Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE
DELLA INFORMAZIONE

Pisa



# *TECNICHE DI VISUALIZZAZIONE VOLUMETRICA*

**Luca MOLINARI, Antonio PLASTINO, Enrico TORACCA e Claudio MONTANI**

Nota Interna, B4-11, Maggio 1993

Indice

# INTRODUZIONE

In un numero sempre crescente di discipline scientifiche viene fatto uso di tecniche di visualizzazione per favorire la comprensione di fenomeni, migliorare la conoscenza di strutture o materiali o accelerare il processo decisionale. In questa vasta classe di problemi, che prende il nome di visualizzazione scientifica, un tema molto importante è rappresentato dalla *visualizzazione volumetrica*. Il problema di base può essere descritto in modo molto semplice: dato un insieme di informazioni puntuali definite in uno generico spazio a tre o quattro dimensioni (dove nell'ultimo caso la dimensione aggiuntiva è quella temporale), la visualizzazione volumetrica ha lo scopo di definire le tecniche ed i metodi che consentono di ottenere una rappresentazione visibile per i fenomeni, le strutture o le forme che l'insieme di dati rappresenta.

A partire dalle diverse organizzazioni topologiche degli insiemi di dati (Capitolo 1) e dalle diverse tecniche di visualizzazione adottabili (Capitoli 2 e 3), questo lavoro si è massimamente concentrato sugli algoritmi di estrazione e presentazione di isosuperfici di interesse da dataset volumetrici basati su grigliati regolari 3D. Il vincolo sulla regolarità della struttura del dataset non deve essere interpretato come una limitazione del progetto in quanto la grande maggioranza delle applicazioni analizza dati campionati o calcolati su grigliati uniformi.

# 1. VISUALIZZAZIONE VOLUMETRICA

Gli aspetti fondamentali che caratterizzano la gestione e visualizzazione di dati volumetrici possono essere classificati secondo quattro criteri:

1) La struttura geometrica del dataset (l'organizzazione topologica dei dati).
2) L'estensione spaziale del dato.
3) Le tecniche di visualizzazione adottate.
4) Le tecniche di shading impiegate, cioè le tecniche utilizzate per esaltare la sensazione di profondità di oggetti tridimensionali (3D) proiettati su uno schermo bidimensionale (2D).

I primi due aspetti di questa classificazione verranno esposti nei due paragrafi seguenti, l'aspetto riguardante le tecniche di visualizzazione verrà trattato nel secondo capitolo; il terzo capitolo, infine, entrerà in qualche dettaglio del problema, non secondario, delle tecniche di shading.

## 1.1. LA STRUTTURA GEOMETRICA DEL DATASET

Col termine *dataset volumetrico* [Fren89], si fa riferimento ad insiemi di valori (in genere scalari) acquisiti secondo una delle due modalità descritte in precedenza e geometricamente associati ai nodi di un generico grigliato 3D.

La caratterizzazione di un dataset volumetrico e' generalmente basata su due aspetti. Il primo aspetto riguarda il generico valore di informazione, detto valore di densità (di solito un numero intero o reale), proprio di ogni campione appartenente al dataset. Il valore di densità rappresenta di solito una qualche proprietà fisica dell'oggetto o del fenomeno indagato ed è ottenuto mediante dispositivi di acquisizione dei dati [Herm80].

Il secondo aspetto caratteristico di un dataset volumetrico riguarda invece la disposizione geometrica nello spazio 3D dei punti di campionamento. Questo secondo aspetto può essere indicato come la "struttura geometrica del dataset" e specifica la geometria del grigliato cui sono associati i valori di densità.

I vari tipi di grigliati che si incontrano nella visualizzazione di un dataset volumetrico possono essere classificati secondo lo schema proposto da Speray [Sper90] nel modo seguente:

**Griglie regolari**: i nodi del grigliato sono posti ad intervalli regolari lungo i tre assi cartesiani; ogni nodo P di coordinate (i,j,k) (che non sia sui bordi) ha pertanto sei nodi ad esso adiacenti, due per ogni asse:

$P_1=(i+dx,j,k)$; $P_2=(i-dx,j,k)$;
$P_3=(i,j+dy,k)$; $P_4=(i,j-dy,k)$;
$P_5=(i,j,k+dz)$; $P_6=(i,j,k-dz)$;
con dx,dy,dz costanti (la Figura1.1 mostra un esempio 2D).

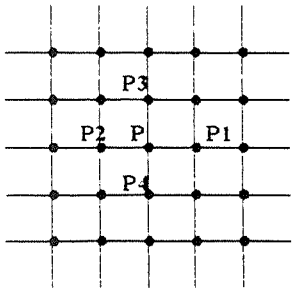



Figura 1.1 Griglia regolare.

**Griglie rettilinee**: le distanze tra i nodi lungo un asse sono arbitrarie. Le celle i cui vertici sono nodi adiacenti del grigliato sono ancora parallelepipedi ed allineate secondo gli assi cartesiani (in Figura 1.2 è mostrato un esempio 2D).

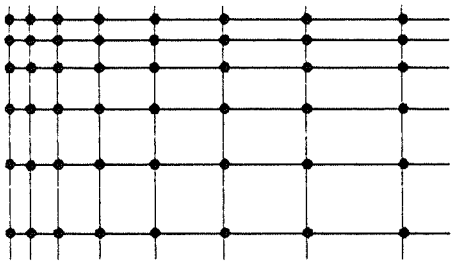


Figura 1.2 Griglia rettilinea.

**Griglie irregolari**: le celle del grigliato sono in questo caso esaedri (in Figura 1.3 è mostrato un esempio 2D). Questi grigliati sono tipici delle applicazioni di fluidodinamica.

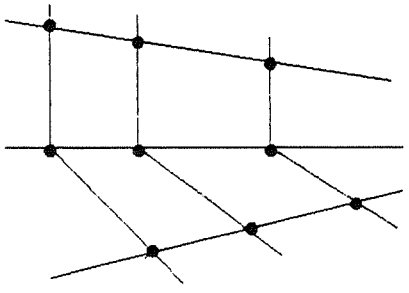


Figura 1.3 Griglia irregolare.

**Griglie non strutturate**: diversamente dai precedenti tipi di griglie, in questo caso non esiste nessun tipo di informazione che permette di individuare i punti tra loro "vicini". Le celle possono essere tetraedri, esaedri, prismi, piramidi, ecc. (in Figura 1.4 è mostrato un esempio 2D).

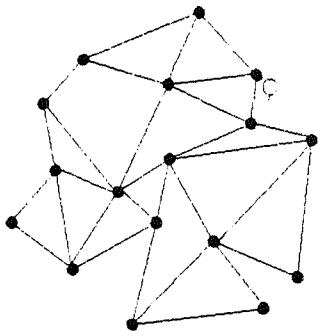


Figura 1.4 Griglia non strutturata.

**Griglie ibride**: sono formate da griglie irregolari e non strutturate assieme (in Figura 1.5 è mostrato un esempio 2D).

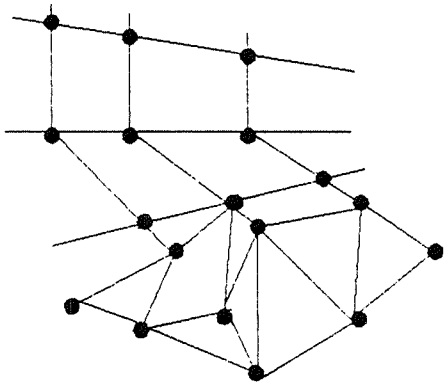


Figura 1.5 Griglia ibrida

Ai fini della visualizzazione, la struttura del dataset è un parametro fondamentale: l'esistenza di una qualche forma di strutturazione implica il poter individuare, per ogni punto dello spazio di interesse, gli elementi del dataset che in qualche modo "insistono" su quel punto. Negli algoritmi di visualizzazione che verranno presentati, sarà fatto sempre riferimento a dataset basati su grigliati regolari data la facilità con cui si possono estrarre da essi informazioni riguardanti la geometria spaziale dei dati.

## 1.2. ESTENSIONE SPAZIALE DEL DATO

Scopo di questo paragrafo è quello di definire come l'informazione associata ad un nodo (o ai nodi) del generico grigliato possa essere estesa nello spazio continuo 3D. Vengono prese in considerazione due rappresentazioni dei dati: la rappresentazione *voxel-based* e quella *cell-based*.

Nella rappresentazione voxel-based il grigliato sul quale è definito il dataset è interpretato come un array 3D di punti ($i\delta_1$, $j\delta_2$, $k\delta_3$), le cui coordinate sono multipli delle distanze $\delta_1$, $\delta_2$, $\delta_3$, tra due nodi adiacenti lungo i tre assi cartesiani. Ad ogni punto $V=(v_1,v_2,v_3)$ dell'array sono associati tutti i punti $(x_1, x_2, x_3)$ dello spazio continuo tali che: $v_i-\delta_i/2 \le x_i \le v_i+\delta_i/2$ per i=1,2,3. L'elemento di volume risultante è chiamato *voxel* (il termine *voxel* è acronimo di "volume element" analogo a *pixel* per

"picture element" nello spazio 2D). Si noti che un nodo del grigliato specifica univocamente un voxel e che ogni voxel contiene un solo nodo del grigliato [Srih81]. I voxel che formano una scena sono quindi elementi di volume rettangolari della stessa dimensione. Nella rappresentazione voxel-based il valore di densità associato al nodo che specifica il voxel viene "esteso" a tutti i punti appartenenti a quel voxel, ed è dunque considerato costante su tutto il voxel. Nell'approccio voxel-based è possibile immaginare una scena 3D come composta da una serie di "slice" parallele ad esempio al piano XY, dove ogni slice della scena è un'insieme di voxel le cui coordinate sull'asse Z sono identiche (Figura 1.6).

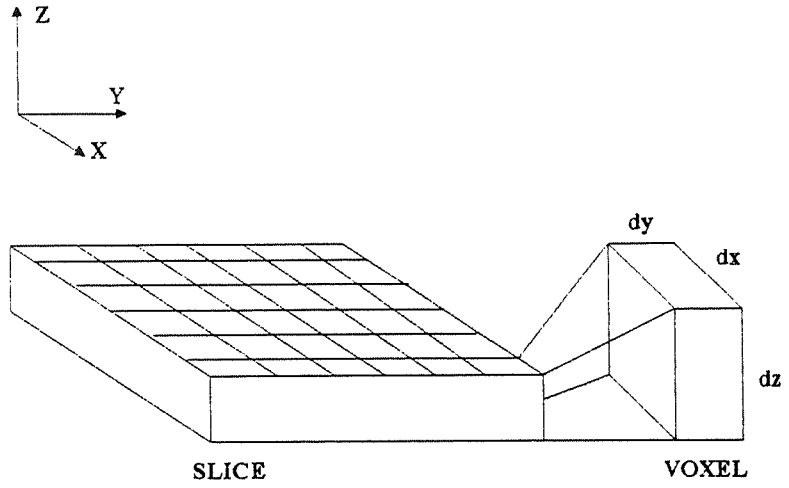



Figura 1.6 Slice e voxel componenti una scena 3D.

Nella rappresentazione cell-based, a differenza di quella voxel-based, la generica informazione non è più considerata costante in un intorno del punto assunto come riferimento. Ogni nodo del grigliato rappresenta un punto campione di una funzione continua. Un' immediata conseguenza di questa rappresentazione è che il calcolo dell'informazione associata ad un punto qualsiasi dello spazio non è più data "per definizione" ma implica necessariamente una fase di interpolazione tra i valori noti associati a punti adiacenti del grigliato. Inoltre le celle differiscono dai voxel in quanto esse sono definite come il volume individuato dagli otto vertici corrispondenti ad otto nodi adiacenti del grigliato (due lungo ogni direzione) e soprattutto per il fatto che il valore del campo scalare è variabile in tutta la cella [Upso88]. E` opportuno sottolineare che le tecniche di visualizzazione che procedono ad una estrazione di superfici ben si adattano a modelli cell-based. La possibilità infatti di interpolare linearmente tra nodi adiacenti del grigliato consente una grande flessibilità nell'individuazione di superfici. La funzione di interpolazione può variare da algoritmo ad algoritmo: nelle implementazioni più comuni essa è lineare lungo ciascuna dimensione del grigliato (Figura 1.7).

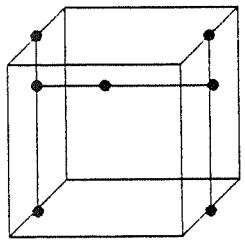


Figura 1.7 Interpolazione lineare su una cella del grigliato.

La scelta tra la rappresentazione voxel-based e quella cell-based non può essere semplicemente una scelta di progetto. In alcune applicazioni la frequenza di scansione e la risoluzione degli strumenti di input conduce a dataset voxel-based. Viceversa forzare a voxel-based un dataset intrinsecamente cell-based può condurre ad una inaccettabile perdita di informazione. Un esempio classico di dataset voxel-based è rappresentato da dataset utilizzati nella medicina diagnostica e prodotti dalla Tomografia Assiale Computerizzata (TAC) o dalla Risonanza Magnetica Nucleare (NMR); al contrario, i dati di rilevazione della concentrazione di anidride solforosa nell'aria di una città sono invece interpretabili come dataset cell-based, data generalmente la bassa risoluzione spaziale dei campioni esaminati rispetto all'estensione della città.

# 2. TECNICHE DI VISUALIZZAZIONE VOLUMETRICA

Tra i vari aspetti che caratterizzano l'elaborazione di *dataset volumetrici*, quello che senza dubbio merita una trattazione più approfondita è l'aspetto legato alle tecniche di visualizzazione. Dato il vasto numero di algoritmi di visualizzazione proposti in letteratura, non è negli scopi di questo capitolo fornire una trattazione completa ed esauriente di tali algoritmi. L'interesse è piuttosto rivolto a fornire una panoramica delle tecniche di visualizzazione volumetrica più note e significative proposte negli ultimi anni. L'approccio seguito per classificare tali tecniche è quello utilizzato in [Mont92]. In tale articolo si opera una distinzione tra:

- tecniche che procedono ad una ricostruzione di superfici di interesse (isosuperfici) convertendo il modello in un insieme di superfici poliedriche connesse che approssimano la superficie dell'oggetto da visualizzare, provvedendo poi alla loro visualizzazione;
- tecniche di visualizzazione diretta che invece procedono direttamente alla visualizzazione di isosuperfici senza la necessità di applicare algoritmi per convertire il modello volumetrico.

Le tecniche che procedono ad una ricostruzione di superfici che prese in esame sono: Triangolazione di Contorni Planari, Marching Cube, Dividing Cube, Surface Tracking. Il vantaggio di tali tecniche rispetto alle tecniche di visualizzazione diretta è che la superficie ricostruita viene esplicitamente determinata prima di generare una rappresentazione della superficie stessa sullo schermo. Ciò permette, una volta costruita la superficie, di ottenere viste diverse senza la necessità di rivisitare il dataset originale. Inoltre, l'uso di rappresentazioni intermedie poliedriche permette l'utilizzo per la visualizzazione delle eventuali funzionalità Hw presenti su molte workstation riducendo notevolmente i tempi di visualizzazione.

Delle tecniche di visualizzazione diretta vengono esaminate invece le tecniche basate su Ray Tracing e le Tecniche Proiettive. Il vantaggio di quest'ultima classe di algoritmi di visualizzazione sta nel fatto che l'utente può derivare una vista complessiva del dataset senza richiederne la conversione in strutture di rappresentazione intermedie.

## 2.1. TRIANGOLAZIONE DI CONTORNI PLANARI

Nei metodi presentati in questo paragrafo [Fuch77], in una prima fase si effettua una ricerca di contorno sulle singole slice che costituiscono il dataset. Ogni contorno rappresenta l'intersezione dell'oggetto rappresentato con il piano associato alla slice. ed è rappresentato mediante una sequenza finita di punti ordinati ricavati di norma percorrendo il contorno stesso in verso antiorario. La porzione di contorno tra due punti consecutivi e' approssimata da un segmento lineare chiamato "contour segment" (figura 2.1). In un dataset voxel-based, la sequenza di punti corrisponde ai voxel connessi, appartenenti ad una slice, che giacciono sul bordo tra l'oggetto ed il materiale circostante.

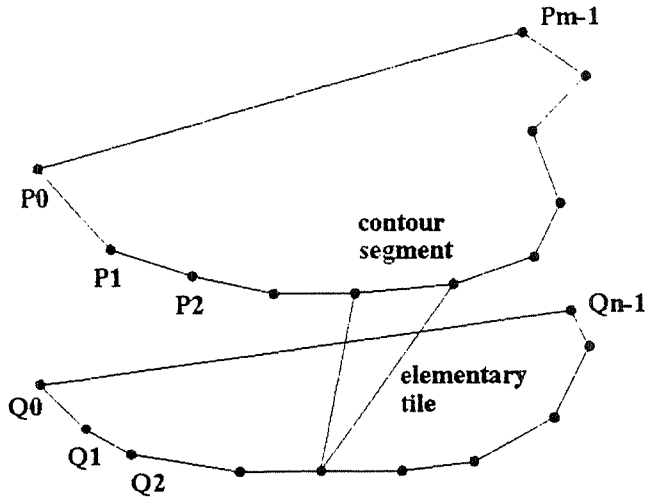



Figura 2.1 Coppia di contorni individuati da due slice

Tali voxel possono essere ricavati tramite una classificazione con soglia (thresholding), in cui i voxel con valori minori di una soglia prefissata sono considerati interni all'oggetto e quelli con valori maggiori della soglia esterni [Styt91].

La superficie dell'oggetto è ricostruita congiungendo coppie di contorni planari adiacenti tramite una triangolazione, cioe` congiungendo punti appartenenti a due contorni vicini in maniera tale da ottenere triangoli che delimitino un poliedro approssimante la superficie di interesse. Formalizzando il problema poniamo $P_0,...,P_{m-1}$ l'insieme di m punti distinti che rappresentano il contorno superiore e $Q_0,...,Q_{n-1}$ l'insieme di punti che rappresenta il contorno inferiore. Un "elementary tile" e` una faccia triangolare composta da un singolo contour segment (che forma la base) e da due lati che connettono i punti estremi del contour segment con un unico punto appartenente al contorno adiacente. I due lati dell'elementary tile sono chiamati rispettivamente destro e sinistro. Per seguire l'algoritmo dovuto a Ganapathy [Gana82], si assume che l'insieme degli elementary tile che definisce una superficie soddisfi, per definizione, due regole:

(1) ciascun contour segment appare in uno ed un solo elementary tile. L'insieme contiene, quindi, n+m elementary tile;

(2) se un lato appare come lato sinistro (destro) di qualche elementary tile nell'insieme, esso apparira` come lato destro (sinistro) di un altro e unico elementary tile nell'insieme.

E` stato mostrato che un insieme di elementary tile che non rispetta questi criteri, non puo` descrivere completamente la superficie tra due contorni planari adiacenti [Gana82]. Un insieme di elementary tile che soddisfi le regole (1) e (2) e` chiamato "superficie accettabile". L'insieme di tutte le superfici accettabili, per una coppia di contorni adiacenti $P_0,..,P_{m-1}$ e $Q_0,..,Q_{n-1}$, puo` essere rappresentato da un grafo diretto G[V,A]. L'insieme di vertici V corrisponde all'insieme di tutte le possibili congiunzioni tra il contorno superiore ed inferiore. L'insieme degli archi A corrisponde a tutti i possibili elementary tile (figura 2.2). Un vertice $v_{i,j}$ rappresenta il collegamento tra il punto $Q_i$ del contorno inferiore ed il punto $P_j$ del contorno superiore, mentre un arco $[v_{i,j},v_{i,j+1}]$ rappresenta un elementary tile formato dai lati $Q_iP_j$, $Q_iP_{j+1}$ e dal contour segment $P_jP_{j+1}$.

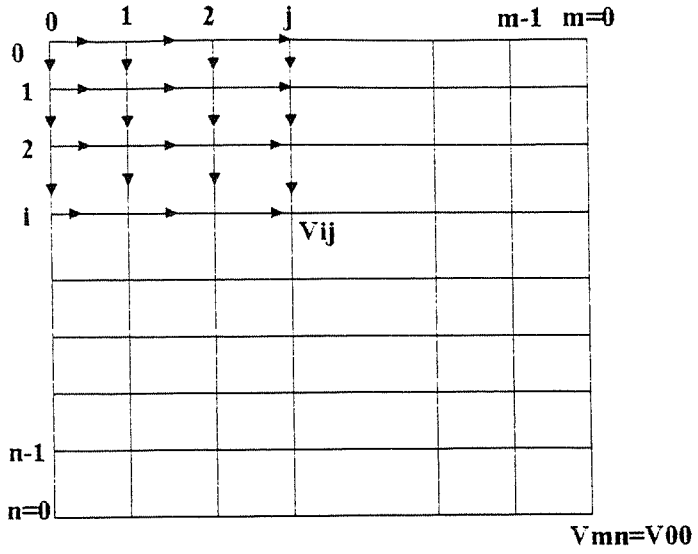



Figura 2.2 Grafo rappresentante tutte le superfici accettabili per una coppia di contorni.

Una superficie accettabile e` rappresentata da un cammino dal vertice $v_{0,0}$ al vertice $v_{m,n}$ con n+m archi. Il numero di superfici accettabili per una coppia di contorni di n e m punti e` una funzione A[m,n] che puo` essere ricorsivamente determinata come:

$$A[m,n]=A[(m-1),n] + A[m,(n-1)]$$
$$A[m,1]=1 \ \forall \ m$$
$$A[1,n]=1 \ \forall \ n$$

quindi

$$A[m,n] = \frac{[(m-1)+(n-1)]!}{(m-1)!+(n-1)!},$$

quantità ad andamento esponenziale in max(m,n).

Un numero così elevato di superfici accettabili preclude una ricerca esaustiva della superficie ritenuta migliore già per valori piccoli di m ed n. A tal scopo si consideri una funzione f: $A \rightarrow \mathbf{R}$ che associa ad ogni elementary tile $a \in A$ un numero reale $f_a \in R$. Il cammino di una superficie avra` esattamente n+m archi. Il cammino relativo alla "superficie piu` accettabile" sara` tale che la funzione

$$F = \sum_{k=1}^{n+m} f_{a_k}$$

soddisfi alcune regole predefinite [Gana82].

Fuchs [Fuch77] definisce $f_a$ come l'area del triangolo associato all'arco a ($f_a$ = peso di a) e definisce F come la minima area di una superficie accettabile. In questo modo dunque il cammino ottimale è rappresentato dal cammino di peso minimo.

Alternativamente Keppel [Kepp81] definisce F come il massimo volume racchiuso da una superficie accettabile considerando cammino ottimale il cammino di peso massimo.

La determinazione di cammini di costo minimo o massimo non può tuttavia essere effettuata in base a decisioni locali (cioè scegliendo opportunamente gli elementary tile in fase di elaborazione dei contorni) ma richiede algoritmi efficienti (e costosi) di ricerca

globale su grafo. Metodi alternativi a quelli "ottimali", basati su decisioni locali ed euristiche, non richiedono invece mai piu` di n+m passi per triangolare una coppia di contorni planari. Questi metodi euristici vengono utilizzati quando la velocita` di elaborazione è di importanza maggiore rispetto ai risultati forniti dai metodi "ottimali".

Un metodo euristico di triangolazione, proposto da Ganapathy e Denney [Gana82], considera il peso $f_a$ di un arco a come la lunghezza del contour segment incluso nel triangolo corrispondente a tale arco, diviso per il perimetro del contorno contenente il segmento. Da questa scelta deriva che per gli m archi orizzontali e gli n archi verticali di una superficie accettabile valgono le seguenti relazioni:

$$F_h = \sum_{i=1}^{n} f_{a_{h_i}} = 1, \qquad F_v = \sum_{j=1}^{m} f_{a_{v_j}} = 1,$$

e inoltre che $F=F_h+F_v=2$ per tute le superfici accettabili.

Tuttavia, mentre F ha un valore uguale per ogni superficie accettabile, queste stesse superfici non sono tutte "ugualmente accettabili". Il fattore che le differenzia è l'ordine nel quale i contour segment sono visitati, che ha una precisa rilevanza fisica nella superficie ricostruita. Per una coppia di contorni di forma e dimensioni simili, la triangolazione dovrebbe procedere approssimativamente con lo stesso "passo" su ogni contorno, senza introdurre tensioni superficiali artificiali. Un approccio si questo genere è giustificato dalla ben nota legge fisica secondo cui un corpo lasciato libero tende ad assumere la conformazione di minore tensione. L'euristica alla base di questo metodo puo` allora essere cosi` formulata: i triangoli sono aggiunti alla superficie in maniera che la differenza assoluta tra la somma dei pesi degli archi orizzontali e la somma dei pesi degli archi verticali sia minimizzata ogni volta. Un passo tipico di questo metodo e` illustrato in figura 2.3. $F'_h$ rappresenta la distanza lungo il contorno superiore visitato. $F'_v$ e` similmente definita per il contorno inferiore.

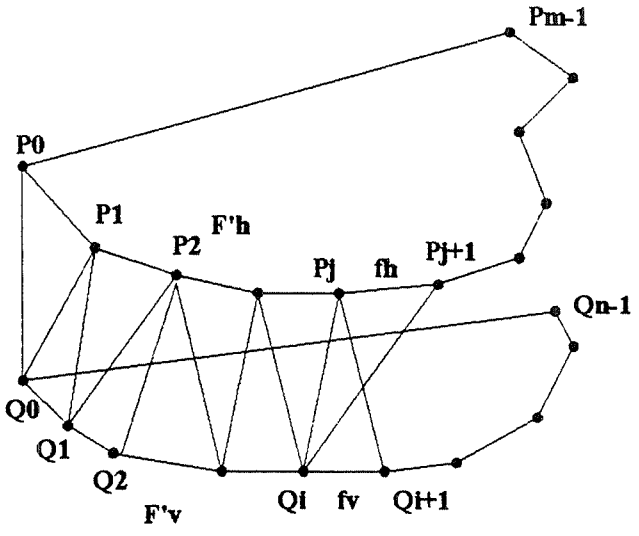



Figura 2.3 Scelta del successivo elementary tile nel metodo di Ganapathy.

Il prossimo elementary tile aggiunto alla superficie puo` essere $P_jP_{j+1}Q_i$ o $P_jQ_iQ_{i+1}$. $P_jP_{j+1}Q_i$ e` scelto se $|F'_h+f_h-F'_v|<|F'_v+f_v-F'_h|$. In questo caso j e` incrementato e si procede lungo il contorno superiore, altrimenti $P_jQ_iQ_{i+1}$ e` scelto, i e` incrementato e si procede lungo il contorno inferiore. Questo metodo richiede esattamente n+m passi per elaborare una

triangolazione completa di una coppia di contorni.

Un altro metodo, basato su decisioni locali ed euristiche, e` quello proposto da Christiansen e Sederberg [Chri78]. Il processo in questo caso comincia definendo il lato di partenza. In figura 2.4 tale lato e` $1_t\ 1_b$ . A questo punto vi sono solo due candidati per il primo triangolo: $1_t\ 2_t\ 1_b$ o $1_t\ 1_b\ 2_b$. Qui entra in gioco la decisione locale per la scelta di uno dei due elementary tile. Il peso $f_a$ per un arco e` definito come la lunghezza del lato, e tra i due triangoli viene selezionato quello con il lato più corto (in figura 2.4 e` selezionato il triangolo $1_t\ 1_b\ 2_b$). La routine prosegue considerando il secondo triangolo. La funzione F e` dunque localmente minimizzata ad ogni vertice.

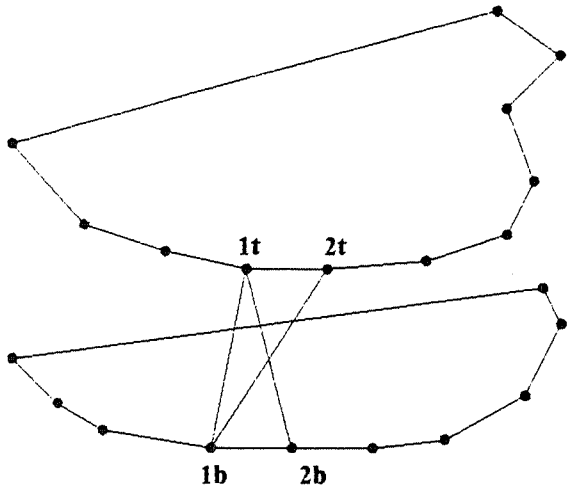



Figura 2.4 Scelta del successivo elementary tile col metodo di Christiansen.

Questo algoritmo e` facilmente realizzabile e veloce, ma e` dimostrato che questo, come gli altri metodi euristici, lavora bene su coppie di contorni che sono mutuamente centrate e ragionevolmente simili in forma e grandezza. Coppie di contorni che violano questi criteri possono comunque essere trasformati in maniera tale da soddisfarli. Un metodo per raggiungere coerenza in grandezza e forma e` quello di trasformare una coppia di contorni in maniera che giacciano all'interno di un quadrato unitario e centrato nell'origine (figura 2.5) [Chri78].

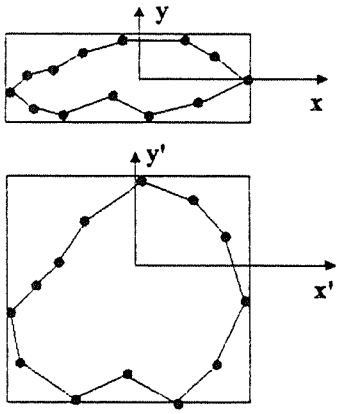



Figura 2.5 Trasformazione di una coppia di contorni planari.

Un'importante caratteristica di quest'ultimo metodo e` la capacita` di manipolare le "ramificazioni" dei contorni. Si consideri un semplice caso dove un contorno si ramifica in due contorni separati (figura 2.6). Un approccio semplice ed economico e` quello di trattare tutte le ramificazioni come un' unica linea continua e chiusa nella seguente maniera:

1) si introduce un nuovo nodo in mezzo ai due nodi piu` vicini della coppia di contorni che rappresenta la ramificazione. La coordinata z di questo nuovo nodo e` la media delle coordinate z dei due contorni implicati.

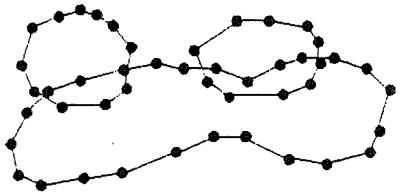


Figura 2.6 Ramificazione dei contorni.

2) Si rinumerano i nodi dei due contorni implicati in maniera che questi possano essere considerati come un unico contorno (figura 2.7). Questo significa che il nuovo nodo ed i suoi immediati vicini sono numerati due volte.
3) Si triangola normalmente.

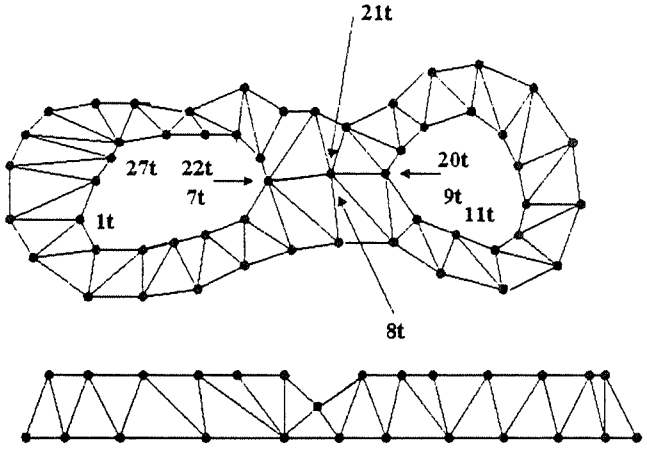



Figura 2.7 Triangolazione di contorni ramificati.

Questo schema lavora bene per un qualsiasi numero di ramificazioni.

Applicazioni degli algoritmi visti ad oggetti reali 3D hanno dimostrato che tali metodi forniscono una buona soluzione al problema della ricostruzione delle superfici. L'accuratezza di una qualsiasi rappresentazione dipende da quanti contorni sono stati usati per descrivere l'oggetto tridimensionale, da quanti punti sono selezionati per ogni contorno e da come essi sono distribuiti [Kepp81]. La proprieta` piu` utile della triangolazione, cosi` come per altri algoritmi che procedono ad una estrazione delle superfici sotto forma di una rete di poligoni, si basa, comunque, su una descrizione della superficie particolarmente adatta alla visualizzazione su un display grafico.

Le superfici triangolari prodotte per triangolazione di contorni planari possono essere

colorate col metodo di Gouraud [Gour71] (tale metodo sara' discusso nel paragrafo 3.1.4) o utilizzando il "constant shading" (par. 3.1.1).

Gli algoritmi di triangolazione di contorni dimostrano i propri limiti all'aumentare delle dimensioni dei dati (e quindi dei contour point da elaborare) e in tutti i casi in cui le conformazioni dei contorni rendano difficile definire tecniche non ambigue di connessione tra i vari punti. In questi ultimi casi, più che complicare l'algoritmo di triangolazione, si preferisce in genere ricorrere ad una qualche forma di interattività con l'utente.

## 2.2. MARCHING CUBES

Negli ultimi anni sono state proposte diverse tecniche che generano superfici poligonali direttamente a partire dai dati. Un punto di riferimento in questo senso è certamente costituito dall'algoritmo Marching Cubes (MC) di Lorensen e Cline [Lore87]. Con questa tecnica vengono generate reti di triangoli che approssimano, con un procedimento di interpolazione lineare, la superficie il cui valore è specificato dall'utente. In questo modo è possibile ricostruire superfici di isointensità, ovvero superfici che delimitano oggetti aventi la stessa densità.

Supponendo di disporre di dati volumetrici cell-based disposti su un grigliato regolare individuato da s slice bidimensionali di dimensione (r*c) e di dover visualizzare un'isosuperficie il cui valore è pari a "value", suddividiamo in passi distinti le azioni da intraprendere.

**Passo 1:** Input.

Si leggono le prime quattro slice in memoria. Il numero dei dati del modello è infatti in pratica tale da non permettere la memorizzazione contemporanea di tutte le slice (generalmente r=c=256 o r=c=128). Come si vedrà, nel caso si adotti una tecnica di shading meno sofisticata del "Gouraud shading", utilizzata da Lorensen e Cline, è sufficiente leggere soltanto due slice in memoria. Per ogni coppia di slice k e k+1, è possibile individuare (r-1)*(c-1) celle o cubi logici. Per un dato elemento i, j, k della slice k, gli 8 vertici della cella corrispondente sono formati dagli elementi di posizione: (i,j,k), (i,j+1,k), (i+1,j+1,k), (i+1,j,k), (i,j,k+1), (i,j+1,k+1), (i+1,j+1,k+1), (i+1,j,k+1), (figura 2.8).

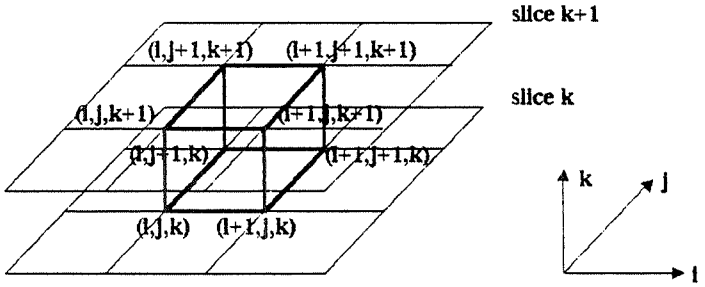



Figura 2.8 Posizione dei vertici di una cella.

**Passo 2:** Classificazione dei vertici.

Una volta individuata la cella, si procede alla classificazione dei suoi vertici per trovare la superficie che eventualmente la interseca. Si assegna cioè valore 1 (e si considerano punti esterni o sulla superficie cercata) a quei vertici il cui valore è maggiore o uguale del valore di soglia fissato che individua la superficie. Viceversa se il valore del vertice è minore del valore di soglia, si assegna 0 a quel vertice e lo si considera un punto interno. Se un lato della cella

presenta un vertice così classificato a 0 e l'altro vertice che delimita il lato ad 1, allora la superficie di valore pari alla soglia interseca quel lato della cella.

**Passo 3:** Costruzione dell'indice.

Poichè ogni cella ha otto vertici e nel passo precedente ad ogni vertice è associato un valore 0 o 1 (rispettivamente dentro o fuori la superficie) ci sono $2^8$=256 modi in cui una superficie può intersecare una cella. Ad ogni cella dopo la classificazione è associato un indice di 8 bit in cui ogni bit $v_i$ rappresenta lo stato del vertice corrispondente:

$v_i$=1 se il vertice corrispondente è stato classificato con 1

$v_i$=0 se il vertice corrispondente è stato classificato con 0

**Passo 4:** Estrazione dei vertici dei triangoli.

Enumerando tutti i 256 casi in cui una superficie interseca una cella, è possibile costruire una tabella precalcolata contenente, per ognuna delle sue 256 entrate, i lati della cella su cui si ha un'intersezione tra lato e superficie. Ogni entrata di tale tabella viene indirizzata dall'indice binario associato alla cella e calcolato al passo 3. L'intersezione sui lati della cella è considerata come il vertice di un triangolo che approssima la superficie al suo interno. Sfruttando la proprietà dei casi complementari dell'indice per cui, scambiando i vertici con valore maggiore della soglia con quelli con valore minore, la superficie interna non cambia, è possibile ridurre il numero di casi distinti a 128. Sfruttando inoltre proprietà di simmetria nelle rotazioni della cella, il numero di casi distinti si riduce a 15. La figura 2.9 mostra i 15 casi evidenziando con un cerchio nero i vertici classificati a uno.

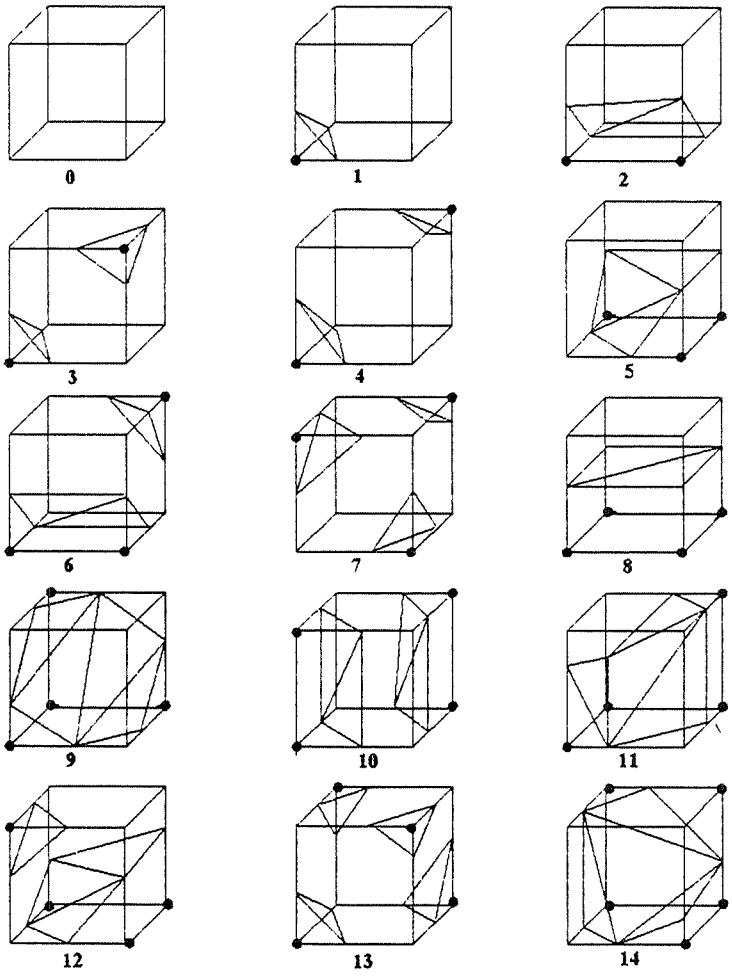



Figura 2.9 I 15 casi canonici di intersezione cella-superficie.

Il caso 0 occorre se tutti gli 8 valori dei vertici sono inferiori al valore di soglia e dunque non produce triangoli. Il caso 1 occorre se la superficie separa un vertice dagli altri 7, producendo un triangolo definito da 3 intersezioni sui lati della cella. Gli altri casi producono più triangoli o disgiunti tra di loro (caso 4) o aventi in comune un lato (caso 9) formando così superfici generalmente non planari.

**Passo 5:** Interpolazione dei vertici dei triangoli.

Una volta individuati i lati della cella che sono intersecati dalla superficie, occorre stimare il punto di intersezione ovvero il vertice del triangolo. Tale stima è effettuata utilizzando l'interpolazione lineare tra i valori dell'intensità ai due vertici del lato.

Posto D(i) il valore della densità al vertice di posizione i, D(i+dx) il valore della densità al vertice di posizione i+dx, e value il valore di soglia rappresentante la superficie, la posizione x del punto di intersezione sul lato (i,i+dx) è data da:

$$x = i + \frac{value - D(i)}{D(i + dx) - D(i)} dx$$

**Passo 6:** Calcolo delle normali ai vertici.

Quest'ultimo passo consiste nel calcolo della normale unitaria ad ognuno dei vertici dei triangoli all'interno della cella. Per il calcolo della normale è necessaria una stima del vettore gradiente agli otto vertici della cella. Come visto nel paragrafo 2.3 il gradiente al vertice i, j, k:

$$G(i,j,k)=[G_x(i,j,k), G_y(i,j,k), G_z(i,j,k)]$$

è stimato usando le differenze centrali:

$$G_x(i,j,k)=(D(i+dx,j,k)-D(i-dx,j,k))/2\cdot dx$$
$$G_y(i,j,k)=(D(i,j+dy,k)-D(i,j-dy,k))/2\cdot dy$$
$$G_z(i,j,k)=(D(i,j,k+dz)-D(i,j,k-dz))/2\cdot dz$$

con dx, dy, dz le distanze tra i vertici della cella rispettivamente lungo i tre assi X, Y, Z (figura 2.10). Per il calcolo del gradiente ai vertici della cella è necessaria la presenza contemporanea di quattro slice in memoria.

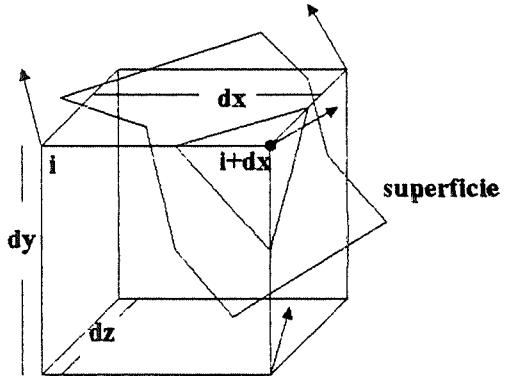



Figura 2.10 I gradienti ai vertici di una cella intersecata dalla superficie.

Interpolando linearmente i gradienti ai vertici della cella sui lati in cui si trova un vertice del triangolo, e quindi normalizzando il vettore ottenuto, è possibile dare una stima della normale richiesta ai vertici dei triangoli. Si osservi infine che tale normale è necessaria se si

vuole applicare ai triangoli ottenuti (come proposto da Lorensen e Cline) la tecnica di shading di Gouraud. Tuttavia ci si può accontentare di applicare ai triangoli una tecnica di shading meno sofisticata quale la "Constant shading" al prezzo di una minore qualità visiva dell'oggetto. Nel caso si adotti la "Constant shading", il passo 6 dell'algoritmo si semplifica notevolmente e si riduce al calcolo di una sola normale unitaria per ogni triangolo. Tale normale unitaria è la normale al piano contenente il triangolo stesso. Se si utilizza la tecnica "Constant shading", non è necessaria la presenza contemporanea in memoria di quattro slice perchè, non dovendo calcolare il gradiente ai vertici della cella, è sufficiente leggere soltanto due slice.

Per ogni coppia di slice si esaminano le (r-1)×(c-1)×(s-1) celle e per ognuna di esse si ricavano i triangoli approssimanti la superficie e le normali ai vertici di questi con il procedimento sopra visto. I triangoli così prodotti sono quindi proiettati sul piano di vista e colorati con intensità di colore dipendente dalle normali ai loro vertici (Gouraud shading).

Per finire notiamo come il processo di determinazione della superficie all'interno della cella così come descritto in precedenza, cioè basato solo sulla valutazione della densità ai vertici, presenta in realtà alcune ambiguità [Whil90]. Una cella è *ambigua* se contiene almeno una faccia ambigua, cioè una faccia in cui i vertici esterni ed interni sono diagonalmente opposti (figura 2.11). In figura 2.12 sono mostrate due triangolazioni egualmente accettabili su una cella ambigua.

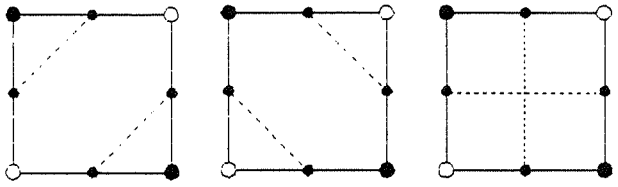


Figura 2.11 - Possibilità di connessione di quattro punti di intersezione

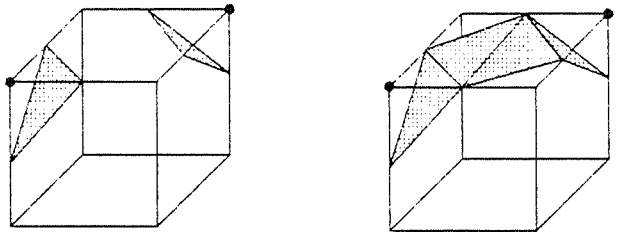


Figura 2.12

In generale non è possibile determinare la corretta topologia di una superficie all'interno di una cella esaminando soltanto i valori dei vertici della cella stessa. Per rimuovere le ambiguità occorre quindi ricavare maggiori informazioni sull'andamento della superficie all'interno della cella. Uno dei metodi più economici [Wivy86] consiste nel confrontare la media dei quattro valori dei vertici della faccia ambigua con la soglia. Il valore medio rappresenta il valore al centro della faccia di una interpolazione bilineare fra i quattro vertici. Naturalmente non sempre la approssimazione del valore della funzione al centro della faccia riesce a sciogliere l'ambiguità. Metodi più sofisticati utilizzano informazioni ricavate all'esterno della cella. Il metodo *quadratic fit* [Whil90] assume che la isosuperficie possa essere rappresentata localmente come una funzione quadratica e ne determina una con la tecnica dei minimi quadrati. Gli zeri della funzione sul piano della faccia rappresentano una

sezione conica che consente di valutare la funzione nel punto "di sella" anzichè nel centro della faccia.

## 2.3. DIVIDING CUBES

L'algoritmo Dividing Cubes (DC), proposto ancora da Lorensen e Cline [Clin88], può essere considerato al confine tra la classe degli algoritmi che ricostruiscono una superficie e quelli, esaminati in seguito, che si limitano a visualizzarla. Il metodo produce infatti punti appartenenti alla superficie da visualizzare e vettori normali alla superficie. La dimensione dei punti della superficie da visualizzare, è scelta in modo che questi ultimi corrispondano il più esattamente possibile ai pixel dello schermo su cui vengono proiettati. La normale in tali punti viene quindi utilizzata per determinare il colore da assegnare al pixel. La differenza principale tra Dividing Cubes e Marching Cubes sta dunque nel tipo di elemento superficiale prodotto: mentre il secondo genera triangoli, il primo genera punti della superficie da visualizzare. Esaminiamo i passi principali di DC.

**Passo 1** Per ogni coppia di slice (m×n) si creano (m-1)×(n-1) celle comprese tra le 2 slice (tale passo è identico al passo iniziale del MC).

**Passo 2**. Si classifica ogni cella costruita al passo 1 secondo il valore scelto della superficie da visualizzare: una cella viene detta interna (esterna) se i valori ai suoi 8 vertici sono tutti minori (maggiori) del valore della superficie (cioè del valore di soglia). Una cella viene detta superficiale se almeno un vertice ha valore maggiore del valore della superficie. Per come è costruita la cella, le celle superficiali sono celle intersecate dalla superficie. Per esse si calcola il vettore gradiente ad ognuno degli otto vertici.

**Passo 3**. Si divide ogni cella superficiale fino alla risoluzione prossima a quella dell'immagine. Questo passo richiede 3 operazioni:

a) Supponendo che le dimensioni delle slice che definiscono le celle siano (n×m) lungo l'asse X e Y, e che le slice disposte lungo l'asse Z siano p, se la risoluzione dello schermo che rappresenta l'immagine è r×r pixel (r>n,m,p) si divide, rispettivamente lungo l'asse X, Y, Z ogni cella superficiale in (r/n)×(r/m)×(r/p) cubi chiamati cubi-pixel e contenuti nella cella superficiale (figura 2.13).

b) Per ogni cubo-pixel calcolato nel passo a) si calcola il valore di densità ai suoi 8 vertici per interpolazione trilineare con i valori di densità ai vertici della cella contenente il cubo-pixel.

c) Si classifica ogni cubo-pixel come nel passo 2 individuando i cubi-pixel superficiali cioè cubi-pixel aventi alcuni valori ai vertici maggiori del valore della superficie ed altri minori.

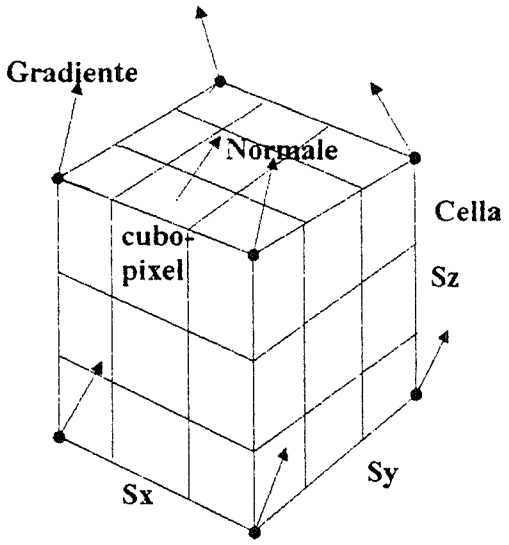



Figura 2.13 Cella e cubi pixel da essa individuati.

**Passo 4**. Per ogni cubo-pixel superficiale si calcola il gradiente al suo centro per interpolazione trilineare con i gradienti ai vertici della cella calcolati nel passo 2. Normalizzando tale gradiente si ottiene così una stima del vettore normale $N=[N_x, N_y, N_z]$ alla superficie passante per il cubo-pixel.

**Passo 5**. Dopo aver impostato le trasformazioni di vista richieste al modello, ogni cubo-pixel superficiale è proiettato sullo schermo individuando (grazie al passo 3) un unico pixel la cui intensità è funzione della normale N al cubo-pixel e della direzione della luce. L'intensità del pixel è calcolata con tecnica "constant shading".

Si Osservi che nel passo 5 la proiezione di ogni pixel sul piano visivo richiede l'applicazione di una tecnica che permetta di rappresentare solo i pixel che sono effettivamente visibili dalla posizione dell'osservatore. Un problema generale che si incontra durante la proiezione di superfici sul piano visivo riguarda infatti la rimozione delle superfici nascoste, cioè di quelle superfici che pur essendo visibili risultano parzialmente o del tutto oscurate da altre superfici interposte tra le prime ed il punto di vista. La tecnica utilizzata in DC al passo 5 per rimuovere le superfici nascoste, e poter così rappresentare solo i pixel visibili, è quella dello Z-buffer. Tale tecnica viene ora esposta brevemente data la sua semplicità di implementazione ed il suo utilizzo in numerosi algoritmi di visualizzazione [Roge85].

L'algoritmo Z-buffer fà uso di 2 strutture dati: una matrice di profondità Zbuf usata per memorizzare la coordinata Z o profondità di ogni pixel visibile dello schermo, ed un frame buffer usato per memorizzare il colore di ogni pixel dello spazio immagine. Inizialmente ogni elemento della matrice Zbuf è impostato al più grande valore di Z rappresentabile, mentre il frame buffer è inizializzato al valore del pixel corrispondente allo sfondo. Per ogni pixel (x,y) appartenente ad un triangolo che deve essere rappresentato sullo schermo, si eseguono i passi 1) e 2):

1) Si calcola la profondità z(x,y) di quel pixel.
2) Se z(x,y)<Zbuf[x,y] allora
   a) Si rimpiazza Zbuf[x,y] con z(x,y).
   b) Si scrive il valore del pixel del poligono nel frame buffer in posizione (x,y).

Quando la condizione 2) è vera, il punto del poligono è più vicino al punto di vista rispetto al punto la cui intensità è correntemente nel frame buffer in posizione (x,y) e perciò nuove profondità ed intensità devono essere memorizzate.

Poichè la rimozione delle superfici nascoste avviene al livello del singolo pixel dello schermo, l'algoritmo Z-buffer permette di rappresentare correttamente le proiezioni sullo schermo dei cubi-pixel prodotti dal DC. L'algoritmo Z-buffer è inoltre semplice da implementare ma presenta lo svantaggio di richiedere una grossa quantità di memoria (pari al numero di pixel dello schermo) necessaria all'implementazione della matrice Zbuf.

## 2.4. SURFACE TRACKING

L'algoritmo Surface Tracking (ST), noto in letteratura anche con il nome di Boundary Detection, delimita l'oggetto da rappresentare con un insieme di superfici elementari tra loro connesse. Assumendo che lo spazio 3D sia composto da voxel di forma cubica, ogni voxel ha sei facce quadrate che condivide con altri 6 voxel adiacenti; ad ogni voxel è inoltre assegnato un valore di densità che specifica il tipo di materiale presente al suo interno. Selezionando un opportuno valore di densità, l'algoritmo ST produce come superfici elementari tutte le facce dei voxel che sono "al confine" della superficie richiesta, cioè quelle facce che separano voxel all'interno della superficie da voxel all'esterno della superficie. Un passo preliminare dell'algoritmo ST è pertanto quello della segmentazione (o thresholding) in cui ad ogni voxel è assegnato un valore 0 o 1 a seconda che il voxel abbia un valore rispettivamente maggiore (esterno alla superficie) o minore (interno alla superficie) del valore della superficie da visualizzare. Dato tale array 3D di valori binari ed una faccia condivisa da un voxel interno detto 0-voxel e da uno esterno detto 1-voxel, ST produce una lista di tutte le facce comuni ad un 1-voxel e ad un 0-voxel che sono in qualche modo connesse alla faccia iniziale [Artz81].

Viene data ora una descrizione dell'algoritmo ST utilizzando i formalismi e le definizioni usate nel lavoro di Gordon e Udupa [Gord89].

Si definisce una scena binaria S=(V,g) con:

$$V=\{v : v=(v_1,v_2,v_3) \text{ con } v_i \text{ intero, } 0\le v_i\le b_i,\ 1\le i\le 3\ \}$$

$b_i$ essendo la dimensione della scena in direzione i e g una funzione che associa un valore 0 o 1 ad ogni voxel in V. Si consideri ora un singolo voxel e si assegni al voxel tre circuiti orientati (R-,G-,B-circuito) che verranno usati per definire l'adiacenza e la connessione delle facce dei voxel (figura 2.14).

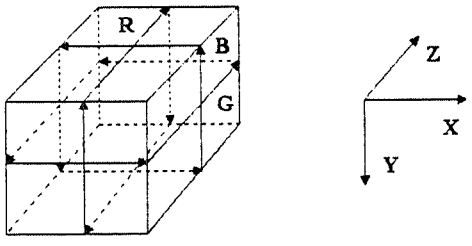



Figura 2.14 Circuiti orientati R-, G-, B- di un voxel.

Le facce ed i lati dei voxel attraverso cui passa un i-circuito vengono chiamate la i-faccia e l' i-lato del voxel, dove i∈(R,G,B). Segue ora una serie di definizioni necessarie per la descrizione dell'algoritmo.

Def.: Due voxel u, v, sono detti $2_i$-adiacenti, i∈(R,G,B), se la loro intersezione è una i-faccia sia per u che per v; essi inoltre sono detti 2-adiacenti se sono $2_i$-adiacenti o $2_j$-

adiacenti o $2_k$-adiacenti per distinti i,j,k∈(R,G,B).

Def.: Una faccia di confine f indicata con f=(u,z) in una scena binaria S è l'intersezione di 2 voxel u e z entrambi in V tali che u è un 1-voxel e z è uno 0-voxel e u e z sono 2-adiacenti.

Def.: Sia f=(u,z) una faccia di confine di S ed una i-faccia di u per i∈(R,G,B); siano $e_1$, $e_2$ due i-lati di f tali che l'i-circuito che passa attraverso f và da $e_1$ ad $e_2$ (figura 2.15) $e_1$ è detto un in-lato di f ed $e_2$ è detto un out-lato di f rispetto al circuito i.

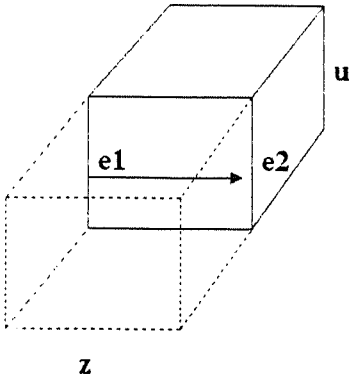



Figura 2.15 In-lato ed out-lato di una faccia f.

La seguente nozione di adiacenza di facce di confine è fondamentale per la descrizione topologica dell'algoritmo ST.

Def.: siano $f_1=(u,z)$, $f_2(u',z')$ due facce di confine di S e i∈(R,G,B), allora $f_1$ è detta $T_i$-adiacente ad $f_2$ $[f_1 T_i f_2]$ se e solo se:

1) $f_1$ e $f_2$ hanno in comune un i-lato: e;
2) e è un out-lato di $f_1$ ed un in-lato di $f_2$ rispetto ad i;
3) esattamente uno dei seguenti casi si verifica:
   3.1) u=u' ed i restanti voxel che condividono e sono 0-voxel (figura 2.16 a).
   3.2) $f_1$ e $f_2$ sono nello stesso piano e u e u' sono dalla stessa parte di questo piano (figura 2.16 b).
   3.3) z=z' (figura 2.16 c).

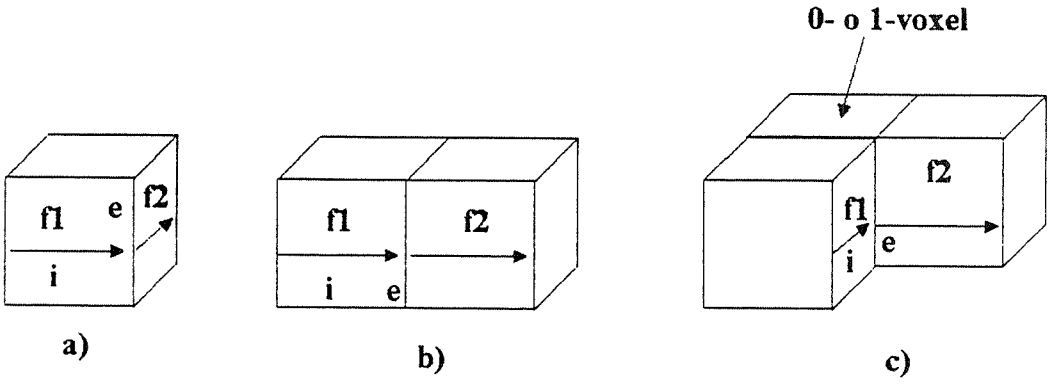



Figura 2.16 Condizioni per la $T_i$-adiacenza di due facce di confine.

Inoltre $f_1$ e $f_2$ sono T-adiacenti [$f_1 T f_2$] se $f_1 T_i f_2$ o $f_1 T_j f_2$ o $f_1 T_k f_2$ per distinti i, j, k∈ (R,G,B).

Def.: Sia S=(V,g) una scena binaria, U l'insieme di tutti gli 1-voxel in V e Z=V-U. Il confine $\delta(u,z)$ è definito come l'insieme di tutte le facce di confine di S.

Il problema di individuare le superfici di confine che delimitano l'oggetto dallo sfondo può essere dunque riformulato dicendo che data una faccia di confine f in $\delta(u,z)$, l'algoritmo ST traccia un insieme (contenente f) di facce di confine di $\delta(u,z)$ opportunamente connesse.

E' stato dimostrato da Herman e Webster [Herm83] che per ogni faccia di confine $f \in \delta(u,z)$:

a) ci sono esattamente 2 facce di confine $f_1$ e $f_2 \in \delta(u,z)$ tali che $fTf_l$ per $1 \le l \le 2$.

b) ci sono esattamente 2 facce di confine $f'_1$ e $f'_2 \in \delta(u,z)$ tali che $f'_l Tf$ per $1 \le l \le 2$.

Data una scena binaria ed una faccia di confine iniziale $f_0$, ST prosegue esaminando le 2 facce che sono T-adiacenti ad $f_0$. Per ogni faccia trovata in questo modo, l'algoritmo procede trovando due nuove facce T-adiacenti e così via. Un meccanismo di "marcatura" è usato per tener traccia delle facce che sono esaminate la prima volta. Quando una faccia è esaminata la seconda volta viene demarcata. Per le proprietà a) e b), $\delta(u,z)$ si può rappresentare come un grafo diretto G=(N,A) in cui una faccia di confine è rappresentata con un nodo con grado uscente 2 (proprietà a)) e grado entrante 2 (proprietà b)); gli archi A sono:

$$A=\{(i,j): i,j \in N, f_i T f_j\}$$

se $f_i$ e $f_j$ sono le facce di confine associate rispettivamente ai nodi i e j di G. L'algoritmo ST può dunque essere schematicamente così riassunto:

INPUT ST: Una scena binaria S ed una faccia di confine $f_0$ (generalmente $f_0$ è specificata dall'utente);
OUTPUT ST: Una lista L inizialmente vuota di facce di confine tra loro T-adiacenti;
STRUTTURE DATI: una coda Q che contiene le facce che devono essere visitate ed una lista M di facce visitate (marcate).

ALGORITMO ST:
**begin**
1) inserisci $f_0$ nella coda Q, nella lista L, e poni 2 copie di $f_0$ in M;
2) **while** Q è non vuota
   a. rimuovi una faccia f da Q;
   b. trova $f_l$ per $1 \le l \le 2$ tali che $fTf_l$;
   c. **for** l=1 **to** 2 **do**
      d1. **if** $f_l \in M$ **then** cancella $f_l$ da M;
      d2. **else** inserisci $f_l$ in Q, in L ed in M;
   **endfor**
**endwhile**
**end ST**

La faccia iniziale $f_0$ è differente dalle altre facce in quanto ogni faccia eccetto $f_0$ è già stata visitata una volta quando viene rimossa da Q nel passo 2(a). Viceversa $f_0$ dovrà ancora essere visitata, questo è il motivo per cui occorrono due copie di $f_0$ in M inizialmente. Herman e Webster [Herm83] hanno dimostrato che ST termina e che al termine L contiene facce di confine (non ripetute) tra loro T-adiacenti.

Nell' esempio che segue viene mostrata l' applicazione dell'algoritmo ST ad un oggetto costituito da tre voxel [Artz81] in cui ogni faccia di confine è identificata da un numero

(figura 2.17).

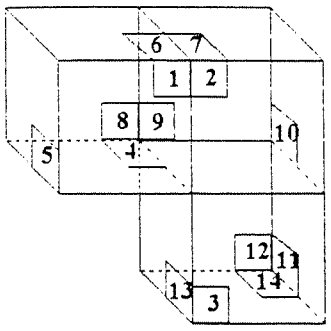



Figura 2.17. Oggetto costituito da tre voxel

Partendo dalla faccia $f_0$ =1 è possibile costruire, come mostrato in precedenza, il grafo diretto rappresentante la superficie dell'oggetto (figura 2.18) e mostrare l'evoluzione dell'algoritmo ST (figura 2.19).

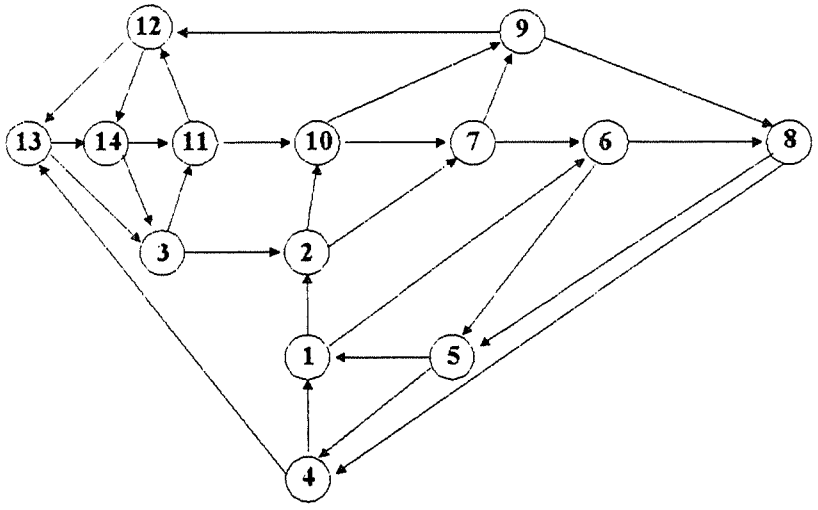



Figura 2.18. Grafo corrispondente all'oggetto di figura 2.15

| Nodo estratto | Q | M | L |
|---|---|---|---|
| | {1} | {1,1} | {1} |
| 1 | {6,2} | {1,1,6,2} | {1,6,2} |
| 6 | {2,5,8} | {1,1,6,2,5,8} | {1,6,2,5,8} |
| 2 | {5,8,7,10} | {1,1,6,2,5,8,7,10} | {1,6,2,5,8,7,10} |
| 5 | {8,7,10,4} | {1,6,2,5,8,7,10,4} | {1,6,,2,5,8,7,10,4} |
| 8 | {7,10,4} | {1,6,2,8,7,10} | {1,6,2,5,8,7,10,4} |
| 7 | {10,4,9} | {1,2,8,7,10,9} | {1,6,2,5,8,7,10,4,9} |

| | | | |
|---|---|---|---|
| 10 | {4,9} | {1,2,8,10} | {1,6,2,5,8,7,10,4,9} |
| 4 | {9,13} | {2,8,10,13} | {1,6,2,5,8,7,10,4,9,13} |
| 9 | {13,12} | {2,10,13,12} | {1,6,2,5,8,7,10,4,9,13,12} |
| 13 | {12,3,14} | {2,10,13,12,3,14} | {1,6,2,5,8,7,10,4,9,13,12,3,14} |
| 12 | {3,14} | {2,10,12,3} | {1,6,2,5,8,7,10,4,9,13,12,3,14} |
| 3 | {14,11} | {10,12,3,11} | {1,6,2,5,8,7,10,4,9,13,12,3,14,11} |
| 14 | {11} | {10,12} | {1,6,2,5,8,7,10,4,9,13,12,3,14,11} |
| 11 | ∅ | ∅ | {1,6,2,5,8,7,10,4,9,13,12,3,14,11} |

Figura 2.19. Evoluzione dell'algoritmo ST applicato all'oggetto di figura 2.15

Da un punto di vista computazionale, il parametro valutativo è rappresentato dal numero di facce di confine. Come si vede dall'algoritmo, ogni faccia di confine è visitata 2 volte perchè ogni nodo di G ha grado entrante 2. Gordon e Udupa hanno proposto un miglioramento dell'algoritmo ST che permette di ridurre il numero di facce di confine visitate 2 volte mediamente ad 1/3 delle facce di confine totali. Ciò permette di ridurre il tempo di esecuzione del programma del 35%; ogni volta che una faccia è visitata, l'algoritmo testa se la faccia è nella lista M dei nodi marcati e questa è l'operazione che richiede maggior tempo. Se ogni faccia è specificata dalle coordinate del voxel cui appartiene e da una direzione (un numero da 1 a 6) e supponendo di disporre di sufficiente memoria per la rappresentazione delle facce, le facce potrebbero essere memorizzate in un array 4D ed il tempo di ricerca sarebbe perciò costante. In questo caso dunque l'algoritmo è lineare nel numero di facce.

Un ultimo problema che riguarda la rappresentazione della superficie dell'oggetto da rappresentare è il colore da assegnare alle facce di confine prodotte dall'algoritmo ST. Questo aspetto verrà trattato in maniera completa nel capitolo 3.

## 2.5. RICOSTRUZIONE DI SUPERFICI DA DATASET VOLUMETRICI AD ALTA RISOLUZIONE

Abbiamo già avuto modo di notare come il poligono rimanga una primitiva grafica molto popolare nelle applicazioni di computer grafica in generale e di rendering volumetrico in particolare. Oltre ad avere una rappresentazione semplice, infatti, la manipolazione dei poligoni (trasformazioni geometriche, rimozione di superfici nascoste, shading) è largamente supportata dall'hardware e dal software grafico in commercio.

Tuttavia, a causa della loro natura lineare, spesso è necessario impiegare migliaia o milioni di poligoni per catturare i dettagli delle geometrie più complesse e modelli di queste dimensioni diventano di difficile gestione anche per le workstation grafiche più moderne. Questo tipo di problemi è in generale amplificato quando si operi su dataset ad alta risoluzione: ad esempio un algoritmo come Marching Cubes (par. 2.2), se applicato a dataset di dimensioni $512^3$ o $1024^3$ (quali quelli impiegati in analisi non distruttiva dei materiali) può produrre da 500k a 2000k triangoli [Schr92].

Una seconda caratteristica dei metodi tipo Marching Cubes che ne limita l'impiego nelle alte risoluzioni è il tempo speso nell'analisi di celle non attraversate dalla isosuperficie di interesse. Una isosuperficie interseca generalmente solo un piccolo sottoinsieme delle celle contenute in un volume, mentre l'algoritmo effettua una visita esaustiva. L'incidenza del costo delle visite infruttuose sul tempo complessivo di elaborazione è evidentemente molto variabile, dipendendo dalle dimensioni complessive del volume, dal numero di celle attraversate dall'isosuperficie e dalle risorse del sistema di calcolo. Ricerche in questo senso sono giunte alla conclusione che tra il 30% e il 70% del tempo complessivo impiegato

nell'estrazione di una isosuperficie è speso nell'analisi di celle vuote [Wilh90a].

Gli approcci alla rappresentazione di oggetti contenuti in modelli volumetrici mediante isosuperfici che tengono conto dei problemi discussi in precedenza possono essere sostanzialmente divisi in due categorie: quelli basati su *approssimazioni progressive* e quelli basati sulla *rimozione di poligoni coplanari*.

Le tecniche che cadono nella prima classe sono accomunate dal fatto di restituire quantità diverse di primitive nelle diverse zone del volume, in dipendenza di una valutazione delle approssimazioni ottenute.

Un campo di applicazione in cui si è fatto uso di tecniche di questo tipo è la rappresentazione di superfici implicite mediante approssimazioni poligonali [Bloom88, Hall90]. Una *superficie implicita* è definita come l'insieme di punti $P$ che soddisfa la funzione implicita $f(P) = 0$. L'approssimazione prodotta consiste in un insieme di poligoni che si intersecano l'un l'altro soltanto in corrispondenza di un lato o di un vertice e che sono posizionati in prossimità della superficie. Gli algoritmi proposti effettuano una partizione dello spazio mediante poliedri (tetraedri o cubi) e valutano la funzione in corrispondenza dei vertici così individuati. Quindi su ogni poliedro viene calcolata l'intersezione della superficie implicita, mediante interpolazione bilineare dei valori ai quattro vertici nel caso di tetraedri o come poligono individuato dai punti di intersezione calcolati sui lati del poliedro, quando questo sia di natura cubica. Anche la valutazione dell'approssimazione dipende dalla primitiva utilizzata per partizionare lo spazio: nel primo caso l'operazione si traduce in una valutazione dei coefficienti di una rappresentazione di Bernstein-Bézier della superficie internamente al tetraedro [Hall90]; nel secondo caso il test riguarda la planarità del poligono e la divergenza delle normali ai vertici con la normale nel centro [Bloom88].

Nell'ambito della ricostruzione di superfici a partire da un numero di punti campionati, il metodo proposto da Schmitt e al. [Schm86] inizia con patch bicubici di Bernstein-Bézier aventi continuità geometrica di tipo $G^1$, per poi dividere in quattro parti quelli che non sono giudicati abbastanza vicini ai punti di campionamento sottostanti. La valutazione dell'approssimazione raggiunta consiste nel calcolo delle distanze fra il patch in esame e ognuno dei punti campionati corrispondenti e nella verifica che esse siano minori di una tolleranza definita dall'utente. Se $Q_{ij}(u, v)$ è il patch in esame e $P$ un punto del dataset, trovare la distanza tra $P$ e $Q_{ij}$ significa trovare i parametri $(u_0, v_0)$ (non necessariamente unici) che minimizzano la distanza euclidea $\|Q_{ij}(u,v)-P\|$; ciò è realizzato efficacemente attraverso tecniche iterative. DeHaemer e Zyda [DeHa91] hanno esteso l'approccio precedente a patch lineari, anche allo scopo di ottenere una riduzione del numero di poligoni restituiti.

Più recente è l'approccio che prevede la rimozione dei poligoni coplanari a seguito della fase di estrazione vera e propria. Un punto di riferimento in questo senso è certamente costituito dall'algoritmo di "decimazione" di Schroeder e al. [Schr92]. L'idea alla base dell'algoritmo è relativamente semplice. Si effettuano diverse passate sull'insieme dei vertici della rete di triangoli. Durante ogni passata ogni vertice è classificato in uno dei modi illustrati in figura 2.20. Tutti i vertici, meno quelli classificati come complessi, sono candidati alla rimozione. La fase di classificazione produce, per ogni vertice candidato, la lista dei triangoli che insistono su di esso e la lista dei relativi vertici. Un vertice semplice $\mathbf{v}$ viene rimosso se la sua distanza dal piano mediano costruito a partire da le normali $\mathbf{n}_i$, i centri $\mathbf{x}_i$ e le aree $A_i$ dei triangoli incidenti

$$\mathbf{N} = \frac{\sum \mathbf{n}_i A_i}{\sum A_i} \qquad \bar{\mathbf{n}} = \frac{\mathbf{N}}{|\mathbf{N}|} \qquad \bar{\mathbf{x}} = \frac{\sum \mathbf{x}_i A_i}{\sum A_i}.$$

è inferiore ad una quantità prestabilita. La distanza dal piano è calcolata come $d = |\bar{\mathbf{n}} \cdot (\mathbf{v} - \bar{\mathbf{x}})|$ (figura 2.21 a).

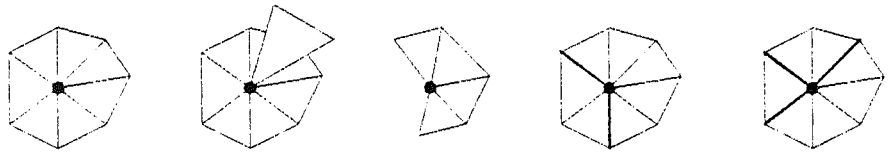



Figura 2.20 - Classificazione dei vertici

Per vertici interni e di frontiera il criterio di decimazione si basa sulla distanza dalla linea definita dai due vertici che formano il lato interno o di frontiera corrispondente (figura 2.21 b).

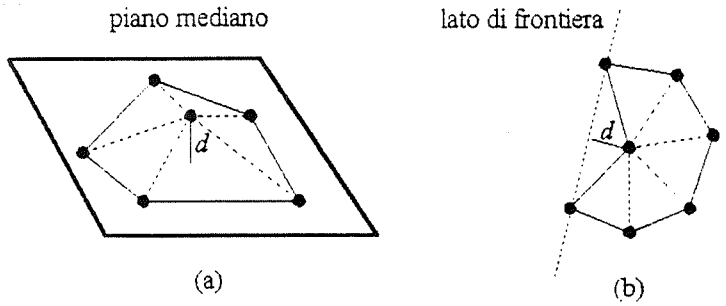



Figura 2.21

Rimuovendo un vertice si crea una discontinuità nella superficie poligonale che viene riempita con una nuova triangolazione. L'algoritmo di triangolazione espressamente sviluppato per operare in tre dimensioni, consiste nel dividere ricorsivamente al lista dei vertici in due parti mediante una linea di separazione individuata da due vertici non adiacenti. Ogni nuova lista così ottenuta è ulterirmente divisa fino al raggiungimento di liste di tre vertici che costituiscono i nuovi triangoli e interrompono così la ricorsione. La scelta della linea di separazione si rivela in effetti una operazione particolarmente complessa: anzichè ricorrere ad algoritmi generali, gli autori ottengono buoni risultati anche con tecniche più semplici che non garantiscono una soluzione per ogni possibile configurazione iniziale dei vertici.

## 2.6. RAY TRACING

Data la complessità di una trattazione, sia pur introduttiva, delle tecniche di visualizzazione basate sul Ray Tracing, questo paragrafo viene suddiviso in tre sottoparagrafi. Il primo descrive un modello di illuminazione/riflessione, che rappresenta un concetto di base per qualsiasi algoritmo di shading (cap. 3) cosi` come per il Ray Tracing. Il secondo descrive l'algoritmo Ray Tracing applicato a modelli geometrici e descritti analiticamente, mentre il terzo sottoparagrafo descrive un'applicazione della tecnica di visualizzazione RT a dataset volumetrici.

### 2.6.1. Un modello di illuminazione e riflessione: il modello di Phong

Un modello di riflessione descrive l'interazione della luce con una superficie in funzione delle proprieta` della superficie e della natura della luce incidente. Diversamente, un modello di illuminazione definisce la natura della luce emanata da una sorgente, la geometria della sua distribuzione e la sua intensità.

L'obiettivo dello studio e dell'utilizzo di questi modelli è quello di rappresentare oggetti tridimensionali su uno schermo bidimensionale in maniera da creare l'illusione della terza dimensione. L'effetto profondità viene simulato dosando opportunamente l'intensità luminosa di ogni pixel rappresentante la superficie dell'oggetto. Il modello standard, nella computer grafica, che stabilisce un compromesso tra il realismo dei risultati ed il costo computazionale è il modello di Phong [Phong75], sebbene esistano modelli più complessi quali quello di Cook-Torrance. Phong considera ogni sorgente di luce come puntiforme e modella la luce riflessa da un punto sulla superficie di un oggetto in termini di una componente diffusiva, di una componente riflessiva e di una terza componente generale di illuminazione "ambientale". L'intensita` di luce riflessa da un punto di una superficie e` dunque una combinazione lineare di queste tre componenti: riflessione diffusa, speculare e luce ambiente.

Per **riflessione diffusa** si intende il paziale assorbimento e la reirradiazione *uniforme* della luce da parte della superficie di un oggetto. Una superficie che si comporta come un perfetto diffusore di luce (superficie molto ruvida), diffonde luce in egual misura in tutte le direzioni (figura 2.22). Ciò significa che "la quantita`" visibile di luce riflessa in maniera diffusa non dipende dalla posizione dell'osservatore.

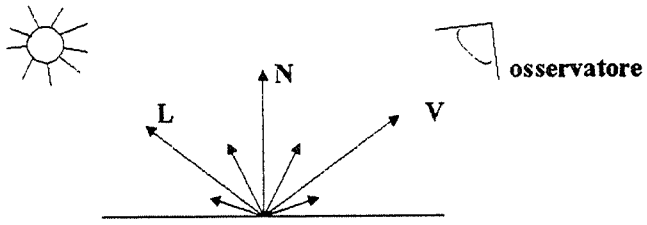



Figura 2.22 Riflessione diffusa su una superficie.

L'intensità $I_d$ di luce riflessa in maniera diffusa, è data dalla legge di Lambert:

$$I_d = I_i K_d (L\ N) = I_i\ K_d \cos(\beta) \quad 0 \leq \beta \leq \pi/2\ .$$

$I_i$ e` l'intensità della sorgente di luce (e quindi della luce incidente), $\beta$ e` l'angolo tra la normale unitaria alla superficie N ed il vettore unitario L che rappresenta la direzione della luce incidente (cioé la direzione della linea che connette il punto sulla superficie con la sorgente di luce). $K_d$ è invece una costante tra 0 e 1, il cui valore dipende dalla natura del materiale e dalla lunghezza d'onda della luce incidente.

La **luce ambiente** e` il risultato di multiple riflessioni diffuse prodotte da qualsiasi altra superficie presente nella scena (si pensi per esempio ai muri di una stanza). La luce ambiente e` incidente su una superficie da tutte le direzioni. Questo significa che le superfici visibili, ma in ombra

rispetto alla sorgente di luce, vengono comunque illuminate dalla luce ambiente e quindi rese effettivamente visibili.

Phong modella questo tipo di illuminazione come un termine costante. La combinazione della luce diffusa e della luce ambiente porta alla seguente equazione:

$$I = I_a K_a + I_i K_d (L\ N)$$

dove $I_a$ e' l'intensita' della luce ambiente e $K_a$ e' un coefficiente costante di riflessione ambiente.

Per certi angoli di vista, una superficie lucente riflette tutta la luce incidente indipendentemente dal suo coefficiente di riflessione. Questo fenomeno, chiamato **riflessione speculare**, produce una caratteristica "macchia" di luce riflessa sulla superficie dell'oggetto.

Se si considera una superficie perfettamente speculare, illuminata da una sorgente puntiforme, tutta la luce incidente su di essa sarà riflessa lungo una precisa direzione, la direzione speculare, e sarà visibile solamente quando questa direzione coincide con quella di vista. L'intensità di luce riflessa specularmente dipende quindi dalla direzione di vista.

La direzione speculare forma un angolo β con la normale N alla superficie, dove β e' l'angolo tra la luce incidente e la normale (figura 2.23) si noti che R, I, N sono complanari e che la direzione di riflessione è un vettore dato da:

$$R=I+2N\cos(\beta).$$

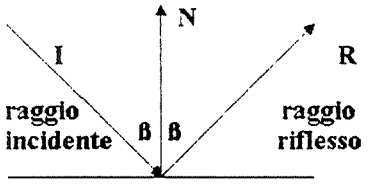



Figura 2.23 Riflessione speculare.

Nelle scene reali, difficilmente la riflessione speculare e' perfetta e la luce riflessa puo' essere vista anche da direzioni vicine a quella del fascio riflesso. L'intensità di luce riflessa viene così modellata :

$$I_s = I_i K_s (R\ V)^n = I_i\ K_s (\cos^n(\Phi))$$

dove Φ e' l'angolo tra la direzione speculare R e la direzione di vista V (R e V sono vettori unitari) e $K_s$ e' il coefficiente di riflessione speculare (una costante che dipende dal materiale) (figura2.24).

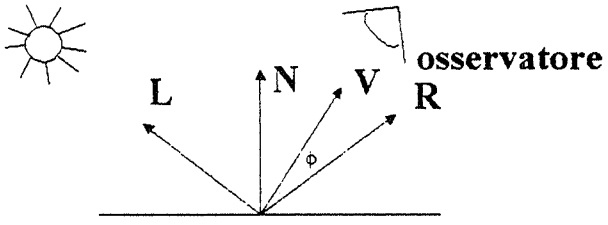



Figura 2.24 Visibilità della luce riflessa.

Per una superficie perfettamente speculare n e' infinito e la luce, come gia' accennato, puo' essere vista solo quando R e V coincidono (cioe' Φ=0). Per una superficie piu' ruvida n e' portato a valori piu' bassi e la luce speculare puo' essere vista anche per angoli maggiori di zero. Si noti inoltre che, a differenza della riflessione diffusa, la luce riflessa specularmente e' dello stesso colore della luce incidente.

Combinando tra loro i termini sin qui discussi si ottiene la seguente equazione:

$$I_{Phong} = I_a K_a + I_i [K_d (L N) + K_s (R V)^n ]/(r+k)$$

dove la divisione per il termine (r+k) è stata aggiunta per considerare la distanza r tra la superficie e la sorgente di luce (k e' una costante arbitraria). Cio' assicura che non venga assegnata la stessa intensita' a superfici identiche ma a distanze differenti.

La riflessione non e' l'unico effetto che deve essere considerato. Se nella scena appaiono oggetti trasparenti, devono essere considerati anche effetti di **rifrazione**. Un raggio che colpisce un oggetto parzialmente o interamente trasparente viene rifratto a causa del cambiamento di direzione della luce dovuto all'attraversamento di mezzi differenti (figura2.25).

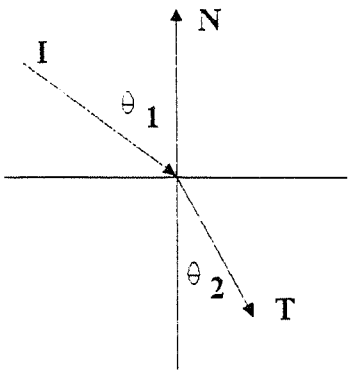



Figura 2.25 Rifrazione della luce su un oggetto trasparente.

Gli angoli di incidenza e rifrazione sono messi in relazione dalla legge di Snell:

$$\sin \theta_1 / \sin \theta_2 = \eta_2 / \eta_1$$

dove $\eta_1$ e $\eta_2$ sono gli indici di rifrazione dei due mezzi, mentre $\theta_1$ è l'angolo di incidenza e $\theta_2$ è l'angolo di rifrazione.

Il raggio incidente I e quello rifratto (o trasmesso) T sono complanari con N. Il raggio trasmesso e' dato da:

$$T = 1/\eta \, I - (\cos(\theta_2) - 1/\cos(\theta_1)) N$$

dove $\eta = \eta_2/\eta_1$ e $\cos(\theta_2) = [1 - 1/\eta^2 (1 - \cos^2(\theta_1))]^{1/2}$

Calcoli per il raggio riflesso e quello rifratto sono usualmente eseguiti nello spazio oggetto utilizzando tecniche Ray Tracing [Styt91].

### 2.6.2. Ray Tracing su modelli geometrici

Il Ray Tracing rappresenta una delle più complete tecniche di simulazione di un modello di illuminazione/riflessione presente oggi nella computer grafica. Questa tecnica, applicata a modelli geometrici, calcola il valore colore di ogni pixel generando un "raggio visivo", cioe' un raggio passante per tale pixel ed avente direzione dipendente dalla posizione dell'osservatore, nel caso di proiezioni prospettiche, o dalla direzione di vista nel caso di proiezioni parallele (figura 2.26). Il raggio attraversa la scena colpendo gli oggetti che incontra lungo la sua direzione e il primo oggetto colpito risulta chiaramente visibile dal particolare pixel associato al raggio [Glas89].

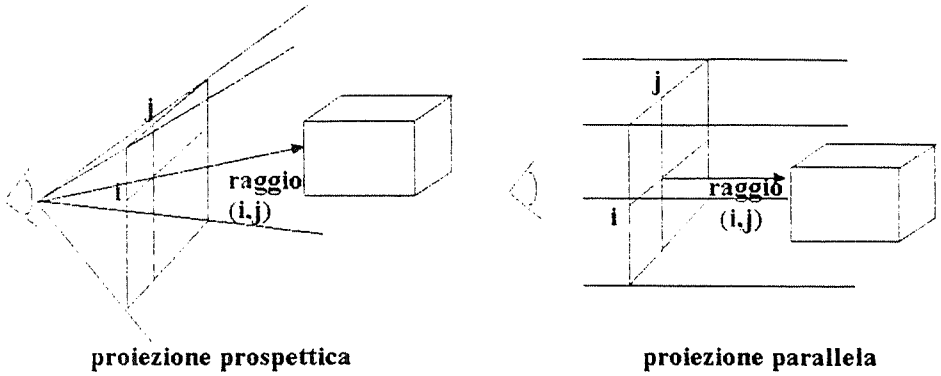



Figura 2.26 Proiezione prospettica e parallela di un oggetto.

L'intensita' di luce che arriva dalla superficie lungo tale raggio e' il risultato di un'illuminazione "diretta" e di una "globale". L'illuminazione diretta è rappresentata dalla luce incidente sulla scena (e quindi sul punto di intersezione trovato) che proviene direttamente dalla sorgente di luce. Nel modello di Phong, precedentemente discusso, i termini usati per l'illuminazione diretta sono: $I_{Phong}$=luce diffusa + luce speculare + luce ambiente.

L'illuminazione diretta, cioé $I_{Phong}$, garantisce pero' solo un contributo "locale" all'illuminazione di un punto sulla superficie. Se la luce che entra nella scena e' incidente su un punto di una superficie dopo aver avuto interazioni con altri oggetti, allora questa illuminazione è classificata come "globale". L'illuminazione globale nasce, di fatto, dalla interazione della luce emanata dalla sorgente con oggetti riflettenti e semitrasparenti. Il contributo globale all'illuminazione di un punto su una superficie (punto ricavato intersecando il raggio con un oggetto), si ottiene tracciando "indietro" un altro raggio nella direzione di riflessione ed uno nella direzione di rifrazione alla ricerca di ulteriori intersezioni [Watt89].

In figura 2.27 un raggio primario (raggio 1) e' tracciato dalla posizione di vista attraverso un pixel sullo schermo in un ambiente contenente sfere semitrasparenti (oggetti cioe' che riflettono e rifraggono la luce). Il colore del pixel equivale al colore attribuito al raggio 1. Tale colore ha tre contributi: il colore "locale" dovuto all'illuminazione diretta sul punto di intersezione (ricavabile come $I_{Phong}$) e due contributi globali ricavabili dal raggio di riflessione (raggio 2) e di rifrazione (raggio 3).

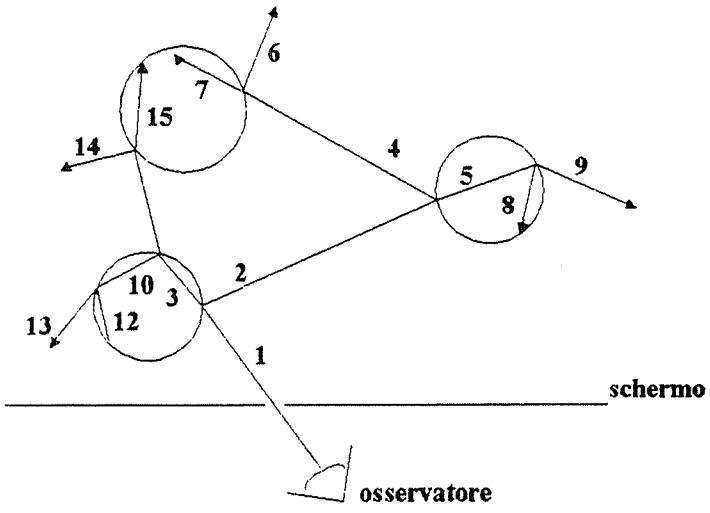



Figura 2.27 Tracciamento di un raggio con la tecnica Ray Tracing.

Il contributo del raggio 2 puo` essere ricavato tracciando questo raggio indietro alla sua prossima intersezione (se c'e`) con un oggetto. Il colore attribuibile al raggio 2 su questa intersezione e` a sua volta calcolato componendo i tre contributi locale, globale riflesso e globale trasmesso (o rifatto). I contributi di riflessione e rifrazione sono ottenuti tracciando rispettivamente i raggi 4 e 5 e cosi` via. Il contributo di luce rifratta al raggio 1 viene calcolato nella stessa maniera tracciando indietro il raggio 3 alla sua prossima intersezione e cosi` via.

In figura 2.28 è mostrata l'analisi necessaria ad ogni intersezione di un raggio con una superficie.

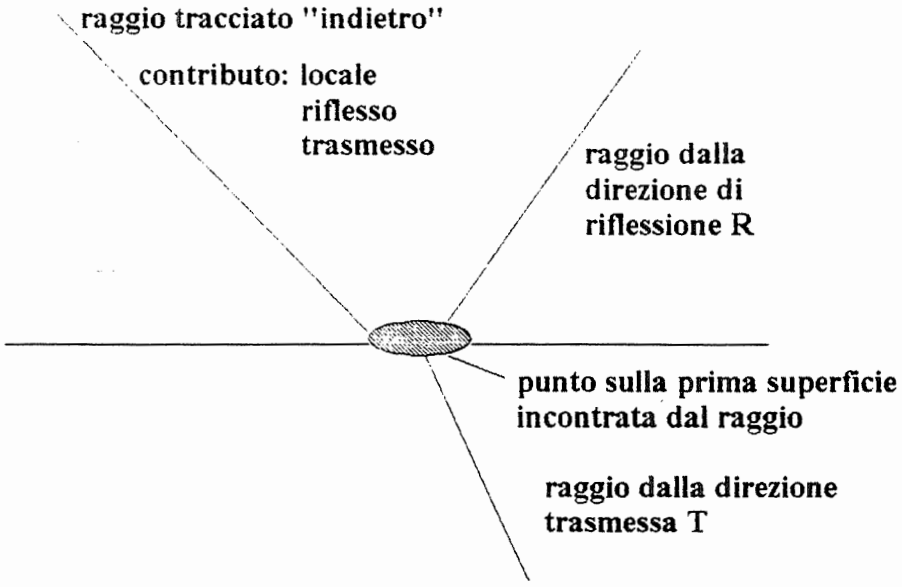



Figura 2.28

La procedura Trace ray riportata di seguito mostra i compiti principali che devono essere eseguiti da una semplice procedura Ray Tracing ricorsiva. Al fine di scoprire il colore del pixel corrispondente al raggio 1 (figura 2.27), deve essere eseguita una chiamata con parametri che rappresentano il punto di partenza e la direzione di questo raggio, insieme ad un valore di profondità che rappresenta la profondità raggiunta e che viene confrontato con una profondità massima, intesa come costante, oltre la quale i raggi non vengono più tracciati.

**Trace_ray** (posizione_vista, posizione_pixel-posizione_vista, profondità, colore_pixel);

Il risultato **sara`** ritornato dal parametro: colore_pixel.

```
Procedure Trace_ray(partenza,direzione:vettori;profondità:intero;var colore:colori);
var punto_intersezione, direzione_riflessa, direzione_trasmessa: vettori;
    colore_locale,colore_riflesso,colore_trasmesso:colori;
begin
   {interseca il raggio con tutti gli oggetti e trova il punto di intersezione (se c'é) che é più
    vicino alla partenza del raggio.};
   if {nessuna intersezione} then colore :=colore_di_sfondo
   else
   begin
      colore_locale:={contributo di illuminazione locale al punto di intersezione}
      if profondità=max_profondità then
```

```
        begin
            colore_riflesso:=nero;colore_trasmesso:=nero;
        end
        else
        begin
            {calcola la direzione del raggio riflesso};
            Trace_ray (punto_intersezione, direzione_riflessa, profondità+1,
                        colore_riflesso);
            {calcola la direzione del raggio trasmesso};
            Trace_ray (punto_intersezione, direzione_trasmessa, profondità+1,
                        colore_trasmesso);
        end;
        Combina (colore_locale ,peso_locale_per_la_superficie, colore_riflesso,
                    peso_di_riflessione_per_la_superficie, colore_trasmesso,
                    peso_di_rifrazione_per_la_superficie, colore);
    end;
end;{Trace_ray}
```

I tre parametri di "peso" della procedura **Combina** dipendono dalle caratteristiche fisiche delle superfici.

Ogni volta che un raggio interseca una superficie viene prodotto, in generale, un raggio riflesso ed uno rifratto alla ricerca di nuove intersezioni. Un ray tracer semplice e convenzionale puo' quindi risultare computazionalmente troppo costoso, dato l'alto numero di intersezioni che deve calcolare. Parecchie tecniche per migliorare questo algoritmo hanno riscosso larghi consensi. Uno degli schemi impiegati per velocizzare il calcolo delle intersezioni consiste nel "racchiudere" un oggetto, o una serie di oggetti da rappresentare, in un solido semplice, la cui intersezione con una linea sia facilmente calcolabile (ad esempio una sfera). Se il raggio non interseca il solido, significa che non puo' intersecare l'oggetto in esso contenuto. Questo metodo evita il continuo test di intersezione con oggetti complessi. Un altro schema si basa sul fatto che quando un raggio e' riflesso (o rifratto) su una superficie, la sua intensita' e' attenuata dal coefficiente di riflessione (o rifrazione) attribuito a tale superficie (i valori dei coefficienti di riflessione\rifrazione sono dipendenti dal tipo di superficie intersecata). Ad ogni intersezione, il coefficiente di attenuazione (sempre tra 0 e 1) viene moltiplicato per quello precedentemente calcolato, diminuendo maggiormente l'intensita' del raggio. L'idea è quella di interrompere il tracciamento dei raggi secondari qualora il coefficiente cumulativo di attenuazione diventi minore di una soglia prefissata, oltre la quale il contributo di ogni raggio risulterebbe ininfluente [Watt89].

Diamo infine un accenno ai "raggi ombra". Questi raggi aggiuntivi, tracciati da ogni punto di intersezione verso la sorgente di luce, determinano se la superficie di un oggetto e' in ombra rispetto alla sorgente di luce. Un oggetto A e' nell'ombra di un altro oggetto B, rispetto ad una data sorgente di luce, se il raggio ombra interseca l'oggetto B tra la superficie di A e la sorgente di luce. Se l'oggetto B e' interamente opaco, allora il contributo di illuminazione locale (illuminazione diretta) e' ridotta al solo termine ambiente. Una attenuazione per $I_{local}$ viene invece calcolata se l'oggetto B è semitrasparente [Styt91].

### 2.6.3. Ray Tracing su modelli volumetrici

Tecniche di visualizzazione Ray Tracing (RT) sono state proposte anche per la rappresentazione di dati volumetrici classificati sia voxel- che cell-based. Punto di forza dell'algoritmo RT, applicato alla visualizzazione volumetrica, è il poter generare immagini con simulazione di effetti di trasparenza ed evidenziazione delle superfici contenute, senza richiedere l'attraversamento completo del volume dei dati. Infatti dalla definizione stessa dell'algoritmo RT, deriva che il tracciamento di ogni raggio ha termine non appena il valore colore del pixel associato sia stato sufficientemente definito.

L'alta qualità delle immagini, quale quella a cui ci ha abituato l'applicazione del Ray Tracing a modelli geometrici, è generalmente considerata non necessaria; è infatti ritenuta sufficiente la simulazione di effetti di trasparenza senza il controllo della rifrazione e della sola componente diffusiva. Eliminare quindi la generazione dei raggi secondari per il calcolo delle componenti di

riflessione speculare ed eventualmente anche quelli necessari al calcolo delle ombre, riduce notevolmente la complessità, di per sè considerevole, del metodo RT [Mont90].

Usando un approccio RT, come già accennato nel paragrafo 2.5.2, possono essere generate sia proiezioni parallele che prospettiche. Tuttavia quando l'obiettivo è la rappresentazione di dati volumetrici sono generalmente ritenute sufficienti le più semplici proiezioni parallele. Questa affermazione è giustificata da due motivi: la prospettiva non aumenta la percezione di oggetti piccoli ed irregolari, ed inoltre le proiezioni parallele lasciano inalterate le distanze ed altre informazioni geometriche che permettono misurazioni dirette sull'immagine rappresentata [Reyn87].

Ogni oggetto da visualizzare nella scena è "individuato" tramite una classificazione a soglia: ciascun voxel con valore minore o uguale alla soglia prefissata viene considerato interno, altrimenti è considerato esterno all'oggetto di interesse. Questa classificazione produce regioni 3D di voxel il cui "strato" più esterno rappresenta la superficie di isovalore desiderata (si noti che il valore di soglia rappresenta l'informazione di base per qualsiasi algoritmo dedicato alla rappresentazione di isosuperfici).

Così come per il Ray Tracing applicato su modelli geometrici e sinteticamente definiti, la tecnica RT applicata a dati volumetrici e dedicata alla visualizzazione di isosuperfici, stima, per ogni pixel sullo schermo, il punto sulla superficie dell'oggetto intersecato dal raggio emesso dal punto sullo schermo [Tuy 84].

Quando il tracciamento di ogni raggio viene realizzato su un modello voxel-based il valore di ogni voxel attraversato dal raggio viene confrontato con le soglie relative agli oggetti da rappresentare (figura 2.29).

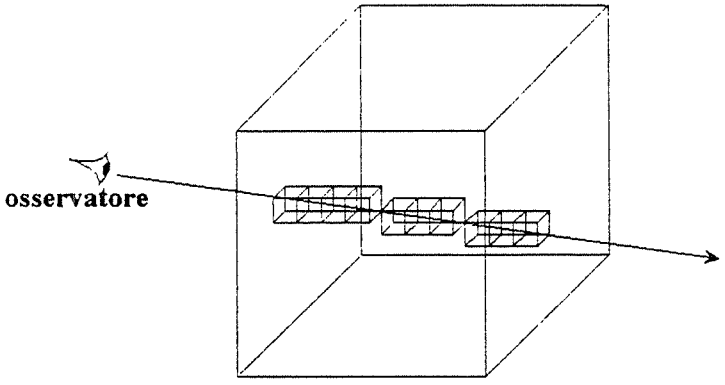



Figura 2.29 Tracciamento di un raggio per RT applicato a modelli voxel-based.

Non appena si passa da un voxel con valore minore ad uno con valore maggiore (o viceversa) di una soglia $f_v$ relativa ad un oggetto v, significa che è stata trovata l'intersezione del raggio con la superficie dell'oggetto v. Per quel punto di intersezione (e quindi per quel raggio) viene calcolata una intensità di colore che dipende dalla normale assunta per la superficie in quel punto. Il valore di tale normale può essere stimato utilizzando la tecnica di shading nello spazio oggetto "binary gradient shading" che verrà discussa nel paragrafo 3.1.6. Se il raggio attraversa esclusivamente i voxel il cui valore è maggiore di ciascuna soglia prefissata ed esce dai limiti dello spazio 3D in cui sono definiti gli oggetti da rappresentare, significa che esso non interseca nessuna superficie e pertanto al pixel ad esso associato viene assegnato il colore relativo allo sfondo.

L'algoritmo RT può anche essere applicato a spazi volumetrici cell-based. In tal caso i dati volumetrici vengono ricampionati su k locazioni ugualmente distanziate lungo il raggio interpolando trilinearmente i valori degli otto dati del grigliato che circondano ciascuna locazione campione (figura 2.30).

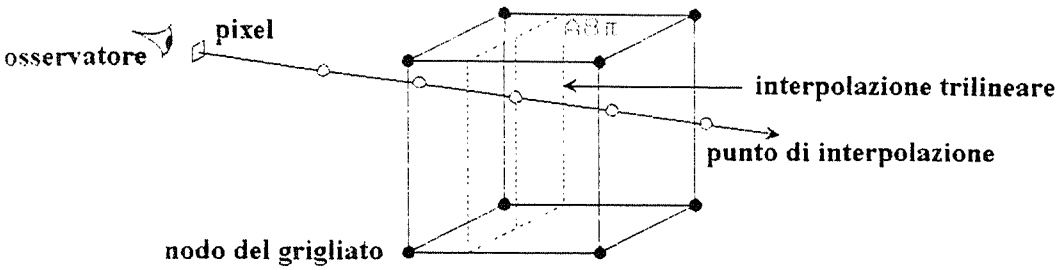



Figura 2.30 Tracciamento di un raggio per RT applicato a modelli cell-based.

I cambiamenti di valore dei punti ricampionati attraversati dal raggio vengono interpretati come cambiamenti di densità ed il passaggio da un valore minore di una soglia $f_v$ ad uno maggiore (o viceversa) viene interpretato come l'attraversamento della superficie dell'oggetto v associata alla soglia $f_v$ [Mont90]. E' possibile poi per ogni locazione campione calcolare una stima del vettore normale all'isosuperficie interpolando i vettori normali ai vertici della cella. In modo analogo al caso del RT applicato a spazi volumetrici voxel based, si determina quindi il colore associato al raggio e dunque al pixel ad esso relativo.

Per completezza d'informazione è importante sottolineare come la tecnica di visualizzazione RT possa essere applicata non solo alla visualizzazione di specifiche superfici individuate da particolari valori di soglia ma anche alla visualizzazione dell'intero volume di dati ("volume rendering") in cui ogni voxel contribuisce all' immagine finale. Le tecniche di questo tipo classificano ogni voxel del dataset associando ad esso un valore di opacità ed un colore. I colori e le opacità di tutti i voxel attraversati vengono quindi "integrati" per ottenere il colore finale del pixel associato al raggio. Per ulteriori approfondimenti in questo senso si rimanda ai lavori reperibili in letteratura [Levo88, Levo90, Ney90].

## 2.7. TECNICHE PROIETTIVE

Con il nome di tecniche proiettive si fa riferimento a due tecniche di visualizzazione dette Back-To-Front (BTF) e Front-to-Back (FTB). Entrambe le tecniche sono caratterizzate dal fatto che ogni immagine del volume di dati è ottenuta visitando opportunamente l'intero volume e proiettando i voxel significativi sul piano di vista senza la necessità di dover estrarre la superficie dell'oggetto.

### 2.7.1. Back-to-Front

L'algoritmo BTF, proposto da Frieder ed alt. [Frie85] è basato su un attraversamento dell'array 3D costituente il modello iniziale ottenuto facendo variare gli indici di slice, riga e colonna in ordine di distanza decrescente rispetto all'osservatore. L'insieme delle slice è quindi esaminato partendo da quella più distante dall'osservatore per finire con quella più vicina. A sua volta ogni slice è esaminata, proiettando i rispettivi voxel sul video facendo variare sia l'indice di riga che quello di colonna in ordine crescente o decrescente in funzione della posizione dell'osservatore rispetto agli assi di riferimento. Più precisamente, se la posizione dell'osservatore è (x,y,z) in un sistema di riferimento come quello in figura 2.31 allora:

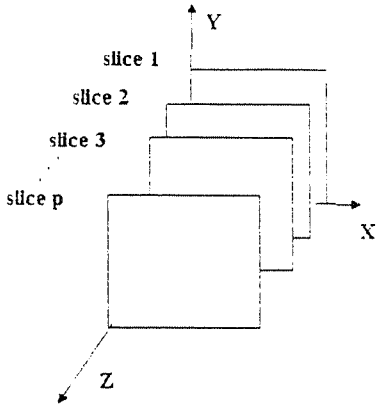



Figura 2.31 Sistema di riferimento contenente le slice del modello

* l'ordine di scansione delle slice è crescente se z ≥0, decrescente se z<0 ;
* l'ordine di scansione delle righe delle slice è crescente se y≥0, decrescente se y<0;
* l'ordine di scansione delle colonne delle slice è crescente se x≥0, decrescente se x<0.

In figura 2.32 si può vedere un'applicazione al caso 2D del metodo BTF in cui una slice binaria è proiettata sullo schermo. Assumendo che i voxel A, B, C, D debbano essere proiettati sullo schermo, i voxel del modello devono essere attraversati per indice di riga (asse y) decrescente e di colonna (asse x) crescente. Sia x che y possono essere scelti come l'indice che cambia più velocemente. Così le sequenze di voxel A, B, C, D (x cambia più velocemente) o B, D, C, A, (y cambia più velocemente) risultano entrambe in una corretta rappresentazione della scena. Il motivo di tale correttezza risiede nel fatto che se un voxel (o parte di esso) con coordinate (x,y) è oscurato da un'altro voxel con coordinate (x',y'), allora x≤x' e y≥y' così che proiettando (x,y) prima di (x',y') si ottiene un'immagine corretta.

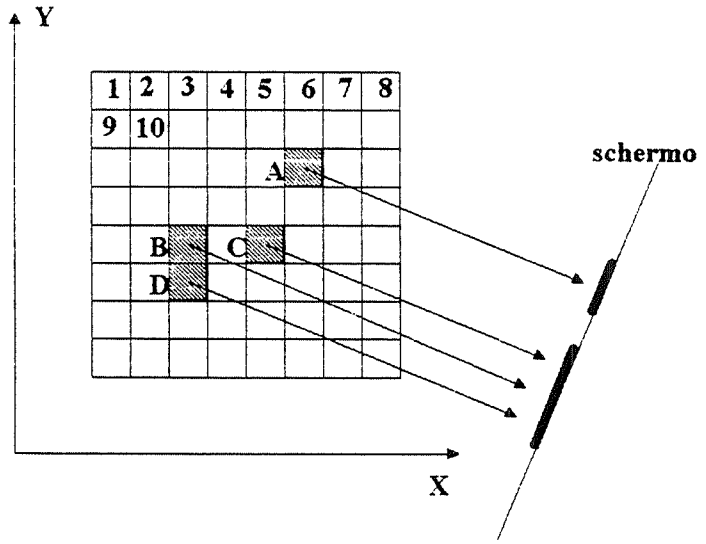



Figura 2.32 Applicazione del metodo BTF ad un caso 2D.

Bisogna notare che prima di applicare la tecnica BTF, occorre un passo preliminare di classificazione che permette di assegnare valore 1 ai voxel appartenenti all'oggetto da rappresentare e 0 a quelli non appartenenti. Una volta individuati i voxel da proiettare sullo schermo, il punto dello schermo in cui, scelta una particolare vista, viene proiettato un qualsiasi voxel (x,y,z) è calcolato utilizzando le ben note tecniche di proiezione (moltiplicazione di matrici 4x4 col vettore coordinate

del voxel [Fole82]). Ogni proiezione di un voxel richiede dunque nove moltiplicazioni. L'illuminazione da assegnare ai pixel su cui si proiettano i voxel viene poi stabilita utilizzando il metodo "Image based gradient shading" (par 3.2.1) che utilizza sia la distanza del voxel dalla sorgente di luce che la normale alla superficie passante per quel voxel.

Un indubbio vantaggio della tecnica BTF, oltre alla sua semplicità, è dovuto al fatto che quando il volume di dati è rappresentato da una lista di slice, qualsiasi sia la vista scelta dall'utente, i voxel possono essere attraversati leggendo una slice alla volta. Ciò permette all'algoritmo di mantenere una sola slice in memoria centrale riducendo così l'occupazione di memoria. Su ogni slice i voxel sono quindi visitati scegliendo opportunamente l'ordine di visita sui 2 assi della slice.

Il problema delle superfici nascoste infine, che in algoritmi di estrazione delle superfici richiede l'utilizzo di tecniche quali lo z-buffer (par. 2.3), viene automaticamente risolto dal procedimento stesso di scansione BTF dei voxel del modello.

### 2.7.2. Front-to-Back

La tecnica FTB proposta da Reynolds ed alt. [Reyn87] ricostruisce l'immagine attraversando il volume di voxel con ordine inverso a quello BTF (prima i voxel più vicini poi i più lontani) per poter elaborare così solo quei voxel che risultino effettivamente visibili (in figura 2.33 i voxel vengono letti nell'ordine: D, C, B, A). Il vantaggio delle tecniche FTB è quindi che l'elaborazione dei voxel la cui proiezione coincide con pixel il cui colore risulta già definito (perchè un altro voxel è già stato proiettato su quel pixel) risulta del tutto inutile.

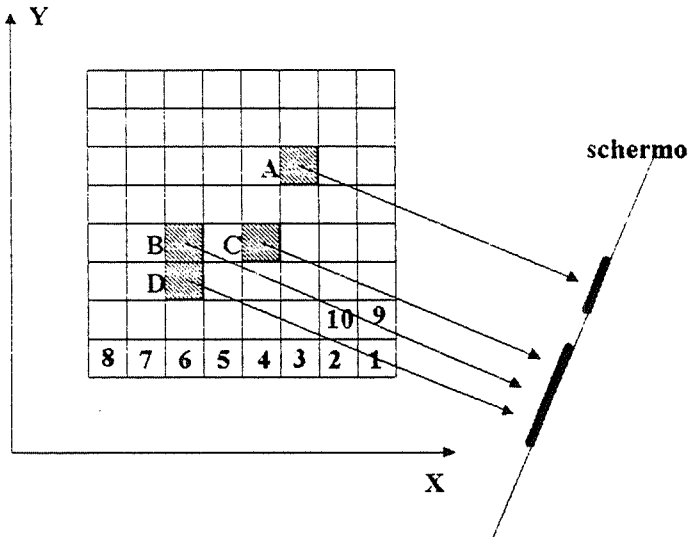



Figura 2.33 Applicazione del metodo FTB ad un caso 2D.

D'altro lato, a sfavore della tecnica FTB gioca il fatto che occorre gestire strutture dati per stabilire quali regioni dello schermo sono già state trattate; ciò comporta una maggiore complessità algoritmica rispetto alla tecnica BTF. Un'assunzione fondamentale dell'algoritmo FTB è che le trasformazioni di vista applicate all'oggetto siano tali che linee orizzontali nello spazio dell'oggetto siano trasformate in linee orizzontali quando proiettate sullo spazio dello schermo. Ciò significa che è imposta una limitazione nella possibilità di scelta nell'orientamento delle viste: in questo tipo di proiezione sono ammesse solo le viste che trasformano le righe di voxel del volume (voxel ad ordinata e profondità z costanti) in righe di pixel paralleli all'asse X' dello spazio immagine (figura 2.34). Tale condizione non è limitativa dal punto di vista dell'analisi dei dati in quanto si dimostra [Reyn87] che una proiezione generica può comunque essere ottenuta come composizione di una proiezione del tipo precedentemente descritto (che si ottiene ruotando l'oggetto prima intorno all'asse $X_1$ e poi intorno all'asse $Y_1$) e di una successiva rotazione dell'immagine ottenuta intorno all'asse $Z_1$.

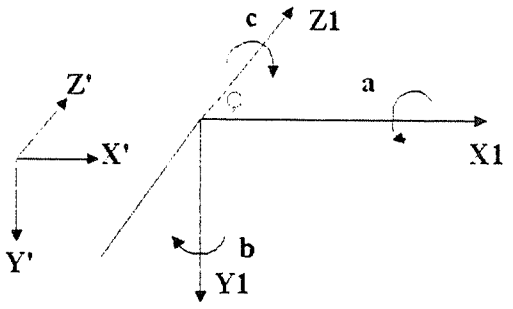



Figura 2.34 Sistema dello spazio immagine (X',Y',Z') e dello spazio oggetto($X_1$,$Y_1$,$Z_1$).

Il dover gestire solo viste del tipo descritto semplifica molto la gestione dei risultati intermedi. Specificata una qualunque orientazione dell'oggetto rispetto a $X_1$,$Y_1$,$Z_1$ è possibile trovare 3 rotazioni a, b, e c che, eseguite nell'ordine dato, portano l'oggetto nell'orientazione specificata. Per come è organizzato l'algoritmo FTB, si eseguono prima le rotazioni a e b in modo da trasformare segmenti paralleli a $X_1$ in segmenti che quando proiettati sul piano X'Y' rimangono paralleli all'asse X'. A questo punto, applicando il FTB, segmenti di linee dell'oggetto così proiettati possono essere facilmente "fusi" con segmenti di linee dell'immagine precedentemente proiettati a cui sono paralleli. Come conseguenza di tale procedimento la rimozione delle superfici nascoste e l'individuazione dei pixel da illuminare diventano operazioni relativamente semplici. Una volta ottenuta l'immagine corrispondente alle rotazioni a e b, la rotazione c è ottenuta ruotando l'immagine 2D sullo schermo.

Si assume che in ogni istante in memoria centrale sia memorizzata una sola slice. Supponendo di visitare i voxel in ordine FTB il problema accennato all'inizio consiste nel testare ogni pixel su cui si proietta un voxel per vedere se esso è già illuminato (lit) o no (unlit). La soluzione adottata da Reynolds ed alt. è di usare una struttura dati dinamica: il *dynamic screen*, per rappresentare segmenti di pixel unlit su ogni linea dello schermo. Assumendo che l'ampiezza dell'immagine sia N×N pixel, il *dynamic screen* è costituito da un array di puntatori YCHAIN[0...N-1] in cui ogni elemento YCHAIN[L] punta ad una lista rappresentante segmenti unlit della linea L dello schermo. Ogni elemento della lista ha 3 campi: MIN e MAX per il minimo e massimo valore della coordinata x del segmento unlit e LINK che contiene un puntatore all'elemento successivo della lista (figura 2.35).

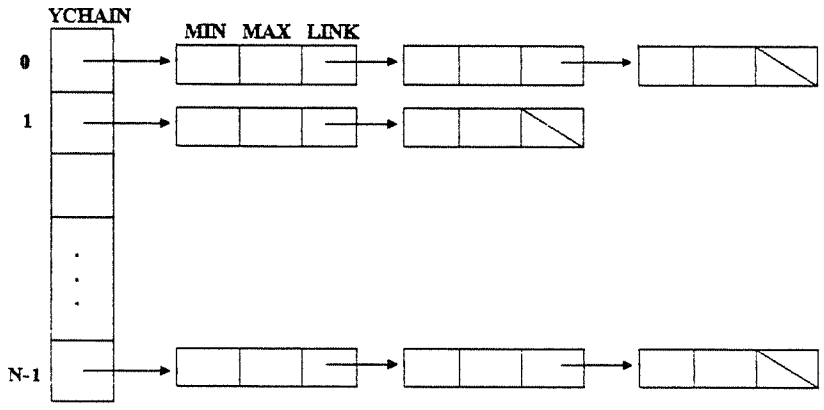



Figura 2.35 Struttura *Dynamic screen*

Il vantaggio di questa tecnica è che sono testati solo i pixel unlit e che man mano che lo schermo

viene illuminato, questi diventano sempre meno. Sotto le condizioni viste l'algoritmo FTB si può sintetizzare nei seguenti punti:

1. Leggi le slice in ordine FTB. Per ogni slice, scorri le righe in ordine FTB.
2. Per ogni riga di voxel dell'oggetto, trova la linea L dello schermo su cui tale riga si proietta.
3. Scorri la riga in ordine FTB trasformando i segmenti dell'oggetto nello spazio immagine ottenendo così i segmenti dell'oggetto proiettati e "fondendoli" con i segmenti dell'immagine nell'appropriata linea dello schermo.
4. Se l'operazione di "fusione" risulta in pixel che devono essere disegnati (lit) si scrive l'appropriato valore del pixel nell'array dell'immagine e si aggiorna opportunamente il *dynamic screen*.

Viene mostrata ora una descrizione formale della procedura Front-to-Back.

```
procedure Front_to_Back;
begin
   for k:=1 to numero_di_slice do
   begin
      s:= NEXT_SLICE; { in ordine front_to_back }
      for j:=1 to numero_di_righe do
      begin
         r:= NEXT_ROW; { in ordine front_to_back }
         if ( r contiene segmenti dell'oggetto ) then
         begin
            L:= {linea dello schermo su cui si proietta r};
            if (L contiene pixel unlit) then
            MERGE ( SEGMENT (L); TRANSFORMED_SEGMENTS (r) );
         end
      end
   end
end; { Front_to_Back }
```

Gli argomenti della procedura MERGE sono: una riga dei segmenti dell'oggetto trasformato ed una linea di segmenti di pixel unlit su cui proiettare la riga. MERGE scorre i segmenti dell'immagine già proiettati e dell'oggetto da proiettare.

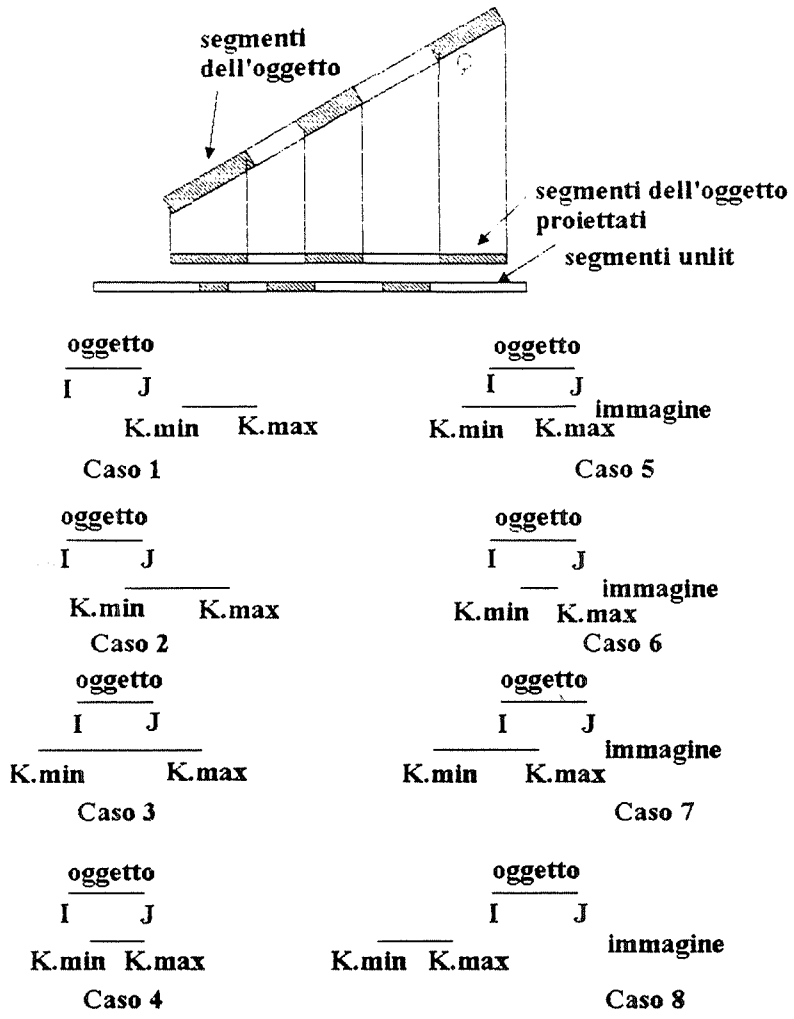



Figura 2.36 Proiezione di segmenti dell'oggetto e dell'immagine.

Siano I e J i pixel agli estremi di un segmento-oggetto e K un segmento-immagine di pixel unlit (figura 2.31); MERGE confronta I e J con gli estremi di K K.MIN e K.MAX. A seconda del risultato del confronto ci sono 8 azioni distinte che possono essere intraprese come mostrato di seguito.

| Caso | Condizione | Azione |
|---|---|---|
| 1 | J<K.min | NEXT_OB_SEG |
| 2 | J<K.min and<br>K.min≤J≤K.max | PAINT(K.min,J);<br>CHANGE(J+1,K.max,K);<br>NEXT_OB_SEG |
| 3 | I>K.min and<br>J<K.max | PAINT(I,J);<br>SPLIT(I-1,J+1,K);<br>NEXT_OB_SEG |
| 4 | I≤K.min and<br>J=K.max | PAINT(K.min,K.max);<br>DELETE(K);<br>NEXT_OB_SEG; |

NEXT_IM_SEG;

| | | |
|---|---|---|
| 5 | I>K.min and J=K.max | PAINT(I,J);<br>CHANGE(K.min,I-1,K);<br>NEXT_OB_SEG;<br>NEXT_IM_SEG; |
| 6 | I≤K.min and J>K.max | PAINT(K.min,K.max);<br>DELETE(K);<br>NEXT_IM_SEG; |
| 7 | J>K.max and K.max≥I≥K.min | PAINT(I,K.max);<br>CHANGE(K.min,I-1,K);<br>NEXT_IM_SEG |
| 8 | I>K.max | NEXT_IM_SEG |

Il risultato dei confronti può essere raggruppato in 4 differenti categorie:

1. un segmento ne precede completamente un altro (casi 1 e 8). In questi casi non si deve fare nessun cambiamento al segmento dell'immagine.
2. Il segmento dell'oggetto "copre" un estremo del segmento dell'immagine di pixel unlit (casi 2,5,7). I pixel coperti sono illuminati ed il segmento dell'immagine è accorciato cambiando K.MIN o K.MAX. Queste operazioni sono eseguite dalle procedure PAINT e CHANGE.
3. L'intero segmento dell'immagine è coperto dal segmento dell'oggetto (casi 4,6), il segmento dell'immagine è disegnato e cancellato dalla lista dalla procedura DELETE.
4. Il segmento dell'oggetto cade all'interno del segmento dell'immagine, in questo caso i pixel coperti sono disegnati ed il segmento dell'immagine è diviso in due segmenti corrispondenti alle parti non coperte. L'operazione di divisione è eseguita dalla procedura SPLIT che aggiunge un nuovo elemento alla lista ed aggiorna i campi degli elementi precedenti. Vediamo infine le procedure utilizzate nel metodo FTB.

```
Procedure PAINT(I,J);
begin {disegna i pixel da I a J}; end;

Procedure DELETE(k);
begin {cancella il segmento K}; end;

Procedure CHANGE(I,J,K);
begin {modifica il segmento K}; end;

Procedure SPLIT(I,J,K);
begin {divide il segmento K in due nuovi segmenti}
    new(K');
    K'.min:=J;
    K'.max:=K.max;
    K'.LINK:=K.LINK;
    K.max:=I;
    K.LINK:=K';
end;
```

L'intensità da attribuire ai pixel risultanti dell'immagine finale, viene stabilita utilizzando il metodo "Image based gradient shading" (par. 3.2.1) che stima le normali alla superficie permettendo così una rappresentazione sufficientemente realistica dell'oggetto. Tale metodo viene analogamente utilizzato per attribuire l'intensità ai pixel nella tecnica BTF. L'algoritmo FTB si è dimostrato molto veloce in applicazioni di tipo medico specialmente grazie alla facilità con cui risolve il problema delle

superfici nascoste, alla semplicità con cui si effettua l'operazione di rasterizzazione sullo schermo (rappresentando solo segmenti paralleli alle linee dello schermo) e alla compressione dei dati fornita dalla codifica run-lenght della struttura *dynamic screen*.

## 2.8. VALUTAZIONI

Gli aspetti che devono essere presi in considerazione nella valutazione di una tecnica di visualizzazione volumetrica, sono da ricercare nell'occupazione di memoria richiesta, nel tempo impiegato per rappresentare l'isosuperficie e nella qualità visiva dell'immagine ottenuta. Tali aspetti sono in parte dipendenti dalla tecnica stessa di visualizzazione impiegata ed in parte dall'hardware grafico a disposizione e dalla tecnica di illuminazione della superficie adottata. Per quanto riguarda l'hardware grafico, si può notare infatti che in un algoritmo quale il Marching Cubes, il tempo di esecuzione impiegato è strettamente legato al numero di interpolazioni e dunque al numero di triangoli complessivamente generati. Tale tempo dipende dunque dalla dimensione del dataset ed è indipendente dall'ampiezza dello schermo su cui vengono proiettati i triangoli. Il numero di triangoli prodotti resta quindi costante all'aumentare della risoluzione dello schermo. Viceversa in una tecnica di visualizzazione quale Dividing Cubes è il numero di punti della superficie che devono essere proiettati a determinare il tempo di esecuzione dell'algoritmo e tale numero di punti dipende sia dall'ampiezza dell' isosuperficie sia dalla risoluzione dello schermo. La dipendenza del tempo di esecuzione dall'ampiezza dello schermo è ancora più evidente nel Ray Tracing applicato a modelli volumetrici in cui infatti occorre tracciare un raggio nella scena 3D per ogni pixel dello schermo.

Oltre ad influenzare il tempo di esecuzione dell'algoritmo, un hardware grafico specializzato è in grado anche di ridurre l'occupazione di memoria richiesta. Nel caso di Dividing Cubes per esempio si è visto come la rimozione delle superfici nascoste sia effettuata utilizzando la tecnica dello Z-buffer in cui è necessario memorizzare una matrice di profondità di ampiezza pari al numero di pixel dello schermo. L'occupazione della memoria centrale risulterà dunque notevolmente ridotta se si dispone di un buffer di memoria dedicato alla memorizzazione di tale matrice di profondità. Viceversa nel caso si adotti una tecnica proiettiva (Back-to-front o Front-to-back) il problema delle superfici nascoste è automaticamente risolto dal metodo di scansione dei voxel adottato e non è dunque richiesta nessuna memoria aggiuntiva per l'implementazione dello Z-buffer.

Per quanto riguarda la tecnica di illuminazione adottata, come si vedrà nel capitolo successivo, esistono parecchie tecniche di illuminazione che forniscono una qualità visiva dell'immagine migliore all'aumentare del costo computazionale della tecnica. Pur essendo la qualità visiva un parametro di valutazione difficilmente quantificabile, è senz'altro un parametro fondamentale che deve essere preso in considerazione e che dipende in gran parte dalla tecnica di illuminazione adottata.

Per quanto brevemente accennato, gli aspetti che intervengono nella valutazione di una tecnica di visualizzazione volumetrica sono molteplici e non facilmente stimabili. Tali tecniche risultano dunque tra di esse difficilmente comparabili considerando anche il fatto che, come già affermato, alcune procedono prima ad una ricostruzione di superfici (Triangolazione di contorni planari, Marching Cubes, Dividing Cubes, Surface Tracking) e poi utilizzano una tecnica di illuminazione per visualizzare tali superfici; altre invece effettuano una visualizzazione diretta delle isosuperfici (Ray Tracing applicato a modelli volumetrici e Tecniche proiettive).

## 3. TECNICHE DI SHADING

I diversi algoritmi di visualizzazione passati in rassegna nel capitolo precedente permettono di individuare i voxel visibili, o le superfici approssimate da poligoni, secondo una particolare vista e quindi permettono di delimitare l'oggetto o gli oggetti di interesse dallo sfondo. Tuttavia affinché nell'immagine 2D dello schermo sia presente anche l'informazione sulla terza dimensione (profondità nello spazio) è necessario assegnare ai pixel rappresentanti l'oggetto una colorazione che tenga conto della curvatura locale della superficie di cui il voxel o il poligono corrispondente fa parte rispetto alla sorgente luminosa ed all'osservatore. Le tecniche di illuminazione che permettono di rendere nell'immagine 2D l'effetto di curvatura delle superfici nello spazio 3D modulando opportunamente il colore prendono il nome di tecniche di **shading**. Il costo computazionale delle tecniche di shading è via via crescente man mano che tali tecniche diventano più sofisticate, cioè tentano di rappresentare l'oggetto creando in modo realistico l'illusione della profondità sullo schermo bidimensionale.

Come già visto nel paragrafo 2.5.1 presentando il modello di illuminazione di Phong, le tre componenti fondamentali per il calcolo dell'illuminazione sono: la normale alla superficie, il punto di vista e la posizione della sorgente luminosa. Una difficoltà che si incontra nel calcolo dello shading di modelli volumetrici è la mancanza di informazione esplicita sull'andamento delle superfici degli oggetti rappresentati e quindi sul valore che la normale alla superficie assume in qualsiasi voxel o poligono posto sulla superficie. Nel caso di modelli volumetrici quindi il valore della normale deve essere approssimato, derivandolo con particolari tecniche dai dati stessi.

Nelle tecniche di shading qui presentate sarà fatto riferimento al sistema di coordinate (X,Y,Z) in cui gli oggetti sono definiti, come spazio oggetto e quello (X',Y',Z') in cui essi sono rappresentati come spazio immagine (figura 3.1).

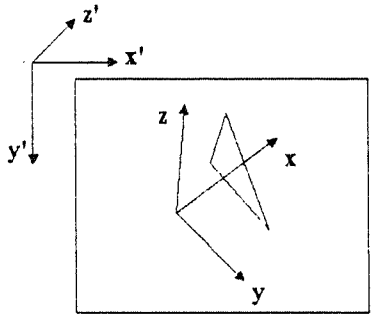



Figura 3.1 Spazio oggetto e spazio immagine.

Lo spazio oggetto e lo spazio immagine sono messi in relazione da una trasformazione di coordinate. Si suppone inoltre che l'immagine della superficie da rappresentare sia ottenuta mediante una proiezione ortografica parallela, cioè un punto P della superficie dell'oggetto è proiettato in un punto Q dello schermo in modo che il segmento QP sia sempre ortogonale allo schermo. Se P è un punto visibile sulla superficie, viene usata l'espressione: "lo shading assegnato a P" per intendere l'intensità luminosa del punto Q sullo schermo su cui P si proietta. A tal fine occorre stimare la normale al punto P. I differenti metodi di shading discussi stimano le normali in modi differenti, ma fanno tutti uso della seguente formula ricavata dal modello di illuminazione di Phong, in cui per semplicità si trascura la componente di illuminazione dovuta alla riflessione speculare:

$$S(P) = \left[ I_a K_a + \frac{(I_{max} - I_a K_a)(D - d)}{D} N(\theta) \right]_L^M$$

$$\text{con} \quad [v]_L^M = \begin{cases} L & \text{se } v < L \\ v & \text{se } L \le v \le M \\ M & \text{se } v > M \end{cases}$$

dove:
M e L sono costanti definite dall'utente;
S(P) è lo shading assegnato al punto P;
$I_aK_a$ rappresenta la luce ambiente;
$I_{max}$ è la massima illuminazione che si ottiene quando $\theta = 0$ (tipicamente uguale a 256 se il display può rappresentare 256 differenti tonalità di grigio).
$N(\theta)$ è una funzione di $\theta$ che simula le proprietà di riflessione diffusa in funzione dell'angolo $\theta$ tra la normale N stimata alla superficie e la direzione della luce L (figura 3.2).
D è la distanza dall'oggetto alla quale l'illuminazione diventa zero (D può essere considerato il diametro di una sfera che racchiude l'oggetto).
d è la distanza di P dalla sorgente di luce.

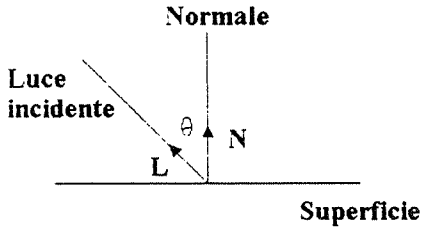



Figura 3.2

Pur rimanendo costante la formula (1), le tecniche di shading differiscono per il modo in cui è stimata la normale (e dunque il valore di $\theta$) o per la scelta della funzione $N(\theta)$. Per semplicità di implementazione si assume che i raggi della luce siano ortogonali allo schermo per non generare ombre.

Le tecniche di shading possono essere classificate in due categorie [Gord85]:

1) Tecniche di shading nello spazio oggetto, in cui ad ogni voxel o poligono da rappresentare sono associate informazioni riguardanti i voxel adiacenti o, nel caso dei poligoni, riguardanti il poligono stesso o quelli ad esso adiacenti.
2) Tecniche di shading nello spazio immagine, che assegnano uno shading all'immagine usando informazioni disponibili nello spazio immagine dopo che una pre-immagine è stata calcolata e che sono state trasformate le coordinate da spazio oggetto a spazio immagine.

## 3.1. TECNICHE DI SHADING NELLO SPAZIO OGGETTO

### 3.1.1. Constant shading

In questo metodo, la normale all'oggetto in un punto P è stimata usando la normale al poligono approssimante la superficie cui P appartiene. Nel caso in cui si rappresentino le facce dei voxel, per ogni particolare vista solo tre distinti valori della normale sono possibili e nell'immagine risultante compaiono molte discontinuità e "scalettature" dipendenti solo dalla particolare rappresentazione e

non dall'andamento effettivo della superficie del corpo rappresentato. A tutti i punti di una faccia di confine viene assegnata in maniera costante l'intensità fornita dalla (1) applicata al suo punto centrale P con

$$N(\theta)= [\cos(\theta/n)]^p \qquad (2)$$

dove i valori di p e di n sono scelti empiricamente in seguito a sperimentazioni per migliorare la qualità delle immagini (come valori di riferimento possono valere p=0.6, n=2). La presenza di discontinuità nella superficie rappresentata con questo metodo è evidenziata in figura 3.3 in cui si vede che il Constant-shading assegna maggiore luminosità alla faccia di confine $f_1$ che alla $f_2$ sebbene la superficie reale dell'oggetto sia lineare.

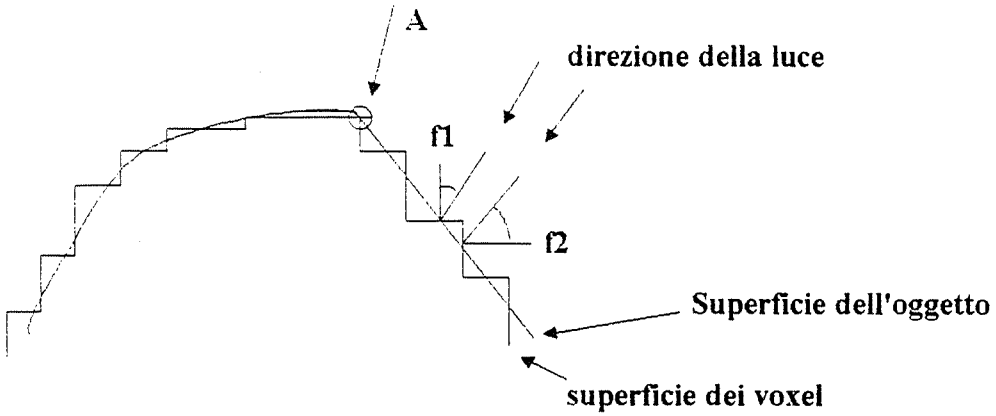



Figura 3.3 Presenza di "scalettature" con il metodo Constant shading.

Nel caso in cui la superficie sia approssimata da triangoli contigui differentemente orientati, vale sempre la (2), però la discontinuità tra due triangoli adiacenti risulta senz'altro molto meno accentuata rispetto al caso in cui si rappresentino le facce dei voxel che presentano solo 3 possibili orientazioni. Anche in questo caso, per quanto siano piccoli i triangoli da rappresentare, nell'immagine risultante si noterà che ogni triangolo che approssima la superficie risulta distinguibile dagli altri poiché ognuno ha un'intensità lievemente differente dai suoi vicini. La differenza di intensità di facce adiacenti è inoltre accentuata dall'effetto "banda di Mach" che consiste in una esaltazione del cambio di intensità ad ogni lato dei poligoni in cui è presente una discontinuità di intensità di illuminazione. Questo fenomeno è dovuto ad un particolare fenomeno fisiologico legato ai recettori presenti nell'occhio umano ed è stato scoperto appunto da E. Mach nel 1865 [Fole82]. Come conseguenza di ciò si ha un'accentuazione dei lati che separano i triangoli adiacenti e quindi una rappresentazione della scena complessiva in cui tali lati sono distinguibili.

### 3.1.2. Image-based contextual shading

Questa tecnica è stata sviluppata per superare i problemi causati dalle false discontinuità e scalettature nella rappresentazione delle facce dei voxel. L'idea è quella di assegnare ai punti sulla faccia del voxel un'intensità che dipenda non solo dall'orientazione della faccia stessa, ma anche dall'orientazione delle 4 facce che condividono con essa un lato (facce confinanti). La presenza di false discontinuità avviene, come visto, quando $\theta$ è piccolo per una particolare faccia ma non per una faccia confinante (figura 3.3). Le facce visibili in cui l'angolo tra la normale e la direzione della luce sia minore di 45° verranno chiamate facce critiche. Le altre facce visibili sono dette non critiche. La tecnica di shading Image-based contextual shading (IBCS) riduce gli effetti di false discontinuità tra facce critiche e non critiche. Sia $\theta_0$ l'angolo tra la normale ad una faccia f e la direzione di luce e $\theta_1$, $\theta_2$, $\theta_3$, $\theta_4$ gli angoli analogamente definiti per le quattro facce adiacenti. A tutti i punti di una faccia f si assegna uno shading determinato dalla formula (1) valutata al centro P della faccia con:

$$N(\theta)=\Sigma_{i=0..4}\, w_i\, [\cos(\theta_i/n)]^p,\ \Sigma_{i=0..4}\, w_i=1 \qquad (3)$$

I pesi $w_i$ dipendono da f e dal numero di facce visibili adiacenti ad f che sono critiche:
* Se f è critica allora viene dato peso 0 alle facce confinanti critiche;
* Se f è non critica allora viene dato peso 0 alle facce confinanti non critiche.

Lo shading assegnato ad una faccia non critica circondata da facce non critiche è perciò lo stesso di quello assegnato con il Constant-shading, mentre lo shading di una faccia critica circondata da facce non critiche risulterà più uniforme. Chen ed alt. [Chen85] stabiliscono i valori di $w_0$ e $w_i$ ($0 \le i \le 4$) in modo sperimentale ed in funzione del numero di facce critiche confinanti. Il principale problema che si incontra con questa tecnica di shading può essere illustrato facendo riferimento alla figura 3.3, in cui si vede che a destra del punto A ci sono sia facce critiche che facce non critiche mentre a sinistra del punto A ci sono solo facce visibili non critiche. Così a sinistra del punto A la tecnica IBCS è identica al Constant-shading perché la (3) diventa uguale alla (2) essendo $w_0=1$, $w_1=w_2=w_3=w_4=0$, mentre a destra del punto A si applica la tecnica IBCS ottenendo un'immagine più uniforme. Viceversa se la luce proviene dall'alto a sinistra di figura 3.3 (piuttosto che dall'alto a destra) si avrà un'immagine più uniforme a sinistra del punto A. Lo svantaggio dell'IBCS può dunque essere riassunto notando che l'apparenza dell'oggetto cambia in modo innaturale in maniera dipendente dalla sua orientazione relativamente alla direzione della luce.

### 3.1.3. Normal-based contextual shading

Questo metodo, riportato in [Chen85], è stato sviluppato per risolvere il problema dell'IBCS sopracitato. L'idea è quella di stimare la normale al centro P di ogni faccia in base all'orientazione della faccia stessa e delle facce confinanti e usando le formule (1) e (3) con $\theta$ posto uguale all'angolo tra la normale stimata in P e la direzione della luce. Il modo in cui è stimata la normale in P è illustrato in figura 3.4.

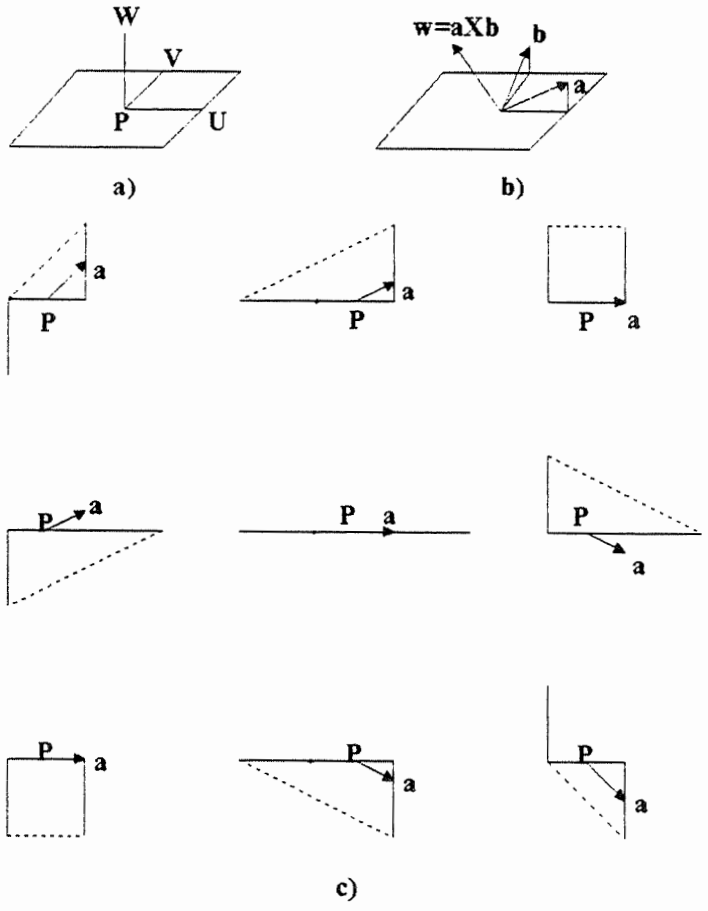



Figura 3.4 Stima del vettore normale nella tecnica IBCS.

Si fissa un sistema di coordinate U,V,W con l'origine in P ed in cui W punta al di fuori dell'oggetto e U e V sono paralleli ai lati della faccia di cui si cerca la normale (figura 3.4a). Il vettore normale *w* è definito come il prodotto vettoriale dei vettori a (nel piano U-W) e b (nel piano V-W) (figura 3.4b). Ognuno dei due vettori a e b può avere una delle 5 possibili direzioni dipendentemente dall'orientazione delle facce confinanti (figura 3.4c). Come si vede, ci sono 9 possibili disposizioni delle facce confinanti per ognuno dei due assi U e V del sistema. In totale ci sono dunque 81 possibili disposizioni delle facce confinanti. Poiché per ogni direzione di a esistono 5 direzioni di b in totale si ottengono 25 possibili orientazioni di *w* che è la stima del vettore normale alla faccia nel punto P.

Il vantaggio di tale tecnica è che a differenza del metodo IBCS l'apparenza dell'oggetto non cambia in modo apprezzabile quando l'oggetto è ruotato. Lo svantaggio tuttavia risiede nel fatto che per ogni faccia f, delle 81 possibili orientazioni delle facce confinanti soltanto 25 producono valori distinti di *w*.

### 3.1.4. Gouraud shading

A differenza dei metodi precedentemente descritti che assegnano la stessa intensità a tutti i punti di un poligono, la presente tecnica e quella descritta nel paragrafo successivo (Phong shading) calcolano separatamente l'illuminazione per ogni punto del poligono che si proietta su di un pixel del video e quindi forniscono una descrizione qualitativamente migliore dell'oggetto. Il Gouraud shading [Gour71] è un metodo semplice ed economico che può essere schematizzato in 4 passi:

passo 1: per ogni poligono da rappresentare viene calcolato il vettore normale al poligono stesso.

passo 2: per ogni vertice V dei poligoni si calcola la normale unitaria $N_v$ al vertice del poligono, come media delle normali dei poligoni che condividono quel vertice ($N_v=(N_1+N_2+N_3+N_4)/4$ nell'esempio di figura 3.5) nel caso queste siano disponibili, o mediante approssimazioni come quelle viste in questi paragrafi quando, come nella visualizzazione volumetrica, non si abbiano informazioni esplicite sull'andamento della superficie da rappresentare.

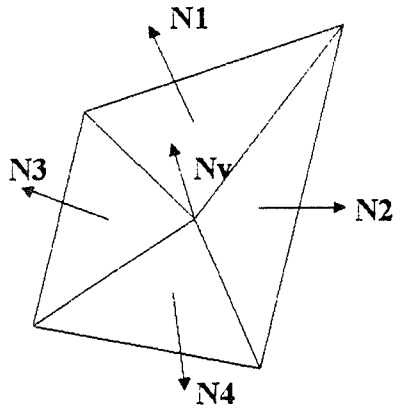



Figura 3.5 Calcolo delle normali ai vertici nel Gouraud shading.

passo 3: per ogni vertice V dei poligoni si calcola un'intensità S(V) utilizzando la formula (1) e $N(\theta)=N\times L$ con L vettore unitario in direzione della luce e N la normalizzazione del vettore $N_v$ calcolato al passo 2. In generale si può tuttavia utilizzare un qualunque modello di shading per ricavare l'intensità S(V).

passo 4: ogni poligono viene rappresentato sullo schermo calcolandone lo shading pixel per pixel. Per ogni linea orizzontale dello schermo (*scan line*) interessata alla proiezione del poligono, si calcolano le intensità $I_a$ e $I_b$ all'intersezione della scan line corrente con i lati del poligono per interpolazione lineare con le intensità ai vertici del poligono calcolate al passo 3 (figura 3.6).

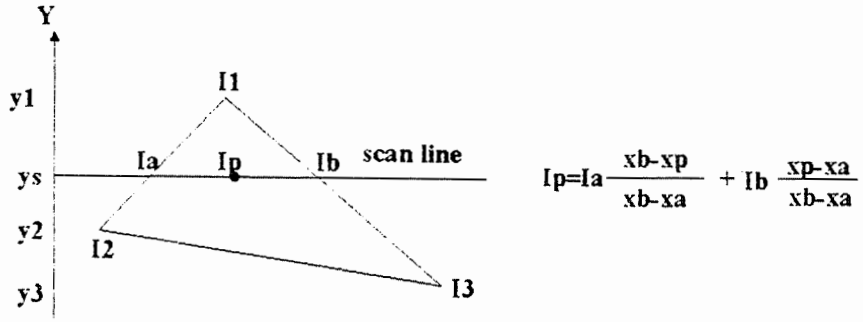



Figura 3.6 Calcolo dell'intensità nel Gouraud shading

L'intensità $I_p$ lungo ogni punto (pixel) della scan line è quindi calcolato per interpolazione lineare tra le intensità $I_a$ e $I_b$.

La tecnica Gouraud shading rappresenta un indubbio miglioramento rispetto alle tecniche precedenti, poiché produce un'intensità che varia con continuità da poligono a poligono riuscendo a dare l'impressione di una superficie curva, ma non riesce ad eliminare una percezione di discontinuità tra i poligoni a causa dell'effetto banda di Mach.

### 3.1.5. Phong shading

Il metodo Phong shading [Phong75], interpola il vettore normale alla superficie piuttosto che le intensità lungo una scan line. Le normali ai vertici del poligono sono calcolate come nel Gouraud shading (nell'esempio di figura 3.7 le normali ai vertici sono $N_1$, $N_2$, $N_3$). Per ogni scan line nel poligono si valutano per interpolazione lineare i vettori normali agli estremi di ogni linea ($N_a$ e $N_b$ in figura 3.7). Questi due vettori sono poi usati per interpolare $N_p$. Le equazioni per ricavare $N_a$, $N_b$ ed $N_p$ sono [Watt89]:

$$\begin{aligned} N_a &= [1/(y_1-y_2)]\,[N_1(y_s-y_2) + N_2(y_1-y_s)] \\ N_b &= [1/(y_1-y_3)]\,[N_1(y_s-y_3) + N_3(y_1-y_s)] \\ N_p &= [1/(x_b-x_a)]\,[N_a(x_b-x_p) + N_b(x_p-x_a)] \end{aligned} \qquad (4)$$

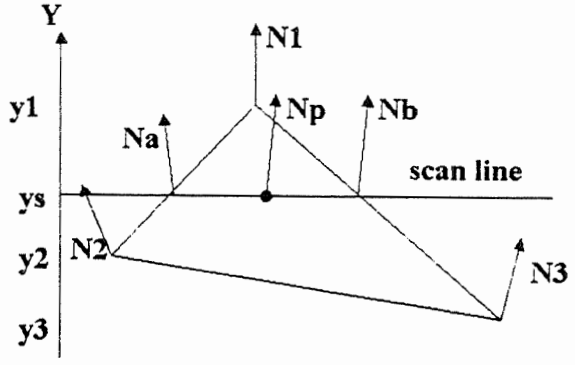



Figura 3.7 Calcolo della normale nel Phong shading.

In questo modo si deriva un vettore normale per ogni pixel componente il poligono proiettato sullo schermo, cioè un' approssimazione della normale reale alla superficie curva approssimata dal poligono. Questa è la caratteristica della buona resa visiva della tecnica di Phong perchè in questo modo viene notevolmente ridotto il problema della banda di Mach. C'è da notare però che essendo le (4) equazioni vettoriali, la fase di interpolazione è tre volte più costosa della fase corrispondente del Gouraud shading. Inoltre per ogni pixel è necessario calcolare l'intensità da assegnargli utilizzando l'equazione (1) con $N(\theta) = N \times L$ (con N uguale al vettore $N_p$ normalizzato e L al vettore unitario orientato nella direzione della luce).

### 3.1.6. Binary gradient shading

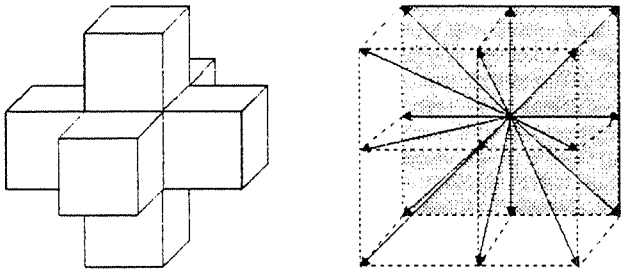
Questo metodo, proposto da Jensen e Hulismans [Jens89], calcola la normale associata ad ogni voxel (i,j,k), operando direttamente nello spazio oggetto su un intorno del voxel (i,j,k). I presupposti per l'applicazione del metodo sono che ogni voxel sia già stato classificato in maniera binaria, cioè gli sia stato assegnato un valore 1 se il voxel è interno all'oggetto da rappresentare e 0 se è esterno. Per ogni voxel interno all'oggetto viene stimato il valore della normale $N_{v0}$ passante per il voxel esaminando i 6 voxel adiacenti ad ognuna delle 6 facce (figura 3.8a). Quando un voxel adiacente ha valore 1, si genera una componente di $N_{v0}$ orientata nella direzione opposta a quella del voxel con valore 1 e con lunghezza unitaria. Se invece un voxel adiacente ha valore 0, si genera una componente di $N_{v0}$ orientata nella direzione del voxel con valore 0 e con lunghezza unitaria. La normale $N_{v0}$ è quindi ottenuta come somma delle sue componenti prodotte, dando luogo a soli 27 possibili valori diversi per la normale approssimata. Infatti su ognuno dei 3 assi corrispondenti alle 3 coordinate le componenti possibili sono 3:

un vettore con direzione la direzione dell'asse e modulo 2;

un vettore con direzione opposta alla direzione dell'asse e modulo 2;

un vettore nullo.

Quindi i valori possibili per il vettore $N_{v0}$ sono $3^3$=27. Le 27 possibilità si riducono a 17 se si considerano solo le normali a prodotto scalare non negativo con la direzione di vista (figura 3.8b).



a) b)

Figura 3.8 Calcolo della normale nel metodo Binary gradient shading.

La qualità delle immagini ottenibili con questo metodo non è ottima data la ridotta casistica di vettori normali generabili. Una volta individuato il vettore normale $N_{v0}$ lo shading dei pixel su cui si proietta il voxel corrispondente può essere calcolato usando la formula (1) con $N(\theta)=N\ L$ (con N uguale al vettore $N_{v0}$ normalizzato e L vettore unitario orientato nella direzione della luce). Il metodo può dare migliori risultati ove siano considerati intorni più ampi di voxel, aumentando tuttavia il numero dei voxel da visitare e quindi il costo computazionale. La complessità del metodo, $O(kV)$ , è dipendente dal numero V di voxel a cui è applicato e dalla cardinalità k dell'intorno di voxel considerato per il calcolo di ogni gradiente. Se applicato nell'ambito di un metodo proiettivo BTF la complessità che ne deriva, $O(kN^3)$ è estremamente gravosa in confronto a quella di altre tecniche.

## 3.2. TECNICHE DI SHADING NELLO SPAZIO IMMAGINE

Lo svantaggio principale delle tecniche di shading nello spazio oggetto (in cui le normali sono memorizzate come parte della descrizione dell'oggetto) consiste nel fatto che i vettori normali alla superficie devono essere ricalcolati se l'oggetto viene modificato interattivamente dall'utente. Questo problema viene invece evitato dalla seguente tecnica di shading.

### 3.2.1. Image-based gradient shading (Z-buffer gradient shading)

Come già accennato, le tecniche di shading basate sullo spazio immagine, non richiedono alcuna rappresentazione oltre alla rappresentazione dell'oggetto, fatta eccezione per una pre-immagine contenente la distanza di ogni pixel dal punto visibile dell'oggetto che si proietta su di esso. In particolare, questa tecnica utilizza come pre-immagine lo Z-buffer (par. 2.3) usato per la rimozione

delle superfici nascoste. Lo Z-buffer contiene cioè una rappresentazione completa della superficie visibile dell'oggetto nella forma z=z(x,y). E' possibile stimare la normale N in ogni punto (x,y) trovando il vettore gradiente $\nabla z$ in quel punto. Assumendo noto il vettore unitario L indicante la direzione della luce, $N(\theta)=N\ L$ e applicando la (1) si ottiene l'intensità da assegnare al pixel (x,y). Data la superficie z=z(x,y), il vettore gradiente è $\nabla z=(dz/dx,\ dz/dy,\ 1)$. Per uno z-buffer N×N, date le distanze $z_{i,j}=z(x=i,y=j)$ $(1\le i\le N,\ 1\le j\le N)$, dz/dx può essere approssimato dalla "backward difference":

$$\delta_b=z_{i,j}-z_{i-1,j} \qquad (2\le i\le N,\ 1\le j\le N)$$

o dalla" forward difference":

$$\delta_f=z_{i+1,j}-z_{i,j} \qquad (1\le i\le N-1,\ 1\le j\le N)$$

o dalla"central difference":

$$\delta_c=1/2(z_{i+1,j}-z_{i-1,j}) \quad (2\le i\le N-1,\ 1\le j\le N)$$

L'ultima è in genere un'approssimazione migliore della *backward* o della *forward* che possono essere usate per i pixel ai bordi dello schermo dove la central difference non si può applicare. In modo analogo si approssima dz/dy. La "central difference" sarebbe una buona approssimazione di dz/dx se z fosse una funzione continua di x, cosa che non sempre accade. Si supponga per esempio che $z_{i-1,j}$ e $z_{i,j}$ siano le distanze da una parte di un oggetto ma $z_{i+1,j}$ sia la distanza da una parte diversa; allora la "backward difference" è un'approssimazione migliore di dz/dx (figura 3.9). Un discorso analogo vale per dz/dy.

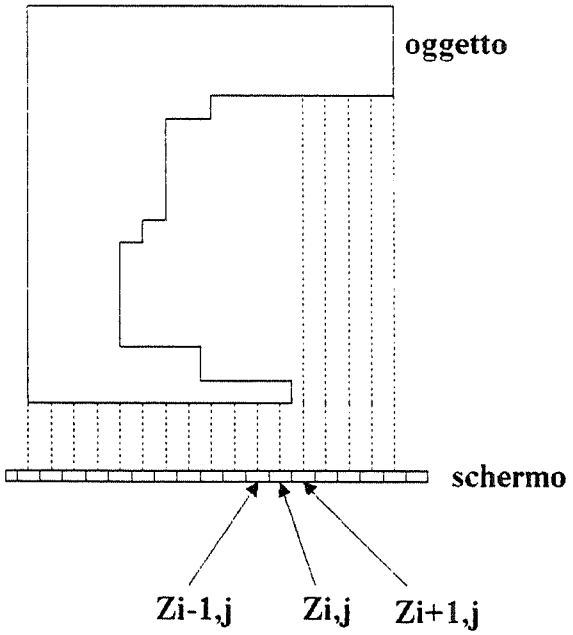



Figura 3.9 Caso in cui applicare la "backward difference" per il calcolo di dz/dx.

Ciò che viene richiesto è dunque un metodo che sia in grado di distinguere cambiamenti graduali lungo una singola superficie da bruschi cambiamenti che si possono avere nel passaggio da una superficie ad un'altra. La soluzione adottata da Gordon e Reynolds [Gord85] è quella di considerare una media pesata della backward e forward difference approssimando dz/dx come:

$$dz/dx \cong [w_b\delta_b + w_f\delta_f]/[w_b+w_f] \quad (5)$$

dove $w_b$ e $w_f$ sono pesi positivi che dipendono da $|\delta_b|$ e $|\delta_f|$ rispettivamente. Piccole differenze

forniscono un peso grande e viceversa. Se $\delta_b$ è piccolo e $\delta_f$ è grande, allora viene dato più peso alla backward difference che più facilmente approssima la reale inclinazione della superficie in quel punto.

Il vantaggio principale del Gradient shading è che non occorrono dati supplementari da memorizzare assieme alla descrizione delle facce. Un secondo vantaggio è che il metodo si dimostra efficiente poichè la complessità è proporzionale al numero di pixel ed è indipendente dal numero di facce dell'oggetto così che l'immagine può essere costruita in un tempo costante indipendentemente dalla complessità dell'oggetto.

### 3.2.2. Distance-only shading

Il modo più semplice e drastico per evitare il costoso calcolo della normale alla superficie visibile è quello di non tenerne conto affatto nel calcolo dello shading, ignorando così l'orientazione della superficie. Un sommario andamento della superficie può essere reso calcolando la luminosità associata ai pixel in funzione della sola distanza d dalla sorgente di luce. Ciò può essere ottenuto ponendo $N(\theta)=1$ nella formula (1). Nella realizzazione data da Chen ed alt. [Chen85] tale tecnica viene usata per calcolare lo shading al centro di ogni faccia visibile di un voxel, che viene estratta con tecnica "surface tracking", e poi assegnando tale shading a tutti i punti visibili sulla stessa faccia. Ciò è giustificato quando i poligoni che approssimano la superficie sono "piccoli" e pertanto la distanza d dalla sorgente di luce non dovrebbe essere molto differente per punti visibili di uno stesso poligono. Notevoli svantaggi di tale tecnica risiedono sia nella perdita dell'informazione di dettaglio che nella perdita di informazione sulle linee di separazione tra superfici diverse. Nel primo caso non vengono rappresentate le particolarità strutturali di ridotte dimensioni della superficie, mentre nel secondo caso non viene rilevata informazione significativa poichè tra i punti circostanti l'intersezione tra due superfici non si presentano generalmente sensibili cambiamenti di profondità. A vantaggio di tale tecnica è invece l'estrema semplicità e la bassa complessità.

Le tecniche di shading prese in esame sono state citate nel capitolo 2 esponendo i diversi algoritmi di visualizzazione volumetrica. Come si è visto alcuni di tali algoritmi procedono ad una ricostruzione di superfici, altri procedono invece direttamente alla visualizzazione delle isosuperfici. E' opportuno pertanto specificare quali tecniche di shading possono essere applicate per visualizzare poligoni (triangoli o facce rettangolari dei voxel nel caso del Surface Tracking o punti della superficie nel caso del Dividing Cubes) e quali tecniche invece sono applicatbili al singolo pixel del piano immagine o voxel dello spazio oggetto. A tale scopo viene mostrata una tabella che mostra le differenze tra le tecniche di shading, specificando lo spazio cui appartengono, gli oggetti e gli algoritmi cui si applicano:

| **Tecnica di shading** | **Spazio** | **Applicazione** | **Algoritmo** |
|---|---|---|---|
| Constant shading | Oggetto | Facce di voxel<br>Triangoli<br><br><br>Punti della superficie | Surface tracking;<br>Marching Cubes;<br>Triang. di contorni planari;<br>Dividing Cubes |
| Image-based contextual shading | Oggetto | Facce di voxel | Surface Tracking |
| Normal-based contextual shading | Oggetto | Facce di voxel | Surface Tracking |

| | | | |
|---|---|---|---|
| Gouraud shading | Oggetto | Triangoli | Triang. di contorni planari; Marching Cubes; |
| Phong shading | Oggetto | Triangoli | Triang. contorni planari; Marching Cubes |
| Binary gradient shading | Oggetto | Voxel | Ray Tracing; |
| Image-based gradient shading | Immagine | Pixel | Back-to-front Front-to-back |
| Distance-only shading | Immagine | Facce di voxel | Surface Tracking |

**Bibliografia**


[Artz81]
E.Artzy, G.Frieder, G.T.Herman, "The Theory, Design, Implementation and Evaluation of a Three Dimensional Surface Detection Algorithm". *Computer Graphics and Image Processing*, Vol. 15, pp. 1-24, 1981.

[Bloom88]
J. Bloomenthal, "Polygonization of implicit surfaces", *Computer Aided Geometric Design*, Vol. 5, No. 4, Novembre 1988, pp. 341-355

[Chen85]
L.S. Chen, G.T. Herman, R.A. Reynolds, "Surface Shading in the Cuberille Environment", *IEEE C.G.&A.*, December 1985, Vol. 5, No.12, pp.33-43.

[Chri78]
H.N.Christiansen, T.W.Sederberg, "Conversion of complex Contour Line definitions into poligonal element mosaics", *A.C.M. Computer Graphics*, Vol. 12, No. 3 Luglio 1978, pp. 187-192.

[Clin88]
H.E.Cline, W.E.Lorensen, S.Ludke, C.R.Crawford, B.C.Teeter, "Two algorithms for the three-dimensional reconstruction of tomograms". *Med. Phys.*, Vol. 15, No. 3, May/Jun, 1988.

[Fole82]
J.D.Foley, A.Van Dam, "Fundamentals of Interactive Computer Graphics", Addison Wesley, Reading (MA), 1989.

[Fren89]
K.A.Frenkel, "Volume Rendering", *Communication A.C.M.*, Aprile 1989, Vol.32, No. 4, pp. 426-435.

[Frie85]
G.Frieder, D.Gordon, R.A.Reynolds, "Back-to-Front Display of Voxel-Based Objects", *IEEE C. G. & A.*, Gennaio 1985, Vol. 5, No. 1, pp. 52-60.

[Fuch77]
H.Fuchs, Z.M.Kendem, S.P.Uselton, "Optimal Surface Reconstruction from Planar Contours", *Communication of the A.C.M.*, Vol. 20, No. 10, Ottobre 1977, pp. 693-702.

[Gana82]
S.Ganapaty, T.G.Denney, "A new general triangulation method for planar contours", *Computer Graphics*, Vol. 16 No. 3, Luglio 1982, pp. 69-75.

[Glas89]
A.S.Glassner, "An Introduction to Ray Tracing", Academic Press, New York, 1989.

[Gord85]
D.Gordon, R.A.Reynolds, "Image Space Shading of Three-Dimensional Objects". *Computer Vision Graphics and Image Processing*, No.29, 1985, pp.361-376.

[Gord89]
D.Gordon, J.K.Udupa, "Fast Surface Tracking in Three-Dimensional Binary Images". *Computer Vision Graphics and Image Processing*. Vol. 45, pp.196-214, 1989.

[Gour71]
H.Gouraud, "Continuous shading of curved surfaces", *IEEE Transaction on Computers C-20*, No. 6, 1971, pp. 623-629.

[Hall90]
M. Hall, J. Warren, "Adaptive Poligonalization of Implicitly Defined Surfaces", *IEEE Computer Graphics & Applications*, Vol. 10, No. 6, Novembre 1990, pp. 33-42

[Herm80]
G.T.Herman, "Image Reconstruction From Projections: The Fundamentals of Computerized Tomography", Academic Press, New York, 1980.

[Herm83]
G.T.Herman, D.Webster, "A Topological proof of a Surface Tracking algorithm". *Computer Vision Graphics and Image Processing*. Vol. 23, pp.162-177, 1983.

[Jens89]
G.J.Jense, D.P.Hulismans, "Interactive Voxel-Based Graphics for 3D Reconstruction of Biological Structures". *Computer & Graphics*, Vol.13, No.2, pp.145-150, 1989.

[Kepp81]
E.Keppel,"Approximating Complex Surfaces by Triangulation of Contour Lines" *IBM J. Res. Develop.*, Gennaio 1975, pp. 2-11.

[Levo88]
M.Levoy, "Volume Rendering: Display of Surfaces from Volume Data" *IEEE, C. G. & A. Maggio 1988*, Vol., 8, No. 5, pp.29-37.

[Levo90]
M.Levoy "Volume Rendering: A Hybrid Ray Tracer for Rendering Polygon and Volume Data", *IEEE C. G. & A.* Marzo 1990, Vol. 10, No. 2, pp. 33-40.

[Lore87]
W.Lorensen, H.Cline, "Marching Cubes: a high resolution 3D surface construction algorithm". *Computer Graphics*, Vol. 21. No. 4, pp. 163-170, 1987.

[Mont87]
C.Montani, M.Re, "Vector and Raster Hidden Surface Removal Using Parallel Connected Stripes", *IEEE Computer Graphics and Applications*, Luglio 1987, Vol.,7, No.7,pp. 14-23.

[Mont90]
C.Montani, R.Scopigno, "Rappresentazione e Visualizzazione di modelli volumetrici", *Consiglio Nazionale delle Ricerche, collana Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo*, No.1/27, Settembre 1990.

[Mont92]
C.Montani, R.Scopigno, "Volume Rendering: Visualizzazione di immagini 3D". *Secondo Convegno Nazionale Conoscenza per Immagini, C.N.R.* Roma, 13-15 Aprile 1992.

[Ney90] D.R.Ney, E.K.Fischman, D.Magid, "Volumetric Rendering of C.T. Data: Principles and Tecniques", *IEEE Computer Graphics & Applications*, Marzo 1990, pp. 24-32.

[Phong75] B-T.Phong, "Illumination for computer generated pictures", *Comm. A.C.M.*, Vol. 18, No. 16, 1975, pp.311-317.

[Reyn87] R.A.Reynolds, D.Gordon, L.S.Chen, "A Dynamic Screen Technique for Shaded Graphics Display of Slice-Represented Objects", *Computer Vision, Graphics, and Image Processing*, Vol. 38, pp. 299-316, 1987.

[Roge85] D.F.Rogers, "Procedural Elements for Computer Graphics", McGraw-Hill, New York, 1985.

[Schr92] W. J. Schroeder, J. A. Zarge, W. E. Lorensen, "Decimation of Triangle Meshes", *Computer Graphics*, Vol. 26, No. 2, Luglio 1992, pp. 65-68.

[Sper90] D.Speray, S.Kennon, "Volume Probes: Interactive Data Exploration on Arbitrary Grids", *Computer Graphics*, Vol. 24, No. 5, Novembre 1990.

[Srih81] S.N.Srihari, "Representation of Three-Dimensional Digital Images", *Computing Surveys*, Vol. 13, No.4, Dicembre 1981, pp. 399-424.

[Styt91] M.R.Stytz, G.Frieder, O.Frieder, "Three-Dimensional Medical Imaging: alghorithms and Computer Systems", *A.C.M Computing Surveys*, Vol. 23, No. 4, Dicembre 1991, pp.421-499.

[Tuy84] H.K.Tuy, L.T.Tuy, "Direct 2-D Display of 3-D Objects", *IEEE C. G. & A*, Ottobre 1984, pp.29-33.

[Upso88] C.Upson, M.Keeler, "V-BUFFER: Visible Volume Rendering", *A.C.M. Computer Graphics*, Vol. 22, No. 4, Agosto 1988, pp. 59-64.

[Watt89] A.Watt, "Fundamentals of Three-Dimensional Computer Graphics", Addison Wesley, Reading (MA), 1989.

[Wilh90] J. Wilhelms, A. Van Gelder, "Topological Considerations in Isosurface Generation Extended Abstract", *Computer Graphics*, Vol. 24, No. 5, Novembre 1990, pp. 79-86.

[Wyvi86] G. Wyvill, C. McPheeters, B. Wyvill, "Data structures for soft objects", *The Visual Computer*, Vol. 2, No. 4, Agosto 1986, pp. 227-234.