x(x_w - x_{wmin}) + x_{vmin} \rfloor$$

$$y_v = \lfloor S_y(y_w - y_{wmin}) + y_{vmin} \rfloor$$

con fattori scala

$$S_x = L_v / L_w = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}$$

$$S_y = H_v / H_w = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}.$$

Qualora le coordinate schermo considerino l'origine coincidente con l'angolo sinistro superiore della viewport (DIRECTX) anzichè l'angolo inferiore sinistro, le coordinate window verranno ribaltate in coordinate viewport.

Per cui si ha

$$x_v = \lfloor S_x(x_w - x_{wmin}) + x_{vmin} \rfloor$$

$$y_v = \lfloor S_y(y_{wmin} - y_w) + y_{vmax} \rfloor.$$

Il fattore **aspect ratio** di window e viewport è definito come i rapporti H_w/L_w e H_v/L_v.

In termini di composizione di trasformazioni, la trasformazione window-viewport si ottiene mediante

- Traslazione di (x_{wmin}, y_{wmin}) in $(0, 0)$
- Scala di S_x, S_y
- Traslazione di $(0, 0)$ in (x_{vmin}, y_{vmin})

In termini matriciali, si ha:

$$T_{wv} = \begin{bmatrix} 1 & 0 & x_{vmin} \\ 0 & 1 & y_{vmin} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_{wmin} \\ 0 & 1 & -y_{wmin} \\ 0 & 0 & 1 \end{bmatrix}$$

Ogni pixel occupa un quadrato unitario e le coordinate intere (x_v, y_v) ne rappresentano il centro. Quindi la trasformazione window-viewport trasforma una window in $[-1, 1] \times [-1, 1]$ in una viewport in $[-0.5, n_x - 0,5] \times [-0.5, n_y - 0,5]$. La effettiva trasformazione window-viewport diventa

$$T_{wv} = \begin{bmatrix} \frac{n_x}{2} & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & \frac{n_y - 1}{2} \\ 0 & 0 & 1 \end{bmatrix}.$$

Chapter 5

Rendering: parte II

La rendering pipeline ha come input un insieme di vertici che specificano la geometria degli oggetti definiti dall'utente, e fornisce in output un array di pixel opportunamente colorati nel frame buffer. All'interno di questa pipeline si possono identificare due principali fasi: geometry stage e rasterizer stage all'interno delle quali avvengono operazioni quali le trasformazioni, clipping, shading, hidden-surface removal e discretizzazione delle primitive grafiche che poi appariranno sul display.

Le variabili in gioco sono quindi l'insieme dei vertici di ogni primitiva geometrica e l'insieme dei pixel; gli algoritmi della pipeline devono elaborare ogni primitiva geometrica (GEOMETRY STAGE) e assegnare un valore ad ogni pixel del frame buffer (RASTERIZER STAGE). Le principali fasi della geometry stage sono state considerate nel capitolo precedente *Rendering parte I* di questa trattazione. In questo capitolo si tratteranno gli

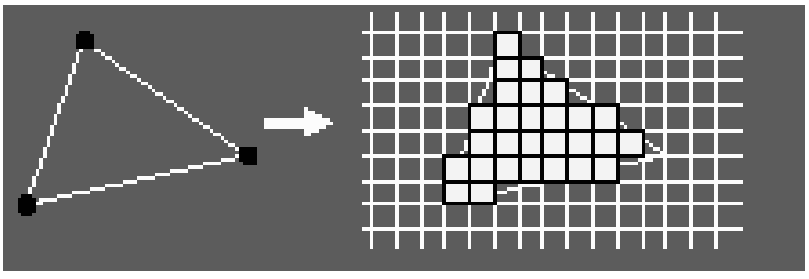


Figure 5.1: Scan-conversion

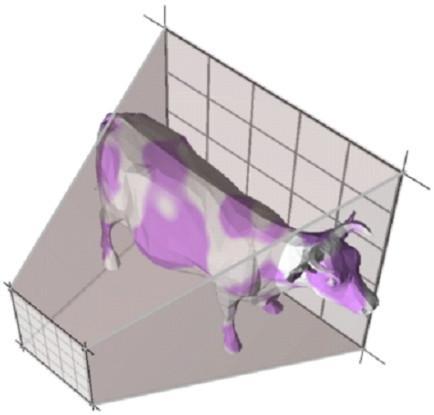


Figure 5.2: Clipping nel volume di vista definito da una proiezione prospettica.

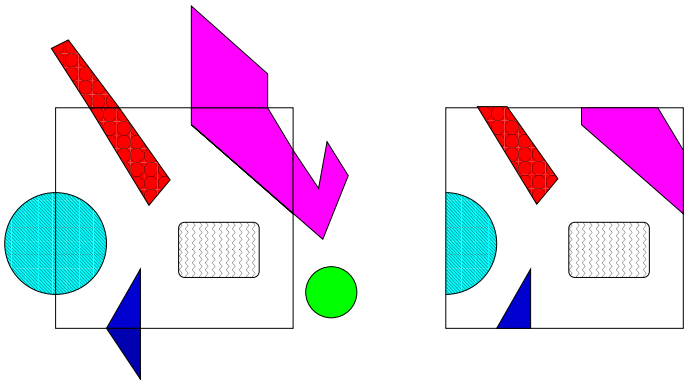


Figure 5.3: clipping bidimensionale

algoritmi di clipping del geometry stage, quindi considereremo gli algoritmi che intervengono nella fase di rasterizzazione. Questa fase infatti si occupa principalmente del problema della visibilità degli oggetti rispetto ad un dato punto di vista e della scan-conversion ovvero conversione delle primitive geometriche in pixel (Fig.5.1).

5.1 Fase di clipping

Il **clipping** è il procedimento che rimuove le parti degli oggetti che rimangono fuori dal volume di vista permettendo quindi di migliorare le performance del resto delle procedure nella pipeline di rendering.

Durante la fase di clipping si determinano quali primitive, o parti di primitive, entrano nel volume di vista (o volume di clipping) e quindi saranno visualizzate nella window (Fig.5.2). E' un'operazione che si effettua nel corso del geometry stage: i vertici che passano il clip (insieme a quelli che sono generati a seguito del clip) continuano nella pipeline, mentre i vertici che non passano il clip sono scartati.

Il clipping è eseguito dopo la normalizzazione della proiezione, ovvero dopo che il volume di vista (un tronco di piramide per proiezioni prospettiche e un parallelepipedo per proiezioni parallele) è stato trasformato canonical view volume (cubo centrato nell'origine con lati di lunghezza 2). L'algoritmo di clipping quindi opera nel Normalized Device Space (spazio immagine) prima della divisione prospettica e non nello spazio schermo 2D.

Una volta stabilito il volume di clipping e quindi prima della proiezione ortografica finale $3D \rightarrow 2D$, i poligoni esterni ad esso sono **scartati**, i poligoni interni sono **mantenuti**, mentre i poligoni che intersecano il boundary del volume sono **tagliati** e solo la parte interna al volume viene mantenuta (vedi Fig.5.3).

5.1.1 Clipping di punti e segmenti

Per semplicità espositiva la seguente trattazione riguarda algoritmi in 2D, sebbene realmente questi algoritmi vengono applicati al 3D e implementati, come già detto, in NDS (spazio immagine 3D).

Per convenzione il bordo della window ($[Xwmin, Xwmax] \times [Ywmin, Ywmax]$) e' parte della window, quindi punti o linee sul bordo sono da considerarsi interni.

Per determinare la visibilità di un punto (x, y) e quindi eventualmente

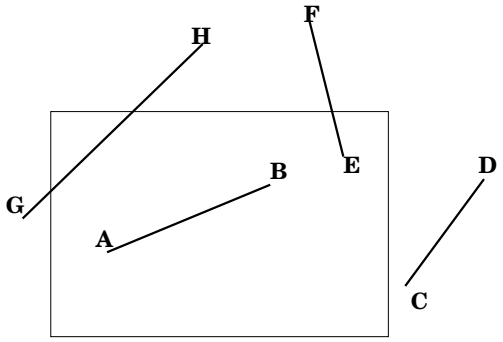


Figure 5.4: clipping di segmenti

scartarlo, si applica banalmente il test di visibilità alle sue coordinate:

$$Xwmin \leq x \leq Xwmax \quad e \quad Ywmin \leq y \leq Ywmax.$$

Qualora una qualsiasi di queste disuguaglianze non valesse, il punto è fuori dal rettangolo di clipping.

Consideriamo ora un metodo per il clipping di segmenti: l' **algoritmo di Cohen e Sutherland**. L'idea dell'algoritmo è la seguente:

- Determina se il segmento giace interamente nello schermo (estremi entrambi interni) oppure se può essere scartato in quanto completamente esterno.
- Se nessuno di questi due casi viene soddisfatto allora il segmento viene diviso in due parti (attraverso le intersezioni del segmento con i lati della window) e gli stessi test vengono applicati a ciascuna delle due parti.

Consideriamo il piano contenente la window suddiviso in nove settori di cui il centrale rappresenta la window stessa. Ogni settore è identificato da un codice a 4 bit, detto **outcode**, e mostrato in Fig.5.5.

Ogni settore è univocamente identificato da 4 bit secondo il seguente schema:

- **bit 1**: 1 se $y > Ywmax$ o altrimenti
- **bit 2**: 1 se $y < Ywmin$ o altrimenti
- **bit 3**: 1 se $x > Xwmax$ o altrimenti
- **bit 4**: 1 se $x < Xwmin$ o altrimenti

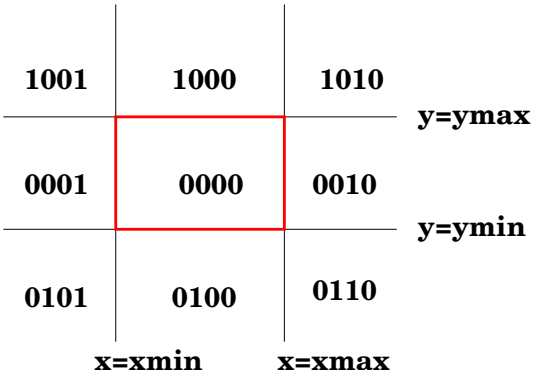


Figure 5.5: Suddivisione dello spazio window in 9 settori e assegnazione outcode

Ad ogni estremo (x, y) del segmento viene associato il proprio outcode indicante in modo univoco il settore di piano a cui appartiene.

Algoritmo di Cohen e Sutherland

Determinare gli outcode per gli estremi del segmento: $O_1 = outcode(x_1, y_1), O_2 = outcode(x_2, y_2)$;

CASO 1 $O_1 = O_2 = 0000$, entrambi gli outcode sono oooo, il segmento e' INTERNO, fine;

CASO 2 $O_1 \& O_2 \neq 0000$: i due punti estremi sono entrambi sotto, sopra, a sinistra o destra della window, il segmento è ESTERNO, fine;

CASO 3 $O_1 \& O_2 = 0000$: gli estremi sono esterni, ma la linea non può essere rifiutata perchè potrebbe intersecare la window; In tal caso si considera uno dei due estremi con outcode non nullo; si determina l'intersezione fra il segmento dato e il lato della window (corrispondente al bit non nullo dell'outcode); se l'outcode dell'intersezione è 0000 si aggiorna l'estremo considerato con l'intersezione trovata; e si ripete la procedura, altrimenti il segmento è da scartare.

Esempio di clipping di segmento

Consideriamo la situazione illustrata in Fig.5.6.

1. il segmento AB non è nè interno nè esterno (CASO 3)
2. sia A l'estremo con outcode non nullo; calcolo A' intersezione di AB con X=Xwmin (retta associata al bit 4); il segmento risultante e' A'B;

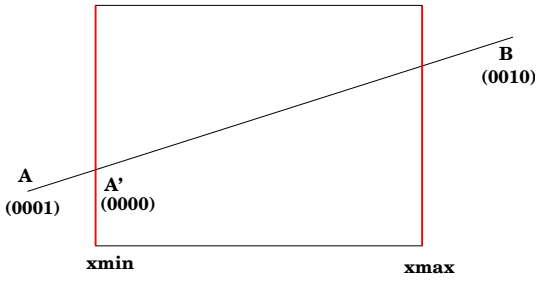


Figure 5.6: Esempio di clipping del segmento AB

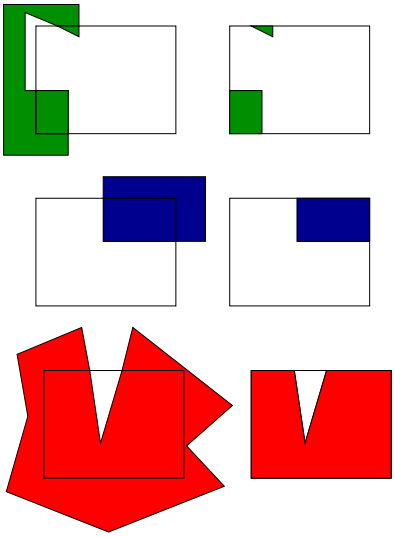


Figure 5.7: Il risultato del clipping è da uno a più poligoni: (alto) poligono concavo che viene suddiviso a seguito del clipping; (mezzo) poligono convesso; (basso) poligono concavo.

3. dagli outcode di A'(0000) e B risulta che A'B non e' nè interno nè esterno; si reitera il CASO 3
4. B è l'unico estremo con outcode non nullo, calcola B' intersezione di A'B con X=Xwmax (retta associata al bit 3); il segmento e' A'B' che è interno.

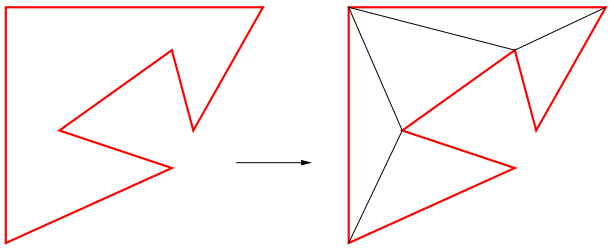


Figure 5.8: Tassellazione di un poligono concavo

5.2 Clipping di poligoni

In generale il clipping di poligoni prevede di avere in input una lista di vertici $v_1, v_2, ..v_n$ del poligono e di fornire in output uno o più poligoni (Fig.5.7). Fare clipping di un poligono è un'operazione più complessa che fare clipping di un segmento, dato che i casi da trattare sono tanti e diversi fra loro. Pertanto, gli algoritmi di clipping assumono che i poligoni siano convessi.

Il clipping di un poligono convesso su una window rettangolare ha come risultato al più un poligono convesso. Mentre il clipping di un poligono concavo può avere come risultato più di un poligono creando quindi ambiguità (Fig.5.7 in alto).

In generale i poligoni concavi vengono prima tassellati come mostrato nell'esempio in Fig.5.8 per decomporli in un insieme di poligoni convessi.

5.2.1 Algoritmo di clipping di Sutherland-Hodgman

L'algoritmo si basa sulla scomposizione del problema in sottoproblemi di clipping dove ciascun **sottoproblema** esegue il clipping del poligono con un solo lato della window. Successivi clip rispetto a tutte e quattro le rette definite dai lati della window risolvono il problema del clip del poligono rispetto all'intera window. Un esempio è illustrato in Fig. 5.9.

Algoritmo

Siano $v_1, v_2, ..v_n$ i vertici del poligono. I lati sono formati collegando v_i

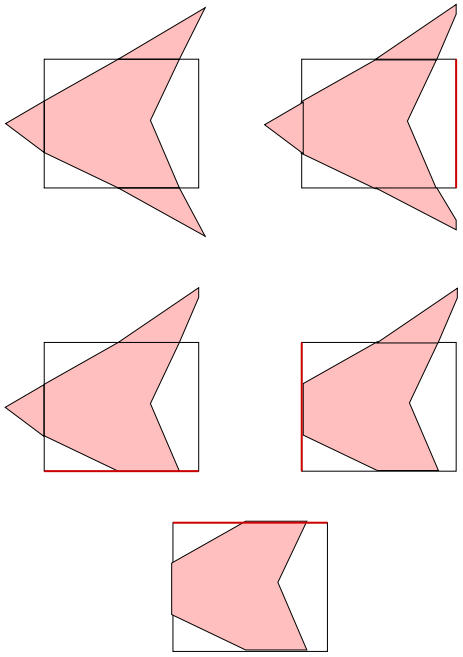


Figure 5.9: Successivi clip rispetto a tutti e quattro i bordi della window (in rosso il lato di clip considerato)

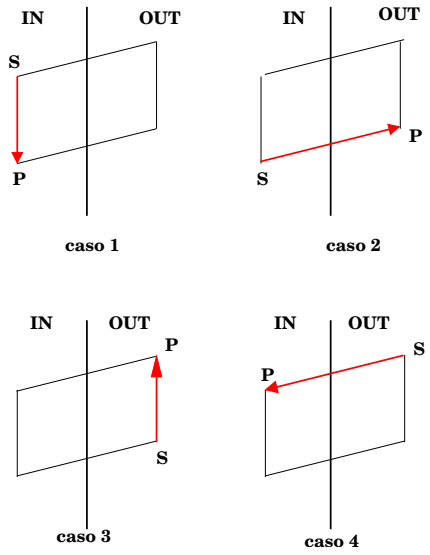


Figure 5.10: Algoritmo di Sutherland-Hodgman: casi nel clipping del lato SP del poligono con il bordo window

a v_{i+1} per $i = 1..n-1$ e v_1 con v_n. L'algoritmo taglia rispetto ai lati e restituisce un'altra successione di vertici che definisce il poligono finale $w_1, w_2, .., w_m$.

Per ogni lato della window (lato di clipping) il poligono viene scansionato in senso antiorario partendo dal vertice v_n fino a v_1 per poi tornare a v_n: si interseca ogni lato del poligono con il lato di clipping e si aggiorna la lista dei vertici w_i finali. Solo dopo aver elaborato l'ultimo vertice del poligono, si uniscono parti opportune del bordo della window per ottenere un poligono chiuso.

Situazioni che si possono presentare come risultato dell'intersezione lato clipping/lato poligono sono le seguenti. Sia SP il lato del poligono che si sta considerando. Si identificano i 4 casi mostrati in Fig.5.10:

- CASO 1: lato del poligono completamente interno $\Rightarrow$ P aggiunto alla lista dei vertici finali
- CASO 2: calcolo dell'intersezione i che viene poi aggiunta alla lista dei vertici finali
- CASO 3: vertici entrambi esterni $\Rightarrow$ non vengono aggiunti vertici
- CASO 4: calcolo del'intersezione i e vengono aggiunti i vertici i e P

5.3 Clipping in 3D

Il clipping in realtà è eseguito nel volume di vista, quindi in uno spazio racchiuso in $\mathbb{R}^3$. Gli algoritmi visti per il caso bidimensionale si estendono facilmente al caso tridimensionale con opportune accortezze.

Per esempio il test di visibilità nella regione di clipping (Fig.5.11) viene modificato nel seguente. Il punto (x, y, z) è visibile se:

$$\begin{aligned} Xwmin \leq x \leq Xwmax \\ Ywmin \leq y \leq Ywmax \\ Zwmin \leq z \leq Zwmax \end{aligned}$$

Nell'algoritmo di Cohen-Sutherland per segmenti, il clipping deve avvenire di segmenti nello spazio rispetto ad un parallelepipedo. Le regioni in cui viene diviso lo spazio sono 27 invece di 9, il codice a 4 bit (outcode) viene

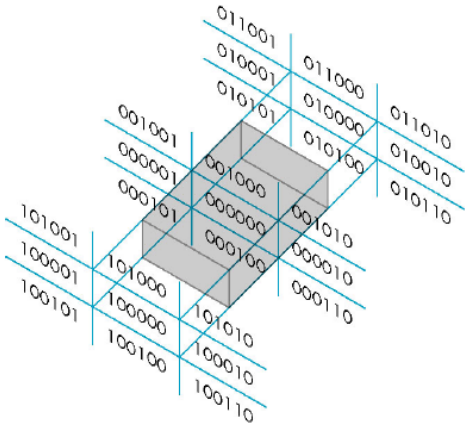


Figure 5.11: Clipping di segmenti nel volume di vista.

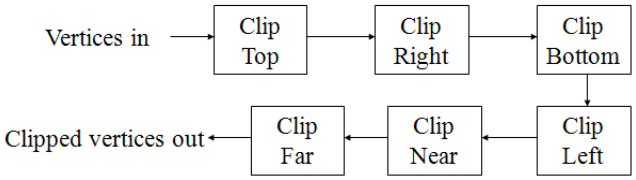


Figure 5.12: Clipping dei poligoni con algoritmo di Sutherland-Hodgman.

modificato in un codice a 6 bit per includere il test dei casi in cui il punto è davanti o dietro il volume di clipping (vedi Fig.5.11). Occorrerà infine calcolare le intersezioni retta/piano invece di retta/retta.

Il problema dell'intersezione retta passante per P_1 e P_0 e piano (passante per P_2 con vettore normale n, si riconduce alla determinazione del parametro t mediante il seguente sistema:

$$\begin{cases} P(t) = (1-t)P_0 + tP_1 \\ \\ n \cdot (P(t) - P_2) = 0 \end{cases}$$

Risolvendo si determina: $t = \frac{n \cdot (P_2 - P_0)}{n \cdot (P_1 - P_0)}$ parametro di intersezione.

Nell'algoritmo di Sutherland-Hodgman si deve applicare l'intersezione con tutte e sei le facce del volume di vista normalizzato come illustrato in

Fig.5.12. Il test per verificare se un punto P è interno o esterno (illustrato in Fig.5.10) si avvale dell'equazione del piano:

$$n \cdot (P - x) = 0,$$

dove P è un punto qualsiasi sul piano/linea e n è la normale al piano/linea rivolta verso la parte esterna. Quindi se per un certo x, si ha $n \cdot (P - x) < 0$ allora il punto x è interno, $n \cdot (P - x) = 0$ punto x sulla linea/piano, mentre $n \cdot (P - x) > 0$ punto x esterno.

5.4 Stadio di rasterizzazione

Questo stadio della pipeline comprende i seguenti algoritmi che da un insieme di vertici che definiscono oggetti geometrici (INPUT) producono frammenti (pixel) nel frame buffer (OUTPUT):

SCAN-CONVERSION
: discretizzazione delle primitive grafiche (punto, linea, poligono)

TEXTURING
: associare immagini a poligoni

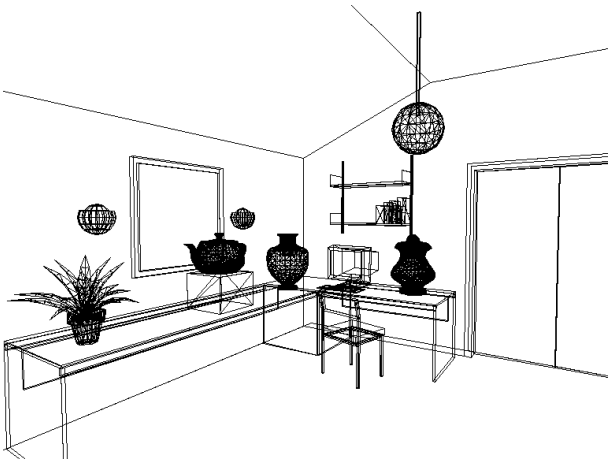
DOUBLE BUFFERING
: gestione front e back buffer.

VISIBILITA'
: algoritmi di hidden surface removal

Dopo che i vertici sono passati attraverso le trasformazioni geometriche richieste e gli oggetti definiti da essi sono stati opportunamente assemblati e 'ritagliati' (clipping), tutte queste entità geometriche (punti, segmenti, poligoni) potrebbero potenzialmente apparire sullo schermo.

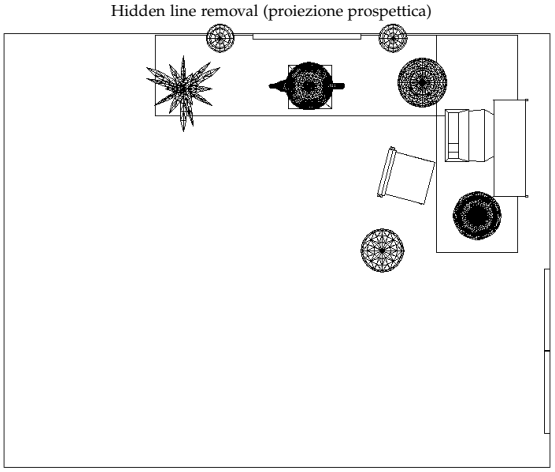
La proiezione nello schermo infatti è applicata a tutte le parti di ogni oggetto in scena, comprese quelle che dovrebbero essere non visibili da un dato punto di osservazione della scena poichè oscurate da altri oggetti in scena. Questo crea rappresentazioni scarsamente comprensibili o addirittura ambigue.

Si osservi in Fig.5.13 la differenza tra le diverse rappresentazioni wireframe della stessa scena.



Rappresentazione wireframe della scena

Hidden line removal (proiezione prospettica)



Hidden line removal (proiezione parallela)

Figure 5.13: Scena wireframe; hidden-line removal

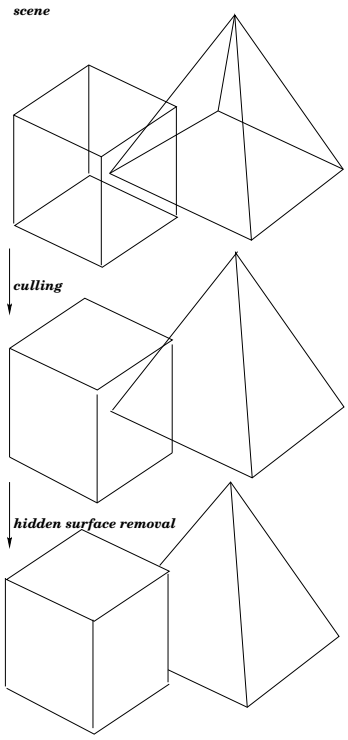


Figure 5.14: Back-face elimination (culling) e hidden surface removal

Per risolvere il problema della visibilità degli oggetti rispetto ad un punto di vista applichiamo algoritmi per rimozione delle linee nascoste (hidden-line) o delle superfici nascoste (hidden-surface) per determinare gli oggetti o parti di essi che sono visibili dalla camera cioè non oscurati da altri oggetti.

Consideriamo nel seguito solo tecniche per scene composte unicamente da poligoni piani, poichè ogni superficie, a questo punto, sarà già stata suddivisa (tassellata) in poligoni.

Distinguiamo nel paragrafo seguente la semplice operazione di culling che rimuove completi poligoni con un semplice test, dalla più sofisticata tecnica di **hidden surface removal** che si occupa del problema di poligoni che sono parzialmente oscurati da altri. Tecnica che verrà considerata in seguito. Un'idea della differenza viene mostrata in Fig. 5.14.

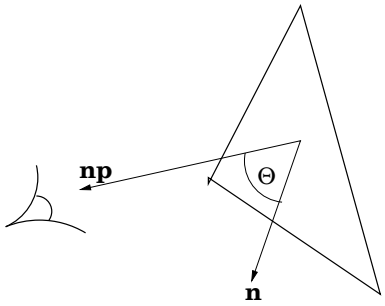


Figure 5.15: Back-face test

5.5 Culling o back-face elimination

Ci sono tre principali ragioni per selezionare ed eliminare un insieme di poligoni prima della loro elaborazione in fase di rasterizzazione.

view frustum culling
: il poligono non giace all'interno del volume di vista (algoritmi di clipping visti)

backface culling
: il poligono è nel volume di vista ma non è visibile dall'osservatore, il test viene eseguito prima del clipping

poligono degenere
: il poligono ha area nulla (il backface culling gestisce anche questo caso).

Occupiamoci allora del preliminare e semplice test, illustrato in Fig.5.15, applicato nel backface culling per decidere se considerare o scartare i poligoni. Questo algoritmo è molto veloce e permette di scartare circa la metà dei poligoni in scena in quanto risultano nascosti all'osservatore.

Si applica il **test di visibilità:**

$$visibility = n_p \cdot n > 0$$

dove n definisce la normale esterna al poligono, n_p la direzione di vista. Ovvero

$$if(n_p \cdot n) \leq 0 \quad triangolo \quad non \quad visibile.$$

Un esempio è mostrato in Fig.5.16.

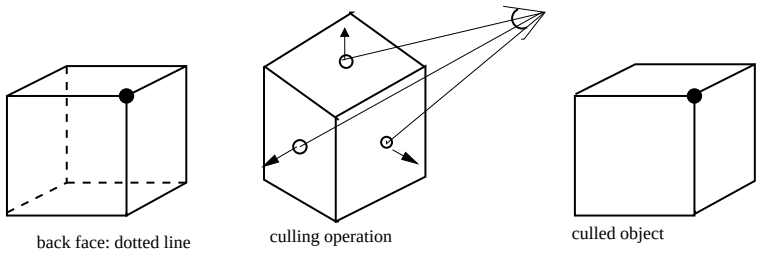


Figure 5.16: Esempio di backface culling

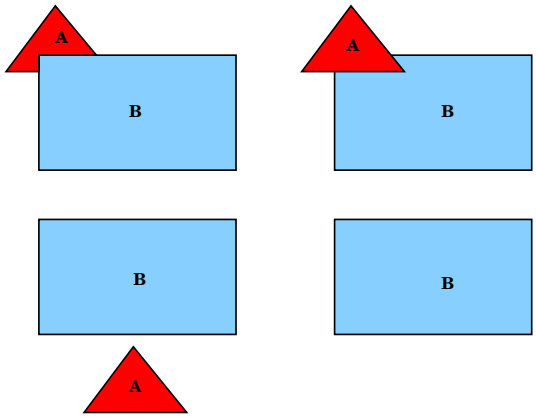


Figure 5.17: (riga in alto) B parzialmente oscura A; A parzialmente oscura B; (riga in basso) A e B sono entrambi visibili; B totalmente oscura A

Se la normale n al triangolo risulta nulla significa che l'area del triangolo è degenere, il backface culling riconosce questo caso e scarta il triangolo.

5.6 Algoritmi di Hidden Surface Removal

Il clipping e il culling hanno risolto i problemi individuati in Fig.5.17 (in basso), mentre l'algoritmo di hidden surface removal che ora consideriamo risolve i casi Fig.5.17 (in alto) ovvero determina la parte visibile di ogni oggetto.

Due approcci al problema:

spazio oggetto: algoritmi che lavorano nel sistema di coordinate in cui l'oggetto è descritto La complessità per k poligoni è data da $O(k^2)$.

spazio immagine: algoritmi che lavorano nel sistema di coordinate dello schermo su cui l'oggetto è visualizzato. La complessità per immagini $n \times m$ è nmk quindi dell'ordine di $O(k)$.

Nel seguito vengono descritti i seguenti algoritmi per l'eliminazione di parti nascoste:

- Depth-sort (painter's) algorithm
- Z-buffer algorithm
- scanline algorithm
- ray tracing algorithm

In generale la pipeline grafica tradizionale usa lo Z-buffer.

5.6.1 Depth-sort (painter's) algorithm

Deve il suo nome alla tecnica utilizzata dai pittori che consiste nel disegnare prima gli oggetti presenti sullo sfondo (in lontananza) poi si disegnano gli oggetti più vicini all'osservatore.

Disegna nel frame buffer in ordine di distanza decrescente dal punto di vista (back-to-front).

1. Ordina tutti i poligoni rispetto alla loro coordinata z più lontana dal punto di vista
2. Risolvi i problemi di sovrapposizione dei poligoni (in z)
3. Per ogni poligono della lista ordinata
   disegna ogni pixel.

Problemi di sovrapposizione sono illustrati in Fig.5.18. In questi casi si estende il test anche alla x e alla y.

Nei casi di sovrapposizione ciclica non c'è un ordine corretto, conviene dividere uno dei poligoni in due parti e iterare nuovamente.

5.6.2 Z-buffer algorithm

L'algoritmo Z-buffer, sviluppato da Catmull(1975), è uno dei più popolari algoritmi di rimozioni superfici nascoste. E' un algoritmo che opera

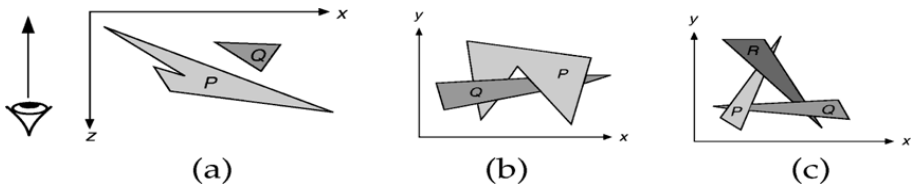


Figure 5.18: Problemi di sovrapposizione

nello spazio immagine, ovvero in screen space. Si avvale di una memoria ausiliaria, detta appunto Z-buffer presente nel frame buffer, che ha dimensione del color buffer e contiene un valore di profondità (depth-value) per ogni pixel. Il depth-value associato ad ogni pixel è la distanza dal COP al poligono più vicino lungo il raggio passante per quel pixel.

I punti (pixel) all'interno di un poligono sono disegnati nel color buffer (rasterizzati) e la loro profondità z (depth-value) è valutata per interpolazione dei valori z associati ai vertici del poligono. Durante il disegno di un poligono, si scrive il valore del colore di un dato pixel nel color buffer, solo se la sua profondità z è minore del depth-value memorizzato nello Z-buffer (cioè il depth-value fino a quel momento incontrato).

```
Inizializza Z-buffer a max distanza dal COP
(depthzbuffer[pixel]=infinito)
Per ogni poligono
        Per ogni pixel della proiezione del poligono
        se (depthpoly[pixel]< depthzbuffer[pixel]) allora
                disegna il pixel (color buffer);
                memorizza profondità del pixel (z-buffer)
```

Uno dei maggiori vantaggi dello Z-buffer è che risulta indipendente dalla forma di rappresentazione degli oggetti e dal numero di oggetti in scena e non richiede alcun ordinamento degli oggetti.

5.6.3 Scanline algorithm

Generazione dell'immagine in ordine di linee di scansione.

Algoritmo N.facce pol. in scena	100	2500	60000
Depth-sort	1	10	507
Z-buffer	54	54	54
Scanline	5	21	100

Table 5.1: Performance degli algoritmi di hidden line removal

```
Per ogni linea di scansione
        determina i punti di intersezione della linea
        di scansione con tutti i lati dei poligoni;
Ordina questi punti rispetto x;
(* una parte di scanline tra due pti di inters. è detta span,
lungo una span la visibilità non cambia *)
Determina le span visibili
```

Il calcolo della profondità è fatto solo per span di diversi poligoni che si sovrappongono.

In tabella 1 sono mostrati i risultati del confronto tra i tre metodi di hidden-surface removal in termini del numero delle facce poligonali da rendere. Si noti come l'efficienza dell'algoritmo z-buffer sia indipendente dalla complessità della scena.

5.6.4 Ray-tracing

Il ray tracing è considerato un approccio backward rendering che si contrappone alla rendering pipeline (forward rendering), ed è una tecnica particolarmente adatta alla rimozione di parti nascoste per superfici curve.

L'idea sulla quale si basa l'algoritmo è la seguente. Tra i raggi di luce che dalla sorgente luminosa vanno verso la scena alcuni non colpiscono oggetti, altri li colpiscono e li illuminano. Questi raggi intersecano la superficie degli oggetti riflettendosi, diffondendosi.

Il procedimento seguito dall'algoritmo di ray tracing è la determinazione dei pixel visibili tracciando raggi dal punto di vista verso gli oggetti in scena attraverso la window.

L'operazione chiave che interessa perciò il processo di ray tracing e ne

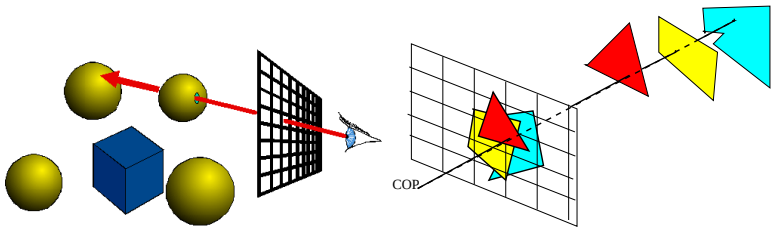


Figure 5.19: Intersezione raggio-scena

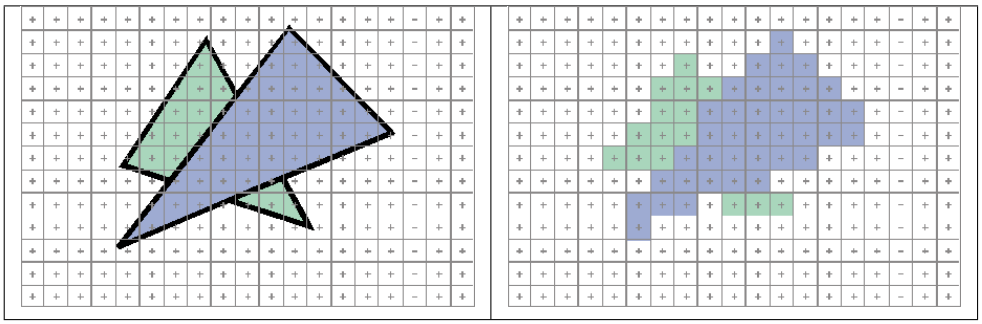


Figure 5.20: Rasterizzazione: discretizzazione di primitive geometriche

caratterizza la sua complessità computazionale, è l'intersezione di un raggio con gli oggetti presenti nel volume di vista, Fig.5.19. Questo algoritmo, di cui si parlerà più dettagliatamente in un prossimo capitolo di questa trattazione, risolve problemi di visibilità, illuminazione e shading della scena.

Algoritmo di Ray Tracing

```
Per ogni pixel dell'immagine
        determina il raggio dal COP al pixel
        Per ogni oggetto in scena
                calcola intersezione raggio/superficie
                se (punto di inters. è più vicino del corrente)
                        allora questa intersezione è visibile;
                memorizza l'oggetto associato;
                disegna il pixel con il colore dell'oggetto associato;
```

5.7 Rasterizzazione o Scan conversion

L'input alla fase di rasterization è costituito da vertici delle primitive grafiche opportunamente trasformati e proiettati sulla viewport, con associati profondità, colore ed eventualmente coordinate texture. Dopo la proiezione, infatti, un oggetto geometrico (es. una linea definita in 3D dai suoi estremi) diventa un segmento di linea nello screen space 2D definito da una coppia di due vertici 2D ciascuno con un valore z (depth-value), un colore ed eventualmente coordinate texture associate.

Il processo di rasterizzazione, a partire dai vertici che definiscono un'entità geometrica, determina i frammenti (potenziali pixel) corrispondenti a tale entità geometrica, e li accende nel color buffer contenuto nel frame buffer. Avviene perciò una **discretizzazione delle primitive grafiche in frammenti** ognuno dei quali è contenuto in un pixel sul frame buffer (vedi Fig.5.20).

Per l'entità geometrica poligono, questo significa determinare tutti i pixel che stanno sul bordo del poligono e quelli che stanno all'interno del poligono 2D definito dai vertici proiettati.

Operazioni per-fragment decidono il valore da assegnare a ciascun pixel. Per assegnare un colore ai pixel tra due estremi di una linea viene usata l'interpolazione lineare tra colori assegnati agli estremi.

5.7.1 Algoritmi per il disegno di linee

Si vuole ora affrontare il problema del disegno di un segmento di estremi interi $(x_1, y_1), (x_2, y_2)$, appartenente alla retta $y = mx + q, m = \frac{y_2 - y_1}{x_2 - x_1}$. Se $|m| \leq 1$ il segmento interseca più colonne che righe e la rasterizzazione conterrà un pixel per ogni colonna; mentre se $|m| > 1$ il segmento interseca più righe che colonne e la rasterizzazione conterrà un pixel per ogni riga. Noi considereremo il caso solo di $|m| \leq 1$: gli algoritmi considerati in questo caso possono essere facilmente estesi agli altri casi.

Metodo incrementale: Digital Differential Analyzer (DDA)

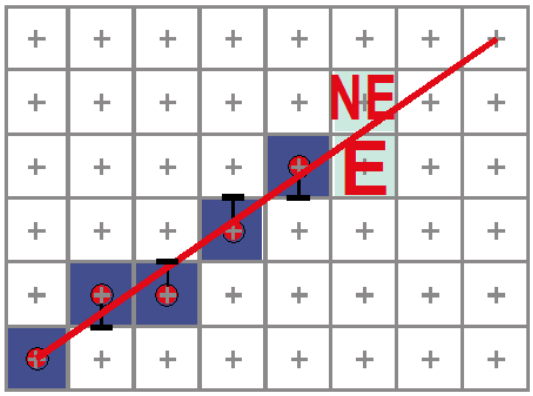


Figure 5.21: Algoritmo di Midpoint

Per disegnare il segmento di estremi interi $(x_1, y_1), (x_2, y_2)$ appartenente alla retta $y = mx + q$, con $m = \frac{y_2 - y_1}{x_2 - x_1}$, si generano passi unitari nella direzione dell'estensione maggiore. Se per esempio l'estensione maggiore è in x:

- incrementare x: $x = x + 1$;
- calcolare $y = mx + q$;
- accendere il pixel di coordinate $(x, round(y))$.

Quindi il punto (x'', y'') ottenuto per incremento da (x', y') viene calcolato nel modo seguente:

$$\begin{aligned} y'' &= mx'' + q = m(x' + 1) + q \\ &= m(x' + 1) + y' - mx' = y' + m \quad (m \leq 1) \\ y'' &= y' + 1, x'' = x' + 1/m \quad (m > 1) \end{aligned}$$

L'algoritmo risulta lento e utilizza valori floating point.

5.7.2 Algoritmo Midpoint

Nel 1985 è stata proposta una leggera variante dell'algoritmo originale di Bresenham (1965) per il disegno di linee, l'algoritmo prende il nome di Midpoint Algorithm. L' algoritmo utilizza solo aritmetica intera.

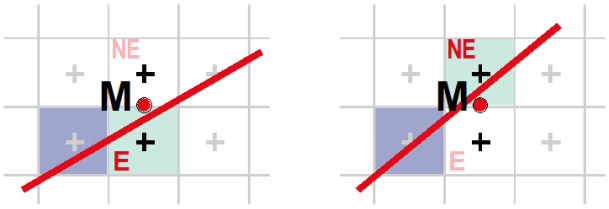


Figure 5.22: Algoritmo di Midpoint

```
void MidpointLine(int x1, int y1,
                  int x2, int y2, int value) {
        int     dx = x2 - x1;
        int     dy = y2 - y1;
        int     d = 2 * dy - dx;
        int     incrE = 2 * dy;
        int     incrNE = 2 * (dy - dx);
        int     x = x1;
        int     y = y1;

        writePixel(x, y, value);

        while (x < x1) {
                if (d <= 0) {  // East Case
                        d = d + incrE;
                } else {                // Northeast Case
                        d = d + incrNE;
                        y++;
                }
                x++;
                writePixel(x, y, value);
        }               /* while */
}                       /* MidpointLine */
```

Figure 5.23: Algoritmo di Midpoint

Se il pixel $P(xp, yp)$ è stato appena disegnato, il prossimo pixel deve essere o $E(xp + 1, yp)$ o $NE(xp + 1, yp + 1)$, vedi Fig.5.21. Si sceglierà E se il segmento da disegnare passa sotto o attraverso il punto medio M, (Fig.5.22, sinistra) altrimenti si disegna NE (Fig.5.22, destra).

Il problema si riconduce quindi a valutare da quale parte del segmento si trova il punto medio $M(xm, ym)$.

Se la linea viene rappresentata in forma implicita $F(x, y) = ax + by + c = 0$, allora basta verificare che se $F(xm, ym) > 0$ allora M è sotto la linea, se $F(xm, ym) < 0$ allora M è sopra la linea, se $F(xm, ym) = 0$ allora M è sulla linea. Per determinare la forma implicita della retta, partiamo dalla forma esplicita:

$$\begin{aligned} y &= mx + B \\ y &= \frac{dy}{dx}x + B \\ dxy &= dyx + dxB \end{aligned}$$

$$F(x, y) = dyx - dxy + dxB$$

Quindi $a = dy, b = -dx, c = Bdx$.

Ci serviamo di una variabile decisionale $d = F(xm, ym)$, con (xm, ym) coordinate del punto medio M. Poichè $M(xp + 1, yp + \frac{1}{2})$, allora $d = F(xp + 1, yp + \frac{1}{2})$.

Se $d < 0$ si sceglie E, se $d > 0$ si sceglie NE, infine se $d = 0$ si sceglie indifferentemente uno dei due pixel E o NE.

Come si incrementa la variabile decisionale d? A seconda della scelta precedente (E o NE) si determina il nuovo M e il corrispondente valore di d.

- Caso E:
  $d_{new} = F(xp + 2, yp + \frac{1}{2}) = a(xp + 2) + b(yp + \frac{1}{2}) + c$;
  $d_{old} = a(xp + 1) + b(yp + \frac{1}{2}) + c$;
  Sottraiamo la prima dalla seconda: $d_{new} = d_{old} + a$ quindi $\Delta E = a = dy$. Il valore della variabile decisionale al prossimo step è ottenuto incrementando il valore decisionale precedente senza rivalutare $F(xm, ym)$: $d_{new} = d_{old} + \Delta E = d_{old} + dy$
- Caso NE:
  $d_{new} = F(xp + 2, yp + \frac{3}{2}) = a(xp + 2) + b(yp + \frac{3}{2}) + c$
  $d_{old} = a(xp + 1) + b(yp + \frac{1}{2}) + c$;
  Sottraiamo la prima dalla seconda: $d_{new} = d_{old} + a + b$ quindi $\Delta NE = a + b = dy - dx$. Il valore della variabile decisionale al prossimo step è ottenuto incrementando il valore decisionale precedente senza rivalutare $F(xm, ym)$: $d_{new} = d_{old} + \Delta NE = d_{old} + dy - dx$

Riassumendo, ad ogni step l'algoritmo sceglie tra due pixel basandosi sul segno della variabile di decisione calcolata nella precedente iterazione. Quindi aggiorna la variabile decisionale d aggiungendo ΔE o ΔNE al vecchio valore di d a seconda della scelta del pixel. Occorrono quindi solo addizioni!

Il primo valore di d è determinato in $(x0 + 1, y0 + \frac{1}{2})$. Quindi $d = F(x0 + 1, y0 + \frac{1}{2}) = F(x0, y0) + a + \frac{b}{2} = a + \frac{b}{2}$ Per eliminare la frazione in d si ridefinisce F moltiplicandolo per 2; $F(x, y) = 2(ax + by + c)$. In Fig5.23 l'algoritmo di Midpoint completo.

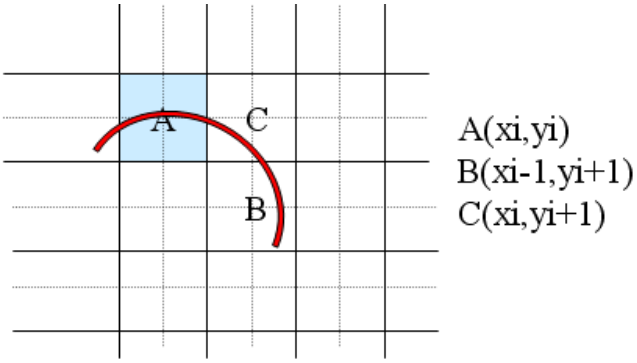


Figure 5.24: Algoritmo per archi di cerchio di Bresenham-Michner

5.7.3 Algoritmi per il disegno di circonferenze

Consideriamo l'equazione del I ottante di cerchio:

$$f(x,y) = x^2 + y^2 - r^2 = 0 \quad con \quad y \leq x$$

quindi sfruttiamo la simmetria per disegnare l'intera circonferenza.

Algoritmo per archi di cerchio di Bresenham-Michner

Si consideri la configurazione in Fig.5.24. Avendo già selezionato A, il metodo sceglie fra i due pixel B e C ad ogni passo. L'espressione del cerchio da approssimare è valutata sia in $B(xb, yb)$ che in $C(xc, yc)$.

Per determinare il discriminatore in modo analogo a quanto fatto nel caso di disegno di linee, ci avvaliamo del seguente teorema.

Teorema Siano $B(xb, yb)$ e $C(xc, yc)$ pixel adiacenti (coordinate intere), sia r intero (raggio del cerchio),se

$$|f(xc, yc)| < |f(xb, yb)| \quad allora \quad d1 < d2.$$

Anche se in generale le distanze d1 e d2 dei punti C e B dall'arco di cerchio non sono proporzionali ai valori f(xc,yc) e f(xb,yb) per il teorema sopra si possono assumere ugualmente f(xc,yx) e f(xb,yb) come indicatori di tali distanze.
Se $|f(xb, yb)| < |f(xc, yc)|$ allora sarà scelto B come il più vicino, altri-

menti C. Poniamo:

$$\begin{aligned} d_i &= f(xc, yc) + f(xb, yb) = \\ &= x_i^2 + (y_{i+1})^2 - r^2 + (x_{i-1})^2 + (y_{i+1})^2 - r^2 = \\ &= x_i^2 + (x_{i-1})^2 + 2(y_{i+1})^2 - 2r^2 \end{aligned}$$

Algoritmo per disegnare il I ottante di un cerchio di centro l'origine

```
x:=r;
y:=0;
while y < x do
begin
    plot(x,y);
       y:=y+1;
    d := x^2 + (x-1)^2 + 2*y^2 - 2*r^2;
    if d ≥ 0 then x := x - 1;
end;
if x = y then plot(x,y);
```

Vediamo ora come determinare il valore del discriminatore al passo i+1 utilizzando il valore del discriminatore al passo i.

- Se $d_i \geq 0$ al passo i, allora si sceglie
  $B(x_{i+1} = x_i - 1, y_{i+1} = y_i + 1)$

  $$\begin{aligned} d_{i+1} &= f(x_{i+1}, y_{i+1} + 1) + f(x_{i+1} - 1, y_{i+1} + 1) = \\ &= di + 4(y_i - x_i) + 10 \end{aligned}$$
- Se $d_i < 0$ al passo i, allora si sceglie
  $C(x_{i+1} = x_i, y_{i+1} = y_i + 1)$

  $$\begin{aligned} d_{i+1} &= f(x_i + 1, y_{i+1} + 1) + f(x_{i+1} - 1, y_{i+1} + 1) = \\ &= d_i + 4y_i + 6 \end{aligned}$$

Al passo $i = 0$ sarà $x = r$ ed $y = 0$ e quindi $d_0 = 3 - 2r$

L' Algoritmo modificato per il disegno di un ottante di cerchio è il seguente:

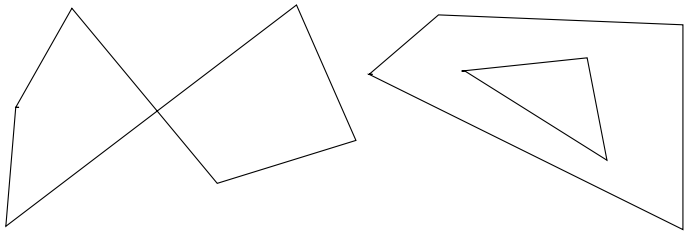


Figure 5.25: Esempi di poligoni non regolari: intrecciati (sinistra), con più cicli (destra)

```
begin
x:=r;
y:=0;
d:=3 - 2*r;
while y<x do
        begin
        plot(x,y);
        if d<0 then d:=d + 4*y +6
        else begin
                d:=d+4*(y-x) +10;
                x:=x-1
                end;
        y:=y+1
        end
if x = y then plot (x,y)
end;
```

5.8 Algoritmi per il riempimento di poligoni

Nella seguente trattazione si considerano solo poligoni regolari e mono-connessi; non si considerano quindi poligoni intrecciati (un vertice può essere estremo di più lati del poligono) o poligoni con più cicli, si veda Fig.5.25.

Un poligono può essere rappresentato da una polyline avente l'ultimo

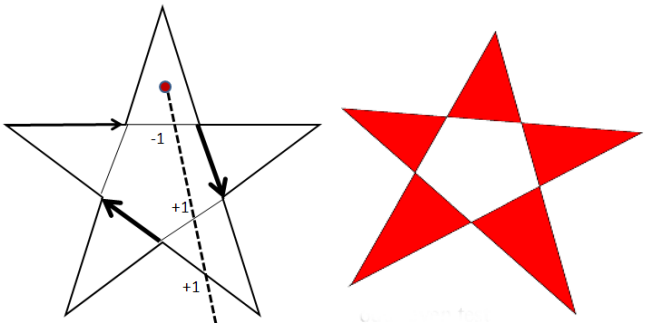


Figure 5.26: Point Membership Classification test; il winding number del punto in rosso è 1 (interno)

punto coincidente con il primo. Riempire l'interno di un poligono con un colore o un pattern equivale a decidere quali punti del piano su cui giace il poligono sono punti interni al poligono stesso. Questo test prende il nome di **Point Membership Classification** (PMC) o inside-outside testing e può essere realizzato mediante una delle due seguenti tecniche:

- **odd-even test:** Un punto risulta interno ad un triangolo se giace nel semipiano positivo di tutte e tre le linee di bordo del triangolo. Per un triangolo con vertici ordinati in senso anti-orario il punto si trova a sinistra di ogni linea di contorno. Siano $E_i = a_i x + b_i y + c_i$, $i = 1, 2, 3$, le tre rette su cui giacciono i lati del triangolo T.

$$(x, y) \in T \quad \Longleftrightarrow \quad E_i(x, y) \geq 0, i = 1, 2, 3. \tag{5.1}$$

  Il test può essere adottato anche per poligoni a più lati purchè convessi. Per un poligono qualsiasi (anche concavo) il test si generalizza nel seguente, dove da un generico punto p nel piano si traccia un raggio verso i bordi della window:
  - Sia p punto interno al poligono. Ogni raggio che fuoriesce da p attraversa un numero dispari di lati.
  - Sia p punto esterno al poligono. Ogni raggio che fuoriesce da p e attraversa il poligono, lo attraversa per un numero pari di lati.
- **winding test** Un punto è interno se il suo winding number è diverso da zero.
  Il calcolo del winding number avviene attraversando i lati di un

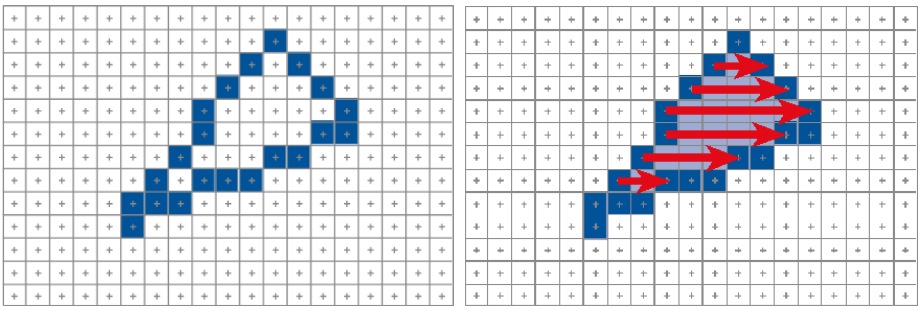


Figure 5.27: Scansione di poligoni

poligono da un vertice di partenza in una data direzione (non importa quale) (Fig.5.26(sinistra)). Dato un punto arbitrario p, per determinare il suo winding number associato si considera una linea da p non parallela ad alcun lato del poligono che attraversa il poligono completamente. Il winding number per p è il numero di lati che attraversa la linea in una direzione, meno il numero di lati che attraversa la linea in direzione opposta. Nell'esempio illustrato in Fig.5.26(sinistra) il punto è interno al poligono rosso mostrato in Fig.5.26(destra).

Per il riempimento dei pixel interni al poligono si hanno due differenti approcci al problema:

- flood fill
- scan-line fill

flood fill: rasterizza i pixel di bordo (lati) nel frame buffer. Determina un punto iniziale (seed point) dentro il poligono, colora i vicini ricorsivamente fino al bordo.

scan-line fill: La porzione di griglia su cui si trova il poligono viene scansita riga per riga, come illustrato in Fig. 5.27; per ogni riga si utilizza il test PMC per determinare se un punto sulla scan-line è interno o esterno; questo identifica le span di quella scan line. Gli estremi di ogni span formano il bordo del poligono; si colorano i pixel interni ad ogni span con un colore costante o si utilizza interpolazione colore.

Nel caso semplice di poligoni convessi, per esempio triangoli, si ha quindi

il seguente algoritmo di scan-conversion:

```
For every triangle
    Compute projection for vertices
    Compute bbox, clip bbox to screen limits
    For all scanlines y in bbox
        Evaluate all Ei's at (x0, y): Ei = ai x0 + bi y + ci
        For all pixels x in bbox
            If all Ei > 0
                Framebuffer[x,y ] = triangleColor
            Increment line equations: Ei+ = ai
```

La bounding box di un triangolo (`bbox`) si ottiene dai valori $x_{min}, x_{max}, y_{min}, y_{max}$ dei vertici proiettati del triangolo stesso. Un semplice test permette inoltre di fare clipping con i bordi della window. L'angolo in alto a sinistra della `bbox` ha coordinate (x_0, y). Riga per riga (scanline y) la `bbox` viene scansita e solo se il pixel verifica l'odd-even test (5.1) viene aggiornato il framebuffer.

Per generalizzare il procedimento scan-line fill al riempimento di poligoni non solo convessi, ma anche concavi e intrecciati si deve tener presente che, in questo caso, ogni riga può contenere più di una span, si devono perciò trovare le intersezioni della scan-line con i lati del poligono, ordinare tali intersezioni per la coordinata x, infine selezionare tutti i pixel, tra le coppie di intersezioni che sono interni al poligono con test PMC.

5.8.1 Riempimento di poligoni con interpolazione bilineare

Dato il triangolo di vertici (v_1, v_2, v_3) si vuole assegnare il valore scalare (colore RGB /depth/texture coordinate) ai punti interni P al triangolo nel frame buffer.

Con riferimento alla Fig.5.28, prima si interpola lungo i lati del triangolo con interpolazione lineare tra i vertici v_1 e v_2, v_2 e v_3:

$$P_1(\alpha) = (1-\alpha)v_1 + \alpha v_2 \quad P_2(\alpha) = (1-\alpha)v_2 + \alpha v_3, \quad \alpha \in [0,1],$$

poi si interpola lungo le righe orizzontali di pixel all'interno del triangolo

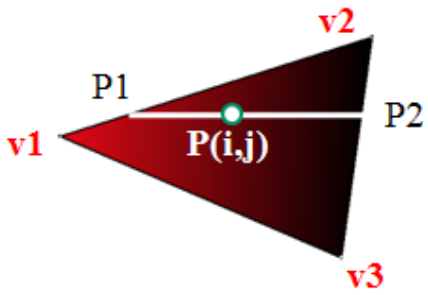


Figure 5.28: interpolazione all'interno di un triangolo.

tra P_1 e P_2. Dati P_1, P_2 e $P(i,j)$ si determina il valore del parametro β mediante

$$\beta = \frac{(P - P_1)(P_2 - P_1)}{\|P_2 - P_1\|^2}.$$

Quindi se i vertici P_1 e P_2 sono caratterizzati rispettivamente dai colori (R_1, G_1, B_1) e (R_2, G_2, B_2), allora il punto $P(i,j)$ avrà assegnato il colore:

$$\begin{aligned} R(i,j) &= (1-\beta)R_1 + \beta R_2 \\ G(i,j) &= (1-\beta)G_1 + \beta G_2 \\ B(i,j) &= (1-\beta)B_1 + \beta B_2. \end{aligned}$$

Chapter 6

Illuminazione e Shading

Fino ad ora abbiamo visto come costruire modelli geometrici tridimensionali e come visualizzarli. Tuttavia la loro natura tridimensionale non viene assolutamente messa in risalto se non introduciamo l'aspetto della illuminazione della scena che li contiene.

Aver assegnato un colore agli oggetti ci permette di vederli di un colore uniforme, che rende la scena 'piatta' (Fig.6.1(c)). Se, per esempio, guardiamo una fotografia di una sfera illuminata non vediamo un cerchio colorato uniformemente, bensì una forma circolare con molte gradazioni di colore sfumato. Le gradazioni del colore che vengono prodotte dall' illuminare l'oggetto colorato con sorgenti luminose danno all'immagine bidimensionale l'apparenza tridimensionale che desideriamo.

Quello che abbiamo trascurato fino ad ora è l'interazione fra la luce e le superfici dei modelli geometrici e tutti gli effetti che da questa ne derivano: ombre, riflessioni, rifrazioni,.. (Fig.6.2).

Se in termini generali definiamo illuminazione il trasporto diretto ed indiretto del flusso luminoso dalle risorse luminose alla scena, allora in computer graphics con il termine ***lighting*** (illuminazione, in lingua inglese) intendiamo il calcolo dell'intensità luminosa di ogni punto 3D (effettuato simulando l'interazione luce-scena). Con il termine ***shading***, invece ci riferiamo all'assegnamento del colore per-pixel, calcolato a partire dalla intensità luminosa del punto 3D.

Considereremo quindi in questo capitolo vari modelli di risorse luminose, e le più comuni interazioni luce-materiali. Verrà poi presentato un

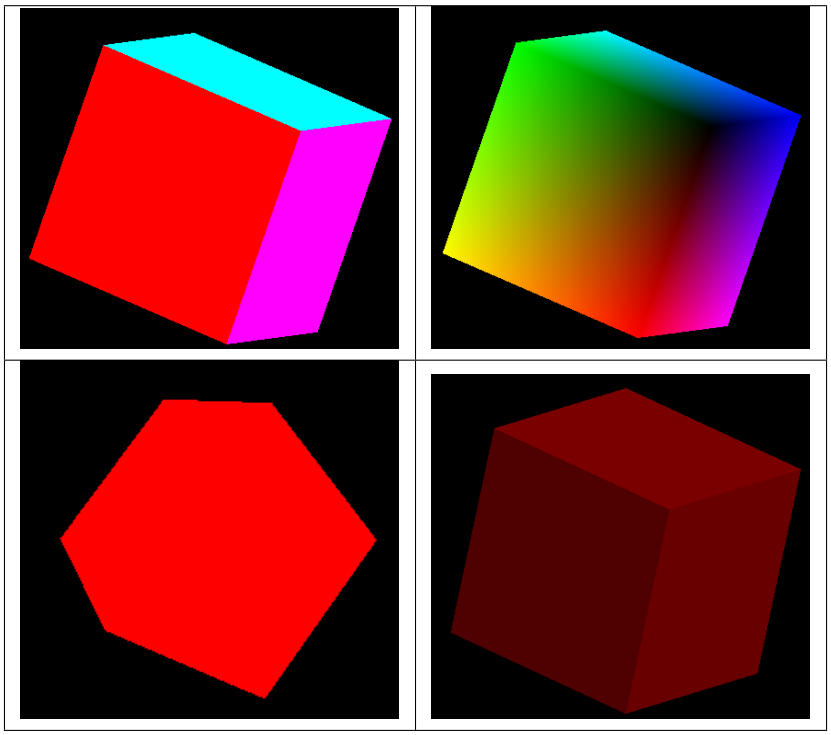


Figure 6.1: (a) e (b) Resa senza illuminazione con colori alle facce (a) o ai vertici (b); (c) Resa senza illuminazione, stesso colore alle facce; (d) resa con illuminazione

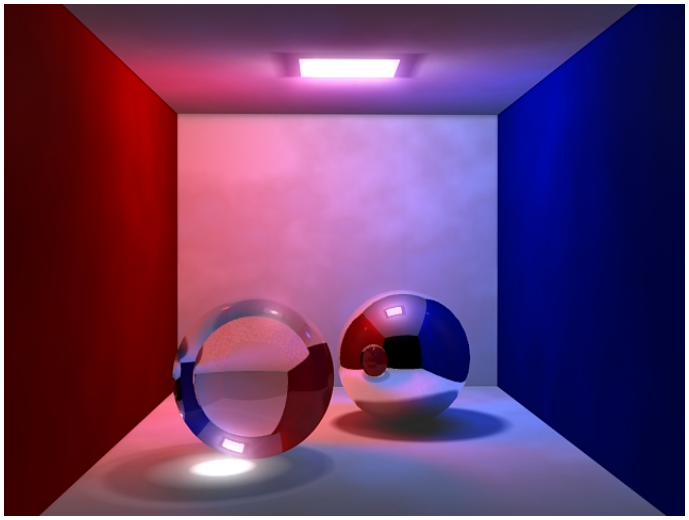


Figure 6.2: Resa con illuminazione, ombre, riflessioni e trasparenze

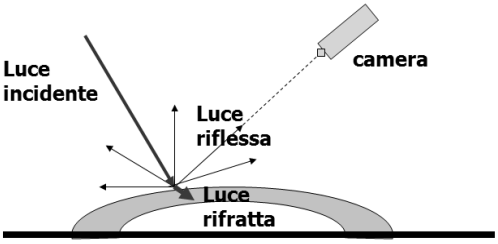


Figure 6.3: La luce può essere riflessa, assorbita o rifratta.

modello di illuminazione locale adatto ad una pipeline grafica per un rendering veloce e infine un modello di illuminazione globale che tiene conto dei contributi di illuminazione indiretti provenienti dagli altri oggetti in scena.

Si presenteranno inoltre le principali tecniche di shading.

La maggioranza delle illustrazione riportate sono tratte dal testo *Alan Watt, 3D Computer Graphics, 3rd Edition, Addison-Wesley, 2000.*

6.1 Illuminazione

La luce visibile è composta da uno spettro di colori che vanno dal rosso al viola. Come illustrato in Fig.6.3 la luce che colpisce un punto di un oggetto, viene in parte **assorbita** (e quindi trasformata in altre forme di energia come calore,vibrazioni), in parte **riflessa** e in parte può essere **rifratta** ovvero trasmessa all'interno della superficie (rifrazione). Il colore che il nostro occhio percepisce di un oggetto è dovuto solo ai fotoni che sono riflessi (o emessi da) dalla superficie dell'oggetto e raggiungono il nostro occhio.

La quantità e il colore della luce riflessa dagli oggetti è ciò che ci consente di vederli come rossi, verdi, o qualsiasi colore dello spettro. Una superficie di un oggetto può emettere luce (lampadina) e/o riflettere luce che la colpisce da altre superfici o direttamente da risorse luminose. Il colore di un punto su di un oggetto è quindi definito da interazioni multiple tra risorse luminose e superfici riflettenti. Queste interazioni possono essere viste come un processo ricorsivo.

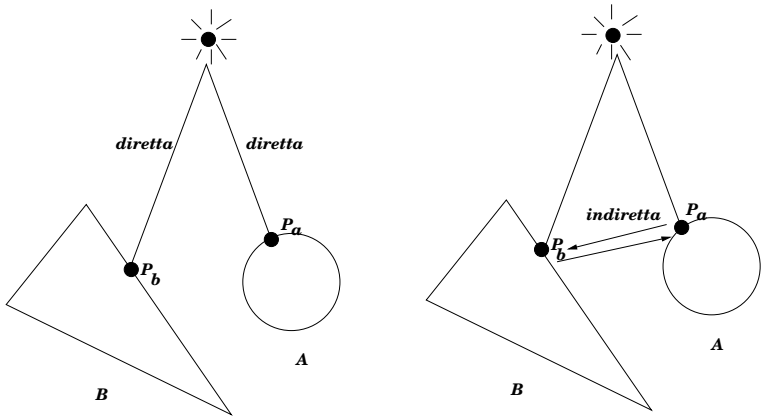


Figure 6.4: Modello locale (sinistra), Modello globale (destra)

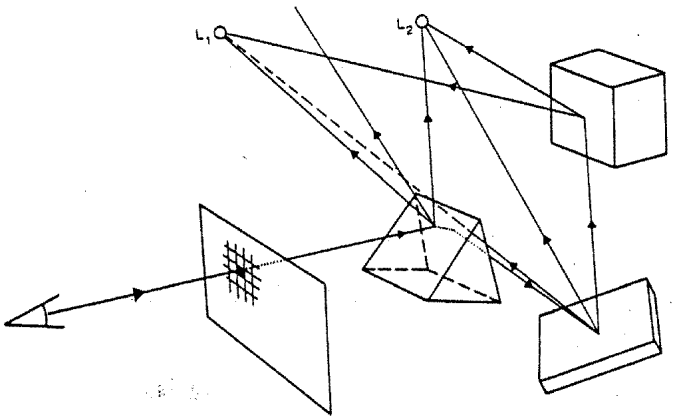


Figure 6.5: Modello globale, scena con tre oggetti e due sorgenti luminose.

Matematicamente, questo processo globale ricorsivo che i raggi luminosi in scena seguono è espresso dalla **Rendering equation**. Questa equazione per la sua complessità non può in generale essere risolta analiticamente. Esistono quindi vari approcci che la approssimano come la radiosity e il ray tracing.

I modelli di illuminazione che si assumono in computer graphics sono approssimazioni della rendering equation e possono essere di tipo locale (diretto) o globale.

Un **modello di illuminazione locale** calcola il colore di un punto in

base alla luce direttamente emessa dalle risorse luminose ed alle proprietà della superficie Fig.6.4(sinistra). In questo caso la luminosità di un punto non è influenzata da altri oggetti in scena.

Un **modello di illuminazione globale** calcola il colore di un punto in base alla luce direttamente emessa dalle risorse luminose e alla luce che raggiunge quel punto riflessa o trasmessa dalla superficie stessa o da altre superfici di oggetti presenti (Fig.6.4(destra), Fig.6.5). L'illuminazione del punto è influenzata quindi anche dagli altri oggetti presenti. Questo rende tale modello incompatibile con una pipeline di rendering.

Nell'approccio locale che consideriamo nel seguito seguiamo il percorso dei raggi emessi dalle risorse luminose che intersecano superfici riflettenti nella scena (solo singole interazioni tra raggi luminosi e superfici).

6.2 Un modello locale

Nel considerare le interazioni tra risorse luminose e superfici dobbiamo affrontare tre parti indipendenti del problema: come modellare le risorse luminose, come definire i materiali e come costruire un modello di riflessione per l'interazione tra materiali e luce.

Partiamo seguendo i raggi luminosi che escono da una sorgente luminosa (Fig. 6.6 (sinistra)). L'osservatore vede solo i raggi che escono dalla risorsa e raggiungono i suoi occhi attraverso un complesso percorso di interazioni multiple con oggetti in scena. Se un raggio luminoso colpisce direttamente l'osservatore, egli vede il colore della risorsa luminosa. Se il raggio di luce colpisce una superficie che è visibile all'osservatore, il colore che vede è dato dall'interazione tra materiale della superficie e risorsa luminosa.

In termini di computer grafica sostituiamo l'osservatore con il piano di proiezione come mostrato in figura Fig. 6.6(destra). Il colore delle risorse luminose e delle superfici determina il colore dei pixel nel frame buffer.

Consideriamo ora solo i raggi che lasciano la risorsa luminosa e raggiungono l'osservatore, o direttamente, o attraverso l'interazione con oggetti. Questi sono i raggi che raggiungono il centro di proiezione (COP) dopo

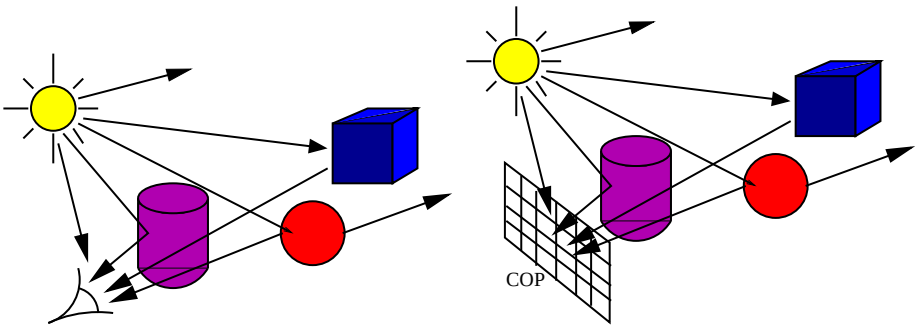


Figure 6.6: luci e superfici

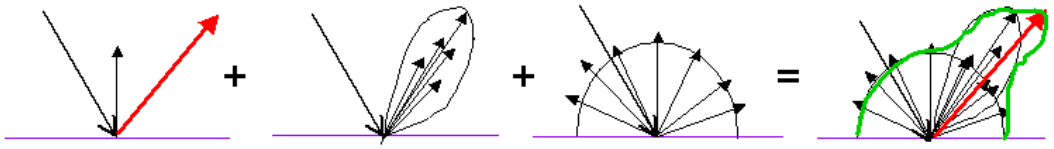


Figure 6.7: Superficie speculare, parzialmente speculare, diffusiva

essere passati attraverso la viewport.

Il colore di un punto sulla superficie è determinato

- dal materiale della superficie;
- dalle risorse luminose che illuminano la superficie;
- dal modello di illuminazione.

6.2.1 Materiali: interazione luce-materiale

Vogliamo simulare l'apparenza di un materiale in base a come la superficie dell'oggetto riflette la luce proveniente da una risorsa luminosa. Con riferimento alla Fig.6.7 un certo materiale è simulato mediante una combinazione delle seguenti tipologie:

- **superfici diffusive:** luce assorbita e irradiata in tutte le direzioni con uguale energia (Es. terreno, muro dipinto)
- **superfici speculari:** luce riflessa direttamente dalla superficie lungo la direzione di riflessione. (Es. lo specchio è perfettamente speculare)

Figure 6.8: Simulazione vari materiali

- **superfici lucide/traslucide:** parzialmente speculari.

Un esempio di simulazione di vari materiali è mostrata in Fig.6.8.

Per caratterizzare quindi un materiale, si sommano vari contributi definiti dai seguenti coefficienti scalari:

K_a ambiente, K_d diffusivo, K_s speculare, n_s grado di specularità per superfici parzialmente speculari, K_e emissivo.

Un oggetto che rappresenta una sorgente luminosa sarà caratterizzato dall'avere K_e diverso da zero.

6.2.2 Risorse luminose

Una risorsa luminosa è caratterizzata da un tipo, da un colore (RGB) e da un'intensità luminosa (AMBIENTE I_a, DIFFUSA I_d, SPECULARE I_s).

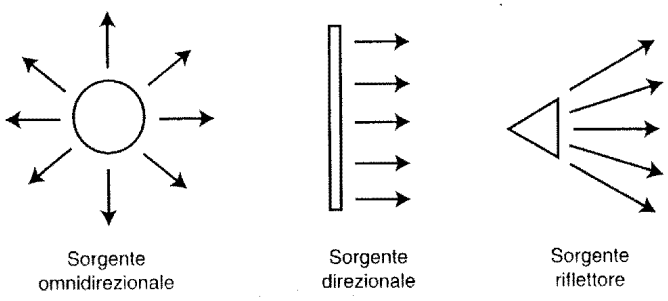


Figure 6.9: Tipi di risorse luminose

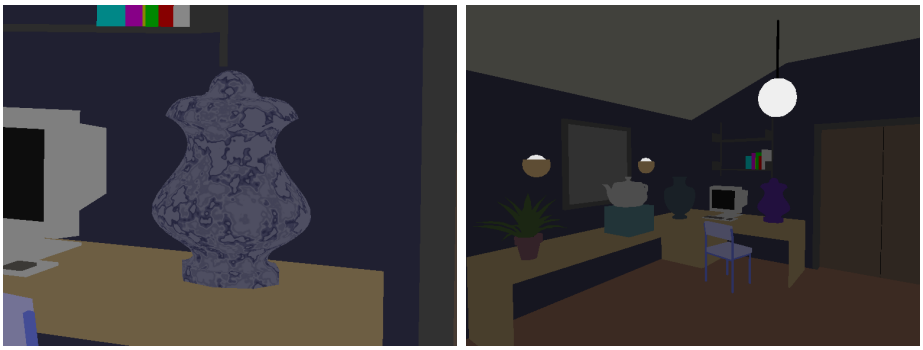


Figure 6.10: Scena resa con la sola luce ambiente

Esaminiamo nel seguito i tipi di sorgenti luminose che vengono comunemente simulate in un modello di illuminazione locale: luce ambiente, luce omnidirezionale (punto), direzionale e spot (Fig. 6.9).

- **luce ambiente**
  luce uniforme in tutto l'ambiente, non emessa da una particolare risorsa luminosa (Fig.6.10); simula il contributo luminoso dovuto alla luce indiretta che in un modello locale viene trascurato;
- **sorgente omnidirezionale (punto luce)**
  emissione di luce da un punto $p_0 = [x, y, z, 1]$; in tutte le direzioni con intensità costante; adatta per simulare lampadine,(Fig.6.11);
- **sorgente direzionale:** nella luce direzionale l' emissione di luce è costante lungo una direzione

$$p_0 = [x, y, z, 0];$$

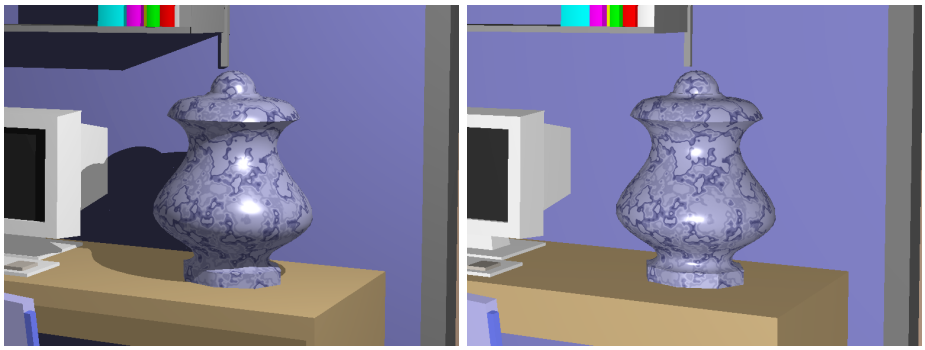


Figure 6.11: (sinistra) scena con luce omnidirezionale posizionata vicino al vaso; (destra) luce omnidirezionale coincidente con l'osservatore, si noti come in questo caso non vengono create ombre.

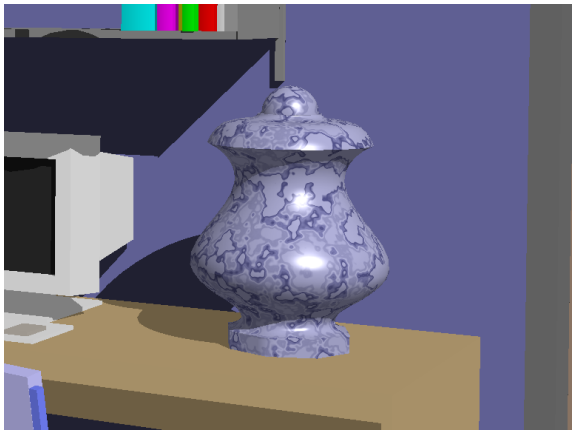


Figure 6.12: Scena resa con luce direzionale

adatta per simulare luci molto lontane dalla scena, per esempio luce solare, (Fig.6.12);

- **sorgente riflettore: luce spot**
  emissione di un cono, o piramide di luce da un singolo punto (apice del cono) p_s in direzione ℓ, con ampiezza θ. L'intensità è concentrata nel centro del cono e si attenua verso $\theta, -\theta$ (Fig.6.13).

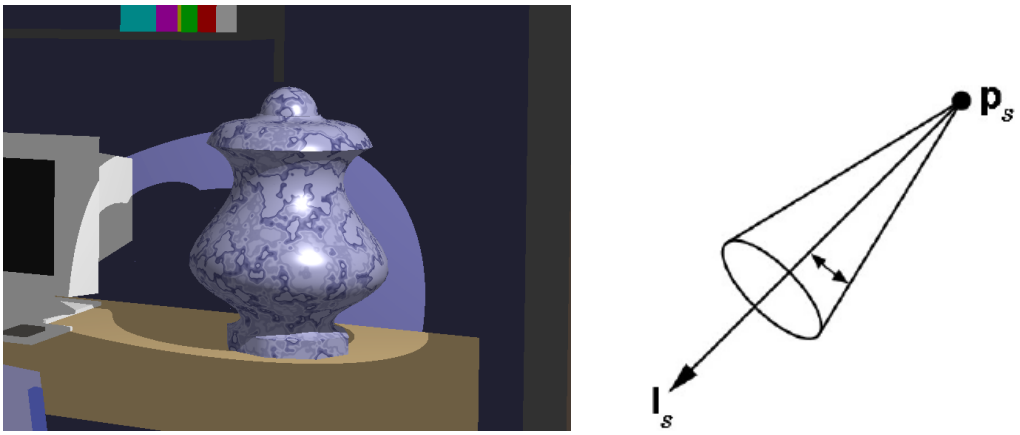


Figure 6.13: Scena resa con luce spot di cono 30 gradi

6.3 Modello di illuminazione locale di Phong

Il modello di illuminazione locale che presentiamo fu introdotto da Phong nel 1975. Esso simula l'interazione luce-materiale ottenendo un ottimo compromesso tra efficienza e buona approssimazione della realtà fisica. Si premette che l' illuminazione nella pipeline di rendering deve essere fatta prima della trasformazione prospettica e della normalizzazione che potrebbero portare il vettore normale n alla superficie a non essere più perpendicolare alla superficie cui si riferisce, quindi i calcoli vanno fatti in coordinate di vista.

L'intensità luminosa riflessa in un punto della superficie, ovvero l' illuminazione di quel punto, è simulata dal contributo di quattro componenti: emissiva, ambiente I_a, diffusiva I_d e speculare I_s, combinati linearmente con coefficienti che dipendono dal materiale in gioco.

La componente emissiva considera gli oggetti come sorgenti luminose da cui la luce scaturisce in modo uniforme da tutti i punti e verso tutte le direzioni.

L'equazione completa del modello di illuminazione locale di Phong in un punto è la seguente:

$$I = k_e + k_a I_a + k_d I_d + k_s I_s$$

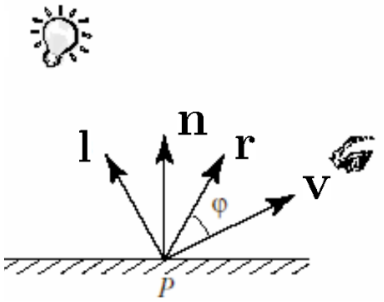


Figure 6.14: n vettore normale alla superficie, ℓ direzione sorgente luminosa, v punto di vista, r raggio riflesso.

dove $0 \leq k_i \leq 1$ sono i coefficienti che caratterizzano i materiali, in particolare k_e rappresenta la luminosità intrinseca dell'oggetto.

6.3.1 Contributo riflessione ambiente

La luce proveniente in modo indiretto da altri oggetti o direzioni e riflessa in tutte le direzioni si può inserire nel modello locale attraverso il contributo della luce ambiente. Essa rappresenta la luce indiretta garantendo che tutti gli oggetti in scena abbiano un minimo di illuminazione.

L'intensità luminosa indiretta (anche se il punto non è direttamente illuminato) è data da:

$$I = k_a I_a$$

dove $k_a \in [0, 1]$ è il coefficiente di riflessione ambientale (compreso nell' intervallo $[0, 1]$), e I_a è l'intensità della luce ambientale ed è uguale per tutti gli oggetti in scena.

6.3.2 Contributo riflessione diffusa

Riflessioni diffuse caratterizzano le superfici opache. Superfici diffusive perfette, che riflettono la luce uniformemente in tutte le direzioni, sono chiamate **superfici Lambertiane**. Un esempio di superficie ideale diffusa è il gesso.

Sia n la normale alla superficie nel punto di incidenza e ℓ il raggio lu-

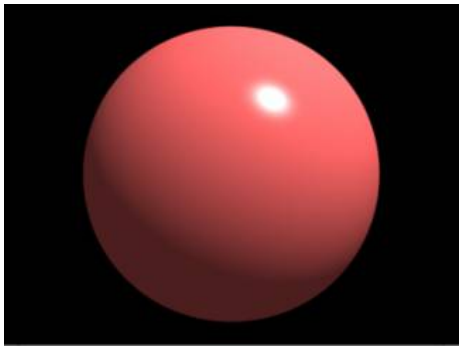


Figure 6.15: Highlight su superficie

minoso incidente, come mostrato in Fig.6.14, mentre θ è l'angolo di incidenza della luce (angolo compreso tra n e l).

Il **modello di Lambert** per l' intensità illuminazione diffusa è definito dalla formula:

$$I = k_d I_d = k_d I_\ell \cos(\theta) = k_d(\ell \cdot n) I_\ell$$

dove I_ℓ è l' intensità della luce incidente, $k_d \in [0,1]$ è il coefficiente di riflessione diffusa del materiale.

La quantità di luce che arriva all'osservatore non dipende dalla posizione dell'osservatore stesso.

Inoltre, l'angolo θ deve avere un valore compreso tra -90 e 90 per contribuire all'illuminazione del punto, in altre parole un punto della superficie non è illuminato da sorgenti luminose che stanno dietro di essa; nel caso quindi di prodotto scalare $\ell \cdot n$ negativo si pone $I = 0$.

6.3.3 Contributo riflessione speculare

La luminosità speculare è quella riflessa da una superficie lucida. La luce viene riflessa in una direzione privilegiata (raggio riflesso r in Fig.6.14). Caratteristici di questo contributo luce sono gli **highlight** su superfici lucide: macchie del colore della luce sull'oggetto, Fig.6.15. Gli highlight riflettono i colori della luce incidente e non sono influenzati dal colore del materiale. Gli highlight sono più visibili lungo la direzione di perfetta specularità (assumendo una superficie speculare ideale). Quindi,

quanto più l'angolo tra il raggio riflesso r e la direzione dell'osservatore v aumenta quanto più l'highlight cresce.

Vediamo ora il calcolo del contributo dell'intensità di illuminazione speculare. Sia ϕ l'angolo tra i vettori r riflesso ottenuto riflettendo ℓ (direzione di incidenza) rispetto al vettore normale n alla superficie, e v (osservatore) illustrati in (Fig.6.14). La luce speculare ha massima intensità per $\phi = 0$ (osservatore v nella direzione del raggio perfetto riflesso r), e diminuisce con ϕ che aumenta. Il contributo dovuto alla riflessione speculare è dato da:

$$I = k_s I_s = k_s I_\ell \cos^{n_s}(\phi) = k_s I_\ell (v \cdot r)^{n_s}$$

dove k_s è il coefficiente di luce riflessa dal materiale, n_s rappresenta l'esponente di riflessione speculare del materiale detto anche **esponente di Phong**. Il termine $cos^{n_s}(\phi)$ rappresenta quindi la distribuzione di energia lungo la direzione di riflessione r, come illustrato in Fig.6.17 e serve per rappresentare superfici non perfettamente speculari. Un esempio di superficie ideale riflessa ($r = v$) è lo specchio ($n_s = \infty$), per $n_s \in [100, 500]$ si simulano invece superfici metalliche o plastica.

In Fig.6.16 è riportata una tabella di immagini che mostrano la teapot resa con il modello di Phong variando k_s e k_d nel range da 0.0 a 1.0 con passo 0.2 ($6 \times 6 = 36$ immagini). $k_a = 0.7, n = 10.0$ in tutte le immagini, $y = k_s, x = k_d$.

6.3.4 Calcolo dei vettori

Esaminiamo brevemente come determinare i vettori che sono in gioco nel calcolo del modello di illuminazione locale di Phong.

- **Vettore normale (n)**: Un piano può essere descritto dal suo vettore normale n e da un punto P_0, mediante l'equazione:

  $$n \cdot < p - P_0 >= 0.$$

  Dati tre punti non collineari p_0, p_1, p_2 sul piano, questi sono sufficienti per determinarlo univocamente e per determinare la normale

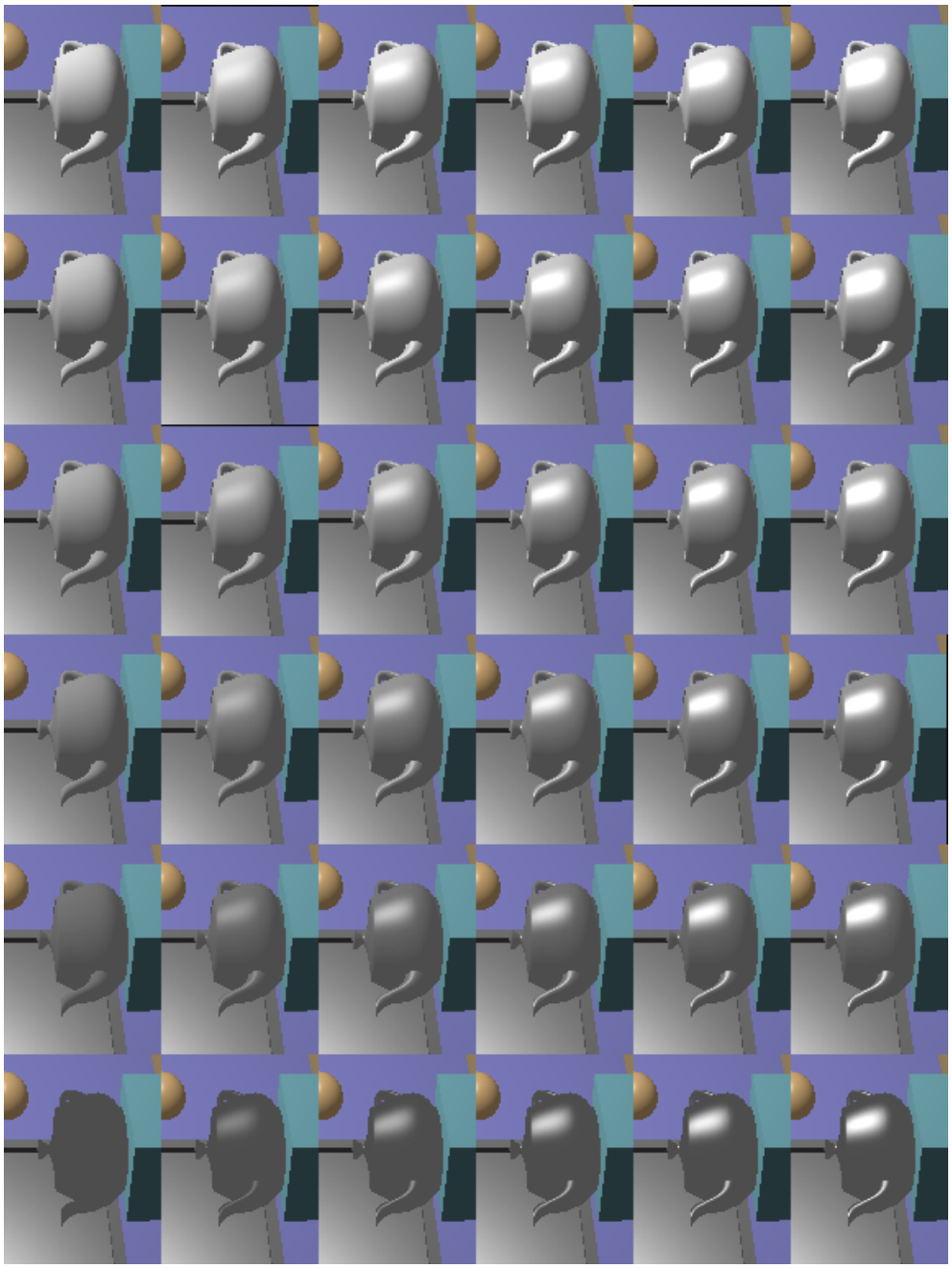


Figure 6.16: Contributo riflessione speculare. L'immagine di coordinate $(0,0)$ corrisponde ai valori $K_s = K_d = 0$, K_s cresce per colonne mentre K_d cresce per righe.

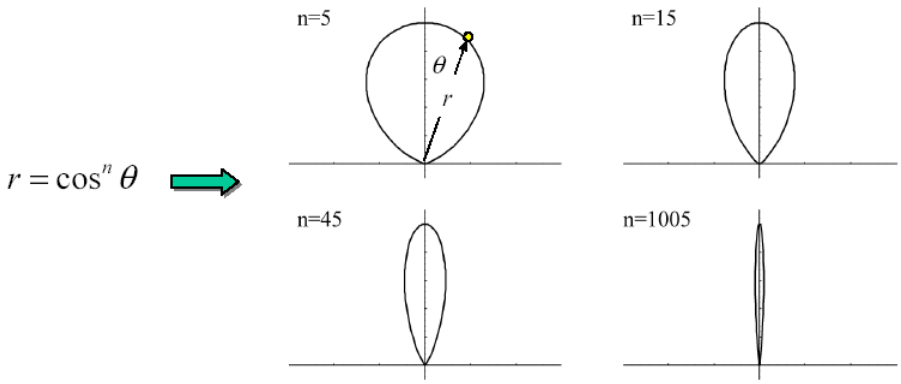


Figure 6.17: Esponente di Phong: distribuzione di energia lungo la direzione di riflessione r

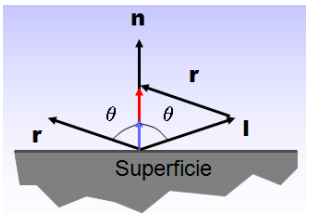


Figure 6.18: Calcolo vettore riflesso

al piano si calcola il prodotto vettoriale:

$$n =< p_2 - p_0 > \times < p_1 - p_0 > .$$

- **Vettore riflesso (r)**: Una superficie riflettente ideale è caratterizzata dalla proprietà che l'angolo θ tra il raggio luce incidente (ℓ) e la normale (n) è uguale all'angolo tra il raggio riflesso (r) e la normale n alla superficie. Se $r + \ell = 2cos(\theta)n$ allora otteniamo:

$$r = 2(\ell \cdot n)n - \ell.$$

Si veda Fig.6.18. Infatti per la regola del parallelogramma

$$r + l = 2\|\ell\|cos(\theta)n$$

dove $\|\ell\|cos(\theta)n$ è la proiezione lungo n del vettore ℓ. Considerando

che ℓ è un vettore unitario si ottiene la formula fornita per il vettore riflessione r.

6.4 Estensioni del modello di Phong

Per meglio simulare gli effetti luminosi reali si considera un fattore di attenuazione dell' intensità luminosa all'aumentare della distanza della sorgente di luce dalla superficie. Si supponga che il punto luce sia a distanza d dall'oggetto. Questo fattore di attenuazione è inversamente proporzionale alla distanza d. In particolare, l'**energia luminosa incidente** sulla superficie d'area unitaria si può supporre proporzionale a $1/d^2$.

L'**energia riflessa** invece è indipendente dalla distanza dall'osservatore. In CG si assume che la quantità di luce lungo un raggio luminoso uscente sia costante.

Se consideriamo l'attenuazione della luce incidente dovuta alla distanza della sorgente per un punto a distanza d dalla sorgente luminosa, il modello di Phong diventa:

$$I = k_e + k_a I_a + I_\ell / d^2 (k_d(\ell \cdot n) + k_s(r \cdot v)^{n_s}).$$

Se le sorgenti luminose sono più di una possiamo sommare il contributo di ciascuna, il modello di Phong risulta quindi il seguente:

$$I = k_e + k_a I_a + \sum_{\ell_s} I_{\ell_s} (k_d(\ell_s \cdot n) + k_s(r \cdot v)^{n_s}).$$

Il colore (R,G,B) degli oggetti è definito dai coefficienti diffusivi per ciascuna delle tre componenti colore, il modello che include anche il colore definito per l'oggetto diventerà:

$$\begin{array}{l} I_r = k_a I_a + \sum_{\ell_s} I_{\ell_s} (k_{dR}(\ell_s \cdot n) + k_s(r \cdot v)^{n_s}) \\ I_g = k_a I_a + \sum_{\ell_s} I_{\ell_s} (k_{dG}(\ell_s \cdot n) + k_s(r \cdot v)^{n_s}) \\ I_b = k_a I_a + \sum_{\ell_s} I_{\ell_s} (k_{dB}(\ell_s \cdot n) + k_s(r \cdot v)^{n_s}) \end{array} .$$

Il coefficiente di riflessione, invece, se la luce è bianca rimane uguale nelle tre equazioni. Il modello può essere esteso per considerare luci colorate.

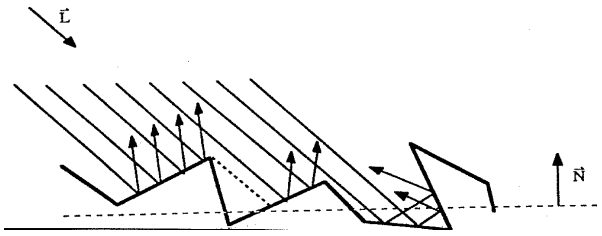


Figure 6.19: Microfaccette della superficie

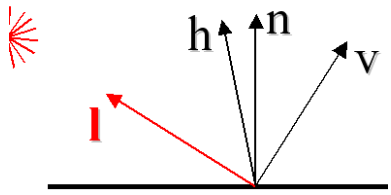


Figure 6.20: Blinn introduce il vettore h.

6.4.1 Modello di Blinn

Un miglioramento al modello di Phong nel calcolo dell'intensità dell 'illuminazione speculare è stato proposto da Blinn nel 1977. Una superficie ruvida è simulata come insieme di micro-faccette riflettenti orientate in modo casuale in tutte le direzioni su tutta la superficie (Fig. 6.19). In tal caso il modello di illuminazione è il seguente:

$$I = k_e + k_a I_a + \sum_{\ell_s} I_{\ell_s}(k_d(\ell_s \cdot n) + k_s \frac{DGF}{n \cdot v})$$

dove D è la funzione di distribuzione delle faccette, G la quantità di ombra delle micro-faccette e F la funzione di Fresnel (funzione del tipo di materiale considerato).

Il fattore $\frac{1}{n \cdot v}$ considera la geometria della vista: più il vettore vista si avvicina alla normale n e più sono le faccette che l'osservatore vede.

Variante più efficiente al modello

Blinn, per evitare di calcolare il vettore riflessione r, introduce il vettore

$$h = \frac{\ell + v}{\|\ell + v\|},$$

mostrato in Fig.6.20, che è la normale al piano passante per p che riflette la luce ℓ perfettamente nella direzione dell'osservatore. Quindi $n \cdot h$ è massimo per $\theta = 0$. Il modello diventa:

$$I = k_e + k_a I_a + \sum_{\ell_s} I_{\ell_s}(k_d(\ell \cdot n) + k_s(h \cdot n)^{n_s}).$$

6.5 Shading

Per calcolare l'intensità luminosa (luce irradiata) di un particolare punto sulla superficie bisogna considerare che, in generale, una superficie è curva, e quindi i vettori r, n, v visti si spostano muovendosi da punto a punto su di essa.

Idealmente quindi bisognerebbe calcolare il vettore n per ogni punto visibile e applicare poi il modello di illuminazione a quel punto. In pratica poligoni o superfici curve sono approssimate da mesh di poligoni. Su quali punti dobbiamo applicare il modello?

La risposta a tale domanda sono i tre differenti tipi di shading di cui si parlerà nel seguito: per-faccette (FLAT), per-vertici (GOURAUD) e per-pixel (PHONG).

6.6 Modello Flat shading

I tre vettori ℓ, n, v possono variare quando ci spostiamo da punto a punto su di una superficie. Per un poligono piatto, tuttavia, la normale n rimane costante.

Il modello flat shading valuta l'equazione di illuminazione di Phong una sola volta per ogni poligono (cioè considera i vettori ℓ, n, v costanti sul poligono) e assegna il colore ottenuto ad ogni punto del poligono.
Questo modello può essere considerato esatto nelle tre ipotesi seguenti:

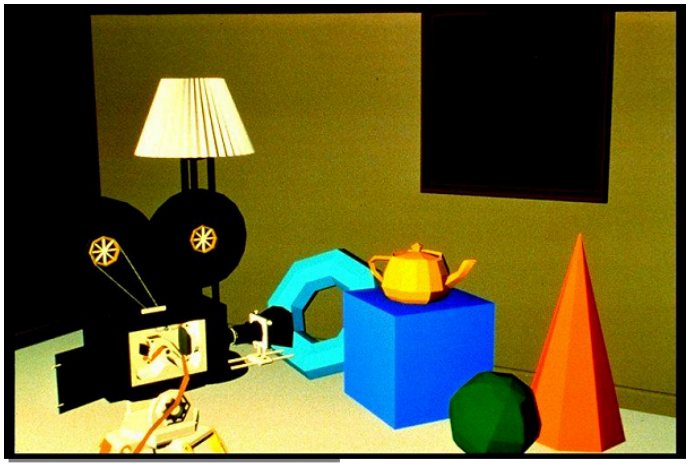


Figure 6.21: Scena resa con flat shading

IP1
: che la risorsa luminosa sia all' infinito, (sorgente direzionale)

IP2
: che il punto di vista sia all'infinito (proiezione parallela)

IP3
: che il poligono rappresenti l'esatta superficie e non ne sia un' approssimazione (es. cubo).

Un esempio di applicazione del flat shading per la resa di una scena, è mostrato in Fig.6.21.

- Se il modello è realmente a faccette e la scena è resa in proiezione prospettica e illuminata da luce puntiforme, questo modello è accurato? NO! La direzione (ℓ) dalla sorgente luminosa puntiforme varia all'interno della faccetta e per riflessione speculare, la direzione v dell'osservatore varia anch'essa per ogni punto della faccetta.
- Cosa succede se valutiamo il modello di illuminazione locale per ogni punto del poligono? La situazione non migliora più di tanto, l'apparenza è ancora chiaramente a faccette!

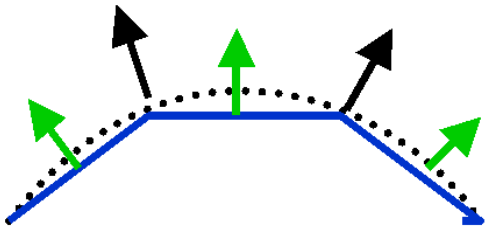


Figure 6.22: Normali ai vertici (nero), normali alle facce (verde)

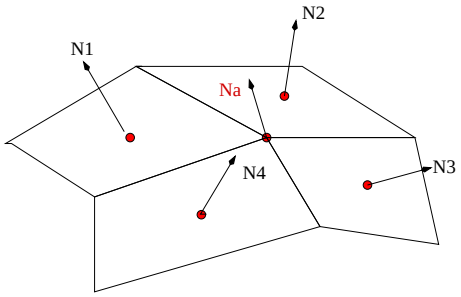


Figure 6.23: Calcolo normale N_a.

6.7 Modello Gouraud shading

Per migliorare il risultato ottenuto con il Flat Shading e quindi far sembrare più lisce le superfici, si introducono le normali ai vertici per ogni vertice (Fig. 6.22). Questo ci consentirà di creare una migliore approssimazione alla superficie reale che i poligoni approssimano.

La normale ai vertici può essere fornita con il modello, calcolata in modo esatto dal modello (es. per superfici spline), oppure calcolata in modo approssimato come media delle normali delle faccette che condividono il vertice.

In tal caso, usando la notazione di Fig.6.23, il vettore normale N_a usato per calcolare l'intensità del vertice a è determinato nel seguente modo:

$$N_a = \frac{N_1 + N_2 + N_3 + N_4}{|N_1 + N_2 + N_3 + N_4|}$$

dove N_i è la normale della faccetta i-esima. La normale N_a al vertice a è comune a tutti i poligoni che condividono il vertice a.

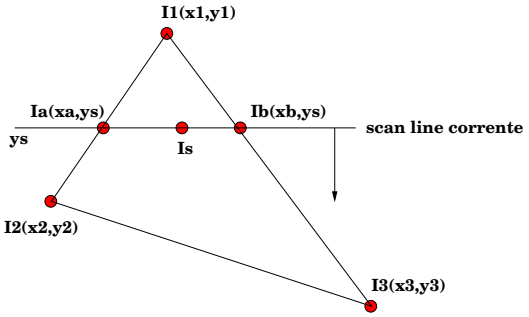


Figure 6.24: Calcolo intensità luminosa all'interno del poligono

Il Gouraud shading calcola l'intensità di ogni vertice di ciascun poligono usando un modello di illuminazione locale e le normali ai vertici. Per determinare l'intensità (il colore) di ciascun pixel all'interno del poligono sarà necessario interpolare bilinearmente i risultanti valori.

Il calcolo dell'interpolazione bilineare è effettuato in fase di rasterizzazione. L'intensità dei pixel interni ad un poligono è ottenuta interpolando i valori delle intensità dei suoi vertici. In riferimento alla Fig.6.24 si deve

- Interpolare linearmente lungo i lati: calcola I_a e I_b interpolando i valori dei relativi vertici;
- Interpolare linearmente lungo la scan-line: calcola i pixel interni al poligono;

Una scena resa con Gouraud shading è illustrata in Fig.6.25.

6.7.1 Gouraud shading: artefatti

La superficie resa con la tecnica Gouraud shading appare scura/opaca poichè la componente speculare, se inclusa, è mediata sull'intero poligono e può risultare difficile catturare effetti highlight come illustrato in Fig.6.26.

L'effetto a faccette prodotto dal flat shading (si veda ad esempio Fig.6.21) è dovuto al fatto che il calcolo dell'illuminazione di un certo poligono è indipendente dal calcolo dell'illuminazione dei poligoni ad esso adiacenti, E questo causa una netta visibilità dei bordi tra due poligoni adiacenti data dalla brusca variazione delle normali alle faccette.

Anche per quanto riguarda il Gouraud shading l'effetto faccette, sebbene

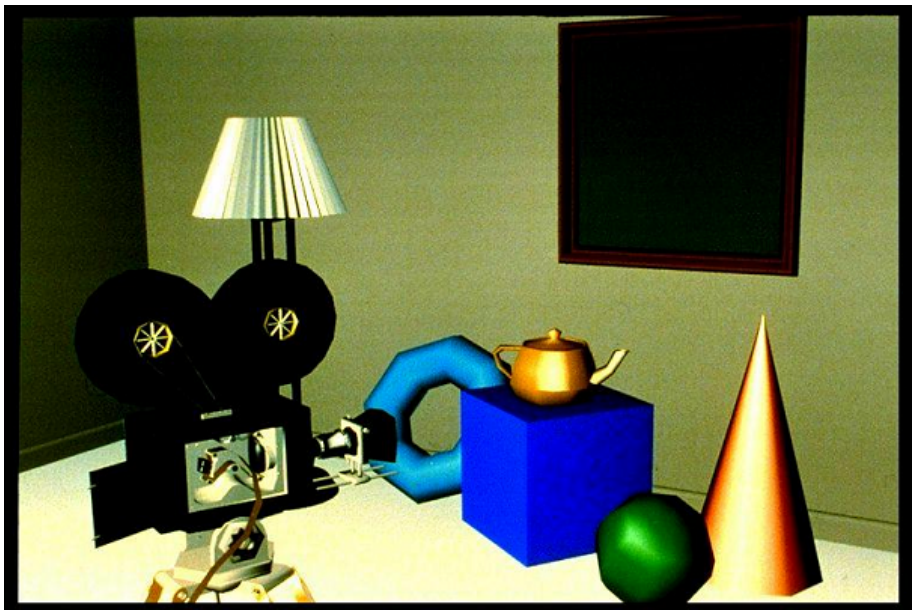


Figure 6.25: Scena resa con Gouraud shading

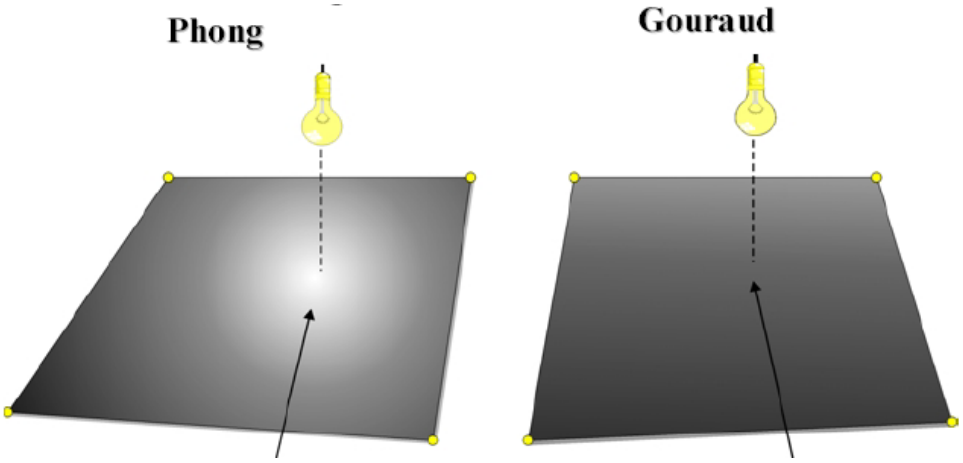


Figure 6.26: Effetti spotlight: Phong shading vs Gouraud shading

meno evidente, è ancora presente, si veda un esempio in Fig.6.27 e Fig.6.28. Nonostante con Gouraud shading le normali ai vertici del lato comune a due poligoni ora sono le stesse e producono per interpolazione gli stessi valori di illuminazione lungo il lato comune, l'interpolazione lineare con incrementi uniformi effettuata in screen space, per i punti in-

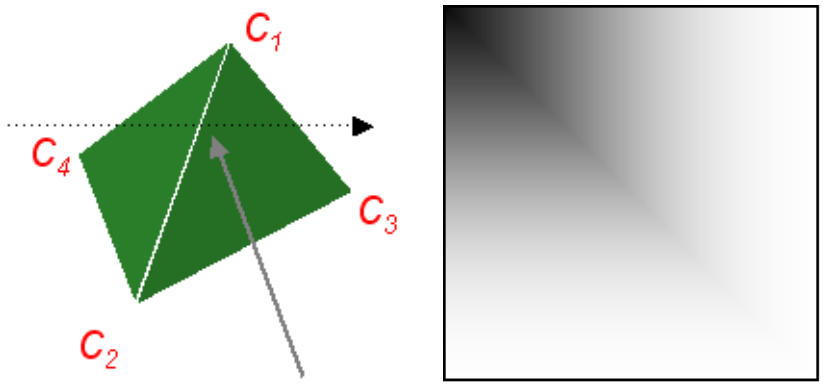


Figure 6.27: Artefatti del Gouraud shading: Mach Band

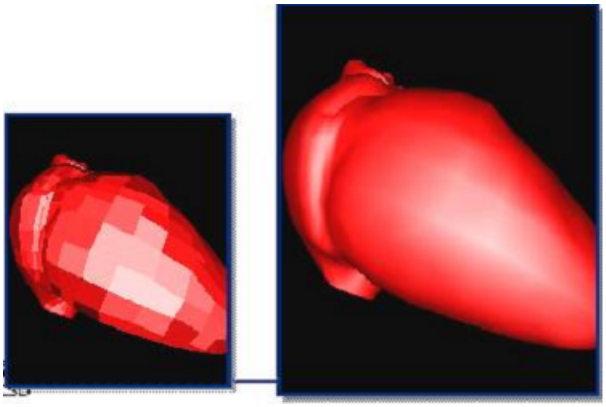


Figure 6.28: Artefatti del Gouraud shading: resa con flat shading (sinistra); resa con Gouraud shading (destra)

terni al poligono, non corrisponde a come realmente l'intensità riflessa varia attraverso le facce del poligono in coordinate mondo (WCS). Poichè abbiamo già elaborato una (non affine) proiezione prospettica per passare da WCS a coordinate schermo.

Questi visibili artefatti sono definiti **Mach Band**.

Una possibile soluzione è rappresentata dal calcolo dello shading in object space e non in screen space o dalla correzione prospettica (che tratteremo in dettaglio nel capitolo sulle texture).

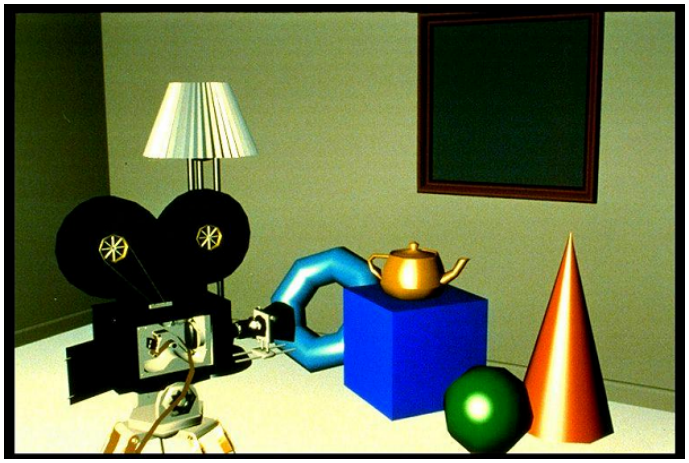


Figure 6.29: Scena resa con Phong shading

6.8 Modello Phong shading

Nel Phong shading (da non confondersi con il modello di illuminazione locale di Phong) la normale si calcola per ogni punto visibile (pixel) del poligono interpolando le normali dei vertici. L'effetto è quello di associare una sorta di curvatura a superfici a faccette. L'algoritmo per il calcolo dell'intensità luminosa per ogni pixel interno della faccetta può quindi schematizzarsi nelle fasi seguenti:

- calcolo delle normali ai vertici come media delle normali delle facette che condividono il vertice;
- interpolazione bilineare per ottenere le normali sui lati e all'interno del poligono;
- applicare il modello di illuminazione ad ogni pixel.

Il metodo risulta ovviamente meno efficiente rispetto al Gouraud shading; inoltre mentre il primo è realizzato in hardware, il Phong shading è realizzato off-line o tramite shaders.

La scena iniziale resa con Phong shading è illustrata in Fig.6.29. Un altro esempio dell'accuratezza delle tre diverse tecniche è illustrato in Fig.6.30.

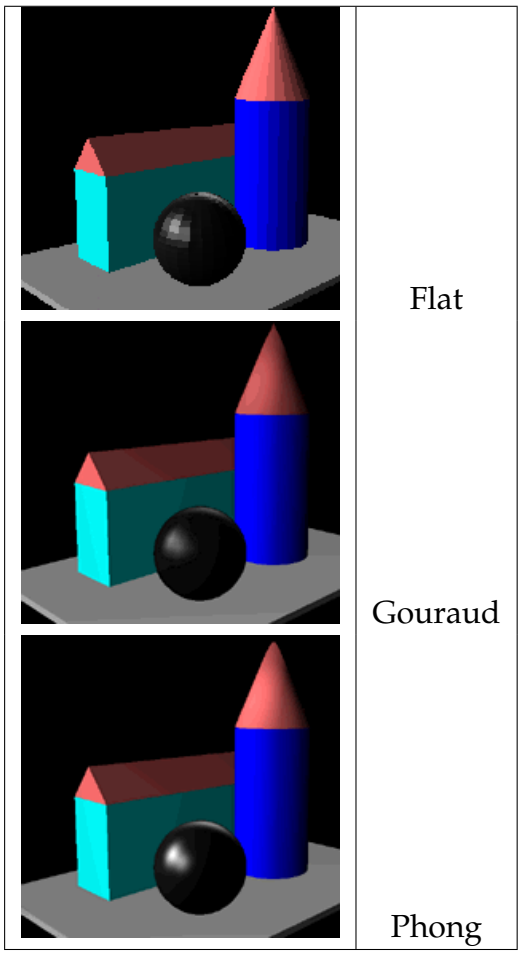


Figure 6.30: Flat, Gouraud e Phong shading a confronto.

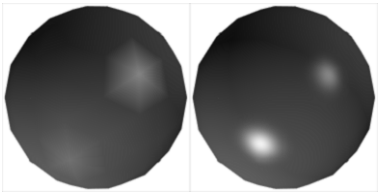


Figure 6.31: Gouraud Shading (sinistra); Phong shading (destra): artefatti nel profilo dell'oggetto

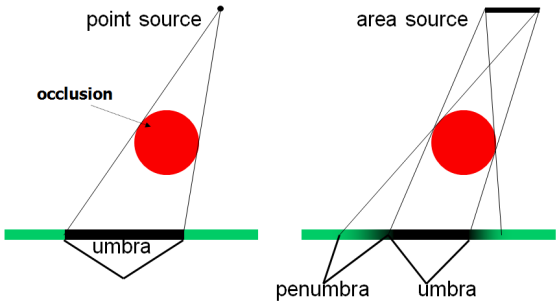


Figure 6.32: Ombra e penombra

6.8.1 Phong shading: artefatti

Se l'effetto è quello di rendere la curvatura dell'oggetto sebbene il modello sottostante sia a faccette, è inevitabile che il profilo dell'oggetto rimanga poligonale. Questo è ben visibile negli esempi illustrati in Fig.6.31.

Un modo per 'rimediare' a questi artefatti consiste ovviamente nel ricorrere ad una più raffinata suddivisione dei poligoni in scena, soprattutto attorno ai lati degli oggetti. Questo comporta un aumento del numero di faccette che permette di ottenere una miglior resa, al prezzo di un maggior onere computazionale.

6.9 Shadows: ombre geometriche

Le ombre sono importanti componenti di immagini realistiche per mettere in maggior risalto le relazioni spaziali tra gli oggetti in scena. L'ombra di un oggetto dipende dalla forma dell'oggetto e dalla posizione della sorgente luminosa.

Poichè un'area in ombra pur non essendo soggetta a luce diretta può ricevere illuminazione indiretta da altri oggetti vicini, l'intensità esatta prodotta dall'ombra viene calcolata solo con modelli di illuminazione GLOBALI.

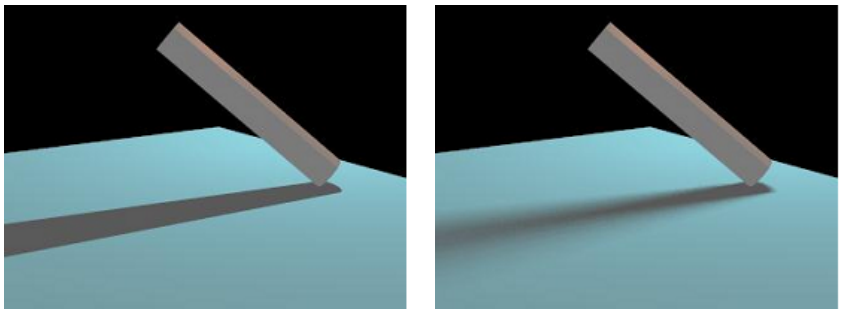


Figure 6.33: Ombre hard-edged e soft-edged

6.9.1 Shadows: alcune proprietà

Un punto è in ombra se non è illuminato da alcuna risorsa luminosa, o , equivalentemente, se un osservatore posto in quel punto non può vedere alcuna risorsa luminosa. Quindi non ci sono ombre se il punto di vista è coincidente con una (UNICA) risorsa luminosa. Il fatto che le ombre siano aree nascoste alla sorgente luminosa è alla base dell'algoritmo shadow map che verrà presentato nel seguito che fa uso dell'algoritmo di hidden surface per risolvere il problema.

Una realistica rappresentazione dell'ombra è definita da ombra e penombra; come illustrato in Fig.6.32. L' area in ombra è completamente nascosta alla risorsa luminosa, mentre l'area in penombra riceve in parte luce dalla sorgente; una penombra circonda un'ombra con cambio graduale di intensità; se le risorse luminose sono puntiformi allora non c'è penombra e l'ombra è HARD-EDGED. Un'ombra SOFT-EDGED contiene sia ombra che penombra come illustrato in Fig.6.33.

Nelle scene statiche l'ombra non cambia al variare del punto di vista, mentre nelle animazioni si deve ricalcolare il contributo dell'ombra prodotta dagli oggetti in movimento (grande overhead in animazioni).

Presentiamo nel seguito tre metodi per aggiungere il contributo ombra ad una scena resa con un modello di illuminazione locale: il primo calcola la forma di un'area in ombra prodotta da un singolo punto luce su un oggetto posto su di un piano, ombra che viene poi aggiunta come oggetto alla scena, il secondo si basa sull'idea di determinare una mappa della scena in ombra, e il terzo determina quali superfici sono visibili mediante

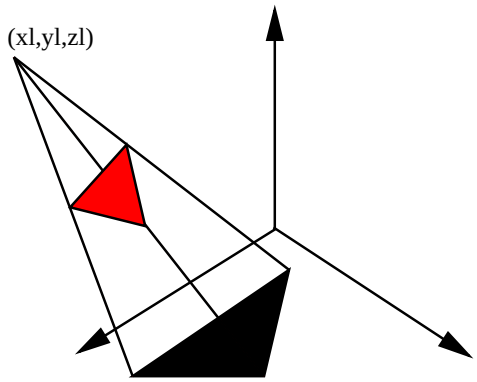


Figure 6.34: Ombre sul piano.

calcolo dello spazio volumetrico in ombra prodotto da ciascun poligono.

6.9.2 Metodo 1: Ombre sul piano (Blinn 1988)

Consideriamo un punto luce che illumina un singolo oggetto isolato che genera l'ombra su un piano. Si consideri l'esempio del singolo poligono mostrato in Fig.6.34.

L'ombra in questo caso ha la forma di un poligono chiamato **shadow polygon**, ottenuto proiettando il poligono sul piano (rappresentato dall' equazione $z = 0$) con centro di proiezione COP posizionato nel punto luce (x_ℓ, y_ℓ, z_ℓ).

Per ogni vertice del poligono $P(x_p, y_p, z_p)$ passa un raggio ombra dal COP=(x_ℓ, y_ℓ, z_ℓ) , punto luce, al piano colpendolo in $S(x_s, y_s, 0)$.

Ricostruiamo il raggio ombra in forma parametrica:

$$S = P - \alpha L.$$

Poichè $z_s = 0$, si determina il parametro α dalla relazione

$$0 = z_p - \alpha z_\ell, \qquad \alpha = z_p / z_\ell.$$

Per ottenere poi le coordinate (x, y) del vertice S dello shadow polygon si

sostituisce l'α ottenuto:

$$\begin{array}{rl} x_s = & x_p - \frac{z_p}{z_\ell} x_\ell \\ y_s = & y_p - \frac{z_p}{z_\ell} y_\ell \end{array} .$$

In forma matriciale:

$$\begin{bmatrix} x_s \\ y_s \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\frac{x_\ell}{z_\ell} & 0 \\ 0 & 1 & -\frac{y_\ell}{z_\ell} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} .$$

Moltiplicando per tale matrice tutti i vertici del poligono che produce l'ombra, si ottengono i vertici dello shadow polygon sul piano.

Ovviamente questo semplice approccio non può essere utilizzato per ottenere l'ombra che un oggetto forma su altri oggetti, nè l'ombra di parti dell'oggetto sull'oggetto stesso.

6.9.3 Metodo 2: shadow Z-buffer (Williams 1978)

L'idea su cui si basa l'algoritmo è che non ci sono ombre se il punto di vista è coincidente con una (UNICA) risorsa luminosa; infatti le ombre sono aree nascoste alla sorgente luminosa. Quindi algoritmi di hidden surface removal possono essere usati per calcolare l'ombra.

L'algoritmo richiede uno z-buffer aggiuntivo nel frame buffer, detto shadow z-buffer per ogni risorsa luminosa. Il processo può essere descritto in due passi:

- **PASSO 1**: rendere la scena usando la risorsa luminosa come punto di vista; si memorizza l'informazione nello shadow Z-buffer (Depth image dei poligoni visibili dalla risorsa luminosa)
- **PASSO 2**: rendere la scena usando l'algoritmo Z-buffer:
  - se un punto (x,y,z) è visibile dal punto di vista, trasforma (x,y,z) in (x',y',z') coord. del punto rispetto al frame che ha come origine la sorgente di luce;
  - Se $z' > shadowzbuffer(x', y')$ allora c'è una superficie più vicina alla risorsa luminosa rispetto al punto considerato e il punto

è in ombra, calcola un'intensità ombra;
– altrimenti il punto è reso normalmente.

6.9.4 Metodo 3: shadow volume

L'algoritmo si basa sulla determinazione dei volumi di spazio in ombra creati da ogni poligono esposto alla luce. Il volume di ombra di un triangolo è definito dal tetraedro troncato di vertice il punto luce e facce laterali i quadrilateri formati dai lati del triangolo e dalla loro proiezione su un piano, e faccia superiore definita dal triangolo stesso.

Per ogni shadow volume si classificano le sue facce in frontfacing (faccia del shadow volume rivolta verso l'osservatore) o backfacing.

Per determinare se un punto su un poligono è in ombra si contano il numero di facce intersecate da un raggio che parte dal punto stesso in direzione del punto di vista. Se si intersecano più frontfacing rispetto a backfacing, allora il punto è in ombra.

L'algoritmo si può realizzare efficientemente utilizzando lo stencil buffer contenuto nel frame buffer. L'algoritmo si svolge mediante 4 passi (Heidmann91):

- Passo 1: azzerare lo stencil buffer, resa della scena con luce ambiente nello Z-buffer (per memorizzare le info su depth). Disattivare aggiornamento Z-buffer e scrittura nel color buffer (è attivo il test nello Z-buffer, disegno solo nello stencil).
- Passo 2: resa delle facce frontfacing di tutti gli shadow volume nello stencil buffer, incrementare contatore.
- Passo 3: resa delle facce backfacing di tutti gli shadow volume nello stencil, decrementare il contatore.
- Passo 4: resa normale (componente diffusiva e speculare) di tutti i soli pixel con corrispondente valore nello stencil buffer pari a 0.

6.10 Modelli di illuminazione Globale

Il modello di illuminazione locale presenta diverse limitazioni. Consideriamo per esempio una fila di sfere su un piano illuminate da una sorgente

in linea con esse. La sfera più vicina alla sorgente blocca parte della luce sulle altre sfere, tuttavia, in un modello locale, ogni sfera è resa indipendentemente dalla presenza di altri oggetti in scena e quindi tutte le sfere appaiono ugualmente illuminate. Inoltre se queste sfere sono riflettenti, parte della luce riflessa contribuisce all'illuminazione delle altre sfere, fino a far riflettere le sfere in loro stesse come effetto speculare. Il nostro modello di illuminazione locale non può gestire questa situazione, nè può produrre ombre corrette. Tutti questi fenomeni, ombre, riflessioni, blocco della luminosità, sono effetti globali e richiedono un modello di illuminazione globale.

I modelli globali però sono incompatibili con la classica **pipeline di rendering** poichè quest'ultima rende ogni poligono in modo indipendente dagli altri poligoni.

Due strategie di rendering alternative che possono gestire effetti di illuminazione globale sono le seguenti:

Ray Tracing
: Global specular interaction; Dipende dal punto di vista,

Radiosity
: Global diffuse interaction; NON dipende dal punto di vista.

Il ray tracing viene proposto negli anni '80 come tentativo di estendere il modello di illuminazione locale ad un modello di illuminazione globale per produrre immagini realistiche. Il modello originale proposto da Whitted e Kay ('80) è un algoritmo di ray tracing di illuminazione globale che gestisce:

- illuminazione e shading (dovuto all'illuminazione diretta)
- hidden surface removal
- contributo di luce riflessa
- trasparenza/ rifrazione
- ombre (shadows).

6.11 Ray Tracing

L'idea generale è la seguente. Un oggetto è visibile da un osservatore grazie ai raggi luminosi che lo colpiscono e arrivano all'osservatore stesso.

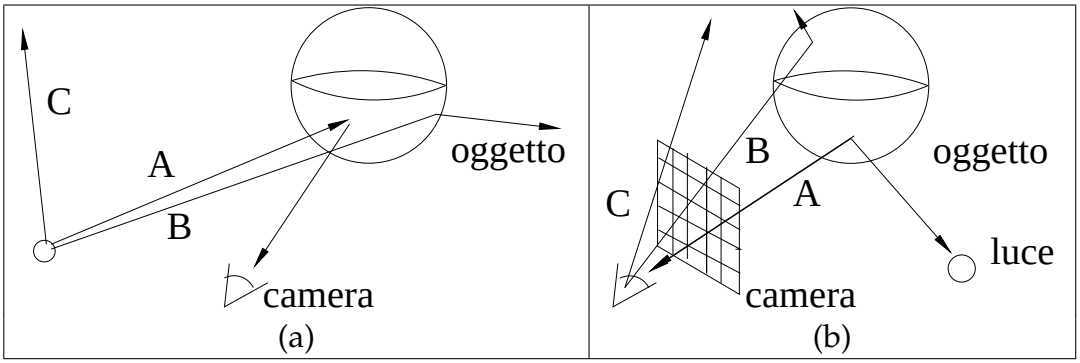


Figure 6.35: (a) Situazione reale; (b) Situazione considerata dal ray tracing

L'idea è quella di far partire i raggi dall'osservatore e quindi seguire il percorso inverso verso le sorgenti luminose. I raggi dall'osservatore verso la scena possono: intersecare una risorsa luminosa, non colpire alcun oggetto e quindi ritornare il colore dello sfondo, altrimenti colpire il primo oggetto intersecato in scena e ritornarne il colore. Ma vediamo passo passo cosa succede a partire da una semplice situazione di un unico oggetto opaco in scena.

Caso di studio: Consideriamo un'unica sfera isolata in scena.

Situazione reale, illustrata in Fig.6.35(a): I raggi luminosi (fotoni) lasciano la risorsa luminosa, colpiscono la scena, la maggioranza dei raggi (C) esce di scena poichè non interseca la sfera, alcuni (A) vengono riflessi verso la camera, e quindi saranno visti, altri (B) saranno riflessi altrove, non colpiranno l'osservatore e non saranno visti.

Poichè siamo interessati solo a quei raggi luminosi che terminano sull' osservatore, ha perciò senso partire dall'osservatore e tracciare i raggi verso la scena. Così, invece di tener traccia dei raggi luminosi che dalla risorsa luminosa colpiscono la superficie in scena, si considera solo un numero limitato di raggi luminosi nella direzione inversa di propagazione dall'osservatore alla scena fino alla sorgente luminosa, situazione illustrata in Fig.6.35(b). In realtà i raggi da considerare sono solo quelli che partono dal centro di proiezione e passano per il piano di vista (viewport).

Poichè dobbiamo assegnare un colore ad ogni pixel, dovremo generare almeno un raggio per ogni pixel. Ogni raggio, o interseca una superficie,

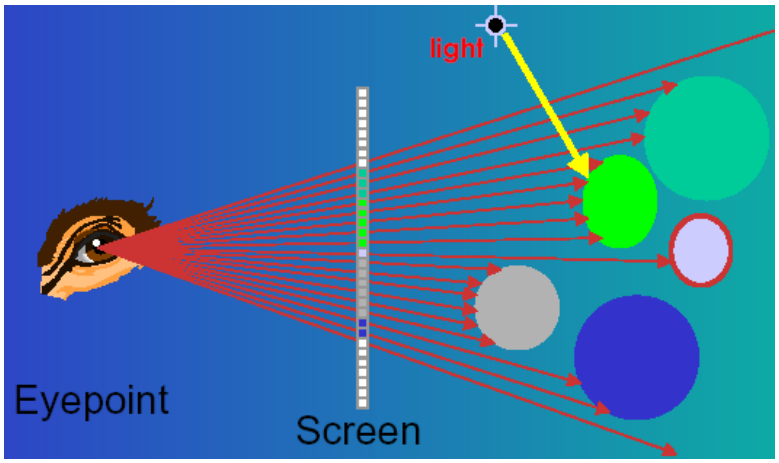


Figure 6.36: Ray-casting

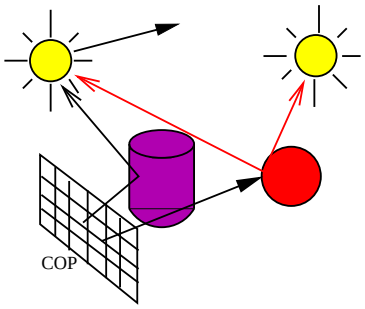


Figure 6.37: Raggi shadow in rosso.

o colpisce una risorsa luminosa o esce di scena senza colpire nulla. A pixel che corrispondono a quest'ultimo caso verrà assegnato un colore di background.

Iniziamo a descrivere nel dettaglio una versione semplificata del ray tracing, il piú semplice algoritmo di ***RAY CASTING***, che non considera il contributo luminoso riflesso dovuto alla presenza di altri oggetti in scena ma che tuttavia gestisce sia l'**illuminazione diretta** che la **visibilità delle superfici** e le **ombre**.

L'algoritmo utilzza il modello di riflessione locale di Phong per il calcolo di illuminazione diretta dei punti visibili dall'osservatore (primi punti di intersezione).

Per ogni pixel (x, y) della viewport, l'algoritmo costruisce un raggio dal punto di vista in scena (ray) e assegna al pixel il colore del primo oggetto intersecato

$$color[x, y] = castRay(ray),$$

si veda Fig.6.36.

```
color c=castRay(ray)
begin
hitObject(ray, hitpoint, n, object);
color = object color;

if (object is light)
    return(color);
else
    return(lighting(hitpoint,n,color);
end
```

Come si realizza l'operazione **castRay(ray)**?

Sia p il primo punto di intersezione (oggetto) che il raggio incontra. L'intensità e il colore di questo punto p è calcolata come contributo di tutte le sorgenti luminose che incidono su p. Tracciando da p un raggio (detto raggio shadow) per ogni risorsa luminosa:

- se un raggio shadow interseca una superficie prima di incontrare la risorsa luminosa, la luce non raggiunge il punto p considerato che quindi non avrà il contributo di quella risorsa luminosa. Di conseguenza, se nessuna risorsa luminosa è raggiungibile dal punto p allora il punto è in **ombra**;
- se almeno un raggio shadow non incontra occlusioni: il punto p risulterà illuminato. L'intensità (colore) del pixel viene calcolata dall' applicazione del modello di illuminazione di Phong nel punto intersezione p sommando il contributo di ogni risorsa luminosa visibile da p.

In Fig. 6.37 i raggi shadow sono mostrati in rosso.

Il raycasting opera un'illuminazione diretta e in più **rimuove superfici non visibili** e aggiunge **ombre**, al costo dei calcoli di intersezione oggetto/risorse luminosa.

L'algoritmo completo di rayCasting è schematizzato nel seguito.

```
col c=castRay(ray)
begin

hitObject(ray, hitpoint,n, object);
col = object color;

For every light L
    Construct rayS(hitpoint, L->getDir());
    For every object ob
          hit2=intersect(rayS, ob);
      If (rayS does NOT intersect any ob)
          col=col+lighting(L,hitpoint, n)
end
```

Per gestire inter-riflessioni e trasparenze l'algoritmo si potenzia e diventa **ray tracing**.

6.11.1 Gestione raggi riflessi

Per ora si sono considerati raggi luminosi che hanno origine dalla risorsa luminosa colpiscono una superficie e si riflettono verso l'osservatore. In realtà la luce compie percorsi molto più articolati, rimbalzando in scena molte volte prima di raggiungere l'osservatore grazie ad oggetti riflettenti presenti, e apportando così un contributo luminoso su tutti gli oggetti che vengono incontrati nel percorso dei raggi (inter-riflessioni).

Per tener conto almeno parzialmente di questo effetto, ad ogni intersezione raggio-oggetto viene generato un nuovo raggio (raggio riflesso) nella direzione della riflessione speculare perfetta. I raggi riflessi 'rimbalzano' da superficie a superficie contribuendo all'intensità luminosa di ciascuna superficie che intersecano, finchè intersecano una risorsa luminosa o raggiungono un massimo numero di 'rimbalzi'.

Questi calcoli sono fatti ricorsivamente e devono tener conto dell' attenuazione dovuta alla distanza, cioè dell'assorbimento della luce da parte delle superfici.

La direzione di riflessione è gestita esattamente come se fosse il raggio dall'osservatore (raggio primario).

Stiamo seguendo un raggio la cui direzione r è determinata a partire dal vettore v (che proviene o direttamente dall'osservatore o in generale da un qualunque altro punto di intersezione con altri oggetti) e dalla normale n alla superficie nel punto.

La formula è derivata in modo identico alla formula del raggio riflesso nel modello di Phong ma attenzione qui c'è v invece della direzione del raggio incidente di luce. Quindi il raggio riflesso cercato si ottiene dalla formula:

$$r = 2(n \cdot v)n - v,$$

dove v ed n sono i vettori unitari rappresentanti rispettivamente il vettore raggio luce incidente (proveniente in generale da un altro oggetto) e la normale alla superficie. Il raggio v che stiamo tracciando ha in realtà direzione opposta, cioè incidente nel punto e non uscente, perciò consideriamo il verso opposto:

$$r = v - 2(n \cdot v)n.$$

6.11.2 Gestione dei raggi rifratti (trasparenza)

Il ray tracing è in grado di gestire sia superfici riflettenti che superfici che lasciano passare parzialmente o totalmente la luce (Fig.6.38).

La luce che colpisce un oggetto trasparente/ semitrasparente viene in parte **trasmessa** (i raggi sono rifratti). La rifrazione permette di rendere visibili oggetti che altrimenti non lo sarebbero. Oggetti visti attraverso materiali semitrasparenti risultano però deformati.

Per gestire la trasparenza degli oggetti con ray tracing ad ogni intersezione raggio-oggetto semitrasparente viene generato un nuovo raggio nella direzione della trasparenza (secondo la legge di rifrazione) e calcolato il contributo trasmesso.

Ogni volta che un raggio colpisce una superficie produce, in generale, un raggio riflesso ed uno rifratto. In figura 6.39 sono mostrati i vettori incidente (ℓ) e trasmesso (T).

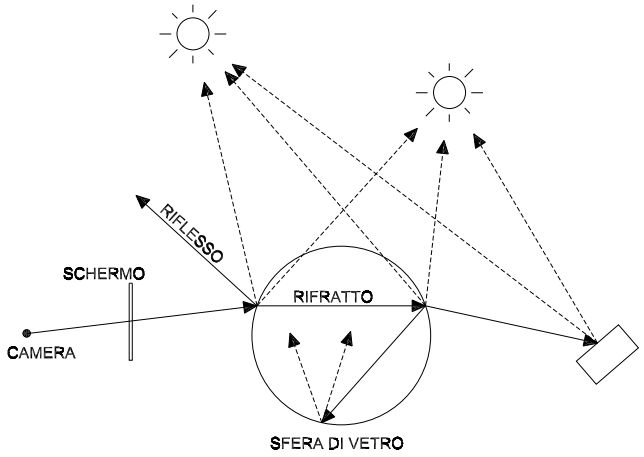


Figure 6.38: Raggi riflesso, trasmesso (rifratto) in una scena con una sfera di vetro e due risorse luminose.

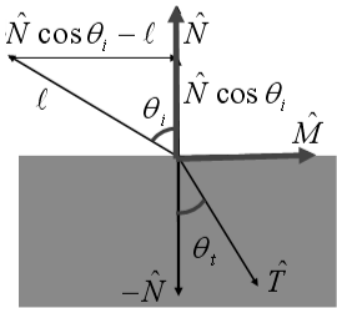


Figure 6.39: Raggio trasmesso T

Un raggio che colpisce un oggetto interamente trasparente è rifratto dovuto al cambio della velocità della luce in differenti mezzi.

Il cambio di direzione (raggio T) è calcolato tramite la **Legge di Snell** che considera gli angoli di incidenza e gli indici di rifrazione dei mezzi:

$$\frac{\sin\theta_t}{\sin\theta_i} = \frac{\mu_2}{\mu_1} = \mu$$

con μ_2, μ_1 coefficienti di rifrazione dei mezzi attraversati.

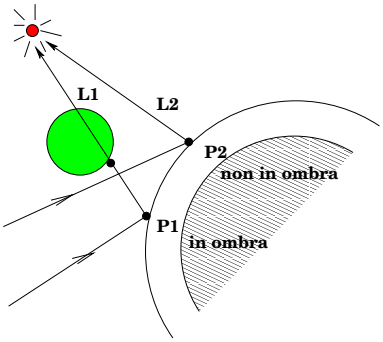


Figure 6.40: Ombre generate con ray tracing

Calcolo del raggio trasmesso. Dalle espressioni

$$T = \sin(\theta_t)M - \cos(\theta_t)N \qquad M = \frac{(N\cos(\theta_i) - \ell)}{\sin(\theta_i)}.$$

Sostituendo l'espressione di M in quella di T si ottiene:

$$T = \frac{\sin\theta_t}{\sin\theta_i}(N\cos(\theta_i) - \ell) - \cos(\theta_t)N = (\mu\cos(\theta_i) - \cos(\theta_t))N - \mu\ell$$

Poichè $\cos(\theta_i) = N \cdot \ell$, e $\cos(\theta_t) = (1 - \mu^2(1 - \cos^2(\theta_i))^{1/2}$ si ha:

$$T = (\mu(N \cdot \ell) - (1 - \mu^2(1 - (N \cdot \ell)^2)^{1/2})N - \mu\ell$$

Angolo critico (ℓ dato da $\sin(\theta_t) = \mu$): la luce trasmessa va nella direzione lungo la superficie.

6.11.3 Ombre: shadow ray

Le ombre con l'algoritmo di ray tracing vengono generate tramite il controllo sugli shadow ray. Il calcolo del colore locale richiede infatti il controllo per vedere se la risorsa luminosa è visibile dal punto di intersezione p.

Quindi, come già visto, si deve generare un raggio shadow L (vettore direzione della luce) da p verso la risorsa luminosa e controllare se questo

interseca qualche altro oggetto (Fig.6.40). Se L interseca un oggetto allora p non avrà il contributo luminoso da quella sorgente e l'intensità luminosa in quel punto è quindi ridotta. Per N risorse luminose si generano N test di intersezione.

Per generare una soft shadow, ovvero zone in ombra e in penombra, è necessario considerare una risorsa luminosa ad area. Raggi shadow multipli vengono generati a partire dal punto di intersezione fino a punti diversi all'interno della sorgente luminosa ad area. Per una più realistica rappresentazione della penombra è necessario coprire con un numero consistente di raggi casuali l'intera area luminosa.

Osservazione 1: Luci riflesse da specchi non provocano ombra. Infatti i raggi shadow sono lanciati solo verso le sorgenti luminose.

Osservazione 2: Il contributo di intensità luminosa diffusa è considerato solo nel modello di illuminazione locale applicato al punto p di prima intersezione. Non ha contributi derivanti da altri oggetti.

Per risorse luminose non puntiformi ma ad area si generano multipli raggi shadow da uno stesso punto intersezione per campionare la risorsa luminosa ad area. Alcuni di questi raggi raggiungono la sorgente producendo un contributo di penombra, mentre se nessun raggio raggiunge la sorgente il punto è in ombra. Si creano quindi *ombre soft* con ombra attorniata da penombra.

6.11.4 Algoritmo di ray tracing ricorsivo

Dall'algoritmo di raycasting proposto da Appel nel 1968, l'idea è stata ripresa da Whitted nel 1980 con una proposta di algoritmo di ray tracing ricorsivo.

Se almeno una risorsa luminosa è visibile da un punto di intersezione sulla superficie, allora dobbiamo:

- generare un raggio nella direzione di ogni sorgente luminosa **(raggi shadow L)**
- calcolare il contributo di quella risorsa luminosa in quel punto usando un modello di illuminazione locale (es. Phong);
- generare un raggio nella direzione della perfetta riflessione **(raggio R)**

Figure 6.41: Scene rese con ray tracing. Courtesy of *xcmodel* CG Group, University of Bologna,ITALY

- generare un raggio nella direzione del raggio trasmesso **(raggio T)**

Questi ultimi due raggi vengono chiamati secondari e verranno trattati come i raggi primari. Ad ogni nuova intersezione che questi raggi hanno con altre superfici, ulteriori raggi di riflessione e trasmissione possono venire generati e il contributo locale viene determinato utilizzando i raggi shadow verso le sorgenti luminose.

Assumiamo per semplicità una sola risorsa luminosa; per più risorse sommare in modo analogo i vari contributi. Di seguito riportiamo in pseudocodice la routine **raytraceImage** per la generazione dei raggi primari per ogni pixel, e la routine **trace** che segue il percorso dei raggi in modo ricorsivo.

```
Image raytraceImage(vector eye)
begin
Image image;
for (each scanline Y)
    for (each pixel X)
        image[X][Y]=trace(eye,dirFromEyeToXY, RecursionStep)
     end
end
return image;
end
```

L'algoritmo ricorsivo di tracing richiama due funzioni base:

- **trace()** funzione ricorsiva che segue un raggio, specificato da un punto e da una direzione, e ritorna il colore (shade) della prima superficie che interseca. (max = numero massimo di passi ricorsivi consentiti)
- **intersect()** funzione che trova il punto di intersezione con la più vicina superficie che il raggio interseca. Deve perciò avere a disposizione tutto l'intero database di oggetti. Se il raggio interseca una risorsa luminosa, allora il colore ritornato dal ray tracer deve essere il colore della risorsa luminosa.

```
color c=trace(point p, vector d, int step)
begin
color locale, riflesso, trasmesso; point q; normal n;

if (step > max) return(background-color);
q=intersect(p,d,status);

if (status== light-source) return(light-source-color);
if (status== no-intersection) return(background-color);

/* calcolo della normale al punto di intersezione,
della direzione del raggio riflesso e del raggio trasmesso */
n=normal(q);
r=reflect(q,n);
t=transmit(q,n);

locale=phong(q,n,r);
riflesso=trace(q,r,step+1);
trasmesso=trace(q,t,step+1);

return(locale+trasmesso+riflesso);
end
```

6.11.5 Quando termina il processo ricorsivo?

Il processo è ricorsivo e termina quando l'energia di un raggio scende al di sotto di una data soglia, o se si è raggiunto un numero massimo di rimbalzi in scena.

Per il calcolo della frazione di energia rimasta e quindi trasmessa dal raggio quando, colpendo una superficie, viene riflesso e/o rifratto aggiungiamo un parametro energia alla chiamata ricorsiva:

color c=trace(point p, vector d, int step, float energy)

quindi si deve solo aggiungere una linea di codice alla routine per controllare se c'è sufficiente energia rimasta per continuare a tracciare il raggio.

Il processo si può visualizzare come un **albero** dove ogni nodo è un punto

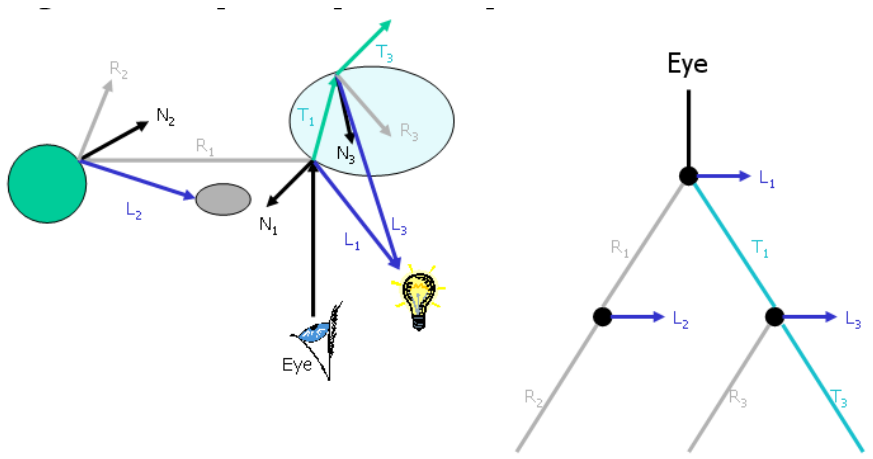


Figure 6.42: Albero dei raggi: tre oggetti in scena ed una sola risorsa luminosa.

colpito sulla superficie. Ad ogni nodo si genera un raggio shadow e un raggio riflesso, un raggio trasmesso (rifratto) o entrambi.

In figura 6.42 è illustrata la generazione di un singolo raggio e il percorso seguito da questo, a destra invece è mostrato il corrispondente albero di raggi generato.

Il modello originale proposto da Whitted e Kay calcola l'intensità luminosa come somma di un contributo locale I_{locale} e di un contributo globale intensità ottenuto ricorsivamente mediante i raggi riflessi $I_{riflessa}$ e a quelli trasmessi $I_{trasmessa}$. L'intensità di un punto è quindi data da:

$$I = I_{locale} + K_r * I_{riflessa} + K_t * I_{trasmessa}$$

dove:

- $K_r \in [0,1]$ coefficiente di riflettività del materiale
- $K_t \in [0,1]$ coefficiente di trasparenza/opacità del materiale
- $I_{riflessa}$ intensità proveniente dalla direzione speculare;
- $I_{trasmessa}$ intensità proveniente dalla direzione trasmessa;
- I_{locale} intensità luminosa diretta calcolata con un modello di illuminazione locale.

Per illustrare gli effetti della profondità di ricorsione sulla scena resa, osserviamo la scena illustrata in Fig.6.43 con profondità di ricorsione vari-

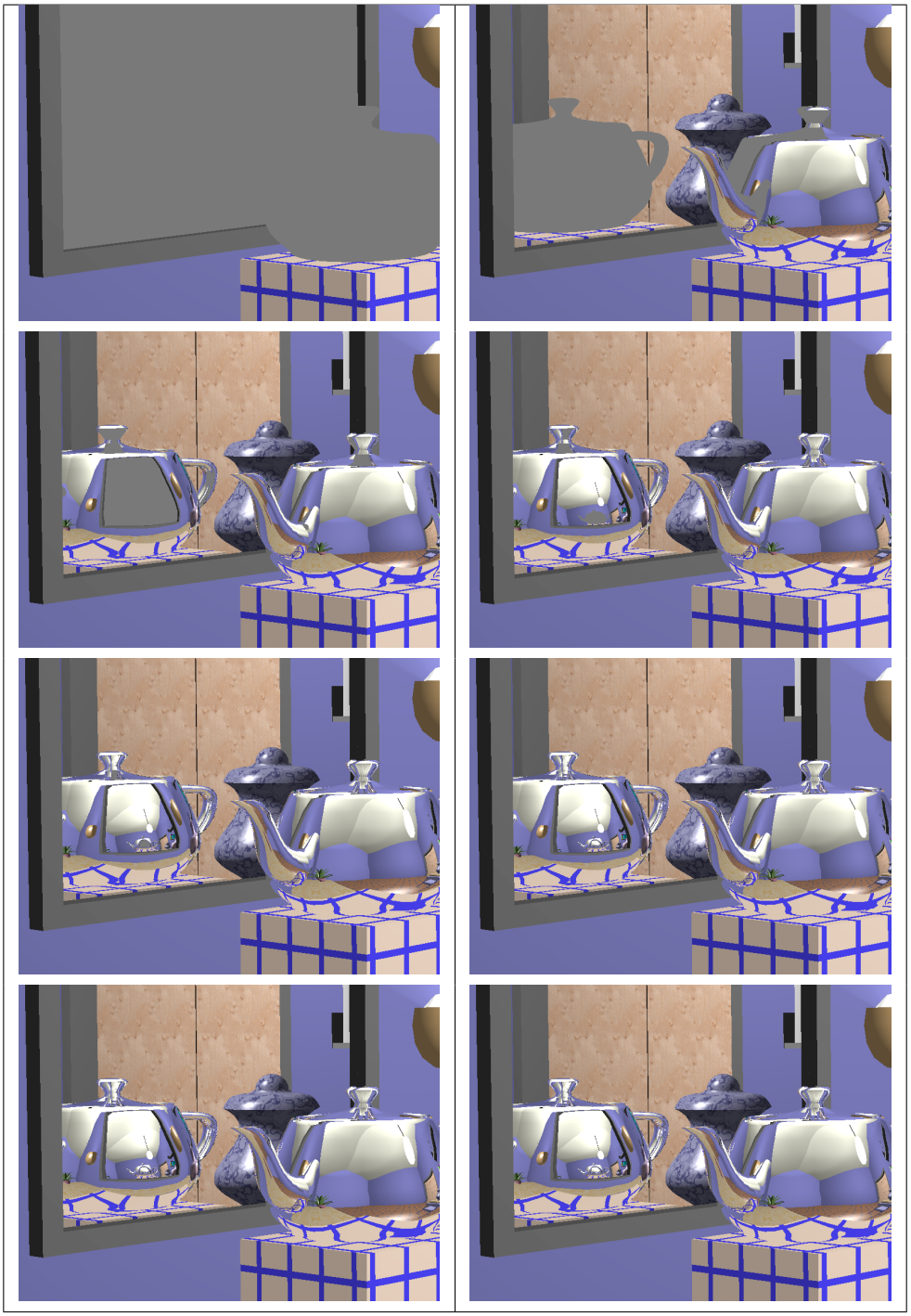


Figure 6.43: Effetti della profondità di ricorsione per numero ricorsioni crescente a partire da 0 (in alto a sinistra) fino a 7 (in basso a destra).

ante da 0 a 7. Teapot e specchio sono perfettamente riflettenti senza componente locale (solo luce ambiente).

Per **depth=0**, i raggi terminano su specchio e teiera e non hanno contributo locale. Per **depth=1**, i raggi osservatore/specchio/teapot terminano alla teapot e restituiscono un'ombra grigia (colore del background) sullo specchio.

6.11.6 Ray tracing: intersezione raggio-superficie

Il compito più oneroso dell'algoritmo è il calcolo dell'intersezione raggio/superficie. Più oggetti si aggiungono in scena e più diventa onerosa l'intersezione. La maggioranza dei ray tracer supportano solamente superfici a faccette piane.

In generale vengono calcolate le bounding box per ogni superficie, evitando calcoli inutili se il raggio non interseca le bounding box, viceversa, se il raggio interseca almeno una bounding box, allora è probabile che intersechi anche la superficie e si procede quindi al calcolo esatto dell'intersezione raggio superficie.

Intersezione raggio/triangolo

Si calcola il punto di intersezione tra il raggio, uscente da P_1 in direzione d, rappresentato in forma parametrica dall'equazione: $P(t) = P_1 + td$ e il piano che contiene il triangolo. Infine, si controlla se tale punto di intersezione è dentro il triangolo mediante verifica che stia dalla stessa parte rispetto alle rette (piani) passanti per i tre lati del triangolo. Utilizzando le coordinate baricentriche di un punto P rispetto ai vertici del triangolo $< a, b, c >$, si ha $P(\alpha, \beta, \gamma) = \alpha a + \beta b + \gamma c$. Se il punto è interno al triangolo le sue coordinate baricentriche sono positive, altrimenti è esterno.

Intersezione raggio/sfera

Si vuole calcolare l'intersezione del raggio in forma parametrica da un punto P_1 in direzione $d =< i, j, k >$:

$$P(t) = P_1 + td = \begin{pmatrix} x = x_1 + it \\ y = y_1 + jt \\ z = z_1 + kt \end{pmatrix}$$

con la sfera di raggio r e centro (l, m, n):

$$f(x, y, z) = (x - l)^2 + (y - m)^2 + (z - n)^2 - r^2 = 0$$

Sostituendo per x, y, z si ha $at^2 + bt + c = 0$ dove

$$\begin{aligned} a &= i^2 + j^2 + k^2 \\ b &= 2i(x_1 - l) + 2j(y_1 - m) + 2k(z_1 - n) \\ c &= l^2 + m^2 + n^2 + x_1^2 + y_1^2 + z_1^2 + 2(-lx_1 - my_1 - nz_1) - r^2 \end{aligned}$$

Risolvendo per t otteniamo nessuna, una o due intersezioni (se il discr.$>$ 0). In caso di due intersezioni otteniamo i due punti di intersezione entrante ed uscente del raggio con la sfera.

Intersezione raggio/superficie NURBS

Sia

$$S(u, v) = \frac{\sum_i \sum_j P_{ij} w_{ij} N_{im}(u) N_{jm}(v)}{\sum_i \sum_j w_{ij} N_{im}(u) N_{jm}(v)}$$

con $P_{ij} = (x_{ij}, y_{ij}, z_{ij})$ superficie NURBS. Il raggio sia definito dall'intersezione tra due piani ortogonali:

$$\begin{aligned} a_1 x + b_1 y + c_1 z + e_1 = 0 \\ a_2 x + b_2 y + c_2 z + e_2 = 0 \end{aligned}$$

L'intersezione fra raggio e superficie è ottenuta determinando gli zeri della seguente superficie NURBS:

$$r(u, v) = \sum_i \sum_j D_{ij} R_{im,jn}(u, v) = 0$$

con $D_{ij} = (\bar{x}_{ij}, \bar{y}_{ij})^T$,

$$D_{ij} = \begin{bmatrix} w_{ij}(a_1 x_{ij} + b_1 y_{ij} + c_1 z_{ij} + e_1) \\ w_{ij}(a_2 x_{ij} + b_2 y_{ij} + c_2 z_{ij} + e_2) \end{bmatrix}$$

Il problema si riconduce quindi al calcolo degli zeri della superficie spline

2D al denominatore $d(u,v)$:

$$d(u,v) = \sum_i \sum_j D_{ij} N_{im}(u) N_{jm}(v) = 0$$

quindi a risolvere il sistema non lineare di due eq. in due incognite.

6.11.7 Tecniche di accelerazione

- **Suddivisione delle superfici**: permette di avere bounding box più piccole e riduce il numero di intersezioni per patch.
- **Suddivisione spaziale (octree)**: si considera il parallelepipedo che racchiude la scena e lo si suddivide in octree. Per ciascuno dei nodi viene costruita una lista delle superfici contenute in esso. Quindi si testa l'intersezione del raggio con le superfici contenute nei nodi attraversati dal raggio.

Caratterizziamo nel seguito un modello di illuminazione globale IDEALE mediante due formalismi, l'uno matematico (equazione di rendering) e l'altro descrittivo (percorsi in scena).

6.11.8 Percorsi nell'illuminazione globale di un semplice ambiente

Consideriamo l'ambiente chiuso in figura 6.44 con oggetti e sorgente luminosa. Ogni superficie obbedisce alla stessa legge fisica, può assorbire, riflettere o emettere luce. Seguiamo ora il percorso dei raggi dalla risorsa luminosa (L) all'osservatore (E) con interazioni tra superfici consecutive lungo il percorso di tipo DD (Diffuse-Diffuse), SD (specular-Diffuse) e SS (specular-specular).

Un generale percorso del raggio è nella forma **L(D|S)*E**, dove | significa **or** e * indica ripetizione. Un modello di illuminazione locale è del tipo **LD|S**. L'algoritmo HSR simula interazioni **LD|SE**.

L'illuminazione di un ambiente di sole superfici diffusive è simulato da RADIOSITY.

Mentre l'illuminazione di un ambiente di sole superfici speculari è simu-

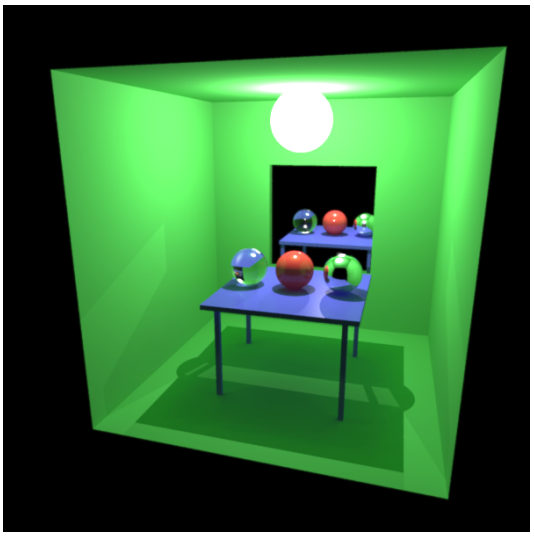


Figure 6.44: Scena resa con illuminazione globale

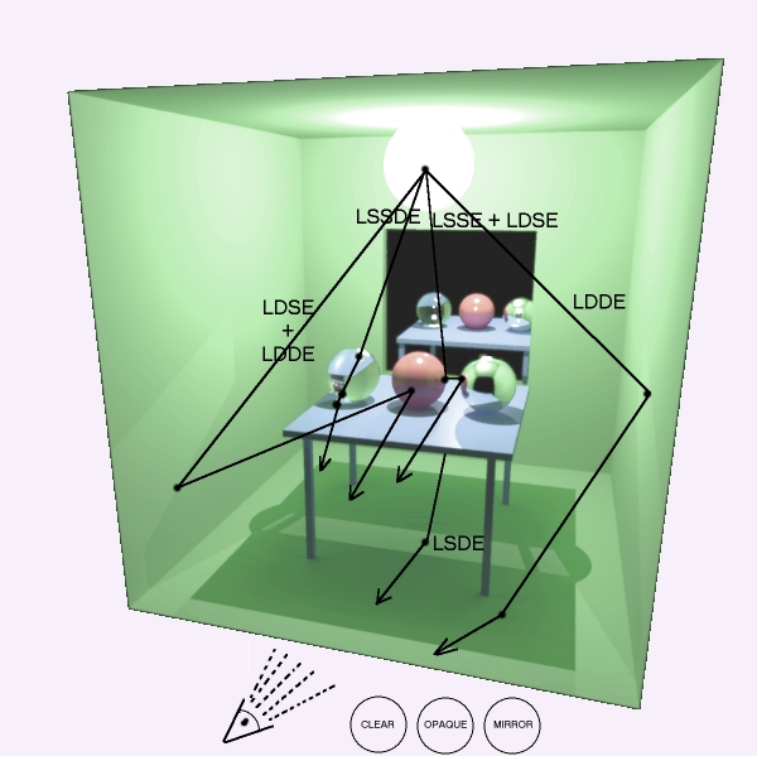


Figure 6.45: Percorsi dei raggi in una scena resa con illuminazione globale

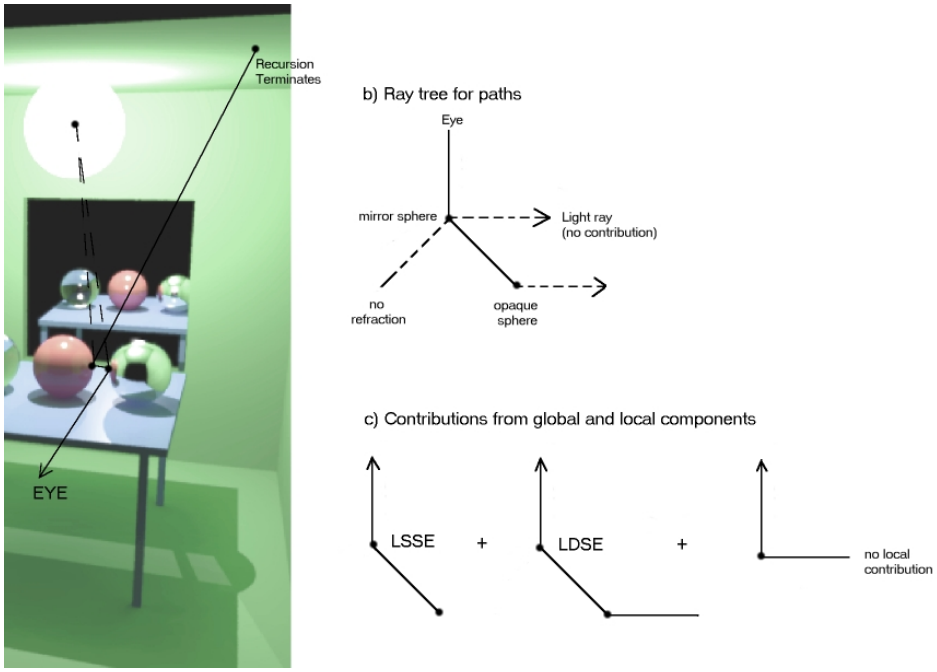


Figure 6.46: Ray tracing, percorso dei raggi

lato da RAY TRACING.

LDDE: viene vista la parte in ombra generata dal tavolo.

LDSE+LDDE: viene vista la parte in ombra della sfera. Poichè la sfera è lucida, la luce viene sia riflessa che diffusa verso l'osservatore.

LSSE+LDSE: la luce è riflessa dalla superficie a specchio verso l'osservatore che quindi vede la sfera opaca riflessa nella superficie a specchio.

LSSDE: questo percorso ha tre interazioni tra L ed E: il tavolo è una superficie che diffonde, la prima interazione speculare alla prima intersezione con la sfera e la seconda quando la luce esce dalla sfera.

Volendo caratterizzare il metodo di ray tracing di Whitted consideriamo la situazione in Fig.6.46 dove è riportato il percorso del raggio **LSSE+LDSE** che colpisce la sfera a specchio che quindi non ha contributo locale. In questo caso con *D* si intende la riflessione diffusa (diretta) dovuta al modello locale.

6.12 L'equazione di Rendering (Jim Kajiva 1986)

La luce è una forma di energia, perciò un'approccio basato sulla conservazione dell'energia sembra essere la simulazione più naturale all'illuminazione globale.

Consideriamo un'unica superficie: i raggi luminosi escono (emissione e/o riflessione) ed entrano in diverse direzioni.

Sia p punto sulla superficie, e p' punto esterno arbitrario raggiunto da un raggio che colpisce la superficie in p.

L'energia proveniente da tutti i possibili punti p' verso p, deve bilanciare l'energia emessa (se p è sorgente) o riflessa/diffusa da p.

Sia $i(p- > p')$ l'intensità luminosa che lascia p' e raggiunge il punto p. L'equazione di rendering proposta da Kajiva (J. Kajiya, The Rendering Equation, SIGGRAPH 1984, pp. 143-150) esprime questo bilanciamento:

$$i(p- > p') = \nu(p < - > p')(\epsilon(p- > p') + \int \rho(p''- > p- > p')i(p''- > p)dp'')$$

dove

- se p' è sorgente allora c'è un termine $\epsilon(p- > p')$ nella direzione di p';
- sia $i(p''- > p)$ il contributo intensità luminosa (radianza) entrante da ogni possibile punto p'' in p.
- $\rho(p''- > p- > p')$, è la Bidirectional Reflectance Distribution Function (BRDF) o funzione di riflessione che caratterizza il materiale in p. Descrive quanta della luce incidente sulla superficie in p dalla direzione p'' lascia la superficie in direzione p'.
- Il termine $\nu(p < - > p')$ è il termine geometrico che considera sia occlusioni (visibilità tra i punti) sia l'angolo tra le superfici di p'' e p, sia la distanza tra i punti. E' uguale a zero se c'è una superficie opaca tra p e p' quindi la luce da p' non raggiunge p. Altrimenti, si deve tener conto della distanza tra p e p' e quindi:

  $$\nu(p < - > p') = \frac{1}{r^2}$$

  dove r è la distanza tra i due punti. Inoltre si deve considerare la

relazione geometrica tra le superfici ovvero tra le normali ad esse nei punti p e p''.

L'energia luminosa che viaggia da un punto p a p' è uguale alla luce emessa da p a p' più l'integrale su tutti i punti di tutte le superfici della riflessione diffusa da un punto p'' su p e p' per l'intesità luminosa da p'' a p, il tutto attenuato da un fattore geometrico.

Questa equazione integrale presenta delle difficoltà di risoluzione analitica. La tecnica di radiosity cerca una buona approssimazione di questa soluzione.

6.13 Radiosity

La radiosity è un modello di illuminazione globale che oltre all'illuminazione diretta tiene conto del contributo di riflessione diffusa indiretto prodotto dagli altri oggetti in scena. Il termine radiosity significa velocità alla quale la luce lascia una superficie.

Ipotesi semplificativa: consideriamo un ambiente nel quale ogni superficie è perfettamente diffusiva, cioè riflette ugual luce in tutte le direzioni. Si vuole ottenere un'approssimazione della rendering equation nella quale tutte le superfici sono perfettamente diffusive: **la radiosity equation**.

Si considera una suddivisione della scena in n piccoli poligoni piani o patch, ciascuno dei quali perfettamente diffusivo (Fig.6.48).

La radiosity b_i del patch i è l'intensità luminosa che lascia il patch per unità d'area. Se a_i è l'area del patch allora l'intensità luminosa che lascia il patch è data da $b_i a_i$ (somma di energia emessa e riflessa):

$$b_i a_i = e_i a_i + \rho_i \sum_{j=0}^{n} f_{ji} b_j a_j;$$

la sommatoria considera i contributi di energia che arrivano al patch i da altri patch dell'ambiente; f_{ji} è detto **fattore forma** tra i patch i e j e rappresenta la frazione di energia che lascia il patch j e raggiunge il patch i e dipende dall'orientamento relativo tra i due patch. La riflettività del

Figure 6.47: Esempio di scena resa con radiosity

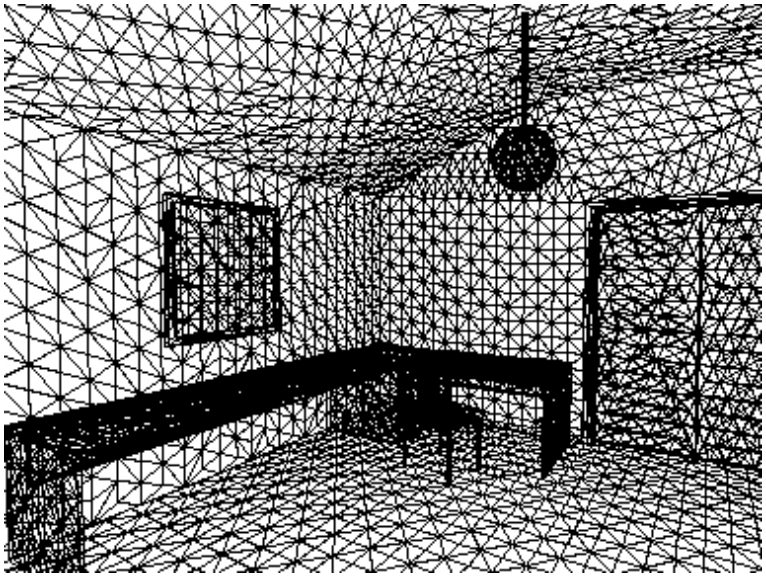


Figure 6.48: Suddivisione della geometria della scena

Figure 6.49: Risultato della resa di Fig.6.48 con radiosity.

patch i è invece ρ_i. Dividiamo per a_i per ottenere la **radiosity equation** del patch:

$$b_i = e_i + \rho_i \sum_{j=0}^{n} f_{ji} b_j;$$

Una volta calcolati i fattori forma, per n poligoni piani si ottiene un sistema di n equazioni lineari in n incognite di radiosità.

In forma matriciale:

$$(I - RF)b = e$$

con soluzione $b = [b_i]_{i=1}^{n}$ che rappresenta un valore di radiosity costante per ogni patch. Per attenuare l'effetto discreto della soluzione si può assegnare la soluzione ai vertici dei patch e poi si deve interpolare bilinearmente ogni poligono (con Gouraud shading) per ottenere i valori di luminosità all'interno di ogni poligono.

La radiosity equation è un caso speciale della rendering equation nella quale tutte le superfici sono considerate perfettamente diffusive.

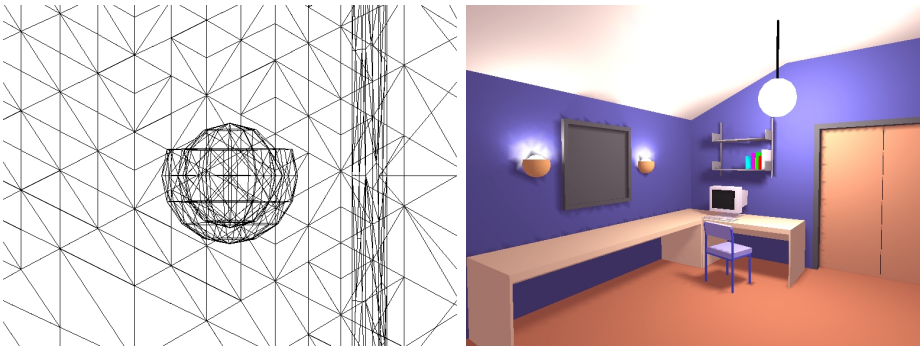


Figure 6.50: Influenza della suddivisione in radiosity.

Un esempio di scena resa con radiosity è illustrata in Fig.6.47.

L'algoritmo di **Radiosity Rendering** si basa sui seguenti passi:

- Passo 1) Suddividiamo la scena in mesh di patch (Fig.6.48);
- Passo 2) Calcolo dei form factor per ogni coppia di patch (procedimento piuttosto costoso);
- Passo 3) Risolviamo la radiosity equation; ora si può posizionare l'osservatore in scena e rendere con un comune renderer.(Fig.6.49).

Artefatti in radiosity

- Non si gestiscono superfici speculari
- Tempo computazionale eccessivo per scene di moderata complessità
- Artefatti in immagine dovuti a meshing: la suddivisione della scena non tiene conto della variazione della funzione radiosity, causando discontinuità.

Consideriamo ora l' influenza della suddivisione nella qualità dell'immagine finale, mediante un esempio illustrato in Fig.6.50.

Facendo un remeshing dell'area attorno alla lampada si fanno coincidere i contorni del patch muro con i contorni del patch lampada, i miglioramenti sono mostrati in Fig.6.51.

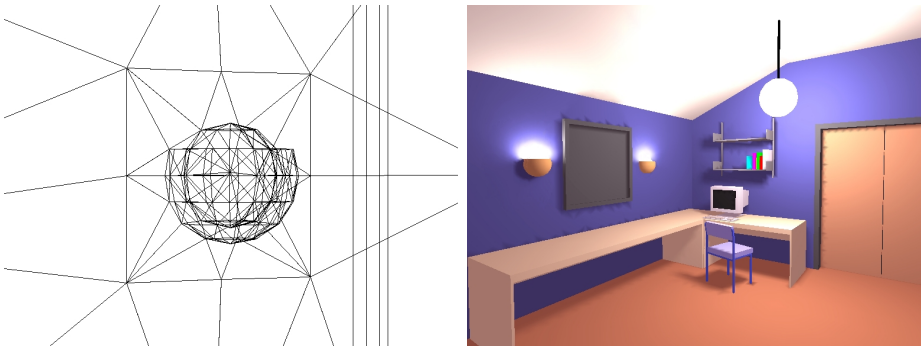


Figure 6.51: Remeshing.

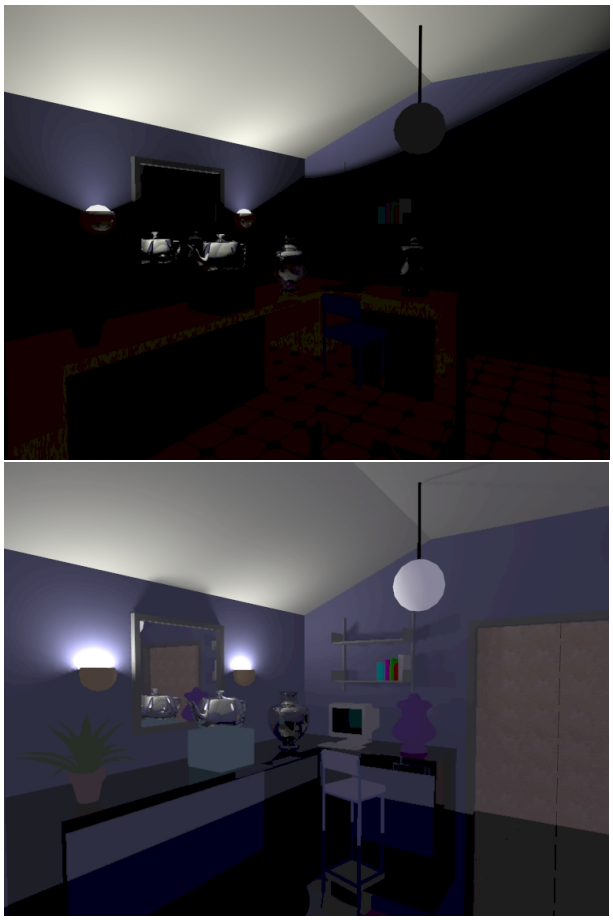


Figure 6.52: Ray tracing (sopra) vs Radiosity (sotto)

6.14 Ray tracing vs Radiosity

Infine un esempio per mettere a confronto i due metodi globali visti nella resa della scena senza considerare le luci principali. In Fig.6.52 sono illustrate le due scene dove però nella scena resa con radiosity la teapot è resa con ray tracing.

RT è view-dependent, Radiosity è view-independent.

RT considera solo LDE e LDS*E. La luce indiretta DD non viene considerata. Radiosity invece considera solo DD.

Chapter 7

Texture mapping

Per molti anni in computer graphics la sola entità che veniva manipolata dall'utente erano oggetti geometrici, come linee, poligoni, poliedri. Sebbene i sistemi grafici raster siano stati introdotti da oltre 60 anni, solo con l'avvento delle schede grafiche programmabili, i programmatori hanno avuto accesso diretto ai pixel del frame buffer. Texture mapping, antialiasing, alpha blending, composizione, sono solo alcune delle tecniche che sono ora disponibili per lavorare direttamente con i vari buffer del frame buffer.

Con l'anti-aliasing si vogliono eliminare gli artefatti dell'immagine prodotta nel frame buffer dovuti ad una rappresentazione discreta di una scena continua.

Con texturing invece si vogliono aggiungere dettagli realistici alla superficie degli oggetti. Là dove, per esempio, i dettagli geometrici sono difficili da modellare e computazionalmente onerosi da rendere le tecniche di texture mapping modificano i parametri di shading di una superficie, applicando una funzione (ad esempio un'immagine 2D) sulla superficie stessa.

7.1 Aliasing in computer graphics

La rasterizzazione provoca artefatti di visualizzazione. Questi sono dovuti al passaggio dalla rappresentazione continua a quella discreta (Fig.7.1).

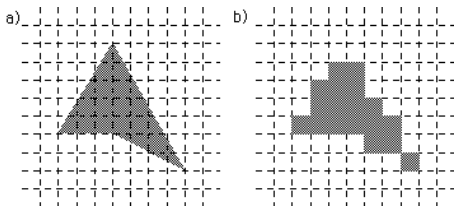


Figure 7.1: Aliasing: passaggio dal continuo (a) al discreto (b).

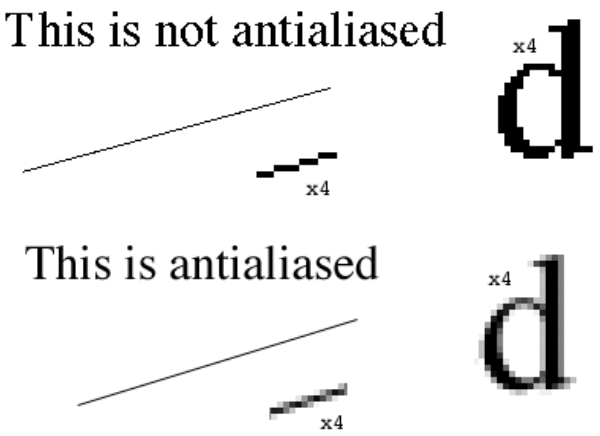


Figure 7.2: Aliasing e antialiasing

L'**Aliasing** è un fenomeno che accade quando ricostruiamo un segnale continuo da un insieme di valori discreti del segnale stesso; si ha una perdita di informazioni (l'occhio umano osserva una scena da una immagine digitale).
Si può aumentare la risoluzione, e questo sicuramente migliora la qualità dell'immagine, a costo di un maggior utilizzo di memoria. L'effetto aliasing è comunque, anche se attenuato, sempre presente.

Si parla di

- Aliasing spaziale: quando si considerano gli artefatti nella singola immagine;
- Aliasing temporale: quando gli artefatti si osservano in una sequenza di immagini (animazioni)- numero di frame al secondo. Quando, ad esempio, oggetti piccoli sono rappresentati in alcuni frame e in altri no a seconda che passino per il centro di un pixel.

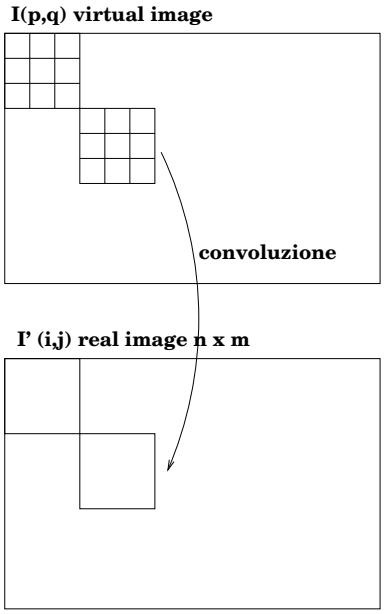


Figure 7.3: Supersampling.

Le tecniche di **anti-aliasing** hanno lo scopo di migliorare la qualità dell'immagine attenuando artefatti dovuti ad aliasing. (Fig.7.2)

L'idea è quella di assegnare una differente intensità di colore ai pixel che compongono l'immagine in funzione della loro distanza dalla linea ideale: in questo modo l'occhio umano vede un'immagine qualitativamente migliore di quella che vedrebbe a causa della limitata risoluzione. Queste tecniche si possono distinguere in:

- supersampling
- edge-detection + supersampling

7.1.1 Anti-aliasing: supersampling

La tecnica di supersampling o oversampling calcola un'immagine virtuale ad una risoluzione maggiore della reale risoluzione schermo, quindi filtra il risultato per ottenere un'immagine alla risoluzione originaria. Il vantaggio primario è quello di riuscire ad implementare l'algoritmo mediante l'ausilio dello Z-buffer.

Dall'immagine ad alta risoluzione si passa a quella a minor risoluzione tramite un filtro passa-basso, il processo è illustrato in Fig.7.3. La convoluzione di un filtro passa-basso h di dimensione k con l'immagine virtuale si effettua nel modo seguente:

$$I'(i,j) = \sum_{p=S_i-k}^{S_i+k} \sum_{q=S_j-k}^{S_j+k} I(p,q)h(S_i - p, S_j - q)$$

Esempio di un filtro 3×3 che rappresenta la discretizzazione di una funzione Gaussiana bivariata:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Operativamente si applica un filtro passa basso (es. Gaussiano) ad un'immagine ad alta risoluzione, quindi si ricampiona il risultato a bassa (originale) risoluzione.

In alternativa si può utilizzare l'**accumulation buffer**.

E' un buffer della stessa dimensione dell'immagine che contiene la somma di N immagini della stessa scena ognuna ottenuta da un punto di vista spostato di una N-esima frazione di pixel in x, y o entrambi. L'immagine viene poi mediata (es. divisa per N) e inviata al display. L'accumulation buffer è gestito nelle API OpenGL.

Antialiasing: Esempio

Il seguente esempio tratto dal testo *Alan Watt, 3D Computer Graphics, 3rd Edition, Addison-Wesley, 2000,* considera la scena in Fig.7.4 senza antialiasing. Mentre in Fig.7.5 è mostrato il risultato dell'applicazione della tecnica supersampling con un'amplificazione $2x$ (sinistra) e $3x$ (destra) rispetto alla risoluzione schermo, ovvero ogni pixel è una matrice 2×2 pixel o 3×3 pixel.

Infine in Fig.7.6 viene mostrato il risultato dell'anti-aliasing effettuato con ray tracing. Invece di considerare l'intera immagine, si fa anti-aliasing solo in quelle aree che lo necessitano (in funzione del gradiente locale dell'immagine). Si generano più raggi nei sotto-pixel coinvolti.

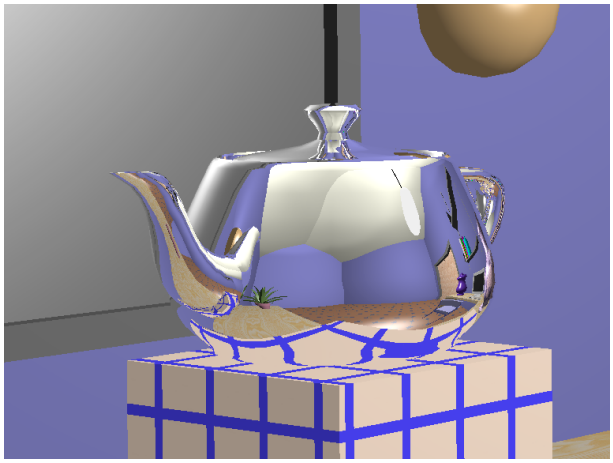


Figure 7.4: Scena senza eliminazione di aliasing

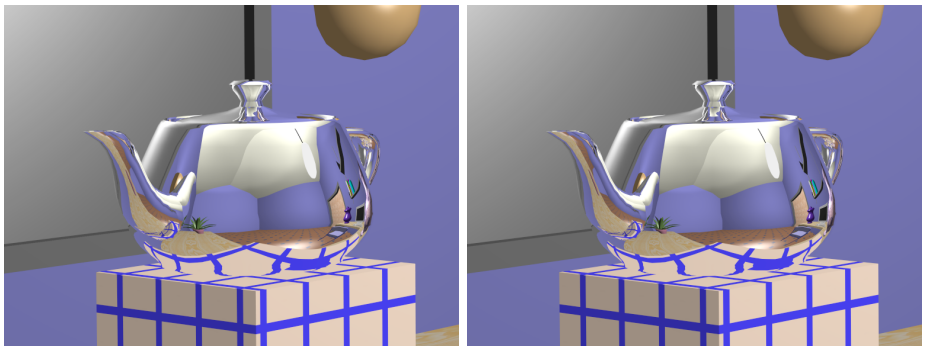


Figure 7.5: Scena con anti-aliasing: $2x$ (sinistra) e $3x$ (destra)

7.2 Texture mapping

In computer graphics, il texturing è un processo che modifica l'apparenza di una superficie in ogni punto, usando immagini, o funzioni.

L'impiego di texture maps è di particolare importanza nell'ambito della Computer Graphics in tempo reale, nella quale il processo di resa delle scene tridimensionali deve rispettare forti vincoli temporali, e le immagini devono essere elaborate rapidamente. Si tende dunque a ridurre il più possibile la complessità geometrica della scena, allo scopo di velocizzare i calcoli necessari al rendering, e viene affidato alle texture maps il compito di mostrare i dettagli e le piccole imperfezioni presenti sulla

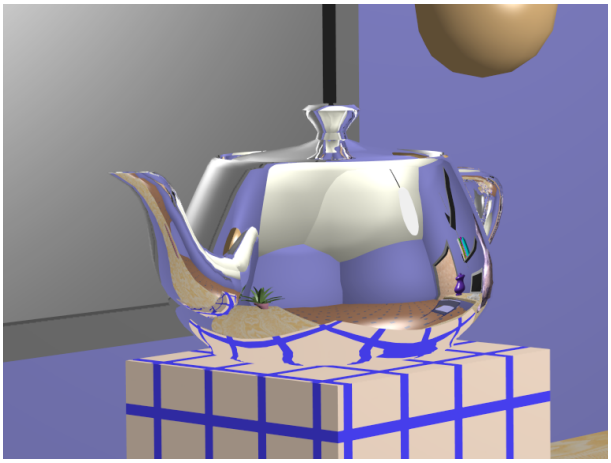


Figure 7.6: Scena con anti-aliasing, resa con ray tracing.

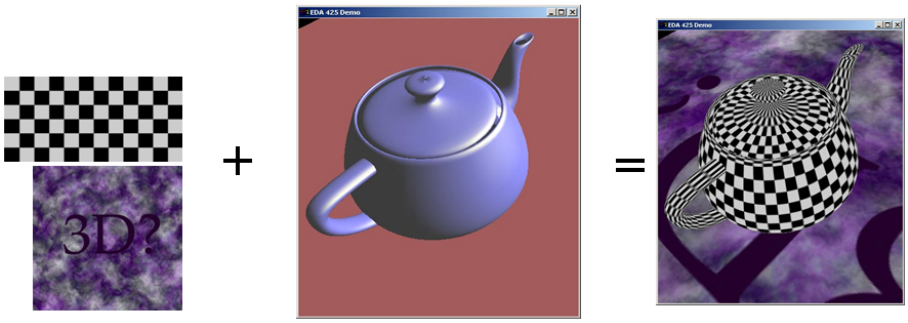


Figure 7.7: Esempio di texture mapping: modello geometrico (centro), texture (sinistra), modello con texture (destra)

superficie degli oggetti. Un semplice esempio di utilizzo di texture è illustrato in Fig.7.7. Si arricchisce un modello geometricamente semplice di dettagli mediante un buon modello visivo con conseguente miglioramento di performance, risparmio di memoria e tempi di modellazione. Il segreto è quindi di riuscire ad aggiungere dettagli visivi all'oggetto senza aggiungere dettagli geometrici. In Fig.7.8 è mostrato un esempio di modelling , modelling+shading e modelling+shading+texture.

La tecnica del texture mapping si è evoluta nel corso degli anni, sviluppando nuove modalità, sempre più realistiche di rappresentare i materiali, e trovando applicazioni diverse da quelle per cui era stata concepita.

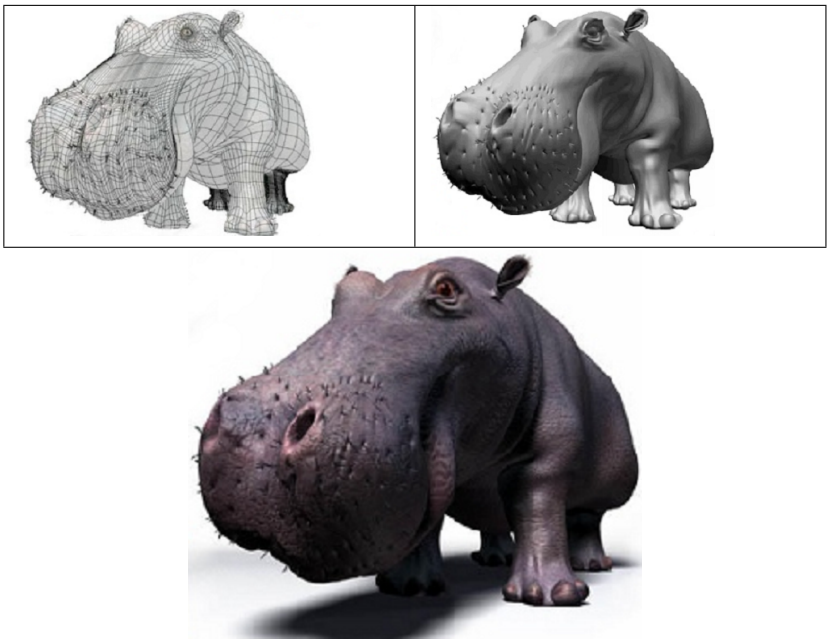


Figure 7.8: Modellazione, modellazione+shading, modellazione + shading + texture (Courtesy of Jeremy Birn)

Un esempio fra tutti è quello delle tecniche di light mapping ed environment mapping, usate per colmare i limiti dei modelli di illuminazione locali utilizzati per la grafica in tempo reale. Questi modelli, semplificati allo scopo di poter essere implementati efficientemente in ambito real-time, non sono in grado di rendere effetti di illuminazione quali le rifrazioni, le superfici riflettenti, o l'effetto delle luci indirette riflesse dall'ambiente su un oggetto. Si rende la scena alla quale vengono applicate tecniche globali di illuminazione, da questa si catturano le texture necessarie a riprodurre in seguito l'effetto di illuminazione, che vengono poi applicate agli oggetti durante il rendering con tecniche locali di illuminazione ad un basso costo computazionale. Questo permette di simulare modelli più complessi di illuminazione.

Le immagini texture possono essere di vario tipo, da immagini fotografiche digitalizzate a immagini generate in modo procedurale. Inoltre le texture possono essere 1D, 2D,3D o 4D.

Gli algoritmi di texturing vanno ad alterare il colore dei pixel definito

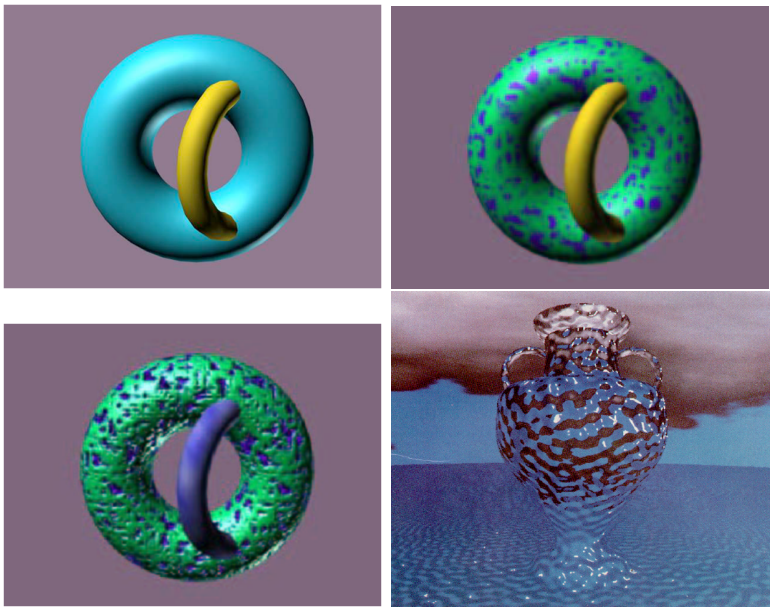


Figure 7.9: Esempi di oggetto reso con normale shading, con texture mapping, bump mapping e trasparenze (solid texture).

nella fase di shading. Le proprietà dell'oggetto che possono essere modificate dal texture mapping, andando a modificare i relativi coefficienti nel modello di illuminazione locale, sono le seguenti:

- Colore (texture mapping): viene modificato il colore (diffuso o speculare) nel modello di illuminazione di Phong con il corrispondente colore dalla texture map;
- Colore riflesso (environment mapping): simula la riflessione di un oggetto speculare che riflette l'ambiente circostante; permette di creare immagini che apparentemente sembrano generate da un ray tracer.
- Perturbazione della normale (bump mapping): perturbazione della normale alla superficie in funzione di un valore corrispondente in una funzione 2D, per simulare un effetto di superficie in rilievo.
- Trasparenza: controllare l'opacità di un oggetto trasparente. Es: bicchiere di vetro inciso all'acquaforte.

Si veda Fig.7.9 per un esempio di applicazione.

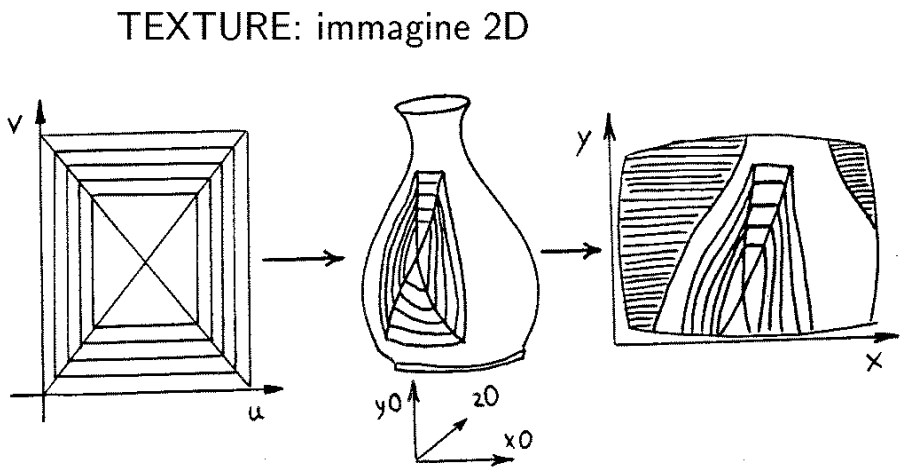


Figure 7.10: Texture mapping: spazio texture (sinistra); spazio oggetto (centro); spazio schermo (destra).

7.3 2D Image Texture Mapping

L'implementazione della tecnica del texture mapping richiede di definire una corrispondenza tra diversi spazi, o sistemi di coordinate, coinvolti durante l'operazione di rendering. Lo **spazio texture**(2D) rappresenta il dominio della texture map e ha due dimensioni che rappresentano righe e colonne dell'immagine. Normalmente questo spazio viene schematizzato come un array bidimensionale di punti, detti texel. Il secondo spazio preso in considerazione è quello nel quale è definita la geometria degli oggetti presenti nella scena: lo **spazio oggetto**, a tre dimensioni. L'ultimo è lo **spazio schermo** (2D), nel quale si visualizza la viewport, che contiene i pixel dell'immagine ottenuta attraverso il processo di rendering della scena.

Vediamo come si può caratterizzare tale mapping.

Data un'immagine, ovvero una funzione 2D da $[0,1]^2$ (coordinate texture) nello spazio dei colori RGB:

$$T(u,v) \rightarrow (r,g,b)$$

Per ogni primitiva geometrica, una parametrizzazione è una funzione S

che stabilisce la corrispondenza tra punti di un dominio parametrico 2D a punti sulla superficie, si definisce un mapping inverso S^{-1} che fa corrispondere a punti sulla superficie punti in un dominio 2D, in coordinate texture:

$$S^{-1}(x, y, z) = (u, v)$$

Nel geometry stage della rendering pipeline, la fase di proiezione P della scena proietta ogni vertice (x, y, z) nel corrispondente pixel (x_s, y_s) nella viewport:

$$(x_s, y_s) = P(x, y, z).$$

Il pixel (x_s, y_s) corrispondente al vertice (x, y, z) sull'oggetto in scena, avrà il colore:

$$(r, g, b) = T(S^{-1}(P^{-1}(x_s, y_s))).$$

Il risultato del texture mapping può essere visto come una operazione di image warping: l'immagine contenuta nella texture map viene trasformata per essere riprodotta nello spazio schermo. Tuttavia questa tecnica viene comunemente sintetizzata nella sequenza delle due fasi:

1. Specificare il mapping di una texture su di una superficie (object space) (**fase di PARAMETRIZZAZIONE**), per il passaggio da spazio texture a spazio oggetto,
2. Proiezione dell'oggetto con texture sul piano immagine (screen space), (**fase di PROIEZIONE**) per il passaggio da spazio oggetto a spazio schermo.

Si veda Fig.7.10.

Il primo passo, che prende il nome di parametrizzazione, stabilisce in che modo la texture deve essere applicata all'oggetto; si tratta dunque di definire una funzione che associ ad ogni punto sulla superficie un texel, ovvero una coppia di coordinate nello spazio texture. Identificare una parametrizzazione che dia risultati soddisfacenti è un problema complesso che deve essere affrontato tenendo conto di possibili effetti indesiderati. Ad esempio l'immagine posizionata sulla superficie potrebbe apparire deformata, o troppo schiacciata lungo una certa direzione. Oppure potrebbe accadere che determinate zone significative dell'immagine, come ad esempio il naso in una texture map da applicare su una faccia umana, vengano a trovarsi nella posizione sbagliata dell'oggetto model-

lato.

Il secondo passaggio, quello da spazio oggetto a spazio schermo, avviene mediante l'operazione di proiezione nello stadio geometry della pipeline, di cui si è già parlato nel capitolo sulla rendering pipeline, che si occupa però solo dei vertici di ogni primitiva geometrica, il colore da associare ai pixel interni alla primitiva viene calcolato in fase di rasterizzazione.

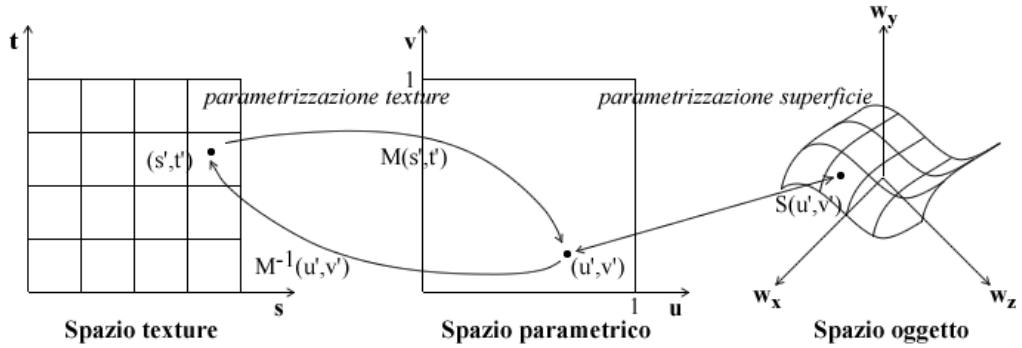


Figure 7.11: Un esempio di texture mapping su superficie parametrica. La funzione M, la cui inversa è M^{-1}, è usata per parametrizzare la texture. La funzione S definisce la parametrizzazione dei punti che appartengono alla superficie.

7.3.1 Parametrizzazione

Se l'oggetto geometrico è definito mediante superfici parametriche come le superfici spline, la parametrizzazione è definita in modo naturale dal dominio parametrico piano $D = U \times V$ della superficie

$$S(u,v) = (\ x(u,v),\ \ y(u,v),\ \ z(u,v)\ )^T, \quad \forall (u,v) \in D.$$

Possiamo quindi applicare una texture map bidimensionale $T(s,t)$ nello spazio texture al dominio parametrico della superficie mediante una trasformazione affine $(u,v) = M(s,t)$ da cui è possibile trovare analiticamente il mapping inverso (M^{-1}). Si veda Fig.7.11.

Per un oggetto definito invece da mesh poligonali il procedimento è un po' più complesso, in quanto non ha associata alcuna parametrizzazione

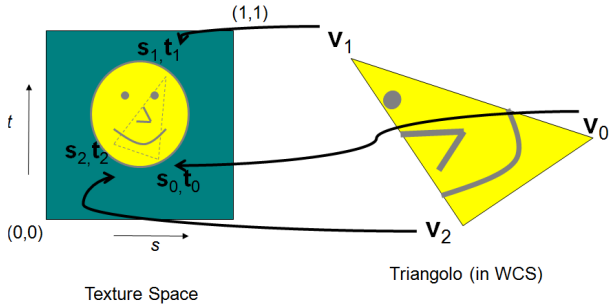


Figure 7.12: Fase di parametrizzazione di un singolo poligono: associare coordinate texture ai vertici del poligono.

'naturale'. In tal caso si può procedere specificando esplicitamente la corrispondenza fra vertici di ogni poligono della mesh e coordinate texture, come illustrato in Fig. 7.12 per un singolo poligono. In alternativa esistono metodi matematici per la parametrizzazione di mesh che procedono a stabilire una corrispondenza tra ogni vertice della mesh e punti sulla texture mediante la risoluzione di un sistema lineare.

Nella fase di rasterizzazione, come vedremo, verranno calcolati i colori texture per un generico punto interno al poligono di una mesh, attraverso un'interpolazione delle coordinate texture dei vertici del poligono, così come avviene per l'informazione colore.

7.4 Texture mapping nella pipeline grafica

Nella pipeline grafica entrambi gli stadi (geometry e rasterizer) concorrono alla formazione di texture nelle immagini.

Il primo stadio, Geometry Stage, applica ai vertici dei poligoni la proiezione dallo spazio oggetto allo spazio schermo.

Lo stadio successivo, rasterization stage, ha invece il compito di rappresentare/colorare su schermo ogni pixel di ogni poligono proiettato.

La parametrizzazione viene sfruttata in questo secondo stadio, tramite un inverse texture mapping, il quale stabilisce per ciascun pixel in screen space le coordinate texture della sua pre-immagine (su texture image).

Poichè non esiste una funzione di mapping inverso per ogni pixel, ma solo per quelli corrispondenti ai vertici di un poligono, le coordinate texture dei punti interni al poligono sono approssimate tramite un'interpolazione.

Tale interpolazione si dovrebbe effettuare mediante un' interpolazione lineare delle coordinate texture dei vertici nello spazio oggetto. Invece, poichè il texture mapping è integrato nell'algoritmo scan-conversion per il disegno nel frame buffer dei poligoni, il calcolo delle coordinate texture di un punto interno ad una primitiva geometrica avviene nello spazio schermo.

L' interpolazione bilineare delle coordinate texture nello spazio schermo coinvolge tre interpolazioni lineari, illustrate in Fig. 7.13.

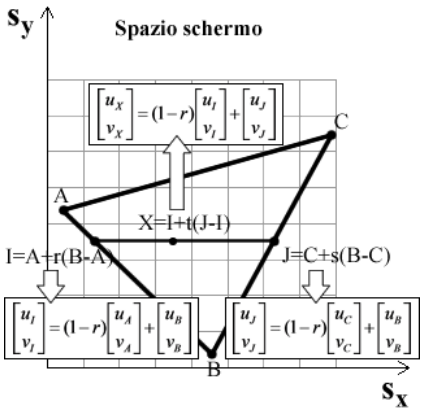


Figure 7.13: Interpolazione bilineare delle coordinate texture dei pixel di un poligono triangolare.

Per disegnare un poligono, la scanconversion traccia, una alla volta, le linee orizzontali di pixel che lo compongono; per ciascuna linea vengono interpolate le coordinate texture degli estremi I e J in funzione di quelle dei vertci del triangolo A, B e C e dei parametri r e s che identificano la linea. Vengono poi calcolati i colori dei pixel che compongono la linea, interpolando per ciascun pixel le coordinate texture degli estremi I e J a seconda del valore del parametro t, che dipende dalla posizione del pixel sulla linea di scansione. Lo stesso procedimento viene utilizzato per interpolare altri parametri definiti a livello di vertici, come, ad esempio, il colore, nell'algoritmo di shading di Gouraud, e la profondità, usata nel test di visibilità.

Questi calcoli, eseguiti sulla base delle coordinate dei punti nello spazio schermo, non danno esattamente lo stesso risultato che si otterrebbe interpolando le loro coordinate in spazio oggetto. Infatti la proiezione, eseguita durante il Geometry Stage, non è una trasformazione lineare; dunque una variazione costante dei parametri in spazio schermo non corrisponde ad una variazione costante nello spazio oggetto. L'effetto visivo di questo errore di approssimazione è uno schiacciamento delle texture, quando vengono applicate su un poligono che abbia alcuni vertici posti a diverse distanze dall'osservatore; tanto maggiore è la differenza di profondità tra i vertici del poligono tanto più l'immagine verrà distorta.

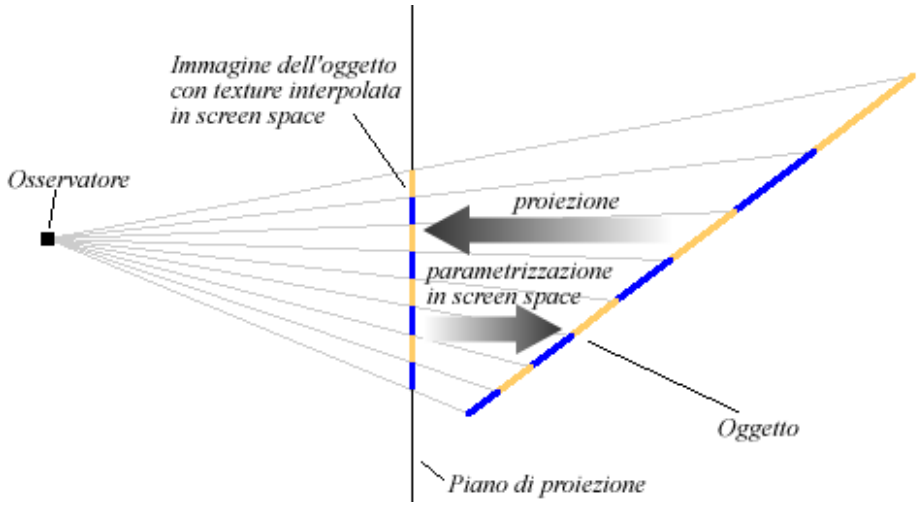


Figure 7.14: Effetto dell'interpolazione lineare delle coordinate texture nello spazio schermo.

Per ovviare a questo problema, alcune implementazioni della pipeline grafica sono dotate di un meccanismo di correzione prospettica, con cui si ricava il valore corretto del parametro di interpolazione nello spazio oggetto, a partire dalla sua proiezione sullo spazio schermo.

7.4.1 Correzione Prospettica

Vediamo di ricavare la causa della deformazione per trovare un'opportuna correzione prospettica per pixel. Consideriamo un'interpolazione dei valori di texture associati ai vertici mediante un'interpolazione lineare.

Siano $v_1 = (x_1, z_1)$ e $v_2 = (x_2, z_2)$ gli estremi del segmento nello spazio oggetto in Fig.7.14, e p_1,p_2 le loro rispettive correzioni nello spazio schermo.

Facendo un'interpolazione lineare nello **spazio schermo**:

$$p(t) = p_1 + t(p_2 - p_1) = \frac{x_1}{z_1} + t(\frac{x_2}{z_2} - \frac{x_1}{z_1})$$

anzichè un'interpolazione lineare nello **spazio oggetto**:

$$p(s) = \begin{bmatrix} x_1 \\ z_1 \end{bmatrix} + s\left(\begin{bmatrix} x_2 \\ z_2 \end{bmatrix} - \begin{bmatrix} x_1 \\ z_1 \end{bmatrix}\right),$$

si ottengono risultati leggermente distorti. Come dobbiamo fare per ottenere un'interpolazione corretta in coord. schermo? Poichè $p(t)$ è la proiezione di $p(s)$, poniamo

$$\frac{x_1}{z_1} + t(\frac{x_2}{z_2} - \frac{x_1}{z_1}) = \frac{x_1 + s(x_2 - x_1)}{z_1 + s(z_2 - z_1)}$$

e ricaviamo il parametro s:

$$s = \frac{tz_1}{z_2 + t(z_1 - z_2)}$$

In realtà in fase di scan conversion non abbiamo il valore z, ma nello Z-buffer è memorizzato $w = 1/z$, quindi

$$s = \frac{t\frac{1}{w_1}}{\frac{1}{w_2} + t(\frac{1}{w_1} - \frac{1}{w_2})}$$

Ora possiamo utilizzare questo valore parametrico s al posto di t per interpolare quantità arbitrarie, per esempio le coordinate texture (u, v) all'interno del triangolo 3D a partire dalle coordinate texture associate ai vertici (u_1, v_1), (u_2, v_2).

$$u = u_1 + s(u_2 - u_1), \qquad v = v_1 + s(v_2 - v_1)$$

$$u = \frac{u_1 w_1 + t(u_2 w_2 - u_1 w_1)}{w_1 + t(w_2 - w_1)} \quad v = \frac{v_1 w_1 + t(v_2 w_2 - v_1 w_1)}{w_1 + t(w_2 - w_1)}$$

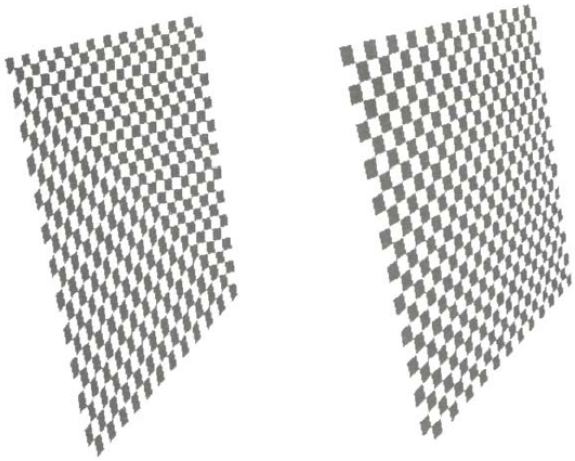


Figure 7.15: Interpolazione lineare (sinistra) ed interpolazione corretta con proiezione prospettica (destra).

Si osservi in Fig.7.4.1 un esempio di applicazione della corrrezione prospettica sulla resa di due triangoli complanari.

7.5 Inverse Mapping mediante Two Step Texture Mapping

Il metodo di inverse mapping descritto nel paragrafo precedente può essere usato solo nel caso in cui sia definita una parametrizzazione, ovvero una corrispondenza vertici-coordinate texture per ogni oggetto della scena. Un'alternativa, per le situazioni in cui non sia possibile stabilire una parametrizzazione, consiste nel servirsi di superficie intermedia virtuale sulla quale sia facile applicare la texture, e proiettare poi la texture da superficie intermedia sull'oggetto. La superficie intermedia viene scelta in modo da assomigliare il più possibile alla forma dell'oggetto e affinchè il metodo risulti conveniente la superficie intermedia deve essere di facile parametrizzazione.

La tecnica **Two Step texture Mapping**, proposta da Bier e Sloan (1986), si basa sui due seguenti passi:

- **PASSO 1: (S Mapping)** la texture è mappata su una superficie sem-

plice intermedia:

$$T(u,v) \rightarrow T'(x_i, y_i, z_i)$$

- **PASSO 2: (O Mapping)** proiezione della texture dalla superficie intermedia sull'oggetto:

$$T'(x_i, y_i, z_i) \rightarrow O(x_w, y_w, z_w)$$

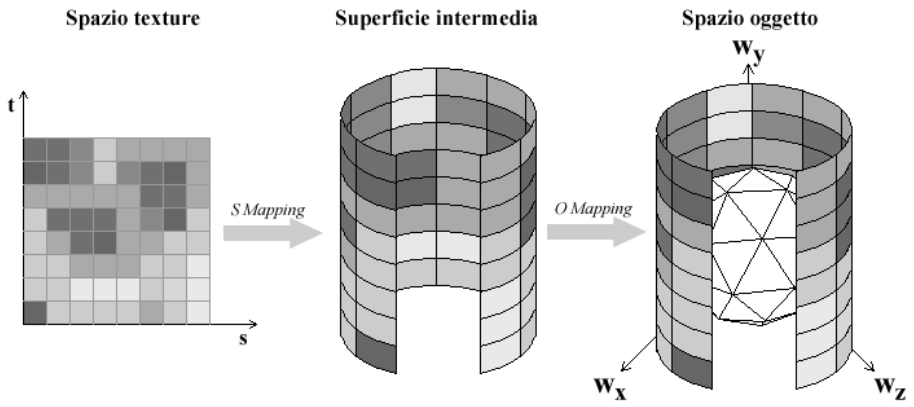


Figure 7.16: I due passi del texture mapping con superficie intermedia.

Nella Fig. 7.16 è illustrato il procedimento, che si sviluppa in due passi, durante ciascuno dei quali viene stabilita una corrispondenza tra punti in sistemi di coordinate diverse. Nel primo passo avviene il mapping tra lo spazio texture e la superficie intermedia (S mapping). Il secondo passo, O mapping, consiste nel definire la corrispondenza tra i punti sulla superficie intermedia e quelli sulla superficie dell'oggetto.

Come **superfici intermedie** possiamo considerare un piano, la parte ricurva di un cilindro, le facce di un cubo, o la superficie di una sfera. Quale superficie utilizzare dipende fortemente dal tipo di oggetto e di texture che dobbiamo rendere, si veda Fig.7.18.

La superficie più semplice da parametrizzare è il piano; in questo caso si può ottenere un S-mapping attraverso una trasformazione affine, composta da traslazioni, rotazioni e scale, che posizionano l'immagine bidimensionale della texture map su una regione del piano.

Quando invece si vuole avvolgere la texture map attorno all'oggetto si usa una superficie chiusa, come ad esempio un cubo, sulle cui facce ven-

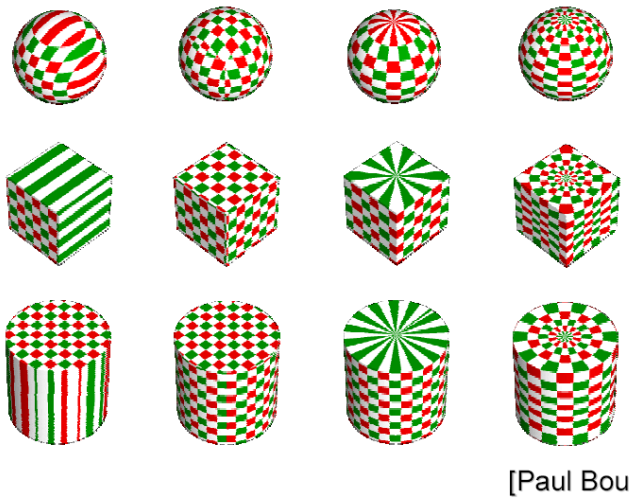


Figure 7.17: Utilizzo di diverse superfici intermedie con una stessa texture. Per colonne: piano, cubo, cilindro, sfera.

gono mappate sei porzioni della texture map. Per solidi di rotazione è conveniente usare delle superfici come il cilindro o la sfera, che garantiscono una distorsione più naturale della texture map sulla superficie dell'oggetto.

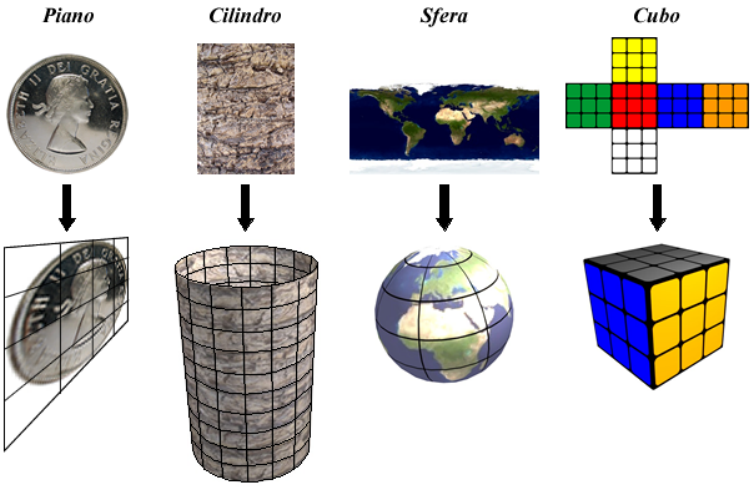


Figure 7.18: Le quattro superfici intermedie usate comunemente per il texture mapping.

Se, per esempio, consideriamo come superficie intermedia un cilindro, allora la parametrizzazione del cilindro di raggio r ed altezza h, ci fornirà un naturale S-mapping:

$$S_{cilindro}(u,v) = (r\cos(2\pi u), r\sin(2\pi u), \frac{v}{h})$$

dove $u, v \in [0,1]^2$. Per posizionare la texture di coordinate (s,t) sul cilindro, si pone $s = u$ e $t = v$.

In Fig.7.17 sono mostrati diversi esempi di mapping della stessa texture utilizzando diverse superfici intermedie.

Per quanto riguarda il secondo step, quello di O mapping, sono previste quattro modalità (si veda Fig. 7.19) per stabilire il punto della superficie intermedia da proiettare su una data posizione della superficie dell'oggetto:

1. L'intersezione tra la normale alla superficie dell'oggetto e la superficie intermedia.
2. L'intersezione della retta per il punto che passa per il baricentro dell'oggetto, e la superficie intermedia.
3. L'intersezione della normale alla superficie intermedia e la superficie dell'oggetto.
4. L'intersezione del raggio che va dall'osservatore al punto in questione riflesso sulla superficie dell'oggetto e la superficie intermedia.

La terza modalità è usata in particolare quando la superficie intermedia è un piano; il risultato ottenuto è paragonabile all'effetto di proiettare sulla superficie dell'oggetto l'immagine della texture map usando un proiettore che trasmetta un fascio di raggi perpendicolari al piano. Usando la quarta modalità, l'oggetto sembra riflettere sulla propria superficie l'immagine applicata alla superficie intermedia, come se fosse uno specchio. Questo principio è alla base della tecnica di Environment Mapping, con la quale si può dare l'impressione di una superficie perfettamente speculare che riflette l'ambiente circostante, senza dover ricorrere a modelli di illuminazione globale che calcolino il contributo delle luci riflesse, indirettamente dall'ambiente.

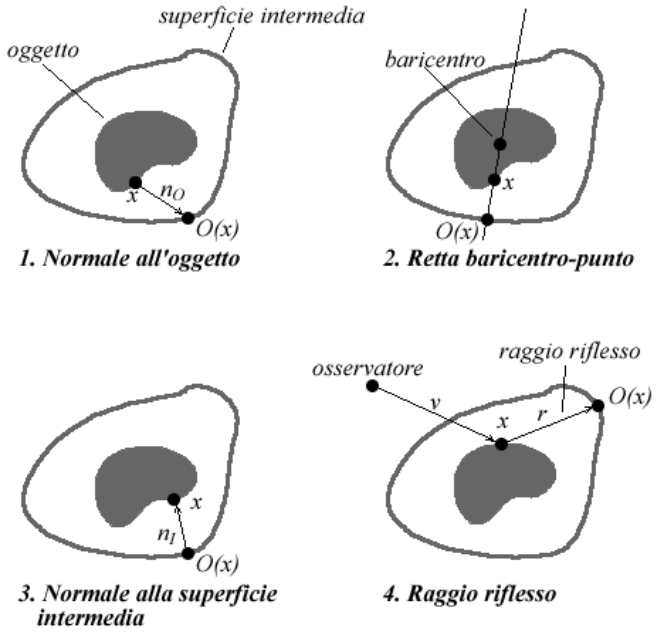


Figure 7.19: Le modalità di O mapping.

7.6 Texture Filtering

La tecnica di inverse texture mapping fa uso della funzione inversa della parametrizzazione per trovare un punto pre-immagine su texture di un pixel nello spazio schermo. Nella pratica però, poichè un pixel non è puntiforme, bensì di forma quadrata con lato non nullo, la sua pre-immagine è un' area quadrangolare curvilinea (come in figura 7.20) sullo spazio texture.

Se i texel che compongono la pre-immagine sono di colore diverso, si pone il problema di scegliere quale colore assegnare al pixel.

La tecnica più semplice detta **point sampling** consiste nel campionare il colore di un solo texel della pre-immagine, ad esempio la pre-immagine di un punto immaginario posizionato al centro del pixel. In tal caso, si verifica il fenomeno dell'aliasing, che porta ad una proiezione poco realistica della texture nello spazio schermo (effetto illustrato in Fig.7.21). L'aliasing si presenta quando vengono campionati dalla texture dei colori poco rappresentativi della pre-immagine, ed è tanto più evidente quanto

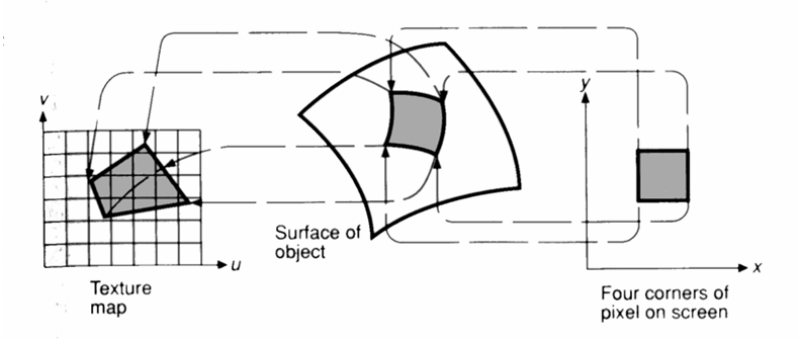


Figure 7.20: A sinistra è mostrata la pre-immagine su texture del pixel evidenziato nello spazio schermo (destra).

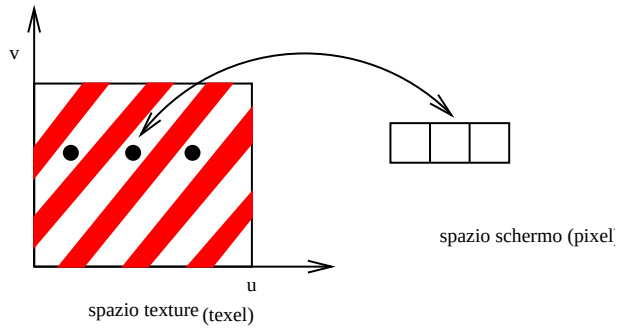


Figure 7.21: Artefatti in 2D texturing: point sampling.

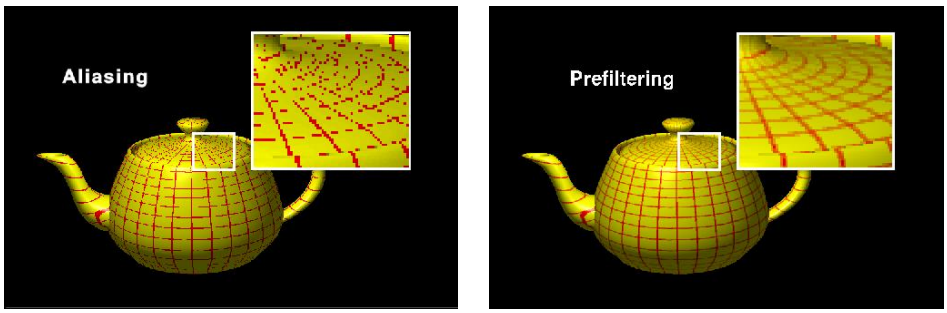


Figure 7.22: Risultati di un texture mapping con point sampling (sinistra) ed area filtering (destra)

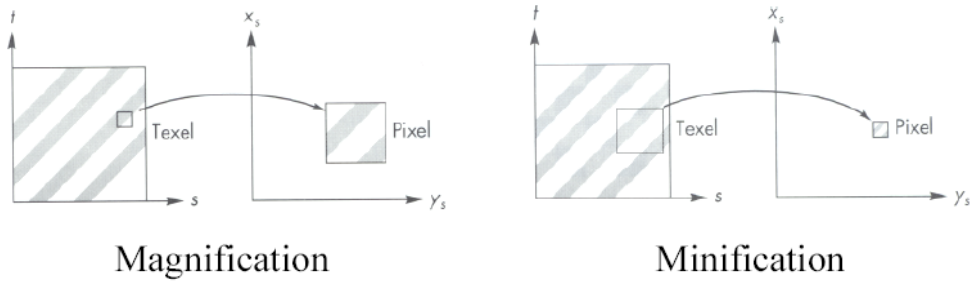


Figure 7.23: Magnification e minification filtering.

più il periodo del motivo presente nell'immagine texture proiettata nello spazio schermo si avvicina alla distanza tra un pixel e l'altro.

Si deve procedere considerando invece del point sampling un' **area sampling** che consiste nel filtrare l'area coinvolta nella pre-immagine per calcolare il valore (colore) pixel.

In Fig.7.22 è illustrato un esempio di point sampling versus area sampling texture mapping.

Una procedura di anti-aliasing ideale dovrebbe:

- trovare la pre-immagine di un pixel in texture space $T(u, v)$ (area quadrilatera curvilinea)
- fare una somma pesata dei valori che cadono dentro l'area della pre-immagine;
- ottenere quindi una singola intensità di texture per quel pixel.

Eseguire questo filtraggio durante il rendering risulta troppo costoso, si usano quindi delle tecniche dette di **pre-filtering** che non gravano sui tempi di resa.

Si gestiscono due casi nella pratica: il caso in cui ad un singolo pixel corrisponde nella pre-immagine una regione più estesa (più texel), e il caso opposto quando a pochi texel nella pre-immagine corrisponde un'area pixel molto più grande.

Questi due casi, illustrati in Fig.7.23, prendono il nome rispettivamente di:

MAGNIFICATION

Le texture appaiono ingrandite sullo schermo, i pixel hanno pre-

immagini di area minore di un texel.

MINIFICATION

Le texture appaioni ridotte in dimensioni nello spazio schermo e conseguentemente ad un pixel corrisponde una grande pre-immagine nel texture space.

Le tecniche adottate per gestire minification e magnification assumono che la pre-immagine di un pixel possa sempre essere approssimata con un quadrato nello spazio texture, di area più o meno grande, a seconda del rapporto tra l'area della texture e l'area della sua proiezione su schermo. Dunque si può pensare di calcolare preventivamente l'area di tutte le possibili pre-immagini quadrate della texture map, e poi memorizzare questi valori in una texture map da usare durante il rendering. Per **Magnification** occorre interpolare in texture space, si possono a tale scopo utilizzare le tecniche nearest neighbor (scelta del colore del texel più vicino) o interpolazione lineare (interpola i valori dei texel più vicini (4 in 2D, 8 per 3D)). La più classica soluzione a **Minification** invece è la tecnica di mip-mapping descritta nel paragrafo seguente.

7.6.1 Mipmapping: pre-calcolo (Williams 1983)

Il più comune metodo per gestire antialiasing di texture è chiamato **mipmapping** ed è implementato anche nelle più modeste GPU. "Mip" è acronimo del latino *Multum in parvo*, quindi molte cose in un piccolo spazio.

La tecnica consiste nel memorizzare non un'unica texture map ma una piramide di texture map con risoluzione decrescente di livello in livello di un fattore in potenza di 2 $(512, 256, 128, ..., 1)$. L'immagine originale è a livello 0 alla base della piramide e ogni livello viene calcolato dal precedente mediante una media (filtro passa basso) dall'immagine originale.

Un esempio di piramide di texture map prodotte per il mip-mapping è illustrato in Fig.7.24.

Il livello di texture da adottare dipenderà dall'area sullo schermo occupata dalla proiezione dell'oggetto a cui applicare la texture.

Se questa è sufficientemente grande i texel vengono selezionati dalla texture image ad alta risoluzione (un esempio è illustrato nella figura 7.25), ovvero ai livelli bassi della piramide.

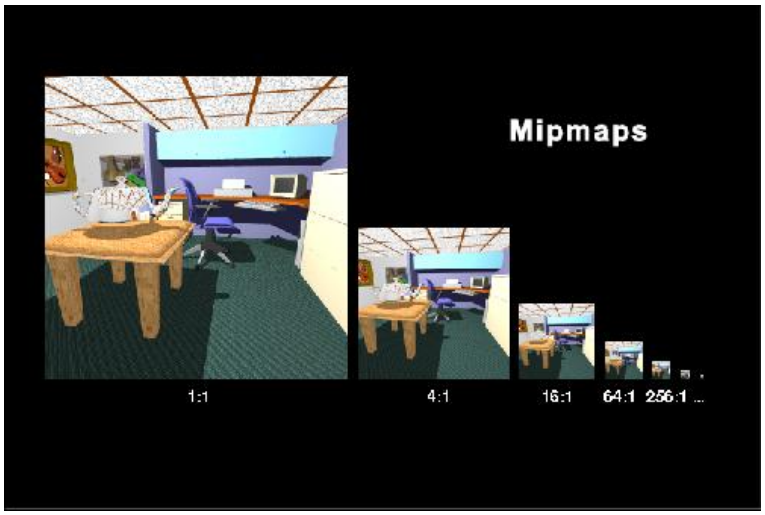


Figure 7.24: Mipmapping

Se, viceversa, l'oggetto viene proiettato in una piccola area dello schermo, occorre un filtering più significativo per evitare l'aliasing e dunque si sceglie una versione maggiormente filtrata (livelli verso l'apice della piramide).

Una versione approssimata della pre-immagine è calcolata approssimando l'area curvilinea con un quadrato nel texture space di area A e lato $b = \sqrt{A}$. Il livello d di mipmap da utilizzare è quindi calcolato come $d = \log_2 b$, dove $d = 0$ è lasciato alla texture di massima risoluzione. Il valore d sarà compreso tra i livelli texture della piramide n e $n + 1$, e dalle coordinate texture (s, t) un'interpolazione trilineare (s, t) sui due livelli n e $n + 1$ e d, permetterà quindi di calcolare il colore finale texture da assegnare al pixel.

7.7 Solid Texturing

Avvolgere una superficie arbitraria 3D con un'immagine bidimensionale risulta spesso difficile o insoddisfacente. L'immagine presenta deformazioni per coprire correttamente l'oggetto solido che rendono poco realistica la texture. Una soluzione a questi problemi è fornita dalla naturale estensione delle immagini texture, ovvero un'immagine tridimensionale

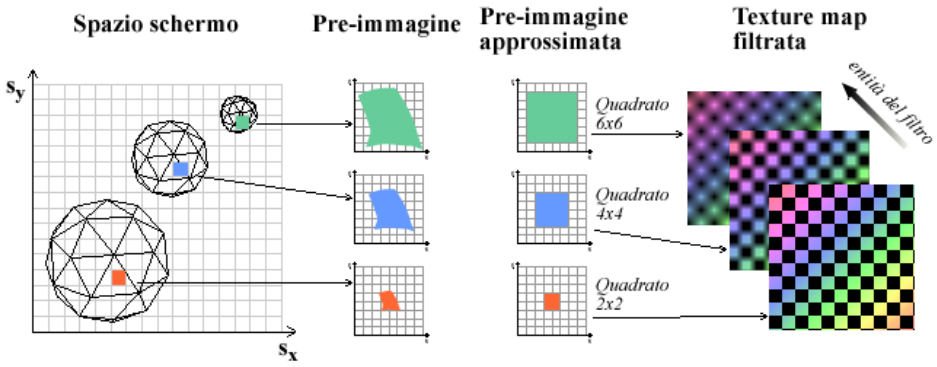


Figure 7.25: Sono evidenziati tre pixel dello spazio schermo e le relative pre-immagini. A seconda dell'area della pre-immagine viene selezionato il grado di filtraggio della texture da applicare (ovvero il livello di risoluzione nella piramide delle immagini).

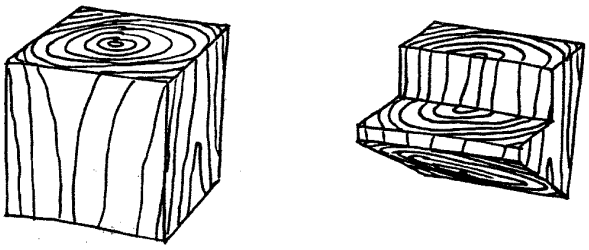


Figure 7.26: Solid texturing

che forma un volume di dati scalari.

Un'esempio classico di utilizzo di texture 3D sono le texture legno o marmo che permettono di rappresentare un oggetto come se questo fosse scolpito nel materiale simulato. L'effetto è quello dell'immersione dell'oggetto nello spazio texture 3D.

Una texture 3D consiste, per esempio, di una funzione che ad ogni punto (x, y, z) di un volume faccia corrispondere la terna (R, G, B) del colore della texture. Quindi per ogni punto sulla superficie dell' oggetto si valuta il valore di tale funzione di texturing.

Texture 3D possono essere create mediante l'utilizzo di un volume di immagini, come ad esempio una texture creata da sweeping di immagini 2D. Questo metodo risulta ovviamente oneroso in termini di occupazione di

memoria. Alternativamente, le texture 3D possono essere create in modo procedurale. Quest'ultima tecnica penalizza l'efficienza computazionale, sebbene il compito di calcolare le texture 3D procedurali possa essere demandato ai pixel shader per un calcolo 'on the fly'.

Uno dei metodi più comuni per generare funzioni texture 3D è stato introdotto da Ken Perlin nel 1985 e fa uso di funzioni rumore per generare i valori pixel.

La funzione Perlin Noise $Noise : \mathbb{R}^n \rightarrow [-1, 1]$ è una funzione continua di interpolazione di numeri pseudo-casuali che ritorna un singolo numero pseudo-casuale nell'intervallo $[-1, 1]$. Il Perlin Noise è una primitiva stocastica che ha lo scopo di apparire casuale, ma non lo è realmente: si può vedere come generatore di numeri casuali a cui dato un ingresso però corrisponde sempre lo stesso valore "random". Per maggiori dettagli si visiti il sito: **http://mrl.nyu.edu/ perlin/noise/**.

Per ottenere una perturbazione anche sui vettori normali si può generalizzare alla generazione di vettori in $\mathbb{R}^3$ pseudo-casuali costruendo un'interpolante cubico di Hermite come funzione $Noise()$.

Per generare effetti di rumore diversi si puo utilizzare una variante del Perlin Noise, chiamata Turbolenza definita dalla somma pesata di armoniche dalla seguente espressione:

$$Turbulence(x) = \sum_{i=0}^{k} \frac{1}{a^i} |Noise(b^i x)|, \tag{7.1}$$

dove i parametri a, b permettono di modificare frequenza e ampiezza. Per esempio $a = b = 2$ rappresenta armoniche di frequenza raddoppiata e ampiezza dimezzata.

7.8 Bump Mapping (Blinn 1978)

Introdotta da Blinn nel 1978, la tecnica di bump mapping permette di caratterizzare la superficie di un oggetto (microstruttura) con un aspetto rugoso, ondulato, in rilievo.

L'effetto dei bumps sulle superfici è una apparente variazione geometrica

Figure 7.27: Procedural texturing 3D.

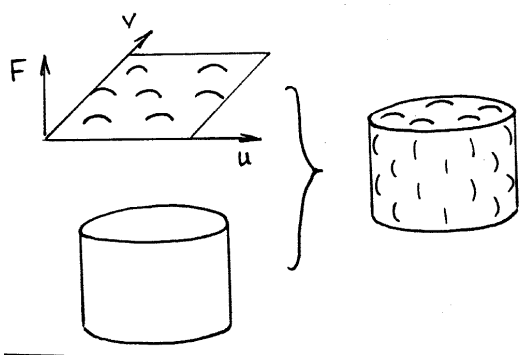


Figure 7.28: Bump mapping

locale dovuta ad una variazione della normale in un punto. La normale geometrica dell'oggetto rimane invariata e così la geometria dell'oggetto, la sola variazione è della normale usata nel modello di illuminazione locale che produce una variazione nell'ombreggiatura sulla superficie.

Problemi tipici del bump mapping si riscontrano nei profili dell'oggetto che seguono la geometria originale non riproducendo quindi le variazioni simulate con il bump. Inoltre, se l'oggetto è poligonale la mesh deve essere sufficientemente raffinata per essere sensibile alle perturbazioni della bump map.

Consideriamo ora i dettagli della realizzazione della tecnica bump mapping definita da una funzione bump $B(u, v) : R^2 \rightarrow R^+$.

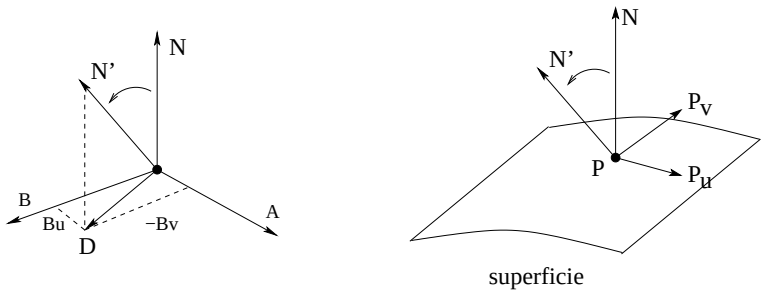


Figure 7.29: Calcolo Bump mapping

Consideriamo un punto $P(u, v)$ su una superficie parametrica, la normale alla superficie in quel punto è data da:

$$N = P_u \times P_v,$$

cioè dal prodotto esterno tra le derivate parziali rispetto alle due direzioni parametriche. Se spostassimo il punto P lungo la normale N di una quantità pari a $B(u, v)$, otteremo il punto $P'(u, v)$ dato da:

$$P'(u, v) = P(u, v) + B(u, v)N.$$

La nuova superficie ottenuta applicando la bump map alla superficie originale avrà nel punto P' vettore normale:

$$N' = P'_u \times P'_v,$$

$$P'_u = P_u + B_u N + B(u, v)N_u, \quad P'_v = P_v + B_v N + B(u, v)N_v.$$

Se $B(u, v)$ definisce uno scostamento 'piccolo' possiamo riscrivere:

$$N' = (P_u + B_u N) \times (P_v + B_v N) = N + (B_u A - B_v B) = N + D,$$

dove D è il vettore che giace nel piano tangente e sposta N nella posizione desiderata, calcolato dalle derivate parziali della bump map e dai due vettori nel piano tangente.

Come bump map può essere utilizzata un'immagine a livelli di grigi ovvero una griglia di pixel (i, j) a ciascuno dei quali corrisponde un valore $B_{i,j}$. Le mappe delle derivate parziali B_u e B_v sono approssimate mediante

Figure 7.30: Esempi di bump mapping

metodi alle differenze finite ($B_u \propto B_{i+1,j} - B_{i,j}$, $B_v \propto B_{i,j+1} - B_{i,j}$) e sono chiamate **normal map**.

Un altro metodo per generare una bump map da applicare ad un oggetto rappresentato con mesh poligonale M_0 è il seguente.

Si genera una parametrizzazione della mesh M_0 così che ad ogni punto 3D della mesh sia associato univocamente un punto 2D sulla texture map 2D. Si semplifica la mesh M_0 in M_1. Sovrapponendo i due modelli M_0 ed M_1 per ogni punto P della mesh semplificata M_1 si determina il punto più vicino P' sul modello originale M_0 (ray casting). Si memorizza la normale a P' nella normal map.

La normal map è quindi memorizzata come immagine a tre componenti.

Esempi di applicazione della tecnica bump mapping sono mostrati in Fig. 7.30. Mentre in Fig.7.31 è mostrato un esempio di composizione di un'immagine a colori (texture image 2D) ed una sua versione a livelli di grigio che rappresenta una bump map, applicati allo stesso modello geometrico.

7.9 Environment Mapping

L' environment mapping (EM) permette di simulare mediante un modello di illuminazione locale la resa che si ottiene con un ray tracer di oggetti caratterizzati da materiali riflettenti.

L'effetto che si vuole riprodurre è quello del riflesso su oggetti particolar-

Figure 7.31: Texture mapping + Bump mapping

mente riflettenti dell'ambiente circostante.

Con l'algoritmo di ray tracing il contributo dell'ambiente riflesso nel punto P su di una superficie, è determinato dal vettore riflesso R che esce da P e rimbalza in scena.

Si vuole ottenere lo stesso effetto usando una funzione di R per indicizzare una mappa bidimensionale (detta environment map).

I passi dell'algoritmo EM sono i seguenti:

- **Fase 1) PREPROCESSING** per costruire la environment map si rende la scena senza l'oggetto riflettente con la camera posizionata al centro dell'oggetto rivolta nella direzione della normale dell'oggetto; otteniamo così un'immagine dell'ambiente come se fosse 'visto' dall'oggetto riflettente;
- **Fase 2) RESA** si riposiziona l'oggetto in scena e si effettua la resa normale, usando questa immagine per ottenere i valori del colore

Figure 7.32: Environment mapping cubico

pixel.

Esistono sostanzialmente tre metodi per ottenere Environment Mapping e vengono classificati a seconda del modo nel quale le informazioni dell'ambiente 3D sono mappate nel 2D: cubico, sferico e latitudine-longitudine.

7.9.1 Environment Mapping: mapping cubico

La fase 1 del mapping cubico è ottenuta posizionanado la camera al centro dell'oggetto e rendendo l'ambiente sulle sei facce di un cubo posizionato con il centro nella posizione della camera. La mappa ambiente è quindi formata da 6 mappe. Quindi occorrono sei rese della scena ciascuna con una diversa direzione di vista. Questo tipo di environment mapping è mostrato in Fig.7.32.

Nella fase 2 del mapping cubico per ogni punto sulla superficie dell'oggetto da rendere si calcola il vettore riflessione R. Il vettore di riflessione R seleziona quindi la faccia del cubo dalla coordinata di più grande valore (es. $R = -0.1, 0.4, -0.84$ seleziona la faccia $-z$). Le altre due coordinate (normalizzate dalla terza coordinata) selezionano il pixel nella immagine associata alla faccia selezionata (es. $(-0.1, 0.4)$ viene mappato in $(0.38, 0.80)$).

Per ogni pixel dovremmo in realtà considerare un fascio di vettori di riflessione piuttosto che uno solo, e filtrare quindi l'area sottesa a questo fascio nella mappa per ottenere il valore del colore del pixel.

Esempio. In Fig.7.33 è illustrata la scena di una teapot di materiale metal-

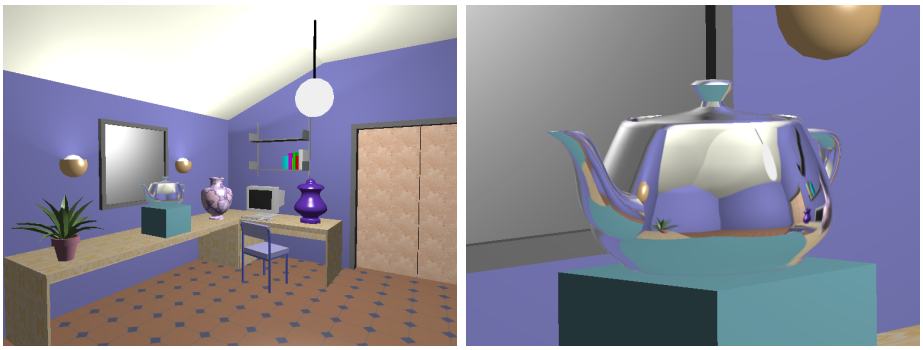


Figure 7.33: Esempio di environment mapping cubico: scena e particolare elaborato con environment mapping.

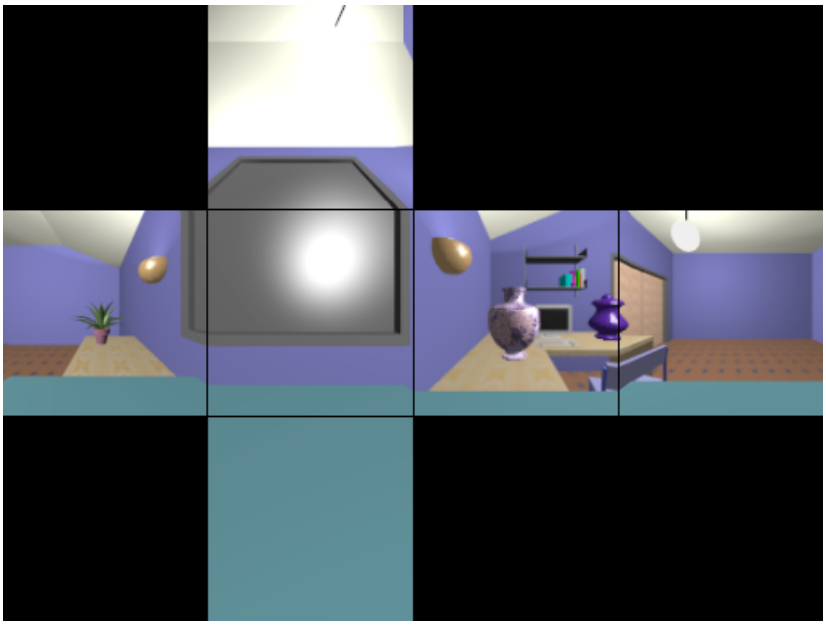


Figure 7.34: Sei immagini generate dalla fase 1 di preprocessing dell'algoritmo di EM cubico.

lico resa con un modello di illuminazione locale e il particolare della teapot che è stata resa con la tecnica di environment mapping cubico.

Il calcolo della mappa ambiente viene effettuato con la teapot fuori scena e la posizione di vista al centro della teapot. La risoluzione qui utilizzata è 128×128 per ognuna delle 6 mappe (Fig.7.34).

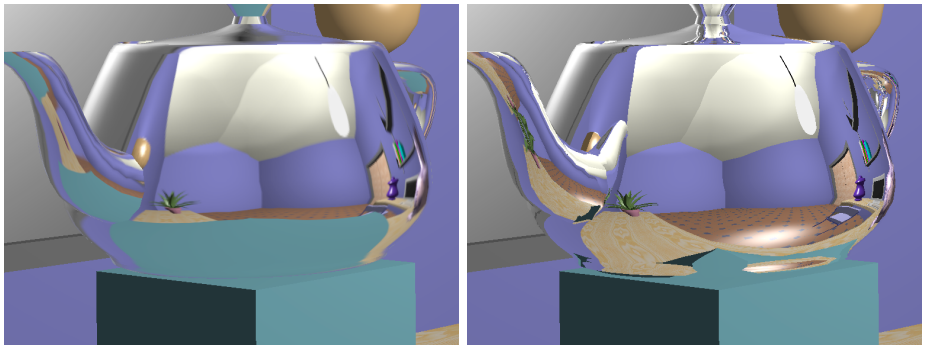


Figure 7.35: Environment mapping (sinistra) vs. ray tracing (destra).

Figure 7.36: Environmental Mapping: mapping sferico

In Fig.7.35 è invece mostrato il risultato dell'environment mapping (sinistra) confrontato con una resa ottenuta con ray tracing (destra). Il calcolo delle mappe ambiente senza l'oggetto non permette di gestire concavità dell'oggetto ovvero riflessioni dell'oggetto su se stesso. Si veda per esempio il pomello del coperchio della teapot e il beccuccio della teapot.

7.9.2 Environment Mapping: Latitude Mapping

Nel 1976 Blinn e Newell svilupparono il primo algoritmo di environment mapping. Per ogni punto sull'oggetto viene calcolato il raggio riflesso R,

Figure 7.37: Environment Mapping: Latitude Mapping

in termini della normale N al punto e della direzione di vista V:

$$R = (R_x, R_y, R_z) = 2(N \cdot V)N - V.$$

Il raggio R punta verso l'ambiente circostante l'oggetto. Quindi ad ogni R si deve far corrispondere la coppia di coordinate sferiche (ρ, ϕ) per accedere alla environment map, Il valore di ϕ, chiamato longitudine, varia da 0 a 2π,

$$\phi = \begin{array}{ll} \arccos(-R_x / \sin(\theta)) & se \quad R_y \geq 0 \\ 2\pi - \arccos(-R_x / \sin(\theta)) & altrimenti \end{array}.$$

e ρ, chiamato latitudine, varia da 0 a π ed è calcolata da

$$\rho = \arccos(-R_z)).$$

Un esempio di latitude mapping è mostrato in Fig.7.37.

I problemi dell'environment mapping possono essere riassunti nei seguenti punti:

- E' geometricamente corretto solo quando l'oggetto è piccolo rispetto all'ambiente che riflette in esso;
- L'oggetto può solo riflettere l'ambiente e non se stesso e quindi la tecnica non è corretta per oggetti concavi.
- Una nuova map è richiesta per ogni punto di vista;

- Costo (off-line) per il calcolo della mappa ambiente.
- Richiede una mappa per ogni oggetto riflettente in scena.

7.9.3 Light Map

Una light map è un'immagine che rappresnta l'ambiente illuminato con modalità globali che comprendono effetti di ombre, riflessioni,..Viene applicata agli oggetti con tecniche di texture mapping per simulare un'illuminazione globale utilizzando solo illuminazione locale.

Si effettua un precalcolo dell'illuminazione in scena mediante un algoritmo di illuminazione globale e si memorizza la resa su una 2D light map (simile all'environment map), quindi in fase di shading si fa riferimento light map per la determinazione del colore del pixel.

Per il precalcolo della light map deve essere usato un metodo di illuminazione globale view independent (es. radiosity).

7.10 Displacement Mapping

La tecnica di displacement mapping utilizza un heightfield per perturbare un punto sulla superficie lungo la direzione della normale alla superficie. La perturbazione modifica realmente la geometria del modello anzichè modulare solo i parametri di shading come la tecnica di bump mapping.

Con **displaced subdivision** si intende la tecnica che aggiunge dettagli geometrici a superfici parametriche e superfici subdivision.

Sia p un punto sulla superficie e n la sua normale, allora il punto sulla superficie 'spostata' è dato da

$$s = p + dn$$

con d scalare che rappresenta lo scostamento del punto p.

L'idea è di utilizzare una mesh di controllo a risoluzione bassa (coarse) e applicare uno schema di subdivision per arrivare ad una superficie ad alta risoluzione (smooth) quindi applicare uno spostamento di un valore scalare lungo la normale (Fig.7.38).

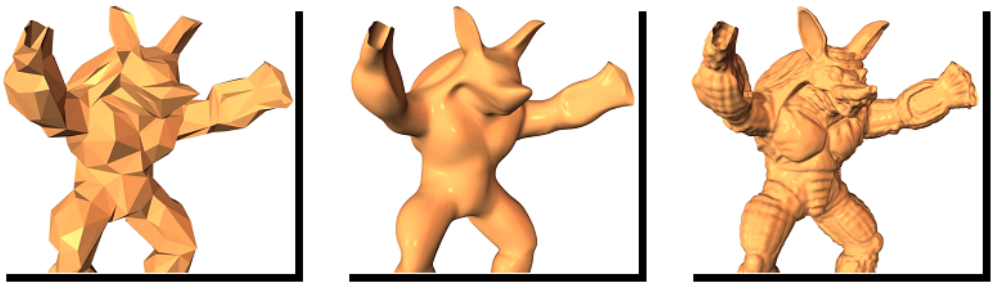


Figure 7.38: Esempio di displaced subdivision.

Per le superfici di subdivision p è il punto limite della superficie a partire da una mesh di controllo coarse. La normale $n = p_u \times p_v$, per il vertice p^k dello schema di suddivisione di Loop è ottenuta calcolando le approssimazioni:

$$p_u = \sum_{i=0}^{n-1} \cos(2\pi i/n) p_i^k \quad p_v = \sum_{i=0}^{n-1} \sin(2\pi i/n) p_i^k$$

E' necessario inoltre calcolare la normale alla superficie n_s di suddivisione 'spostata' per poterla rendere correttamente:

$$n_s = s_u \times s_v.$$

dove

$$s_u = \frac{\partial s}{\partial u} = p_u + d_u n + d n_u, \quad s_v = \frac{\partial s}{\partial v} = p_v + d_v n + d n_v,$$

Il terzo termine può essere ignorato per piccoli scostamenti.

Si definisce la displacement map (un height field) per ogni triangolo della coarse mesh. Questa segue i raffinamenti della coarse mesh: quando un triangolo è suddiviso tre nuovi punti vengono creati e viene determinato lo scostamento per questi 3 nuovi punti.

Implementazioni di questa tecnica si trovano in API come DirectX 9 e Renderman.

Figure 7.39: Alfa Blending

7.11 Opacità e alpha Blending

L'alpha channel è la quarta componente colore nella modalità colore $RGB\alpha$. Come per le altre componenti colore un'applicazione può controllare il valore di α per ogni pixel.

Il valore *alfa* in $RGB\alpha$ viene utilizzato per creare effetti interessanti. Per esempio permette la miscelazione di due colori.
Se un oggetto semitrasparente si trova davanti ad un altro, il risultato nel frame buffer è un misto tra i colori dei due oggetti.

L'opacità di una superficie è una misura di quanta luce penetra attraverso la superficie. Un valore di opacità pari a 1 ($\alpha = 1$) corrisponde ad una superficie completamente opaca. Una superficie con opacità 0 ($\alpha = 0$) è trasparente: tutta la luce passa attraverso ad essa.

Si possono utilizzare i piani di bit aggiuntivi nel frame buffer per memorizzare α per ogni pixel che corrisponde al valore miscelato. Si veda Fig.7.39 per un esempio.