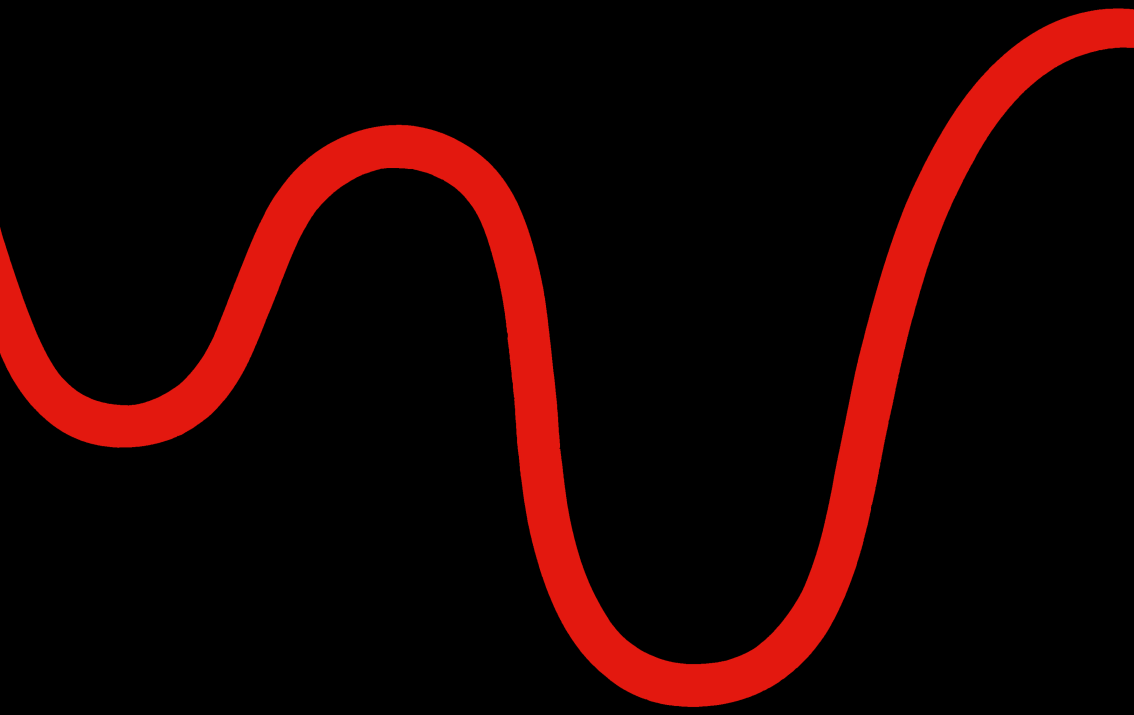
COMPUTER GRAPHICS

CORSO DI PROGRAMMAZIONE

STEVEN HARRINGTON



McGraw-Hill

COMPUTER GRAPHICS

Corso di programmazione

Sezione di Computer Graphics
coordinata da Umberto Cugini

COMPUTER GRAPHICS

CORSO DI PROGRAMMAZIONE

STEVEN HARRINGTON

McGRAW-HILL Book Company GmbH

Amburgo New York St Louis San Francisco Auckland
Bogotá Città del Guatemala Città del Messico Johannesburg
Lisbona Londra Madrid Montreal Nuova Delhi Panama
Parigi San Juan San Paolo Singapore Sydney Tokyo Toronto

Titolo originale: *COMPUTER GRAPHICS - A Programming Approach*
Copyright © 1983 McGraw-Hill, Inc.

Copyright © 1985 McGraw-Hill Book Co. GmbH
Lademannbogen 136
D 2000 Hamburg 63, RFT

I diritti di traduzione, di riproduzione, di memorizzazione elettronica e di adattamento totale e parziale con qualsiasi mezzo (compresi i microfilm e le copie fotostatiche) sono riservati per tutti i paesi.

Realizzazione editoriale EDIGEO srl, via del Lauro 3, 20121 Milano
Traduzione: Claudio Devoti, Matilde Giuffra,
Roberto Mazzucchelli e Roberto Pozzoli
Grafica di copertina: Valentina Boffa
Composizione e stampa: Litovelox, Trento

ISBN 88-7700-601-3

1ª edizione giugno 1985

Indice

Prefazione all'edizione italiana 11

Introduzione 13
Un approccio pratico *13*
Hardware non specializzato *13*
Software standard *14*
Requisiti software *14*
Requisiti dell'elaboratore *14*
Conoscenze necessarie *15*
Tempo occorrente *15*
Rapporto con il sistema CORE *15*
Semplicità ed efficienza *16*
Approcci a un corso di grafica *16*
Organizzazione del testo *16*
Struttura degli algoritmi *17*

Capitolo 1 **Geometria e generazione di rette** 23
Introduzione *23*
1.1 Punti e rette *25*
1.2 Piani e coordinate *27*
1.3 Segmenti *30*
1.4 Rette perpendicolari *33*
1.5 Vettori *34*
1.6 Pixel e frame buffer *35*
1.7 Generazione di vettori *36*
1.8 Generazione di caratteri *41*
1.9 Visualizzazione del frame buffer *42*
Esercizi *45*
Problemi di programmazione *46*

Capitolo 2 **Primitive grafiche 51**
Introduzione *51*
2.1 Dispositivi grafici *52*
2.2 Operazioni elementari *55*
2.3 L'interprete del display file *58*
2.4 Coordinate normalizzate del dispositivo *59*
2.5 Struttura del display file *60*
2.6 Algoritmi per il display file *64*
2.7 Controllo dello schermo *67*
2.8 Testi *69*
2.9 Tipi di linee *77*
Applicazione pratica *80*
Esercizi *82*
Problemi di programmazione *84*

Capitolo 3 **I poligoni 93**
Introduzione *93*
3.1 Generalità sui poligoni *94*
3.2 Rappresentazione dei poligoni *95*
3.3 Introduzione dei poligoni nel display file *95*
3.4 Test di appartenenza *97*
3.5 Interfacciamento dei poligoni con il sistema *98*
3.6 Riempimento dei poligoni *101*
Applicazione pratica *116*
Esercizi *118*
Problemi di programmazione *119*

Capitolo 4 **Le trasformazioni 127**
Introduzione *127*
4.1 Matrici *128*
4.2 Trasformazioni di scala *130*
4.3 Funzioni trigonometriche *134*
4.4 Relazioni riguardanti le somme di angoli *135*
4.5 Rotazioni *137*
4.6 Coordinate omogenee e traslazioni *139*
4.7 Rotazioni attorno a un punto arbitrario *142*
4.8 Altre trasformazioni *144*
4.9 Algoritmi di trasformazione *148*
4.10 Procedure di display *156*
Applicazione pratica *159*
Esercizi *161*
Problemi di programmazione *163*

Capitolo 5 **Segmentazione del display file 167**
Introduzione *167*
5.1 Tabella dei segmenti *168*

5.2 Creazione di un segmento *170*
5.3 Chiusura di un segmento *171*
5.4 Cancellazione di un segmento *172*
5.5 Cambiamento di nome a un segmento *176*
5.6 Visibilità di un segmento *178*
5.7 Trasformazione dell'immagine *179*
5.8 Aggiornamento delle routine di trasformazione *181*
5.9 Memorizzazione e visualizzazione dei segmenti *183*
5.10 Altre strutture del display file *186*
5.11 Alcune tecniche raster *188*
Applicazione pratica *191*
Esercizi *192*
Problemi di programmazione *193*

Capitolo 6 Windowing e clipping 197
Introduzione *197*
6.1 Trasformazione di visualizzazione *198*
6.2 Algoritmi per la trasformazione di visualizzazione *204*
6.3 Clipping *208*
6.4 Clipping di poligoni *214*
6.5 Aggiunta del clipping al sistema *219*
6.6 Evitare le divisioni *220*
6.7 Clipping generalizzato *221*
6.8 Posizione di un punto rispetto a una retta arbitraria *222*
6.9 Windowing multiplo *224*
Applicazione pratica *224*
Esercizi *226*
Problemi di programmazione *227*

Capitolo 7 Interazione fra macchina e utente 231
Introduzione *231*
7.1 Hardware *232*
7.2 Algoritmi di gestione dei dispositivi di input *238*
7.3 Trattamento degli eventi asincroni *242*
7.4 Dispositivi campionati *251*
7.5 Attributi di reperibilità *252*
7.6 Simulazione di un locator con un dispositivo di pick *256*
7.7 Simulazione di un dispositivo di pick con un locator *257*
7.8 Eco corrispondente alle operazioni di input *262*
7.9 Tecniche di interazione *263*
Esercizi *272*
Problemi di programmazione *272*

Capitolo 8 Grafica tridimensionale 275
Introduzione *275*

8.1 Geometria 3-D *276*
8.2 Primitive 3-D *280*
8.3 Trasformazioni 3-D *283*
8.4 Rotazione attorno a un asse arbitrario *289*
8.5 Proiezione parallela *294*
8.6 Proiezione prospettica *297*
8.7 Parametri di visualizzazione *300*
8.8 Conversione nelle coordinate del piano di proiezione *310*
8.9 Trasformazione di visualizzazione 3-D *314*
8.10 Proiezioni di tipo speciale *318*
Applicazione pratica *323*
Esercizi *326*
Problemi di programmazione *328*

Capitolo 9 Clipping tridimensionale 331
Introduzione *331*
9.1 Quando eseguire il clipping *332*
9.2 Clipping di volumi *334*
9.3 Piani di clipping *337*
9.4 Determinazione dei parametri di clipping *340*
9.5 Clipping con un piano *343*
9.6 Algoritmi di clipping 3-D *346*
9.7 Modifiche ulteriori *352*
Applicazione pratica *355*
Esercizi *356*
Problemi di programmazione *357*

Capitolo 10 Superfici e linee nascoste 359
Introduzione *359*
Parte 1 Superfici nascoste convesse: algoritmo BACK-FACE *360*
10.1 Facce nascoste *360*
10.2 Algoritmi di tipo BACK-FACE *365*
Parte 2 Superfici nascoste convesse: algoritmo del pittore *369*
10.3 L'algoritmo del pittore *369*
10.4 Memorizzazione temporanea dei poligoni *372*
10.5 Memorizzazione del tipo di linea utilizzato *375*
10.6 Ricerca delle superfici nascoste *376*
10.7 Scomposizione dei poligoni in triangoli *378*
10.8 Confronto dei triangoli *384*
10.9 Test di minimax *385*
10.10 Sovrapposizione dei lati *388*
10.11 Contenimento di un punto *392*
10.12 Reperimento di un punto nel piano del triangolo *395*
10.13 Contenimento dell'intero triangolo *397*
10.14 Determinazione dell'ordine di profondità *399*

10.15 Ordinamento geometrico *401*
10.16 Liste concatenate *401*
10.17 Ordinamento dei triangoli *404*
Parte 3 Linee nascoste *409*
10.18 Eliminazione delle linee nascoste *409*
10.19 Copertura delle linee da parte dei triangoli *411*
10.20 Suddivisione di una linea *418*
10.21 Controllo delle linee nascoste per i lati del triangolo *423*
Parte 4 Superfici nascoste concave *427*
10.22 Generalizzazione ai poligoni concavi *427*
Esercizi *435*
Problemi di programmazione *435*

Capitolo 11 Ombreggiatura 437
Introduzione *437*
11.1 Illuminazione diffusa *437*
11.2 Illuminazione da sorgente puntiforme *440*
11.3 Riflessione speculare *442*
11.4 Algoritmi di ombreggiatura *446*
11.5 Trasparenza e ombre *456*
Esercizi *456*
Problemi di programmazione *457*

Capitolo 12 Le curve 459
Introduzione *459*
12.1 Generazione di curve *459*
12.2 Interpolazione *462*
12.3 Algoritmi di interpolazione *467*
12.4 Arrotondamento di poligoni *473*
12.5 *B*-spline *475*
12.6 *B*-spline e angoli *481*
Applicazione pratica *483*
Esercizi *486*
Problemi di programmazione *486*

Appendice A Il sistema grafico CORE 487

Appendice B Grafica su un terminale non grafico 495

Appendice C Interfacciamento con il PLOT10 501

Appendice D Interfacciamento con le routine CALCOMP 505

Bibliografia 509

Indice analitico 511

Prefazione all'edizione italiana

I risultati derivanti dall'impiego dell'elaboratore nel settore della grafica — la *Computer Graphics*, per utilizzare il termine anglosassone, sintetico e significativo, ormai entrato nell'uso comune — sono ormai una realtà che è possibile incontrare quotidianamente, dalla sigla televisiva ai videogame, dal disegno tecnico alla rappresentazione dei dati statistici.
Un'altra realtà è la sempre crescente produzione e diffusione di letteratura specialistica tecnica e divulgativa, che ha lo scopo di abbattere la cosiddetta «barriera informatica», che smitizzi cioè le difficoltà e l'atmosfera un po' magica che avvolge le attività connesse con l'uso degli elaboratori, limitandone l'accesso ai soli addetti.
Il libro di Steven Harrington copre un'area finora lasciata scoperta, compresa tra i testi dedicati ad un pubblico specialistico (testi per corsi a livello universitario inseriti in piani di studio nell'area informatica) e testi molto, a volte troppo, divulgativi che si riducono poi ad una serie di programmi già pronti da copiare e utilizzare, senza connessione tra loro, con poche e frammentarie descrizioni degli algoritmi utilizzati e della teoria che li origina.
L'approccio seguito da Harrington è molto pragmaticamente quello di descrivere le basi della Computer Graphics in modo rigoroso ma, allo stesso tempo, con il minimo indispensabile di formalismo, rendendo quindi accessibile il libro ad un pubblico con basi matematiche elementari e concatenando la spiegazione dei vari algoritmi in modo da portare il lettore anche alla realizzazione di un package grafico ben costruito e basato su concetti di standardizzazione delle funzioni. Il libro ha così vari livelli di lettura e si rivolge contemporaneamente a varie tipologie di lettori.
Può essere usato come testo per corsi di Computer Graphics, sia nell'ambito della scuola media superiore, che in un corso di base a livello universitario, vista la sua struttura progressiva e ben articolata e la chiarezza di esposizione: per questo tipo di utilizzo, gli esercizi — che ben si integrano con le parti svolte nel testo — sono articolati in modo da approfondire quelle parti che risultereb-

bero troppo specialistiche per l'impostazione generale del libro.
Può essere usato come testo divulgativo, dato l'approccio e lo stile utilizzato, permettendo anche una lettura non totale e sequenziale dei vari argomenti trattati da parte di chiunque abbia interesse nelle problematiche della Computer Graphics, senza per questo richiedere una preparazione specifica precedente. Infatti, anche le parti relative ai fondamenti sulle trasformazioni e sulle rappresentazioni di entità geometriche fanno riferimento a concetti elementari di geometria analitica.
Può essere usato — da professionisti o hobbisti — come insieme di specifiche per produrre un sistema grafico di base, sia per della grafica bidimensionale, che per rappresentazioni tridimensionali, con il notevole vantaggio di un'impostazione sistemistica del package che verrà prodotto, che si basa su concetti di standardizzazione della grafica di base.
Può, infine, essere usato come «libro di ricette» a cui attingere algoritmi per problemi grafici specifici. Infatti, la modularità d'impostazione sia degli argomenti che della struttura delle procedure illustrate permette di poter facilmente enucleare le descrizioni di procedure specifiche, che possono essere implementate indipendentemente o inserite in altri contesti software che forniscono il supporto richiesto per la visualizzazione.
Il libro inoltre è impostato in modo da far riferimento a funzioni estremamente elementari, che possono essere realizzate anche disponendo di un personal computer dotato di stampante, senza la necessità di costose e sofisticate apparecchiature.
Grazie a questa multifunzionalità il libro di Harrington costituisce un passo fondamentale verso la divulgazione completa e rigorosa della Computer Graphics.

Umberto Cugini

Introduzione

UN APPROCCIO PRATICO

Questo libro vuole essere un testo introduttivo alle tecniche interattive riguardanti la Computer Graphics e i suoi concetti di base. L'approccio seguito si differenzia da quello dei testi precedenti poiché spinge il lettore a sperimentare o comunque ad imparare mettendo immediatamente in pratica quanto apprende. Il libro rappresenta una guida per lo sviluppo di un sistema grafico e fornisce anche molti suggerimenti per programmi applicativi.
I vantaggi di un tale tipo di approccio sono: un elevato grado di coinvolgimento del lettore, una realistica intuizione delle possibilità del software grafico e della sua facilità d'uso e un appagamento più gratificante nel raggiungere i risultati.

HARDWARE NON SPECIALIZZATO

La Computer Graphics (C.G.) è stata spesso associata ad un hardware specializzato per la visualizzazione. Benché questi dispositivi permettano la generazione di immagini di elevato grado qualitativo, non sono però necessari per insegnare la grafica. L'obiettivo di questo libro è quello di far capire in cosa consiste la grafica e per questo scopo è più che sufficiente la bassa risoluzione di una comune stampante o di un normale terminale video. È naturale che, ove sia disponibile un terminale grafico, questo venga utilizzato, ma resta il fatto che esso non è un requisito imprescindibile. Questo libro può essere utilizzato per un corso di Computer Graphics anche se non si dispone di un hardware specializzato. Si potranno così evitare code di studenti in attesa di usare il solo terminale grafico disponibile.

SOFTWARE STANDARD

L'esistenza di diversi sistemi di software grafico di base è un'altra ragione per la quale in precedenza non è stato seguito un approccio pratico. C'era innanzitutto il problema di scegliere quale sistema software usare. Solo recentemente è emersa la proposta di uno standard grafico: il sistema CORE proposto dal *Graphics Standards Planning Committee*. Questo libro segue le specifiche del sistema CORE e le sue estensioni. Il sistema CORE è stato sviluppato per rendere più portabili i programmi di grafica e quindi si basa sulla indipendenza da qualsiasi particolare dispositivo fisico. Si prefigge inoltre di specificare le prestazioni grafiche di base, fornendo il supporto per tecniche più avanzate. Entrambi questi obiettivi fanno di questo standard un ottimo modello per un sistema didattico.

REQUISITI SOFTWARE

Ci si potrebbe chiedere a questo punto se sia strettamente necessario per il corso avere a disposizione una implementazione del sistema CORE. La risposta è no. Il testo ha come obiettivo la definizione di algoritmi dettagliati per la costruzione del sistema grafico. Lo studente può anche implementarli come parte applicativa del corso realizzando autonomamente il proprio sistema grafico. Se lo scopo del corso deve essere l'applicazione della grafica, allora il sistema verrà implementato dal docente ad uso degli studenti. L'implementazione è più semplice di quanto potrebbe sembrare, grazie al dettaglio col quale sono scritti gli algoritmi ed alla modularità con la quale il sistema è strutturato, che ne consente l'implementazione per estensioni in fasi successive e indipendenti.

REQUISITI DELL'ELABORATORE

Il sistema completo non è di dimensioni ridotte. Un'implementazione scritta in FORTRAN consta di 4000 linee di programma sorgente che compilate, ad esempio, dall'elaboratore PRIME 400 costituiscono un codice oggetto che occupa 16 Kparole per le istruzioni ed altri 16 Kparole per i dati. È comunque necessario disporre di un linguaggio ad alto livello come PL/1, Pascal, ALGOL, FORTRAN o APL. Il BASIC e il COBOL possono essere usati, ma con una difficoltà di implementazione molto maggiore. Ad esempio, l'implementazione in linguaggio BASIC su un microcomputer come l'Apple II non presenta problemi per le routine bidimensionali, ma comincia ad avere grossi problemi di memoria se si tenta un'estensione alle tre dimensioni.

CONOSCENZE NECESSARIE

Questo libro è destinato a studenti che abbiano conoscenze di algebra, geometria analitica e trigonometria a livello di scuola media superiore e che conoscano almeno uno dei linguaggi di programmazione ad alto livello. Gran parte della C.G. fa infatti uso della geometria analitica. Il libro illustra un sistema grafico in quanto sequenza di problemi di programmazione ed è per questo che viene richiesta anche la conoscenza di almeno un linguaggio ad alto livello; non ne è richiesto nessuno in particolare, poiché gli algoritmi sono presentati nel modo più indipendente possibile da ogni singolo linguaggio.

TEMPO OCCORRENTE

Il testo è stato usato per un corso semestrale alla State University of New York di Brockport. È possibile raggiungere una cadenza di apprendimento di circa un capitolo alla settimana. È necessario invece un tempo leggermente superiore per assimilare gli argomenti riguardanti le tre dimensioni e le linee nascoste. Per ciascun capitolo è prevista un'applicazione pratica: un programma alla settimana non è un carico di lavoro eccessivo anche se abbastanza impegnativo. Se necessario, gli insegnanti possono dare agli studenti, già pronti, i programmi meno interessanti dal punto di vista propedeutico.
Al termine del corso gli studenti non solo avranno sviluppato un'esperienza operativa sui fondamenti della C.G., ma potranno disporre di un intero sistema grafico, o per lo meno di una serie di programmi grafici dei quali potranno, a ragione, andare fieri. Alla fine di ciascun capitolo sono suggeriti parecchi problemi di programmazione; i più difficili sono contrassegnati da uno o due asterischi, a seconda della difficoltà.

RAPPORTO CON IL SISTEMA CORE

Come abbiamo già detto, il testo si sforza di seguire le specifiche CORE del *Graphics Standards Planning Committee* nella stesura degli algoritmi. Ciò non significa che il sistema presentato sia il CORE: la differenza è dovuta al fatto che il testo ha obiettivi diversi da quelli del *Graphics Standards Planning Committee*. L'obiettivo del libro è quello di illustrare i fondamenti della grafica e a questo scopo sono stati scelti gli argomenti più significativi; sono state quindi deliberatamente tralasciate alcune routine che devono essere necessariamente inserite in un'implementazione del CORE, per considerarne altre che sono, attualmente, solo una proposta di estensione del CORE. Tutto ciò è l'opposto della filosofia CORE, che evita di selezionare proprio per assicurare l'uniformità dei sistemi e quindi la portabilità dei programmi.

SEMPLICITÀ ED EFFICIENZA

Qua e là nel testo ci sono casi in cui si sarebbero potuti usare diversi tipi di algoritmi, o dove molti problemi avrebbero potuto essere risolti in modo diverso. La regola seguita è di presentare sempre una sola soluzione, considerando che esporre due diversi metodi per risolvere lo stesso problema significhi creare confusione. Ciò anche se la soluzione scelta non è la più intelligente o la più efficiente. L'obiettivo primario non è infatti di creare il miglior sistema grafico, bensì di insegnare i principi fondamentali della grafica e proprio per questo gli algoritmi sono stati scelti in base alla chiarezza e comprensibilità con la quale vengono applicati tali principi. Se sorge la domanda «perché non fare in quest'altro modo?», significa che la scelta è stata ottimale, perché chi pone la domanda avrà recepito esattamente lo scopo e la funzione dell'algoritmo.

APPROCCI A UN CORSO DI GRAFICA

Ci sono almeno due modi di affrontare un corso di grafica usando questo libro. Il primo consiste nel fare implementare agli studenti il sistema grafico descritto. Seguendo un tale approccio lo studente comprenderà a fondo come funzionano ed interagiscono le varie componenti del sistema. Occorre eventualmente che gli vengano fornite le soluzioni agli esercizi assegnati la settimana precedente, poiché ogni nuova estensione si basa su ciò che è stato precedentemente consolidato. Esiste il pericolo che, fornendo gli algoritmi per esteso, essi vengano copiati senza essere veramente capiti; a ciò si può ovviare con una spiegazione in aula ed assegnando alcuni esercizi proposti dal testo.
Il secondo tipo di approccio è orientato più all'utilizzo di un sistema grafico che non alla sua realizzazione. Da questo punto di vista gli algoritmi servono come esempi di routine del sistema grafico, ma non come traccia per l'implementazione. Gli studenti dovrebbero allora disporre di questo o di un altro sistema equivalente. Alcune lezioni possono allora essere dedicate alla spiegazione delle possibilità e al modo di lavorare del sistema utilizzato. In questo caso gli esercizi di programmazione sono orientati all'uso della funzionalità del sistema piuttosto che alla loro realizzazione. Alla fine di ciascun capitolo vengono proposti a questo scopo alcuni problemi di programmazione.

ORGANIZZAZIONE DEL TESTO

L'organizzazione del testo è tale da introdurre ad ogni capitolo un nuovo argomento di C.G.; ogni nuovo capitolo si basa sul precedente e quindi l'ordine di lettura è piuttosto fisso.
Il **Capitolo 1** inizia con un elementare ripasso di geometria analitica ed introduce la generazione di vettori. Le routine riportate possono fornire risultati grafici

su comuni terminali e possono essere implementate anche se non si dispone di un hardware particolare.
Il **Capitolo 2** si occupa dei comandi per disegnare con linee e caratteri, oltre che degli strumenti per interfacciare i dispositivi di visualizzazione usati con il resto del sistema. In questo capitolo viene inoltre introdotto il concetto fondamentale di display file.
Il **Capitolo 3** prende in considerazione le superfici poligonali. Sebbene il sistema CORE sia orientato a display che lavorano per linee, il testo considera talune estensioni grafiche tipicamente *raster* quali i poligoni e le superfici nascoste. I concetti della grafica raster non vengono quindi tralasciati.
Il **Capitolo 4** si occupa delle principali trasformazioni: traslazione, variazione di scala e rotazione.
Il **Capitolo 5** tratta della segmentazione del display file e degli attributi di visibilità dei segmenti stessi. Questi primi cinque capitoli, che si occupano tutti della gestione e dell'interpretazione del display file, costituiscono un tutt'uno se si considerano i terminali grafici intelligenti che gestiscono direttamente il loro display file.
Il **Capitolo 6** completa l'argomento della grafica bidimensionale considerando i problemi del windowing e del clipping.
Nel **Capitolo 7** vengono esaminate le tecniche di interazione. Qui, ancora una volta, le routine fungono da interfaccia tra il resto del sistema e il particolare tipo di hardware grafico di input; se occorre, la tastiera può simulare ciascuno dei tipi di input.
Nel **Capitolo 8** il sistema viene esteso alla grafica 3-D. Vengono trattati gli argomenti della geometria 3-D, delle trasformazioni di visualizzazione, della proiezione parallela e della proiezione prospettica.
Nel **Capitolo 9** si affronta l'estensione del clipping al caso tridimensionale.
Il **Capitolo 10** si occupa dell'argomento delle superfici nascoste e di quello dell'eliminazione delle linee nascoste; esso è diviso in quattro parti: la prima è il semplice controllo per la cancellazione di facce nascoste, la seconda è una prima versione dell'algoritmo del pittore, la terza allarga il discorso dalle superfici nascoste al problema della cancellazione delle linee nascoste ed infine la quarta generalizza l'algoritmo ai poligoni concavi.
Il **Capitolo 11** considera il problema dell'ombreggiatura. Per far questo è indispensabile l'implementazione della parte prima del Capitolo 10.
Il **Capitolo 12**, l'ultimo, si occupa del disegno di curve. Vengono trattati sia la generazione di archi che l'interpolazione. Allo sviluppo di un semplice schema di interpolazione segue la presentazione delle *B*-spline.

STRUTTURA DEGLI ALGORITMI

Gli algoritmi presentati in questo libro sono scritti in un immaginario linguaggio di programmazione di tipo ALGOL. Questo linguaggio è stato scelto soprattutto perché permette di concentrarsi sulla «sostanza» degli algoritmi, senza

preoccuparsi troppo delle dichiarazioni e delle sintassi. L'implementazione del sistema necessita di una traduzione in un linguaggio reale di programmazione, anche se ciò richiede un leggero sforzo. La traduzione è possibile in PASCAL, PL/1, FORTRAN indifferentemente. In questo paragrafo viene presentata la descrizione formale dello pseudo-ALGOL usato, per aiutare a comprendere gli algoritmi sviluppati.

COMMENTI
I commenti possono essere inseriti in ogni punto dell'algoritmo. Si distinguono dal resto per il fatto che sono scritti a lettere minuscole.

IDENTIFICATORI
I nomi di variabili e procedure possono essere costituiti da numeri, lettere maiuscole e simboli speciali, purché inizino con una lettera.

VARIABILI
Le variabili possono essere considerate come *argomenti*, come *variabili locali* o *globali.*
Le variabili locali esistono solo finché il controllo è mantenuto dalla routine nella quale sono definite e non mantengono il loro valore per chiamate successive della stessa routine.
Le variabili globali sono conosciute da ogni routine. Ogni variabile globale ha un unico nome attraverso il quale è identificata per tutta la durata del programma.
Gli argomenti vengono considerati come vincolati da un meccanismo di chiamata del tipo *call by reference*, cosicché un'alterazione nel valore di un argomento della routine chiamata porta ad un corrispondente cambiamento della variabile associata nella routine chiamante.
Non vengono specificati i tipi delle variabili.

ARRAY
Possono essere usati array di dati. I singoli elementi degli array sono specificati dal nome seguito da una lista di indici racchiusi tra parentesi quadre: A[I, J, K]. Non esistono limitazioni sulle dimensioni degli array e non esiste alcuna istruzione per il loro dimensionamento. Un intero array può essere passato da una routine all'altra indicando il suo nome (senza indici) fra gli argomenti.

COSTANTI
Le costanti possono essere associate a degli identificatori. Questi identificatori possono essere usati come costanti per rendere più comprensibile il loro ruolo nella routine.

OPERATORI
Gli operatori aritmetici disponibili sono:

↑ elevazione a potenza
* moltiplicazione
/ divisione
+ addizione
− sottrazione e negazione

Gli operatori relazionali sono:

= uguale
≠ diverso
< minore
> maggiore
≤ minore o uguale
≥ maggiore o uguale

Il risultato fra operatori relazionali è un valore TRUE o FALSE. Gli operatori logici sono:

not negazione
and congiunzione
or disgiunzione

FUNZIONI INCORPORATE
Alcune funzioni matematiche e trigonometriche di uso corrente sono considerate presenti nel linguaggio. Esse sono:

INT(X) parte intera di X
MAX(X, Y) massimo tra X e Y
MIN(X, Y) minimo tra X e Y
|X| valore assoluto di X
SIN(X) seno di X
COS(X) coseno di X
ARCTAN(X) arcotangente di X
SQRT(X) radice quadrata di X

ISTRUZIONI
Le routine sono composte da istruzioni, dove ogni istruzione causa l'esecuzione di un'azione. Possono esserci più istruzioni su una stessa riga, oppure un'istruzione può richiedere più righe. Le istruzioni devono terminare con un punto e virgola (;).

<istruzione 1>; <istruzione 2>; <istruzione 3>; <istruzione 4>;

Ogni istruzione contiene almeno una parola chiave (in minuscolo neretto) o un simbolo di assegnamento (←).

BLOCCHI
Più istruzioni possono venir raggruppate per formare una singola istruzione. Questo blocco di istruzioni deve essere racchiuso dalle parole chiave **begin** ed **end**.

```
begin
    <istruzione 1>;
    <istruzione 2>;
    <istruzione 3>;
end;
```

Ogni routine consiste di un blocco di istruzioni. L'incolonnamento delle istruzioni non ha alcun significato formale ed è stato fatto solamente per facilitare la lettura degli algoritmi.

ASSEGNAMENTI
Alle variabili si possono assegnare dei valori tramite le istruzioni di assegnamento.

```
<variabile> ← <espressione>;
```

come ad esempio

```
X←3;
```

L'operatore di assegnamento è ←.

ISTRUZIONI DI CONTROLLO E LOOP
Le decisioni sono prese dall'istruzione **if**, che si presenta in due forme

```
if <espressione> then <istruzione>;
```

oppure

```
if <espressione> then <istruzione 1> else <istruzione 2>;
```

<espressione> potrà essere valutata TRUE (vera) o FALSE (falsa). Se <espressione> è TRUE, sarà eseguita <istruzione> (nella prima forma) oppure <istruzione 1> (nella seconda forma). Se <espressione> è FALSE, non succede niente nella prima forma mentre sarà eseguita <istruzione 2> nella seconda forma. Se la struttura è ad **if** annidati, la clausola **else** è associata all'**if** incompleto più recente.

```
if <E1> then if <E2> then <I1> else <I2>;
```

L'istruzione <I2> dipende dal test su <E2>.

Il costrutto **for** permette di ripetere una o più istruzioni un numero finito di volte, variando un indice o contatore. La sua forma è la seguente:

for <indice> = <partenza> **to** <fine> **do** <istruzione>;

Ad ogni ripetizione viene incrementato <indice> di 1; la prima volta che <istruzione> viene eseguita, <indice> avrà valore uguale a <partenza>. Le ripetizioni proseguono fino a che <indice> avrà raggiunto il valore <fine>. Se <fine> è minore di <partenza>, l'indice sarà decrementato. Le espressioni <partenza> e <fine> vengono valutate una sola volta all'entrata del ciclo.

Il costrutto **while** ha la seguente forma:

while <espressione> **do** <istruzione>;

<istruzione> viene ripetuta finché <espressione> diventa FALSE. Se <espressione> vale già FALSE prima di entrare nel ciclo, <istruzione> non sarà eseguita.

FUNZIONI E PROCEDURE

Ogni routine è considerata come un sottoprogramma, richiamabile da ogni altro sottoprogramma (ad eccezione delle chiamate recursive). L'ordinamento dei vari sottoprogrammi è arbitrario. Ogni routine è numerata in ordine progressivo all'interno di ogni capitolo e inizia con il suo nome seguito da una lista di argomenti racchiusi fra parentesi tonde. Ad esempio:

Algoritmo 14.1 ALG (X, Y, Z)

Le funzioni sono trattate allo stesso modo delle procedure, con l'eccezione che, in un punto della routine stessa, al nome della funzione deve essere assegnato il valore da ritornare alla routine chiamante.

```
Algoritmo 14.2  FUNZ (Y)
  begin
    FUNZ←Y*2;
  end;
```

L'istruzione **return** può essere usata in ogni punto di una routine per ridare il controllo alla routine chiamante.

Esiste anche l'istruzione **return-error** <stringa>; essa stamperà il messaggio di errore contenuto in <stringa> prima di passare il controllo alla routine chiamante.

La chiamata ad una procedura è composta semplicemente dal nome della procedura da chiamare e da un'eventuale lista di argomenti:

```
ALG(2, 4, 6);
```

Le funzioni sono chiamate usando il loro nome all'interno di un'espressione.

```
Z←2*FUNZ(4)+7;
```

ISTRUZIONI DI INPUT/OUTPUT
Un input può essere eseguito con l'istruzione

read <variabile>;

dove <variabile> è l'identificatore della variabile a cui devono essere assegnati i valori.

L'output è eseguito da un'istruzione del tipo:

print <espressione 1>, <espressione 2>, ...<espressione N>;

dove <espressione 1>...<espressione N> rappresentano i valori che si vogliono stampare. Ogni istruzione **print** genera un nuovo record di output (una nuova riga). La stampa di un array può essere effettuata nel seguente modo:

print for <indice> = <partenza> **to** <fine>, <array>[<indice>];

NOTA ALL'EDIZIONE ITALIANA
Data l'universalità — nell'ambito della C.G. — della lingua inglese, si è preferito lasciare i nomi delle variabili e delle procedure così come sono nell'edizione originale, nella convinzione che ciò concorra ad aumentare, anziché diminuire, la chiarezza dell'esposizione; sono stati quindi tradotti solo i messaggi d'errore e le eventuali stringhe di output.

1

Geometria e generazione di rette

INTRODUZIONE

Probabilmente la nostra era passerà alla storia come quella della rivoluzione informatica o della rivoluzione del computer, poiché stiamo assistendo ad un'incredibile crescita e sviluppo della tecnologia e delle applicazioni degli elaboratori. Il computer è una macchina per elaborare informazioni, uno strumento per immagazzinare, manipolare e correlare dati. Siamo in grado di generare o raccogliere e in seguito elaborare informazioni in un campo di vastità prima impensabili. Queste informazioni ci possono aiutare a prendere delle decisioni, a capire il mondo in cui viviamo e a controllarne l'evoluzione.
Con l'aumento del volume delle informazioni sorge però un problema: come è possibile trasferire dalla macchina all'uomo queste informazioni in modo efficiente ed efficace? La macchina può agevolmente generare tabelle di numeri lunghe centinaia di pagine, ma una stampa di queste proporzioni può essere senza significato se chi la legge non ha il tempo per comprenderla. La Computer Graphics è orientata direttamente alla risoluzione di questo problema. Si tratta infatti di uno studio di tecniche atte a migliorare la comunicazione tra uomo e macchina.
Un grafico può sostituire quell'enorme tabella di numeri e permettere a chi lo legge di notarne gli elementi fondamentali e le caratteristiche in modo immediato. Dare al computer la possibilità di esprimere i suoi dati in forma grafica significa aumentare enormemente la capacità di fornire informazioni all'utente. Questo è l'aspetto passivo della grafica, ma la comunicazione può avvenire anche a doppio senso. Può essere infatti utile e conveniente fornire informazioni al computer in modo grafico. A questo scopo esistono dispositivi grafici sia di input che di output. Spesso occorre che l'input dell'utente modifichi l'output presentato dalla macchina: può essere così stabilito un dialogo attraverso il mezzo grafico. Tutto questo va sotto il nome di Computer Graphics interattiva, poiché l'utente interagisce con la macchina.
La Computer Graphics consente di comunicare tramite figure, schemi e dia-

grammi e offre un'alternativa stimolante alle stringhe di simboli della tastiera. L'antico adagio «una figura val più di mille parole» è certamente vero. Attraverso la C.G. si è in grado di pilotare navi spaziali, passeggiare tra palazzi che non sono ancora stati costruiti, assistere al crollo di ponti, alla nascita di una stella e alla rotazione degli atomi. Molte sono le applicazioni della C.G. Le informazioni gestionali possono essere visualizzate attraverso schemi e diagrammi; teorie e modelli scientifici possono essere descritti in forma grafica; nel campo della progettazione assistita dal calcolatore (*Computer Aided Design*, C.A.D.) si è in grado di disegnare l'ala di un velivolo, una mappa, un circuito stampato o la planimetria di un edificio; si possono creare mappe per qualsiasi tipo di informazione geografica; è possibile rendere più vivace una lezione tenuta in aula tramite diagrammi e simulazioni.
Il computer è diventato persino uno strumento per artisti e cineasti. Nei videogames, la C.G. fornisce una nuova forma di divertimento. Attraverso gli anni sono stati sviluppati molti dispositivi di visualizzazione grafica: esistono anche molti sistemi software e linguaggi per grafici. Il problema che sorge con una tale eterogeneità risiede nella difficoltà di trasferire un programma grafico da una installazione all'altra.
Recentemente comunque, il *Graphics Standards Planning Committee* della *Association for Computing Machinery* ha sviluppato uno standard chiamato sistema CORE. Questo sistema è progettato per essere indipendente da qualsiasi periferica grafica di input e di output e contiene le primitive grafiche di base dalle quali si possono derivare routine grafiche più complesse o specializzate. L'idea è quella di poter mandare in esecuzione un programma scritto per il sistema CORE su una qualsiasi delle installazioni possibili per il CORE. In questo libro verranno usati il sistema CORE e le sue estensioni come modello per gli algoritmi. Anche se ciò renderà talvolta le cose un po' più verbose o complicate del necessario, tali difetti saranno superati dai benefici derivanti dalla familiarizzazione con il sistema standard proposto.
La trattazione della Computer Graphics inizierà affrontando la questione fondamentale di come posizionare e visualizzare punti e segmenti.
Ci sono parecchi dispositivi hardware (terminali grafici) che possono essere usati per visualizzare immagini generate dal computer. Alcuni di questi verranno descritti nel Capitolo 2. Prima di parlare di dispositivi che visualizzano punti, è il caso di rivedere la geometria elementare che ne rappresenta il fondamento. Considereremo la natura di punti e rette e come possiamo specificarli e manipolarli. Concluderemo questo capitolo con una trattazione del modo in cui la descrizione matematica di questi elementi geometrici fondamentali può essere implementata su una periferica di visualizzazione. Verranno presentati gli algoritmi per realizzare un'implementazione su una stampante o su un comune terminale grafico che utilizzi un tubo a raggi catodici (*cathode ray tube*, CRT). Questi algoritmi permetteranno, se necessario, di usare la stampante o il CRT come un primitivo ma efficace dispositivo di visualizzazione grafica, per dimostrare gli elementi di grafica descritti nel resto del testo.

1.1 PUNTI E RETTE

Che cos'è un punto? Per rispondere, ci avvarremo della nozione intuitiva di posizione. Si è soliti pensare al punto come a una posizione nello spazio. Lo spazio è formato da un numero infinito di punti possibili. Ogni locazione corrisponde a uno e un solo punto (vedi Figura 1.1).
Quanto è grande un punto? Si dice che un punto sia infinitesimo: ciò significa che non ha dimensioni. Se volessimo vedere un oggetto molto piccolo ma finito, usando un microscopio si potrebbe aumentare l'ingrandimento finché l'oggetto appaia grande come vogliamo. Se si volesse invece vedere un punto al microscopio, esso ci apparirebbe sempre allo stesso modo. Il fattore di ingrandimento potrebbe essere aumentato a piacere e il punto resterebbe sempre un punto.
Dati due punti, quando possiamo dire che sono uguali? Solitamente si dice che due punti sono uguali se sono lo stesso punto. Due punti sono due posizioni nello spazio; se sono distinti, significa che c'è una certa distanza tra di loro. Questa distanza può essere molto piccola, ma se si esaminassero i due punti al microscopio, si potrebbe aumentare l'ingrandimento fino a notare chiaramente la separazione tra di essi. Se i due punti fossero uguali (cioè specificassero entrambi la stessa posizione nello spazio), qualsiasi ingrandimento non sarebbe in grado di renderli distinti. Quando due punti sono uguali non esistono nella realtà due punti, ma piuttosto due rappresentazioni o due nomi per un solo punto (vedi Figura 1.2).
Cos'è una retta? È possibile rispondere a questa domanda con la nozione intuitiva di direzione. Due punti distinti specificano una direzione (o meglio due direzioni: dal primo al secondo e viceversa).
Si consideri ora un terzo punto distinto dai primi due. Il primo punto e il terzo definiscono anch'essi una direzione; se questa direzione è la stessa definita dal primo e dal secondo punto, allora si dirà che tutti e tre i punti giacciono sulla stessa retta. La retta definita da due punti è l'insieme formato dai due punti e da tutti gli altri punti allineati con essi. Ciò equivale a dire che una retta può essere considerata come l'insieme di tutti i punti che giacciono su di essa (vedi Figura 1.3).

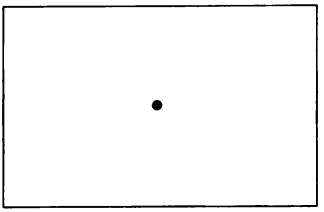


Figura 1.1 Un punto

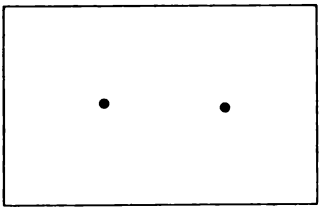


Figura 1.2 Due punti

Come possiamo definire un punto? Si consideri innanzitutto un caso semplice: come è possibile specificare un punto su una retta? Si supponga di avere due punti che vengono definiti come punto 0 e punto 1. Questi due punti definiscono una retta. Come può essere definito il punto medio tra i punti 0 e 1? Una definizione adeguata può essere punto 1/2 o 0.5, visto che è equidistante dai due punti considerati. Con lo stesso criterio il punto simmetrico del punto 1 rispetto al punto 0 potrebbe essere definito come -1 ed il punto che si trova nella stessa direzione del punto 1 ad una distanza doppia dal punto 0 può essere definito con 2. Si può quindi usare la distanza dal punto 0 come nome per ogni altro punto. Questo è esattamente quello che si fa utilizzando un righello. Il righello infatti specifica una posizione come la distanza lungo una retta da un particolare punto (il punto 0). Se sono definite le posizioni dei punti 0 e 1 è possibile specificare qualsiasi altro punto sulla retta da essi definita utilizzando la sua distanza (vedi Figura 1.4).

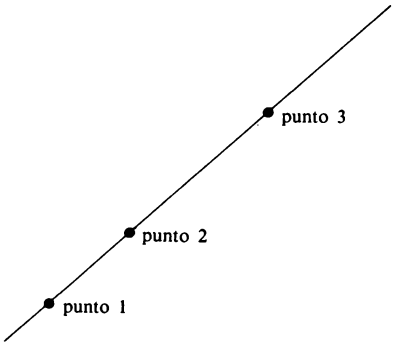


Figura 1.3 Una retta

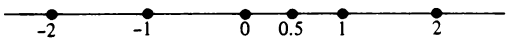


Figura 1.4 Definizione di punti su una retta

1.2 PIANI E COORDINATE

Che cos'è un piano? Una retta viene definita come un'entità monodimensionale dotata soltanto di lunghezza. Un piano è una entità bidimensionale dotata sia di lunghezza che di larghezza. Un piano può essere immaginato come un foglio di carta molto sottile e piatto che si estende a piacimento sia in lunghezza che in larghezza.

Come potrà essere specificato un punto su un piano? Si è visto come su di una retta (entità monodimensionale) occorra un solo numero per specificare un punto. La superficie di uno schermo o di un pezzo di carta è bidimensionale, dotata quindi sia di lunghezza che di larghezza; per poter specificare un punto su di essa è possibile ad esempio usare rette giacenti sulla superficie che si intersecano. Solitamente, per convenzione, queste rette vengono scelte perpendicolari, una orizzontale e l'altra verticale. La retta orizzontale viene definita come asse x e quella verticale come asse y. Analogamente al caso monodimensionale, queste rette vengono impiegate per misurare le distanze. L'asse x definisce la distanza orizzontale (a destra o a sinistra) e l'asse y definisce la distanza verticale (verso il basso o verso l'alto). Il punto dove le rette si intersecano è il punto 0 per entrambe ed è chiamato *origine*. Attraverso questo schema è possibile

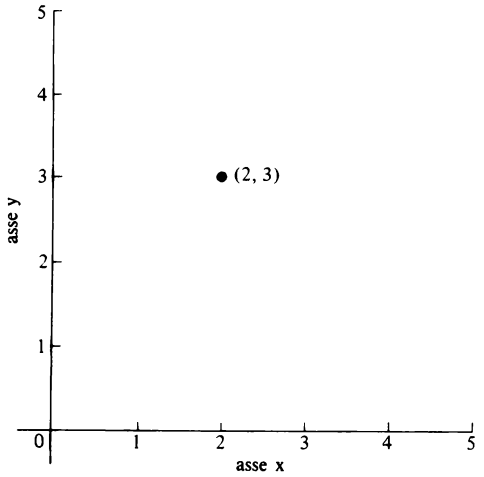


Figura 1.5 Coordinate cartesiane di un piano

specificare qualsiasi punto sul piano bidimensionale usando due numeri. Il primo numero (detto coordinata x) definisce quanto dista il punto dall'origine lungo l'asse x. Il secondo numero (detto coordinata y) definisce quanto dista il punto dall'origine lungo l'asse y. Le coordinate sono scritte sotto forma di coppie ordinate; perciò la coppia (2, 3) definisce il punto che si trova due unità a destra e tre sopra l'origine. Questa griglia rettangolare usata per specificare i punti è chiamata sistema di coordinate cartesiane (vedi Figura 1.5).

È possibile specificare una retta in termini di coordinate? La risposta è sì. Si consideri un generico punto (x, y) dove x è la coordinata x e y è la coordinata y. È possibile scrivere un'equazione in x e y che sia soddisfatta se e solo se (x, y) è un punto appartenente alla retta. Per la definizione della retta, ricordando che sono necessari due punti distinti, si considerino i punti: $p_1 = (x_1, y_1)$ e $p_2 = (x_2, y_2)$ (vedi Figura 1.6). Si consideri poi un terzo punto $p = (x, y)$ giacente sulla retta. Per facilitare la dimostrazione possiamo costruire due triangoli. Innanzitutto si traccia una retta orizzontale passante per p_1, i cui punti avranno la stessa distanza dall'asse x pari al valore della coordinata y_1 del punto p_1. Si conduce poi una retta verticale passante per il punto p_2 (i cui punti hanno la coordinata $x = x_2$) che intersecherà la retta orizzontale nel punto $A = (x_2, y_1)$. Una retta verticale passante per p individua il punto $B = (x, y_1)$ di intersezione con la retta orizzontale. Consideriamo ora i due triangoli $p_1 p_2 A$ e $p_1 pB$. Si tratta di due triangoli simili poiché hanno il vertice p_1 in comune e gli angoli in A e B sono entrambi retti. Il rapporto dei loro lati corrispondenti è quindi costante; si avrà cioè che, in particolare, l'altezza $p_2 A$ sta all'altezza pB come la base $p_1 A$ sta alla base $p_1 B$. Si può così determinare l'altezza $p_2 A$ come la differenza delle coordinate y dei due punti p_2 e A. Si scrive quindi:

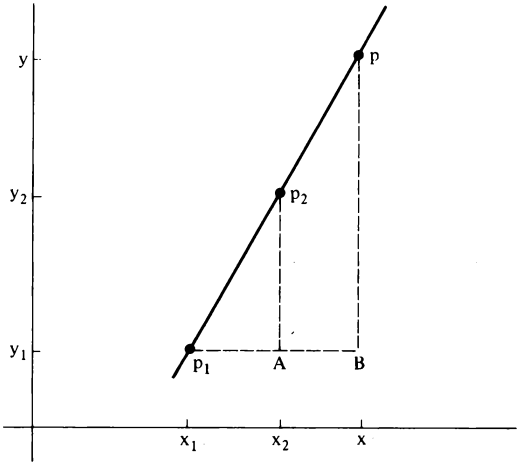


Figura 1.6 Definizione di una retta nel piano

$$\text{altezza } p_2A = y_2 - y_1$$

e analogamente

$$\text{altezza } pB = y - y_1$$
$$\text{base } p_1A = x_2 - x_1$$
$$\text{base } p_1B = x - x_1$$

Da queste si ottiene

$$\frac{y_2 - y_1}{y - y_1} = \frac{x_2 - x_1}{x - x_1} \tag{1.1}$$

o, moltiplicando entrambi i membri dell'equazione per il comun denominatore,

$$(x - x_1)(y_2 - y_1) = (y - y_1)(x_2 - x_1) \tag{1.2}$$

Questa è un'equazione della retta passante per i punti (x_1, y_1) e (x_2, y_2). Un ulteriore passaggio algebrico permette di risolvere rispetto ad y l'equazione (1.2) ottenendo

$$y = \frac{(y_2 - y_1)}{(x_2 - x_1)}(x - x_1) + y_1 \tag{1.3}$$

o

$$y = mx + b \tag{1.4}$$

dove si è posto

$$m = \frac{(y_2 - y_1)}{(x_2 - x_1)} \tag{1.4.1}$$

e

$$b = y_1 - mx_1 \tag{1.4.2}$$

L'equazione (1.4) viene detta in forma esplicita, il parametro m rappresenta la pendenza della retta, cioè l'incremento della coordinata y corrispondente ad un incremento unitario della coordinata x. Il parametro b costituisce invece la coordinata y del punto in cui la retta interseca l'asse y (intercetta). Questo può essere verificato semplicemente osservando che il punto $(0, b)$ soddisfa l'equazione della retta.

È possibile, a questo punto, determinare dove si intersecano due rette? Osserviamo innanzitutto che quando si dice che le due rette si incontrano, si intende che esse abbiano un punto in comune. Le coordinate del punto saranno tali da soddisfare le equazioni di entrambe le rette. Il problema è la determinazione dei

valori di queste coordinate. Si supponga di conoscere le equazioni delle due rette in forma esplicita:

$$\begin{aligned} &\text{retta 1:} \quad y = m_1 x + b_1 \\ &\text{retta 2:} \quad y = m_2 x + b_2 \end{aligned} \tag{1.5}$$

Ora, se esiste un punto (x_i, y_i) comune alle due rette, allora

$$y_i = m_1 x_i + b_1 \qquad \text{e} \qquad y_i = m_2 x_i + b_2 \tag{1.6}$$

saranno entrambe verificate. Imponendo l'uguaglianza dei valori di y_i si ottiene

$$m_1 x_i + b_1 = m_2 x_i + b_2 \tag{1.7}$$

e risolvendo rispetto a x_i

$$x_i = \frac{b_2 - b_1}{m_1 - m_2} \tag{1.8}$$

Sostituendo infine la (1.8) nell'equazione di una delle due rette (1.5), si ricava

$$y_i = \frac{b_2 m_1 - b_1 m_2}{m_1 - m_2} \tag{1.9}$$

Quindi il punto di coordinate

$$\left(\frac{b_2 - b_1}{m_1 - m_2}, \frac{b_2 m_1 - b_1 m_2}{m_1 - m_2} \right) \tag{1.9.1}$$

è il punto di intersezione. Si noti che due rette parallele avranno lo stesso coefficiente angolare e quindi non ci sarà alcun punto di intersezione. In questo caso accade che il denominatore della (1.9) è uguale a 0. Si può concludere perciò dicendo che, quando non è possibile risolvere questa equazione, significa che non esiste alcun punto di intersezione tra le rette considerate.

1.3 SEGMENTI

Che cos'è un segmento? Attraverso le equazioni di una retta si è in grado di definire tutti i punti appartenenti ad una data direzione. Le rette però si estendono all'infinito sia in un senso che nell'altro e non è certo questo ciò di cui abbiamo bisogno per ottenere rappresentazioni grafiche con un calcolatore. Nelle nostre applicazioni infatti occorre rappresentare soltanto porzioni di retta, solo quei punti cioè compresi tra due estremi p_1 e p_2 (vedi Figura 1.7). Defi-

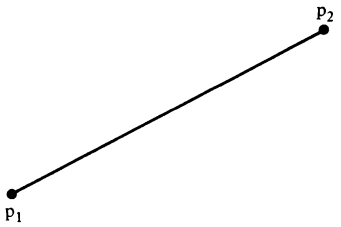


Figura 1.7 Segmento di retta

niamo quindi un segmento tramite i suoi due estremi, dai quali è possibile risalire all'equazione della retta passante per essi. Da questa equazione e dalle coordinate degli estremi è possibile stabilire se un punto si trova sul segmento oppure no. Si supponga che gli estremi siano i punti $p_1 = (x_1, y_1)$ e $p_2 = (x_2, y_2)$ e l'equazione della retta sia $y = mx + b$; allora un terzo punto $p_3 = (x_3, y_3)$ giace su questo segmento se si verificano le seguenti tre condizioni:

1. $y_3 = mx_3 + b$
2. $\min(x_1, x_2) \leq x_3 \leq \max(x_1, x_2)$
3. $\min(y_1, y_2) \leq y_3 \leq \max(y_1, y_2)$

La notazione $\min(x_1, x_2)$, sta ad indicare il minore tra x_1 e x_2 e $\max(x_1, x_2)$ indica invece il maggiore tra i due numeri.
Quanto è lungo un segmento? Se di un segmento sono note le coordinate dei due estremi p_1 e p_2, è possibile determinarne la lunghezza L. Per far ciò deve essere costruito un triangolo rettangolo p_1p_2A ottenuto facendo passare una retta verticale per p_2 ed una orizzontale per p_1 (vedi Figura 1.8). Il teorema di

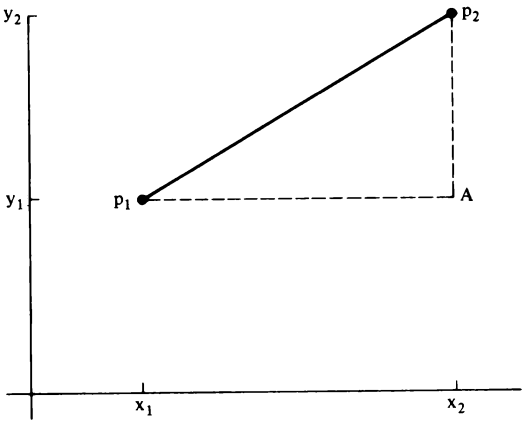


Figura 1.8 Lunghezza di un segmento

Pitagora stabilisce che il quadrato della lunghezza dell'ipotenusa di un triangolo rettangolo (p_1p_2) è uguale alla somma dei quadrati delle lunghezze dei due cateti (p_1A e p_2A). Siano (x_1, y_1) le coordinate di p_1 e (x_2, y_2) quelle di p_2: il punto A avrà coordinate (x_2, y_1). La lunghezza del segmento sarà allora definita come

$$L^2 = (x_2-x_1)^2+(y_2-y_1)^2 \tag{1.10}$$

dalla quale si ricava

$$L = [(x_2-x_1)^2+(y_2-y_1)^2]^{1/2} \tag{1.11}$$

Che cos'è il punto medio di un segmento? Il punto medio di un segmento è spesso utile ed è facile da calcolare. Si osservi intuitivamente che il punto medio tra gli estremi di un segmento avrà la coordinata x media fra le coordinate x degli estremi e la coordinata y media fra le coordinate y (vedi Figura 1.9). Da ciò le coordinate del punto medio sono

$$(x_m, y_m) = \left(\frac{(x_1+x_2)}{2}, \frac{(y_1+y_2)}{2}\right) \tag{1.12}$$

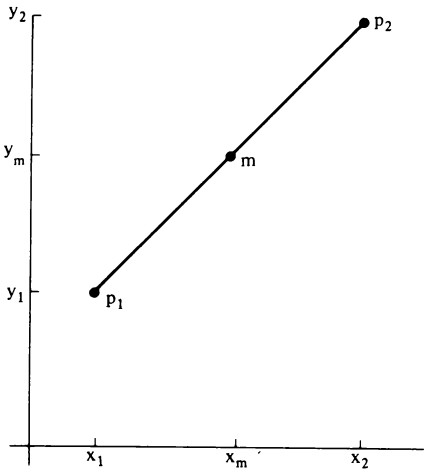


Figura 1.9 Punto medio di un segmento

1.4 RETTE PERPENDICOLARI

Possiamo stabilire se due rette sono perpendicolari? È possibile rispondere a questa domanda esaminando i loro coefficienti angolari. Si considerino le due rette di equazioni

$$\begin{aligned} y &= m_1 x + b_1 \\ y &= m_2 x + b_2 \end{aligned} \tag{1.13}$$

Se la prima retta è perpendicolare alla seconda, allora una retta parallela alla prima (cioè una retta con la stessa pendenza) sarà anch'essa perpendicolare alla seconda. Per esempio, la retta $y = m_1 x$ dovrà essere perpendicolare alla retta $y = m_2 x + b_2$. Lo stesso ragionamento si può applicare in modo equivalente alla seconda retta concludendo che $y = m_2 x$ rappresenta una retta perpendicolare a quella di equazione $y = m_1 x$. Queste due rette si intersecano nell'origine (vedi Figura 1.10).

Si consideri ora un punto (x_1, y_1) sulla retta $y = m_1 x$ tale che sia $y_1 = m_1 x_1$ e un secondo punto (x_2, y_2) sulla retta $y = m_2 x$ per il quale risulti $y_2 = m_2 x_2$. I tre punti di coordinate (x_1, y_1), (x_2, y_2), e (0, 0) formano un triangolo che sarà rettangolo se le due rette in esame sono perpendicolari. Dal teorema di Pitagora è possibile affermare che la somma dei quadrati delle distanze tra i punti (0, 0) e (x_1, y_1) e tra i punti (0, 0) e (x_2, y_2) è uguale al quadrato della distanza

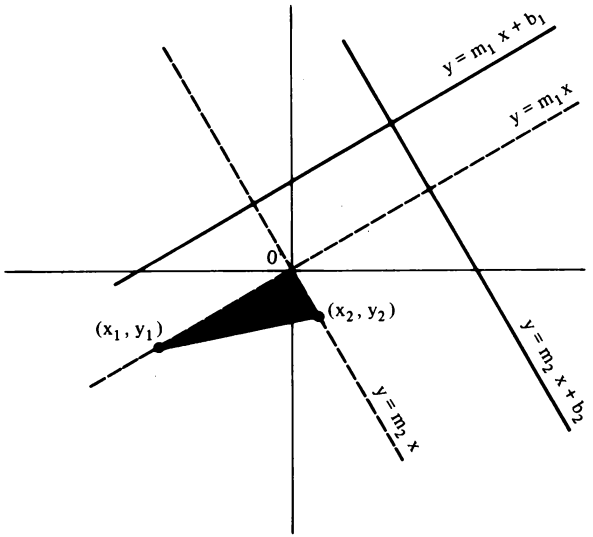


Figura 1.10 Costruzione per il test di perpendicolarità fra rette

tra i punti di coordinate (x_1, y_1) e (x_2, y_2). Applicando la formula delle distanze tra i punti otteniamo

$$x_1^2 + y_1^2 + x_2^2 + y_2^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2 \tag{1.14}$$

che semplificata risulta

$$-2y_1y_2 = -2x_1x_2 \tag{1.14.1}$$

o, più elegantemente

$$\frac{y_1}{x_1} = -\frac{x_2}{y_2} \tag{1.15}$$

Ricordando infine che $y_1 = m_1x_1$ e che $y_2 = m_2x_2$ otteniamo la relazione che lega i coefficienti angolari di due rette perpendicolari:

$$m_1 = -\frac{1}{m_2} \tag{1.16}$$

È possibile perciò concludere dicendo che se due rette sono perpendicolari, il coefficiente angolare della prima è uguale al reciproco dall'altro cambiato di segno.

1.5 VETTORI

Che cos'è un vettore? Un vettore è definito dall'insieme di una direzione, di un verso e di una lunghezza. Un vettore può essere descritto dalla coppia di numeri $[D_x, D_y]$ dove D_x indica di quanto muoversi nella direzione dell'asse x e D_y indica di quanto muoversi nella direzione dell'asse y. Assumiamo qui che il ver-

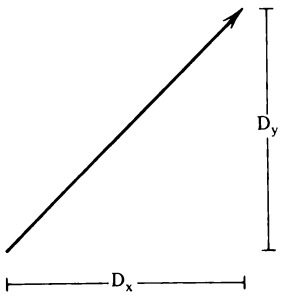


Figura 1.11 Vettore

so sia concorde con quello degli assi x e y, cioè che D_x sia positivo per uno spostamento nel verso dell'asse x e ugualmente per D_y (vedi Figura 1.11). Diversamente dai segmenti, i vettori non hanno una posizione fissa nello spazio, infatti indicano una distanza e una direzione in cui muoversi ma non da dove si deve partire. L'idea di vettore è utile ai nostri scopi in quanto rappresenta correttamente il modo in cui una penna si muove sulla carta per disegnare un segmento di retta. Il comando di muovere una penna può essere infatti indicato specificando la distanza da percorrere in una certa direzione. Nel Capitolo 4 verranno usate le trasformazioni sui vettori per cambiare il fattore di scala delle figure, traslarle e ruotarle.

1.6 PIXEL E FRAME BUFFER

Come si applica tutto ciò che è stato definito nei paragrafi precedenti ad un terminale grafico? Si osservi prima di tutto che la nozione di retta come numero infinito di punti dalle dimensioni infinitesime è ovviamente inapplicabile a un dispositivo reale. Con un calcolatore non è possibile cioè rappresentare un numero infinito di punti per lo stesso motivo per cui non è possibile rappresentare tutta l'infinità dei numeri. Il calcolatore è una macchina con una memoria finita e quindi è in grado di memorizzare solo un numero finito di punti per comporre una retta (solitamente non più di poche centinaia o migliaia di punti). Il massimo numero di punti distinti che compongono una retta è una misura della *risoluzione* del dispositivo di visualizzazione. Maggiore è il numero di punti e proporzionalmente maggiore è la risoluzione. Il limite del numero di punti memorizzabili non ci deve preoccupare, a causa delle limitate capacità di risoluzione dell'occhio umano che non è in grado di distinguere i dettagli con una precisione altissima. Dovendo costruire le rette utilizzando un numero finito di punti, ciascuno di essi dovrà avere una data dimensione; ciò significa che non si tratta di punti nel senso in cui sono stati definiti nei paragrafi precedenti. Questa nuova entità viene detta *pixel* (abbreviazione di *picture element*). Il pixel è l'elemento più piccolo visualizzabile sullo schermo di un terminale o su un foglio di carta. Esso è quindi la porzione più piccola di schermo che possiamo controllare, individuandola attraverso un indirizzo costituito dalle sue coordinate, che permettono di identificare il punto. Questo meccanismo è simile al modo con cui gli indici permettono di individuare un elemento in un array. Tutte le immagini possono essere generate assegnando l'intensità luminosa e il colore ai pixel dello schermo. I segmenti di retta, ad esempio, si possono disegnare assegnando un'intensità — cioè una luminosità — ai pixel che stanno tra il pixel iniziale e quello finale del segmento e che corrispondono ai suoi estremi. Si potrebbe pensare allo schermo come a una griglia o un array bidimensionale di pixel individuabili assegnando i valori interi delle corrispondenti coordinate. Per convenzione si numerano le colonne partendo da sinistra con il primo elemento uguale a 1 e le righe partendo dal basso (vedi Figura 1.12).

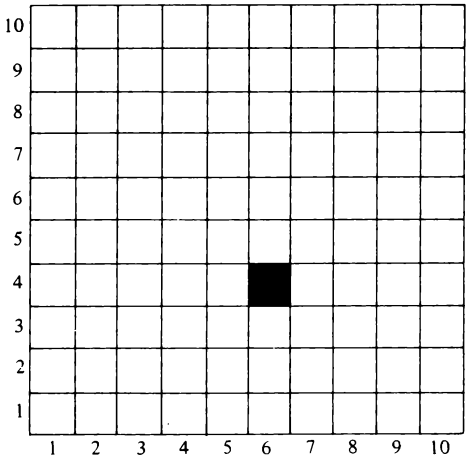


Figura 1.12 Pixel

I valori dell'intensità luminosa di ciascun pixel possono essere memorizzati in un array all'interno della memoria del calcolatore. Il dispositivo grafico utilizzerà questo array per determinare l'intensità con la quale deve essere visualizzato ciascuno dei pixel che costituiscono una data figura. Questo array, che contiene una rappresentazione interna dell'immagine, viene detto *frame buffer*. La sua funzione è di memorizzare i pixel che si aggiungono di volta in volta fino a formare l'immagine completa, durante i calcoli necessari alla loro determinazione.

1.7 GENERAZIONE DI VETTORI

Il processo di «accensione» dei pixel di un segmento di retta viene detto *generazione di vettori*. Se sono noti gli estremi che definiscono un segmento, come si potrà decidere quali siano i pixel tra i due estremi ai quali si deve modificare l'intensità luminosa al fine di renderli visibili? Ci sono diversi metodi per effettuare questa selezione di pixel. Quello presentato in questo capitolo è detto analizzatore differenziale simmetrico e digitale (*symmetrical digital differential analyzer*, DDA). Il nome deriva dal fatto che viene usata la stessa tecnica applicata in un metodo numerico per la soluzione di equazioni differenziali. L'equazione differenziale che dobbiamo risolvere è quella che definisce una retta ed è quindi particolarmente semplice. Inoltre, invece di calcolare la soluzione, si richiede di disegnarla.

Si consideri una diversa forma dell'equazione di una retta, detta forma *parametrica*, nella quale i valori delle coordinate x e y sono definiti in termini di un

solo parametro u. Supponendo che sia richiesto di tracciare il segmento compreso tra gli estremi (x_1, y_1) e (x_2, y_2), per ottenere i punti appartenenti al segmento è necessario che le coordinate di questi punti varino uniformemente tra x_1 e x_2. Ciò può essere espresso con l'equazione

$$x = x_1 + (x_2 - x_1)u \tag{1.17}$$

Quando u vale 0, x è uguale a x_1; se u viene fatto variare incrementandolo fino al valore 1, x varia con continuità fino al valore x_2. Per un segmento di retta, si desidera che le coordinate y si muovano da y_1 a y_2 in modo analogo a quanto si è detto per le x. Si ottiene quindi

$$y = y_1 + (y_2 - y_1)u \tag{1.18}$$

Le due equazioni insieme descrivono una retta. È possibile dimostrarlo facilmente risolvendo rispetto a u. Con pochi passaggi algebrici si ricostruisce l'equazione della retta di partenza nella forma nota. Queste due equazioni suggeriscono un metodo per selezionare i pixel ai quali si deve cambiare l'intensità, che consiste nel partire con $u = 0$ e incrementarlo di piccole quantità fino ad arrivare a 1. Mentre si esegue questa operazione, x e y si muovono lungo il segmento, anch'essi di piccoli passi come è specificato dalle equazioni parametriche, assegnando l'intensità al pixel che contiene il punto di coordinate (x, y). La routine DDA parte da uno degli estremi del segmento e per incrementi successivi percorre il segmento fino al secondo estremo. In ciascun passo viene assegnata l'intensità al pixel che corrisponde al punto della retta che viene calcolato. L'algoritmo DDA dovrà partire dalla posizione iniziale (X1, Y1), e avrà la posizione finale in (X2, Y2), inoltre riceverà attraverso il parametro INTENSITY il valore dell'intensità luminosa che deve essere assegnata al pixel. L'algoritmo DDA è il seguente:

Algoritmo 1.1 DDA (X1, Y1, X2, Y2, INTENSITY)
Cambia il valore dell'intensità dei pixel all'interno del frame buffer.

Argomenti X1, Y1 ← coordinate del punto di partenza
X2, Y2 ← coordinate del punto di arrivo
INTENSITY ← valore dell'intensità dei pixel

Variabili globali FRAME ← array bidimensionale che costituisce il frame buffer

Variabili locali DX, DY ← componenti del vettore da disegnare
STEPS ← numero di passi lungo il vettore nei quali viene assegnata l'intensità ai pixel
DX-STEP, DY-STEP ← dimensioni del passo
X, Y ← punto generico sul segmento
U ← contatore dei passi

begin
determina le componenti del vettore
DX ← X2 − X1;

```
    DY←Y2−Y1;
    calcola il numero dei passi
    STEPS←INT(MAX(|DX|, |DY|))+1;
    calcola le dimensioni del passo
    DX-STEP←DX/STEPS;
    DY-STEP←DY/STEPS;
    incrementa la posizione iniziale per arrotondamento
    X←X1+0.5;
    Y←Y1+0.5;
    for U = 1 to STEPS
      do begin
          assegna il pixel più vicino nel frame buffer
          FRAME [INT(X), INT(Y)]←INTENSITY;
          passa al punto successivo
          X←X+DX-STEP;
          Y←Y+DY-STEP;
      end;
    assegna l'ultimo pixel
    FRAME [INT(X), INT(Y)]←INTENSITY;
    return;
end;
```

La prima parte di questo algoritmo serve al calcolo del numero di passi da effettuare, determinando le due componenti sugli assi x e y [DX, DY] del segmento da disegnare. Questo vettore va diviso in un numero sufficiente di passi, in modo che ciascuno di essi non superi la dimensione di un pixel. Se si eseguono pochi passi si avranno degli intervalli vuoti nel segmento in corrispondenza delle regioni nelle quali non si sono calcolati i punti. La lunghezza di un segmento di retta va determinata come numero di pixel e quindi il numero minimo di passi necessari è dato dalla lunghezza in pixel della componente x o y del vettore. È possibile anche eseguire un numero di passi maggiore del valore minimo così determinato: in questo caso si ottiene una retta dallo spessore più grande. Inoltre, assegnando più volte un valore ad uno stesso pixel, diminuisce l'efficienza dell'operazione, fatto comunque meno grave che non avere interruzioni visibili nella continuità del segmento (vedi Figura 1.13).
Un passo di incremento pari a un pixel è la soluzione ottimale, ma nell'algoritmo precedentemente descritto si è scelto di avere un numero di passi maggiore. La ragione che spinge a questa scelta è l'impossibilità, causata dagli errori di arrotondamento, di valutare esattamente il valore del passo per eseguire spostamenti di un pixel. Tali errori possono causare interruzioni nella continuità della retta.
Un'altra operazione influenzata dagli arrotondamenti è il calcolo della lunghezza delle componenti DX e DY del vettore. Se queste assumono i valori dell'intero inferiore (a causa dell'arrotondamento e dei troncamenti che vengono applicati per la conversione dei valori reali ad interi) allora il numero di passi

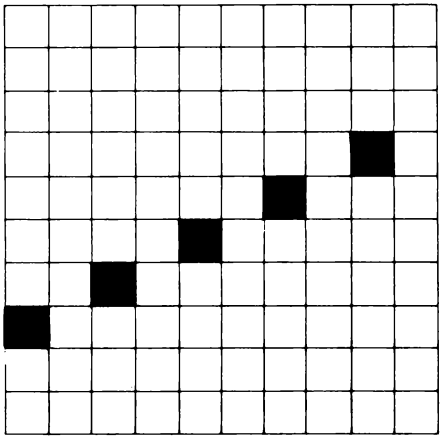


Figura 1.13a Passi insufficienti

(STEPS) può risultare inferiore di una unità rispetto al valore che dovrebbe assumere. Una soluzione a questo problema è calcolare la variabile STEPS come INT(MAX(|DX|,|DY|)+ROUNDOFF) dove ROUNDOFF è un numero positivo piccolo ma maggiore del più grande errore di arrotondamento.
Dopo aver valutato il numero di passi, si dividono le componenti x e y per il passo stesso al fine di ottenere le lunghezze degli incrementi DX-STEP e DY-STEP. Per ogni incremento di u, x aumenterà di una quantità pari a DX-STEP

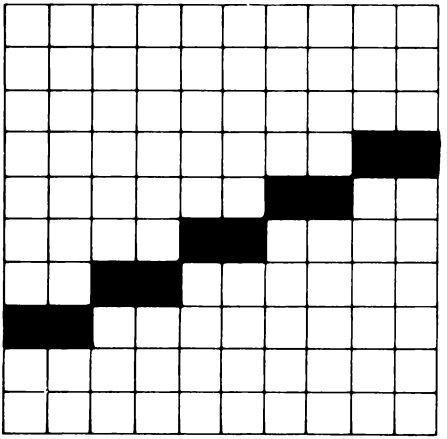


Figura 1.13b Passi sufficienti

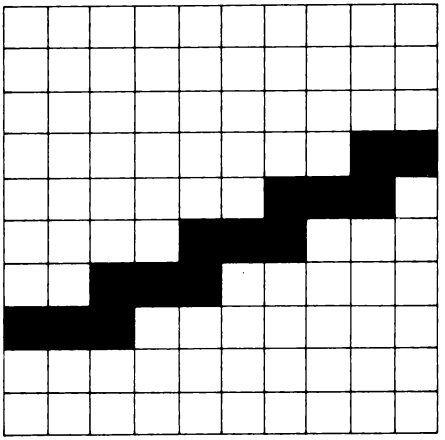


Figura 1.13c Passi più che sufficienti

e y di DY-STEP; in questo modo non dobbiamo usare esplicitamente le equazioni (1.17) e (1.18) nel nostro algoritmo. Infatti, aumentando un poco u e calcolando x e y attraverso quelle equazioni, si scopre che x e y sono anch'essi aumentati di poco, ma se u cambia della stessa quantità ad ogni passo, altrettanto faranno x e y. Avendo compreso questo è possibile dimenticarsi di u e sommare direttamente gli incrementi costanti a x e y, come è stato fatto nell'algoritmo presentato.

Alcuni calcolatori meno potenti possono essere privi di metodi efficienti per il calcolo delle divisioni. Per tali macchine l'efficienza può essere mantenuta dividendo per potenze di 2, operazione che può essere realizzata con uno spostamento di bit. Come si è visto, prendendo un maggior numero di passi non si danneggia la qualità della retta, per cui è possibile modificare l'algoritmo ponendo STEPS pari alla più piccola potenza di 2 maggiore del più grande tra DX e DY.

L'algoritmo usa la funzione INT per convertire le coordinate reali in indici interi dei pixel. Assumiamo qui che INT tronchi al più grande intero minore o uguale al valore reale dato; ad esempio,

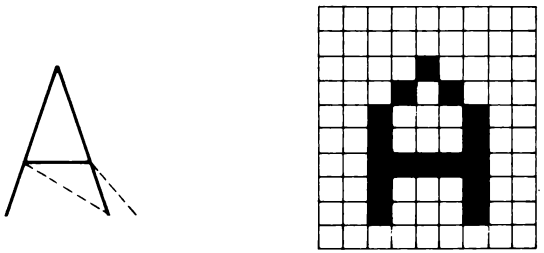
$$\text{INT}(3.3) = 3$$
$$\text{INT}(2.7) = 2$$
$$\text{INT}(-1.2) = -2$$

Troncando in questo modo al fine di calcolare le coordinate dei pixel, questi si troveranno sempre leggermente al di sotto della retta vera e propria. Una soluzione più corretta consiste nel selezionare pixel che stiano per metà al di sotto e per metà al di sopra della retta. Ciò può essere fatto con un arrotondamento

anziché con un troncamento nella conversione tra reale ed intero. L'effetto dell'arrotondamento può essere ottenuto ugualmente sommando 0.5 prima del troncamento. Si assegna quindi ad X e Y il valore, incrementato di 0.5, dei corrispondenti X1 e Y1, assicurando così che il troncamento produca lo stesso effetto di un arrotondamento.
Di seguito, nell'algoritmo, si trova un ciclo che cambia lo stato dei pixel nel frame buffer assegnando l'intensità luminosa desiderata per il vettore che stiamo tracciando e che quindi incrementa X e Y per il punto successivo. Il ciclo inizia con un estremo del segmento e termina all'altro estremo quando si raggiungono i valori X2 e Y2; l'ultimo pixel non viene però assegnato durante l'esecuzione del ciclo ma subito dopo la fine del ciclo.
Nel caso in cui la velocità sia un parametro molto importante del nostro sistema grafico, gli algoritmi per la generazione di vettori (come quello che abbiamo visto), possono essere implementati anche in hardware o firmware. Questo viene fatto solitamente per quei dispositivi che, per evitare il costo della memoria di un grande frame buffer, fanno in modo che il generatore di vettori controlli direttamente il dispositivo tracciante (solitamente un pennello elettronico in un tubo a raggi catodici), visualizzando i pixel direttamente sullo schermo. Invece di assegnare un valore ad un elemento (x, y) del frame buffer, la penna o il pennello elettronico vengono mossi fino alla posizione (x, y) illuminando quindi un punto dello schermo.

1.8 GENERAZIONE DI CARATTERI

Oltre alle rette e ai punti, spesso devono essere visualizzate anche sequenze di caratteri utili per etichettare e arricchire con annotazioni i disegni e che costituiscono istruzioni e informazioni per chi legge il disegno. I caratteri sono solitamente costruiti all'interno del dispositivo grafico, spesso cablati nell'hardware; altre volte sono memorizzati e trattati nel software per mezzo di programmi.
Ci sono due metodi principali per la generazione di caratteri. Il primo, che viene detto metodo *a vettori* (*stroke*), compone i caratteri per mezzo di segmenti di retta, come i segmenti tracciati da una penna. È possibile implementare nel nostro sistema questo metodo generando i caratteri attraverso chiamate alla routine DDA. Basterà decidere quali segmenti di retta sono necessari per ciascun carattere e richiamare con opportuni parametri la routine DDA per i caratteri che si desidera visualizzare. Questo metodo si presta ad essere implementato sia come hardware che come software. Il metodo a vettori permette di cambiare facilmente il fattore di scala dei caratteri che possono essere ad esempio raddoppiati, moltiplicando semplicemente per 2 la lunghezza di ogni segmento che compone il carattere.
Il secondo metodo per la generazione dei caratteri è il metodo a *matrice di punti*. In questo schema i caratteri vengono rappresentati per mezzo di un array di punti che spesso è largo 5 punti e alto 7, mentre altre volte si trovano dimen-



(*a*) (*b*)

Figura 1.14 Generazione di un carattere: (*a*) metodo a vettori, (*b*) metodo a matrice di punti

sioni di 7×9 e 9×13. Questo array è un piccolo frame buffer, grande abbastanza per contenere un solo carattere. I punti che compongono il carattere sono i pixel di questa piccola matrice. Il posizionamento di un carattere sullo schermo è una operazione che consiste nel copiare lo stato dei pixel di un certo carattere in una porzione del frame buffer (solitamente per i comuni terminali alfanumerici la matrice di punti serve per controllare direttamente l'intensità di piccole parti dello schermo, eliminando la necessità di avere un grande frame buffer). La memoria che contiene la matrice di punti di un carattere è di solito un dispositivo hardware chiamato «chip per la generazione dei caratteri». Le dimensioni della matrice sono fisse e ciò non consente quindi di variare le dimensioni dei caratteri.

1.9 VISUALIZZAZIONE DEL FRAME BUFFER

Si è considerato nei paragrafi precedenti come l'array che costituisce il frame buffer venga modificato per contenere segmenti di retta e caratteri. Questo array è in corrispondenza diretta con lo schermo e memorizza una forma codificata dei valori delle intensità luminose dei punti che compongono la figura da visualizzare. Al fine di visualizzare l'immagine, partendo dal contenuto del frame buffer, sono necessari alcuni passaggi.

Per incominciare esso va ripulito: occorre iniziare con un array vuoto così come per fare un disegno occorre un foglio di carta bianca.

Un'altra operazione necessaria è il trasferimento dell'informazione contenuta nel frame buffer sul supporto di visualizzazione (schermo o foglio di carta). Per i terminali grafici di tipo raster questa operazione è cablata nell'hardware del terminale. Il frame buffer è continuamente scandito dall'hardware per comporre l'immagine ed ogni modifica del frame buffer viene immediatamente mostrata sullo schermo. Nel caso di visualizzazione della figura su stampante invece non si dispone di hardware che visualizzi automaticamente il contenuto del

frame buffer; in questo caso le operazioni per la visualizzazione devono essere eseguite da una routine software.
Infine possono essere necessarie delle routine per inizializzare e per disattivare il sistema grafico. Quando inizia un programma grafico devono essere allocati i dispositivi hardware e la memoria necessaria e le variabili devono essere inizializzate da una routine opportuna. Per quanto riguarda la conclusione del lavoro è necessaria una routine che si occupi della deallocazione della memoria, del rilascio dei dispositivi hardware e di altre operazioni di gestione delle risorse.
A completamento di questo capitolo verranno descritti gli algoritmi necessari per ottenere la visualizzazione delle informazioni grafiche su una stampante o su un comune terminale CRT. Il primo algoritmo — il generatore di vettori (DDA) — è già stato presentato. Per tutte queste routine avremo bisogno di un array che rappresenti il frame buffer contenente i pixel dell'immagine. Le dimensioni di questo array dipendono dalla risoluzione dei dispositivi di visualizzazione. Per un CRT con 24 linee di 80 caratteri, il frame buffer dovrà essere dimensionato come FRAME[80,24]. Questa non è però la sola scelta possibile: molti dispositivi sono tali per cui l'assegnamento del pixel che si trova nell'angolo inferiore destro causa uno scorrimento automatico dell'immagine verso l'alto con la corrispondente perdita della linea superiore. Ciò può essere evitato non utilizzando la linea inferiore e quindi dimensionando FRAME[80,23]. D'altra parte è possibile che lo schermo non sia quadrato e che, per poter trattare egualmente le due coordinate x e y, si decida di selezionare una sottoarea quadrata che corrisponde ad un dimensionamento di FRAME[60,23].
Nel caso di una stampante, potremo usare un array definito come FRAME[90,50] o anche più grande, a seconda della memoria disponibile o del numero di caratteri per linea; il frame buffer sarà allora un array di caratteri. La variabile INTENSITY utilizzata dall'algoritmo 1.1 sarà un carattere, come ad esempio il punto o l'asterisco che verranno scelti a seconda della figura da costruire. Per avere un frame buffer vuoto o «bianco», inizializzato per contenere una nuova figura, l'array dovrà essere riempito di caratteri bianchi. Questo viene eseguito dal seguente algoritmo.

Algoritmo 1.2 ERASE
Ripulisce il frame buffer assegnando a ciascun pixel un valore di sfondo.

Variabili globali FRAME←array bidimensionale del frame buffer
WIDTH-START, HEIGHT-START←indici di partenza di FRAME
WIDTH-END, HEIGHT-END←indici finali di FRAME
Variabili locali X, Y←indici del frame buffer

```
begin
   for Y = HEIGTH-START to HEIGHT-END
   do for X = WIDTH-START to WIDTH-END
   do FRAME [X, Y]←' ';
   return;
end;
```

Una chiamata alla routine ERASE ripulisce lo schermo e può essere utile ogni volta che si desideri disegnare una nuova figura.
La routine DISPLAY viene usata per mostrare il contenuto del frame buffer su una stampante. Si osservi che, mentre le coordinate *y* partono dal basso e si incrementano verso l'alto della figura, le stampanti scrivono partendo dall'alto della pagina verso il basso. Ciò significa che le coordinate *y* devono essere visualizzate partendo dal valore più grande verso il più piccolo.

Algoritmo 1.3 DISPLAY
Visualizza il contenuto del frame buffer.
Variabili globali FRAME←array del frame buffer
WIDTH-START, HEIGHT-START←indici di partenza di FRAME
WIDTH-END, HEIGHT-END←indici finali di FRAME
Variabili locali X, Y←pixel da visualizzare

```
begin
   for Y = HEIGHT-END to HEIGHT-START
         do print for X = WIDTH-START to WIDTH-END,
         FRAME[X, Y];
   return;
end;
```

Per completare questo sistema grafico, si deve includere una routine che inizializza tutti i parametri per la dimensione del dispositivo ed esegue ogni operazione di inizializzazione dipendente dal dispositivo. Tra le varie inizializzazioni ci dovrà essere l'assegnamento dei parametri per il dimensionamento del frame buffer e la pulizia del dispositivo di visualizzazione.

Algoritmo 1.4 INITIALIZE-1
Variabili globali WIDTH-START, HEIGHT-START←indici di partenza di FRAME
WIDTH-END, HEIGHT-END←indici finali di FRAME
WIDTH, HEIGTH←dimensioni di FRAME

```
begin
   esegue ogni allocazione di memoria, assegnamento di dispositivi hardware e ogni altra operazione di gestione delle risorse necessarie;
   HEIGHT-START←indice di partenza per le colonne di FRAME
   WIDTH-START←indice di partenza per le righe di FRAME
   HEIGHT-END←indice finale per le colonne di FRAME
   WIDTH-END←indice finale per le righe di FRAME
   HEIGHT←HEIGHT-END−HEIGHT-START;
   WIDTH←WIDTH-END−WIDTH-START;
   ERASE;
   return;
end;
```

Algoritmo 1.5 TERMINATE
begin
rilascia tutti i dispositivi hardware assegnati;
esegue tutte le necessarie operazioni finali;
stop;
end;

ESERCIZI

1.1 Disegnare gli assi coordinati e i seguenti punti:
a) (1, 1)
b) (2, 3)
c) (−1, 0)
d) (2, −1)
e) (1.5, 2)

1.2 Scrivere l'equazione in forma esplicita delle rette passanti per le coppie di punti sotto elencate:
a) (0, 0); (1, 1)
b) (0, 0); (6, 2)
c) (1, 0); (7, 2)
d) (0, 1); (6, 3)
e) (2, 3); (4, 2)
f) (1, 1); (2, 3)
g) (1, 1); (3, 2)
h) (0, 2.5); (−1, 3.5)
i) (0, 7); (−2, −3)
j) (3, 6); (−4, 6)

1.3 Stabilire se le seguenti coppie di rette si intersecano. In caso affermativo, calcolare le coordinate del punto di intersezione.
a) $y = x$; $y = 2x+6$
b) $y = x+4$; $y = -2x-1$
c) $y = x+4$; $y = 2x+6$
d) $y = -2x-1$; $y = 2x+6$
e) $y = -x/3+6$; $y = 2x+6$
f) $y = -x/3+6$; $y = x/3+4$
g) $y = -x/3+6$; $y = x+4$
h) $y = x$; $y = x+4$
i) $y = x$; $y = x/3+4$
j) $y = -3$; $y = x+4$

1.4 Calcolare la lunghezza dei segmenti definiti dalle seguenti coppie di punti:
a) (0, 0); (5, 0)
b) (1, 3); (4, 7)
c) (4, 4); (1, 7)
d) (1, 1); (−3,4)
e) (3, 5); (−2, −7)
f) (2, 0); (0, 2)

g) (2, 1); (2, 4)
h) (2, 1); (4, 2)

1.5 Scrivere l'equazione in forma parametrica di ciascuno dei segmenti definiti nell'esercizio 1.4. L'equazione deve fornire il primo estremo per $u = 0$ ed il secondo per $u = 1$.

1.6 Spiegare perché, nell'algoritmo 1.1 per la generazione dei vettori, alla variabile STEPS viene assegnato il valore INT(MAX(|DX|,|DY|)) e non invece INT(|MIN(|DX|,|DY|)).

1.7 Spiegare perché l'incremento di 0.5 per realizzare l'arrotondamento viene effettuato una sola volta, ovvero perché l'incremento di 0.5 non viene effettuato per ciascuno dei punti considerati.

PROBLEMI DI PROGRAMMAZIONE

1.1 Si implementino gli algoritmi dall'1.1 all'1.5 per costruire un sistema grafico in grado di disegnare segmenti di rette.

1.2 Collaudare il sistema costruito nel problema di programmazione 1.1 scrivendo un programma che disegni una casetta. Un esempio di tale programma è costituito dai seguenti passi:

1) chiamata alla INITIALIZE-I per assegnare i parametri del frame buffer
2) chiamata alla ERASE per pulire lo schermo
3) chiamate alla routine DDA per disegnare una casetta come, ad esempio
DDA (10, 5, 10, 10, '*')
DDA (10, 10, 25, 15, '*')
DDA (25, 15, 40, 10, '*')
DDA (40, 10, 40, 5, '*')
DDA (40, 5, 10, 5, '*')



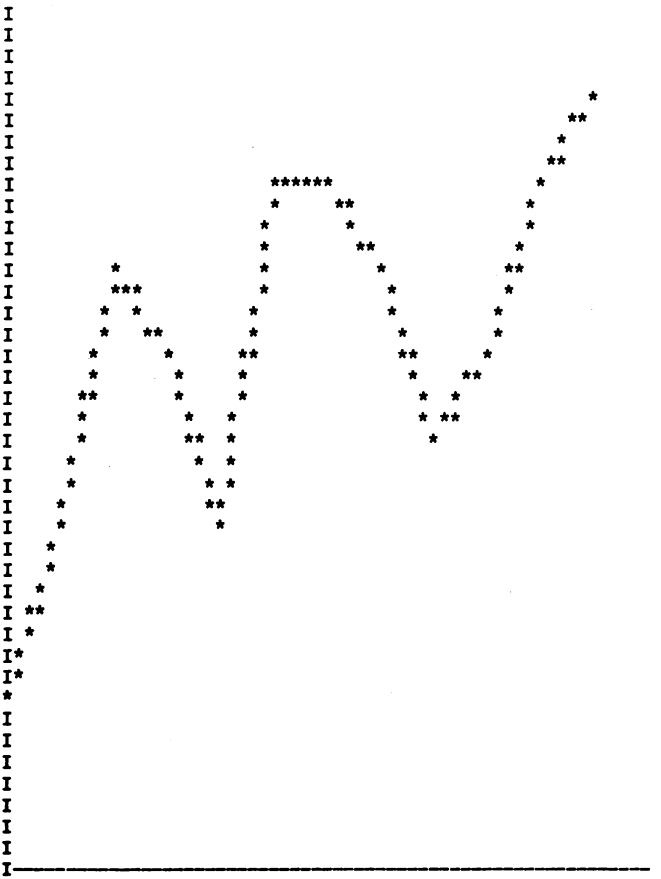
4) chiamate successive alla DISPLAY
5) chiamata alla TERMINATE per concludere il programma

1.3 Una società manifatturiera ha ottenuto i seguenti risultati di vendita:

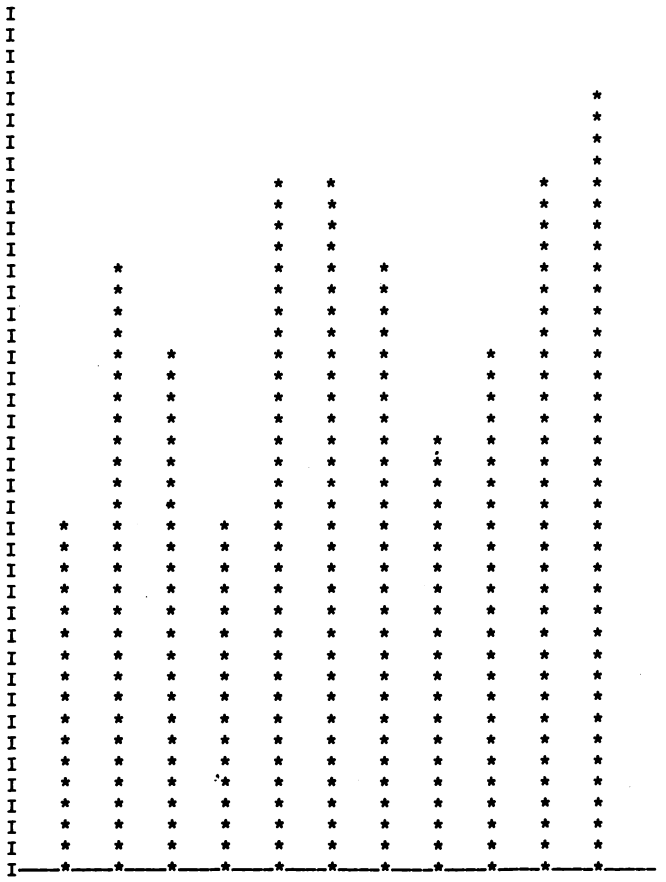
Gen	2000	Mar	7000
Feb	4000	Apr	6000

Mag	4000	Set	5000
Giu	8000	Ott	6000
Lug	8000	Nov	8000
Ago	7000	Dic	9000

Usando il sistema di programmi per disegnare i segmenti definito nel problema 1.1, scrivere un programma per disegnare un grafico delle vendite annuali.



1.4 Usando i dati del problema precedente, costruire l'istogramma delle vendite annuali.

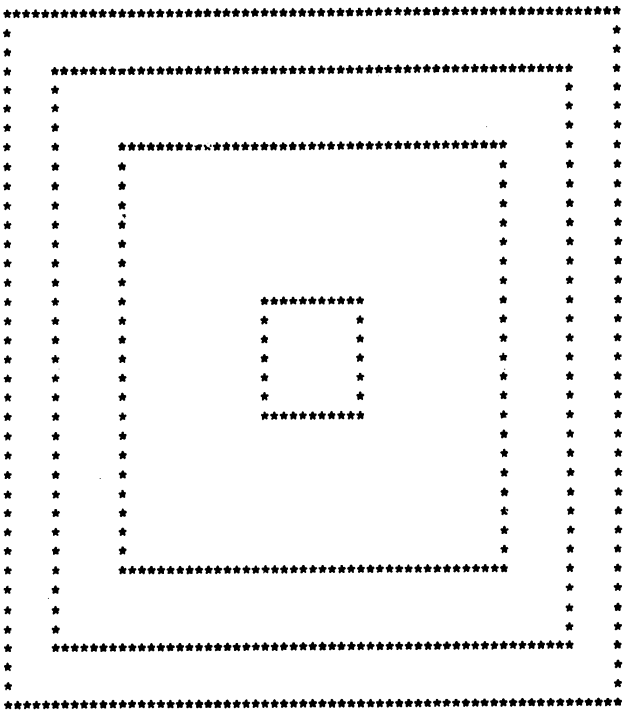


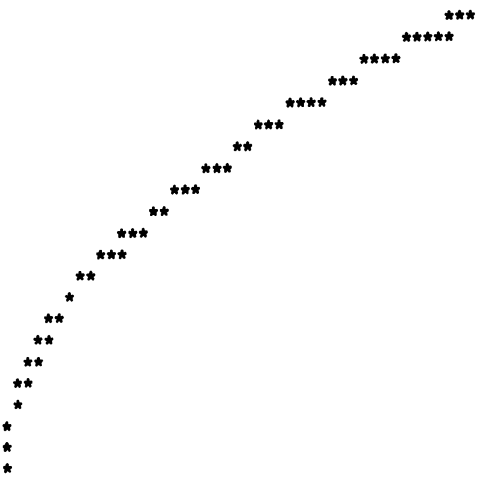
1.5 Scrivere una routine per disegnare un quadrato di lato A centrato nello schermo. Il valore di A va considerato come argomento della routine. Collaudare la routine disegnando una figura con quattro quadrati di dimensioni diverse.

1.6 Scrivere un programma per disegnare in modo approssimato la curva $x = y^2/10$. Per questo dividete l'asse y in intervalli $y_1, y_2, \ldots, y_n$ e trovate quindi il punto sulla curva corrispondente a ciascun valore di x.

$$(x, y) = (y^2/10, y)$$

in questo modo risulterà: per y_1, $(y_1^2/10, y_1)$; per y_2, $(y_2^2/10, y_2)$; e così via. Si congiungano questi punti con segmenti per ottenere la spezzata approssimante.





*1.7 Alcuni terminali CRT consentono di posizionare il cursore tramite caratteri di controllo o sequenze di *escape*. Se disponete di un tale tipo di terminale, modificate l'algoritmo DDA per variare i caratteri direttamente sullo schermo anziché cambiarne il valore contenuto nel frame buffer.

2

Primitive grafiche

INTRODUZIONE

L'utente di un sistema grafico non dovrà in generale partire con la pretesa di costruire da solo tutto ciò di cui ha bisogno. Disporrà invece di un sistema grafico già pronto che comprenderà un hardware specializzato per l'input e l'output di informazioni grafiche ed un software di base costituito da routine per eseguire le operazioni grafiche elementari.
Lo scopo di un sistema grafico è di facilitare l'utente nel lavoro di programmazione; allo stesso modo in cui i linguaggi ad alto livello facilitano la programmazione, mettendo a disposizione operazioni potenti e costrutti che soddisfano i requisiti di un problema, così un sistema grafico fornisce le operazioni e i costrutti utili alla creazione di immagini per facilitare lo sviluppo di un programma grafico.
Alcuni sistemi grafici si presentano addirittura come linguaggi ad alto livello, specializzati per risolvere problemi grafici. Altri si presentano sotto forma di estensioni di linguaggi ad alto livello non specializzati, come il FORTRAN, il PL/1 e il Pascal. Tali estensioni possono essere realizzate tramite un package di sottoprogrammi o l'aggiunta di nuovi costrutti nel linguaggio. Sebbene la forma dei diversi sistemi grafici possa variare, alcune operazioni fondamentali sono sempre presenti, come quella di scrivere un carattere o una stringa di testo e di cambiare il tipo di tratto di una linea.
In questo capitolo si inizierà la costruzione del nostro sistema grafico.
Entrando più in dettaglio sui tipi di dispositivi grafici, verranno presentati gli algoritmi per alcune operazioni fondamentali e verranno introdotti il concetto di indipendenza dal dispositivo grafico utilizzato ed il concetto di display file.

2.1 DISPOSITIVI GRAFICI

Nel capitolo precedente si è detto che le immagini sintetiche generate da un sistema di Computer Graphics sono composte da un numero finito di entità grafiche elementari dette pixel. Un dispositivo con buona risoluzione può avere 1000 divisioni in entrambe le direzioni x e y. Lo schermo visualizzerà quindi 1000×1000, cioè un milione di pixel. Ciascun pixel richiede almeno un bit per memorizzare l'informazione sulla sua intensità (acceso o spento) e ulteriori bit saranno necessari se si desidera ottenere vari toni di grigio o differenti colori. Quindi se l'informazione relativa a ciascun pixel viene inserita nella memoria del computer, è necessaria una notevole estensione di memoria. Questo è ciò che accade in quei terminali grafici che usano la tecnologia raster.
Come si è detto nel Capitolo 1, la porzione di memoria usata per contenere le informazioni relative ai pixel viene detta frame buffer. La memoria è solitamente scandita e le informazioni in essa contenute vengono visualizzate da un hardware specializzato che vi accede direttamente (*direct memory access*, DMA) indipendentemente dall'unità centrale: ciò lascia il processore libero per la generazione dell'immagine (vedi Figura 2.1).
Nei terminali grafici raster il frame buffer può essere esaminato per determinare che cosa deve venir visualizzato. Sui dispositivi di questo tipo possono essere visualizzati tanto linee quanto superfici. Poiché le immagini possono essere indirizzate su di un monitor di tipo televisivo, i dispositivi con tecnologia raster possono spesso avvantaggiarsi della ricerca tecnologica per la produzione di massa dell'industria televisiva; un terminale raster può inoltre generare immagini a colori. Uno dei principali problemi associati a questi sistemi è il tempo necessario per alterare ciascun pixel ogni volta che un'immagine viene modificata. Un altro svantaggio è il costo della memoria necessaria, che viene spesso aggirato in alcuni dispositivi usando una bassa risoluzione (cioè un numero minore di pixel). Il progresso tecnologico ha abbassato drasticamente il prezzo delle memorie e a tutt'oggi sembra che la grafica raster sia destinata a giocare un ruolo dominante nel futuro, mentre nel passato il costo della memoria aveva fatto apparire questa tecnologia molto meno promettente.
Sono stati sviluppati differenti progetti di terminali grafici al fine di ridurne il costo: una tecnica utilizzata prevedeva che il dispositivo di visualizzazione memorizzasse l'immagine invece di utilizzare per questa operazione la memoria del

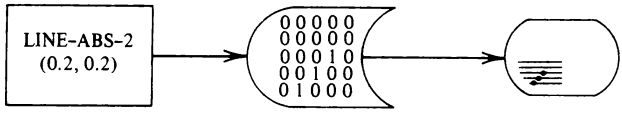


Figura 2.1 Sistema di visualizzazione *raster*

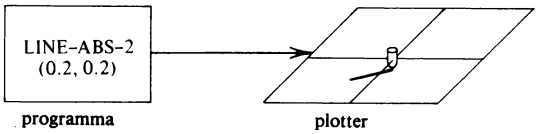


Figura 2.2 Sistema di plottaggio

computer. Ciò è quanto viene fatto dai plotter; una penna viene a contatto con la carta muovendosi nella direzione dettata da un algoritmo che genera vettori (vedi Figura 2.2). Dopo che è stata disegnata una linea, l'inchiostro lasciato sulla carta «memorizza» l'informazione e il computer non la deve più tenere in memoria. Il principale svantaggio di un tale approccio è che una volta disegnata, una linea non può essere facilmente cancellata.

Se si vuole modificare l'immagine sul plotter cancellando una linea, si deve utilizzare un nuovo foglio di carta e ridisegnare la figura modificata. Ciò può richiedere molto tempo e può fare sprecare molta carta. Per questo motivo i plotter non sono sicuramente i migliori dispositivi per una grafica interattiva. Il primo terminale CRT a basso costo fu prodotto dalla *Tektronix*. Grazie alla buona risoluzione ed al basso costo, questi terminali vennero largamente utilizzati e sono tutt'oggi disponibili. Questi terminali usano speciali tubi a raggi catodici chiamati *direct view storage tubes* (DVST), che funzionano più o meno come un plotter. Un pennello elettronico viene diretto sulla superficie dello schermo e la sua posizione è controllata da un campo elettrico o magnetico all'interno del tubo. Quando viene illuminato dal pennello elettronico, il fosforo dello schermo di questi speciali tubi rimane luminoso (vedi Figura 2.3).

Come nel caso del plotter, si può modificare un'immagine su un DVST soltanto cancellando l'intero schermo e ridisegnando. Anche se ciò può essere fatto più velocemente che non su un plotter, il processo è comunque lungo, rende difficile l'interazione e non consente a questi dispositivi di essere utilizzati per animazioni in tempo reale.

Un altro tipo di dispositivo che memorizza l'immagine come i plotter o i tubi a memoria, ma consente la cancellazione selettiva è il *plasma panel*. Il plasma panel — detto anche schermo a elettroluminescenza — contiene un gas a bassa pressione posto tra una griglia verticale ed una orizzontale di sottili conduttori. Un'elevata differenza di potenziale tra un conduttore verticale ed uno orizzontale induce una scarica nel gas così come accade in un tubo al neon. Una ten-

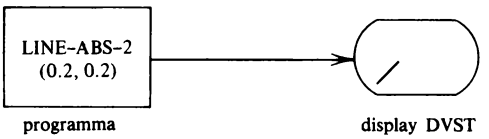


Figura 2.3 Sistema di visualizzazione a *direct view storage*

sione più bassa non innesca la scarica ma la mantiene attivata. Normalmente fra conduttori orizzontali e verticali c'è una bassa tensione. Per accendere un pixel la tensione viene momentaneamente aumentata in corrispondenza dei conduttori che si intersecano nel punto desiderato; per spegnerlo, il voltaggio sui conduttori corrispondenti viene ridotto fino a un valore che non permette la scarica. I plasma panel sono molto affidabili, tanto che vengono spesso usati per applicazioni militari. Sono stati usati anche nel sistema didattico denominato PLATO.

Un altro dispositivo usato per la visualizzazione di informazioni grafiche che ha avuto una notevole rilevanza tra i moderni metodi della grafica, è il *vector refresh display*. Con questa tecnologia si memorizza l'immagine nella memoria del computer, ma lo si fa in modo più efficiente rispetto alla tecnologia raster. Per definire un segmento di retta servono solamente le coordinate dei suoi estremi, mentre nel frame buffer di un sistema raster non vengono memorizzati solo gli estremi, ma anche tutti i pixel che stanno in mezzo, così come tutti gli altri che compongono lo schermo.

Il vector refresh display memorizza invece solo i comandi necessari per disegnare il segmento di retta. Viene memorizzato quindi l'input al generatore di vettori invece dell'output.

Questi comandi vengono memorizzati in ciò che viene chiamato *display file*. I comandi sono esaminati e le linee vengono disegnate usando un algoritmo di generazione di vettori simile al DDA precedentemente illustrato. Ciò viene fatto su un normale tubo a raggi catodici, e quindi l'immagine scompare velocemente. Per poter presentare un'immagine stabile, il sistema deve ridisegnarla continuamente. Ciò significa che la generazione di vettori deve essere applicata a tutte le linee di una figura abbastanza velocemente, in modo da ridisegnare l'intera immagine prima che si noti una variazione di luminosità (più di trenta volte al secondo).

Per far ciò, il generatore di vettori dei display a vector refresh è solitamente firmwarizzato nell'hardware (vedi Figura 2.4). Tali display permettono modifiche in tempo reale dell'immagine, ma hanno qualche svantaggio; infatti le immagini sono costituite solo da linee e non da superfici e figure molto complesse possono presentare sfarfallii (*flicker*) per il troppo lungo tempo necessario per disegnarle.

Il concetto di display file si è dimostrato molto utile e può essere applicato anche ad altri tipi di dispositivi. In questo caso si parla di *pseudo display file*.

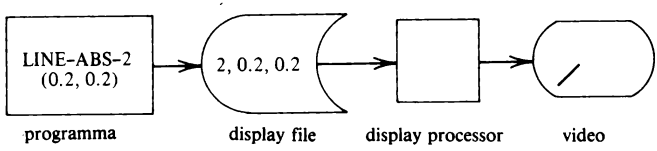


Figura 2.4 Sistema di visualizzazione a *vector refresh*

Tutti i terminali grafici che sono stati descritti possono visualizzare l'immagine così come è stata costruita. Quando viene spedito un comando per tracciare un segmento, questo può apparire immediatamente sullo schermo o sulla carta. La differenza principale riguarda la possibilità di alterare l'immagine. Per modificare l'immagine su un terminale a vector refresh, è necessario modificare solo il display file. Per la stessa modifica su un terminale raster è necessario cambiare il frame buffer, e questo vale anche per un plasma panel. Una modifica su un plotter o un DVST richiede invece la chiamata a una routine di salto pagina, che sposti la carta o ripulisca lo schermo e, successivamente, è necessario ridisegnare l'intera immagine.
Si potrebbe menzionare un'ulteriore classe di dispositivi grafici. Si tratta delle normali stampanti alfanumeriche a linee o di terminali CRT che non siano espressamente progettati per la grafica. Questi dispositivi possono essere usati per uscite grafiche; la loro risoluzione non è di solito molto buona, ma sono sempre disponibili e per certe necessità possono essere sufficienti.
In questo libro tutti gli esercizi saranno formulati in modo tale da poter essere eseguiti disponendo di una normale stampante o di un terminale alfanumerico. Come si è detto nel Capitolo 1, per usare un comunissimo terminale alfanumerico si definisce una matrice che verrà usata come frame buffer. L'immagine viene costruita alterando il contenuto di questa matrice, così come viene fatto per un terminale raster. La differenza rispetto ad un vero sistema raster è che l'utente non può vedere l'immagine in formazione, poiché questa potrà essere visualizzata sul terminale o sulla stampante solo quando è terminata la sua generazione. Questi dispositivi richiedono quindi una procedura in più per mostrare l'immagine.

2.2 OPERAZIONI ELEMENTARI

Indipendentemente dalle tecnologie di visualizzazione, la maggior parte dei sistemi grafici offre una serie di comandi primitivi molto simili tra loro (sebbene la forma dei comandi possa differire da sistema a sistema). Il primo che viene preso in esame è quello per disegnare un segmento. Sebbene un segmento possa essere definito dai suoi due estremi, accade spesso che nella figura più segmenti siano uniti tramite i rispettivi estremi (vedi Figura 2.5).
In questo caso il punto finale dell'ultimo segmento diventa il primo punto del segmento successivo. Per evitare di specificare due volte lo stesso punto, il sistema può memorizzare la posizione corrente della penna o del pennello elettronico. Il comando diventa quindi: disegna una linea dalla posizione corrente al punto specificato, cioè:

LINE-ABS-2(X, Y)

che viene chiamato comando di posizionamento assoluto, poiché vengono pas-

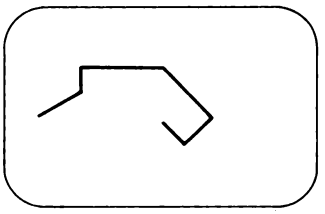


Figura 2.5 Spezzata

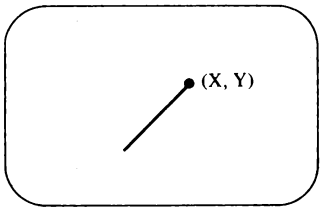


Figura 2.6 Comando LINE assoluto

sate le coordinate assolute del punto finale (vedi Figura 2.6). Il numero 2 contenuto nel nome del comando indica solamente che ci si riferisce alla grafica in due dimensioni, quindi con soli due parametri per punto. Esiste anche un comando di posizionamento relativo con il quale si indica di quanto ci si deve spostare dalla posizione corrente (vedi Figura 2.7):

LINE-REL-2(DX, DY)

L'estremo del segmento, in termini assoluti, può essere determinato dalla posizione corrente e dalla specifica di posizionamento relativo. Se (XC, YC) definisce la posizione corrente:

LINE-REL-2(DX, DY)

è equivalente a:

LINE-ABS-2(XC + DX, YC + DY)

Le procedure sopra riportate servono a produrre una catena di segmenti consecutivi, ma capita anche di voler disegnare due segmenti staccati (vedi Figura 2.8).
Questo può essere fatto con lo stesso meccanismo, a patto di disegnare i due

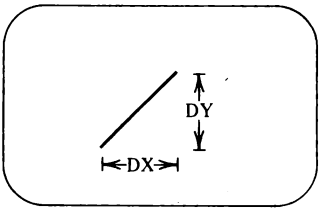


Figura 2.7 Comando LINE relativo

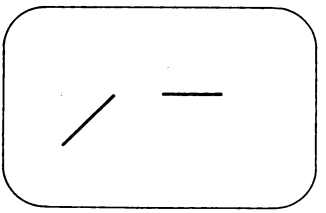


Figura 2.8 Segmenti staccati

segmenti come se fossero uniti da un segmento intermedio invisibile. A questo scopo sono necessari dei comandi per spostare la posizione della penna senza tracciare una retta:

MOVE-ABS-2(X, Y)
MOVE-REL-2(DX, DY)

Come prima, ci potranno essere degli spostamenti definiti in modo assoluto o relativo. È ora possibile costruire un disegno composto da segmenti (ad esempio una casa) tramite una serie di comandi LINE e MOVE. Se tali comandi si trovano in un sottoprogramma, ogni volta che questo viene chiamato viene disegnata l'immagine di una casa. Usando i comandi assoluti, la casa sarà sempre posta nella stessa posizione dello schermo. Se invece si usano soltanto i comandi relativi, la sua posizione dipenderà da quella assunta dalla penna (o pennello elettronico) nel momento in cui è stato chiamato il sottoprogramma. Questa procedura può essere usata per costruire immagini eseguendo copie di figure di base.
Il sottoprogramma per disegnare ciascun tipo di componente di base dovrà essere scritto usando solo comandi relativi. Il disegno dell'intera figura si riduce così al posizionamento della penna ed alla chiamata al sottoprogramma.
Si supponga ad esempio di scrivere il seguente sottoprogramma per disegnare una casa.

```
SOTTOPROGRAMMA CASA
begin
    LINE-REL-2(0, 0.2);
    LINE-REL-2(0.1, 0.1);
    LINE-REL-2(0.1 −0.1);
    LINE-REL-2(0, −0.2);
    LINE-REL-2(−0.2, 0);
end;
```

Questo inizierà il disegno dalla posizione corrente della penna, che diventa l'angolo inferiore sinistro della figura. Esso disegna in sequenza la parete sinistra, il

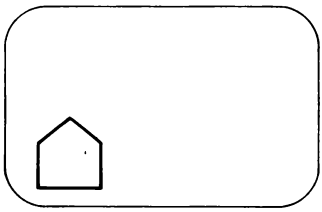


Figura 2.9 Sottoprogramma CASA

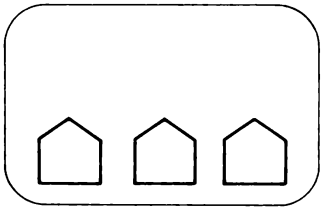


Figura 2.10 Tre chiamate di CASA con differenti posizioni iniziali della penna

tetto, la parete destra ed infine il pavimento (vedi Figura 2.9). Avendo usato solo comandi relativi, è possibile disegnare tre case eseguendo semplicemente questo sottoprogramma con tre differenti posizioni iniziali (vedi Figura 2.10):

```
begin
   MOVE-ABS-2(0.1, 0.2);
   CASA;
   MOVE-ABS-2(0.4, 0.2);
   CASA;
   MOVE-ABS-2(0.7, 0.2);
   CASA;
end;
```

2.3 L'INTERPRETE DEL DISPLAY FILE

Benché sia possibile che il comando LINE modifichi direttamente il frame buffer, il nostro sistema non verrà organizzato in questo modo. Verrà introdotto invece un passaggio intermedio. I comandi LINE e MOVE inseriranno delle informazioni in quello che chiameremo display file. Successivamente l'informazione contenuta nel display file verrà usata per creare l'immagine.
Ci sono due ragioni per dividere in due passi questa operazione. Per prima cosa non tutti i dispositivi di visualizzazione hanno lo stesso frame buffer e quindi differenti dispositivi di visualizzazione richiedono programmi differenti per il loro funzionamento. Isolando ciò nel secondo passo dell'operazione, aumenterà l'indipendenza del sistema grafico dai dispositivi di visualizzazione; sarà più facile cambiare la posizione, la dimensione e l'orientamento dell'immagine. Queste trasformazioni verranno eseguite durante il secondo passo e le relative tecniche saranno descritte nel Capitolo 4.
Per ora ci si concentrerà sulla struttura del display file. Esso contiene tutte le informazioni necessarie per costruire la figura; queste informazioni sono sotto forma di istruzioni come ad esempio «disegna una retta» o «muovi la penna». Registrare istruzioni come queste di solito richiede meno memoria che non me-

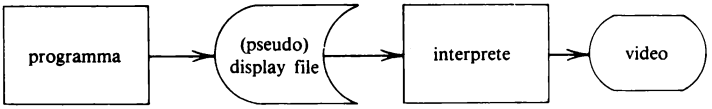


Figura 2.11 Display file e interprete

morizzare la figura pixel per pixel. Inoltre queste istruzioni possono essere considerate come un programma per creare l'immagine. Ogni istruzione indica un'azione di MOVE o LINE per il dispositivo fisico di visualizzazione. Verrà realizzato un interprete per il display file in modo da convertire le istruzioni nella corrispondente immagine (vedi Figura 2.11).
L'interprete può essere pensato come una macchina che esegue tutte queste istruzioni. Il risultato dell'esecuzione di un display file è l'immagine visibile. In alcuni sistemi grafici, l'interprete è effettivamente un calcolatore separato detto *display processor*, che si trova all'interno del terminale grafico. In altri sistemi invece si simula il comportamento del display processor. Qualora esista un display processor separato, è necessario prestare attenzione alla modifica del display file, in quanto il display processor potrebbe trovarsi ad interpretare delle istruzioni mentre queste vengono modificate.
L'interprete del display file rappresenta l'interfaccia fra un programma grafico e il dispositivo di visualizzazione. Se dei programmi grafici vengono scritti appositamente per un certo tipo di terminale grafico, non saranno eseguibili con altri dispositivi. La portabilità di questi programmi è quindi limitata. Se, al contrario, si scrivono dei programmi che generano un codice nel display file, tutto quello che serve è un interprete per ogni diverso dispositivo, che converta le istruzioni standard in azioni di quel particolare dispositivo.
Per concludere, d'ora in poi si farà riferimento ad un dispositivo di visualizzazione con il relativo interprete come ad un'unica macchina che esegue programmi standardizzati.

2.4 COORDINATE NORMALIZZATE DEL DISPOSITIVO

I diversi dispositivi di visualizzazione possono avere dimensioni dello schermo differenti, misurate in pixel. Se si vogliono rendere indipendenti i programmi dal particolare dispositivo, si devono specificare le coordinate in unità di misura che non sia il pixel e usare l'interprete per convertirle negli appropriati valori in pixel per il dispositivo che si sta usando.
Le unità di misura indipendenti dal dispositivo vengono dette *coordinate normalizzate del dispositivo*.
In queste unità di misura, lo schermo è largo 1 unità e alto 1 unità. L'angolo in basso a sinistra dello schermo viene assunto come origine, e quello in alto a destra è il punto (1, 1). Il punto (0.5, 0.5) è nel centro dello schermo a prescinde-

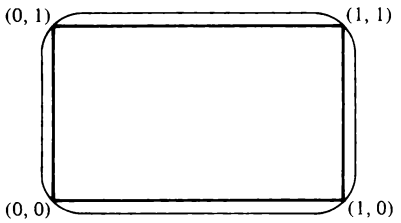


Figura 2.12 Coordinate normalizzate del dispositivo

re dalle dimensioni fisiche o dalla risoluzione del dispositivo di visualizzazione usato (vedi Figura 2.12).
L'interprete si serve di una semplice formula lineare per convertire le coordinate normalizzate del dispositivo in quelle fisiche. Supponiamo che per il terminale grafico che stiamo usando, l'indice del pixel più a sinistra sia WIDTH-START e che ci sia un numero WIDTH di pixel in direzione orizzontale. Supponiamo anche che il pixel più in basso sia HEIGHT-START e che il numero di pixel in direzione verticale sia HEIGHT.
In coordinate normalizzate lo schermo è largo 1 unità, ma in quelle fisiche è largo WIDTH unità, cosicché la posizione x normalizzata deve essere moltiplicata per WIDTH/1 per convertirla nelle unità schermo in uso. Nella posizione $x_n = 0$ in coordinate normalizzate, avremo in coordinate schermo x_s = WIDTH-START e la formula di conversione sarà:

$$x_s = \text{WIDTH} * x_n + \text{WIDTH-START} \tag{2.1}$$

Analogamente per la direzione verticale:

$$y_s = \text{HEIGHT} * y_n + \text{HEIGHT-START} \tag{2.2}$$

2.5 STRUTTURA DEL DISPLAY FILE

Si consideri ora la struttura del display file. Ciascun comando del display file è composto di due parti, un *codice operativo* (*opcode*), che indica di quale tipo di comandi si tratti (ad esempio LINE o MOVE) e gli *operandi*, che sono le coordinate di un punto (x, y). Il display file è strutturato come una serie di istruzioni di questo tipo. Un metodo possibile per memorizzare queste istruzioni è quello di usare tre array separati: uno per il codice operativo (DF-OP), uno per la coordinata x (DF-X) e uno per la coordinata y (DF-Y). Ad esempio, per riferirsi alla settima istruzione del display file basterà prendere il settimo elemento di ciascuno dei tre array DF-OP[7], DF-X[7], DF-Y[7]. Naturalmente il display

file deve essere dimensionato in modo da poter contenere tutti i comandi necessari per creare la nostra immagine.
Si dovrà assegnare un significato ai codici operativi possibili per poterli poi interpretare. A questo punto ci sono soltanto due possibili istruzioni da considerare, MOVE e LINE, nella forma di comando assoluta, poiché da quella relativa è possibile ottenere quella assoluta tramite una conversione prima che i comandi siano introdotti nel display file. Si assegna al codice operativo il valore 1 per specificare un comando MOVE ed il valore 2 per il comando LINE. Un comando per muovere nella posizione $x=0.3$ e $y=0.7$ appare quindi come: 1, 0.3, 0.7. Le istruzioni per la memorizzazione nella terza posizione del display file sono quindi:

DF-OP[3]←1;
DF-X[3]←0.3;
DF-Y[3]←0.7;

Se DF-OP[4] contenesse il valore 2, DF-X[4] il valore 0.5 e DF-Y[4] il valore 0.8, sul video apparirebbe un segmento da (0.3, 0.7) a (0.5, 0.8), una volta che le istruzioni 3 e 4 del display file fossero state interpretate (vedi Figura 2.13). Si possono sviluppare ora gli algoritmi per l'inserimento di istruzioni nel display file. Per definire un segmento occorrono i due estremi, ma in questo caso si inserisce solo un estremo assumendo che l'altro sia la posizione corrente della penna. Occorrono quindi le variabili DF-PEN-X e DF-PEN-Y per trattenere il valore corrente della posizione della penna che servirà anche per convertire i comandi da relativi in assoluti. Sarà necessario disporre anche di una variabile FREE per indicare la posizione della successiva cella libera (non utilizzata) del display file. Queste variabili, unitamente al display file stesso, vengono usate da tutta una serie di routine e pertanto vanno conservate durante tutte le chiamate alle diverse routine, diventando perciò variabili globali. Il primo algoritmo considerato inserisce un'istruzione nel display file.

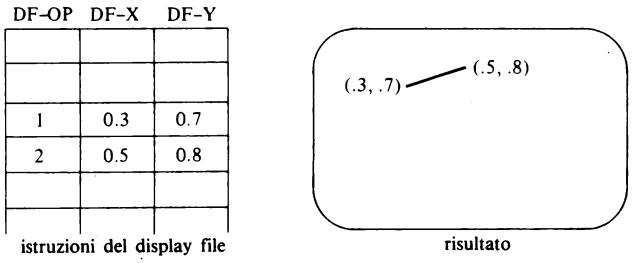


Figura 2.13 Istruzioni del display file

Algoritmo 2.1 PUT-POINT (OP, X, Y)
Inserisce un'istruzione completa nel display file.
Argomenti OP, X,Y←istruzione che deve essere inserita
Variabili globali DF-OP, DF-X, DF-Y←array per le istruzioni che costituiscono il display file
FREE←posizione della prima cella libera
Costanti DFSIZE←lunghezza dei tre array

```
begin
   if FREE > DFSIZE then return-error 'DISPLAY FILE PIENO';
   DF-OP[FREE] ← OP;
   DF-X[FREE] ← X;
   DF-Y[FREE] ← Y;
   FREE ← FREE + 1;
   return;
end;
```

Questo algoritmo memorizza il codice operativo e le coordinate in una posizione del display file. Il puntatore FREE viene incrementato in modo da indicare la prossima cella libera disponibile alla successiva chiamata della routine. Poiché si desidera accedere direttamente ad un elemento del display file, viene definito un metodo di accesso in una routine apposita in modo che una modifica nella struttura di dati usata per il display file non influenzi il resto del sistema di software grafico.

Algoritmo 2.2 GET-POINT (NTH, OP, X, Y)
Restituisce l'istruzione NTH del display file.
Argomenti NTH←numero dell'istruzione richiesta
OP, X, Y←istruzione che deve essere restituita
Variabili globali DF-OP, DF-X, DF-Y←array per il display file

```
begin
   OP ← DF-OP[NTH];
   X ← DF-X[NTH];
   Y ← DF-Y[NTH];
   return;
end;
```

Le istruzioni MOVE e LINE devono aggiornare la posizione corrente della penna ed inserire nel display file l'opportuno comando. Se l'aggiornamento della posizione corrente della penna viene eseguito per primo, può essere usato come operando per l'istruzione del display file. Nei capitoli successivi risulterà conveniente avere delle routine separate che prelevano il codice operativo e la posizione corrente della penna e li inseriscono nel display file come istruzione.

Algoritmo 2.3 DISPLAY-FILE-ENTER (OP)
Combina l'operazione e la posizione corrente per formare un'istruzione che inserisce poi nel display file.
Argomenti OP←operazione da inserire
Variabili globali DF-PEN-X, DF-PEN-Y←posizione corrente della penna

```
begin
   PUT-POINT(OP, DF-PEN-X, DF-PEN-Y);
   return;
end;
```

Usando la procedura DISPLAY-FILE-ENTER per inserire un'istruzione nel display file, l'istruzione MOVE assoluta diventa:

Algoritmo 2.4 MOVE-ABS-2 (X, Y)
Routine utente che memorizza un comando di movimento della penna.
Argomenti X, Y←punto nel quale muovere la penna
Variabili globali DF-PEN-X, DF-PEN-Y←posizione corrente della penna

```
begin
   DF-PEN-X←X;
   DF-PEN-Y←Y;
   DISPLAY-FILE-ENTER(1);
   return;
end;
```

Il punto (DF-PEN-X, DF-PEN-Y) memorizza la posizione nella quale si vuole che vada la penna. Assegnando (DF-PEN-X, DF-PEN-Y) a (X, Y) comunichiamo che la penna deve essere mossa nella posizione (X, Y).
L'algoritmo per introdurre un comando LINE è analogo.

Algoritmo 2.5 LINE-ABS-2 (X, Y)
Routine utente che memorizza un comando per disegnare un segmento di retta.
Argomenti X, Y←punto fino al quale disegnare la retta
Variabili globali DF-PEN-X, DF-PEN-Y←posizione corrente della penna

```
begin
   DF-PEN-X←Y;
   DF-PEN-Y←Y;
   DISPLAY-FILE-ENTER(2)
   return;
end;
```

Cambiando DF-PEN-X e DF-PEN-Y si indica che la penna verrà posizionata in (X, Y), ma inserendo il codice di operazione 2 anziché 1 si comunica all'interprete di disegnare una linea mentre la penna è in movimento.

Si possono quindi scrivere in modo analogo gli algoritmi per i comandi di posizionamento relativo.

Algoritmo 2.6 MOVE-REL-2 (DX, DY)
Routine utente che memorizza un comando di movimento della penna.
Argomenti DX, DY ← variazione della posizione della penna
Variabili globali DF-PEN-X, DF-PEN-Y ← posizione corrente della penna

```
begin
   DF-PEN-X ← DF-PEN-X + DX;
   DF-PEN-Y ← DF-PEN-Y + DY;
   DISPLAY-FILE-ENTER(1);
   return;
end;
```

Algoritmo 2.7 LINE-REL-2 (DX, DY)
Routine utente per memorizzare un comando per disegnare un segmento di retta.
Argomenti DX, DY ← incrementi con i quali disegnare la retta
Variabili globali DF-PEN-X, DF-PEN-Y ← posizione corrente della penna

```
begin
   DF-PEN-X ← DF-PEN-X + DX;
   DF-PEN-Y ← DF-PEN-Y + DY;
   DISPLAY-FILE-ENTER(2);
   return;
end;
```

Le routine LINE e MOVE relative agiscono in un modo analogo a quelle assolute, poiché anch'esse dicono dove deve essere posizionata la penna e come giungervi. Si differenziano solo per il fatto che la nuova posizione della penna è calcolata come variazione incrementale della vecchia.

2.6 ALGORITMI PER IL DISPLAY FILE

Gli algoritmi precedentemente esposti indicano come introdurre le istruzioni nel display file. La restante parte del problema è quella di analizzare il display file ed eseguirne i comandi. Verranno ora considerati gli algoritmi necessari per la realizzazione dell'interprete del display file.
L'interprete legge le istruzioni da una porzione del display file e realizza gli opportuni comandi di LINE e MOVE tramite chiamate ad una routine che genera un vettore dello stesso tipo di quello usato nella DDA del Capitolo 1. La routine che alla fine permette di effettuare i comandi di LINE e MOVE sullo schermo dipende dal tipo del dispositivo di visualizzazione e dal suo software.
Vengono di seguito riportate le versioni degli algoritmi che possono essere usati

con una DDA come quella fornita nel Capitolo 1. Nelle appendici si possono trovare esempi di routine DOMOVE e DOLINE per altri sistemi. La routine DOMOVE deve solamente aggiornare la posizione corrente della penna. I calcoli necessari sono soltanto quelli relativi alla conversione da coordinate normalizzate del dispositivo a coordinate assolute dello schermo.

Algoritmo 2.8 DOMOVE (X, Y)
Realizza un movimento della penna.
Argomenti X, Y ← punto nel quale muovere la penna (in coordinate normalizzate)
Variabili globali FRAME-PEN-X, FRAME-PEN-Y ← posizione della penna (coordinate schermo)
WIDTH, HEIGHT ← dimensioni dello schermo
WIDTH-START, HEIGHT-START ← coordinate dell'angolo inferiore sinistro
WIDTH-END, HEIGHT-END ← coordinate dell'angolo superiore destro

```
begin
   FRAME-PEN-X ← MAX(WIDTH-START,MIN(WIDTH-END,
                        X * WIDTH + WIDTH-START));
   FRAME-PEN-Y ← MAX(HEIGHT-START, MIN(HEIGHT-END,
                        Y * HEIGHT + HEIGHT-START));
   return;
end;
```

In questo algoritmo si possono vedere le formule (2.1) e (2.2) per convertire i valori delle coordinate normalizzate (gli argomenti) nelle coordinate schermo. Le funzioni MAX e MIN sono state aggiunte alla formula per motivi di sicurezza, per evitare in ogni caso la generazione di un valore al di fuori dei limiti del terminale. Se (X, Y) dovessero corrispondere ad un punto fuori dall'area dello schermo, le funzioni MAX e MIN «limiterebbero» le coordinate della posizione corrispondente sullo schermo al suo confine.
L'algoritmo DOLINE aggiorna la posizione della penna e chiama la DDA per inserire il segmento nel frame buffer; converte inoltre da coordinate normalizzate del dispositivo in coordinate schermo.

Algoritmo 2.9 DOLINE (X, Y)
Disegna un segmento di retta.
Argomenti X, Y ← punto fino al quale disegnare la retta (coordinate normalizzate)
Variabili globali FRAME-PEN-X, FRAME-PEN-Y ← posizione della penna (coordinate schermo)
WIDTH, HEIGHT ← dimensioni dello schermo

```
                    WIDTH-START, HEIGHT-START ← coordinate dell'an-
                        golo inferiore sinistro
                    WIDTH-END, HEIGHT-END ← coordinate dell'angolo
                        superiore destro
                    LINECHR←tipo di linea
Variabili locali    X1, Y1←estremo del precedente segmento
begin
   X1←FRAME-PEN-X;
   Y1←FRAME-PEN-Y;
   FRAME-PEN-X←MAX(WIDTH-START,MIN(WIDTH-END,
                    X*WIDTH+WIDTH-START));
   FRAME-PEN-Y←MAX(HEIGHT-START,MIN(HEIGHT-END,
                    Y*HEIGHT+HEIGHT-START));
   DDA(X1, Y1, FRAME-PEN-X, FRAME-PEN-Y, LINECHR);
   return;
end;
```

Nelle appendici sono riportate alcune versioni della routine DOLINE dipendenti dal dispositivo, per consentire l'utilizzo di diversi tipi di sistemi grafici.
Si può ora iniziare la stesura dell'interprete, cioè la routine che esamina il display file e genera le chiamate a DOMOVE e DOLINE a seconda delle istruzioni che trova. Apparirà utile nei capitoli successivi avere la possibilità di interpretare soltanto una porzione del display file; all'interprete verranno quindi passati come argomenti la posizione di partenza (START) e il numero di istruzioni da interpretare (COUNT).

Algoritmo 2.10 INTERPRET (START, COUNT)
Scandisce il display file eseguendone le istruzioni.

```
Argomenti           START←indice di partenza della scansione del display
                        file
                    COUNT←numero delle istruzioni da interpretare
Variabili globali   WIDTH-START, HEIGHT-START←coordinate dell'an-
                        golo inferiore sinistro dello schermo
Variabili locali    NTH←indice del display file
                    OP, X, Y←istruzione del display file
begin
   ciclo per eseguire le istruzioni desiderate
   for NTH = START to START+COUNT-1
      do begin
         GET-POINT(NTH, OP, X, Y);
               if OP = 1
               then DOMOVE(X, Y)
               else if OP = 2
                     then DOLINE(X, Y)
                     else return-error 'ERRORE DI OP-CODE';
```

```
        end;
    return;
end;
```

Un ciclo passa attraverso tutte le istruzioni del display file scelte, recuperandole attraverso la routine GET-POINT.
L'algoritmo verifica se si tratta di una istruzione di MOVE o di LINE esaminandone il codice operativo e chiama la routine DOMOVE o DOLINE per eseguire l'appropriata azione nell'area di visualizzazione.

2.7 CONTROLLO DELLO SCHERMO

Per visualizzare la figura descritta all'interno del display file, sono necessarie tre azioni. Per prima cosa è necessario ripulire lo schermo; secondariamente si dovrà interpretare il display file; infine, su alcuni dispositivi (come stampanti a linee o terminali CRT standard) è necessario un comando apposito per visualizzare il contenuto del frame buffer. Per convenienza di uso, queste tre operazioni saranno combinate in una sola routine. Prima di presentare la routine, comunque, occorre dire qualcosa a riguardo dell'operazione di pulizia dello schermo (o del frame buffer). Potrebbe non essere utile ripulire lo schermo ogni volta che si interpreta il display file; infatti, se le sole modifiche apportate ad una figura sono delle aggiunte, non c'è ragione di cancellarla e ridisegnarla ogni volta, poiché disegni molto complessi possono essere generati un pezzo alla volta. Naturalmente, variazioni all'immagine di natura diversa richiedono una cancellazione con ridisegno dell'intera figura. Cancellare porzioni della figura, spostarne la posizione, la dimensione o l'orientamento, sono tutte operazioni che richiedono di ricostruire una nuova immagine partendo dallo schermo ripulito. Si vedrà come apportare queste modifiche nei Capitoli 4 e 5; la cosa importante da notare a questo punto è che la richiesta di pulizia dello schermo può essere considerata come facente parte di questi cambiamenti: in questi casi, infatti, essa può essere fatta automaticamente. Ciononostante l'utente deve essere in grado di richiedere una routine di cancellazione in ogni momento. Nel Capitolo 1 si è introdotta una routine di pulizia dello schermo che potrebbe essere richiamata dall'utente; può comunque succedere che il programma dell'utente valuti di aver bisogno di una cancellazione in un momento precedente a quello in cui deve essere fatta la chiamata alla routine che la esegue. La pulizia del frame buffer, sarà controllata tramite un flag (una variabile il cui valore è TRUE o FALSE): un valore TRUE indica che lo schermo dovrà essere pulito prima della interpretazione del display file ed un valore FALSE significa che le istruzioni del display file possono essere disegnate «sopra» la vecchia immagine. Se la macchina valuta necessaria una cancellazione, pone il valore dell'indicatore a TRUE.
Fintantoché il display file non è pronto ad essere interpretato, sullo schermo

non appare nulla. A questo punto l'indicatore di cancellazione è sottoposto al test e, se risulta TRUE, il frame buffer viene pulito e l'indicatore viene riportato a FALSE.

Algoritmo 2.11 NEW-FRAME
Routine utente che segnala che il frame buffer deve essere pulito prima di visualizzare il contenuto del display file.
Variabili globali ERASE-FLAG←indica se il frame buffer deve essere pulito

```
begin
   ERASE-FLAG←TRUE;
   return;
end;
```

Tramite la sola routine MAKE-PICTURE-CURRENT vengono ora unificate l'operazione di pulizia del frame buffer (quando richiesta), l'interpretazione del display file e la visualizzazione del nuovo frame buffer.

Algoritmo 2.12 MAKE-PICTURE-CURRENT
Routine utente che mostra il display file corrente.
Variabili globali FREE←indice della cella del display file libera più vicina
ERASE-FLAG←indica se il frame-buffer deve essere pulito

```
begin
   if ERASE-FLAG
   then begin
        ERASE;
        ERASE-FLAG←FALSE;
        end;
   if FREE>1 then INTERPRET(1,FREE-1);
   DISPLAY;
   FREE←1;
   return;
end;
```

Dopo aver ricontrollato, per sicurezza, se il display file non sia vuoto, vengono interpretati i comandi in esso contenuti. Successivamente, ciascuna azione necessaria per mostrare i risultati dell'interpretazione viene effettuata tramite la routine DISPLAY. Se viene usato un terminale grafico in grado di mostrare immediatamente il risultato di ciascun comando, allora la chiamata alla DISPLAY non è necessaria ed essa diventa un'operazione nulla. Alla fine, l'indice del display file è posto nuovamente a 1 per indicare che il display file stesso è vuoto e quindi pronto ad accettare i nuovi comandi di disegno.

Si ha ora la necessità di inizializzare alcune variabili: questo viene eseguito dall'algoritmo seguente.

Algoritmo 2.13 INITIALIZE-2A
Inizializzazione delle variabili per il disegno di rette.
Variabili globali FREE←indice della cella di display file libera più vicina
DF-PEN-X, DF-PEN-Y←posizione della penna del display file

```
begin
   INITIALIZE-1;
   FREE←1;
   DF-PEN-X←0;
   DF-PEN-Y←0;
   NEW-FRAME;
   return;
end;
```

2.8 TESTI

Un'altra operazione fondamentale è la generazione dei testi. La maggior parte delle immagini da visualizzare sono composte da dati di tipo grafico e alfanumerico. Note, istruzioni, comandi, valori, messaggi e così via possono essere presentati assieme all'immagine. Il comando di base per tali funzioni è la generazione in output di un carattere o di una stringa di caratteri. Mentre di solito si hanno a disposizione comandi per la generazione di un'intera stringa, si può trovare a volte una routine di generazione di un singolo carattere. Non è comunque difficile costruire una procedura per applicare tale funzione di base ad un'intera stringa. I caratteri stessi possono essere disegnati con il metodo a matrice di punti o con il metodo a vettori. Essi vengono di solito creati per mezzo di hardware specializzato sebbene si possa anche trovare il software corrispondente che li crea come sequenza di segmenti. Il vantaggio dell'hardware è la velocità di esecuzione ed un risparmio di spazio nel display file. Con terminali sofisticati ci possono essere opzioni, specificabili dall'utente, come la spaziatura fra i caratteri, la dimensione dei caratteri, il rapporto tra la loro altezza e la loro larghezza, l'inclinazione della stringa di caratteri, l'orientamento e la forma dei caratteri stessi.
L'interprete verrà esteso con le aggiunte necessarie per generare scritte. Si estenderà il numero di codici operativi nella misura di un codice per ciascun carattere. L'operando di un'istruzione determinerà dove il carattere andrà scritto sullo schermo. Il codice operativo che verrà usato per un certo carattere sarà il suo codice ASCII (un numero compreso tra 32 e 126). Usare il codice ASCII faciliterà, in molti sistemi, la conversione dall'istruzione al valore del carattere usato per la sua visualizzazione. Verranno esclusi i codici ASCII relativi ai ca-

spazio	32	8	56	P	80	h	104
!	33	9	57	Q	81	i	105
"	34	:	58	R	82	j	106
#	35	;	59	S	83	k	107
$	36	<	60	T	84	l	108
%	37	=	61	U	85	m	109
&	38	>	62	V	86	n	110
'	39	?	63	W	87	o	111
(	40	@	64	X	88	p	112
)	41	A	65	Y	89	q	113
*	42	B	66	Z	90	r	114
+	43	C	67	[	91	s	115
,	44	D	68	\	92	t	116
-	45	E	69	]	93	u	117
.	46	F	70	^	94	v	118
/	47	G	71	_	95	w	119
0	48	H	72	'	96	x	120
1	49	I	73	a	97	y	121
2	50	J	74	b	98	z	122
3	51	K	75	c	99	{	123
4	52	L	76	d	100	|	124
5	53	M	77	e	101	}	125
6	54	N	78	f	102	~	126
7	55	O	79	g	103		

Figura 2.14 Codice ASCII dei caratteri

ratteri di controllo e quindi un codice valido sarà maggiore di 31: i codici da 3 a 31 saranno usati nel prossimo capitolo per indicare i poligoni (vedi Figura 2.14). Si possono usare codifiche per i caratteri diverse da quella ASCII, purché usino valori numerici maggiori di 31.

Un comando per la generazione di caratteri ha quindi un codice operativo che specifica quale carattere deve essere visualizzato e gli operandi x, y che ne specificano la posizione sullo schermo. Una parola o una stringa di caratteri viene memorizzata come sequenza di comandi per disegnare ciascuno dei suoi caratteri. Poiché si assegna la posizione di ciascun carattere, si può controllare la spaziatura tra i caratteri della stringa. Inoltre, assegnando le coordinate relative alla posizione di ciascun carattere, la stringa potrà essere scritta in orizzontale, in verticale o in diagonale.

Finora non è stato introdotto alcun meccanismo per assegnare l'orientamento di un carattere. Alcuni dispositivi grafici permettono di visualizzare i caratteri con qualunque angolazione. Altri permettono solo incrementi di 90 gradi, mentre la maggior parte (come tutti i normali terminali) visualizzano solo caratteri in posizione verticale.

Per mantenere l'indipendenza dai diversi dispositivi e nel tentativo di semplificare le cose, si stabilisce che tutti i caratteri della stessa stringa siano orientati nella stessa direzione e alla fine gli algoritmi verranno definiti per dispositivi che permettano solo caratteri in posizione verticale. Può essere abbastanza macchinoso scrivere una stringa di caratteri se deve essere poi chiamata una singola procedura per ognuno di essi. Verrà realizzata per questo una routine

per inserire intere stringhe di caratteri nel display file per mezzo di sequenze di inserimento di un singolo carattere. Sarà compito di questa routine spostare automaticamente i diversi caratteri per lasciare gli interspazi desiderati. La spaziatura tra i caratteri verrà assegnata attraverso le variabili globali XCHRSP e YCHRSP. Sarà quindi necessario disporre di una routine per assegnare i valori desiderati a questi parametri di spaziatura.
Il sistema grafico CORE permette una vasta gamma di possibilità per il formato dei testi; è possibile cambiare le dimensioni, l'orientamento, la spaziatura, il tipo di carattere e la direzione della riga di testo. Il sistema proposto in questo volume non sarà così ambizioso. Permetterà di controllare solo la spaziatura tra i caratteri e la direzione di una riga di testo. Il sistema CORE determina la direzione della riga di testo dall'orientamento dei caratteri.
C'è infatti un comando SET-CHARUP(DX, DY) nel quale [DX, DY] definiscono il vettore che specifica quale sia l'asse verticale dei caratteri. La linea di testo viene quindi stampata alla «destra» di questa direzione. In questo sistema non si cercherà di cambiare l'orientamento dei caratteri e quindi il comando SET-CHARUP risulta inutile. Nondimeno, per essere coerenti con le specifiche CORE, la direzione di una linea di testo verrà determinata come il vettore perpendicolare a quello specificato dal comando SET-CHARUP(DX, DY). La spaziatura tra i caratteri nella direzione della linea di testo viene cambiata attraverso il comando SET-CHARSPACE(CHARSPACE). Il sistema considererà automaticamente lo spazio occupato in larghezza da ogni carattere.
Il parametro CHARSPACE definisce una spaziatura addizionale, che viene specificata in termini di dimensioni del carattere, cosicché il valore 0.5 per CHARSPACE significa uno spazio addizionale di mezzo carattere (per un totale di una ampiezza di 1.5 caratteri tra il centro di due caratteri successivi). Siccome il parametro CHARSPACE è misurato in termini di ampiezze dei caratteri invece che di coordinate normalizzate dello schermo, una linea di testo scritta con ampia spaziatura su un certo dispositivo, risulterà avere un'ampia spaziatura su qualunque altro dispositivo anche se l'ampiezza assoluta in coordinate schermo

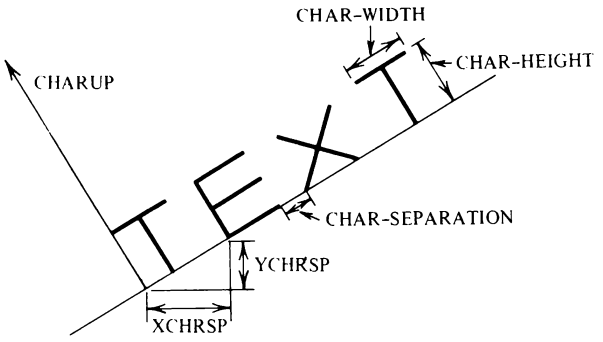


Figura 2.15 Parametri per il testo con caratteri orientabili

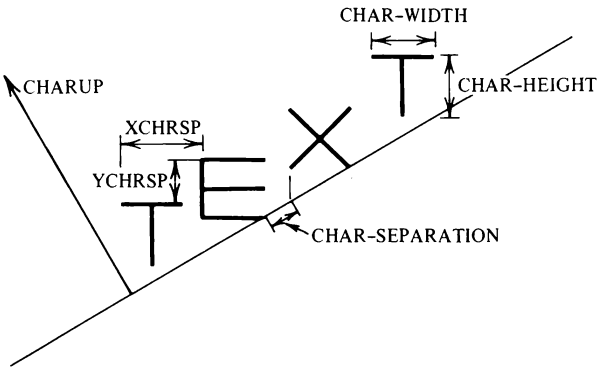


Figura 2.16 Parametri per il testo con caratteri non orientabili

normalizzate dei caratteri può variare. Il problema da risolvere è quello di convertire i valori di CHARUP e di CHARSPACE e le dimensioni del carattere in spostamenti lungo l'asse x (XCHRSP) e lungo l'asse y (YCHRSP) di ciascun carattere della stringa (vedi Figura 2.15 e 2.16).
Il calcolo dell'ampiezza di questi spostamenti è complicato dal fatto che non sono permesse rotazioni dei simboli ed inoltre la larghezza e l'altezza di un carattere sono spesso diverse. Se il testo va scritto orizzontalmente, lo spostamento equivale all'ampiezza del carattere. Se la stringa va scritta in verticale allora l'altezza dei caratteri sarà equivalente alla dimensione del passo standard. Per tutti gli altri orientamenti si dovrà usare per il passo un valore compreso tra l'altezza e l'ampiezza. Lo spostamento standard viene calcolato in modo che la sua componente orizzontale sia proporzionale alla larghezza del carattere e lo spostamento verticale sia proporzionale all'altezza del carattere nel modo seguente:

$$\text{DEFAULT-STEP} = \frac{|(\text{CHAR-WIDTH}) * (\text{DY})| + |(\text{CHAR-HEIGHT}) * (\text{DX})|}{(\text{DX}^2 + \text{DY}^2)^{1/2}} \tag{2.3}$$

Dove DX e DY sono le componenti del vettore CHARUP (con inclinazione DX/DY), perpendicolare rispetto alla direzione della linea di testo (di inclinazione $-$DY/DX); questo è il motivo per cui DX e DY sembrano essere invertiti nella (2.3).

$$\text{TRUE-STEP} = \text{DEFAULT-STEP} * (1 + \text{CHAR-SEPARATION}) \tag{2.4}$$

Infine, il valore del passo viene scomposto nelle sue componenti orizzontale e verticale. Ancora una volta, poiché [DX, DY] è un vettore perpendicolare alla

direzione della stringa, DY farà le funzioni della componente x e $-$DX quelle della componente y.

$$\begin{aligned} \text{XCHRSP} &= \text{TRUE-STEP}\ \frac{\text{DY}}{(\text{DX}^2+\text{DY}^2)^{1/2}} \\ \text{YCHRSP} &= \text{TRUE-STEP}\ \frac{-\text{DX}}{(\text{DX}^2+\text{DY}^2)^{1/2}} \end{aligned} \tag{2.5}$$

Questo calcolo viene sviluppato in modo più efficiente nel seguente algoritmo. (Occorre notare che combinando le equazioni (2.3), (2.4) e (2.5) si elimina il calcolo della radice quadrata).

Algoritmo 2.14 SET-CHARUP (DX, DY)
Routine utente che indica in quale direzione deve essere stampata la stringa.

Argomenti	DX, DY←array in cui assegnare la direzione verticale del carattere
Variabili globali	XCHRSP, YCHRSP←distanza tra i centri di due caratteri CHAR-WIDTH, CHAR-HEIGHT←dimensione del carattere CHAR-SEPARATION←dimensione della spaziatura rispetto alla dimensione del carattere
Variabili locali	S, S1, S2←variabili temporanee usate per memorizzare i risultati intermedi durante il calcolo
Costanti	ROUNDOFF←numero piccolo a piacere maggiore del più grande errore di arrotondamento

```
begin
   S←DX↑2+DY↑2;
   if S<ROUNDOFF then return-error
      'NESSUNA DIREZIONE CHARUP';
   S1←(|CHAR-WIDTH*DY|+|CHAR-HEIGHT*DX|);
   S2←S1*(1+CHAR-SEPARATION)/S;
   XCHRSP←DY*S2;
   YCHRSP←−DX*S2;
   return;
end;
```

La routine SET-CHARSPACE non modifica solo il valore globale CHAR-SEPARATION ma aggiorna anche la dimensione del passo che è in uso. Fra gli argomenti della chiamata alla SET-CHARUP ci sono ancora i valori di x e y che definiscono la direzione verticale, mediante i quali si deduce la direzione della stringa di testo.

Algoritmo 2.15 SET-CHARSPACE (SPACE)
Routine utente che fornisce la spaziatura fra i caratteri.

Argomenti SPACE←frazione della dimensione del carattere che separa i caratteri
Variabili globali CHAR-SEPARATION←valore della spaziatura tra i caratteri
XCHRSP, YCHRSP←spazio tra i centri dei caratteri

```
begin
   CHAR-SEPARATION←SPACE;
   SET-CHARUP(−YCHRSP, XCHRSP);
   return;
end;
```

A questo punto è possibile scrivere l'algoritmo per inserire una stringa nel display file a partire dalla posizione corrente della penna.

Algoritmo 2.16 TEXT (STRING)
Routine utente che inserisce nel display file le istruzioni per stampare una stringa.
Argomenti STRING←caratteri che devono essere inseriti
Variabili globali DF-PEN-X, DF-PEN-Y←posizione della penna
XCHRSP, YCHRSP←spaziatura tra i caratteri
Variabili locali LEN←lunghezza della stringa
X, Y←posizione del carattere
I←contatore dei caratteri
CHR←carattere da memorizzare
OP←codice di operazione corrispondente al carattere

```
begin
   determina la lunghezza della stringa
   LEN←LENGTH(STRING)
   salva la posizione della penna per ripristinarla dopo l'introduzione della
      stringa
   X←DF-PEN-X;
   Y←DF-PEN-Y;
   introduce la stringa
   for I = 1 to LEN
   begin
      considera l'i-esimo carattere della stringa
      CHR←GETCHR(STRING, I);
      forma il suo codice di carattere
      OP←DECODE(CHR);
      lo introduce nel display file
      DISPLAY-FILE-ENTER(OP);
      muove la penna nella successiva posizione del carattere
      DF-PEN-X←DF-PEN-X + XCHRSP;
      DF-PEN-Y←DF-PEN-Y + YCHRSP;
```

```
    end;
    rimette la penna nella sua posizione originale
    MOVE-ABS-2(X, Y);
    return;
end;
```

Le routine LENGTH, GETCHR e DECODE dipendono dalle modalità di rappresentazione di stringhe nel sistema che si sta usando e gli algoritmi per eseguirle non vengono riportati.
La routine LENGTH restituisce il numero dei caratteri nella stringa. La routine GETCHR restituisce l'i-esimo carattere della stringa. La routine DECODE converte, se necessario, il carattere in codice ASCII. Si deve ora modificare l'interprete perché possa trattare questi nuovi comandi di carattere che sono stati introdotti nel display file.

Algoritmo 2.17 INTERPRET (START, COUNT)
(Algoritmo 2.10 aggiornato)
Scandisce il display file eseguendo le istruzioni.
Argomenti START←indice di inizio della scansione del display file
COUNT←numero delle istruzioni che devono essere interpretate
Variabili locali: NTH←indice del display file
OP, X, Y←istruzione del display file

```
begin
   ciclo per eseguire tutte le istruzioni richieste
   for NTH = START to START+COUNT-1
   do begin
         GET-POINT(NTH, OP, X, Y);
         if OP = 1
         then DOMOVE(X, Y)
         else if OP = 2
               then DOLINE(X, Y)
               else if OP>31
                     then DOCHAR(OP, X, Y)
                     else return-error 'ERRORE DI OP-CODE';
   end;
   return;
end;
```

Le istruzioni per introdurre il carattere nel frame buffer sono localizzate nella routine DOCHAR.

Algoritmo 2.18 DOCHAR (OP, X, Y)
Pone un carattere sullo schermo.
Argomenti OP←indica quale carattere deve essere usato

```
                    X, Y←indica la posizione sullo schermo
Variabili globali   WIDTH, HEIGHT←dimensioni dello schermo
                    WIDTH-START, HEIGHT-START←coordinate schermo dell'angolo inferiore sinistro
                    WIDTH-END, HEIGHT-END←coordinate schermo dell'angolo superiore destro
Variabili locali    CHR←carattere da visualizzare
                    X1, Y1←coordinate schermo del carattere
begin
   CHR←CODE(OP);
   X1←MAX(WIDTH-START,MIN(WIDTH-END,X*WIDTH+
                          WIDTH-START));
   Y1←MAX(HEIGHT-START,MIN(HEIGHT-END,Y*HEIGHT+
                          HEIGHT-START));
   GENERATE-CHAR(X1, Y1, CHR);
   return;
end;
```

Ancora una volta ci sono delle routine dipendenti dal particolare sistema che si usa, la CODE e la GENERATE-CHAR (che non vengono riportate). CODE converte da ASCII nella forma richiesta dalla GENERATE-CHAR. GENERATE-CHAR genera un carattere sullo schermo o nel frame buffer nella posizione (X1, Y1). Se si usa una stampante a linee per l'uscita, allora il DDA visto in precedenza può essere usato per la GENERATE-CHAR:

GENERATE-CHAR(X1, Y1, CHR) diventa DDA(X1, Y1, X1, Y1, CHR).

Alcuni esempi della routine GENERATE-CHAR per altri sistemi sono presentati nelle appendici. Ancora una volta si sono introdotte alcune variabili globali che vanno inizializzate dalla routine seguente.

Algoritmo 2.19 INITIALIZE-2B

```
Variabili globali   CHAR-WIDTH, CHAR-HEIGHT←dimensioni del carattere
                    CHAR-SEPARATION←proporzione della dimensione del carattere usata come spaziatura
begin
   INITIALIZE-2A;
   CHAR-WIDTH←ampiezza di un carattere in coordinate normalizzate
   CHAR-HEIGHT←altezza di un carattere in coordinate normalizzate
   CHAR-SEPARATION←0;
   CHARUP(0,1);
   return;
end;
```

2.9 TIPI DI LINEE

Molti dispositivi di visualizzazione consentono di scegliere fra diverse modalità per rappresentare una linea che verranno chiamate *tipo di linea*. Le linee possono essere continue, tratteggiate o punteggiate. Può essere inoltre possibile selezionare il colore, l'intensità e lo spessore e cambiare il tipo in fase di visualizzazione. È quindi necessario disporre di un comando di display file per eseguire tali cambiamenti. Quando l'interprete incontra questo comando, viene eseguito il cambiamento e tutte le linee successive sono disegnate nel nuovo stile. I comandi di display file del sistema proposto sono composti da tre parti: un codice operativo e i due operandi relativi alle coordinate x, y. Per indicare il cambiamento di tipo (di colore o di intensità) si può usare un codice operativo speciale che non richiede però alcun operando. Il display file potrebbe essere organizzato in modo da consentire istruzioni con diversi formati, ma qui si preferirà una soluzione più semplice, che consiste nel fornire degli operandi fittizi che verranno poi ignorati. Si useranno a questo scopo codici operativi negativi per i comandi di cambiamento del tipo di linea. L'identificazione dei codici operativi con i relativi tipi di linea dipenderà dai tipi di linea disponibili. Comunque si assumerà che un codice pari a -1 corrisponda a una linea retta continua. Questo potrebbe essere il tipo assegnato quando il sistema viene inizializzato. Altri tipi corrisponderanno ai codici -2, -3 e così via. Per la visualizzazione su una stampante, così come è stato discusso nel Capitolo 1, il tipo della linea corrisponde al carattere che viene inserito nel frame buffer. Cambiare il tipo di linea significa quindi cambiare questo carattere (vedi Figura 2.17).
L'interprete verrà esteso per poter eseguire i comandi che cambiano il tipo di linea. Per far questo saranno necessarie due nuove routine, una per inserire il comando che definisce il tipo di linea nel display file, l'altra che realizza effettivamente il cambiamento nel momento in cui l'interprete incontra il comando che è stato inserito.
Sebbene i codici operativi per il tipo delle linee siano negativi (per distinguerli da quelli usati per MOVE, LINE e per i caratteri) non c'è ragione di obbligare l'utente ad usare effettivamente dei numeri negativi. L'utente specificherà i codici come positivi 1, 2, ... per ciascun tipo di linea e si farà in modo che sia

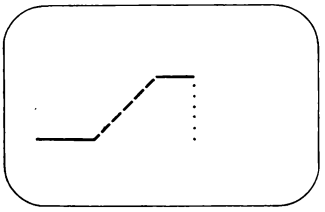


Figura 2.17 Cambiamento del tipo di linea

l'algoritmo SET-LINESTYLE a inserirli come negativi nel display file. L'algoritmo per assegnare il tipo di linea sarà così strutturato:

Algoritmo 2.20 SET-LINESTYLE (LSTYLE)
Routine utente che assegna il tipo di linea.
Argomenti LSTYLE←tipo specificato dall'utente
begin
    DISPLAY-FILE-ENTER(←LSTYLE);
    return;
end;

Quando l'interprete trova un comando di definizione del tipo di linea, deve eseguire il cambiamento. Questa operazione, che dipende dal sistema in uso, verrà enucleata in una routine separata per permettere un semplice interfacciamento con differenti dispositivi.

Algoritmo 2.21 DOSTYLE (OP)
Cambia il tipo di linea.
Argomenti OP←assegna il tipo di linea desiderato
begin
    decodifica OP;
    assegna il tipo di linea;
    return;
end;

Se deve essere usato l'algoritmo DDA del Capitolo 1 per avere un'uscita su stampante o su un CRT con differenti caratteri per rappresentare diversi tipi di linee, l'Algoritmo 2.21 diventerà:

Algoritmo 2.21A DOSTYLE (OP)
Cambia il tipo di linea.
Argomenti OP←assegna il tipo di linea desiderato
Variabili globali LINECHAR←carattere della linea usato da DDA
begin
  LINECHR←LINE-CODE(←OP);
  return;
end;

dove LINE-CODE converte gli interi 1, 2, ... in appropriati codici di carattere, ad esempio *, +, ... Esempi di altri tipi di routine DOSTYLE sono riportati in appendice.
Ancora una volta sarà necessario estendere l'interprete in modo che possa riconoscere i codici operativi negativi.

Algoritmo 2.22 INTERPRET (START, COUNT)
(Algoritmo 2.17 aggiornato)
Scandisce il display file eseguendo le istruzioni.
Argomenti START←indice di partenza della scansione del display file
COUNT←numero di istruzioni che devono essere interpretate
Variabili locali NTH←indice del display file
OP, X, Y←istruzione del display file

```
begin
   ciclo per eseguire le istruzioni desiderate
   for NTH = START to START + COUNT - 1
   do begin
      GET-POINT(NTH, OP, X, Y);
      if OP < 0 then DOSTYLE(OP)
            else if OP = 1
            then DOMOVE(X, Y)
            else if OP = 2
                  then DOLINE (X, Y)
                  else if OP > 31
                        then DOCHAR(OP, X, Y)
                        else return-error
                              'ERRORE DI OP-CODE';
   end;
   return;
end;
```

È necessario un piccolo cambiamento nell'algoritmo DISPLAY-FILE-ENTER in modo che si possa distinguere un'istruzione che usa le informazioni sulle coordinate (LINE e MOVE) da una che non ne fa uso (STYLE). Questa distinzione si mosterà utile nei capitoli successivi laddove non verranno inserite istruzioni per tracciare rette, ma solo cambiamenti del tipo di linea.

Algoritmo 2.23 DISPLAY-FILE-ENTER (OP)
(Algoritmo 2.3 aggiornato)
Combina operazione e posizione per formare un'istruzione da inserire nel display file.
Argomenti OP←operando che deve essere inserito
Variabili globali DF-PEN-X, DF-PEN-Y←posizione corrente della penna

```
begin
   if OP < 0
   then PUT-POINT(OP, 0, 0)
```

```
    else PUT-POINT(OP, DF-PEN-X, DF-PEN-Y);
    return;
end;
```

C'è un'ultima routine che è necessaria per completare questo stadio del sistema: devono essere assegnati i valori iniziali o di default per le variabili globali. In aggiunta alle inizializzazioni precedenti, andrà assegnato anche il tipo di linea iniziale.

Algoritmo 2.24 INITIALIZE-2
Inizializzazione delle variabili che riguardano le rette, i caratteri e il tipo di linea.

```
begin
    INITIALIZE-2B;
    DOSTYLE(-1);
    return;
end;
```

APPLICAZIONE PRATICA

Si voglia scrivere un programma che disegni il grafico di alcuni dati, usando gli algoritmi che sono stati descritti. Inizialmente potranno essere disegnati gli assi verticali e orizzontali del grafico, cioè due rette. Con una chiamata a MOVE-ABS-2 viene raggiunto uno degli estremi e con una LINE-ABS-2 si disegna fino all'altro estremo (vedi Figura 2.18).
Il passo successivo potrebbe essere l'etichettatura degli assi, che può essere fatta per mezzo di una serie di comandi TEXT. Ad esempio, volendo etichettare l'asse orizzontale con i numeri da 1 a 5, decidendo dove ogni numero debba essere posto sul grafico, si eseguirà una MOVE-ABS-2 per mettere la penna nella posizione desiderata, seguita da un comando TEXT per scrivere la cifra (vedi Figura 2.19).
Se si volesse etichettare ciascun asse con una stringa di caratteri che indichi cosa si sta rappresentando lungo quell'asse, una MOVE-ABS-2 posizionerà la penna all'inizio della stringa e un ulteriore comando TEXT la produrrà in uscita (vedi Figura 2.20).
Per l'asse verticale si dovrà cambiare la direzione CHAR-UP dei caratteri, in modo che la stringa venga disegnata in verticale (vedi Figura 2.21).
L'ultima cosa che resta da fare è riportare i dati sul grafico. Occorrerà usare la SET-LINESTYLE per cambiare il tipo di linea in modo che la curva di interpolazione dei dati si distingua dagli assi. Le informazioni da visualizzare dovranno essere fornite al pacchetto grafico sotto forma di matrici di dati o di misure. Per ciascuna di queste misure andranno eseguiti alcuni calcoli al fine di trovare la posizione corrispondente sullo schermo. Il grafico verrà realizzato con delle

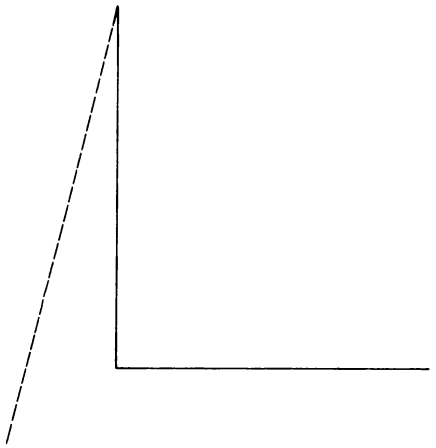


Figura 2.18 Disegno di due assi

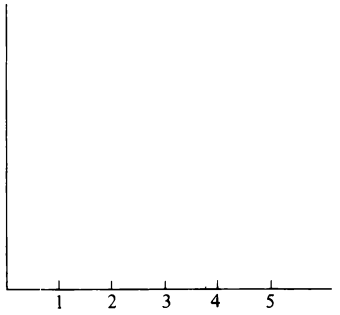


Figura 2.19 Etichettatura di un asse

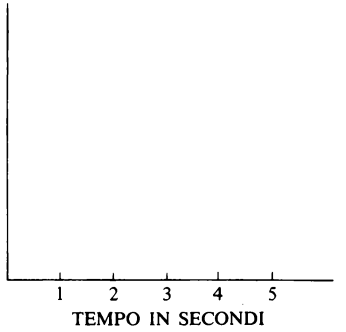


Figura 2.20 Etichettatura ulteriore

chiamate alla routine MOVE-ABS-2 e alla LINE-ABS-2 (vedi Figura 2.22). Il disegno di tutti i punti eccetto il primo viene realizzato solitamente attraverso un ciclo che preleva la misura, calcola la corrispondente posizione sullo schermo e chiama la LINE-ABS-2 per tracciare una retta.

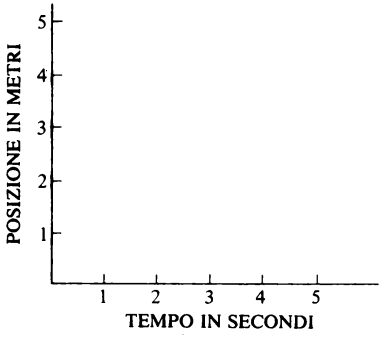


Figura 2.21 Etichettatura con un diverso CHAR-UP

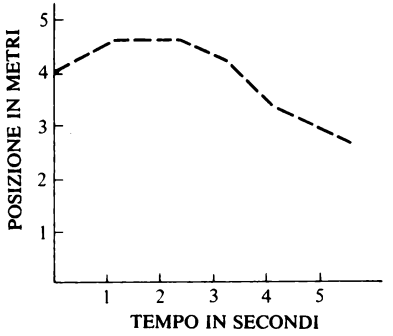


Figura 2.22 Tracciamento della curva con un tipo di linea diverso

ESERCIZI

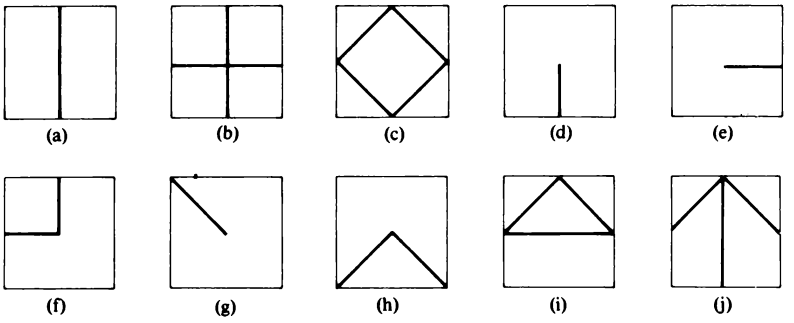
2.1 Si supponga che il dispositivo di visualizzazione a disposizione abbia una risoluzione di 1024×1024. I pixel siano numerati da 0 a 1023 sia nella direzione x che y, cosicché $0 \leq x_s \leq 1023$).

a) Trovare la funzione di conversione $f(x_n)$ per passare da coordinate normalizzate x_n alle corrispondenti coordinate schermo x_s, cioè

$$x_s = f(x_n)$$

b) Se i pixel fossero numerati da 1 a 1024 ($1 \leq x_s \leq 1024$), quale sarebbe la funzione di conversione?

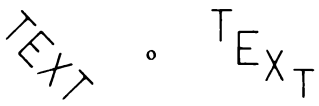
2.2 Scrivere i segmenti di programma che, usando LINE-ABS-2 e MOVE-ABS-2, disegnano ciascuna delle seguenti figure.

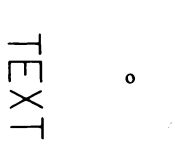


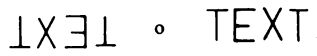
2.3 Assumendo che la penna sia posizionata inizialmente in (0, 0) scrivere i programmi che disegnano le figure dell'esercizio 2.2 usando le procedure LINE-REL-2 e MOVE-ABS-2.

2.4 Scrivere le chiamate alla SET-CHARUP per produrre ciascuna delle seguenti scritte inclinate

a) TEXT

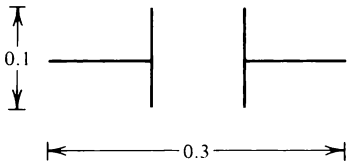
b) 

c) 

d) 

e) 

2.5 Scrivere un singolo sottoprogramma che disegni il simbolo

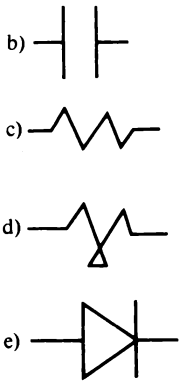


usando solo comandi relativi. L'intero simbolo dovrà essere alto 0.1 unità e largo 0.3 unità. Usando il sottoprogramma 3 volte costruire poi la figura seguente:



2.6 Usando le routine descritte in questo capitolo, scrivere dei segmenti di programma che generino le seguenti immagini.

a) +5 VOLT



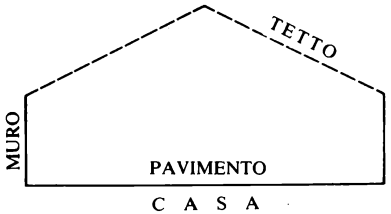
PROBLEMI DI PROGRAMMAZIONE

2.1 a) Implementare gli algoritmi dal 2.1 al 2.12.
b) Implementare gli algoritmi dal 2.13 al 2.18.
c) Implementare gli algoritmi dal 2.19 al 2.23.

2.2 Collaudare le routine del problema di programmazione 2.1 disegnando una casetta.
a) Usare tutte le routine LINE-ABS-2, LINE-REL-2, MOVE-ABS-2 e MOVE-REL-2 per costruire l'immagine della casa.
b) Collaudare le routine per i caratteri etichettando il PAVIMENTO con una strin-

ga orizzontale, il MURO con una stringa verticale, il TETTO con una stringa diagonale e l'intera CASA con una stringa a più ampia spaziatura.

c) Disegnare il tetto della casa con un differente tipo di linea.



2.3 Scrivere le quattro routine
INQUIRE-LINESTYLE(L)
INQUIRE-CHARSPACE(S)
INQUIRE-CHARSIZE(W, H)
INQUIRE-CHARUP(DX, DY)
che restituiscono all'utente lo stato corrente del tipo di retta, la spaziatura tra i caratteri, l'ampiezza e l'altezza dei caratteri e l'ultima verticale del carattere.

2.4 Scrivere due routine chiamate
POLYLINE-ABS-2 (X-ARRAY, Y-ARRAY, N)
POLYLINE-REL-2 (DX-ARRAY, DY-ARRAY, N)

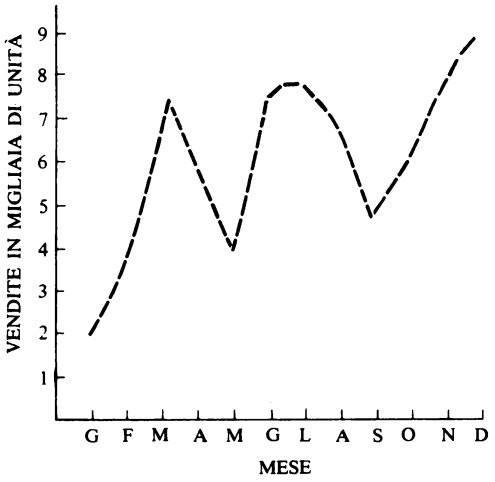
Queste routine inseriscono una sequenza di rette connesse. La sequenza comincia dalla posizione corrente della penna e nel caso della POLYLINE-ABS-2 traccia una retta che connette la sequenza di punti assoluti contenuti in X-ARRAY e Y-ARRAY. Nel caso della POLYLINE-REL-2 la sequenza di punti è data dalle variazioni relative tra di esse, memorizzate in DX-ARRAY e DY-ARRAY. Il numero di elementi degli array è dato da N.

2.5 a) Usando le routine LINE e MOVE descritte in questo capitolo, oppure le routine POLYLINE del problema di programmazione 2.4, disegnare un diagramma dei seguenti dati di vendita.

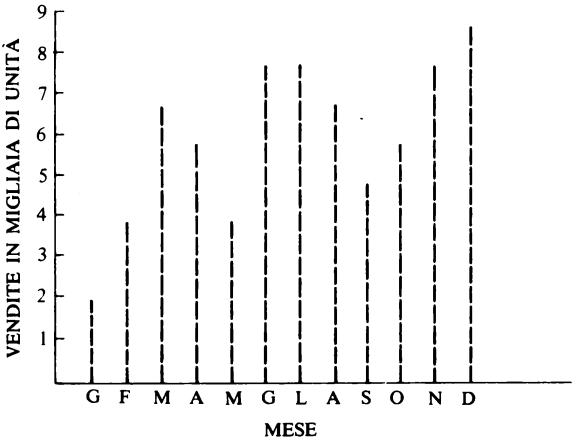
Gen	Feb	Mar	Apr	Mag	Giu	Lug	Ago	Set	Ott	Nov	Dic
2000	4000	7000	6000	4000	8000	8000	7000	5000	6000	8000	9000

b) Disegnare gli assi etichettati per il grafico.

c) Usare linee continue per gli assi e un diverso tipo di linea per rappresentare l'andamento della curva interpolata.

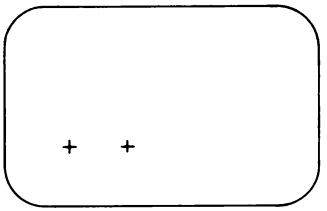


2.6 Costruire un istogramma dei dati del problema di programmazione 2.5 usando le routine LINE e MOVE.

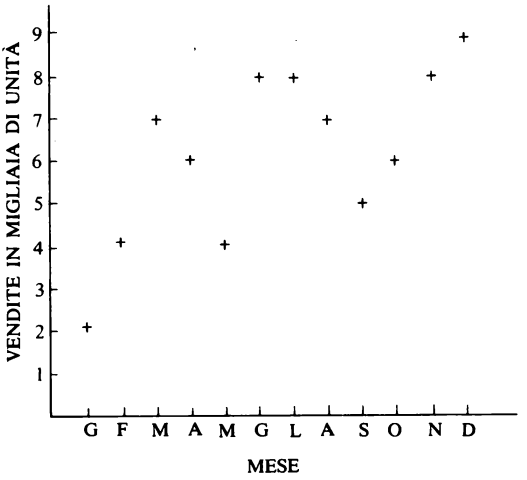


2.7 a) Scrivere tre sottoprogrammi utente chiamati
SET-MARKER-SYMBOL(SYMBOL)
MARKER-ABS-2(X, Y)
MARKER-REL-2(DX, DY)
La routine SET-MARKER-SYMBOL memorizza un carattere o un simbolo perché possa essere usato dalle altre due routine. I due sottoprogrammi MARKER

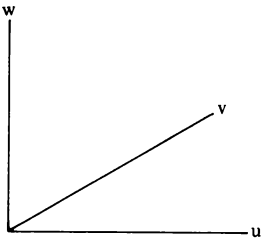
muovono la penna senza disegnare e posizionano il simbolo marker in quella posizione. In questo modo i comandi
SET-MARKER-SYMBOL(2);
MARKER-ABS-2(0.2, 0.3);
MARKER-REL-2(0.2, 0.0);
costruiranno la figura



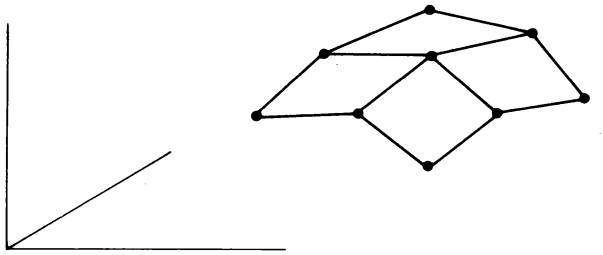
b) Usare le routine precedenti per visualizzare le informazioni sulle vendite del problema di programmazione 2.5.



***2.8** La proiezione di costruzioni tridimensionali e la visualizzazione di superfici verrà considerata nei Capitoli 9 e 10; comunque, in questo problema verrà considerato un semplice modo di presentare informazioni tridimensionali usando solo le routine bidimensionali che sono state viste. Si consideri una funzione a due variabili $w = f(u, v)$. Si possono costruire tre assi coordinati per u, v, w e disegnare un punto (u, v, w) nello spazio.



L'insieme di questi punti forma una superficie. Si può produrre un'immagine di questa superficie disegnando alcuni punti campione connessi attraverso segmenti.



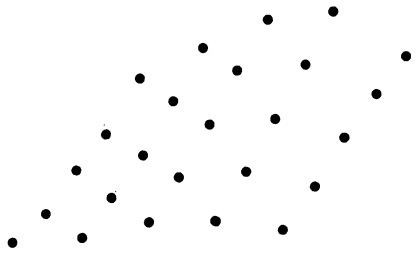
Per posizionare i punti tridimensionali sul nostro schermo bidimensionale, si osservi che se w aumenta il punto si muove verso l'alto, se u aumenta il punto si muove verso destra e se v aumenta il punto si muove sia verso l'alto che a destra.

$$x = u + v/2$$
$$y = w + v/2$$

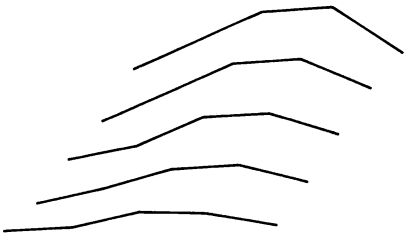
Come esempio di questa tecnica si veda la seguente matrice dei valori in w

$v \setminus u$	0.1	0.2	0.3	0.4	0.5
0.1	0.10	0.10	0.13	0.15	0.20
0.2	0.10	0.12	0.15	0.20	0.25
0.3	0.12	0.15	0.20	0.25	0.30
0.4	0.11	0.15	0.20	0.25	0.30
0.5	0.08	0.11	0.15	0.18	0.20

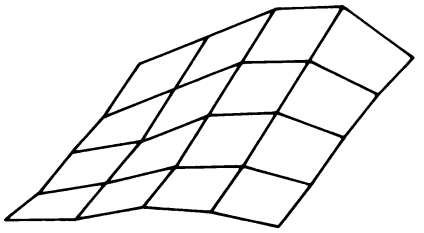
a) Scrivere un programma per disegnare i punti della tabella, usando la formula suddetta per determinare le coordinate (x, y) appropriate.



b) Unire gli elementi successivi di ciascuna riga della tabella con segmenti di retta.



c) Unire con segmenti gli elementi di una stessa riga e di una stessa colonna.



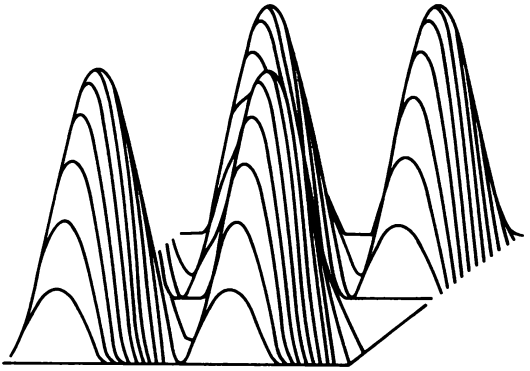
***2.9** Per raggiungere un realismo tridimensionale nel visualizzare una funzione di due variabili non si devono disegnare le linee che sono nascoste dalla superficie. Il seguente metodo è un esempio semplice ma efficace per ottenere questo effetto. Si desidera visualizzare una funzione del tipo $w = f(u, v)$. Come nell'esempio 2.8, si trasformano questi punti nelle coordinate x e y con

$$x = u + v/2$$
$$y = w + v/2$$

Per un dato valore di v si disegneranno delle curve che rappresentano come w varia al variare di u, e ciò sarà ottenuto incrementando i valori di u e calcolando w attraverso la funzione f; noti u, v, w verranno calcolate le coordinate x e y; infine i punti saranno connessi per mezzo di segmenti di retta. Per eliminare le linee nascoste, si potranno disegnare le curve in primo piano per prime. Questo significa partire con i valori più piccoli per v, disegnare la curva rispettiva, incrementare v, disegnare la curva successiva e così via. Si supponga ora di disegnare una curva con un massimo relativo. Questa curva che passa per il massimo nasconderà la curva successiva se il valore y del massimo è maggiore del valore y della curva in esame. Si potrà evitare di tracciare la linea nascosta se, anziché disegnare il segmento di linea al nuovo valore della y, lo si traccerà verso un valore della y che sia il massimo fra quelli trovati per quel valore di x.
A questo fine è necessario mantenere aggiornato un array contenente, per ogni valore di x, il massimo valore di y. Il programma seguente usa questo metodo per disegnare una funzione nell'intervallo $0<u<0.5$, $0<v<0.5$; NUM-CURVES è il numero delle curve disegnate e MAX-TABLE è l'array dei valori massimi delle y e deve contenere almeno il triplo dei valori di NUM-CURVES.

```
begin
   inizializza l'array dei valori massimi
   for I = 1 to 3*NUM-CURVES do MAX-TABLE[I]←0;
      assegna il passo di incremento
      DV←0.5/(NUM-CURVES −1);
      DU←DV*0.5;
      V←0;
      ciclo per disegnare tutte le curve
      for T = 1 to NUM-CURVES
      do begin
         U←0;
         ciclo per disegnare una curva
         for K = 1 to 2*NUM-CURVES −1
         do begin
            calcola un punto della curva
            W←F(U, V);
            X←U+V*0.5;
            Y←W+V*0.5;
            disegna il punto
            MAX-TABLE[T+K−1]←MAX(MAX-TABLE[T+K−1], Y)
            if K = 1 then MOVE-ABS-2(X, MAX-TABLE[T+K−1])
            else LINE-ABS-2(X, MAXTABLE [T+K−1]);
            si muove lungo la curva
            U←U+DU;
            end;
         passa alla curva successiva
         V = V+DV;
         end;
   return;
end;
```

Collaudate questo algoritmo applicandolo alla funzione $f(u, v) = (1/2)(\sin 8\pi u)(\sin 4\pi v)$ per $0 \leq u \leq 0.5$, $0 \leq v \leq 0.5$. La figura seguente è stata prodotta con un valore di NUM-CURVES pari a 25.



3
I poligoni

INTRODUZIONE

Fino a questo momento si è parlato solamente di linee e di caratteri, ma il mondo sembrerebbe piuttosto privo di risalto se fosse fatto di sole rette; sarebbe un mondo di figure filiformi e di contorni privi di qualsiasi consistenza. Il mondo delle figure colorate ed ombreggiate è molto più interessante.
Sfortunatamente le prime applicazioni di Computer Graphics hanno trattato disegni per linee, essenzialmente perché i dispositivi di visualizzazione disponibili (plotter, DVST e terminali a vector refresh) erano dispositivi per il disegno di linee.
I terminali raster, invece, possono visualizzare solidi con la stessa facilità con cui tracciano i loro contorni. Con la tecnologia raster è possibile quindi colorare ed ombreggiare le superfici. Con l'aumento della diffusione dei terminali raster è perciò cresciuta anche l'attenzione per la rappresentazione di superfici. La rappresentazione della superficie degli oggetti è importante persino per i terminali che disegnano per linee.
Nel Capitolo 10 affronteremo il problema di come non mostrare le linee che sarebbero normalmente nascoste; non è sufficiente conoscere gli spigoli per risolvere questo problema: bisogna anche sapere quali superfici siano presenti in modo da poter decidere quale linea possa essere vista e quale no.
In questo capitolo estenderemo il nostro sistema per includere una nuova primitiva grafica: il poligono. Verranno definiti i poligoni e le modalità per rappresentarli; si vedrà come stabilire se un punto è interno ad un poligono ed infine verrà sviluppato un metodo per riempire tutti i pixel interni ad esso.

3.1 GENERALITÀ SUI POLIGONI

Per rappresentare una superficie, la primitiva di base è un *poligono* (cioè una figura piana definita da più lati). Un poligono può essere rappresentato come un insieme di segmenti uniti ai loro estremi per formare una figura chiusa. Oppure può anche essere rappresentato dai punti in cui si congiungono i lati del poligono. I segmenti che formano il perimetro del poligono vengono chiamati *lati* e gli estremi dei lati vengono chiamati *vertici* del poligono. Il più semplice poligono è il triangolo che ha tre lati e tre vertici (vedi Figura 3.1).
I poligoni possono essere divisi in due classi: poligoni *convessi* e poligoni *concavi*. Un poligono convesso è un poligono tale che per ciascuna coppia di punti interni ad esso, tutti i punti sul segmento di retta delimitato dai primi due appartengono al poligono. Un poligono concavo è un poligono non convesso. Un triangolo è sempre convesso (vedi Figura 3.2).

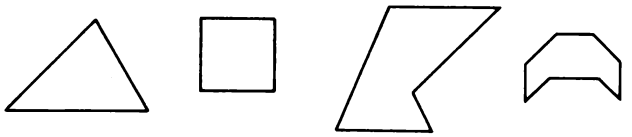


Figura 3.1 Poligoni

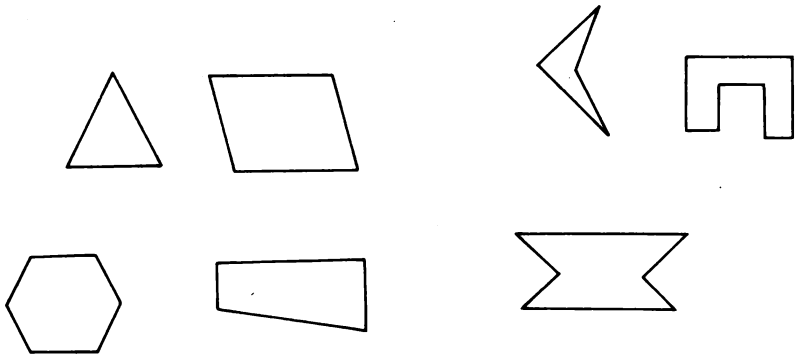


Figura 3.2 Poligoni concavi e convessi

3.2 RAPPRESENTAZIONE DEI POLIGONI

Per aggiungere al nostro sistema grafico i poligoni, si deve prima decidere come rappresentarli. Si potrebbero introdurre soltanto dei comandi per inserire ciascuno dei lati nel display file, ma questo approccio comporta due difetti. Il primo è che non si avrebbe alcun mezzo per sapere quali comandi per disegnare una retta, fra tutti quelli presenti nel display file, debbano essere raggruppati per formare l'entità grafica poligono; il secondo è che la penna non viene correttamente posizionata sul lato iniziale.
È possibile superare questi problemi anteponendo ai comandi per disegnare i lati del poligono un nuovo comando che definirà di quanti lati è composto il poligono in modo da sapere quanti successivi comandi di tracciamento di un segmento riguardano il poligono. Questo nuovo comando, una volta interpretato, provocherà un movimento a penna sollevata nella posizione corretta per disegnare il primo lato. Poiché non sono ancora stati usati i codici di operazione dal 3 a 31 incluso, questi verranno usati per indicare i poligoni. Il valore del codice indicherà il numero dei lati. Si è così limitato il poligono ad un massimo di 31 lati. Gli operandi X e Y del comando per i poligoni saranno le coordinate del punto di inizio del primo lato. Poiché i poligoni sono curve chiuse, il punto iniziale sarà anche il punto finale dell'ultimo lato da disegnare.
Per quanto riguarda l'esecuzione, l'istruzione per i poligoni (codice operativo compreso fra 3 e 31) segnalerà che le istruzioni che seguono appartengono a un poligono, ma si comporterà come un comando di MOVE.

3.3 INTRODUZIONE DEI POLIGONI NEL DISPLAY FILE

Si consideri ora l'algoritmo per introdurre i poligoni nel display file. Le informazioni richieste per definire il poligono sono il numero di lati e le coordinate dei vertici. Possono essere utilizzati degli array per passare le coordinate dei punti alla routine. Se vengono fornite le coordinate assolute si potrà utilizzare il seguente algoritmo (vedi Figura 3.3).

Algoritmo 3.1 POLYGON-ABS-2 (AX, AY, N)
Inserisce un poligono definito in coordinate assolute nel display file.
Argomenti N←numero dei lati del poligono
AX, AY←array per i vertici del poligono
Variabili globali DF-PEN-X, DF-PEN-Y←posizione corrente della penna
Variabili locali I←contatore dei lati del poligono

```
begin
  if N>31 or N<3 then return-error
    'ERRORE NEI LATI DEL POLIGONO';
  inserisce l'istruzione del poligono
  DF-PEN-X←AX[N];
```

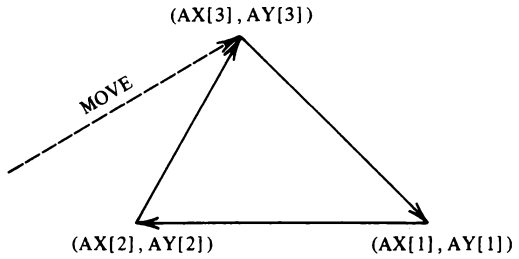


Figura 3.3 POLYGON assoluto

```
    DF-PEN-Y←AY[N];
    DISPLAY-FILE-ENTER(N);
    inserisce l'istruzione per i lati
    for I = 1 to N do LINE-ABS-2(AX[I], AY[I]);
    return;
end;
```

Si vorrà inoltre essere in grado di costruire poligoni posizionati in modo relativo rispetto alla posizione corrente della penna.
In questo caso si dovrà interpretare il primo punto specificato come uno spostamento relativo rispetto alla posizione corrente ed i punti successivi individueranno i lati definendo la posizione di ogni vertice in modo relativo rispetto al vertice precedente (vedi Figura 3.4).

Algoritmo 3.2 POLYGON-REL-2 (AX, AY, N)
Inserisce un poligono definito in coordinate relative nel display file.

Argomenti	N←numero dei lati del poligono
	AX, AY←array per i vertici del poligono
Variabili globali	DF-PEN-X, DF-PEN-Y←posizione corrente della penna
Variabili locali	I←contatore dei lati del poligono

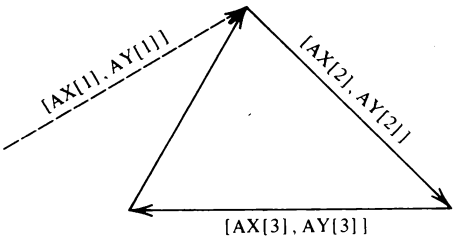


Figura 3.4 POLYGON relativo

```
                    TMPX, TMPY←posizione del punto di chiusura del poli-
                      gono
begin
   if N>31 or N<3 then return-error
      'ERRORE NEI LATI DEL POLIGONO';
   DF-PEN-X←DF-PEN-X+AX[1];
   DF-PEN-Y←DF-PEN-Y+AY[1];
   salva il punto di partenza per poi richiudere il poligono
   TMPX←DF-PEN-X;
   TMPY←DF-PEN-Y;
   inserisce l'istruzione del poligono
   DISPLAY-FILE-ENTER(N);
   inserisce l'istruzione per i lati
   for I = 2 to N do LINE-REL-2(AX[I], AY[I]);
   chiude il poligono
   LINE-ABS-2(TMPX, TMPY);
   return;
end;
```

Nell'algoritmo precedente la posizione di partenza deve essere calcolata a partire dallo spostamento iniziale specificato e dalla posizione corrente della penna. Questa posizione deve essere memorizzata temporaneamente per poterla usare nell'istruzione finale che chiude la figura.

3.4 TEST DI APPARTENENZA

Avendo introdotto i comandi nel display file, è possibile mostrare i poligoni tramite i loro contorni, semplicemente modificando l'interprete in modo che consideri i codici di comando compresi fra il 3 e il 31 come comandi di movimento. Esiste poi la possibilità di visualizzare anche l'area interna ai poligoni accendendo i pixel contenuti fra i contorni dei poligoni. Un metodo per stabilire se un punto è interno ad un poligono è quello di costruire un segmento passante per il punto in questione ed un punto esterno al poligono.
È facile trovare un punto esterno al poligono: si potrebbe per esempio scegliere un punto avente la coordinata x minore della minima fra tutte le coordinate x dei vertici del poligono. Contando il numero di intersezioni fra il segmento ed il contorno del poligono, se questo è un numero dispari, il punto in questione è interno al poligono; un numero pari indica invece che esso è esterno (vedi Figura 3.5).
Contando il numero di punti di intersezione occorre prestare attenzione al caso in cui tale punto di intersezione sia anche un vertice. In questo caso occorrerà considerare anche gli altri estremi dei due segmenti che si intersecano in tale vertice: se questi punti si trovano dalla stessa parte rispetto alla retta costruita,

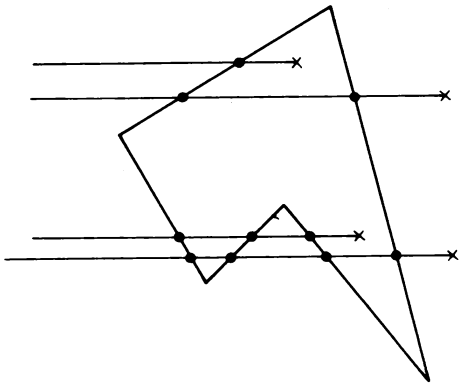


Figura 3.5 Test di appartenenza

il punto d'intersezione viene contato due volte; se giacciono invece da parti opposte rispetto alla retta di costruzione, il punto viene contato come una singola intersezione (vedi Figura 3.6).

3.5 INTERFACCIAMENTO DEI POLIGONI CON IL SISTEMA

Prima di affrontare in modo approfondito le tecniche di riempimento dei poligoni, è meglio considerare l'algoritmo necessario a collegare i metodi di rappresentazione dei poligoni con il resto del sistema grafico che si sta sviluppando.

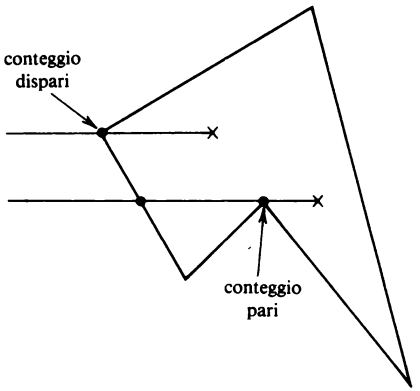


Figura 3.6 Conteggio delle intersezioni con i vertici

Come si è visto, è possibile rappresentare un poligono tracciandone il contorno o riempiendone l'area con tratti di linea: l'utente deve essere in grado di scegliere la tecnica di rappresentazione preferita, mediante l'assegnazione di un valore ad una variabile che verrà controllata nel momento in cui dovrà essere disegnato il poligono.

Algoritmo 3.3 SET-FILL (ON-OFF)
Routine utente che assegna l'opportuno valore alla variabile che indica se deve essere eseguito il riempimento del poligono.
Argomenti ON-OFF←valore indicato dall'utente per la scelta della rappresentazione del poligono
Variabili globali SOLID←indicatore del tipo di rappresentazione del poligono

```
begin
   SOLID←ON-OFF;
   return;
end;
```

Tramite un'altra routine l'utente può scegliere il tipo di riempimento di un poligono, che può essere realizzato utilizzando colori o livelli di grigio differenti, oppure diversi schemi di tracciamento delle linee.

Algoritmo 3.4 SET-FILL-STYLE (STYLE)
Routine utente che assegna il tipo di riempimento di un poligono.
Argomenti STYLE←tipo di riempimento richiesto dall'utente

```
begin
   DISPLAY-FILE-ENTER(-(30+STYLE));
   return;
end;
```

I tipi di riempimento dell'area dei poligoni sono trattati allo stesso modo dei tipi di linea usati per disegnare i contorni: il codice corrispondente ad un tipo di riempimento viene inserito nel display file. In questo sistema verranno usati interi compresi tra -30 e -1 per indicare i tipi di linea ed interi minori di -30 per i tipi di riempimento. Anche in questo caso, l'utente dovrà usare un intero positivo per indicare il tipo di riempimento scelto che verrà convertito in un numero minore di -30 prima dell'inserimento nel display file.
L'algoritmo DOSTYLE, trattato nel Capitolo 2, deve essere modificato perché vengono interpretati i codici corrispondenti sia ai tipi di linea che ai tipi di riempimento.

Algoritmo 3.5 DOSTYLE (OP) (Algoritmo 2.21 aggiornato)
Routine utente che interpreta un comando per il cambiamento del tipo.
Argomenti OP←indica il tipo di linea o di riempimento desiderato dall'utente

```
begin
   if OP< -30
   then decodifica OP e stabilisce il tipo di riempimento
   else decodifica OP e stabilisce il tipo di linea;
   return;
end;
```

Negli algoritmi successivi, il riempimento dell'area di un poligono verrà fatto da una serie di segmenti di retta orizzontali e richiederà l'assegnamento del tipo di linea con cui dovranno essere disegnati tali segmenti e della distanza fra di essi. L'algoritmo INTERPRET va esteso perché dovranno essere interpretati anche i comandi relativi ai poligoni, in modo che il controllo del flusso venga trasferito alla routine DOPOLYGON che elabora le istruzioni per il disegno di un poligono.

Algoritmo 3.6 INTERPRET (START, COUNT)
(Algoritmo 2.22 aggiornato)
Scandisce il display file interpretando le istruzioni in esso contenute.
Argomenti START←posizione da cui occorre iniziare l'interpretazione del display file
COUNT←numero di istruzioni da interpretare
Variabili locali NTH←indice del display file
OP, X, Y←istruzione contenuta nel display file

```
begin
ciclo per eseguire tutte le istruzioni desiderate
   for NTH = START to START+COUNT-1
   do begin
      GET-POINT(NTH, OP, X, Y);
      if OP<0 then DOSTYLE(OP)
         else if OP = 1
         then DOMOVE(X, Y)
         else if OP = 2
            then DOLINE(X, Y)
            else if OP<32
               then DOPOLYGON(OP, X, Y, NTH)
               else if OP>31
                  then DOCHAR(OP, X, Y)
                  else return-error
                  'ERRORE DI OP-CODE';
      end;
   return;
end;
```

Se l'utente ha richiesto un poligono con area piena, assegnando il valore opportuno alla variabile SOLID, questa informazione verrà passata alla routine

FILL-POLYGON che realizzerà il riempimento, altrimenti verrà effettuato un semplice spostamento.

Algoritmo 3.7 DOPOLYGON (OP, X, Y, INDEX)
Elabora un'istruzione relativa ad un poligono.
Argomenti OP, X, Y←istruzione del display file
INDEX←posizione dell'istruzione nel display file
Variabili globali SOLID←variabile che indica se devono essere tracciate le linee di riempimento del poligono

```
begin
   if SOLID then FILL-POLYGON(OP, X, Y, INDEX);
   DOMOVE(X, Y);
   return;
end;
```

3.6 RIEMPIMENTO DEI POLIGONI

Per disegnare anche l'area interna dei poligoni si potrebbe considerare ciascun pixel dello schermo, applicarvi il test di appartenenza ed accenderlo se soddisfa questo test.
Questo metodo, comunque, sarebbe piuttosto costoso in termini di tempo. Molti pixel possono essere eliminati fin dall'inizio confrontandoli con il massimo e il minimo delle coordinate dei punti di contorno. Si considerano quindi soltanto quei pixel che giacciono all'interno del rettangolo più piccolo che contiene il poligono. Se si calcolano dapprima i valori di y rispettivamente più grande e più piccolo dei punti appartenenti al poligono, è possibile considerare

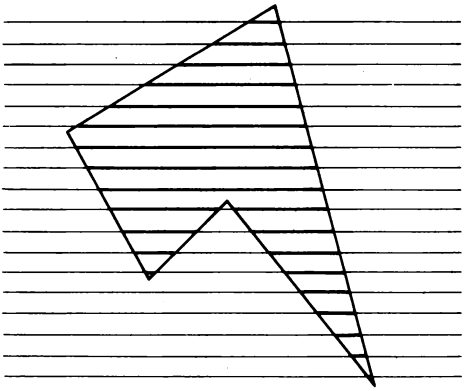


Figura 3.7 Riempimento lungo le linee di scansione

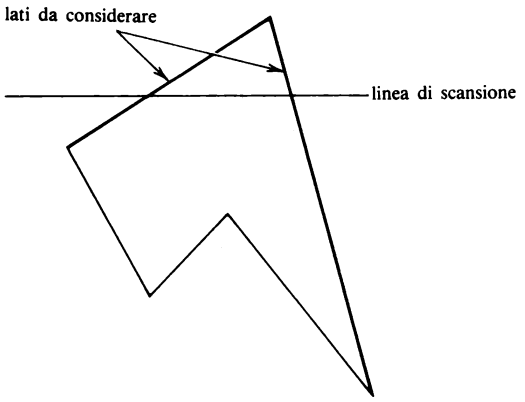


Figura 3.8 Vanno considerati solo i lati che intersecano la linea di scansione

solo i punti compresi fra di essi. Partendo con il massimo valore di y e decrementandolo man mano, si può operare un procedimento di scansione da sinistra verso destra nel modo stesso con cui funziona un terminale raster; le linee così costruite per il test saranno linee orizzontali alla y corrente corrispondente al valore di scansione (vedi Figura 3.7). Per sottoporre un punto al test non occorre calcolare il punto di intersezione del segmento orizzontale di costruzione con ciascun lato del poligono: basta considerare soltanto quei lati del poligono con gli estremi situati da parti opposte rispetto alla linea di test (vedi Figura 3.8). Sarà più semplice identificare quali lati del poligono devono essere sottoposti a test, se i lati vengono ordinati secondo il valore massimo di y. Scandendo l'intero poligono, l'ordine col quale vengono considerati i lati corrisponderà a quello nel quale sono memorizzati.

Ogni volta che il valore di y viene decrementato, contemporaneamente vanno esaminati i lati presi precedentemente in considerazione per stabilire se i loro estremi inferiori sono stati superati. Se il valore inferiore della y di un lato è stato superato, il lato va tolto dall'insieme dei lati che devono essere presi in considerazione. In questo modo, per ciascun valore di y, sappiamo esattamente quali lati del poligono possono essere intersecati (vedi Figura 3.9).

Non resta ora che accendere quei pixel posti sulla linea di scansione orizzontale che appartengono al poligono. Non è necessario esaminare ciascun pixel sulla linea orizzontale: il poligono suddivide la linea di scansione orizzontale in alcuni tratti (vedi Figura 3.10), che sono alternativamente accesi o spenti. Conoscendo gli estremi dei tratti, cioè i punti in cui la linea di scansione attraversa i lati del poligono, è possibile usare la routine DDA (o una equivalente) per riempire l'intero tratto e non occorre quindi considerare ciascun pixel del segmento di scansione separatamente.

Si supponga di calcolare i valori della coordinata x di tutti i punti di intersezione dei lati del poligono con una linea orizzontale e in seguito di ordinarli. Il più

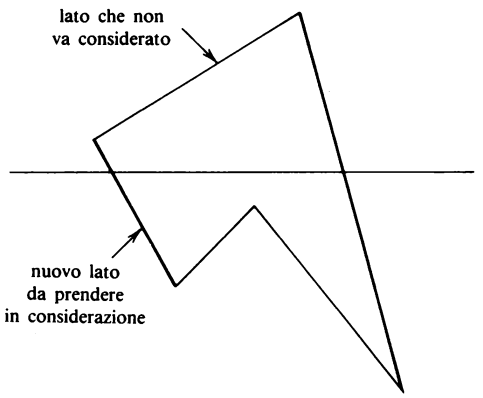


Figura 3.9 Quando la linea di scansione oltrepassa l'estremo *inferiore* di un lato, questo non va più considerato. Quando la linea di scansione passa per l'estremo *superiore* di un lato, questo va preso in esame

piccolo valore di x rappresenta il confine sinistro del poligono: è in questo punto che inizia il poligono. Il successivo valore x indica dove termina il poligono. Un segmento tracciato fra questi due valori riempirà la porzione di poligono compresa. È possibile accoppiare i valori x ordinati in questo modo per passarli alla routine che disegna tratti di retta (vedi Figura 3.11).
Concludendo, un algoritmo per il riempimento di un'area poligonale deve iniziare con l'ordinamento dei lati del poligono secondo valori decrescenti di y. Per ciascun valore y, si deve stabilire quale lato può essere intersecato e calco-

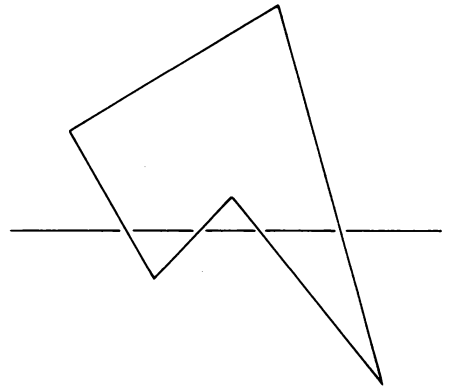


Figura 3.10 Il poligono suddivide la linea di scansione in segmenti

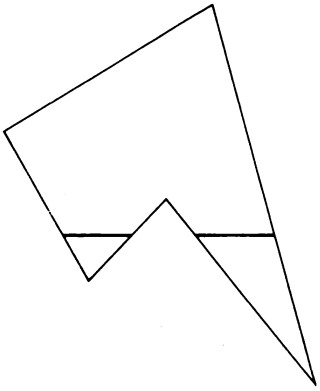


Figura 3.11 Ciascuna coppia di valori x è usata per tracciare un segmento

lare quindi i valori x dei punti di intersezione. I valori x vengono ordinati ed accoppiati per poi essere passati alla routine per disegnare rette.
L'algoritmo che realizza la scansione yx e riempie l'area del poligono è chiamato FILL-POLYGON. Esso inizia attingendo le informazioni riguardanti la forma del poligono dal display file e ordinandole secondo valori decrescenti delle y. Ciò viene fatto tramite l'algoritmo LOAD-POLYGON, mentre il riempimento dell'area del poligono viene fatto tramite la ripetizione di quattro fasi. La prima è per stabilire se esiste un altro lato del poligono che la linea di scansione deve prendere in considerazione: di questo si occupa la routine INCLUDE. La seconda fase esclude quei lati che sono stati già oltrepassati dalla linea di scansione e ne calcola i punti d'intersezione con quelli rimanenti: questa operazione viene eseguita in UPDATE-X-VALUES. La terza fase riempie il segmento della linea di scansione ed è eseguita dalla FILL-SCAN. La quarta fase serve per decrementare la posizione della linea di scansione. Tutti questi passi vengono ripetuti finché i lati del poligono siano stati superati dal procedimento di scansione.

Algoritmo 3.8 FILL-POLYGON (OP, X, Y, INDEX)
Riempie un poligono.

Argomenti	OP, X, Y←istruzione relativa al poligono
	INDEX←indice dell'istruzione nel display file
Variabili globali	YMAX←array che contiene le coordinate y massime dei lati del poligono
	SCAN-DECREMENT←dimensione del decremento delle linee di scansione
Variabili locali	EDGES←numero di lati del poligono considerato
	SCAN←valore y della linea di scansione
	START-EDGE, END-EDGE←lato del poligono che è attraversato dalla linea di scansione

```
begin
   carica l'array globale che contiene le informazioni sui vertici del poli-
      gono
   LOAD-POLYGON(OP, X, Y, INDEX, EDGES);
   verifica se ci sono altri lati da considerare
   if EDGES<2 then return;
   sceglie la linea di scansione
   SCAN←INT(MAX[1]);
   inizializza i valori degli indici di partenza e di arrivo per i lati considerati
   START-EDGE←1;
   END-EDGE←1;
   riempie il poligono
      determina ogni nuovo lato che deve essere incluso in questa scansione
      INCLUDE(END-EDGE, EDGES, SCAN);
      determina le intersezioni del lato con la linea di scansione rimuoven-
         do ogni lato già considerato
      UPDATE-X-VALUES(END-EDGE, START-EDGE, SCAN);
      ripete il riempimento fino a che tutti i lati non siano stati oltrepassati
   while END-EDGE≠START-EDGE;
      do begin
      riempie la linea di scansione
      FILL-SCAN(END-EDGE, START-EDGE, SCAN);
      SCAN = SCAN−SCAN-DECREMENT;
      INCLUDE(END-EDGE, EDGES, SCAN);
      UPDATE-X-VALUES(END-EDGE, START-EDGE, SCAN);
      end;
   return;
end;
```

Si considerino ora più dettagliatamente le informazioni riguardanti i lati del poligono. Interessa per esempio sapere il valore massimo e minimo della coordinata y di un singolo lato. Quello massimo (Y-TOP) indica in quale punto nel procedimento di scansione comincia ad essere preso in considerazione il lato. Il valore minimo di y (Y-BOTTOM) indica quando la linea è stata completamente considerata e quindi non deve essere più presa in esame. Memorizzare la coordinata x potrebbe anche servire per poter determinare dove il lato interseca la linea di scansione. A questo scopo occorre memorizzare anche il valore della x dell'estremo che ha il valore y massimo.
Si avranno così a disposizione le coordinate x e y dell'estremo incontrato per primo durante il procedimento di scansione. Scendendo passo passo con le successive linee di scansione, il punto di intersezione con lo spigolo si sposterà orizzontalmente. È possibile determinare l'entità di tale spostamento conoscendo la variazione di x corrispondente ad una variazione di y lungo lo spigolo, conoscendo cioè il reciproco della pendenza del lato (INVERSE-SLOPE). Le

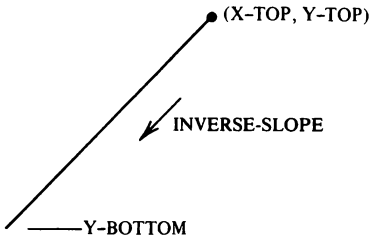


Figura 3.12 Parametri memorizzati per ciascun lato del poligono

quattro entità (X-TOP, Y-TOP, Y-BOTTOM, INVERSE-SLOPE) sono quindi tutto ciò che interessa sapere di ciascun lato (vedi Figura 3.12).
Queste informazioni, riguardanti tutte le linee non orizzontali, verranno memorizzate; quelle orizzontali possono essere tralasciate, essendo parallele alla linea di scansione e non potendo quindi essere intersecate. L'ordine di memorizzazione di queste informazioni è quello a partire da Y-TOP, poiché è questo l'ordine nel quale verrà eseguito l'accesso.
Il trattamento e la memorizzazione dell'informazione riguardante il singolo lato vengono eseguiti da una routine chiamata LOAD-POLYGON. L'algoritmo inizia assegnando un estremo del lato come vertice nel comando originale riguardante il poligono. Il punto è convertito da coordinate normalizzate in coordinate assolute schermo. Il valore della y viene spostato in modo che si trovi tra due pixel: ciò viene fatto per evitare che si verifichino errori quando un vertice del poligono si trova su una linea di scansione. La soluzione adottata è quella di distorcere leggermente la figura in modo che tale situazione non si verifichi

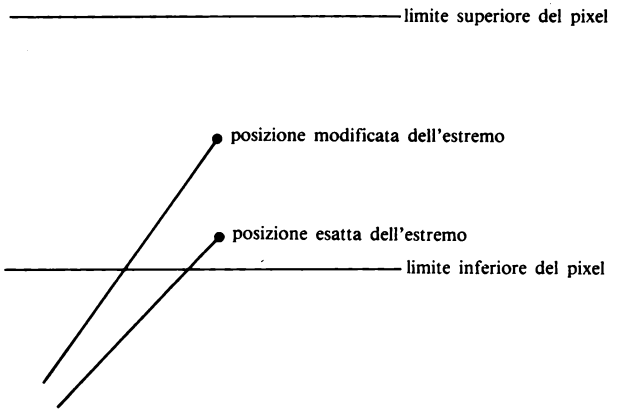


Figura 3.13 Gli estremi vengono spostati al centro del pixel

mai. I vertici del poligono distorto si troveranno sempre almeno a mezzo pixel di distanza dalla linea di scansione (vedi Figura 3.13).
L'algoritmo LOAD-POLYGON prosegue quindi nella ricerca all'interno del display file di un nuovo vertice per mezzo della routine GET-POINT. Il vertice diventa il secondo estremo di un lato del poligono (il primo estremo proveniva dal passo precedente). I lati orizzontali vengono ignorati ma l'informazione riguardante le linee non orizzontali è memorizzata ordinatamente dalla routine POLY-INSERT. Quando tutti i lati sono stati considerati, l'algoritmo restituisce il numero di lati memorizzati fino a quel momento.

Algoritmo 3.9 LOAD-POLYGON (OP, X, Y, I, EDGES)
Recupera le informazioni riguardanti i lati del poligono dal display file. Le coordinate posizionali sono convertite in coordinate schermo assolute.

Argomenti	OP, X, Y←istruzione riguardante i poligoni
	I←indice dell'istruzione del display file
	EDGES←numero di lati memorizzati
Variabili globali	WIDTH-START, HEIGHT-START←indice di partenza dei punti dello schermo
	WIDTH←larghezza dello schermo in pixel
	HEIGHT←altezza dello schermo in pixel
Variabili locali	X1, Y1, X2, Y2←estremi di un lato in coordinate dello schermo assolute
	I1←indice relativo al display file
	K←indice relativo ai lati del poligono
	DUMMY←argomento fittizio
Costanti	ROUNDOFF←numero piccolo maggiore del massimo errore di arrotondamento

```
begin
   assegna il punto di partenza per un lato
   X1←X*WIDTH+WIDTH-START;
   corregge la coordinata y perché si trovi tra due linee di scansione
   Y1←INT(Y*HEIGHT+HEIGHT-START)+0.5;
   preleva l'indice per il comando relativo al primo lato
   I1←I+1;
   inizializza un indice per memorizzare le informazioni sui lati
   EDGES←1;
   ciclo per prelevare le informazioni riguardanti ciascun lato
   for K = 1 to OP
         do begin
            preleva il successivo vertice
            GET-POINT(I1, DUMMY, X2, Y2);
            X2←X2*WIDTH+WIDTH-START;
            Y2←INT(Y2*HEIGHT+HEIGHT-START)+0.5;
            decide se una retta è orizzontale
            if|Y1−Y2|<ROUNDOFF then
```

```
            begin
               X1←X2;
               I1←I1+1;
            end
         else begin
            salva i dati di un lato ordinatamente dalla y
               più grande
            POLY-INSERT(EDGES, X1, Y1, X2, Y2);
            incrementa l'indice del vettore per i dati sui lati
            EDGES←EDGES+1;
            riassegna il valore del punto di partenza
            Y1←Y2:
            X1←X2;
            I1←I1+1;
            end;
         end;
   assegna ad EDGES il numero dei lati effettivamente memorizzati
   EDGES←EDGES−1;
   return;
end;
```

L'algoritmo POLY-INSERT realizza in pratica un ordinamento. Esso calcola il valore massimo di *y* dei due estremi e lo confronta con i lati precedentemente considerati per stabilire in che punto della sequenza si trova il nuovo lato. Il confronto inizia dall'ultimo elemento.
Se il nuovo valore massimo della *y* di un lato è minore di quello dei lati precedenti, esso viene immediatamente accodato. In caso contrario, l'ultimo lato viene spostato in basso di una posizione, lasciando libera una posizione di memoria. Questa operazione di confronto e successivo spostamento di lati viene continuata finché non si trova la giusta posizione per il nuovo lato; è solo a quel punto che l'informazione riguardante il nuovo lato viene inserita.
I dati vengono memorizzati in quattro array separati (uno per ciascun tipo di informazione). Questi array devono essere dimensionati a 31 poiché è questo il massimo numero possibile per i lati di un poligono consentito dal sistema proposto. YMAX è l'array per memorizzare i valori di Y-TOP, YMIN per memorizzare quelli di Y-BOTTOM, XA per memorizzare quelli di X-TOP e DX per contenere il reciproco della pendenza.

Algoritmo 3.10 POLY-INSERT (J, X1, Y1, X2, Y2)
Inserisce ordinatamente le informazioni riguardanti i lati dei poligoni nel display file.

Argomenti	J←indice dell'inserzione
	X1, Y1, X2, Y2←estremi del lato del poligono
Variabili globali	YMAX, YMIN, XA, DX←array per le informazioni sui lati

```
Variabili locali   J1←indice relativo ai lati memorizzati
                   YM←valore massimo del nuovo lato
begin
   inizio dell'ordinamento a partire dal valore massimo della y
   J1←J;
   trova il massimo valore della y
   YM←MAX(Y1, Y2);
   trova il punto corretto di inserzione, spostando opportunamente gli altri
      dati
   while J1≠1 and YMAX(J1−1)<YM
   do begin
      YMAX[J1]←YMAX[J1−1];
      YMIN[J1]←YMIN[J1−1];
      XA[J1]←XA[J1−1];
      DX[J1]←DX[J1−1];
      J1←J1−1;
      end;
   inserisce le informazioni sul lato
   YMAX[J1]←YM;
   DX[J1]←(X2−X1)/(Y2−Y1);
   if Y1>Y2 then
      begin
      YMIN[J1]←Y2;
      XA[J1]←X1;
      end
   else begin
      YMIN[J1]←Y1;
      XA[J1]←X2:
      end;
   return;
end;
```

L'informazione riguardante il lato di un poligono è stata memorizzata senza alcun bisogno di considerare l'intersezione di tutti i lati con ogni linea di scansione: interessano solo quei lati che intersecano la linea di scansione corrente. I lati che devono venir considerati saranno posti solo in una porzione degli array utilizzati per memorizzarli. Per delimitare tale zona verranno usati due puntatori, START-EDGE ed END-EDGE e quindi dovranno venir considerati tutti i lati individuati da indici di valore compreso tra questi due. Un lato che sia memorizzato in una cella dell'array il cui indice è al di fuori di questi due limiti non va considerato (vedi Figura 3.14).
L'algoritmo INCLUDE aggiunge nuovi lati al gruppo di quelli da considerare. A causa dell'ordine con cui essi vengono memorizzati, il lato successivo — che deve essere aggiunto nella lista — sarà il successivo nell'array. Per aggiungere il nuovo lato occorre solamente incrementare la variabile END-EDGE. Quindi,

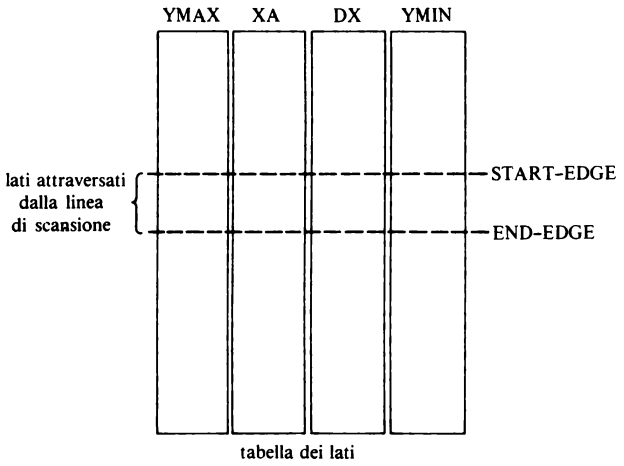


Figura 3.14 Elementi della tabella dei lati che intersecano la linea di scansione corrente

per ciascuna nuova linea di scansione, l'algoritmo INCLUDE controlla il valore massimo di y relativo al nuovo lato; se la linea di scansione si trova al di sotto di questo valore, END-EDGE viene incrementato per aggiungere il lato in questione.

A questo punto vengono eseguite altre due correzioni dell'informazione riguardante il lato. Il primo riguarda la posizione della coordinata x. Il valore di x voluto è quello del punto in cui il lato interseca una linea di scansione, ma il valore che abbiamo memorizzato è l'estremo del segmento che sicuramente non si trova su una linea di scansione per quanto si è detto precedentemente. Per

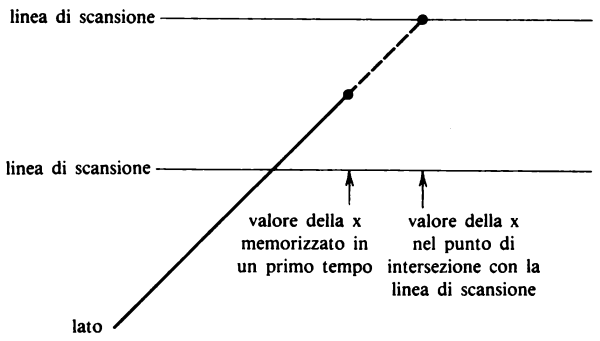


Figura 3.15 Estensione di un lato fino alla linea di scansione

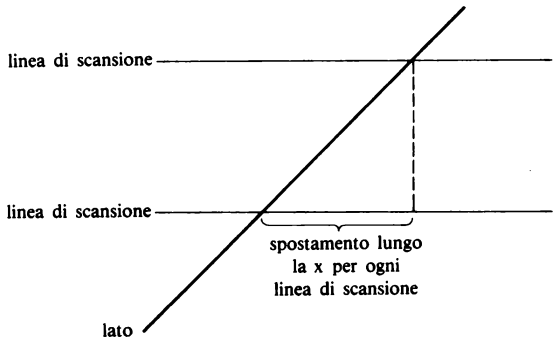


Figura 3.16 Cambiamento della x in corrispondenza di ogni linea di scansione

disporre di un valore corretto per x, estendiamo il lato all'indietro fino alla linea di scansione precedente (vedi Figura 3.15).
La seconda correzione riguarda il passo di variazione della x. Per un decremento costante della scansione, ci sarà una variazione fissa del valore della x del punto di intersezione. È possibile calcolare il valore di tale spostamento moltiplicando la variazione di x in corrispondenza di una variazione unitaria della y per il valore del decremento di scansione. Questo valore è più utile che non l'inverso della pendenza e per questo motivo lo si memorizza (vedi Figura 3.16).

Algoritmo 3.11 INCLUDE (END-EDGE, LAST-EDGE, SCAN)
Aggiunge un nuovo lato intersecato dalla linea di scansione.

Argomenti END-EDGE←indice dell'ultimo elemento della lista corrente dei lati
LAST-EDGE←indice dell'ultimo lato
SCAN←posizione della linea di scansione corrente

Variabili globali YMAX, XA, DX←array per le informazioni relative ai lati
SCAN-DECREMENT←valore del decremento di una linea di scansione

```
begin
  while END-EDGE ≤ LAST-EDGE
  and YMAX[END-EDGE] ≥ SCAN
  do begin
       assegna il punto di partenza alla precedente linea di scansione
       XA[END-EDGE]←XA[END-EDGE]
           +DX[END-EDGE]*(SCAN-DECREMENT
           +SCAN−YMAX[END-EDGE]);
       salva il valore della variazione di x per scansione
```

```
            DX[END-EDGE]←DX[END-EDGE]*
                  (−SCAN-DECREMENT);
            END-EDGE←EDGE-END+1;
            end;
      return;
end;
```

Per ciascuna nuova linea di scansione vanno esaminati i lati identificati degli indici compresi fra START-EDGE e END-EDGE. Se un lato deve ancora essere considerato, bisogna calcolare il valore della sua intersezione con la nuova linea di scansione; se invece è stato superato dalla linea di scansione, il lato non deve venir considerato. Questa operazione è svolta dall'algoritmo UPDATE-X-VALUES. Per stabilire se un lato interseca ancora la linea di scansione, viene esaminato il più piccolo tra i valori delle y degli estremi. L'aggiornamento del punto di intersezione viene eseguito sommando il valore costante del passo calcolato in INCLUDE. La cancellazione di un lato viene eseguita spostando verso l'alto di una posizione l'informazione riguardante i lati precedenti, causando una ricopertura del lato da cancellare e quindi un incremento del valore di START-EDGE (vedi Figura 3.17).

Algoritmo 3.12 UPDATE-X-VALUES (END-EDGE, START-EDGE, SCAN)

Aggiorna i punti di intersezione tra la linea di scansione e i lati del poligono

Argomenti START-EDGE, END-EDGE←limiti della lista corrente dei lati
SCAN←posizione della corrente linea di scansione

Variabili globali YMIN, XA, DX←array per le informazioni dei lati

Variabili locali BEGIN-EDGE, STOP-EDGE←limiti sui lati usati per l'aggiornamento

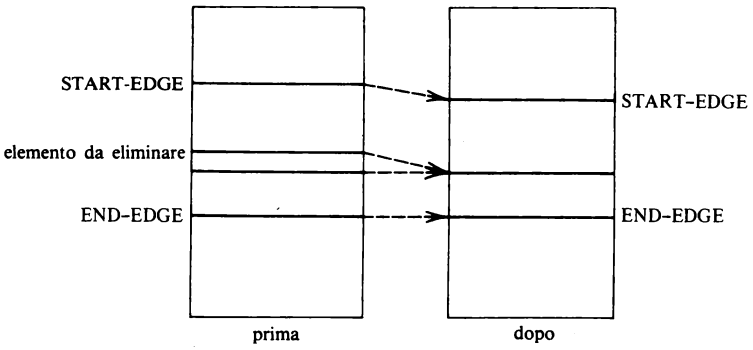


Figura 3.17 Eliminazione di un elemento della tabella

```
                K←indice della lista dei lati
                I←indice della lista dei lati da spostare verso l'alto
begin
   STOP-EDGE←END-EDGE−1;
   BEGIN-EDGE←START-EDGE;
   for K = BEGIN-EDGE to STOP-EDGE
      do begin
            if YMIN[K]<SCAN
            then begin
                  XA[K]←XA[K]+DX[K];
                  XSORT(START-EDGE, K);
            end
            else begin
                  START-EDGE←START-EDGE+1;
                  if START-EDGE ≤ K then
                  for I = K to START-EDGE
                  do begin
                     YMIN[I]←YMIN[I−1];
                     XA[I]←XA[I−1];
                     DX[I]←DX[I−1];
                     end;
            end;
      end;
   return;
end;
```

L'informazione riguardante i lati compresi tra START-EDGE e END-EDGE viene memorizzata in modo ordinato rispetto ai valori della x di intersezione (vedi Figura 3.18).

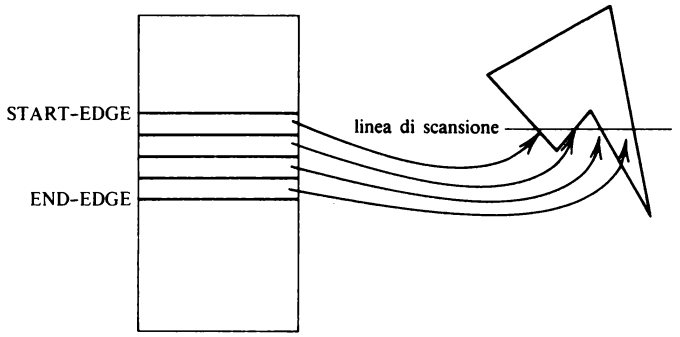


Figura 3.18 Ordinamento secondo x degli elementi della tabella

Il compito di mantenere questo ordinamento viene svolto dall'algoritmo chiamato XSORT. Quando ogni punto di intersezione viene aggiornato da UPDATE-X-VALUES, XSORT viene chiamato per controllare la sua posizione nell'array: se la posizione è corretta non succede nulla, in caso contrario l'elemento è fuori posto e quindi viene fatto risalire alla sua posizione corretta scambiandolo con l'elemento con il quale è stato confrontato. Quando viene introdotto un nuovo lato, può essere necessario spostare verso il basso tutta una parte dell'array, per inserirlo nella posizione corretta. Nei successivi controlli, i lati saranno quasi sempre nella loro posizione corretta, tranne quando i lati di un poligono si attraversano.

Algoritmo 3.13 XSORT (START-EDGE, K)
Controlla che i valori x dei punti di intersezione siano in ordine.
Argomenti START-EDGE←indice del primo lato considerato
K←indice del lato di cui va controllato l'ordine
Variabili globali YMIN, XA, DX←array per le informazioni sui lati
Variabili locali L←indice relativo ai lati

```
begin
   L←K;
   while L>START-EDGE and XA[L]<XA[L−1];
   do begin
      XCHANGE(YMIN[L], YMIN[L−1]);
      XCHANGE(XA[L], XA[L−1]);
      XCHANGE(DX[L], DX[L−1]);
      L←L−1;
      end;
   return;
end;
```

L'algoritmo XSORT usa una semplice routine per scambiare due elementi.

Algoritmo 3.14 XCHANGE (A, B)
Scambia due elementi.
Argomenti A, B←elementi che devono essere scambiati
Variabili locali T←variabile per la memorizzazione temporanea

```
begin
   T←A;
   A←B;
   B←T;
   return;
end;
```

Il successivo algoritmo, FILL-SCAN, visualizza i tratti interni al poligono di una linea di scansione. Esso contiene un ciclo che considera tutti i punti d'in-

tersezione correnti e li connette a due a due con dei segmenti. L'operazione di disegno del segmento è realizzata da FILLIN.

Algoritmo 3.15 FILL-SCAN (END-EDGE, START-EDGE, SCAN)
Riempie la linea di scansione.

Argomenti	START-EDGE, END-EDGE←indicazione dei lati intersecati dalla linea di scansione
	SCAN←posizione della linea di scansione
Variabili globali	XA←array per le posizioni delle intersezioni con i lati
Variabili locali	NX←numero di segmenti che devono essere disegnati
	J←indice degli elementi della lista dei lati
	K←indice degli elementi della lista dei segmenti
	X1←coordinata x iniziale del segmento
	X2←coordinata x finale del segmento
	Y←coordinata y dei segmenti

```
begin
   NX←(END-EDGE−START-EDGE)/2;
   J←START-EDGE;
   for K = 1 to NX
      do begin
         X1←XA[J];
         Y←SCAN;
         X2←XA[J+1];
         FILLIN(X1, X2, Y);
         J←J+2;
         end;
   return;
end;
```

La routine FILLIN può dipendere dal tipo di sistema di visualizzazione usato. In generale essa deve memorizzare il parametro che definisce il tipo di linea corrente in modo che al suo posto venga ricopiato il tipo di riempimento del poligono; infine basterà muoversi in uno degli estremi, disegnare una retta fino all'altro estremo e ripristinare il tipo della linea e la posizione della penna ai loro valori originali. Molti sistemi consentono un'implementazione più semplice. Per una comune stampante, per esempio, è possibile usare soltanto l'algoritmo DDA del Capitolo 1.

L'algoritmo sopra riportato consente di riempire l'area dei poligoni in modo efficiente. Occorre notare, comunque, che il metodo usato per determinare i confini del poligono da riempire può essere diverso da quello che viene usato per tracciare il contorno del poligono. Può succedere che, nel riempire l'area interna del poligono, si ottenga un poligono più ampio di un pixel rispetto a quello definito dai contorni, dando l'impressione di una più bassa risoluzione.

A conclusione degli algoritmi di questo capitolo, è necessario aggiungere una

routine di inizializzazione per assegnare i valori iniziali dei parametri e del tipo di riempimento.

Algoritmo 3.16 INITIALIZE-3

```
begin
   INITIALIZE-2;
   DOSTYLE(-31);
   SET-FILL(FALSE);
   return;
end;
```

APPLICAZIONE PRATICA

A conclusione di questo capitolo viene presentato un esempio di applicazione di alcune delle routine finora sviluppate. Si potrebbe scrivere un programma per disegnare un istogramma di alcuni dati. Si è già visto nel Capitolo 2 che gli assi possono essere disegnati ed etichettati: vediamo ora come generare le barre. È possibile realizzare una barra tramite un poligono a quattro lati (vedi Figura 3.19).

Si supponga di costruire un tale poligono avente l'angolo inferiore sinistro nella posizione corrente della penna, usando il comando POLYGON-REL-2 (AX, AY, 4). I valori necessari, da inserire negli array AX e AY sarebbero:

$$AX = 0, DX, 0, -DX$$
$$AY = 0, 0, DY, 0$$

dove DX è la base del poligono e DY la sua altezza. Fissando DX al valore voluto come ampiezza della barra, tutte le barre avranno la stessa larghezza, ma

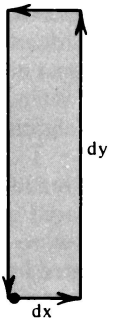


Figura 3.19 Costruzione di una barra tramite un poligono

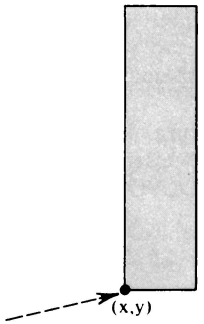


Figura 3.20 Un comando MOVE posiziona il poligono

l'altezza di ciascuna barra dipenderà ovviamente dei dati di ingresso. Ciascuna barra può avere diversa altezza. Per cambiare l'altezza di una barra occorre soltanto modificare il valore del parametro DY nell'array AY; perciò, introducendo il valore corretto e chiamando la routine POLYGON-REL-2, è possibile generare una barra con un'altezza qualsiasi. Ciò che resta da fare a questo punto è disegnare una serie di barre ciascuna in una differente posizione orizzontale e ciascuna con un'altezza che rappresenti il valore del dato per quella posizione. Questo è semplificato dal fatto di aver usato un comando relativo per i poligoni. Per posizionare la barra occorrerà soltanto spostare la penna nella posizione voluta prima di disegnare il poligono (vedi Figura 3.20).

```
MOVE-ABS-2(X,Y);
POLYGON-REL-2(AX, AY, 4);
```

Inserendo queste istruzioni all'interno di un ciclo, ad ogni ripetizione si avrà un dato diverso da disegnare. Una volta che è stato incrementato il valore di X per posizionare la penna nel punto di partenza della successiva barra, si calcola

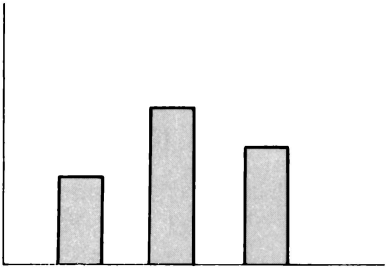
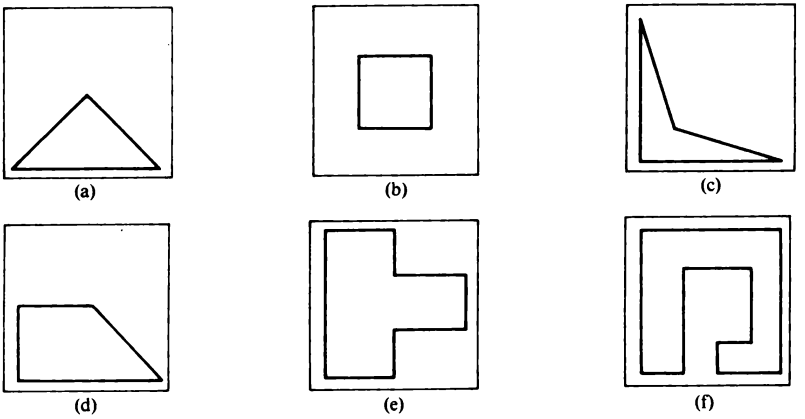


Figura 3.21 Un grafico a barre costruito con successivi comandi POLYGON

l'altezza della barra in base al valore del dato che rappresenta e la si pone nell'array AY. Gli algoritmi MOVE e POLYGON verranno poi utilizzati per disegnare la barra corrente. A ciascun ciclo viene disegnata una nuova barra fino a completare il grafico (vedi Figura 3.21).
Occorre notare che se le barre sono disegnate con spaziatura costante e in modo sequenziale, il comando esplicito MOVE-ABS-2 può essere sostituito dal movimento relativo implicito realizzato dall'algoritmo POLYGON-REL-2.

ESERCIZI

3.1 Per ciascuno dei seguenti poligoni, indicare quali sono concavi e quali convessi:



3.2 Fare uno schizzo di quale sarebbe l'uscita grafica disegnata dal comando POLYGON-ABS-2(AX, AY, N) per ciascuna delle seguenti coppie di array di coordinate:

a) N = 8
AX = 0.2, 0.2, 0.1, 0.1, 0.4, 0.4, 0.3, 0.3
AY = 0.1, 0.3, 0.3, 0.4, 0.4, 0.3, 0.3, 0.1
b) N = 8
AX = 0.5, 0.5, 0.3, 0.3, 0.5, 0.5, 0.1, 0.1
AY = 0.1, 0.2, 0.2, 0.3, 0.3, 0.4, 0.4, 0.1
c) N = 8
AX = 0.2, 0.3, 0.2, 0.3, 0.2, 0.1, 0.2, 0.1
AY = 0.1, 0.1, 0.2, 0.3, 0.4, 0.4, 0.2, 0.2
d) N = 5
AX = 0.1, 0.1, 0.5, 0.9, 0.9
AY = 0.1, 0.6, 0.8, 0.6, 0.1
e) N = 7
AX = 0.5, 0.3, 0.4, 0.4, 0.6, 0.6, 0.7
AY = 0.7, 0.5, 0.5, 0.1, 0.1, 0.5, 0.5

3.3 Tracciare il disegno prodotto da
MOVE-ABS-2(0, 0)
POLYGON-REL-2(AX, AY, N)
per ciascuna delle seguenti coppie di array di coordinate:

a) N = 4
AX = 0.1, 0.1, 0.1, −0.1
AY = 0.2, −0.1, 0.1, 0.1
b) N = 10
AX = 0.4, 0, −0.2, 0.3, 0.1, 0.1, 0.3, −0.2, 0, −0.2
AY = 0.2, 0.2, 0.2, 0, 0.2, −0.2, 0, −0.2, −0.2, 0.2
c) N = 5
AX = 0.1, 0.1, 0.1, 0.1, 0.1
AY = 0.2, 0.1, −0.1, 0.1, −0.1
d) N = 10
AX = 0.5, −0.4, 0, 0.1, 0, 0.1, 0, 0.1, 0, 0.1
AY = 0.3, 0, −0.1, 0, −0.1, 0, 0.1, 0, −0.1, 0
e) N = 11
AX = 0.1, 0.2, −0.1, 0.2, −0.1, 0.2, 0.2, −0.1, 0.2, −0.1, 0.2
AY = 0.1, 0.2, 0, 0.2, 0, 0.2, −0.2, 0, −0.2, 0, −0.2

3.4 Calcolare alcuni valori approssimati per gli argomenti del comando POLYGON-REL-2 per disegnare ciascuno dei poligoni dell'esercizio 3.1.

3.5 Calcolare alcuni valori approssimati per gli argomenti del comando POLYGON-REL-2 per disegnare ciascuno dei poligoni dell'esercizio 3.1, assumendo che la penna sia inizialmente nell'origine del sistema di riferimento.

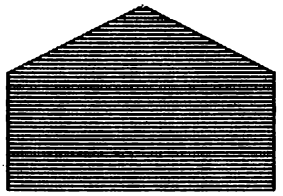
3.6 Si supponga che ad ogni pixel in coordinate assolute schermo corrisponda una linea di scansione. Per ciascuno dei segmenti di retta elencati di seguito (definito attraverso i suoi estremi), si calcoli:

1) La posizione della prima linea di scansione che interseca il segmento di retta.
2) La coordinata x di questo primo punto di intersezione.
3) La variazione della coordinata x e il punto di intersezione per ciascun passo di scansione (evitando, in questo caso, l'arrotondamento della coordinata y al valore medio fra due pixel).
a) (9.4, 3.5); (15.2, 7.8)
b) (2.3, 10.6); (4.3, 5.5)
c) (8.2, 6.6); (14.5, 6.8)
d) (17.7, 8.5); (2.4, 12.2)
e) (18.4, 9.3); (12.3, 4.9)

PROBLEMI DI PROGRAMMAZIONE

3.1 Implementare gli algoritmi dal 3.1 al 3.16 compresi. Adattare le routine DOSTYLE e FILLIN in modo appropriato per il sistema a disposizione.

3.2 Collaudare il problema di programmazione 3.1 disegnando una casetta come poligono di 5 lati. Disegnare il contorno e riempirne l'area con almeno due tipi di riempimento diversi.



3.3 Scrivere le routine chiamate

INQUIRE-FILL-STYLE(I)

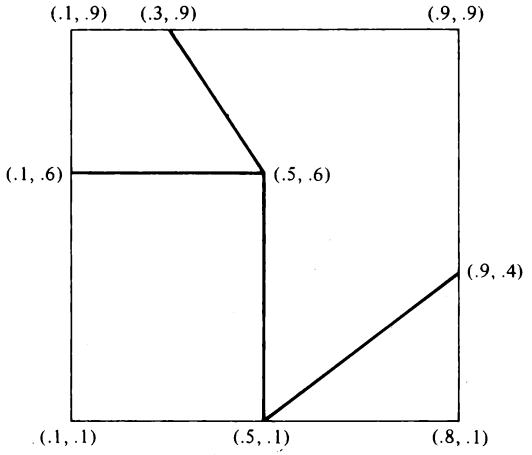
e

INQUIRE-FILLED(ON-OFF)

che restituiscono il tipo di riempimento corrente e il valore dell'indicatore SOLID.

3.4 Scrivere un programma per costruire una figura costituita da N quadrati (ciascuno dei quali sarà un poligono di quattro lati). I vertici dei quadrati saranno di coordinate (J/N, 0) (1, J/N), (1−(J/N), 1) e (0,1−(J/N)), dove J varia da 1 a N compresi.

3.5 Si consideri la seguente mappa di quattro regioni di uno stato immaginario.



a) Costruire la mappa utilizzando dei poligoni, uno per ogni regione (senza però etichettare i vertici con le coordinate).

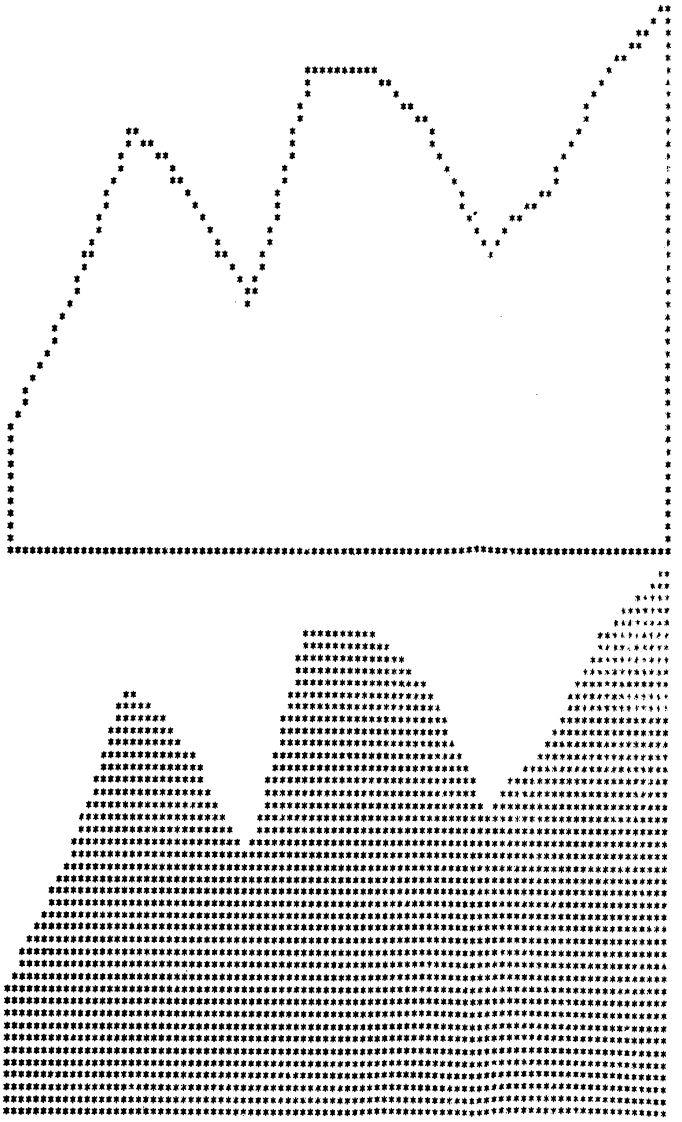
b) Visualizzare la mappa riempiendo ciascuna regione con un motivo diverso.

3.6 Si consideri la seguente tabella di dati di vendite e acquisti.

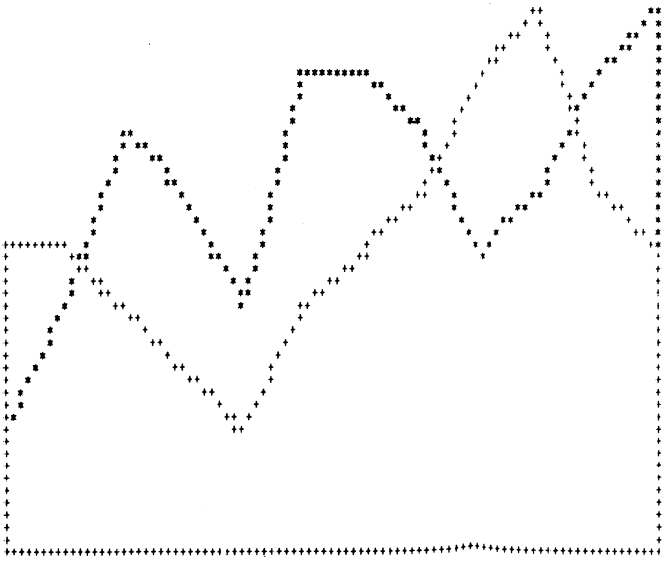
Rapporto annuale

Mese	Gen	Feb	Mar	Apr	Mag	Giu	Lug	Ago	Sett	Ott	Nov	Dic
Vendite	2000	4000	7000	6000	4000	8000	8000	7000	5000	6000	8000	9000
Acquisti	5000	5000	4000	5000	2000	4000	5000	6000	8000	9000	6000	5000

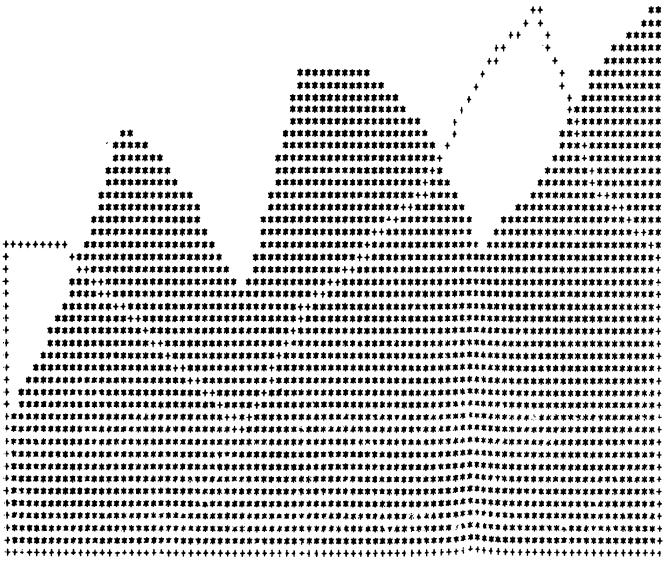
a) Disegnare un poligono delimitato dai dati di vendita e dagli angoli inferiori destro e sinistro del grafico e visualizzarlo sia per contorni che riempito.



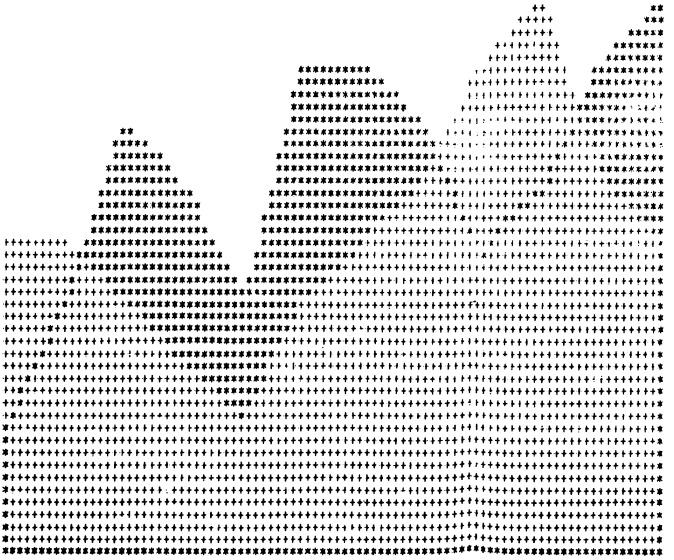
b) Disegnare un analogo poligono per i dati relativi agli acquisti e disegnare entrambi i poligoni insieme senza riempirli.



c) Disegnare per primo il poligono delle vendite riempito e quindi sovrapporvi il poligono degli acquisti non riempito con un tipo di linea per i contorni diverso dal primo.



d) Disegnare il poligono delle vendite riempito, il poligono degli acquisti anch'esso riempito e quindi di nuovo il poligono delle vendite tracciato solo tramite il suo contorno.



e) Costruire un unico poligono più grande, delimitato dal grafico delle vendite e da quello degli acquisti inserito in ordine cronologicamente inverso e visualizzato riempiendo l'area interna.

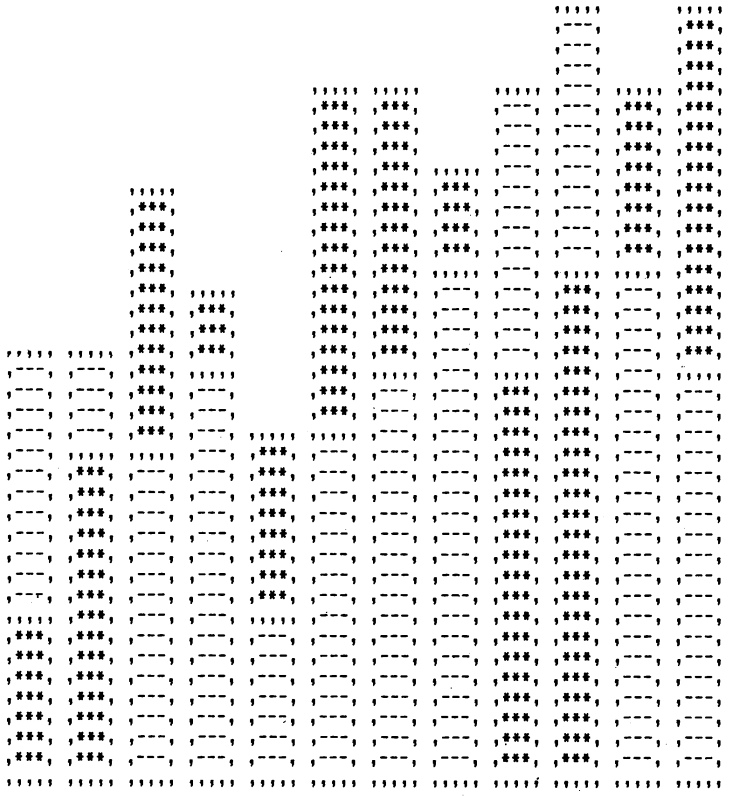


3.7 a) Scrivere un sottoprogramma per disegnare un quadrato con lunghezza dei lati pari a 0.1 usando solo comandi relativi e riempiendo l'area interna. Si disegni quindi una scacchiera riposizionando la penna in modo opportuno e chiamando ogni volta questa subroutine.

b) Costruire altre composizioni di figure ripetitive utilizzando diversi tipi di poligoni.

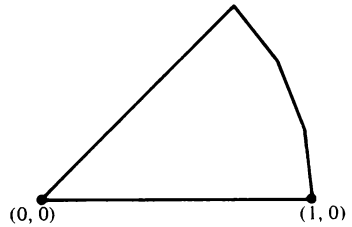
3.8 a) Usando un poligono rettangolare, tracciare il diagramma a barre (istogramma) delle vendite del problema di programmazione 3.6.

b) Disegnare nello stesso diagramma a barre sia i dati riguardanti gli acquisti che quelli delle vendite che sono stati forniti nel problema di programmazione 3.6, utilizzando nei due casi barre con differenti tipi di riempimento e diversi tipi di linea per i contorni. Posizionare le barre di entrambi i gruppi di dati sullo stesso mese ma inserire la maggiore delle due per prima.



3.9 Scrivere una routine WEDGE(A) che disegni un poligono con la forma di una fetta di torta. La fetta deve avere raggio unitario e un lato deve giacere sull'asse x con un

estremo nell'origine. L'angolo che definisce la fetta vale A radianti. Il bordo viene approssimato con segmenti di retta che connettono dei punti appartenenti ad una circonferenza. Il tratto i-esimo viene dato da $X_i = COS(A_i)$, $Y_i = SIN(A_i)$ dove $A_i = (i*A)/N$ e N = INT(4*A). L'indice i deve variare tra 0 e N compresi.



***3.10** Costruire il *font* (forma) per caratteri di grandi dimensioni e/o cifre numeriche costituiti da poligoni riempiti. Scrivere una procedura distinta per ogni simbolo e usare solo comandi relativi in modo tale che i simboli così generati possano essere posizionati a piacere.

****3.11** Se, invece di riempire un poligono, il programma di riempimento per mezzo delle linee di scansione rendesse visibili solo i pixel corrispondenti a punti che giacciono sui lati del poligono, si avrebbe un programma che disegna per linee scandendo la figura dall'alto verso il basso. Questo è esattamente l'ordine con il quale vanno forniti i pixel da visualizzare ad alcuni tipi di schermi di tipo raster ed alle stampanti. Se si immagina che siano state memorizzate informazioni non solo per un poligono ma per un intero display file allora, utilizzando questo programma, si è in grado di fornire i pixel che devono essere visualizzati nel corretto ordine richiesto da questo tipo di dispositivi. Inoltre, per mezzo di un opportuno ordinamento delle istruzioni del display file, è possibile evitare l'uso di un frame buffer così esteso come quello che si è utilizzato fino ad ora. Si assuma che la figura possa essere generata per mezzo dei soli comandi MOVE e LINE. Si progettino le routine DOLINE, DOMOVE e DISPLAY, per tracciare le figure su una stampante, ricavandole per mezzo di una modifica delle routine FILL-POLYGON e di quelle da essa richiamate, eliminando in questo modo la necessità della matrice che costituisce il frame buffer.

4

Le trasformazioni

INTRODUZIONE

Una delle principali caratteristiche funzionali della Computer Graphics è la facilità con cui si possono modificare le figure. Il manager può cambiare la scala di un grafico presente in una sua relazione, l'architetto può vedere un edificio da diverse angolature, il cartografo può variare le dimensioni di una mappa, l'animatore può cambiare la posizione di un personaggio. Queste modifiche sono facilmente realizzabili poiché le informazioni grafiche sono state codificate come numeri e memorizzate in un computer. Ai numeri si possono applicare delle operazioni matematiche che vengono dette *trasformazioni.*

Le trasformazioni permettono di fare delle modifiche uniformi sull'intera figura. Di solito, infatti, è più facile modificare l'intera immagine disegnata dal computer piuttosto che modificarne una sola parte. Ciò può costituire un utile complemento alle tecniche manuali di trattamento delle figure, nelle quali solitamente è più facile modificare piccole porzioni del disegno che creare una figura interamente modificata.

In questo capitolo vengono esaminate le trasformazioni geometriche di scala, traslazione e rotazione e come queste possano essere espresse semplicemente in termini di prodotti di matrici. Verranno introdotte le coordinate omogenee al fine di trattare uniformemente le traslazioni e come anticipazione delle trasformazioni prospettiche tridimensionali. Gli algoritmi presentati descriveranno le routine relative al caso bidimensionale per eseguire le trasformazioni di scala, traslazione e rotazione.

4.1 MATRICI

Le immagini prodotte da un sistema grafico sono generate utilizzando una serie di segmenti di retta definiti dalle coordinate dei loro estremi. Alcune modifiche ad una figura possono essere facilmente fatte tramite operazioni matematiche sui valori di queste coordinate. Prima di considerare alcune delle trasformazioni possibili, occorre richiamare le nozioni algebriche di base relative al prodotto di matrici.
Si considerano qui solamente matrici a due indici; ad esempio:

$$\begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} \qquad \begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{vmatrix} \qquad \begin{vmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{vmatrix}$$

Si supponga di definire la matrice A come:

$$A = \begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} \tag{4.1}$$

Allora l'elemento nella seconda riga e nella terza colonna sarà rappresentato come $A(2, 3)$ e avrà valore 6.
L'operazione tra matrici che interessa maggiormente è la moltiplicazione. Quest'operazione tra matrici è più complessa del semplice prodotto di due numeri, poiché essa si realizza attraverso prodotti e somme di tutti gli elementi della matrice.
Non è possibile applicare la moltiplicazione a qualunque coppia di matrici. Ad esempio è possibile moltiplicare tra loro le matrici A e B se il numero di colonne della prima matrice (A) è uguale al numero di righe della seconda (B). Ad esempio, scegliendo come prima matrice (A) quella nell'equazione (4.1) e la matrice B come

$$B = \begin{vmatrix} 1 & 0 \\ -1 & 2 \\ 0 & 1 \end{vmatrix} \tag{4.2}$$

è possibile moltiplicare A per B in quanto A ha 3 colonne e B ha tre righe. Diversamente dal prodotto fra numeri, il prodotto di matrici non è un'operazione *commutativa* (cioè, se è possibile moltiplicare A per B non è possibile moltiplicare B per A, a causa del fatto che B ha solo 2 colonne, mentre il numero delle righe di A è 3). Il prodotto di due matrici fornisce come risultato una terza matrice che avrà lo stesso numero di righe della prima matrice e lo stesso numero di colonne della seconda. Il prodotto della matrice A che ha dimensioni 3×3 per la matrice B che è di 3×2 fornisce come risultato C di dimensioni 3×2.
Gli elementi della matrice C vengono calcolati a partire dagli elementi di A e di

B applicando la seguente formula:

$$C(i, k) = \sum_j A(i, j)B(j, k) \tag{4.3}$$

Nell'esempio visto $C = AB$, ovvero

$$C = \begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} \quad \begin{vmatrix} 1 & 0 \\ -1 & 2 \\ 0 & 1 \end{vmatrix} \tag{4.4}$$

L'elemento $C(1, 1)$ viene determinato moltiplicando ogni elemento della prima riga di A per il corrispondente elemento della prima colonna di B e sommando, alla fine, tutti i prodotti ottenuti.

$$\begin{aligned} C(1,1) &= A(1, 1)B(1, 1) + A(1, 2)B(2, 1) + A(1, 3)B(3, 1) \\ &= (1)(1) + (2)(-1) + (3)(0) = -1 \end{aligned} \tag{4.5}$$

L'elemento $C(3, 2)$ sarà invece

$$\begin{aligned} C(3,2) &= A(3, 1)B(1, 2) + A(3, 2)B(2, 2) + A(3, 3)B(3, 2) \\ &= (7)(0) + (8)(2) + (9)(1) = 25 \end{aligned} \tag{4.6}$$

Applicando questi calcoli a ciascun elemento di C si otterrà alla fine il valore di C

$$C = \begin{vmatrix} -1 & 7 \\ -1 & 16 \\ -1 & 25 \end{vmatrix} \tag{4.7}$$

Il prodotto di matrici gode della proprietà *associativa*. Questo significa che disponendo di più matrici da moltiplicare fra loro, non ha importanza l'ordine nel quale esse vengono moltiplicate. In notazione matematica:

$$A(BC) = (AB)C \tag{4.8}$$

Questa proprietà è molto utile poiché permette di combinare diverse trasformazioni grafiche per ottenere un'unica trasformazione che ne assomma gli effetti, rendendo il calcolo estremamente più efficiente.
Esiste una matrice che, moltiplicata per una seconda matrice, dà come risultato ancora la seconda matrice. Per questa ragione essa viene detta *matrice identità*. Si tratta di una matrice quadrata (che ha cioè lo stesso numero di righe e di colonne) i cui elementi sono tutti uguali a 0 tranne quelli della diagonale principale che sono tutti uguali ad uno. Per esempio:

$$\begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} \qquad \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Nel caso in cui:

$$I = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \tag{4.9}$$

allora

$$A = AI \tag{4.10}$$

Ad esempio:

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} = \begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \tag{4.11}$$

4.2 TRASFORMAZIONI DI SCALA

A questo punto sorge spontanea la domanda: come si applica tutto questo alla grafica? Si consideri ad esempio il punto $P_1 = [x_1, y_1]$ come una matrice A di dimensioni 1×2. Se la si moltiplica per una qualsiasi matrice T di dimensioni 2×2, si ottiene un'altra matrice 1×2 che può essere interpretata come la rappresentazione di un altro punto

$$[x_2\ y_2] = P_2 = P_1 T \tag{4.12}$$

In questo modo la matrice T definisce una corrispondenza tra il punto di partenza P_1 ed il nuovo punto P_2. Ricordando che un'immagine è memorizzata come lista dei punti estremi dei segmenti, cosa succede se si trasforma ogni punto dell'immagine moltiplicandolo per la matrice T e si visualizza la nuova immagine ottenuta? Come apparirà la nuova figura trasformata? La risposta naturalmente dipenderà dai valori degli elementi della matrice T. Se, ad esempio, T è la matrice identità

$$T = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} \tag{4.13}$$

allora l'immagine risulterà la stessa.
Se invece la matrice di trasformazione è la matrice T_1

$$T_1 = \begin{vmatrix} 2 & 0 \\ 0 & 1 \end{vmatrix} \tag{4.14}$$

allora

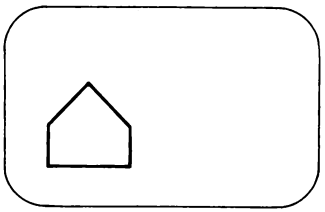
$$[x_2\ y_2] = [x_1\ y_1] \begin{vmatrix} 2 & 0 \\ 0 & 1 \end{vmatrix} = [2x_1\ y_1] \tag{4.15}$$

Ogni nuova coordinata x sarà raddoppiata. Le linee orizzontali diventeranno allora lunghe il doppio rispetto alla vecchia immagine e la nuova immagine sarà alta come la precedente ma apparirà schiacciata e allungata esattamente del doppio (vedi Figura 4.1).
La matrice di trasformazione

$$T_2 = \begin{vmatrix} 0.5 & 0 \\ 0 & 1 \end{vmatrix} \tag{4.16}$$

ha invece l'effetto di ridurre le coordinate x di un mezzo del valore originale. L'immagine avrà la stessa altezza di quella vecchia ma risulterà ristretta della metà.
Se si allunga l'immagine raddoppiandone la larghezza e quindi la si accorcia diminuendola di nuovo della metà,

$$P_2 = (P_1 T_1) T_2 \tag{4.17}$$



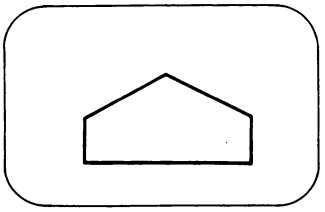


Figura 4.1 Raddoppio della scala per la coordinata x

ci si aspetta di ritrovare alla fine la stessa immagine iniziale. Ciò si dimostra moltiplicando tra loro le due matrici di trasformazione T_1 e T_2. La proprietà associativa della moltiplicazione di matrici permette di scrivere

$$P_2 = P_1(T_1T_2) \tag{4.17.1}$$

Il prodotto delle due matrici le combina quindi in un'unica trasformazione.

$$T_1T_2 = \begin{vmatrix} 2 & 0 \\ 0 & 1 \end{vmatrix} \quad \begin{vmatrix} 0.5 & 0 \\ 0 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} \tag{4.18}$$

Il risultato è esattamente la matrice identità che ci si aspettava, che non modifica l'immagine a cui viene applicata.
È possibile trovare una matrice che rende l'immagine alta il doppio e con la stessa larghezza, scegliendo i suoi elementi in modo tale che le coordinate y siano moltiplicate per 2. La matrice di trasformazione è la seguente

$$T_3 = \begin{vmatrix} 1 & 0 \\ 0 & 2 \end{vmatrix} \tag{4.19}$$

Moltiplicando un punto qualunque per questa matrice si dimostra che ciò è vero (vedi Figura 4.2)

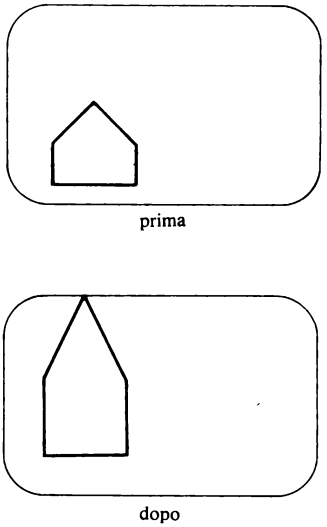


Figura 4.2 Raddoppio della scala per la coordinata y

$$[x_2\ y_2] = [x_1\ y_1] \begin{vmatrix} 1 & 0 \\ 0 & 2 \end{vmatrix} = [x_1 2y_1] \tag{4.20}$$

Applicando entrambe le trasformazioni T_1 e T_3 è possibile generare un'immagine di dimensioni doppie in altezza e in larghezza.

$$P_2 = P_1 T_1 T_3 \tag{4.21}$$

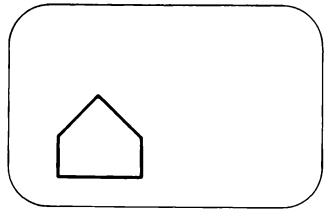
Ancora una volta, moltiplicando le due matrici tra loro, si ottiene una singola trasformazione che serve per raddoppiare le dimensioni di una qualunque immagine (vedi Figura 4.3).

$$T_4 = T_1 T_3 = \begin{vmatrix} 2 & 0 \\ 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 \\ 0 & 2 \end{vmatrix} = \begin{vmatrix} 2 & 0 \\ 0 & 2 \end{vmatrix} \tag{4.22}$$

In generale una trasformazione del tipo

$$S = \begin{vmatrix} S_x & 0 \\ 0 & S_y \end{vmatrix} \tag{4.23}$$

cambia la dimensione e le proporzioni dell'immagine. Le trasformazioni di questo tipo vengono dette *trasformazioni di scala*. S_x è il fattore di scala per le coordinate x e S_y per le coordinate y.



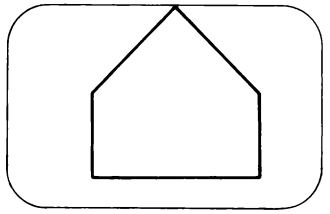


Figura 4.3 Cambiamento di scala per entrambe le coordinate

4.3 FUNZIONI TRIGONOMETRICHE

Si consideri ora la trasformazione che permette di ruotare un'immagine. Per iniziare l'argomento, occorre rivedere i fondamenti di trigonometria. Si supponga di avere un punto $P_1 = (x_1, y_1)$ e di ruotarlo rispetto all'origine di un angolo pari a θ raggiungendo così la nuova posizione $P_2 = (x_2, y_2)$. Bisogna trovare una trasformazione che cambi i valori (x_1, y_1) nei valori (x_2, y_2). Per prima cosa serve sapere come si ottengono i valori di (x_2, y_2) a partire da (x_1, y_1) e da θ. Per determinare questa relazione bisogna disporre delle funzioni trigonometriche *seno* e *coseno* (a cui sono associati i simboli sin e cos). È possibile definire il seno e il coseno di un angolo θ nel seguente modo: si disegni un segmento passante per l'origine, inclinato di un angolo pari a θ misurato in senso antiorario a partire dall'asse x e si supponga inoltre che il segmento disegnato sia di lunghezza L (vedi Figura 4.4). Il segmento di retta avrà quindi per estremi i punti (0, 0) e (x, y) e lunghezza

$$L = (x^2+y^2)^{1/2} \tag{4.24}$$

Si definisce seno dell'angolo θ il rapporto tra l'altezza del punto (x, y) rispetto all'asse x (cioè il valore della coordinata y) e la lunghezza del segmento

$$\sin\theta = \frac{y}{(x^2+y^2)^{1/2}} \tag{4.25}$$

mentre il rapporto tra la distanza del punto dall'asse y (che è la coordinata x del punto) e la lunghezza del segmento viene definito come il coseno dell'angolo θ

$$\cos\theta = \frac{x}{(x^2+y^2)^{1/2}} \tag{4.26}$$

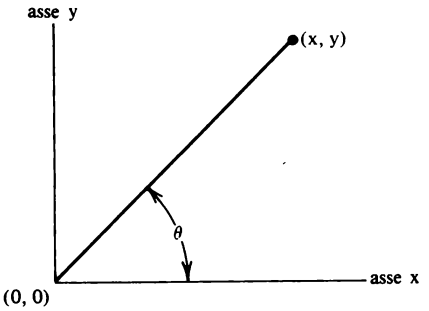


Figura 4.4 Definizione di angolo

Si osservi infine che se la lunghezza del segmento disegnato fosse $L = 1$ si avrebbe

$$\sin\theta = y \qquad \text{e} \qquad \cos\theta = x \tag{4.27}$$

Per eseguire la rotazione di un punto si utilizzeranno le due identità trigonometriche

$$\begin{aligned} \sin(\theta+\phi) &= \cos\phi \sin\theta + \sin\phi \cos\theta \\ \cos(\theta+\phi) &= \cos\phi \cos\theta - \sin\phi \sin\theta \end{aligned} \tag{4.28}$$

Il paragrafo successivo dimostra come si ricavano queste identità; il lettore esperto può evitare questo paragrafo senza problemi.

4.4 RELAZIONI RIGUARDANTI LE SOMME DI ANGOLI

Si dimostra la validità delle identità (4.28) considerando i due triangoli simili mostrati in Figura 4.5. Il segmento di retta che unisce l'origine al punto A forma un angolo pari a θ con l'asse x. Ruotando il punto A attorno all'origine di un angolo ϕ si ottiene il punto B. Siccome viene eseguita soltanto una rotazione attorno all'origine, la lunghezza OA è uguale alla lunghezza OB. Utilizzando le definizioni del capitolo precedente, segue che

$$\frac{CA}{OA} = \sin\theta \qquad \frac{OC}{OA} = \cos\theta \tag{4.29}$$

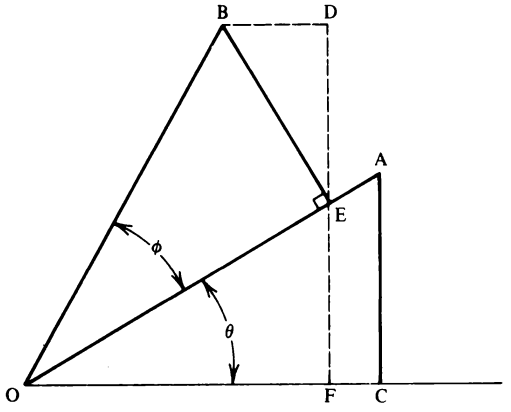


Figura 4.5 Somma di angoli

Se si immagina per un istante che l'asse x coincida col segmento OA invece che col segmento OC, si vede che

$$\frac{EB}{OB} = \sin\phi \qquad \frac{OE}{OB} = \cos\phi \tag{4.30}$$

La lunghezza FE e la lunghezza OF possono essere calcolate osservando che i due triangoli OEF e OAC sono simili, e quindi

$$\frac{OE}{OA} = \frac{FE}{CA} = \frac{OF}{OC} \tag{4.31}$$

$$FE = \frac{OE}{OB}\frac{CA}{OA}OB = \cos\phi \sin\theta\, OB \tag{4.32}$$

$$OF = \frac{OE}{OB}\frac{OC}{OA}OB = \cos\phi \cos\theta\, OB \tag{4.33}$$

Ma anche i triangoli EBD e OAC sono simili e questo consente di determinare la lunghezza dei segmenti ED e DB.

$$\frac{EB}{OA} = \frac{DB}{CA} = \frac{ED}{OC} \tag{4.34}$$

$$DB = \frac{EB}{OB}\frac{CA}{OA}OB = \sin\phi \sin\theta\, OB \tag{4.35}$$

$$ED = \frac{EB}{OB}\frac{OC}{OA}OB = \sin\phi \cos\theta\, OB \tag{4.36}$$

Ora sostituendo

$$\sin(\theta+\phi) = \frac{FD}{OB} = \frac{FE}{OB} + \frac{ED}{OB} = \cos\phi \sin\theta + \sin\phi \cos\theta \tag{4.37}$$

e analogamente per il coseno

$$\cos(\theta+\phi) = \frac{OF}{OB} - \frac{DB}{OB} = \cos\phi \cos\theta - \sin\phi \sin\theta \tag{4.38}$$

Le (4.37) e (4.38) sono, come si può notare, le formule (4.28).

4.5 ROTAZIONI

Si hanno ora a disposizione tutti gli strumenti per determinare la matrice di trasformazione per ruotare un punto rispetto all'origine (vedi Figura 4.6). Se il punto A ha coordinate (x_1, y_1) e una distanza dall'origine pari a

$$L = (x_1^2 + y_1^2)^{1/2} \tag{4.24.1}$$

allora

$$\sin\theta = \frac{y_1}{L}$$

$$\cos\theta = \frac{x_1}{L} \tag{4.39}$$

L'angolo θ è compreso tra la retta che congiunge l'origine con il punto e l'asse x. Se il punto viene di nuovo ruotato dell'angolo ϕ fino alla posizione (x_2, y_2) allora

$$\sin(\theta + \phi) = \frac{y_2}{L} = \cos\phi \sin\theta + \sin\phi \cos\theta \tag{4.40}$$

dalla quale si ottiene

$$y_2 = \cos\phi \sin\theta L + \sin\phi \cos\theta L = \cos\phi\, y_1 + \sin\phi\, x_1 \tag{4.41}$$

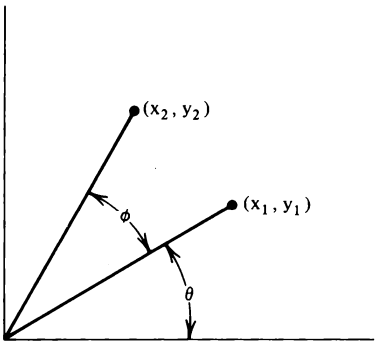


Figura 4.6 Rotazione di un punto attorno all'origine

Analogamente si avrà

$$\cos(\theta+\phi) = \frac{x_2}{L} = \cos\phi\cos\theta - \sin\phi\sin\theta \tag{4.42}$$

che fornisce l'altra coordinata

$$x_2 = \cos\phi\, x_1 - \sin\phi\, y_1 \tag{4.43}$$

Per rappresentare queste equazioni è possibile usare una matrice di trasformazione.

$$[x_2\ y_2] = [x_1\ y_1] \begin{vmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\theta \end{vmatrix} \tag{4.44}$$

La matrice di trasformazione per una rotazione antioraria pari ad un angolo ϕ attorno all'origine è

$$R = \begin{vmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{vmatrix} \tag{4.45}$$

Si supponga, ad esempio, di volere ruotare il punto (2, 3) in senso antiorario di un angolo pari a $\pi/6$ radianti; la matrice di rotazione sarà

$$\begin{vmatrix} \cos(\pi/6) & \sin(\pi/6) \\ -\sin(\pi/6) & \cos(\pi/6) \end{vmatrix} = \begin{vmatrix} 0.866 & 0.5 \\ -0.5 & 0.866 \end{vmatrix} \tag{4.46}$$

e il punto ottenuto con la rotazione sarà

$$[2\ 3] \begin{vmatrix} 0.866 & 0.5 \\ -0.5 & 0.866 \end{vmatrix} = [0.232\ \ 3.598] \tag{4.47}$$

(vedi Figura 4.7). Per ruotare un segmento di retta basta ruotare entrambi gli estremi.
Per poter eseguire rotazioni in senso orario si utilizzano angoli negativi e la matrice di rotazione per un angolo θ sarà

$$R = \begin{vmatrix} \cos(-\theta) & \sin(-\theta) \\ -\sin(-\theta) & \cos(-\theta) \end{vmatrix} \tag{4.48}$$

e poiché

$$\cos(-\theta) = \cos\theta \tag{4.49}$$

e

$$\sin(-\theta) = -\sin\theta \tag{4.50}$$

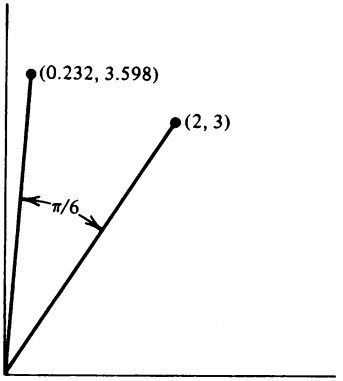


Figura 4.7 Rotazione di $\pi/6$ del punto (2, 3)

sarà meglio riscriverla nella forma seguente

$$R = \begin{vmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{vmatrix} \tag{4.51}$$

che serve per indicare le rotazioni in senso orario attorno all'origine.

4.6 COORDINATE OMOGENEE E TRASLAZIONI

Si voglia ruotare un punto rispetto ad un altro punto qualunque, diverso dall'origine. Disponendo di un modo per muovere l'intera immagine sullo schermo, si potrebbe realizzare tale rotazione muovendo l'immagine fino a che il punto, che costituisce il centro di rotazione, coincida con l'origine, operando la rotazione nel modo che è già stato descritto ed infine riportando l'immagine nel punto di partenza.
Muovere l'immagine parallelamente a se stessa è un'operazione che viene detta *traslazione*. Questa operazione si realizza facilmente sommando a ciascuno dei punti che compongono l'immagine, le quantità di cui vogliamo traslare la figura. Se, ad esempio, si vuole eseguire una traslazione verso destra di 2 unità, si dovrà sommare 2 alle coordinate x di tutti i punti. Oppure, per muoverla verso il basso di 1 unità, si sommerà -1 a tutte le coordinate y (vedi Figura 4.8). Più in generale, per traslare una figura verso destra e verso l'alto delle quantità (T_x, T_y), occorre trasformare ciascun punto (x_1, y_1) nel punto (x_2, y_2) tale che

$$x_2 = x_1 + T_x, \quad y_2 = y_1 + T_y \tag{4.52}$$

Sfortunatamente, questo modo di descrivere le traslazioni non fa uso di matrici

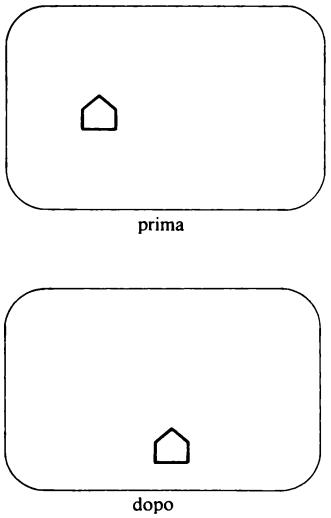


Figura 4.8 Traslazione

e perciò non è possibile combinarlo con altre trasformazioni utilizzando la semplice operazione di prodotto tra matrici, anche se la combinazione sarebbe utile; per esempio, si è visto che la rotazione attorno ad un punto diverso dall'origine può essere ottenuta attraverso una traslazione, una rotazione ed un'altra traslazione. Sarebbe utile combinare queste tre trasformazioni in una singola per guadagnare in efficienza ed eleganza. Un modo per risolvere questo problema è di usare matrici 3×3 invece che 2×2, introducendo una coordinata fittizia w: questo metodo è detto delle *coordinate omogenee*. In coordinate omogenee, un punto è definito da tre numeri anziché due. La prima coordinata omogenea sarà il prodotto di x per w, la seconda sarà il prodotto tra y e w e la terza sarà proprio w. Un punto di coordinate (x, y) sarà rappresentato dalla tripla (xw, yw, w). Le coordinate x e y si possono facilmente ottenere dividendo il primo e il secondo numero per il terzo. In realtà la coordinata aggiuntiva w non verrà usata finché non saranno considerate le trasformazioni prospettiche tridimensionali. In due dimensioni il suo valore è solitamente uguale a 1 per semplicità. La questione, nella sua generalità, verrà trattata quando si parlerà di trasformazioni tridimensionali.
In coordinate omogenee la matrice per le trasformazioni di scala

$$\begin{vmatrix} S_x & 0 \\ 0 & S_y \end{vmatrix}$$

diventa

$$S = \begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix} \tag{4.53}$$

Applicando questa matrice al punto (xw, yw, w) si otterrà

$$[xw \quad yw \quad w] \begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix} = [S_x xw \quad S_y yw \quad w] \tag{4.54}$$

Dividendo per il terzo elemento w si avrà

$$(S_x x, S_y y) \tag{4.54.1}$$

che rappresenta il punto ridotto di scala.
La matrice per le rotazioni in senso antiorario

$$\begin{vmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{vmatrix} \tag{4.54.2}$$

diventa, usando le coordinate omogenee

$$R = \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} \tag{4.55}$$

Applicando tale matrice al punto (x, y), che con le coordinate omogenee diventa (xw, yw, w), si ottiene

$$[xw \quad ym \quad w] \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} =$$

$$= [(xw\cos\theta - yw\sin\theta) \quad (xw\sin\theta + yw\cos\theta) \quad w] \tag{4.56}$$

che corrisponde al punto ruotato

$$(x\cos\theta - y\sin\theta, \; x\sin\theta + y\cos\theta) \tag{4.56.1}$$

La matrice di trasformazione per le traslazioni pari a T_x e T_y, in coordinate omogenee è

$$T = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{vmatrix} \tag{4.57}$$

Per dimostrare questo fatto si deve applicare la matrice

$$[xw \quad yw \quad w] \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{vmatrix} = [(xw + T_x w) \quad (yw + T_y w) \quad w] \quad (4.58)$$

ottenendo il punto traslato di coordinate $(x + T_x, y + T_y)$.

4.7 ROTAZIONI ATTORNO A UN PUNTO ARBITRARIO

Si consideri a questo punto la matrice di trasformazione che realizza la rotazione in senso antiorario attorno ad un punto qualunque (x_C, y_C) (vedi Figura 4.9). Come si è già detto questa operazione viene eseguita in tre passi: traslazione del punto (x_C, y_C) nell'origine, rotazione attorno all'origine ed infine nuova traslazione nel punto di partenza (vedi Figura 4.10).
La matrice di trasformazione che trasla il punto (x_C, y_C) nell'origine è

$$T_1 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_C & -y_C & 1 \end{vmatrix} \quad (4.59)$$

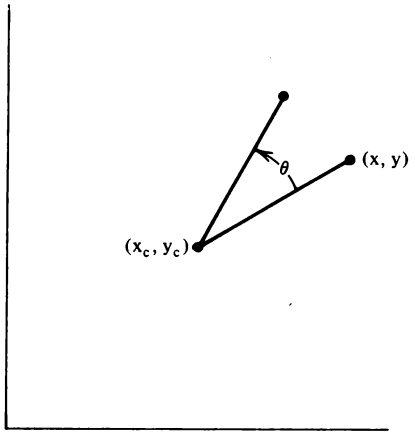


Figura 4.9 Rotazione attorno a un punto arbitrario

Quella di rotazione è

$$R = \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} \tag{4.60}$$

e la traslazione che riporta il centro di rotazione nella posizione corretta è rappresentata da

$$T_2 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_C & y_C & 1 \end{vmatrix} \tag{4.61}$$

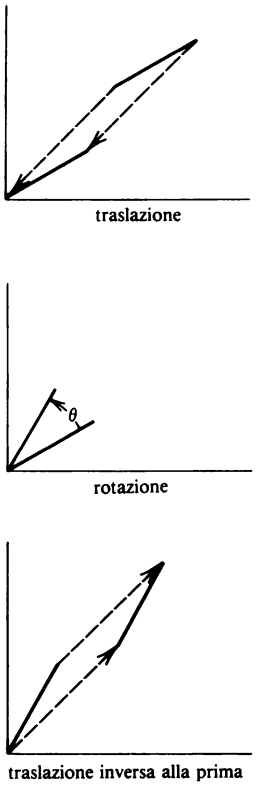


Figura 4.10 Le tre fasi della rotazione attorno a un punto arbitario

Per eseguire una trasformazione su un generico punto basterà moltiplicare

$$((([xw \quad yw \quad w]\,T_1)R)\,T_2)$$

ottenendo le tre matrici di trasformazione unificate in un'unica matrice

$$[xw \quad yw \quad w](T_1(R\,T_2))$$

$$T_1\,R\,T_2 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_C & -y_C & 1 \end{vmatrix} \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_C & y_C & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_C & -y_C & 1 \end{vmatrix} \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ x_C & y_C & 1 \end{vmatrix}$$

$$= \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_C\cos\theta + y_C\sin\theta + x_C & -x_C\sin\theta - y_C\cos\theta + y_C & 1 \end{vmatrix} \tag{4.62}$$

Questa è la matrice che complessivamente realizza una rotazione antioraria θ attorno al punto (x_C, y_C); la stessa trasformazione può essere ottenuta applicando una rotazione iniziale θ seguita da una sola traslazione descritta dai valori della terza riga della matrice.

4.8 ALTRE TRASFORMAZIONI

Le tre trasformazioni di scala, rotazione e traslazione sono quelle più comunemente utilizzate, anche se esistono altri tipi di trasformazioni. Una qualunque matrice di dimensioni 2×2 del tipo

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix}$$

può essere convertita in una matrice 3×3 in coordinate omogenee come la seguente

$$\begin{vmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Verranno qui di seguito riportate solo le matrici 2×2 per alcune delle trasformazioni più usate:

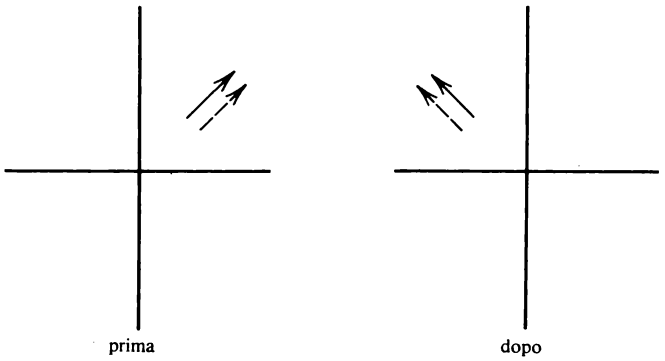


Figura 4.11 Simmetria rispetto all'asse y

$\begin{vmatrix} -1 & 0 \\ 0 & 1 \end{vmatrix}$ simmetria rispetto all'asse y (vedi Figura 4.11)

$\begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix}$ simmetria rispetto all'asse x (vedi Figura 4.12)

$\begin{vmatrix} -1 & 0 \\ 0 & -1 \end{vmatrix}$ simmetria rispetto all'origine (vedi Figura 4.13)

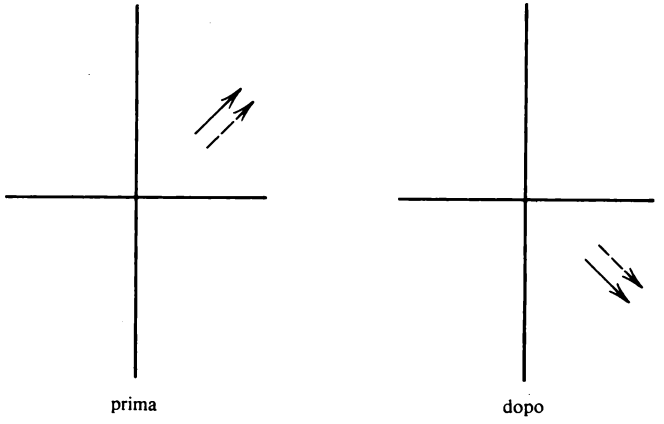


Figura 4.12 Simmetria rispetto all'asse x

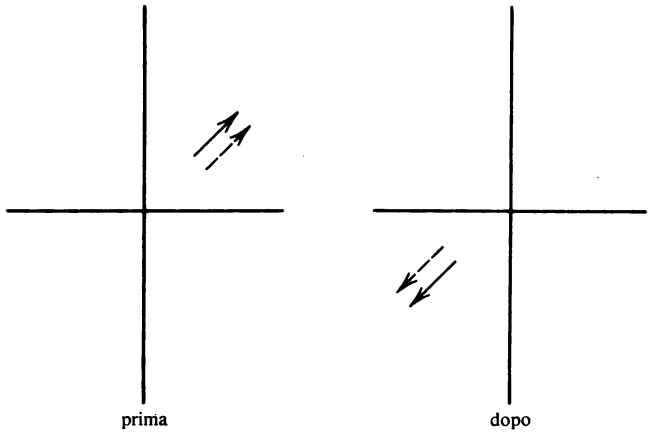


Figura 4.13 Simmetria rispetto all'origine

$\begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix}$ simmetria rispetto alla retta di equazione $y = x$ (vedi Figura 4.14)

$\begin{vmatrix} 0 & -1 \\ -1 & 0 \end{vmatrix}$ simmetria rispetto alla retta di equazione $y = -x$ (vedi Figura 4.15)

$\begin{vmatrix} 1 & a \\ 0 & 1 \end{vmatrix}$ deformazione di taglio (*shear*) lungo l'asse y (vedi Figura 4.16)

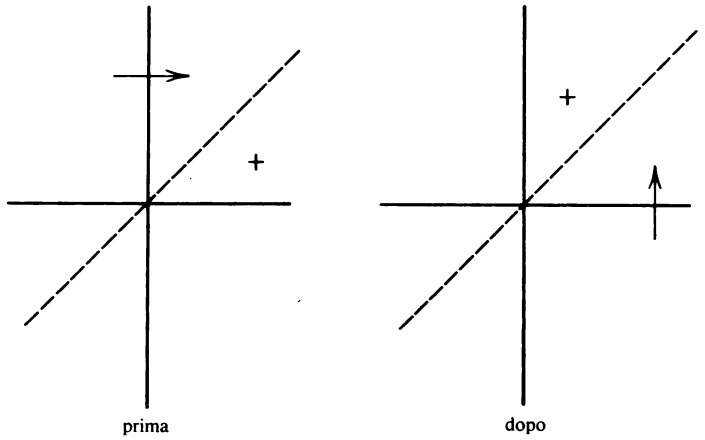


Figura 4.14 Simmetria rispetto alla retta $y = x$

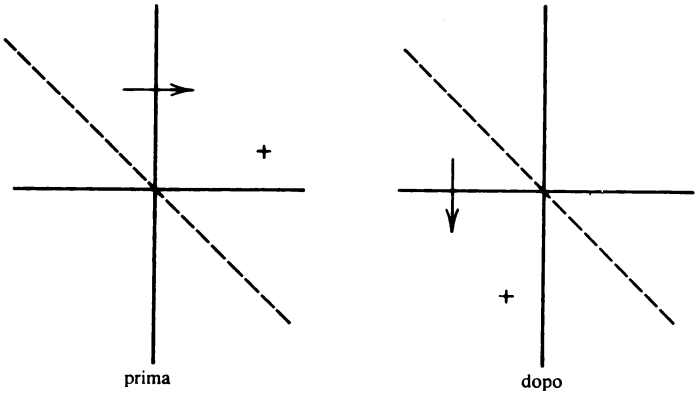


Figura 4.15 Simmetria rispetto alla retta $y = -x$

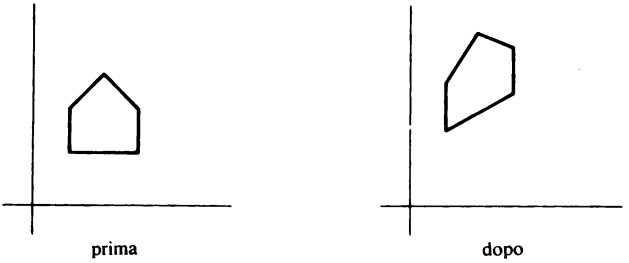


Figura 4.16 Deformazione di taglio lungo l'asse y

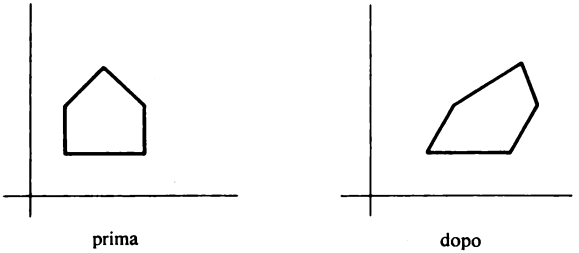


Figura 4.17 Deformazione di taglio lungo l'asse x

$\begin{vmatrix} 1 & 0 \\ b & 1 \end{vmatrix}$ deformazione di taglio (*shear*) lungo l'asse x (vedi Figura 4.17)

Le prime tre simmetrie sono state calcolate utilizzando fattori di scala negativi. Le simmetrie rispetto alle rette $y = x$ e $y = -x$ si possono ottenere mediante una riduzione di scala seguita da una traslazione. È possibile eseguire trasformazioni di *shear* per mezzo di sequenze di rotazioni e cambiamenti di scala, ma è molto più semplice costruire la matrice direttamente. È anche possibile costruire rotazioni e trasformazioni di scala per mezzo di trasformazioni di *shear*.

4.9 ALGORITMI DI TRASFORMAZIONE

Dopo aver analizzato le trasformazioni dal punto di vista algebrico, vengono esaminati alcuni algoritmi per la loro implementazione. Le routine che realizzano le trasformazioni convertono la matrice di trasformazione in coordinate omogenee. Questa matrice può essere poi applicata ad un punto per ottenere il corrispondente punto trasformato. Benché si usi la nozione di coordinata omogenea, in realtà la terza coordinata w non viene mai sfruttata e le trasformazioni saranno tali da avere w sempre uguale ad uno.
Per questa ragione l'ultima colonna della matrice di trasformazione non verrà mai utilizzata. Essa conterrebbe sempre i valori 0, 0, 1 e si può quindi evitare di conservarli in memoria. La matrice di trasformazione 3×3 in coordinate omogenee sarà memorizzata nella matrice 3×2 chiamata H. Una routine la inizializza con i valori della matrice identità.

Algoritmo 4.1 IDENTITY-MATRIX
Crea la matrice identità.
Variabili globali H←matrice di trasformazione di 3×2 elementi
Variabili locali I, J←indici di scansione della matrice H

```
begin
   for I = 1 to 3
      for J = 1 to 2
         if I = J
            then H[I, J]←1
            else H[I, J]←0;
   return;
end;
```

L'algoritmo successivo genera una trasformazione di scala, moltiplicando la matrice H per la matrice di trasformazione di scala

$$\begin{vmatrix} SX & 0 & 0 \\ 0 & SY & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Si noti che, comparendo molti zeri in questa matrice, quando si esegue la moltiplicazione molti termini non danno alcun contributo. L'algoritmo fornito permette di evitare questi calcoli inutili considerando solo i termini diversi da zero.

Algoritmo 4.2 MULTIPLY-IN-SCALE (SX, SY)
Moltiplica la matrice di trasformazione per la matrice di scala.
Argomenti SX←fattore di scala sull'asse *x*
SY←fattore di scala sull'asse *y*
Variabili globali H←matrice di trasformazione 3×2
Variabili locali I←indice di riga della matrice

```
begin
   for I = 1 to 3
      begin
            H[I, 1]←H[I, 1]*SX;
            H[I, 2]←H[I, 2]*SY;
      end;
   return;
end;
```

Per traslare delle quantità TX, TY si moltiplica H per la matrice di traslazione

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ TX & TY & 1 \end{vmatrix}$$

Anche qui si semplifica la moltiplicazione escludendo i termini nulli. Poiché la terza colonna della matrice di trasformazione è sempre [0, 0, 1], l'algoritmo per la traslazione è il seguente.

Algoritmo 4.3 MULTIPLY-IN-TRANSLATION (TX, TY)
Moltiplica la matrice di trasformazione per una matrice di traslazione.
Argomenti TX←traslazione nella direzione *x*
TY←traslazione nella direzione *y*
Variabili globali H←matrice di trasformazione 3×2

```
begin
   H[3, 1]←H[3, 1]+TX;
   H[3, 2]←H[3, 2]+TY;
   return;
end;
```

Per ruotare in senso antiorario di un angolo A espresso in radianti si moltiplica per la matrice

$$\begin{vmatrix} \cos A & \sin A & 0 \\ -\sin A & \cos A & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

L'algoritmo ha come argomento l'angolo, ne calcola il seno e il coseno e quindi esegue il prodotto di matrici evitando come al solito i termini nulli.

Algoritmo 4.4 MULTIPLY-IN-ROTATION (A)
Moltiplica la matrice di trasformazione per una rotazione.
Argomenti A←angolo di rotazione in senso antiorario
Variabili globali H←matrice di trasformazione 3×2
Variabili locali C, S←valori del seno e del coseno
I←indice della matrice
TEMP←locazione temporanea per la prima colonna

```
begin
   C←COS(A);
   S←SIN(A);
   for I = 1 to 3
      begin
         TEMP←H[I, 1]*C−H[I, 2]*S;
         H[I, 2]←H[I, 1]*S+H[I, 2]*C;
         H[I, 1]←TEMP;
      end;
   return;
end;
```

Le routine viste finora servono per creare una matrice di trasformazione, ma occorre ora una routine per applicare al punto la trasformazione risultante. La routine che segue trasforma un singolo punto: gli argomenti della subroutine sono le sue coordinate e le stesse variabili sono utilizzate per restituire il punto trasformato.

Algoritmo 4.5 DO-TRANSFORMATION (X, Y)
Trasforma un punto.
Argomenti X, Y←coordinate del punto da trasformare
Variabili globali H←matrice di trasformazione 3×2
Variabili locali TEMP←locazione temporanea per il nuovo valore della variabile X

```
begin
   TEMP←X*H[1, 1]+Y*H[2, 1]+H[3, 1];
   Y←X*H[1, 2]+Y*H[2, 2]+H[3, 2];
   X←TEMP;
   return;
end;
```

Per eseguire una trasformazione bisogna formare l'opportuna matrice di trasformazione e quindi applicarla a tutti i punti da visualizzare. Ci sono diversi modi per eseguire quest'operazione. Verrà seguito qui l'approccio utilizzato nel sistema grafico CORE, che non permette all'utente di usare direttamente le routine

MULTIPLY-IN-SCALE, MULTIPLY-IN-TRANSLATION e MULTIPLY-IN-ROTATION. L'utente non sarà in grado di realizzare matrici di trasformazione particolarmente complicate, moltiplicando diverse trasformazioni di scala, rotazioni e traslazioni (se vuol farlo dovrà programmarsi da solo le opportune routine). All'utente sarà consentito applicare, nell'ordine da lui scelto, una sola variazione di scala, una sola rotazione e una sola traslazione per volta.
La trasformazione verrà applicata una volta che il display file è stato interpretato. L'utente può modificare i valori di queste trasformazioni in ogni momento, ma l'immagine verrà disegnata usando i valori delle trasformazioni presenti al momento in cui viene interpretato il display file. I seguenti tre algoritmi descrivono le routine utente che servono a salvare i parametri delle trasformazioni in attesa che venga il momento di interpretare il display file.

Algoritmo 4.6 TRANSLATE(TX, TY)
Routine utente che assegna i parametri di una traslazione.
Argomenti TX, TY←dimensioni della traslazione
Variabili globali TRNX, TRNY←locazioni per i parametri della traslazione

```
begin
   TRNX←TX;
   TRNY←TY;
   return;
end;
```

Algoritmo 4.7 SCALE (SX, SY)
Routine utente che assegna i fattori di scala.
Argomenti SX, SY←fattori di scala
Variabili globali SCLX, SCLY←locazioni per i parametri di scala

```
begin
   SCLX←SX;
   SCLY←SY;
   return;
end;
```

Algoritmo 4.8 ROTATE (A)
Routine utente che assegna l'angolo di una rotazione.
Argomenti A←angolo di rotazione
Variabili globali ANGL←locazione per il parametro della rotazione

```
begin
   ANGL←A;
   return;
end;
```

A questo punto occorre una routine che utilizzi i parametri delle trasformazioni

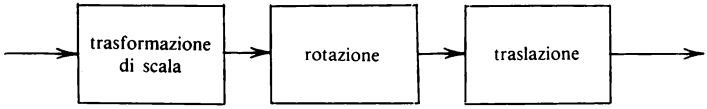


Figura 4.18 Trasformazione dell'immagine

che l'utente ha assegnato, per costruire la matrice di trasformazione vera e propria eseguendo il prodotto di matrici definito precedentemente (vedi Figura 4.18).

Algoritmo 4.9 BUILD-TRANSFORMATION
Costruisce la matrice di trasformazione di una figura.
Variabili globali ANGL, SCLX, SCLY, TRNX, TRNY←parametri delle trasformazioni

```
begin
   IDENTITY-MATRIX;
   MULTIPLY-IN-SCALE(SCLX, SCLY);
   MULTIPLY-IN-ROTATION(ANGL);
   MULTIPLY-IN-TRANSLATION(TRNX, TRNY);
   return;
end;
```

Per applicare queste trasformazioni ad una figura, vengono modificati tre algoritmi dei capitoli precedenti. In particolare verranno estesi gli algoritmi MAKE-PICTURE-CURRENT per introdurvi una chiamata alla BUILD-TRANSFORMATION. In questo modo i parametri assegnati con la routine MAKE-PICTURE-CURRENT divengono i valori per la trasformazione da applicare.

Algoritmo 4.10 MAKE-PICTURE-CURRENT (Algoritmo 2.11 aggiornato)
Routine utente che visualizza il display file corrente.
Variabili globali FREE←indice della successiva cella del display file libera
ERASE-FLAG←indica se lo schermo deve essere ripulito prima della visualizzazione

```
begin
   if ERASE-FLAG
   then begin
        ERASE;
        ERASE-FLAG←FALSE;
        end;
   BUILD-TRANSFORMATION;
```

```
    if FREE>1 then INTERPRET(1, FREE-1);
    DISPLAY;
    FREE←1;
    return;
end;
```

La routine GET-TRANSFORMED-POINT riceve un'istruzione dal display file e gli applica la trasformazione.

Algoritmo 4.11 GET-TRANSFORMED-POINT (NTH, OP, X, Y)
Reperisce e trasforma l'istruzione NTH del display file.
Argomenti NTH←indice dell'istruzione considerata
OP, X, Y←istruzione che viene restituita trasformata

```
begin
    GET-POINT(NTH, OP, X, Y);
    if OP>0 then DO-TRANSFORMATION(X, Y);
    return;
end;
```

Ciascun punto che compare sullo schermo deve essere moltiplicato per la matrice trasformazione ed il risultato del prodotto è quello che deve essere visualizzato. Questo può essere fatto modificando le routine INTERPRET e la LOAD-POLYGON in modo che chiamino la GET-TRANSFORMED-POINT invece della GET-POINT (vedi Figura 4.19).

Algoritmo 4.12 INTERPRET (START, COUNT)
(Algoritmo 3.6 aggiornato)
Scandisce il display file eseguendone le istruzioni.
Argomenti START←indice da cui iniziare la scansione del display file
COUNT←numero di istruzioni che devono essere interpretate.
Variabili locali NTH←indice del display file
OP, X, Y←istruzione del display file

```
begin
    ciclo per eseguire tutte le istruzioni richieste
    for NTH = START to START+COUNT-1
    do begin
        GET-TRANSFORMED-POINT(NTH, OP, X, Y);
        if OP<0 then DOSTYLE (OP)
        else if OP = 1
            then DOMOVE(X, Y)
            else if OP = 2
                then DOLINE(X, Y)
                else if OP<32
```

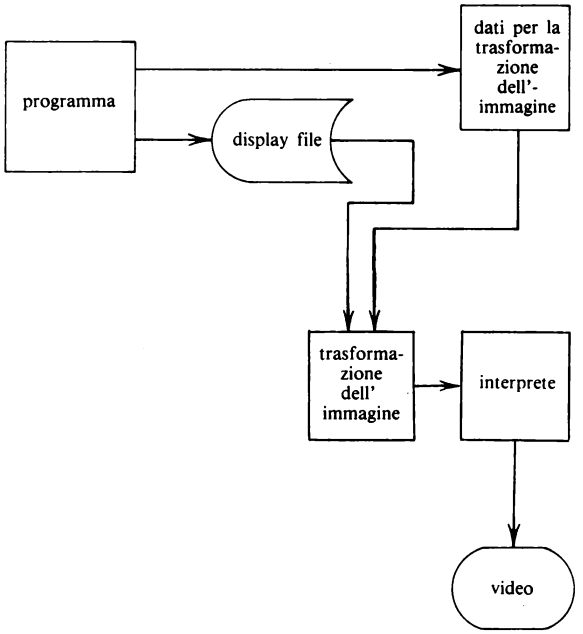


Figura 4.19 Generazione di una figura tramite trasformazione dell'immagine

```
                    then DOPOLYGON(OP, X, Y, NTH)
                    else if OP>31
                         then DOCHAR(OP, X, Y)
                         else return error
                         'ERRORE DI OP-CODE';
    end;
  return;
end;
```

Algoritmo 4.13 LOAD-POLYGON (OP, X, Y, I, EDGES)
(Algoritmo 3.9 aggiornato)
Preleva le informazioni sui lati del poligono a partire dalle posizioni del display file convertite in coordinate schermo.

Argomenti OP, X, Y←istruzione per il poligono
I←indice dell'istruzione nel display file
EDGES←restituisce il numero dei lati memorizzati

Variabili globali WIDTH-START, HEIGHT-START←indici di partenza dello schermo
WIDTH←ampiezza dello schermo in pixel

```
                    HEIGHT←altezza dello schermo in pixel
Variabili locali    X1, Y1, X2, Y2←estremi di un segmento in coordinate
                      schermo
                    I1←indice per scandire il display file
                    K←indice per scandire i lati del poligono
                    DUMMY←argomento fittizio
Costanti            ROUNDOFF←numero piccolo maggiore del più grande
                      errore di arrotondamento
begin
   assegna il punto di partenza per il lato
   X1←X*WIDTH+WIDTH-START;
   corregge le coordinate Y in modo che giaccia fra due linee di scansione
   Y1←INT(Y*HEIGHT+HEIGHT-START)+0.5;
   preleva l'indice del comando per il primo lato
   I1←I+1;
   inizializza un indice per memorizzare i dati sui lati
   EDGES←1;
   ciclo per ricercare le informazioni di ciascun lato
   for K = 1 to OP
         do begin
               preleva il vertice successivo
               GET-TRANSFORMED-POINT(I1, DUMMY, X2, Y2);
               X2←Y2*WIDTH+WIDTH-START;
               Y2←INT(Y2*HEIGHT+HEIGHT-START)+0.5;
               controlla se la retta è orizzontale
               if|Y1−Y2|<ROUNDOFF then
                  begin
                     X1←X2;
                     I1←I1+1;
                  end
               else begin
                     memorizza i dati sui lati in ordine decrescente per le
                     y POLY-INSERT(EDGES, X1, Y1, X2, Y2);
                     incrementa l'indice in cui memorizzare i dati sui lati
                     EDGES←EDGES+1;
                     riassegna il punto precedente
                     Y1←Y2;
                     X1←X2;
                     I1←I1+1;
                     end;
               end;
   assegna a EDGES il numero dei lati memorizzati
   EDGES←EDGES−1;
   return;
end;
```

Sarebbe utile inoltre avere una routine che inizializzi i parametri, in modo tale che l'utente non debba assegnarli a meno che non voglia farlo esplicitamente. I parametri di default daranno la trasformazione identità.

Algoritmo 4.14 IDENTITY-PARAMETERS
Assegna i parametri della trasformazione ai valori della matrice identità.
Variabili globali SCLX, SCLY, ANGL, TRNX, TRNY←parametri delle trasformazioni

```
begin
   SCLX←1;
   SCLY←1;
   ANGL←0;
   TRNX←0;
   TRNY←0;
   return;
end;
```

Di conseguenza dovrà essere scritta una nuova routine globale di inizializzazione.

Algoritmo 4.15 INITIALIZE-4

```
begin
   INITIALIZE-3;
   IDENTITY-PARAMETERS;
   return;
end;
```

4.10 PROCEDURE DI DISPLAY

Può accadere a volte che la trasformazione di visualizzazione che abbiamo presentato sia insufficiente; questo accade nel caso che l'utente desideri applicare più di una sola trasformazione di rotazione, traslazione o cambiamento del fattore di scala.
Ad esempio, una rotazione attorno ad un punto qualsiasi è facilmente realizzabile con una traslazione seguita da una rotazione attorno all'origine e da un'altra traslazione. In questi casi l'utente dovrà aggiungere più livelli di trasformazioni al sistema. Egli dovrà cioè scrivere le routine LINE e MOVE personalizzate che moltiplichino i loro argomenti per l'opportuna matrice di trasformazione prima di chiamare le routine di sistema LINE-ABS-2 e MOVE-ABS-2 per tracciare effettivamente il disegno. Una matrice di trasformazione progettata e creata dall'utente sarà più o meno complessa a seconda delle particolari esigenze, includendo diverse trasformazioni elementari. Le routine progettate opportunamente per questi scopi trasformeranno i valori delle coordinate dei punti prima di inserirli nel display file. Le estensioni del sistema grafico dettagliate

negli algoritmi precedenti, d'altra parte, operano sui punti al momento in cui essi vengono letti dal display file. Più trasformazioni possono quindi essere applicate nei diversi stadi del processo di visualizzazione.
Una situazione in cui è utile disporre di una serie multipla di trasformazioni è il caso che una figura sia costituita da poche componenti primitive combinate in una struttura gerarchica. Ad esempio, si abbia una routine che disegna il petalo di un fiore. Combinando petali con differenti posizioni e orientamenti è possibile disegnare dei fiori completi. Combinando fiori di diverse dimensioni e in differenti posizioni, potremo comporre dei mazzi di fiori; applicando altre trasformazioni ai mazzi possiamo formare un giardino, e così via (vedi Figura 4.20).
Figure che abbiano una struttura gerarchica di questo tipo si prestano ad essere realizzate attraverso sottoprogrammi. Un programma che disegni un giardino sarà costituito da chiamate alla routine che disegna un mazzo di fiori. Il sottoprogramma che disegna il mazzo lo farà per successive chiamate al sottoprogramma che disegna un fiore e infine, analogamente agli altri, il sottoprogramma del fiore chiamerà più volte la routine che produce un petalo. Osserviamo che, comunque, sono un po' più complicati dei soliti programmi scritti con i normali linguaggi di programmazione: essi infatti richiedono di costruire una matrice di trasformazione. Una semplice chiamata ad un sottoprogramma (**call** PETALO) produce sempre la stessa immagine, della stessa dimensione e con il medesimo orientamento. Ciò che serve è un tipo di chiamata al sottoprogramma che assegni i valori ad una trasformazione e che questa venga applicata a tutti i punti generati nel sottoprogramma prima che essi vengano introdotti nel display file; ad esempio

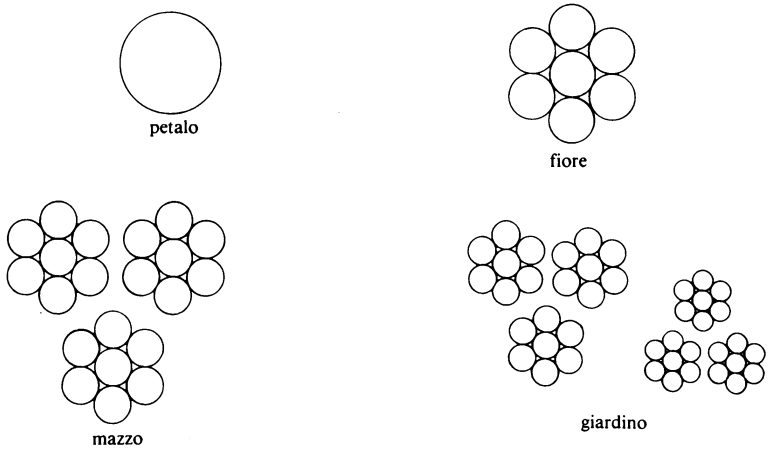


Figura 4.20 Struttura gerarchica di figure

call PETALO con SIZE(SX, SY), ANGLE(A), TRANSLATION(TX, TY)

Questo tipo di chiamate a sottoprogrammi, che richiedono di fissare una trasformazione, vengono dette *chiamate a procedure di display* (o di visualizzazione) e i sottoprogrammi che disegnano le figure componenti si dicono *procedure di display*. A causa del fatto che le procedure di display possono essere annidate, possono esserci di conseguenza più trasformazioni da applicare. Supponiamo ad esempio che il programma GIARDINO chiami la procedura di display MAZZO applicandogli la trasformazione T_1. MAZZO chiama la procedura FIORE applicandogli la trasformazione T_2. Infine FIORE chiama la procedura di display PETALO con la trasformazione T_3. La trasformazione complessiva che viene applicata ai punti generati nella procedura PETALO sarà allora data dal prodotto $T_3T_2T_1$. Si osservi che ogni volta che viene eseguita una nuova chiamata ad una procedura di display, la matrice di trasformazione complessiva viene moltiplicata alla sinistra per la trasformazione di quella procedura di display.

Nella procedura GIARDINO viene applicata la matrice identità I. Al momento della chiamata alla MAZZO, questa viene moltiplicata per T_1 ottenendo ancora T_1 come risultato. All'interno di essa viene chiamata la FIORE e si moltiplica la trasformazione T_2 ottenendo T_2T_1. Infine la chiamata alla PETALO combina anche la terza trasformazione ottenendo $T_3T_2T_1$. Cosa accade quando termina la procedura PETALO e il controllo del programma torna alla FIORE che l'ha chiamata? Ci si aspetta di avere la trasformazione complessiva appropriata per FIORE e cioè T_2T_1. Analogamente, quando il controllo viene ritornato dalla routine FIORE alla MAZZO, la trasformazione sarà T_1 e quando il controllo, infine, viene restituito al programma GIARDINO, la matrice di trasformazione complessiva deve essere ancora la matrice identità I. Per questo deve esserci un modo di salvare la matrice complessiva prima di moltiplicarla per la trasformazione aggiuntiva da applicare nella procedura di display chiamata, in modo tale che possa essere recuperata quando il controllo ritorna alla routine chiamante. Nel momento in cui la FIORE chiama la PETALO, viene salvata la trasformazione complessiva corrente T_2T_1, che viene poi moltiplicata per T_3 per ottenere la nuova trasformazione complessiva $T_3T_2T_1$ e viene poi trasferito il controllo alla PETALO. Quando la PETALO ha concluso le sue operazioni, prima di restituire il controllo alla procedura chiamante, ripristina il valore della trasformazione globale al valore che era stato precedentemente salvato, e cioè T_2T_1, dopodiché il controllo del programma può essere passato di nuovo alla FIORE. Ciascuna chiamata ad una procedura di display deve salvare la matrice di trasformazione corrente. A causa del fatto che vi possono essere annidamenti di procedure, può capitare di avere memorizzate diverse matrici di trasformazione simultaneamente. Una struttura di dati adatta a memorizzare queste matrici nell'ordine opportuno è uno stack. L'ultimo oggetto memorizzato nello stack è il primo che può essere estratto; questo corrisponde esattamente al comportamento delle subroutine nelle quali l'ultima subroutine entrata è la prima ad uscire.

In sintesi la chiamata ad una procedura di display richiede i seguenti passi:

1. Salvare la matrice di trasformazione complessiva
2. Moltiplicare la matrice di trasformazione sulla sinistra della trasformazione passata al momento della chiamata per formare la nuova matrice di trasformazione complessiva
3. Trasferire il controllo alla procedura di display

Il ritorno da una procedura di display alla procedura chiamante richiede invece le seguenti operazioni:

1. Ripristino della matrice di trasformazione complessiva al valore salvato nello stack
2. Ritorno del controllo al programma chiamante

I comandi LINE e MOVE dell'utente all'interno del corpo di una routine di display dovranno comportarsi nel seguente modo:

1. Moltiplicare le coordinate dei punti per la matrice complessiva corrente per ottenere i punti opportunamente trasformati
2. Inserire i valori trasformati nel display file utilizzando i comandi del sistema grafico LINE-ABS-2 e MOVE-ABS-2.

APPLICAZIONE PRATICA

Si consideri a questo punto come il sistema grafico possa essere utilizzato per produrre sequenze di animazione. L'animazione viene eseguita fotografando una sequenza di disegni, ciascuno dei quali differisce di poco dal precedente. Ad esempio, volendo mostrare una persona che muove un braccio, bisognerà fotografare una serie di disegni, ciascuno dei quali mostra il braccio in una posizione differente. Quando poi le figure vengono visualizzate una dopo l'altra attraverso un proiettore, viene percepito il movimento continuo del braccio (vedi Figura 4.21).
Per ottenere un risultato valido, è necessario avere solo un piccolo spostamento per volta. L'immagine complessiva può essere costituita da uno sfondo — che non viene modificato — e da un primo piano che cambia fotogramma per fotogramma. Il primo piano può venire disegnato su di un foglio di plastica tra-

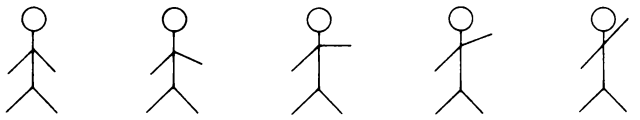


Figura 4.21 Animazione del movimento di un braccio tramite una serie di figure

sparente che viene sovrapposto allo sfondo. Se la parte dell'immagine in movimento è piccola, la figura di primo piano, che deve essere ridisegnata ad ogni fotogramma, è piccola e richiede quindi meno lavoro: quello che risulta difficile al disegnatore è invece modificare l'intera scena. Ad esempio, se si vuole dare l'impressione di movimento all'interno della scena, bisogna cambiare il fattore di scala dello sfondo incrementandolo ad ogni passo. Questo tipo di cambiamenti, difficile per l'operatore umano, risulta invece particolarmente facile per il calcolatore. Sarà quindi di grande aiuto avere un calcolatore a disposizione per generare la scena di sfondo, mentre l'artista sovrappone l'azione che si svolge in primo piano. Per utilizzare il sistema grafico per generare gli sfondi, innanzitutto si dovrà costruire l'intero disegno dello sfondo usando i comandi LINE e POLYGON descritti nei precedenti capitoli.
Assumendo che questo sia già stato fatto e avendo quindi una routine che inserisce le opportune istruzioni nel display file, se si eseguono ripetute volte questi comandi, verrà generato lo sfondo di ciascun fotogramma. Si esamini per esempio una scena che si svolge in una strada di città: per mostrare un personaggio che cammina attraverso la città, occorre mantenere il personaggio fermo e muovere lo scenario sotto di esso. Ciò può essere fatto semplicemente spostando l'immagine di sfondo con il comando TRANSLATE: prima di visualizzare ogni figura bisogna aggiungere un piccolo fattore di traslazione al valore di TX. Ciascun fotogramma mostrerà lo sfondo traslato un po' più a destra, il che darà l'impressione che il personaggio si sia spostato verso sinistra (vedi Figura 4.22).

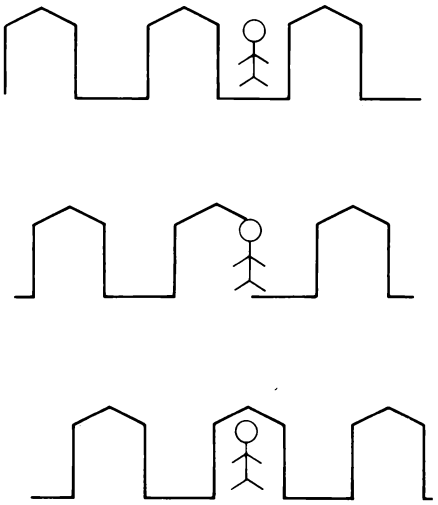


Figura 4.22 Movimento apparente creato dalla traslazione dello sfondo

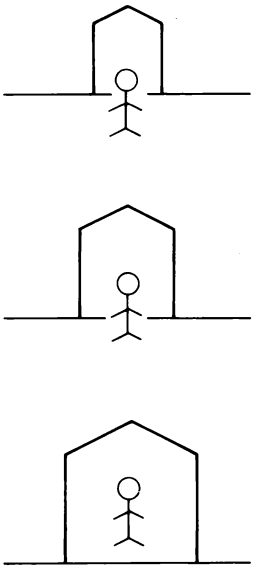


Figura 4.23 Movimento apparente creato modificando il fattore di scala dello sfondo

Volendo avvicinare il personaggio ad un edificio, dovremo ingrandire la figura che rappresenta l'edificio, usando la routine SCALE(SX, SY) per ingrandire l'edificio e la routine TRANSLATE(TX, TY) per centrarlo correttamente. In ogni fotogramma, SX e SY vengono incrementati in modo da fare apparire l'edificio un po' più ravvicinato (vedi Figura 4.23).

ESERCIZI

4.1 Eseguire le seguenti moltiplicazioni di matrici:

a) $\begin{vmatrix} 1 & 0 \\ 2 & 3 \end{vmatrix} \quad \begin{vmatrix} 2 & 1 \\ 1 & 1 \end{vmatrix}$

b) $\begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} \quad \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}$

c) $\begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix} \quad \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}$

d) $|2 \quad 4| \quad \begin{vmatrix} 3 & 5 \\ 2 & 0 \end{vmatrix}$

e) $\begin{vmatrix} 3 & 5 \\ 2 & 0 \end{vmatrix} \quad \begin{vmatrix} 2 \\ 4 \end{vmatrix}$

f) $|3 \quad 4 \quad 1| \qquad \begin{vmatrix} 1 & 2 & 0 \\ -2 & 3 & 0 \\ 0 & 0 & 1 \end{vmatrix}$

g) $|0 \quad 0 \quad 1| \qquad \begin{vmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 3 & 5 & 1 \end{vmatrix}$

h) $|1 \quad 1 \quad 1| \qquad \begin{vmatrix} 3 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 1 \end{vmatrix}$

i) $|1 \quad 0 \quad 1| \qquad \begin{vmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{vmatrix}$

j) $\begin{vmatrix} 2 & 1 & 0 \\ 3 & 1 & 0 \\ 1 & 0 & 1 \end{vmatrix} \qquad \begin{vmatrix} 1 & 4 & 0 \\ 1 & 5 & 0 \\ 0 & 2 & 1 \end{vmatrix}$

4.2 Scrivere una matrice di trasformazione 2×2 per ciascuna delle seguenti trasformazioni di scala:

a) La figura ingrandita complessivamente del triplo.
b) La figura rimpicciolita complessivamente di un terzo.
c) La direzione x ingrandita di quattro volte e la y immutata.
d) La direzione y ridotta di due terzi del valore originario e la x immutata.
e) La direzione x ridotta di tre quarti rispetto al valore originario e la direzione y aumentata di un fattore pari a sette quinti.

4.3 Scrivere la matrice di trasformazione in coordinate omogenee 3×3 per ciascuna delle trasformazioni date nell'esercizio 4.2.

4.4 Scrivere la matrice di trasformazione 2×2 per ciascuna delle seguenti rotazioni attorno all'origine:

a) Antioraria di π
b) Antioraria di $\pi/2$
c) Oraria di $\pi/2$
d) Antioraria di $\pi/4$
e) Antioraria di $5\pi/4$

4.5 Scrivere la matrice di trasformazione 3×3 in coordinate omogenee per ciascuna delle rotazioni date nell'esercizio 4.4.

4.6 Scrivere la matrice di trasformazione 3×3 in coordinate omogenee per ciascuna delle seguenti traslazioni:

a) Traslare l'immagine verso destra di 3 unità.
b) Traslare l'immagine verso l'alto di 2 unità.
c) Muovere l'immagine verso il basso di 1/2 unità e a destra di 1 unità.
d) Muovere l'immagine verso il basso di 2/3 di unità e a sinistra di 4 unità.

4.7 Scrivere la matrice di trasformazione in coordinate omogenee 3×3 che ha lo stesso effetto complessivo delle seguenti sequenze di trasformazioni:

a) Cambiare la scala dell'immagine in modo che sia larga il doppio e quindi traslare di 1 unità verso sinistra.
b) Cambiare la scala nella direzione x in modo che sia larga la metà e quindi ruotare in senso antiorario di $\pi/2$ attorno all'origine.
c) Ruotare di $\pi/2$ in senso antiorario attorno all'origine e quindi cambiare la scala della direzione x in modo che sia la metà.
d) Traslare verso il basso di 1/2 unità, verso destra di 1/2 unità e quindi ruotare in senso antiorario di $\pi/4$.
e) Cambiare la scala per le coordinate y per rendere l'immagine alta il doppio, traslarla verso il basso di 1 unità e quindi ruotarla in senso orario di $\pi/6$.

4.8 Dimostrare come sia possibile realizzare le simmetrie rispetto alla retta $y = x$ e rispetto alla retta $y = -x$ per mezzo di una trasformazione di scala, seguita da una rotazione. (È possibile utilizzare fattori di scala con valori negativi).

4.9 Una generica figura sia centrata rispetto al punto (0.5, 0.5):

a) Scrivere le chiamate alla SCALE, ROTATE e TRANSLATE che mostrino la figura raddoppiata di scala ma ancora centrata nello schermo.
b) Scrivere le chiamate alla SCALE, ROTATE e TRANSLATE che mostrino la figura cambiata di scala di un generico fattore S ma ancora centrata nello schermo.
c) Scrivere le chiamate alla SCALE, ROTATE e TRANSLATE che ruotino la figura in senso antiorario di un generico angolo A ma ancora centrata sullo schermo.
d) Scrivere le chiamate alla SCALE, ROTATE e TRANSLATE che mostrino la figura simmetrica, rispetto alla retta di equazione $x = 0.5$, di quella originale.
e) Scrivere le chiamate alla SCALE, ROTATE e TRANSLATE che mostrino la figura simmetrica rispetto al punto (0.5, 0.5); la trasformazione complessiva è $x_1 = 1-x$ e $y_1 = 1-y$.

PROBLEMI DI PROGRAMMAZIONE

4.1 Implementare gli algoritmi dal 4.1 al 4.15 che estendono il sistema grafico per includere le trasformazioni delle immagini.

4.2 Scrivere un programma che disegni un poligono di cinque lati a forma di casa. Mostrare questa figura modificata in scala, ruotata, traslata e trasformata da una combinazione di tutti questi tipi di trasformazioni. Mostrare la figura sia con l'area riempita che vuota.

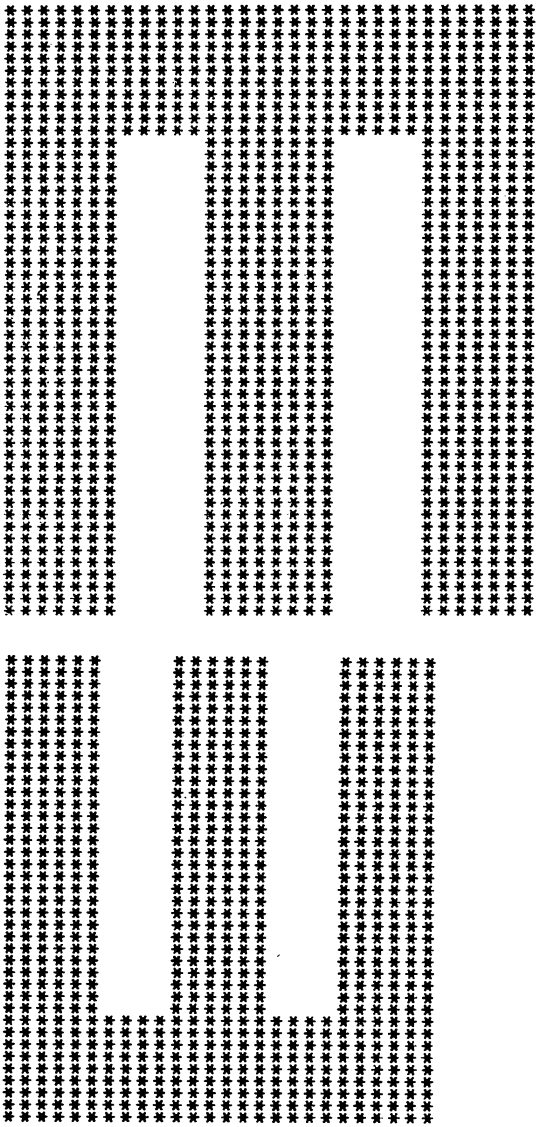
4.3 Scrivere tre routine

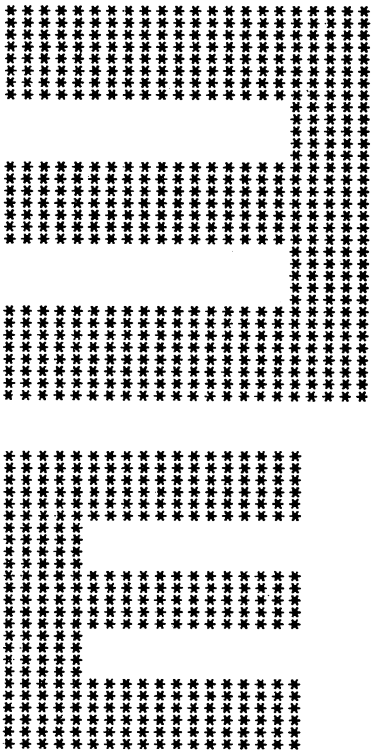
INQUIRE-SCALE(SX, SY)
INQUIRE-ROTATE(A)
INQUIRE-TRANSLATE(TX, TY)

che restituiscono i parametri della trasformazione corrente dell'immagine.

4.4 Programma per creare un tabellone per il controllo della vista: usando un poligono riempito per costruire la lettera E, scrivere un programma per mostrare questa lettera

con differenti dimensioni e orientamenti. L'orientamento dovrà essere uno dei seguenti quattro: E, ∃, ш, m e deve essere scelto a caso per ciascuna immagine. La dimensione delle lettere dovrà essere grande all'inizio ma diventare sempre più piccola. Le lettere devono essere sempre centrate sullo schermo.

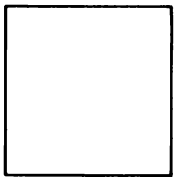


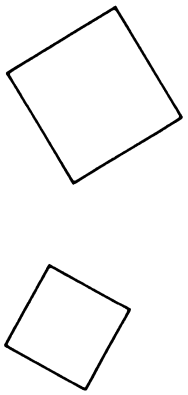


4.5 Scrivere una procedura PIE(A1, A2, R) che disegni un segmento circolare per mezzo di trasformazioni sull'immagine prodotta come risultato dalla procedura WEDGE del problema di programmazione 3.9. Il segmento di circonferenza dovrà avere il vertice centrato nel punto (0.5, 0.5) e raggio R e giacere tra gli angoli A_1 e A_2.

4.6 Supponiamo che la procedura FOO abbia dei comandi LINE, MOVE e POLYGON per costruire una generica immagine. Scrivere un programma per ripetere i seguenti passi:

1) Chiamare la procedura FOO per creare l'immagine.
2) Creare una nuova trasformazione di immagine che mostri la figura ruotata di $\pi/5$ in senso antiorario e ridotta di scala di un fattore pari a 0.8 rispetto alla prima immagine visualizzata. L'immagine deve rimanere centrata nello schermo.
3) Produrre la seguente immagine risultante:





Collaudare questo programma per 5 iterazioni con FOO che disegna il quadrato con vertici in (0, 0), (0, 1), (1, 1), (1, 0).

**4.7 Implementare il meccanismo di chiamata delle procedure di visualizzazione.

5

Segmentazione del display file

INTRODUZIONE

Nei capitoli precedenti si sono considerate rappresentazioni grafiche costituite da singole figure. In realtà, l'immagine sullo schermo di un terminale è spesso composta da diverse figure o tipi di informazioni grafiche. Per esempio, una singola immagine potrebbe contenere diverse viste di uno stesso oggetto, mostrare una panoramica dell'intero oggetto e contemporaneamente uno scorcio di un particolare componente o infine contenere informazioni riguardanti l'oggetto, istruzioni per l'utente o segnalazioni d'errore.
Si consideri, come esempio, il progetto degli impianti di un edificio in costruzione, che comprenderà i diagrammi relativi alla struttura, all'impianto elettrico, a quello idraulico e a quello per il riscaldamento: per alcune applicazioni potrebbero servire soltanto singoli elementi dell'intero progetto. Come secondo esempio si pensi ad un'astronave che si vuole rappresentare in movimento nel cielo: la si potrebbe mostrare in posizioni differenti, mantenendo fermo lo sfondo oppure la si potrebbe fissare al centro dello schermo muovendo, invece, lo sfondo. L'ultimo capitolo visto, che tratta delle trasformazioni, parla di come cambiare la posizione di un'immagine intera, ma, in questo caso, si desidera trasformare soltanto una porzione della scena (o l'astronave o lo sfondo, non entrambi).
Per riflettere tale struttura a «sotto-figure» occorrerà riorganizzare il display file, suddividendolo in *segmenti* a cui saranno associati univocamente i diversi componenti dell'intera scena. Ad ogni segmento verrà assegnata una serie di *attributi*, tra cui l'attributo di *visibilità*, il quale, a seconda del proprio valore, indicherà se un segmento dovrà essere o meno visualizzato. Variando i valori degli attributi di visibilità associati ai vari segmenti che compongono una scena, l'utente sarà in grado di costruire la scena per mezzo di sotto-figure opportunamente selezionate. Si consideri nuovamente l'esempio del progetto degli impianti di un edificio. Sullo schermo si potranno visualizzare le informazioni relative alla struttura e all'impianto elettrico dell'edificio, rendendo visibili il primo e il

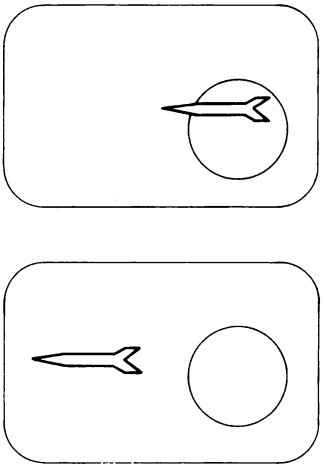


Figura 5.1 Trasformazione di una zona del video

secondo segmento ed invisibili il terzo e il quarto. In un altro momento si potrebbero far comparire sullo schermo solo la struttura e l'impianto idraulico rendendo visibili unicamente il primo ed il terzo segmento. Mentre tutte le informazioni sono presenti nella memoria del calcolatore, l'utente sarà in grado di mostrare a piacere porzioni della scena indicando quale segmento del display file debba essere interpretato.
Un altro attributo associato ad ogni segmento è la trasformazione dell'immagine, che permette l'applicazione indipendente di rotazioni, cambiamenti di scala e traslazioni a ciascun segmento. Nel caso dell'astronave, si potrebbe definire il veicolo in un segmento e lo sfondo in un altro, applicare una trasformazione ad uno dei due e lasciare immutato l'altro (vedi Figura 5.1).
L'argomento centrale di questo capitolo sarà la segmentazione del display file: si parlerà di come creare, chiudere, richiamare e cancellare segmenti, utilizzando nel display file attributi come quello di visibilità e di trasformazione dell'immagine.

5.1 TABELLA DEI SEGMENTI

L'argomento ha inizio con la presentazione di alcune delle informazioni da associare ad ogni segmento e la loro organizzazione. A ciascun segmento va assegnato un nome che lo identifichi univocamente e serva a distinguerlo da tutti gli altri segmenti, ogni qual volta occorra cambiare — ad esempio — l'attributo di visibilità. Infatti, quando si fa riferimento ad un segmento del display file, è

necessario sapere quali istruzioni contenute nel file gli appartengono; ciò è possibile conoscendo l'istruzione iniziale del segmento e il numero delle istruzioni che seguono. Per ciascun segmento occorre un metodo che consenta di metterne in relazione il nome con le informazioni relative alla posizione all'interno del display file, e con quelle relative ai suoi attributi. L'organizzazione di queste informazioni sarà tale che, dato il nome di un segmento, si potranno ottenere o cambiare i valori degli attributi ad esso associati; in altre parole, sarà subito possibile interpretare le istruzioni del display file corrispondenti al segmento. In questo sistema si utilizzerà, a tal fine, una *tabella dei segmenti* che sarà costituita da array, i quali serviranno a mantenere in memoria le proprietà dei segmenti; usando inoltre dei numeri come nomi per i segmenti, questi numeri avranno anche la funzione di indici degli array. Esisterà dunque un array per le locazioni iniziali nel display file, uno per le dimensioni dei segmenti, un terzo indicherà la visibilità e così via. Per conoscere, ad esempio, la dimensione del terzo segmento, si dovrà ricercare il terzo elemento dell'array che contiene appunto le dimensioni di tutti i segmenti (vedi Figura 5.2).
L'indice 1 farà riferimento al primo segmento della tabella, l'indice 2 identificherà il secondo segmento e così via. La tabella risulta così essere formata da diverse aree riservate alle locazioni iniziali nel display file, alle dimensioni dei segmenti, agli attributi come quello di visibilità e di trasformazione dell'immagine. Se si desidera che, per esempio, il segmento numero 3 non sia visibile, si assegnerà l'opportuno valore alla locazione corrispondente a tale segmento,

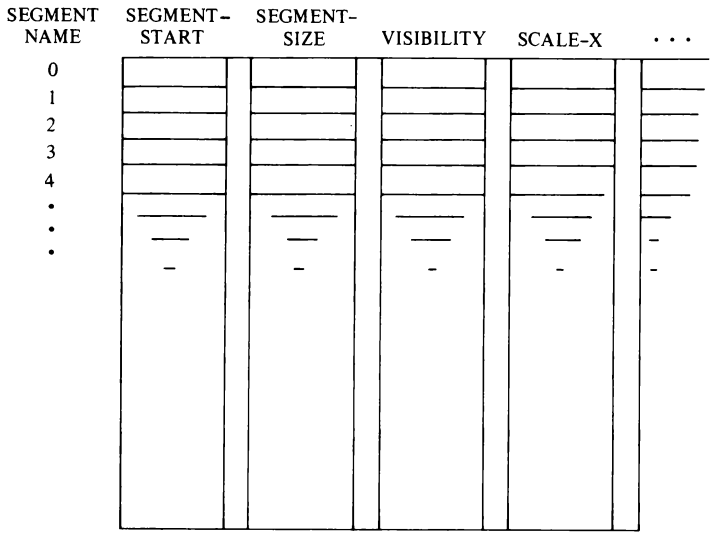


Figura 5.2 Tabella dei segmenti

nell'array che contiene le informazioni sulla visibilità. Volendo far apparire sullo schermo la scena, si consulterà la tabella dei segmenti per determinare quali siano visibili al momento. Per ogni segmento si considereranno la posizione iniziale e la dimensione, e si passeranno tali informazioni all'interprete del nostro display file; in questo modo l'interprete elaborerà solo quei segmenti dei display file che sono visibili.
Esistono altri possibili schemi per l'implementazione della tabella dei segmenti, molti dei quali presentano sostanziali vantaggi rispetto a quello proposto, che è stato scelto proprio perché permette semplici accessi ai dati: non richiede strutture dati concettualmente nuove e complesse e permette aggiornamenti immediati.
A questo punto occorrerebbe fare in modo che le estensioni al sistema grafico che si sta sviluppando fossero compatibili con i programmi già implementati, che non prevedevano la segmentazione dell'immagine. Forse il metodo più semplice per integrare i due procedimenti è quello di porre le istruzioni in un segmento speciale «senza nome», quando appunto non viene specificato alcun nome di segmento; anche in questo caso devono essere memorizzate informazioni come la posizione iniziale e la dimensione del segmento, in un opportuno spazio all'interno della tabella, riservato al segmento senza nome. In questa implementazione si è scelto di associargli le locazioni della tabella corrispondenti all'indice 0: dunque SEGMENT-SIZE[1] sarà ancora il numero di istruzioni del segmento 1, mentre SEGMENT-SIZE[0] corrisponderà al numero di istruzioni del segmento non identificato.

5.2 CREAZIONE DI UN SEGMENTO

In questo paragrafo si parlerà del processo di creazione o apertura di un segmento. Quando si crea un segmento, tutti i comandi che seguiranno, come ad esempio i comandi LINE, MOVE, TEXT o POLYGON, apparterranno a questo segmento cui dovrà essere assegnato un nome; supponiamo, per esempio, di assegnargli il numero 3: tutti i comandi che seguiranno alla creazione del segmento 3 apparterranno ad esso. A questo punto supponiamo di voler chiudere il segmento 3 e di voler aprire il segmento 5; i successivi comandi MOVE e LINE apparterranno al segmento 5.
Quando viene creato un nuovo segmento, per prima cosa si controlla se esistono altri segmenti aperti; infatti non è possibile mantenere aperti due o più segmenti contemporaneamente, in quanto nascerebbe un'ovvia ambiguità: a quale segmento assegnare le istruzioni di disegno? Se, quindi, al momento della creazione di un segmento, ne esiste un altro ancora aperto, si incorre in un errore. In secondo luogo, occorre controllare che al segmento in questione sia stato assegnato un nome corretto, cioè che non esistano altri segmenti con quello stesso nome. In questo caso viene nuovamente segnalato un errore.
Al momento della crezione del nuovo segmento, vengono inizializzate le loca-

zioni nella tabella dei segmenti corrispondenti all'indice assegnato. La prima istruzione appartenente al segmento verrà inserita nella prima area di memoria disponibile all'interno del display file. La dimensione del segmento sarà pari a 0 poiché al momento della creazione non si sono ancora inserite istruzioni nel file. Gli attributi vengono inizializzati ai valori del segmento senza nome, che sono poi i valori assegnati per default. Infine, si segnala che da questo momento in poi esiste un segmento aperto (quello che è stato appena creato).

Algoritmo 5.1 CREATE-SEGMENT (SEGMENT-NAME)
Routine utente che crea un segmento identificato da un nome.

Argomenti SEGMENT-NAME←nome del segmento

Variabili globali NOW-OPEN←segmento corrente
FREE←indice della successiva cella libera nel display file
SEGMENT-START, SEGMENT-SIZE, VISIBILITY, ANGLE, SCALE-X, SCALE-Y, TRANSLATE-X, TRANSLATE-Y←array per la tabella dei segmenti

Costanti NUMBER-OF-SEGMENTS←dimensione della tabella dei segmenti

```
begin
   if NOW-OPEN>0 then return-error 'SEGMENTO ANCORA APERTO';
   if SEGMENT-NAME<1 or SEGMENT-NAME>NUMBER-OF-
      SEGMENTS then return-error 'NOME DEL SEGMENTO NON
      VALIDO';
   if SEGMENT-SIZE[SEGMENT-NAME]>0 then return-error
      'SEGMENTO GIÀ ESISTENTE';
   SEGMENT-START[SEGMENT-NAME]←FREE;
   SEGMENT-SIZE[SEGMENT-NAME]←0;
   VISIBILITY[SEGMENT-NAME]←VISIBILITY[0];
   ANGLE[SEGMENT-NAME]←ANGLE[0];
   SCALE-X[SEGMENT-NAME]←SCALE-X[0];
   SCALE-Y[SEGMENT-NAME]←SCALE-Y[0];
   TRANSLATE-X[SEGMENT-NAME]←TRANSLATE-X[0];
   TRANSLATE-Y[SEGMENT-NAME]←TRANSLATE-Y[0];
   NOW-OPEN←SEGMENT-NAME;
   return;
end;
```

5.3 CHIUSURA DI UN SEGMENTO

Una volta aperto un segmento, si può accedere alle istruzioni contenute nel display file, come si è visto nei capitoli precedenti. In questo caso, comunque, tutti i comandi in lista sono associati al segmento aperto. Dopo aver completato le istruzioni di disegno appartenenti al segmento, esso dovrebbe venire chiu-

so. A questo punto non vi è altro da fare che cambiare il valore della variabile NOW-OPEN, la quale indica il nome del segmento correntemente aperto: gli verrà assegnato il valore 0 cosicché il segmento senza nome diverrà il segmento corrente. Per evitare un inutile ed eccessivo spreco di memoria, occorre anche cancellare ogni istruzione che precedentemente era stata assegnata al segmento non identificato: quest'operazione realizza l'inizializzazione di tale segmento, la quale prevede che esso non contenga alcuna istruzione, ma sia pronto a ricevere nella prima locazione libera del display file tutti i comandi che si presenteranno sino all'apertura del successivo segmento.

Algoritmo 5.2 CLOSE-SEGMENT
Variabili globali NOW-OPEN←nome del segmento corrente
FREE←indice della successiva cella libera nel display file
SEGMENT-START, SEGMENT-SIZE←array contenenti la posizione iniziale e il dimensionamento dei segmenti

```
begin
   if NOW-OPEN = 0 then return-error 'NESSUN SEGMENTO APERTO';
   DELETE-SEGMENT(0);
   SEGMENT-START[0]←FREE;
   SEGMENT-SIZE[0]←0;
   NOW-OPEN←0;
   return;
end;
```

5.4 CANCELLAZIONE DI UN SEGMENTO

L'algoritmo più complesso di questo capitolo è quello per la cancellazione di un segmento. Se un segmento non è più necessario, si vorrebbe riacquistare lo spazio occupato all'interno del display file, per poter usare questa porzione di memoria per qualche altro segmento, dato che le istruzioni ad esso relative non saranno mai più eseguite. D'altra parte, non avrebbe senso distruggere e ricostruire l'intero display file, azione che potrebbe comportare l'alterazione di parti di un'immagine magari esigue, ma molto costose da ricostruire. L'ideale sarebbe cancellare (e forse ricreare con modifiche) solo un segmento alla volta, preservando il resto del display file. Il metodo per eseguire tale operazione dipende dalla struttura dei dati scelta per l'organizzazione del display file stesso. Per questo sistema si sono usati degli array e l'immagazzinamento di un blocco di dati in un array è un'operazione semplice e immediata, anche se non efficiente come altre tecniche di gestione dei dati. Ciò che si avrebbe l'intenzione di fare, è considerare tutte le istruzioni inserite dopo la chiusura del segmento da cancellare per spostarle nello spazio da esso occupato all'interno del display file: in questo modo riusciamo a riempire lo spazio lasciato dal blocco di istru-

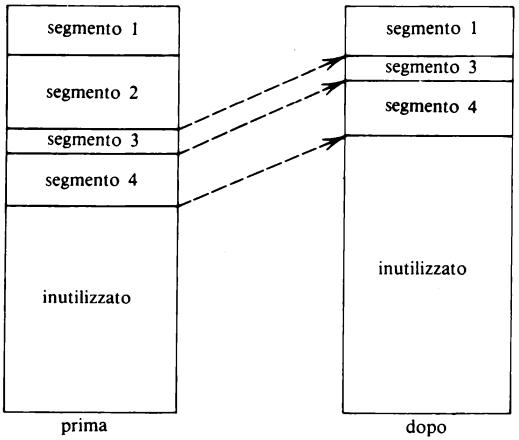


Figura 5.3 Cancellazione del segmento 2 dal display file

zioni cancellate recuperando una porzione di memoria equivalente (vedi Figura 5.3). L'algoritmo di cancellazione di un segmento inizia controllando che il nome che lo identifica sia corretto. In caso affermativo, controlla che il segmento non sia aperto: infatti i segmenti aperti sono ancora utilizzabili; ogni tentativo di cancellazione di un segmento aperto è considerato un errore. A questo punto viene controllata la dimensione del segmento: se questa è uguale a zero, il segmento non possiede istruzioni da eliminare, quindi non è necessaria l'esecuzione di alcuna operazione. Finalmente si possono spostare le istruzioni che si vogliono sostituire al segmento da cancellare: la prima istruzione da rilocare è posta sopra la prima istruzione del segmento da eliminare. Percorrendo il display file, le istruzioni vengono traslate verso il basso finché non si incontra una locazione vuota. Quando non vi sono più istruzioni da spostare, viene aggiornato il valore della variabile che indica la prima istruzione libera nel display file.
Si noti che non bisogna risistemare solo il display file, ma anche la tabella dei segmenti, poiché saranno senz'altro cambiate tutte le posizioni iniziali dei segmenti creati dopo quello che si sta cancellando. A tal fine, si scandisce la tabella considerando tutti quei segmenti, il cui indirizzamento iniziale è posto oltre la posizione di partenza del segmento cancellato; ad essi viene sottratta la dimensione del segmento cancellato che, dopo questa operazione, può essere posta a zero per indicare che il segmento non esiste più. Nel caso venga cancellato un segmento visibile, si renderà necessaria un'azione di NEW-FRAME. Le operazioni descritte sono presentate in dettaglio nel seguente algoritmo.

Algoritmo 5.3 DELETE-SEGMENT (SEGMENT-NAME)
Routine utente che cancella un segmento.
Argomenti SEGMENT-NAME←nome del segmento

Variabili globali NOW-OPEN←segmento corrente
FREE←indice della successiva cella libera nel display file
SEGMENT-START, SEGMENT-SIZE←due degli array che fanno parte della tabella dei segmenti
Costanti NUMBER-OF-SEGMENTS←dimensione della tabella dei segmenti
Variabili locali GET←locazione dell'istruzione da rimuovere
PUT←locazione in cui deve essere posta l'istruzione che si sta considerando
SIZE←dimensione del segmento da cancellare
I←indice della tabella dei segmenti

```
begin
   if SEGMENT-NAME<0 or SEGMENT-NAME>NUMBER-
     OF-SEGMENTS then return-error 'NOME DEL SEGMENTO
     NON VALIDO';
   if SEGMENT-NAME = NOW-OPEN and SEGMENT-NAME≠0 then
      return-error 'SEGMENTO ANCORA APERTO';
   if SEGMENT-SIZE[SEGMENT-NAME] = 0 then return;
   PUT←SEGMENT-START[SEGMENT-NAME];
   SIZE←SEGMENT-SIZE[SEGMENT-NAME];
   GET←PUT+SIZE;
   shifta gli elementi del display file
   while GET<FREE do
      begin
         GET-POINT(GET, OP,X,Y);
         SET-POINT(PUT, OP,X,Y);
         PUT←PUT+1;
         GET←GET+1;
      end;
   ripristina lo spazio cancellato
   FREE←PUT;
   aggiorna la tabella dei segmenti
   for I = 0 to NUMBER-OF-SEGMENTS do
   if SEGMENT-START[I]>SEGMENT-START[SEGMENT-NAME]
   then SEGMENT-START[I]←SEGMENT-START[I]-SIZE;
   SEGMENT-SIZE[SEGMENT-NAME]←0;
   if VISIBILITY[SEGMENT-NAME] then NEW-FRAME;
   return;
end;
```

In questo algoritmo si utilizza una procedura detta SET-POINT per accedere ad un'istruzione del display file in una specifica posizione (non si è usata la routine PUT-POINT perché la sua azione è limitata alla ricerca della successiva posizione libera nel file). L'algoritmo che esplica la procedura SET-POINT, nel

caso si scelga una struttura dati ad array, è molto semplice, ma viene descritta per completezza.

Algoritmo 5.4 SET-POINT (INDEX, OP, X, Y)
Sostituisce un'istruzione nel display file.

Argomenti OP, X, Y←istruzione da sostituire
INDEX←posizione dell'istruzione da sostituire

Variabili globali DF-OP, DF-X, DF-Y←array che costituiscono il display file

Costanti DFSIZE←dimensione del display file

```
begin
  if INDEX>DFSIZE then return-error 'OVERFLOW NEL DISPLAY
       FILE';
  DF-OP[INDEX]←OP;
  DF-X[INDEX]←X;
  DF-Y[INDEX]←Y;
  return;
end;
```

Per alcune applicazioni servirà anche una routine che cancelli tutti i segmenti creati, di grande utilità all'utente nel caso egli desideri iniziare una nuova rappresentazione grafica, o anche al momento di un'inizializzazione. Per eseguire tale operazione si potrebbero cancellare individualmente tutti i segmenti: questo metodo è indipendente dalla struttura dati con cui è stato organizzato il display file. Comunque, per avere una maggiore efficienza, nel caso della particolare struttura ad array adottata in questo testo, si potrebbero porre a zero le dimensioni di tutti i segmenti, assegnare la prima locazione del display file alla variabile FREE, che indica appunto la successiva posizione libera e inizializzare al valore 1 tutte le posizioni iniziali per evitare, ad esempio, che in seguito vengano prese in considerazione dalla routine DELETE-SEGMENT, che ne aggiornerebbe inutilmente i valori.

Algoritmo 5.5 DELETE-ALL-SEGMENTS
Routine utente che cancella tutti i segmenti.

Variabili globali NOW-OPEN←segmento corrente
FREE←indice della successiva cella disponibile del display file
SEGMENT-SIZE←array contenente le dimensioni dei segmenti
SEGMENT-START←array contenente le posizioni iniziali dei segmenti

Variabili locali I←indice della tabella dei segmenti

Costanti NUMBER-OF-SEGMENTS←dimensione della tabella dei segmenti

```
begin
   for I = 0 to NUMBER-OF-SEGMENTS
   do begin
      SEGMENT-START[I]←1;
      SEGMENT-SIZE[I]←0;
      end;
   NOW-OPEN←0;
   FREE←1;
   NEW-FRAME;
   return;
end;
```

5.5 CAMBIAMENTO DI NOME A UN SEGMENTO

Un'altra routine di grande utilità, soprattutto per le applicazioni in tempo reale, è quella che permette di cambiare nome a un segmento. Supponiamo di voler mostrare un'astronave in movimento attraverso lo schermo; questo vuol dire presentare una sequenza di immagini, ciascuna con l'astronave in una posizione differente. Immaginiamo di avere un segmento contenente l'astronave: anziché usare una trasformazione di immagine per cambiarne la posizione, si potrebbero generare delle nuove istruzioni nel display file associate a ciascuna posizione; in questo modo, per ogni nuova immagine, è possibile cancellare il segmento, ricrearlo con l'astronave nella sua nuova posizione e mostrare il risultato. Il problema è che nell'intervallo di tempo intercorrente fra la creazione della seconda immagine e la cancellazione della prima sullo schermo, può comparire un'immagine incompleta dell'astronave; per evitare questo fatto si deve imporre il divieto di cancellare un segmento finché non sarà completata la sua ricostruzione, rendendo però necessaria l'esistenza contemporanea di due segmenti nel display file; per aggirare questo ostacolo si costruisce la nuova immagine invisibile in un segmento con un nome temporaneo; una volta completatane la costruzione, si cancella l'immagine originale, la si sostituisce con la nuova e si cambia il nome del nuovo segmento assegnandogli il nome di quello vecchio. Questi passi si possono ripetere per simulare il movimento e si realizzano con l'algoritmo RENAME-SEGMENT, il quale controlla che i nomi dei segmenti siano validi, che non siano stati ancora aperti e che non abbiano il nome di segmenti già esistenti. Se sono soddisfatte queste condizioni, la parte di tabella dei segmenti relativa al vecchio segmento è copiata nella posizione corrispondente al nuovo segmento e la dimensione del vecchio viene riportata a zero.

Algoritmo 5.6 RENAME-SEGMENT (SEGMENT-NAME-1, SEGMENT-NAME-2)

Routine utente che assegna un nuovo nome a un segmento.

Argomenti SEGMENT-NAME-OLD←nome del segmento che deve essere cambiato
SEGMENT-NAME-NEW←nuovo nome del segmento
Variabili globali SEGMENT-START, SEGMENT-SIZE, VISIBILITY, ANGLE, SCALE-X, SCALE-Y, TRANSLATE-X, TRANSLATE-Y ←tabella dei segmenti
NOW-OPEN←segmento attualmente aperto
Costanti NUMBER-OF-SEGMENTS←dimensione della tabella dei segmenti

```
begin
   if SEGMENT-NAME-OLD < 1 or SEGMENT-NAME-NEW < 1
      or SEGMENT-NAME-OLD > NUMBER-OF-SEGMENTS
      or SEGMENT-NAME-NEW > NUMBER-OF-SEGMENTS
      then return-error 'NOME DEL SEGMENTO NON VALIDO';
   if SEGMENT-NAME-OLD = NOW-OPEN
      or SEGMENT-NAME-NEW = NOW-OPEN
      then return-error 'SEGMENTO ANCORA APERTO';
   if SEGMENT-SIZE[SEGMENT-NAME-NEW] ≠ 0
      then return-error 'SEGMENTO GIÀ ESISTENTE';
   copia gli indici della tabella relativi al vecchio segmento nelle nuove posizioni
   SEGMENT-START[SEGMENT-NAME-NEW]
      ←SEGMENT-START[SEGMENT-NAME-OLD];
   SEGMENT-SIZE[SEGMENT-NAME-NEW]
      ←SEGMENT-SIZE[SEGMENT-NAME-OLD];
   VISIBILITY[SEGMENT-NAME-NEW]
      ←VISIBILITY[SEGMENT-NAME-OLD];
   ANGLE[SEGMENT-NAME-NEW]
      ←ANGLE[SEGMENT-NAME-OLD];
   SCALE-X[SEGMENT-NAME-NEW]
      ←SCALE-X[SEGMENT-NAME-OLD];
   SCALE-Y[SEGMENT-NAME-NEW]
      ←SCALE-Y[SEGMENT-NAME-OLD]
   TRANSLATE-X[SEGMENT-NAME-NEW]
      ←TRANSLATE-X[SEGMENT-NAME-OLD];
   TRANSLATE-Y[SEGMENT-NAME-NEW]
      ←TRANSLATE-Y[SEGMENT-NAME-OLD];
   cancella il vecchio segmento
   SEGMENT-SIZE[SEGMENT-NAME-OLD]←0;
   return;
end;
```

5.6 VISIBILITÀ DI UN SEGMENTO

Si è già accennato all'attributo di visibilità, associato a ciascun segmento, che viene memorizzato in un vettore appartenente alla tabella dei segmenti. L'accesso a questo vettore nella posizione relativa ad un determinato segmento permette di determinare se il segmento stesso dovrà essere visualizzato o meno. L'utente deve poter cambiare il valore di questo attributo a seconda che desideri far apparire o scomparire sullo schermo l'immagine contenuta in un segmento, senza conoscere la struttura interna con cui è stata implementata la segmentazione del display file. L'algoritmo che segue permette l'aggiornamento dell'attributo di visibilità e realizza un'azione di NEW-FRAME ogni volta che si deve rendere visibile un segmento (vedi Figura 5.4).

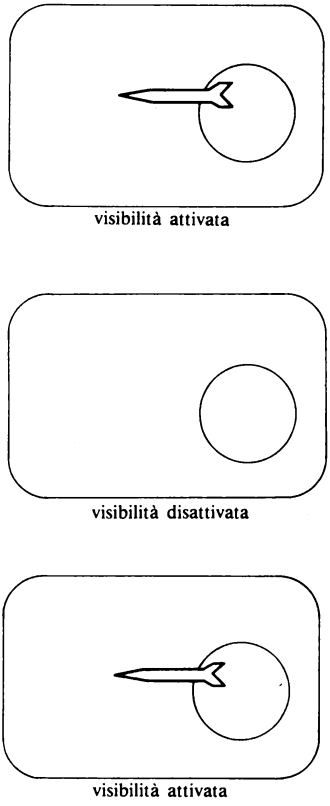


Figura 5.4 Cambiamento dell'attributo di visibilità

Algoritmo 5.7 SET-VISIBILITY (SEGMENT-NAME, ON-OFF)
Routine utente che aggiorna l'attributo di visibilità.
Argomenti SEGMENT-NAME←nome del segmento
ON-OFF←nuovo valore da assegnare all'attributo
Variabili globali VISIBILITY←array appartenente alla tabella dei segmenti contenente gli attributi di visibilità
Costanti NUMBER-OF-SEGMENTS←dimensione della tabella dei segmenti

```
begin
   if SEGMENT-NAME < 1
   or SEGMENT-NAME > NUMBER-OF-SEGMENTS
   then return-error 'NOME DEL SEGMENTO NON VALIDO';
   VISIBILITY[SEGMENT-NAME] ← ON-OFF;
   if not ON-OFF then NEW-FRAME;
   return;
end;
```

5.7 TRASFORMAZIONE DELL'IMMAGINE

Vediamo ora come e quando è possibile modificare i valori degli attributi per la trasformazione dell'immagine contenuta in un segmento. Come si è detto nel precedente capitolo, una trasformazione di immagine può essere specificata da cinque valori: i fattori di scala x e y, l'angolo di rotazione e i valori delle traslazioni lungo gli assi x e y. Devono quindi essere associati a ciascun segmento altri cinque attributi. Mentre nel Capitolo 4 alcune variabili globali sono servite per memorizzare questi parametri, ora verranno utilizzati degli array per poter memorizzare ogni singolo parametro dei segmenti: ognuno di questi array entrerà a far parte della tabella dei segmenti. Naturalmente l'utente potrà aggiornare i parametri delle trasformazioni a suo piacimento. Il primo algoritmo che verrà illustrato è quello che realizza una traslazione.

Algoritmo 5.8 SET-IMAGE-TRANSLATION (SEGMENT-NAME, TX, TY)
Routine utente che trasla l'immagine contenuta in un segmento.
Argomenti SEGMENT-NAME←il segmento da trasformare
TX, TY←parametri di traslazione
Variabili globali TRANSLATE-X, TRANSLATE-Y←parte della tabella dei segmenti contenente i parametri di traslazione.
Costanti NUMBER-OF-SEGMENTS←dimensione della tabella dei segmenti

```
begin
   if SEGMENT-NAME < 1 or SEGMENT-NAME > NUMBER-OF-
   SEGMENTS
```

```
    then return error 'NOME DEL SEGMENTO NON VALIDO';
    TRANSLATE-X[SEGMENT-NAME]←TX;
    TRANSLATE-Y[SEGMENT-NAME]←TY;
    if VISIBILITY[SEGMENT-NAME] then NEW-FRAME;
    return;
end;
```

Occorre notare che viene eseguita un'azione di NEW FRAME solo se il segmento che viene modificato è visibile. Il sistema CORE non ha routine indipendenti che permettono il cambiamento di scala e la rotazione dell'immagine contenuta in un segmento poiché spesso, dopo una rotazione ed un mutamento di scala, è necessaria una traslazione per perfezionare l'immagine. Quindi la routine che segue realizza una trasformazione composta da una trasformazione di scala, una rotazione e una traslazione (vedi Figura 5.5).

Algoritmo 5.9 SET-IMAGE-TRANSFORMATION (SEGMENT-NAME, SX, SY, A, TX, TY)

Routine utente che esegue una trasformazione composta.

Argomenti SEGMENT-NAME←segmento da trasformare
SX, SY, A, TX, TY←parametri di trasformazione

Variabili globali VISIBILITY, SCALE-X, SCALE-Y, ANGLE, TRANSLATE-X, TRANSLATE-Y←array contenenti gli attributi di trasformazione nella tabella dei segmenti

Costanti NUMBER-OF-SEGMENTS←dimensione della tabella dei segmenti

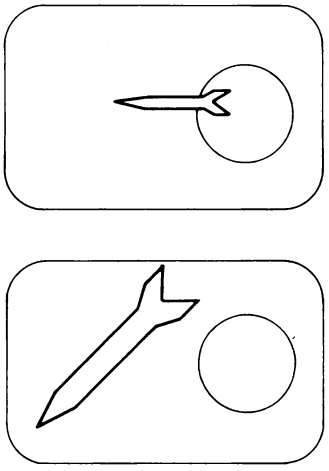


Figura 5.5 Trasformazione dell'immagine

```
begin
   if SEGMENT-NAME < 1
   or SEGMENT-NAME > NUMBER-OF-SEGMENTS
   then return-error 'NOME DEL SEGMENTO NON VALIDO';
   SCALE-X[SEGMENT-NAME] ← SX;
   SCALE-Y[SEGMENT-NAME] ← SY;
   ANGLE[SEGMENT-NAME] ← A;
   TRANSLATE-X[SEGMENT-NAME] ← TX;
   TRANSLATE-Y[SEGMENT-NAME] ← TY;
   if VISIBILITY[SEGMENT-NAME] then NEW-FRAME;
   return;
end;
```

5.8 AGGIORNAMENTO DELLE ROUTINE DI TRASFORMAZIONE

In questo paragrafo si cercherà di rendere compatibili gli algoritmi del Capitolo 4 con la segmentazione del display file, applicando le trasformazioni al segmento senza nome.

Algoritmo 5.10 TRANSLATE (TX, TY) (Algoritmo 4.6 aggiornato)
Assegna i parametri di traslazione relativi al segmento senza nome.
Argomenti TX, TY ← parametri di traslazione
Variabili globali TRANSLATE-X, TRANSLATE-Y ← array della tabella dei segmenti contenenti i parametri di traslazione

```
begin
   TRANSLATE-X[0] ← TX;
   TRANSLATE-Y[0] ← TY;
   NEW-FRAME;
   return;
end;
```

Algoritmo 5.11 SCALE (SX, SY) (Algoritmo 4.7 aggiornato)
Assegna i parametri di trasformazione di scala relativi al segmento senza nome.
Argomenti SX, SY ← fattori di scala
Variabili globali SCALE-X, SCALE-Y ← array della tabella dei segmenti contenenti i parametri delle trasformazioni di scala

```
begin
   SCALE-X[0] ← SX;
   SCALE-Y[0] ← SY;
   NEW-FRAME;
   return;
end;
```

Algoritmo 5.12 ROTATE (A) (Algoritmo 4.8 aggiornato)
Assegna parametri di rotazione relativi al segmento senza nome.
Argomenti A←angolo di rotazione
Variabili globali ANGLE←array della tabella dei segmenti contenente i parametri di rotazione

```
begin
   ANGLE[0]←A;
   NEW-FRAME;
   return;
end;
```

Anche l'algoritmo che inizializza i parametri della trasformazione identità dovrà essere modificato.

Algoritmo 5.13 IDENTITY-PARAMETERS (Algoritmo 4.14 aggiornato)
Variabili globali ANGLE, SCALE-X, SCALE-Y, TRANSLATE-X, TRANSLATE-Y←array della tabella dei segmenti contenenti i parametri delle trasformazioni

```
begin
   SCALE-X[0]←1;
   SCALE-Y[0]←1;
   ANGLE[0]←0;
   TRANSLATE-X[0]←0;
   TRANSLATE-Y[0]←0;
   return;
end;
```

L'algoritmo per costruire la matrice di trasformazione completa avrà ora come parametro il nome del segmento da trasformare.

Algoritmo 5.14 BUILD-TRANSFORMATION (SEGMENT-NAME)
(Algoritmo 4.9 aggiornato)
Costruisce la matrice di trasformazione.
Argomenti SEGMENT-NAME←segmento da trasformare
Variabili globali ANGLE, SCALE-X, SCALE-Y, TRANSLATE-X, TRASLATE-Y←array della tabella dei segmenti contenenti i parametri delle trasformazioni

```
begin
   IDENTITY-MATRIX;
   MULTIPLY-IN-SCALE(SCALE-X[SEGMENT-NAME],
      SCALE-Y[SEGMENT-NAME]);
   MULTIPLY-IN-ROTATION(ANGLE[SEGMENT-NAME]);
   MULTIPLY-IN-TRANSLATION(TRANSLATE-X[SEGMENT-
      NAME];
```

```
    TRANSLATE-Y[SEGMENT-NAME]);
    return;
end;
```

Anche questo capitolo necessiterà di una routine di inizializzazione, che renderà visibile il segmento senza nome ed effettuerà una chiamata alla routine DELETE-ALL-SEGMENTS, poiché all'inizio di ogni sessione grafica non deve esistere alcun segmento. Chiaramente, la variabile NOW-OPEN, che contiene il nome del segmento aperto in quell'istante, verrà inizializzata a zero.

Algoritmo 5.15 INITIALIZE-5

Variabili globali VISIBILITY←array della tabella dei segmenti contenente gli attributi di visibilità
NOW-OPEN←segmento corrente

```
begin
    INITIALIZE-4;
    DELETE-ALL-SEGMENTS;
    VISIBILITY[0]←TRUE;
    NOW-OPEN←0;
    return;
end;
```

5.9 MEMORIZZAZIONE E VISUALIZZAZIONE DEI SEGMENTI

Si sono visti finora molti algoritmi per la creazione e la memorizzazione di informazioni relative ai segmenti nella tabella corrispondente; a questo punto, deve essere introdotta la struttura di segmentazione nelle procedure che gestiscono il display file (vedi Figura 5.6). La prima routine da modificare è PUT-POINT. Ad essa deve essere aggiunta un'istruzione che incrementa la dimensione del segmento correntemente aperto ogni volta che viene inserita una nuova istruzione nel display file.

Algoritmo 5.16 PUT-POINT (OP, X, Y) (Algoritmo 2.1 aggiornato)
Estensione dell'algoritmo presentato in precedenza che include l'aggiornamento della tabella dei segmenti.

Argomenti OP, X, Y←istruzione del display file
Variabili globali NOW-OPEN←segmento corrente
SEGMENT-SIZE←array della tabella contenente la dimensione dei segmenti
FREE←posizione della successiva cella libera nel display file

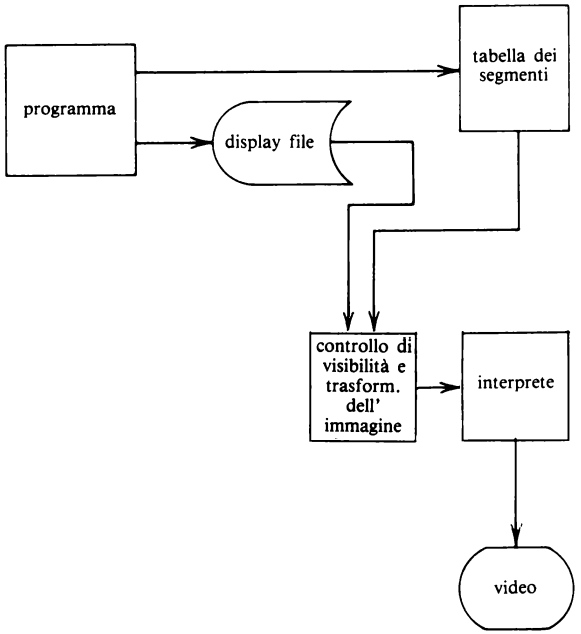


Figura 5.6 Generazione di figure con i segmenti

```
begin
   SEGMENT-SIZE [NOW-OPEN] ← SEGMENT-SIZE [NOW-OPEN] + 1;
   SET-POINT (FREE, OP, X, Y);
   FREE ← FREE + 1;
   return;
end;
```

La seconda routine da modificare è MAKE-PICTURE-CURRENT, che serviva a visualizzare il contenuto dell'intero display file, a partire dall'istruzione 1 per terminare all'istruzione corrispondente all'indice (FREE − 1). La modifica da apportare consiste in un ciclo che esamina ogni segmento, verificando se la dimensione è maggiore di zero e se il segmento è visibile. Nel caso in cui entrambe le condizioni siano verificate, viene costruita la matrice di trasformazione e viene interpretato il segmento. La tabella dei segmenti indicherà dove inizia il segmento e quante istruzioni devono essere interpretate. Queste informazioni vengono passate alla routine INTERPRET come argomenti (vedi Figura 5.7).

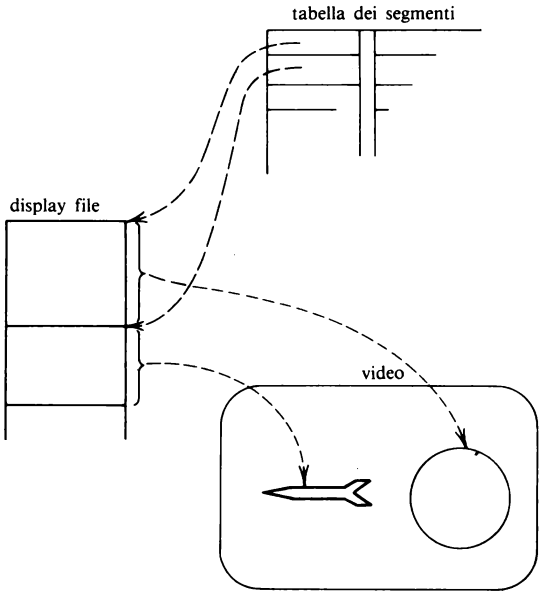


Figura 5.7 La tabella dei segmenti rappresenta la porzione del display file usata per la costruzione della figura

Algoritmo 5.17 MAKE-PICTURE-CURRENT
(Algoritmo 4.10 aggiornato)

Variabili globali SEGMENT-START, SEGMENT-SIZE,
VISIBILITY—tabella dei segmenti
ERASE-FLAG←indicatore per cancellare il display
Variabili locali I←indice della tabella dei segmenti
Costanti NUMBER-OF-SEGMENTS←dimensione della tabella dei segmenti

```
begin
  if ERASE-FLAG then
       begin
       ERASE;
       ERASE-FLAG←FALSE;
       end;
  for I = 0 to NUMBER-OF-SEGMENTS
  do if SEGMENT-SIZE[I]≠0 and VISIBILITY[I]
       then begin
       BUILD-TRANSFORMATION(I);
       INTERPRET(SEGMENT-START[I], SEGMENT-SIZE[I]);
```

```
        end;
    DISPLAY;
    DELETE-SEGMENT(0);
    return;
end;
```

5.10 ALTRE STRUTTURE DEL DISPLAY FILE

Le operazioni che generalmente vengono eseguite sul display file sono: l'inserimento, quando si costruisce un'immagine; la selezione, quando vengono interpretate e visualizzate le istruzioni contenute nel file; la cancellazione, quando un segmento non serve più. Tra le molte strutture dati che potevano essere utilizzate, è stata scelta la più semplice: la tabella costituita da array monodimensionali. Mentre l'inserimento e la selezione sono facili da trattare con gli array, la cancellazione può non essere molto efficiente poiché, dovendo spostare un'istruzione all'inizio del display file, occorre spostare anche tutte le istruzioni che seguono e, se il display file è di grandi dimensioni, per eseguire quest'operazione occorrerà elaborare un numero notevole di istruzioni. Una struttura dati alternativa potrebbe essere la *lista concatenata* (vedi Figura 5.8). In essa le istruzioni non vengono memorizzate in un ordine preciso, bensì viene utilizzato un elemento della lista che fornisce la posizione dell'istruzione successiva. In questo modo si può passare da un'istruzione alla successiva seguendo questa serie di legami realizzati tramite l'uso di un puntatore. Quando si aggiunge una nuova istruzione al display file, si memorizzano il codice operazione e gli operandi in una nuova cella e questa nuova cella così creata viene concatenata alla lista del display file. La cancellazione di celle da una lista è un'operazione molto semplice, poiché è sufficiente cambiare il puntatore a quella cella in modo che indichi la posizione della cella successiva (vedi Figura 5.9), mentre le celle eliminate dalla lista possono essere riutilizzate.
In una struttura a lista, cancellare degli elementi significa cambiare due legami: questo comporta una maggiore efficienza rispetto allo spostamento di istruzioni necessario per realizzare la medesima operazione su di una struttura ad array. Gli svantaggi di un'organizzazione a lista sono la grande quantità di memoria necessaria per memorizzare i puntatori oltre alle altre informazioni e l'alto costo implicito nella locazione arbitraria delle celle.
Il terzo schema proposto, che si pone tra la struttura ad array e la struttura a lista, è l'organizzazione a pagine: il display file è strutturato in un numero di piccoli array chiamati *pagine* che sono concatenati in modo da formare una lista. Ciascun segmento inizia in testa ad una pagina; se il segmento termina in un punto che non è il confine della pagina, il resto della pagina non viene utilizzato. L'accesso all'interno di una pagina è quello definito per gli array. Ogni volta che si giunge alla fine di una pagina, si considera il puntatore per accedere ad una pagina successiva (vedi Figura 5.10).
Raggruppando le istruzioni in pagine, abbiamo ridotto il numero di legami, ma

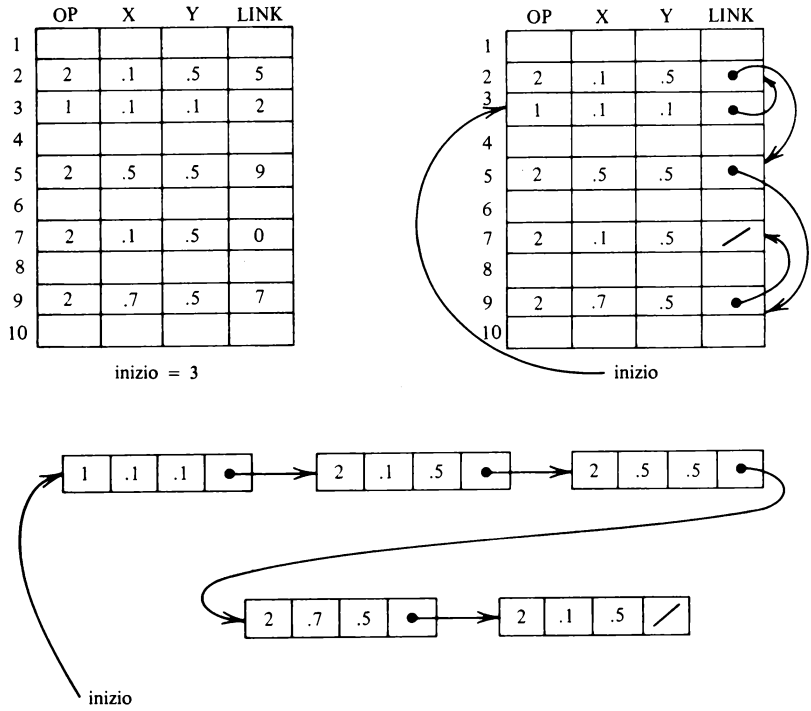


Figura 5.8 Il display file come lista concatenata (in tre modi differenti)

è ancora possibile cancellare un segmento con il semplice mutamento dei due puntatori. Anche in questo tipo di struttura le pagine cancellate si possono riutilizzare. Gli svantaggi legati a questo tipo di struttura sono la perdita dello spazio, nel caso in cui un segmento non completi una pagina, e l'accesso più laborioso alle informazioni rispetto alle strutture precedenti. Come abbiamo già visto, una volta che un segmento è stato chiuso non può più essere modificato né tanto meno esteso. Questa è una restrizione imposta dal sistema CORE. Altri sistemi permettono operazioni simili (anche se potrebbero sorgere alcuni problemi, come ad esempio la modifica di attributi quali lo stile della linea o il formato dei caratteri al momento della riapertura del segmento). Nel caso siano permesse operazioni di editing nel display file, è più naturale utilizzare una

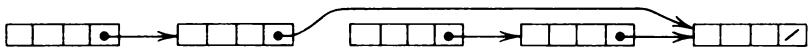


Figura 5.9 Cancellazione di un'istruzione del display file da una lista concatenata

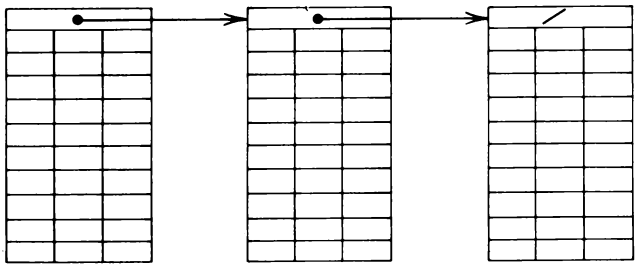
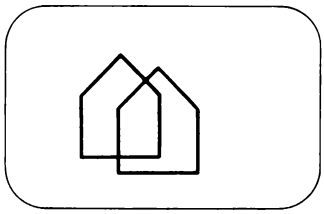


Figura 5.10 Pagine concatenate di istruzioni del display file

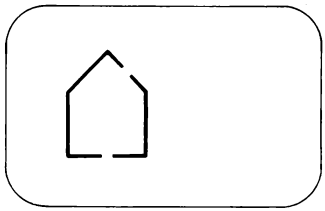
struttura a lista che non una struttura ad array. Queste estensioni verranno considerate nel problema di programmazione 5.9.
Esistono molte altre strutture dati oltre alle tre considerate. Gli accessi al display file sono stati eseguiti solo nelle routine GET-POINT e SET-POINT (PUT-POINT può essere implementata come una chiamata alla SET-POINT) sebbene anche la routine di cancellazione di un segmento dipenda dalla struttura del display file. In questo modo si può organizzare il display file con strutture dati differenti, apportando un esiguo numero di modifiche all'algoritmo.

5.11 ALCUNE TECNICHE RASTER

Nel pacchetto grafico che si sta sviluppando, l'unico metodo che si ha a disposizione per modificare un'immagine è quello di cambiare il display file o trasformare l'immagine e interpretare nuovamente l'intera figura. Questo metodo va bene per terminali CRT a memoria e a vector refresh, ma è inefficiente per terminali a tecnologia raster, in cui la pulizia del video e il calcolo dei nuovi valori dei pixel sono operazioni costose in termini di tempo. Con questi dispositivi si consiglia una tecnica di modifica locale dell'immagine che permetta di apportare cambiamenti ad una porzione del display, lasciando inalterata la rimanente parte del frame buffer. Per esempio, supponiamo di rendere invisibile un segmento e di lasciare inalterato lo stato corrente dei rimanenti; invece di eseguire una pulizia del frame buffer e ridisegnare l'intera figura, possiamo ricostruire il segmento che vogliamo rendere invisibile assegnando ad ogni pixel il valore dello sfondo, cancellando così tutte le linee tracciate precedentemente. Questa tecnica potrebbe causare delle interruzioni sulle linee appartenenti ad altri segmenti che hanno dei pixel in comune con il segmento da rendere invisibile (per esempio nei punti di intersezione appartenenti ad un segmento visibile e ad uno da rendere invisibile). Per risolvere questo problema, si devono reinterpretare i segmenti ancora visibili senza la pulizia totale del frame buffer (vedi Figura 5.11). Un'altra tecnica simile può essere usata per eseguire delle efficienti

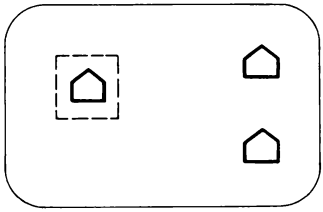


prima

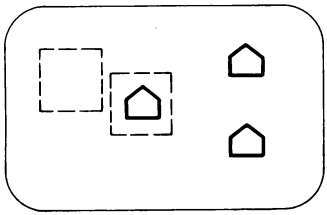


dopo

Figura 5.11 Cancellazione di un'immagine ottenuta ridisegnandola con il valore dello sfondo



prima



dopo

Figura 5.12 Modifica dei soli pixel contenuti nel rettangolo

traslazioni di immagini sullo schermo copiando i pixel da una posizione all'altra: nel caso in cui la figura sia contenuta in un rettangolo, si possono copiare solo quei pixel interni al rettangolo. Questa tecnica è vantaggiosa rispetto a quella che assegna ai pixel nel frame buffer il valore dello sfondo e ricalcola i pixel corrispondenti all'immagine traslata (vedi Figura 5.12).
Talvolta si adotta la tecnica di lavorare soltanto su un sottoinsieme del frame buffer di dimensioni fissate (ad esempio 8×8 pixel); i cambiamenti effettuati in questi sottoinsiemi risultano essere più veloci che non quelli eseguiti sull'intero frame buffer. Si è già introdotta questa idea a proposito della generazione dei caratteri: i valori dei pixel relativi ad un carattere si possono copiare da uno schema predefinito in un sottoinsieme del frame buffer. Alcune semplici operazioni su sottoinsiemi del frame buffer (del tipo attivazione o disattivazione dei pixel, copia dei valori relativi al pixel da un altro sottoinsieme, operazioni logiche come AND, OR, XOR, traslazione dei valori di una riga o di una colonna) si possono implementare a livello hardware, ottenendo così una maggiore velocità di esecuzione. Tali operazioni sono talvolta chiamate BITBLT, abbreviazione del termine inglese *bit block transfer* che significa trasferimento di blocchi di bit; questa tecnica permette anche l'animazione in tempo reale di porzioni di schermo (vedi Figura 5.13).

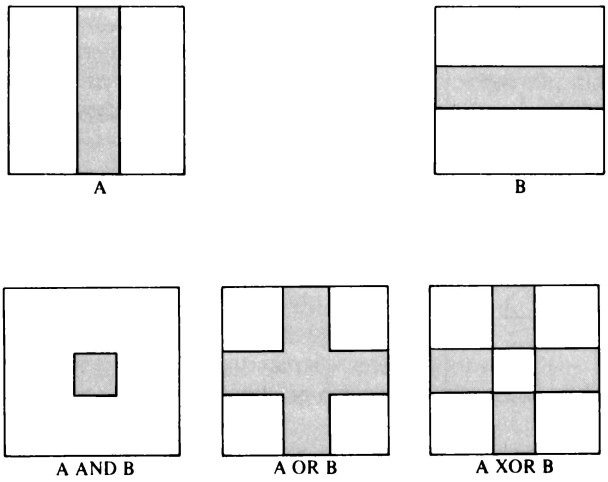


Figura 5.13 Operazioni logiche eseguite su figure

APPLICAZIONE PRATICA

Si supponga di voler utilizzare un sistema grafico computerizzato per simulare l'ormeggio di una nave. La posizione della nave rispetto alla banchina (misurata dai sensori della nave) è presentata graficamente con una vista dall'alto. Come si può scrivere questo programma nel sistema presentato? Si possono usare, per esempio, i comandi LINE e POLYGON per generare le immagini della banchina e della nave. Inserendo le istruzioni relative a queste figure in differenti segmenti del display file, si possono applicare in modo indipendente diverse trasformazioni a questi segmenti, ottenendo le varie posizioni della nave rispetto alla banchina (vedi Figura 5.14).

```
CREATE-SEGMENT(1);
disegna la nave
CLOSE-SEGMENT;
CREATE-SEGMENT(2);
disegna la banchina
CLOSE-SEGMENT;
```

Ora è necessario un ciclo per aggiornare ripetutamente la posizione della nave sullo schermo, che verrà ricavata per mezzo dei sensori della nave, applicando rotazioni e traslazioni. Naturalmente, se la posizione della nave non è cambiata dall'ultimo controllo, non sarà effettuato alcun aggiornamento. Questo programma può essere esteso per simulare l'ormeggio in diversi porti, inserendo in differenti segmenti le rappresentazioni delle banchine di ciascun porto.

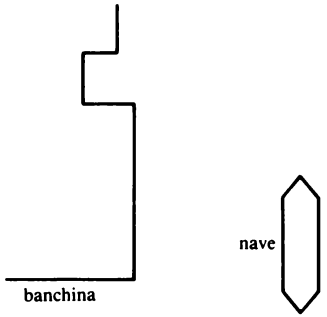


Figura 5.14 Rappresentazione grafica della posizione di una nave in fase di attracco

ESERCIZI

5.1 Si supponga che la tabella dei segmenti abbia spazio per soli 5 segmenti più il segmento senza nome. Si descriva la tabella dei segmenti generata da ciascuna delle seguenti porzioni di programma.

a) INITIALIZE-5
MOVE-ABS-2(0.5, 0.5)
LINE-ABS-2(0.5, 1.0)
LINE-ABS-2(0.0, 1.0)
LINE-ABS-2(0.5, 0.5)

b) INITIALIZE-5
CREATE-SEGMENT(1)
MOVE-ABS-2(0.5, 0.5)
LINE-ABS-2(0.5, 1.0)
LINE-ABS-2(0.0, 1.0)
LINE-ABS-2(0.5, 0.5)
CLOSE-SEGMENT

c) INITIALIZE-5
CREATE-SEGMENT(4)
MOVE-ABS-2(0.5, 0.5)
LINE-ABS-2(0.5, 1.0)
LINE-ABS-2(0.0, 1.0)
LINE-ABS-2(0.5, 0.5)
CLOSE-SEGMENT

d) INITIALIZE-5
CREATE-SEGMENT(3)
MOVE-ABS-2(0.5, 0.5)
LINE-ABS-2(0.5, 1.0)
CLOSE-SEGMENT
LINE-ABS-2(0.0, 1.0)
LINE-ABS-2(0.5, 0.5)

e) INITIALIZE-5
MOVE-ABS-2(0.5, 0.5)
CREATE-SEGMENT(2)
LINE-ABS-2(0.5, 1.0)
LINE-ABS-2(0.0, 1.0)
CLOSE-SEGMENT
LINE-ABS-2(0.5, 0.5)

f) INITIALIZE-5
CREATE-SEGMENT(2)
MOVE-ABS-2(0.5, 0.5)
LINE-ABS-2(0.5, 1.0)
CLOSE-SEGMENT
CREATE-SEGMENT(4)
LINE-ABS-2(0.0, 1.0)
LINE-ABS-2(0.5, 0.5)
CLOSE-SEGMENT

g) INITIALIZE-5
CREATE-SEGMENT(2)

MOVE-ABS-2(0.5, 0.5)
LINE-ABS-2(0.5, 1.0)
CLOSE-SEGMENT
CREATE-SEGMENT(4)
LINE-ABS-2(0.0, 1.0)
LINE-ABS-2(0.5, 0.5)
CLOSE-SEGMENT
DELETE-SEGMENT(4)

h) INITIALIZE-5
CREATE-SEGMENT(2)
MOVE-ABS-2(0.5, 0.5)
LINE-ABS-2(0.5, 1.0)
CLOSE-SEGMENT
CREATE-SEGMENT(4)
LINE-ABS-2(0.0, 1.0)
LINE-ABS-2(0.5, 0.5)
CLOSE-SEGMENT
DELETE-SEGMENT(2)

i) INITIALIZE-5
CREATE-SEGMENT(2)
MOVE-ABS-2(0.5, 0.5)
LINE-ABS-2(0.5, 1.0)
CLOSE-SEGMENT
RENAME-SEGMENT(2,4)

5.2 Suggerire altri attributi, oltre a quelli di visibilità e di trasformazione della figura, da associare ad un segmento del display file.

5.3 In questa implementazione si è usato uno spazio nella tabella riservato a ciascun segmento tale che, se si volessero creare 1000 segmenti, la dimensione dell'array dovrebbe essere di almeno 1000 unità. Supponiamo che siano permessi all'utente degli indici da 1 a 10000 (come nome dei segmenti) con la restrizione di non poter usare più di 100 segmenti alla volta. Si provi ad organizzare la tabella dei segmenti seguendo questa regola, usando array di solo 100 elementi.

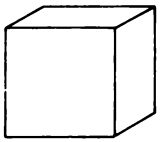
5.4 Si organizzi il display file mediante la struttura dati a liste concatenate e si rivedano gli algoritmi PUT-POINT, GET-POINT, SET-POINT e DELETE-SEGMENT.

PROBLEMI DI PROGRAMMAZIONE

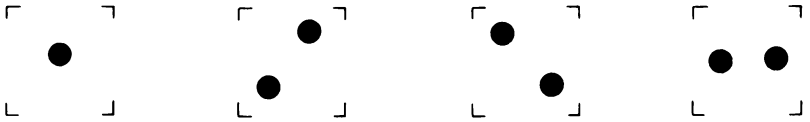
5.1 Si implementino gli algoritmi dal 5.1 al 5.17, che permettono la segmentazione del display file.

5.2 Scrivere le routine:
INQUIRE-SEGMENT-OPEN(SEGMENT-NAME)
INQUIRE-VISIBILITY(SEGMENT-NAME, ON-OFF)
INQUIRE-IMAGE-TRANSFORMATION(SEGMENT-NAME, SX, SY, A, TX, TY)
che restituiscono all'utente il nome del segmento corrente, lo stato di visibilità e i parametri di trasformazione.

5.3 Scrivere il programma DADI(N) che visualizza l'immagine di un dado con N punti (con N che va da 1 a 6). Usare un segmento per disegnare il cubo



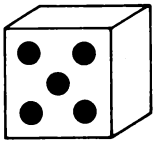
e quattro altri segmenti contenenti i seguenti schemi.



Visualizzare N punti rendendo visibili i segmenti dei 4 schemi. Per esempio, se considerassimo N uguale a 5 dovremo visualizzare i primi 3 schemi e rendere invisibile il quarto, ottenendo



che assieme al cubo darebbe



5.4 Si consideri la mappa del problema di programmazione 3.5. Si vuole ridisegnare la stessa mappa (non riempita) con etichette diverse, nelle seguenti maniere:

a) con la posizione dei distributori di caramelle
b) con la posizione dei distributori di cioccolatini della concorrenza
c) con la posizione di entrambi.

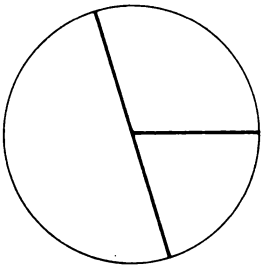
Per far ciò, mettere la mappa in un segmento, dei simboli per i distributori di caramelle in un secondo e simboli differenti per i distributori di cioccolatini in un terzo. Creare le mappe desiderate assegnando i rispettivi attributi di visibilità. I distributori devono venir mostrati nelle seguenti posizioni:

caramelle (0.2, 0.7), (0.3, 0.5), (0.8, 0.8), (0.6, 0.5)
cioccolatini (0.8, 0.2), (0.2, 0.3), (0.3, 0.7), (0.7, 0.7)

5.5 Scrivere una routine per disegnare 4 immagini che diano l'effetto del caleidoscopio, ciascuna delle quali sarà ottenuta dalla precedente applicando una trasformazione: alla prima si dovrà applicare la trasformazione identità; la seconda, la terza e la quarta si otterranno rispettivamente mediante una simmetria rispetto alla retta $x = 0.5$, alla retta $y = 0.5$ ed al punto (0.5, 0.5).

5.6 Considerare il comando POLYGON per disegnare un quadrato di vertici (0,0), (0.1), (1,1), (1,0) ed una trasformazione che lo riduca di un fattore 0.8, lo ruoti di un fattore $\pi/5$ e lo posizioni in (0.5, 0.5). Costruire una successione di segmenti contenenti un quadrato più piccolo ed ulteriormente ruotato rispetto al precedente. Infine visualizzare tutti i segmenti.

5.7 Scrivere la routine PIE-CHART(FA, N, R) che abbia come argomenti un array FA di frazioni, il contatore N delle frazioni ed un raggio R. Il programma dovrà disegnare separatamente dei settori circolari per ogni frazione, in modo da formare un cerchio. Costruire ciascuno spicchio chiamando la routine PIE del problema di programmazione 4.5, facendo in modo che tutti gli spicchi appartengano a segmenti differenti. Per esempio supponiamo che FA abbia come parametri attuali i valori 0.3, 0.5 e 0.2; si otterrà la figura



5.8 a) Disegnare una faccia usando segmenti diversi per la forma del viso, gli occhi, il naso, la bocca.
b) Mutare l'espressione del viso applicando trasformazioni diverse.

***5.9** Esistono due routine che, pur non facendo parte del sistema CORE, sono utili per alcune applicazioni. La prima routine COPY-SEGMENT(SEGMENT-1, SEGMENT-2) fa disegnare da SEGMENT-2 la stessa figura (eventualmente trasformata) di SEGMENT-1. Per realizzare quest'operazione si possono copiare tutte le istruzioni di SEGMENT-1 alla fine del display file ed associarle a SEGMENT-2; tuttavia un metodo più efficiente è quello di assegnare a SEGMENT-2 lo stesso indirizzo iniziale nel display file di SEGMENT-1. In questo caso, le istruzioni vengono memorizzate una sola volta, pur essendo condivise da entrambi i segmenti. Il problema è che la routine DELETE-SEGMENT diviene ulteriormente complicata. Una seconda routine di grande utilità è EXTEND-SEGMENT (SEGMENT). Questa routine riapre un segmento che è stato precedentemente chiuso, non cancellando alcuna istruzione già esistente, ma permettendo all'utente di introdurne delle nuove.

a) Progettare ed implementare la routine COPY-SEGMENT
b) Progettare ed implementare la routine EXTEND-SEGMENT.

6

Windowing e clipping

INTRODUZIONE

Si supponga che un architetto sia in possesso di un programma grafico in grado di disegnare un intero edificio, ma che sia interessato al solo piano terreno, oppure che un uomo d'affari disponga di una cartina con l'andamento delle vendite dell'intera nazione ma sia interessato al solo Nord-Ovest e Sud-Est; un progettista di circuiti integrati, pur avendo un programma per la visualizzazione di un intero chip, può essere interessato di volta in volta a pochi particolari. Spesso il calcolatore è usato nella fase di progettazione poiché è possibile creare, archiviare e modificare molto facilmente e con grande accuratezza disegni estremamente complessi. In questi casi è utile poter visualizzare solo quelle porzioni del disegno che sono di immediato interesse, come se si potesse guardare un'immagine attraverso un riquadro. Inoltre, è utile ingrandire queste porzioni di immagine per utilizzare al meglio la superficie di visualizzazione disponibile. Il metodo per selezionare e ingrandire porzioni di un disegno viene detto *windowing* (finestratura). La tecnica per eliminare le parti del disegno che non interessano viene detta invece *clipping*. In questo capitolo verranno considerate le tecniche di windowing, clipping e le trasformazioni di visualizzazione, con le quali è possibile posizionare correttamente e ingrandire il particolare che si vuole visualizzare. Si aggiungeranno quindi al sistema grafico le routine per assegnare le dimensioni della window e dello spazio di visualizzazione sullo schermo (viewport) e quelle per rimuovere i segmenti di retta che giacciono fuori dalla regione in cui si vuole visualizzare (clipping). Le routine per il trattamento delle trasformazioni descritte nel Capitolo 4 saranno utili per assegnare i valori alle trasformazioni di visualizzazione che verranno applicate ad ogni istruzione del display file al momento della loro esecuzione, in modo da ottenere una visualizzazione del particolare desiderato.

6.1 TRASFORMAZIONE DI VISUALIZZAZIONE

Può essere utile pensare a due modelli distinti dell'entità da visualizzare: c'è, da una parte, il modello dell'oggetto e, dall'altra, la sua immagine che appare sullo schermo. Quando si parla dell'oggetto, si fa riferimento al suo modello memorizzato nel calcolatore; si dice per questo che il modello è una rappresentazione all'interno di uno *spazio-oggetto*, poiché si rappresentano le dimensioni dell'oggetto utilizzando la sua unità di misura fisica. Nello spazio-oggetto le lunghezze dell'oggetto possono essere espresse in qualunque unità di misura, dagli anni-luce agli angstrom. La lunghezza dell'immagine sullo schermo deve essere invece misurata in coordinate schermo normalizzate, tali cioè che possano variare tra 0 e 1 (vedi Figura 6.1). Nello spazio-oggetto, una posizione verrà misurata, ad esempio, in metri; nello spazio-immagine invece la stessa posizione è data in coordinate schermo normalizzate.
Il sistema grafico dovrà quindi disporre di un metodo che converta le unità di misura dallo spazio-oggetto in quelle dello *spazio-immagine* (coordinate schermo normalizzate). Ciò può essere fatto utilizzando la trasformazione di scala descritta nel Capitolo 4. Cambiando il fattore di scala, si potranno ridurre uniformemente le dimensioni dell'oggetto finché le sue misure siano comprese tra 0 e 1. Se si desidera visualizzare oggetti molto piccoli, questi possono essere in-

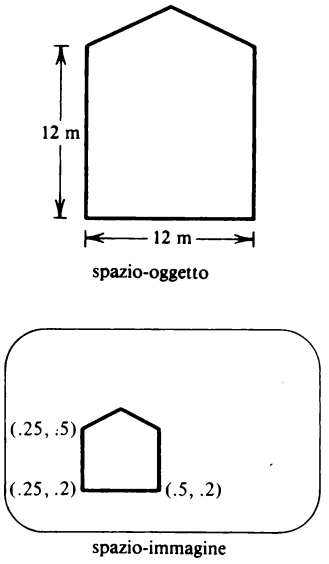


Figura 6.1 Nello spazio-oggetto, la posizione viene misurata in unità fisiche (per esempio in metri). Nello spazio-immagine, invece, la posizione viene fornita in coordinate schermo normalizzate

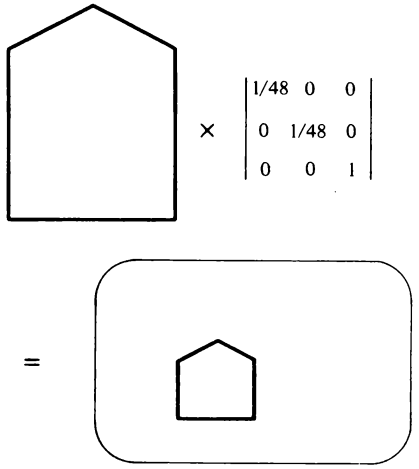
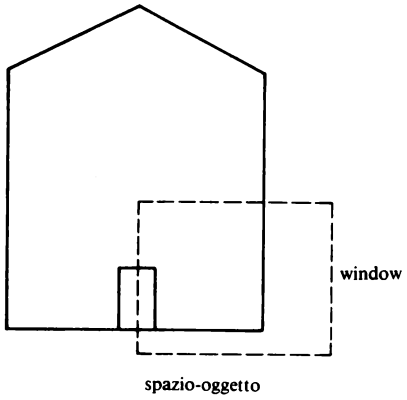


Figura 6.2 Una trasformazione di scala converte le unità di coordinate oggetto in unità di coordinate schermo normalizzate



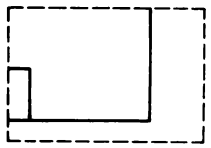


Figura 6.3 Una window usata per vedere solo una porzione dell'oggetto

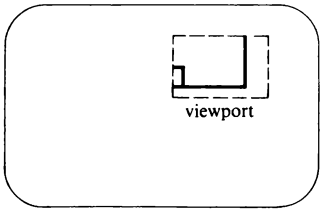


Figura 6.4 Una viewport definisce la parte di schermo che si intende utilizzare

granditi fino ad occupare tutto lo schermo disponibile o fino a riempire la porzione dello schermo nella quale deve comparire la proiezione dell'oggetto. Il fattore di scala sarà tale che le quote di ingombro della figura coincidano con le dimensioni della window di visualizzazione e quindi siano comprese tra 0 e 1. Se invece l'oggetto è particolarmente complesso, e quindi se ne vuole visualizzare una sola parte, si dovrà determinare una trasformazione di scala in grado di visualizzare la parte desiderata, che viene detta *window*, cioè finestra (vedi Figura 6.3).
In generale accade che non si desideri usare l'intero schermo per la visualizzazione: si vuole cioè che sullo schermo compaia un rettangolo e che l'immagine sia confinata in esso. Tale rettangolo sullo schermo è detto *viewport* (vedi Figure 6.4, 6.5 e 6.6). Una volta che siano state definite sia la window che la viewport, saranno disponibili abbastanza informazioni per determinare la traslazio-

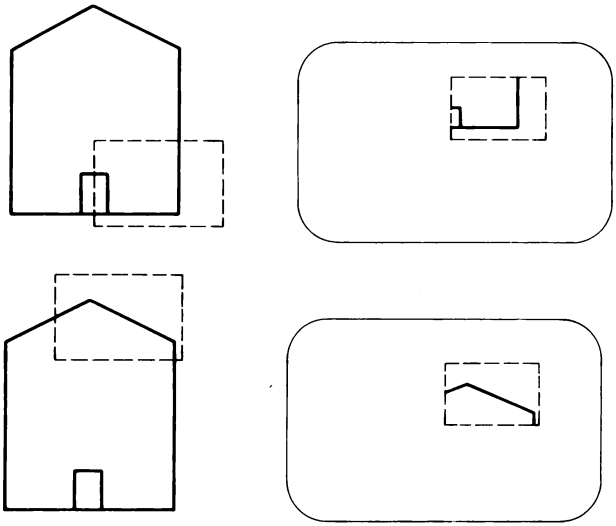


Figura 6.5 Diverse window, stessa viewport

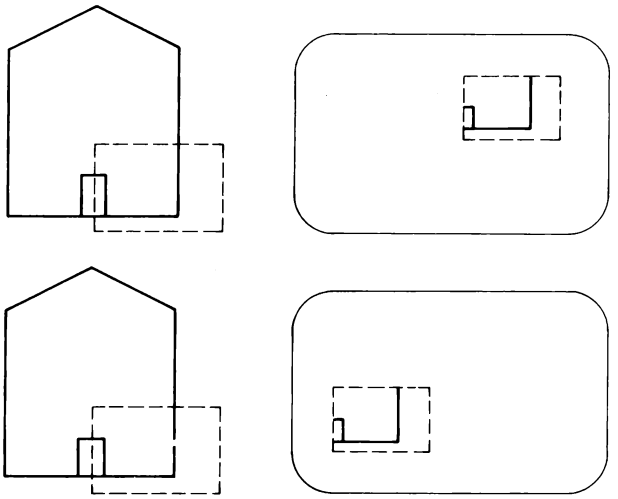


Figura 6.6 Stessa window, diverse viewport

ne e le trasformazioni di scala necessarie per convertire lo spazio-oggetto in spazio-immagine. Questa conversione può essere scomposta nei seguenti tre passi: inizialmente l'oggetto e la window che individua la porzione che si vuole visualizzare devono essere traslati finché l'angolo inferiore sinistro della window è nell'origine del sistema di riferimento; il secondo passo consiste nel cambiare il fattore di scala dell'oggetto e della window in modo che la window abbia le stesse dimensioni della viewport. Quest'operazione ha l'effetto di convertire le dimensioni dell'oggetto in coordinate dell'immagine e la window che individua la porzione dell'oggetto che deve essere visualizzata viene trasformata in viewport. L'ultimo passo di conversione è una seconda traslazione che muove la viewport nella sua corretta posizione sullo schermo (vedi Figura 6.7).
In realtà i passi che si devono fare sono solo due. Si modificano le dimensioni della window per farla diventare delle dimensioni della viewport (modificando il fattore di scala) e si posiziona la viewport nella posizione desiderata sullo schermo (con una trasformazione di visualizzazione). Il posizionamento consiste nel muovere l'angolo inferiore sinistro della window fino all'angolo inferiore sinistro della viewport, ma si dovrà eseguire quest'operazione in due passi. Per prima cosa si dovrà muovere l'angolo inferiore sinistro della window fino all'origine e in un secondo tempo lo si sposta fino a farlo coincidere con il corrispondente angolo della viewport. Si è costretti ad impiegare due passi a causa del fatto che è possibile effettuare la riduzione di scala senza disturbare la posizione dell'angolo solo quando questo è nell'origine. La trasformazione complessiva che realizza questi tre passi verrà chiamata in seguito *trasformazione di visualizzazione* (vedi Figura 6.8).

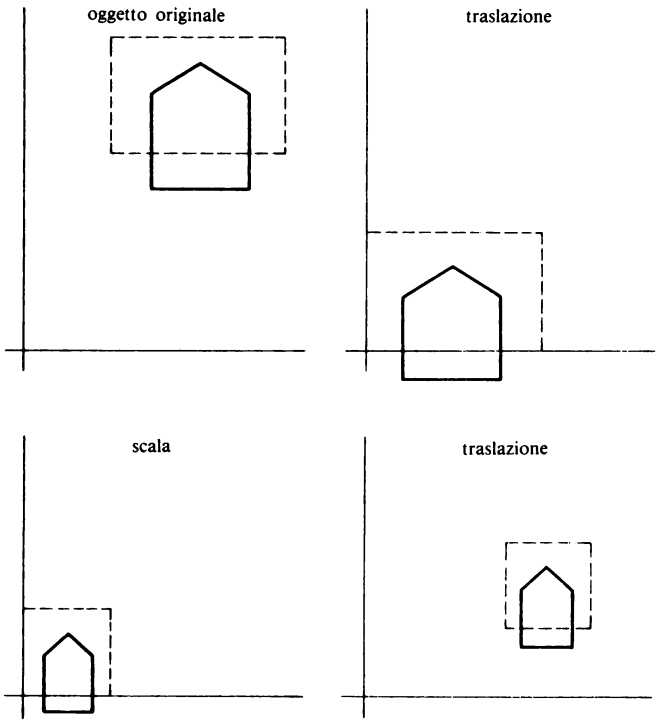


Figura 6.7 Fasi relative alla trasformazione di visualizzazione

Consideriamo ora un esempio di trasformazione di visualizzazione. Se la window ha come limiti sinistro e destro rispettivamente i valori 3 e 5 e come limiti inferiore e superiore i valori 0 e 4, allora la prima matrice di traslazione corrispondente sarà:

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{vmatrix}$$

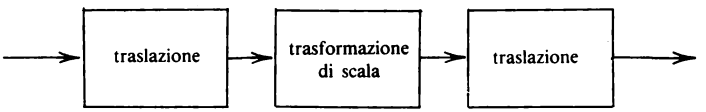


Figura 6.8 Trasformazione di visualizzazione

Si supponga che la viewport sia il quadrante superiore a destra dello schermo limitato tra 0.5 e 1.0 in entrambe le direzioni x e y. La lunghezza della window è quindi $5-3 = 2$ nella direzione x e la lunghezza corrispondente della viewport è $1.0-0.5 = 0.5$. Così il fattore di scala per la conversione delle coordinate x può essere calcolato come $0.5/2 = 0.25$. Con procedimento analogo il fattore di scala nella direzione y si ottiene come $0.5/4 = 0.125$. La matrice di trasformazione per cambiare il fattore di scala sarà:

$$\begin{vmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Infine per posizionare la viewport è richiesta una traslazione data da:

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0.5 & 1 \end{vmatrix}$$

La trasformazione di visualizzazione è allora:

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{vmatrix} \begin{vmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0.5 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ -0.25 & 0.5 & 1 \end{vmatrix} \tag{6.1}$$

In generale la trasformazione di visualizzazione è definita come:

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -WXL & -WYL & 1 \end{vmatrix} \begin{vmatrix} \frac{(VXH-VXL)}{(WXH-WXL)} & 0 & 0 \\ 0 & \frac{(VYH-VYL)}{(WYH-WYL)} & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ VXL & VYL & 1 \end{vmatrix}$$

$$= \begin{vmatrix} \frac{(VXH-VXL)}{(WYH-WXL)} & 0 & 0 \\ 0 & \frac{(VYH-VYL)}{(WYH)-(WYL)} & 0 \\ VXL-WXL\,\frac{(VXH-VXL)}{(WXH-WXL)} & VYL-WYL\,\frac{(VYH-VYL)}{(WYH-WYL)} & 1 \end{vmatrix} \tag{6.2}$$

6.2 ALGORITMI PER LA TRASFORMAZIONE DI VISUALIZZAZIONE

Il primo passo nella trasformazione di visualizzazione consiste nello specificare le dimensioni della window. Si impone come restrizione che la forma delle window sia rettangolare con i lati paralleli agli assi x e y. Con queste limitazioni è necessario specificare solo i valori massimi e minimi sia per la coordinata x che per la y. La routine che assegna le dimensioni della window memorizzerà questi valori in variabili globali del sistema grafico, cosicché essi saranno disponibili al momento di eseguire la trasformazione. Allo stesso modo si devono specificare i confini della viewport. Per determinare le dimensioni della viewport e della window sia nella direzione x che y si dovrà semplicemente fare la differenza tra i limiti superiore e inferiore che definiscono i rettangoli. Si osservi che non dovrà mai accadere che il limite inferiore sia uguale al limite superiore per una certa window perché questo obbligherebbe ad eseguire una divisione di un numero per zero nel calcolo dei parametri per determinare la trasformazione di scala. Di seguito vengono descritti gli algoritmi per assegnare le dimensioni della window e della viewport.

Algoritmo 6.1 SET-VIEWPORT (XL, XH, YL, YH)
Routine utente che definisce le dimensioni della viewport.
Argomenti XL, XH←confini laterali sinistro e destro della viewport
YL, YH←confini superiore e inferiore della viewport
Variabili globali VXL-HOLD, VXH-HOLD, VYL-HOLD, VYH-HOLD← parametri dei confini della viewport assegnati dall'utente

```
begin
   if XL≥XH or YL≥YH
   then return-error 'VIEWPORT ERRATA';
   VXL-HOLD←XL;
   VXH-HOLD←XH;
   VYL-HOLD←YL;
   VYH-HOLD←YH;
   return;
end;
```

Algoritmo 6.2 SET-WINDOW (XL, XH, YL, YH)
Routine utente che definisce le dimensioni della window.
Argomenti XL, XH←confini laterali sinistro e destro della window
YL, YH←confini superiore e inferiore della window
Variabili globali WXL-HOLD, WXH-HOLD, WYL-HOLD, WYH-HOLD← parametri dei confini della window assegnati dall'utente

```
begin
   if XL≥XH or YL≥YH
   then return-error 'WINDOW ERRATA';
   WXL-HOLD←XL;
   WXH-HOLD←XH;
   WYL-HOLD←YL;
   WYH-HOLD←YH;
   return;
end;
```

Il sistema CORE stabilisce che non è possibile cambiare i parametri di visualizzazione all'interno delle istruzioni che definiscono un segmento. Usando un'analogia, si può pensare che ciascun segmento venga considerato nel sistema CORE come la fotografia di un oggetto e in modo corrispondente la trasformazione di visualizzazione descrive come viene sistemata la macchina fotografica. In questa analogia è ragionevole impedire dei movimenti della macchina fotografica mentre viene ripresa l'immagine. Nella costruzione del sistema grafico che si sta via via definendo, si seguirà questa regola. Per questo si dovranno avere due insiemi di parametri di visualizzazione: il primo verrà usato dell'utente per apportare le modifiche, il secondo nel momento in cui vengono eseguite le routine di windowing e clipping. Il cambiamento della window sarà quindi un processo composto di due fasi. Inizialmente l'utente modifica i valori di uno dei due insiemi di parametri che definiscono la window e la viewport; in seguito, al momento in cui questi devono essere usati, i valori definiti dall'utente vengono copiati nelle variabili usate dalle routine di windowing e di clipping. Eseguendo questa copia dei parametri durante il processo di creazione dei segmenti, si assicura che nessuna variazione dei parametri di visualizzazione possa avvenire durante l'interpretazione delle istruzioni che costituiscono il segmento. Possiamo notare che questa restrizione sul cambiamento dei parametri di visualizzazione è stata definita solo nell'ambito del sistema CORE. Non c'è fonda-

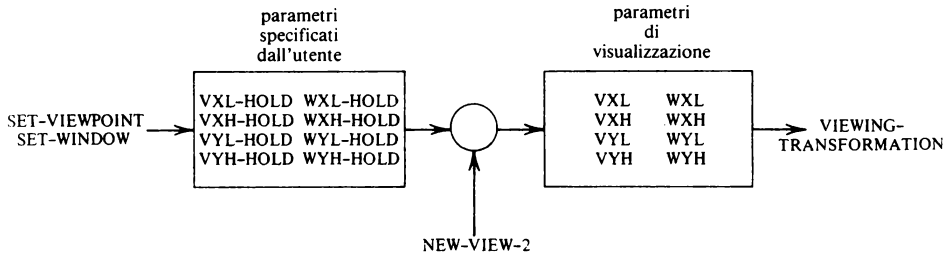


Figura 6.9 L'esecuzione della NEW-VIEW è necessaria per trasformare le specifiche dell'utente in parametri di visualizzazione

mentalmente alcuna ragione per cui un sistema non possa essere scritto in modo da permettere ad ogni momento la variazione del tipo di visualizzazione, tanto più che a volte quest'operazione può risultare utile (vedi Figura 6.9). L'algoritmo che verrà definito si attiene alle specifiche CORE e quindi dovrà copiare i valori specificati dell'utente nei parametri del sistema. La routine presentata di seguito calcola inoltre il fattore di scala per convertire le coordinate oggetto della window in coordinate della viewport.

Algoritmo 6.3 NEW-VIEW-2
Assegna il valore dei parametri di clipping e di visualizzazione partendo dalla window e dalla viewport specificate dall'utente.
Variabili globali WXL-HOLD, WYL-HOLD, WXH-HOLD, WYH-HOLD←parametri della window definiti dall'utente
VXL-HOLD, VYL-HOLD, VXH-HOLD, VYH-HOLD←parametri della viewport definiti dall'utente
WXL, WYL, WXH, WYH, VXL, VYL, VXH, VYH←parametri del clipping corrente
WSX, WSY←fattori di scala per la conversione da window a viewport

```
begin
   WXL←WXL-HOLD;
   WYL←WYL-HOLD;
   WXH←WXH-HOLD;
   WYH←WYH-HOLD;
   VXL←VXL-HOLD;
   VYL←VYL-HOLD;
   VXH←VXH-HOLD;
   VYH←VYH-HOLD;
   WSX←(VXH-VXL)/(WXH-WXL);
   WSY←(VYH-VYL)/(WYH-WYL);
   return;
end;
```

I valori assegnati per definire le dimensioni di una window devono essere applicati ad un intero segmento del display file. Questa restrizione può essere realizzata imponendo che la copia dei parametri descritta nell'algoritmo precedente avvenga quando si crea un segmento. D'altra parte, si dovrà modificare la routine di creazione di un segmento in modo da azzerare la trasformazione di visualizzazione per soddisfare l'ultima definizione dei parametri richiesta dall'utente.

Algoritmo 6.4 CREATE-SEGMENT (SEGMENT-NAME)
(Algoritmo 5.1 aggiornato)
Routine utente che crea il segmento indicato.
Argomenti SEGMENT-NAME←nome del segmento
Variabili globali NOW-OPEN←segmento correntemente aperto
FREE←indice della prima cella libera nel display file
SEGMENT-START, SEGMENT-SIZE, VISIBILITY, ANGLE, SCALE-X, SCALE-Y, TRANSLATE-X, TRANSLATE-Y←tabella dei segmenti
Costanti NUMBER-OF-SEGMENTS←dimensione della tabella dei segmenti

```
begin
   if NOW-OPEN>0
        then return-error 'SEGMENTO ANCORA APERTO';
   if SEGMENT-NAME<1
   or SEGMENT-NAME>NUMBER-OF-SEGMENTS
        then return-error 'NOME DEL SEGMENTO NON VALIDO';
   if SEGMENT-SIZE[SEGMENT-NAME]>0
        then return-error 'SEGMENTO GIÀ ESISTENTE';
   NEW-VIEW-2;
   SEGMENT-START[SEGMENT-NAME]←FREE;
   SEGMENT-SIZE[SEGMENT-NAME]←0;
   VISIBILITY[SEGMENT-NAME]←VISIBILITY[0];
   ANGLE[SEGMENT-NAME]←ANGLE[0];
   SCALE-X[SEGMENT-NAME]←SCALE-X[0];
   SCALE-Y[SEGMENT-NAME]←SCALE-Y[0];
   TRANSLATE-X[SEGMENT-NAME]←TRANSLATE-X[0];
   TRANSLATE-Y[SEGMENT-NAME]←TRANSLATE-Y[0];
   NOW-OPEN←SEGMENT-NAME;
   return;
end;
```

Le trasformazioni che si devono eseguire, come già detto, sono le seguenti: prima di tutto si deve eseguire una traslazione pari ai valori che definiscono la x minima e la y minima dei confini della window; quest'operazione sposta l'angolo inferiore sinistro della window sull'origine del sistema di riferimento. Quindi si deve modificare il fattore di scala di un valore pari al rapporto tra le dimensioni della viewport e quelle della window; ciò modifica le dimensioni della window riportandole in quelle della viewport. Infine si vuole traslare del valore delle coordinate x e y del limite inferiore della viewport, per muovere l'angolo inferiore sinistro dall'origine alla corretta posizione della viewport. Possiamo costruire ciascuna di queste matrici di trasformazione nel modo descritto nel Capitolo 4 e moltiplicare quindi tra loro le matrici, per formare una sola trasformazione, come nella (6.2), per poi applicarla ad un generico punto. Questo porta al seguente algoritmo per la trasformazione di visualizzazione.

Algoritmo 6.5 VIEWING-TRANSFORM (OP, X, Y)
Esegue la trasformazione di visualizzazione di un punto.
Argomenti OP, X, Y – istruzione del display file
Variabili globali WXL, WYL, WSX, WSY, VXL, VYL – parametri della window e della viewport
Variabili locali X1, Y1 – coordinate dal punto trasformato

```
begin
   X1←(X−WXL)*WSX+VXL;
   Y1←(Y−WYL)*WSY+VYL;
   PUT-POINT(OP, X1, Y1);
   return;
end;
```

Bisogna osservare che il precedente algoritmo, non solo esegue la trasformazione di visualizzazione, ma inserisce anche l'istruzione risultante nel display file. Il display file conterrà il modello dello spazio-immagine.

6.3 CLIPPING

Ora che si è visto come la figura può essere modificata nel fattore di scala e posizionata correttamente, si è in grado di considerare l'eliminazione dei segmenti di retta che compaiono al di fuori della window, così che solo quelli contenuti vengano visualizzati. Questo processo è chiamato *clipping*. L'operazione di clipping consiste nell'esaminare ciascuna porzione di retta appartenente al segmento del display file che deve essere visualizzato, per determinare se sia completamente all'interno della window, completamente al di fuori o se ne attraversa i confini. Nel primo caso il segmento di retta viene visualizzato; se si trova interamente all'esterno non si disegna nulla. Nel caso invece che attraversi i confini, si dovrà determinare il punto di intersezione e disegnare solo il tratto che giace all'interno dei confini (vedi Figura 6.10). Si osservi però che tipi di elementi grafici diversi richiedono differenti tecniche di clipping. Ad esempio, un carattere può essere incluso interamente od omesso a seconda che il suo centro stia o meno nella window: questa tecnica però non può essere utilizzata per le rette, così come un metodo valido per le rette può essere inefficace per applicare il clipping ai poligoni. Esistono diversi tipi di algoritmi disponibili; quello che verrà presentato nelle pagine seguenti è in grado di trattare i poligoni, i segmenti di retta e i caratteri. Esso si basa su di un metodo ideato da Sutherland e Hodgman (1974). L'idea che sta alla base di questo algoritmo è che l'operazione di clipping rispetto all'intera finestra può essere facilmente realizzata limitando ciascun segmento di retta rispetto ad uno solo dei confini della window alla volta e poi comporre i clipping parziali per ottenere il risultato complessivo. È possibile cioè realizzare il clipping complessivo attraverso i risultati parziali del clipping eseguito rispetto a ciascuna delle quattro rette che costituisco-

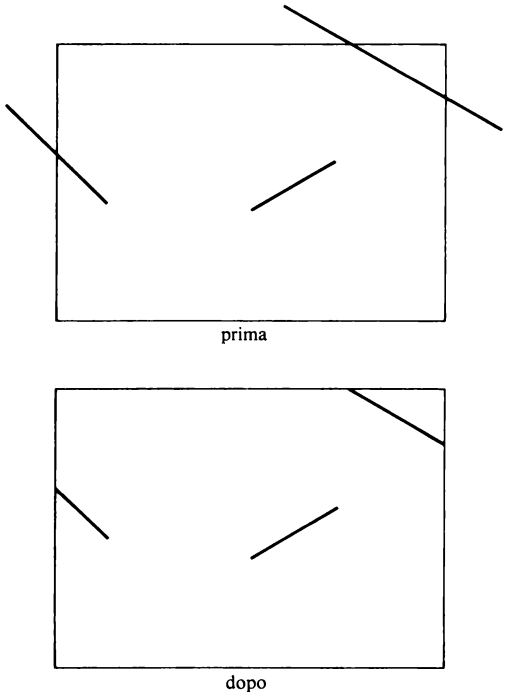


Figura 6.10 Clipping

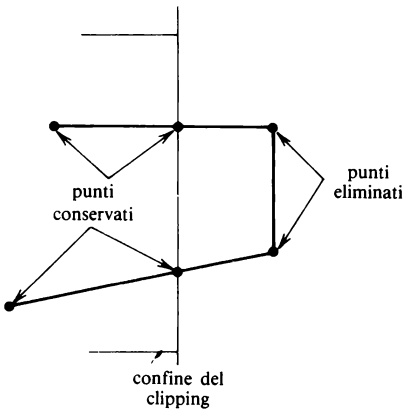


Figura 6.11 Clipping rispetto a un solo lato

no i confini della window. Per eseguire questa operazione di clipping rispetto ad uno solo dei confini, si dovranno percorrere le istruzioni del display file che descrivono il disegno: nel momento in cui vengono considerati gli estremi di un segmento di retta, si potrà decidere se essi appartengono ad una retta che attraversa il confine che si sta considerando e, se ciò accade, il punto di intersezione dovrà essere determinato e passato alla routine successiva. Attraverso quattro routine — ciascuna delle quali considera un solo confine — si otterrà alla fine il risultato completo: in ognuna di queste routine vengono esaminati tutti i punti della figura, controllando se essi giacciono all'interno del confine considerato. I punti estremi dei segmenti che giacciono all'interno della window e tutti i punti di intersezione tra le rette e i confini passano da ognuna di queste routine alle successive, mentre i punti che si trovano oltre i confini vengono «filtrati» (vedi Figura 6.11). In questo modo si esegue il clipping dell'intera figura rispetto a ciascuno dei confini della window prima di passare al confine successivo. Si osservi che questo metodo ha una caratteristica importante per quanto riguarda l'efficienza dell'algoritmo. Il processo di clipping avanza sequenzialmente attraverso le istruzioni di disegno della figura, in modo tale che è possibile iniziare il clipping rispetto al secondo confine della window prima che si sia conclusa l'operazione di clipping dell'intera figura rispetto al primo confine. Infatti, ciascun punto può essere elaborato attraverso tutte e quattro le routine di clipping e inserito nel display file prima che venga considerato il punto successivo (vedi Figura 6.12).

Di seguito vengono descritti gli algoritmi per il clipping di una figura rispetto a ciascuno dei quattro confini, con la seguente tecnica: si esaminano due punti per vedere se il segmento di retta con questi estremi attraversa il confine; se ciò è vero, viene eseguito un nuovo comando, che dipende dal punto eliminato. Se il segmento ha il suo punto iniziale fuori dalla window — e andrebbe quindi disegnato dall'esterno verso l'interno — oppure si tratta del punto centrale di un carattere, il comando verrà sostituito, eseguendo al suo posto un MOVE; se invece il punto non va eliminato, il comando che viene inserito nel display file è il medesimo di quello che si trovava precedentemente. Ciò significa che, se la figura esce dai limiti della window, la penna si muoverà lungo la frontiera fino al punto nel quale la figura rientra nella regione che deve essere disegnata.

Gli algoritmi aggiornano il punto corrente assegnandogli le coordinate del nuovo punto da considerare e decidono se il punto corrente si trova nella window. In caso affermativo anche questa istruzione viene eseguita e così di seguito per tutti i punti. In questo contesto si usa l'espressione «un comando viene eseguito» per intendere che il comando viene passato alla routine successiva. Per i primi tre algoritmi di clipping la «routine successiva» è quella che esegue l'algoritmo di clipping lungo il confine successivo. L'ultima routine manderà invece i comandi di disegno direttamente nel display file (vedi Figura 6.13).

Algoritmo 6.6 CLIP-LEFT (OP, X, Y)
Esegue il clipping rispetto al confine sinistro della window.
Argomenti OP, X, Y←istruzione del display file

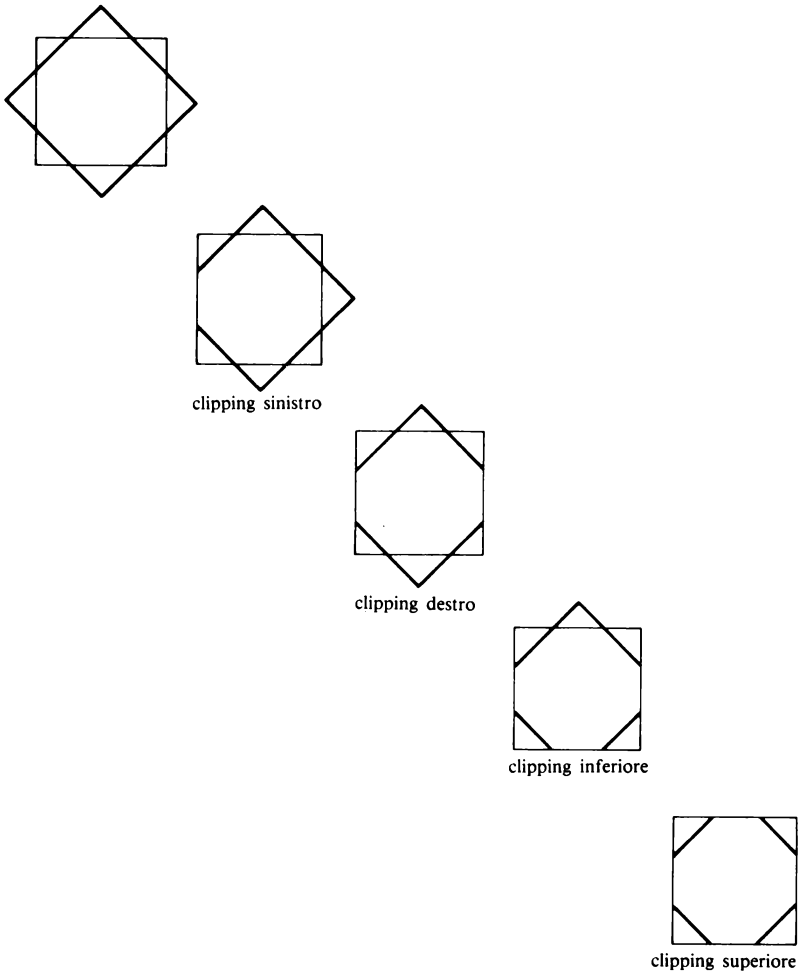


Figura 6.12 Clipping rispetto a tutti i confini della window

Variabili globali WXL←valore del confine di sinistra della window.
XS, YS←array per l'ultimo punto disegnato

begin

Caso in cui il segmento debba essere disegnato dall'esterno della window verso l'interno

if X≥WXL **and** XS[1]<WXL
then CLIP-RIGHT(1, WXL,
(Y−YS[1])*(WXL−X)/(X−XS[1])+Y);

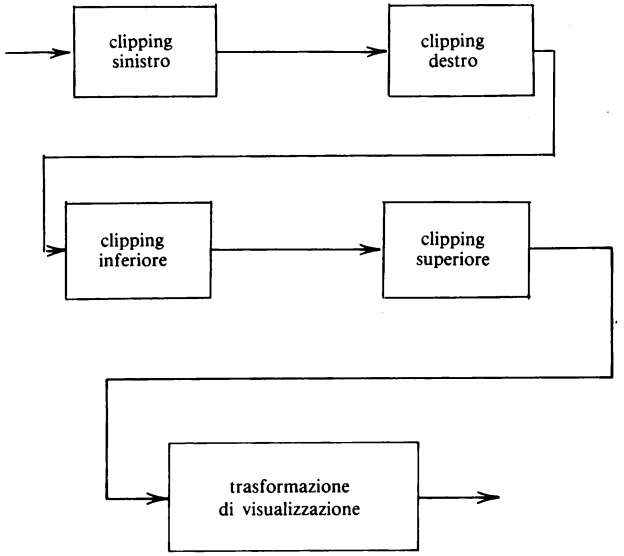


Figura 6.13 Procedura di clipping

```
    Caso in cui il segmento debba essere disegnato dall'interno della window
    verso l'esterno
    if X≤WXL and XS[1]>WXL
    then if OP<32
     then CLIP-RIGHT(OP, WXL,
          (Y-YS[1])*(WXL-X)/(X-XS[1])+Y)
     else CLIP-RIGHT(1, WXL,
          (Y-YS[1]*(WXL-X)/(X-XS[1])+Y)
    memorizza il punto da utilizzare come secondo estremo del successivo
    segmento di retta
    XS[1]←X;
    YS[1]←Y;
    Caso in cui il punto è interno
    if X≥WXL then CLIP-RIGHT(OP, X, Y);
    return;
end;
```

Le operazioni che vengono svolte all'interno della chiamata alla CLIP-RIGHT nella routine precedente consistono nella valutazione della coordinata *y* del

punto nel quale la retta interseca i confini della window. La coordinata x di questo punto è la posizione della retta che costituisce il confine della window.

Algoritmo 6.7 CLIP-RIGHT (OP, X, Y)
Esegue il clipping rispetto al confine destro della window.
Argomenti OP, X, Y←istruzione del display file
Variabili globali WXH←valore del confine di destra della window
XS, YS←ultimo punto disegnato

```
begin
   if X≤WXH and XS[2]>WXH
   then CLIP-BOTTOM(1, WXH,
        (Y-YS[2])*(WXH-X)/(X-XS[2])+Y);
   if X≥WXH and XS[2]<WXH
   then if OP<32
   then CLIP-BOTTOM(OP, WXH,
        (Y-YS[2])*(WXH-X)/(X-XS[2])+Y)
   else CLIP-BOTTOM(1, WXH,
        (Y-YS[2])*(WXH-X)/(X-XS[2])+Y);
   XS[2]←X;
   YS[2]←Y;
   if X≤WXH then CLIP-BOTTOM(OP, X, Y);
   return;
end;
```

Algoritmo 6.8 CLIP-BOTTOM (OP, X, Y)
Esegue il clipping rispetto al confine inferiore della window.
Argomenti OP, X, Y←istruzione del display file
Variabili globali WYL←valore del confine inferiore della window
XS, YS←ultimo punto disegnato

```
begin
   if Y≥WYL and YS[3]<WYL
   then CLIP-TOP(1, (X-XS[3])*(WYL-Y)/(Y-YS[3])+X, WYL);
   if Y≤WYL and YS[3]>WYL
   then if OP<32
   then CLIP-TOP(OP, (X-XS[3])*(WYL-Y)/(Y-YS[3])+X,
        WYL)
   else CLIP-TOP(1, (X-XS[3])*(WYL-Y)/(Y-YS[3])+X, WYL);
   XS[3]←X;
   YS[3]←Y;
   if Y≥WYL then CLIP-TOP(OP, X, Y);
   return;
end;
```

Algoritmo 6.9 CLIP-TOP (OP, X, Y)
Esegue il clipping rispetto al confine superiore della window.

```
Argomenti          OP, X, Y←istruzione del display file
Variabili globali  WYH←valore del confine superiore della window
                   XS, YS←ultimo punto disegnato
begin
   if Y≤WYH and YS[4]>WYH
   then SAVE-CLIPPED-POINT
        (1, (X-XS[4])*(WYH-Y)/(Y-YS[4])+X, WYH);
   if Y≥WYH and YS[4]<WYH
   then if OP<32
   then SAVE-CLIPPED-POINT
        (OP, (X-XS[4])*(WYH-Y)/(Y-YS[4])+X, WYH)
   else SAVE-CLIPPED-POINT
        (1, (X-XS[4])*(WYH-Y)/(Y-YS[4])+X, WYH);
   XS[4]←X;
   YS[4]←Y;
   if Y≤WYH then SAVE-CLIPPED-POINT(OP, X, Y);
   return;
end;
```

La routine SAVE-CLIPPED-POINT, che è stata usata per inserire i comandi nel display file, verrà descritta nel prossimo paragrafo.

6.4 CLIPPING DI POLIGONI

In questo paragrafo si estenderanno le routine di clipping in modo da poterle applicare anche ai poligoni oltre che ai segmenti di retta. Per questo, bisogna sapere cosa accade se un poligono attraversa i confini della window. La routine di clipping dovrà eliminare alcuni lati del poligono e inserirà dei comandi MOVE invece dei comandi DRAW in modo che, invece di disegnare dei segmenti di retta fuori dalla window, la penna si sposti lungo i confini. Ciò implica una variazione del numero di lati del poligono che deve riflettersi sulle istruzioni del display file per il disegno del poligono. In pratica si userà il comando MOVE per tracciare un lato invisibile del poligono, occupando un'istruzione che muove la penna invece di un'istruzione che disegna una retta. A causa del cambiamento del numero di lati, non si è in grado di sapere come sarà il comando che disegna l'intero poligono da inserire nel display file (se pure esiste) prima che sia stato valutato il clipping dell'intero poligono. Tutte queste istruzioni dovranno essere memorizzate in un'area di memoria temporanea. Quando tutti i lati sono «filtrati» attraverso il clipping, si sarà in grado di contare quanti lati sono rimasti, ricostruire l'appropriato comando per disegnare il poligono (che riassume le istruzioni che sono state salvate per i lati visibili) e inserirlo all'interno del display file. Le istruzioni che rimangono dopo l'esecuzione delle routine di clipping sono trattate in due modi diversi: quelle che non appartengono

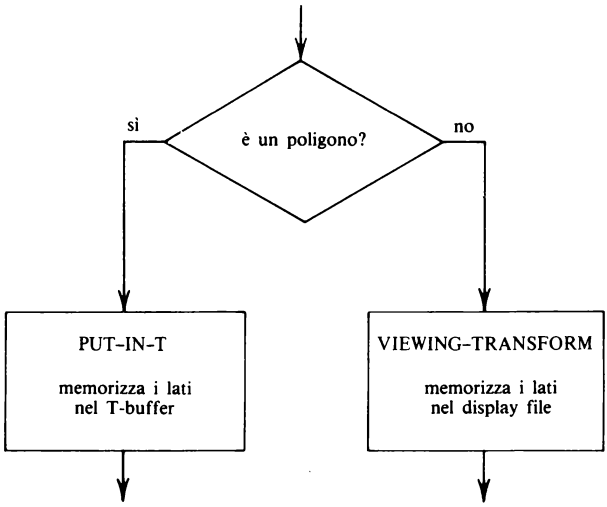


Figura 6.14 SAVE-CLIPPED-POINT

ad un poligono vengono inserite nel display file, dopo che gli è stata assegnata la trasformazione di visualizzazione mentre le istruzioni che corrispondono alla visualizzazione di un poligono vengono inserite in un buffer temporaneo. Questa decisione viene presa nell'algoritmo SAVE-CLIPPED-POINT, che si basa per la sua scelta sul valore dell'indicatore PFLAG che segnala che si sta eseguendo l'elaborazione di un poligono. L'algoritmo mantiene inoltre una traccia del numero di lati del poligono che sono stati salvati e che quindi dovranno comparire nel poligono risultante (vedi Figura 6.14).

Algoritmo 6.10 **SAVE-CLIPPED-POINT (OP, X, Y)**
Argomenti OP, X, Y←istruzione del display file
Variabili globali COUNT-OUT←contatore del numero di lati finali del poligono dopo il clipping
PFLAG←indicatore che segnala che si sta considerando un poligono

```
begin
   if PFLAG
   then begin
            COUNT-OUT←COUNT-OUT + 1;
            if COUNT-OUT < 33 then
            PUT-IN-T(OP, X, Y, COUNT-OUT);
            end
   else VIEWING-TRANSFORM(OP, X, Y);
   return;
end;
```

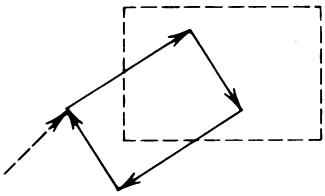
Nell'algoritmo PUT-IN-T gli array che forniscono la memoria temporanea per i lati dei poligoni sono stati indicati con IT, XT e YT. L'algoritmo precedente usa la routine PUT-IN-T per memorizzare le istruzioni in questi array.

Algoritmo 6.11 PUT-INT-T (OP, X, Y, INDEX)
Argomenti OP, X, Y←istruzione del display file
INDEX←posizione in cui memorizzare l'istruzione
Variabili globali IT, XT, YT←array di 32 locazioni per memorizzare temporaneamente i lati del poligono

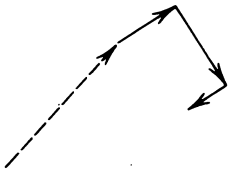
```
begin
   IT[INDEX]←OP;
   XT[INDEX]←X;
   YT[INDEX]←Y
   return;
end;
```

Per i poligoni è necessario scrivere una routine di clipping in più, che conti quanti lati del poligono originario sono stati considerati. Quando tutti questi lati sono stati considerati, ci si deve assicurare che il poligono sia chiuso. Il problema della chiusura sorge in quanto l'algoritmo inserisce un lato «invisibile» lungo il confine solo quando la penna rientra nella porzione visibile all'interno della window. Ma se accade che il punto iniziale del poligono si trova fuori dalla window, la chiusura del poligono richiederebbe il disegno dell'ultimo segmento all'esterno. Il punto iniziale viene però eliminato dal clipping e ciò significa che non verrà disegnato l'ultimo lato «invisibile» per chiudere il poligono (vedi Figura 6.15).
Per correggere questa situazione si deve eseguire il clipping su un ulteriore comando, quello che muove la penna nel primo punto visibile. Il risultato di quest'operazione è un lato che chiuderà il poligono, se necessario inserendo un'istruzione in più. L'istruzione superflua viene quindi rimossa decrementando il numero dei lati del poligono. Infine, il numero dei lati del poligono viene confrontato per vedere se si trova tra 3 e 31. Se è più grande di 31, non è possibile creare un'istruzione con il codice operativo adatto ad esso. Se è minore di 3, il poligono è degenerato in un segmento di retta, in un punto o addirittura rimane completamente invisibile. In questo caso non si dovrà inserire alcun comando nel display file. Se invece il nuovo poligono ha un numero accettabile di lati, è necessario aggiornare il comando che lo disegna. Si dovranno inoltre inserire le opportune coordinate x e y nel comando, così da disegnare sicuramente un poligono chiuso. Tutto questo viene eseguito dall'algoritmo POLYGON-CLIP.

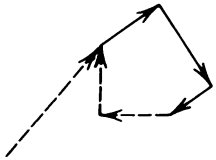
Algoritmo 6.12 POLYGON-CLIP (OP, X, Y)
Argomenti OP, X, Y←istruzione del display file
Variabili globali PFLAG←indicatore che segnala che si sta considerando un poligono



eseguendo il clipping
su tutti i lati...



...si ottiene un
poligono aperto...



...che deve essere
chiuso correttamente

Figura 6.15 Chiusura di un poligono

```
                    COUNT-IN←contatore del numero dei lati che devono
                      ancora essere esaminati
                    COUNT-OUT←contatore del numero dei lati finali del
                      poligono che vengono inseriti nel display file
                    IT, XT, YT←array di 32 locazioni per memorizzare tem-
                      poraneamente i lati di un poligono
Variabili locali    I←passo di scansione per i lati del poligono
begin
  COUNT-IN←COUNT-IN−1;
  CLIP-LEFT(OP, X, Y);
  if COUNT-IN≠0 then return;
  chiusura del poligono
  if COUNT-OUT>0 then CLIP-LEFT(2, XT[1], YT[1]);
  PFLAG←FALSE;
```

```
    rimozione del lato aggiunto per chiudere il poligono
    COUNT-OUT←COUNT-OUT−1;
    if COUNT-OUT<3 then return;
    if COUNT-OUT<32
       then begin
            inserimento del poligono nel display file
            VIEWING-TRANSFORM(COUNT-OUT,
                 XT[COUNT-OUT], YT[COUNT-OUT]);
            for I = 1 to COUNT-OUT
            do VIEWING-TRANSFORM(IT[I], XT[I], YT[I]);
            end
       else return-error 'POLIGONO TROPPO GRANDE';
    return;
end;
```

I comandi per i poligoni dovranno essere trattati in modo che, quando un poligono viene trovato, venga inserito nel file temporaneo e ai contatori opportuni vengano quindi assegnati i valori del numero di lati che devono essere considerati e il numero dei lati del poligono risultante. Le variabili che definiscono l'ultimo punto considerato verranno inizializzate per ciascuna delle routine di clipping; verrà poi assegnato il valore ad un flag in modo tale che le chiamate alle successive routine di clipping siano in grado di riconoscere che le informazioni che ricevono sono relative al lato di un poligono; in questo modo ogni routine può decidere se deve trattare un poligono o una delle altre primitive grafiche. Queste operazioni vengono effettuate dalla routine CLIP, che rappresenta la procedura di clipping più esterna.

Algoritmo 6.13 **CLIP (OP, X, Y)**

Argomenti OP, X ,Y←istruzione del display file

Variabili globali PFLAG←indicatore che segnala che si sta considerando un poligono
COUNT-IN←contatore del numero dei lati che devono ancora essere esaminati
COUNT-OUT←contatore del numero dei lati finali del poligono che vengono inseriti nel display file
XS, YS←ultimo punto disegnato

Variabili locali I←inizializzazione delle quattro routine di clipping

```
begin
  if PFLAG
     then POLYGON-CLIP(OP, X, Y)
     else if OP>2 and OP<31
     then begin
        PFLAG←TRUE;
        COUNT-IN←OP;
        COUNT-OUT←0;
```

```
            for I = 1 to 4
                begin
                XS[I]←X;
                YS[I]←Y;
                end;
        end
        else CLIP-LEFT(OP, X, Y);
    return;
end;
```

6.5 AGGIUNTA DEL CLIPPING AL SISTEMA

L'algoritmo CLIP esegue il clipping, trasforma opportunamente e memorizza nel display file le istruzioni rimanenti, che disegnano solo la parte presente nella window. Al fine di integrare questa nuova funzione con le routine che sono già state definite, sarà sufficiente includerla durante il processo di memorizzazione delle istruzioni nel display file. Per far questo, si deve modificare la routine DISPLAY-FILE-ENTER in modo che prelevi la posizione corrente della penna in coordinate dello spazio-oggetto e la inserisca nel display file passando attraverso la routine di clipping, trasformandola quindi in coordinate dello spazio-immagine (vedi Figura 6.16).

Algoritmo 6.14 DISPLAY-FILE-ENTER (OP)
(Algoritmo 2.23 aggiornato)
Combina le operazioni e le posizioni per formare un'istruzione che memorizza poi nel display file.

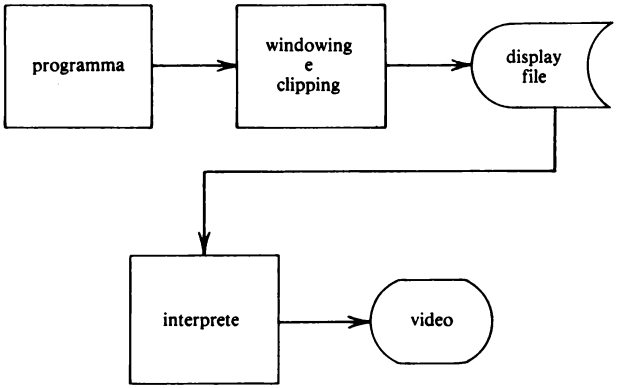


Figura 6.16 Estensione del sistema con le operazioni di windowing e clipping

```
Argomenti          OP←operazione che deve essere inserita
Variabili globali  DF-PEN-X, DF-PEN-Y←posizione corrente della penna
begin
   if OP<1
   then PUT-POINT(OP, 0, 0)
   else CLIP(OP, DF-PEN-X, DF-PEN-Y);
   return;
end;
```

Inoltre è necessario avere una routine di inizializzazione che assegna i valori iniziali dei confini della viewport e della window in modo tale che siano espressi in coordinate schermo normalizzate, cioè limitate tra 0 e 1 in entrambe le direzioni x e y. Ciò rende le trasformazioni per window e viewport trasparenti all'utente che non desidera usarle direttamente.

```
Algoritmo 6.15  INITIALIZE-6
Variabili globali  PFLAG←indicatore che segnala che si sta considerando
                     un poligono
                   XS, YS←posizioni limitate all'interno dei confini della
                     window
Variabili locali   I←inizializzazione delle quattro routine di clipping
begin
   INITIALIZE-5
   SET-VIEWPORT(0.0, 1.0, 0.0, 1.0);
   SET-WINDOW(0.0 ,1.0, 0.0, 1.0);
   NEW-VIEW-2;
   for I = 1 to 4
   do begin
      XS[I]←0;
      YS[I]←0;
      end;
   PFLAG←FALSE;
   return;
end;
```

6.6 EVITARE LE DIVISIONI

Nell'algoritmo di clipping è necessario calcolare il punto di intersezione fra le rette che limitano la window e i segmenti di retta che attraversano i suoi confini. Questo calcolo richiede un'operazione di divisione. Se si dispone di un calcolatore che è sprovvisto di hardware specializzato per eseguire quest'operazione, può accadere che le routine che la eseguono siano troppo lente. In questi casi, può essere utile un metodo alternativo per trovare il punto di intersezione.

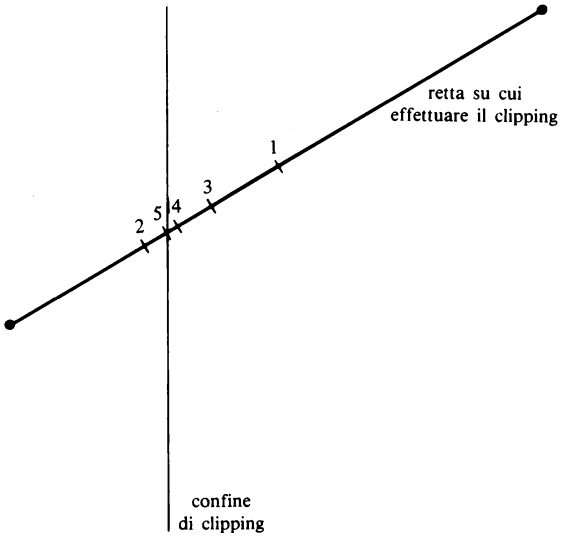


Figura 6.17 Uso del punto medio per calcolare l'intersezione di una retta con il confine di clipping

Il metodo che viene suggerito permette di trovare l'intersezione tra il segmento di retta e i confini della window per approssimazioni successive. La prima approssimazione è il punto medio del segmento; quindi si guarda se il punto medio si trova dentro o fuori dai confini, considerando la metà del segmento che contiene il confine della window e valutando di nuovo il punto medio di questo sottosegmento. Il processo si ripete finché si arriva sufficientemente vicino al punto cercato, il che significa a meno di un pixel dal confine (vedi Figura 6.17).

6.7 CLIPPING GENERALIZZATO

Fino ad ora sono state usate quattro routine di clipping distinte, una per ciascun confine della window; è evidente però che queste routine sono abbastanza simili tra loro: in pratica esse differiscono solo nel controllo che determina se un punto è all'interno o all'esterno dal confine della window. È possibile scrivere queste routine in una forma più generale, così che siano identiche e ricevano come parametro l'informazione relativa al confine rispetto al quale occorre eseguire il clipping. In un linguaggio di programmazione che permette la recursività, ciò significa che invece di avere quattro routine separate, si avrà bisogno

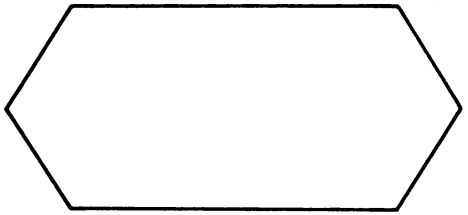


Figura 6.18 Una window con sei confini di clipping

di scriverne una sola. Essa richiamerà se stessa quattro volte (recursivamente), ogni volta passando la descrizione del confine specificato dagli opportuni parametri. Se si dispone della recursività, la routine di clipping può essere ulteriormente generalizzata per eseguire il clipping rispetto a una qualunque retta (non solo quelle verticali e orizzontali); ciò toglie all'algoritmo la limitazione al clipping entro confini rettangolari con i lati paralleli agli assi. Eseguire il clipping rispetto a una retta arbitraria permette ai lati della window di essere inclinati a piacere ed inoltre la window può avere così più di quattro lati. Infatti basterà chiamare recursivamente l'algoritmo di clipping, tante volte quanti sono i lati della window. L'algoritmo generalizzato in un linguaggio recursivo può quindi essere usato per eseguire il clipping entro un qualunque poligono convesso (vedi Figura 6.18).

6.8 POSIZIONE DI UN PUNTO RISPETTO A UNA RETTA ARBITRARIA

Si consideri brevemente il problema di decidere in quale dei due semipiani individuati da una retta si trovi un dato punto. Si supponga che la retta sia definita tramite i punti (x_1, y_1) e (x_2, y_2) e si ricordi che, come è stato spiegato nel Capitolo 1, un terzo punto (x, y) si trova sulla linea se vale l'equazione

$$(x-x_2)\,(y_1-y_2) = (y-y_2)\,(x_1-x_2) \tag{6.3}$$

Se invece l'equazione non è verificata, allora si deve concludere che il punto non si trova sulla retta. Se accade che l'espressione alla sinistra del segno di uguaglianza è maggiore della parte destra, cioè è valida la relazione

$$(x-x_2)\,(y_1-y_2) > (y-y_2)\,(x_1-x_2) \tag{6.4}$$

allora il punto si trova da una parte, se invece vale la relazione inversa

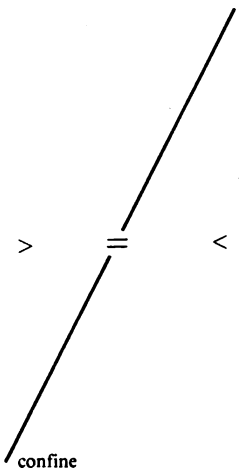


Figura 6.19 Decisione della posizione di un punto rispetto a una retta

$$(x-x_2)\,(y_1-y_2) < (y-y_2)\,(x_1-x_2) \tag{6.5}$$

allora il punto si trova dalla parte opposta (vedi Figura 6.19). Per scegliere quale dei due semipiani corrisponde alla relazione «maggiore di», bisogna sapere qual è il punto di coordinate (x_1, y_1).
Si consideri ad esempio la retta alla quale appartengono i punti $(x_1, y_1) = (1, 2)$ e $(x_2, y_2) = (4, 5)$. Un terzo punto di coordinate $(x, y) = (3, 4)$ giace sulla retta; infatti si verifica che

$$(3-4)\,(2-5) = (4-5)\,(1-4) \tag{6.6}$$

Il punto (2, 5) invece si trova nel semipiano in alto a sinistra rispetto alla retta; infatti accade che

$$(2-4)\,(2-5) > (5-5)\,(1-4) \tag{6.7}$$

Come conseguenza della scelta fatta per i punti (x_1, y_1) e (x_2, y_2), tutti i punti per i quali è verificata la (6.4) risulteranno stare dalla parte in alto a sinistra rispetto alla retta. Il punto (4, 3) è invece appartenente al semipiano in basso a destra delimitato dalla retta. È facile verificarlo osservando che la relazione tra i termini risulta essere

$$(4-4)\,(2-5) < (3-5)\,(1-4) \tag{6.8}$$

Questo confronto può essere usato nell'algoritmo di clipping per determinare se un punto si trova all'interno o all'esterno rispetto a una qualunque retta di confine.

6.9 WINDOWING MULTIPLO

Alcuni sistemi permettono l'uso di window multiple; ciò significa che una prima immagine può venire creata applicando più trasformazioni all'oggetto da visualizzare; quest'immagine viene poi inserita in una window per formarne una seconda. Si potranno eseguire altre trasformazioni per l'inserimento delle immagini in una window finché non venga ottenuta la figura desiderata. Ciascuna trasformazione di windowing permette all'utente si ritagliare una parte della figura e di riposizionarla in un'altra posizione dello schermo. In questo modo, utilizzando più window, l'utente ha la massima libertà nel comporre sullo schermo più particolari della figura. Lo stesso effetto può essere ottenuto, comunque, applicando un certo numero di trasformazioni su una singola window all'oggetto da visualizzare.

APPLICAZIONE PRATICA

Una delle più importanti applicazioni della Computer Graphics è la progettazione di circuiti integrati. Questi componenti miniaturizzati possono essere prodotti per mezzo di tecniche fotografiche a partire da disegni molto grandi del circuito. I disegni descrivono le aree occupate dai materiali conduttori, semiconduttori e isolanti. Opportune configurazioni geometriche di questi materiali costituiscono singoli diodi o transistor collegati in modo opportuno. La stesura dei disegni corretti di strutture tanto elaborate può essere un lavoro così complesso che la Computer Graphics rappresenta un aiuto indispensabile in queste applicazioni. Nella struttura di tali circuiti c'è di solito un elevato grado di regolarità e di ripetitività. Per questo risulterà molto utile la possibilità di poter riprodurre più volte una figura per mezzo di ripetute chiamate a un sottoprogramma che genera una sola volta l'immagine desiderata. Inoltre, il programma grafico che disegna il circuito può essere inserito in programmi più complessi, in grado di effettuare controlli sulla correttezza del circuito stesso. Il disegno completo di un circuito può avere dimensioni di 1 o 2 metri quadri. Se si riduce un tale disegno alle dimensioni del terminale del progettista, i dettagli appariranno confusi e la figura stessa risulterà illeggibile, sempre che si riesca veramente a visualizzarla interamente. Ciò di cui si ha bisogno è una window che mostri solo una parte del circuito, quella su cui il progettista sta lavorando. È possibile eseguire quest'operazione attraverso una chiamata alla routine SET-WINDOW. Se il progettista richiede di guardare contemporaneamente due differenti porzioni del circuito, la superficie dello schermo può esse-

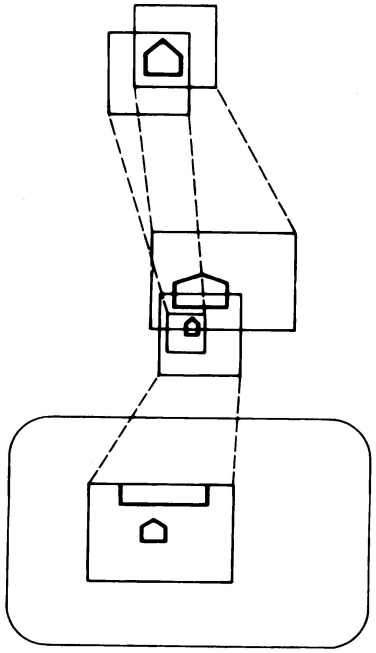


Figura 6.20 Windowing multiplo

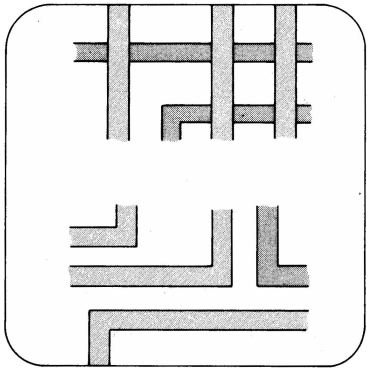


Figura 6.21 Uso di window e viewport per esaminare due zone di un circuito integrato

re divisa in due distinte viewport e le due porzioni del circuito saranno visualizzate attraverso le due window che le individuano. Il progettista può specificare un'associazione tra le coppie di window, aprire un segmento del display file, disegnare il circuito (eseguendo il clipping di tutte le rette tranne quelle desiderate), chiudere il segmento e quindi ripetere queste stesse operazioni per la seconda porzione di circuito che deve essere visualizzata (vedi Figura 6.21).

```
SET-WINDOW(20.0, 30.0, 40.0, 50.0);
SET-VIEWPORT(0.2, 0.8, 0.6, 1.0);
CREATE-SEGMENT(1);
DRAW-CIRCUIT;
CLOSE-SEGMENT;
SET-WINDOW(20.0, 30.0, 10.0, 20.0);
SET-VIEWPORT(0.2, 0.8, 0.0, 0.4);
CREATE-SEGMENT(2);
DRAW-CIRCUIT;
CLOSE-SEGMENT;
```

ESERCIZI

6.1 Nel caso che la viewport sia l'intero schermo (SET-VIEWPORT(0, 1, 0, 1)), quale sarà la matrice di trasformazione in coordinate omogenee per ciascuno dei seguenti assegnamenti?

a) SET-WINDOW (0, 1, 0, 1)
b) SET-WINDOW (0, 1, 0, 2)
c) SET-WINDOW (0, 3, 0, 1)
d) SET-WINDOW (2, 3, 0, 1)
e) SET-WINDOW (−10, 10, −2, 2)

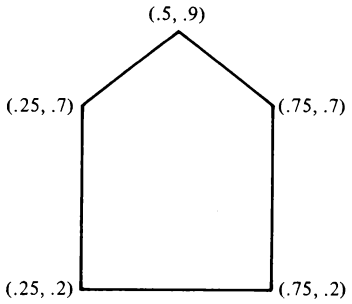
6.2 Se la window viene mantenuta come nell'assegnamento di default (SET-WINDOW(0, 1, 0, 1)), quale sarà la matrice di trasformazione in coordinate omogenee per ciascuna delle seguenti viewport?

a) SET-VIEWPORT (0, 0.5, 0, 1)
b) SET-VIEWPORT (0, 1, 0, 0,3)
c) SET-VIEWPORT (0.5, 1, 0, 1)
d) SET-VIEWPORT (0, 1, 0.25, 0.75)
e) SET-VIEWPORT (0.25, 0.75, 0.5, 1.0)

6.3 Per ciascuna delle seguenti coppie window-viewport, fornire la matrice di trasformazione in coordinate omogenee.

a) SET-VIEWPORT (0, 1, 0.25, 0.75)
   SET-WINDOW (0, 1, 0, 0.5)
b) SET-VIEWPORT (0, 0.5, 0, 0.5)
   SET-WINDOW (0, 2, 0, 4)
c) SET-VIEWPORT (0.5, 1, 0.5, 1)
   SET-WINDOW (10, 20, 0, 10)

6.4 Si consideri la seguente figura:



Si esegua uno schizzo di ciò che deve apparire sullo schermo per ciascuno dei seguenti assegnamenti della window e della viewport.

a) SET-VIEWPORT (0, 1, 0, 1)
   SET-WINDOW (0, 2, 0, 2)
b) SET-VIEWPORT (0, 0.5, 0, 0.5)
   SET-WINDOW (0, 1, 0, 1)
c) SET-VIEWPORT (0, 1, 0, 1)
   SET-WINDOW (0, 0.5, 0, 0.7)
d) SET-VIEWPORT (0.5, 1, 0.5, 1)
   SET-WINDOW (0, 1, 0, 1)
e) SET-VIEWPORT (0, 1, 0, 0.5)
   SET-WINDOW (0, 1, 0.5, 1)

6.5 Si supponga che una window abbia l'angolo inferiore sinistro nel punto di coordinate $(-2, -1)$ e l'angolo superiore destro si trovi invece nel punto (3, 2). Si indichi, per ciascuno dei segmenti definiti dalle seguenti coppie di punti, se essi sono totalmente visibili, totalmente nascosti o parzialmente visibili e in quest'ultimo caso dare le coordinate del punto di clipping.

a) $(-1, 0)$; (1, 1)
b) (1, 3); (1.6, 1)
c) $(-1.5, 0)$; $(-5, -2)$
d) $(-3, 1)$; (4, 1)
e) $(-2, 3)$; (1, 4)
f) $(-1, 3)$; (4, 0.5)
g) (2, 3); (4, 1.5)
h) (0, 1.5); (2.5, 0)

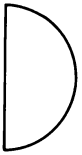
PROBLEMI DI PROGRAMMAZIONE

6.1 Si implementino gli algoritmi dal 6.1 al 6.14 estendendo il sistema grafico con le operazioni di definizione della window e della viewport.

6.2 a) Si costruisca una circonferenza di raggio 1 centrata nell'origine (0, 0), connetten-

do con segmenti di retta alcuni suoi punti (cos A, sin A) distanti un angolo non superiore a $\pi/15$.

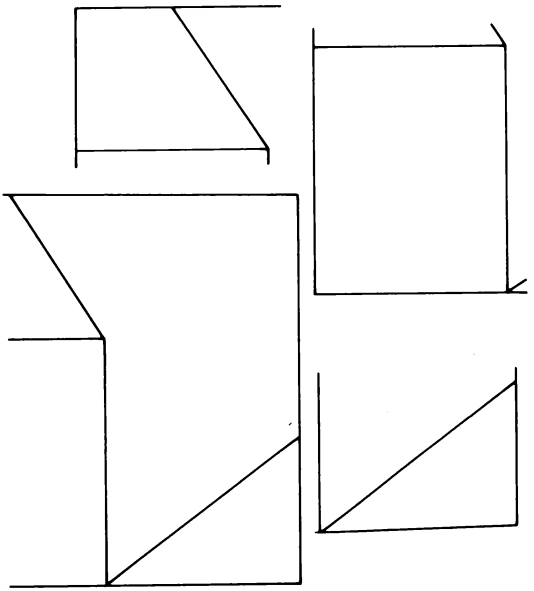
b) Usando una window sulla circonferenza, costruire il simbolo di una porta logica AND.



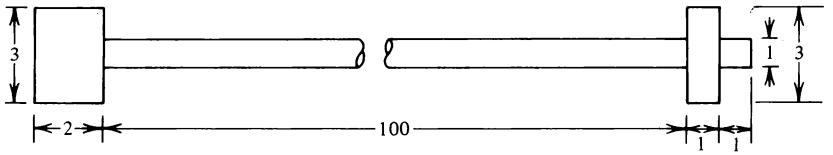
c) Usando tre finestre per la stessa circonferenza, costruire il simbolo per una porta logica OR.



6.3 Utilizzando delle window sulla mappa del problema di programmazione 3.5 si disegni la mappa di ciascuna delle regioni.



6.4 Si consideri l'asse molto lungo qui raffigurato:



a) Si tenti di disegnare l'intera figura sullo schermo utilizzando una window quadrata (più grande di 104) e un'opportuna viewport.
b) Si usi una window rettangolare più lunga che alta (simile al rettangolo che rappresenta l'ingombro dell'asse) in modo che la lunghezza venga compressa.
c) Si usino infine due window e due viewport per visualizzare solo gli estremi dell'asse.

***6.5** Si generalizzi l'algoritmo di clipping in modo che sia possibile eseguirlo con delle rette arbitrarie come confini. Si assuma che una retta di confine venga definita da quattro parametri x_1, y_1, x_2, y_2, passati all'algoritmo e si ricordi inoltre che un punto (x, y) è visibile quando

$$(x-x_2)\,(y_1-y_2) > (y-y_2)\,(x_1-x_2)$$

****6.6** Se il linguaggio di programmazione di cui si dispone permette la recursività, si formuli l'algoritmo di clipping generalizzato del problema di programmazione 6.4 in modo che le operazioni di clipping rispetto a ciascun confine della window possano essere eseguite come una serie di chiamate recursive della stessa routine. Tale routine permette di eseguire il clipping utilizzando poligoni convessi arbitrari come window.

7

Interazione fra macchina e utente

INTRODUZIONE

La Computer Graphics fornisce nuovi mezzi per la comunicazione tra l'utente e la macchina; le informazioni riguardanti la dimensione, la forma, la posizione e l'orientamento di figure si presentano all'utente in modo estremamente naturale. Strutture e relazioni complesse possono essere visualizzate in modo semplice e chiaro. La comunicazione deve essere però un processo bidirezionale: l'utente deve poter rispondere a questo tipo di informazioni e di messaggi. La forma più comune di messaggio proveniente dal calcolatore è una stringa di caratteri, stampata su un foglio di carta o sullo schermo di un terminale CRT. La corrispondente forma di risposta è una sequenza di caratteri proveniente da una tastiera o da una scheda perforata. Nella C.G. la forma dei risultati che il calcolatore è in grado di fornire viene estesa comprendendo le immagini bidimensionali composte da segmenti di retta, poligoni e archi, con differenti colori e intensità luminose. Un terminale grafico è in grado cioè di mostrare la posizione degli oggetti e le relazioni tra essi.

Ci si chiede quale sia la forma appropriata per la risposta dell'utente a messaggi di questo genere. Si risponderà a questa domanda considerando innanzitutto quali siano le categorie di risposte che l'utente deve essere in grado di fornire ai terminali grafici. L'utente, ad esempio, può aver bisogno di selezionare un particolare oggetto che compare sullo schermo, specificare la posizione di un punto sullo schermo, richiedere di modificare l'orientamento di un oggetto, la sua dimensione o la sua posizione. Egli dovrà anche essere in grado di inserire un nuovo oggetto. L'utilizzo di stringhe di caratteri introdotte attraverso la tastiera costituisce un metodo piuttosto innaturale per esprimere queste funzioni. L'utente, per esempio, non troverà comodo doversi determinare «manualmente» le coordinate di un certo punto rispetto a un sistema di riferimento, per poi inserirne i valori nel calcolatore sotto forma di cifre; preferirà piuttosto indicarne la posizione direttamente sullo schermo. Creare un disegno, inserendo le coordinate degli estremi di tutti i segmenti che lo compongono o fornendo le

istruzioni che muovono una penna, è un'operazione molto noiosa. La maggior parte degli utenti preferisce avere fisicamente in mano una penna che può muovere sullo schermo. Per questo motivo sono stati sviluppati alcuni metodi adatti ad inserire in questo modo le informazioni di tipo grafico. Gli utenti cioè, muovendo uno stilo simile ad una penna, possono veder comparire sullo schermo le linee tracciate come se usassero realmente una penna ad inchiostro. Un altro dei metodi che verranno descritti permette all'utente di «appendere» una parte dell'immagine allo stilo per poi riposizionarla muovendo lo stilo stesso. È possibile poi alterare il fattore di scala o l'orientamento delle figure premendo pulsanti o inclinando leve, oppure selezionare una parte della figura indicandola direttamente sullo schermo. In questo capitolo verranno considerati tutti i diversi tipi di dispositivi che permettono all'utente di fornire informazioni grafiche in modo naturale ed alcune tecniche che avranno lo scopo di rendere efficiente l'interazione tra l'uomo e la macchina.

7.1 HARDWARE

Per permettere all'utente di interagire in modo grafico, sono stati sviluppati molti e diversi tipi di dispositivi hardware. Questi dispositivi si possono suddividere in due classi: i sensori di posizione (*locator*) e i selettori (*selector*). I primi forniscono le informazioni relative alla posizione: il calcolatore riceve da un locator le coordinate di un punto che corrisponde ad una certa posizione sullo schermo. I selettori, invece, sono usati per indicare una particolare entità di tipo grafico; un selettore è in grado di selezionare una particolare entità attraverso un'immagine che compare sullo schermo, ma non fornisce alcuna informazione sulla posizione in cui si trova l'entità stessa.
Si analizzeranno dapprima alcuni locator. Un esempio è costituito da una coppia di rotelle (*paddle*) come quelle che si tovano sul terminale grafico Tektroniks 4010. Si tratta di due potenziometri montati sulla tastiera, regolabili dall'utente. Uno è usato per indicare un movimento nella direzione x, l'altro viene usato per il movimento nella direzione y. Due convertitori analogico-digitali trasformano la posizione fornita dai potenziometri in valori digitali che possono essere letti dal calcolatore. La lettura di entrambi i potenziometri fornisce le coordinate di un punto. Perché questo schema sia utile bisogna fornire all'utente l'informazione grafica che corrisponde al punto che sta indicando con i potenziometri. È necessario, cioè, un meccanismo di retroazione che può essere realizzato per mezzo di uno speciale simbolo posto sullo schermo nel punto che viene individuato. Allo stesso scopo si può anche utilizzare una coppia di rette parallele agli assi dello schermo che si intersecano nel punto indicato. Quando un potenziometro viene ruotato, il simbolo speciale o una delle rette citate si muove attraverso lo schermo mostrando all'utente qual è la posizione ricevuta dal calcolatore (vedi Figura 7.1).
Un altro dispositivo di posizionamento è il *joystick*. Un joystick è dotato di

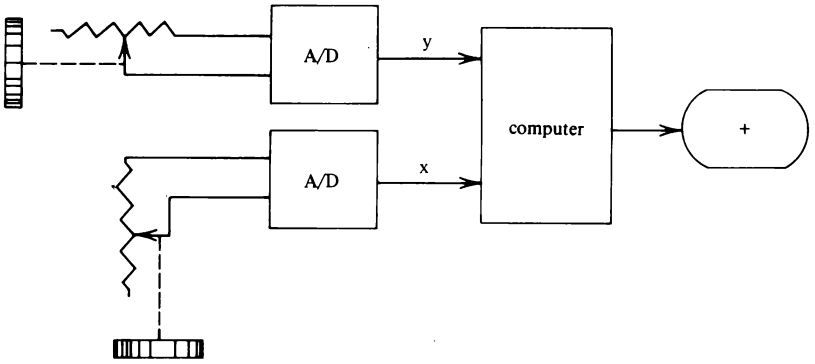


Figura 7.1 Inserimento di coordinate posizionali tramite dei potenziometri a rotella

due potenziometri, proprio come nel caso precedente, che però sono vincolati ad una sola leva. Muovendo la leva in avanti o indietro, cambia la posizione di un potenziometro, mentre muovendola a sinistra o a destra cambia la posizione dell'altro. Con un joystick si possono modificare simultaneamente i valori di posizione sia della coordinata x che quelli della y tramite lo spostamento di una sola leva. Le posizioni sul potenziometro vengono elaborate allo stesso modo di quelle delle rotelle; diversamente dal primo caso, un joystick ritorna nella posizione zero non appena la leva viene abbandonata, mentre le rotelle rimangono nell'ultima posizione a loro assegnata finché non vengano nuovamente ruotate. I joystick sono poco costosi e sono piuttosto comuni nelle applicazioni in cui sia richiesto soltanto un posizionamento approssimativo dei punti sullo schermo (vedi Figura 7.2).

Se si dovesse memorizzare all'interno di un calcolatore un disegno tracciato su di un foglio di carta o su di un'eliocopia, si troverebbe che per questa operazione di digitalizzazione il joystick non è molto pratico. Con il joystick si è in grado di indicare una posizione sullo schermo, ma non è possibile far corrispondere precisamente la posizione dello schermo alla corrispondente posizione del foglio sul quale compare il disegno. Per applicazioni di questo tipo serve un dispositivo detto *digitalizzatore* o *tavoletta grafica*, costituita da una superficie piana e da uno stilo simile ad una penna oppure di un cursore per tavoletta dotato di mirino (vedi Figura 7.3). La tavoletta è in grado di «sentire» la posizione dello stilo o del cursore quando essi vengono appoggiati sulla sua superficie. Per rilevare la posizione dello stilo sono stati impiegati numerosi e differenti principi fisici: la maggior parte di essi non richiede un contatto diretto fra lo stilo e la superficie della tavoletta, in modo tale che sia possibile porre un disegno o un'eliocopia su di essa ed usare lo stilo per indicare i punti che compaiono sul disegno. Non è necessario un meccanismo di retroazione sullo schermo per vedere dove si trovano i punti indicati, come nel caso del joystick, poiché l'utente può guardare la tavoletta per controllare quale posizione sta indicando.

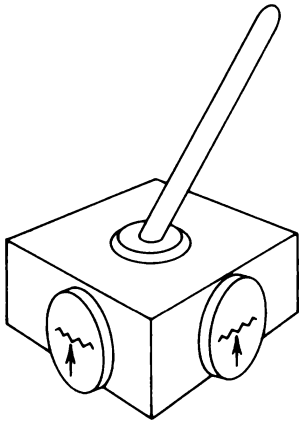


Figura 7.2 Joystick

Nonostante ciò, nel caso in cui i dati in ingresso siano correlati con elementi già presenti sullo schermo, può tornare utile disporre di una forma di retroazione come, ad esempio, un cursore.
Si osservi però che l'utente, in generale, non vuole inserire nel calcolatore la posizione di tutti i punti indicati durante gli spostamenti dello stilo; qualche volta vorrà muovere lo stilo solamente al fine di scegliere e raggiungere la posizione di un nuovo punto. Occorre quindi un meccanismo per attivare la tavoletta e per disattivarla, cosicché il calcolatore potrà leggere i valori delle coordinate del punto indicato solo quando l'utente lo desidera. Un modo semplice e conveniente per fare ciò consiste nel porre un interruttore sulla punta dello stilo in modo tale che si chiuda solo quandò la punta viene premuta. L'utente può quindi sollevare lo stilo, posizionarlo e premerne la punta per inserire le coordinate del punto indicato.

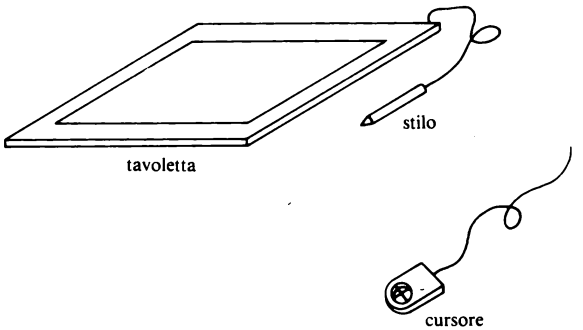


Figura 7.3 Tavoletta grafica

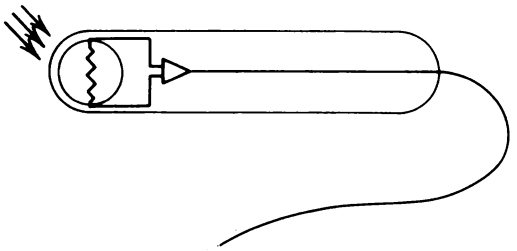


Figura 7.4 Penna ottica

Un dispositivo selettore può essere costituito ad esempio da una penna ottica, costituita da una fotocellula montata sulla punta di uno stilo (vedi Figura 7.4). Questa penna può essere usata per indicare i punti dello schermo di un terminale a vector refresh. La penna è in grado di generare un impulso, ogni volta che i fosfori di fronte alla fotocellula vengono illuminati. L'immagine visualizzata sul terminale, che all'occhio appare stabile, in realtà lampeggia ad una frequenza maggiore di quella che l'occhio è in grado di percepire. Le variazioni di luminosità a questa frequenza sono però riconoscibili dalla penna ottica che può essere utilizzata, quindi, per determinare il momento in cui il fosforo sotto di essa viene illuminato. Si ricordi che l'immagine viene tracciata da un unico pennello elettronico, che percorre lo schermo del terminale illuminando i punti che compongono la figura appartenenti ad una sola linea di scansione alla volta. Non è quindi possibile che sullo schermo vengano tracciate simultaneamente due linee. Quando la penna ottica viene eccitata dai fosfori colpiti dal pennello elettronico, viene trasmesso un segnale di interruzione al processore grafico che sta interpretando il display file. All'interno del processore grafico il registro istruzione ha la funzione di memorizzare l'istruzione del display file che deve essere interpretata: nel momento in cui sopraggiunge il segnale generato dalla penna ottica, si interrompe il corso normale di interpretazioni delle istruzioni e si memorizza il valore contenuto nel registro istruzione. Quindi, dopo aver estratto quest'informazione fornita dalla penna ottica, il processore può riprendere correttamente l'interpretazione del display file. In questo modo la penna ha individuato l'istruzione del display file che era in esecuzione, permettendo così di stabilire cosa essa indicava; infatti, determinando la parte della figura contenente l'istruzione memorizzata nel momento in cui è partito un impulso dalla penna, è possibile scoprire quale oggetto l'utente stava indicando in quel momento. Spesso è anche possibile abilitare o disabilitare questo meccanismo di interruzione e quindi fare in modo che le entità visualizzate sullo schermo siano selezionabili oppure no.

Una penna ottica è un esempio di dispositivo asincrono, cioè controllato da eventi che si svolgono in un momento qualunque (*event-driven*). Diversamente da quanto accade per un locator, che può essere campionato in ogni momento, in questo caso il processore deve rimanere in attesa che la penna incontri un fo-

sforo iluminato sullo schermo: il calcolatore deve attendere finché tale evento non si verifichi per poter ottenere l'informazione desiderata. La tastiera è l'esempio più tipico di dispositivo asincrono; infatti il calcolatore rimane in attesa che venga premuto un tasto per poter acquisire il carattere scelto dall'utente.
Altri dispositivi asincroni disponibili sono i pulsanti e gli interruttori. La gestione dei dispositivi di tipo asincrono può essere svolta in due modi distinti: il primo consiste nell'esecuzione di un *polling loop*, che è un ciclo all'interno del quale viene controllato lo stato del dispositivo asincrono finché questo non trasmette un segnale. Nel momento in cui si verifica questo evento, il ciclo viene interrotto riportando come risultato le informazioni relative al tipo di evento associato al segnale (vedi Figura 7.5). In molti linguaggi ad alto livello, la lettura dei caratteri provenienti dalla tastiera viene realizzata in questo modo. Lo svantaggio principale di questo schema di comportamento è che il processore resta inutilizzato mentre si è in attesa di un evento. Questo fatto può essere addirittura disastroso per il buon funzionamento del sistema nel caso che un unico processore debba eseguire il ciclo di attesa ed anche il refresh dell'immagine sullo schermo.
Un metodo alternativo a questo consiste nell'inviare un segnale di *interruzione* ogni volta che si verifica un evento asincrono. Un'interruzione è un segnale

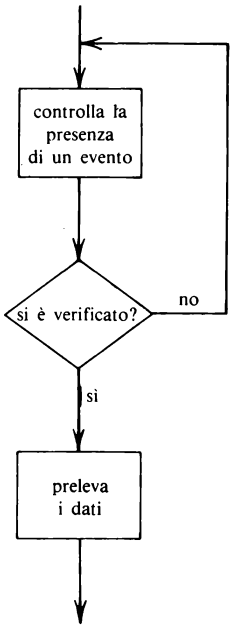


Figura 7.5 Un ciclo di polling

elettrico che blocca l'esecuzione della sequenza normale delle istruzioni e trasferisce il controllo ad una particolare routine di gestione dell'interruzione. Usando questo meccanismo, il processore è libero di eseguire qualunque operazione mentre è in attesa di un evento. Ogni volta che un'interruzione segnala un evento asincrono, l'elaborazione viene sospesa e viene eseguita la routine che gestisce quel particolare tipo di evento. Al termine di questa routine l'elaborazione può continuare regolarmente (vedi Figura 7.6).
Si supponga, ad esempio, di usare un programma che visualizza una figura e quindi, in un momento successivo, richiede all'utente delle informazioni da tastiera. L'utente esperto è in grado di fornire anticipatamente i caratteri di risposta, prima che il sistema abbia finito di elaborare e visualizzare la figura. Questi caratteri andrebbero persi se si usasse la tecnica di polling, in quanto il calcolatore, mentre sta visualizzando la figura, non è disponibile a riceverli. At-

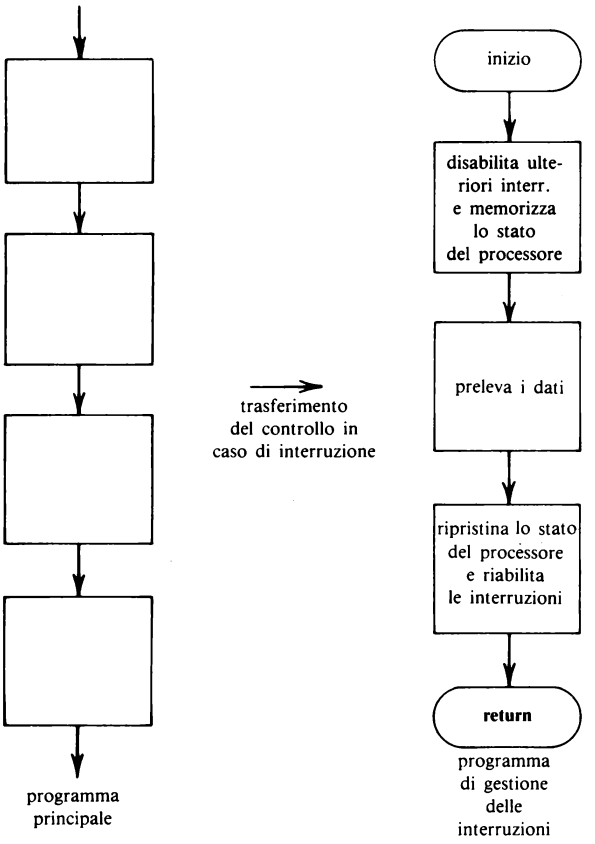


Figura 7.6 Schema di interruzione per raccogliere i dati di input

traverso il metodo delle interruzioni il calcolatore può invece sospendere temporaneamente il programma, memorizzare i caratteri introdotti dalla tastiera per poi riesaminarli al momento opportuno. Il trattamento degli eventi asincroni deve essere scomposto in due momenti: quando si verifica l'evento, le informazioni ad esso associate vengono inserite in fondo ad una coda, poiché, prima che il programma sia disponibile per rispondere all'interruzione, possono verificarsi numerosi eventi che dovranno essere tutti memorizzati per essere reperibili nel momento opportuno. Quando poi si è in grado di trattare l'informazione proveniente dal dispositivo di tipo asincrono, allora si dovrà cercare all'interno della coda il primo evento in attesa di essere trattato. Nel caso che esso sia presente nella coda, l'informazione relativa viene rimossa dalla coda ed elaborata. Se invece la coda è vuota, significa che non si è verificato alcun evento; può essere necessario eseguire a questo punto un ciclo di polling per controllare ripetutamente lo stato della coda in modo equivalente a quello che si esegue per controllare lo stato di un certo dispositivo.
Esistono differenti tipi di dispositivi asincroni in grado di inviare un segnale di interruzione; i dati in ingresso relativi ai diversi dispositivi possono avere forme o dimensioni diverse. Per questo può essere utile non salvare le informazioni relative all'evento direttamente in una coda, ma in qualche altra area di memoria, utilizzando invece la coda per memorizzare solamente un puntatore, che indica ove trovare queste informazioni. In alternativa, si possono avere più code specializzate a contenere le informazioni relative a ciascuna classe di dispositivi. Riassumendo, quindi, si può dire che la ricerca delle informazioni relative ad un evento si svolge in due fasi. Innanzitutto viene esaminata la coda per determinare se è presente un evento e quindi si recuperano le informazioni che in essa erano state memorizzate al momento dell'interruzione. Il primo passo è quindi la ricerca dell'evento, mentre il secondo è l'utilizzo delle informazioni associate ad esso.

7.2 ALGORITMI DI GESTIONE DEI DISPOSITIVI DI INPUT

Questo paragrafo tratta alcuni argomenti che dipendono fortemente dai particolari dispositivi hardware disponibili. Come viene realizzata nel sistema CORE l'indipendenza dai dispositivi? Ciò avviene specificando le routine disponibili e le loro funzioni ma non come sono state implementate. Il sistema si presenta cioè come un'interfaccia tra un programma applicativo indipendente dai dispositivi e i particolari mezzi hardware disponibili. La struttura interna delle routine di interfacciamento dipenderà dall'hardware, mentre all'esterno esse appariranno all'utente sempre nello stesso modo, con la stessa funzionalità e nella medesima forma. Questo modo di presentare le funzionalità del sistema è un vantaggio per l'utente, ma costituisce una difficoltà per chi deve costruire il sistema, poiché non si possono scrivere algoritmi generali. Per questo motivo, le

routine di questo capitolo si presenteranno meno precise di quelle precedenti, stabilendo solo in termini generali ciò che gli algoritmi devono fare. Se non si è in possesso di dispositivi per l'interazione grafica, questi possono essere simulati utilizzando solo la tastiera; in questo caso la struttura interna delle routine varia però in modo considerevole rispetto al caso generale. Ciò dipende da due fattori: innanzitutto gli ingressi provenienti da una tastiera attraverso un linguaggio ad alto livello vengono letti per campionamento invece che attraverso un meccanismo di acquisizione asincrono. In ogni momento può essere eseguita una istruzione **read** che restituisce sempre un valore. Durante l'esecuzione il programma resta realmente sospeso, fino al momento in cui vengono introdotte tutte le informazioni in ingresso. Per poter agire in modo simile a un dispositivo asincrono, bisogna che l'istruzione di lettura venga completata, sia che ci siano nuove informazioni nel buffer di ingresso, sia che non ve ne siano; purtroppo, la maggior parte dei linguaggi ad alto livello che si utilizzano per implementare queste routine, non si comporta in questo modo. La seconda differenza riguarda il fatto che si assumerà di avere un solo dispositivo di interazione e non verranno affrontati problemi come la risposta a due eventi simultanei.

Considerando la forma generale delle routine di gestione dei dispositivi di input, si esamineranno cinque classi di dispositivi. Il primo tipo che verrà trattato è il *pulsante*; esso è un dispositivo asincrono che produce un segnale di interruzione quando viene premuto, senza fornire nessun'altra informazione associata. La seconda classe di dispositivi è costituita da quelli di *pick* (indicatori): un tipico esempio è la penna ottica. Si tratta anche in questo caso di un dispositivo di tipo asincrono, che serve per selezionare una porzione dello schermo. La terza classe di dispositivi è la *tastiera*, mentre la quarta è la classe dei *locator*. Un locator è un dispositivo a campionamento che restituisce due numeri, corrispondenti alle coordinate di una certa posizione dello schermo. Infine verranno descritti i dispositivi di tipo *valuator* che, analogamente ad un locator, sono gestiti per mezzo del campionamento, ma restituiscono come risultato un solo valore. Un valuator può essere realizzato con un potenziometro o una manopola controllabile dall'utente. Non c'è motivo per cui un sistema non possa avere parecchi pulsanti, penne ottiche, tastiere, locator o valuator. Per identificare un particolare dispositivo hardware tra quelli di cui si è in possesso, si dovrà perciò indicarne non solo la classe ma anche, con precisione, di quale si tratta. Il sistema grafico proposto dovrà anche disporre di un meccanismo per accendere o spegnere logicamente questi dispositivi, realizzato attraverso una routine che permette all'utente di attivare o disattivare ciascun dispositivo in modo che i dati da esso provenienti — se ce ne sono — vengano eventualmente ignorati. La descrizione delle routine riguardanti la grafica interattiva inizierà considerando le operazioni di attivazione e disattivazione dei dispositivi. Queste operazioni verranno descritte nel caso più semplice, in cui il procedimento di attivazione è applicato alle classi di dispositivi e non a ciascuno di essi singolarmente. Ad ogni classe verrà associato un flag, il cui valore stabilisce se quella classe è attiva o meno. Per facilitare l'individuazione delle varie classi, è possibile identificarle attraverso dei numeri. La classe 1 rappresenta un pulsante, la 2 un

pick, la classe 3 una tastiera, la classe 4 un locator e la 5 un valuator. La routine di disattivazione ha come argomento il numero della classe che deve essere disattivata e svolgerà qualsiasi azione necessaria per spegnerla logicamente, ponendo il valore FALSE nel flag corrispondente. Analogamente, la routine di attivazione, preso il numero della classe, svolgerà tutte le azioni necessarie ad accenderla, per poi assegnare il valore TRUE al flag corrispondente a quella classe.

Algoritmo 7.1 ENABLE-GROUP (CLASS)
Attiva una classe di dispositivi di input.
Argomenti CLASS←codice di identificazione della classe che deve essere attivata
Variabili globali BUTTON, PICK, KEYBOARD, LOCATOR, VALUATOR←segnalatori dei dispositivi

```
begin
   if CLASS = 1
      then begin
           OPERAZIONI NECESSARIE A PERMETTERE
              L'INPUT DA PULSANTI;
           BUTTON←TRUE;
           end;
   if CLASS = 2
      then begin
           OPERAZIONI NECESSARIE A PERMETTERE
              L'INPUT DA DISPOSITIVI DI PICK;
           PICK←TRUE;
           end;
   if CLASS = 3
      then begin
           OPERAZIONI NECESSARIE A PERMETTERE
              L'INPUT DA TASTIERA;
           KEYBOARD←TRUE;
           end;
   if CLASS = 4
      then begin
           OPERAZIONI NECESSARIE A PERMETTERE
              L'INPUT DA LOCATOR;
           LOCATOR←TRUE;
           end;
   if CLASS = 5
      then begin
           OPERAZIONI NECESSARIE A PERMETTERE
              L'INPUT DA VALUATOR;
```

```
            VALUATOR←TRUE;
            end;
   return;
end;
```

Algoritmo 7.2 DISABLE-GROUP (CLASS)
Disattiva una classe di dispositivi di input.
Argomenti CLASS←codice di identificazione della classe che deve essere disattivata
Variabili globali BUTTON, PICK, KEYBOARD, LOCATOR, VALUATOR←segnalatori dei dispositivi

```
begin
   if CLASS = 1
      then begin
           OPERAZIONI NECESSARIE A DISABILITARE I
             PULSANTI;
           BUTTON←FALSE;
           end;
   if CLASS = 2
      then begin
           OPERAZIONI NECESSARIE A DISABILITARE I
             DISPOSITIVI DI PICK;
           PICK←FALSE;
           end;
   if CLASS = 3
      then begin
           OPERAZIONI NECESSARIE A DISABILITARE LE
             TASTIERE;
           KEYBOARD←FALSE;
           end;
   if CLASS = 4
      then begin
           OPERAZIONI NECESSARIE A DISABILITARE I
             LOCATOR;
           LOCATOR←FALSE;
           end;
   if CLASS = 5
      then begin
           OPERAZIONI NECESSARIE A DISABILITARE I
             VALUATOR;
           VALUATOR←FALSE;
           end;
   return;
end;
```

È possibile anche fornire all'utente una sola routine per disattivare tutti i dispositivi di input.

Algoritmo 7.3 DISABLE-ALL
Disattiva tutti i dispositivi di input.
Variabili locali CLASS←contatore delle possibili classi di dispositivi
begin
for CLASS = 1 **to** 5 **do** DISABLE-GROUP(CLASS);
return;
end;

7.3 TRATTAMENTO DEGLI EVENTI ASINCRONI

Un dispositivo asincrono viene definito come un generatore di interruzioni: ogni volta che l'elaboratore rileva un segnale di interruzione proveniente da un dispositivo di questo tipo, viene interrotta l'esecuzione dell'attività che è in corso e vengono trattate opportunamente le informazioni associate al dispositivo. Al termine di queste operazioni si ritorna al normale processo di elaborazione. Quali operazioni sono necessarie per rispondere ad una interruzione? Di seguito verrà presentata una proposta per un algoritmo che svolga queste funzioni. Si supponga per semplicità di lavorare con una sola coda per gli eventi, anche se, in generale, potrebbero esserci più code. Rispondere ad un'interruzione significa identificare quale dispositivo l'ha causata e ottenere, da quel dispositivo, i dati associati. Queste informazioni devono essere poi memorizzate nella coda, tranne le stringhe che sono trattate in maniera differente poiché necessitano di una quantità di memoria maggiore di ogni altro dispositivo e di dimensioni variabili. Ogni stringa di caratteri viene memorizzata in un'area di memoria riservata, mentre nella coda viene inserito solo il puntatore che indica dove inizia l'area di memoria che contiene i caratteri.

Algoritmo 7.4 EVENT
Questo algoritmo è un esempio di trattamento delle informazioni che provengono dal dispositivo di input che ha generato un'interruzione.
begin
DISABILITA LE INTERRUZIONI; per impedire l'interruzione del processo di interruzione
SALVA LO STATO DEL PROCESSORE;
DETERMINA QUALE DISPOSITIVO (CLASSE E NUMERO)
HA PROVOCATO L'INTERRUZIONE;
if IL DISPOSITIVO È ABILITATO
then begin
if ALL'EVENTO È ASSOCIATA UNA STRINGA DI CARATTERI
then begin
PRELEVA LA STRINGA IN INPUT;
ADD-STRING-Q(STRING, DATA);
end

```
    else PRELEVA I DATI ASSOCIATI AL DISPOSITIVO;
    ADD-EVENT-Q(CLASS, NUMBER, DATA);
    end;
    RIPRISTINA LO STATO DEL PROCESSORE:
    RIABILITA LE INTERRUZIONI;
    return;
end;
```

La struttura dati utilizzata dall'algoritmo EVENT è una coda di tipo *FIFO* (first-in-first-out, cioè primo-arrivato-primo-servito) che viene gestita in un modo che ricorda il comportamento di una fila di persone in attesa di acquistare un biglietto ad uno sportello: il primo della fila è il primo ad essere servito; quando arrivano altre persone, si mettono in coda alla fila. Nel sistema proposto questa struttura è implementata mediante una serie di array per contenere i dati e utilizzando due puntatori. Il primo di essi è QREAR, che indica la posizione successiva all'ultimo elemento dalla coda; il valore di questo puntatore viene incrementato ogni volta che si presenta un nuovo evento: se durante il ciclo di incremento si raggiunge l'ultima posizione dell'array, si continuerà ad inserire elementi a partire dalla prima posizione. Il secondo puntatore di cui si fa uso indica il primo elemento estraibile dalla coda; questo puntatore viene indicato con QFRONT. Se QFRONT = 0 significa che la coda è vuota; quando si inserisce il primo elemento della coda gli verrà assegnato il valore 1 (vedi Figura 7.7).

Algoritmo 7.5 ADD-EVENT-Q (CLASS, NUMBER, DATA)
Inserisce un elemento nella coda degli eventi.

Argomenti	CLASS←classe del dispositivo di input
	NUMBER←numero del dispositivo di input
	DATA←dati provenienti dal dispositivo di input
Variabili globali	EVENTQC, EVENTQN, EVENTQD←array di dimensione QSIZE che formano la coda
	QFRONT, QREAR←puntatori all'inizio e alla fine della coda.
Costanti	QSIZE←dimensione massima della coda

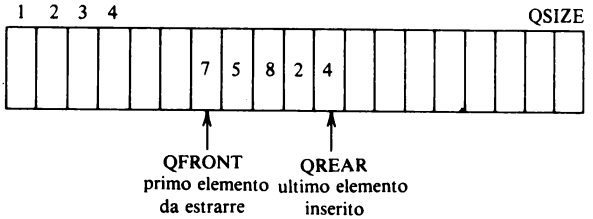


Figura 7.7 Un array usato come coda, contenente 7, 5, 8, 2 e 4

```
begin
   if QREAR = QSIZE then QREAR←1
   else QREAR←QREAR + 1;
   if QFRONT = QREAR
   then return-error 'OVERFLOW NELLA CODA DI EVENTI';
   EVENTQC[QREAR]←CLASS;
   EVENTQN[QREAR]←NUMBER;
   EVENTQD[QREAR]←DATA:
   if QFRONT = 0 then QFRONT←1;
   return;
end;
```

La coda per le stringhe di caratteri viene trattata in modo diverso. Ciascuna nuova stringa viene aggiunta in un array opportuno, mentre nella coda viene inserito solamente l'indice corrispondente alla loro posizione all'interno di questo array.

Algoritmo 7.6 ADD-STRING-Q (STRING, DATA)
Salva STRING e restituisce un suo puntatore in DATA.

Argomenti STRING←stringa che deve essere memorizzata nell'array
DATA←restituisce l'indice della posizione nella quale la stringa è memorizzata

Variabili globali STRINGQ←array di stringhe
SQREAR←puntatore all'ultima stringa introdotta

Costanti SQSIZE←dimensione dell'array delle stringhe di caratteri

```
begin
   if SQREAR = SQSIZE then SQREAR←1
   else SQREAR←SQREAR + 1;
   STRINC Q[SQREAR]←STRING;
   DATA←SQREAR;
   return;
end;
```

L'utente deve essere in grado di trattare, con opportune routine, gli eventi presenti nella coda, in modo che si possa stabilire ad esempio se la coda contiene le informazioni relative a qualche evento. A questo scopo sarà necessaria una routine di attesa dell'evento AWAIT-EVENT, che controlli la coda e restituisca il numero del dispositivo che ha provocato l'evento. Se non è stato inserito alcun evento e la coda è vuota, viene restituito un opportuno valore che rappresenta questa situazione.
Nel caso che non si siano verificate interruzioni, questa routine controlla ripetutamente il contenuto della coda per un certo periodo di tempo. Se allo scadere di questo periodo la coda rimane vuota, viene restituito un indicatore che specifica che non si sono verificati eventi. Diversamente la routine permette di

ottenere dalla coda la classe e il numero del dispositivo che ha provocato l'evento.
Se il dispositivo di input non genera interruzioni, allora è possibile simulare questo comportamento controllando direttamente lo stato del dispositivo. Esistono due modi per realizzare il polling: in un primo caso si possono utilizzare le prestazioni offerte direttamente dal linguaggio di programmazione e/o dal sistema operativo che si sta usando. I dati di input provengono a richiesta da uno specifico dispositivo, nello stesso modo con cui si opera per gestire i dispositivi campionati; l'istruzione **read** di un linguaggio ad alto livello, ad esempio, opera in questo modo. Il secondo metodo consiste nell'utilizzare un ciclo all'interno del quale si controlla lo stato del dispositivo. Questo è ciò che viene chiamato *polled device*. In generale la routine AWAIT-EVENT utilizza tutte queste tecniche. Per esempio, l'operazione di pick può essere eseguita tramite vere e proprie interruzioni, mentre il controllo se è stato premuto un pulsante può essere fatto tramite un ciclo di polling e la lettura dei dati provenienti da una tastiera può venir contemporaneamente realizzata tramite una semplice istruzione **read**. Negli algoritmi seguenti si riportano alcuni esempi dei tre metodi. Nell'implementazione di questi algoritmi si dovranno considerare solo quelle parti relative ai dispositivi che sono disponibili in una certa configurazione.

Algoritmo 7.7 AWAIT-EVENT (WAIT, CLASS, DEVICE)
Controlla lo stato della coda di eventi.

Argomenti	WAIT←tempo di attesa per l'evento
	CLASS, DEVICE←classe e tipo dell'evento
Variabili globali	BUTTON, PICK, KEYBOARD←segnalatori dei dispositivi abilitati
	INPUT-STRING, PICKED-SEGMENT←memoria per l'input proveniente da tastiera e dai dispositivi di pick
	DETECTABLE←array che contiene i segmenti degli attributi di reperibilità
	BUTTON-POLL, PICK-POLL, KEYBOARD-POLL←indicatori dello stato di questi dispositivi
Variabili locali	TIME-LIMIT←tempo massimo di attesa
	DATA←memoria per i dati provenienti dalla coda di eventi

begin
se si sta simulando un pulsante mediante un dispositivo campionato, si inseriscano le seguenti istruzioni

```
if BUTTON
then begin
        read DEVICE;
        CLASS←1;
        return;
        end;
```

se si sta simulando un dispositivo di pick mediante un dispositivo campiona-

to, si inseriscano le seguenti istruzioni

```
if PICK
then begin
        PICKED-SEGMENT←0;
        while PICKED-SEGMENT<1
        or PICKED-SEGMENT>NUMBER-OF-SEGMENTS
        or not DETECTABLE[PICKED-SEGMENT];
        do read PICKED-SEGMENT;
        CLASS←2;
        DEVICE←1;
        return;
        end;
```

se si sta simulando una tastiera mediante un dispositivo campionato, si inseriscano le seguenti istruzioni

```
if KEYBOARD
then begin
        read INPUT-STRING;
        CLASS←3;
        DEVICE←1;
        return;
        end;
```

se sono disponibili dei dispositivi che generano interruzioni nella coda, si aggiungano le seguenti istruzioni per controllare il contenuto della coda

```
TIME-LIMIT←TIME( )+WAIT;
while TIME( )≤TIME-LIMIT
do begin
```

se i dispositivi che generano interruzioni inseriscono le loro informazioni nella coda, si aggiungano le seguenti istruzioni per controllare il contenuto della coda

```
   begin
      GETQ(CLASS, DEVICE, DATA);
      if CLASS≠0
      then if CLASS = 2
      then PICKED-SEGMENT←DATA
      else if CLASS = 3
      then INPUT-STRING←STRINGQ[DATA];
      return;
   end;
```

se si sta simulando un pulsante mediante un dispositivo campionato con un polling, si inseriscano le seguenti istruzioni

```
if BUTTON and BUTTON-POLL
then begin
        CLASS←1;
        read BUTTON(DEVICE);
        return;
```

```
        end;
```

se si sta simulando un dispositivo di pick mediante un dispositivo campionato con un polling, si inseriscano le seguenti istruzioni

```
    if PICK and PICK-POLL
    then begin
            read PICK(DEVICE, PICKED-SEGMENT);
            if DETECTABLE[PICKED-SEGMENT]
            then begin
                    CLASS←2;
                    return;
                    end;
            end;
```

se si sta trattando la tastiera come dispositivo campionato con un polling, si inseriscano le seguenti istruzioni

```
    if KEYBOARD and KEYBOARD-POLL
    then begin
            CLASS←3;
            read KEYBOARD(DEVICE, INPUT-STRING);
            return;
            end;
    end;
    CLASS←0;
    DEVICE←0;
    return;
end;
```

Se si dispone di un meccanismo di interruzione e quindi si deve gestire una vera coda di eventi, occorre una routine per prelevare l'elemento che è in testa alla coda. L'algoritmo utilizza il puntatore QFRONT per indicare la posizione dell'elemento iniziale. Se QFRONT è uguale a 0 significa che la coda è vuota; diversamente viene restituito il valore del primo elemento e il puntatore viene spostato ad indicare il successivo.

Algoritmo 7.8 GETQ (CLASS, DEVICE, DATA)
Restituisce i dati relativi al primo elemento della coda e — nel caso che la coda sia vuota — restituisce 0.

Argomenti	CLASS←classe di eventi DEVICE←dispositivo all'interno di una stessa classe DATA←informazioni associate all'evento
Variabili globali	EVENTQC, EVENTQN, EVENTQD←array che costituiscono la coda degli eventi QFRONT, QREAR←puntatori all'inizio e alla fine della coda
Costanti	QSIZE←dimensione della coda di eventi

```
begin
   CLASS←0;
   if QFRONT = 0 then return;
   CLASS←EVENTQC[QFRONT];
   DEVICE←EVENTQN[QFRONT];
   DATA←EVENTQD[QFRONT];
   if QFRONT = QREAR
   then begin
      QFRONT←0;
      QREAR←0;
      end;
   else if QFRONT = QSIZE
      then QFRONT←1
      else QFRONT←QFRONT+1;
   return;
end;
```

La coda degli eventi permette di trattenere le informazioni associate a molti eventi prima che l'informazione ad essi associata venga utilizzata. La coda rappresenta tutta la memoria necessaria per conservare le informazioni fino al momento in cui queste vengono utilizzate. A volte però può essere necessario iniziare nuove operazioni, partendo con la coda nello stato iniziale e ignorando tutti gli eventi in essa contenuti, anche se non sono stati ancora considerati. La routine seguente ripulisce la coda distruggendo tutti gli eventi inutili.

Algoritmo 7.9 FLUSH-ALL-EVENTS
Rimuove tutti gli eventi dalla coda.
Variabili globali QFRONT, QREAR←puntatori all'inizio e alla fine della coda
SQREAR←puntatore al file di tipo stringa

```
begin
   QFRONT←0;
   QREAR←0;
   SQREAR←1;
   return;
end;
```

La routine AWAIT-EVENT simula l'arrivo di un evento che, nel caso si tratti di una tastiera o di dispositivi di pick, avrà diverse informazioni associate che sarà possibile recuperare per mezzo di apposite routine. Nel caso si tratti di una tastiera, la routine GET-KEYBOARD-DATA restituisce la stringa di caratteri che è stata inserita e la sua lunghezza. Nel caso si tratti invece di un dispositivo di pick, la routine GET-PICK-DATA restituisce il numero del segmento selezionato dall'utente.

Algoritmo 7.10 GET-KEYBOARD-DATA (STRING, LEN)
Restituisce le informazioni provenienti dalla tastiera.
Argomenti STRING←stringa di caratteri
LEN←lunghezza della stringa
Variabili globali INPUT-STRING←memoria per le informazioni provenienti dalla tastiera

```
begin
   LEN←LENGTH(INPUT-STRING);
   STRING←INPUT-STRING;
   return;
end;
```

Algoritmo 7.11 GET-PICK-DATA (SEGMENT-NAME)
Restituisce il valore selezionato con il dispositivo di pick.
Argomenti SEGMENT-NAME←nome del segmento selezionato
Variabili globali PICKED-SEGMENT←memoria per le informazioni provenienti dal dispositivo di pick

```
begin
   SEGMENT-NAME←PICKED-SEGMENT;
   return;
end;
```

La routine AWAIT-EVENT permette a più dispositivi di essere attivi nello stesso tempo; la maggior parte delle applicazioni però esige informazioni da un solo dispositivo alla volta e inoltre accade che i sistemi time-sharing, di cui si dispone, non permettono l'uso simultaneo di più dispositivi. Per tutti questi motivi, il trattamento generale delle code di eventi risulta superfluo. Allo scopo di evitare queste situazioni, il sistema CORE fornisce delle routine che accettano le informazioni da una sola classe di dispositivi alla volta: pulsanti, tastiere o dispositivi di pick. Ancora una volta queste routine dipendono dal sistema e dalla configurazione hardware disponibile. Gli algoritmi seguenti forniscono indicazioni per implementare le routine per i pulsanti e per i dispositivi di pick.

Algoritmo 7.12 AWAIT-BUTTON (WAIT, BUTTON-NUM)
Routine utente per la gestione di un pulsante.
Argomenti WAIT←tempo di attesa per l'evento
BUTTON-NUM←restituisce il numero del dispositivo fisico
Variabili globali BUTTON←segnalatore dei dispositivi abilitati
BUTTON-POLL←segnalatore dello stato del pulsante nel caso che si tratti di un dispositivo campionato
Variabili locali TIME-LIMIT←tempo massimo di attesa
DUMMY←argomento fittizio

```
begin
   if not BUTTON
```

```
        then return-error 'PULSANTE NON ABILITATO';
    se si sta simulando un pulsante mediante un dispositivo campionato, si
    inserisca la seguente istruzione
    read BUTTON-NUM;
    se si stanno usando dei dispositivi campionati con un ciclo di polling o a
    generazione di interruzione, si inserisca il seguente ciclo
    TIME-LIMIT←TIME( ) + WAIT;
    while TIME( ) ≤ TIME-LIMIT
        do begin
        se si stanno usando dei pulsanti che generano interruzioni, le infor-
        mazioni possono essere reperite per mezzo delle seguenti istruzioni
            begin
                GETQ(CLASS, BUTTON-NUM, DUMMY);
                if CLASS = 1
                then return;
            end;
            se si sta simulando un pulsante mediante un dispositivo campiona-
            to con un polling, si inserisca la seguente istruzione
            if BUTTON-POLL
            then begin
                    read BUTTON(BUTTON-NUM);
                    return;
                    end;
            end;
    BUTTON-NUM←0;
    return;
end;
```

Algoritmo 7.13 AWAIT-PICK (WAIT, PICK-NUM)
Routine utente per la gestione di un dispositivo di pick.

Argomenti WAIT←tempo di attesa per l'evento
PICK-NUM←restituisce il numero del segmento selezionato

Variabili globali PICK←segnalatore dei dispositivi abilitati
PICK-POLL←segnalatore dello stato del dispositivo di pick se si tratta di un dispositivo campionato

Variabili locali TIME-LIMIT←tempo massimo di attesa

```
begin
    if not PICK
    then return-error 'PICK NON ABILITATO';
    se si sta simulando un dispositivo di pick mediante un dispositivo cam-
    pionato, si inserisca la seguente istruzione
    read PICK-NUM;
    se si stanno usando dei dispositivi campionati con un ciclo di polling o a
    generazione di interruzione, si inserisca il seguente ciclo
```

```
    TIME-LIMIT←TIME( ) + WAIT;
    while TIME( ) ≤ TIME-LIMIT
       do begin
       se si stanno usando dei dispositivi di pick che generano interruzioni,
       le informazioni possono essere reperite per mezzo delle seguenti istru-
       zioni
          begin
             GETQ(CLASS, DUMMY, PICK-NUM);
             if CLASS = 1
             then return;
          end;
          se si sta simulando un dispositivo di pick mediante un dispositivo
          campionato con un polling, si inserisca la seguente istruzione
          if PICK-POLL
          then begin
                  read PICK(PICK-NUM);
                  return;
                  end;
          end;
    PICK-NUM←0;
    return;
end;
```

7.4 DISPOSITIVI CAMPIONATI

Un locator è un dispositivo campionato e può quindi essere letto in ogni momento in cui l'utente lo desideri; non è necessario attendere il verificarsi di un evento che andrebbe inserito nella coda opportuna. La sola cosa necessaria per la gestione di questi dispositivi è una routine che legga lo stato del dispositivo e ritorni i valori attuali delle coordinate x, y del punto indicato.
A causa del fatto che i locator sono dispositivi campionati, non si avrà bisogno della routine AWAIT-EVENT. Per un certo dispositivo di tipo locator i valori delle coordinate correnti vengono lette dal dispositivo fisico e restituite direttamente. Per simulare questo comportamento utilizzando una tastiera, i valori delle coordinate potranno essere letti in forma alfanumerica.

Algoritmo 7.14 READ-LOCATOR (X, Y)
Argomenti X, Y←coordinate passate dal dispositivo
Variabili globali LOCATOR←segnalatore di abilitazione del locator

```
begin
   if not LOCATOR
   then return-error 'LOCATOR NON ABILITATO';
   PRELEVA (X, Y) DAL LOCATOR O DAL DISPOSITIVO CHE LO
      SIMULA;
```

```
    SE NECESSARIO CONVERTE IN COORDINATE NORMALIZ-
      ZATE;
    return;
end;
```

I valuator vengono trattati essenzialmente allo stesso modo dei locator, sebbene venga restituito un solo valore invece che una coppia di coordinate.

```
Algoritmo 7.15  READ-VALUATOR (V)
Argomenti          V←valore passato dal dispositivo
Variabili globali  VALUATOR←segnalatore di abilitazione del valuator
begin
   if not VALUATOR
   then return-error 'VALUATOR NON ABILITATO';
   PRELEVA V DAL VALUATOR O DAL DISPOSITIVO CHE LO
     SIMULA;
   return;
end;
```

7.5 ATTRIBUTI DI REPERIBILITÀ

Un'utile funzionalità per i dispositivi di pick è la possibilità di reperire porzioni dell'immagine. Può essere infatti necessario individuare un elemento in un sottoinsieme di elementi che compaiono contemporaneamente sullo schermo. Tutti gli elementi non selezionabili verranno ignorati dal dispositivo di pick. Questo comportamento può essere realizzato disabilitando il meccanismo di interruzione quando la porzione della figura che non può essere selezionata è in fase di visualizzazione. Il meccanismo di interruzione viene riabilitato quando si stanno disegnando elementi che devono essere reperibili dal pick. L'utente disporrà quindi di alcune routine per rendere identificabili gli elementi che compaiono sullo schermo. Per questo si aggiungeranno degli *attributi di reperibilità* ai segmenti, in modo tale che si possa sapere se sono individuabili o meno per mezzo del dispositivo di pick. Questa possibilità è garantita associando a ciascun segmento un flag e fornendo all'utente una routine che assegna il valore a questo indicatore (considerando che il valore iniziale sia tale da rendere il segmento non selezionabile). Il valore assunto da questo indicatore di reperibilità potrà essere utilizzato dalle routine che generano l'immagine su un terminale a refresh per disabilitare o abilitare le interruzioni provenienti dalla penna ottica. Questo indicatore sarà utilizzato anche simulando le operazioni di pick con altri tipi di dispositivi. Un esempio di tale comportamento è illustrato nella routine AWAIT-EVENT, descritta precedentemente per la simulazione di un dispositivo di pick campionato: nel caso che il pick di un certo segmento venga simulato attraverso la tastiera, l'indicatore di reperibilità per quel segmento dovrà ve-

nire controllato dalla routine AWAIT-EVENT e — nel caso che il segmento risulti non reperibile — verrà richiesto un altro dato in ingresso; se invece il segmento è reperibile, allora il suo nome verrà memorizzato e la routine AWAIT-EVENT restituirà un parametro per indicare che è stata eseguita un'operazione di pick valida.
Se si dispone di dispositivi che generano interruzioni per eseguire un'operazione di pick, allora la routine MAKE-PICTURE-CURRENT o la INTERPRET dovranno essere estese per includere la possibilità di abilitare o disabilitare le interruzioni, in corrispondenza del valore assunto dall'indicatore di reperibilità associato al segmento che si sta interpretando.
Il seguente algoritmo permette all'utente di assegnare il valore all'attributo di reperibilità di un dato segmento.

Algoritmo 7.16 SET-DETECTABILITY (SEGMENT-NAME, ON-OFF)
Routine utente che assegna il valore all'attributo di reperibilità di un segmento.

Argomenti SEGMENT-NAME ← segmento del display file interessato
ON-OFF ← valore dell'indicatore di reperibilità
Variabili globali DETECTABLE ← array di attributi di reperibilità
Costanti NUMBER-OF-SEGMENTS ← dimensione dell'array DETECTABLE

```
begin
   if SEGMENT-NAME < 1
   or SEGMENT-NAME > NUMBER-OF-SEGMENTS
   then return-error 'SEGMENTO NON VALIDO';
   DETECTABLE [SEGMENT-NAME] ← ON-OFF;
   return;
end;
```

In questo modo è stato introdotto un nuovo attributo per i segmenti. Ciò significa che la routine che gestisce la tabella dei segmenti dovrà essere estesa per contenere questa nuova proprietà. Ci sono due routine che vanno modificate di conseguenza: la routine CREATE-SEGMENT, che inizializza i valori di tutti gli attributi e la RENAME-SEGMENT, che copia tutti gli attributi in una nuova posizione della tabella. La versione modificata di queste routine viene descritta di seguito.

Algoritmo 7.17 CREATE-SEGMENT (SEGMENT-NAME)
(Algoritmo 6.4 aggiornato)
Routine utente che crea un segmento identificato da un nome.

Argomenti SEGMENT-NAME ← nome del segmento
Variabili globali NOW-OPEN ← segmento correntemente aperto
FREE ← indice della successiva cella libera nel display file
SEGMENT-START, SEGMENT-SIZE, VISIBILITY, ANGLE, SCALE-X, SCALE-Y, TRANSLATE-X,

TRANSLATE-Y, DETECTABLE←tabella dei segmenti

Costanti NUMBER-OF-SEGMENTS←dimensione della tabella dei segmenti

```
begin
   if NOW-OPEN > 0
        then return-error 'SEGMENTO ANCORA APERTO';
   if SEGMENT-NAME < 1
   or SEGMENT-NAME > NUMBER-OF-SEGMENTS
        then return-error 'NOME DEL SEGMENTO NON VALIDO';
   if SEGMENT-SIZE [SEGMENT-NAME] > 0
        then return-error 'SEGMENTO GIÀ ESISTENTE';
   NEW-VIEW-2
   SEGMENT-START [SEGMENT-NAME] ← FREE;
   SEGMENT-SIZE [SEGMENT-NAME] ← 0;
   VISIBILITY [SEGMENT-NAME] ← VISIBILITY [0];
   ANGLE [SEGMENT-NAME] ← ANGLE [0];
   SCALE-X [SEGMENT-NAME] ← SCALE-X [0];
   SCALE-Y [SEGMENT-NAME] ← SCALE-Y [0];
   TRANSLATE-X [SEGMENT-NAME] ← TRANSLATE-X [0];
   TRANSLATE-Y [SEGMENT-NAME] ← TRANSLATE-Y [0];
   DETECTABLE [SEGMENT-NAME] ← DETECTABLE [0];
   NOW-OPEN ← SEGMENT-NAME;
   return;
end;
```

Algoritmo 7.18 RENAME-SEGMENT (SEGMENT-NAME-1, SEGMENT-NAME-2) (Algoritmo 5.6 aggiornato)
Routine utente che assegna al segmento SEGMENT-NAME-1 il nome di SEGMENT-NAME-2.

Argomenti SEGMENT-NAME-1←vecchio nome del segmento
SEGMENT-NAME-2←nuovo nome del segmento

Variabili globali SEGMENT-START, SEGMENT-SIZE, VISIBILITY, ANGLE, SCALE-X, SCALE-Y, TRANSLATE-X, TRASLATE-Y, DETECTABLE←tabella dei segmenti
NOW-OPEN←segmento attualmente aperto

Costanti NUMBER-OF-SEGMENTS←dimensione della tabella dei segmenti

```
begin
   if SEGMENT-NAME-1 < 1 or SEGMENT-NAME-2 < 1
   or SEGMENT-NAME-1 > NUMBER-OF-SEGMENTS
   or SEGMENT-NAME-2 > NUMBER-OF-SEGMENTS
        then return-error 'NOME DEL SEGMENTO NON VALIDO';
   if SEGMENT-NAME-1 = NOW-OPEN
   or SEGMENT-NAME-2 = NOW-OPEN
```

```
            then return-error 'SEGMENTO ANCORA APERTO';
        if SEGMENT-SIZE[SEGMENT-NAME-2] ≠ 0
            then return-error 'SEGMENTO GIÀ ESISTENTE';
        copia il vecchio segmento nella nuova posizione
        SEGMENT-START[SEGMENT-NAME-2]
            ←SEGMENT-START[SEGMENT-NAME-1];
        SEGMENT-SIZE[SEGMENT-NAME-2]
            ←SEGMENT-SIZE[SEGMENT-NAME-1];
        VISIBILITY[SEGMENT-NAME-2]
            ←VISIBILITY[SEGMENT-NAME-1];
        ANGLE[SEGMENT-NAME-2]←ANGLE[SEGMENT-NAME-1];
        SCALE-X[SEGMENT-NAME-2]←SCALE-X[SEGMENT-NAME-1];
        SCALE-Y[SEGMENT-NAME-2]←SCALE-Y[SEGMENT-NAME-1];
        TRANSLATE-X[SEGMENT-NAME-2]
            ←TRANSLATE-X[SEGMENT-NAME-1];
        TRANSLATE-Y[SEGMENT-NAME-2]
            ←TRANSLATE-Y[SEGMENT-NAME-1];
        DETECTABLE[SEGMENT-NAME-2]
            ←DETECTABLE[SEGMENT-NAME-1];
        cancella il vecchio segmento
        SEGMENT-SIZE[SEGMENT-NAME-1]←0;
        return;
    end;
```

Come nei precedenti capitoli, l'utente disporrà di una routine di inizializzazione. Questa routine assegna dei valori iniziali tali per cui nessun dispositivo sia abilitato, la coda degli eventi sia vuota e nessun segmento sia reperibile.

Algoritmo 7.19 INITIALIZE-7

Variabili globali DETECTABLE ←array contenente gli attributi di reperibilità

```
begin
    INITIALIZE-6;
    DISABLE-ALL;
    FLUSH-EVENT-Q;
    DETECTABLE[0]←FALSE;
    return;
end;
```

7.6 SIMULAZIONE DI UN LOCATOR CON UN DISPOSITIVO DI PICK

Come si è detto, una penna ottica può essere utilizzata per selezionare un oggetto che compare sullo schermo, ma non ne fornisce la posizione. Inoltre, questo dispositivo non può essere usato per indicare una posizione dello schermo nella quale non vi siano oggetti, operazione che è possibile invece utilizzando un joystick o una tavoletta. Se si vuole usare la penna ottica per ottenere informazioni sulla posizione di un punto, è necessario realizzare un meccanismo che viene detto *tracking cross*. Una tracking cross consiste di una piccola croce composta da quattro segmenti di retta separati, che viene fatta comparire in una posizione nota sullo schermo ed è individuabile per mezzo della penna ottica (vedi Figura 7.8).
Nella figura la penna si trova in posizione tale da indicare il centro della croce. Non appena la penna viene mossa, incontrerà uno dei segmenti che compongono la croce. Se la penna si muove ad esempio verso destra, incontrerà il segmento più a destra della croce generando un'interruzione. Questa informazione viene usata per muovere la croce verso destra di una quantità opportuna. Dopo ogni refresh dello schermo, la posizione della croce viene aggiornata mantenendola in modo tale che il suo centro indichi sempre il punto dove la penna tocca lo schermo (vedi Figura 7.9).
Operando in questo modo per ognuno dei segmenti di retta che formano la croce, questa seguirà i movimenti della penna ottica. Poiché la posizione del centro della croce sullo schermo è nota, si potranno inserire informazioni di posizione semplicemente «trascinando» la croce con la penna ottica fino alla posizione desiderata.

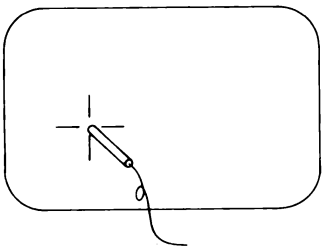


Figura 7.8 Penna ottica con la corrispondente tracking cross

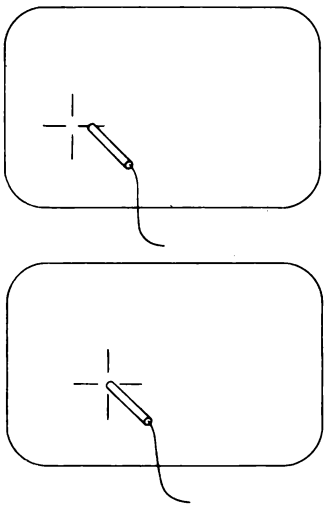


Figura 7.9 Se la penna ottica incontra uno dei segmenti della croce, l'intera croce viene spostata

7.7 SIMULAZIONE DI UN DISPOSITIVO DI PICK CON UN LOCATOR

Si supponga di avere a disposizione un dispositivo di tipo locator ma di non disporre invece di alcun dispositivo adatto ad eseguire operazioni di pick. Utilizzando il locator, si è in grado di conoscere il punto che l'utente sta indicando ma non si è in grado di sapere quale sia il segmento che è stato visualizzato vicino a quella posizione. Per conoscere quest'informazione si dovranno percorrere le istruzioni del display file, segmento per segmento, come se si stesse disegnando la figura, ma invece di visualizzarne i vari elementi, si calcolerà la distanza tra le diverse rette e i caratteri che la compongono e la posizione indicata dal locator. Se una retta o un carattere sono abbastanza vicini a quel punto, si potrà concludere che il segmento a cui appartengono è l'entità che l'utente voleva indicare. Il numero di quel segmento è quindi il valore risultante dalla operazione di pick. Per calcolare la distanza tra un punto (x, y) e un carattere, la posizione del carattere verrà trattata come un punto (x_c, y_c) e si userà quindi la seguente formula per calcolare la distanza:

$$D = [(x-x_c)^2+(y-y_c)^2]^{1/2} \tag{7.1}$$

se la distanza è abbastanza piccola, cioè si verifica che

$$D<\text{APERTURE} \tag{7.2}$$

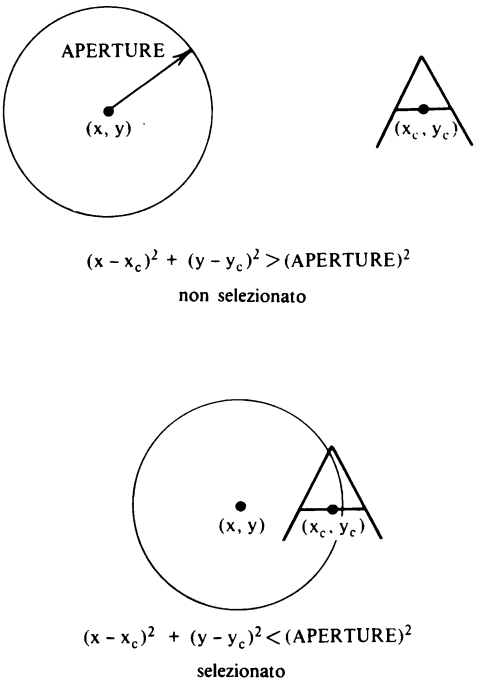


Figura 7.10 Selezione di un punto

allora il segmento attualmente in esame è quello desiderato (vedi Figura 7.10). Le (7.1) e (7.2) selezionano un carattere quando il suo punto centrale giace all'interno di un cerchio di raggio APERTURE e centro nel punto indicato dal locator e richiede quindi l'esecuzione di operazioni di moltiplicazione che sono costose in termini di tempo di calcolo. Un controllo più efficiente sotto questo punto di vista può essere fatto controllando che il punto sia contenuto in un quadrato di lato uguale a 2*APERTURE, centrato nel punto indicato dal locator. In questo caso la condizione di prossimità diviene:

$$|(x-x_c)| + |(y-y_c)| < \text{APERTURE} \tag{7.3}$$

Questa formula è più efficiente in quanto il calcolo del valore assoluto è più facile da eseguire che non quello dell'elevazione a potenza (vedi Figura 7.11). Leggermente più complesso è il calcolo della distanza tra un punto ed un segmento. Infatti a questo scopo si deve determinare l'equazione di una retta, perpendicolare al segmento, che passi per il punto, trovare l'intersezione tra le due

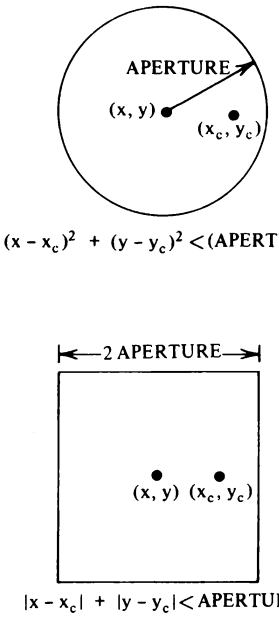


Figura 7.11 La verifica di inclusione in un'area quadrata è più efficiente

rette e infine determinare la distanza tra il loro punto di intersezione ed il punto dato dal locator. Dopo opportuni passaggi algebrici si ottiene la seguente formula:

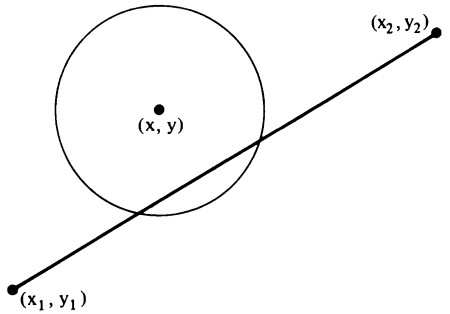
$$D = \frac{|(x-x_1)(y_2-y_1)-(y-y_1)(x_2-x_1)|}{[(x_2-x_1)^2+(y_2-y_1)^2]^{1/2}} \tag{7.4}$$

nella quale (x, y) è il punto fornito dal locator e (x_1, y_1), (x_2, y_2) sono gli estremi del segmento. Si osservi però che questa formula per calcolare la distanza si applica a tutta la retta che contiene il segmento. Per questo motivo è necessario un controllo preliminare al fine di determinare se il punto di intersezione tra la normale passante per il punto e la retta stessa sia all'esterno del segmento. Si potrà cioè ignorare il segmento se

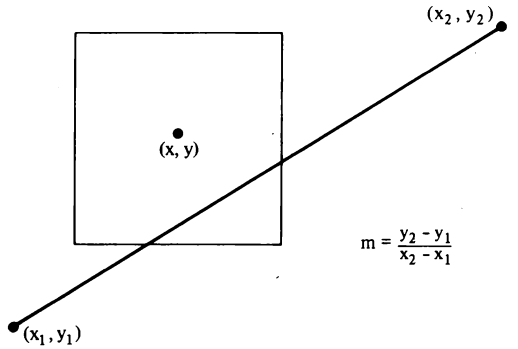
$$\begin{aligned} x &< \text{MIN}(x_1, x_2) - \text{APERTURE} \\ \text{oppure } x &> \text{MAX}(x_1, x_2) + \text{APERTURE} \\ \text{oppure } y &< \text{MIN}(y_1, y_2) - \text{APERTURE} \\ \text{oppure } y &> \text{MAX}(y_1, y_2) + \text{APERTURE} \end{aligned} \tag{7.5}$$

Anche in questo caso nella formula scelta compaiono diverse moltiplicazioni che sono costose in termini di tempo di calcolo; ciò accade perché, analogamente al caso precedente, si è scelto di accettare i segmenti che intersecano una circonferenza tracciata attorno al punto fornito dal locator. Se invece si considera un quadrato, si userà la seguente formula che è estremamente più efficiente (vedi Figura 7.12):

$$\text{MIN}\left(\left|\frac{(y_2-y_1)(x-x_1)}{x_2-x_1}+y_1-y\right|,\ \left|\frac{(x_2-x_1)(y-y_1)}{y_2-y_1}+x_1-x\right|\right) < \text{APERTURE} \qquad (7.6)$$



$$\frac{[(x-x_1)(y_2-y_1)-(y-y_1)(x_2-x_1)]^2}{(x_2-x_1)^2+(y_2-y_1)^2} < (\text{APERTURE})^2$$



$$\text{MIN}\left(|m(x-x_1)+y_1-y|, \left|\frac{(y-y_1)}{m}+x_1-x\right|\right) < \text{APERTURE}$$

Figura 7.12 Verifiche per la selezione di un segmento di retta

Infine si osservi che i segmenti verticali e orizzontali devono essere trattati come casi particolari. Di seguito viene descritto un algoritmo per la simulazione dei dispositivi di pick utilizzando la penna ottica. Il programma percorre la tabella dei segmenti allo stesso modo della routine MAKE-PICTURE-CURRENT, considerando solo i segmenti visibili e selezionabili. Si scandisce il display file di ciascuno di questi segmenti in modo simile a quello della routine INTERPRET ma, invece di eseguire le istruzioni, si applicano i controlli che sono stati definiti precedentemente. Il primo segmento contenente un elemento che soddisfa le condizioni di prossimità viene restituito come risultato dell'operazione di pick. Se non viene individuato alcun segmento, il valore restituito è zero.

Algoritmo 7.20 PICK-SEARCH (PICK, X, Y)
Routine utente che simula l'operazione di pick utilizzando un locator che indica quale segmento viene visualizzato nei pressi di X, Y.

Argomenti
PICK←restituisce il numero che identifica il segmento trovato
X, Y←posizione del pick simulato

Variabili globali
APERTURE←sensibilità del pick
SEGMENT-START, SEGMENT-SIZE←tabella dei segmenti
DETECTABLE←attributo di reperibilità del segmento
VISIBILITY←attributo di visibilità del segmento

Variabili locali
SEGMENT←indice di scansione per tutti i segmenti possibili
X1, Y1, X2, Y2←estremi del segmento considerato
INSTRUCTIONS←indice per percorrere i segmenti nel display file

Costanti
ROUNDOFF←numero piccolo più grande del massimo errore di arrotondamento
NUMBER-OF-SEGMENTS←dimensioni della tabella dei segmenti

```
begin
   X1←0;
   Y1←0;
   for SEGMENT = 1 to NUMBER-OF-SEGMENTS
   do if SEGMENT-SIZE[SEGMENT]≠0
         and VISIBILITY[SEGMENT]
         and DETECTABLE[SEGMENT]
   then begin
         BUILD-TRANSFORMATION(SEGMENT);
         DO-TRANSFORMATION(X1, Y1);
         PICK←SEGMENT;
         for INSTRUCTION = SEGMENT-START[SEGMENT] to
            SEGMENT-START[SEGMENT]
             +SEGMENT-SIZE[SEGMENT]-1
```

```
        do begin
              GET POINT(INSTRUCTION, OP, X2, Y2);
              if OP>0
                 then begin
                    DO-TRANSFORMATION(X2, Y2);
                    if OP>31
                    and |X-X2|+|Y-Y2|<APERTURE
                       then return;
                    if OP = 2
                    then if X>MIN(X1, X2)- APERTURE and
                            X<MAX(X1, X2)+APERTURE and
                            Y>MIN(Y1, Y2)-APERTURE and
                            Y<MAX(Y1, Y2)+APERTURE
                    then if|Y2-Y1|<ROUDOFF
                         or|X2-X1|<ROUNDOFF
                    then return;
                    else if MIN
                            (|(Y2-Y1)*(X-X1)/(X2-X1)+(Y1-Y)|,
                            |(X2-X1)*(Y-Y1)/(Y2-Y1)+(X1-X)|)
                            <APERTURE
                    then return;
                    X1←X2;
                    Y1←Y2;
                    end;
           end;
     end;
     PICK←0;
     return;
end;
```

7.8 ECO CORRISPONDENTE ALLE OPERAZIONI DI INPUT

In un sistema interattivo, una componente importante è l'eco. L'eco fornisce all'utente informazioni circa le azioni che ha effettuato. Questo permette di confrontare l'effetto delle operazioni fatte con ciò che si intendeva fare. L'eco corrispondente alle informazioni provenienti da una tastiera, per esempio, consiste nel visualizzare i caratteri corrispondenti ai tasti premuti. Per quanto riguarda i dispositivi di tipo locator, l'eco può essere fatto visualizzando un cursore nella posizione corrente; questo permette all'utente di mettere in relazione questa posizione con l'oggetto che compare sullo schermo. L'eco corrispondente ai dispositivi di pick consiste nell'identificare visivamente l'oggetto che è stato selezionato sullo schermo, che può essere evidenziato, ad esempio, cambian-

done il colore, la luminosità o facendolo lampeggiare; ciò permette all'utente di determinare se l'oggetto selezionato sia veramente quello desiderato. Per quanto riguarda i pulsanti, l'eco può essere realizzato facendo lampeggiare o aumentare la luminosità del campo selezionato, nel caso ad esempio che i pulsanti vengano usati per selezionare un campo di un menu. L'eco adatto ad un dispositivo valuator è la visualizzazione del numero corrispondente allo stato del dispositivo. Ciò che è importante, comunque, è la presenza di qualche forma di eco, indipendentemente dalla particolare scelta che può essere fatta nell'implementazione. Senza di esso, l'utente è costretto a lavorare «al buio», cioè senza conoscere immediatamente l'effetto delle operazioni che ha eseguito ed in questa situazione l'uso dei programmi risulta essere evidentemente difficoltoso.

7.9 TECNICHE DI INTERAZIONE

Quest'ultimo paragrafo tratta di alcune tecniche che consentono di creare e modificare interattivamente le figure. Si consideri innanzitutto il problema di aggiungere nuovi elementi grafici ad un'immagine che è già stata costruita precedentemente. Una prima funzionalità necessaria è la tecnica detta *point plotting*, che permette all'utente di selezionare un punto sullo schermo, solitamente utilizzando contemporaneamente un locator e un pulsante. Il locator consente di conoscere le coordinate del punto selezionato dall'utente, mentre tramite il pulsante si può confermare il corretto posizionamento del locator. L'algoritmo che realizza l'operazione di point plotting, comunicata attraverso il pulsante, legge le coordinate provenienti dal locator; la selezione di un punto può essere utilizzata per scopi differenti: ad esempio è possibile fornire gli estremi di un segmento, la posizione in cui scrivere una stringa di caratteri o i valori della traslazione da applicare ad un segmento appartenente alla figura. Quest'operazione viene usata così frequentemente da rendere opportuno fornire una routine apposita.

Algoritmo 7.21 AWAIT-BUTTON-GET-LOCATOR (WAIT, BUTTON-NUM, X, Y)
Routine utente che seleziona interattivamente un punto.
Argomenti WAIT←tempo massimo di attesa del consenso dato dal pulsante
BUTTON-NUM←variabile che restituisce il pulsante selezionato dall'utente
X, Y←punto selezionato dall'utente

```
begin
   AWAIT-BUTTON(WAIT, BUTTON-NUM);
   READ-LOCATOR(X, Y);
   return;
end;
```

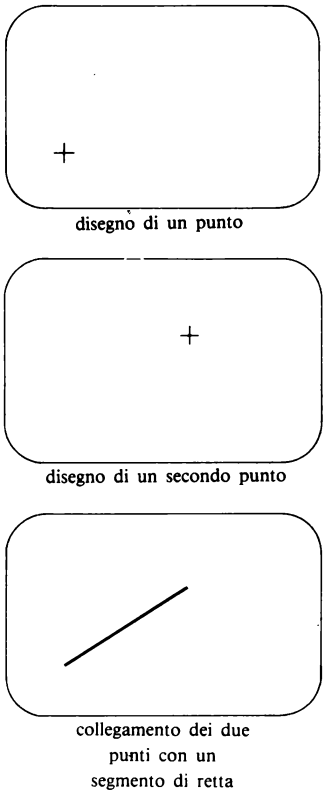


Figura 7.13 Point plotting

Un'altra delle applicazioni dell'operazione di point plotting è l'inserimento di linee formate da segmenti di retta. L'idea è quella di unire più punti successivi, per ottenere delle linee spezzate. L'utente può decidere di continuare o di terminare il disegno premendo un pulsante (vedi Figura 7.13). Un esempio di programma per il point plotting potrebbe essere il seguente:

```
begin
   BUTTON-NUM←CONTINUE;
   while BUTTON-NUM = CONTINUE
      do begin
      AWAIT-BUTTON-GET-LOCATOR(WAIT,
            BUTTON-NUM, X, Y);
      LINE-ABS-2(X, Y);
```

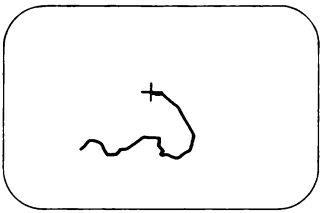


Figura 7.14 Inking

```
        MAKE-PICTURE-CURRENT;
        end;
end;
```

Una variante di questa tecnica è quella che viene detta di *inking*, con la quale, per mezzo di un locator, è possibile far comparire automaticamente sullo schermo una traccia costituita da una spezzata poligonale, come una traccia d'inchiostro lasciata da una penna. Non è necessario che l'utente prema un pulsante per confermare il tracciamento di un segmento; quest'operazione viene fatta automaticamente tutte le volte che il locator si muove di una distanza sufficiente (vedi Figura 7.14). La seguente porzione di programma illustra la tecnica di inking:

```
begin
   PEN-ON←TRUE;
   READ-LOCATOR(XOLD, YOLD);
   while PEN-ON
      do begin
            READ-LOCATOR(X, Y);
            if |X−XOLD)| + |Y−YOLD| >NO-MOVEMENT
               then begin
                     LINE-ABS-2(X, Y);
                     XOLD←X;
                     YOLD←Y;
                     MAKE-PICTURE-CURRENT;
                     end;
            UPDATE(PEN-ON);
            end;
end;
```

Una chiamata alla routine UPDATE(PEN-ON) pone termine all'operazione di inking, non appena si verifica un evento proveniente da un dispositivo qualunque, come ad esempio un pulsante, che può essere rilasciato sollevando la penna, oppure il superamento di un tempo limite, o infine un limite sul numero di

segmenti consentiti. I terminali a vector refresh ed alcuni terminali raster consentono l'uso di un'altra interessante tecnica, detta *rubber band line*. Tale tecnica consente all'utente di verificare come apparirà una retta prima di fissarla in una determinata posizione. Il procedimento è uguale a quello usato per tracciare le rette: una volta che l'estremo del segmento è stato fissato nella posizione voluta, tramite il locator, non resta che confermare la scelta di questo punto premendo un pulsante. La differenza sta nel fatto che il segmento che va dal secondo estremo dell'ultimo segmento introdotto fino alla posizione indicata dal locator, viene immediatamente e costantemente ridisegnato, seguendo gli spostamenti del secondo estremo indicato istante per istante dal locator. In questo modo, si ha l'impressione di un elastico che venga teso fra l'ultimo punto fissato e il punto indicato dal locator (vedi Figura 7.15).
Questa tecnica è difficilmente realizzabile direttamente utilizzando le routine del sistema grafico descritte fino a questo momento. È necessario che si aggiungano altre routine, per modificare porzioni del display file senza aprire e chiudere

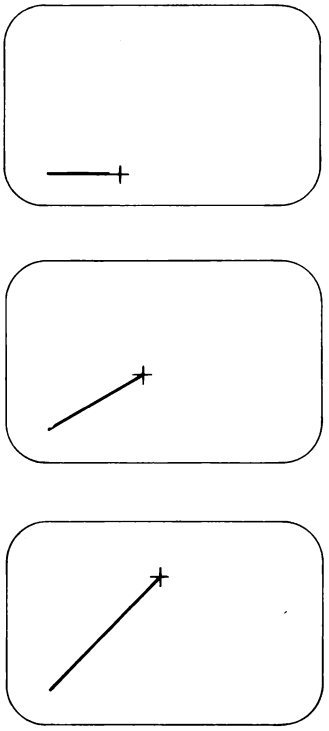


Figura 7.15 Rubber band line

i segmenti, oppure routine per aggiungere nuove istruzioni in un segmento che era già stato chiuso. È possibile però realizzare questa tecnica applicando le trasformazioni di visualizzazione ad una singola retta contenuta in un segmento chiuso ed utilizzando un segmento aperto per contenere il disegno finale definito dall'utente. La stesura dei dettagli per l'implementazione di questa tecnica sarà argomento di un esercizio.

L'utente generalmente non desidera solo creare nuove immagini, ma vuole avere anche la possibilità di modificare e correggere immagini già esistenti. La prima cosa necessaria a questo scopo è l'identificazione della porzione di figura che deve essere cambiata eseguendo una operazione di pick. Se, per esempio, si deve cancellare una parte della figura, si potrà individuarla tramite una chiamata alla routine AWAIT-PICK che permette di scegliere il segmento da rimuovere ed in seguito una chiamata alla routine SET-VISIBILITY permetterà di renderlo invisibile. Si osservi che con questa tecnica non si può ritornare allo stato precedente alla cancellazione, in quanto non è possibile selezionare un segmen-



Figura 7.16 Selezione di un segmento con un pick



Figura 7.17 Posizionamento

to invisibile (vedi Figura 7.16). Un altro tipo di modifica apportabile ad una figura è il cambiamento della trasformazione di visualizzazione. Ad esempio, l'utente può definire una nuova zona dello schermo in cui deve essere disegnato il segmento. Il modo più semplice per indicare questa nuova posizione è quello di indicare un punto sullo schermo utilizzando un locator. L'utente dovrà comportarsi in questo modo: selezionerà il segmento che deve essere traslato per mezzo di un'operazione di pick, quindi dovrà indicare, per mezzo di un locator, un punto dell'immagine che servirà da riferimento; infine, il locator dovrà essere portato nel punto in cui va riposizionato il punto di riferimento dell'immagine, che quindi verrà ridisegnata riferendosi ad esso (vedi Figura 7.17).
Di seguito viene mostrata una porzione di programma che realizza le operazioni sopra descritte. All'interno di questo programma compare la routine INQUIRE-IMAGE-TRANSLATION (TX, TY), che restituisce i parametri della traslazione corrente applicata al segmento.

```
begin
   AWAIT-PICK(WAIT, SEGMENT);
   AWAIT-BUTTON-GET-LOCATOR(WAIT, BUTTON-NUM, XOLD,
      YOLD);
   AWAIT-BUTTON-GET-LOCATOR(WAIT, BUTTON-NUM, XNEW,
      YNEW);
   INQUIRE-IMAGE-TRANSLATION(SEGMENT, TX, TY);
   SET-IMAGE-TRANSLATION(SEGMENT, TX + XNEW - XOLD, TY
      + YNEW - YOLD);
   MAKE-PICTURE-CURRENT;
end;
```

Se si dispone di uno schermo a vector refresh di un terminale raster, l'immagine può essere spostata e ridisegnata con continuità, seguendo gli spostamenti del locator. L'immagine sembra «appesa» al locator e l'utente è quindi in

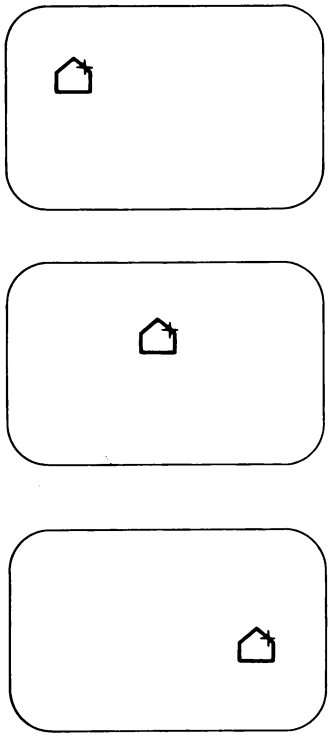


Figura 7.18 Dragging

grado di vedere come appare la figura risultante prima di fissare i valori della traslazione (vedi Figura 7.18): questa tecnica viene detta *dragging* (trascinamento). L'esempio di una possibile procedura di dragging è costituito dal seguente programma:

```
begin
   AWAIT-PICK(WAIT, SEGMENT);
   AWAIT-BUTTON-GET-LOCATOR(WAIT, BUTTON-NUM, XOLD,
   YOLD);
   DRAGGING←TRUE;
   while DRAGGING
   do begin
      INQUIRE-IMAGE-TRANSLATION(SEGMENT, TX, TY);
      READ-LOCATOR(XNEW, YNEW);
      if XNEW≠XOLD or YNEW≠YOLD
         then begin
            SET-IMAGE-TRANSLATION(SEGMENT, TX+XNEW
               -XOLD, TY+YNEW-YOLD);
            XOLD←XNEW;
            YOLD←YNEW
         MAKE-PICTURE-CURRENT;
         end;
   UPDATE(DRAGGING);
   end;
end;
```

La routine UPDATE(DRAGGING), chiamata alla fine di questo programma, serve per interrompere il processo di trascinamento interattivo della figura. Solitamente il trascinamento viene interrotto in corrispondenza di un certo evento, ma può essere anche interrotto nel caso venga superato un tempo limite.
È possibile modificare interattivamente anche il fattore di scala e l'orientamento di una figura: le tecniche sono le stesse usate per il cambiamento interattivo dei parametri di traslazione, tranne per il fatto che il dispositivo di input corrispondente dovrà essere un valuator invece di un locator. Un valuator, comunque, potrà venire simulato per mezzo di un locator, utilizzando una scala disegnata sullo schermo (di solito parallela all'asse x o all'asse y). Il cursore del locator viene posizionato in un punto della scala e la posizione rilevata viene convertita in un valore per il valuator (vedi Figura 7.19).
Un programma interattivo può offrire all'utente molte funzionalità opzionali. Ad esempio, l'utente potrà inserire un segmento di retta, tracciare una linea spezzata, rendere invisibile un segmento e spostarlo; naturalmente avrà anche la possibilità di terminare la sessione di lavoro. L'utente deve selezionare l'opzione che desidera, utilizzando la tastiera o dei pulsanti, associando a ciascuno di essi una particolare operazione. Come si può informare l'utente delle diverse modalità di lavoro che sono a sua disposizione? E come si può indicare attra-

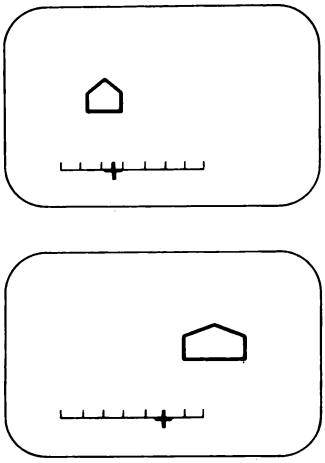


Figura 7.19 Uso di scale graduate

verso quali pulsanti sono richiamabili? Queste esigenze possono venir soddisfatte costruendo una lista di opzioni che viene detta *menu* (vedi Figura 7.20). I diversi menu andranno inseriti in uno o più segmenti (diversi da quelli dedicati a contenere le figure definite dall'utente) che si possono rendere invisibili quando non sono più necessari. Si osservi però che all'utente può risultare sgradevole spostare continuamente l'attenzione tra diversi pulsanti, tasti, locator e dispositivi per il pick. Per questo, ad esempio, alla penna ottica o allo stilo della tavoletta è di solito applicato un interruttore. Ciò evita all'utente di cambiare continuamente il tipo di dispositivo di interazione, permettendogli di usare la penna ottica o lo stilo per indicare i diversi campi del menu. A questo scopo è necessario che ciascuna opzione del menu sia inserita in un segmento distinto, che può essere poi identificato con una operazione di pick; questo tipo di selezione viene detto a *light buttons*. La tecnica, pur essendo analoga a quella che utilizza i pulsanti, consente all'utente di non spostare la sua attenzione dallo schermo.

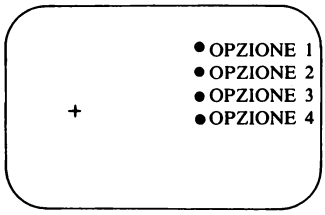


Figura 7.20 Un menu

ESERCIZI

7.1 Progettare la routine RUBBERBAND, che visualizza ripetutamente un segmento di retta che congiunge il punto indicato dal dispositivo locator nel momento in cui viene chiamata la routine, con la posizione corrente del locator. Quindi, quando sopraggiunge un evento di qualsiasi tipo, il segmento aperto al momento della chiamata della routine riceve il comando di disegnare una retta fino alla posizione corrente del locator. Dopodiché si esce dalla routine. Si scriva l'algoritmo RUBBERBAND per i seguenti casi.

a) Il sistema sia esteso con la routine REPLACE-INSTRUCTION(OP, X, Y), che sostituisce l'ultima istruzione del display file con quella indicata dai suoi argomenti.
b) Il sistema sia esteso con la routine EXTEND-SEGMENT(SEGMENT-NAME), che riapre un segmento in modo di aggiungervi una nuova istruzione.
c) Il sistema non sia esteso; bisognerà quindi utilizzare le trasformazioni di visualizzazione per simulare la rubber band line.

7.2 Ricavare la formula che esprime la distanza tra un punto ed una retta.

7.3 Progettare un algoritmo per gestire gli spostamenti della croce che costituisce l'eco della penna ottica.

PROBLEMI DI PROGRAMMAZIONE

7.1 Usando gli algoritmi dal 7.1 al 7.17 come guida, si estenda il sistema grafico in modo da includere le operazioni interattive.

7.2 Si scriva un programma che permette all'utente di creare interattivamente una figura, usando i pulsanti disponibili e il dispositivo locator. Si usi il pulsante numero 1 per indicare che si vuole muovere la penna alla posizione indicata dal locator, senza disegnare nessuna retta (MOVE-ABS-2). Si usi il pulsante numero 2 per indicare che si vuole disegnare una retta (LINE-ABS-2) ed il pulsante 3 per interrompere il programma (TERMINATE).

7.3 Si scriva un programma che realizza la tecnica di inking. Si usi un pulsante per abilitare o disabilitare il tracciamento delle rette (cioè la fuoriuscita di inchiostro). Il programma dovrà campionare la posizione del locator ripetutamente. Se il locator è stato mosso dalla posizione precedente in misura sufficiente, bisogna estendere la linea spezzata prolungandola con un nuovo segmento fino alla posizione corrente del locator.

7.4 Si scriva un programma per spostare una figura. Si selezioni l'immagine che deve essere spostata, utilizzando il pick. Dopo l'esecuzione del pick per individuarla, si legga la posizione del locator; a questo punto occorre attendere che venga premuto un pulsante. Quando accade questo secondo evento, si deve leggere nuovamente la posizione del locator. Infine si deve determinare la variazione di posizione del locator e si applica questa variazione alla trasformazione che viene usata per visualizzare il segmento precedentemente selezionato.

7.5 a) Si scriva un programma che crea tre figure (un triangolo, un quadrato e una croce) in tre differenti segmenti del display file. Inizialmente l'attributo di visibilità delle tre figure sarà posto a FALSE. Si visualizzi un menu che permetta all'utente di rendere visibili le figure premendo opportuni pulsanti.

b) Si inserisca ciascuna delle diverse opzioni del menu in diversi segmenti del display file, in modo che i campi del menu possano essere selezionati con un'operazione di pick invece che con il pulsante.
c) Si faccia in modo che si possa rendere invisibile una delle figure precedentemente visualizzate, indicandole direttamente sullo schermo con un'operazione di pick.

***7.6** Modificando la trasformazione di visualizzazione di un segmento chiuso che contiene un segmento orizzontale di lunghezza 1, si implementi il meccanismo della rubber band line.

****7.7** Spesso, durante la costruzione di una figura, l'utente desidera unire l'estremo di un segmento con l'estremo di un altro. Posizionare correttamente il locator nei punti esatti che devono essere uniti è una operazione difficile e noiosa. Si semplifichi questa operazione, scrivendo una routine che restituisce come punto l'estremo del segmento vicino al quale si trova il locator, utilizzando come guida l'algoritmo 7.16.

8

Grafica tridimensionale

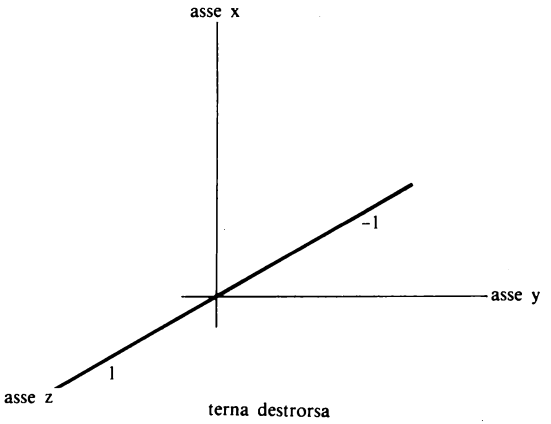
INTRODUZIONE

Gran parte delle applicazioni grafiche sono bidimensionali: diagrammi e grafici, mappe di vario genere e molti tipi di immagini artistiche e pubblicitarie possono venir disegnate a due sole dimensioni. Essendo però il mondo tridimensionale, vi sono molte applicazioni nel campo della progettazione e del design che necessitano di descrivere oggetti tridimensionali. Se ad esempio un architetto vuole vedere l'aspetto complessivo della struttura che ha progettato, gli sarà necessario un modello tridimensionale che possa venir osservato da differenti angolazioni. Analogamente un progettista di velivoli richiede un modello tridimensionale per la simulazione del comportamento aerodinamico, per l'analisi degli sforzi e delle deformazioni delle strutture che egli progetta. Altri tipi di applicazioni, come ad esempio la simulazione della messa in orbita di una nave spaziale o dell'atterraggio di un areoplano, richiedono necessariamente una rappresentazione tridimensionale nel sistema.

In questo capitolo si espanderà il sistema grafico per trattare le rappresentazioni degli oggetti in uno spazio tridimensionale. Le trasformazioni geometriche verranno estese per poter effettuare traslazioni nelle tre direzioni e rotazioni attorno ai tre assi coordinati. Poiché si dispone di una superficie piana per la visualizzazione delle informazioni grafiche, occorrerà determinare un metodo per proiettare oggetti tridimensionali sulla superficie piana dello schermo nella quale compare l'immagine. Verranno discussi, a questo scopo, i metodi di proiezione parallela e di proiezione prospettica.

8.1 GEOMETRIA 3-D

Viene ora trattata la geometria dello spazio tridimensionale, rivedendo brevemente i metodi della geometria analitica per definire le entità geometriche. L'oggetto più semplice da cui iniziare è, naturalmente, il punto. Analogamente al caso bidimensionale, è possibile definire un punto fissando un sistema di assi coordinati come riferimento e specificando le coordinate del punto rispetto ad esso. Si può estendere quindi il sistema di riferimento aggiungendo un asse per



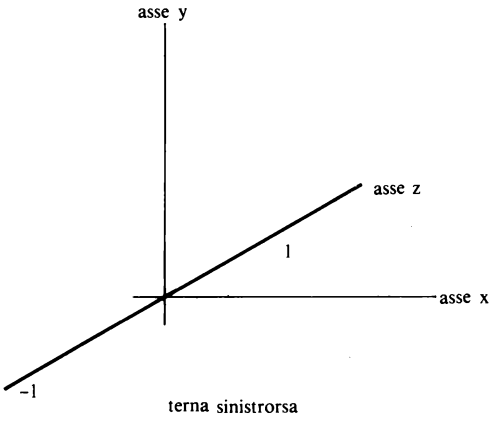


Figura 8.1 Sistemi di coordinate relativi alle terne destrorse e sinistrorse

la terza dimensione (gli assi quindi rappresenteranno la larghezza, la profondità e l'altezza). I tre assi ortogonali che costituiscono la terna di riferimento verranno etichettati con le lettere x, y (gli stessi assi che corrispondono a quelli del sistema di riferimento bidimensionale) e z (l'asse aggiunto per la terza dimensione). L'ultima cosa che resta da definire per questa terna di riferimento è quale sia, delle due possibili, la direzione positiva per l'asse z (vedi Figura 8.1). Se si fa corrispondere all'asse y il dito medio della mano destra e all'asse x si fa corrispondere l'indice, allora il pollice è diretto nella direzione positiva di un sistema di riferimento che viene chiamato *terna destrorsa*. Se invece si fa corrispondere agli assi x e y le medesime dita della mano sinistra, allora il pollice risulta diretto nella direzione opposta e il sistema di riferimento si dice essere una *terna sinistrorsa*. Sebbene nella maggior parte dei trattati di geometria si consideri come sistema di riferimento una terna destrorsa, nell'ambito della Computer Graphics viene più spesso usata la terna sinistrorsa. In questo testo comunque si farà riferimento ad una terna destrorsa. Quindi un punto sarà definito dalla tripla di numeri (x, y, z) secondo questa convenzione (vedi Figura 8.2). Nelle rappresentazioni bidimensionali una retta viene definita come luogo dei punti che verifica l'identità in funzione delle coordinate x e y:

$$\frac{y-y_1}{x-x_1} = \frac{y_2-y_1}{x_2-x_1} \tag{8.1}$$

Questa equazione è tale che se x cambia valore la coordinata y deve cambiare proporzionalmente. Il rapporto tra x e y resta cioè costante. Se si estende questa equazione alle tre dimensioni, una variazione della coordinata x deve riper-

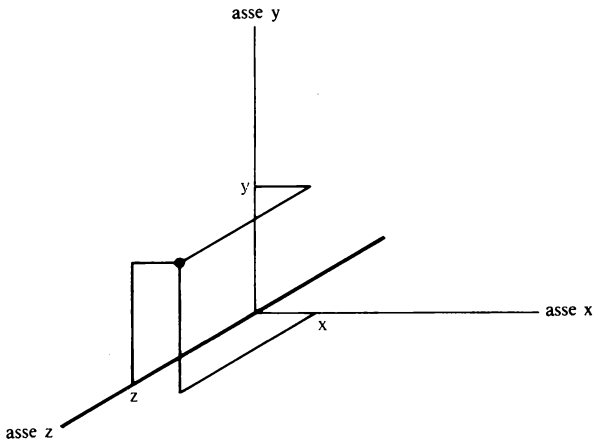


Figura 8.2 Posizione del punto (x, y, z)

cuotersi proporzionalmente sia sul valore della coordinata y che sul valore della z. Una retta nello spazio 3-D può quindi essere definita dalle due equazioni

$$\begin{aligned} \frac{y-y_1}{x-x_1} &= \frac{y_2-y_1}{x_2-x_1} \\ \frac{z-z_1}{x-x_1} &= \frac{z_2-z_1}{x_2-x_1} \end{aligned} \tag{8.2}$$

Analogamente alla retta nel piano, questa equazione della retta nello spazio 3-D è definita da due punti (x_1, y_1, z_1) e (x_2, y_2, z_2). Per questa equazione una forma più elegante è la forma parametrica, che si ottiene esprimendo le tre coordinate nei termini di un solo parametro u.

$$\begin{aligned} x &= (x_2-x_1)u+x_1 \\ y &= (y_2-y_1)u+y_1 \\ z &= (z_2-z_1)u+z_1 \end{aligned} \tag{8.3}$$

Per questi scopi, occorre anche poter rappresentare un piano; l'equazione che rappresenta il luogo dei punti che lo compongono è del tipo seguente:

$$Ax+By+Cz+D = 0 \tag{8.4}$$

Si osservi che è possibile semplificare l'equazione del piano dividendo per una costante (ad esempio per il valore di A), ottenendo

$$x+B_1y+C_1z+D_1 = 0 \tag{8.5}$$

nella quale i coefficienti sono nella seguente relazione:

$$B_1 = B/A, \qquad C_1 = C/A \qquad \text{e} \qquad D_1 = D/A \tag{8.6}$$

Si può quindi concludere che per definire un piano occorrono solo tre costanti e cioè i valori dei parametri B_1, C_1, D_1. L'equazione di un piano può essere definita specificando le coordinate di tre punti, purché non appartenenti alla stessa retta. Si abbiano ad esempio le coordinate di tre punti (x_1, y_1, z_1), (x_2, y_2, z_2) e (x_3, y_3, z_3): è possibile determinare l'equazione del piano nella forma (8.5) imponendo che l'equazione del piano venga verificata dalle coordinate di ciascuno dei tre punti dati.

$$\begin{aligned} x_1+B_1y_1+C_1z_1+D_1 &= 0 \\ x_2+B_1y_2+C_1z_2+D_1 &= 0 \\ x_3+B_1y_3+C_1z_3+D_1 &= 0 \end{aligned} \tag{8.7}$$

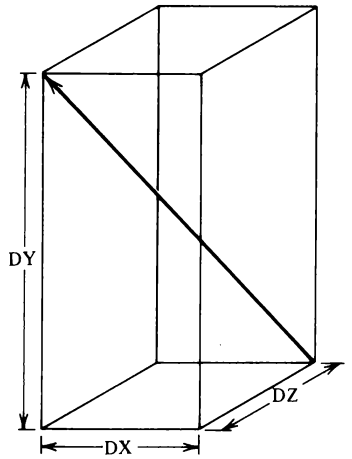


Figura 8.3 Vettore in tre dimensioni

In questo modo, si ha un sistema di tre equazioni da risolvere rispetto alle incognite B_1, C_1, D_1. Con pochi passaggi algebrici è possibile ottenere i loro valori in funzione delle coordinate dei punti noti che definiscono il piano. Un altro modo per definire un piano consiste nel fornire un punto e la direzione perpendicolare ad esso. Il vettore perpendicolare al piano che serve per specificare la direzione che lo definisce viene detto *vettore normale*. Verranno chiamate [*DXN, DYN, DZN*] le componenti del vettore normale e (x_p, y_p, z_p) le coordinate del punto che giace nel piano (vedi Figure 8.3 e 8.4).

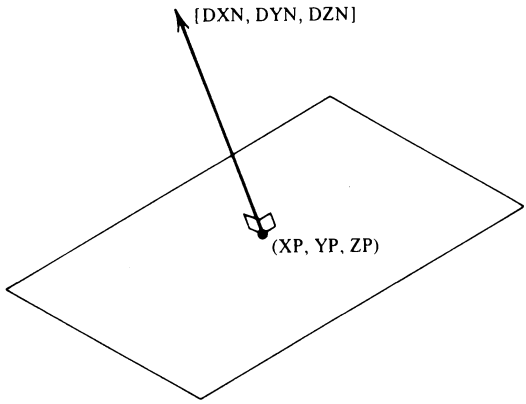


Figura 8.4 Vettore normale a un piano

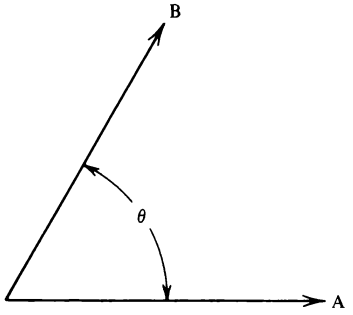


Figura 8.5 Angolo compreso tra due vettori

Preso un punto arbitrario (x, y, z) che appartiene al piano, il vettore $[x-x_p, y-y_p, z-z_p]$ è interamente contenuto nel piano. Utilizzando queste informazioni, si può trovare l'equazione del piano per mezzo dell'operazione di prodotto scalare tra vettori. Il prodotto scalare tra due vettori è la sommatoria dei prodotti tra le componenti corrispondenti. Se ad esempio $A = [A_x, A_y, A_z]$ e $B = [B_x, B_y, B_z]$, il risultato del prodotto scalare tra i due vettori è:

$$A \cdot B = A_x B_x + A_y B_y + A_z B_z \tag{8.8}$$

Questo valore si ottiene anche dal prodotto delle lunghezze dei due vettori moltiplicato per il coseno dell'angolo tra essi compreso (vedi Figura 8.5).

$$A \cdot B = |A||B|\cos\theta \tag{8.9}$$

Nel caso in cui l'angolo tra un qualunque vettore che giace nel piano e il vettore normale sia $\pi/2$ radianti (come è ovvio dalla definizione di vettore normale), il coseno di quest'angolo è uguale a 0. Quindi il risultato del prodotto tra un vettore appartenente al piano e il vettore normale fornisce l'equazione cercata

$$DXN(x-x_p) + DYN(y-y_p) + DZN(z-z_p) = 0 \tag{8.10}$$

che è verificata per qualunque punto (x, y, z) appartenente al piano.

8.2 PRIMITIVE 3-D

Dopo aver visto come è possibile definire punti, rette e piani nello spazio 3-D, è possibile ora costruire modelli di oggetti solidi. All'utente del sistema grafico sono stati forniti dei comandi per muovere una penna sullo schermo e tracciare rette e poligoni su una superficie bidimensionale. Se quindi si vogliono estende-

re questi comandi alle tre dimensioni, basterà richiedere all'utente tre coordinate per definire un punto anziché due. Per visualizzare questi comandi di tracciamento 3-D, occorre un'operazione di proiezione che verrà eseguita da una opportuna routine, chiamata DISPLAY-FILE-ENTER, che verrà descritta nei successivi paragrafi. Analogamente al caso 2-D, l'utente utilizzerà per il tracciamento di segmenti nello spazio una penna mossa da opportuni comandi e si manterrà l'informazione della posizione corrente della penna nelle variabili globali DF-PEN-X, DF-PEN-Y, DF-PEN-Z. Di seguito vengono presentati gli algoritmi per eseguire le operazioni MOVE e LINE nello spazio 3-D.

Algoritmo 8.1 MOVE-ABS-3 (X, Y, Z)
Muove la penna nello spazio 3-D.
Argomenti X, Y ,Z←coordinate assolute del punto in cui muovere la penna
Variabili globali DF-PEN-X, DF-PEN-Y, DF-PEN-Z←posizione corrente della penna

```
begin
   DF-PEN-X←X;
   DF-PEN-Y←Y;
   DF-PEN-Z←Z;
   DISPLAY-FILE-ENTER(1);
   return;
end;
```

Algoritmo 8.2 MOVE-REL-3 (DX, DY, DZ)
Muove in modo relativo la penna nello spazio 3-D.
Argomenti DX, DY, DZ←coordinate relative del punto in cui muovere la penna
Variabili globali DF-PEN-X, DF-PEN-Y, DF-PEN-Z←posizione corrente della penna

```
begin
   DF-PEN-X←DF-PEN-X+DX;
   DF-PEN-Y←DF-PEN-Y+DY;
   DF-PEN-Z←DF-PEN-Z+DZ;
   DISPLAY-FILE-ENTER(1);
   return;
end;
```

Algoritmo 8.3 LINE-ABS-3 (X, Y, Z)
Disegna un segmento di retta nello spazio 3-D.
Argomenti X, Y, Z←coordinate del punto da raggiungere muovendo la penna e tracciando il segmento
Variabili globali DF-PEN-X, DF-PEN-Y, DF-PEN-Z←posizione corrente della penna

```
begin
```

```
    DF-PEN-X←X;
    DF-PEN-Y←Y;
    DF-PEN-Z←Z;
    DISPLAY-FILE-ENTER(2);
    return;
end;
```

Algoritmo 8.4 LINE-REL-3 (DX, DY, DZ)
Disegna un segmento di retta nello spazio 3-D in modo relativo.
Argomenti DX, DY, DZ←coordinate relative del punto da raggiungere muovendo la penna e tracciando il segmento
Variabili globali DF-PEN-X, DF-PEN-Y, DF-PEN-Z←posizione corrente della penna

```
begin
    DF-PEN-X←DF-PEN-X + DX;
    DF-PEN-Y←DF-PEN-Y + DY;
    DF-PEN-Z←DF-PEN-Z + DZ;
    DISPLAY-FILE-ENTER(2);
    return;
end;
```

Analogamente, possono essere adattati i comandi per tracciare i poligoni, aggiungendo un ulteriore array per la terza coordinata dei punti che compongono il poligono. Di seguito vengono presentate le versioni degli algoritmi relativi ai poligoni adattate al caso 3-D.

Algoritmo 8.5 POLYGON-ABS-3 (AX, AY, AZ, N)
Disegna un poligono nello spazio 3-D.
Argomenti N←numero dei lati del poligono
AX, AY, AZ←array contenenti le coordinate dei vertici
Variabili globali DF-PEN-X, DF-PEN-Y, DF-PEN-Z←posizione corrente della penna
Variabili locali I←indice che percorre i lati del poligono

```
begin
    if N>31 or N<3 then return-error 'ERRORE NEI LATI DEL POLI-
        GONO';
    DF-PEN-X←AX[N];
    DF-PEN-Y←AY[N];
    DF-PEN-Z←AZ[N];
    DISPLAY-FILE-ENTER(N);
    for I = 1 to N do LINE-ABS-3(AX[I], AY[I], AZ[I]);
    return;
end;
```

Algoritmo 8.6 POLYGON-REL-3 (AX, AY, AZ, N)
Disegna un poligono nello spazio 3-D in modo relativo.
Argomenti N←numero dei lati del poligono
AX, AY, AZ←array contenenti le coordinate dei vertici del poligono
Variabili globali DF-PEN-X, DF-PEN-Y, DF-PEN-Z←posizione corrente della penna
Variabili locali I←indice relativo ai lati del poligono
TMPX, TMPY, TMPZ←coordinate del punto di chiusura del poligono

```
begin
  if N>31 or N<3 then return error 'ERRORE NEI LATI DEL POLIGONO';
  muove la penna fino al vertice di partenza
  DF-PEN-X←DF-PEN-X+AX[1];
  DF-PEN-Y←DF-PEN-Y+AY[1];
  DF-PEN-Z←DF-PEN-Z+AZ[1];
  memorizza il vertice di chiusura del poligono
  TMPX←DF-PEN-X;
  TMPY←DF-PEN-Y;
  TMPZ←DF-PEN-Z;
  DISPLAY-FILE-ENTER(N);
  inserisce i lati del poligono
  for I = 2 to N do LINE-REL-3(AX[I], AY[I], AZ[I]);
  chiude il poligono
  LINE-ABS-3(TMPX, TMPY, TMPZ);
  return;
end;
```

8.3 TRASFORMAZIONI 3-D

L'uso delle routine descritte nel paragrafo precedente consente di tracciare un disegno dell'oggetto che si vuole rappresentare nello spazio 3-D. Al fine di visualizzare, rispetto ad un particolare punto di vista, le linee tracciate, verrà definita una trasformazione che esegue una proiezione sulla superficie dello schermo. Questa trasformazione può essere usata, come si è già visto, per modificare il fattore di scala, ruotare o traslare le immagini. A questo scopo è necessario innanzitutto estendere anche le trasformazioni geometriche in modo che possano essere applicate nelle tre dimensioni. Una matrice di trasformazione bidimensionale ha la forma

$$\begin{vmatrix} S_x & 0 \\ 0 & S_y \end{vmatrix}$$

che in coordinate omogenee diventa

$$\begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Nel caso 3-D, viene aggiunta la terza coordinata, che dovrà quindi avere sulla diagonale della matrice il corrispondente fattore di scala: per questo motivo la matrice di trasformazione sarà di dimensione 3×3 e quindi del tipo seguente

$$\begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{vmatrix} \tag{8.11.1}$$

In coordinate omogenee la matrice diventa di dimensioni 4×4

$$S = \begin{vmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{8.11.2}$$

Per trasformare le coordinate di un punto, bisogna eseguire il prodotto di matrici come indicato nel Capitolo 4.

$$|x_1 y_1 z_1 w_1| = |x\ y\ z\ w| \begin{vmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = |S_x x \quad S_y y \quad S_z z \quad w| \tag{8.12}$$

La riga inferiore della matrice viene utilizzata per i valori dei fattori di traslazione T_x, T_y, T_z; la matrice di trasformazione corrispondente alla traslazione è quindi la seguente:

$$T = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{vmatrix} \tag{8.13}$$

Per le trasformazioni di rotazione, è stata costruita una matrice per il caso bidimensionale che realizza una rotazione attorno all'origine dei due assi coordinati.

$$\begin{vmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{vmatrix}$$

Si può generalizzare questa matrice allo spazio 3-D osservando che una rotazio-

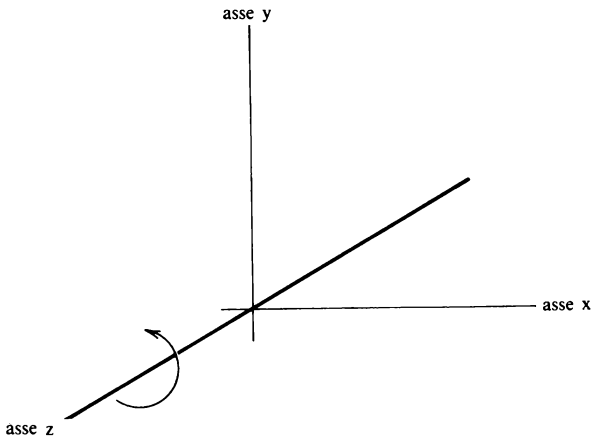


Figura 8.6 Rotazione attorno all'asse z

ne attorno all'asse z mantiene immutati i valori delle coordinate z e cambia opportunamente i valori delle x e delle y come nel caso 2-D (vedi Figura 8.6). La matrice che realizza la rotazione attorno all'asse z quindi è

$$R_z = \begin{vmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{8.14}$$

La trasformazione precedente può essere vista in due modi: pensando che mentre gli assi sono fissi l'oggetto ruoti, oppure che mentre l'oggetto resta fermo viene mosso il sistema di riferimento. La differenza tra queste due modalità di operare riguarda il senso della rotazione: una rotazione antioraria dell'oggetto rispetto agli assi è equivalente a una rotazione oraria degli assi rispetto all'oggetto.
Le rotazioni attorno agli assi x e y saranno eseguite tramite una matrice della stessa forma di quella indicata nella (8.14) come consegue dalla simmetria degli assi. Una rotazione attorno all'asse x sarà quindi (vedi Figura 8.7):

$$R_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{8.15}$$

Una rotazione rispetto all'asse y sarà invece realizzata dalla matrice di trasformazione (vedi Figura 8.8):

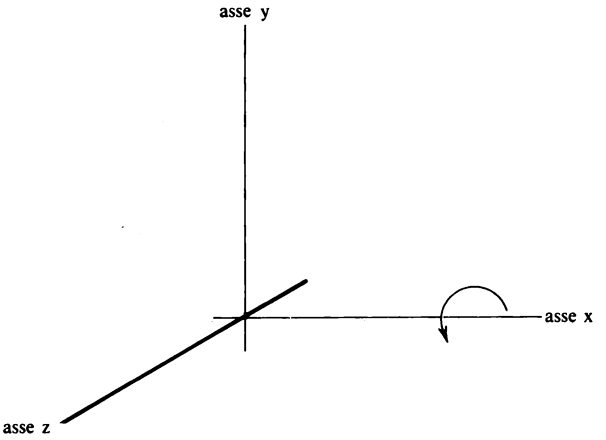


Figura 8.7 Rotazione attorno all'asse x

$$R_y = \begin{vmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{8.16}$$

Queste trasformazioni serviranno per comporre le trasformazioni di visualizzazione. Gli algoritmi per creare queste matrici di trasformazione inizieranno con

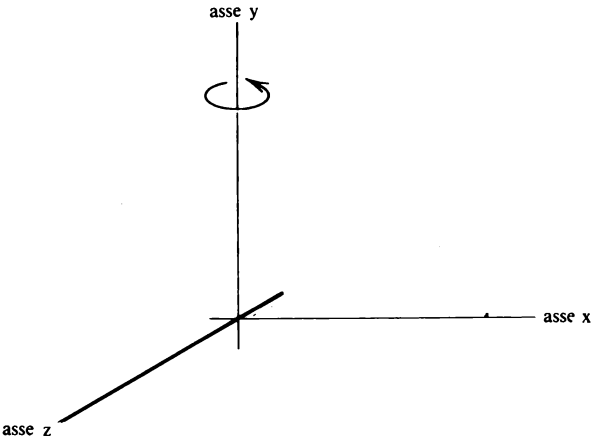


Figura 8.8 Rotazione attorno all'asse y

la creazione della matrice identità (NEW-TRANSFORMATION-3). Sarà possibile poi comporle, moltiplicando la matrice di trasformazione corrente per l'opportuna matrice di rotazione, di traslazione o di trasformazione di scala. Si ricordi che — come per il caso bidimensionale — la quarta colonna di una qualsiasi matrice di trasformazione è sempre composta di tre 0 e di un 1, quindi non sarà necessario memorizzare questa colonna e basterà avere una matrice 4×3. Questa matrice di trasformazione sarà memorizzata nella variabile TMATRIX. L'algoritmo seguente assegna a TMATRIX i valori della matrice identità.

Algoritmo 8.7 NEW-TRANSFORMATION-3
Inizializza la matrice identità.
Variabili globali TMATRIX←matrice di trasformazione 4×3
Variabili locali I, J←indici relativi alle righe e alle colonne della matrice

```
begin
   for I = 1 to 4
   do begin
         for J = 1 to 3 do TMATRIX[I, J]←0;
         if I≠4 then TMATRIX[I, I]←1;
         end;
   return;
end;
```

L'algoritmo seguente moltiplica la matrice TMATRIX per una traslazione.

Algoritmo 8.8 TRANSLATE-3 (TX, TY, TZ)
Genera la matrice di trasformazione relativa ad una traslazione.
Argomenti TX, TY, TZ←valori delle traslazioni lungo i tre assi coordinati
Variabili globali TMATRIX←matrice di trasformazione 4×3

```
begin
   TMATRIX[4, 1]←TMATRIX[4, 1]+TX;
   TMATRIX[4, 2]←TMATRIX[4, 2]+TY;
   TMATRIX[4, 3]←TMATRIX[4, 3]+TZ;
   return;
end;
```

Di seguito vengono descritti gli algoritmi per le rotazioni rispetto agli assi principali. Come argomenti sono stati introdotti per brevità il coseno e il seno dell'angolo invece del loro valore in radianti.

Algoritmo 8.9 ROTATE-X-3(S, C)
Genera la matrice di trasformazione relativa ad una rotazione attorno all'asse x (y verso z)
Argomenti S, C←seno e coseno dell'angolo di rotazione
Variabili globali TMATRIX←matrice di trasformazione 4×3

Variabili locali I←indice relativo alle righe ed alle colonne della matrice
TMP←variabile di memorizzazione temporanea

```
begin
   for I = 1 to 4
   do begin
      TMP←TMATRIX[I, 2]*C−TMATRIX[I, 3]*S;
      TMATRIX[I, 3]←TMATRIX[I, 2]*S+TMATRIX[I, 3]*C;
      TMATRIX[I, 2]←TMP;
      end;
   return;
end;
```

Algoritmo 8.10 ROTATE-Y-3 (S, C)
Genera la matrice di trasformazione relativa ad una rotazione attorno all'asse y (z verso x).

Argomenti S, C←seno e coseno dell'angolo di rotazione
Variabili globali TMATRIX←matrice di trasformazione 4×3
Variabili locali I←indice relativo alle righe ed alle colonne della matrice
TMP←variabile di memorizzazione temporanea

```
begin
   for I = 1 to 4
   do begin
      TMP←TMATRIX[I, 1]*C+TMATRIX[I, 3]*S;
      TMATRIX[I, 3]←TMATRIX[I, 1]*S+TMATRIX[I, 3]*C;
      TMATRIX[I, 1]←TMP;
      end;
   return;
end;
```

Algoritmo 8.11 ROTATE-Z-3 (S, C)
Genera la matrice di trasformazione relativa ad una rotazione attorno all'asse z (x verso y).

Argomenti S, C←seno e coseno dell'angolo di rotazione
Variabili globali TMATRIX←matrice di trasformazione 4×3
Variabili locali I←indice relativo alle righe ed alle colonne della matrice
TMP←variabile di memorizzazione temporanea

```
begin
   for I = 1 to 4
   do begin
      TMP←TMATRIX[I, 1]*C−TMATRIX[I, 2]*S;
      TMATRIX[I, 2]←TMATRIX[I, 1]*S+TMATRIX[I, 2]*C;
      TMATRIX[I, 1]←TMP;
      end;
   return;
end;
```

8.4 ROTAZIONE ATTORNO A UN ASSE ARBITRARIO

Per familiarizzare con le trasformazioni tridimensionali si proverà a risolvere un problema. Anche se sono state finora sviluppate le trasformazioni di rotazione attorno ad uno degli assi coordinati, in generale una qualsiasi retta dello spazio può essere considerata asse di rotazione. Il problema è allora quello di ricavare una matrice di trasformazione per una rotazione di un certo angolo θ attorno ad una retta arbitraria. Innanzitutto deve essere applicata una traslazione per riportare l'origine del sistema di riferimento sull'asse di rotazione e si devono effettuare due rotazioni, una attorno all'asse x e una attorno all'asse y, in modo da far coincidere l'asse di rotazione con l'asse z, cosicché la rotazione attorno all'asse arbitrario diventi una rotazione attorno all'asse z. Infine, si applicano le trasformazioni inverse per le due rotazioni attorno agli assi y e x e per la traslazione effettuata in precedenza, in modo da rimettere la retta e le coordinate nel sistema di riferimento originario (vedi Figura 8.9).
Una rappresentazione conveniente per la retta che va usata come asse di rotazione si ottiene assegnando la sua direzione e un punto ad essa appartenente. Tale definizione è sufficiente per questo scopo, poiché il punto fornisce le informazioni per la traslazione, e la direzione suggerisce gli angoli di rotazione necessari per l'allineamento con l'asse z. Data la retta in forma parametrica

$$\begin{aligned} x &= Au + x_1 \\ y &= Bu + y_1 \\ z &= Cu + z_1 \end{aligned} \tag{8.17}$$

un punto su di essa è (x_1, y_1, z_1), mentre la direzione è definita dal vettore $[A, B, C]$. Si può ora scrivere la matrice di trasformazione per la rotazione attorno ad un asse generico. La traslazione iniziale per portare l'origine sull'asse di rotazione è data dalla

$$T = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_1 & -y_1 & -z_1 & 1 \end{vmatrix} \tag{8.18}$$

Applicando questa trasformazione, il punto (x_1, y_1, z_1) sulla retta verrà spostato nell'origine. La traslazione della stessa quantità — ma in direzione opposta — necessaria per riportare l'origine nella posizione di partenza è la seguente:

$$T^{-1} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_1 & y_1 & z_1 & 1 \end{vmatrix} \tag{8.19}$$

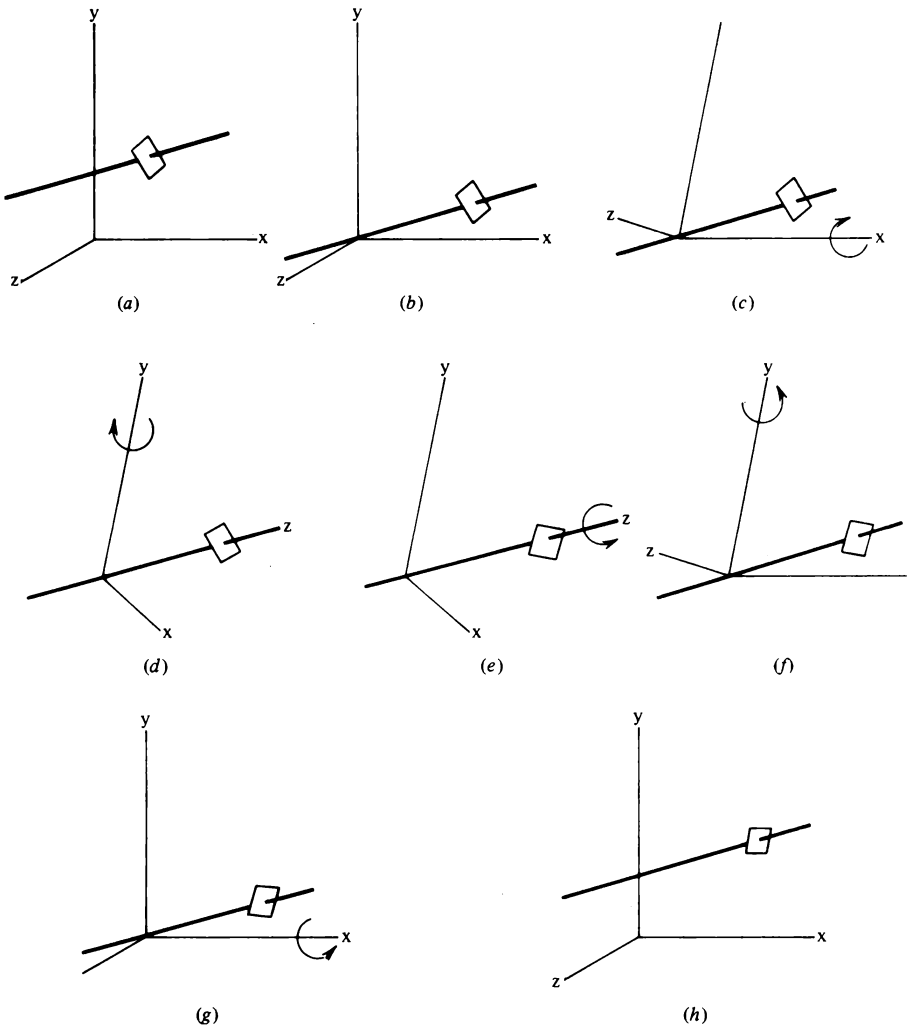


Figura 8.9 (*a*) Per eseguire una rotazione attorno a un asse arbitrario, (*b*) si trasla l'asse nell'origine, (*c*) lo si ruota attorno all'asse x fino a portarlo a giacere nel piano xz, (*d*) lo si ruota attorno all'asse y fino a farlo coincidere con l'asse z, (*e*) si esegue la rotazione richiesta attorno all'asse z (che ora coincide con l'asse di rotazione) invece di fissare l'oggetto e ruotare gli assi (in questa figura sono stati fissati gli assi e si è ruotato l'oggetto), quindi (*f*) si effettua una rotazione inversa attorno all'asse y, (*g*) una seconda rotazione inversa attorno all'asse x e infine (*h*) una traslazione inversa alla prima operazione effettuata.

Con una prima rotazione attorno all'asse x, si dovrà far in modo che l'asse di rotazione giaccia nel piano xz. Per determinare il valore della rotazione si proietterà l'asse di rotazione sul piano yz; la proiezione sul piano yz può essere costruita nel seguente modo: il segmento congiungente i punti (0, 0, 0) e (A, B, C) è diretto come l'asse e, poiché il sistema di riferimento è stato traslato in modo che l'origine appartenga all'asse, il segmento giace interamente lungo l'asse stesso. Proiettando il segmento sul piano yz si ottiene un segmento con estremi di coordinate (0, 0, 0) e (0, B, C) (vedi Figura 8.10).
Se si ruota attorno all'asse x di un angolo I sufficiente affinché l'asse arbitrario si trovi nel piano xz, il segmento proiettato giacerà sull'asse z e quindi è possibile far coincidere il generico asse di rotazione con l'asse z. Quanto vale quest'angolo I? (vedi Figura 8.11). Si sa che la lunghezza V del segmento proiettato avente gli estremi in (0, 0, 0) e (0, B, C) vale

$$V = (B^2 + C^2)^{1/2} \tag{8.20}$$

e dalla definizione di seno e coseno, si ricava

$$\begin{aligned} \sin I &= B/V \\ \cos I &= C/V \end{aligned} \tag{8.21}$$

La rotazione attorno all'asse x sarà quindi

$$R_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos I & \sin I & 0 \\ 0 & -\sin I & \cos I & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{8.22}$$

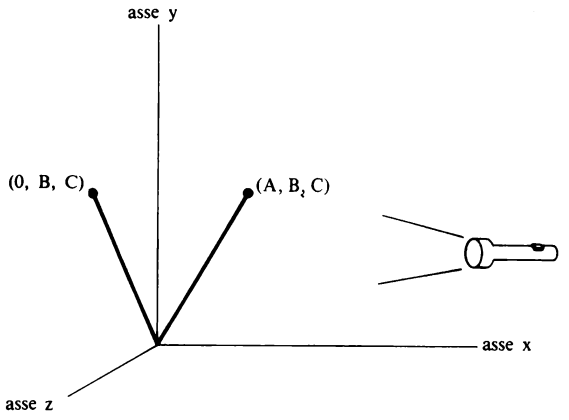


Figura 8.10 Proiezione di un segmento di retta sul piano yz

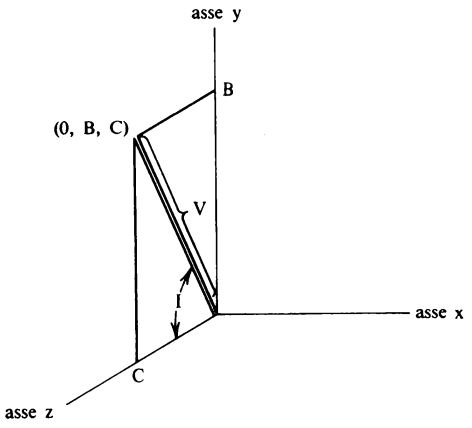


Figura 8.11 Parametri relativi alla proiezione di un segmento di retta

Sostituendo i parametri direttori dell'asse di rotazione, si avrà

$$R_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & C/V & B/V & 0 \\ 0 & -B/V & C/V & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{8.23}$$

La trasformazione inversa può essere ottenuta cambiando il segno dell'angolo di rotazione, che ha l'effetto di cambiare il segno del seno, ma non quello del coseno. La rotazione inversa risulta quindi essere

$$R_x^{-1} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & C/V & -B/V & 0 \\ 0 & B/V & C/V & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{8.24}$$

Si deve costruire ora la rotazione che porta il generico asse nel piano xz (vedi Figura 8.12). La lunghezza del segmento di estremi (0, 0, 0) e (A, B, C)

$$L = (A^2 + B^2 + C^2)^{1/2} \tag{8.25}$$

è immutata; la coordinata z misura quindi

$$(L^2 - A^2)^{1/2} = (B^2 + C^2)^{1/2} = V \tag{8.26}$$

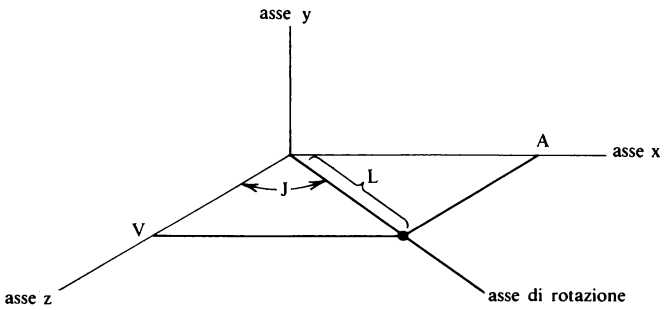


Figura 8.12 Asse di rotazione giacente nel piano xz

L'angolo J della rotazione da effettuare attorno all'asse y in modo che il segmento si allinei con l'asse z risulta essere:

$$\begin{aligned} \sin J &= A/L \\ \cos J &\doteq V/L \end{aligned} \tag{8.27}$$

La matrice di rotazione è quindi

$$R_y = \begin{vmatrix} \cos J & 0 & \sin J & 0 \\ 0 & 1 & 0 & 0 \\ -\sin J & 0 & \cos J & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} V/L & 0 & A/L & 0 \\ 0 & 1 & 0 & 0 \\ -A/L & 0 & V/L & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{8.28}$$

E la trasformazione inversa è

$$R_y^{-1} = \begin{vmatrix} V/L & 0 & -A/L & 0 \\ 0 & 1 & 0 & 0 \\ A/L & 0 & V/L & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{8.29}$$

Ora si può effettuare la rotazione dell'angolo θ attorno all'asse arbitrario. Poiché questo asse è stato allineato con l'asse z, è sufficiente una rotazione di θ attorno all'asse z.

$$R_z = \begin{vmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{8.30}$$

Finalmente si è in grado di completare la matrice di trasformazione che realizza

la rotazione rispetto all'asse arbitrario di un angolo pari a θ, moltiplicando tra loro le matrici che sono state costruite:

$$R_\theta = TR_xR_yR_zR_y^{-1}R_x^{-1}T^{-1} \tag{8.31}$$

8.5 PROIEZIONE PARALLELA

In questo paragrafo si vedrà come utilizzare le trasformazioni esaminate, per proiettare sulla superficie piana di visualizzazione gli oggetti tridimensionali (vedi Figura 8.13).
Il modo più semplice per eseguire quest'operazione consiste nell'ignorare la coordinata z. Questo è un caso particolare del metodo di proiezione, che viene detto *proiezione parallela*. Una proiezione parallela si costruisce per mezzo di

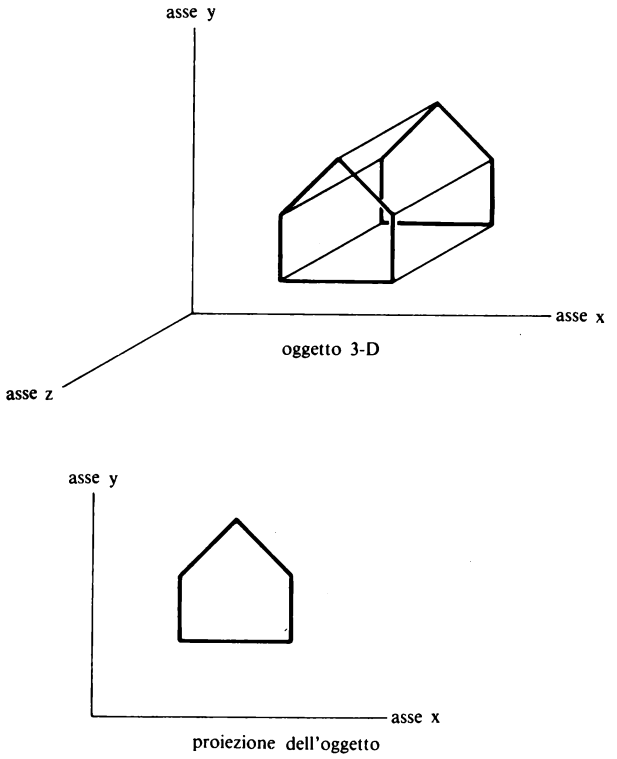


Figura 8.13 Un oggetto tridimensionale e la sua proiezione

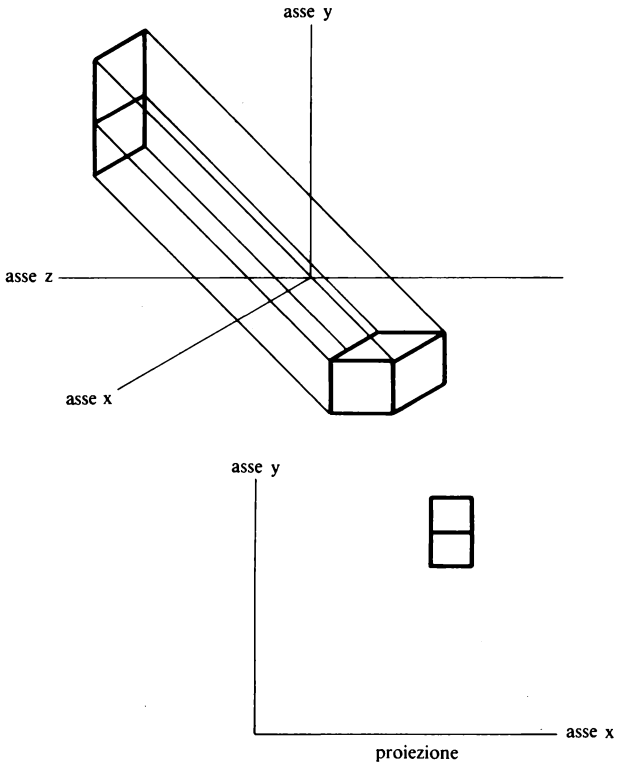


Figura 8.14 Proiezione parallela

un fascio di rette parallele, ciascuna delle quali passa per uno dei vertici dell'oggetto. Il punto di intersezione di una di queste rette con il piano di proiezione è la proiezione del vertice considerato. Connettendo con dei segmenti le proiezioni dei vertici, nello stesso modo in cui sono collegati nell'oggetto, si otterrà la sua immagine proiettata (vedi Figura 8.14). Il caso particolare di proiezione parallela, che consiste nell'ignorare la coordinata z, è il caso in cui lo schermo su cui proiettare sia parallelo al piano xy e il fascio delle rette di proiezione è costituito da rette parallele all'asse z. Se ci si muove infatti lungo le linee di proiezione, si ha una variazione della sola coordinata z. Perciò anche il punto di intersezione dei raggi di proiezione con il piano avrà le stesse coordinate x e y del vertice appartenente all'oggetto che si sta proiettando. In una generica proiezione parallela, è possibile scegliere qualsiasi valore per la direzione delle linee di proiezione, che quindi non saranno sempre ortogonali al piano di proiezione.

Indicando la direzione della proiezione con il vettore $[x_p, y_p, z_p]$ e supponendo

di proiettare nel piano xy, si dovrà quindi determinare, conoscendo le coordinate di un dato punto (x_1, y_1, z_1), quali sono le coordinate della sua proiezione (x_2, y_2). Si scrivano innanzitutto le equazioni di una retta passante per il punto (x, y, z) e con direzione parallela a quella di proiezione. In forma parametrica sarà semplicemente

$$\begin{aligned} x &= x_1 + x_p u \\ y &= y_1 + y_p u \\ z &= z_1 + z_p u \end{aligned} \tag{8.32}$$

I valori delle coordinate x e y del punto di intersezione della retta con il piano xy (quindi con $z = 0$) si ottengono nel modo seguente: imponendo nella terza equazione che z sia uguale a 0, il parametro u varrà

$$u = -\frac{z_1}{z_p} \tag{8.33}$$

Sostituendo questo valore nelle prime due equazioni si ottiene

$$\begin{aligned} x_2 &= x_1 - z_1 (x_p / z_p) \\ y_2 &= y_1 - z_1 (y_p / z_p) \end{aligned} \tag{8.34}$$

Queste formule di trasformazione possono essere tradotte nella corrispondente matrice

$$[x_2 \quad y_2] = [x_1 \quad y_1 \quad z_1] \begin{vmatrix} 1 & 0 \\ 0 & 1 \\ -x_p/z_p & -y_p/z_p \end{vmatrix} \tag{8.35}$$

che in coordinate omogenee risulta essere

$$[x_2 \quad y_2 \quad z_2 \quad 1] = [x_1 \quad y_1 \quad z_1 \quad 1] \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -x_p/z_p & -y_p/z_p & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{8.36}$$

L'algoritmo che segue realizza la proiezione parallela di un punto, assumendo che i rapporti di trasformazione

$$\begin{aligned} S_{xp} &= x_p / z_p \\ S_{yp} &= y_p / z_p \end{aligned} \tag{8.37}$$

siano già stati memorizzati, in modo che non sia necessario ricalcolarne il valore ogni volta che si deve proiettare un punto differente.

Algoritmo 8.12 PARALLEL-TRANSFORM (X, Y, Z)
Esegue una proiezione parallela di un punto.
Argomenti X, Y, Z←in entrata coordinate del punto che deve essere proiettato e in uscita coordinate del punto risultante dalla proiezione
Variabili globali SXP, SYP←rapporti di proiezione

```
begin
   X←X−Z*SXP;
   Y←Y−Z*SYP;
   return;
end;
```

8.6 PROIEZIONE PROSPETTICA

Un altro tipo di proiezione possibile è quella *prospettica*, per mezzo della quale più lontano è l'oggetto dall'osservatore, più questo appare piccolo nella proiezione. Questa tecnica di proiezione rende il senso della profondità dell'oggetto, in quanto indica quali parti dell'immagine rappresentano particolari dell'oggetto vicini o lontani. In una proiezione prospettica, le rette di proiezione non sono parallele, ma convergono tutte in un solo punto detto *centro di proiezione*. Questo è una conseguenza del fatto che i raggi luminosi provenienti dall'oggetto convergono verso l'occhio dell'osservatore: sono le intersezioni di queste rette convergenti col piano dello schermo a determinare l'immagine proiettata (vedi Figura 8.15).
Se il centro di proiezione è nel punto (x_c, y_c, z_c) e si vuole proiettare il punto dell'oggetto di coordinate (x_1, y_1, z_1), allora il raggio di proiezione ha equazioni

$$\begin{aligned} x &= x_c + (x_1 - x_c)u \\ y &= y_c + (y_1 - y_c)u \\ z &= z_c + (z_1 - z_c)u \end{aligned} \tag{8.38}$$

Il punto proiettato (x_2, y_2) è l'intersezione di questa retta con il piano xy. Dalla terza delle (8.38), imponendo che il punto di intersezione abbia coordinata $z = 0$, si ricava il parametro u corrispondente

$$u = -\frac{z_c}{z_1 - z_c} \tag{8.39}$$

Sostituendo questo valore nelle altre due equazioni (8.38) si ottiene

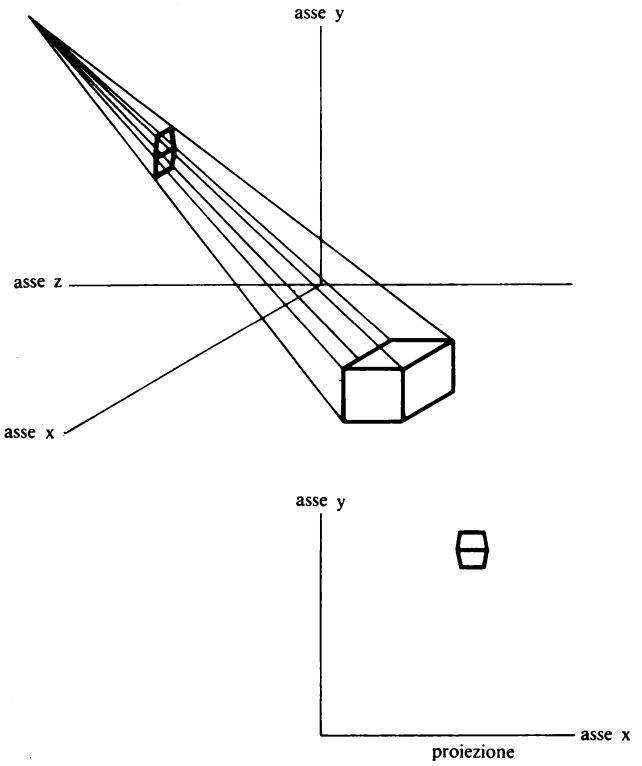


Figura 8.15 Proiezione prospettica.

$$
\begin{aligned}
x_2 &= x_c - z_c \frac{x_1 - x_c}{z_1 - z_c} \\
y_2 &= y_c - z_c \frac{y_1 - y_c}{z_1 - z_c}
\end{aligned}
\tag{8.40}
$$

Con opportuni passaggi algebrici, le equazioni precedenti possono essere riformulate come segue

$$
\begin{aligned}
x_2 &= \frac{x_c z_1 - x_1 z_c}{z_1 - z_c} \\
y_2 &= \frac{y_c z_1 - y_1 z_c}{z_1 - z_c}
\end{aligned}
\tag{8.41}
$$

Queste formule di proiezione possono essere poste nella forma di una matrice

di trasformazione, per poterle applicare ai punti in coordinate omogenee. La matrice di trasformazione che si ottiene è:

$$P = \begin{vmatrix} -z_c & 0 & 0 & 0 \\ 0 & -z_c & 0 & 0 \\ x_c & y_c & 0 & 1 \\ 0 & 0 & 0 & -z_c \end{vmatrix} \tag{8.42}$$

Al fine di mostrare come agisce la matrice di trasformazione, si consideri il punto (x_1, y_1, z_1); in coordinate omogenee si ha

$$[x_1 w_1 \quad y_1 w_1 \quad z_1 w_1 \quad w_1]$$

Moltiplicando questo punto per la matrice di trasformazione, si avrà

$$[x_2 w_2 \quad y_2 w_2 \quad z_2 w_2 \quad w_2] = [x_1 w_1 \quad y_1 w_1 \quad z_1 w_1 \quad w_1] \begin{vmatrix} -z_c & 0 & 0 & 0 \\ 0 & -z_c & 0 & 0 \\ x_c & y_c & 0 & 1 \\ 0 & 0 & 0 & -z_c \end{vmatrix}$$

$$= [-x_1 w_1 z_c + z_1 w_1 x_c \quad -y_1 w_1 z_c + z_1 w_1 y_c \quad 0 \quad z_1 w_1 - z_c w_1] \tag{8.43}$$

quindi

$$w_2 = z_1 w_1 - z_c w_1 \tag{8.44}$$

e

$$z_2 w_2 = 0 \tag{8.45}$$

dalla quale risulta

$$z_2 = 0 \tag{8.46}$$

e inoltre

$$x_2 w_2 = -x_1 w_1 z_c + z_1 w_1 x_c \tag{8.47}$$

dalla quale risulta

$$x_2 = \frac{x_c z_1 - x_1 z_c}{z_1 - z_c} \tag{8.48}$$

ed infine

$$y_2 w_2 = -y_1 w_1 z_c + z_1 w_1 y_c \tag{8.49}$$

dalla quale risulta

$$y_2 = \frac{y_c z_1 - y_1 z_c}{z_1 - z_c} \tag{8.50}$$

Il punto risultante (x_2, y_2) è la corretta proiezione del punto dato.
Il seguente algoritmo esegue la proiezione prospettica di un punto.

Algoritmo 8.13 PERSPECTIVE-TRANSFORM (X, Y, Z)
Esegue la proiezione prospettica di un punto.
Argomenti X, Y, Z←coordinate del punto sul piano di proiezione
Variabili globali XC, YC, ZC←centro di proiezione
Costanti ROUNDOFF←numero piccolo maggiore del massimo errore di arrotondamento
VERY-LARGE←numero molto grande per approssimare l'infinito

```
begin
   if |ZC-Z|<ROUNDOFF
   then begin
      X←(X-XC)*VERY-LARGE;
      Y←(Y-YC)*VERY-LARGE;
      end
   else begin
      X←(X*ZC-XC*Z)/(ZC-Z);
      Y←(Y*ZC-YC*Z)/(ZC-Z);
      end;
   return;
end;
```

8.7 PARAMETRI DI VISUALIZZAZIONE

Si è visto come formare le immagini bidimensionali che costituiscono la vista frontale di oggetti tridimensionali, tramite proiezioni parallele e prospettiche. Si supponga ora di voler proiettare la vista laterale, quella dall'alto o da dietro: tutto quello che occorre fare è applicare alcune trasformazioni di rotazione prima di eseguire la proiezione. Vi sono due modi di affrontare questo problema: il primo consiste nell'immaginare il *piano di visualizzazione* (cioè il *piano di proiezione*, ovvero quello dello schermo) fisso e l'oggetto ruotato; il secondo considera invece l'oggetto fisso e il piano di visualizzazione ruotato (vedi Figura 8.16). Il sistema CORE usa il secondo tipo di approccio, che considera il piano di proiezione e ciascun segmento del display file come se fossero rispettivamente la pellicola di una macchina fotografica e un fotogramma impressionato (vedi Figura 8.17). In questo modo è possibile muovere la macchina fotogra-

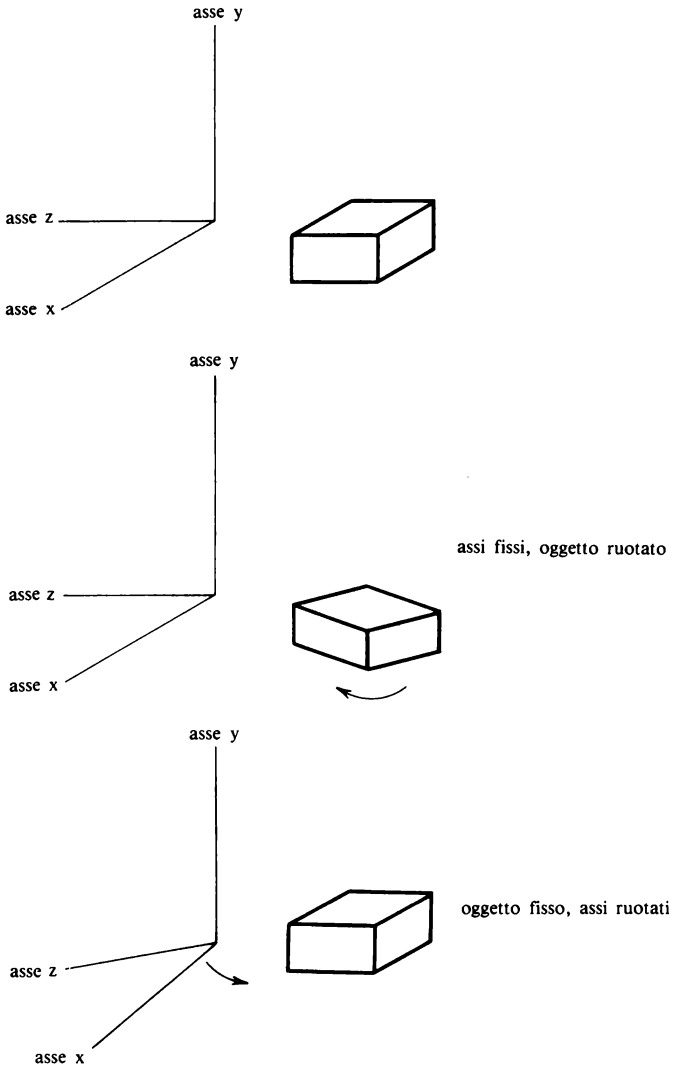


Figura 8.16 Ruotare un oggetto equivale a ruotare gli assi in senso opposto

fica in ogni direzione e vedere l'oggetto da qualsiasi angolazione. La fotografia ottenuta è ciò che viene mostrato sullo schermo (come se la pellicola venisse sviluppata e proiettata sullo schermo). Variando i parametri di visualizzazione, è possibile cambiare la posizione di questa macchina fotografica ideale (vedi Fi-

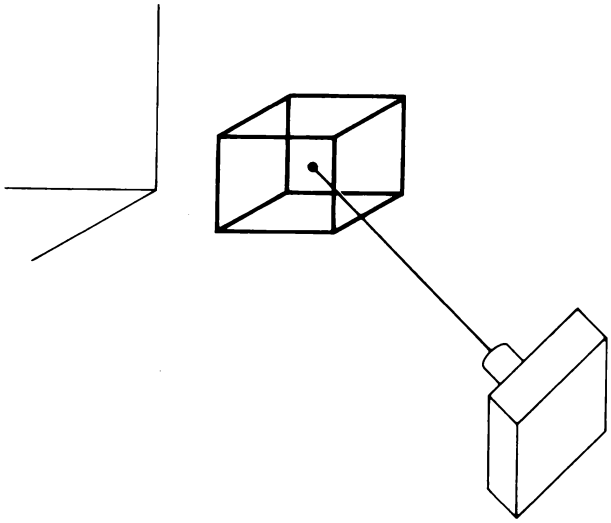


Figura 8.17 L'analogia con la macchina fotografica

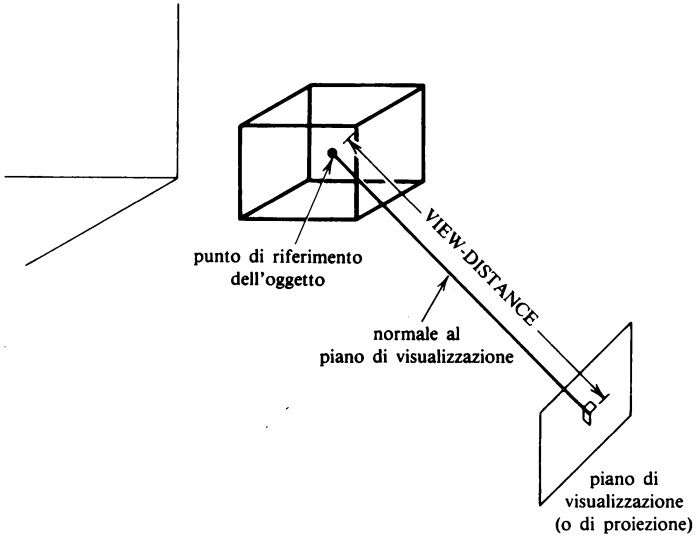


Figura 8.18 Parametri di visualizzazione tridimensionale

gura 8.18); i primi parametri da considerare sono le coordinate del *punto di riferimento* dell'oggetto (XR, YR, ZR): tutti gli altri parametri di visualizzazione sono riferiti ad esso. Ruotare il punto di vista significa ruotare attorno a questo punto di riferimento (e non attorno all'origine del sistema di riferimento), spostando il punto di vista sulla superficie di una sfera con il centro in questo punto. Cambiando altri parametri di visualizzazione, si può muovere il punto di vista lungo un arco o allungare il raggio della superficie sferica su cui ci si muove. La direzione della retta congiungente il punto di vista con il punto di riferimento dell'oggetto è data dal vettore normale al piano di proiezione [DXN, DYN, DZN]. Questo vettore rappresenta la direzione in cui si sta guardando l'oggetto; la sua lunghezza è data dal parametro VIEW-DISTANCE. In questo modo sono stati definiti due sistemi di riferimento: uno relativo all'oggetto rispetto al quale sono state riferite le coordinate dei suoi punti, l'altro relativo al piano di visualizzazione rispetto al quale vengono riferiti i punti proiettati. Ritornando all'esempio della macchina fotografica, si può pensare che il sistema di riferimento dell'oggetto sia disegnato sul «pavimento» sul quale è appoggiato l'oggetto e il sistema di riferimento del punto di vista sia idealmente tracciato sulla pellicola della macchina fotografica. L'origine del sistema di riferimento verrà posta nel punto dove una retta, parallela al vettore normale al piano di proiezione e passante per il punto di riferimento posto sull'oggetto, interseca il piano di proiezione.

Resta da definire, a questo punto, un ultimo parametro di visualizzazione. Si supponga di ruotare il piano di proiezione attorno all'asse costituito dalla retta che congiunge il punto di riferimento dell'oggetto e l'origine del sistema di rife-

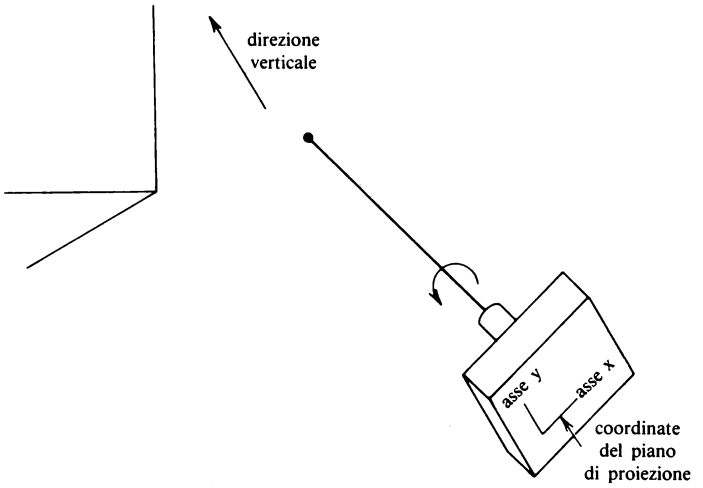


Figura 8.19 Direzione verticale e coordinate del piano di proiezione

rimento locale al piano di proiezione (cioè attorno al vettore normale). Per ogni diversa rotazione, la proiezione sarà la stessa, ma ruotata attorno all'asse ortogonale allo schermo (vedi Figura 8.19). Il parametro che definisce questa direzione sarà un vettore che indica quella che deve essere considerata la direzione verticale «verso l'alto» [XUP, YUP, ZUP]. Utilizzando questo vettore si determina l'angolo di rotazione.
Riassumendo, i diversi parametri di vista hanno le seguenti funzioni: se si modifica il punto di riferimento posto sull'oggetto, risulterà modificata la parte dell'oggetto che si proietta nell'origine del sistema di riferimento del piano di proiezione (vedi Figura 8.20).
Se invece viene modificata la normale al piano di proiezione, l'oggetto sarà ripreso nella stessa posizione, ma da angolazione differente (vedi Figura 8.21).
È possibile modificare, inoltre, il valore della distanza tra l'oggetto ed il piano di proiezione (vedi Figura 8.22).
Infine, il cambiamento della direzione verticale ci permette di posizionare in modo opportuno l'immagine dell'oggetto sul piano di proiezione (vedi Figura 8.23).

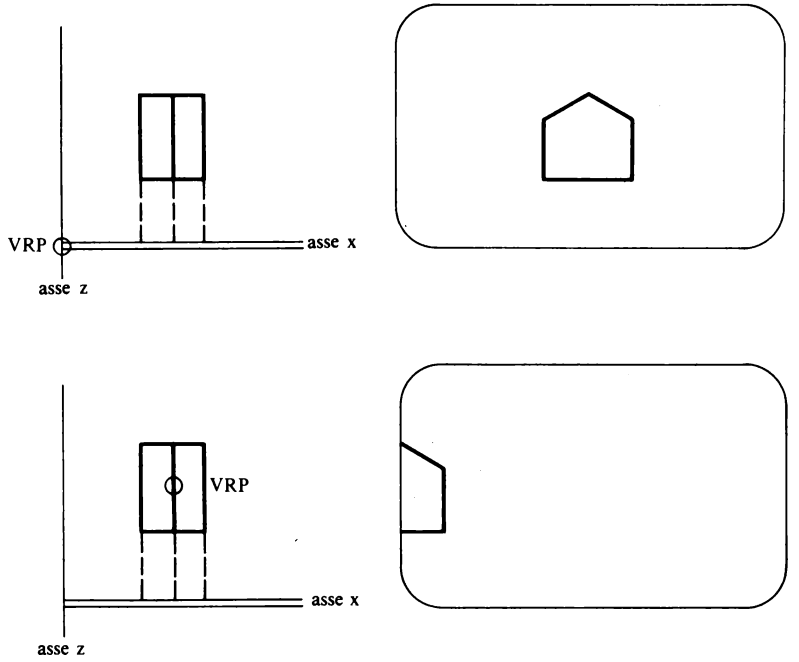


Figura 8.20 Variazione del punto di riferimento dell'oggetto

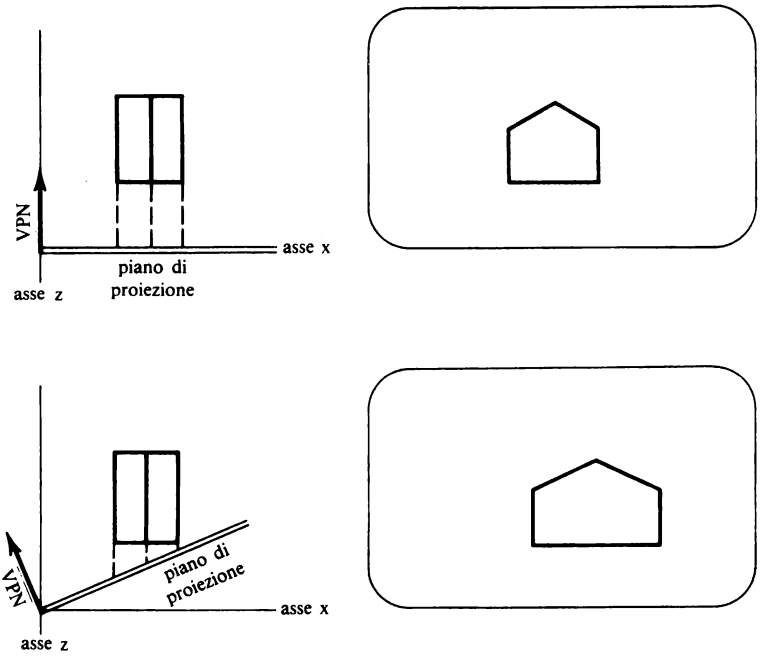


Figura 8.21 Variazione della normale al piano di proiezione

Questi parametri consentono all'utente di scegliere arbitrariamente il punto di vista da cui osservare l'oggetto. Le routine del sistema grafico dovranno quindi permettere all'utente di assegnare i valori di questi parametri, che saranno memorizzati in opportune variabili globali.
Gli algoritmi seguenti realizzano l'interfaccia tra l'utente e la trasformazione tridimensionale che esegue la proiezione.

Algoritmo 8.14 SET-VIEW-REFERENCE-POINT (X, Y, Z)
Aggiorna il punto di riferimento dell'oggetto.
Argomenti X, Y, Z—coordinate del punto
Variabili globali XR, YR, ZR—area di memoria permanente che contiene le coordinate del punto di riferimento

```
begin
   XR←X;
   YR←Y;
   ZR←Z;
   return;
end;
```

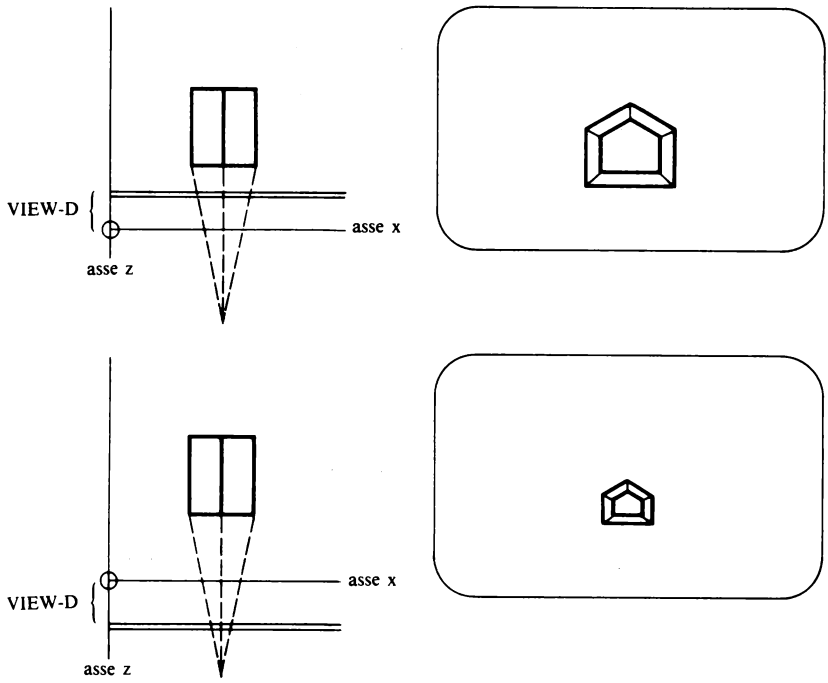


Figura 8.22 Variazione della distanza fra il piano di proiezione e il punto di riferimento dell'oggetto

Algoritmo 8.15 SET-VIEW-PLANE-NORMAL (DX, DY, DZ)
Aggiorna la normale al piano di proiezione.

Argomenti	DX, DY, DZ←vettore normale al piano di proiezione
Variabili globali	DXN, DYN, DZN←area di memoria permanente che contiene il vettore normale al piano di proiezione
Variabili locali	D←lunghezza del vettore
Costanti	ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
   D←SQRT(DX↑2+DY↑2+DZ↑2);
   IF D<ROUNDOFF
   then return-error 'NORMALE AL PIANO NON VALIDA';
   DXN←DX/D;
   DYN←DY/D;
   DZN←DZ/D;
   return;
end;
```

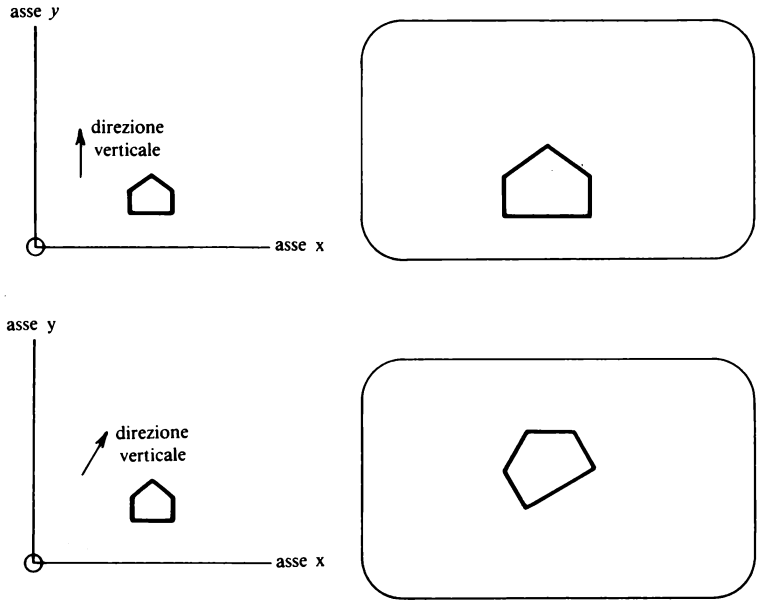


Figura 8.23 Variazione della direzione verticale

Algoritmo 8.16 SET-VIEW-DISTANCE (D)
Aggiorna la distanza tra il punto di riferimento e il piano di proiezione.
Argomenti D←nuova distanza
Variabili globali VIEW-DISTANCE←area di memoria permanente che contiene il valore della distanza

```
begin
   VIEW-DISTANCE←D;
   return;
end;
```

Algoritmo 8.17 SET-VIEW-UP (DX, DY, DZ)
Aggiorna la direzione verticale della figura.
Argomenti DX, DY, DZ←vettore della direzione verticale
Variabili globali DXUP, DYUP, DZUP←area di memoria permanente che contiene il vettore della direzione verticale
Costanti ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
   if |DX| + |DY| + |DZ| < ROUNDOFF
   then return-error 'NESSUNA DIREZIONE SET-VIEW-UP';
```

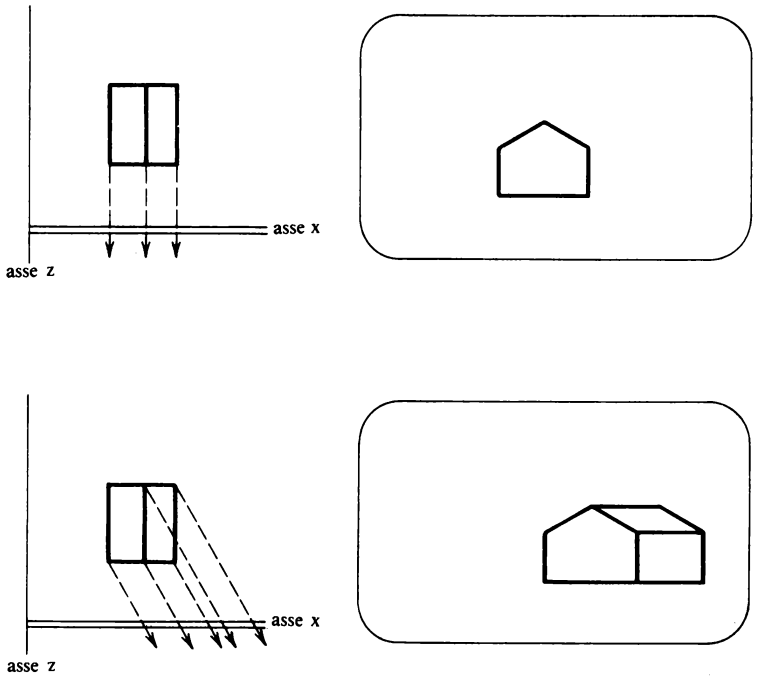


Figura 8.24 Variazione della direzione di proiezione parallela

```
    DXUP←DX;
    DYUP←DY;
    DZUP←DZ;
    return;
end;
```

L'utente deve essere in grado di assegnare anche il tipo di proiezione (vedi Figure 8.24 e 8.25). Nel caso di proiezioni parallele, l'utente deve specificare la direzione delle linee di proiezione, mentre per una proiezione prospettica occorrono le coordinate del centro di proiezione. Tutti questi valori saranno memorizzati in variabili globali; sarà necessario, inoltre, un flag per specificare quale dei due tipi di proiezione applicare.

Algoritmo 8.18 SET-PARALLEL (DX, DY, DZ)
Routine utente che specifica la direzione della proiezione parallela.
Argomenti DX, DY, DZ←vettore della direzione di proiezione
Variabili globali PERSPECTIVE-FLAG←indicatore del tipo di proiezione

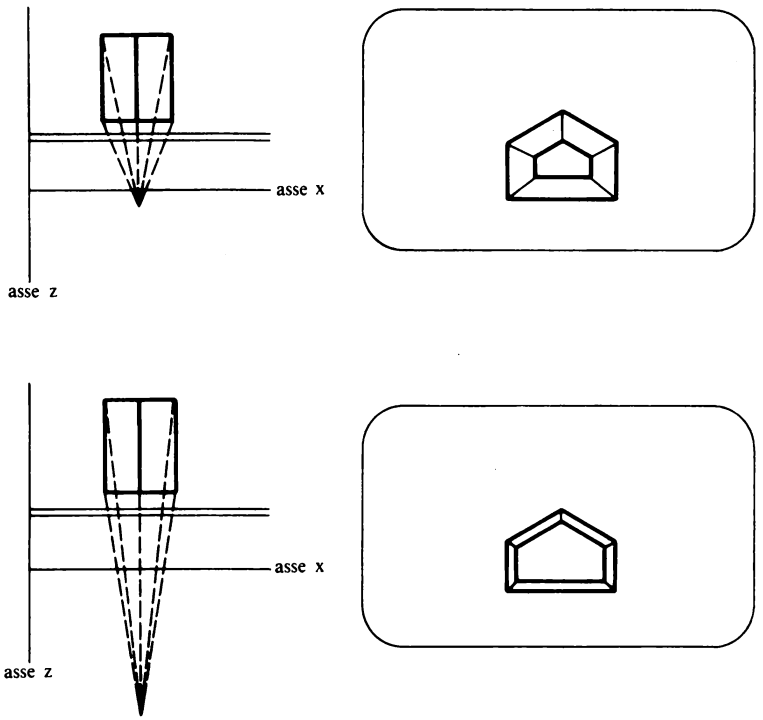


Figura 8.25 Variazione del centro di proiezione prospettica

DXP, DYP, DZP←area di memoria permanente che contiene il vettore della direzione di proiezione

Costanti ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
   IF |DX| + |DY| + |DZ| < ROUNDOFF
   then return-error 'NESSUNA DIREZIONE DI PROIEZIONE';
   PERSPECTIVE-FLAG←FALSE;
   DXP←DX;
   DYP←DY;
   DZP←DZ;
   return;
end;
```

Algoritmo 8.19 SET-PERSPECTIVE (X, Y, Z)
Assegna il tipo di proiezione parallela e memorizza il centro di proiezione.

Argomenti X, Y, Z←coordinate del centro di proiezione
Variabili globali XPCNTR, YPCNTR, ZPCNTR←area di memoria permanente per il centro di proiezione
PERSPECTIVE-FLAG←indicatore del tipo di proiezione

```
begin
   PERSPECTIVE-FLAG←TRUE;
   XPCNTR←X;
   YPCNTR←Y;
   ZPCNTR←Z;
   return;
end;
```

8.8 CONVERSIONE NELLE COORDINATE DEL PIANO DI PROIEZIONE

L'utente fornisce la descrizione dell'oggetto in termini di coordinate dell'oggetto; ma ciò che è visibile di esso da un particolare punto di vista si ottiene esprimendo l'oggetto nel sistema di riferimento del piano di proiezione. Il processo di generazione di una particolare vista dell'oggetto è una trasformazione da un sistema di coordinate ad un altro. Sebbene questo problema possa apparire complesso, in realtà è stato risolto in modo soddisfacente. I passi da effettuare sono gli stessi che si devono eseguire per la rotazione attorno a un asse arbitrario.
Il primo passo è una traslazione per muovere l'origine del sistema di riferimento dell'oggetto nella posizione corretta per il sistema di coordinate del piano di proiezione; dapprima la si sposta nel punto di riferimento dell'oggetto e quindi lungo la normale al piano di proiezione di una quantità pari alla distanza tra il punto di riferimento e il piano stesso. Dopo che l'origine è stata posizionata correttamente, si procede ad allineare l'asse z. Nella descrizione della rotazione attorno a un asse arbitrario, si è visto come ruotare una retta in modo che si sovrapponga all'asse z; la retta in questione sarà, nel nostro caso, l'asse z del sistema di coordinate oggetto che deve essere ruotato in modo da sovrapporsi all'asse z del sistema di coordinate del piano di proiezione (normale al piano). Questo viene realizzato in due passi: dapprima una rotazione attorno all'asse x porta la retta a giacere sul piano xz del sistema di riferimento del piano di proiezione; quindi una rotazione attorno all'asse y muove l'asse z nell'appropriata posizione. Infine, occorre ruotare attorno all'asse z fino a che gli assi x e y non siano al loro posto nel sistema di coordinate del piano di proiezione. L'intera sequenza di trasformazione è data da:

$$\text{TMATRIX} = T\,R_X R_Y R_Z \tag{8.51}$$

dove

$$T = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -(XR + DXN * VIEW\text{-}DISTANCE) & -(ZR + DZN * VIEW\text{-}DISTANCE) & \\ -(YR + DYN * VIEW\text{-}DISTANCE) & & \end{vmatrix} \tag{8.52}$$

$$R_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & -DZN/V & -DYN/V & 0 \\ 0 & DYN/V & -DZN/V & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{8.53}$$

e

$$V = (DYN^2 + DZN^2)^{1/2} \tag{8.54}$$

e anche

$$R_y = \begin{vmatrix} V & 0 & -DXN & 0 \\ 0 & 1 & 0 & 0 \\ DXN & 0 & V & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{8.55}$$

e

$$R_z = \begin{vmatrix} YUP\text{-}VP/RUP & XUP\text{-}VP/RUP & 0 & 0 \\ -XUP\text{-}VP/RUP & YUP\text{-}VP/RUP & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{8.56}$$

dove

$$[XUP\text{-}VP \quad YUP\text{-}VP \quad Z \quad 1] = [DXUP \quad DYUP \quad DZUP \quad 1]R_xR_y \tag{8.57}$$

e

$$RUP = (XUP\text{-}VP^2 + YUP\text{-}VP^2)^{1/2} \tag{8.58}$$

Si analizzi ora il significato delle variabili XUP-VP, YUP-VP; si sta cercando di portare a termine il procedimento di trasformazione ruotando gli assi x e y fino al corretto posizionamento, che si raggiunge quando l'asse y è allineato con la proiezione sul piano di visualizzazione della direzione verticale (indicata dal vettore VIEW-UP), mediante le due trasformazioni parziali (R_x R_y). Non è necessaria alcuna traslazione, in quanto si sta operando su un vettore che non

ha un punto d'applicazione fisso, ma solo modulo, direzione e verso. Ignorando la coordinata z, poiché si proietta sul piano xy, calcolando la distanza dall'origine al punto proiettato ed infine dividendo i valori di x e y per questa distanza, otteniamo il seno e il coseno dell'angolo di rotazione. Il seguente algoritmo crea la matrice di trasformazione.

Algoritmo 8.20 MAKE-VIEW-PLANE-TRANSFORMATION
Crea la trasformazione di proiezione.

Variabili globali XR, YR, ZR←punto di riferimento sull'oggetto
DXN, DYN, DZN←normale al piano di proiezione
DXUP, DYUP, DZUP←direzione verticale della vista
TMATRIX←matrice 4×3 per la trasformazione
PERSPECTIVE-FLAG←indicatore del tipo di proiezione
VIEW-DISTANCE←distanza tra il punto di riferimento e il piano di proiezione

Variabili locali V, XUP-VP, YUP-VP, RUP←variabili per memorizzare i risultati parziali

Costanti ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
  inizializzazione della matrice identità
  NEW-TRANSFORM-3;
  traslazione affinché il centro del piano di proiezione diventi la nuova
    origine
  TRANSLATE-3(-(XR + DXN * VIEW-DISTANCE),
    -(YR + DYN * VIEW-DISTANCE),
    -(ZR + DZN * VIEW-DISTANCE));
  rotazione che rende la normale al piano di proiezione coincidente con
    l'asse z
  V←SQRT(DYN↑2 + DZN↑2);
  if V > ROUNDOFF then ROTATE-X-3(-DYN/V, -DZN/V);
  ROTATE-Y-3(DXN, V);
  determina la direzione verticale in questo nuovo sistema di riferimento
  XUP-VP←DXUP * TMATRIX[1, 1] + DYUP * TMATRIX[2, 1] +
    DZUP * TMATRIX[3, 1];
  YUP-VP←DXUP * TMATRIX[1, 2] + DYUP * TMATRIX[2, 2] +
    DZUP * TMATRIX[3, 2];
  determina la rotazione necessaria per posizionare correttamente la verti-
    cale del piano di proiezione
  RUP←SQRT(XUP-VP↑2 + YUP-VP↑2);
  if RUP < ROUNDOFF
  then return-error 'IL VETTORE SET-VIEW-UP GIACE LUNGO LA
    NORMALE AL PIANO';
  ROTATE-Z-3(XUP-VP/RUP, YUP-VP/RUP);
  if PERSPECTIVE-FLAG
```

```
    then MAKE-PERSPECTIVE-TRANSFORMATION
      else MAKE-PARALLEL-TRANSFORMATION;
    return;
end;
```

L'algoritmo precedente esegue anche la conversione dei parametri di proiezione, nei casi di proiezione parallela e prospettica, da coordinate oggetto a coordinate del piano di proiezione, chiamando a seconda dei casi la routine MAKE-PARALLEL-TRANSFORMATION o la MAKE-PERSPECTIVE-TRANSFORMATION. Quest'ultima routine applica la matrice di trasformazione al solo centro di proiezione ed inoltre segnala un errore nel caso che il centro di proiezione si trovi dalla parte sbagliata del piano di proiezione, cioè nello stesso semispazio in cui si trova anche l'oggetto. La routine MAKE-PARALLEL-TRANSFORMATION converte la direzione della proiezione in coordinate del piano di visualizzazione, omettendo di applicare la traslazione non necessaria alla trasformazione del vettore ed inoltre segnala un errore se la direzione di proiezione risulta essere parallela al piano di proiezione.

Algoritmo 8.21 MAKE-PERSPECTIVE-TRANSFORMATION
Converte il centro di proiezione in coordinate del piano di proiezione.
Variabili globali XPCNTR, YPCNTR, ZPCNTR←centro di proiezione
XC, YC, ZC←centro di proiezione in coordinate del piano di proiezione

```
begin
  XC←XPCNTR;
  YC←YPCNTR;
  ZC←ZPCNTR;
  VIEW-PLANE-TRANSFORM(XC, YC, ZC);
  if ZC<0 then return-error 'CENTRO DI PROIEZIONE DIETRO AL
    PIANO';
  return;
end;
```

Algoritmo 8.22 MAKE-PARALLEL-TRANSFORMATION
Calcola la direzione di proiezione in coordinate del piano di proiezione.
Variabili globali VXP, VYP, VZP←direzione della proiezione in coordinate del piano di proiezione
DXP, DYP, DZP←vettore della direzione di proiezione parallela
SXP, SYP←inclinazioni di proiezione rispetto alla direzione Z
TMATRIX←matrice 4×3 per la trasformazione
Costanti ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
   VXP←DXP*TMATRIX[1, 1]+DYP*TMATRIX[2, 1]+
            DZP*TMATRIX[3, 1];
   VYP←DXP*TMATRIX[1, 2]+DYP*TMATRIX[2, 2]+
            DZP*TMATRIX[3, 2];
   VZP←DXP*TMATRIX[1, 3]+DYP*TMATRIX[2, 3]+
            DZP*TMATRIX[3, 3];
   if|VZP|<ROUNDOFF
   then return-error 'PROIEZIONE PARALLELA AL PIANO';
   SXP←VXP/VZP;
   SYP←VYP/VZP;
   return;
end;
```

8.9 TRASFORMAZIONE DI VISUALIZZAZIONE 3-D

Finalmente si hanno a disposizione tutti i mezzi necessari per disegnare la proiezione, su un piano, di oggetti definiti nelle tre dimensioni. Supponendo di aver fissato i parametri di visualizzazione e di aver definito il sistema di riferimento del piano di proiezione con la routine MAKE-VIEW-PLANE-TRANSFORMATION, l'utente può usare uno dei comandi di disegno tridimensionale come, per esempio, LINE-ABS-3(X, Y, Z), che aggiorna la posizione della penna e chiama la routine DISPLAY-FILE-ENTER per proiettare l'immagine sul piano di proiezione prima di inserirla nel display file. Questa procedura applica la matrice di trasformazione, che converte i punti nelle coordinate del piano di proiezione, quindi proietta l'immagine sul piano (annullando la coordinata z) ed infine la memorizza nel display file dopo aver effettuato un'operazione di clipping. Se quest'ultima operazione viene effettuata dopo la conversione delle coordinate e la proiezione sul piano, il risultato è tale che la window può essere considerata come applicata sul piano di proiezione (vedi Figura 8.26). Di seguito viene descritto l'algoritmo che serve per convertire un punto dal sistema di coordinate oggetto al sistema di coordinate del piano di proiezione.

Algoritmo 8.23 VIEW-PLANE-TRANSFORM (X, Y, Z)
Trasforma un punto nel sistema di riferimento del piano di proiezione.

Argomenti	X, Y, Z←coordinate del punto da trasformare, usate anche come parametri in uscita
Variabili globali	TMATRIX←matrice di trasformazione 4×3
Variabili locali	T←array di tre elementi in cui vengono memorizzati i risultati parziali dei calcoli
	I←indice delle colonne della matrice TMATRIX

```
begin
   for I = 1 to 3
```

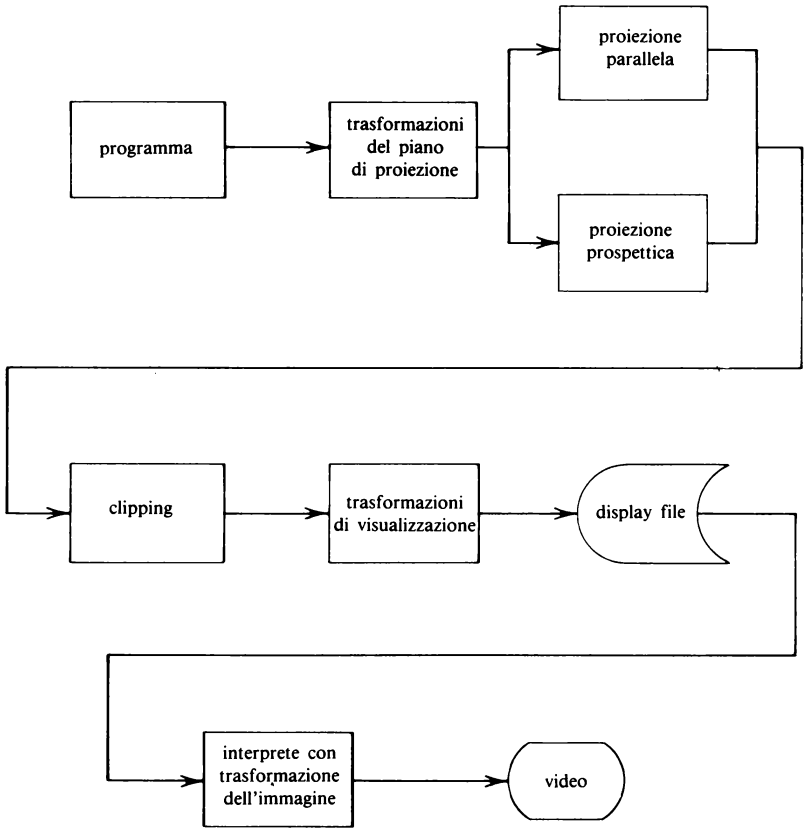


Figura 8.26 Estensione del sistema grafico con le operazioni di visualizzazione tridimensionale

```
    do T[I]←X*TMATRIX[1, I]+Y*TMATRIX[2, I]+
       Z*TMATRIX[3, I]+TMATRIX[4, I];
    X←T[1];
    Y←T[2];
    Z←T[3];
    return;
end;
```

La routine DISPLAY-FILE-ENTER viene modificata nel modo seguente:

Algoritmo 8.24 DISPLAY-FILE-ENTER (OP) (Algoritmo 6.14 aggiornato)
Inserisce un'istruzione nel display file.

Argomenti OP←codice operativo dell'istruzione da inserire
Variabili globali DF-PEN-X, DF-PEN-Y, DF-PEN-Z←posizione corrente della penna
PERSPECTIVE-FLAG←indicatore del tipo di proiezione
Variabili locali X, Y, Z←coordinate del punto che deve essere trsformato

```
begin
   if OP<0 then PUT-POINT(OP, 0, 0)
   else begin
      X←DF-PEN-X;
      Y←DF-PEN-Y;
      Z←DF-PEN-Z;
      VIEW-PLANE-TRANSFORM(X, Y, Z);
      if PERSPECTIVE-FLAG
      then PERSPECTIVE-TRANSFORM(X, Y, Z)
      else PARALLEL-TRANSFORM(X, Y, Z);
      CLIP(OP, X, Y);
      end;
   return;
end;
```

Per finire, occorre estendere la routine CREATE-SEGMENT inserendo una chiamata alla routine NEW-VIEW-3 che definisce una nuova trasformazione di visualizzazione, fornendo all'utente la possibilità di applicare delle trasformazioni a ciascun segmento del display file; l'utente può assegnare gli opportuni valori ai corrispondenti parametri di trasformazione, che non avranno effetto finché non verrà creato il relativo segmento.

Algoritmo 8.25 NEW-VIEW-3
Crea una nuova trasformazione di visualizzazione globale.

```
begin
   NEW-VIEW-2;
   MAKE-VIEW-PLANE-TRANSFORMATION;
   return;
end;
```

Algoritmo 8.26 CREATE-SEGMENT (SEGMENT-NAME)
(Algoritmo 7.15 aggiornato)
Routine utente che crea un segmento identificato da un nome.

Argomenti SEGMENT-NAME←nome del segmento
Variabili globali NOW-OPEN←segmento aperto corrente
FREE←indice della successiva cella libera nel display file
SEGMENT-START, SEGMENT-SIZE, VISIBILITY, ANGLE, SCALE-X, SCALE-Y, TRANSLATE-X,

```
                    TRANSLATE-Y, DETECTABLE←array che com-
                    pongono la tabella dei segmenti
Costanti            NUMBER-OF-SEGMENTS←dimensione della tabella dei
                    segmenti
begin
   if NOW-OPEN>0
        then return-error 'SEGMENTO ANCORA APERTO';
   if SEGMENT-NAME<1
   or SEGMENT-NAME>NUMBER-OF-SEGMENTS
        then return-error 'NOME DEL SEGMENTO NON VALIDO';
   if SEGMENT-SIZE[SEGMENT-NAME]>0
        then return-error 'SEGMENTO GIÀ ESISTENTE';
   NEW-VIEW-3;
   SEGMENT-START[SEGMENT-NAME]←FREE;
   SEGMENT-SIZE[SEGMENT-NAME]←0;
   VISIBILITY[SEGMENT-NAME]←VISIBILITY[0];
   ANGLE[SEGMENT-NAME]←ANGLE[0];
   SCALE-X[SEGMENT-NAME]←SCALE-X[0];
   SCALE-Y[SEGMENT-NAME]←SCALE-Y[0];
   TRANSLATE-X[SEGMENT-NAME]←TRANSLATE-X[0];
   TRANSLATE-Y[SEGMENT-NAME]←TRANSLATE-Y[0];
   DETECTABLE[SEGMENT-NAME]←DETECTABLE[0];
   NOW-OPEN←SEGMENT-NAME;
   return;
end;
```

Infine, l'ultima cosa di cui si ha bisogno è una routine per assegnare i valori iniziali dei parametri di visualizzazione, che sono: l'origine del sistema di riferimento dell'oggetto come punto di riferimento della vista, il piano xy del sistema di coordinate oggetto come piano di proiezione, la direzione y (nel sistema di coordinate oggetto) come direzione verticale e la proiezione parallela nella direzione z.

Algoritmo 8.27 INITIALIZE-8

```
begin
   INITIALIZE-7;
   SET-VIEW-REFERENCE-POINT(0, 0, 0);
   SET-VIEW-PLANE-NORMAL(0, 0, −1);
   SET-VIEW-DISTANCE(0);
   SET-VIEW-UP(0, 0, 1);
   SET-PARALLEL(0, 0, 1);
   NEW-VIEW-3;
   return;
end;
```

8.10 PROIEZIONI DI TIPO SPECIALE

Il problema di realizzare viste in due dimensioni di oggetti tridimensionali è nato molto prima dell'introduzione dei calcolatori nella grafica. Un tipo di proiezione molto usata è l'*assonometria*, che è una proiezione parallela avente come direzione di proiezione la normale al piano di visualizzazione. Consideriamo un cubo con gli spigoli paralleli agli assi del sistema di coordinate rispetto al quale è definito l'oggetto; guardando di fronte una faccia, il cubo apparirà come un quadrato (vedi Figura 8.27). Se ci spostiamo un poco di lato, si comincerà ad intravedere anche una delle facce laterali, mentre gli spigoli della faccia frontale appariranno più corti. Se si aumenta l'angolatura della proiezione, in modo da osservare il cubo dall'alto verso il basso, gli spigoli della faccia superiore si allungheranno in modo da essere visibili, mentre alcuni spigoli della faccia laterale in vista appariranno più corti (vedi Figura 8.28). Vi è una particolare proiezione assonometrica chiamata *proiezione isometrica*, che riduce tutti gli spigoli

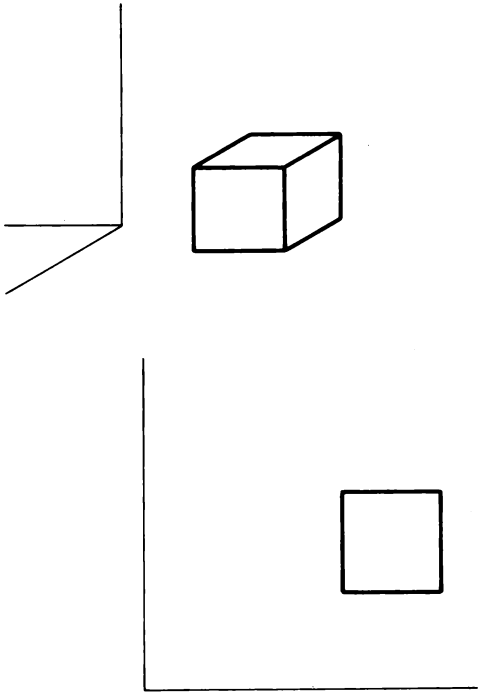


Figura 8.27 Vista frontale di una faccia del cubo

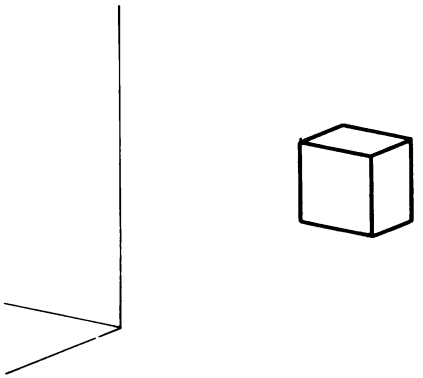


Figura 8.28 Cambiare il punto di vista significa allungare alcuni spigoli e accorciarne altri

di uno stesso fattore (vedi Figura 8.29). Una proiezione isometrica di un cubo ha come risultato uno spigolo centrale circondato da tre facce tutte della stessa dimensione. I comandi necessari per realizzare una proiezione isometrica di un oggetto con lati paralleli agli assi sono:

```
SET-PARALLEL(1, 1, 1);
SET-VIEW-PLANE-NORMAL(-1, -1, -1);
```

Questa proiezione ha come punto di riferimento lo spigolo in alto a destra dell'oggetto, mentre gli appropriati comandi relativi a proiezioni isometriche aventi come centro di proiezione gli altri sette spigoli sono proposti come esercizi.
Se si sceglie la trasformazione di visualizzazione in modo che solo gli spigoli

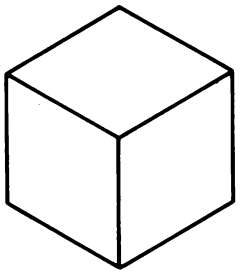


Figura 8.29 La proiezione isometrica accorcia le distanze lungo tutti gli assi della stessa misura

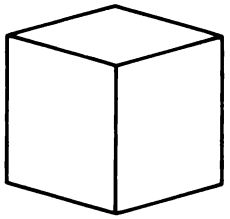


Figura 8.30 La proiezione dimetrica accorcia le distanze lungo due soli assi della stessa misura

paralleli a due degli assi siano ridotti in modo uguale dalla proiezione, allora tale tipo di proiezione viene detta *dimetrica* (vedi Figura 8.30).
Una proiezione *trimetrica* sarà quindi — estendendo la definizione precedente — una proiezione nella quale gli spigoli paralleli a ciascuno dei tre assi vengono ridotti in misura differente (vedi Figura 8.31).
Nel caso che la direzione della proiezione parallela non sia parallela alla normale del piano di proiezione, si avrà quella che viene definita *proiezione obliqua*. Di seguito vengono brevemente trattati due soli tipi di proiezione obliqua, chiamati *proiezione cavaliera* e *proiezione cabinet*. Il primo tipo viene descritto facendo riferimento ancora al cubo con gli spigoli paralleli agli assi e utilizzando un piano parallelo al piano *xy* come superficie di proiezione. Per ottenere una proiezione cavaliera, bisogna che la direzione di proiezione sia inclinata in modo che i punti con coordinata *z* positiva siano proiettati in basso e alla sinistra sul piano di proiezione mentre, viceversa, i punti con coordinata *z* negativa siano proiettati in alto e a destra. L'angolo dell'asse *z* proiettato (che corrisponde a quanto si deve salire muovendosi di una unità verso destra) può essere qualunque, ma la distanza di cui un punto viene traslato lungo la direzione *z* proiettata deve essere uguale al valore tridimensionale vero della componente *z* della distanza dal piano di proiezione. Il comando che crea una proiezione cavaliera inclinata di un angolo *A* è

```
SET-PARALLEL(COS(A), SIN(A), 1)
```

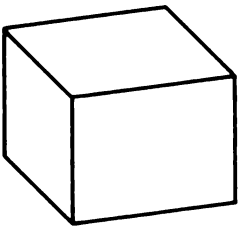


Figura 8.31 La proiezione trimetrica accorcia le distanze lungo i tre assi in misura ineguale

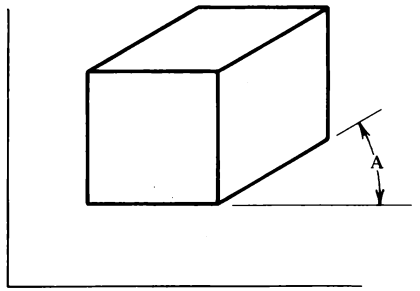


Figura 8.32 Proiezione cavaliera

Ogni comando SET-PARALLEL i cui argomenti siano nello stesso rapporto di quelli dell'esempio precedente andrà ugualmente bene (vedi Figura 8.32). La realizzazione di una proiezione cavaliera, con le distanze sull'asse *z* proiettato uguali a quelle vere, è molto semplice da realizzare, anche manualmente, ma il risultato non è soddisfacente perché l'oggetto ci appare, nella direzione dell'asse *z*, più lungo di quanto non sia in realtà.
Un metodo alternativo a questo e altrettanto semplice da applicare consiste nel ridurre le distanze sull'asse *z* proiettato della metà di quelle vere. Questo tipo di proiezione è quella che viene detta *cabinet*. Utilizzando il comando SET-PARALLEL, possiamo creare una proiezione di questo tipo fornendo i seguenti parametri come argomenti (vedi Figura 8.33).

```
SET-PARALLEL(COS(A), SIN(A), 2).
```

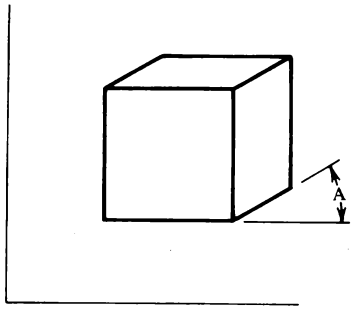


Figura 8.33 Proiezione cabinet

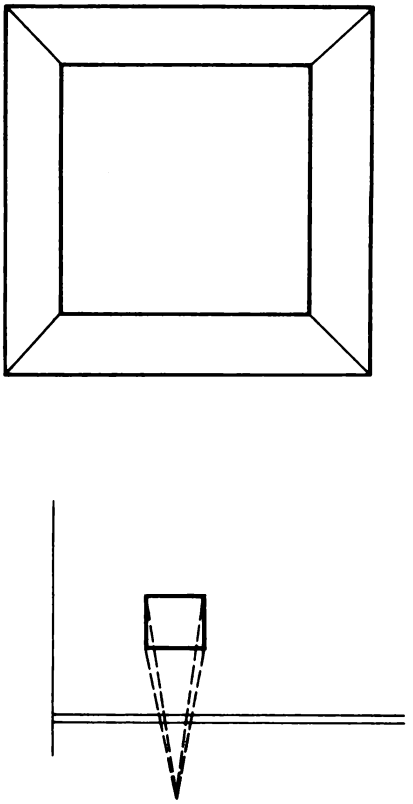


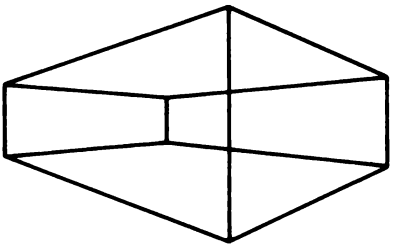
Figura 8.34 Prospettiva a un punto

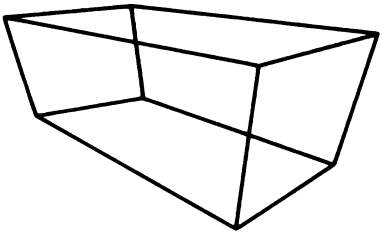
Le proiezioni di tipo prospettico possono essere classificate come relative ad un punto, due punti o tre punti. Queste tre categorie sono diverse per il numero di *punti di fuga* richiesti al disegnatore che costruisce il disegno. Le routine che sono state descritte sono sufficientemente generali per permetterci di costruire tutti questi diversi tipi di proiezione.

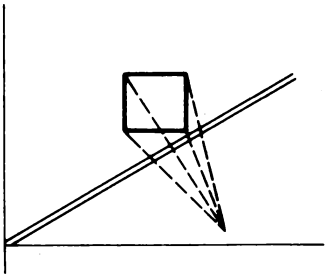
Una prospettiva ad un solo punto di fuga viene prodotta quando una delle facce di un oggetto cubico è parallela al piano di proiezione (vedi Figura 8.34).

La prospettiva con due punti di fuga si ha quando un insieme degli spigoli del cubo è composto da elementi tutti paralleli al piano di proiezione senza che nessuna delle facce lo sia (vedi Figura 8.35).

Infine la prospettiva a tre punti corrisponde al caso in cui nessuno degli spigoli — e quindi ovviamente nessuna delle facce — risulta essere parallela al piano di proiezione (vedi Figura 8.36).







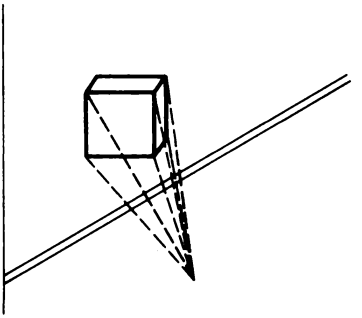


Figura 8.35 Prospettiva a due punti

Figura 8.36 Prospettiva a tre punti

APPLICAZIONE PRATICA

Una delle applicazioni che fa uso massiccio della grafica a tre dimensioni è il simulatore di volo. Si tratta di un sistema che può essere usato nella fase iniziale dell'addestramento di un pilota. Il simulatore appare esattamente come una cabina di pilotaggio, nella quale però il parabrezza è sostituito dall'immagine prodotta dal calcolatore che simula l'ambiente circostante al velivolo. L'immagine si modifica in conseguenza delle azioni del pilota nello stesso modo in cui si modifica ciò che il pilota vedrebbe volando realmente. Così facendo, gli eventuali errori del pilota non avranno esito catastrofico come potrebbe avvenire durante un vero volo di addestramento.

La prima cosa che si deve fare per costruire un simulatore di volo, utilizzando il sistema grafico descritto, è la costruzione di un modello del mondo entro il quale il pilota può volare. Utilizzando le primitive di LINE e POLYGON, si potrà costruire il disegno di edifici, strade, laghi ed ogni altro tipo di paesaggio scelto. Questa costruzione può essere molto complessa ma risulta indispensabi-

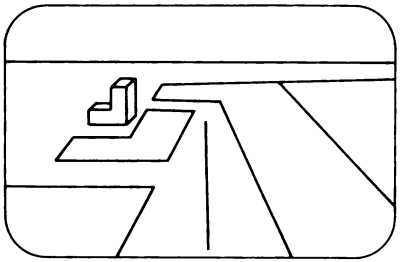


Figura 8.37 Simulatore di volo

le. Se si suppone che questa operazione sia fatta dalla procedura BUILD-WORLD, non resta che proiettare l'immagine sullo schermo in modo che corrisponda alla parte del paesaggio visibile dal pilota. Supponendo inoltre che il piano di proiezione coincida con il parabrezza della cabina di pilotaggio, spostando il punto di riferimento (considerato solidale con il velivolo) si sposta anche il piano di proiezione e di conseguenza si potrà porre il valore del parametro VIEW-DISTANCE uguale a 0.
Se si impone che il punto di riferimento sia centrato nella finestra sullo schermo, il comando SET-WINDOW risulterà essere

```
SET-WINDOW(-0.5, 0.5, -0.5, 0.5);
```

L'orientamento del piano di proiezione corrisponde all'orientamento del para-

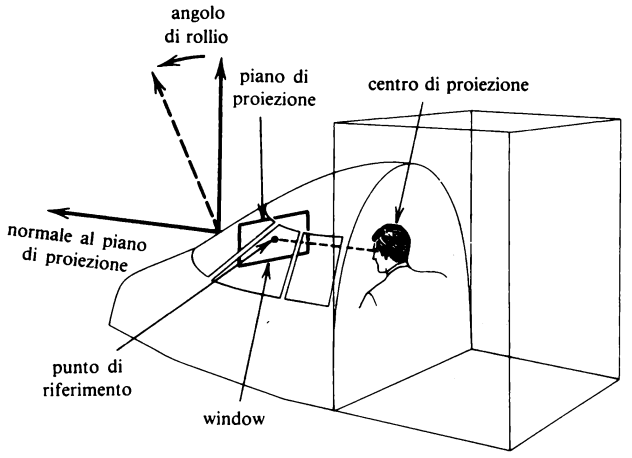


Figura 8.38 Parametri per il simulatore di volo

brezza del velivolo, che viene assegnato dalla routine SET-VIEW-PLANE-NORMAL. Il pilota potrà imporre una rotazione di beccheggio o di rollio modificando il valore del vettore che specifica la direzione verticale. Infine, gli ultimi parametri da assegnare sono quelli che fissano la proiezione di tipo prospettico ed il punto di vista coincidente con gli occhi del pilota che si trova a circa 0.5 metri dallo schermo (vedi Figura 8.38).
Si possono evidenziare due routine principali fra quelle che realizzano la simulazione del volo: una viene richiamata nel corso degli spostamenti del velivolo e un'altra aggiorna l'assetto del velivolo corrispondentemente ai comandi del pilota. Queste operazioni di aggiornamento della posizione dell'aereo si ottengono con la modifica del punto di riferimento per l'osservazione e del centro di proiezione.

```
begin
   SET-VIEW-REFERENCE-POINT(EAST-WEST, ALTITUDE,
   NORTH-SOUTH);
   SET-PERSPECTIVE(EAST-WEST - 0.5 * EW-DIRECTION,
      ALTITUDE - 0.5 * AL-DIRECTION,
      NORTH-SOUTH - 0.5 * NS-DIRECTION);
   CREATE-SEGMENT(2);
   SET-VISIBILITY(2, FALSE);
   BUILD-WORLD;
   CLOSE-SEGMENT;
   SET-VISIBILITY(1, FALSE);
   SET-VISIBILITY(2, TRUE);
   DELETE-SEGMENT(1);
   RENAME-SEGMENT(2, 1);
   return;
end;
```

In questo programma si sono usate le variabili EAST-WEST, ALTITUDE, NORTH-SOUTH come coordinate della posizione del velivolo. Il vettore [EW-DIRECTION, AL-DIRECTION, NS-DIRECTION] fornisce l'orientamento del velivolo. Nel comando SET-PERSPECTIVE si usa 0.5 come distanza del pilota dal parabrezza. Questo valore viene proiettato nella direzione del velivolo e il risultato viene sottratto dalla posizione del velivolo in modo da ottenere la posizione del pilota. Per assicurare che la vecchia immagine rimanga sullo schermo finché la nuova non è completa, si utilizzano quegli strumenti del sistema grafico che permettono di cambiare il nome ai segmenti e modificarne l'attributo di visibilità. La routine che aggiorna l'orientamento dell'aereo può essere scritta nel modo seguente:

```
begin
   SET-VIEW-PLANE-NORMAL(EW-DIRECTION, AL-DIRECTION,
      NS-DIRECTION);
```

```
        SET-VIEW-UP(SIN(BANK-ANGLE)*NS-DIRECTION,
            COS(BANK-ANGLE)*SQRT(EW-DIRECTION↑2+
            NS-DIRECTION↑2),-SIN(BANK-ANGLE)*EW-DIRECTION);
        return;
    end;
```

In questa procedura, la variabile BANK-ANGLE contiene il valore dell'inclinazione del velivolo verso sinistra. Se quest'angolo è uguale a zero il volo è livellato (volo rettilineo orizzontale), se invece è negativo significa che l'aereo è inclinato verso destra. Tramite la chiamata alla SET-VIEW-UP, viene calcolato un vettore — espresso in termini di coordinate oggetto — che rappresenta la direzione verticale nel volo livellato.

ESERCIZI

8.1 Scrivere l'equazione in forma parametrica della retta passante per le seguenti coppie di punti:

a) (0, 0, 0); (1, 2, 3)
b) (1, 5, 4); (2, 2, 3)
c) (8, 4, 1); (5, 2, −1)

8.2 Scrivere l'equazione dei seguenti piani:

a) passante per il punto (0, 0, 0) e normale al vettore [0, 0, 1]
b) passante per il punto (0, 0, 0) e normale al vettore [0, 0, −1]
c) passante per il punto (0, 0, 0) e normale al vettore [1, 1, 0]
d) passante per il punto (1, 1, 2) e normale al vettore [4, 2, 3]

8.3 Scrivere la matrice di trasformazione tridimensionale in coordinate omogenee per le seguenti trasformazioni:

a) traslazione di 0.5 in direzione x, di 0 in direzione y e di −0.2 in direzione z
b) dimezzamento del fattore di scala per l'asse z
c) triplicazione del fattore di scala per gli assi x e y
d) rotazione di $\pi/4$ attorno all'asse x (y verso z)
e) rotazione di $\pi/3$ attorno all'asse y (x verso z)
f) rotazione di π rispetto alla linea congiungente i punti (0, 0, 0) e (1, 0, 1)

8.4 Sia dato l'oggetto di Figura 8.39. Si mostri come apparirà la figura sullo schermo utilizzando ciascuno dei seguenti parametri di proiezione:

a) tutti i parametri di default tranne SET-VIEW-REFERENCE-POINT (0.2, 0.2, −0.2)
b) tutti i parametri di default tranne SET-PARALLEL(0.2, 0, 1)
c) tutti i parametri di default tranne SET-PARALLEL(−0.2, 0, 1)
d) tutti i parametri di default tranne SET-PARALLEL(0, −0.2, 1)

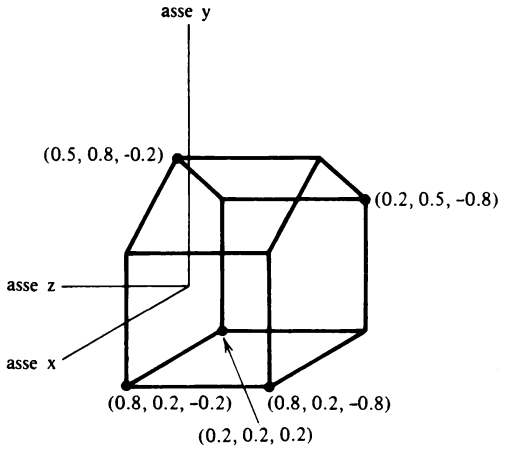


Figura 8.39

e) tutti i parametri di default tranne SET-PARALLEL(0, 0.2, 1)
f) tutti i parametri di default tranne SET-VIEW-PLANE-NORMAL(−0.5, 0, −1)
g) tutti i parametri di default tranne SET-VIEW-PLANE-NORMAL(0, 0.5, −1)
h) tutti i parametri di default tranne SET-PARALLEL(0.5, 0, 1)
SET-VIEW-PLANE-NORMAL
(−0.5, 0, −1)
i) tutti i parametri di default tranne SET-PARALLEL(0, 0.5, 1)
SET-VIEW-PLANE-NORMAL
(0, −0.5, −1)
j) tutti i parametri di default tranne SET-PARALLEL(0.5, 0.5, 1)
SET-VIEW-PLANE-NORMAL
(−0.5, −0.5, −1)
k) tutti i parametri di default tranne SET-PARALLEL(0.5, 0, 1)
SET-VIEW-PLANE-NORMAL(−0.5, 0, −1)
SET-VIEW-REFERENCE-POINT
(0.2, 0.2, −0.2)
l) tutti i parametri di default tranne SET-PARALLEL(1, 0, 0)
SET-VIEW-PLANE-NORMAL(−1, 0, 0)
SET-VIEW-REFERENCE-POINT
(0.2, 0.2, −0.2)
m) tutti i parametri di default tranne SET-PARALLEL(0.2, 0, 1)
SET-VIEW-DISTANCE(0.2)
n) tutti i parametri di default tranne SET-PARALLEL(0.2, 0, 1)
SET-VIEW-DISTANCE(−0.2)
o) tutti i parametri di default tranne SET-VIEW-PLANE-NORMAL(−1, 0, 0)
SET-PARALLEL(1, 0, 0)
SET-VIEW-REFERENCE-POINT
(0.8, 0.2, −0.2)

p) tutti i parametri di default tranne
```
SET-VIEW-UP(1, 1, 0)
```
q) tutti i parametri di default tranne
```
SET-PARALLEL(0, 1, 0)
SET-VIEW-PLANE-NORMAL(0, -1, 0)
SET-VIEW-REFERENCE-POINT
      (0.2, 0.2, -0.2)
SET-VIEW-UP(0, 0, -1)
```
r) tutti i parametri di default tranne
```
SET-VIEW-UP(-1, 0, 0)
SET-VIEW-REFERENCE-POINT
      (0.8, 0.2, -0.2)
SET-VIEW-PLANE-NORMAL(0, -1, 0)
SET-PARALLEL(0, 1, 0)
```

PROBLEMI DI PROGRAMMAZIONE

8.1 Implementare gli algoritmi dall'8.1 all'8.27 estendendo il sistema grafico alle tre dimensioni.

8.2 Collaudare queste routine, inserendo la descrizione di una casetta in tre dimensioni, visualizzandola sia in proiezione prospettica che parallela. Quindi si modifichino leggermente i parametri di visualizzazione per vedere come agiscono.

8.3 Scrivere le routine:
```
INQUIRE-VIEW-REFERENCE-POINT(X, Y, Z)
INQUIRE-VIEW-PLANE-NORMAL(DX, DY, DZ)
INQUIRE-VIEW-DISTANCE(D)
INQUIRE-VIEW-UP(DX, DY, DZ)
INQUIRE-PARALLEL(ON-OFF, DX, DY, DZ)
INQUIRE-PERSPECTIVE(ON-OFF, X, Y, Z)
```
che restituiscono all'utente le ultime assegnazioni dei parametri di visualizzazione.

8.4 a) Rappresentare un cubo in proiezione isometrica, dimetrica e trimetrica.
b) Rappresentare un cubo in proiezione cavaliera e cabinet.
c) Rappresentare un cubo in proiezione prospettica ad un punto, a due punti e a tre punti.

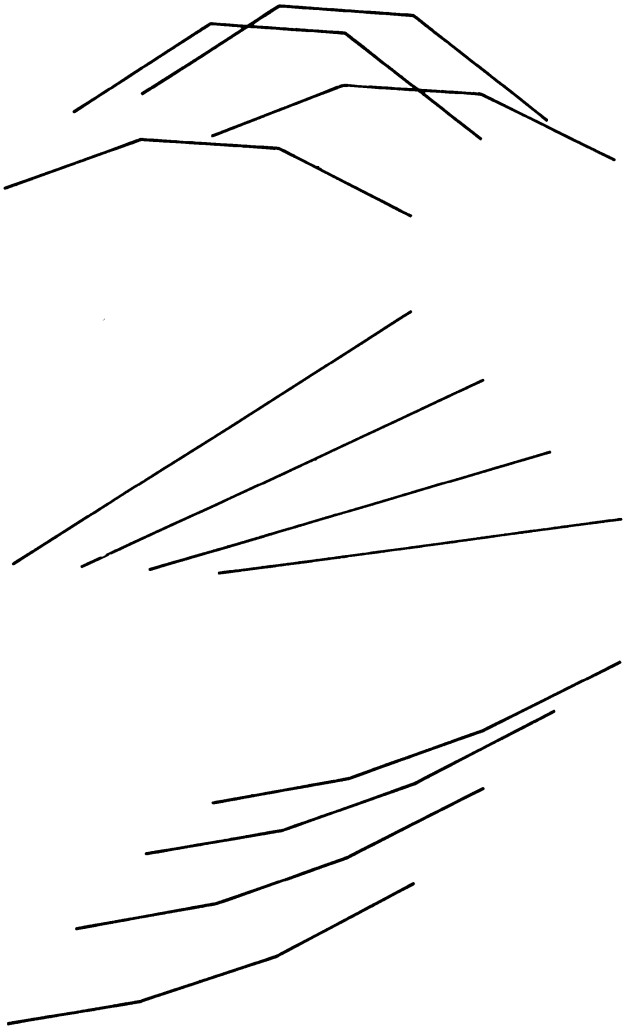
8.5 Scrivere un programma che permette la visualizzazione di oggetti tridimensionali generati con la procedura FOO, in modo da dare l'impressione di girare attorno alla scena. Considerare l'oggetto contenuto in un cubo con vertici di riferimento in (0, 0, 0) e (1, 1, −1).

8.6 Scrivere un programma che, data la funzione

$$Y = FUN(X, Z)$$

mostri la superficie da essa rappresentata tramite una serie di curve di livello (a z costante) per $z = 0.2, 0.4, 0.6, 0.8$ che colleghino i punti con $x = 0.2, 0.4, 0.6, 0.8$. Scegliere una trasformazione che la visualizzi al meglio. La funzione sia per esempio:

a) FUN(X, Z) = SIN(X)*SIN(Z)
b) FUN(X, Z) = X*Z
c) FUN(X, Z) = 1+X12−Z12



***8.7** Scrivere un programma interattivo che permetta la visualizzazione di oggetti tridimensionali, consentendo all'utente di selezionarli da un menu e di modificare i parametri di visualizzazione.

9

Clipping tridimensionale

INTRODUZIONE

Si immagini di essere di fronte a un edificio e di fissare l'attenzione sull'ingresso principale; via via che ci si avvicina, le dimensioni della porta aumentano finché non la si attraversa e i muri esterni spariscono: ci si trova nell'atrio dell'edificio. Se si gira a destra e si oltrepassa un'altra porta, si giunge in un ufficio (vedi Figura 9.1).
Per descrivere graficamente queste scene, bisogna considerare due nuove tecniche: la prima è l'eliminazione di linee e superfici nascoste, l'altra è l'estensione degli algoritmi di clipping al caso tridimensionale.
In questo capitolo verrà considerato il clipping in tre dimensioni, mentre l'eliminazione delle linee e delle superfici nascoste verrà affrontata nel Capitolo 10. I cambiamenti che verranno apportati alla tecnica di clipping consentiranno di eliminare la parte anteriore e quella posteriore di un oggetto, non soltanto i suoi lati. Se si pensa all'esempio dell'edificio, è possibile notare che la tecnica del clipping viene applicata non appena l'osservatore oltrepassa la soglia e quindi non è più in grado di vedere la facciata dell'edificio (vedi Figura 9.2).
Un altro motivo per eseguire il clipping 3-D è dato dalla possibilità di determinare le coordinate z dei punti di intersezione tra le linee e il *piano di clipping*: quest'informazione risulta utilissima nell'eliminazione delle linee nascoste.
L'algoritmo che realizza l'eliminazione delle linee nascoste è piuttosto pesante dal punto di vista computazionale ed è quindi auspicabile eseguire il clipping prima di mandarlo in esecuzione, così da ridurre al minimo il numero di linee da elaborare; l'eliminazione delle linee nascoste, inoltre, richiede l'ordinamento per profondità di linee e superfici di cui fanno parte gli spigoli generati dal clipping. Per questi motivi il clipping va eseguito prima che l'informazione sulla profondità dell'oggetto vada persa con l'esecuzione della sua proiezione.

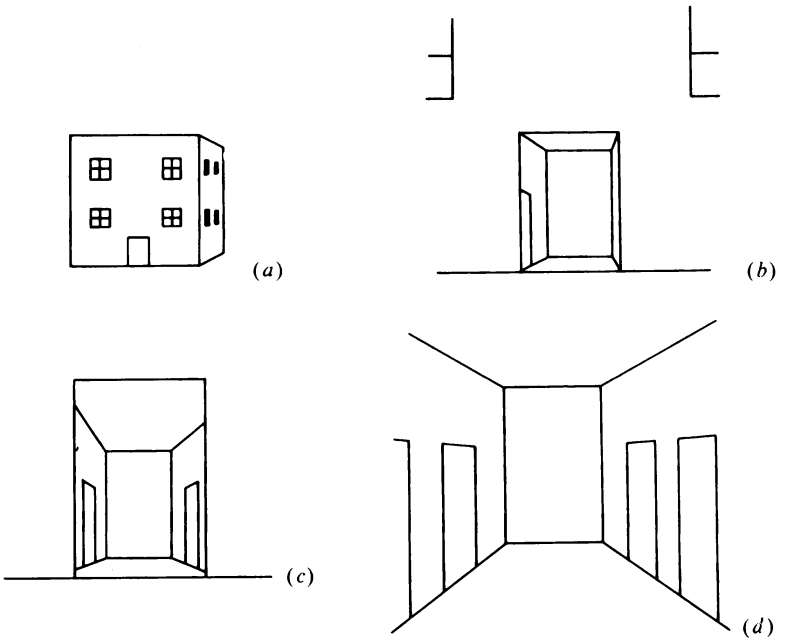


Figura 9.1 Percorso all'interno di un palazzo

9.1 QUANDO ESEGUIRE IL CLIPPING

Nel Capitolo 8 è stata proposta una serie di trasformazioni dell'immagine che eseguivano dapprima una conversione in coordinate piane, proiettavano sul piano di visualizzazione (fornendo un'immagine bidimensionale) ed infine eseguivano un clipping lungo i bordi della finestra (vedi Figura 9.3).
L'ordine delle operazioni va ora modificato nel modo seguente: conversione in coordinate piane, clipping dell'oggetto tridimensionale eseguito con un parallelepipedo ed infine proiezione del risultato sul piano di visualizzazione (vedi Figura 9.4). In parole povere, occorre estrarre l'operazione di proiezione dalla routine DISPLAY-FILE-ENTER e porla alla fine della procedura di clipping. La routine DISPLAY-FILE-ENTER diventerà:

Algoritmo 9.1 **DISPLAY-FILE-ENTER (OP)** (Algoritmo 8.24 aggiornato)
Argomenti OP←codice operativo dell'istruzione da inserire
Variabili globali DF-PEN-X, DF-PEN-Y, DF-PEN-Z←posizione corrente della penna
PERSPECTIVE-FLAG indicatore del tipo di proiezione (parallela o prospettica)

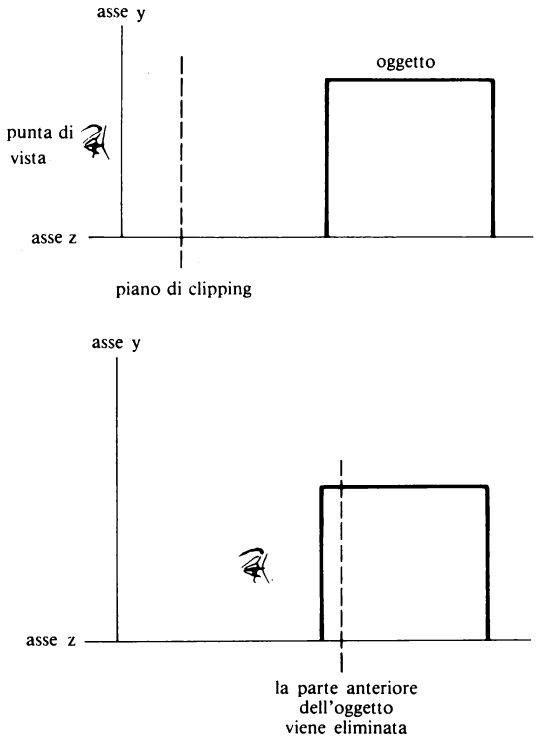


Figura 9.2 Spostamento del piano di clipping attraverso un oggetto

```
begin
   if OP<0 then PUT-POINT(OP, 0, 0)
   else begin
          X←DF-PEN-X;
          Y←DF-PEN-Y;
          Z←DF-PEN-Z;
          VIEW-PLANE-TRANSFORM(X, Y, Z);
          CLIP(OP, X, Y, Z);
          end;
   return;
end;
```

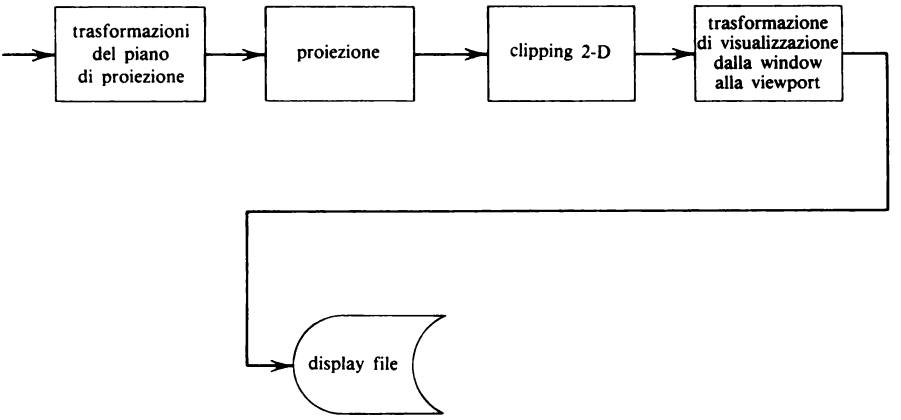


Figura 9.3 Operazioni di visualizzazione effettuate con il clipping bidimensionale

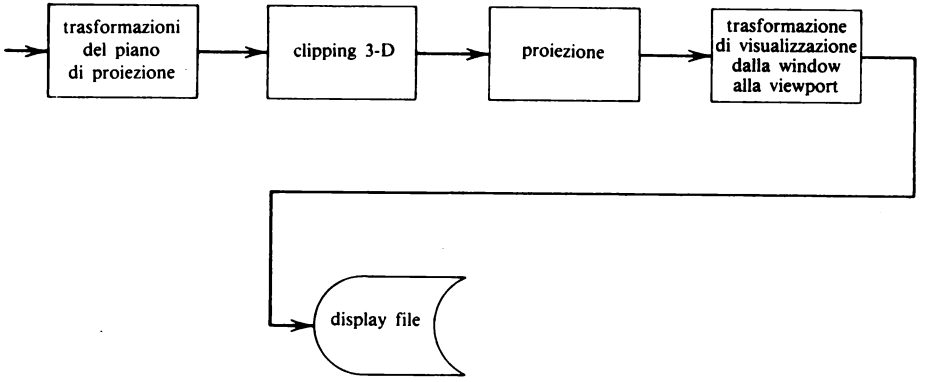


Figura 9.4 Operazioni di visualizzazione effettuate con il clipping tridimensionale

9.2 CLIPPING DI VOLUMI

Estendere l'operazione di clipping bidimensionale al caso tridimensionale non è un'operazione difficile, in quanto il metodo resta fondamentalmente lo stesso. La differenza tra i due casi consiste nel modo con cui si decide se un punto appartiene ad una regione visibile sullo schermo: in pratica, si dovrà stabilire dove si trova un punto rispetto a un piano invece che rispetto a una retta. Occorre distinguere fra due tipi di regioni entro cui deve trovarsi l'oggetto da visualizza-

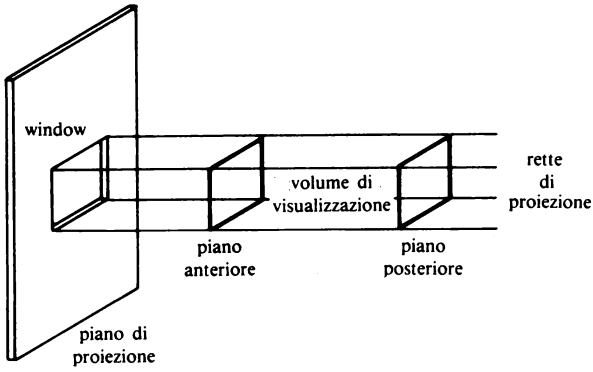


Figura 9.5 Volume di visualizzazione per una proiezione parallela

re, a seconda che la proiezione sia parallela o prospettica. Questa regione viene detta *volume di visualizzazione*. Nel caso di proiezione parallela, l'immagine da visualizzare è contenuta in un tubo di sezione rettangolare uscente dalla window e i cui piani sono paralleli alla direzione di proiezione. I due ulteriori piani di clipping necessari a trasformare il tubo in un parallelepipedo saranno perpendicolari alla direzione di proiezione. Gli oggetti che si trovano in questa regione dello spazio vengono visualizzati (vedi Figura 9.5).
Nel caso invece di proiezione prospettica, una serie di raggi, uscenti dal centro di proiezione e passanti per i contorni della window, formano quella che viene chiamata *piramide di visualizzazione*, che può essere troncata con due piani analogamente al caso della proiezione parallela (vedi Figura 9.6).

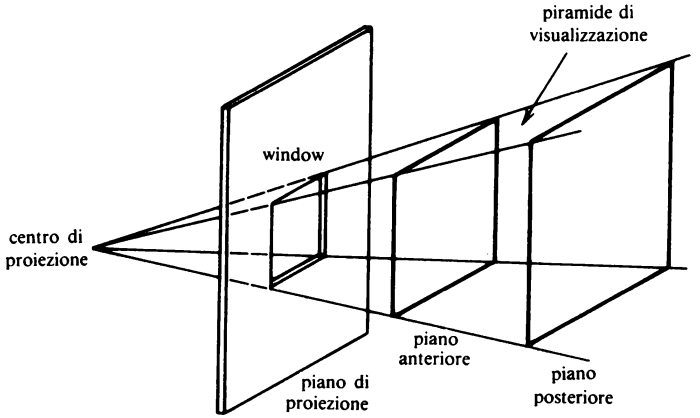


Figura 9.6 Piramide di visualizzazione per una proiezione prospettica

In entrambi i casi, si avrà a che fare con un volume delimitato da sei piani di clipping: i piani ottenuti proiettando la window vengono definiti automaticamente, mentre i piani di troncamento vanno specificati con routine apposite. Si tratta sostanzialmente di memorizzare le posizioni di questi piani, assegnate dall'utente come variabili globali. L'algoritmo seguente consente la memorizzazione di queste posizioni in termini di distanza dal punto di vista nella direzione della normale al piano di proiezione.

Algoritmo 9.2 SET-VIEW-DEPTH (FRONT-DISTANCE, BACK-DISTANCE)

Routine utente che posiziona i piani di troncamento.

Argomenti FRONT-DISTANCE, BACK-DISTANCE←distanza del piano dal punto di vista lungo la normale al piano di proiezione

Variabili globali FRONT, BACK←posizioni dei piani

```
begin
   if FRONT-DISTANCE>BACK-DISTANCE
   then return-error 'PIANO ANTERIORE DIETRO AL PIANO
      POSTERIORE';
   FRONT←FRONT-DISTANCE;
   BACK←BACK-DISTANCE;
   return;
end;
```

Il test di clipping relativo ai piani di troncamento può essere omesso usando un flag, che fornisce così all'utente la possibilità di attivarlo o meno. Di seguito sono riportate le routine per assegnare il valore del flag.

Algoritmo 9.3 SET-FRONT-PLANE-CLIPPING (ON-OFF)

Routine utente che assegna il valore al flag che attiva il test di clipping relativo al piano di troncamento anteriore.

Argomenti ON-OFF←valore assegnato dall'utente

Variabili globali FRONT-FLAG←indicatore relativo al piano di troncamento anteriore

```
begin
   FRONT-FLAG←ON-OFF;
   return;
end;
```

Algoritmo 9.4 SET-BACK-PLANE-CLIPPING (ON-OFF)

Routine utente che assegna il valore al flag che attiva il test di clipping relativo al piano di troncamento posteriore.

Argomenti ON-OFF←valore assegnato dall'utente

Variabili globali BACK-FLAG←indicatore relativo al piano di troncamento posteriore

```
begin
   BACK-FLAG←ON-OFF;
   return;
end;
```

9.3 PIANI DI CLIPPING

Il problema centrale del clipping è quello di decidere da quale parte si trova un punto rispetto a un piano, problema che risulta analogo a quello di stabilire da quale parte si trova un punto di un piano rispetto a una retta ad esso appartenente.
L'equazione di un piano è del tipo

$$Ax + By + Cz + D = 0 \tag{9.1}$$

Sostituendo le coordinate di un punto (x_1, y_1, z_1) nell'equazione, esso giace sul piano se

$$Ax_1 + By_1 + Cz_1 + D = 0 \tag{9.2}$$

Se il punto non giace sul piano, il risultato è diverso da 0, positivo se si trova da una parte rispetto al piano

$$Ax_1 + By_1 + Cz_1 + D > 0 \tag{9.3}$$

e negativo se si trova dall'altra

$$Ax_1 + By_1 + Cz_1 + D < 0 \tag{9.4}$$

È possibile quindi stabilire se un punto è all'interno dei confini di clipping, esaminando il segno dell'espressione ottenuta dall'equazione del piano.
I due piani di troncamento devono essere paralleli al piano di proiezione e nel sistema di coordinate ad esso relative sono

$$z = \text{FRONT-Z} \quad \text{e} \quad z = \text{BACK-Z} \tag{9.5}$$

dove FRONT-Z e BACK-Z sono costanti che forniscono la posizione di questi piani sull'asse z (vedi Figura 9.7).
Per essere visibile, il punto (x_1, y_1, z_1) deve essere dietro al piano di troncamento anteriore, davanti al piano di troncamento posteriore, oppure appartenente a uno dei due. Le verifiche devono essere

$$z_1 \leq \text{FRONT-Z} \quad \text{e} \quad z_1 \geq \text{BACK-Z} \tag{9.6}$$

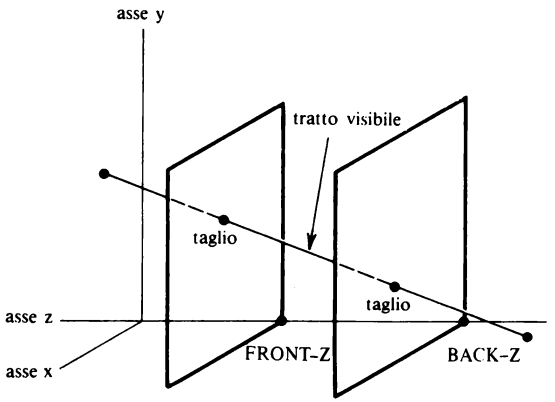


Figura 9.7 Piani di clipping anteriore e posteriore relativi a un segmento di retta

È possibile definire le equazioni degli altri piani in modo semplice, se si immagina di osservare i piani di proiezione con una vista laterale (vedi Figura 9.8). Per esempio, osservando il piano superiore del tubo rettangolare, esso apparirà come una retta; nel caso di una proiezione parallela in direzione [VXP VYP VZP], si tratta di una retta passante per il punto appartenente allo spigolo della window

$$(y, z) = (\text{WYH}, 0) \tag{9.7}$$

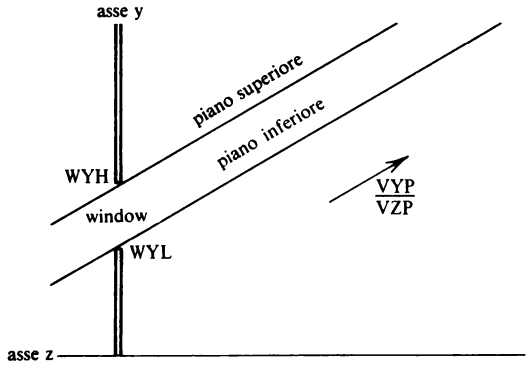


Figura 9.8 Piani di clipping superiore e inferiore per una proiezione parallela

avente pendenza

$$\frac{\text{incremento } y}{\text{incremento } z} = \frac{\text{VYP}}{\text{VZP}} \tag{9.8}$$

L'equazione finale risulta

$$y = \frac{\text{VYP}}{\text{VZP}} z + \text{WYH} \tag{9.9}$$

Essa rappresenta anche l'equazione del piano parallelo all'asse x. Analogamente, il piano inferiore del tubo e quelli laterali sono dati rispettivamente dalle

$$\begin{aligned} y &= \frac{\text{VYP}}{\text{VZP}} z + \text{WYL} \\ x &= \frac{\text{VXP}}{\text{VZP}} z + \text{WXH} \\ x &= \frac{\text{VXP}}{\text{VZP}} z + \text{WXL} \end{aligned} \tag{9.10}$$

Nel caso della prospettiva, il piano di clipping superiore apparirà come una retta passante per il punto superiore della window

$$(y, z) = (\text{WYH}, 0) \tag{9.11}$$

e il centro di proiezione sarà

$$(y, z) = (y_c, z_c) \tag{9.12}$$

L'equazione di tale retta è

$$y = \frac{y_c - \text{WYL}}{z_c} z + \text{WYH} \tag{9.13}$$

L'equazione è la stessa di quella del piano parallelo all'asse x. Allo stesso modo il piano inferiore e quelli laterali (vedi Figura 9.9) sono dati rispettivamente dalle

$$\begin{aligned} y &= \frac{y_c - \text{WYL}}{z_c} z + \text{WYL} \\ x &= \frac{x_c - \text{WXH}}{z_c} z + \text{WXH} \\ x &= \frac{x_c - \text{WXL}}{z_c} z + \text{WXL} \end{aligned} \tag{9.14}$$

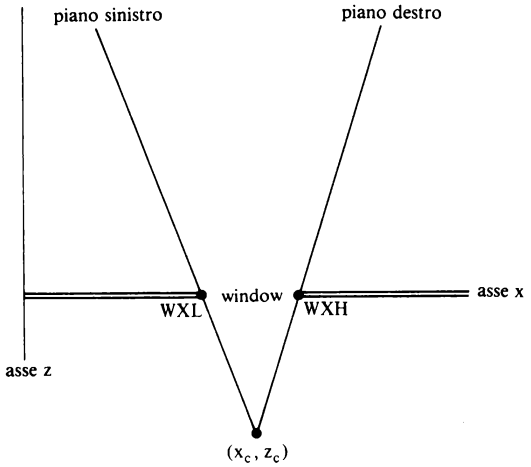


Figura 9.9 Piani di clipping destro e sinistro per una proiezione prospettica

Le equazioni di questi piani di clipping sono le stesse sia per proiezioni parallele che per proiezioni prospettiche, a patto che si sostituisca l'espressione della pendenza con una variabile. Per esempio, il piano superiore nel caso di proiezione parallela è

$$y = \text{YHM}\, z + \text{WYH} \tag{9.15}$$

dove

$$\text{YHM} = \frac{\text{VYP}}{\text{VZP}} \tag{9.16}$$

e, nel caso di proiezione prospettica, è

$$\text{YHM} = \frac{y_c - \text{WYH}}{z_c} \tag{9.17}$$

9.4 DETERMINAZIONE DEI PARAMETRI DI CLIPPING

Il sistema proposto in questo testo è strutturato in modo che i parametri di visualizzazione del segmento di display file vengono assegnati solo quando il segmento è stato creato tramite la routine NEW-VIEW-3, che viene richiamata dalla CREATE-SEGMENT. Ora servono anche i parametri di clipping e quindi

l'algoritmo NEW-VIEW-3 va esteso con l'aggiunta delle chiamate a due ulteriori routine: MAKE-CLIPPING-CONSTANTS e MAKE-CLIPPING-TESTS.

Algoritmo 9.5 NEW-VIEW-3 (Algoritmo 8.25 aggiornato)
Crea una nuova trasformazione di visualizzazione globale.

```
begin
   NEW-VIEW-2;
   MAKE-VIEWING-TRANSFORM;
   MAKE-CLIPPING-CONSTANTS;
   MAKE-CLIPPING-TESTS;
   return;
end;
```

La routine MAKE-CLIPPING-TESTS, come si vedrà in seguito, definisce le condizioni per il test con il piano di clipping relative al «vecchio» estremo del segmento da troncare.
Nella routine MAKE-CLIPPING-CONSTANTS, la conversione delle posizioni dei piani di troncamento anteriore e posteriore avviene da distanze lungo la normale al piano di proiezione a valori di z nel sistema di coordinate del piano di proiezione (vedi Figura 9.10). Si può anche calcolare la pendenza dei piani di clipping della window, tramite l'algoritmo seguente:

Algoritmo 9.6 MAKE-CLIPPING-CONSTANTS
Calcola alcuni parametri relativi al clipping 3-D.
Variabili globali FRONT, BACK←distanze dei piani di troncamento anteriore e posteriore dal punto di vista
VIEW-DISTANCE←distanza del piano di proiezione dal punto di vista

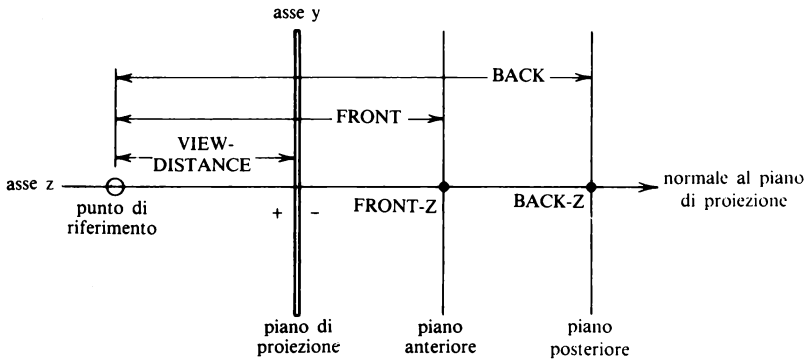


Figura 9.10 Parametri di definizione dei piani anteriore e posteriore

```
                    FRONT-Z, BACK-Z←posizioni dei piani di troncamento
                      nelle coordinate del piano di proiezione
                    PERSPECTIVE-FLAG←tipo di proiezione
                    VXP, VYP, VZP←vettori di proiezione
                    XC, YC, ZC←centro di proiezione prospettica
                    WXL, WXH, WYL, WYH←dimensioni della window
                    XLM, XHM, YLM, YHM←pendenze dei piani di clip-
                      ping relativi alla window
begin
   FRONT-Z←VIEW-DISTANCE-FRONT;
   BACK-Z←VIEW-DISTANCE-BACK;
   if PERSPECTIVE-FLAG
   then begin
        XLM←(XC-WXL)/ZC;
        XHM←(XC-WXH)/ZC;
        YLM←(YC-WYL)/ZC;
        YHM←(YC-WYH)/ZC;
        end
   else begin
        XLM←VXP/VZP;
        XHM←XLM;
        YLM←VYP/VZP;
        YHM←YLM;
        end;
   return;
end;
```

La routine MAKE-CLIPPING-TESTS calcola le condizioni di verifica corrispondenti ai valori iniziali del «vecchio» estremo del segmento da troncare.

Algoritmo 9.7 MAKE-CLIPPING-TESTS
Inizializza le condizioni dei «vecchi» estremi.

```
Variabili globali  OLD-POINT-TEST←array contenente le condizioni di
                     verifica dei 4 piani di clipping uscenti dalla window
                   XLM, XHM, YLM, YHM←pendenze dei piani di clip-
                     ping
                   WXL, WXH, WYL, WYH←contorni della window
                   ZS←array contenente le coordinate z dei «vecchi» estremi
begin
   OLD-POINT-TEST[1]←XLM*ZS[1]+WXL;
   OLD-POINT-TEST[2]←XHM*ZS[2]+WXH;
   OLD-POINT-TEST[3]←YLM*ZS[3]+WYL;
```

```
    OLD-POINT-TEST[4] ← YHM * ZS[4] + WYH;
    return;
end;
```

9.5 CLIPPING CON UN PIANO

In questo paragrafo verranno usate le costanti definite precedentemente. Si ricordi che, per essere visibile, un punto (x_1, y_1, z_1) deve stare al di sotto del piano di clipping superiore e quindi la verifica corrispondente sarà

$$y_1 \leq \text{YHM } z_1 + \text{WYH} \tag{9.18}$$

Il punto dovrà anche stare al di sopra del piano inferiore e la verifica che esprime questo vincolo sarà

$$y_1 \geq \text{YLM } z_1 + \text{WYL} \tag{9.19}$$

Analogamente alle precedenti, si definiscono le condizioni rispetto ai piani che delimitano a destra e a sinistra la regione di clipping (vedi Figura 9.11)

$$x_1 \leq \text{XHM } z_1 + \text{WXH} \quad \text{e} \quad x_1 \geq \text{XLM } z_1 \times \text{WXL} \tag{9.20}$$

Le variabili OLD-POINT-TEST, che vengono valutate nell'algoritmo 9.7, costituiscono i risultati delle verifiche applicate alla parte destra delle equazioni (9.18), (9.19) e (9.20), relativamente ad uno dei punti che devono essere controllati.

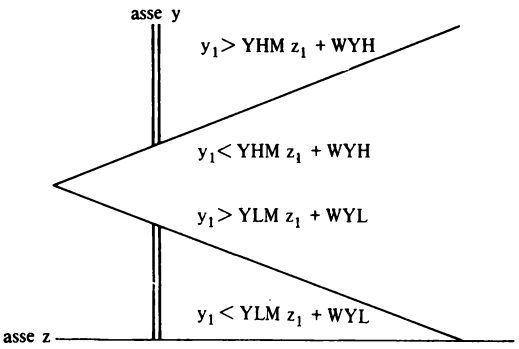


Figura 9.11 Verifica della posizione di un punto rispetto ai piani di clipping superiore e inferiore

Vengono ora esaminate le operazioni da compiere nel caso in cui un segmento intersechi i confini della regione visibile. Ciò che si vuol ottenere è che questo segmento risulti invisibile a partire dal punto in cui interseca il piano. L'intersezione tra il segmento di estremi (x_1, y_1, z_1) e (x_2, y_2, z_2) e il piano si può determinare considerando le equazioni della retta

$$\begin{aligned} x &= \frac{x_2-x_1}{z_2-z_1}(z-z_1)+x_1 \\ y &= \frac{y_2-y_1}{z_2-z_1}(z-z_1)+y_1 \end{aligned} \tag{9.21}$$

e ciascuno dei piani di clipping.
Per quanto riguarda i piani che limitano anteriormente e posteriormente la regione, per determinare il punto è sufficiente sostituire nelle equazioni (9.21) il valore della coordinata z che definisce ciascun piano. Ad esempio, per il piano anteriore si avrà:

$$z = \text{FRONT-Z} \tag{9.22}$$

ed il punto di intersezione avrà coordinate

$$\begin{aligned} x &= \frac{x_2-x_1}{z_2-z_1}(\text{FRONT-Z}-z_1)+x_1 \\ y &= \frac{y_2-y_1}{z_2-z_1}(\text{FRONT-Z}-z_1)+y_1 \\ z &= \text{FRONT-Z} \end{aligned} \tag{9.23}$$

Nel caso invece dei piani laterali, bisognerà risolvere l'equazione rispetto alla coordinata z prima di eseguire le sostituzioni. Considerando ad esempio il piano superiore, si avrà

$$y = \text{YHM}\ z + \text{WYH} \tag{9.24}$$

e il punto di intersezione si determina tramite l'equazione

$$\text{YHM}\ z + \text{WYH} = y = \frac{y_2-y_1}{z_2-z_1}(z-z_1)+y_1 \tag{9.25}$$

che risolta rispetto a z risulta essere

$$z = \frac{(y_2-y_1)z_1+(\text{WYH}-y_1)(z_2-z_1)}{(y_2-y_1)-\text{YHM}(z_2-z_1)} \tag{9.26}$$

Questo valore della z può essere quindi sostituito nell'equazione (9.21), permettendo di determinare i valori delle coordinate x e y del punto in cui va eseguito il clipping. Con procedimento analogo, si possono determinare le equazioni per il calcolo del punto di intersezione degli altri piani.
L'algoritmo seguente rappresenta il procedimento per calcolare l'intersezione per tutti e quattro i piani laterali di clipping.

Algoritmo 9.8 CLIPPED-Z (A1, Z1, A2, Z2, M, B)
Calcola il valore della coordinata z del punto di intersezione tra una retta e uno dei quattro piani laterali di clipping.
Argomenti A1, Z1←coordinate di uno degli estremi del segmento
A2, Z2←coordinate dell'altro estremo del segmento
M←inclinazione del piano di clipping
B←intercetta del piano di clipping con l'asse z

```
begin
   CLIPPED-Z←((A2-A1)*Z1+(B-A1)*(Z2-Z1)) /
                (A2-A1-M*(Z2-Z1));
   return;
end;
```

Un'altra coordinata del punto di intersezione può essere determinata a partire dall'equazione del piano di clipping; considerando, ad esempio, il piano superiore, è possibile calcolare la coordinata y in questo modo

$$\text{Y-CLIP} = \text{YHM} * \text{Z-CLIP} + \text{WYH} \tag{9.27}$$

Per determinare il valore della coordinata x verranno usate, infine, le equazioni della retta. In questo caso è possibile scrivere una generica routine per sostituire il valore della z nell'equazione della retta e calcolare il valore dalla x. Bisogna però considerare il caso particolare in cui la retta è perpendicolare all'asse z, per cui risulta $z_1 = z_2 =$ costante e quindi il valore del coefficiente angolare risulta infinito. Bisogna quindi riscrivere le equazioni in termini di una delle altre due coordinate:

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1$$

oppure (9.28)

$$x = \frac{x_2 - x_1}{y_2 - y_1}(y - y_1) + x_1$$

Come negli altri casi, si potranno quindi sostituire i valori delle altre coordinate ottenute dalle equazioni dei piani di clipping. Se poi il coefficiente angolare diviene infinito per la seconda forma delle equazioni della retta, allora si potrà concludere che la retta è contenuta nel piano di clipping. In questo caso, si po-

trà fornire come risultato dell'intersezione uno qualsiasi degli estremi del segmento che si sta considerando.
Il seguente algoritmo calcola la terza coordinata del punto di intersezione.

Algoritmo 9.9 CLIPPED-X-OR-Y (A1, B1, Z1, A2, B2, Z2, A-CLIP, Z-CLIP)

Calcola la terza coordinata del punto di intersezione tra il segmento e il piano di clipping.

Argomenti A1, B1, Z1 e A2, B2, Z2←coordinate degli estremi del segmento
B1, B2←coordinata da determinare
Z-CLIP←coordinata z del punto di intersezione
A-CLIP←seconda coordinata nota del punto di intersezione

Costanti ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
   if |Z2 − Z1| > ROUNDOFF
   then CLIPPED-X-OR-Y
            ←(B2 − B1)/(Z2 − Z1) * (Z-CLIP − Z1) + B1;
   else if |A2 − A1| > ROUNDOFF
      then CLIPPED-X-OR-Y
               ←(B2 − B1)/(A2 − A1) * (A-CLIP − A1) + B1;
      else CLIPPED-X-OR-Y←B1;
   return;
end;
```

9.6 ALGORITMI DI CLIPPING 3-D

È possibile ora scrivere gli algoritmi di clipping del caso tridimensionale, come estensioni di quelli dati nel Capitolo 6. Ciascun punto visibile viene valutato dai diversi algoritmi che controllano la sua posizione rispetto a ciascuno dei piani di clipping (vedi Figura 9.12).
Tramite una serie di test sul nuovo estremo del segmento e sul vecchio (memorizzato insieme alle condizioni di verifica ad esso relative), si stabilisce se il segmento attraversa il piano di clipping: se lo attraversa, vengono calcolate le coordinate del punto di intersezione, che sono poi passate alla successiva fase di clipping, nella quale verrà memorizzato come «vecchio» estremo assieme alle condizioni di verifica ad esso relative.
Infine, il punto viene sottoposto ad un controllo di visibilità (poiché potrebbe essere nascosto da altri oggetti) e, se lo supera, viene passato alla successiva fase di clipping.

Algoritmo 9.10 CLIP-LEFT (OP, X, Y, Z) (Algoritmo 6.6 aggiornato)
Esegue il clipping sul lato sinistro.

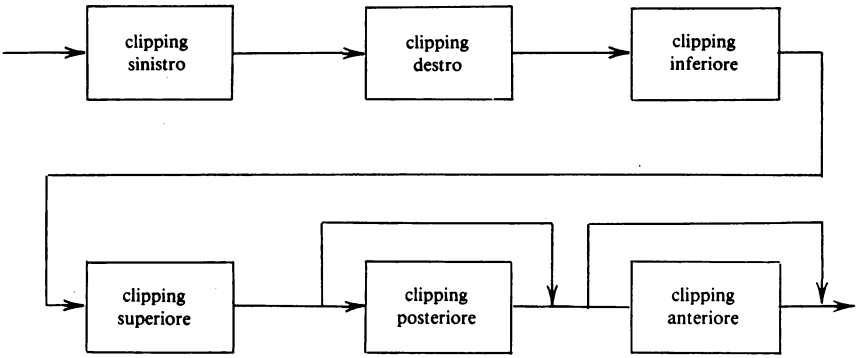


Figura 9.12 Procedura di clipping 3-D

Argomenti OP, X, Y, Z←dati relativi al nuovo estremo
Variabili globali WXL←lato sinistro della window
XLM←pendenza del piano di clipping sinistro
XS[1], YS[1], ZS[1]←vecchio estremo
OLD-POINT-TEST[1]←condizione di test relativa al vecchio estremo
Variabili locali NEW-POINT-TEST←condizione di test relativa al nuovo punto

```
begin
  NEW-POINT-TEST←XLM*Z+WXL;
  if (X ≥ NEW-POINT-TEST and XS[1]<OLD-POINT-TEST[1])
     or (X ≤ NEW-POINT-TEST and XS[1]>OLD-POINT-TEST[1])
  then begin
     calcolo del punto di intersezione
     Z-CLIP←CLIPPED-Z(X, Z, XS[1], ZS[1], XLM, WXL);
     X-CLIP←XLM*Z-CLIP+WXL;
     Y-CLIP←CLIPPED-X-OR-Y(X, Y, Z, XS[1], YS[1],
        ZS[1], X-CLIP, Z-CLIP);
     if XS[1]<OLD-POINT-TEST[1] or OP>31
        caso in cui la linea è disegnata dall'esterno verso l'interno
     then CLIP-RIGHT(1, X-CLIP, Y-CLIP, Z-CLIP)
        caso in cui la linea è disegnata dall'interno verso l'esterno
     else CLIP-RIGHT(OP, X-CLIP, Y-CLIP, Z-CLIP);
     end;
     memorizza il punto che verrà considerato come estremo del segmento
        successivo
```

```
    XS[1]←X;
    YS[1]←Y;
    ZS[1]←Z;
    memorizza anche la sua condizione di verifica
    OLD-POINT-TEST[1]←NEW-POINT-TEST;
    caso in cui il punto è interno
    if X ≥ NEW-POINT-TEST then CLIP-RIGHT(OP, X, Y, Z);
    return;
end;
```

Algoritmo 9.11 CLIP-RIGHT (OP, X, Y, Z) (Algoritmo 6.7 aggiornato)
Esegue il clipping sul lato destro.

Argomenti OP, X, Y, Z←dati relativi al nuovo estremo
Variabili globali WXH←lato destro della window
XHM←pendenza del piano di clipping destro
XS[2], YS[2], ZS[2]←vecchio estremo
OLD-POINT-TEST[2]←condizione di test relativa al vecchio estremo
Variabili locali NEW-POINT-TEST←condizione di test relativa al nuovo punto
X-CLIP, Y-CLIP, Z-CLIP←coordinate del punto di intersezione del segmento con il piano di clipping

```
begin
    NEW-POINT-TEST←XHM*Z+WXH;
    if (X≤NEW-POINT-TEST and XS[2]>OLD-POINT-TEST[2])
       or (X≥NEW-POINT-TEST and XS[2]<OLD-POINT-TEST[2])
    then begin
       Z-CLIP←CLIPPED-Z(X, Z, XS[2], ZS[2], XHM, WXH);
       X-CLIP←XHM*Z-CLIP+WXH;
       Y-CLIP←CLIPPED-X-OR-Y(X, Y, Z, XS[2], YS[2], ZS[2], X-CLIP,
          Z-CLIP);
       if XS[2]>OLD-POINT-TEST[2] or OP>31
       then CLIP-BOTTOM(1, X-CLIP, Y-CLIP, Z-CLIP)
       else CLIP-BOTTOM(OP, X-CLIP, Y-CLIP, Z-CLIP);
       end;
    XS[2]←X;
    YS[2]←Y;
    ZS[2]←Z;
    OLD-POINT-TEST[2]←NEW-POINT-TEST;
    if X≤NEW-POINT-TEST then CLIP-BOTTOM(OP, X, Y, Z);
    return;
end;
```

Algoritmo 9.12 CLIP-BOTTOM (OP, X, Y, Z) (Algoritmo 6.8 aggiornato)
Esegue il clipping sul lato inferiore.

Argomenti OP, X, Y, Z←dati relativi al nuovo estremo
Variabili globali WYL←lato inferiore della window
YLM←pendenza del piano di clipping inferiore
XS[3], YS[3], ZS[3]←vecchio estremo
OLD-POINT-TEST[3]←condizione di test relativa al vecchio estremo
Variabili locali NEW-POINT-TEST←condizione di test relativa al nuovo punto
X-CLIP, Y-CLIP, Z-CLIP←coordinate del punto di intersezione del segmento con il piano di clipping

```
begin
   NEW-POINT-TEST←YLM*Z+WYL;
   if (Y ≥ NEW-POINT-TEST and YS[3]<OLD-POINT-TEST[3])
      or (Y ≤ NEW-POINT-TEST and YS[3]>OLD-POINT-TEST[3])
   then begin
      Z-CLIP←CLIPPED-Z(Y, Z, YS[3], ZS[3], YLM, WYL);
      Y-CLIP←YLM*Z-CLIP+WYL;
      X-CLIP←CLIPPED-X-OR-Y(Y, X, Z, YS[3], XS[3], ZS[3],
         Y-CLIP, Z-CLIP);
      if YS[3]<OLD-POINT-TEST[3] or OP>31
      then CLIP-TOP(1, X-CLIP, Y-CLIP, Z-CLIP)
      else CLIP-TOP(OP, X-CLIP, Y-CLIP, Z-CLIP);
      end;
   XS[3]←X;
   YS[3]←Y;
   ZS[3]←Z;
   OLD-POINT-TEST[3]←NEW-POINT-TEST;
   if Y ≥ NEW-POINT-TEST then CLIP-TOP(OP, X, Y, Z);
   return;
end;
```

Algoritmo 9.13 CLIP-TOP (OP, X, Y, Z) (Algoritmo 6.9 aggiornato)
Esegue il clipping sul lato superiore.

Argomenti OP, X, Y, Z←dati relativi al nuovo estremo
Variabili globali WYH←lato superiore della window
YHM←pendenza del piano di clipping superiore
XS[4], YS[4], ZS[4]←vecchio estremo
OLD-POINT-TEST[4]←condizione di test relativa al vecchio estremo
Variabili locali NEW-POINT-TEST←condizione di test relativa al nuovo punto
X-CLIP, Y-CLIP, Z-CLIP←coordinate del punto di intersezione del segmento con il piano di clipping

```
begin
   NEW-POINT-TEST←YHM*Z+WYH;
```

```
    if (Y ≤ NEW-POINT-TEST and YS[4] > OLD-POINT-TEST[4])
       or (Y ≥ NEW-POINT-TEST and YS[4] < OLD-POINT-TEST[4])
    then begin
       Z-CLIP ← CLIPPED-Z(Y, Z, YS[4], ZS[4], YHM, WYH);
       Y-CLIP ← YHM * Z-CLIP + WYH;
       X-CLIP ← CLIPPED-X-OR-Y(Y, X, Z, YS[4], XS[4], ZS[4],
          Y-CLIP, Z-CLIP);
       if YS[4] > OLD-POINT-TEST[4] or OP > 31
       then CLIP-BACK(1, X-CLIP, Y-CLIP, Z-CLIP)
       else CLIP-BACK(OP, X-CLIP, Y-CLIP, Z-CLIP);
       end;
    XS[4] ← X;
    YS[4] ← Y;
    ZS[4] ← Z;
    OLD-POINT-TEST[4] ← NEW POINT-TEST;
    if Y ≤ NEW-POINT-TEST then CLIP-BACK(OP, X, Y, Z);
    return;
end;
```

Dal momento che il piano anteriore e quello posteriore sono perpendicolari all'asse *z*, i relativi algoritmi di clipping sono più semplici dei precedenti e controllano anche il flag usato per evitare il clipping ad essi relativo.

Algoritmo 9.14 CLIP-BACK (OP, X, Y, Z)
Esegue il clipping rispetto al piano posteriore.

Argomenti	OP, X, Y, Z ← dati relativi al nuovo estremo
Variabili globali	BACK-Z ← posizione del piano di clipping posteriore
	BACK-FLAG ← indicatore della richiesta di clipping
	XS[5], YS[5], ZS[5] ← vecchi estremi del segmento
Variabili locali	X-CLIP, Y-CLIP ← coordinate del punto di clipping
	Z-CHANGE ← fattore di correzione relativo alla coordinata *z*

```
begin
   if BACK-FLAG
   then begin
        if (Z ≥ BACK-Z and ZS[5] < BACK-Z)
           or (Z ≤ BACK-Z and ZS[5] > BACK-Z)
        then begin
           Z-CHANGE ← (BACK-Z - Z)/(Z - ZS[5]);
           Z-CLIP ← (X - XS[5]) * Z-CHANGE + X;
           Y-CLIP ← (Y - YS[5]) * Z-CHANGE + Y;
           if ZS[5] < BACK-Z or OP > 31
           then CLIP-FRONT(1, X-CLIP, Y-CLIP, BACK-Z)
           else CLIP-FRONT(OP, X-CLIP, Y-CLIP, BACK-Z)
           end;
```

```
            XS[5]←X;
            YS[5]←Y;
            ZS[5]←Z;
            if Z ≥ BACK-Z then CLIP-FRONT(OP, X, Y, Z);
            end
      else CLIP-FRONT(OP, X, Y, Z);
      return;
end;
```

Algoritmo 9.15 CLIP-FRONT (OP, X, Y, Z)
Esegue il clipping rispetto al piano anteriore.

Argomenti	OP, X, Y, Z←dati relativi al nuovo estremo
Variabili globali	FRONT-Z←posizione del piano di clipping anteriore
	FRONT-FLAG←indicatore della richiesta di clipping
	XS[6], YS[6], ZS[6]←vecchi estremi del segmento
Variabili locali	X-CLIP, Y-CLIP←coordinate del punto di clipping
	Z-CHANGE←fattore di correzione relativo alla coordinata z

```
begin
   if FRONT-FLAG
   then begin
         if (Z ≤ FRONT-Z and ZS[6] > FRONT-Z)
            or (Z ≥ FRONT-Z and ZS[6] < FRONT-Z)
         then begin
            Z-CHANGE←(FRONT-Z − Z)/(Z − ZS[6]);
            X-CLIP←(X − XS[6]) * Z-CHANGE + X;
            Y-CLIP←(Y − YS[6]) * Z-CHANGE + Y;
            if ZS[6] > FRONT-Z or OP > 31
            then SAVE-CLIPPED-POINT(1, X-CLIP, Y-CLIP,
               FRONT-Z)
            else SAVE-CLIPPED-POINT(OP, X-CLIP, Y-CLIP,
               FRONT-Z)
            end;
         XS[6]←X;
         YS[6]←Y;
         ZS[6]←Z;
         if Z ≤ FRONT-Z then SAVE-CLIPPED-POINT(OP, X, Y, Z);
         end
   else SAVE-CLIPPED-POINT(OP, X, Y, Z);
   return;
end;
```

La routine SAVE-CLIPPED-POINT memorizza le istruzioni di un poligono in un buffer temporaneo e mette altre istruzioni nel display file (come è già stato

fatto nel Capitolo 6); ora deve solo essere aggiunto un valore di proiezione, per inserire le istruzioni tridimensionali nel display file bidimensionale.

Algoritmo 9.16 SAVE-CLIPPED-POINT (OP, X, Y, Z)
(Algoritmo 6.10 aggiornato)
Memorizza le istruzioni di clipping.
Argomenti OP, X, Y, Z←istruzioni di disegno tridimensionali
Variabili globali PFLAG←indica che si sta trattando un poligono
COUNT-OUT←contatore dei lati del poligono su cui eseguire il clipping

```
begin
   if PFLAG
   then begin
          COUNT-OUT←COUNT-OUT+1;
          if COUNT-OUT<33
          then PUT-IN-T(OP, X, Y, Z, COUNT-OUT);
          end
   else DOPROJECTION(OP, X, Y, Z);
   return;
end;
```

La memorizzazione temporanea dei poligoni è stata estesa per includere la coordinata z.

Algoritmo 9.17 PUT-IN-T (OP, X, Y, Z, INDEX)
(Algoritmo 6.11 aggiornato)
Memorizza i lati del poligono nel buffer temporaneo.
Argomenti OP, X, Y, Z←istruzione da memorizzare
INDEX←posizione in cui memorizzarla
Variabili globali IT, XT, YT, ZT←array del buffer temporaneo

```
begin
   IT[INDEX]←OP;
   XT[INDEX]←X;
   YT[INDEX]←Y;
   ZT[INDEX]←Z;
   return;
end;
```

9.7 MODIFICHE ULTERIORI

Nel Capitolo 6, la routine SAVE-CLIPPED-POINT chiamata dalla VIEWING-TRANSFORM effettua la trasformazione da window a viewport e memorizza il

risultato nel display file. Il sistema nella versione tridimensionale chiama invece la routine DOPROJECTION.
La routine DOPROJECTION proietta dapprima l'oggetto sul piano di visualizzazione e chiama poi la routine VIEWING-TRANSFORM per trasformare e memorizzare il risultato.

Algoritmo 9.18 DOPROJECTION (OP, X, Y, Z)
Proietta e memorizza i comandi di disegno.
Argomenti OP, X, Y, Z←istruzioni da memorizzare
Variabili globali PERSPECTIVE-FLAG←indicatore del tipo di proiezione
Variabili locali PX, PY, PZ←coordinate proiettate

```
begin
   PX←X;
   PY←Y;
   PZ←Z;
   if PERSPECTIVE-FLAG
   then PERSPECTIVE-TRANSFORM(PX, PY, PZ)
   else PARALLEL-TRANSFORM(PX, PY, PZ);
   VIEWING-TRANSFORM(OP, PX, PY);
   return;
end;
```

Va modificata anche la routine POLYGON-CLIP, in modo tale che, quando estrae i comandi per i lati dei poligoni dal buffer temporaneo e li mette nel display file, lo faccia attraverso la routine DOPROJECTION. Anche in questo caso le istruzioni tridimensionali devono essere trasformate in bidimensionali per poter accedere al display file.

Algoritmo 9.19 POLYGON-CLIP (OP, X, Y, Z)
(Algoritmo 6.12 aggiornato)
Argomenti OP, X, Y, Z←istruzione del display file
Variabili globali PFLAG←indica che si sta trattando un poligono
COUNT-IN←numero di lati che restano da considerare
COUNT-OUT←numero di lati inseriti nel display file
IT, XT, YT, ZT←array temporaneo di 32 elementi per la memorizzazione del poligono

```
begin
   COUNT-IN←COUNT-IN−1;
   CLIP-LEFT(OP, X, Y, Z);
   if COUNT-IN≠0 then return;
   chiude il poligono su cui è stato eseguito il clipping
   if COUNT-OUT>0 then CLIP-LEFT(2, XT[1], YT[1], ZT[1]);
   PFLAG←FALSE;
   COUNT-OUT←COUNT-OUT−1;
   if COUNT-OUT<3 then return;
```

```
    if COUNT-OUT<32
       then begin
       DOPROJECTION(COUNT-OUT, XT[COUNT-OUT],
          YT[COUNT-OUT], ZT[COUNT-OUT]);
       for I = 1 to COUNT-OUT
          do DOPROJECTION(IT[I], XT[I], YT[I], ZT[I]);
       end
    else return-error 'ERRORE NEI LATI DEL POLIGONO';
    return;
end;
```

Un'altra routine che deve essere modificata è la routine CLIP. Le chiamate alle routine CLIP-LEFT e POLYGON-CLIP devono ora includere anche la coordinata z, come pure la routine CLIP stessa. Ci sono ora tre coordinate per i vecchi estremi dei segmenti, che devono essere inizializzate per sei routine e non più per quattro. Le condizioni da verificare per i vecchi estremi devono anch'esse venir inizializzate in corrispondenza dei vecchi vertici del poligono. Queste operazioni sono effettuate dalla routine MAKE-CLIPPING-TESTS.

Algoritmo 9.20 **CLIP (OP, X, Y, Z)** (Algoritmo 6.13 aggiornato)

Argomenti OP, X, Y, Z←istruzioni che hanno subito un clipping

Variabili globali PFLAG←indica che si sta trattando un poligono
COUNT-IN←lati del poligono che devono essere inseriti
COUNT-OUT←lati memorizzati del poligono che hanno subito un clipping
XS, YS, ZS←array per memorizzare l'ultimo punto disegnato

```
begin
   if PFLAG then POLYGON-CLIP(OP, X, Y, Z)
   else if OP>2 and OP<31
   then begin
         PFLAG←TRUE;
         COUNT-IN←OP;
         COUNT-OUT←0;
         for I = 1 to 6
            begin
               XS[I]←X;
               YS[I]←Y;
               ZS[I]←Z;
            end;
         MAKE-CLIPPING-TESTS;
         end
   else CLIP-LEFT(OP, X, Y, Z);
   return;
end;
```

Per finire, viene riportata una routine di inizializzazione per i parametri relativi ai piani anteriore e posteriore e per le costanti della window, nonché per le posizioni degli estremi del «vecchio» segmento.

Algoritmo 9.21 INITIALIZE-9
Variabili globali XS, YS, ZS←array per memorizzare i vecchi estremi di un segmento
Variabili locali I←indice del ciclo

```
begin
   SET-VIEW-DEPTH(0, 0);
   INITIALIZE-8;
   SET-FRONT-PLANE-CLIPPING(FALSE);
   SET-BACK-PLANE-CLIPPING(FALSE);
   for I = 1 to 6
   do begin
         XS[I]←0;
         YS[I]←0;
         ZS[I]←0;
         end;
   return;
end;
```

APPLICAZIONE PRATICA

Nel capitolo precedente si è vista l'applicazione delle operazioni di visualizzazione degli oggetti 3-D in un programma per la simulazione del volo.
Anche in quel caso è comunque necessaria un'estensione per eliminare tutte le parti di paesaggio che si trovano dietro all'aeroplano e comunque dietro al parabrezza della cabina di pilotaggio (piano di proiezione). Con il solo piano di clipping anteriore è possibile simulare una visibilità frontale illimitata. Se, invece, si vuole simulare una visibilità limitata, dovuta per esempio a cattive condizioni atmosferiche, si possono eliminare tutti gli oggetti che si trovano al di là del limite di visibilità.
Le istruzioni per attivare questi due tipi di clipping sono:

```
SET-VIEW-DEPTH(0, VISIBILITY-DISTANCE);
SET-FRONT-PLANE-CLIPPING(TRUE);
SET-BACK-PLANE-CLIPPING(BAD-WEATHER);
```

in cui VISIBILITY-DISTANCE è la distanza massima percepibile dall'occhio del pilota e BAD-WEATHER è posta a TRUE in caso di visibilità ridotta.

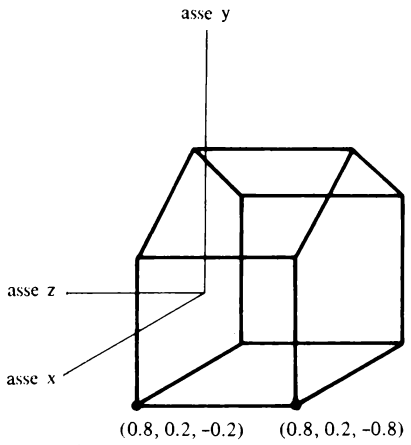
ESERCIZI

9.1 Scrivere l'equazione dei piani di clipping in base ai seguenti dati specificati dall'utente:

a) SET-WINDOW(0, 1, 0, 1)
SET-PARALLEL(1, 0, 2)
b) SET-WINDOW(−0.5, 0.5, 0.5, 1)
SET-PARALLEL(2, −2, 1)
c) SET-WINDOW(0, 1, 0, 1)
SET-PERSPECTIVE(0.5, 0.5, 1)
d) SET-WINDOW(−0.5, 0.5, 0.5, 1)
SET-PERSPECTIVE(1, 0.5, 2)

9.2 Calcolare il punto di intersezione fra la retta passante per i punti (0, 1, −1) e (2, 5, −3) e il piano $x+3y-2z-11 = 0$.

9.3 Si consideri la figura



Assumendo tutti i parametri di visualizzazione di default, tranne
SET-PERSPECTIVE(0.5, 0.5, 1)
SET-DEPTH-CLIPPING-PLANES(0.3, 0.7)
tracciare che cosa verrebbe visualizzato per

a) SET-FRONT-CLIPPING(TRUE)
SET-BACK-CLIPPING(FALSE)
b) SET-FRONT-CLIPPING(FALSE)
SET-BACK-CLIPPING(TRUE)
c) SET-FRONT-CLIPPING(TRUE)
SET-BACK-CLIPPING(TRUE)
d) SET-VIEW-REFERENCE-POINT(0, 0, −0.2)
SET-FRONT-CLIPPING(TRUE)
SET-BACK-CLIPPING(TRUE)

PROBLEMI DI PROGRAMMAZIONE

9.1 Implementare gli algoritmi dal 9.1 al 9.21 compresi.

9.2 Collaudare gli algoritmi di clipping tridimensionale costruendo il modello di un oggetto 3-D, per esempio una casa, e generare le viste che corrispondono al clipping rispetto a ciascuno dei piani del parallelepipedo o della piramide tronca che definisce la regione visibile, nel caso di proiezione parallela e di proiezione prospettica.

9.3 Scrivere le routine

INQUIRE-DEPTH-CLIPPING-PLANES(FRONT, BACK)
INQUIRE-FRONT-CLIPPING(ON-OFF)
INQUIRE-BACK-CLIPPING(ON-OFF)

che fornisce all'utente le ultime assegnazioni dei valori dei parametri di clipping relativi alla profondità.

9.4 Si visualizzi un qualsiasi oggetto solido in proiezione prospettica; si attivi poi il clipping anteriore, muovendo, in viste successive, il piano di clipping attraverso l'oggetto cambiando il punto di riferimento della vista e il centro di proiezione.

***9.5** Modificare il programma di visualizzazione interattiva del problema di programmazione 8.7, includendo l'assegnamento da parte dell'utente dei valori dei parametri di clipping per i piani anteriore e posteriore della regione visibile.

10

Superfici e linee nascoste

INTRODUZIONE

All'inizio del capitolo precedente è stato descritta la simulazione di un percorso all'interno di un palazzo. Nella simulazione venivano mostrate di volta in volta solo le parti del palazzo che potevano essere osservate, a seconda del punto di vista. Con la vista frontale era visibile la facciata del palazzo, mentre il retro della struttura rimaneva nascosto, ma con una vista da dietro la situazione sarebbe risultata opposta. Non si era in grado di vedere il contenuto del palazzo dall'esterno poiché esso rimaneva nascosto dai muri del palazzo stesso; occorreva entrare nel palazzo, cioè porre il punto di vista al suo interno per visualizzare quanto esso conteneva. D'altra parte, nei capitoli precedenti si è imparato a disegnare e progettare oggetti tridimensionali visualizzando sempre ogni loro parte, come se fossero trasparenti: figure di questo tipo vengono chiamate disegni *wire-frame*, cioè ricostruzioni reticolari di oggetti supposti solidi, che trasformano oggetti complessi in un confuso intreccio di segmenti. In queste rappresentazioni grafiche può essere difficile giudicare quali linee appartengano alla parte anteriore dell'oggetto e quali alla parte posteriore. Bisogna quindi considerare attentamente il problema dell'eliminazione di quelle linee che normalmente sarebbero nascoste da altre parti dell'oggetto. Affidando tale compito alla macchina, l'utente sarà libero di costruire l'intero modello (parte anteriore, parte posteriore, interno ed esterno) e di vederlo poi come appare nella realtà. Questo problema non è così banale come potrebbe sembrare; è anzi il problema più complesso, in termini di programmazione, rispetto a tutti gli altri argomenti presentati in questo testo. Ciò che la natura fa con facilità (con una grande quantità di elaborazioni parallele), il calcolatore deve realizzare attraverso calcoli estremamente complessi. Le soluzioni proposte al problema dell'eliminazione delle linee e superfici nascoste sono molte, ma probabilmente la soluzione ottimale deve ancora venir trovata. In questo testo verrà presentata una sola soluzione, che non è la più elegante né la più efficiente, ma è abbastanza im-

mediata. Il capitolo è diviso in quattro parti, in cui si cerca di risolvere il problema a passi successivi, partendo dalle situazioni più semplici sino ad arrivare al caso più generale. La prima parte tratta l'eliminazione delle facce nascoste in oggetti convessi per mezzo dell'algoritmo BACK-FACE. Tale algoritmo verrà usato nelle tecniche di ombreggiatura delle superfici del Capitolo 11 e dovrà quindi essere implementato, mentre le routine degli altri paragrafi possono non venir considerate. La seconda parte tratta l'eliminazione delle superfici nascoste di tipo convesso per mezzo dell'algoritmo del «pittore». Nella terza parte il sistema viene esteso all'eliminazione delle linee non visibili. La quarta parte considera superfici nascoste di tipo concavo.

PARTE 1 — SUPERFICI NASCOSTE CONVESSE: ALGORITMO BACK-FACE

10.1 FACCE NASCOSTE

L'eliminazione delle superfici nascoste è, in generale, un'operazione costosa ed è quindi conveniente applicare dei test che semplifichino il problema al massimo prima di affrontare l'analisi vera e propria. Esiste una semplice verifica che elimina la maggior parte delle superfici nascoste di un oggetto, identificando quelle che non si presentano alla vista dell'osservatore, cioè quelle superfici che costituiscono la parte posteriore dell'oggetto, che non possono essere visibili poiché tra esse e l'osservatore è interposto il nucleo dell'oggetto stesso. Questo non risolve completamente il problema delle superfici nascoste, poiché potrebbe ancora accadere che la faccia frontale di un oggetto sia nascosta da un secondo oggetto o da un'altra parte dell'oggetto stesso; in ogni caso, il test è in grado di scartare quasi la metà delle superfici, semplificando notevolmente il problema: gli argomenti restanti verranno discussi nella seconda parte di questo capitolo. Per il momento si considereranno solo poligoni convessi: le linee non possono nascondere niente, ma possono essere nascoste; ne consegue che in genere esse vengono considerate solo come lati delle superfici di un oggetto; per questo motivo è sufficiente, per la maggior parte dei disegni, considerare — nella risoluzione del problema delle superfici nascoste — solamente dei poligoni. In un poligono, considerato nello spazio tridimensionale, si possono distinguere due superfici, una anteriore e una posteriore, colorandone, ad esempio, una con colore chiaro e l'altra con colore scuro, come se il poligono fosse un foglio di carta. Se si rappresenta poi il poligono attraverso i suoi vertici, le cui coordinate vengono memorizzate in opportuni array, le due facce potranno essere distinte definendo in senso orario il verso di percorrenza del contorno del poligono dalla parte della superficie chiara, e — di conseguenza — in senso antiorario il verso di percorrenza del contorno del poligono dalla parte della superficie scura (vedi Figura 10.1).



Figura 10.1 La superficie di un poligono sarà chiara o scura a seconda che sia anteriore o posteriore e, quindi, a seconda del suo verso di percorrenza

Il verso di percorrenza del contorno si stabilisce conoscendo il vettore che rappresenta la superficie considerata, dato dalla normale al piano su cui giace la faccia stessa. Per calcolarlo, occorre considerare un'operazione detta *prodotto vettoriale*. Il prodotto vettoriale di due vettori è un terzo vettore, con lunghezza uguale al prodotto delle lunghezze dei due vettori per il seno dell'angolo tra loro compreso e con direzione perpendicolare al piano contenente i due vettori. Per il verso di questo vettore, valgono gli stessi ragionamenti fatti a proposito del verso dell'asse z nel paragrafo 8.1: in questo capitolo si farà riferimento ad un sistema di coordinate destrorso (vedi Figura 10.2).
Le formule per un prodotto vettoriale tridimensionale in tale sistema, per

$$[R_x, R_y, R_z] = [P_x, P_y, P_z] \times [Q_x, Q_y, Q_z] \tag{10.1}$$

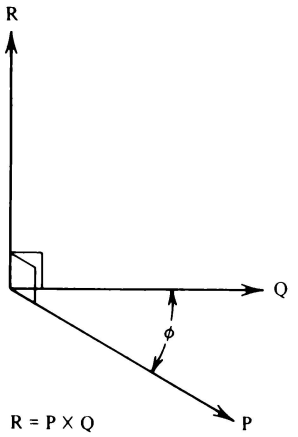


Figura 10.2 Prodotto vettoriale

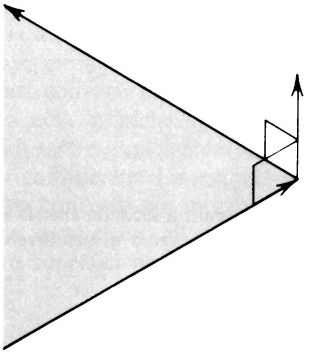


Figura 10.3 Su una faccia scura, il prodotto vettoriale di due lati in un vertice convesso è diretto verso l'esterno (secondo la regola della terna destrorsa)

sono:

$$\begin{aligned} R_x &= P_y Q_z - P_z Q_y \\ R_y &= P_z Q_x - P_x Q_z \\ R_z &= P_x Q_y - P_y Q_x \end{aligned} \tag{10.2}$$

Due lati di un poligono rappresentano due vettori nel piano del poligono stesso, cosicché il prodotto vettoriale dei due lati del poligono è un vettore perpendicolare alla faccia del poligono. Tale vettore punterà fuori dalla faccia scura (con verso di percorrenza del perimetro antiorario) o dalla faccia chiara (con verso di percorrenza orario)? Ciò dipende dal fatto che i due lati formino un angolo convesso o concavo.

Si considerino due lati adiacenti che non giacciono sulla stessa retta e che si incontrano in un vertice convesso (cioè un vertice in cui i due lati formano un angolo convesso); in questo caso, il prodotto vettoriale sarà diretto verso l'esterno della faccia scura (cioè con verso di percorrenza antiorario, vedi Figura 10.3). Se si suppone che le superfici esterne di un solido siano quelle chiare e che le interne siano quelle scure, quando si osserva la faccia di un oggetto dall'esterno, il verso di percorrenza del suo contorno sarà quello orario (vedi Figura 10.4). Se un poligono è visibile, la superficie chiara si troverà di fronte all'osservatore, mentre la scura sarà invisibile; in questo caso, calcolando il prodotto vettoriale che fornisce la direzione della faccia scura, si troverà che questo vettore punta verso l'osservatore. In conclusione, se il prodotto vettoriale di due lati è diretto verso l'osservatore, la faccia è una faccia posteriore o nascosta (*back face*) e va quindi eliminata; se invece esso punta nella direzione opposta, la faccia è anteriore e visibile (*front face*). È possibile capire se il vettore normale alla faccia punta o meno nella direzione dell'osservatore confrontandolo con quello della direzione di proiezione (dopo la trasformazione nelle

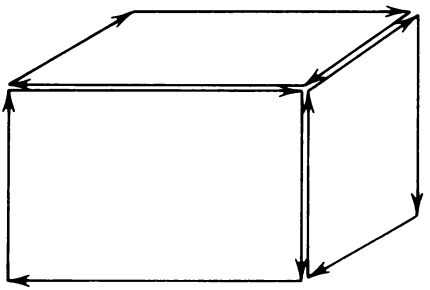


Figura 10.4 Tutte le facce esterne sono chiare (percorse in senso orario)

coordinate del piano di proiezione). Per quanto riguarda la proiezione prospettica, il vettore può essere definito dal punto di proiezione e da un punto sul poligono, mentre occorre stare attenti a definire la proiezione parallela in modo che il vettore sia diretto dall'oggetto verso l'osservatore. Si vedrà in seguito che la normale al piano di proiezione deve essere diretta dall'osservatore verso l'oggetto, cioè in senso opposto a quello delle proiezioni.
Per confrontare le direzioni di due vettori (R ed S, per esempio) occorre considerarne il prodotto scalare.

$$a = R \cdot S = |R||S|\cos\theta \tag{10.3}$$

Come si è visto nel Capitolo 8, il prodotto scalare è dato dal prodotto delle lunghezze dei due vettori per il coseno dell'angolo fra essi compreso. Il fattore coseno è molto importante, poiché se l'angolo compreso tra i due vettori è acuto ($0 \leq \theta < \pi/2$), il coseno e quindi il prodotto scalare sono positivi; se l'angolo è ottuso ($\pi/2 < \theta \leq \pi$), il coseno e il prodotto scalare sono entrambi negativi (vedi Figura 10.5).
Per stabilire se un dato poligono è una faccia anteriore o posteriore, si usa quindi il prodotto vettoriale di due lati — per calcolare il vettore normale alla superficie — e in seguito si calcola il prodotto scalare di questo con il vettore proiezione. Se si ottiene un valore negativo, significa che i due vettori hanno

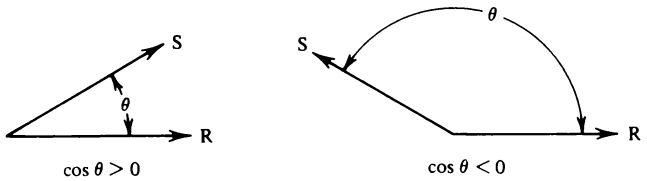


Figura 10.5 Il coseno dell'angolo è positivo se l'angolo compreso fra i due vettori è acuto ($0 \leq \theta < \pi/2$) ed è negativo se l'angolo è ottuso ($\pi/2 < \theta \leq \pi$)

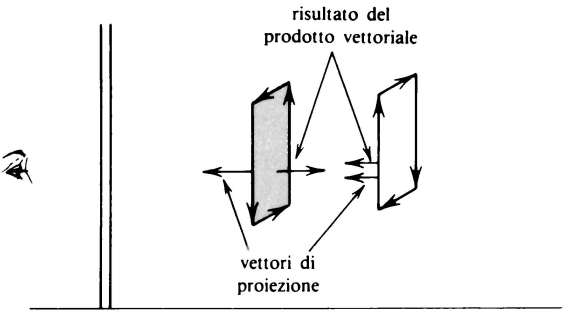


Figura 10.6 Il piano di sinistra risulta visibile da sinistra perché il prodotto scalare fra il vettore di proiezione e il vettore risultante dal prodotto vettoriale di due lati è negativo. Il piano di destra invece risulta una faccia nascosta.

verso opposto: la faccia scura (interna) non è rivolta in direzione dell'osservatore e la faccia chiara può essere vista; il poligono è quindi una faccia anteriore. Un valore positivo invece indica che la faccia non è visibile. La formula per calcolare il prodotto scalare a partire dalle componenti dei vettori è la seguente

$$a = R \cdot S = [R_x, R_y, R_z] \cdot [S_x, S_y, S_z] = R_x S_x + R_y S_y + R_z S_z \qquad (10.4)$$

Il primo test per stabilire se una faccia è nascosta è quindi un controllo sul segno del prodotto scalare a (vedi Figura 10.6).
Bisogna stare attenti a scegliere i vertici del poligono per i vettori P e Q. Questi vettori devono incontrarsi in un vertice convesso. Com'è possibile trovare un vertice convesso? Si sceglie un estremo qualsiasi ed il vertice che lo precede e si calcola la differenza vettoriale fra i due punti, che definisce uno dei due vettori. Si considera quindi il vertice successivo, calcolando nuovamente la differenza con un estremo dell'altro vettore. Occorre però inserire un controllo per essere sicuri di non avere selezionato tre punti allineati; se il terzo punto è allineato con i primi due occorre prendere in considerazione un altro vertice del poligono e utilizzarlo come terzo punto, quindi riprovare con il test. Se la verifica dà esito negativo per tutti i vertici del poligono, si deve concludere che si è in presenza di una retta, ovvero di un poligono degenere, da scartare proprio come se fosse una faccia nascosta.
Dove devono essere fatti tutti questi controlli? Certamente dopo le trasformazioni che portano in coordinate del piano di proiezione e dopo che sia stata fatta l'operazione di clipping per evitare di prendere in considerazione poligoni esterni alla window. C'è un punto nella routine POLYGON-CLIP in cui il poligono su cui è stato eseguito il clipping risiede in una locazione di memoria temporanea in attesa di essere inserito nel display file. Occorre controllare che il numero dei lati sia compreso tra 3 e 31 prima di memorizzare il poligono.

10.2 ALGORITMI DI TIPO BACK-FACE

Vengono ora esaminati nei dettagli gli algoritmi di eliminazione delle facce nascoste. Prima di iniziare, occorre fornire all'utente un mezzo per attivare o disattivare il procedimento a suo piacimento.

Algoritmo 10.1 SET-HIDDEN-LINE-REMOVAL (ON-OFF)
Routine utente che attiva o disattiva il procedimento di eliminazione delle linee e delle facce nascoste.
Argomenti ON-OFF←specifiche dell'utente per l'eliminazione delle linee e delle facce nascoste
Variabili globali HIDDEN←indicatore per l'eliminazione delle linee e delle facce nascoste

```
begin
   HIDDEN←ON-OFF;
   return;
end;
```

La routine che segue è la POLYGON-CLIP, che include ora un controllo sulle facce nascoste quando il flag HIDDEN è posto al valore TRUE.

Algoritmo 10.2 POLYGON-CLIP (OP, X, Y, Z)
(Algoritmo 9.19 aggiornato)
Argomenti OP, X, Y, Z←istruzione del display file
Variabili globali PFLAG←indica che si sta trattando un poligono
COUNT-IN←numero di lati che devono ancora essere elaborati
COUNT-OUT←numero di lati da introdurre nel display file
IT, XT, YT, ZT←array temporanei per memorizzare il poligono
HIDDEN←indicatore per l'eliminazione delle linee nascoste

```
begin
   COUNT-IN←COUNT-IN−1;
   CLIP-LEFT(OP, X, Y, Z);
   if COUNT-IN≠0 then return;
   chiude il poligono su cui è stato eseguito il clipping
   if COUNT-OUT>0 then CLIP-LEFT(2, XT[1], YT[1], ZT[1]);
   PFLAG←FALSE;
   COUNT-OUT←COUNT-OUT−1;
   if COUNT-OUT<3 then return;
   if COUNT-OUT<32
      then if HIDDEN and IS-BACK-FACE then return;
      else begin
```

```
                DOPROJECTION(COUNT-OUT, XT[COUNT-OUT],
                YT[COUNT-OUT], ZT[COUNT-OUT]);
                for I = 1 to COUNT-OUT
                do DOPROJECTION(IT[I], XT[I], YT[I], ZT[I]);
                end
        else return-error 'ERRORE NEI LATI DEL POLIGONO';
    return;
end;
```

L'algoritmo sopra riportato usa la funzione IS-BACK-FACE per decidere se il poligono è una faccia nascosta. Questa funzione assume il valore TRUE se il poligono è una faccia nascosta (o una linea) e FALSE in caso contrario. L'algoritmo usa anche la funzione LEFT-MOST per determinare la posizione del vertice più a sinistra all'interno del buffer temporaneo.

Algoritmo 10.3 IS-BACK-FACE

Funzione che restituisce il valore TRUE se il poligono posto nel T-buffer è una faccia nascosta.

Variabili globali XT, YT, ZT←array che costituiscono il T-buffer contenente i vertici del poligono
PERSPECTIVE-FLAG←indicatore del tipo di proiezione
COUNT-OUT←numero dei vertici del poligono
VXP, VYP, VZP←vettori di proiezione parallela
XC, YC, ZC←centro della proiezione prospettica

Variabili locali L←indice del vertice posto più a sinistra
LA, LB←indici dei vertici che vengono dopo e prima il vertice posto più a sinistra
SX, SY, SZ←vettore che punta in direzione dell'osservatore
XV1, YV1, ZV1←componenti del primo vettore
XV2, YV2, ZV2←componenti del secondo vettore
RX, RY, RZ←componenti del prodotto vettoriale
DIR←prodotto scalare tra la normale alla superficie e il vettore proiezione

Costanti ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
  IS-BACK-FACE←TRUE;
  trova il vertice più a sinistra
  L←LEFT-MOST(1, COUNT-OUT);
  trova la direzione del vettore di proiezione
  if not PERSPECTIVE-FLAG
       then begin
          SX←VXP;
          SY←VYP;
```

```
            SZ←VZP;
            end
        else begin
            SX←XC−XT[L];
            SY←YC−YT[L];
            SZ←ZC−ZT[L];
            end;
    XV1←0;
    YV1←0;
    ZV1←0;
    trova il vertice che precede quello posto più a sinistra
    LB←L;
    while|XV1|+|YV1|+|ZV1|<ROUNDOFF
    do begin
        riduzione del contatore del ciclo
        if LB = 1 then LB←COUNT-OUT else LB←LB−1;
        termina il ciclo se ha esaminato tutti i vertici
        if LB = L then return;
        calcola il vettore
        XV1←XT[L]−XT[LB];
        YV1←YT[L]−YT[LB];
        ZV1←ZT[L]−ZT[LB];
        end;
    LA←L;
    DIR←0;
    trova il vertice successivo a quello posto più a sinistra e tale che il pro-
    dotto vettoriale non sia nullo
    while|DIR|<ROUNDOFF
    do begin
        if LA = COUNT-OUT then LA←1 else LA←LA+1;
        if LA = LB then return;
        XV2←XT[LA]−XT[L];
        YV2←YT[LA]−YT[L];
        ZV2←ZT[LA]−ZT[L];
        trova il vettore normale tramite il prodotto scalare
        RX←YV1*ZV2−ZV1*YV2;
        RY←ZV1*XV2−XV1*ZV2;
        RZ←XV1*YV2−YV1*XV2;
        trova la direzione della superficie tramite il prodotto scalare
        DIR←RX*SX+RY*SY+RZ*SZ;
    end;
    controlla se la faccia è nascosta
    IS-BACK-FACE←(DIR>0);
    return;
end;
```

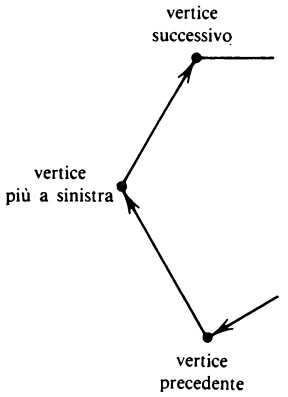


Figura 10.7 Vertici più a sinistra di un poligono

La funzione LEFT-MOST scandisce tutti i vertici del poligono memorizzando la locazione del vertice con valore della coordinata x minimo (vedi Figura 10.7).

Algoritmo 10.4 LEFT-MOST (M, N)
Stabilisce quale vertice del poligono posto tra l'M-esimo e l'N-esimo si trovi più a sinistra.

Argomenti M, N←indici del primo e dell'ultimo vertice del poligono
Variabili globali XT←array del T-buffer contenente le coordinate x del vertice
Variabili locali J←indice dei vertici

```
begin
   LEFT-MOST←M;
   for J = M+1 to N
   do if XT[LEFT-MOST]>XT[J] then LEFT-MOST←J;
   return;
end;
```

Poiché si è aggiunto il nuovo indicatore globale HIDDEN, è necessario includerlo nelle inizializzazioni. Assegnandogli FALSE come valore di default per l'eliminazione delle linee e delle superfici nascoste, si potrà ottenere l'applicazione del procedimento soltanto se essa è esplicitamente richiesta dall'utente (vedi Figura 10.8).

Algoritmo 10.5 INITIALIZE-10A

```
begin
   INITIALIZE-9;
```

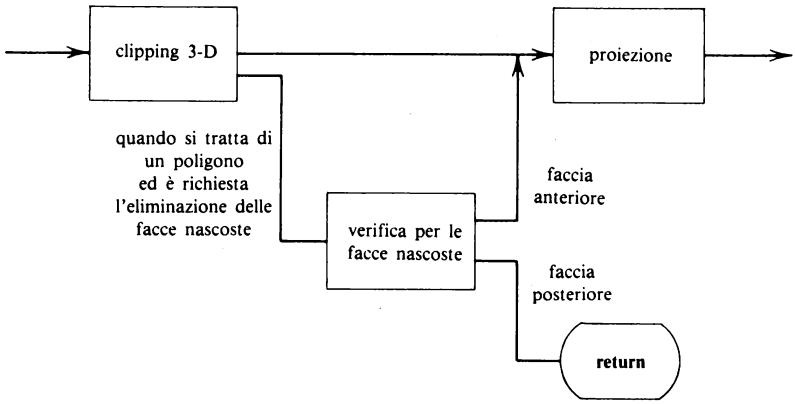


Figura 10.8 Estensione del sistema con la verifica relativa alle facce nascoste

```
    SET-HIDDEN-LINE-REMOVAL(FALSE);
    return;
end;
```

Si è visto come eliminare molte delle superfici che sono nascoste dalle parti visibili dell'oggetto. Il metodo è sufficiente per gli oggetti a singola convessità, ma può essere inadeguato quando si ha a che fare con oggetti a doppia o multipla convessità e/o con superfici concave. Il lettore che non è interessato particolarmente al problema delle superfici e delle linee nascoste, può saltare direttamente al Capitolo 11. Le routine di ombreggiatura presentate richiedono la conoscenza degli algoritmi per l'eliminazione delle facce nascoste di tipo BACK-FACE (da 10.1 fino a 10.5), ma non di quelli descritti nelle restanti parti di questo capitolo.

PARTE 2 — SUPERFICI NASCOSTE CONVESSE: ALGORITMO DEL PITTORE

10.3 L'ALGORITMO DEL PITTORE

Nonostante gli algoritmi per l'eliminazione delle facce nascoste di tipo BACK-FACE diano dei risultati abbastanza soddisfacenti, essi non risolvono completamente il problema; infatti dati due oggetti separati, una faccia in vista di un oggetto potrebbe coprire una faccia in vista del secondo oggetto, che però non verrebbe eliminata con l'algoritmo presentato (vedi Figura 10.9). Il passo successivo dello sviluppo di questo sistema è l'applicazione della tecnica conosciuta

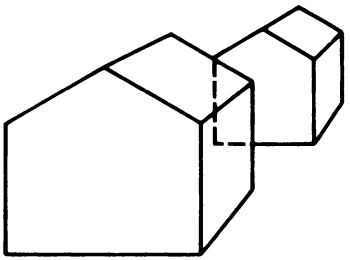


Figura 10.9 Porzioni nascoste di facce anteriori

col nome di *algoritmo del pittore*, che risolve questo problema per le superfici convesse su terminali raster. Il metodo fa uso delle proprietà del frame buffer dei terminali raster. Si devono considerare a questo scopo solo poligoni dotati di superficie. L'algoritmo prende nome dal modo in cui il pittore realizza un dipinto ad olio; l'artista inizia riempiendo l'intera tela con la scena di sfondo e dipingendovi sopra gli oggetti in primo piano, senza che ci sia bisogno di cancellare parti dello sfondo; il nuovo dipinto copre il vecchio di modo che solo il nuovo strato di pittura rimane visibile. Un frame buffer ha questa stessa proprietà. Quando si inserisce un poligono nel frame buffer, si assegna a ciascun pixel nella posizione occupata dal poligono un valore associato allo stile di riempimento dell'interno del poligono. Se in seguito viene inserito un secondo poligono «sovrapposto» al precedente, alcuni pixel del primo poligono verranno cambiati nel valore corrispondente allo stile di riempimento del secondo poligono. Dove giace il secondo poligono, i pixel assegnati per il primo poligono vengono sostituiti poiché il secondo ha «coperto» il primo. L'algoritmo del pittore suggerisce quindi di disegnare dapprima quei poligoni che sono lontani dall'osservatore (lo sfondo) e per ultimi gli oggetti che gli sono più vicini (in primo piano). Disegnando le superfici nell'ordine opportuno, si possono «coprire» di volta in volta quelle nascoste traendo vantaggio dalle proprietà dei frame buffer (vedi Figura 10.10).

L'algoritmo del pittore è un'idea semplice, ma presenta notevoli difficoltà quando si passa alla fase di implementazione. Prima di tutto non è possibile elaborare ciascun poligono indipendentemente, come si è fatto nell'algoritmo BACK-FACE, ma ogni poligono va confrontato con tutto il resto per esaminare l'ordine in cui sono disposti dinnanzi all'osservatore. In effetti, la parte centrale dell'algoritmo consiste in un ordinamento geometrico delle superfici, nozione basilare per l'eliminazione di linee e superfici nascoste. Tale procedura determinerà l'ordine nel quale i poligoni verranno introdotti nel display file, che sarà poi l'ordine nel quale verranno disegnati. Occorrerà utilizzare una certa quantità di memoria temporanea per le istruzioni relative ai poligoni, finché non si è pronti per eseguirne l'ordinamento; è necessario anche un mezzo che indichi quando sono state esaminate tutte le superfici e si può quindi procedere

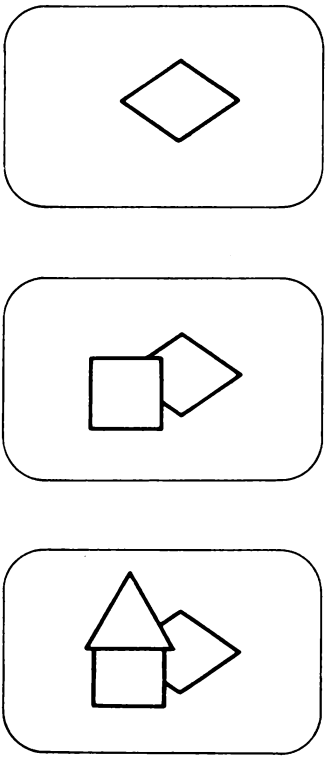


Figura 10.10 Il poligono inserito per ultimo ricopre quelli già presenti e sostituisce i valori precedenti dei pixel

con l'ordinamento. Un'altra informazione da memorizzare durante questa fase è il tipo di linea usato per disegnare ciascun poligono, poiché i comandi per il cambiamento del tipo di linea dovranno essere inseriti nello stesso ordine con il quale vengono disegnati i poligoni, che non è necessariamente lo stesso con il quale vengono introdotti dall'utente. Per eseguire l'ordinamento dei poligoni, occorrerà infine scegliere una tecnica che agevoli il loro confronto: al fine di rendere omogenee le quantità da confrontare, sarà utile scomporre i poligoni in triangoli.

Nei successivi paragrafi verranno affrontati tutti questi problemi e verranno presentate le routine per l'implementazione dell'algoritmo del pittore. La prima parte riguarderà la memorizzazione delle informazioni riguardanti il poligono nelle aree di memoria temporanee, in attesa di eseguire l'ordinamento. In seguito verrà affrontato il problema della scomposizione dei poligoni in triangoli. Si vedrà poi come confrontare due triangoli per stabilirne l'ordinamento, utiliz-

zando le tecniche di confronto presentate. Infine i poligoni verranno introdotti nel display file in ordine a partire dallo sfondo sino al primo piano.

10.4 MEMORIZZAZIONE TEMPORANEA DEI POLIGONI

Prima di applicare l'algoritmo del pittore, è possibile eseguire un test di tipo BACK-FACE, per ridurre la mole di lavoro, memorizzando temporaneamente le istruzioni di disegno relative ai poligoni, prima che questi vengano ordinati. Quest'operazione può essere eseguita dalla routine POLYGON-CLIP, la cui funzione principale è quella di realizzare un clipping dell'immagine.
Si è visto che se un poligono supera il test BACK-FACE, esso viene proiettato sul piano e l'istruzione di disegno associata viene memorizzata nel display file. Questo procedimento deve essere modificato in modo tale che, se un poligono supera il test e il flag HIDDEN ha valore TRUE, allora il tipo di linea usato per disegnare il poligono venga memorizzato e i poligoni stessi vengano proiettati e inseriti in una diversa area di memorizzazione detta C-buffer, la quale

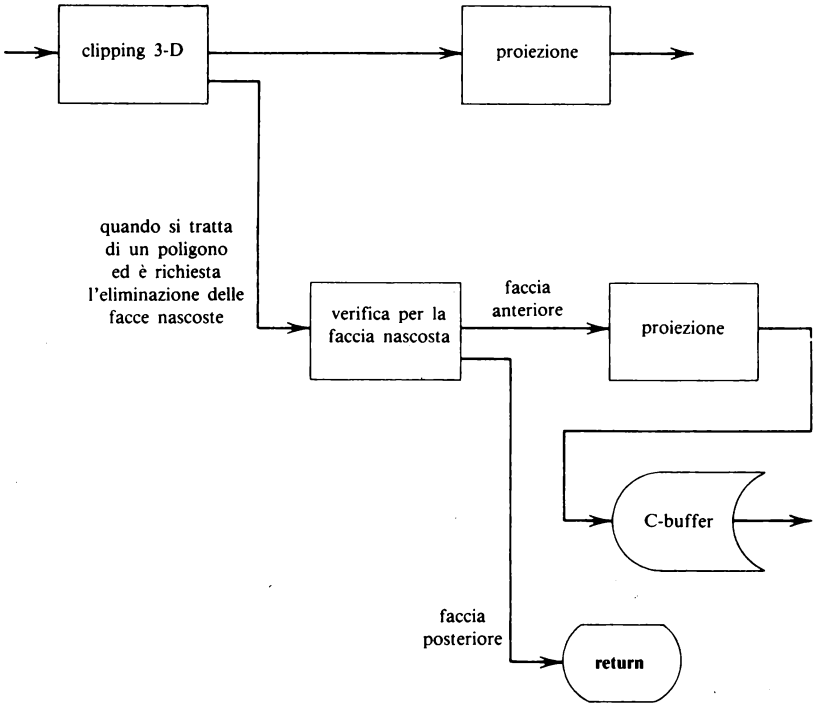


Figura 10.11 Le facce anteriori vengono raccolte una ad una fino al loro esaurimento

conterrà le istruzioni riguardanti tutti i poligoni finché non saranno pronti per l'ordinamento (vedi Figura 10.11).
La proiezione e la memorizzazione nel C-buffer viene eseguita dalla routine DOPROJECTION-TO-C.

Algoritmo 10.6 POLYGON-CLIP (OP, X, Y, Z)
(Algoritmo 10.2 aggiornato)
Argomenti OP, X, Y, Z←istruzione del display file
Variabili globali PFLAG←indica che si sta trattando un poligono
COUNT-IN←numero di lati che devono ancora essere elaborati
COUNT-OUT←numero di lati da introdurre nel display file
IT, XT, YT, ZT←array temporanei per memorizzare le informazioni relative al poligono
HIDDEN←indicatore per l'eliminazione delle linee nascoste

```
begin
  COUNT-IN←COUNT-IN−1;
  CLIP-LEFT(OP, X, Y, Z);
  if COUNT-IN≠0 then return;
  chiude il poligono su cui è stato eseguito il clipping
  if COUNT-OUT>0 then CLIP-LEFT(2, XT[1], YT[1], ZT[1]);
  PFLAG←FALSE;
  COUNT-OUT←COUNT-OUT−1;
  if COUNT-OUT<3 then return;
  if COUNT-OUT<32
    then if HIDDEN
          then if IS-BACK-FACE
                then return;
                else begin
                     SAVE-STYLE;
                     DOPROJECTION-TO-C(COUNT-OUT,
                       XT[COUNT-OUT], YT[COUNT-OUT],
                       ZT[COUNT-OUT]);
                     for I = 1 to COUNT-OUT
                     do DOPROJECTION-TO-C(IT[I], XT[I], YT[I],
                       ZT[I])
                     end
          else begin
               DOPROJECTION(COUNT-OUT,
                     XT[COUNT-OUT], YT[COUNT-OUT],
                     ZT[COUNT-OUT]);
               for I = 1 to COUNT-OUT
               do DOPROJECTION(IT[I], XT[I], YT[I], ZT[I]);
```

```
            end
        else return-error 'ERRORE NEI LATI DEL POLIGONO';
    return;
end;
```

La routine DOPROJECTION-TO-C è identica alla routine DOPROJECTION (Algoritmo 9.18), tranne che i valori proiettati vengono memorizzati nel C-buffer anziché nel display file.

Algoritmo 10.7 DOPROJECTION-TO-C (OP, X, Y, Z)
Esegue una proiezione e memorizza un comando di disegno.
Argomenti OP, X, Y, Z—istruzione da memorizzare
Variabili globali PERSPECTIVE-FLAG—indicatore di proiezione
Variabili locali PX, PY, PZ—coordinate proiettate

```
begin
    PX←X;
    PY←Y;
    PZ←Z;
    if PERSPECTIVE-FLAG
    then PERSPECTIVE-TRANSFORM(PX, PY, PZ)
    else PARALLEL-TRANSFORM(PX ,PY, PZ);
    PUT-IN-C(OP, PX, PY, Z);
    return;
end;
```

Si noti che il C-buffer contiene la coordinata z — che fornisce un'informazione sulla profondità del poligono — e le coordinate x, y del piano di proiezione. La memorizzazione nel C-buffer viene effettuata dalla routine PUT-IN-C.

Algoritmo 10.8 PUT-IN-C (OP, X, Y, Z)
Memorizza un'istruzione di disegno nel C-buffer.
Argomenti OP, X, Y, Z—istruzione da inserire nel C-buffer.
Variabili globali CFREE—indice della successiva cella libera del C-buffer
IC, XC, YC, ZC—array che costituiscono il C-buffer
Costanti C-SIZE—dimensione degli array del C-buffer, che devono essere sufficientemente grandi per contenere tutti i poligoni da ordinare

```
begin
    if CFREE>C-SIZE then return-error 'C-BUFFER PIENO';
    IC[CFREE]←OP;
    XC[CFREE]←X;
    YC[CFREE]←Y;
    ZC[CFREE]←Z;
    CFREE←CFREE+1;
    return;
end;
```

10.5 MEMORIZZAZIONE DEL TIPO DI LINEA UTILIZZATO

La routine usata per memorizzare i tipi di linea usati per il riempimento delle aree e per il disegno dei contorni dei poligoni è chiamata SAVE-STYLE. I valori correnti vengono memorizzati in array, in modo che una cella di ciascuno di questi venga usata per ogni singolo poligono.

Algoritmo 10.9 SAVE-STYLE
Memorizza il tipo di linea utilizzato per tracciare le aree e i contorni dei poligoni.

Variabili globali CURRENT-LINE-STYLE←tipo di linea corrente per il contorno
CURRENT-FILL-STYLE←tipo di linea corrente per il riempimento
EDGE-STYLE-LIST, FILL-STYLE-LIST←array per memorizzare i tipi di linea
STYLE-COUNT←indice dell'ultima cella occupata negli array dei tipi di linea
Costanti STYLE-SAVE-SIZE←dimensione degli array usati

```
begin
   if STYLE-COUNT = STYLE-SAVE-SIZE
   then return-error 'ARRAY STYLE PIENI';
   STYLE-COUNT←STYLE-COUNT + 1;
   EDGE-STYLE-LIST[STYLE-COUNT]←CURRENT-LINE-STYLE;
   FILL-STYLE-LIST[STYLE-COUNT]←CURRENT-FILL-STYLE;
   return;
end;
```

Occorre anche estendere le routine SET-LINESTYLE e SET-FILL-STYLE dei Capitoli 2 e 3, per stabilire la sequenza con cui devono essere considerati i tipi di linea per disegnare i poligoni memorizzati temporaneamente, prima dell'applicazione dell'algoritmo del pittore.

Algoritmo 10.10 SET-LINESTYLE (STYLE) (Algoritmo 2.20 aggiornato)
Routine utente che modifica il tipo di linea da usare per il contorno di un'immagine.

Argomenti STYLE←tipo di linea scelto dall'utente
Variabili globali CURRENT-LINE-STYLE←tipo di linea corrente

```
begin
   CURRENT-LINE-STYLE← -STYLE;
   DISPLAY-FILE-ENTER(CURRENT-LINE-STYLE);
   return;
end;
```

Algoritmo 10.11 SET-FILL-STYLE (STYLE) (Algoritmo 3.4 aggiornato) Routine utente che modifica il tipo di linea da usare per riempire l'area interna di un poligono.

Argomenti STYLE←tipo di linea scelto dall'utente
Variabili globali CURRENT-FILL-STYLE←tipo di linea corrente

```
begin
   CURRENT-FILL-STYLE←−(30+STYLE);
   DISPLAY-FILE-ENTER(CURRENT-FILL-STYLE);
   return;
end;
```

Infine, bisogna inizializzare le variabili introdotte per memorizzare il tipo di linea utilizzato per tracciare il contorno e l'area dei poligoni.

Algoritmo 10.12 INITIALIZE-10B

Variabili globali CURRENT-LINE-STYLE, CURRENT-FILL-STYLE←tipi di linea correnti
STYLE-COUNT←indice dell'ultima cella usata negli array che contengono i tipi di linea
CFREE←indice della successiva cella libera nel C-buffer

```
begin
   INITIALIZE-10A;
   CURRENT-LINE-STYLE←−1;
   CURRENT-FILL-STYLE←−31;
   STYLE-COUNT←0;
   CFREE←1;
   return;
end;
```

10.6 RICERCA DELLE SUPERFICI NASCOSTE

Le routine sopra riportate memorizzano i poligoni così come vengono incontrati, finché non si è pronti per ordinarli. Come si farà a sapere quando saranno stati inseriti tutti i poligoni e sarà il momento di ordinarli e memorizzarli nel display file? Nel sistema proposto, occorre considerare le linee nascoste da eliminare come singoli segmenti del display file. Gli oggetti contenuti in un segmento vanno poi ordinati, senza che venga eseguito alcun controllo per vedere se un oggetto posto in un dato segmento nasconde un oggetto posto in un altro segmento. Con una tale organizzazione, un punto ragionevole nel quale eseguire l'ordinamento e la memorizzazione dei poligoni è appena prima della chiusura del segmento di display file. Quando viene eseguita tale chiusura, tutti i poligoni principali saranno stati inseriti e quindi nessun altro poligono del segmento sarà stato dimenticato creando confusione. L'algoritmo CLOSE-SEGMENT

deve quindi comprendere una chiamata alla routine per l'eliminazione delle superfici nascoste.

Algoritmo 10.13 **CLOSE-SEGMENT** (Algoritmo 5.2 aggiornato)
Variabili globali NOW-OPEN←nome del segmento corrente
FREE←indice della successiva cella libera del display file
HIDDEN←indicatore per la rimozione delle superfici nascoste
SEGMENT-START, SEGMENT-SIZE←array contenenti la posizione iniziale e la dimensione dei segmenti del display file

```
begin
   if NOW-OPEN = 0 then return-error 'NESSUN SEGMENTO
      APERTO';
   DELETE-SEGMENT(0);
   if HIDDEN then HIDDEN-SURFACE-CHECK;
   SEGMENT-START[0]←FREE;
   SEGMENT-SIZE[0]←0;
   NOW-OPEN←0;
   return;
end;
```

La routine per l'eliminazione delle superfici nascoste è stata chiamata HIDDEN-SURFACE-CHECK. Per prima cosa questa routine controlla se esiste qualche elemento da eliminare; in caso affermativo, il procedimento viene svolto in tre passi successivi. Il primo passo consiste nello scomporre i poligoni in triangoli. Nel secondo passo si confrontano tutti i triangoli e, per ciascuno di essi, viene creata una lista indicante quale triangolo sta davanti e a quale altro. Il terzo passo usa la lista citata per porre i triangoli nel display file in ordine a partire dallo sfondo sino ad arrivare al primo piano (vedi Figura 10.12).
La routine riassegna inoltre gli indici CFREE e STYLE-COUNT in modo che si abbia la massima disponibilità di memoria per inserire le informazioni relative al segmento di display file che verrà creato successivamente.

Algoritmo 10.14 **HIDDEN-SURFACE-CHECK**
Elimina le superfici nascoste.
Variabili globali CFREE←indice della successiva cella libera del C-buffer
STYLE-COUNT←indice delle successive celle libere per la memorizzazione dei tipi di linea
Variabili locali TRIANGLES←numero di triangoli generati dalla scomposizione dei poligoni

```
begin
   if CFREE ≤ 1 then return;
   SPLIT-INTO-TRIANGLES(TRIANGLES);
   COMPARE-ALL-TRIANGLES(TRIANGLES);
```

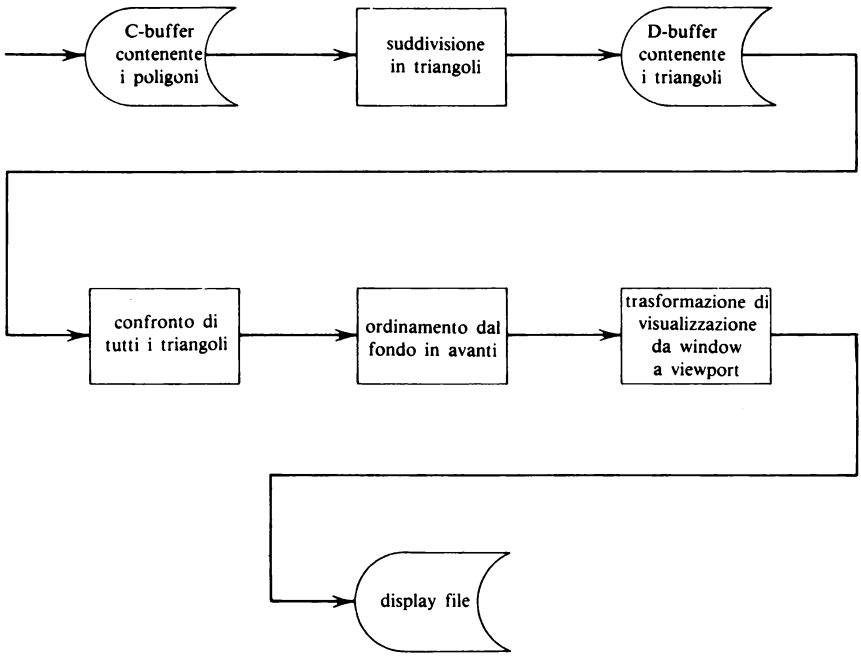


Figura 10.12 Trattamento dei poligoni con la rimozione delle facce nascoste tramite l'algoritmo del pittore

```
    SORT-AND-SAVE-TRIANGLES(TRIANGLES);
    CFREE←1;
    STYLE-COUNT←0;
    return;
end;
```

10.7 SCOMPOSIZIONE DEI POLIGONI IN TRIANGOLI

La prima parte della HIDDEN-SURFACE-CHECK consiste nella scomposizione dei poligoni in triangoli tramite la routine SPLIT-INTO-TRIANGLES (vedi Figura 10.13). In questo paragrafo viene considerata solo la scomposizione di poligoni convessi: la generalizzazione a poligoni concavi sarà argomento della Parte 4 di questo capitolo. Nella routine SPLIT-INTO-TRIANGLES, i poligoni vengono copiati uno per uno dal C-buffer in un'altra area di memoria chiamata B-buffer. Quest'operazione viene eseguita in previsione di quanto verrà fatto nella Parte 4, in cui sarà necessario maggior spazio per il procedimento di

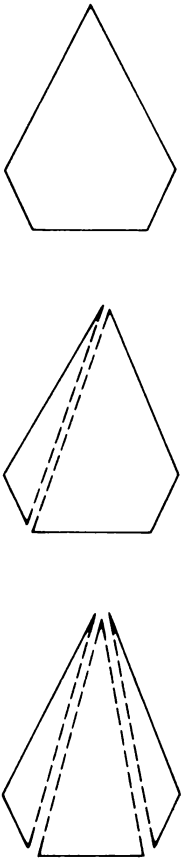


Figura 10.13 Suddivisione di un poligono in triangoli

suddivisione dei poligoni concavi in triangoli. Verranno copiati solo i comandi relativi al disegno di spigoli e non i comandi preliminari relativi al poligono (OP-CODE tra 3 e 31), poiché non c'è alcun bisogno di segnalare che le istruzioni che seguono sono relative al lato di un poligono. Una volta che un poligono è stato copiato nel B-buffer, viene chiamata la routine POLYGON-SPLIT per realizzare l'operazione di scomposizione. I triangoli risultanti vengono memorizzati in una terza area chiamata D-buffer.

Algoritmo 10.15 SPLIT-INTO-TRIANGLES (TRIANGLES)
Scompone i poligoni contenuti nel C-buffer in triangoli, che vengono poi memorizzati nel D-buffer.

Argomenti TRIANGLES←restituisce alla routine chiamante il numero di triangoli generati
Variabili globali CFREE←indice della successiva cella libera contenuta nel C-buffer
IC, XC, YC, ZC←array che costituiscono il C-buffer
Variabili locali DFREE←indice della successiva cella libera del D-buffer
BFREE←indice della successiva cella libera del B-buffer
SIDES←numero di lati del poligono contenuti nel C-buffer
POLY-NUM←contatore dei poligoni elaborati
I←indice del C-buffer

```
begin
   I←1;
   DFREE←1;
   POLY-NUM←0;
   while I<CFREE
         do begin
               POLY-NUM←POLY-NUM+1;
               SIDES←IC[I]
               for BFREE = 1 to SIDES
               do begin
                     I←I+1;
                     PUT-IN-B(IC[I], XC[I], YC[I], ZC[I],
                        BFREE);
                     end;
               POLYGON-SPLIT(SIDES, DFREE, POLY-NUM);
               I←I+1;
               end;
   TRIANGLES←(DFREE−1)/3;
   return;
end;
```

La routine PUT-IN-B memorizza un'istruzione nel B-buffer.

Algoritmo 10.16 PUT-IN-B (OP, X, Y, Z, INDEX)
Inserisce un'istruzione nel B-buffer.
Argomenti OP, X, Y, Z←istruzione da inserire
INDEX←posizione nel B-buffer nella quale inserire l'istruzione
Variabili globali IB, XB, YB, ZB←array che costituiscono il B-buffer
Costanti BSIZE←dimensione degli array del B-buffer. In questo caso sono sufficienti 31 celle, ma per il caso di poligoni concavi può essere necessario un dimensionamento triplo

```
begin
   if INDEX>BSIZE then return-error 'B-BUFFER PIENO';
```

```
    IB[INDEX]←OP;
    XB[INDEX]←X;
    YB[INDEX]←Y;
    ZB[INDEX]←Z;
    return;
end;
```

La routine POLYGON-SPLIT realizza la scomposizione di un poligono. Questa versione parte dall'ipotesi che il poligono sia convesso, cioè tale che ciascun segmento congiungente due vertici sia contenuto all'interno del poligono (o al più giaccia lungo il suo perimetro). Il poligono viene scomposto scegliendo tre vertici per volta, per formare un triangolo con due lati appartenenti al contorno del poligono originale; il terzo lato — sotteso tra il primo e il terzo vertice — deve venir aggiunto: è proprio questo terzo lato che separa il triangolo dal resto del poligono. Poiché questo lato viene aggiunto in modo artificioso, deve essere reso invisibile con un comando MOVE. Se anche la restante parte del poligono viene chiusa con un lato invisibile, il poligono originale risulterà scomposto in un triangolo e in un poligono più piccolo. Questo procedimento può essere continuato finché rimangono solo triangoli.
Nell'algoritmo va inserito un controllo che permetta di riconoscere se i tre vertici scelti per la scomposizione sono allineati. I tre punti scelti per formare un triangolo corrispondono agli indici del B-buffer più bassi tra quelli relativi al poligono che deve essere scomposto. La funzione IN-A-LINE controlla se i punti giacciono su una stessa retta, nel qual caso il triangolo non viene copiato nel buffer. Se invece si tratta di un triangolo non degenere, viene memorizzato il poligono a cui esso appartiene e tramite la routine PUT-IN-D vengono copiati i vertici nel D-buffer. La routine SHIFT-BUFFER chiude il poligono originale ed inserisce il codice di operazione 1 per indicare che questo lato di chiusura è invisibile. Il procedimento viene ripetuto finché il poligono non è interamente suddiviso.

Algoritmo 10.17 POLYGON-SPLIT (SIDES, DFREE, POLY-NUM)
Scompone un poligono in triangoli.

Argomenti SIDES←numero di lati del poligono da scomporre
DFREE←indice della successiva cella libera del D-buffer
POLY-NUM←numero che identifica il poligono da scomporre

Variabili globali IB←array del B-buffer contenente i codici di operazione
SOURCE-POLY←array per memorizzare il poligono da cui è stato tratto il triangolo

Variabili locali N←contatore del B-buffer
OPSAVE←codici operazione di un lato

```
begin
    inizializzazione per il primo triangolo
    N←3;
```

```
        il nuovo lato del triangolo deve essere invisibile
        OPSAVE←IB[1];
        IB[1]←1;
        ciclo che esegue la scomposizione
        while N ≤ SIDES
        do begin
              se si tratta dell'ultimo triangolo, fissa il lato da rendere invisi-
                 bile
              if N = SIDES then IB[N−2]←OPSAVE;
              if not IN-A-LINE(N)
                 then begin
                 memorizza il poligono da cui proviene il triangolo
                 SOURCE-POLY[(DFREE+2)/3]←POLY-NUM;
                 memorizza il triangolo
                 PUT-IN-D(N−2, N, DFREE);
                 end;
              chiude il poligono
              SHIFT-BUFFER(N−2, N−1);
              rende invisibile il lato di chiusura
              IB[N]←1;
              N←N+1;
              end;
        return;
     end;
```

La funzione IN-A-LINE controlla se i tre punti di un triangolo giacciono tutti sulla stessa retta, per evitare di considerare triangoli degeneri (vedi Figura 10.14). In questo modo verrà considerato un minor numero di triangoli, a vantaggio dell'ordinamento successivo. Questa funzione verrà usata anche nella Parte 3, dove si tratterà l'eliminazione delle linee nascoste e quindi si avrà a che fare con poligoni descritti tramite il loro contorno.
I tre vertici di un triangolo giacciono sulla stessa retta se ciascuna coppia di vertici definisce la stessa direzione (e quindi la stessa pendenza). Se i punti (x_1, y_1), (x_2, y_2) e (x_3, y_3) si trovano sulla stessa retta, allora

$$\frac{y_1-y_2}{x_1-x_2} = \frac{y_1-y_3}{x_1-x_3} \tag{10.5}$$

da cui

$$0 = (x_1-x_2)(y_1-y_3)-(y_1-y_2)(x_1-x_3) \tag{10.6}$$

Il test sopra riportato viene eseguito dalla routine IN-A-LINE su tre punti del B-buffer, con una leggera modifica per compensare gli errori di arrotondamento.

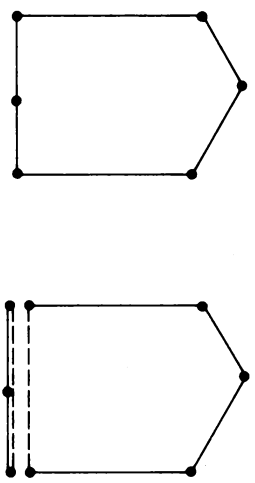


Figura 10.14 Se i tre vertici sono allineati, il triangolo è degenere e deve venir scartato

Algoritmo 10.18 IN-A-LINE (N)
Funzione booleana che determina se tre punti giacciono sulla stessa retta.

Argomenti	N←indice superiore del B-buffer contenente i tre punti
Variabili globali	XB, YB←array del B-buffer contenenti le coordinate x e y dei punti
Variabili locali	L, M←indici relativi agli altri due punti
Costanti	ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
   M←N−1;
   L←N−2;
   IN-A-LINE← |(XB[M]−XB[L])*(YB[M]−YB[N])
         −(XB[M]−XB[N])*(YB[M]−YB[L])| <ROUNDOFF;
   return;
end;
```

L'algoritmo POLYGON-SPLIT usa due routine di servizio: una, chiamata PUT-IN-D, per copiare i vertici del triangolo nel D-buffer; l'altra, chiamata SHIFT-BUFFER, per traslare la posizione di un'istruzione nel B-buffer. Gli algoritmi per queste routine sono riportati di seguito.

Algoritmo 10.19 PUT-IN-D (M, N, DFREE)
Copia le informazioni dalla posizione M alla posizione N del B-buffer nel D-buffer a partire da DFREE.

Argomenti M, N←indici iniziale e finale della porzione di B-buffer da copiare
DFREE←indice della successiva cella libera nel D-buffer
Variabili globali IB, XB, YB, ZB←array che costituiscono il B-buffer
ID, XD, YD, ZD←array che costituiscono il D-buffer
Variabili locali I←indice del B-buffer

```
begin
   for I = M to N
   do begin
         ID[DFREE]←IB[I];
         XD[DFREE]←XB[I];
         YD[DFREE]←YB[I];
         ZD[DFREE]←ZB[I];
         DFREE←DFREE+1;
      end;
   return;
end;
```

Algoritmo 10.20 SHIFT-BUFFER (J, K)
Sposta un elemento nel B-buffer.
Argomenti J←vecchio indice dell'elemento da spostare
K←nuovo indice dell'elemento
Variabili globali IB, XB, YB, ZB←array che costituiscono il B-buffer

```
begin
   IB[K]←IB[J];
   XB[K]←XB[J];
   YB[K]←YB[J];
   ZB[K]←ZB[J];
   return;
end;
```

10.8 CONFRONTO DEI TRIANGOLI

A questo punto, tutti i poligoni sono stati scomposti in triangoli e memorizzati nel D-buffer. Il prossimo passo sarà costituito dal confronto di questi triangoli. Dati due triangoli A e B, esistono tre possibili posizioni reciproche:

1. A copre tutto o parte di B
2. B copre tutto o parte di A
3. i due triangoli non sono sovrapposti

Verranno considerati soltanto oggetti aventi facce che non si compenetrano e costituite da poligoni strettamente convessi, in quanto i poligoni concavi possono avere facce situate in parte davanti e in parte dietro ad altri oggetti. I triangoli saranno ordinati secondo la loro posizione rispetto alla direzione z nelle

coordinate del piano di proiezione: maggiore è il valore della z positiva, più vicino all'osservatore sarà il triangolo. Poiché la normale al piano di proiezione ha verso contrario a quello dell'asse z, per eseguire un ordinamento corretto l'utente deve specificare una normale al piano di proiezione che punta dall'osservatore verso l'oggetto.
Per confrontare due triangoli, si possono utilizzare diversi test, alcuni semplici da applicare, ma che non garantiscono sempre un risultato, altri che forniscono risultati validi, ma che richiedono una grande mole di calcoli. La soluzione più logica per organizzare le verifiche è quella di applicare dapprima le più semplici e, se non si ottiene una risposta corretta, tentare con altre più sofisticate.

10.9 TEST DI MINIMAX

Il test più semplice è chiamato *minimax* e si basa sul fatto che se due triangoli vengono circoscritti con dei rettangoli e i rettangoli non si sovrappongono, è certo che anche i triangoli non si sovrappongono (vedi Figura 10.15).
I rettangoli da considerare devono essere i più piccoli possibili tra quelli che contengono i triangoli: questi rettangoli avranno una base di lunghezza pari alla differenza fra i valori massimo e minimo delle coordinate x dei vertici del triangolo ed un'altezza pari alla differenza fra il valore massimo e minimo delle coordinate y. Due rettangoli sicuramente non saranno sovrapposti se uno dei due si trova completamente al di sopra dell'altro, cioè se la base del secondo è più elevata del lato superiore del primo: ciò significa che la minima coordinata y dei vertici di un triangolo è maggiore della massima coordinata y dei vertici dell'altro. È anche possibile confrontare i lati sinistro e destro con un test analogo eseguito sulle coordinate x. Di seguito viene riportato il test completo (modificato per considerare anche la possibilità di errori di arrotondamento).

Algoritmo 10.21 MINIMAX (L1, L2)
Realizza il test di minimax tra due triangoli.
Argomenti L1, L2←posizioni del D-buffer contenenti l'ultimo vertice di ciascun triangolo
Variabili globali XD, YD←array del D-buffer contenenti le coordinate x e y dei vertici del triangolo
Costanti ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
  MINIMAX←MAX(XD[L1], MAX(XD[L1-1], XD[L1-2]))
          -MIN(XD[L2], MIN(XD[L2-1], XD[L2-2]))
          <ROUNDOFF
          or MAX(YD[L1], MAX(YD[L1-1], YD[L1-2]))
          -MIN(YD[L1], MIN(YD[L2-1], YD[L2-2]))
          <ROUNDOFF
```

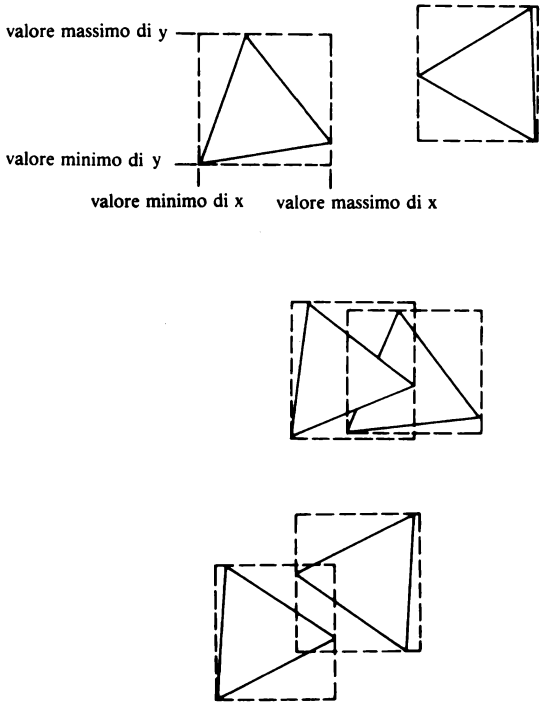


Figura 10.15 Test di minimax. Se i rettangoli non si sovrappongono, anche i triangoli sicuramente non si sovrapporranno

```
            or MAX(XD[L2], MAX(XD[L2-1], XD[L2-2]))
            -MIN(XD[L1], MIN(XD[L1-1], XD[L1-2]))
            <ROUNDOFF
            or MAX(YD[L2], MAX(YD[L2-1], YD[L2-2]))
            -MIN(YD[L1], MIN(YD[L1-1], YD[L1-2]))
            <ROUNDOFF;
    return;
end;
```

Se il test di minimax risulta vero, i triangoli non si sovrappongono, ma se i rettangoli si sovrappongono, non si può essere ancora certi che i triangoli contenuti siano a loro volta sovrapposti. Si dovranno effettuare ulteriori test che, oltre a rivelare l'esistenza di un'area comune ai due triangoli, permettano di decidere quale triangolo sta davanti. Anche in questo caso un test di minimax è il modo più semplice per stabilire quale triangolo sia posto davanti (vedi Figura 10.16): se il valore minimo delle coordinate z di un triangolo è maggiore del va-

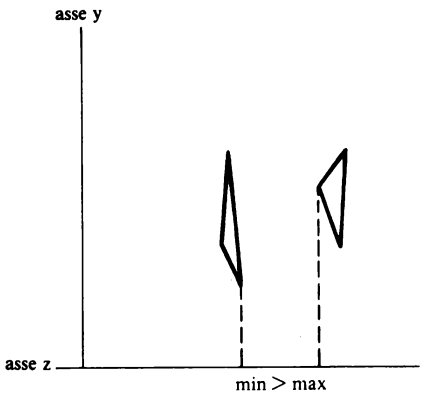


Figura 10.16 Test di minimax relativo alla coordinata z. Se la z minima di un triangolo è maggiore della z massima di un altro triangolo, allora il primo è posto davanti al secondo

lore massimo delle z dell'altro, allora il primo triangolo si trova davanti. Se il test di minimax fallisce, occorrerà usare un altro mezzo per stabilire l'ordine. La funzione, per questo test di minimax sulle coordinate z, assumerà il valore 1 se il secondo triangolo è davanti al primo, -1 se il primo è davanti al secondo, 0 se non è possibile alcuna decisione.

Algoritmo 10.22 Z-MINIMAX (I0, J0)
Test di minimax sulle coordinate z dei due triangoli.
Argomenti I0, J0←posizione nel D-buffer dell'ultimo vertice di ciascun triangolo
Variabili globali ZD←array del D-buffer contenente le coordinate z dei vertici del triangolo
Costanti ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
  if MIN(ZD[J0], MIN(ZD[J0-1], ZD[J0-2]))
       -MAX(ZD[I0], MAX(ZD[I0-1], ZD[I0-2]))
       >ROUNDOFF
  then Z-MINIMAX←1
  else if MIN(ZD[I0], MIN(ZD[I0-1], ZD[I0-2]))
          -MAX(ZD[J0], MAX(ZD[J0-1], ZD[J0-2]))
          >ROUNDOFF
  then Z-MINIMAX←-1
  else Z-MINIMAX←0;
  return;
end;
```

10.10 SOVRAPPOSIZIONE DEI LATI

Esistono diversi altri test che possono essere fatti per stabilire se due triangoli si sovrappongono; un esempio è rappresentato dal test di intersezione, il quale verifica che due triangoli sono sovrapposti se un lato del primo interseca un lato del secondo. Al test di intersezione si può far precedere un test di minimax (vedi Figura 10.17).
Questa volta verranno circoscritti con rettangoli i singoli lati del triangolo, evitando così di confrontare lati inclusi in rettangoli che non si sovrappongono. Il controllo riguardante l'intersezione di due segmenti viene effettuato calcolando il punto di intersezione delle rette a cui appartengono e verificando, in seguito, che il punto di intersezione sia compreso fra i loro estremi. Se le proiezioni di due lati si intersecano, va stabilito l'ordinamento secondo z: se il test Z-MINIMAX ha esito positivo, la risposta è affidabile; se invece il test fallisce, l'ordine dei due triangoli va stabilito confrontando le coordinate z in corrispon-

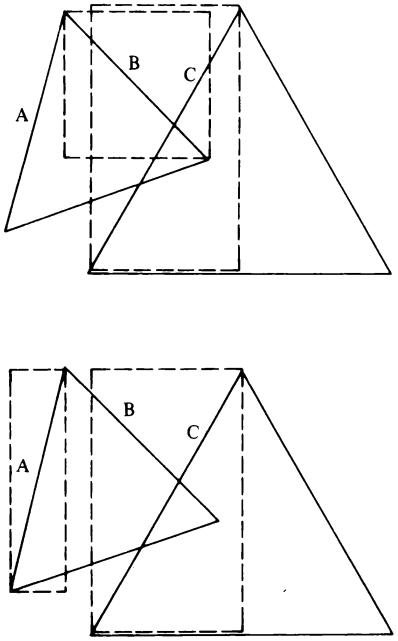


Figura 10.17 Test di minimax relativo a ciascun lato di un triangolo. I lati B e C possono intersecarsi, poiché i relativi rettangoli si sovrappongono; i lati A e C invece non si intersecano, perché i rettangoli che li contengono non si sovrappongono

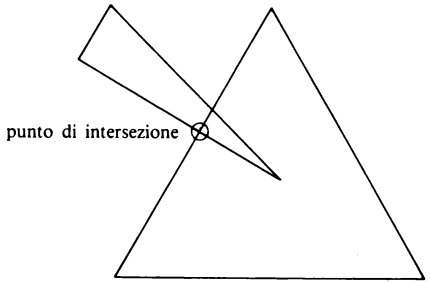


Figura 10.18 Un punto di intersezione fra i lati proiettati di due triangoli può servire per confrontare i valori delle z e determinare l'ordine dei triangoli

denza di un punto (x, y) comune ad entrambi, ovvero dell'intersezione dei due lati (vedi Figura 10.18).
Tutti questi calcoli, peraltro piuttosto noiosi, sono eseguiti in una routine chiamata COMPARE-SIDES.

Algoritmo 10.23 COMPARE-SIDES (I1, I2, I0, J0, L1)
Esegue il confronto tra le coordinate z degli estremi se il segmento compreso tra punti I1 e I2 attraversa un lato del triangolo J0.

Argomenti
I1, I2←posizione nel D-buffer degli estremi di un lato
I0←posizione nel D-buffer del lato del secondo triangolo
J0←posizione nel D-buffer dell'ultimo vertice del triangolo
L1←risultato del test Z-MINIMAX

Variabili globali
XD, YD, ZD←array che contengono le coordinate dei punti nel D-buffer

Variabili locali
XUI, XLI, YUI, YLI←perimetro del rettangolo che circoscrive il primo lato del triangolo
XUJ, XLJ, YUJ, YLJ←perimetro del rettangolo che circoscrive il secondo lato
J1, J2←indici degli altri due vertici del triangolo
D←denominatore usato nel calcolo del punto di intersezione
X←coordinata x del punto di intersezione
ZI, ZJ←coordinata z del punto di intersezione del primo e secondo triangolo

Costanti
ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
  COMPARE-SIDES←0;
  XUI←MAX(XD[I1], XD[I2]);
  XLI←MIN(XD[I1], XD[I2]);
```

```
YUI←MAX(YD[I1], YD[I2]);
YLI←MIN(YD[I1], YD[I2]);
J1←J0−2;
J2←J0−1;
se necessario, prova tutti e tre i lati del triangolo J
while COMPARE-SIDES = 0 and J1 ≤ J0
do begin
      XUJ←MAX(XD[J1], XD[J2]);
      XLJ←MIN(XD[J1], XD[J2]);
      YUJ←MAX(YD[J1], YD[J2]);
      YLJ←MIN(YD[J1], YD[J2]);
      if XUI−XLJ>ROUNDOFF
      and YUI−YLJ>ROUNDOFF
      and XUJ−XLI>ROUNDOFF
      and YUJ−YLI>ROUNDOFF
      then begin
            D←(XD[I1]−XD[I2])*(YD[J1]−YD[J2])
            −(XD[J1]−XD[J2])*(YD[I1]−YD[I2]);
            if|D|>ROUNDOFF
            then begin
                  X←((XD[I1]−XD[I2])*(XD[J2]*YD[J1]
                  −XD[J1]*YD[J2])−(XD[J1]
                  −XD[J2])*(XD[I2]*YD[I1]−XD[I1]*YD[I2]))/D;
                  controlla se il punto di intersezione appartiene ai seg-
                  menti
                  if XLI−X<ROUNDOFF
                  and XLJ−X<ROUNDOFF
                  and X−XUI<ROUNDOFF
                  and X−XUJ<ROUNDOFF
                  then if L1≠0
                  then COMPARE-SIDES←L1
                  else begin
                        stabilisce l'ordine secondo z nel punto di inter-
                        sezione
                        if XUI−XLI<ROUNDOFF
                        then ZI←ZD[I1]
                        else ZI←(ZD[I2]−ZD[I1])*(X−XD[I1])
                              /(XD[I2]−XD[I1])+ZD[I1];
                        if XUJ−XLJ<ROUNDOFF
                        then ZJ←ZD[J1]
                        else ZJ←(ZD[J2]−ZD[I2])*(X−XD[J1])
                              /(XD[J2]−XD[J1])+ZD[J1];
                        if|ZI−ZJ|>ROUNDOFF
                        then if ZJ>ZI
                        then COMPARE-SIDES←1
```

```
                    else COMPARE-SIDES←−1;
                    else begin
                        COMPARE-SIDES
                        ←COMPARE-AT-ENDS(I1, I2, J0);
                        if COMPARE-SIDES = 0
                        then COMPARE-SIDES
                        ←COMPARE-AT-ENDS(J1, J2, I0);
                        end;
                end;
            end;
        J1←J1+1;
        if J1 = J0 then J2←J0−2 else J2←J2+1;
        end;
    return;
end;
```

L'algoritmo sopra riportato contiene delle chiamate alla routine COMPARE-AT-ENDS: questa routine fornisce il risultato nel caso in cui la coordinata z del punto di intersezione abbia lo stesso valore per i due triangoli. In questo caso infatti il punto non fornisce informazioni sufficienti a stabilire l'ordinamento dei triangoli secondo la direzione z: occorre determinare qualche altro punto (vedi Figura 10.19). Considerando allora gli estremi dei segmenti che si intersecano, è possibile stabilire se un estremo è contenuto all'interno dell'altro triangolo: in questo caso si potrà stabilire l'ordinamento dei due triangoli confrontando le coordinate z di tale estremo.
La routine COMPARE-AT-ENDS utilizza altre due routine: INSIDE, che stabilisce se un punto appartiene ad un triangolo, e COMPARE-POINT-PLANE, che confronta il valore z di un punto con la coordinata z del piano del triangolo in corrispondenza della stessa posizione (x, y). Questi due algoritmi vengono dettagliatamente descritti nel prossimo paragrafo.

Algoritmo 10.24 COMPARE-AT-ENDS (I1, I2, J0)
Esamina un estremo di un lato di un triangolo per stabilire l'ordine di profondità rispetto ad un altro triangolo.

Argomenti I1, J2←indici degli estremi di un lato di un triangolo
J0←indice dell'altro triangolo

Variabili globali XD, YD, ZD←array del D-buffer contenenti i vertici del triangolo

```
begin
    if INSIDE(XD[I1], YD[I1], J0)
    then COMPARE-AT-ENDS←COMPARE-POINT-PLANE(XD[I1],
        YD[I1], ZD[I1], J0, 0)
    else if INSIDE(XD[I2], YD[I2], J0)
```

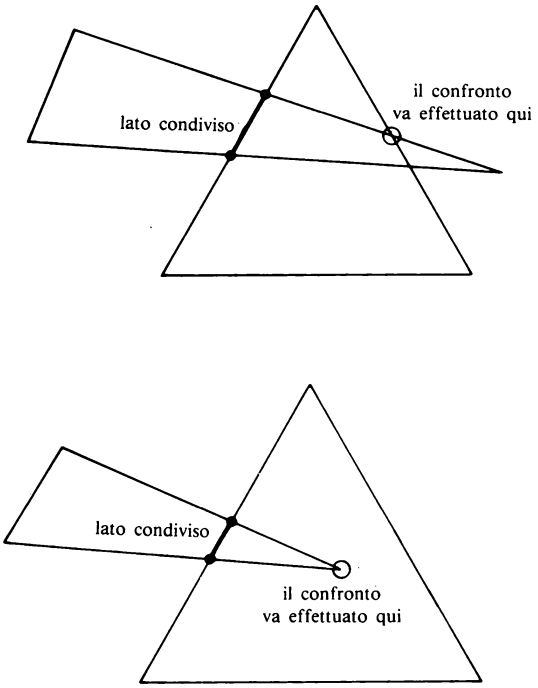


Figura 10.19 Se, nel punto di intersezione, i triangoli hanno identici valori della z (e quindi quel tratto di lato è condiviso fra i due triangoli), il confronto va effettuato in un altro punto di intersezione oppure in un estremo del primo triangolo che sia contenuto nel secondo

```
            then COMPARE-AT-ENDS
            ←COMPARE-POINT-PLANE(XD[I2], YD[I2], ZD[I2], J0, 0)
        else COMPARE-AT-ENDS←0;
        return;
    end;
```

10.11 CONTENIMENTO DI UN PUNTO

Nell'algoritmo precedente si è vista la necessità di stabilire se un punto appartiene ad un triangolo. Nel Capitolo 3 è stato illustrato un metodo per stabilire se un punto è interno ad un poligono; qui verrà introdotto un metodo diverso, che può essere usato per poligoni convessi come i triangoli. Ciascun lato di un triangolo divide il piano che lo contiene in due semipiani, uno dei quali contie-

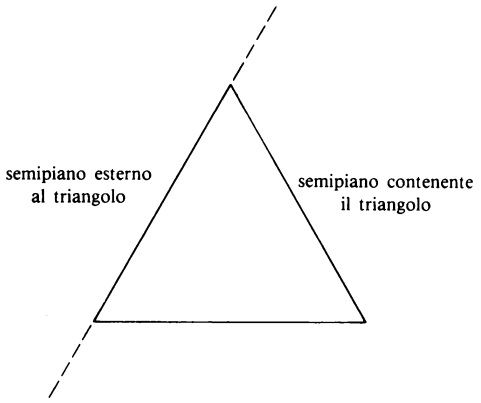


Figura 10.20 La retta cui appartiene il lato di un triangolo divide il piano in due semipiani

ne l'intero triangolo. È possibile eseguire un test per stabilire in quale semipiano giace un certo punto: il test viene effettuato rispetto a ciascuno dei tre lati del triangolo. Se il punto è esterno al triangolo, esso giace in almeno uno dei semipiani esterni (vedi Figura 10.20).
L'equazione della retta passante per i punti (x_1, y_1) e (x_2, y_2) è

$$(x-x_1)(y_2-y_1)-(x_2-x_1)(y-y_1) = 0 \qquad (10.7)$$

Si definisce una funzione chiamata HALF-PLANE, che assume il valore 1, 0 o -1 a seconda che il punto appartenga ad un semipiano, alla retta o all'altro semipiano. Applicando questa funzione a due diversi punti, si può facilmente stabilire se essi giacciono o meno nello stesso semipiano. La retta di suddivisione sarà definita da due vertici del triangolo. I due punti da confrontare sono il punto in esame ed il terzo vertice del triangolo. Se questi due punti sono in semipiani diversi, il punto in esame non appartiene al triangolo. La funzione chiamata INSIDE assume il valore TRUE se un punto è interno ad un triangolo, eseguendo il test HALF-PLANE per ogni lato del triangolo (vedi Figura 10.21). INSIDE può iniziare con un test di minimax, che pone un rettangolo attorno al triangolo e controlla se il punto in esame è esterno al rettangolo. Se ciò è vero allora il punto è anche esterno al triangolo.

Algoritmo 10.25 HALF-PLANE (X, Y, X1, Y1, X2, Y2)
Stabilisce da quale lato della retta, definita dai punti (X1, Y1) e (X2, Y2), si trova il punto (X, Y).
Argomenti X, Y←punto da esaminare
X1, Y1, X2, Y2←due punti sulla retta

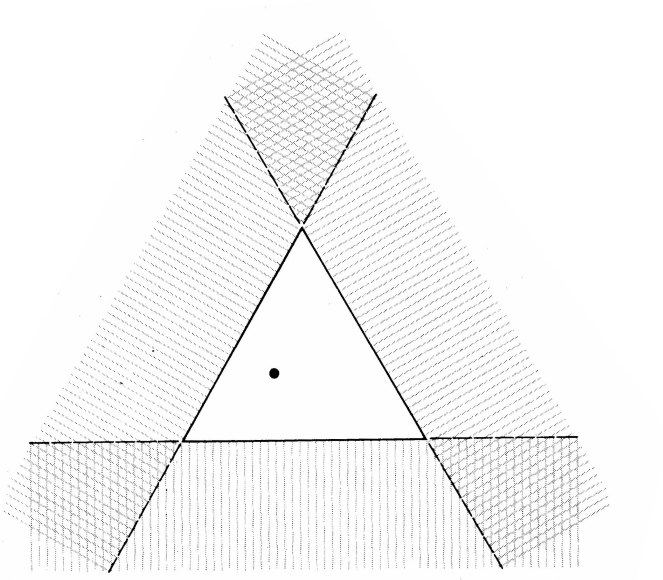


Figura 10.21 Un punto appartiene a un triangolo se, per ognuno dei lati del triangolo, esso giace nel semipiano contenente il triangolo

Variabili locali A, B←due termini dell'espressione di controllo
begin
A←(X−X1)*(Y2−Y1);
B←(X2−X1)*(Y−Y1);
if A>B **then** HALF-PLANE←1
else if A<B **then** HALF-PLANE←−1
else HALF-PLANE←0;
return;
end;

Algoritmo 10.26 INSIDE (X, Y, J0)
Controlla se il punto X, Y è interno al triangolo J0.
Argomenti X, Y←coordinate del punto da controllare
J0←indice del D-buffer corrispondente all'ultimo vertice inserito
Variabili globali XD, YD←array del D-buffer
Costanti ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento
begin
INSIDE←FALSE;
if X−MAX(XD[J0−2], MAX(XD[J0−1], XD[J0]))>ROUNDOFF

```
        or MIN(XD[J0-2], MIN(XD[J0-1], XD[J0]))-X
        >ROUNDOFF
        or Y-MAX(YD[J0-2], MAX(YD[J0-1], YD[J0]))
        >ROUNDOFF
        or MIN(YD[J0-2], MIN(YD[J0-1], YD[J0]))-Y
        >ROUNDOFF
    then return;
    if HALF-PLANE(X, Y, XD[J0-2], YD[J0-2],
            XD[J0-1], YD[J0-1])≠HALF-PLANE(XD[J0], YD[J0],
            XD[J0-2], YD[J0-2], XD[J0-1], YD[J0-1])
    then return;
    if HALF-PLANE(X, Y, XD[J0-1], YD[J0-1], XD[J0], YD[J0])
            ≠HALF-PLANE(XD[J0-2], YD[J0-2], XD[J0-1],
            YD[J0-1], XD[J0], YD[J0])
    then return;
    INSIDE←HALF-PLANE(X, Y, XD[J0], YD[J0], XD[J0-2],
            YD[J0-2])=HALF-PLANE(XD[J0-1], YD[J0-1],
            XD[J0], YD[J0], XD[J0-2], YD[J0-2]);
    return;
end;
```

10.12 REPERIMENTO DI UN PUNTO NEL PIANO DEL TRIANGOLO

Dopo aver stabilito che un punto appartenente ad un triangolo giace all'interno di un altro triangolo (per quanto riguarda i valori delle coordinate x, y), si possono confrontare le coordinate z in tale punto (x, y) per stabilire l'ordine di profondità dei due triangoli. Il problema è che non si conosce ancora la coordinata z corrispondente al triangolo contenente il punto. Per trovare questo valore, basta sostituire nell'equazione del piano che contiene il triangolo più esterno la posizione scelta (x, y) e risolvere rispetto a z. Il problema è quindi trovare l'equazione di un piano passante per tre punti (i tre vertici del triangolo (x_0, y_0, z_0), (x_1, y_1, z_1), (x_2, y_2, z_2)). Nel Capitolo 8 si è visto che un piano è dato dall'equazione

$$A(x-x_0)+B(y-y_0)+C(z-z_0) = 0 \tag{10.8}$$

in cui $[A, B, C]$ è un vettore normale al piano. Si è anche visto come definire un vettore normale al piano tramite il prodotto vettoriale di due vettori ad esso appartenenti. Con i tre punti (x_0, y_0, z_0), (x_1, y_1, z_1), (x_2, y_2, z_2) si possono definire due vettori, per esempio:

$$[x_1-x_2 \quad y_1-y_2 \quad z_1-z_2]$$

e

$$[x_0-x_2 \quad y_0-y_2 \quad z_0-z_2] \tag{10.8.1}$$

Il loro prodotto vettoriale fornisce

$$\begin{aligned} A &= (y_1-y_2)(z_0-z_2)-(z_1-z_2)(y_0-y_2) \\ B &= (z_1-z_2)(x_0-x_2)-(x_1-x_2)(z_0-z_2) \\ C &= (x_1-x_2)(y_0-y_2)-(y_1-y_2)(x_0-x_2) \end{aligned} \tag{10.9}$$

Risolvendo l'equazione del piano rispetto a z, si ottiene

$$z = z_0-[A(x-x_0)+B(y-y_0)]/C \tag{10.10}$$

Queste formule vengono usate dall'algoritmo COMPARE-POINT-PLANE per calcolare e confrontare la profondità del punto e dell'altro triangolo.

Algoritmo 10.27 COMPARE-POINT-PLANE (XM, YM, ZM, J0, L1)
Confronta la profondità di un punto e di un triangolo.

Argomenti	XM, YM, ZM←punto considerato
	J0←indice più alto del D-buffer
	L1←risultato del test Z-MINIMAX
Variabili globali	XD, YD, ZD←array del D-buffer
Variabili locali	A, B, C←parametri dell'equazione del piano del triangolo
	J1, J2←indici dei vertici del triangolo j-esimo
	Z←profondità del triangolo j-esimo
Costanti	ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
   COMPARE-POINT-PLANE←L1;
   if COMPARE-POINT-PLANE≠0 then return;
   J1←J0−1;
   J2←J0−2;
   A←(YD[J1]−YD[J2])*(ZD[J0]−ZD[J2])
      −(ZD[J1]−ZD[J2])*(YD[J0]−YD[J2]);
   B←(ZD[J1]−ZD[J2])*(XD[J0]−XD[J2])
      −(XD[J1]−XD[J2])*(ZD[J0]−ZD[J2]);
   C←(XD[J1]−XD[J2])*(YD[J0]−YD[J2])
      −(YD[J1]−YD[J2])*(XD[J0]−XD[J2]);
   Z←ZD[J0]−(A*(XM−XD[J0])+B*(YM−YD[J0]))/C;
   if Z−ZM>ROUNDOFF then COMPARE-POINT-PLANE←1;
   if ZM−Z>ROUNDOFF then COMPARE-POINT-PLANE←−1;
   return;
end;
```

10.13 CONTENIMENTO DELL'INTERO TRIANGOLO

Se nessuno dei lati dei due triangoli si interseca, rimangono due possibilità: o le proiezioni dei due triangoli non hanno area in comune, o uno dei due triangoli è contenuto interamente all'interno dell'altro (vedi Figura 10.22). Per distinguere questi due casi, occorre stabilire se un punto giace all'interno di un triangolo: sapendo che i lati non si intersecano infatti, se un punto di un triangolo è interno all'altro, si può concludere che ogni punto del primo triangolo giace internamente al secondo. Ancora una volta si confrontano i rettangoli che contengono le proiezioni dei triangoli per decidere l'ordine di contenimento: se la proiezione del triangolo A è contenuta nella proiezione del triangolo B, allora il più piccolo rettangolo contenente A sarà interno al più piccolo rettangolo contenente B. Dopo aver stabilito quale dei due triangoli potrebbe essere quello più interno, si sceglierà un punto appartenente ad esso (il baricentro garantisce l'assenza di problemi di calcolo o di contatto con altri triangoli) e si applicherà la routine INSIDE per completare il test di contenimento. Infatti, se le proiezioni del triangolo sono l'una contenuta nell'altra, occorre stabilirne l'ordinamento nella direzione z, mediante la funzione COMPARE-POINT-PLANE.

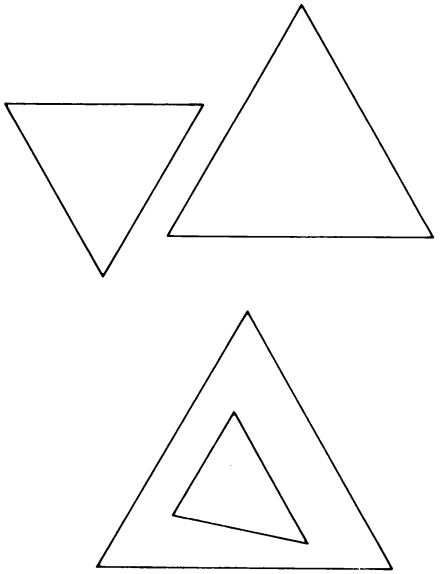


Figura 10.22 Se nessuno dei lati di un triangolo interseca un lato di un secondo triangolo, o i due triangoli non si sovrappongono, oppure uno dei due è contenuto nell'altro. In questo caso anche il suo baricentro sarà contenuto nel triangolo più esterno

L'algoritmo per il confronto di due triangoli i cui lati non si sovrappongono è chiamato COMPARE-CONTAINED.

Algoritmo 10.28 COMPARE-CONTAINED (I0, J0, L1)
Calcola l'ordine di contenimento dei triangoli.

Argomenti I0, J0←indici del D-buffer corrispondenti ai vertici dei due triangoli inseriti nelle posizioni più alte

Variabili globali L1←risultato del test Z-MINIMAX
XD, YD, ZD←array del D-buffer

Variabili locali I1, I2, J1, J2←indici dei vertici dei triangoli
XM, YM, ZM←coordinate di un punto nel centro del triangolo interno
ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
  J1←J0−1;
  J2←J0−2;
  I1←I0−1;
  I2←I0−2;
  if MAX(XD[J0], MAX(XD[J1], XD[J2]))
       −MAX(XD[I0], MAX(XD[I1], XD[I2]))> −ROUNDOFF
  and MIN(XD[J0], MIN(XD[J1], XD[J2]))
       −MIN(XD[I0], MIN(XD[I1], XD[I2]))<ROUNDOFF
  and MAX(YD[J0], MAX(YD[J1], YD[J2]))
       −MAX(YD[I0], MAX(YD[I1], YD[I2]))> −ROUNDOFF
  and MIN(YD[J0], MIN(YD[J1], YD[J2]))
       −MIN(YD[I0], MIN(YD[I1], YD[I2]))<ROUNDOFF
  then begin
    XM←(XD[I0]+XD[I1]+XD[I2])/3;
    YM←(YD[I0]+YD[I1]+YD[I2])/3;
    ZM←(ZD[I0]+ZD[I1]+ZD[I2])/3;
    if INSIDE(XM, YM, J0)
    then COMPARE-CONTAINED
      ←COMPARE-POINT-PLANE(XM, YM, ZM, J0, L1)
    else COMPARE-CONTAINED←0;
    end
  else if MAX(XD[I0], MAX(XD[I1], XD[I2]))
    −MAX(XD[J0], MAX(XD[J1], XD[J2]))> −ROUNDOFF
  and MIN(XD[I0], MIN(XD[I1], XD[I2]))
    −MIN(XD[J0], MIN(XD[J1], XD[J2]))<ROUNDOFF
  and MAX(YD[I0], MAX(YD[I1], YD[I2]))
    −MAX(YD[J0], MAX(YD[J1], YD[J2]))> −ROUNDOFF
  and MIN(YD[I0], MIN(YD[I1], YD[I2]))
    −MIN(YD[J0], MIN(YD[J1], YD[J2]))<ROUNDOFF
  then begin
```

```
        XM←(XD[J0] + XD[J1] + XD[J2])/3;
        YM←(YD[J0] + YD[J1] + YD[J2])/3;
        ZM←(ZD[J0] + ZD[J1] + ZD[J2])/3;
        if INSIDE(XM, YM, I0)
        then COMPARE-CONTAINED
        ←COMPARE-POINT-PLANE(XM, YM, ZM, I0, −L1)
        else COMPARE-CONTAINED←0;
        end;
    return;
end;
```

10.14 DETERMINAZIONE DELL'ORDINE DI PROFONDITÀ

Per completare lo schema di confronto tra triangoli, occorre raggruppare i test presentati finora sotto il controllo di una procedura di supervisione che verrà detta TRIANGLE-COMPARE (vedi Figura 10.23).
La funzione TRIANGLE-COMPARE inizia con un test di minimax. Se il test fallisce, invoca la routine COMPARE-SIZE per controllare se ciascun lato del primo triangolo si interseca con i lati del secondo. Se anche questo controllo fallisce, viene chiamata la routine COMPARE-CONTAINED per completare l'analisi.

Algoritmo 10.29 TRIANGLE-COMPARE (I0, J0)
Confronta due triangoli per stabilirne l'ordine di profondità.
Argomenti I0, J0←locazioni nel D-buffer dell'ultimo vertice per ciascuno dei due triangoli
Variabili locali L1←risultato del test Z-MINIMAX
I1, I2←indici dei vertici del triangolo J-esimo

```
begin
    nel caso in cui i due triangoli non si sovrappongano restituisce zero
    TRIANGLE-COMPARE←0;
    if MINIMAX(I0, J0) then return;
    esegue il test di minimax relativo alla coordinata z
    L1←Z-MINIMAX(I0, J0);
    I1←I0−2;
    I2←I0−1;
    confronta i lati per ottenere il punto di intersezione da usare per il confronto delle coordinate z
    while TRIANGLE-COMPARE = 0 and I1 ≤ I0
    do begin
        TRIANGLE-COMPARE←COMPARE-SIDES(I1, I2, I0, J0, L1);
        I1←I1 + 1;
```

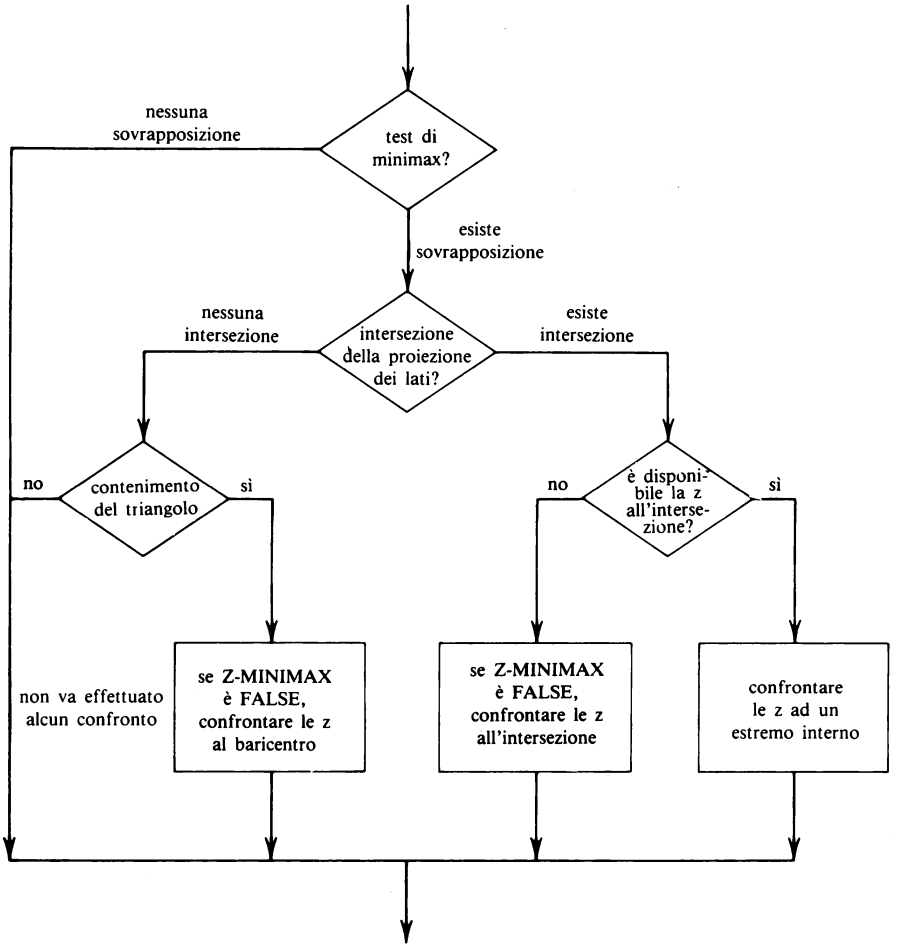


Figura 10.23 Confronto di due triangoli

```
        if I1 = I0 then I2←I0−2 else I2←I2+1;
        end;
        if TRIANGLE-COMPARE≠0
        then TRIANGLE-COMPARE
            ←COMPARE-CONTAINED(I0, J0, L1);
    return;
end;
```

10.15 ORDINAMENTO GEOMETRICO

Sono stati compiuti due passi fondamentali verso l'implementazione dell'algoritmo del pittore: la scomposizione dei poligoni in triangoli (che sono figure convesse e facili da trattare) e il confronto di due triangoli per stabilire l'ordine col quale si presentano all'osservatore. La fase successiva consiste nel confrontare tutti i triangoli tra loro e — per ciascuno — fornire un elenco dei triangoli che lo precedono ed il numero di quelli retrostanti.

10.16 LISTE CONCATENATE

La struttura dati fondamentale usata in questo libro è costituita dagli array. Sfortunatamente, questo tipo di struttura non è dotata della flessibilità necessaria a gestire le informazioni sull'ordinamento dei triangoli. Bisogna quindi usare una diversa struttura dati del tipo di quella introdotta molto sinteticamente nel Capitolo 5, chiamata *lista concatenata* (*linked list*). In un array è possibile reperire l'elemento successivo incrementando un indice, mentre in una struttura a lista si accede all'elemento successivo seguendo un puntatore memorizzato in un elemento della lista. È possibile definire una struttura a liste concatenate tramite due array: il primo contiene i dati da inserire nella lista, mentre il secondo contiene gli indici che puntano agli elementi successivi. L'ordine con il quale vengono introdotte le informazioni negli array non è importante, poiché per accedere ad un elemento non si fa riferimento alla posizione nell'array, ma vengono utilizzati i puntatori che collegano tra loro le celle della lista. Un esempio può chiarire il concetto: si supponga di avere un array detto INFO contenente i dati ed un secondo array chiamato LINK contenente i puntatori (vedi Figura 10.24). La lista può partire in un qualsiasi punto degli array, per esempio dalla posizione corrispondente all'indice 8. Il dato nel primo elemento della lista è 23. Qual è il valore contenuto nel secondo elemento della lista? È il valore 2; per trovare questo secondo elemento si segue il puntatore posto nella corrispondente cella di LINK: il valore del puntatore è 7, il che significa che il secondo elemento della lista è contenuto nella settima posizione dell'array INFO. Analogamente, per trovare il terzo elemento della lista occorre seguire ancora il puntatore contenuto nella seconda cella, che ha valore 2: il terzo elemento della lista è contenuto nella seconda posizione dell'array INFO ed è il valore 14. Ad un certo punto la lista finirà: è necessario quindi un puntatore speciale che indichi la fine della lista, come per esempio il valore 0. In una struttura a liste concatenate gli elementi possono essere inseriti, cancellati o copiati senza spostare la loro posizione nell'array; occorre soltanto cambiare i valori dei puntatori. Per una gestione corretta di questa struttura occorre anche un puntatore che segnali la posizione di partenza della lista.
Assegnando a ciascun triangolo un numero che lo identifichi, è possibile definire l'elenco dei triangoli posti davanti ad un dato triangolo mediante una lista

	indice	dati	puntatori
	1	10	5
	2	14	3
	3	8	6
	4	27	0
	5	39	4
	6	14	1
	7	2	2
inizio	8	23	7

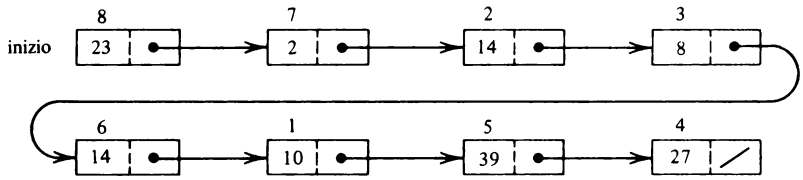


Figura 10.24 Lista concatenata

concatenata. A tale scopo viene utilizzato l'array INFRONT, che conterrà nella posizione *n*-esima un puntatore alla lista dei triangoli posti davanti al triangolo identificato dal numero *n*. Un secondo array, chiamato INBACK, conterrà — in ciascuna posizione — il numero di triangoli posti dietro al triangolo corrispondente. Per accedere, ad esempio, al primo elemento della lista dei triangoli che si trovano davanti al triangolo 3, bisogna considerare l'elemento INFRONT[3] (vedi Figura 10.25). La lista verrà memorizzata negli array INLIST e INLINK.

Si supponga di aver stabilito che il triangolo IFR si trova davanti al triangolo IBK; questa informazione dovrà essere aggiunta alla lista dei triangoli che si trovano di fronte a IBK. Sia NXT l'indice della successiva cella libera nella lista; per prima cosa verrà realizzato il legame tra questa nuova cella ed il primo elemento nella lista

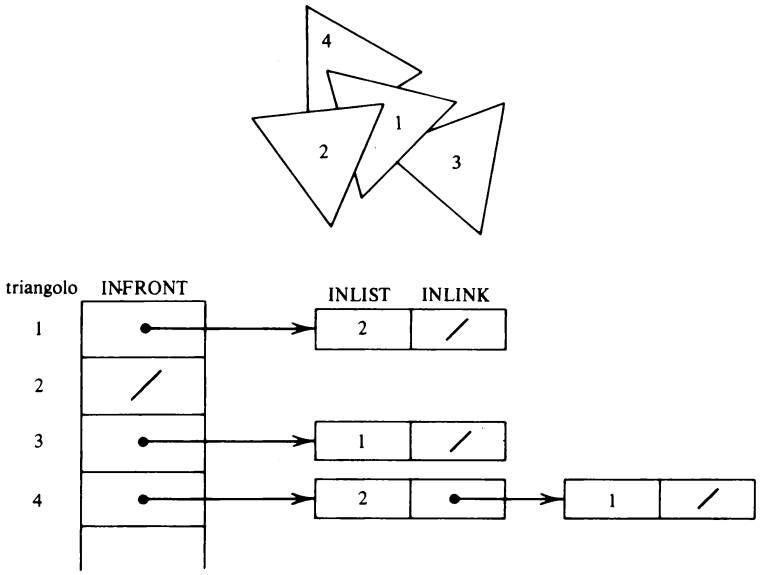


Figura 10.25 Uso di una lista concatenata per ricordare quali triangoli sono posti davanti

INLINK[NXT]←INFRONT[IBK]

Per segnalare che IFR è uno dei triangoli che si trovano davanti a IBK, lo si memorizzerà in questa nuova cella della lista.

INLIST[NXT]←IFR

A questo punto occorre aggiornare INFRONT assegnando il puntatore al nuovo elemento iniziale della lista.

INFRONT[IBK]←NXT

Infine bisogna predisporre lo stato necessario ad un successivo inserimento, considerando una nuova cella libera nella lista

NXT←NXT+1

L'array INBACK contiene, nella posizione *n*-esima, il numero di triangoli retrostanti al triangolo *n*; nell'esempio presentato si dovrà incrementare il contatore dei triangoli posti dietro al triangolo IFR

```
INBACK[IFR]←INBACK[IFR]+1
```

Dati quindi due triangoli ordinati IFR e IBK, il seguente algoritmo aggiornerà le loro liste:

Algoritmo 10.30 ADD-TO-LISTS (IFR, IBK)
Aggiunge IFR alla lista dei triangoli che si trovano davanti al triangolo IBK e incrementa il contatore dei triangoli posti dietro al triangolo IFR.
Argomenti IFR, IBK←triangoli rispettivamente anteriore e posteriore
Variabili globali INFRONT←array dei puntatori alla lista costituita dai vettori INLIST, INLINK
INBACK←array dei contatori dei triangoli retrostanti
NXT←puntatore alla successiva cella libera nella lista
INLIST, INLINK←array che costituiscono la lista

```
begin
   NXT←1;
   INLINK[NXT]←INFRONT[IBK];
   INLIST[NXT]←IFR;
   INFRONT[IBK]←NXT;
   NXT←NXT+1;
   INBACK[IFR]←INBACK[IFR]+1;
   return;
end;
```

10.17 ORDINAMENTO DEI TRIANGOLI

A questo punto serve una routine che consideri tutte le coppie di triangoli, le confronti e le memorizzi in ordine. Ciò viene eseguito dalla routine COMPARE-ALL-TRIANGLES: essa inizializza a zero gli array INFRONT e INBACK.

Algoritmo 10.31 COMPARE-ALL-TRIANGLES (NUMBER-OF-TRIANGLES)
Confronta tutti i triangoli per stabilirne l'ordine di profondità.
Argomenti NUMBER-OF-TRIANGLES←numero di triangoli
Variabili globali INFRONT←array di puntatori alle liste dei triangoli posti davanti
INBACK←array di contatori dei triangoli restrostanti. La dimensione di questi array dovrebbe essere un terzo di quella del D-buffer

```
begin
   for I = 1 to NUMBER-OF-TRIANGLES
   do begin
```

```
        INFRONT[I]←0;
        INBACK[I]←0;
        end;
    considera tutte le coppie di triangoli
    for I = 1 to NUMBER-OF-TRIANGLES−1
    do for K = I+1 to NUMBER-OF-TRIANGLES
        do begin
        li confronta
        L←TRIANGLE-COMPARE(3*I, 3*K);
        if L<0 then ADD-TO-LISTS(K, I);
        if L>0 then ADD-TO-LISTS(I, K);
        end;
si noti che se L è uguale a zero non viene eseguita alcuna operazione
    return;
end;
```

Rimane soltanto una cosa da fare per ottenere l'algoritmo del pittore: bisogna inserire i triangoli nel display file, ordinati a partire dallo sfondo fino al primo piano. Si supponga di voler inserire un triangolo: è noto che esso può venir inserito solo quando tutti i triangoli retrostanti sono stati inseriti e numerati. Per stabilire quali triangoli devono essere introdotti per primi, è sufficiente generare una lista di quelli che hanno il contatore dei triangoli retrostanti nullo, ottenendo così lo sfondo. Questi triangoli possono essere inseriti senza temere che possano nascondere parti in vista. Ogni volta che un triangolo viene aggiunto al display file, esso viene tolto dalla lista INFRONT e quindi per ogni triangolo nella lista occorre diminuire il contatore dei triangoli retrostanti. Quando il contatore va a zero, significa che sono stati inseriti tutti i triangoli posti dietro a quello considerato il quale può, a sua volta, essere aggiunto alla lista dei triangoli che si possono introdurre nel display file. La routine che esegue questa introduzione ordinata di triangoli è chiamata SORTED-ENTRY; l'array contenente i triangoli che possono essere aggiunti al display file è detto TO-BE-DONE e verrà gestito a stack; esso sarà inizializzato da un'altra routine che verrà presentata in seguito. Per copiare il triangolo nel display file, viene usata una procedura chiamata SEND-TO-DF.

Algoritmo 10.32 SORTED-ENTRY

Inserisce in sequenza ordinata i triangoli nel display file.

Variabili globali TO-BE-DONE←array contenente i triangoli che devono essere inseriti nel display file (le cui dimensioni devono essere un terzo di quelle del D-buffer) organizzato a stack

NEXT-TO-DO←posizione dell'ultimo elemento inserito nello stack

INLIST, INLINK←array che realizzano la struttura a lista

```
                    INBACK←array di contatori dei triangoli retrostanti
                    INFRONT←array di puntatori alle liste dei triangoli antistanti
Variabili locali    TEST-TRIANGLE←numero dei triangoli che devono essere inseriti
                    IFT←indice che considera tutti i triangoli posti davanti
                    FRONT-TRIANGLE←identificatore di un triangolo posto davanti
Costanti            TO-BE-DONE-SIZE←dimensione dello stack
begin
  while NEXT-TO-DO≠0
  do begin
     esamina una faccia
     TEST-TRIANGLE←TO-BE-DONE[NEXT-TO-DO];
     INBACK[TEST-TRIANGLE]←−1;
     NEXT-TO-DO←NEXT-TO-DO−1;
     IFT←INFRONT[TEST-TRIANGLE];
     while IFT≠0
     do begin
          FRONT-TRIANGLE←INLIST[IFT];
          INBACK[FRONT-TRIANGLE]
            ←INBACK[FRONT-TRIANGLE]−1;
          if INBACK[FRONT-TRIANGLE] = 0
          then begin
                 NEXT-TO-DO←NEXT-TO-DO+1;
                 if NEXT-TO-DO>TO-BE-DONE-SIZE
                 then return-error 'OVERFLOW NELL'ARRAY
                      TO-BE-DONE'
                 else TO-BE-DONE[NEXT-TO-DO]
                      ←FRONT-TRIANGLE;
                 end;
          IFT←INLINK[IFT];
          end;
     SEND-TO-DF(TEST-TRIANGLE);
     end;
  return;
end;
```

La routine SEND-TO-DF inserisce il triangolo nel display file. La prima cosa da fare è controllare se si è verificato un cambiamento del tipo di riempimento o di linea, nel qual caso il triangolo è preceduto da un comando di cambiamento di tipo: questo controllo viene eseguito dalla routine CHECK-STYLE. Dopodiché, viene posto nel display file un comando per disegnare il triangolo (codice operativo 3) tramite la routine VIEWING-TRANSFORM, la quale inserisce anche i tre lati del triangolo.

Algoritmo 10.33 SEND-TO-DF (TRIANGLE)
Inserisce un triangolo nel display file.
Argomenti TRIANGLE←numero che identifica il triangolo
Variabili globali ID, XD, YD←array del D-buffer
Variabili locali IDX←indice per accedere ad un triangolo contenuto nel D-buffer
I←indice dei lati del triangolo

```
begin
   CHECK-STYLE(TRIANGLE);
   IDX←3*TRIANGLE;
   VIEWING-TRANSFORM(3, XD[IDX], YD[IDX]);
   for I = IDX−2 to IDX
         do VIEWING-TRANSFORM(ID[I], XD[I], YD[I]);
   return;
end;
```

La routine CHECK-STYLE confronta i tipi di linea dei contorni e del riempimento del triangolo con quelli correnti. Se sono diversi, viene generato un comando da porre nel display file per correggerne l'assegnazione.

Algoritmo 10.34 CHECK-STYLE (TRIANGLE)
Controlla che il triangolo venga disegnato utilizzando il tipo di linea e di riempimento corretti.
Argomenti TRIANGLE←triangolo da inserire nel display file
Variabili globali CURRENT-LINE-STYLE←tipo di linea corrente
CURRENT-FILL-STYLE←tipo di riempimento corrente
EDGE-STYLE-LIST←array contenente i tipi di linea dei contorni dei poligoni
FILL-STYLE-LIST←array contenente i tipi di riempimento dei poligoni
SOURCE-POLY←array che indica il punto iniziale per tracciare ciascun triangolo
Variabili locali PLY←poligono a cui appartiene il triangolo

```
begin
   PLY←SOURCE-POLY[TRIANGLE];
   if CURRENT-LINE-STYLE≠EDGE-STYLE-LIST[PLY]
   then begin
         DISPLAY-FILE-ENTER(EDGE-STYLE-LIST[PLY]);
         CURRENT-LINE-STYLE←EDGE-STYLE-LIST[PLY];
         end;
   if CURRENT-FILL-STYLE≠FILL-STYLE-LIST[PLY]
   then begin
         DISPLAY-FILE-ENTER(FILL-STYLE-LIST[PLY]);
         CURRENT-FILL-STYLE←FILL-STYLE-LIST[PLY];
         end;
```

```
    return;
end;
```

Infine, occorre una routine per inizializzare la lista dei triangoli di sfondo ed inserire i triangoli nel display file tramite la SORTED-ENTRY. Questa routine, chiamata SORT-AND-SAVE-TRIANGLES, costituisce l'ultima fase dell'algoritmo del pittore per l'eliminazione delle superfici nascoste.

Algoritmo 10.35 SORT-AND-SAVE-TRIANGLES (NUMBER-OF-TRIANGLES)

Inserisce tutti i triangoli nel display file in sequenza ordinata, a partire dallo sfondo fino al primo piano.

Argomenti NUMBER-OF-TRIANGLES←numero dei triangoli
Variabili globali TO-BE-DONE←array dei triangoli che devono ancora essere visualizzati, organizzato a stack
NEXT-TO-DO←ultimo elemento inserito nello stack
INBACK←array di contatori dei triangoli retrostanti
Variabili locali I←contatore dei triangoli
Costanti TO-BE-DONE-SIZE←dimensione dello stack

```
begin
   NEXT-TO-DO←0;
   ciclo per stabilire l'ordine di inserimento dei triangoli
   for I = 1 to NUMBER-OF-TRIANGLES
   do if INBACK[I] = 0
         then begin
                 NEXT-TO-DO←NEXT-TO-DO + 1;
                 if NEXT-TO-DO > TO-BE-DONE-SIZE
                 then return-error 'OVERFLOW NELL'ARRAY
                    TO-BE-DONE'
                 else TO-BE-DONE[NEXT-TO-DO]←I;
                 end;
   SORTED-ENTRY;
   return;
end;
```

Questa routine completa l'implementazione dell'algoritmo del pittore. Non si può fare a meno di notare che ordinare i poligoni prima di disegnarli è una procedura piuttosto costosa in termini di tempo di elaborazione. Bisognerebbe generare delle scene con parametri di visualizzazione prefissati, obbligando l'utente ad assegnare le figure in sequenza ordinata a partire dallo sfondo, vanificando in questo modo il procedimento fin qui esposto. Ciò significa impegnare più l'utente che il sistema, rendendo il suo lavoro difficile, tanto più che l'ordine dei poligoni può cambiare anche semplicemente osservando la scena da un altro punto di vista. Le routine descritte permettono di visualizzare le superfici dell'oggetto, eliminando le parti nascoste, ma richiedono che il sistema la-

vori utilizzando il frame buffer. Cosa accadrebbe se si avesse a disposizione un terminale a vector refresh o di tipo DVST, adatti al tracciamento di linee? In tutti questi casi, i poligoni verranno mostrati solo tramite i loro contorni (non come aree piene). Anche se i contorni fossero disegnati nell'ordine corretto — cioè partendo dallo sfondo — sarebbero comunque tutti visibili, poiché non c'è alcuna superficie che li nasconda. Per tale tipo di terminali, l'algoritmo presentato non risolve il problema; occorre un algoritmo per l'eliminazione delle linee nascoste.

PARTE 3 — LINEE NASCOSTE

10.18 ELIMINAZIONE DELLE LINEE NASCOSTE

Nei paragrafi seguenti, le routine per l'eliminazione delle superfici nascoste verranno estese per risolvere anche il problema dell'eliminazione delle linee nascoste che definiscono il contorno di poligoni convessi (vedi Figura 10.26). Ogni lato verrà esaminato prima di essere inserito nel display file: sarà confrontato con tutti i triangoli antistanti per vedere se è parzialmente o totalmente nascosto da qualcuno di essi. Tutti i tratti di segmento che formano i lati di un triangolo e che sono classificati come non visibili verranno sostituiti con un comando MOVE, in modo che al momento di interpretare il display file non possano venir visualizzati. Occorre notare che la maggior parte del lavoro preliminare necessario per l'eliminazione delle linee nascoste è già stato fatto nei paragrafi precedenti. Ciò che serve a questo punto, oltre all'operazione di suddivisione dei poligoni in triangoli, è sapere quali triangoli si trovano davanti a quello che si sta considerando, per poter stabilire da che cosa può essere nascosto il lato di

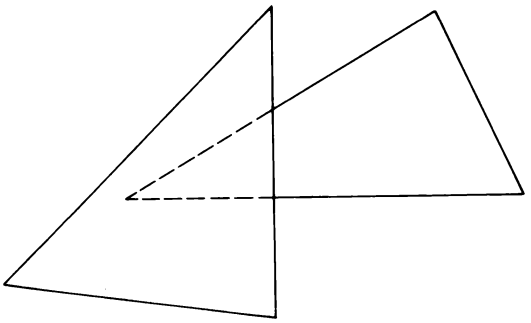


Figura 10.26 Nel caso di eliminazione di linee nascoste, i tratti nascosti dei segmenti vengono sostituiti da comandi MOVE

un triangolo. Nella Parte 2 di questo capitolo, una volta che la routine SORTED-ENTRY aveva stabilito che tutto era pronto per l'inserimento di un triangolo nel display file, veniva chiamata la SEND-TO-DF per realizzare tale operazione. Ora invece la SORTED-ENTRY verrà modificata di modo che, una volta stabilito che un triangolo può essere inserito nel display file, essa chiami la procedura CHECK-SIDES invece della SEND-TO-DF. La CHECK-SIDES controlla tutti i lati del triangolo e sostituisce le parti nascoste con comandi MOVE, prima di porli nel display file.

Algoritmo 10.36 SORTED-ENTRY (Algoritmo 10.32 aggiornato)
Inserisce in sequenza ordinata i triangoli nel display file.

Variabili globali TO-BE-DONE←array contenente i triangoli che devono essere inseriti nel display file (le cui dimensioni devono essere un terzo di quelle del D-buffer), organizzato a stack
NEXT-TO-DO←posizione dell'ultimo elemento inserito nello stack
INLIST, INLINK←array che realizzano la struttura a lista
INBACK←array di contatori dei triangoli retrostanti
INFRONT←array di puntatori alle liste dei triangoli antistanti

Variabili locali TEST-TRIANGLE←numero dei triangoli che devono essere inseriti
IFT←indice per considerare tutti i triangoli posti davanti
FRONT-TRIANGLE←identificatore di un triangolo posto davanti

Costanti TO-BE-DONE-SIZE←dimensione dello stack

```
begin
  while NEXT-TO-DO≠0
  do begin
        esamina una faccia
        TEST-TRIANGLE←TO-BE-DONE[NEXT-TO-DO];
        INBACK[TEST-TRIANGLE]←−1;
        NEXT-TO-DO←NEXT-TO-DO−1;
        IFT←INFRONT[TEST-TRIANGLE];
        while IFT≠0
        do begin
              FRONT-TRIANGLE←INLIST[IFT];
              INBACK[FRONT-TRIANGLE]
              ←INBACK[FRONT-TRIANGLE]−1;
              if INBACK[FRONT-TRIANGLE] = 0
              then begin
                    NEXT-TO-DO←NEXT-TO-DO+1;
                    if NEXT-TO-DO>TO-BE-DONE-SIZE
```

```
                    then return-error 'OVERFLOW NELL'ARRAY
                        TO-BE-DONE'
                    else TO-BE-DONE[NEXT-TO-DO]
                            ←FRONT-TRIANGLE;
                    end;
                IFT←INLINK[IFT];
                end;
            CHECK-SIDES(TEST-TRIANGLE);
            end;
        return;
    end;
```

10.19 COPERTURA DELLE LINEE DA PARTE DEI TRIANGOLI

Prima di completare la descrizione della routine CHECK-SIDES è opportuno vedere tutti i casi che si possono presentare quando un triangolo ricopre un segmento: si userà il test INSIDE sugli estremi del segmento considerato, per controllare se giacciono all'interno del triangolo. Se la risposta al test è affermativa, si può concludere che l'intero segmento è nascosto e va reso completamente invisibile (vedi Figura 10.27). Se dal test risulta che un estremo è contenuto nel triangolo mentre l'altro è all'esterno, si può dedurre che solo parte del segmento è nascosta. In questo caso, per ogni lato del triangolo, occorre calcolare il punto di intersezione con il segmento; tale punto, se esiste, suddivide il

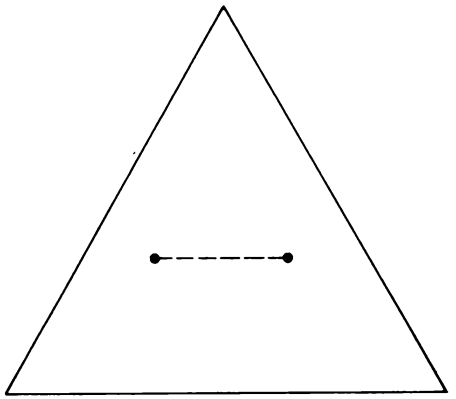


Figura 10.27 Se entrambi gli estremi del segmento sono contenuti nel triangolo, allora l'intero segmento è nascosto

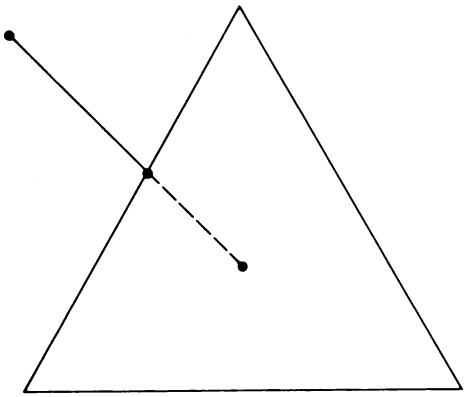


Figura 10.28 Se un solo estremo del segmento è contenuto nel triangolo, allora solo una parte di esso è nascosto

segmento in due parti, una esterna e l'altra appartenente al triangolo: mentre il tratto interno deve essere reso invisibile, quello esterno è ancora visibile e deve essere confrontato con tutti i triangoli posti di fronte a quello considerato (vedi Figura 10.28).
Se entrambi gli estremi giacciono fuori dal triangolo infine si possono verificare due casi:

1. il segmento è completamente esterno al triangolo e quindi visibile per intero (sempre che non sia nascosto da un altro triangolo);
2. il segmento attraversa il triangolo intersecandone due lati: viene quindi suddiviso in tre parti, di cui quella centrale deve essere nascosta. A questo punto i segmenti che potrebbero essere visibili e che vanno confrontati con tutti gli altri triangoli che giacciono di fronte a quello in esame sono due. Potendo controllare solo un segmento alla volta, occorrerà memorizzare in uno stack il secondo segmento da considerare, fintantoché non si è terminata l'elaborazione di quello corrente (vedi Figura 10.29).

Ora non resta altro da fare che calcolare il punto di intersezione tra due segmenti. Questo è un problema già incontrato nella Parte 2, quando si trattava di intersecare i lati dei triangoli nella routine COMPARE-SIDES: verrà quindi utilizzato lo stesso procedimento nella routine chiamata CROSS-CHECK, che restituirà le coordinate del punto di intersezione e una variabile logica per indicare se i due segmenti si intersecano. Gli estremi dei segmenti vengono passati alla routine come variabili globali.

Algoritmo 10.37 CROSS-CHECK (X, Y, YESNO)
Calcola il punto di intersezione fra due segmenti.

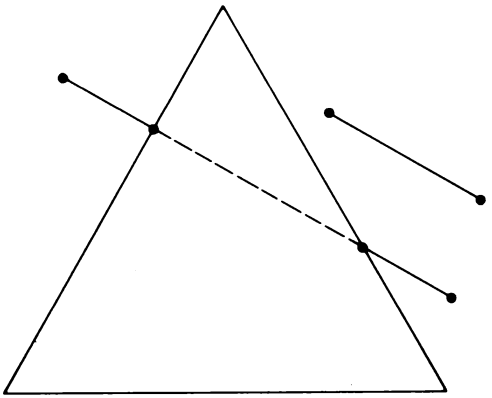


Figura 10.29 Se entrambi gli estremi del segmento sono esterni al triangolo, allora o il segmento è interamente visibile, oppure è diviso in tre parti e la parte centrale è nascosta

Argomenti X, Y←coordinate del punto di intersezione
YESNO←indica se è stata trovata un'intersezione tra i due segmenti

Variabili globali XA, YA, XB,YB←coordinate degli estremi del primo segmento da confrontare
XP, YP, XQ, YQ←coordinate degli estremi del secondo segmento

Variabili locali D, C1, C2←termini di espressioni usate durante il calcolo del punto di intersezione

Costanti ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
   YESNO←FALSE;
   D←(YQ−YP)*(XB−XA)−(XQ−XP)*(YB−YA);
   controlla se i due segmenti sono paralleli
   if|D|<ROUNDOFF then return;
   C1←YA*XB−XA*YB;
   C2←YP*XQ−XP*YQ;
   X←((XQ−XP)*C1−(XB−XA)*C2)/D;
   controlla se il punto di intersezione è interno ai segmenti
   YESNO←MIN(XA, XB)−X<ROUNDOFF
            and MIN(XP, XQ)−X<ROUNDOFF
            and X−MAX(XA, XB)<ROUNDOFF
            and X−MAX(XP, XQ)<ROUNDOFF;
   if not YESNO then return;
```

```
    Y←((YQ−YP)*C1−(YB−YA)*C2))/D;
    YESNO←MIN(YA, YB)−Y<ROUNDOFF
            and MIN (YP, YQ)−Y<ROUNDOFF
            and Y−MAX(YA, YB)<ROUNDOFF
            and Y−MAX(YP, YQ)<ROUNDOFF;
    return;
end;
```

La routine CROSS-CHECK viene applicata per ognuno dei lati del triangolo, finché non si trova un punto di intersezione valido, mediante la procedura INTERSECTION-POINT.

Algoritmo 10.38 INTERSECTION-POINT (X, Y, YESNO, IDX)
Calcola le coordinate X, Y del punto di intersezione fra un segmento e un triangolo.

Argomenti	X, Y←coordinate del punto di intersezione YESNO←indica se è stata trovata un'intersezione IDX←posizione del triangolo nel D-buffer
Variabili globali	XD, YD←array del D-buffer che contengono le coordinate x, y XP, YP, XQ, YQ←estremi di un lato del triangolo considerato XA, YA, XB, YB←estremi del segmento da confrontare con il triangolo
Variabili locali	MARGINAL←indica un punto accettabile coincidente con un estremo (solo se non esistono altri punti di intersezione) X1, Y1←variabili per memorizzare un punto di intersezione coincidente con un estremo
Costanti	ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
   controlla il primo lato
   MARGINAL←FALSE;
   XP←XD[IDX−2];
   YP←YD[IDX−2];
   XQ←XD[IDX−1];
   YQ←YD[IDX−1];
   CROSS-CHECK(X, Y, YESNO);
   if YESNO
      then if|X−XA|+|Y−YA|>ROUNDOFF
           and|X−XB|+|Y−YB|>ROUNDOFF
           then return;
           else begin
                  X1←X;
```

```
                Y1←Y;
                MARGINAL←TRUE;
                end;
    se non c'è intersezione controlla il secondo lato
    XP←XD[IDX];
    YP←YD[IDX];
    CROSS-CHECK(X, Y, YESNO);
    if YESNO
       then if|X−XA|+|Y−YA|>ROUNDOFF
            and|X−XB|+|Y−YB|>ROUNDOFF
            then return;
            else begin
                 X1←X;
                 Y1←Y;
                 MARGINAL←TRUE;
                 end;
    se non c'è intersezione controlla il terzo lato
    XQ←XD[IDX−2];
    YQ←YD[IDX−2];
    CROSS-CHECK(X, Y, YESNO);
    if not YESNO and MARGINAL
       then begin
            X←X1;
            Y←Y1;
            YESNO←TRUE;
            end;
    return;
end;
```

La routine che segue calcola i punti di intersezione quando entrambi gli estremi di un segmento giacciono al di fuori di un triangolo: in questo caso o non vi saranno punti di intersezione, oppure ve ne saranno due. La CROSS-CHECK infatti cerca un punto di intersezione: se la ricerca fallisce per tutti i lati del triangolo, si può concludere che il segmento non lo attraversa. Se invece la ricerca ha successo, occorre allora calcolare l'eventuale secondo punto di intersezione, controllando che i due punti non coincidano.

Algoritmo 10.39 INTERSECTION-PAIR (X, Y, U, V, YESNO, IDX)
Routine che viene applicata nel caso in cui non esistono intersezioni oppure ne esistono due.

Argomenti X, Y, U, V←coordinate dei punti di intersezione
YESNO←indica se sono stati trovati punti di intersezione
IDX←posizione del triangolo nel D-buffer

Variabili globali XD, YD←array del D-buffer contenenti le coordinate x, y
XP, YP, XQ, YQ←estremi di un lato del triangolo considerato
Variabili locali ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
   XP←XD[IDX−2];
   YP←YD[IDX−2];
   XQ←XD[IDX−1];
   YQ←YD[IDX−1];
   calcola la prima intersezione
   CROSS-CHECK(X, Y, YESNO);
   controlla il lato successivo
   XP←XD[IDX];
   YP←YD[IDX];
   if YESNO
   then begin
            CROSS-CHECK(U, V, YESNO);
            if YESNO and (|U−X|+|V−Y|>ROUNDOFF)
            then return;
            end
   else begin
            CROSS-CHECK(X, Y, YESNO);
            if not YESNO
            then return;
            end;
   controlla il terzo lato
   XQ←XD[IDX−2];
   YQ←YD[IDX−2];
   CROSS-CHECK(U, V, YESNO);
   YESNO←YESNO and (|U−X|+|V−Y|>ROUNDOFF);
   return;
end;
```

È necessario, inoltre, usufruire di uno stack in cui inserire il tratto di segmento che verrà considerato una volta terminata l'elaborazione del tratto corrente. A questo punto del calcolo tutte le informazioni riguardanti il poligono sono nel D-buffer; poiché il C-buffer non viene usato, potrà essere impiegato per realizzare lo stack desiderato. Le informazioni riguardanti la profondità non sono più necessarie, poiché l'ordinamento è già stato eseguito: il campo z del C-buffer potrà essere usato per memorizzare un puntatore alla lista di triangoli che si trovano davanti al segmento. La routine PUT-IN-C della Parte 2 verrà applicata per inserire le informazioni nello stack. Per ottenere l'elemento in cima allo stack verrà usata la routine POP-C.

Algoritmo 10.40 POP-C (OP, X, Y, INFR)
Estrae dal C-buffer un lato del poligono e la lista dei triangoli che si trovano di fronte ad esso.

Argomenti OP, X, Y←istruzione per tracciare il lato del poligono in cima allo stack
INFR←puntatore alla lista dei triangoli posti davanti al lato

Variabili globali IC, XC, YC, ZC←array del C-buffer
CFREE←indice della successiva cella libera nello stack costituita dal C-buffer

Variabili locali ICF←indice dell'elemento posto in cima allo stack

```
begin
   ICF←CFREE−1;
   OP←IC[ICF];
   X←XC[ICF];
   Y←YC[ICF];
   INFR←ZC[ICF];
   return;
end;
```

Ogni volta che si deve confrontare un triangolo con tutti quelli antistanti, bisogna procedere considerando tre segmenti (i tre lati del triangolo) ed elaborandone uno per volta. Si possono allora memorizzare i lati di ciascun triangolo nel C-buffer ed estrarli per l'elaborazione seguendo la logica di gestione a stack. Occorre notare che in realtà vengono inserite nello stack le coordinate dei punti corrispondenti agli estremi dei segmenti, da cui si otterrà il segmento intero tracciando il tratto di linea compreso fra la posizione corrente della penna e la posizione letta dallo stack. La seguente routine inizia il procedimento descritto, inserendo nello stack i lati di un triangolo e assegnando la posizione corrente della penna.

Algoritmo 10.41 PUSH-TRIANGLE (TRIANGLE)
Inserire i lati di un triangolo nel C-buffer.

Argomenti TRIANGLE←identificatore di un triangolo

Variabili globali ID, XD, YD←array del D-buffer, che contiene il triangolo
CFREE←indice della successiva cella libera nel C-buffer
INFRONT←array contenente i puntatori alle liste INFRONT
XA, YA←vecchio estremo del segmento

Variabili locali IDX←indice del D-buffer relativo ad un triangolo

```
begin
   CFREE←1;
   IDX←3*TRIANGLE;
   PUT-IN-C(ID[IDX], XD[IDX], YD[IDX], INFRONT[TRIANGLE]);
   PUT-IN-C(ID[IDX−1], XD[IDX−1], YD[IDX−1],
```

```
            INFRONT[TRIANGLE]);
        PUT-IN-C(ID[IDX-2], XD[IDX-2], YD[IDX-2],
            INFRONT[TRIANGLE]);
        XA←XD[IDX];
        YA←YD[IDX];
        return;
    end;
```

10.20 SUDDIVISIONE DI UNA LINEA

Vengono ora riesaminati i vari casi che si verificano quando un triangolo nasconde parzialmente o per intero un segmento, allo scopo di sviluppare le routine che rendono invisibili i tratti di segmento nascosti dal triangolo. Si supponga di aver estratto dallo stack le coordinate (XB, YB) di una nuova posizione della penna che, unitamente alla posizione corrente (XA, YA), definisce il segmento da esaminare. Inoltre, l'estremo A sia esterno al triangolo e sia quindi visibile, mentre l'estremo B sia interno e quindi nascosto. Per calcolare il punto di separazione fra la parte visibile e la parte nascosta del segmento, viene utilizzata la routine INTERSECTION-POINT. Occorre inoltre cambiare l'istruzione per il tracciamento del segmento — dal punto di separazione all'estremo B — in un comando di solo movimento della penna, poiché il segmento va tracciato solo fino al punto di intersezione. Viene quindi creata un'istruzione che consente di tracciare una linea fino al punto di intersezione e che va inserita nello stack. L'istruzione, posta nello stack, che porta dal punto A al punto B, è così sostituita da due istruzioni: la prima, che porta dal punto A al punto di intersezione, è un'istruzione LINE; la seconda, che sposta la penna dal punto di intersezione al punto B, è un'istruzione MOVE. L'unica eccezione si verifica quando il punto di intersezione coincide con il punto A: in tale caso, la parte visibile del segmento sarà troppo piccola e verrà considerata quindi come tratto invisibile. Tutte queste operazioni vengono eseguite dalla routine CHOP-OUT-TO-IN.

Algoritmo 10.42 CHOP-OUT-TO-IN (INFR, IDX)
Suddivide un segmento nel caso in cui l'estremo A è visibile e l'estremo B è nascosto.

Argomenti	INFR←lista dei triangoli antistanti che devono ancora essere esaminati
	IDX←indice del triangolo che nasconde in parte il segmento
Variabili globali	XA, YA, XB, YB←vecchio e nuovo estremo del segmento
Variabili locali	X, Y←coordinate del punto di intersezione
	YESNO←indica se è stato trovato un punto di intersezione
Costanti	ROUNDOFF←numero piccolo maggiore del più grande

```
                    errore di arrotondamento
begin
   calcola il punto di intersezione
   INTERSECTION-POINT(X, Y, YESNO, IDX);
   rende invisibile l'estremo B
   CHANGE-OP-CODE(1);
   divide il lato
   if |XA-X| + |YA-Y| > ROUNDOFF
   then PUT-IN-C(2, X, Y, INFR);
   return;
end;
```

Per cambiare un codice di operazione nello stack del C-buffer, viene usata la routine CHANGE-OP-CODE.

Algoritmo 10.43 **CHANGE-OP-CODE (OP)**

Argomenti OP←codice di operazione da sostituire

Variabili globali IC←array del C-buffer contenente il codice di operazione
CFREE←posizione dello stack contenuta nel C-buffer in cui è possibile inserire un elemento

```
begin
   IC[CFREE - 1]←OP;
   return;
end;
```

Il caso successivo da considerare è quello in cui il vecchio estremo (A) è nascosto e il nuovo estremo (B) è visibile. Per calcolare il punto di suddivisione del segmento, si utilizza ancora la routine INTERSECTION-POINT. Anche in questo caso, se il punto di intersezione coincide con il punto B, la parte visibile è troppo piccola e va considerata quindi invisibile. In caso contrario, un tratto di segmento risulterà nascosto e non saranno necessari ulteriori confronti, mentre l'altro tratto risulterà visibile e andrà confrontato con gli elementi rimanenti della lista INFRONT. Via via che i segmenti vengono esaminati e suddivisi in parti visibili e non visibili, devono essere memorizzati in aree temporanee prima di essere inseriti nel display file, poiché fanno parte di un poligono avente un numero di lati non ancora noto. I segmenti devono quindi essere memorizzati finché non sia stato elaborato l'intero triangolo e solo in seguito il risultato globale potrà essere inserito nel display file. Poiché il B-buffer, a questo punto del procedimento, non viene utilizzato, può essere sfruttato per contenere i segmenti già elaborati. I tratti invisibili del segmento originale possono subito essere caricati nel B-buffer: si tratta soltanto di un'istruzione per muovere la penna nel punto di intersezione, mentre l'istruzione per completare il segmento fino all'estremo B è già nello stack contenuto nel C-buffer. Non resta che aggiornare la lista INFRONT, poiché è stato eseguito il confronto del segmento con un triangolo. Spostando il punto A nel punto di intersezione, il tratto di segmento invisibile non verrà più considerato e il procedimento sarà terminato.

Algoritmo 10.44 CHOP-IN-TO-OUT (INFR, BFREE, IDX)
Suddivide un segmento, nel caso in cui l'estremo A sia nascosto e l'estremo B sia visibile.

Argomenti INFR←puntatore agli elementi rimasti nella lista IN-FRONT
BFREE←indice della successiva cella libera nel B-buffer
IDX←triangolo antistante da esaminare
Variabili globali XA, YA, XB, YB←vecchi e nuovi estremi del segmento
Variabili locali X, Y←punto di intersezione
YESNO←indica se è stato trovato un punto di intersezione
Costanti ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
   INTERSECTION-POINT(X, Y, YESNO, IDX);
   if|XB-X|+|YB-Y|<ROUNDOFF
   then CHANGE-OP-CODE(1)
   else begin
        PUT-IN-B(1, X, Y, 0, BFREE);
        BFREE←BFREE+1;
        LEFT-IN-FRONT(INFR);
        XA←X;
        YA←Y;
        end;
   return;
end;
```

La routine LEFT-IN-FRONT è stata usata per cambiare la lista dei puntatori contenuta nel C-buffer.

Algoritmo 10.45 LEFT-IN-FRONT (INFR)
Aggiorna la lista degli oggetti antistanti contenuta del C-buffer.

Argomenti INFR←puntatore alla nuova lista degli oggetti antistanti
Variabili globali CFREE←posizione dello stack contenuto nel C-buffer in cui è possibile inserire un elemento
ZC←lista degli oggetti antistanti contenuta nel C-buffer

```
begin
   ZC[CFREE-1]←INFR;
   return;
end;
```

Il caso più complicato da trattare è quello in cui entrambi gli estremi di un segmento giacciono al di fuori del triangolo in esame. Per prima cosa occorre applicare la routine INTERSECTION-PAIR, che stabilisce se il triangolo copre il segmento. Se non c'è alcuna sovrapposizione, viene solo aggiornata la lista IN-

FRONT associata al segmento considerato; analogamente, se i due punti di intersezione sono coincidenti, si può concludere che tutto il segmento è visibile. Se d'altro canto esistono due punti di intersezione distinti da trattare, il passo successivo consiste nello stabilire dove si trovano questi punti rispetto ai due estremi A e B. Se necessario, si possono scambiare i punti fra loro, in modo che siano ordinati come segue: (XA, YA), (X, Y), (U, V), (XB, YB). Successivamente si controlla se (X, Y) coincide con (XA, YA), nel qual caso si ripetono le operazioni eseguite dalla routine CHOP-IN-TO-OUT. Nello stesso modo, se il punto (U, V) coincide con il punto (XB, YB), il caso viene trattato come nella CHOP-OUT-TO-IN. Infine, se il segmento viene suddiviso in tre tratti, bisogna aggiungere le istruzioni relative ai due tratti esterni nello stack contenuto nel C-buffer. Queste operazioni sono eseguite in successione dall'algoritmo CHOP-OUT-IN-OUT.

Algoritmo 10.46 CHOP-OUT-IN-OUT (INFR, BFREE, IDX)
Suddivide un segmento nel caso in cui entrambi gli estremi del segmento sono esterni al triangolo e quindi visibili.

Argomenti INFR←lista dei triangoli antistanti che devono essere controllati
BFREE←posizione dello stack contenuto nel B-buffer in cui è possibile inserire un elemento
IDX←triangolo posto di fronte al segmento che va controllato

Variabili globali XA, YA XB, YB←estremi del segmento

Variabili locali X, Y, U, V←coordinate dei punti di intersezione
NO-CROSS←variabile logica che indica se una linea interseca il triangolo
T←variabile di transizione

Costanti ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
   LEFT-IN-FRONT(INFR);
   calcola i punti di intersezione X, Y e U, V
   INTERSECTION-PAIR(X, Y, U, V, NO-CROSS, IDX);
   se non esiste alcuna intersezione, termina
   if NO-CROSS then return;
   if |X-U|+|Y-V|<ROUNDOFF then return;
   ordina i punti di intersezione
   if SIGNOF(XA-XB)≠SIGNOF(X-U)
      or SIGNOF(YA-YB)≠SIGNOF(Y-V)
   then begin
         T←U;
         U←X;
         X←T;
         T←V;
```

```
            V←Y;
            Y←T;
            end;
      if|X−XA|+|Y−YA|<ROUNDOFF
      then if|U−XB|+|V−YB|<ROUNDOFF
            then CHANGE-OP-CODE(1)
            else begin
                  PUT-IN-B(1, U, V, 0, BFREE);
                  BFREE←BFREE+1;
                  XA←U;
                  YA←V;
                  end
      else if|U−XB|+|V−YB|<ROUNDOFF
            then begin
                  CHANGE-OP-CODE(1);
                  PUT-IN-C(2, X, Y, INFR);
                  end
            else begin
                  PUT-IN-C(1, U, V, INFR);
                  PUT-IN-C(2, X, Y, INFR);
                  end;
      return;
end;
```

L'algoritmo sopra riportato utilizza una routine chiamata SIGNOF, per stabilire il segno di un'espressione. Se tale funzione non è disponibile, può essere costruita nel modo seguente:

Algoritmo 10.47 SIGNOF (A)
Restituisce i valori −1, 0, 1 a seconda del segno di A.
Argomenti A←valore del quale occorre stabilire il segno
Costanti ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
   SIGNOF←0;
   if A<−ROUNDOFF then SIGNOF←−1;
   if A>ROUNDOFF then SIGNOF←1;
   return;
end;
```

Non resta che vedere come estrarre un segmento dallo stack. Un segmento verrà tolto dallo stack quando su di esso non dovrà essere effettuato alcun altro controllo, cioè quando il segmento è invisibile, oppure quando sono già stati controllati tutti i triangoli che potrebbero nasconderlo. In questi casi, l'istruzione relativa al segmento viene copiata nel B-buffer che la conserverà finché non

si è pronti per l'inserimento nel display file. Si osservi che per estrarre un elemento dallo stack, occorre decrementare il puntatore alla posizione più alta nella struttura. Si dovrà inoltre aggiornare la posizione (XA, YA) che costituisce l'estremo iniziale del successivo segmento da estrarre. La routine SIDE-IS-DONE (BFREE,OP) esegue tali operazioni.

Algoritmo 10.48 SIDE-IS-DONE (BFREE, OP)
Estrae un'istruzione dallo stack del C-buffer e la memorizza nel B-buffer.
Argomenti BFREE←posizione dello stack contenuta nel B-buffer in cui è possibile inserire un elemento
OP←codice di operazione relativo all'istruzione che deve essere memorizzata
Variabili globali XA, YA, XB, YB←estremi del segmento posto in cima allo stack
CFREE←posizione dello stack contenuto nel C-buffer in cui è possibile inserire un elemento

```
begin
   CFREE←CFREE−1;
   PUT-IN-B(OP, XB, YB, 0, BFREE);
   BFREE←BFREE+1;
   XA←XB;
   YA←YB;
   return;
end;
```

10.21 CONTROLLO DELLE LINEE NASCOSTE PER I LATI DEL TRIANGOLO

Bisogna ora raggruppare tutto ciò che si è visto nell'algoritmo CHECK-SIDES, che sostituirà le istruzioni relative alle parti nascoste del perimetro di un triangolo con comandi MOVE e che porrà poi il poligono risultante nel display file (vedi Figura 10.30).
La CHECK-SIDES, dopo aver inizializzato lo stack inserendo i tre lati del triangolo, esegue un ciclo nel quale considera dapprima il segmento posto più in alto nello stack, controlla che tutte le verifiche ad esso relative siano state eseguite e quindi lo estrae dallo stack. Per ogni segmento si considerano tutti i triangoli contenuti nella lista INFRONT e si controlla se i suoi estremi sono esterni o interni al triangolo e — a seconda del caso — viene chiamata una delle routine CHOP-OUT-TO-IN, CHOP-IN-TO-OUT, CHOP-OUT-IN-OUT oppure SIDE-IS-DONE. Quando tutti i lati sono stati considerati e quindi lo stack è stato svuotato, il controllo viene trasferito alla routine SEND-B-TO-DF che inserisce il poligono risultante nel display file.

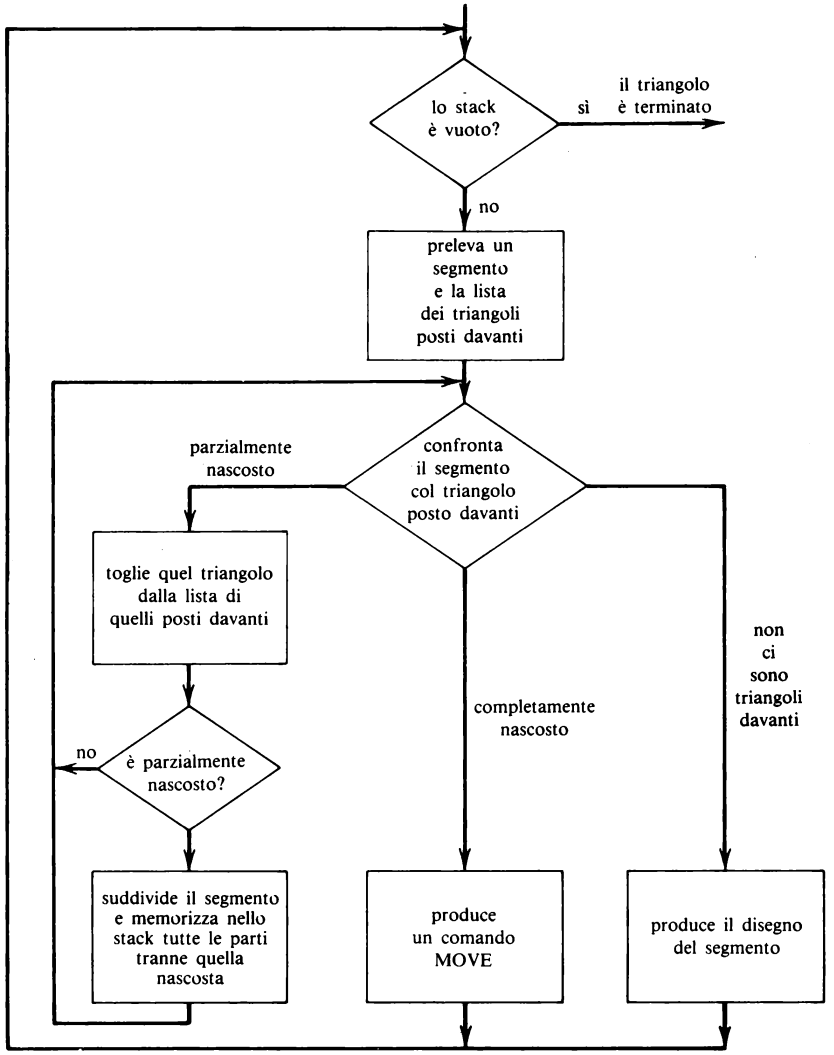


Figura 10.30 Suddivisione di un triangolo in un poligono con lati visibili e nascosti

Algoritmo 10.49 CHECK-SIDES (TRIANGLE)
Elimina le linee nascoste da un triangolo e inserisce il risultato nel display file.
Argomenti TRIANGLE←identificatore del triangolo da visualizzare
CFREE←indice della successiva cella libera nel C-buffer

```
                    XA, YA, XB, YB←estremi del segmento da sottoporre al
                      test
                    INLIST, INLINK←liste dei triangoli antistanti il seg-
                      mento
Variabili locali    BFREE←indice della successiva cella libera nel B-buffer
                    OP←codice di operazione per l'istruzione relativa al seg-
                      mento considerato
                    INFR←puntatore alla lista dei triangoli antistanti
                    IDX←indice del D-buffer relativo al triangolo considerato
begin
   PUSH-TRIANGLE(TRIANGLE);
   BFREE←1;
   while CFREE≠1
   do begin
         POP-C(OP, XB, YB, INFR);
         while INFR≠0 AND OP≠1
         do begin
            IDX←3*INLIST[INFR];
            INFR←INLINK[INFR];
            if INSIDE(XA, YA, IDX)
            then if INSIDE(XB, YB, IDX)
                 then CHANGE-OP-CODE(1);
                 else CHOP-IN-TO-OUT(INFR, BFREE, IDX)
            else if INSIDE(XB, YB, IDX)
                 then CHOP-OUT-TO-IN(INFR, IDX)
                 else CHOP-OUT-IN-OUT(INFR, BFREE, IDX);
            POP-C(OP, XB, YB, INFR);
            end;
         SIDE-IS-DONE(BFREE, OP);
         end;
   SEND-B-TO-DF(TRIANGLE, BFREE);
   return;
end;
```

La routine SEND-B-TO-DF funziona come la routine SEND-TO-DF presentata nella Parte 2: essa controlla la correttezza dei tipi relativi al contorno e di quelli di riempimento ed inserisce il poligono risultante nel display file. La differenza sta nel fatto che ora si può avere a che fare con più di tre lati e che essi vengono memorizzati nel B-buffer e non più nel D-buffer.

Algoritmo 10.50 SEND-B-TO-DF (TRIANGLE, BFREE)
Inserisce un poligono nel display file.

Argomenti TRIANGLE←identificatore di un triangolo facente parte del poligono da visualizzare
BFREE←indice della successiva cella libera del B-buffer

```
Variabili globali   IB, XB, YB←array del B-buffer
Variabili locali    SIDES←numero di lati del poligono
                    I←indice relativo ai lati del poligono
begin
   CHECK-STYLE(TRIANGLE);
   SIDES←BFREE−1;
   VIEWING-TRANSFORM(SIDES, XB[SIDES], YB[SIDES]);
   for I = 1 to SIDES
   do VIEWING-TRANSFORM(IB[I], XB[I], YB[I]);
   return;
end;
```

Le routine di questo paragrafo hanno esteso l'algoritmo per l'eliminazione delle superfici nascoste anche alle linee. Si possono ora visualizzare in modo realistico oggetti tridimensionali, tracciando i contorni dei poligoni che li descrivono o evidenziandone le aree. Le linee nascoste possono essere eliminate su terminali a vector refresh, su terminali DVST e su plotter. Questi algoritmi però possono essere applicati soddisfacentemente solo a poligoni convessi e, in generale, non funzionano in presenza di sovrapposizioni cicliche o di facce che si compenetrano. Nell'ultima parte di questo capitolo verranno presentate alcune estensioni dell'algoritmo, che permetteranno di considerare anche poligoni concavi (vedi Figura 10.31).

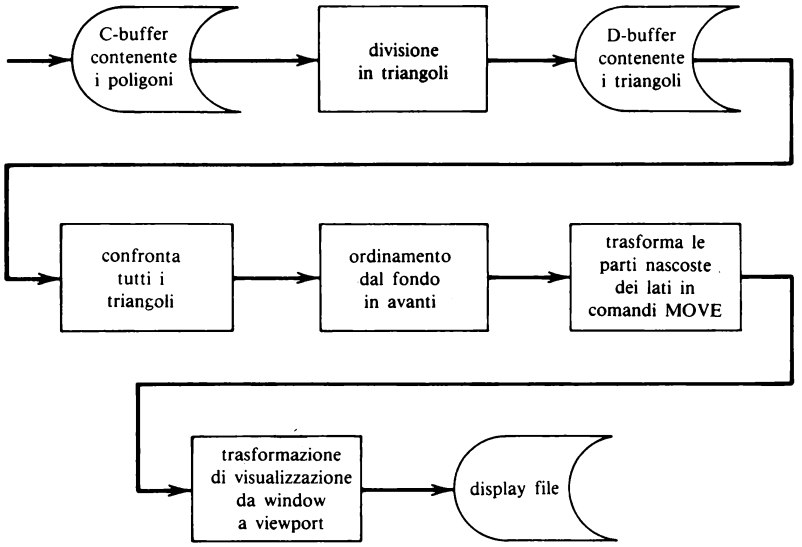


Figura 10.31 Procedimento di rimozione delle linee nascoste

PARTE 4 — SUPERFICI NASCOSTE CONCAVE

10.22 GENERALIZZAZIONE AI POLIGONI CONCAVI

Nelle parti precedenti di questo capitolo sono stati considerati soltanto oggetti costituiti da poligoni convessi, poiché il metodo usato per suddividere i poligoni in triangoli è molto semplice, ma è limitato a questo tipo di figure: isolare ogni volta tre vertici consecutivi e svincolarli dal resto del poligono, non è però sufficiente se si considerano anche poligoni concavi (vedi Figura 10.32).
La prima cosá da fare per poter trattare casi del genere è riconoscere con quale tipo di poligono si ha a che fare. Se, applicando il procedimento di suddivisione visto in precedenza, accade che almeno un vertice del poligono sia contenuto all'interno del triangolo ottenuto, la suddivisione va eseguita usando una diversa strategia.
Si inizia col considerare il punto posto più a sinistra ed appartenente al poligono (come nel caso di un poligono convesso) ed i lati da esso uscenti, fino a formare un triangolo che potrebbe andar bene per la scomposizione.
Se un altro vertice del poligono giace all'interno di tale triangolo, è possibile un'ulteriore suddivisione lungo il segmento che congiunge il punto posto più a sinistra (in assoluto) col punto posto più a sinistra tra quelli interni al triangolo (vedi Figura 10.33). È sempre possibile suddividere un poligono con più di tre lati, separando ogni volta il triangolo più a sinistra, oppure effettuando la sud-

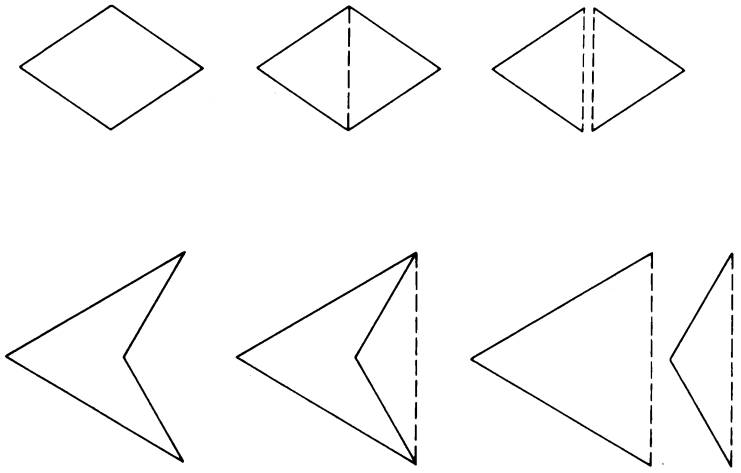


Figura 10.32 Il metodo descritto per la suddivisione di poligoni convessi non funziona con poligoni concavi

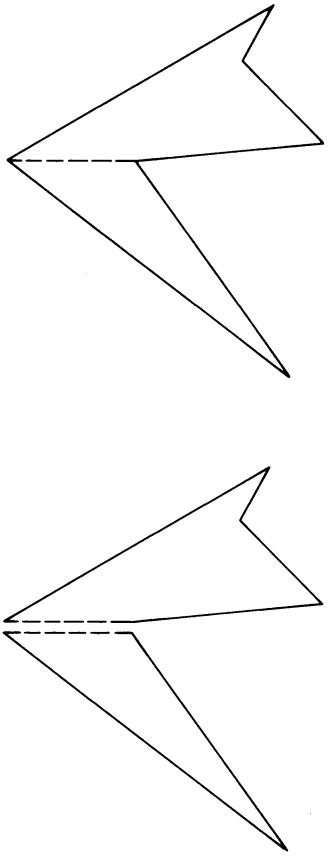


Figura 10.33 Suddivisione ottenuta collegando il punto più a sinistra in assoluto con quello interno più a sinistra

divisione lungo la retta che congiunge il punto più a sinistra del poligono col punto più a sinistra interno al triangolo (vedi Figura 10-34).
L'algoritmo SPLIT-VERTEX esamina il poligono per determinare due punti che possano servire come estremi per una suddivisione. I vertici del poligono si trovano nel B-buffer ed i loro indici sono compresi fra M ed N. Alla routine vengono passati come argomenti il vertice più a sinistra (L) unitamente a quello che lo precede (LB) ed a quello che lo segue (LA), che saranno i tre vertici del triangolo proposto per la suddivisione. Essa restituisce il vertice posto più a sinistra tra quelli che giacciono all'interno del triangolo, col nome di SPLIT-VERTEX; per trovarlo, bisogna dapprima determinare il lato superiore e quello inferiore del triangolo e stabilire quale dei due punti LA e LB è posto più in alto.

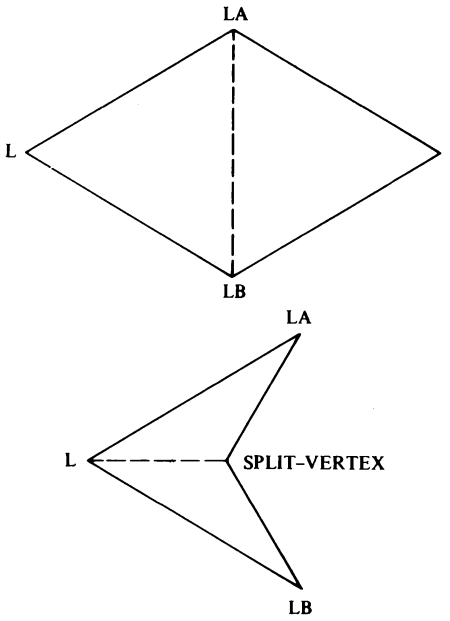


Figura 10.34 Ogni poligono può essere suddiviso in una di queste due maniere

SPLIT-VERTEX è inizializzato al punto posto più a destra fra LA ed LB per eseguire il confronto con tutti gli altri vertici tranne che con L, LA ed LB, che vengono tralasciati perché non interessano punti sul margine del triangolo. Lo stesso dicasi per i vertici che si trovano esternamente ai lati superiore ed inferiore del triangolo. Il punto cercato deve essere a sinistra del vertice SPLIT-VERTEX corrente e posto sul lato «interno» dei segmenti di estremi L-LA ed L-LB. Se un punto supera tutti questi test, diventa il nuovo SPLIT-VERTEX. Se, dopo aver esaminato tutti i vertici, SPLIT-VERTEX coincide ancora con LA o LB, la suddivisione può essere effettuata lungo la congiungente di LA con LB, oppure lungo la congiungente di L con SPLIT-VERTEX.

Algoritmo 10.51 SPLIT-VERTEX (L, LA, LB, M, N)
Determina due vertici tra i quali è possibile tracciare una retta di suddivisione del poligono concavo.

Argomenti
L←vertice posto più a sinistra
LA, LB←vertici che precedono e seguono quello posto più a sinistra
M, N←primo e ultimo vertice del poligono

Variabili globali
XB, YB←array del B-buffer contenenti le coordinate x e y dei punti

```
Variabili locali    YU, YL←lati superiore ed inferiore del triangolo
                    LPU, LPL←punti di suddivisione superiore ed inferiore,
                      relativi al triangolo
                    K←variabile di scansione lungo tutti i vertici del poligono
begin
   YU←MAX(YB[L], MAX(YB[LA], YB[LB]));
   YL←MIN(YB[L], MIN(YB[LA], YB[LB]));
   if YB[LB]>YB[LA]
   then begin
            LPU←LB;
            LPL←LA;
            end
   else begin
            LPU←LA;
            LPL←LB;
            end;
   if XB[LB]>XB[LA] then SPLIT-VERTEX←LB
   else SPLIT-VERTEX←LA;
   for K = M to N
   do if K≠LA and K≠L and K≠LB
            and YB[K] ≤ YU and YB[K] ≥ YL
            and XB[K] < XB[SPLIT-VERTEX]
            and (YB[K]−YB[L])*(XB[LPU]−XB[L])
               ≤(YB[LPU]−YB[L])*(XB[K]−XB[L])
            and (YB[K]−YB[L])*(XB[LPL]−XB[L])
               ≥(YB[LPL]−YB[L])*(XB[K]−XB[L])
            then SPLIT-VERTEX←K;
   return;
end;
```

Spezzando il poligono lungo la retta congiungente i vertici L e SPLIT-VERTEX, si otterranno due poligoni più piccoli che possono non essere triangoli. Non è possibile quindi copiarne immediatamente uno nel D-buffer, come si faceva nella Parte 2, ma occorre continuare per entrambi il procedimento di suddivisione. Essi possono venir memorizzati nel B-buffer, con qualche piccolo spostamento, a patto che siano noti gli indici iniziale e finale delle porzioni occupate dai due poligoni. Si supponga di considerare l'ultimo poligono contenuto nel B-buffer (quello con indice più elevato), i cui vertici abbiano indici compresi fra M ed N. Si supponga inoltre di aver usato la funzione SPLIT-VERTEX e di aver stabilito che il poligono deve venir suddiviso lungo una retta passante per i vertici con indici M1 ed N1, con M1<N1. Uno dei sottopoligoni conterrà le istruzioni memorizzate fra la posizione M1 e la posizione N1, l'altro quelle dalla M alla M1 e dalla N1 alla N. Le istruzioni di indici M1 ed N1 vengono duplicate per formare i due nuovi lati creati dalla suddivisione. Un modo per raggruppare correttamente queste istruzioni è quello di copiare il

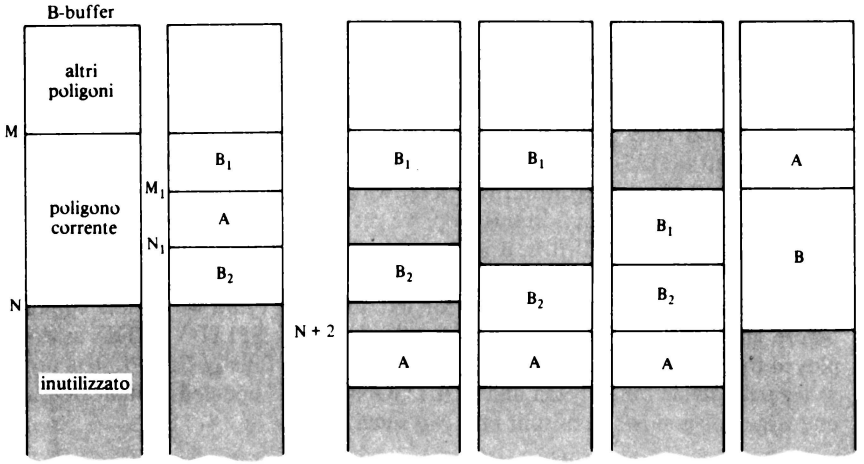


Figura 10.35 Suddivisione di un poligono nel B-buffer

blocco compreso fra gli indici M1 e N1 in una zona non sfruttata del B-buffer, per liberare spazio e compattare gli altri due blocchi di istruzioni relativi al poligono. Il blocco da N1 ad N deve essere spostato di due posizioni per ospitare i due nuovi lati che si sono formati con la suddivisione ed il blocco di estremi M e M1 va avvicinato al blocco di indici compresi fra N1 e N in modo tale che tutte le istruzioni del poligono siano contigue. Infine, il blocco di estremi M1 e N1 può essere ricopiato nello spazio che precede i due blocchi M-M1, N1-N (vedi Figura 10.35).
Le istruzioni relative ai lati lungo la linea di suddivisione sono poste nelle celle M e N1 + 2 e devono contenere un codice di operazione uguale a 1 perché questi lati siano invisibili. La routine DOSPLIT esegue lo spostamento dei lati del poligono usando la routine SHIFT-BUFFER già incontrata nella Parte 2 di questo capitolo (algoritmo 10.20).

Algoritmo 10.52 **DOSPLIT (M, M1, N1, N)**
Suddivide in due parti il poligono.
Argomenti M, N←estremi del poligono da suddividere contenuto nel B-buffer
N1, M1←estremi di uno dei sottopoligoni
Variabili globali IB←array del B-buffer contenente i codici di operazione
Variabili locali K, K1, K2←fattori di spostamento delle istruzioni relative al poligono

begin
copia il sottopoligono in una zona libera del B-buffer

```
    K←N+3−M;
    for J = M1 to N1 do SHIFT-BUFFER(J, J+K);
    sposta la metà superiore del secondo sottopoligono: si noti che ciò viene
    eseguito decrementando l'indice
    for J = N to N1 do SHIFT-BUFFER(J,J+2);
    sposta la metà inferiore del secondo sottopoligono
    K←N1−M1+1;
    for J = M1 to M do SHIFT-BUFFER(J, J+K);
    ricopia il primo sottopoligono
    K1←N+3−M1;
    K2←M−M1;
    for J = M1 to N1 do SHIFT-BUFFER(J+K1, J+K2);
    rende invisibile la suddivisione
    IB[M]←1;
    IB[N1+2]←1;
    return;
end;
```

Non resta ora che modificare la POLYGON-SPLIT per poter usare le routine SPLIT-VERTEX e DOSPLIT, e completare così l'estensione ai poligoni concavi. Poiché potrebbero essere presenti contemporaneamente parecchi poligoni nel B-buffer, bisogna essere in grado di distinguerli tra loro. Verrà quindi memorizzato l'indice dell'ultima istruzione di ciascun poligono in un array chiamato POLYGON-END. Poiché un poligono inizia laddove finisce il precedente, l'array fornirà sia la posizione dell'istruzione di partenza, sia quella dell'istruzione finale. All'inizio della POLYGON-SPLIT si avrà solo il singolo poligono non ancora suddiviso, con un numero di lati pari a SIDES; l'array POLYGON-END è quindi inizializzato ai valori zero e SIDES per indicare il punto iniziale e quello finale del poligono. La parte principale dell'algoritmo è costituita da un ciclo che continua a suddividere i poligoni finché POLYGON-END non sia vuoto. All'interno del ciclo viene considerato il poligono superiore, per vedere se si tratta di un triangolo. In caso affermativo, esso viene copiato nel D-buffer e tolto dallo stack di poligoni che devono ancora essere suddivisi. Se non è un triangolo, viene suddiviso iniziando con la ricerca del vertice più a sinistra e di quelli che lo precedono e lo seguono per poter in seguito creare un triangolo.
La routine SPLIT-VERTEX viene impiegata per stabilire dove deve essere effettuata la suddivisione. Infine, viene usata la routine DOSPLIT per scomporre effettivamente il poligono e POLYGON-END viene aggiornato per indicare che un poligono è stato suddiviso.

Algoritmo 10.53 POLYGON-SPLIT (SIDES, DFREE, KNT)
(Algoritmo 10.17 aggiornato)
Suddivide un poligono concavo in triangoli.
Argomenti SIDES←numero di lati del poligono

```
                    DFREE←successiva cella libera del D-buffer
                    KNT←numero di poligoni da considerare
Variabili globali   SOURSE-POLY←array contenente il numero di identifi-
                      cazione del poligono originale
Variabili locali    POLYGON-END←istruzioni relative all'ultimo lato del
                      poligono
                    DO-NEXT←posizione di testa di POLYGON-END
                    M, N←estremi del poligono
                    JP←identificatore di un triangolo
                    L←vertice più a sinistra di un poligono
                    LA, LB←vertici che precedono e seguono quello posto
                      più a sinistra
                    LS←vertice utile all'eventuale suddivisione
                    M1, N1←vertici della linea di suddivisione
begin
  DO-NEXT←2;
  POLYGON-END[1]←0;
  POLYGON-END[2]←SIDES;
  while DO-NEXT>1
      do begin
          M←POLYGON-END[DO-NEXT−1]+1;
          N←POLYGON-END[DO-NEXT];
          if N−M = 2
          then begin
              L←M+1;
              if not IN-A-LINE(M, N)
              then begin
                  JP←INT((DFREE+2)/3);
                  SOURSE-POLY[JP]←KNT;
                  PUT-IN-D(M, N, DFREE);
                  end;
              DO-NEXT←DO-NEXT−1;
              end
          else begin
              L←LEFT-MOST(M, N);
              if L = N then LA←M else LA←L+1;
              if L = M then LB←N else LB←L−1;
              LS←SPLIT-VERTEX (L, LA, LB, M, N);
              if LS = LA or LS = LB
              then begin
                  M1←MIN(LA, LB);
                  N1←MAX(LA, LB);
                  end
              else begin
                  M1←MIN(L, LS);
```

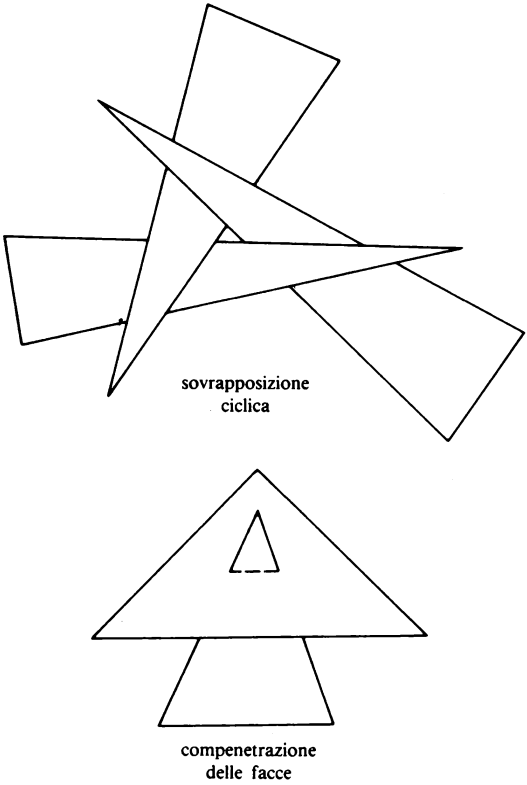


Figura 10.36 Casi che il sistema non è in grado di trattare

```
                    N1←MAX(L, LS);
                    end;
                DOSPLIT(M, M1, N1, N);
                POLYGON-END[DO-NEXT]←M+N1−M1;
                DO-NEXT←DO-NEXT+1;
                POLYGON-END[DO-NEXT]←N+2;
                end;
            end;
    return;
end;
```

Con questo vengono completati gli algoritmi per l'eliminazione di linee e di superfici nascoste, che possono essere applicati alla maggior parte delle scene reali, sebbene permangano alcuni casi di difficile risoluzione, come, per esempio,

quello della *sovrapposizione ciclica* o quello della *compenetrazione delle facce* (vedi Figura 10.36).
In questi casi, un poligono si trova contemporaneamente con parti visibili e parti nascoste; le tecniche descritte non sono in grado di risolvere una situazione del genere. Per poterla trattare, bisognerebbe per prima cosa poterla riconoscere ed in seguito occorrerebbe eseguire una suddivisione del poligono tale che le parti risultanti siano o completamente in vista o completamente nascoste.

ESERCIZI

10.1 Per ciascuno dei seguenti triangoli, calcolare il vettore normale al piano che lo contiene

a) (0, 1, 2), (2, 1, 2), (3, 5, 2)
b) (4, 3, 1), (2, 3, 1), (3, 3, 2)
c) (0, 1, 1), (1, 0, 1), (0, 0, 0)
d) (3, 2, 1), (1, 3, 2), (2, 1, 3)
e) (−5, 3, 7), (4, −2, 1), (0, 6, −2)

10.2 Si supponga che per i triangoli dell'esercizio 10.1 siano stati usati i comandi POLYGON-ABS-3, aventi come argomento i punti nella stessa sequenza dell'esercizio. Usando i parametri di visualizzazione di default, stabilire per ciascun triangolo se è in vista o se è nascosto.

10.3 Indicare per ciascuna delle seguenti coppie di triangoli quale dei seguenti test va applicato:

1. minimax in x e y
2. confronto dei lati
3. contenimento degli estremi di un lato
4. contenimento del punto medio di un triangolo

a) (2, 3, −1), (3, 2, −1), (1, 1, −1)
   e (9, 8, −1), (5, 6, −2), (7, 8, −3)
b) (3, 5, −2), (5, 7, −2), (7, 5, −2)
   e (5, 6, −1), (6, 4, −3), (4, 4, −3)
c) (4, 2, −3), (5, 4, −3), (6, 2, −3)
   e (1, 1, −1), (5, 7, −1), (8, 1, −1)
d) (6, 2, −1), (1, 1, −1), (6, 7, −1)
   e (2, 5, −3), (6, 8, −2), (1, 2, −2)
e) (1, 3, −3), (1, 5, −3), (9, 6, −3)
   e (3, 8, −4), (6, 2, −4), (4, 1, −4)

PROBLEMI DI PROGRAMMAZIONE

10.1 Implementare gli algoritmi dal 10.1 al 10.5 per estendere il sistema grafico con l'eliminazione delle facce nascoste.

***10.2** Implementare gli algoritmi dal 10.6 al 10.35 che realizzano l'algoritmo del pittore.

***10.3** Implementare gli algoritmi dal 10.36 al 10.51 per l'eliminazione delle linee nascoste.

***10.4** Implementare gli algoritmi dal 10.52 al 10.54 per generalizzare il sistema ai poligoni concavi.

10.5 Applicare l'algoritmo BACK-FACE per visualizzare una casetta fatta di poligoni rappresentati mediante il contorno. Scegliere una vista che mostri la facciata, un lato ed il tetto.

10.6 Applicare l'algoritmo del pittore per disegnare più poligoni pieni con diversi tipi di riempimento, sovrapposti in modi differenti.

10.7 Applicare l'algoritmo del pittore per visualizzare due casette composte da poligoni pieni con diversi tipi di riempimento. Scegliere una vista che mostri una casa parzialmente nascosta dall'altra.

10.8 Provare ad eliminare le linee nascoste da una figura che mostri due case, composte da poligoni vuoti e in cui la prima nasconda parzialmente l'altra.

10.9 Provare il trattamento di poligoni concavi, costruendo una serie di poligoni concavi che si sovrappongono e mostrarli sia pieni che vuoti. Cercare di scegliere esempi significativi.

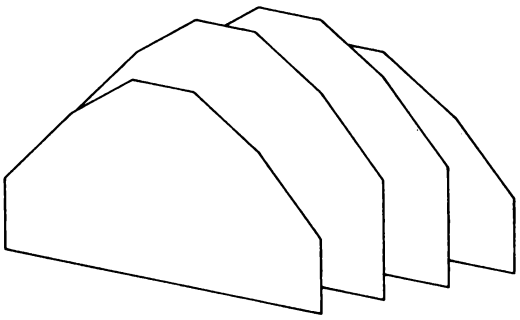
10.10 Scrivere un programma che fornisca più viste successive di un oggetto disegnato dalla procedura FOO all'interno del cubo unitario. Le immagini devono essere quelle viste da un osservatore che sta girando attorno all'oggetto.

***10.11** Le funzioni del tipo $y = F(x, z)$ possono essere visualizzate tramite un certo numero di poligoni delimitati dalla curva ottenuta per un valore fisso di z con x campione e dai contorni. Per esempio, un poligono potrebbe essere definito dai punti $z = 0.5$ e $x = 0.0, 0.2, 0.4, 0.6, 0.8, 1.0$ e dai contorni in $x = 1$, in $y = 0$ e in $x = 0$. In tale modo ciascun poligono rappresenta una «fetta» di funzione. Più fette per diversi valori di z possono fornire l'andamento dell'intera funzione.

a) Scrivere un programma che realizzi quanto è stato esposto per una superficie del tipo

$$y = \sin(\pi * x) + \cos(\pi * z)$$

b) Usare l'algoritmo per la rimozione delle linee nascoste per eliminare le parti non visibili.



c) Scegliere i parametri di visualizzazione per ottenere una soddisfacente rappresentazione della superficie.

11

Ombreggiatura

INTRODUZIONE

Nei capitoli precedenti si è visto come costruire oggetti tridimensionali tramite i poligoni e come proiettare e visualizzare immagini prospettiche di tali oggetti per conferir loro un senso di profondità. Si è visto inoltre come eliminare linee e superfici nascoste per dare alle immagini un maggiore realismo. Spesso è utile disegnare i poligoni con diversi tipi di riempimento dell'area, in modo da ottenere l'effetto di diverse intensità luminose e ombre sulle superfici del poligono. In questo capitolo si affronta il problema dell'ombreggiatura di oggetti tridimensionali. Sarà possibile scegliere automaticamente i diversi tipi di riempimento dei poligoni e fornire ulteriore realismo all'immagine. Verrà sviluppato un modello in grado di simulare il modo in cui le sorgenti luminose illuminano gli oggetti, usato per determinare quanto debba essere luminosa ciascuna faccia di un oggetto.

11.1 ILLUMINAZIONE DIFFUSA

Si può iniziare considerando una sorgente luminosa indiretta: si supponga che l'oggetto sia illuminato da una luce che non proviene da una particolare sorgente, bensì da tutte le direzioni. Si tratta della luce di sfondo, che è riflessa dai muri, dal pavimento e dal soffitto. Si supponga inoltre che questa luce sia uniforme, cioè che abbia la stessa intensità in ogni punto dello spazio considerato. Non ci saranno macchie luminose, né verrà privilegiata alcuna direzione particolare: ci sarà la stessa quantità di luce dall'alto verso il basso e viceversa, così come da sinistra verso destra e viceversa. Una superficie riceverà quindi la stessa quantità di energia luminosa *diffusa*, indipendentemente dal suo orientamento. Parte di questa energia verrà assorbita dalla superficie, mentre il resto

verrà riflesso. Il rapporto tra la luce riflessa dalla superficie e la quantità totale di luce che la investe è chiamato *coefficiente di riflessione*. Una superficie bianca riflette quasi tutta la luce incidente ed il suo coefficiente di riflessione dovrebbe quindi essere prossimo a uno. Una superficie nera, d'altro canto, assorbe la maggior parte della luce incidente e il suo coefficiente è perciò prossimo a zero. Una superficie grigia avrà un coefficiente di riflessione intermedio ai valori precedenti. La maggior parte degli oggetti riflette alcuni colori meglio di altri: si consideri un oggetto che abbia un alto coefficiente di riflessione per la luce rossa ed uno basso per gli altri colori. Illuminando l'oggetto con luce bianca (che è la somma di tutti i colori fondamentali), la porzione rossa di luce sarà riflessa dall'oggetto mentre gli altri colori verranno assorbiti: l'oggetto apparirà rosso. Questo è infatti il meccanismo che conferisce il colore agli oggetti. Un terminale grafico a colori, che è in grado di assegnare a ciascun pixel un valore di intensità per il rosso, il verde e il blu, necessita di tre valori del coefficiente di riflessione — uno per ciascuno dei tre colori — per stabilire le proprietà di ombreggiatura e il colore dell'oggetto. Per semplificare le cose, si considererà un terminale con differenti livelli di grigio ed un solo coefficiente di riflessione (R) che indichi la tonalità di grigio con cui è riempito un oggetto. La quantità di luce che investe l'oggetto non cambia con la sua posizione, poiché la luce proviene uniformemente da tutte le direzioni; ma l'intensità della luce riflessa verso il nostro occhio cambia mutandone l'orientamento? La risposta è che l'oggetto apparirà della stessa luminosità indipendentemente dal suo orientamento. La causa di ciò sta nel reciproco bilanciamento di due effetti: da una parte esiste una maggior emissione di luce in direzione perpendicolare alla superficie, rispetto a quella emessa in direzione parallela. Il rapporto preciso tra queste due quantità è regolato dalla *legge di Lambert*, che afferma che la quantità di luce riflessa da una superficie perfettamente diffondente varia con il coseno dell'angolo formato dalla normale alla superficie con la direzione del raggio riflesso.

Ne segue che la quantità di luce proveniente da un punto sulla superficie dimi-

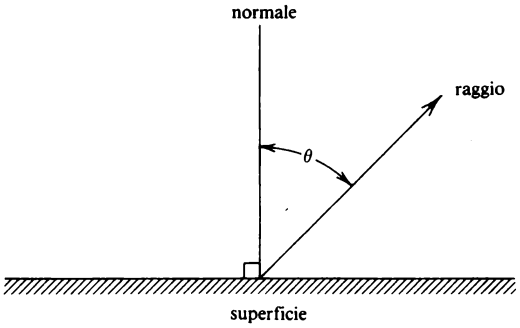


Figura 11.1 La direzione del raggio luminoso è misurata rispetto alla normale alla superficie

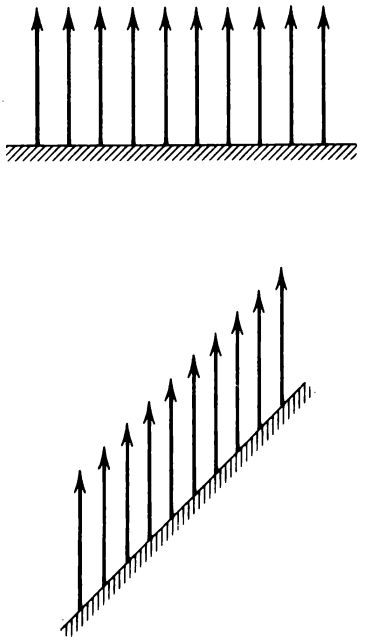
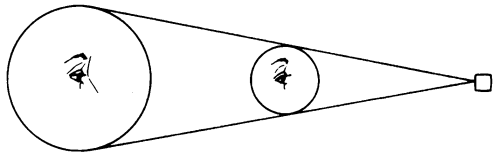


Figura 11.2 Se si inclina la superficie rispetto alla direzione di vista, la luminosità globale non cambia

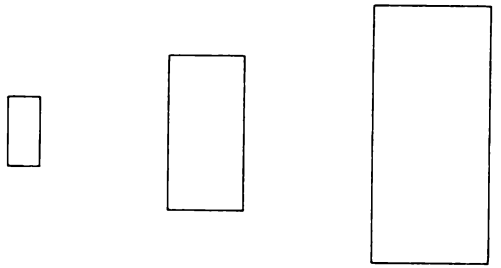
nuirebbe con il coseno dell'angolo al ruotare della superficie rispetto al punto di vista. D'altra parte, poiché l'occhio umano percepisce aree e non punti, si verifica un effetto di compensazione: se la superficie viene ruotata rispetto alla direzione di vista, la densità dei punti di emissione in una prefissata area di vista aumenta in quantità inversamente proporzionale al coseno. Perciò, al ruotare della superficie, viene percepita meno luce proveniente da ciascun punto, ma i punti appaiono molto addensati e quindi, come effetto globale, la luminosità della superficie non cambia (vedi Figura 11.2).

Una compensazione di effetti analoga alla precedente si verifica quando si allontana la superficie dal punto di vista: la luce proveniente dalla superficie si distribuisce su un'area più vasta. Tale area aumenta con il quadrato della distanza e la quantità di luce percepita dall'occhio diminuisce quindi dello stesso fattore. Sempre a causa della distanza fra il punto di vista e la superficie, però, l'oggetto apparirà rimpicciolito dello stesso fattore. Anche se ci sarà meno luce, essa investirà un'area più piccola della retina e la luminosità globale della superficie rimarrà immutata (vedi Figura 11.3).

Si può fornire ora un'espressione della luminosità dell'oggetto determinata da una luce diffusa dallo sfondo. Se B è l'intensità della luce di sfondo e R è il



la quantità di luce dimuinuisce
con l'aumentare della distanza



la dimensione apparente degli oggetti
diminuisce con l'aumentare della distanza

Figura 11.3 Compensazione degli effetti dovuti alla variazione della distanza

coefficiente di riflessione dell'oggetto, l'intensità della luce proveniente da una qualsiasi superficie visibile (che ai fini del sistema grafico può essere anche chiamata «ombreggiatura» della superficie stessa) sarà:

$$\text{SHADE} = BR \tag{11.1}$$

L'utente può creare scene luminose o buie e colorare l'oggetto con diverse tonalità di grigio assegnando diversi valori a B e a R.
Nel semplice modello proposto, ciascun piano di un particolare oggetto sarà ombreggiato allo stesso modo e ciò non corrisponde ad una situazione reale. Per realizzare un modello di ombreggiatura più realistico, si devono considerare anche le sorgenti luminose puntiformi.

11.2 ILLUMINAZIONE DA SORGENTE PUNTIFORME

Le *sorgenti puntiformi* sono un'astrazione delle sorgenti luminose del mondo reale, come ad esempio le lampadine, le candele o il sole. La luce ha origine in un dato punto, proviene da una data direzione percorrendo una data distanza. Nel caso di sorgenti puntiformi, la posizione e l'orientamento della superficie

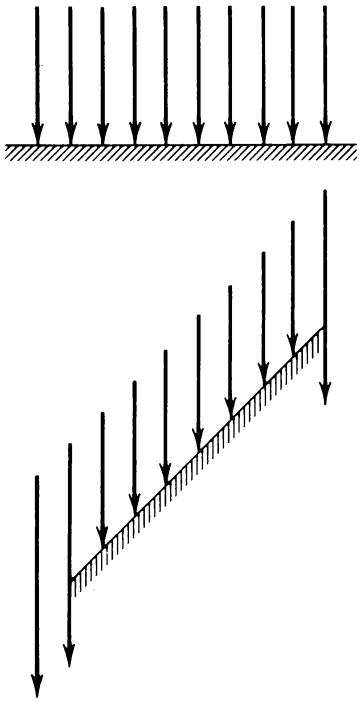


Figura 11.4 Una superficie inclinata rispetto alla direzione dei raggi luminosi riceve meno luce di una non inclinata

dell'oggetto rispetto alla sorgente luminosa determinano la quantità di luce che investe la superficie e di conseguenza la sua luminosità. Le superfici rivolte verso la sorgente luminosa e poste vicino ad essa, riceveranno più luce di quelle diversamente orientate e più lontane. Si considerino dapprima gli effetti dell'orientamento: per le stesse argomentazioni discusse in precedenza, ruotando una faccia rispetto alla sorgente luminosa, l'area della superficie su cui si proietta un fascio luminoso di dimensioni prefissate aumenta. Ciò significa che per ciascun punto della superficie viene messa a disposizione meno luce. La superficie è quindi meno illuminata (vedi Figura 11.4).
L'illuminazione diminuisce proporzionalmente a $\cos(I)$, dove I è l'angolo compreso fra la direzione del fascio luminoso e la normale al piano. L'angolo I viene chiamato *angolo di incidenza* (vedi Figura 11.5).
Dato un vettore L di lunghezza unitaria, diretto verso la sorgente luminosa, ed un vettore N anch'esso di lunghezza unitaria, diretto lungo la normale alla superficie, con verso uscente, il loro prodotto scalare è

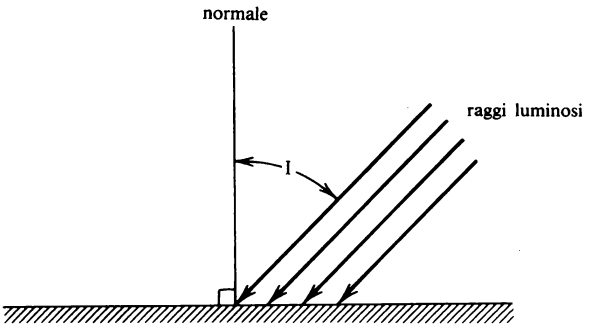


Figura 11.5 Angolo di incidenza

$$\cos I = L \cdot N \tag{11.2}$$

Supponendo che dalla sorgente puntiforme provenga la quantità di luce P, l'ombreggiatura di una superficie dell'oggetto sarà data da

$$\text{SHADE} = BR + P(L \cdot N)R \tag{11.3}$$

Il termine relativo alla sorgente puntiforme è dato dal prodotto della luce da essa proveniente per il coseno dell'angolo di incidenza (ovvero la quantità di luce che investe ciascun punto della superficie) moltiplicato per il coefficiente di riflessione (indicante la quantità di luce che viene riflessa e raggiunge l'osservatore).

11.3 RIFLESSIONE SPECULARE

Il modello proposto può essere ancora perfezionato. Un oggetto è in grado di riflettere la luce in due modi: tramite una riflessione diffusa, della quale si è già parlato, e tramite una riflessione speculare. La riflessione diffusa dipende unicamente dall'angolo di incidenza e la luce riflessa può essere variamente colorata a seconda del coefficiente di riflessione. La riflessione speculare si comporta abbastanza diversamente: è il tipo di riflessione che si verifica sulla superficie di uno specchio. Tutta la luce viene riflessa, non solo alcuni colori, e viene riflessa praticamente in una sola direzione, senza seguire quindi la legge di Lambert. Nel caso della riflessione speculare, la luce investe la superficie e rimbalza via. L'angolo che il raggio riflesso forma con la normale alla superficie è chiamato *angolo di riflessione* O e ha un'ampiezza pari all'angolo di incidenza (vedi Figura 11.6). La luce proveniente da una riflessione speculare può essere percepita quando la direzione della congiungente tra l'oggetto e l'occhio dell'osserva-

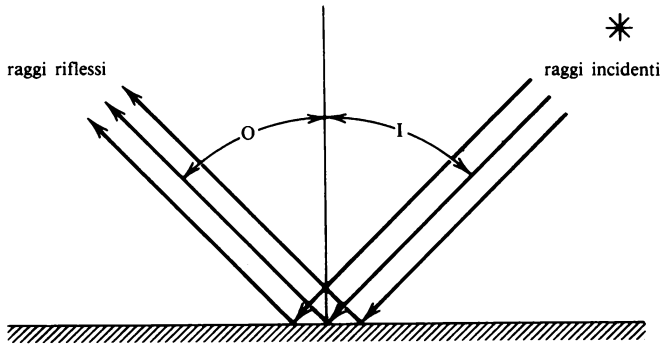


Figura 11.6 Riflessione speculare

tore coincide con la direzione della luce riflessa. Un altro modo di illustrare il concetto precedente consiste nel definire un vettore intermedio fra la direzione della luce incidente e quella di osservazione (vedi Figura 11.7): si confronti questo vettore con la normale alla superficie (che è intermedia tra il raggio incidente e quello riflesso); se le direzioni coincidono, allora l'occhio può percepire i raggi riflessi (vedi Figura 11.8). Se L è un vettore di lunghezza unitaria diretto come il raggio incidente e con verso uscente dalla superficie ed E è un vettore anch'esso unitario diretto verso l'osservatore, segue che

$$H = \frac{L+E}{|L+E|} \tag{11.4}$$

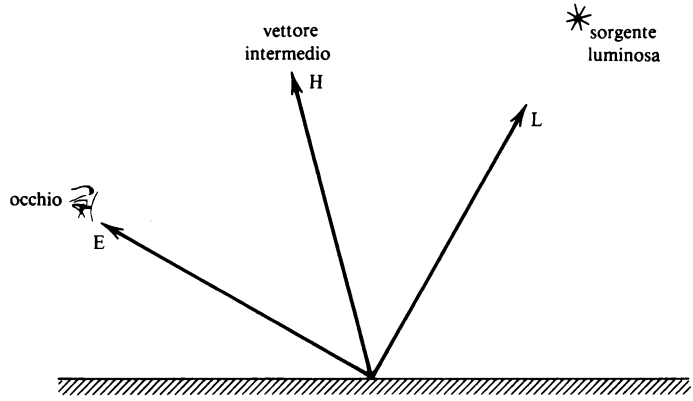


Figura 11.7 Definizione del vettore intermedio tra il raggio incidente e la direzione di vista

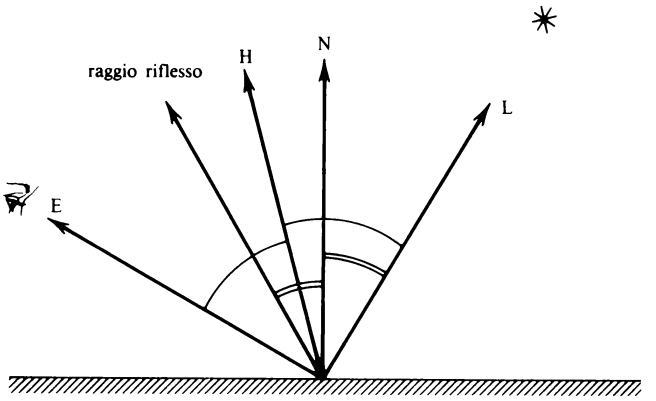


Figura 11.8 Vettori utilizzati per determinare se il raggio riflesso può essere visto

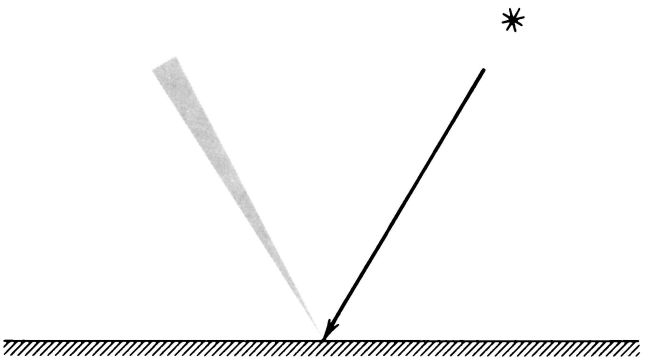
sarà un vettore di lunghezza unitaria (versore) in direzione intermedia ad essi. Se N è il versore normale alla superficie, il prodotto scalare $N \cdot H$ fornirà il coseno dell'angolo fra il versore intermedio ed il versore normale alla superficie. Il raggio riflesso può essere percepito quando l'angolo formato fra questi due versori è prossimo a 0 ed il coseno è quindi prossimo a 1. Per altri valori di tale angolo, la luce non può essere percepita ed il coseno è minore di 1. Sarebbe opportuno disporre di una funzione che assuma il valore 1 quando l'angolo in questione è prossimo a 0, ma che valga 0 per qualsiasi altro angolo. Ciò si può ottenere elevando il coseno a una potenza con esponente elevato:

$$F = (N \cdot H)^a \tag{11.5}$$

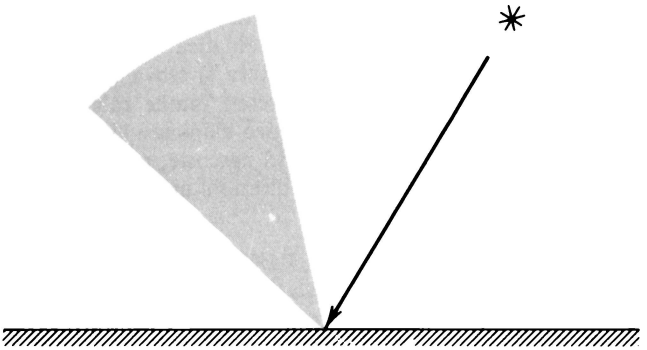
L'esponente a rappresenta il grado di *levigatezza* dell'oggetto: un elevato valore di a fornisce l'effetto di uno specchio perfettamente riflettente, mentre per un valore più piccolo di a l'oggetto appare meno lucido (vedi Figura 11.9). Se S è la quantità di luce riflessa specularmente, la funzione di ombreggiatura diventa

$$\text{SHADE} = BR + P(L \cdot N)R + S(N \cdot H)^a \tag{11.6}$$

La quantità di luce riflessa specularmente può dipendere dall'angolo di incidenza; il parametro S dovrebbe essere, in realtà, funzione di I. Tale funzione può essere complessa e diversa a seconda del materiale. Per semplificare il modello prodotto, comunque, si assume tale funzione costante per qualsiasi oggetto. S tiene anche conto della potenza della sorgente puntiforme P.
Si considerino infine gli effetti della distanza. Teoricamente, l'illuminazione dovuta ad una sorgente puntiforme dovrebbe diminuire col quadrato della sua distanza dall'oggetto illuminato. Tale relazione renderebbe però il modello troppo sensibile alle variazioni della distanza: la discrepanza fra teoria e realtà è do-



a grande:
effetto specchio



a piccolo:
effetto smorzato

Figura 11.9 Effetto dell'esponente *a* dell'equazione (11.5)

vuta, in parte, al fatto che la maggior parte delle sorgenti luminose sono molto più grandi di un punto. Un oggetto illuminato da una sorgente luminosa molto grande e posto vicino ad essa, difficilmente mostrerebbe cambiamenti d'illuminazione in seguito a piccole variazioni della distanza. In ogni caso, occorre una funzione che decresca con la distanza meno rapidamente di una funzione quadratica. Bisogna inoltre porre un limite finito in caso di distanze molto piccole, cosicché non si presenti il problema di una divisione per zero nel caso che la sorgente luminosa non sia correttamente posizionata. La seguente funzione risponde ai requisiti sopra citati:

$$G = \frac{1}{C+D} \tag{11.7}$$

Dove C è una costante e D è la distanza tra la sorgente luminosa e l'oggetto. L'espressione dell'ombreggiatura diventa

$$\text{SHADE} = BR + \frac{[P(L \cdot N)R + S(N \cdot H)^a]}{C+D} \tag{11.8}$$

Generalizzata a più sorgenti luminose puntiformi, essa diventa

$$\text{SHADE} = BR + \sum_j \frac{[P_j(L_j \cdot N)R + S_j(N \cdot H_j)^a]}{C+D_j} \tag{11.9}$$

11.4 ALGORITMI DI OMBREGGIATURA

Disponendo di un modello per ombreggiare gli oggetti, è possibile aggiungere al sistema grafico una procedura di ombreggiatura automatica. Il punto più adatto per introdurre il calcolo relativo all'ombreggiatura si trova alla fine della routine IS-BACK-FACE (vedi Figura 11.10). Sarebbe inutile calcolare l'ombreggiatura di superfici non visibili ed è perciò logico eliminare le facce non in vista. IS-BACK-FACE non solo elimina le superfici nascoste, ma calcola anche il vettore normale alla superficie, che è una delle direzioni utili nelle fasi succes-

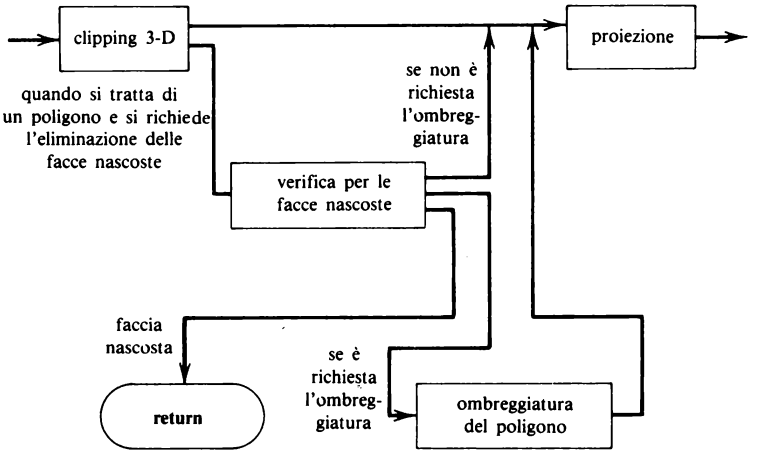


Figura 11.10 Estensione del sistema con l'ombreggiatura

sive. L'algoritmo IS-BACK-FACE va modificato in modo tale che, in presenza di una superficie visibile e se l'utente richiede l'ombreggiatura automatica, chiami la routine di ombreggiatura MAKE-SHADE.

Algoritmo 11.1 IS-BACK-FACE (Algoritmo 10.3 aggiornato)
Assume il valore TRUE se il poligono nel T-buffer rappresenta una faccia non visibile.

Variabili globali XT, YT, ZT←array usati per la memorizzazione nel T-buffer dei vertici del poligono
PERSPECTIVE-FLAG←indicatore del tipo di proiezione
COUNT-OUT←numero dei vertici del poligono
VXP, VYP, VZP←vettore di proiezione parallela
XC, YC, ZC←centro della proiezione prospettica
SHADE-FLAG←indicatore relativo alla richiesta di ombreggiatura automatica

Variabili locali L←vertice posto più a sinistra
LA, LB←vertice che precede e che segue quello posto più a sinistra
SX, SY, SZ←vettore diretto verso l'osservatore
XV1, YV1, ZV1←componenti del primo vettore
XV2, YV2, ZV2←componenti del secondo vettore
RX, RY, RZ←componenti del prodotto vettoriale
DIR←prodotto scalare della normale alla superficie con il vettore proiezione
ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
  IS-BACK-FACE←TRUE;
  trova il vertice più a sinistra
  L←LEFT-MOST(1,COUNT-OUT);
  trova una direzione del vettore di proiezione
  if not PERSPECTIVE-FLAG
    then begin
           SX←VXP;
           SY←VYP;
           SZ←VZP;
           end
    else begin
           SX←XC−XT[L];
           SY←YC−YT[L];
           SZ←ZC−ZT[L];
           end;
  XV1←0;
  YV1←0;
  ZV1←0;
```

```
    trova un vertice che precede quello più a sinistra
    LB←L;
    while|XV1|+|YV1|+|ZV1|<ROUNDOFF
    do begin
       riduzione del contatore del ciclo
       if LB = 1 then LB←COUNT-OUT else LB←LB−1;
       termina se ha considerato tutti i vertici.
       if LB = L then return;
       calcola il vettore
       XV1←XT[L]−XT[LB];
       YV1←YT[L]−YT[LB];
       ZV1←ZT[L]−ZT[LB];
       end;
    LA←L;
    DIR←0;
    trova il vertice che segue quello più a sinistra, che non dia un prodotto
    vettoriale nullo
    while|DIR|<ROUNDOFF
    do begin
          if LA = COUNT-OUT then LA←1 else
             LA←LA+1;
          if LA = LB then return;
          XV2←XT[LA]−XT[L];
          YV2←YT[LA]−YT[L];
          ZV2←ZT[LA]−ZT[L];
          RX←YV1*ZV2−ZV1*YV2;
          RY←ZV1*XV2−XV1*ZV2;
          RZ←XV1*YV2−YV1*XV2;
          DIR←RX*SX+RY*SY+RZ*SZ;
       end;
    controlla che non sia una faccia nascosta
    IS-BACK-FACE←(DIR>0);
    if not IS-BACK-FACE and SHADE-FLAG
    then MAKE-SHADE (COUNT-OUT, −RX, −RY, −RZ);
    return;
end;
```

L'utente può richiedere tramite il flag SHADE-FLAG un'ombreggiatura automatica.

Algoritmo 11.2 SET-SHADING (ON-OFF)
Routine utente che richiede l'ombreggiatura automatica.
Argomenti ON-OFF←scelta dell'utente
Variabili globali SHADE-FLAG←indicatore di ombreggiatura automatica
begin

```
    SHADE-FLAG←ON-OFF;
    return;
end;
```

Le formule di ombreggiatura illustrate usano il prodotto scalare di vettori unitari per la determinazione dei coseni: si dovranno quindi normalizzare spesso i vettori (portandoli a lunghezza unitaria). Sarà conveniente per questo disporre di una procedura di servizio: la routine seguente normalizza un vettore dividendolo per la sua lunghezza; inoltre, viene restituita la lunghezza originale del vettore.

Algoritmo 11.3 UNITY (DX, DY, DZ, LENGTH)
Normalizza un vettore e ne calcola la lunghezza.
Argomenti DX, DY, DZ←vettore da normalizzare
LENGTH←lunghezza del vettore
ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
    LENGTH←SQRT(DX↑2+DY↑2+DZ↑2);
    IF LENGTH<ROUNDOFF THEN RETURN;
    DX←DX/LENGTH;
    DY←DY/LENGTH;
    DZ←DZ/LENGTH;
    return;
end;
```

Un'altra routine indispensabile è quella che fornisce un punto utile a definire la posizione del poligono. Poiché l'ombreggiatura di un poligono riguarda tutta la sua area e non singoli punti interni al suo perimetro, occorre un punto per determinare la direzione che va dal poligono alla sorgente luminosa e dal poligono all'occhio dell'osservatore. Si usa a tale scopo il baricentro del poligono.

Algoritmo 11.4 AVERAGE-POSITION (SIZE, X, Y, Z)
Calcola il baricentro di un poligono.
Argomenti SIZE←numero di vertici del poligono
X, Y, Z←coordinate del baricentro
Variabili globali XT, YT, ZT←array del T-buffer che contengono il poligono

```
begin
    X←0;
    Y←0;
    Z←0;
    for I = 1 to SIZE
    do begin
        X←X+XT[I];
```

```
            Y←Y+YT[I];
            Z←Z+ZT[I];
            end;
        X←X/SIZE;
        Y←Y/SIZE;
        Z←Z/SIZE;
        return;
    end;
```

Si può ora considerare la routine di ombreggiatura. Essa ha come argomenti il numero di lati del poligono e il vettore normale al piano cui esso appartiene. Il vettore normale va normalizzato tramite la routine UNITY. Il numero dei lati del poligono viene passato dalla routine AVERAGE-POSITION, che ha già calcolato le coordinate del punto di riferimento del poligono. La differenza fra la posizione della sorgente luminosa e la posizione del punto di riferimento del poligono definisce un vettore che fornisce la direzione del raggio luminoso. Il calcolo della direzione di osservazione dipende dal tipo di proiezione usata: parallela o prospettica. Per una proiezione parallela si prende il vettore di proiezione, mentre per una proiezione prospettica si prende la direzione di osservazione, che è data dalla differenza fra il centro di proiezione e la posizione del poligono. Dopo che il vettore di illuminazione e quello di osservazione sono stati normalizzati, possono essere sommati per calcolare il vettore intermedio. Si passa quindi alla determinazione dei coseni degli angoli formati fra i vettori sopra definiti. Il coseno dell'angolo di incidenza è il prodotto scalare del vettore normale e del vettore di illuminazione: un risultato negativo significa che la luce è posta dietro la superficie e ciò significa che quella faccia non viene illuminata. Per segnalare questo fatto, può essere usata la funzione MAX, che assume valore uguale a 0 quando la sorgente è posta dietro la superficie. Il prodotto scalare fra il vettore normale e il vettore intermedio determina l'intensità della riflessione speculare, anche in questo caso solo se la sorgente è posta di fronte alla superficie del poligono. Infine, si assegnano tutti i valori dei parametri presenti nella formula di ombreggiatura, per ottenere il numero che caratterizza la «brillantezza» dell'oggetto. Si assume qui che la luminosità dello sfondo B sia compresa fra i valori zero e uno e che la sorgente puntiforme sia

$$P = 1-B \tag{11.10}$$

Il risultato è in seguito modificato da un fattore di brillantezza globale BRIGHT. La quantità di luce proveniente dalla sorgente luminosa puntiforme e non riflessa in modo diffuso è data da $(1-B)(1-R)$ e se il parametro SP viene usato per indicare la porzione di questa luce che viene riflessa specularmente, allora

$$S = (1-B)(1-R)SP \tag{11.11}$$

Dopo aver determinato la brillantezza di un poligono, il risultato viene passato alla routine INTENSITY che seleziona l'appropriato tipo di ombreggiatura all'interno del poligono.

Algoritmo 11.5 MAKE-SHADE (SIDES, XN, YN, ZN)
Calcola ed assegna il tipo di ombreggiatura di un poligono.

Argomenti
SIDES←numero di lati del poligono
XN, YN, ZN←vettore normale al poligono

Variabili globali
XL, YL, ZL←posizione della sorgente luminosa puntiforme
PERSPECTIVE-FLAG←identificatore di proiezione prospettica o parallela
XC, YC, ZC←centro della proiezione prospettica
VXP, VYP, VZP←direzione della proiezione prospettica
B←coefficiente di illuminazione dello sfondo
R←coefficiente di riflessione del poligono
A←levigatezza del poligono
SP←porzione di luce riflessa specularmente
BRIGHT←fattore di scala relativo all'illuminazione globale

Variabili locali
DXL, DYL, DZL←direzione di illuminazione
D←distanza dalla sorgente luminosa
DXE, DYE,DZE←direzione di osservazione
XM, YM, ZM←posizione del poligono
DXH, DYH, DZH←vettore intermedio fra quello di illuminazione e quello di osservazione
COSI←coseno dell'angolo di incidenza
COSN←coseno dell'angolo formato fra il vettore normale ed il vettore intermedio
SHADE-SETTING←valore di ombreggiatura calcolato
DUMMY←variabile fittizia

```
begin
   normalizza il vettore normale alla superficie
   UNITY(XN, YN, ZN, DUMMY);
   calcola la posizione del poligono
   AVERAGE-POSITION(SIDES, XM, YM, ZM);
   calcola gli altri vettori
   DXL←XL−XM;
   DYL←YL−YM;
   DZL←ZL−ZM;
   UNITY(DXL, DYL, DZL, D)
   if PERSPECTIVE-FLAG
   then begin
            DXE←XC−XM;
            DYE←YC−YM;
```

```
            DZE←ZC−ZM;
            end
    else begin
            DXE←VXP;
            DYE←VYP;
            DZE←VZP;
            end;
    UNITY (DXE, DYE, DZE, DUMMY);
    DXH←DXE+DXL;
    DYH←DYE+DYL;
    DZH←DZE+DZL;
    UNITY(DXH, DYH, DZH, DUMMY);
    calcola i coseni
    COSI←MAX(XN*DXL+YN*DYL+ZN*DZL, 0);
    if COSI>0
    then COSN←MAX(XN*DXH+YN*DYH+ZN*DZH, 0)
    else COSN←0;
    calcola il valore di ombreggiatura
    SHADE-SETTING←MIN(1, (B*R+(1−B)*(R*COSI+
                    (1−R)*SP*(COSN↑A))/(1+D))*BRIGHT);
    INTENSITY(SHADE-SETTING);
    return;
end;
```

La routine INTENSITY converte i valori di ombreggiatura nei codici dei tipi di riempimento dell'interno dei poligoni e quindi questa routine può dipendere dal tipo di terminale usato. In particolare, il numero di valori di intensità disponibili e i loro corrispondenti parametri di rappresentazione possono variare a seconda delle implementazioni. L'intervallo dei valori di SHADE-SETTING passati alla routine INTENSITY varia fra 0 e 1. Se il terminale dispone di N codici di intensità si possono generare gli interi da 1 a N con la seguente formula:

```
INT(SHADE-SETTING*(N−1)+1)
```

Si tratta di una trasformazione lineare che ipotizza una variazione lineare dell'intensità fra gli estremi. Ciò non è vero per alcuni terminali ed in alcuni casi si dovrebbero usare formule diverse per compensare l'hardware non lineare. Se il sistema è stato scritto in modo che la tonalità di colorazione 1 è la più scura, la tonalità 2 è più chiara e così via fino alla tonalità N, la formula sopra riportata fornirà gli esatti valori di tonalità. Un caso in cui le intensità possono non corrispondere ai numeri di tonalità interna è quello in cui l'output viene visualizzato su una stampante e vengono usati caratteri diversi per simulare le differenti intensità. Quando la relazione fra il numero delle tonalità e l'intensità è complessa, si può usare un array per la conversione. Il codice di intensità più scuro è memorizzato nella prima cella dell'array, il seguente nella seconda e

così via, cosicché il codice di più alta intensità viene a trovarsi nella N-esima cella. Allora, usando la formula sopra indicata per determinare quale cella esaminare, si estrae da quella cella il corretto valore di intensità. Per cambiare la tonalità si pone l'istruzione relativa alla nuova tonalità nel display file, ricordando che da quel punto essa viene considerata come tonalità di colorazione corrente. Un esempio della routine INTENSITY, che può essere usata per una stampante, è la seguente:

Algoritmo 11.6 INTENSITY (SHADE-SETTING)
Esempio di routine per acquisire i caratteri di ombreggiatura.
Argomenti SHADE-SETTING←valore di ombreggiatura del poligono
Variabili globali SHADE-STYLE←array di 20 caratteri per l'ombreggiatura
CURRENT-FILL-STYLE←valore corrente dello stile
Variabili locali SHADE-INDEX←indice dello stile scelto per ombreggiare STYLE←stile scelto

```
begin
   SHADE-INDEX←INT(19*SHADE-SETTING+1);
   STYLE←SHADE-STYLE[SHADE-INDEX];
   PUT-POINT(STYLE);
   CURRENT-FILL-STYLE←STYLE;
   return;
end;
```

Per caratterizzare la sorgente luminosa e l'oggetto illuminato, è stato usato un certo numero di parametri; occorre naturalmente a questo punto fornire all'utente i mezzi per scegliere i valori da assegnare a tali parametri. Si scriveranno quindi alcune routine atte a convertire le specifiche dell'utente negli opportuni valori delle variabili globali. Gli oggetti saranno caratterizzati dal loro coefficiente di riflessione (R), dalla porzione di luce interessata dalla riflessione speculare (SP) e dalla loro levigatezza (A). La routine seguente assegna i valori ai parametri dell'oggetto.

Algoritmo 11.7 SET-OBJECT-SHADE (REFLECTIVITY, SHINE, GLOSS)
Routine utente che assegna all'oggetto i parametri di ombreggiatura.
Argomenti REFLECTIVITY←coefficiente di riflessione dell'oggetto
SHINE←porzione di luce interessata alla riflessione speculare
GLOSS←ampiezza dei raggi riflessi
Variabili globali R, SP, A←memorizzazione globale dei parametri dell'oggetto

```
begin
   R←REFLECTIVITY;
```

```
    SP←SHINE;
    A←GLOSS;
    return;
end;
```

Si è ammesso che i parametri dell'oggetto possano essere modificati in qualsiasi momento in modo tale che, una volta costruita la scena, i diversi oggetti possano assumere differenti aspetti. Si vuole però poterli anche manipolare in modo diverso. Le variazioni di luminosità saranno considerate parte delle specifiche di visualizzazione e, come gli altri parametri di visualizzazione discussi nei Capitoli 8 e 9, i parametri di illuminazione andrebbero fissati tramite il segmento del display file. A questo scopo si disporrà di due serie di parametri: una che l'utente può variare a piacere, l'altra che è formata dalle specifiche dell'utente, aggiornate ogniqualvolta venga creato un segmento di display file. Nei calcoli di ombreggiatura viene usata questa seconda serie di parametri. La routine che segue acquisisce le specifiche dell'utente.

Algoritmo 11.8 SET-LIGHT (X, Y, Z, BRIGHTNESS, BACKGROUND)
Routine utente che acquisisce i parametri di illuminazione.

Argomenti X, Y, Z←posizione della sorgente luminosa
BRIGHTNESS←fattore di scala dell'intensità globale
BACKGROUND←porzione di luce diffusa

Variabili globali XL1, YL1, ZL1, BRT, BKG←specifiche dell'utente

```
begin
    XL1←X;
    YL1←Y:
    ZL1←Z;
    BRT←BRIGHTNESS;
    BKG←MAX(0, MIN(1, BACKGROUND));
    return;
end;
```

Quando viene creato un segmento di display file, si copieranno i parametri di illuminazione attivi nelle variabili globali che verranno usate nel calcolo dell'ombreggiatura. Si convertirà inoltre la posizione della sorgente luminosa nel sistema di coordinate del piano di proiezione. La routine COPY-LIGHT svolge queste operazioni.

Algoritmo 11.9 COPY-LIGHT
Aggiorna i parametri di illuminazione attivati.

Variabili globali XL1, YL1, ZL1←posizione della sorgente luminosa
BRT←valore corrente della lucentezza
BKG←valore corrente dell'illuminazione di sfondo
XL, YL, ZL←posizione di illuminazione per il segmento di display file

```
                    BRIGHT←lucentezza per il segmento di display file
                    B←illuminazione di sfondo per il segmento di display file
begin
    BRIGHT←BRT;
    B←BKG;
    XL←XL1;
    YL←YL1;
    ZL←ZL1;
    VIEW-PLANE-TRANSFORM(XL, YL, ZL);
    return;
end;
```

La routine COPY-LIGHT andrebbe chiamata nella fase di assegnazione dei parametri di visualizzazione e quindi la routine NEW-VIEW-3 va modificata nel seguente modo.

Algoritmo 11.10 NEW-VIEW-3 (Algorimo 9.5 aggiornato)
Crea una nuova trasformazione di visualizzazione globale.

```
begin
    NEW-VIEW-2;
    MAKE-VIEWING-TRANSFORM;
    MAKE-CLIPPING-CONSTANTS;
    MAKE-CLIPPING-TESTS;
    COPY-LIGHT;
    return;
begin
```

A completamento dell'algoritmo di ombreggiatura, vanno inizializzati tutti i parametri generati.

Algoritmo 11.11 INITIALIZE-11

```
begin
    SET-LIGHT(0, 0, 1, 1, 0);
    INITIALIZE-10;
    SET-OBJECT-SHADE(1, 0, 40);
    SET-SHADING(FALSE);
    return;
end;
```

La sorgente luminosa è inizialmente posta in posizione (0, 0, 1) con luminosità uguale a 1. I valori di luminosità variano normalmente fra 1 e 2 ma, per compensare la distanza dalla sorgente luminosa o un basso coefficiente di riflessione di un oggetto, questo valore può essere aumentato. L'illuminazione dello sfondo è posta inizialmente a 0. In realtà, il parametro varierebbe fra 0 (assenza di luce indiretta) e 1 (illuminazione totalmente indiretta). Gli oggetti vengo-

no inizialmente considerati completamente riflettenti ma senza alcuna riflessione di tipo speculare. Ciò significa che l'esponente di levigatezza che compare nell'espressione della riflessione speculare (11.5) viene inizialmente posto uguale a 40, essendo il suo intervallo di definizione compreso fra i valori 40 e 60.

11.5 TRASPARENZA E OMBRE

La *trasparenza* e le *ombre* non sono state considerate nel modello di ombreggiatura proposto. L'effetto di trasparenza può essere ottenuto mostrando parte della luce proveniente da un oggetto «nascosto». Più grande è la quantità di luce in grado di attraversare l'oggetto, più l'oggetto è considerato trasparente. Per poter considerare anche oggetti trasparenti, occorrerà modificare l'algoritmo del pittore in modo tale che invece di ricoprire gli oggetti precedentemente disegnati riassegnandone i valori dei pixel, si riduca la loro intensità, che andrà poi sommata ai valori di luminosità dovuti all'oggetto in primo piano.
Il problema delle ombre è molto simile al problema delle superfici nascoste, poiché un oggetto in ombra si può considerare come nascosto rispetto alla sorgente luminosa. Per ottenere tempi di calcolo accettabili, le procedure relative ad una superficie nascosta e ad un'ombra vengono condotte contemporaneamente. Sfortunatamente, occorre un algoritmo per l'eliminazione delle superfici nascoste diverso dall'algoritmo del pittore, poiché nel caso delle ombre è necessario definire le parti nascoste di una superficie (e non solo mascherarle).

ESERCIZI

11.1 Dato il rettangolo di vertici (0, 0, −1), (0, 1, −1), (1, 1, 0), (1, 0, 0), si supponga di volerlo visualizzare tramite una proiezione parallela. Calcolare i valori di ombreggiatura relativi a ciascuna delle seguenti assegnazioni dei parametri.

a) Di default
b) SET-OBJECT-SHADE(0.4, 0, 40)
   SET-LIGHT(0.5, 0.5, 1, 1, 0)
c) SET-OBJECT-SHADE(0.4, 0, 40)
   SET-LIGHT(0.5 ,0.5, 1, 1, 1)
d) SET-OBJECT-SHADE(0.4, 0.5, 40)
   SET-LIGHT(0.5, 0.5, 1, 1, 0)
e) SET-OBJECT-SHADE(0.4 ,0.5, 40)
   SET-LIGHT(−1, 0.5, −0.5, 1, 0)

11.2 Stabilire il significato di ciascuno dei simboli che compaiono nell'equazione (11.8).

PROBLEMI DI PROGRAMMAZIONE

11.1 Implementare gli algoritmi dall'11.1 all'11.10 compreso per aggiungere al sistema grafico l'ombreggiatura automatica.

11.2 Provare le routine di ombreggiatura, disegnando una casetta, scegliendo una vista che mostri almeno tre superfici e generando un certo numero di immagini tutte diverse tra loro tramite il cambiamento di un parametro di ombreggiatura per volta, valutando l'effetto prodotto.

11.3 Scrivere le routine:

INQUIRE-SHADING(ON-OFF)
INQUIRE-LIGHT(X, Y, Z, BRIGHTNESS, BACKGROUND)
INQUIRE-OBJECT-SHADE(REFLECTIVITY, SHINE, GLOSS)

che ritornano all'utente il valore dei parametri di ombreggiatura.

11.4 Estendere il problema di programmazione 10.10, che permette successive viste di un oggetto, per mostrare l'oggetto stesso ombreggiato.

11.5 Scrivere un programma per ottenere successive immagini ombreggiate di un oggetto contenuto all'interno di un cubo unitario tramite la procedura FOO. Si usino i parametri di visualizzazione di default, ma nelle successive immagini dovrà sembrare che la sorgente luminosa ruoti intorno all'oggetto.

11.6 Si estenda il problema di programmazione 9.5, aggiungendo l'assegnazione dei parametri relativi alla sorgente luminosa.

12
Le curve

INTRODUZIONE

In questo testo, finora, sono state trattate esclusivamente linee rette che, come si è visto nel Capitolo 1, sono facili da manipolare e visualizzare in quanto hanno una forma matematica semplice che permette di effettuare agevolmente operazioni di trasformazione e di clipping. La realtà tuttavia non è costituita da figure composte da sole linee rette; molti degli oggetti che si vorrebbero rappresentare con un modello e visualizzare su uno schermo coinvolgono delle curve, come ad esempio una funzione matematica, la forma di un missile o il viso di una donna.
In questo capitolo verranno introdotti alcuni metodi che permettono di generare vari tipi di curve; essi saranno presentati in modo molto semplice, dal momento che una trattazione rigorosa non rientra negli scopi di questo libro; si svilupperanno tuttavia gli algoritmi di interpolazione di tipo *spline* per la generazione di curve.

12.1 GENERAZIONE DI CURVE

Esistono due diversi approcci per la generazione e la rappresentazione di linee curve. Il primo approccio consiste nello sviluppare algoritmi simili a quello della routine DDA per generare curve «vere», cioè rappresentate da un'equazione matematica. Il secondo tipo consiste nel generare la curva approssimandola con piccoli segmenti rettilinei, dopo aver usato le tecniche di interpolazione; quest'ultimo verrà considerato nel prossimo paragrafo.
Si supponga di avere un arco di circonferenza e di volerlo disegnare. Conoscendo le equazioni della curva, è possibile scrivere un algoritmo che calcola le coordinate dei punti sulla curva; cambiando le intensità dei pixel che contengo-

no questi punti, si ottiene sullo schermo la sua rappresentazione. Anche se il calcolo è richiesto più volte, l'idea di base è quella già vista nel Capitolo 1 nell'algoritmo DDA per la generazione di rette.
Le equazioni per una curva semplice come un cerchio o un'ellisse sono facili da risolvere e l'algoritmo per la loro visualizzazione può essere implementato nell'hardware della macchina, cosicché si potrà disporre di dispositivi in grado di disegnare archi di cerchio o ellissi con la stessa facilità e velocità con cui si possono disegnare i segmenti di retta. Per la definizione dell'algoritmo DDA applicato alla generazione di archi di circonferenza, occorrono innanzitutto le equazioni dell'arco, che possono essere scritte in forma parametrica rispetto ad un parametro angolare A come segue:

$$\begin{aligned} x &= R\cos A + x_0 \\ y &= R\sin A + y_0 \end{aligned} \tag{12.1}$$

dove (x_0, y_0) sono le coordinate del *centro di curvatura* e R è il *raggio* (vedi Figura 12.1).
Differenziando e cambiando x in dx, y in dy e A in dA si ottiene

$$\begin{aligned} dx &= -R\sin A\, dA \\ dy &= R\cos A\, dA \end{aligned} \tag{12.2}$$

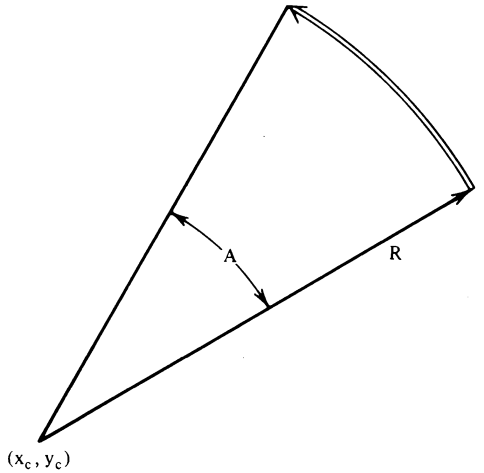


Figura 12.1 Parametri di definizione di un arco di cerchio

Poiché dall'equazione (12.1) si ottiene

$$\begin{aligned} R\cos A &= x - x_0 \\ R\sin A &= y - y_0 \end{aligned} \tag{12.3}$$

sarà

$$\begin{aligned} dx &= -(y - y_0)\,dA \\ dy &= (x - x_0)\,dA \end{aligned} \tag{12.4}$$

Le variazioni dx e dy indicano di quanto incrementare il vecchio punto per ottenere il nuovo.

$$\begin{aligned} x_2 &= x_1 + dx = x_1 - (y_1 - y_0)\,dA \\ y_2 &= y_1 + dy = y_1 + (x_2 - x_0)\,dA \end{aligned} \tag{12.5}$$

Queste equazioni costituiscono le operazioni elementari di un algoritmo per la generazione di archi, in cui i dati di ingresso sono il centro di curvatura (x_0, y_0), l'angolo A, un punto iniziale dell'arco (x, y) e l'intensità luminosa da attribuire ad ogni pixel. Si osservi che per avere una rappresentazione continua, il passo costituito dal parametro DA dovrà essere sufficientemente piccolo.

Algoritmo 12.1 ARCDDA (X0,Y0, A, X, Y, INTENSITY)
Generazione di un arco di cerchio a partire dal punto (X, Y), in senso antiorario.

Argomenti	X0, Y0←centro di curvatura
	A←angolo dell'arco
	X, Y←punto da cui iniziare a disegnare
Variabili globali	FRAME←array del frame buffer
Variabili locali	XARC, YARC←coordinate del successivo punto da disegnare
	A-DRAWN←angolo corrispondente alla parte di arco già disegnato
	DA←incremento angolare
Costanti	ROUNDOFF←numero piccolo maggiore del più grande errore di arrotondamento

```
begin
  if |X-X0| + |Y-Y0| < ROUNDOFF then return;
  A-DRAWN←0;
  ricerca un incremento angolare che permetta di avere una rappresenta-
  zione continua dell'arco
  DA←MIN(0.01, 1/(3.2*(|X-X0| + |Y-Y0|)));
  stabilisce il primo estremo dell'arco
  XARC←X;
  YARC←Y;
  genera l'arco fino al valore angolare desiderato
```

```
    while A-DRAWN<A
    do begin
          cerca un nuovo punto
          XARC←XARC+(Y0−YARC)*DA;
          YARC←YARC+(XARC−X0)*DA;
          A-DRAWN←A-DRAWN+DA;
          cambia intensità ai pixel corrispondenti
          FRAME[INT(XARC), INT(YARC)]←INTENSITY;
          end;
    return;
end;
```

Per definire un tratto di curva, occorrono maggiori informazioni oltre ai punti estremi del segmento; ciò implica l'uso di una differente struttura per il display file. Un altro problema è l'applicazione delle trasformazioni geometriche: infatti, se ad un segmento di retta viene applicata una riduzione di scala, questo rimane ancora un segmento, mentre altre curve si comportano differentemente; ad esempio un cerchio che venga deformato in una sola direzione degenera in un'ellisse. Se si ha a disposizione un algoritmo che genera archi di circonferenza nel modo descritto, non è possibile utilizzare l'algoritmo di clipping, che per il momento è applicabile solo ai segmenti. Volendo eseguire il clipping di archi occorrerà sviluppare un nuovo algoritmo. Questi problemi non sono insormontabili, ma ne esistono altri che impediscono di ottenere dei risultati soddisfacenti con la rappresentazione delle curve sotto forma matematica; infatti le curve che rappresentano gli oggetti che si vogliono disegnare in generale non sono costituite da soli archi di circonferenza: le curve di un profilo alare in un aereo, il tettuccio di una macchina sportiva, il viso di una donna non sono rappresentabili con circonferenze o ellissi. Bisognerà approssimarle unendo assieme piccoli archi di circonferenza, di ellissi o di spirali; ma a questo punto si potrebbero unire assieme tanti piccoli segmenti di retta, con il vantaggio di poter utilizzare il display file, le trasformazioni e il clipping così come sono stati presentati fino ad ora.

12.2 INTERPOLAZIONE

Come è possibile rappresentare una curva che non ha una descrizione matematica semplice? Si può ottenere una sua approssimazione inserendo in un array una serie di punti ad essa appartenenti, mediante i quali sarà possibile valutare come dovrà apparire la curva approssimante. Se la curva gode di certe proprietà di continuità e i punti sono abbastanza vicini fra loro, si è in grado di stimare l'andamento di una curva interpolante che li congiunge. La curva interpolante probabilmente non sarà esatta, ma sarà abbastanza prossima alla curva reale. Le porzioni di curva mancanti, comprese tra due punti dati, verranno rimpiaz-

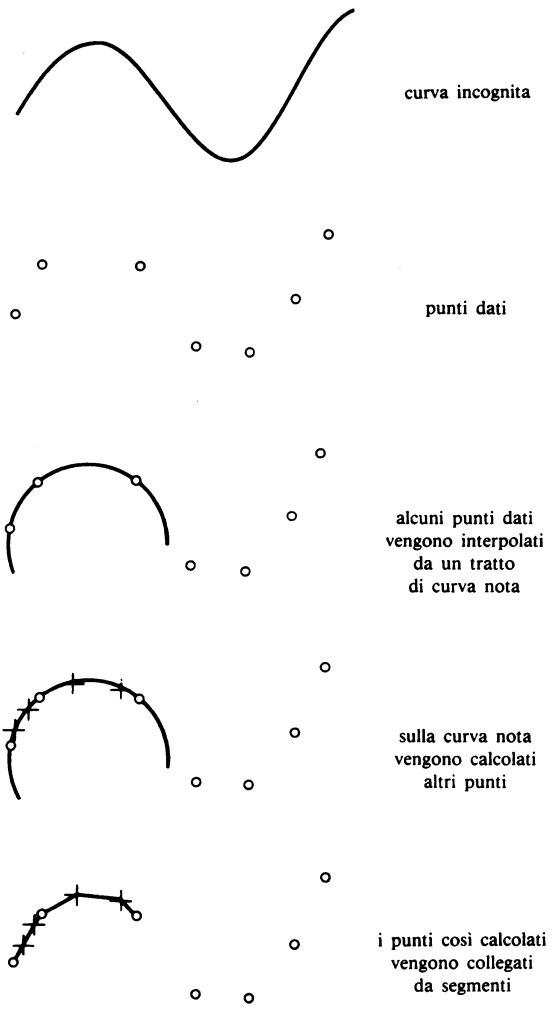


Figura 12.2 Procedimento d'interpolazione

zate con tratti di curve di cui si conosce l'equazione matematica; poiché le due curve condividono gli stessi punti, in una regione piccola, l'approssimazione sarà buona. Per riempire la discontinuità fra i punti dati, si calcolano le coordinate di alcuni punti lungo la curva nota, unendoli tramite dei segmenti (vedi Figura 12.2).

A questo punto, è necessario scegliere il tipo di curva da usare per l'interpola-

zione. Esistono parecchie forme di funzioni che, tramite il controllo di alcuni parametri, possono essere fatte passare per dei punti dati. A questo scopo sono state usate funzioni polinomiali, trigonometriche ed esponenziali e si conoscono diversi metodi possibili per ciascuna di tali classi. Di seguito si applicheranno i metodi relativi alle funzioni polinomiali, del tipo $y = f(x)$, preferendo l'espressione in *forma parametrica* anziché quella esplicita.

$$\begin{aligned} x &= fx(u) \\ y &= fy(u) \\ z &= fz(u) \end{aligned} \tag{12.6}$$

Questa scelta è dovuta ad almeno due valide ragioni: per prima cosa, il passaggio dalle due alle tre dimensioni implica solo l'aggiunta di una terza equazione per la coordinata z e inoltre la forma parametrica permette di esprimere curve a più valori, che non corrispondono cioè alla definizione di funzione matematica: esse possono avere più valori di y per un dato x, in modo che una curva possa riavvolgersi su se stessa e persino autointersecarsi (vedi Figura 12.3).
Bisogna ora definire una funzione che possa essere usata per l'interpolazione di un insieme di punti; in questa prima approssimazione, sarà possibile commettere qualche errore, ma ciò che importa è illustrare la metodologia.
Si supponga di voler far passare una curva polinomiale per n punti dati:

$$(x_1, y_1, z_1),(x_2, y_2, z_2),\ldots, (x_n, y_n, z_n)$$

La funzione verrà costruita come somma di termini, ciascuno relativo ad un punto.

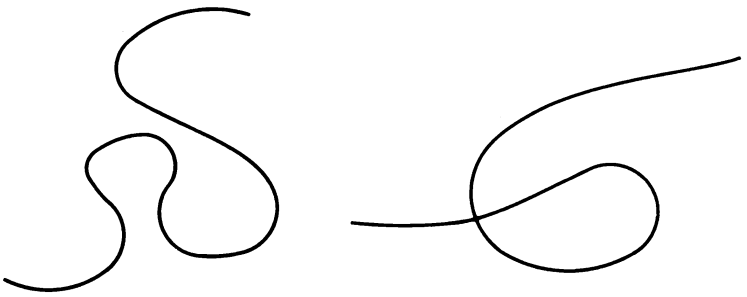


Figura 12.3 La forma parametrica consente di rappresentare curve che si riavvolgono su loro stesse o che si autointersecano

$$fx(u) = \sum_{i=1}^{n} x_i B_i(u)$$

$$fy(u) = \sum_{i=1}^{n} y_i B_i(u) \qquad (12.7)$$

$$fz(u) = \sum_{i=1}^{n} z_i B_i(u)$$

Le funzioni approssimanti $B_i(u)$ sono dette *blending function*; per ciascun valore di u (che rappresenta il parametro «corrente» lungo la curva), determinano l'influenza dell'i-esimo punto sulla curva. Si immagini che il punto «attragga» verso di sé la curva: la funzione $B_i(u)$ esprime la forza con cui il punto sta tirando. Se per un qualsiasi valore di u e per ogni $j \neq i$ si ha che $B_j(u) = 0$ e $B_i(u) = 1$, allora l'i-esimo punto ha il completo controllo della curva e la curva passerà per l'i-esimo punto. Ciò di cui si ha bisogno sono delle blending function che, per diversi valori di u, assegnino di volta in volta il controllo della curva a ciascuno dei punti dati. Non è importante conoscere i valori del parametro u per i quali i punti dati appartengano alla curva, come non lo è il fatto che i punti assumano il controllo della curva in ordine prefissato.
Per questo si definiscono delle blending function per le quali il primo punto dato (x_1, y_1, z_1) assuma il controllo quando $u = -1$, il secondo per $u = 0$ ed il terzo per $u = 1$ e così via, in modo da rendere più semplice l'implementazione: con tale scelta occorre una funzione $B_1(u)$ che valga 1 per $u = -1$ e 0 per $u = 0, 1, 2, \ldots, n-2$.
Un'espressione che soddisfa a questi requisiti è la seguente

$$u(u-1)(u-2)\ldots(u-(n-2)) \qquad (12.8)$$

Per $u = -1$ risulta:

$$(-1)(-2)(-3)\ldots(1-n)$$

Dividendo la (12.8) per questa costante, essa vale 1 per $u = -1$, come si voleva, per cui

$$B_1(u) = \frac{u(u-1)(u-2)\ldots(u-(n-2))}{(-1)(-2)(-3)\ldots(1-n)} \qquad (12.9)$$

Lo stesso accade per la i-esima blending function, che varrà 1 per $u = i-2$ e 0 negli altri casi.

$$B_i(u) = \frac{(u+1)(u)(u-1)\ldots(u-(i-3))(u-(i-1))\ldots(u-(i-2))}{(i-1)(i-2)(i-3)\ldots(1)(-1)\ldots(i-n)} \qquad (12.10)$$

Si consideri il caso in cui si disponga di quattro punti noti, per interpolare i quali occorreranno quattro blending function. La (12.9) fornisce le seguenti:

$$\begin{aligned} B_1(u) &= \frac{u(u-1)(u-2)}{(-1)(-2)(-3)} \\ B_2(u) &= \frac{(u+1)(u-1)(u-2)}{(1)(-1)(-2)} \\ B_3(u) &= \frac{(u+1)u(u-2)}{(2)(1)(-1)} \\ B_4(u) &= \frac{(u+1)u(u-1)}{(3)(2)(1)} \end{aligned} \tag{12.11}$$

Usando tali funzioni e i quattro punti, è possibile costruire una curva che li interpoli:

$$\begin{aligned} x &= x_1 B_1(u) + x_2 B_2(u) + x_3 B_3(u) + x_4 B_4(u) \\ y &= y_1 B_1(u) + y_2 B_2(u) + y_3 B_3(u) + y_4 B_4(u) \\ z &= z_1 B_1(u) + z_2 B_2(u) + z_3 B_3(u) + z_4 B_4(u) \end{aligned} \tag{12.12}$$

Questa curva giacerà piuttosto vicina alla curva che si vuole realmente rappresentare con i quattro punti dati. L'approssimazione sarà migliore fra il secondo ed il terzo punto, tratto in cui u varia fra 0 e 1.

Si possono calcolare alcuni punti sulla curva interpolante con valori di u compresi fra 0 e 1, che potranno essere usati come estremi dei segmenti che costituiscono l'approssimazione della curva di interpolazione. Per esempio, si supponga di voler approssimare la curva fra il secondo e il terzo punto con tre segmenti. Si conoscono le coordinate di questi due punti per $u = 0$ e $u = 1$ e quindi si possono applicare le equazioni per calcolare i punti corrispondenti a $u = 1/3$ e ad $u = 2/3$. In questo modo si ottengono i quattro punti estremi dei tre segmenti approssimanti. Tutta la curva può essere trasformata in una linea spezzata ripetendo questo procedimento. Si prendano quattro punti consecutivi (1, 2, 3, 4) e si costruisce la curva approssimante fra i due punti centrali (2, 3). Quindi si aggiunge un nuovo punto al di là di uno degli estremi noti e si elimina quello che corrisponde all'altro estremo, ottenendo la sequenza (2, 3, 4, 5). A questo punto si può approssimare il tratto di curva compreso tra gli estremi (3, 4) e si continua a iterare questo procedimento muovendosi su tutti i punti dati, finché la curva non sarà disegnata completamente. Il tratto iniziale e quello finale richiedono un trattamento particolare. Per i primi quattro punti (1, 2, 3, 4) si disegna il tratto di curva compreso tra i punti (1, 2) assegnando al parametro u dei valori appartenenti all'intervallo $[-1, 0]$. Analogamente, la blending function per l'ultimo tratto di curva dovrà essere calcolata per valori di u compresi nell'intervallo [1, 2]. Questi casi speciali non comportano particolari difficoltà, ma vanno trattati separatamente.

12.3 ALGORITMI DI INTERPOLAZIONE

Prima di trattare nei particolari le blending function e di presentarne una formulazione migliore, si consideri come potrebbe essere scritto un programma di interpolazione. La prima cosa da notare è che, per ciascun tratto della curva da disegnare, occorre che i valori dei coefficienti della blending function siano i medesimi. Se ciascun tratto viene approssimato con tre segmenti, occorrerà disporre dei coefficienti delle blending function per $u = 0$, $u = 1/3$, $u = 2/3$ e $u = 1$ (vedi Figura 12.4). Tali valori possono essere calcolati e memorizzati in un array che verrà usato per disegnare ciascun tratto di curva.
Il primo algoritmo che verrà presentato consente all'utente di specificare il numero di segmenti che devono essere usati per rappresentare un tratto di curva. Questo numero (NUMBER-OF-LINES) indicherà i valori di u per i quali calcolare e memorizzare le blending function. Il valore di $B_i(u)$ viene memorizzato nell'array BLEND, nella posizione BLEND[I, U]. I valori delle blending function necessari per il primo e l'ultimo tratto della curva vengono calcolati e inseriti negli array FIRST-BLEND e LAST-BLEND.

Algoritmo 12.2 SET-SMOOTH (NUMBER-OF-LINES)
Routine utente che assegna il numero di segmenti con cui approssimare un tratto di curva.

Argomenti	NUMBER-OF-LINES←numero di segmenti per tratto di curva
Variabili globali	LINES-PER-SECTION←contiene il valore di NUMBER-OF-LINES
	BLEND, FIRST-BLEND, LAST-BLEND←array di dimensione 4×MAX-NUMBER-OF-LINES, contenenti i valori delle blending function
Variabili locali	U, V, W←parametri per la valutazione delle blending function

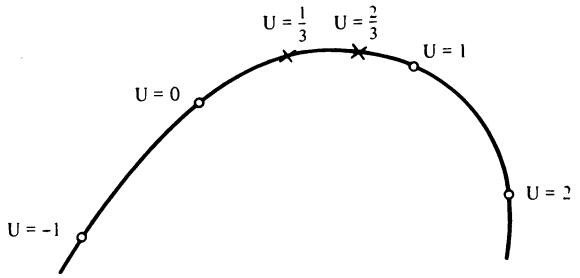


Figura 12.4 Calcolo dei punti necessari per approssimare la curva in corrispondenza di valori del parametro u compresi fra 0 e 1

```
                    I←indice per i punti necessari per definire la curva
Costanti            MAX-NUMBER-OF-LINES←massimo numero di segmenti consentito per ogni tratto di curva
begin
  if NUMBER-OF-LINES<1
  or NUMBER-OF-LINES>MAX-NUMBER-OF-LINES
  then return-error 'NUMERO DI SEGMENTI NON VALIDO';
  LINES-PER-SECTION←NUMBER-OF-LINES;
  for I = 1 to NUMBER-OF-LINES
  do begin
    valore della blending function per il tratto intermedio
      U←I/NUMBER-OF-LINES;
      BLEND[1, I]←U*(U-1)*(U-2)/(-6);
      BLEND[2, I]←(U+1)*(U-1)*(U-2)/2;
      BLEND[3, I]←(U+1)*U*(U-2)/(-2);
      BLEND[4, I]←(U+1)*U*(U-1)/6;
      V←U-1;
    valore della blending function per il primo tratto
      FIRST-BLEND[1, I]←V*(V-1)*(V-2)/(-6);
      FIRST-BLEND[2, I]←(V+1)*(V-1)*(V-2)/2;
      FIRST-BLEND[3, I]←(V+1)*V*(V-2)/(-2);
      FIRST-BLEND[4, I]←(V+1)*V*(V-1)/6;
      W←U+1;
    valore della blending function per l'ultimo tratto
      LAST-BLEND[1, I]←W*(W-1)*(W-2)/(-6);
      LAST-BLEND[2, I]←(W+1)*(W-1)*(W-2)/2;
      LAST-BLEND[3, I]←(W+1)*W*(W-2)/(-2);
      LAST-BLEND[4, I]←(W+1)*W*(W-1)/6;
      end;
  return;
end;
```

La prossima routine esegue il prodotto tra i punti assegnati e le blending function, ottenendo i punti sulla curva approssimata, che vengono uniti con una spezzata; gli array XSM, YSM, ZSM conterranno le coordinate di questi punti, mentre i valori delle blending function sono passati alla procedura come argomenti.

Algoritmo 12.3 MAKE-CURVE (B)
Traccia un tratto di curva.

Argomenti B←array di dimensione 4×LINES-PER-SECTION contenente i coefficienti delle blending function

Variabili globali LINES-PER-SECTION←numero di segmenti per tratto di curva
XSM, YSM, ZSM←array di 4 elementi contenenti i punti

```
                    dati
Variabili locali    I←indice dei punti dati
                    J←contatore del numero di segmenti del tratto di curva
                    X, Y, Z←coordinate di un punto sulla curva approssi-
                      mante
begin
   for J = 1 to LINES-PER-SECTION
   do begin
      X←0;
      Y←0;
      Z←0;
      somma il contributo di ciascun punto
      for I = 1 to 4
      do begin
            X←X+XSM[I]*B[I, J];
            Y←Y+YSM[I]*B[I, J];
            Z←Z+ZSM[I]*B[I, J];
            end;
      disegna il segmento approssimante
      LINE-ABS-3(X, Y, Z);
      end;
   return;
end;
```

Dopo che è stato disegnato un tratto di curva, si traslano i punti dati per applicare il procedimento di approssimazione mediante le blending function al tratto successivo.

Algoritmo 12.4 NEXT-SECTION

Sposta i punti dati per preparare il tratto di curva successivo.

```
Variabili globali   XSM, YSM, ZSM←array di 4 elementi contenente i pun-
                      ti dati
Variabili locali    I←indice per accedere ai punti
begin
   for I = 1 to 3
   do begin
         XSM[I]←XSM[I+1];
         YSM[I]←YSM[I+1];
         ZSM[I]←ZSM[I+1];
         end;
   return;
end;
```

Con l'algoritmo START-CURVE si possono disegnare i primi due tratti di curva, aventi come estremi i 4 punti che costituiscono gli argomenti della procedu-

ra; questa routine inserisce i punti negli array XSM, YSM, ZSM, posiziona la penna sul primo punto e mediante la routine MAKE-CURVE disegna i primi due tratti della curva. Quindi i punti vengono spostati per approntare il disegno del tratto di curva successivo.

Algoritmo 12.5 START-CURVE (XA, YA, ZA)
Routine utente che disegna i tratti di curva iniziali.
Argomenti XA, YA, ZA—array di 4 elementi contenenti i primi 4 punti dati
Variabili globali XSM, YSM, ZSM—array di 4 elementi contenenti le coordinate dei primi 4 punti
FIRST-BLEND, BLEND—array di dimensione 4 × LINES-PER-SECTION contenenti i coefficienti delle blending function
Variabili locali I—contatore del numero di punti

```
begin
   carica i punti
   for I = 1 to 4
   do begin
      XSM[I]←XA[I];
      YSM[I]←YA[I];
      ZSM[I]←ZA[I];
      end;
   posizionamento all'inizio della curva
   MOVE-ABS-3(XA[1], YA[1], ZA[1]);
   disegna i primi due tratti di curva
   MAKE-CURVE(FIRST-BLEND);
   MAKE-CURVE(BLEND);
   preparazione per il successivo tratto
   NEXT-SECTION;
   return;
end;
```

Dopo che la curva è stata iniziata, i tratti successivi possono essere aggiunti uno alla volta (vedi Figura 12.5).
Per ogni nuovo punto è possibile disegnare un nuovo tratto di curva; infatti, aggiungendo il quarto punto mentre si interpola fra il secondo ed il terzo, il tratto di curva disegnata si estende fino al punto che precede quello introdotto. La seguente routine serve per inserire il nuovo punto negli array XSM, YSM, ZSM.

Algoritmo 12.6 PUT-IN-SM (X, Y, Z)
Introduce un nuovo punto negli array XSM, YSM, ZSM.
Argomenti X, Y, Z—nuovo punto
Variabili globali XSM, YSM, ZSM—array di 4 elementi per memorizzare i punti dati

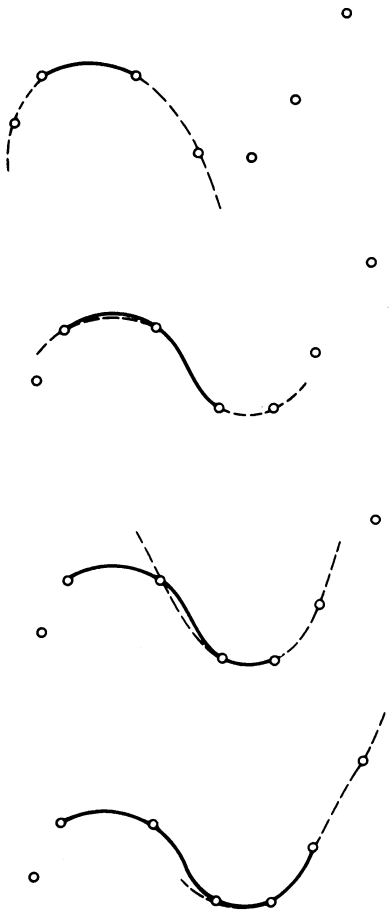


Figura 12.5 Disegno di una curva ottenuta interpolando gruppi consecutivi di quattro punti

```
begin
   XSM[4]←X;
   YSM[4]←Y;
   ZSM[4]←Z;
   return;
end;
```

Algoritmo 12.7 CURVE-ABS-3 (X, Y, Z)
Routine utente che estende la curva.
Argomenti X, Y, Z←coordinate del nuovo punto per la curva
Variabili globali BLEND←array di dimensione 4×LINES-PER-SECTION contenente la blending function

```
begin
   PUT-IN-SM(X, Y, Z);
   MAKE-CURVE(BLEND);
   NEXT-SECTION;
   return;
end;
```

La curva può essere arbitrariamente estesa, ripetendo la chiamata alla CURVE-ABS-3; quando però si desidera terminarla, occorre elaborare l'ultimo tratto in modo speciale (vedi Figura 12.6).

Algoritmo 12.8 END-CURVE (X, Y, Z)
Routine utente che termina la curva.
Argomenti X, Y, Z←ultimo punto sulla curva
Variabili globali BLEND, LAST-BLEND←array di dimensione di 4×LINES-PER-SECTION contenente i valori della blending function

```
begin
   PUT-IN-SM(X, Y, Z)
   MAKE-CURVE(BLEND);
   MAKE-CURVE(LAST-BLEND);
   return;
end;
```

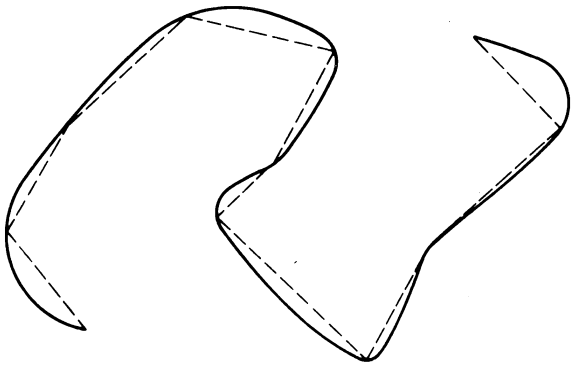


Figura 12.6 Arrotondamento con interpolazione

12.4 ARROTONDAMENTO DI POLIGONI

Con le blending function proposte è possibile arrotondare anche i lati di un poligono. Il procedimento, tra l'altro, è semplice, poiché non si avrà a che fare con i tratti estremi della curva, che in questo caso è chiusa. Basterà arrotondare ciascun lato del poligono, sostituendolo con tanti segmenti: si parte da un poligono di pochi lati per arrivare, a seconda del grado di arrotondamento voluto, ad un poligono con moltissimi lati. Sfortunatamente, la rappresentazione di un poligono nel display file è stata limitata ad un massimo di 31 lati e quindi non si potrà ottenere un arrotondamento più marcato. Ad esempio, un triangolo potrà al massimo essere rappresentato con 10 piccoli segmenti per ciascun lato e in un poligono con 15 lati, ciascuno di essi potrà essere suddiviso solamente in due segmenti. L'algoritmo SMOOTH-POLY-ABS-3 esegue l'arrotondamento di un poligono, calcolando il numero di segmenti con i quali sostituire ciascun lato, chiamando la routine SET-SMOOTH per assegnare i valori dell'array BLEND e disegnando alla fine il poligono arrotondato.

Algoritmo 12.9 SMOOTH-POLY-ABS-3 (AX, AY, AZ, M)
Routine utente che disegna un poligono arrotondato.

Argomenti	AX, AY, AZ←array contenenti i vertici del poligono originale
	M←numero di lati del poligono originale
Variabili globali	BLEND←array contenente i valori delle blending function
Variabili locali	BX, BY, BZ←array di 30 elementi contenenti i vertici del poligono arrotondato
	NSIDES←numero di lati del poligono arrotondato
	NL←numero di segmenti usati per rappresentare ciascun

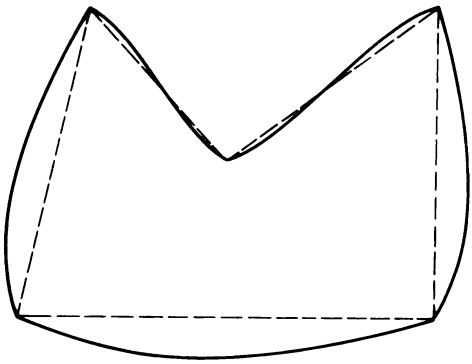


Figura 12.7 Arrotondamento di un poligono

```
                    lato del poligono originale
                J←indice per i lati del poligono arrotondato
                K←variabile usata per i segmenti presenti in ciascun lato
                  del poligono
                I, I1←variabili per ciascun gruppo di 4 punti dati
                L←variabile relativa alle 3 blending function
begin
   if M<3 or M>31
   then return-error 'ERRORE NEI LATI DEL POLIGONO';
   NL←INT(31/M);
   NSIDES←NL*M;
   if NL = 1
   then POLYGON-ABS-3(AX, AY, AZ, M)
   else begin
        SET-SMOOTH(NL);
        J←1;
        for I1 = 1 to M
        arrotonda tutti i lati
        do for K = 1 to NL
        arrotonda un lato del poligono originale
        do begin
             BX[J]←0;
             BY[J]←0;
             BZ[J]←0;
             I←I1;
             for L = 1 to 4
             un lato del poligono arrotondato
             do begin
                BX[J]←BX[J]+AX[I]*BLEND[L, K];
                BY[J]←BY[J]+AY[I]*BLEND[L, K];
                BZ[J]←BZ[J]+AZ[I]*BLEND[L, K];
                if I = M then I←1 else I←I+1;
                end;
             J←J+1;
             end;
        disegna il risultato
        POLYGON-ABS-3(BX, BY, BZ, NSIDES);
        end;
   return;
end;
```

12.5 B-SPLINE

Dopo aver scritto un programma di interpolazione usando le blending function di prima approssimazione, occorre esaminarne le inadeguatezze. Per prima cosa bisogna notare che la somma delle blending function non è uguale a 1 per ogni valore di u: si ricordi che la loro somma risultava 1 per valori interi del parametro u, ma non per valori reali. Si supponga che tutti i punti dati abbiano la stessa coordinata $x = x_0$; si vuole che la curva approssimante abbia valori costanti di x anche nei punti compresi fra i punti dati; la curva approssimante è data dalla

$$x = \sum_{i=1}^{n} x_i B_i(u) \tag{12.13}$$

e per $x_i = x_0$ si ha

$$x = x_o \sum_{i=1}^{n} B_i(u) \tag{12.14}$$

Si possono ottenere valori di x costanti anche nei punti intermedi a quelli dati, solo se la somma delle blending function è uguale a 1 per tutti i valori di u. Ciò significa che le curve che dovrebbero giacere in un piano possono in realtà attorcigliarsi dentro e fuori di esso. Questo particolare problema potrebbe essere superato dividendo i coefficienti delle blending function (calcolate nella procedura SET-SMOOTH) per la loro somma, per ciascun valore di u, cioè normalizzando tali funzioni.

Purtroppo, però, questo non è il solo problema. Ciascun tratto di curva è collegato al successivo in uno dei punti dati, ma le pendenze dei due tratti non sono necessariamente le stesse in tale punto e potrebbero esistere dei punti angolosi; in questi casi non si otterrà certamente una curva completamente liscia. Inoltre, per fare in modo che la curva passi per un dato punto, bisogna annullare l'effetto di tutti gli altri; ma al variare del parametro u — e cioè spostandosi dal punto interpolato lungo la curva — anche gli altri punti dati devono influenzarne l'andamento; perciò si vedrebbe il controllo da parte di un punto dato «pulsare» al variare di u. Un comportamento più naturale potrebbe essere ottenuto facendo variare in modo continuo — per ogni punto dato — il controllo della curva, da un valore nullo per un punto posto molto lontano ad un valore massimo per un punto molto vicino, senza costringere la curva a passare dal punto, ma piuttosto cercando di mantenerla in un intorno dello stesso. Il risultato sarà una curva che in generale segue l'andamento dei punti assegnati, ma che non necessariamente passa attraverso di essi. Un insieme di blending function che soddisfa queste condizioni e la cui somma è sempre uguale a 1 viene chiamato insieme di *B-spline*. Queste funzioni generano tratti di curva con continuità di pendenza, in modo che la curva risultante sia liscia (vedi Figura 12.8). Mostrare come si ricavano in generale le funzioni *B*-spline va al di là del-

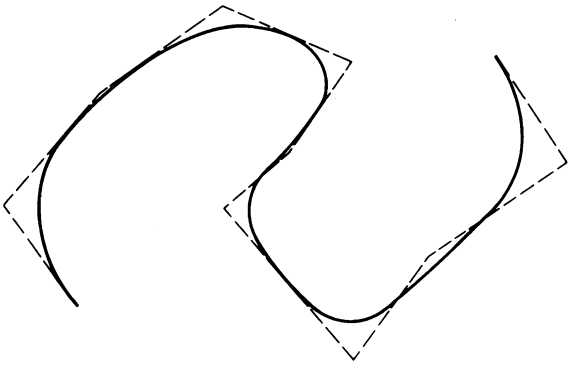


Figura 12.8 Arrotondamento tramite *B*-spline

lo scopo di questo libro; verranno comunque presentate e implementate le funzioni *B-spline cubiche*, che sono utili in molte applicazioni. Le blending function *B*-spline cubiche approssimano 4 punti dati e sono polinomiali cubiche in u, proprio come lo erano le blending function di prima approssimazione usate precedentemente. Le funzioni *B*-spline cubiche per il tratto intermedio della curva sono le seguenti:

$$\begin{aligned} B_1(u) &= (1-u)^3/6 \\ B_2(u) &= u^3/2-u^2+2/3 \\ B_3(u) &= u^3/2+u^2/2+u/2+1/6 \\ B_4(u) &= u^3/6 \end{aligned} \tag{12.15}$$

L'applicazione delle blending function *B*-spline cubiche alla parte iniziale e finale della curva richiede un trattamento particolare; le funzioni che seguono devono essere usate per il primo tratto di curva:

$$\begin{aligned} B'_1(u) &= (1-u)^3 \\ B'_2(u) &= 21u^3/12-9u^2/2+3u \\ B'_3(u) &= -11u^3/12+3u^2/2 \\ B'_4(u) &= u^3/6 \end{aligned} \tag{12.16}$$

Mentre per l'ultimo tratto di curva esse sono:

$$\begin{aligned} B'_{1R}(u) &= B'_4(1-u) \\ B'_{2R}(u) &= B'_3(1-u) \\ B'_{3R}(u) &= B'_2(1-u) \\ B'_{4R}(u) &= B'_1(1-u) \end{aligned} \tag{12.17}$$

Il secondo tratto di curva deve anch'esso usare blending function speciali:

$$\begin{aligned} B''_1(u) &= (1-u)^3/4 \\ B''_2(u) &= 7u^3/12 - 5u^2/4 + u/4 + 7/12 \\ B''_3(u) &= -u^3/2 + u^2/2 + u/2 + 1/6 \\ B''_4(u) &= u^3/6 \end{aligned} \tag{12.18}$$

e così pure il penultimo tratto della curva:

$$\begin{aligned} B''_{1R}(u) &= B''_4(1-u) \\ B''_{2R}(u) &= B''_3(1-u) \\ B''_{3R}(u) &= B''_2(1-u) \\ B''_{4R}(u) &= B''_1(1-u) \end{aligned} \tag{12.19}$$

Si potrà implementare il metodo per disegnare curve *B*-spline, in modo analogo a quello che si è fatto per le blending function precedentemente introdotte: invece della routine SET-SMOOTH, si avrà bisogno di un algoritmo che calcola i 5 gruppi di blending function *B*-spline. Questa operazione è realizzata con una sola chiamata alla routine SET-B-SPLINE.

Algoritmo 12.10 SET-B-SPLINE (NUMBER-OF-LINES)
Routine utente che assegna il numero di segmenti per ciascun tratto di curva per l'interpolazione con le *B*-spline.

Argomenti NUMBER-OF-LINES←numero di segmenti per approssimare un tratto di curva

Variabili globali LINES-PER-SECTION←memorizzazione di NUMBER-OF-LINES
BLEND, FIRST-BLEND, SECOND-BLEND, NEXT-TO-LAST-BLEND, LAST-BLEND←array di dimensioni 4×MAX-NUMBER-OF-LINES per memorizzare le blending function

Variabili locali I←indice per percorrere i punti necessari

Costanti MAX-NUMBER-OF-LINES←massimo numero di segmenti per approssimare un tratto di curva

```
begin
  if NUMBER-OF-LINES < 1
  or NUMBER-OF-LINES > MAX-NUMBER-OF-LINES
  then return-error 'NUMERO DI SEGMENTI NON VALIDO';
  LINES-PER-SECTION ← NUMBER-OF-LINES;
  for I = 1 to NUMBER-OF-LINES
  do begin
       U ← I/NUMBER-OF-LINES
       FIRST-BLEND[1, I] ← (1 − U)↑3;
```

```
            FIRST-BLEND[4, I]←(U↑3)/6;
            FIRST-BLEND[3, I]←(3/2−11*U/12)*(U↑2);
            FIRST-BLEND[2, I]←1−FIRST-BLEND[1, I]
                    −FIRST-BLEND[3, I]−FIRST-BLEND[4, I];
            SECOND-BLEND[1, I]←FIRST-BLEND[1, I]/4;
            SECOND-BLEND[4, I]←FIRST-BLEND[4, I];
            SECOND-BLEND[3, I]←(((1−U)*U+1)*U+1/3)/2;
            SECOND-BLEND[2, I]←1−SECOND-BLEND[1, I]
                    −SECOND-BLEND[3, I]−SECOND-BLEND[4, I];
            BLEND[1, I]←FIRST-BLEND[1, I]/6;
            BLEND[4, I]←FIRST-BLEND[4, I];
            BLEND[3, I]←SECOND-BLEND[3, I];
            BLEND[2, I]←1−BLEND[1, I]−BLEND[3, I]−BLEND[4, I];
            J←NUMBER-OF-LINES−I
            if J≠0
            then begin
                NEXT-TO-LAST-BLEND[1, J]←SECOND-BLEND[4, I];
                NEXT-TO-LAST-BLEND[2, J]←SECOND-BLEND[3, I];
                NEXT-TO-LAST-BLEND[3, J]←SECOND-BLEND[2, I];
                NEXT-TO-LAST-BLEND[4, J]←SECOND-BLEND[1, I];
                LAST-BLEND[1, J]←FIRST-BLEND[4, I];
                LAST-BLEND[2, J]←FIRST-BLEND[3, I];
                LAST-BLEND[3, J]←FIRST-BLEND[2, I];
                LAST-BLEND[4, J]←FIRST-BLEND[1, I];
                end;
            end;
        NEXT-TO-LAST-BLEND[1, NUMBER-OF-LINES]←0;
        NEXT-TO-LAST-BLEND[2, NUMBER-OF-LINES]←1/6;
        NEXT-TO-LAST-BLEND[3. NUMBER-OF-LINES]←7/12;
        NEXT-TO-LAST-BLEND[4, NUMBER-OF-LINES]←1/4;
        LAST-BLEND[1, NUMBER-OF-LINES]←0;
        LAST-BLEND[2, NUMBER-OF-LINES]←0;
        LAST-BLEND[3, NUMBER-OF-LINES]←0;
        LAST-BLEND[4, NUMBER-OF-LINES]←1;
        return;
    end;
```

Per iniziare il calcolo della curva *B*-spline, sono necessari i primi 5 punti di controllo. Analogamente a quanto si è fatto con la blending function di prima approssimazione, si memorizza il primo punto negli array XSM, YSM, ZSM e si effettua una chiamata alla routine MAKE-CURVE con le blending function speciali di partenza. L'algoritmo che segue inizia la costruzione di una curva *B*-spline.

Algoritmo 12.11 START-B-SPLINE (AX, AY, AZ)
Routine utente che inizia ad approssimare i punti di controllo con una cuva *B*-spline.

Argomenti AX, AY, AZ←array di 5 elementi contenenti i primi 5 punti

Variabili globali XSM, YSM, ZSM←array di 4 elementi contenenti i punti di interpolazione
FIRST-BLEND, SECOND-BLEND←array di dimensione 4×LINES-PER-SECTION contenenti i valori della blending function

Variabili locali I←indice per percorrere i punti necessari

```
begin
   for I = 1 to 4
   do begin
      XSM[I]←AX[I];
      YSM[I]←AY[I];
      ZSM[I]←AZ[I];
      end;
   MOVE-ABS-3(AX[1], AY[1], AZ[1]);
   MAKE-CURVE[FIRST-BLEND];
   NEXT-SECTION;
   PUT-IN-SM(AX[5], AY[5], AZ[5]);
   MAKE-CURVE[SECOND-BLEND];
   NEXT-SECTION;
   return;
end;
```

Una volta che la curva è stata iniziata, può essere continuata chiamando la routine CURVE-ABS-3. Ogni chiamata a questa routine aggiunge un nuovo tratto di curva al precedente. Per concludere la curva *B*-spline, si devono disegnare gli ultimi due tratti utilizzando la routine END-B-SPLINE, che ha come argomenti gli ultimi due punti da interpolare e utilizza le blending function NEXT-TO-LAST-BLEND e LAST-BLEND.

Algoritmo 12.12 END-B-SPLINE (X1, Y1, Z1, X2, Y2, Z2)
Routine utente che conclude una curva *B*-spline.

Argomenti X1, Y1, Z1←penultimo punto dato
X2, Y2, Z2←ultimo punto dato

Variabili globali NEXT-TO-LAST-BLEND, LAST-BLEND←array di dimensioni 4×LINES-PER-SECTION contenenti i valori della blending function

```
begin
   PUT-IN-SM(X1, Y1, Z1);
   MAKE-CURVE(NEXT-TO-LAST-BLEND);
   NEXT-SECTION;
```

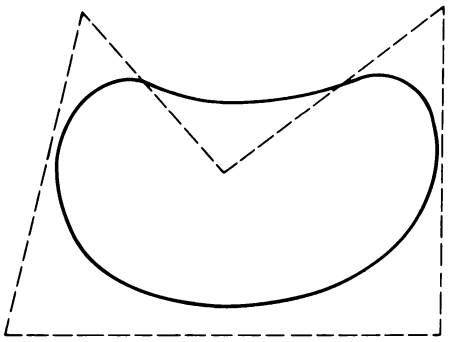


Figura 12.9 Arrotondamento di un poligono tramite *B*-spline

```
    PUT-IN-SM(X2, Y2, Z2);
    MAKE-CURVE(LAST-BLEND);
    return;
end;
```

Le blending function speciali usate all'inizio (e alla fine) della curva fanno in modo che la curva inizi (finisca) nel primo (ultimo) punto assegnato. La curva passerà quindi per il primo e per l'ultimo punto di controllo. La pendenza della curva in tali estremi è uguale alla pendenza del segmento che unisce il primo e il secondo (penultimo e ultimo) punto di controllo.
Le blending function *B*-spline possono anche essere usate per costruire poligoni con gli spigoli smussati (vedi Figura 12.9). Per far ciò, bisogna soltanto modificare l'algoritmo SMOOTH-POLY-ABS-3, sostituendo la chiamata a SET-SMOOTH con la chiamata a SET-B-SPLINE.

Algoritmo 12.13 B-SPLINE-POLY-ABS-3 (AX, AY, AZ, M)
Routine utente che disegna un poligono con gli spigoli smussati.

Argomenti AX, AY, AZ←array contenenti i vertici del poligono originale
M←numero dei lati del poligono di partenza

Variabili globali BLEND←array contenente i valori delle blending function

Variabili locali BX, BY, BZ←array di 30 elementi per contenere i vertici del poligono smussato
NL←numero di tratti di segmento per ciascuno dei lati del poligono originale
NSIDES←numero di lati del poligono arrotondato
J←indice per memorizzare i lati del poligono
K←indice per considerare i segmenti componenti tutti i

```
                    lati del poligono
                I, I1←indici per accedere ai quattro punti dati
                L←indice per considerare le quattro blending function
begin
   if M<3 or M>31
   then return-error 'ERRORE NEI LATI DEL POLIGONO';
   NL←INT(31/M);
   NSIDES←NL*M;
   if NL = 1
   then POLYGON-ABS-3(AX, AY, AZ, M)
   else begin
        SET-B-SPLINE(NL);
        J←1;
        for I1 = 1 to M
        do for K = 1 to NL
           do begin
                BX[J]←0;
                BY[J]←0;
                BZ[J]←0;
                I←I1;
                for L = 1 to 4
                do begin
                     BX[J]←BX[J]+AX[I]*BLEND[L, K];
                     BY[J]←BY[J]+AY[I]*BLEND[L, K];
                     BZ[J]←BZ[J]+AZ[I]*BLEND[L, K];
                     if I = M then I←1 else I←I+1;
                     end;
                J←J+1;
                end;
        POLYGON-ABS-3(BX, BY, BZ, NSIDES);
        end;
   return;
end;
```

12.6 B-SPLINE E ANGOLI

Le blending function *B*-spline sono state progettate per eliminare i punti angolosi. In generale una curva *B*-spline si avvicina ai punti di controllo senza però passare per essi; se però lo si desidera è possibile imporre a una curva *B*-spline di passare per un punto o di formare un punto angoloso, usando più punti di controllo coincidenti. La chiamata

```
CURVE-ABS-3(X0, Y0, Z0)
```

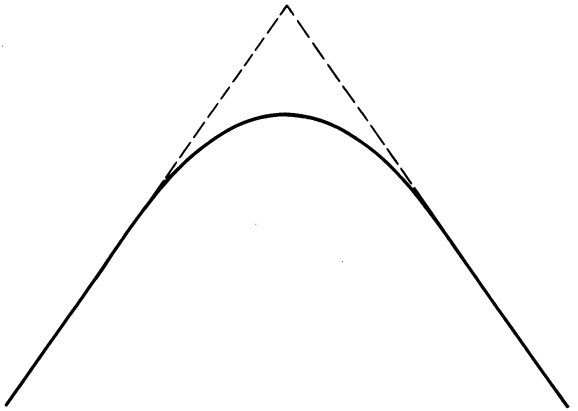


Figura 12.10a *B*-spline con un punto

genera un punto di controllo di coordinate (X0, Y0, Z0), che spinge la curva verso quella direzione (vedi Figura 12.10a).
Le due chiamate

```
CURVE-ABS-3(X0, Y0, Z0)
CURVE-ABS-3(X0, Y0, Z0)
```

producono due punti di controllo coincidenti, che costringono la curva verso lo stesso punto di coordinate (X0, Y0, Z0). La curva però verrà spinta più vicina al punto e il vertice risulterà più acuto (vedi Figura 12.10b).
Se vengono usati tre punti di controllo coincidenti

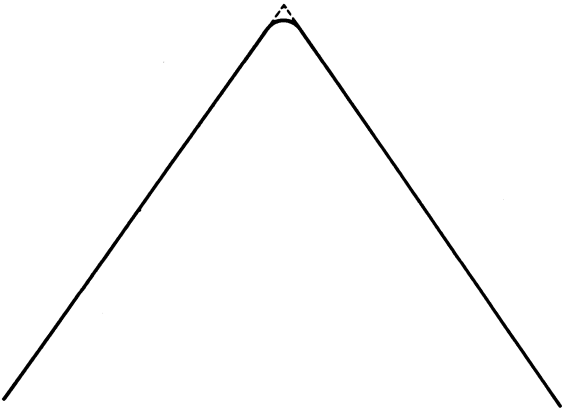


Figura 12.10b *B*-spline con due punti

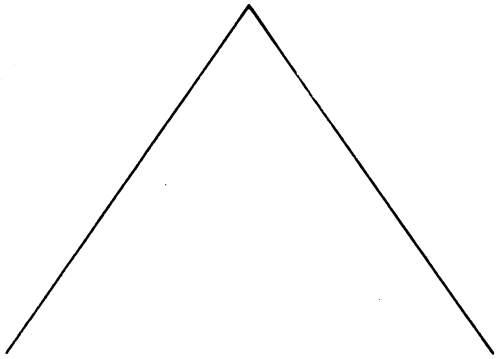


Figura 12.10c *B*-spline con tre punti

```
CURVE-ABS-3(X0, Y0, Z0)
CURVE-ABS-3(X0, Y0, Z0)
CURVE-ABS-3(X0, Y0, Z0)
```

la curva verrà forzata a passare per tale punto (vedi Figura 12.10c).
L'interpolazione con le blending function e l'approssimazione tramite piccoli segmenti di retta ci permettono di disegnare curve di forma matematica complessa e di applicare trasformazioni di immagini e metodi di clipping che sono già stati sviluppati. L'aspetto negativo di tale tecnica è l'eccessiva quantità di memoria richiesta, relativamente al display file. Per ogni punto di controllo possono essere generati 5 o 10 segmenti, che vengono ad occupare una considerevole porzione di display file.

APPLICAZIONE PRATICA

Un'area nella quale può essere utile disegnare curve è quella della progettazione di oggetti quali l'ala di un aeroplano o il parabrezza di un'automobile. Le routine sviluppate in questo capitolo possono essere usate in un programma che disegni interattivamente la curva di un profilo alare. Il progettista potrà selezionare con operazioni di pick 10 punti di controllo, che muoverà per mezzo del locator. Le routine che generano le funzioni *B*-spline verranno usate per approssimare i punti selezionati. Poiché, quando un punto di controllo viene spostato, la curva cambia, il progettista continuerà a spostare i punti finché non otterrà la curva desiderata. Per prima cosa vengono inizializzati i punti di controllo, che si evidenzieranno sullo schermo con il carattere X. I punti verranno costruiti nell'origine del sistema di coordinate assolute e posizionati sullo schermo tramite una traslazione.

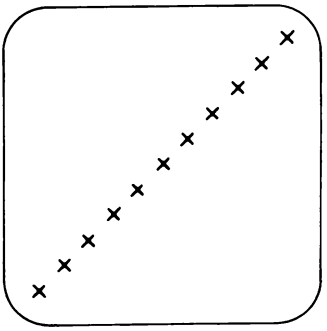


Figura 12.11 Dieci punti di controllo lungo la diagonale

```
begin
   for I = 1 to 10
   do begin
         CREATE-SEGMENT(I);
         MOVE-ABS-2(0, 0);
         TEXT("X");
         CLOSE-SEGMENT;
         SET-VISIBILITY(I, TRUE);
         SET-DETECTABILITY(I, TRUE);
         SET-IMAGE-TRANSLATION(I, I/10, I/10);
         end;
   SET-B-SPLINE(8);
end;
```

Si avranno così 10 punti lungo la diagonale dello schermo (vedi Figura 12.11). La parte principale del programma consiste in un ciclo che permette di eseguire ripetute modifiche e visualizzazioni della curva. La routine che serve a spostare un punto di controllo può essere del tipo seguente:

```
begin
   SAMPLE←0;
   while SAMPLE = 0 do AWAIT-PICK(WAIT, SAMPLE);
   BUTTON-NUM←0;
   while BUTTON-NUM = 0
      do AWAIT-BUTTON-GET-LOCATOR(WAIT,
         BUTTON-NUM, X, Y);
   SET-IMAGE-TRANSLATION(SAMPLE, X, Y);
end;
```

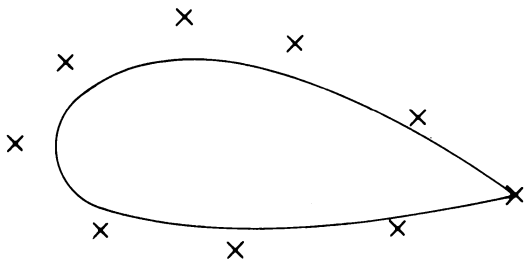


Figura 12.12 Posizionamento dei punti di controllo per ottenere la curva della forma desiderata

Ora occorrono alcune istruzioni che permettano di disegnare la curva utilizzando le posizioni dei punti di controllo e alcuni array per contenerli. AX, AY, AZ siano array di cinque elementi e AZ contenga valori nulli; le prime cinque posizioni dei punti di controllo dovranno essere memorizzate in questi array in modo tale che possano essere usate dalla routine START-B-SPLINE. I tre punti successivi saranno trattati dalla CURVE-ABS-3, mentre gli ultimi due serviranno per la END-B-SPLINE. La curva verrà disegnata nel segmento 11 del display file (vedi Figura 12.12).

```
begin
   DELETE-SEGMENT(11);
   CREATE-SEGMENT(11);
   for I = 1 to 5
      do INQUIRE-IMAGE-TRANSLATION(I, AX[I], AY[I]);
   START-B-SPLINE(AX, AY, AZ);
   for I = 6 to 8
      do begin
            INQUIRE-IMAGE-TRANSLATION(I, X, Y);
            CURVE-ABS-3(X, Y, 0);
            end;
   INQUIRE-IMAGE-TRANSLATION(9, X, Y);
   INQUIRE-IMAGE-TRANSLATION(10, U, V);
   END-B-SPLINE(X, Y, 0, U, V, 0);
   CLOSE-SEGMENT;
   SET-VISIBILITY(11, TRUE);
   SET-DETECTABILITY(11, FALSE);
end;
```

ESERCIZI

12.1 a) Calcolare i valori B_1, B_2, B_3, B_4 delle blending function presentate nella prima parte del capitolo che servono per approssimare ciascun tratto di curva con 5 segmenti di retta.
b) Fare uno schizzo dell'andamento delle funzioni.
c) Usando i seguenti quattro punti di controllo (1, 1, 0), (2, 1, 0), (2, 2, 0), (1, 2, 0), calcolare e disegnare la curva interpolata nel suo tratto centrale.

12.2 a) Calcolare i valori delle blending function *B*-spline cubiche per il primo tratto, per il secondo e per il tratto medio (B_i', B_i'', B_i per $i = 1, 2, 3, 4$) in modo che ciascun tratto di curva sia approssimato da tre segmenti.
b) Disegnare queste funzioni.
c) Se i primi 6 punti di controllo sono (1, 1, 0), (2, 1, 0), (2, 2, 0), (1, 2, 0), (0, 2, 0), (0, 3, 0), calcolare e disegnare i primi tre tratti della curva.

PROBLEMI DI PROGRAMMAZIONE

12.1 a) Implementare gli algoritmi dal 12.2 al 12.9 compreso
b) Implementare anche gli algoritmi dal 12.10 al 12.13.

12.2 Controllare i metodi di interpolazione proposti costruendo un insieme di punti di controllo. Mostrare i punti di controllo come segue:
a) uniti con un singolo segmento di retta
b) interpolati usando lo schema di interpolazione proposto con otto segmenti per ciascun tratto di curva
c) interpolati usando *B*-spline cubiche con otto segmenti per ciascun tratto di curva.

12.3 Usare le routine di interpolazione proposte per presentare una curva liscia riguardante i dati di vendita di un prodotto, di cui si è parlato nel problema di programmazione 1.3.

12.4 Rappresentare graficamente un viso ed usare l'algoritmo *B*-spline per arrotondarlo.

12.5 Mostrare l'effetto dell'arrotondamento su poligoni regolari aventi da 3 a 8 lati. Mostrare ciascuna figura sia prima che dopo tale operazione.

A

Il sistema grafico CORE

In questa appendice vengono elencate le routine disponibili nel sistema grafico standard CORE. I progettisti di questo sistema, rendendosi conto che non tutti gli utenti hanno bisogno dell'intera capacità del sistema e che non tutte le configurazioni hardware sono in grado di supportarla, hanno diviso il sistema CORE in livelli sovrapponibili in ciascuna delle tre aeree di dimensionamento, output e input.
Per l'area di dimensionamento esistono due livelli: il più basso contiene solo le operazioni bidimensionali, il più alto aggiunge a queste le operazioni tridimensionali.
Nell'area di output ci sono tre livelli: il livello più basso supporta solo i segmenti temporanei di display file; al secondo livello (output bufferizzato) sono disponibili i segmenti relativi alla visualizzazione, mentre al terzo livello (output dinamico) vengono attribuiti ai segmenti i parametri di trasformazione dell'immagine.
Anche per l'input esistono tre livelli, il primo dei quali non consente alcun ingresso; il secondo livello riguarda l'input sincrono, mentre il terzo livello consente un'interazione asincrona.
Attualmente non esiste un solo sistema CORE, bensì 18 sistemi diversi, ciascuno dei quali è caratterizzato da un livello di output, di input e di dimensionamento. Ciascun programma può essere eseguito ad un livello più alto di quello per il quale è stato scritto. Vengono qui di seguito riportate le procedure del sistema CORE, ordinate a seconda dei livelli. Una descrizione di queste procedure è reperibile in *Computer Graphics*, vol. 13, n. 3 (Agosto 1979), pp. II/1-105.

Il sistema bidimensionale fondamentale, privo di input, è composto dalle seguenti routine:

BEGIN-BATCH-OF-UPDATES
CLOSE-TEMPORARY-SEGMENT

CREATE-TEMPORARY-SEGMENT
DESELECT-FOR-SEGMENT-CONSTRUCTION
END-BATCH-OF-UPDATES
ESCAPE
INITIALIZE-CORE
INITIALIZE-VIEW-SURFACE
INQUIRE-CHARJUST
INQUIRE-CHARPATH
INQUIRE-CHARPRECISION
INQUIRE-CHARSIZE
INQUIRE-CHARSPACE
INQUIRE-CHARUP-2
INQUIRE-COLOR
INQUIRE-CONTROL-STATUS
INQUIRE-CURRENT-POSITION-2
INQUIRE-ESCAPE
INQUIRE-FONT
INQUIRE-INTENSITY
INQUIRE-LINESTYLE
INQUIRE-LINEWIDTH
INQUIRE-MARKER-SYMBOL
INQUIRE-NDC-SPACE-2
INQUIRE-OUTPUT-CAPABILITIES
INQUIRE-PEN
INQUIRE-PICK-ID
INQUIRE-PRIMITIVE-ATTRIBUTES-2
INQUIRE-SELECTED-SURFACES
INQUIRE-VIEW-UP-2
INQUIRE-VIEWING-CONTROL-PARAMETERS
INQUIRE-VIEWPORT-2
INQUIRE-WINDOW
INQUIRE-WORLD-COORDINATE-MATRIX-2
INQUIRE-OPEN-TEMPORARY-SEGMENT
INQUIRE-TEXT-EXTENT-2
LINE-ABS-2
LINE-REL-2
LOG-ERROR
MAKE-PICTURE-CURRENT
MAP-NDC-TO-WORLD-2
MAP-WORLD-TO-NDC-2
MARKER-ABS-2
MARKER-REL-2
MOVE-ABS-2
MOVE-REL-2
NEW-FRAME

POLYMARKER-ABS-2
POLYMARKER-REL-2
POLYLINE-ABS-2
POLYLINE-REL-2
REPORT-MOST-RECENT-ERROR
SELECT-FOR-SEGMENT-CONSTRUCTION
SET-CHARJUST
SET-CHARPATH
SET-CHARPRECISION
SET-CHARSIZE
SET-CHARSPACE
SET-CHARUP-2
SET-COLOR
SET-FONT
SET-IMMEDIATE-VISIBILITY
SET-INTENSITY
SET-LINESTYLE
SET-LINEWIDTH
SET-MARKER-SYMBOL
SET-NDC-SPACE-2
SET-PEN
SET-PICK-ID
SET-PRIMITIVE-ATTRIBUTES-2
SET-VIEW-UP-2
SET-VIEWPORT-2
SET-VISIBILITIES
SET-WINDOW
SET-WINDOW-CLIPPING
SET-WORLD-COORDINATE-MATRIX-2
TERMINATE-CORE
TERMINATE-VIEW-SURFACE
TEXT

Per un sistema tridimensionale privo di input e con output minimo, si possono aggiungere le seguenti routine:

INQUIRE-CHARPLANE
INQUIRE-CHARUP-3
INQUIRE-CURRENT-POSITION-3
INQUIRE-NDC-SPACE-3
INQUIRE-PRIMITIVE-ATTRIBUTES-3
INQUIRE-PROJECTION
INQUIRE-VIEW-DEPTH
INQUIRE-VIEW-PLANE-DISTANCE
INQUIRE-VIEW-PLANE-NORMAL

INQUIRE-VIEW-REFERENCE-POINT
INQUIRE-VIEW-UP-3
INQUIRE-VIEWING-PARAMETERS
INQUIRE-VIEWPORT-3
INQUIRE-TEXT-EXTENT-3
INQUIRE-WORLD-COORDINATE-MATRIX-3
LINE-ABS-3
LINE-REL-3
MAP-NDC-TO-WORLD-3
MAP-WORLD-TO-NDC-3
MARKER-ABS-3
MARKER-REL-3
MOVE-ABS-3
MOVE-REL-3
POLYLINE-ABS-3
POLYLINE-REL-3
POLYMARKER-ABS-3
POLYMARKER-REL-3
SET-BACK-PLANE-CLIPPING
SET-CHARPLANE
SET-CHARUP-3
SET-COORDINATE-SYSTEM-TYPE
SET-FRONT-PLANE-CLIPPING
SET-NDC-SPACE-3
SET-PRIMITIVE-ATTRIBUTES-3
SET-PROJECTION
SET VIEW-DEPTH
SET-VIEW-PLANE-DISTANCE
SET-VIEW-PLANE-NORMAL
SET-VIEW-REFERENCE-POINT
SET-VIEW-UP-3
SET-VIEWING-PARAMETERS
SET-VIEWPORT-3
SET-WORLD-COORDINATE-MATRIX-3

L'estensione con un output bufferizzato include le seguenti routine:

CLOSE-RETAINED-SEGMENT
CREATE-RETAINED-SEGMENT
DELETE-ALL-RETAINED-SEGMENTS
DELETE-RETAINED-SEGMENT
INQUIRE-HIGHLIGHTING
INQUIRE-OPEN-RETAINED-SEGMENT
INQUIRE-RETAINED-SEGMENT-NAMES
INQUIRE-RETAINED-SEGMENT-SURFACES

INQUIRE-SEGMENT-HIGHLIGHTING
INQUIRE-SEGMENT-VISIBILITY
INQUIRE-VISIBILITY
RENAME-RETAINED-SEGMENT
SET-HIGHLIGHTING
SET-SEGMENT-HIGHLIGHTING
SET-SEGMENT-VISIBILITY
SET-VISIBILITY

Le routine disponibili per un output dinamico in due dimensioni sono:

INQUIRE-IMAGE-TRANSFORMATION-2
INQUIRE-IMAGE-TRANSFORMATION-TYPE
INQUIRE-IMAGE-TRANSLATE-2
INQUIRE-SEGMENT-IMAGE-TRANSFORMATION-2
INQUIRE-SEGMENT-IMAGE-TRANSFORMATION-TYPE
INQUIRE-SEGMENT-IMAGE-TRANSLATE-2
SET-IMAGE-TRANSFORMATION-2
SET-IMAGE-TRANSFORMATION-TYPE
SET-IMAGE-TRANSLATE-2
SET-SEGMENT-IMAGE-TRANSFORMATION-2
SET-SEGMENT-IMAGE-TRANSLATE-2

Per un output dinamico tridimensionale esistono anche le seguenti:

INQUIRE-IMAGE-TRANSFORMATION-3
INQUIRE-SEGMENT-IMAGE-TRANSFORMATION-3
SET-IMAGE-TRANSFORMATION-3
SET-SEGMENT-IMAGE-TRANSFORMATION-3

Se si dispone di un input sincrono, sono presenti le seguenti routine:

AWAIT-ANY-BUTTON
AWAIT-ANY-BUTTON-GET-LOCATOR-2
AWAIT-ANY-BUTTON-GET-VALUATOR
AWAIT-KEYBOARD
AWAIT-STROKE-2
INITIALIZE-DEVICE
INITIALIZE-GROUP
INQUIRE-BUTTON
INQUIRE-ECHO
INQUIRE-ECHO-POSITION
INQUIRE-ECHO-SEGMENTS
INQUIRE-ECHO-SURFACE
INQUIRE-INPUT-CAPABILITIES

INQUIRE-INPUT-DEVICE-CHARACTERISTICS
INQUIRE-KEYBOARD
INQUIRE-LOCATOR-2
INQUIRE-LOCPORT-2
INQUIRE-STROKE
INQUIRE-VALUATOR
SET-ALL BUTTONS
SET-BUTTON
SET-ECHO
SET-ECHO-GROUP
SET-ECHO-POSITION
SET-KEYBOARD
SET-LOCATOR-2
SET-LOCPORT-2
SET-STROKE
SET-VALUATOR
TERMINATE-DEVICE
TERMINATE-GROUP

Se inoltre esiste un output bufferizzato:

AWAIT-PICK
INQUIRE-DETECTABILITY
INQUIRE-PICK
INQUIRE-SEGMENT-DETECTABILITY
SET-DETECTABILITY
SET-PICK
SET-SEGMENT-DETECTABILITY

Per un sistema tridimensionale con input sincrono sono disponibili le seguenti routine di ingresso:

AWAIT-ANY-BUTTON-GET-LOCATOR-3
AWAIT-STROKE-3
INQUIRE-LOCATOR-DIMENSION
INQUIRE-LOCATOR-3
INQUIRE-LOCPORT-3
INQUIRE-STROKE-DIMENSION
SET-LOCATOR-3
SET-LOCPORT-3

Un input completo richiede l'aggiunta delle seguenti routine:

ASSOCIATE
AWAIT-EVENT

DISABLE-ALL
DISABLE-DEVICE
DISABLE-GROUP
DISASSOCIATE
DISASSOCIATE-ALL
DISASSOCIATE-DEVICE
DISASSOCIATE-GROUP
ENABLE-DEVICE
ENABLE-GROUP
FLUSH-ALL-EVENTS
FLUSH-DEVICE-EVENTS
FLUSH-GROUP-EVENTS
GET-KEYBOARD-DATA
GET-LOCATOR-DATA-2
GET-STROKE-DATA-2
GET-VALUATOR-DATA
INQUIRE-DEVICE-ASSOCIATIONS
INQUIRE-DEVICE-STATUS
READ-LOCATOR-2
READ-VALUATOR

Un output bufferizzato richiede l'aggiunta delle routine:

GET-PICK-DATA

Ed infine, le tre dimensioni comportano le seguenti routine:

GET-LOCATOR-DATA-3
GET-STROKE-DATA-3
READ-LOCATOR-3

B

Grafica su un terminale non grafico

In questo testo si è continuamente ribadito che un output grafico può essere ottenuto su un qualsiasi terminale CRT o su una stampante a linee. In questa appendice vengono illustrati i sottoprogrammi FORTRAN necessari per interfacciare il sistema grafico ad una stampante. In modo analogo, la stessa operazione può essere fatta con un terminale CRT. Le routine di visualizzazione lavorano su di una immagine larga 90 caratteri ed alta 50 linee. Vengono usati diversi caratteri per ottenere differenti tipi di riempimento e di contorno. L'algoritmo 1.1 DDA deve essere incluso in queste routine d'interfacciamento dipendenti dal dispositivo usato; esso non viene qui riportato, ma ricalca in tutto e per tutto la stesura riportata nel Capitolo 1. Come frame buffer viene usato un array di dimensione 90×50 chiamato IFRAME.
L'algoritmo 1.2 ERASE pulisce il frame buffer assegnando a ciascun elemento il carattere spazio.

```
      SUBROUTINE ERASE
      COMMON /SBLK/ IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
        IHIGHT
      COMMON /FBLK/ IFRAME(90,50)
      DATA IBLNK /' '/
C     SCANDISCE GLI ARRAY ASSEGNANDO UNO SPAZIO A OGNI
        ELEMENTO
      DO 120 I = IWSTRT, IWEND
      DO 110 J = IHSTRT, IHEND
      IFRAME(I,J) = IBLNK
  110 CONTINUE
  120 CONTINUE
      RETURN
      END
```

L'algoritmo 1.3 DISPLAY stampa il contenuto del frame buffer dall'alto verso

il basso. Per ottenere un frame per pagina vengono stampate anche delle linee di spazi.

```
      SUBROUTINE DISPLY
C     STAMPA IL CONTENUTO DEL FRAME BUFFER
      COMMON /SBLK /IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
        IHIGHT
      COMMON /FBLK/ IFRAME(90,50)
      IHEND1 = IHEND+1
C     STAMPA GLI ARRAY
      WRITE (6,123)
  123 FORMAT (///////)
      DO 220 I = IHSTRT, IHEND
      WRITE (6,210) (IFRAME(J,IHEND1-I), J = IWSTRT,IWEND)
  210 FORMAT(90A1)
  220 CONTINUE
      WRITE (6,123)
      RETURN
      END
```

L'algoritmo 1.4 INITIALIZE-1 assegna i parametri di dimensionamento utili alla visualizzazione e, non essendo disponibile il controllo carrello, richiede all'utente il posizionamento della carta.

```
      SUBROUTINE INITL1
      COMMON /SBLK /IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
        IHIGHT
      IWEND = 90
      IHEND = 50
      IWSTRT = 1
      IHSTRT = 1
      IWIDTH = IWEND-IWSTRT
      IHIGHT = IHEND-IHSTRT
      WRITE(6,10)
   10 FORMAT('POSIZIONARE LA CARTA E PREMERE RETURN')
      READ (5,20) IDUMMY
   20 FORMAT(A2)
      RETURN
      END
```

L'algoritmo 1.5 TERMINATE può essere una routine fittizia di termine lavoro (andrebbe aggiunta un'istruzione di STOP).

```
      SUBROUTINE TRMNAT
      RETURN
      END
```

L'algoritmo 2.8 DOMOVE necessita solamente l'aggiornamento delle variabili usate per il posizionamento corrente della penna.

```
      SUBROUTINE DOMOVE(X, Y)
C     SPOSTA LA POSIZIONE CORRENTE DELLA PENNA
      COMMON /PDBLK / PXAT, PYAT
      COMMON /SBLK/ IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
        IHIGHT
      PXAT = AMAX1(FLOAT(IWSTRT),AMIN1(FLOAT(IWEND),
        X*IWIDTH+IWSTRT))
      PYAT = AMAX1(FLOAT(IHSTRT),AMIN1(FLOAT(IHEND),
        Y*IHIGHT+IHSTRT))
      RETURN
      END
```

L'Algoritmo 2.9 DOLINE permette l'ingresso di una linea nel frame buffer, tramite una chiamata alla routine DDA. Il formato corrente del carattere per rappresentare una linea è LINCHR.

```
      SUBROUTINE DOLINE(X, Y)
C     INSERISCE UNA LINEA NEL FRAME BUFFER
      COMMON /CBLK/ LINCHR
      COMMON /SBLK/ IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
        IHIGHT
      COMMON /PDBLK/ PXAT, PYAT
      X1 = PXAT
      Y1 = PYAT
      PXAT = AMAX1(FLOAT(IWSTRT),AMIN1(FLOAT(IWEND),
        X*IWIDTH+IWSTRT))
      PYAT = AMAX1(FLOAT(IHSTRT),AMIN1(FLOAT(IHEND),
        Y*IHIGHT+IHSTRT))
      CALL DDA(X1,Y1,PXAT,PYAT,LINCHR)
      RETURN
      END
```

L'algoritmo 2.18 DOCHAR mette un carattere nel frame buffer. La routine CODE converte un codice di carattere ASCII in una forma compatibile per la stampa.

```
      SUBROUTINE DOCHAR(IOP,X,Y)
C     INSERISCE UN CARATTERE NEL FRAME BUFFER
      COMMON /SBLK /IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
        IHIGHT
      COMMON /FBLK/ IFRAME(90, 50)
      ICXAT = MAX0(IWSTRT,MIN0(IWEND,INT(X*IWIDTH+
```

```
      IWSTRT)))
      ICYAT = MAX0(IHSTRT,MIN0(IHEND,INT(Y*IHIGHT+
        IHSTRT)))
      ICHR = KODE(IOP)
      IFRAME(ICXAT, ICYAT) = ICHR
      RETURN
      END
```

Le inizializzazioni degli algoritmi 2.13, 2.19 e 2.23 possono essere fatte insieme attraverso il seguente sottoprogramma:

```
      SUBROUTINE INITL2
      COMMON /CSZBLK /CHARHT,CHARWD,CHRSPC
      COMMON /DFBLK/ IFREE
      COMMON /SBLK/ IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
        IHIGHT
      COMMON /PINBLK/ XAT,YAT
      CALL INITL1
      IFREE = 1
      CALL DOSTYL(-1)
      CHARWD = 1.0 /FLOAT(IWIDTH)
      CHARHT = 1.0 / FLOAT(IHIGHT)
      CHRSPC = 0.0
      CALL CHARUP(0.0, 1.0)
      XAT = 0.0
      YAT = 0.0
      CALL NEWFRM
      RETURN
      END
```

L'algoritmo 3.5 DOSTYLE cambia il tipo di riempimento e dei contorni, sostituendo il carattere usato nel disegno. Il formato di default è un asterisco (*).

```
      SUBROUTINE DOSTYL(IOP)
      COMMON /PLYBLK/ IPLYCH
      COMMON /CBLK/ LINCHR
      IF (IOP. LT. -20) GO TO 228
      LINCHR = KODE(41-IOP)
      RETURN
  228 IPLYCH = KODE(-IOP+11)
      RETURN
      END
```

La routine FILLIN che viene chiamata nell'Algoritmo 3.15 si serve di una chiamata alla DDA per riempire una linea di scansione.

```
SUBROUTINE FILLIN(X1,Y1,X2,Y2)
COMMON /PLYBLK/ IPLYCH
CALL DDA(X1,Y1,X2,Y2,IPLYCH)
RETURN
END
```

C

Interfacciamento con il PLOT10

Il PLOT10 è un sistema grafico sviluppato dalla *Tektronix Corporation* per l'utilizzo dei suoi terminali grafici. Esso consiste di una serie di sottoprogrammi richiamabili da FORTRAN per disegnare linee e caratteri, eseguire windowing e clipping. In questa appendice vengono fornite alcune routine FORTRAN che dimostrano come il sistema grafico possa essere costruito con il pacchetto PLOT10 utilizzando solo le seguenti risorse di base:

ANCHO	output di caratteri
DRWABS	disegno di linee continue
DSHABS	disegno di linee tratteggiate
ERASE	pulisce lo schermo
FINITT	termina l'esecuzione
INITT	inizializza l'esecuzione
MOVABS	muove il cursore

Le routine sono state provate con un terminale Tektronix 4010 ad una velocità di trasmissione di 120 caratteri al secondo. Lo schermo ha dimensioni 781×1024 pixel, con 35 linee di 74 caratteri per linea.
L'algoritmo 1.2 ERASE si trasforma nella routine ERASE del PLOT10.
L'algoritmo 1.3 DISPLAY può diventare una routine fittizia (in alternativa può diventare una chiamata ad un dispositivo di hard-copy).

```
      SUBROUTINE DISPLY
      RETURN
      END
```

L'algoritmo 1.4 INITIALIZE-1 potrebbe essere così scritto:

```
      SUBROUTINE INITL1
      COMMON /SBLK/ IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
        IHIGHT
```

```
CALL INITT(120)
IWEND = 1023
IHEND = 780
IWSTRT = 0
IHSTRT = 0
IWIDTH = IWEND-IWSTRT
IHIGHT = IHEND-IHSTRT
RETURN
END
```

L'algoritmo 1.5 TERMINATE svolge le funzioni della FINITT del PLOT10.

```
SUBROUTINE TRMNAT
CALL FINITT(0, 0)
RETURN
END
```

L'algoritmo 2.8 DOMOVE in questo contesto deve chiamare la routine MOVABS del PLOT10.

```
SUBROUTINE DOMOVE(X, Y)
COMMON /SBLK /IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
   IHIGHT
IX = MAX1(FLOAT(IWSTRT),AMIN1(FLOAT(IWEND),X*IWIDTH
   +IWSTRT))
IY = MAX1(FLOAT(IHSTRT),AMIN1(FLOAT(IHEND),Y*IHIGHT
   +IHSTRT))
CALL MOVABS(IX,IY)
RETURN
END
```

L'implementazione dell'algoritmo 2.9 DOLINE richiede l'uso dei sottoprogrammi DRWABS e DSHABS per ottenere diversi tipi di linea.

```
SUBROUTINE DOLINE(X,Y)
COMMON /STLBLK/ LINSTL
COMMON /SBLK/ IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
   IHIGHT
IX = MAX1(FLOAT(IWSTRT),AMIN1(FLOAT(IWEND),X*IWIDTH
   +IWSTRT))
IY = MAX1(FLOAT(IHSTRT),AMIN1(FLOAT(IHEND),Y*IHIGHT
   +IHSTRT))
IF (LINSTL .EQ. 1) GO TO 610
CALL DSHABS(IX,IY,LINSTL)
```

```
      RETURN
  610 CALL DRWABS(IX,IY)
      RETURN
      END
```

L'algoritmo 2.18 DOCHAR sostituisce il vero algoritmo DOCHAR tramite la routine ANCHO del PLOT10, nel caso di uscita con caratteri alfanumerici.

```
      SUBROUTINE DOCHAR(IOP,X,Y)
      CALL DOMOVE(X, Y)
      CALL ANCHO(IOP)
      RETURN
      END
```

Le inizializzazioni degli algoritmi 2.13, 2.19 e 2.23 vengono eseguite assieme con la seguente routine:

```
      SUBROUTINE INITL2
      COMMON /CSZBLK/ CHARHT,CHARWD,CHRSPC
      COMMON /DFBLK/ IFREE
      COMMON /SBLK/ IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
        IHIGHT
      COMMON /PINBLK/ XAT,YAT
      CALL INITL1
      IFREE = 1
      CALL DOSTYL(-1)
      CHARWD = 1.0 / 74.0
      CHARHT = 1.0 / 35.0
      CHRSPC = 0.0
      CALL CHARUP(0.0, 1.0)
      XAT = 0.0
      YAT = 0.0
      RETURN
      END
```

L'algoritmo 3.5 DOSTYLE cambia il tipo di riempimento e quello delle linee di poligoni. Per ottenere tipi diversi di linee, uno dei tre numeri è usato come argomento di DSHABS. I differenti tipi di riempimento vengono creati diminuendo le linee di scansione.

```
      SUBROUTINE DOSTYL(IOP)
      COMMON /PLSBLK/ SCNDEC
      COMMON /STLBLK/ LINSTL
      IF (IOP. GE.-1) LINSTL = 1
      IF (IOP. LT.-30) GO TO 116
```

```
      IF (IOP.LT.-1) = LINSTL = 12+MOD(-IOP,4)*2
      RETURN
  116 SCNDEC = (-IOP-29.0)*2.0
      RETURN
      END
```

La routine FILLIN chiamata nell'algoritmo 3.15 svolge le funzioni della MOVE e della LINE.

```
      SUBROUTINE FILLIN(X1,Y1,X2,Y2)
      CALL MOVABS(INT(X1),INT(Y1))
      CALL DRWABS(INT(X2),INT(Y2))
      RETURN
      END
```

D

Interfacciamento con le routine CALCOMP

Le routine CALCOMP sono una serie di sottoprogrammi richiamabili da FORTRAN, scritti dalla *California Computer Products, Inc.*, per poter usare i loro plotter. Le routine sono state adattate anche per altri dispositivi grafici. Per interfacciare il sistema con queste routine servono solo le seguenti risorse di base:

FIN	terminazione
PLOT	controllo della penna
PLOTS	inizializzazione
SYMBOL	disegno di un carattere

In questa appendice vengono elencate alcune routine FORTRAN che dimostrano come il sistema grafico possa essere costruito a partire dal pacchetto CALCOMP. Le routine sono scritte per un plotter avente una superficie di 11 pollici quadrati e 100 pixel per pollice (1 pollice = 2.54 cm). I punti sono indirizzati in unità di pollice. I caratteri hanno la dimensione di un decimo di pollice quadrato, ma sono trattati come se fossero leggermente più ampi per lasciare spazio tra i caratteri.
Per quanto riguarda l'algoritmo 1.2 ERASE, bisogna notare che non è possibile cancellare qualcosa che è stato disegnato; è possibile però far scorrere la carta per posizionarsi su una regione pulita. Questa è l'idea del comando END-OF-PLOT. Il terzo argomento negativo nella routine PLOT assegna l'origine del nuovo posizionamento della carta.

```
SUBROUTINE ERASE
CALL PLOT(16.0,0.0,-3)
RETURN
END
```

L'algoritmo 1.3 DISPLAY può essere semplicemente una routine fittizia.

```
      SUBROUTINE DISPLY
      RETURN
      END
```

L'algoritmo 1.4 INITIALIZE-1 usa le routine CALCOMP PLOTS per inizializzare il sistema di plottaggio. Gli argomenti usati in queste routine dipendono dal sistema che si sta usando. Un esempio di procedura di inizializzazione può essere il seguente:

```
      SUBROUTINE INITL1
      COMMON·/SBLK/ IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
        IHIGHT
      CALL PLOTS(30)
      IWEND = 1100
      IHEND = 1100
      IWSTRT = 0
      IHSTRT = 0
      IWIDTH = IWEND-IWSTRT
      IHIGHT = IHEND-IHSTRT
      RETURN
      END
```

L'algoritmo 1.5 TERMINATE potrebbe essere realizzato alzando prima la penna dal disegno e poi, se necessario, chiamando la routine CALCOMP FIN per terminare il lavoro.

```
      SUBROUTINE TRMNAT
      CALL PLOT(16.0,0.0,-3)
      CALL FIN
      RETURN
      END
```

L'algoritmo 2.8 DOMOVE utilizza una chiamata alla routine PLOT con il terzo argomento uguale a 3. Il fattore 0.01 è necessario per convertire da unità di pixel ad unità di pollice.

```
      SUBROUTINE DOMOVE(X,Y)
      COMMON /SBLK/ IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
        IHIGHT
      X1 = AMAX1(FLOAT(IWSTRT),AMIN1(FLOAT(IWEND),
           X*IWIDTH+IWSTRT))
      Y1 = AMAX1(FLOAT(IHSTRT),AMIN1(FLOAT(IHEND),
           Y*IHIGHT+IHSTRT))
      CALL PLOT(X1*0.01, Y1*0.01,3)
      RETURN
      END
```

L'algoritmo 2.9 DOLINE usa la routine PLOT, con il terzo argomento uguale a 2. C'è solo un tipo di linea implementata (quella continua) ma è possibile rimediare usando la routine DASHP per ottenere linee tratteggiate.

```
SUBROUTINE DOLINE(X,Y)
COMMON /SBLK/ IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
   IHIGHT
X1 = AMAX1(FLOAT(IWSTRT),AMIN1(FLOAT(IWEND),
      X*IWIDTH+IWSTRT))
Y1 = AMAX1(FLOAT(IHSTRT),AMIN1(FLOAT(IHEND),
      Y*IHIGHT+IHSTRT))
CALL PLOT(X1*0.01, Y1*0.01,2)
RETURN
END
```

L'algoritmo 2.18 DOCHAR usa la routine SYMBOL, che disegna stringhe di caratteri con differenti angolature e formati. Tuttavia, in questa implementazione, è possibile usare singoli caratteri con forma e angolazione prefissate.

```
SUBROUTINE DOCHAR(IOP,X,Y)
COMMON /SBLK/ IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
   IHIGHT
X1 = AMAX1(FLOAT(IWSTRT),AMIN1(FLOAT(IWEND),
      X*IWIDTH+IWSTRT))
Y1 = AMAX1(FLOAT(IHSTRT),AMIN1(FLOAT(IHEND),
      Y*IHIGHT+IHSTRT))
ICHR = KODE(IOP)
CALL SYMBOL(X1*0.01, Y1*0.01,0.1,ICHR,0.0,1)
RETURN
END
```

KODE è una funzione che converte il codice ASCII, passato come IOP, in una forma accettabile dalla routine SYMBOL. In questa particolare implementazione, per allineare a sinistra i caratteri è necessario uno scorrimento.
Le inizializzazioni degli algoritmi 2.13, 2.19 e 2.23 possono essere combinate nella seguente routine:

```
SUBROUTINE INITL2
COMMON /IFBLK/IFLIP
COMMON /CSZBLK/ CHARHT,CHARWD,CHRSPC
COMMON /DFBLK/ IFREE
COMMON /SBLK/ IWEND,IHEND,IWSTRT,IHSTRT,IWIDTH,
   IHIGHT
COMMON /PINBLK/ XAT,YAT
CALL INITL1
```

```
      IFLIP = 1
      IFREE = 1
      CALL DOSTYL(-1)
      CHARWD = 0.012
      CHARHT = 0.012
      CHRSPC = 0.0
      CALL CHARUP(0.0,1.0)
      XAT = 0.0
      YAT = 0.0
      CALL NEWFRM
      RETURN
      END
```

La variabile IFLIP può essere usata dalla routine FILLIN per incrementare la velocità di colorazione del poligono.
L'algoritmo 3.5 DOSTYLE dovrebbe cambiare i tipi di linea e di riempimento, ma in questa implementazione è ammesso un solo tipo di linea. I riempimenti vengono modificati decrementando le linee di scansione.

```
      SUBROUTINE DOSTYL(IOP)
      COMMON /PLSBLK/ SCNDEC
      IF (IOP .LT.-30) SCNDEC = (-IOP-29.0)
      RETURN
      END
```

La routine FILLIN chiamata nell'algoritmo 3.15 cerca di riempire i poligoni in modo efficiente con una serie di linee di scansione, disegnando le linee in direzioni alternate.

```
      SUBROUTINE FILLIN(X1,Y1,X2,Y2)
      COMMON /IFBLK/IFLIP
      IFLIP = -IFLIP
      IF (IFLIP.GT.0) GO TO 118
      CALL PLOT(X1*0.01, Y1*0.01,3)
      CALL PLOT(X2*0.01, Y2*0.01,2)
      RETURN
  118 CALL PLOT(X2*0.01, Y2*0.01,3)
      CALL PLOT(X1*0.01, Y1*0.01,2)
      RETURN
      END
```

Bibliografia

Barrett R.C. e Jordan B.W.: *Scan-Conversion Algorithms for a Cell Organized Raster Display* in *Comm. ACM*, vol. 17, n. 3, 1980, pp. 157-163.

Bergeron R.D., Bono P.R., e Foley J.D.: *Graphics Programming Using the CORE System* in *ACM Computing Surveys*, vol. 10, n. 4, 1978, pp. 389-394.

Blinn J.F.: *Models of Light Reflection for Computer Synthesized Pictures* in *Computer Graphics*, vol. 11, n. 2, 1977, pp. 193-198.

Carlbom I. e Paciorek J.: *Planar Geometric Projections and Viewing Transformations* in *ACM Computing Surveys*, vol. 10, n. 4, 1978, pp. 465-502.

Chasen S.H.: *Geometric Principles and Procedures for Computer Graphic Application*, Prentice-Hall, 1978.

Danielsson P.E.: *Incremental Curve Generation* in *IEEE Trans. on Computers*, vol. C-19, 1970, pp. 783-793.

Faux I.D. e Pratt M.J.: *Computational Geometry for Design and Manufacture*, Ellis Horwood, 1981.

Foley J.D. e Wallace V.L.: *The Art of Natural Graphic Man-Machine Conversation* in *Proc. of IEEE*, 1974.

Forrest A.R.: *Curves for Computer Graphics* in *Pertinent Concepts in Computer Graphics*, Univ. of Illinois Press, 1969, pp. 31-47.

Garret M.T.: *Logical Pick Device Simulation Algorithms for the CORE System* in *Computer Graphics*, vol. 13, n. 4, 1980, pp. 301-313.

Giloi W.K.: *Interactive Computer Graphics*, Prentice-Hall, 1978.

Graphics Standards Planning Committee: *Status Report Part II: General Methodology and the Proposed CORE System* in *Computer Graphics*, vol. 13, n. 3, 1979, pp. II/1-179.

Graphics Standards Planning Committee: *Status Report Part III: Raster Extensions to the CORE System* in *Computer Graphics*, vol. 3, n. 3, 1979, pp. III/1-39.

Griffiths J.G.: *Bibliography of Hidden-Line and Hidden-Surface Algorithms* in *Computer Aided Design*, vol. 10, n. 3, 1978.

Hobbs I.C.: *Computer Graphics Display Hardware* in *Computer Graphics and Applications*, vol. 1, n. 1, 1981, pp. 25-32.

Jordan B.W. e Barret R.C.: *A Scan Conversion Algorithm with Reduced Storage Requirements* in *Comm. ACM*, vol. 16, n. 11, 1973, pp. 672-683.

Loceff M.: *A New Approach to High-Speed Computer Graphics: The Line* in *Computer*, vol. 13, n. 6, 1980, pp. 56-66.

Loutrell P.P.: *A Solution to the Hidden-Line Problem for Computer-Drawn Polyhedra* in *IEEE Trans. on Computers*, vol. C-19, 1970, pp. 205-213.

Lucido A.P.: *A Overview of Directed Beam Graphics Display Hardware* in *Computer*, vol. 11, n. 11, 1978, pp. 29-37.

Lucido A.P.: *Software Systems for Computer Graphics* in *Computer*, vol. 9, n. 8, 1976, pp. 23-26.

Newman W.M.: *A Graphical Technique for Computer Input* in *Computer Journal*, vol. 11, 1978, pp. 63-64.

Newman W.M.: *Instance Rectangles and Picture Structure* in *Proc. of the Computer Graphics, Pattern Recognition & Data Structures Conf.*, IEEE, 1975, pp. 297-301.

Newman W.M.: *Display Procedures* in *Comm. ACM*, vol. 14, n. 10, 1971, pp. 651-660.

Newman W.M. e Sproull R.F.: *Principles of Interactive Computer Graphics*, McGraw-Hill, 1979.

Ohlson M.: *System Design Considerations for Graphics Input Devices* in *Computer*, vol. 11, n. 11, 1978, pp. 9-19.

Peckham H.D.: *Computer Graphics: Three-Dimensional Projections*, Hewlett-Packard, 1974.

Phong B.T.: *Illumination for Computer-Generated Pictures* in *Comm. ACM*, vol. 18, n. 6, 1975, pp. 311-317.

Preiss R.B.: *Storage CRT Display Terminals, Evolution and Trends* in *Computer*, vol. 11, n. 11, 1978, pp. 20-28.

Rogers D.F.: *Procedural Elements for Computer Graphics*, McGraw-Hill, 1985.

Rogers D.F. e Adams J.A.: *Mathematical Elements for Computer Graphics*, McGraw-Hill, 1976.

Slotow H.G.: *Plasma Displays* in *IEEE Trans. on Electron Devices*, vol. ED-23, n. 7, 1976, pp. 760-772.

Sproull R.F. e Sutherland I.E.: *A Clipping Divider* in *Proc. Fall Joint Computer Conf.*, AFIPS Press, 1968.

Sutherland I.E. e Hodgman G.W.: *Reentrant Polygon Clipping* in *Comm. ACM*, vol. 17, n. 1, 1974, pp. 32-42.

Sutherland I.E., Sproull R.F. e Schumacker R.A.: *Sorting and the Hidden-Surface Problem* in *Proc. National Computer Conf.*, vol. 42, AFIPS Press, 1973, pp. 685-693.

Sutherland I.E., Sproull R.F. e Schumacker R.A.: *A Characterization of Ten Hidden-Surface Algorithms* in *Computing Surveys*, vol. 6, n. 1, 1974, pp. 1-55.

Indice analitico

Accesso diretto alla memoria (DMA) *21*
ADD-EVENT-Q *243*
ADD-STRING-Q *244*
ADD-TO-LISTS *404*
ALGOL *17*
Algoritmi, struttura formale *17*
Algoritmo del pittore *369*
Analizzatore digitale differenziale (DDA) *36*
and, operatore *19*
Angolare, coefficiente *29*
Angoli
— di incidenza *441*
— di riflessione *442*
Animazione *159*
Apertura dei segmenti del display file *171*
Approssimante, curva *463*
ARCDDA *461*
ARCTAN *19*
Arco *460*
Argomenti *18*
Array *18*
ASCII, codice dei caratteri *70*
Assegnamento *20*
Assi coordinati *27*
Associativa, proprietà *129*
Assonometrica, proiezione parallela *318*
Attributi dei segmenti del display file *167*
AVERAGE-POSITION *449*
AWAIT-BUTTON *249*
AWAIT-BUTTON-GET-LOCATOR *263*
AWAIT-EVENT *245*
AWAIT-PICK *250*

B-spline *475*
— cubiche, funzioni *476*
B-SPLINE-POLY-ABS-3 *480*
BACK-FACE algoritmi *365*
begin *20*
BITBLT *190*
Blending function *465*
Brillantezza *450*
BUILD-TRANSFORMATION *152, 182*

Cabinet, proiezione parallela *320*
CALCOMP, routine di interfaccia *505*
Cambiamento di nome ai segmenti del display file *176*
Campionamento, punto *462*
Campionato, dispositivo *251*
Cancellazione dei segmenti del display file *172*
Caratteri
— generazione *41*
— parametri *71*
Cartesiane, coordinate *28*
Cavaliera, proiezione parallela *320*
Centro
— di curvatura *460*
— di proiezione *297*
CHANGE-OP-CODE *419*
CHECK-SIDES *424*
CHECK-STYLE *407*
Chiusura dei segmenti del display file *171*
— dei poligoni *216*
CHOP-IN-TO-OUT *420*
CHOP-OUT-IN-OUT *421*

CHOP-OUT-TO-IN *418*
Ciclica, sovrapposizione *435*
Ciclo di polling *236*
Circonferenza *460*
CLIP *218, 354*
CLIP-BACK *350*
CLIP-BOTTOM *213, 348*
CLIP-FRONT *351*
CLIP-LEFT *210, 346*
CLIP-RIGHT *213, 348*
CLIP-TOP *213, 349*
CLIPPED-X-OR-Y *346*
CLIPPED-Z *345*
Clipping *208*
— con un piano *343*
— di poligoni *214*
— di volumi *335*
— generalizzato *221*
— piani di *337*
— piano anteriore *338*
— piano posteriore *338*
— piramide *335*
— senza l'operazione di divisione *220*
— tridimensionale *331*
CLOSE-SEGMENT *172, 377*
Coda di eventi *238*
CODE *76*
Codice dei caratteri ASCII *70*
Codice operativo *60*
Coefficiente
— angolare *29*
— di levigatezza *444*
— di riflessione *438*
Commenti *18*
Commutativa, proprietà *128*
COMPARE-ALL-TRIANGLES *404*
COMPARE-AT-ENDS *391*
COMPARE-CONTAINED *398*
COMPARE-POINT-PLANE *396*
COMPARE-SIDES *389*
Compenetrazione di facce *435*
Computer Graphics interattiva *231*
— standardizzazione *14*
Concavo, poligono *427*
Confronto tra triangoli *384*
Contenimento, controllo *392*
— di interi triangoli *397*
— di un punto in triangoli *392*
— in un poligono, controllo *97*
Controllo di contenimento *392*
— di contenimento in un poligono *97*
Convesso, poligono *94*
— vertice *362*
Coordinate *362*
— cartesiane *28*
— del dispositivo *60*
— del piano di proiezione *303*
— del piano di proiezione, trasformazione *303*
— normalizzate *59*
— omogenee *139*
— tridimensionali *276*
COPY-LIGHT *454*
CORE sistema grafico *15, 487*
COS *19*
Coseno *134*
Costanti *18*
CREATE-SEGMENT *171, 207, 253, 316*
Creazione dei segmenti del display file *170*
CROSS-CHECK *412*
CRT *53*
Cubiche, funzioni *B*-spline *476*
Cursore *256*
Curva approssimante *463*
Curvatura, centro *460*
Curve *459*
-- generazione *459*
— interpolazione *462*
CURVE-ABS-3 *472*

DDA *37*
DECODE *75*
Deformazione di taglio *146*
DELETE-ALL-SEGMENTS *175*
DELETE-SEGMENT *173*
Destrorsa, terna di riferimento *277*
Diffusa, illuminazione *437*
Digitalizzatore *233*
Dimetrica, proiezione parallela *320*
DISABLE-ALL *242*
DISABLE-GROUP *241*
DISPLAY *44*
Display file *54*
— interprete *58*
— istruzioni *58*
— segmenti *167*
— struttura *60, 186*
DISPLAY-FILE-ENTER *63, 79, 218, 316*
Dispositivo
indipendenza *58*

— campionato *251*
— controllato da eventi *235*
— coordinate *60*
DMA Accesso diretto alla memoria *52*
do *21*
DO-TRANSFORMATION *150*
DOCHAR *75*
DOLINE *65*
DOMOVE *65*
DOPOLYGON *101*
DOPROJECTION *353*
DOPROJECTION-TO-C *374*
DOSPLIT *431*
DOSTYLE *78, 99*
Dragging *270*
DVST *53*

Eco *262*
Ellisse *460*
else *20*
ENABLE-GROUP *240*
end *20*
END-B-SPLINE *479*
END-CURVE *472*
Equazioni parametriche
— di curve *464*
— di rette *37*
ERASE *43*
EVENT *242*
Eventi, coda *238*
— dispositivo controllato da *235*

Facce, compenetrazione *435*
Facce nascoste *360*
Faccia anteriore e posteriore *362*
Fattore di scala *131*
FILL-POLYGON *104*
FILL-SCAN *115*
FILLIN *115*
FLUSH-ALL-EVENTS *248*
for *21*
Frame buffer *36*
— visualizzazione *42*
Funzioni *21*
— *B*-spline cubiche *476*
— blending *462*
— di due variabili *276*
— di ombreggiatura *437*
— incorporate *19*

GENERATE-CHAR *76*
Generazione
— di caratteri *41*
— di curve *459*
— di vettori *36*
Geometrico, ordinamento *401*
Gestione delle interruzioni *236*
GET-KEYBOARD-DATA *249*
GET-PICK-DATA *249*
GET-POINT *62*
GET-TRANSFORMED-POINT *153*
GETCHR *75*
GETQ *247*
Globali, variabili *18*
Grafica su un terminale non-grafico *495*
Graphics Standards Planning Committee *14*

HALF-PLANE *393*
HIDDEN-SURFACE-CHECK *377*

Identificatori *18*
Identità, matrice *129*
Identità trigonometriche *135*
IDENTITY-MATRIX *148*
IDENTITY-PARAMETERS *156, 182*
if *20*
Illuminazione
— diffusa *437*
— sorgente puntiforme *440*
Immagine
— spazio *198*
— trasformazione di *203*
IN-A-LINE *383*
Incidenza, angolo *441*
INCLUDE *111*
Indipendenza dal dispositivo *58*
INITIALIZE-1 *44*
INITIALIZE-2 *80*
INITIALIZE-2A *69*
INITIALIZE-2B *76*
INITIALIZE-3 *116*
INITIALIZE-4 *156*
INITIALIZE-5 *183*
INITIALIZE-6 *220*
INITIALIZE-7 *255*
INITIALIZE-8 *317*
INITIALIZE-9 *355*
INITIALIZE-10A *369*
INITIALIZE-10B *376*
INITIALIZE-11 *455*

Inking *265*
Input-output, istruzioni *22*
Inserimento di un poligono *100*
INSIDE *394*
INT *19*
INTENSITY *453*
Interattiva, Computer Graphics *231*
Intercetta *29*
Interpolazione di curve *462*
INTERPRET *66, 75, 79, 99, 153*
Interprete del display file *58*
Interruzione *236*
— risposta alla *236*
INTERSECTION-PAIR *415*
INTERSECTION-POINT *414*
Intersezione tra rette *30*
IS-BACK-FACE *366, 447*
Isometrica, proiezione parallela *318*
Istruzione *19*
— composta *20*
— del display file *58*
— di controllo e loop *20*
— di input/output *22*

Joystick *232*

Lambert, legge *438*
Lati *94*
LEFT-IN-FRONT *420*
LEFT-MOST *368*
LENGTH *75*
Levigatezza, coefficiente *444*
LINE comando assoluto *55*
LINE comando relativo *56*
LINE-ABS-2 *63*
LINE-ABS-3 *281*
LINE-CODE *78*
LINE-REL-2 *64*
LINE-REL-3 *282*
Linea di scansione *101*
Linee nascoste
(vedi anche Triangoli) *409*
Liste concatenate *186, 401*
LOAD-POLYGON *107, 154*
Locator *232*
Luce di sfondo *437*
Lunghezza di un segmento di retta *31*

MAKE-CLIPPING-CONSTANTS *341*
MAKE-CLIPPING-TESTS *342*
MAKE-CURVE *468*
MAKE-PARALLEL-TRANSFORMATION *313*
MAKE-PERSPECTIVE-TRANSFORMATION *313*
MAKE-PICTURE-CURRENT *68, 152, 185*
MAKE-SHADE *451*
MAKE-VIEW-PLANE-TRANSFORMATION *312*
Matrice *128*
— di proiezione parallela *296*
— di proiezione prospettica *299*
— di rotazione *139*
— di scala *133*
— di trasformazione *141*
— di traslazione *141*
— identità *129*
Matrici, prodotto di *128*
MAX *19*
Medio, punto *32*
Memorizzazione dei segmenti del display file *183*
Menu *271*
MIN *19*
MINIMAX *385*
Minimax, test *385*
MOVE comando *57*
MOVE-ABS-2 *63*
MOVE-ABS-3 *281*
MOVE-REL-2 *64*
MOVE-REL-3 *281*
MULTIPLY-IN-ROTATION *150*
MULTIPLY-IN-SCALE *149*
MULTIPLY-IN-TRANSLATION *149*

Nascoste, linee
(vedi anche Triangoli) *409*
NEW-FRAME *68*
NEW-TRANSFORMATION-3 *287*
NEW-VIEW-2 *206*
NEW-VIEW-3 *316, 341, 455*
NEXT-SECTION *469*
Normale al piano di proiezione *300*
Normalizzate, coordinate *59*
not, operatore *19*

Obliqua, proiezione parallela *320*
Oggetto, spazio *198*
Ombre *456*
Ombreggiatura *437*
— con luce diffusa, *437*
— con sorgente puntiforme *440*
Omogenee, coordinate *139*
OPCODE (codice operativo) *60*
Operando *60*
Operatore *18*
or, operatore *19*
Ordinamento geometrico *401*
Origine degli assi *27*
Ottica, penna *235*

Pagina *186*
PARALLEL-TRANSFORM *297*
Parallela, proiezione *294*
Parallele, rette *30*
Parametri dei caratteri *71*
Parametriche, equazioni
— di curve *464*
— di rette *37*
Penna ottica *235*
Perpendicolari, rette *33*
PERSPECTIVE-TRANSFORM *300*
Piani di clipping *337*
Piano *27*
— clipping *337*
— di clipping anteriore *338*
— di clipping posteriore *338*
— di proiezione *300*
— di proiezione, coordinate *303*
— di proiezione, normale *300*
Pick *239*
PICK-SEARCH *261*
Piramide di clipping *335*
Pitagora, teorema *31*
Pittore, algoritmo *369*
Pixel *35*
Plasma panel *53*
PLOT10, sistema *501*
Plotter *53*
Point plotting *263*
Poligono *93*
— arrotondamento *473*
— chiusura *216*
— clipping *214*
— concavo *427*
— controllo di contenimento *97*
— convesso *94*
— inserimento *100*
— lato *94*
— riempimento *101*
— scomposizione in triangoli *378*
— vertice di un *94*
Polling, ciclo *236*
POLY-INSERT *108*
POLYGON-ABS-2 *95*
POLYGON-ABS-3 *282*
POLYGON-CLIP *216, 353, 365, 373*
POLYGON-REL-2 *96*
POLYGON-REL-3 *283*
POLYGON-SPLIT *381, 432*
POP-C *417*
Posizione relativa ad una retta *222*
Potenziometri *232*
print *22*
Procedure *21*
— di visualizzazione *157*
Prodotto di matrici *128*
— scalare *280*
— vettoriale *361*
Profondità *399*
Proiezione
— centro *297*
— coordinate del piano *303*
— dimetrica *320*
— direzione verticale del piano *300*
— isometrica *318*
— obliqua *320*
— parallela *294*
— parallela assonometrica *318*
— parallela cabinet *320*
— parallela cavaliera *320*
— parallela, matrice *296*
— parallela obliqua *320*
— piano *300*
— prospettica *297*
— prospettica a due punti di fuga *322*
— prospettica a tre punti di fuga *322*
— prospettica a un punto di fuga *322*
— prospettica, matrice *311*
— punto di riferimento *303*
— trasformazione *314*
— trimetrica *320*
Proprietà
— associativa *129*
— commutativa *128*
Prospettica, proiezione *297*

Pseudo-ALGOL *18*
Pseudo-display file *54*
Pulsanti *236*
Puntatore *401*
Puntatori, liste a *401*
Punto, definizione *25*
— di campionamento *462*
— di fuga *322*
— di riferimento della proiezione *303*
— medio di un segmento *32*
PUSH-TRIANGLE *417*
PUT-IN-B *380*
PUT-IN-C *374*
PUT-IN-D *383*
PUT-IN-SM *470*
PUT-IN-T *216, 352*
PUT-POINT *62, 183*

Raggio di un arco *460*
Raster, tecniche *188*
— terminali di tipo *52*
read *22*
READ-LOCATOR *251*
READ-VALUATOR *252*
RENAME-SEGMENT *176, 254*
Reperibilità, attributi *252*
Retroazione *263*
Retta, definizione *25*
— equazioni *29*
— intersezione *30*
— posizione relativa *222*
— segmento *31*
Rette
— parallele *30*
— perpendicolari *33*
return *21*
return-error *21*
Riempimento di poligoni *104*
Riflessione *442*
— angolo *442*
— coefficiente *438*
— speculare, funzioni di ombreggiatura *442*
Risoluzione *35*
Risposta all'interruzione *236*
ROTATE *151, 182*
ROTATE-X-3 *287*
ROTATE-Y-3 *288*
ROTATE-Z-3 *288*
Rotazione *127, 289*
— matrice *138, 144, 294*
Rubber band line *266*

SAVE-CLIPPED-POINT *215, 352*
SAVE-STYLE *375*
Scala, fattore *131*
— trasformazione *130*
Scalare, prodotto *280*
SCALE *151, 181*
Scansione, linea *101*
Schermo, cursore *256*
Scomposizione di poligoni in triangoli *378*
Segmenti del display file *167*
— apertura *171*
— attributi *167*
— cambiamento del nome *176*
— cancellazione *172*
— chiusura *171*
— creazione *170*
— memorizzazione *183*
— senza nome *170*
— tabelle *168*
— visibilità *178*
— visualizzazione *183*
Segmento di retta *31*
— lunghezza *31*
— punto medio *32*
Selector *232*
SEND-B-TO-DF *425*
SEND-TO-DF *407*
Seno *134*
SET-B-SPLINE *477*
SET-BACK-PLANE-CLIPPING *336*
SET-CHARSPACE *73*
SET-CHARUP *73*
SET-DETECTABILITY *253*
SET-FILL *99*
SET-FILL-STYLE *99, 376*
SET-FRONT-PLANE-CLIPPING *336*
SET-HIDDEN-LINE-REMOVAL *365*
SET-IMAGE-TRANSFORMATION *180*
SET-IMAGE-TRANSLATION *179*
SET-LIGHT *454*
SET-LINESTYLE *78, 375*
SET-OBJECT-SHADE *453*
SET-PARALLEL *308*
SET-PERSPECTIVE *309*
SET-POINT *175*
SET-SHADING *448*

SET-SMOOTH *467*
SET-VIEW-DEPTH *336*
SET-VIEW-DISTANCE *307*
SET-VIEW-PLANE-NORMAL *306*
SET-VIEW-REFERENCE-POINT *305*
SET-VIEW-UP *307*
SET-VIEWPORT *204*
SET-VISIBILITY *179*
SET-WINDOW *204*
Sfondo, luce *440*
SHIFT-BUFFER *384*
SIDE-IS-DONE *423*
SIGNOF *422*
Simmetria *145*
Simmetrico, analizzatore differenziale digitale *36*
Simulatore di volo *323*
SIN *19*
Sinistrorsa, terna di riferimento *276*
Sistema grafico CORE *15, 487*
SMOOTH-POLY-ABS-3 *473*
Sorgente puntiforme *440*
SORT-AND-SAVE-TRIANGLES *408*
SORTED-ENTRY *405, 410*
Sovrapposizione ciclica *435*
Sovrapposizione di lati di triangoli *388*
Spazio-immagine *198*
Spazio-oggetto *198*
Speculare, riflessione *442*
SPLIT-INTO-TRIANGLES *379*
SPLIT-VERTEX *429*
SQRT *19*
Standardizzazione della Computer Graphics *14*
START-B-SPLINE *479*
START-CURVE *470*
Superfici
— nascoste concave *427*
— nascoste convesse *360*
— ricerca *376*

Tabelle per i segmenti del display file *168*
Taglio, trasformazione *146*
Tastiera *239*
Tavoletta grafica *233*
Tecniche raster *188*
Teorema di Pitagora *31*
Terminale non grafico *495*
Terminali a vector refresh *54*
— di tipo raster *52*
TERMINATE *45*
Terna di riferimento destrorsa *277*
— sinistrorsa *277*
Test del minimax *385*
Testo *69*
TEXT *74*
Text, primitiva di tipo *69*
then *20*
Tipo di linea *77*
to *21*
TRANSLATE *151, 181*
TRANSLATE-3 *287*
Trasformazione *127*
— di immagine *179*
— di rotazione *144*
— di scala *130*
— di simmetria *145*
— di taglio *146*
— di traslazione *141*
— di visualizzazione *201, 314*
— tridimensionale *283*
Traslazione *139, 284*
Trasparenza *456*
TRIANGLE-COMPARE *399*
Triangoli, confronto *384*
— contenimento *397*
— contenimento di un punto *392*
— copertura di linee *411*
— ordinamento *404*
— scomposizione di poligoni *378*
— sovrapposizione di lati *388*
Tridimensionali, coordinate *276*
Trigonometriche, identità *135*
Trimetrica, proiezione parallela *320*
Tubo a raggi catodici (CRT) *53*

UNITY *449*
UPDATE-X-VALUES *112*

Valuator *239*
Variabili *18*
— funzioni di due *276*
— globali *18*
— locali *18*
Vector refresh, terminali *54*
Vertice *94*
Vettore *34*
— normale *279*
Vettori, generazione di *36*
— prodotto scalare *280*

— prodotto vettoriale *361*
VIEW-DISTANCE (parametro) *303*
VIEW-PLANE-TRANSFORM *314*
VIEWING-TRANSFORM *208*
Viewport *200*
Visualizzazione dei segmenti del display file *183*
Volume di visualizzazione *335*
Volumi, clipping *335*

while *21*
Window *200*
Windowing *197*
— multiplo *224*

X, asse *27*
XCHANGE *114*
XSORT *114*
|X| *19*

Y, asse *27*

Z, asse *277*
Z-MINIMAX *387*

In preparazione nella stessa serie:

88 7700 602 1	O. Lecarme e J.L. Nebut, *Pascal - Guida per programmatori*
88 7700 603 X	M. McGilton e R. Morgan, *Il sistema operativo UNIX*
88 7700 604 8	E. Rich, *Intelligenza Artificiale*

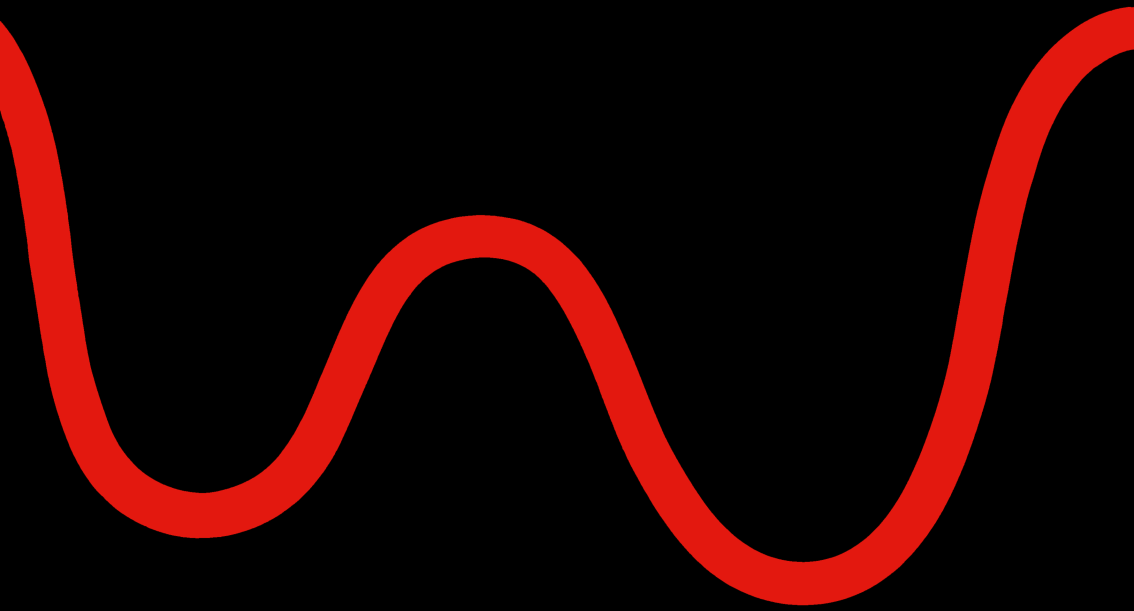
Altri titoli pubblicati in lingua italiana:

88 7700 001 5	J. Heilborn e R. Talbott, *Guida al Commodore 64*
88 7700 002 3	C.A. Street, *La gestione delle informazioni con lo ZX Spectrum*
88 7700 003 1	T. Woods, *L'Assembler per lo ZX Spectrum*
88 7700 004 X	R. Jeffries, G. Fisher e B. Sawyer, *Divertirsi giocando con il Commodore 64*
88 7700 005 8	G. Bishop, *Progetti hardware con lo ZX Spectrum*
88 7700 006 6	H. Mullish e D. Kruger, *Il BASIC Applesoft*
88 7700 007 4	N. Williams, *Progettazione di giochi d'avventura con lo ZX Spectrum*
88 7700 008 2	H. Peckham, *Il BASIC e il PC-IBM in pratica*
88 7700 009 0	H. Peckham, *Il BASIC e il Commodore 64 in pratica*
88 7700 010 4	S. Nicholls, *Tecniche avanzate in Assembler con lo ZX Spectrum*
88 7700 011 2	K. Skier, *L'Assembler per il Commodore 64 e il VIC-20*
88 7700 012 0	S. Kamins e M. Waite, *Programmazione umanizzata in Applesoft*
88 7700 013 9	A. Pennell, *Guida allo ZX Microdrive e all'Interface 1*
88 7700 015 5	P. Cohen, *Grafica e animazione con gli Apple II*
88 7700 016 3	C. Duff, *Guida al Macintosh*
88 7700 017 1	G. Kane, *Il manuale MC68000*
88 7700 018 X	P. Hoffman e T. Nicoloff, *Il manuale MS-DOS*
88 7700 020 1	S. Nicholls, *Grafica avanzata con lo ZX Spectrum*
88 7700 021 X	L.J. Graham e T. Field, *Guida al PC-IBM*
88 7700 022 8	T. Field, *Come usare MacWrite e MacPaint*
88 7700 024 4	H. Peckham, *Il BASIC e gli Apple II in pratica*
88 7700 025 2	C. Morgan e M. Waite, *Il manuale 8086/8088*
88 7700 027 9	G. Mainis, *Il manuale ProDOS*
88 7700 030 9	W. Ettlin e G. Solberg, *Il BASIC Microsoft*

Steven Harrington

Computer Graphics

Corso di programmazione



Lire 39 000
(IVA 2% inclusa)

ISBN 88 7700 601 3

HARRINGTON COMPUTER GRAPHICS