1

Linii şi Suprafeţe Ascunse: Metode de Sortare în Adâncime

1 Introducere

Multe obiecte tridimensionale au suprafeţe care sunt poligoane plane, de exemplu cuburi, piramide, prisme etc. Multe obiecte mai complexe, cum ar fi o casă, sunt compuse din acestea. Chiar şi atunci când un obiect conţine suprafeţe curbe, de obicei acestea pot fi foarte bine aproximate prin poligoane plane, care să simuleze suprafaţa reală. De exemplu, suprafaţa unui cilindru poate fi reprezentată prin mai multe dreptunghiuri. Atunci când o asemenea aproximare este nepotrivită, descrierea suprafeţei curbe trebuie realizată printr-o formulă matematică în trei variabile. Deseori, totuşi, o suprafaţă poate fi reprezentată prin poligoane plane, astfel încât aceasta să apară ca fiind curbă.

Dacă putem defini un obiect sau o scenă grafică utilizând numai poligoane plane, atunci algoritmul de afişare poate beneficia de acest lucru şi afişarea se poate face trasând laturile poligoanelor componente sau prin reprezentarea poligoanelor ca suprafeţe umplute. Afişarea obiectelor care într-adevăr au suprafeţe curbe este mult mai complexă şi nu face obiectul studiului de faţă.

Pe parcursul acestui capitol obiectele în spaţiu sunt considerate ca având suprafeţe poligonale plane. Când un asemenea obiect este afişat numai prin trasarea laturilor, rezultatul este un model *cadru-de-sârmă* (*wireframe*). Aceasta este cea mai simplă reprezentare şi modelul wireframe este utilizat şi atunci când obiectul real este opac. Totuşi, o asemenea reprezentare a unui obiect solid poate să apară ambiguă, întrucât se reprezintă toate laturile obiectului, inclusiv cele care nu sunt vizibile. Figura 1.1 este un exemplu în acest sens. Multe din aceste ambiguităţi dispar dacă liniile ascunse sunt îndepărtate. Aceasta face ca obiectul să aibă şi adâncime.

Distingerea laturilor vizibile şi a celor invizibile ale unui obiect este dificilă şi necesită calcule laborioase. Este mult mai uşor să determinăm poligoanele care nu trebuie reprezentate, decât să găsim laturile lor. Totuşi, la început s-au depus eforturi în găsirea tehnicilor de îndepărtare a liniilor ascunse şi nu a suprafeţelor ascunse. Aceasta deoarece iniţial toate display-urile erau vectoriale şi acestea nu pot reprezenta suprafeţe pline, operaţie simplă

pentru display-urile raster de astăzi. Atunci când se reprezintă un obiect utilizând suprafeţe pline, problema afişării se reduce la problema simplă a eliminării suprafeţelor ascunse.

Chiar şi aşa, algoritmii de îndepărtare a liniilor ascunse merită atenţie, nu atât pentru că încă mai există display-uri vectoriale, ci pentru că plotter-ele sunt *fizic* dispozitive vectoriale şi au probleme în afişarea suprafeţelor solide. Algoritmii de îndepărtare a suprafeţelor nu sunt adecvaţi în acest caz.

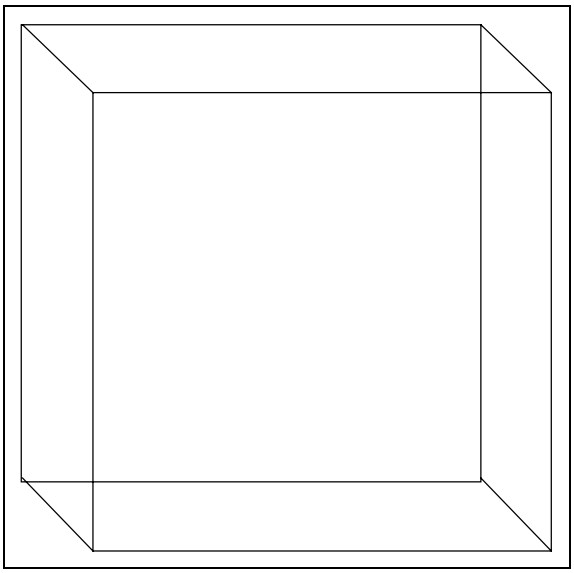

Figura 1.1 Model wireframe ambiguu

Unul din scopurile graficii pe calculator este generarea de imagini ale obiectelor din lumea reală cât mai realist posibil. Un alt scop este realizarea de imagini abstracte din imaginaţie (artă pe calculator). Am putea dori să modelăm contururi desenând cu mâna liberă. O aplicaţie importantă este proiectarea asistată de calculator, în care obiectele sunt specificate complet. În toate cazurile, contururile de afişat sunt de obicei complexe.

Teoretic, descrierea completă a unui contur 3D oarecare poate necesita o infinitate de triplete de numere pentru a-i defini suprafaţa. La cealaltă extremă, o sferă poate fi descrisă printr-o singură formulă matematică. La fel pentru un tor, ca şi pentru alte contururi ideale. Un cilindru este ceva mai complicat de descris. Toate aceste obiecte pot fi aproximate prin poligoane plane cu orice fineţe, dar o asemenea reprezentare devine incomodă când numărul de faţete este prea mare.

Uneori este posibil să construim un obiect complex din mai multe componente simple; uneori o componentă poate fi "extrasă" din ansamblu. Punctele de pe suprafaţa fiecărei componente sunt atunci descrise prin formula matematică ataşată componentei. Construirea reprezentării unui contur în această manieră poartă numele de *modelare solidă*. Unele sisteme de CAD oferă o modalitate interactivă pentru modelarea solidă. Nu toate tipurile de obiecte din lumea reală sunt uşor de construit prin modelare solidă.

Multe din formele pe care dorim să le afişăm sunt prea complexe pentru a fi descrise printr-un asemenea set de formule matematice. În asemenea cazuri, se obişnuieşte să se precizeze un număr limitat de puncte importante de pe suprafaţă şi apoi se generează alte puncte de pe suprafaţă prin interpolare sau aproximare. Este un procedeu intermediar între utilizarea contururilor simple şi utilizarea tripletelor de coordonate.

Să considerăm un exemplu clasic. Dacă dorim să reprezentăm un ceainic, nu există forme simple ideale ale căror formule să descrie întregul ceainic. Dacă utilizăm modelarea solidă, va trebui să construim ceainicul din mai multe forme ideale diferite (cilindri, conuri, sfere, toruri etc). Întrucât numărul componentelor este foarte mare, acest procedeu nu prezintă nici un avantaj. De asemenea, nu este posibil să descriem ceainicul prin înşirarea a peste 1000 de triplete de numere. Soluţia constă în definirea câtorva puncte de pe suprafaţa obiectului şi specificarea tipului suprafeţei care va interpola corect aceste puncte. Sprafeţele bicubice sunt foarte adecvate pentru asemenea exemple.

O altă modalitate este construirea interactivă. Acesta este un lucru obişnuit pentru

forme care nu sunt copia fidelă a unui obiect din lumea reală. Pentru aceasta se precizează mai întâi un număr limitat de puncte care aproximează conturul formei dorite. Apoi se calculează şi se afişează suprafeţele care interpolează sau aproximează aceste puncte. Dacă rezultatul nu este satisfăcător, putem modifica poziţia unor puncte, putem adăuga sau elimina puncte sau să impunem restricţii suprafeţei de interpolare. Se continuă aceste operaţii până se obţine forma dorită.

Atunci când generăm o suprafaţă pe baza câtorva puncte, numim aceasta *formă construită sintetic*. Întrucât imaginea rezultată poate avea proprietăţi diferite de obiectele din lumea reală, această metodă face reprezentarea posibilă. Asemenea sinteză a formelor este des întâlnită în CAD.

Înainte de a discuta despre îndepărtarea de linii şi faţete este necesar să vedem cum poate fi reprezentat intern un obiect 3D. Există două metode uzuale: prin reţele de poligoane şi prin suprafeţe parametrice bicubice. Vom studia în continuare reţelele de poligoane.

2 Reţele de Poligoane

O *reţea de poligoane* ("polygon mesh") este o modalitate de a descrie aproape orice obiect tridimensional din lumea reală, dar metoda este mai potrivită pentru obiecte care conţin multe suprafeţe plane şi laturi drepte. În această categorie intră clădiri, cuburi, cutii. Totuşi, metoda reţelei de poligoane nu se limitează numai la obiecte ale căror suprafeţe sunt plane. În acest fel poate fi descris şi un obiect cu suprafaţă curbă. Reţeaua de poligoane iniţială poate fi apoi netezită prin aproximarea suprafeţelor. Pe de altă parte, poate fi prezentată astfel încât să pară o suprafaţă curbă netedă, chiar dacă este de fapt o mulţime de poligoane (vom discuta despre aceasta mai târziu, când va fi vorba despre umbriri).

O reţea de poligoane nu este doar un set de poligoane, fără nici o legătură între ele. Poligoanele constitutive au proprietatea de *adiacenţă*. O reţea de poligoane trebuie să reflecte această proprietate şi să conţină secvenţa de *vertex*-uri (vârfuri) sau laturi care compun poligoanele individuale. Există mai multe modalităţi de reprezentare a unei reţele de poligoane; fiecare prezintă avantaje şi dezavantaje.

O reţea de poligoane trebuie să permită:

- identificarea unui anumit poligon din reţea;
- identificarea tuturor laturilor aparţinând unui poligon;
- identificarea acelor poligoane care au comună o anumită latură;
- identificarea vertex-urilor (capetelor) oricărei laturi;
- modificarea reţelei;
- afişarea reţelei.

Orice reprezentare a unei reţele trebuie să îndeplinească cerinţele de mai sus. Totuşi, calitatea reprezentării este determinată de uşurinţa în obţinerea informaţiilor. În plus, mai contează viteză şi spaţiul ocupat. Vom descrie în continuare două tipuri de reţele de poligoane.

2.1 Reţele de Poligoane Explicite

Într-o *reţea de poligoane explicite* fiecare vertex este memorat o singură dată ca triplet,

într-o tabelă de vertex-uri. Apoi definim un poligon ca o secvenţă de asemenea vertex-uri. Aceasta se poate face prin definirea poligoanelor ca liste înlănţuite de pointeri în tabela de vertex-uri. Figura 1.2 ilustrează acest lucru; structura de date corespunzătoare este prezentată în Figura 1.3.

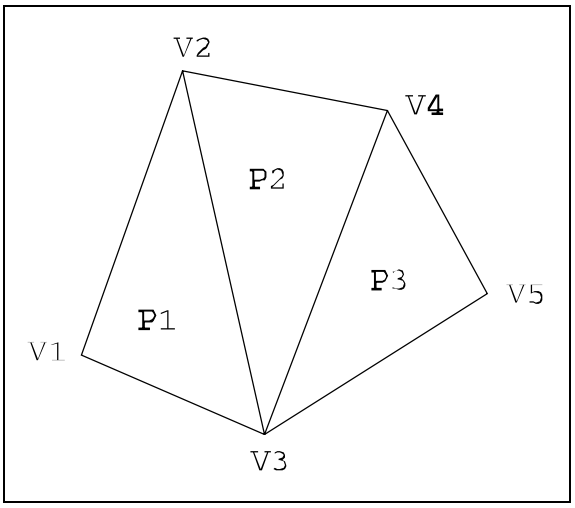

Figura 1.2 Reţea de poligoane.

Această reprezentare a unei reţele are avantajul că ocupă minimum de memorie. Modificarea reţelei este simplă şi eficientă. Pentru a modifica un vertex trebuie modificat doar tripletul de numere din tabelă. Ştergerea sau adăugarea de poligoane este de asemenea simplă.

Acest procedeu ridică însă o problemă: dacă trebuie determinate poligoanele care au comună o latură, atunci trebuie parcursă toată lista de poligoane pentru a vedea dacă conţin sau nu acea latură. La fel dacă trebuie determinate toate poligoanele care au comun un anumit vertex. În cazul reţelelor de dimensiuni reduse, aceasta nu este o problemă, dar dificultăţile cresc dacă reţeaua este mai mare. Mai mult, afişarea reţelei necesită parcurgerea întregii liste de poligoane şi conectarea vertex-urilor prin linii drepte. Aceasta afişează corect reţeaua, dar aproape fiecare latură este trasată de două ori. Din nou, acest lucru nu deranjează în cazul reţelelor mici, dar pentru o reţea mare poate să conteze faptul că viteza de afişare se înjumătăţeşte. Acest lucru ar putea să deranjeze în cazul animaţiei sau a simulatoarelor de zbor.

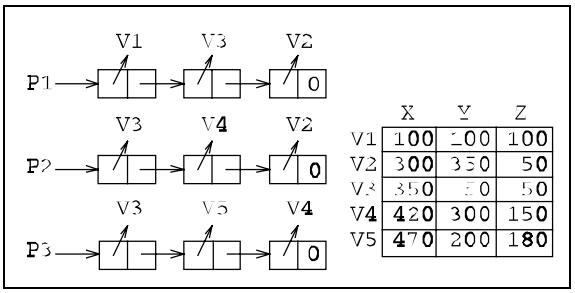

Figura 1.3 Structura de date.

Un exemplu de implementare Atunci când toate poligoanele sunt triunghiuri, reprezentarea internă este simplă dar foarte utilă. O înregistrare poate să conţină informaţii despre cele trei vertex-uri. În locul pointerilor putem memora indicii în tabloul de vertex-uri (în ordine inversă acelor de ceasornic). În înregistrare s-ar putea memora mai multă informaţie, dacă aceasta este util pentru aplicaţie: coeficienţii planului triunghiului, coeficienţii transformării perspectivă, culoarea sau iluminarea triunghiului, dacă este sau nu o faţetă ascunsă etc.

Se dă mai jos un exemplu de implementare pentru o reţea de triunghiuri:

```
#define NUM_VERTEX ...
  /* nr de vertex-uri */
#define NUM_TRIUNG ...
  /* nr de triunghiuri */

typedef struct { double x, y,
z; } PUNCT;
```

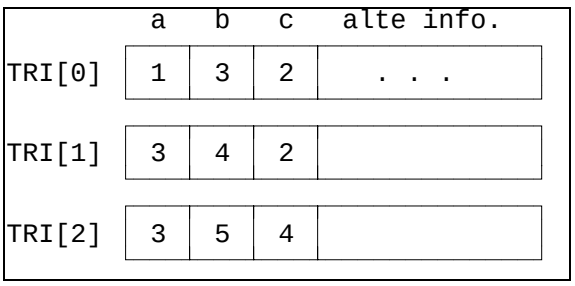

Figura 1.4

```
typedef struct { int a, b, c;
        /* alte informaţii */
} TRIUNGHI;

PUNCT vertex[NUM_VERTEX];
TRIUNGHI triunghiuri[NUM_TRIUNG];
```

În Figura 1.4 este prezentat un tablou de triunghiuri reprezentând reţeaua din Figura 1.2. O variantă ar fi să memorăm datele triunghiurilor nu ca tablou ci ca listă înlănţuită. În acest fel manipularea triunghiurilor devine mult mai uşoară. Un astfel de exemplu va fi prezentat în algoritmul Warnock.

2.2 Reţele de Laturi Explicite

Reţelele de laturi explicite sunt o reprezentare alternativă care elimină practic toate dezavantajele reţelelor de poligoane explicite. Ele sunt formate din trei structuri şi utilizează în mod intensiv pointerii.

În primul rând, avem o tabelă de vertex-uri, la fel ca şi la reţelele de poligoane explicite. Fiecare vertex este memorat o singură dată în această tabelă, ca triplet de numere. A doua structură este o listă înlănţuită a tuturor laturilor din reţea. O latură este determinată de două vertex-uri, deci fiecare înregistrare din listă conţine o pereche de pointeri în tabela vertex-urilor. În plus, înregistrarea mai conţine pointeri spre o listă de poligoane (descrisă mai jos) şi un contor. Pentru fiecare poligon care conţine latura avem câte un pointer; în cazurile normale, cu obiecte solide fără goluri, mărginite de faţete poligonale, vom avea câte doi asemenea pointeri la poligoane. Exemplul de mai jos nu este un asemenea obiect solid închis. Contorul precizează numărul de poligoane care au comună acea latură. (Există aplicaţii speciale în care o latură aparţine la mai mult de două poligoane.)

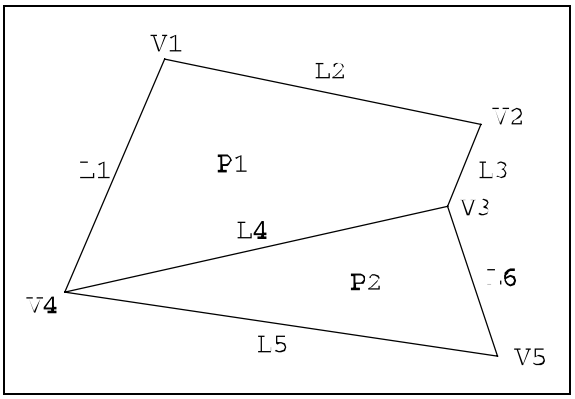

Figura 1.5

A treia structură este o listă de poligoane. Aceasta este o listă înlănţuită de pointeri în lista de laturi. Aceşti pointeri indică, în ordinea corectă, toate laturile din care este compus poligonul.

Figura 1.5 prezintă un exemplu de reţea de laturi explicite. Tabela vertex-urilor pentru poligoanele din Figura 1.5 este:

$$V1 = (x_1, y_1, z_1)$$
$$V2 = (x_2, y_2, z_2)$$
$$V3 = (x_3, y_3, z_3)$$
$$V4 = (x_4, y_4, z_4)$$
$$V5 = (x_5, y_5, z_5)$$

Lista de laturi este:

```
LLIST = &L1
```

cu

```
L1 = (1,&V1,&V4,&P1,NULL,&L2)
L2 = (1,&V1,&V2,&P1,NULL,&L3)
L3 = (1,&V2,&V3,&P1,NULL,&L4)
L4 = (2,&V3,&V4,&P1,&P2,&L5)
L5 = (1,&V4,&V5,&P2,NULL,&L6)
L6 = (1,&V3,&V5,&P2,NULL,NULL)
```

LLIST este un pointer la prima latură. Fiecare latură constă dintr-un contor întreg (numărul de poligoane care conţin acea latură) şi cinci pointeri: doi pointeri la capetele laturii, doi pointeri la poligoane şi un pointer la latura următoare. Unul din pointerii la poligoane este NULL dacă latura aparţine unui singur poligon. Ultima latură se termină cu NULL. Înlănţuirea face posibilă parcurgerea tuturor laturilor, de la prima până la ultima. Lista de poligoane este:

```
PLIST = (&P1,&P2,NULL)
```

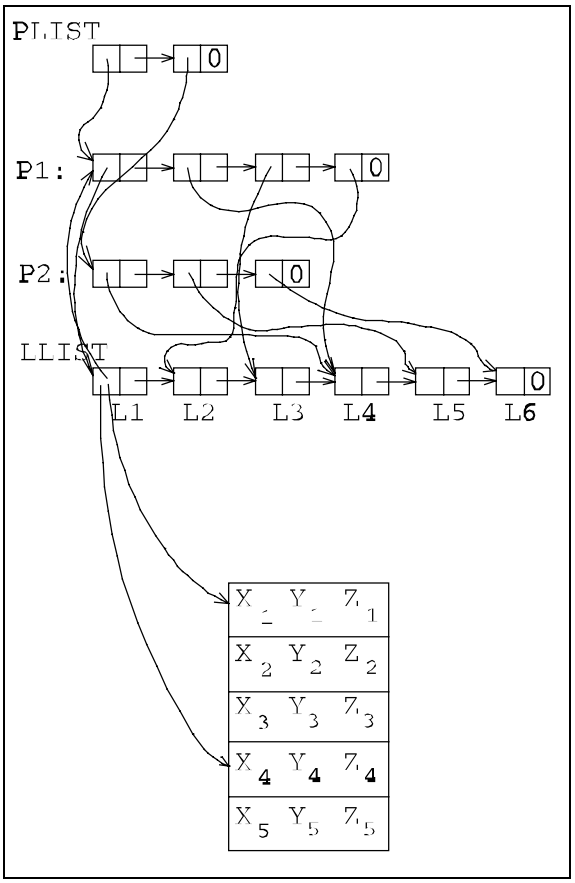

Figura 1.6 Structura listelor înlănţuite

cu

```
P1 = (&L1,&L4,&L3,&L2,NULL)
P2 = (&L4,&L5,&L6,NULL)
```

Aici, PLIST este o listă de pointeri la poligoane. Întrucât numărul de poligoane este variabil, lista se termină cu un pointer NULL. Fiecare poligon este la rândul său o listă înlănţuită de pointeri către laturi; lista se termină cu NULL pentru că şi numărul de laturi poate fi variabil. În descrierea poligoanelor, laturile se precizează în sens antiorar (ccw - "counterclock-wise"). Această ordine este esenţială pentru multe aplicaţii, cum ar fi determinarea orientării în spaţiu a poligonului, pentru îndepărtarea faţetelor ascunse. În definirea laturilor, ordinea celor două capete nu are importanţă. Figura 1.6 prezintă structura pentru exemplul din Figura 1.5.

Pentru afişarea reţelei se parcurge lista de laturi; fiecare latură este trasată o singură dată. Găsirea tuturor laturilor care compun un poligon nu este o problemă. La fel, modificarea unui vertex nu are efecte colaterale. Adăugarea sau ştergerea unui vertex este însă mai complicată, întrucât trebuie create sau şterse referinţe la acel vertex.

Totuşi, pe baza acestei reprezentări este dificil să găsim laturile care conţin un anumit vertex; pentru aceasta trebuie parcurse toate laturile. Pentru rezolvarea unor asemenea probleme s-ar mai putea include nişte informaţii în descrierea reţelei, dar cu cât se includ mai multe legături şi relaţii cu atât se complică generarea şi modificarea.

Deoarece această reţea conţine informaţie redundantă, ea poate deveni inconsistentă. O valoare incorectă poate să nu conducă la o reţea diferită, dar în general creează contradicţii. De exemplu, într-o descriere a unei laturi ar putea figura faptul că latura aparţine unui

poligon, când în realitate nu este aşa. Sunt posibile şi alte inconsistenţe. Din această cauză, sunt necesari algoritmi care să verifice consistenţa reţelei.

Din păcate, corectarea inconsistenţelor nu se poate realiza prin progam. De exemplu, contorul din structura înregistrării unei laturi are doar rolul de a verifica un aspect al consistenţei. Un program ar putea anula toate contoarele şi apoi să parcurgă toate poligoanele şi să incrementeze contorul unei laturi când se găseşte o referinţă la ea. Apoi trebuie parcurse toate laturile pentru a verifica dacă contorul corespunde cu numărul de pointeri la poligoane. Dacă diferă, trebuie afişată această informaţie pentru laturile inconsistente.

3 Îndepărtarea Faţetelor Ascunse

Dacă un obiect spaţial este format din poligoane, este destul de uşor să identificăm suprafeţele obiectului care nu sunt vizibile din punctul de observare. Aceste suprafeţe nu vor fi afişate. Această tehnică poartă numele de *îndepărtarea faţetelor ascunse*.

Pentru început, este necesară o trecere în revistă a calculului vectorial. Un vector este caracterizat printr-o direcţie în spaţiu şi prin lungime; punctul de start al vectorului nu are importanţă, deci doi vectori paraleli cu aceeaşi lungime sunt identici. Aceasta înseamnă că putem deplasa un vector în orice poziţie, singura condiţie este să rămână paralel cu el însuşi. Atunci când un vector porneşte din origine, capătul său este punctul (x,y,z). Din acest punct de vedere un punct sau un vector pornind din origine sunt acelaşi lucru. Pentru formalismul matematic nu este necesar să facem distincţie între ele, dar diferenţa este importantă. În continuare vom utiliza triplete de numere pentru a defini şi punctele şi vectorii şi vom numi tripletul punct sau vector, după nevoie.

Dacă doi vectori (A,B,C) şi (x,y,z) sunt normali unul pe celălalt în spaţiu, atunci produsul lor scalar este 0:

$$(A,B,C)\cdot\begin{bmatrix}x\\y\\z\end{bmatrix}=0 \qquad (6)$$

Toţi vectorii (punctele) (x,y,z) pentru care Ax+By+Cz = 0 se află într-un plan care conţine originea; vectorul (A,B,C) este normal la acel plan. De asemenea, vom spune că planul este normal la acel vector. Planul are două feţe. Vom folosi termenul de *faţa pozitivă* pentru a desemna faţa de pe partea normalei.

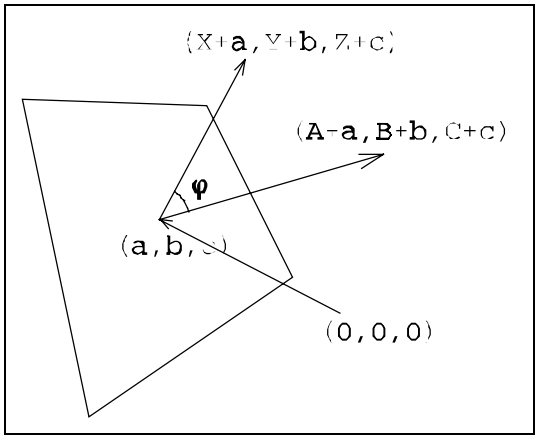

Figura 1.7 Unghiul φ este mai mic decât π/2

Să mergem mai departe. Dacă (a,b,c) este un alt vector, atunci toate punctele (x-a,y-b,z-c) pentru care A(x-a)+B(y-b)+C(z-c) = 0 se situează într-un alt plan normal la (A,B,C). Dacă (x-a,y-b,z-c) se află în acest plan, atunci (x,y,z) se află într-un plan paralel cu acesta, translatat cu vectorul (a,b,c). Deci A(x-a)+B(y-b)+C(z-c) = 0 este reprezentarea unui plan care trece prin punctul (a,b,c) şi este normal la vectorul (A,B,C). Mai pe scurt, Ax+By+Cz-D = 0, cu D = Aa+Bb+Cc.

Vectorul (A,B,C) indică o anumită direcţie. Putem plasa punctul de pornire al vectorului în acest plan prin translatare în punctul (a,b,c), care aparţine planului. Capătul

vectorului este acum punctul (A+a,B+b,C+c) (vezi Figura 1.7).

Dacă un punct (x,y,z) se află de aceeaşi parte a planului (faţa pozitivă) ca şi (A+a,B+b,C+c) atunci poate fi exprimat ca (x+a,y+b,z+c), unde (x,y,z) trebuie să fie astfel încât unghiul φ dintre vectorii (x,y,z) şi (A,B,C) trebuie că fie mai mic decât π/2. Rezultă că cos(φ)>0. Rezultă

$$(A,B,C)\cdot\begin{bmatrix}x\\y\\z\end{bmatrix} = |(A,B,C)|\cdot|(x,y,z)|\cdot\cos\varphi > 0. \qquad (7)$$

Inserând (x+a,y+b,z+c) în ecuaţia planului, obţinem relaţia (3). Aceasta ne arată că ecuaţia planului dă un rezultat pozitiv pentru orice punct situat pe aceeaşi parte faţă de plan ca şi direcţia indicată de (A,B,C). Pentru punctele de pe faţa cealaltă rezultatul va fi negativ.

$$A(x+a)+B(y+b)+C(z+c)-D = (A,B,C)\cdot\begin{bmatrix}x\\y\\z\end{bmatrix}+D-D>0 \qquad (8)$$

De exemplu, să presupunem că (A,B,C)=(5,-3,1). Fie (a,b,c)=(0,2,4). Atunci planul care trece prin (a,b,c) normal la (A,B,C) este Ax+By+Cz=D, cu

$$D = (5)(0)+(-3)(2)+(1)(4) = -2$$

Ecuaţia planului este 5x-3y+z+2=0. Punctul (3,5,-6) este pe partea pozitivă?

$$(5)(3)+(-3)(5)+(1)(-6)+2 = -4, \quad nu$$

Dar (0,0,0)?

$$(5)(0)+(-3)(0)+(1)(0)+2 = 2, \quad da.$$

Ce se poate spune despre (A,B,C)+(a,b,c) = (5,-1,4)?

$$(5)(5)+(-3)(-1)+(1)(4)+2 = 34, \quad desigur.$$

Deseori, dorim să determinăm ecuaţia unui plan determinat de trei puncte. Dacă P_1, P_2, şi P_3 cu coordonatele (x_1,y_1,z_1), (x_2,y_2,z_2) şi (x_3,y_3,z_3) sunt cele trei puncte necoliniare, atunci coeficienţii ecuaţiei planului, Ax+By+Cz-D=0, sunt soluţia sistemului liniar:

$$Ax_i+By_i+Cz_i-D = 0, \quad \text{pentru } i = 1,2,3$$

cu necunoscutele A, B, C şi D. Acest sistem liniar are întotdeauna soluţie. O soluţie este

$$A=\begin{vmatrix}1 & y_1 & z_1\\1 & y_2 & z_2\\1 & y_3 & z_3\end{vmatrix} \quad B=\begin{vmatrix}x_1 & 1 & z_1\\x_2 & 1 & z_2\\x_3 & 1 & z_3\end{vmatrix}$$

$$C=\begin{vmatrix}x_1 & y_1 & 1\\x_2 & y_2 & 1\\x_3 & y_3 & 1\end{vmatrix} \quad D=\begin{vmatrix}x_1 & y_1 & z_1\\x_2 & y_2 & z_2\\x_3 & y_3 & z_3\end{vmatrix}$$

Putem preciza cele trei puncte ca $P_1P_2P_3$, $P_2P_3P_1$, sau $P_3P_1P_2$ (se păstrează ordinea ciclică). Totuşi, dacă se modifică ordinea ciclică în care sunt specificate punctele, atunci direcţia normalei se inversează.

Să presupunem că precizăm cele trei puncte în sens antiorar privite dintr-un punct Z

din spaţiu şi că ecuaţia planului care trece prin cele trei puncte este Ax+By+Cz-D=0. O proprietate foarte utilă a ecuaţiei planului este că va da rezultat pozitiv pentru punctul Z (după cum s-a văzut în exemplul de mai sus). Rezultă că orice punct care se află de aceeaşi parte faţă de plan ca şi punctul Z va da un rezultat pozitiv.

Aceasta ne permite să determinăm dacă o suprafaţă este vizibilă din orice punct dat în spaţiu. Dacă suprafaţa este determinată de un poligon, acest poligon se află într-un plan (lucrăm cu poligoane plane). Determinăm ecuaţia planului specificând trei puncte din plan, de obicei trei vertex-uri ale poligonului, şi convenim că punctele se precizează în sens antiorar, văzute dintr-un punct Z din spaţiu, din care considerăm suprafaţa ca fiind vizibilă. Atunci orice punct din spaţiu pentru care ecuaţia planului dă rezultat pozitiv se află de aceeaşi parte a planului ca şi punctul Z, deci suprafaţa este vizibilă dintr-un asemenea punct. Dacă un punct dă un rezultat negativ, atunci el se află de partea cealaltă a planului şi considerăm că planul nu este vizibil din acel punct.

Vom defini în continuare noţiunea de vizibilitate pentru un obiect solid. Presupunem că un obiect solid este mărginit de poligoane. Fiecare suprafaţă a obiectului are un interior şi un exterior. Desigur, interiorul nu este vizibil, întrucât este mascat de masa obiectului. (Şi suprafeţele exterioare pot fi mascate, dar aceasta este altă problemă, pe care o vom trata mai târziu.) Dacă determinăm ecuaţiile planelor tuturor suprafeţelor astfel încât punctele exterioare să dea un rezultat pozitiv iar punctele interioare -- rezultat negativ, atunci putem determina orientarea oricărui punct din spaţiu în raport cu suprafeţele obiectului.

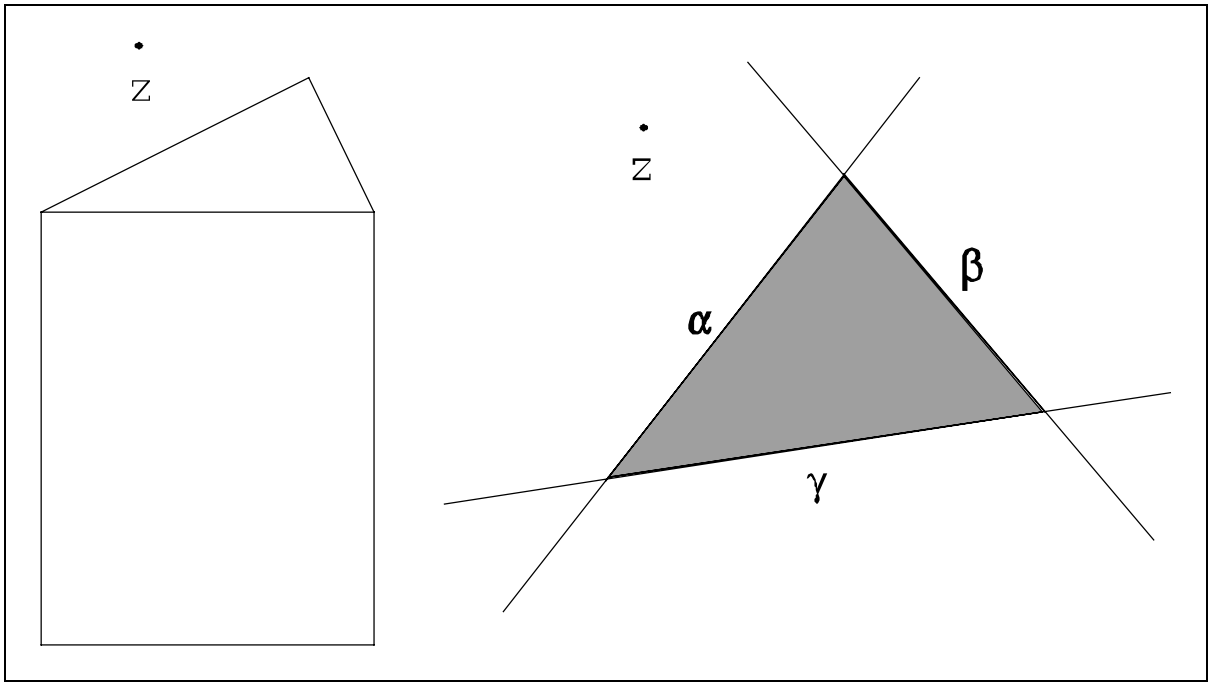

Figura 1.8 Prismă văzută din lateral şi de sus

Figura 1.8 prezintă o prismă solidă şi un punct Z, în vedere laterală şi de sus. Vederea de sus este mai sugestivă pentru ceea ce ne propunem. Cele trei suprafeţe laterale sunt indicate prin α, β, γ, iar planele în care ele sunt situate sunt reprezentate prin drepte. Punctul Z este "înafară" în relaţie cu planul α, "înăuntru" în relaţie cu β şi "înăuntru" faţă de γ ("înăuntru" nu înseamnă neapărat în interiorul obiectului). Orice punct din care suprafaţa β

este vizibilă trebuie să se afle de partea opusă punctului Z faţă de planul β. Acelaşi lucru pentru γ. Se observă că înăuntru sau înafară sunt singurele informaţii pe care le putem furniza sistemului, întrucât noi putem doar să vedem cum arată obiectul.

Furnizăm aceste informaţii atunci când specificăm cele trei puncte care definesc planul care conţine suprafaţa respectivă. De exemplu, suprafaţa γ este cea pe care o vedem din lateral. Când specificăm planul lui γ, trebuie să specificăm cele trei vertex-uri alese în sens antiorar. La fel, cele trei puncte alese pentru suprafaţa α trebuie să fie în sens antiorar când le privim dintr-un punct din apropierea lui Z.

Odată determinate ecuaţiile planelor, putem afla dacă o suprafaţă este vizibilă dintr-un punct dat. Dacă Z era poziţia observatorului, putem insera punctul Z în ecuaţiile planelor α, β şi γ, obţinem rezultatele >0, <0 respectiv <0 şi rezultă că afişăm numai suprafaţa α. Faţa de sus şi cea de jos au fost ignorate în acest exemplu, dar ele trebuie tratate exact în acelaşi fel.

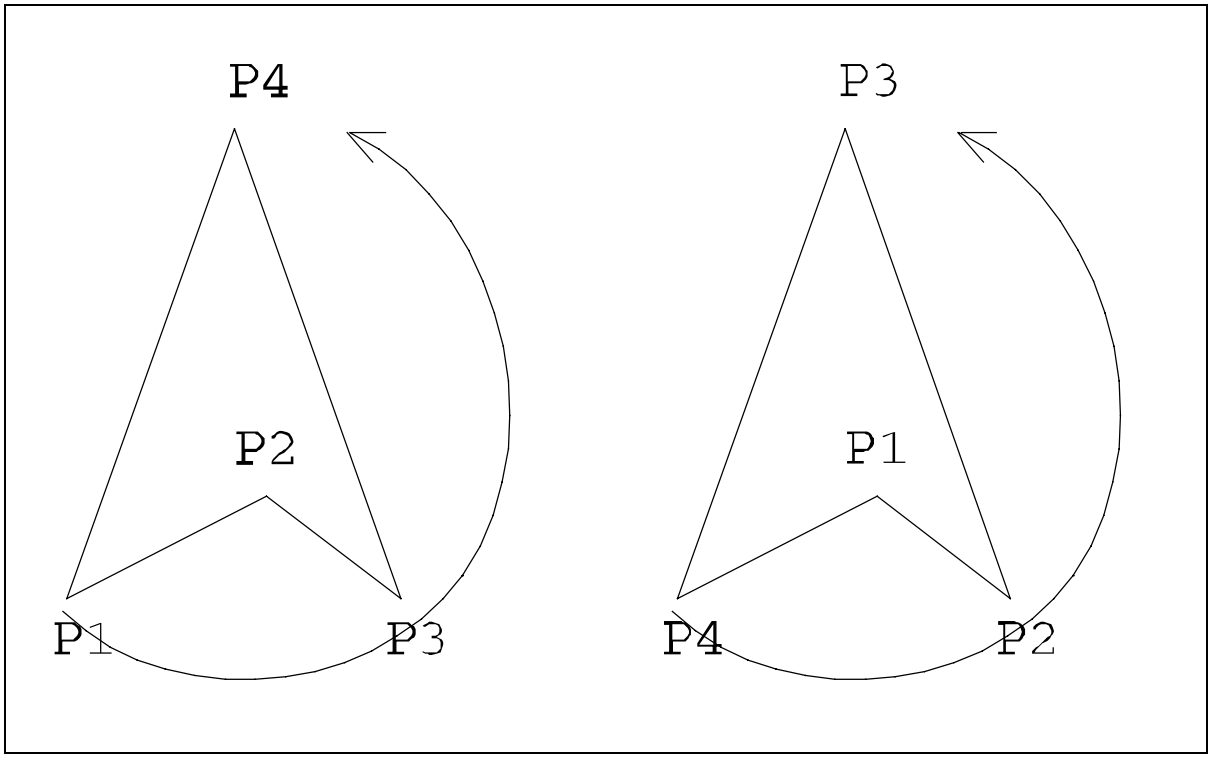

Figura 1.9

Îndepărtarea faţetelor ascunse (IFA) poate fi efectuată dacă toate poligoanele care descriu un obiect au fost introduse astfel încât orientarea unui plan poate fi determinată într-un mod neambiguu din informaţiile referitoare la poligon. Uneori definim poligoanele prin vertex-uri, alteori prin laturi. În ambele cazuri, ele trebuie precizate în sens antiorar când sunt privite din exteriorul obiectului. În unele structuri de date coeficienţii reprezentărilor planelor se calculează o singură dată şi se memorează. În alte cazuri, se calculează la cerere. Calculul se poate baza pe primele trei vertex-uri din structura de date care defineşte poligonul. (Dacă poligonul este definit prin laturi, atunci primele două laturi vor da cele trei vertex-uri). Atunci când un obiect este supus unei transformări în spaţiu, coeficienţii planelor trebuie recalculaţi din vertex-urile transformate sau se transformă şi coeficienţii. A doua metodă este de obicei

mai simplă.

Parcurgerea în sens antiorar a unui poligon furnizează întotdeauna o ordine antiorară a vârfurilor întâlnite. Trebuie acordată mai mare atenţie cazului când intervin poligoane concave. În Figura 1.9 se vede că punctele P_1, P_2, P_3 nu sunt în sens antiorar în poligonul din stânga, chiar dacă acesta a fost parcurs în sens antiorar. Întotdeauna este însă posibil să renumerotăm vertex-urile astfel încât primele trei să fie în sens antiorar.

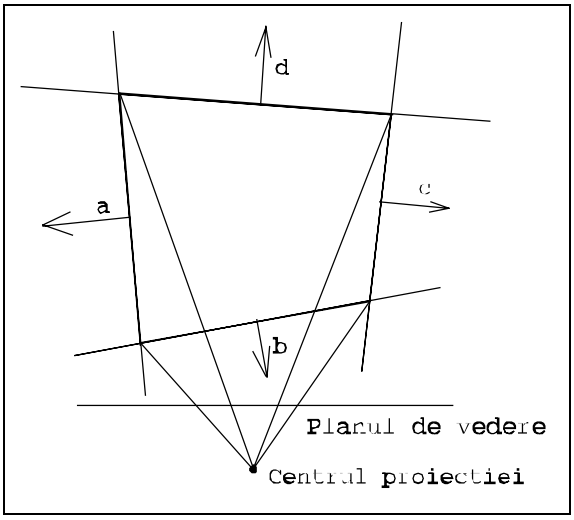

Figura 1.10 Proiecţie perspectivă şi ortografică

IFA depinde de tipul proiecţiei -- perspectivă sau ortografică. La proiecţia perspectivă, punctul pentru care se verifică ecuaţiile planelor este centrul proiecţiei. Dacă rezultatul este pozitiv, suprafaţa este vizibilă. Prin contrast, în cazul proiecţiei ortografice testul este mai simplu. Vectorul normal unui plan este (A,B,C). Dacă el indică spre observator, atunci faţeta este vizibilă. Rezultă că o coordonată z pozitivă indică o faţetă vizibilă. Nu trebuie efectuate calcule; verificăm doar semnul lui C. Este mai indicat să considerăm o faţetă ca fiind ascunsă chiar dacă rezultatul nu este 0, dar este foarte mic, deoarece trebuie luate în considerare şi erorile de rotunjire. Se face deci verificarea dacă C > 0.0001.

Figura 1.10 prezintă un obiect pentru care se face IFA. Rezultatele diferă după tipul proiecţiei. În cazul proiecţiei ortografice, suprafeţele a, b şi c sunt vizibile, deoarece normalele la ele au o componentă înspre planul de vedere. Dacă este vorba de o proiecţie perspectivă, atunci doar suprafaţa b este vizibilă.

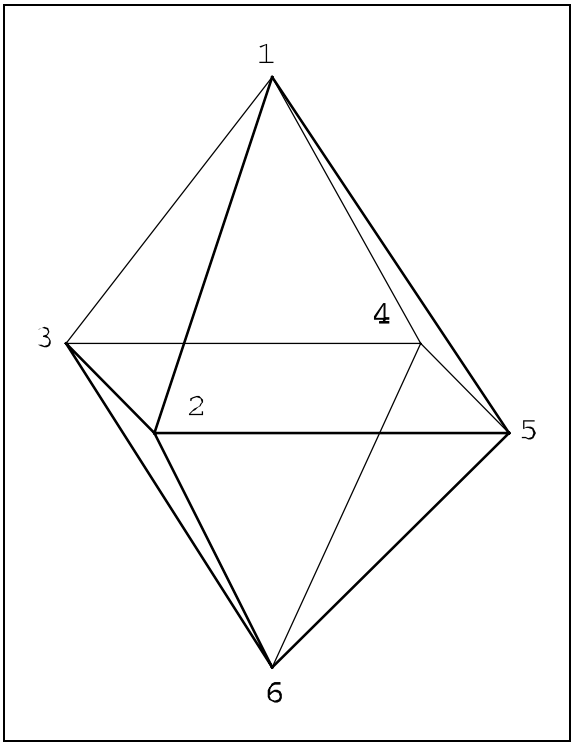

Figura 1.11 Octaedru reprezentat prin contur

Afişarea unui Singur Obiect Convex

Metoda prezentată poate fi utilizată pentru îndepărtarea suprafeţelor sau liniilor ascunse când se afişează un singur obiect convex, indiferent dacă obiectul se afişează prin suprafeţe pline sau doar prin laturi (model wireframe cu liniile ascunse îndepărtate). În cazul unui asemenea obiect, nici o faţetă dinspre observator nu poate fi ascunsă; ele sunt toate vizibile. Deci IFA este singura operaţie care trebuie efectuată. Dacă trebuie îndepărtate liniile ascunse, tehnica este simplă. Obiectul este definit ca o colecţie de poligoane şi se efectuează IFA. Apoi se face trasarea ca şi

cum s-ar desena suprafeţe poligonale, cu diferenţa că se trasează doar laturile lor.

În Figura 1.11 este prezentat un octaedru, cu vertex-urile indicate prin numere. Laturile trasate de două ori sunt prezentate îngroşat, cele trasate o singură dată - ca linii normale, iar laturile care nu se afişează sunt punctate. Din punctul de observare sunt vizibile patru faţete, iar patru faţete nu sunt vizibile. Aceasta face ca laturile 21, 23, 25 şi 26 să fie trasate de două ori.

Conturul obiectului se trasează o singură dată, deoarece este format din laturi care aparţin unui singur poligon vizibil. Dacă planul unui poligon este pe direcţia de observare (în cazul proiecţiei ortogonale, coeficientul C este 0; la proiecţia perspectivă, ecuaţia planului dă rezultatul 0 pentru centrul de proiecţie), el este o singură dreaptă, imaterială indiferent dacă această linie este trasată sau nu. Întotdeauna există o altă latură a unui poligon vizibil care coincide cu acea linie.

Obiecte Concave; Mai Multe Obiecte Dacă un obiect este concav, unele faţete dinspre observator pot fi parţial sau total acoperite de alte părţi ale obiectului. Similar, dacă se afişează mai multe obiecte, chiar dacă ele sunt convexe, un obiect poate să acopere părţi din alt obiect. IFA nu rezolvă problema, întrucât se afişează toate faţetele dinspre observator, fără a ţine seama de acoperiri. În Figura 1.12 avem un obiect concav în care faţeta a, care ar fi afişată, este acoperită parţial de alte faţete.

Chiar dacă nu rezolvă acest tip de probleme, IFA este o metodă utilă de aplicat înaintea altor tehnici generale de îndepărtare a suprafeţelor şi liniilor ascunse, întrucât reduce cu circa 50% numărul de poligoane pe care trebuie să le trateze algoritmii mai generali şi mai complecşi pe care îi vom studia în continuare.

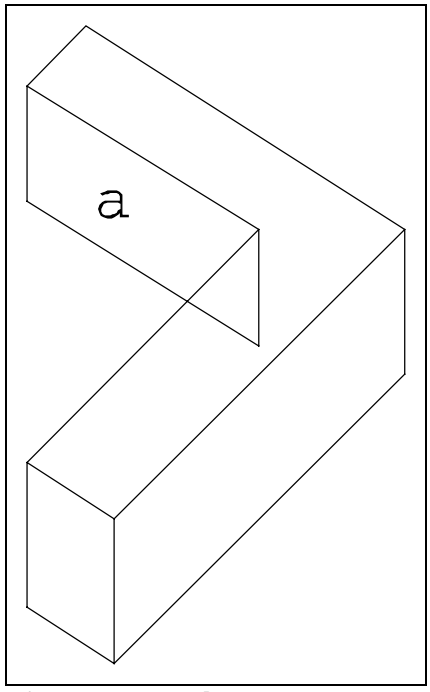

Figura 1.12 Obiect concav

4 Metode de Sortare în Adâncime

Metodele descrise în această secţiune sunt mai puternice decât cele prezentate anterior, deoarece nu se restrâng la un singur obiect convex. Şi în continuare vom lucra cu scene grafice ale căror obiecte sunt definite prin suprafeţe poligonale plane, dar pot exista mai multe obiecte, care pot să fie nu numai convexe ci şi concave. Aceasta este situaţia în majoritatea cazurilor. Se impun însă două restricţii. Scenele nu pot să conţină suprafeţe care se întrepătrund reciproc sau care se acoperă ciclic. Evident, poligoanele care compun scena sunt memorate într-o structură de date. Fără a restrânge din generalitate, facem câteva presupuneri:

a. Poligoanele sunt memorate în una sau mai multe reţele;
b. Se efectuează o transformare a planului de vedere;
c. Se efectuează o transformare perspectivă.

Pentru a desena scena grafică cu liniile sau suprafeţele ascunse îndepărtate, mai întâi se efectuează două operaţii:

1) se transformă toate vertex-urile în planul de vedere dorit;
2) se aplică o transformare perspectivă asupra tuturor vertex-urilor.

Aceste operaţii aduc obiectele de afişat în poziţia dorită.

Cu poligoanele în poziţia dorită, efectuăm un test de faţetă ascunsă asupra fiecărui poligon în parte. Dacă este vorba de o faţetă ascunsă, nu o vom mai lua în considerare. Dacă acel poligon nu este o faţetă ascunsă, îi memorăm toate vertex-urile într-un tablou temporar de dimensiune redusă (un tablou este mai uşor de prelucrat decât o listă înlănţuită).

În continuare, descompunem poligonul în triunghiuri; când extragem un triunghi din poligon, îi adăugăm vertex-urile în tabloul triunghiurilor. Acest tablou va conţine probabil faţetele frontale ale întregii scene, sub formă de triunghiuri. În Figura 1.13 este prezentat un patrulater, reprezentarea lui în tabloul temporar şi cele două triunghiuri care rezultă în urma descompunerii. Săgeţile indică începutul fiecărui triunghi. Valorile *mo* şi *dr* vor fi explicitate mai târziu.

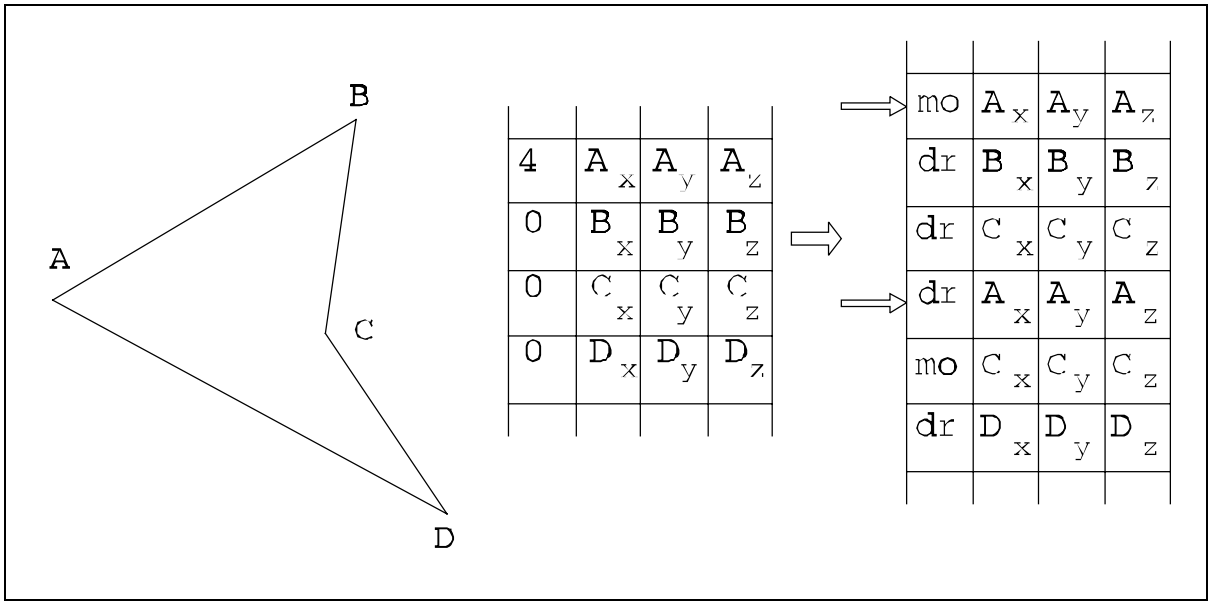

Figura 1.13 Patrulater, structura sa de date şi descompunerea în triunghiuri.

Pasul următor este sortarea acestor triunghiuri, într-o ordine bazată pe adâncimea faţă de planul de vedere. După sortarea în adâncime, putem continua în două moduri. În primul rând, dacă desenăm scena ca fiind compusă din obiecte solide, trebuie să îndepărtăm părţile acoperite ale triunghiurilor. În acest caz, putem desena triunghiurile începând cu cel mai îndepărtat; atunci când trasăm un alt triunghi peste unul desenat anterior, acesta este ascuns în mod automat. Această metodă poartă numele de *algoritmul pictorului*. Pe de altă parte, dacă scena este compusă doar din linii, trebuie să îndepărtăm liniile ascunse. Această operaţie este mult mai complexă, întrucât o linie poate fi complet ascunsă de un triunghi, sau ar putea fi decupată în una sau în două bucăţi.

4.1 Descompunerea în Triunghiuri

Determinarea celui mai îndepărtat dintre două poligoane este mai uşoară dacă cele două poligoane sunt triunghiuri, deoarece triunghiul este cel mai simplu poligon şi întotdeauna este convex. Scopul nostru este deci să descompunem toate poligoanele cu patru sau mai multe laturi în triunghiuri. Această descompunere este simplă dacă poligonul este convex, dar este mult mai complicată în cazul poligoanelor concave. Nu luăm în considerare poligoanele stelate, pentru că ele nu au sens ca suprafeţe de obiecte solide.

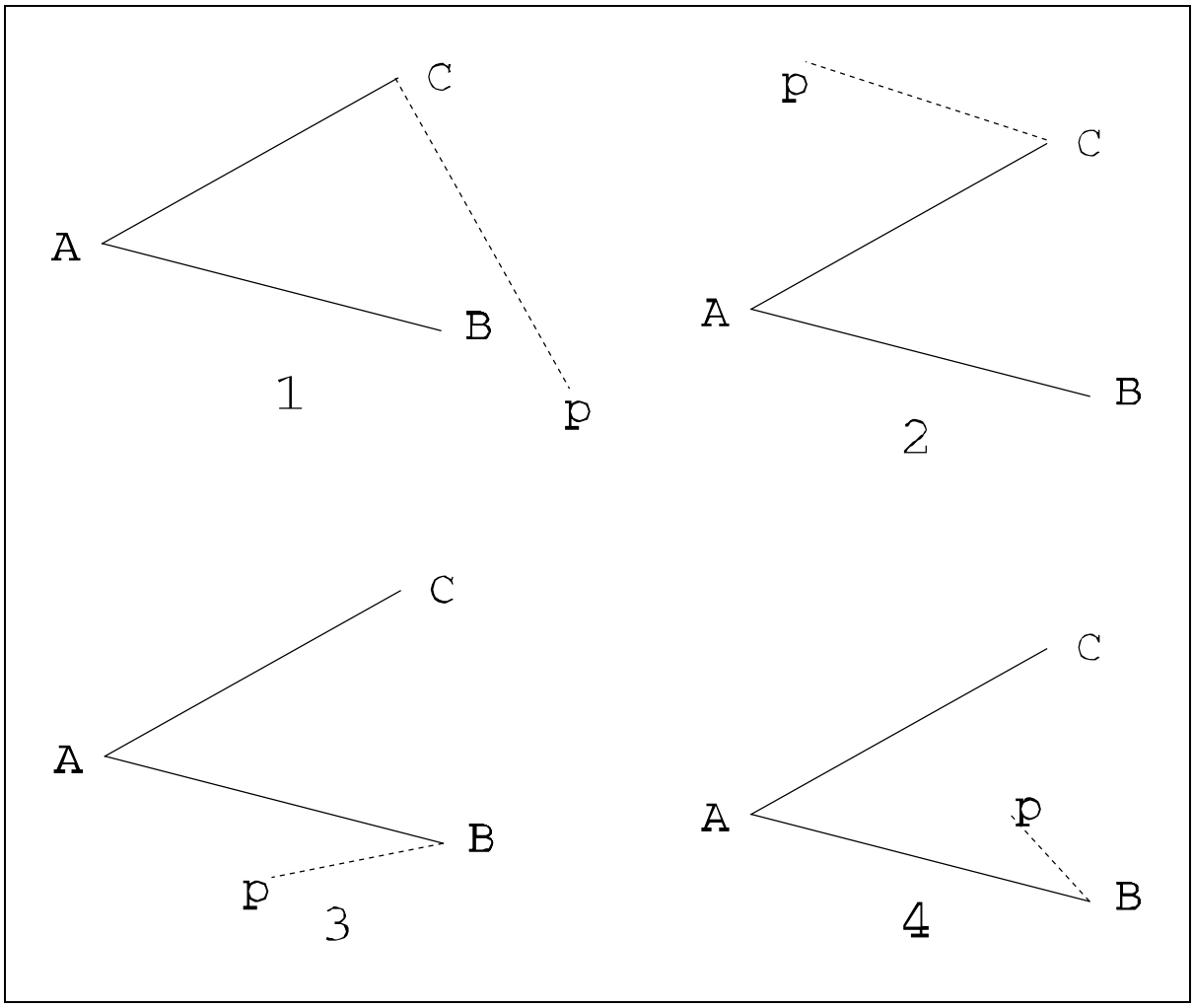

Figura 1.14 Câteva situaţii ale triunghiului cel mai din stânga

Metoda pe care o vom folosi descompune poligoane convexe sau concave. Se porneşte cu vertex-ul cel mai din stânga al poligonului (A, vezi Figura 1.14). El poate fi determinat căutând vertex-ul cu coordonata x cea mai mică. Luăm vertex-urile vecine lui, B şi C. A, B şi C formează triunghiul cel mai din stânga. Un al patrulea vertex, care nu aparţine acestui triunghi, p, este prezentat legat printr-o linie punctată. Această linie nu este neapărat o latură, întrucât p poate să nu fie adiacent lui B sau C. Motivul pentru care studiem punctul p este ca să vedem dacă poligonul este sau nu concav; dacă da, p este situat în interiorul triunghiului ABC.

Dacă nici un alt vertex p al poligonului nu se află în interiorul triunghiului, putem extrage acest triunghi din poligon. Figura 1.14 prezintă câteva exemple de situaţii în care se poate afla p faţă de triunghi. Vom utiliza testul *minimax* pentru a elimina unele din cazurile simple (1 şi 2 din Figura 1.14). Un test *minimax* constă din găsirea celui mai mic dreptunghi care conţine triunghiul şi apoi verificăm dacă p este în interiorul sau în exteriorul acestui dreptunghi. Fie (x_1,y_1), (x_2,y_2) şi (x_3,y_3) cele trei vertex-uri ale triunghiului. Cel mai mic dreptunghi este determinat de colţul stânga jos:

$$\min(x_1,x_2,x_3)\ ,\ \min(y_1,y_2,y_3)$$

şi colţul dreapta sus:

$$\max(x_1,x_2,x_3)\ ,\ \max(y_1,y_2,y_3)$$

Pentru a verifica dacă p se află sau nu în acest dreptunghi, sunt necesare patru teste (vezi Figura 1.15):

$$p_x < \min(x_1,x_2,x_3)$$
$$p_y < \min(y_1,y_2,y_3)$$
$$p_x > \max(x_1,x_2,x_3)$$
$$p_y > \max(y_1,y_2,y_3)$$

De fapt, aceste teste se pot simplifica, deoarece ştim deja că triunghiul ABC este cel mai din stânga. Deci avem nevoie doar de trei teste:

$$p_x > \max(B_x,C_x)$$
$$p_y > \max(A_y,B_y,C_y)$$
$$p_y < \min(A_y,B_y,C_y)$$

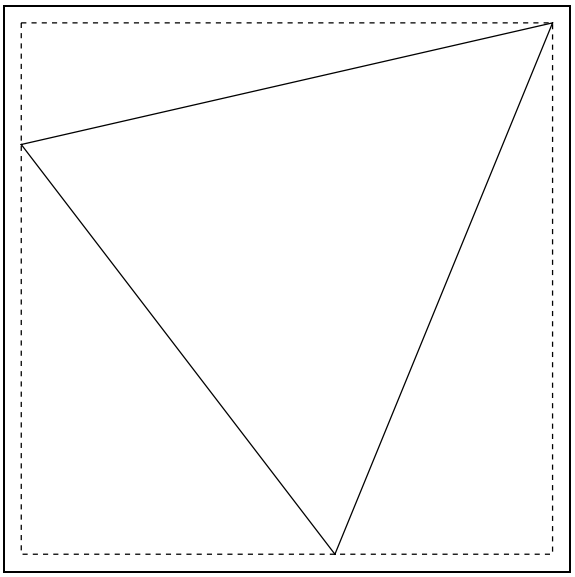

Figura 1.15

Dacă vreuna din aceste condiţii este îndeplinită, atunci în mod sigur p este înafara dreptunghiului (şi deci şi a triunghiului).

Testul minimax necesită mult mai puţine calcule decât testul general de interior de triunghi, descris mai jos. Îl vom utiliza mai întâi, în speranţa că vom economisi timp. Dacă el nu exclude toate punctele p, trebuie aplicate testele următoare.

Testul de Interior de Triunghi În cazurile 3 şi 4 din Figura 1.14, testul minimax nu poate să excludă punctul p. Expresia

$$f(x,y) = (x-x_1)\cdot(y_2-y_1)-(x_2-x_1)\cdot(y-y_1)$$

determină dreapta care trece prin punctele (x_1,y_1) şi (x_2,y_2). Punctul (x,y) se poate afla în trei poziţii:

`f(x,y)=0` $\Rightarrow$ `(x,y)` se află pe dreaptă
`f(x,y)<0` $\Rightarrow$ `(x,y)` se află de o parte a dreptei
`f(x,y)>0` $\Rightarrow$ `(x,y)` se află de cealaltă parte a dreptei

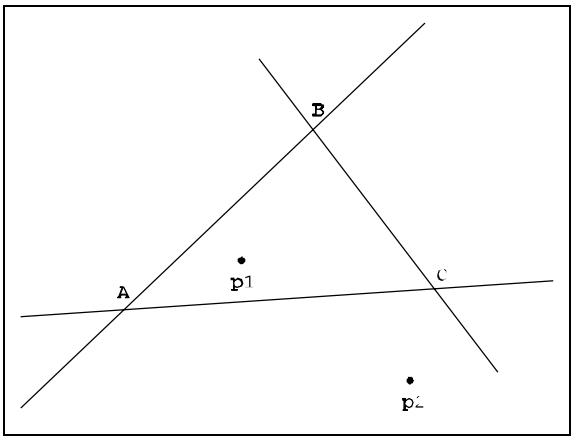

Figura 1.16 Puncte în interiorul şi în exteriorul unui triunghi

Două puncte oarecare se află de aceeaşi parte a dreptei dacă funcţia f are acelaşi semn pentru cele două puncte.

Acest lucru poate fi utilizat pentru testul de interior de triunghi. Un punct poate fi în interiorul triunghiului dacă se află de aceeaşi parte a unei laturi ca şi al treilea vertex. Este nevoie să efectuăm testul pentru toate cele trei laturi (vezi Figura 1.16). Punctul p_1 se află de aceeaşi parte cu A faţă de latura BC. Analog pentru celelalte două laturi. Punctul p_2 nu se află de aceeaşi parte cu C faţă de latura AB, deci este înafara triunghiului.

Funcţiile de mai jos implementează testul descris mai sus:

```
type point = record x, y : real
             end;

function SameSide(p1,p2,l1,l2: point) : boolean;
{ true dacă p1 şi p2 sunt de aceeaşi parte
  a dreptei l1l2 }
begin
  SameSide :=
    ((p1.x-l1.x)*(l2.y-l1.y)-(l2.x-l1.x)*(p1.y-l1.y)) *
    ((p2.x-l1.x)*(l2.y-l1.y)-(l2.x-l1.x)*(p2.y-l1.y)) > 0
end;

function inside(p, a, b, c : point) : boolean;
{ true dacă p este în interiorul triunghiului
  a,b,c; false dacă p este înafară sau pe
  laturi }
begin
  inside :=
     SameSide(p, a, b, c)
       and SameSide(p, b, a, c)
         and SameSide(p, c, a, b)
end;
```

Testul trebuie efectuat pentru triunghiul cel mai din stânga, pentru fiecare vertex al poligonului care nu aparţine acestui triunghi şi care nu a fost eliminat de testul minimax. Desigur, s-ar putea efectua doar testul general de interior, dar testul minimax economiseşte timp.

Reprezentăm poligonul original din tabloul temporar prin vertex-urile sale A, B, C, D ... (De reţinut că un vertex constă din coordonatele x, y, z) Precedăm fiecare vertex printr-o comandă de trasare - *dr* (draw). Ordinea de memorare implică ordinea (ciclică) de trasare, deci trasăm de la primul vertex la al doilea, de la al doilea la al treilea şi aşa mai departe,

până când trasăm de la ultimul la primul. În tabloul de triunghiuri, comanda corespunzătoare unui vertex înseamnă că pentru acel vertex se face o mutare - *mo* (move) sau o trasare - *dr*.

Diferenţierea acestor comenzi nu este foarte importantă dacă se afişează corpuri solide, dar este esenţială dacă trebuie executat un algoritm de îndepărtare a liniilor ascunse. De aceea, facem această diferenţiere în avans. Când un poligon este descompus, linia după care se face secţionarea va forma o nouă latură, având comanda de mutare înaintea fiecăruia din cele două capete.

Poligonul din Figura 1.17 are cinci vertex-uri. Numele lor nu implică ordinea lor -- aceasta este determinată de ordinea din tablou. Vertex-ul cel mai din stânga este A, iar vecinii săi sunt B şi C. Tabloul care descrie poligonul este

```
drC drD drE drB drA
```

Atenţie, linia de la A la C este implicită !

Nu există nici un vertex în interiorul triunghiului ABC, deci îl putem extrage din poligon. Procedura de extragere generează două noi poligoane. Se creează noile secvenţe, pornind de la tabloul temporar, prin înlăturarea unor vertex-uri şi modificarea comenzii `dr` într-o comandă `mo` la primul vertex (în ordine ciclică). Acest prim vertex este întotdeauna cunoscut, întrucât vertex-urile care se înlătură sunt întotdeauna în ordine ciclică.

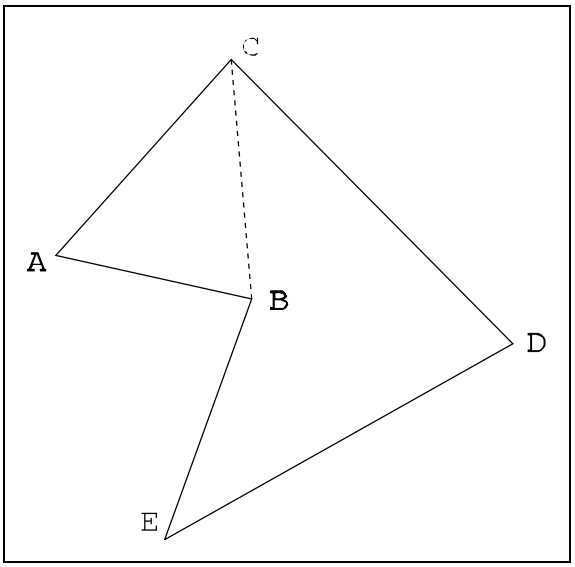

Figura 1.17 Extragerea unui triunghi; în interiorul ABC nu există nici un vertex

Primul poligon nou este triunghiul. Secvenţa care îl compune se obţine prin înlăturarea tuturor vertex-urilor din poligon, cu excepţia celui mai din stânga şi a celor doi vecini (se înlătură D şi E). Primul vârf rămas după îndepărtare este B, deci îl facem `mo`. Adăugăm secvenţa la tabloul de triunghiuri.

Al doilea poligon nou format se obţine prin îndepărtarea doar a celui mai din stânga vertex (A). Primul vertex din set după A, C, trebuie să fie `mo`. Dacă şi al doilea poligon este tot un triunghi, el este adăugat la tabloul de triunghiuri; altfel se păstrează în tabloul temporar şi procesul de descompunere continuă. Ne rezultă:

`drCdrDdrEdrBdrA` → `drCmoBdrA şi moCdrDdrEdrB`

Dacă există puncte în interiorul celui mai din stânga triunghi ABC (de exemplu p şi q în Figura 1.18), atunci trebuie să-l luăm pe cel mai din stânga (p) şi să descompunem poligonul, unindu-l pe A cu p. Rezultă astfel două *poligoane*. Dacă unul din ele este triunghi, îl adăugăm la tabloul de triunghiuri. În Figura 1.18 avem o asemenea situaţie.

Unul din poligoane se obţine prin înlăturarea tuturor vertex-urilor dintre p şi A în ordine ciclică (este vorba de B); A primeşte comanda de mutare. Celălalt poligon rezultă prin îndepărtarea tuturor vertex-urilor dintre A şi p (C şi q); p primeşte atributul `mo`. Avem:

`drCdrqdrpdrBdrA` → `drCdrqdrpmoA şi mopdrBdrA`

Dacă unul dintre cele două noi poligoane este triunghi (pBA), acesta se adaugă la tabloul triunghiurilor. Dacă ambele părţi au mai mult de trei vertex-uri, trebuie să le descompunem. Pentru aceasta, pe unul îl memorăm într-o stivă pentru a fi prelucrat mai târziu. Întreţinerea explicită a stivei se poate evita prin apelul recursiv al procedurii de descompunere.

Fiecare poligon va fi descompus în triunghiuri, astfel încât întreaga scenă grafică să fie compusă numai din triunghiuri. Acestea urmează să fie *sortate în adâncime*, proces numit şi *sortare geometrică*.

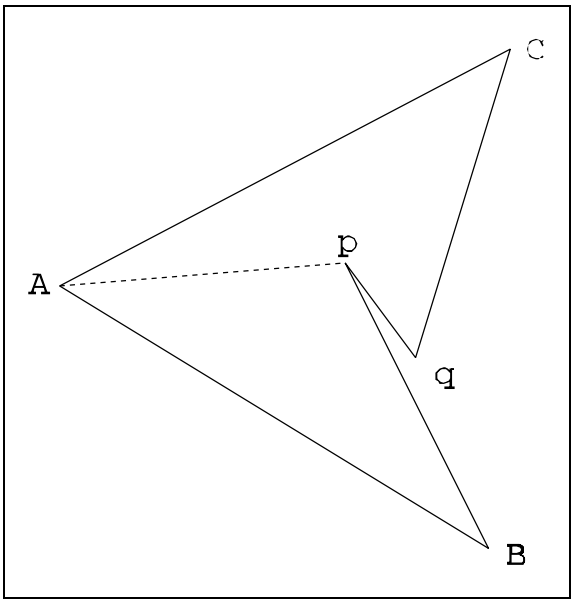

Figura 1.18 Două vertex-uri în interiorul lui ABC

4.2 Sortarea Geometrică

Putem afla care din triunghiurile care compun scena grafică pot să acopere alte triunghiuri dacă le sortăm într-o *ordine de acoperire*. Aceasta înseamnă să le aranjăm într-o astfel de secvenţă încât un triunghi poate să acopere (sau să ascundă) triunghiurile care urmează după el. Aceasta nu este o sarcină simplă. În situaţii de acoperire ciclică sau de întrepătrundere a triunghiurilor, ca în Figurile 1.19 şi 1.20, o asemenea ordine nici nu există. Pentru a trata asemenea situaţii ar fi necesară o descompunere suplimentară a triunghiurilor în altele mai mici.

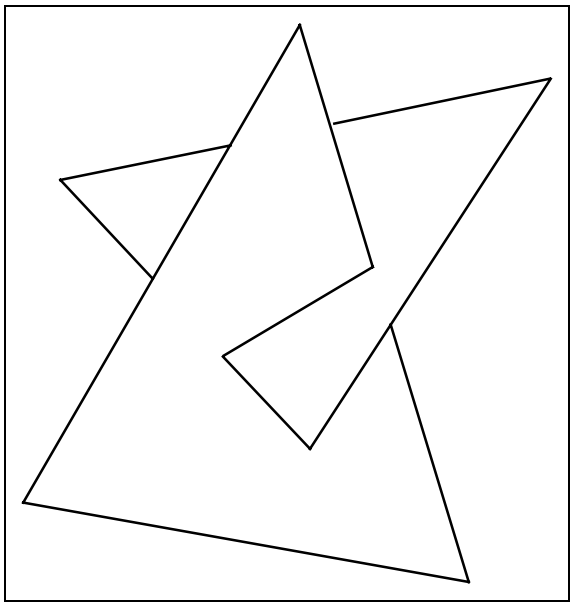

Figura 1.19 Două triunghiuri care se întrepătrund

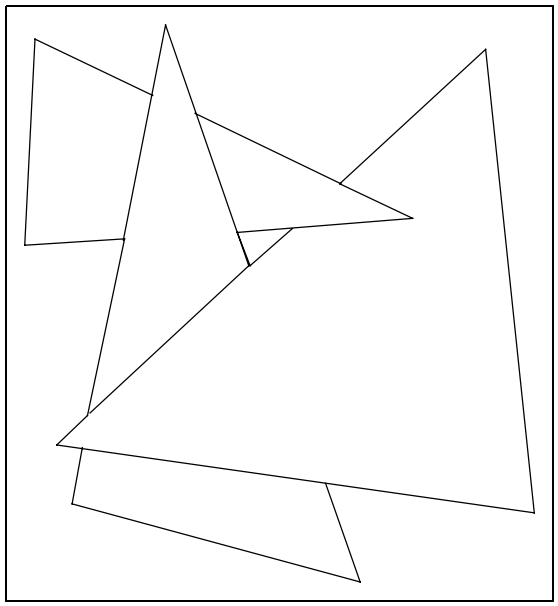

Figura 1.20 Trei triunghiuri care se acoperă ciclic

Dacă excludem asemenea situaţii (care se întâlnesc destul de rar în lumea reală), ordinea de acoperire se poate determina prin compararea triunghiurilor fiecare cu fiecare.

Pentru aceasta, trebuie să găsim câte un punct în fiecare din triunghiuri care să aibă aceleaşi coordonate x,y dar care să difere prin z. Atunci ordinea în adâncime este dată de coordonatele z. Este necesară această comparaţie în cazul că două triunghiuri se suprapun.

Testul de Suprapunere a Triunghiurilor Pentru a determina dacă două triunghiuri se suprapun vom utiliza doar coordonatele (x,y), deci lucrăm cu proiecţiile triunghiurilor. Primul test de efectuat este testul minimax pentru două triunghiuri, care ne poate spune dacă două triunghiuri *nu* se suprapun. Pentru fiecare triunghi calculăm dreptunghiul minim care îl conţine şi le comparăm. Dacă dreptunghiurile nu se suprapun, înseamnă că nici triunghiurile nu se suprapun. Altfel, sunt necesare alte teste (Figura 1.21).

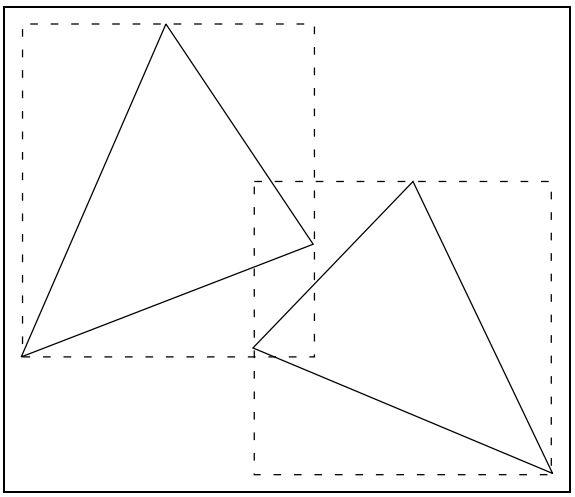

Figura 1.21 Dreptunghiurile minime se suprapun

Pentru triunghiurile rămase, pentru care încă nu putem spune dacă se suprapun sau nu, în continuare căutăm intersecţii ale unor laturi ale triunghiului T_1 cu laturi ale lui T_2. (Lucrăm cu proiecţii!) Dacă găsim cel puţin o intersecţie, nu mai efectuăm alte teste; triunghiurile se suprapun. Totuşi, două triunghiuri se pot suprapune chiar dacă laturile lor nu se intersectează -- atunci când unul din triunghiuri este în interiorul celuilalt.

Dacă două segmente sunt date prin capetele lor, (x_1,y_1), (x_2,y_2) şi (x_3,y_3), (x_4,y_4), căutăm intersecţia lor efectuând mai întâi un test minimax:

$$\max(x_1,x_2) < \min(x_3,x_4)$$
$$\max(x_3,x_4) < \min(x_1,x_2)$$
$$\max(y_1,y_2) < \min(y_3,y_4)$$
$$\max(y_3,y_4) < \min(y_1,y_2)$$

Dacă una din aceste condiţii este adevărată, atunci cele două segmente nu se intersectează. Acest test este mult mai puţin costisitor decât următorul.

Dacă nici una din condiţiile de mai sus nu este îndeplinită, continuăm după cum urmează. În primul rând, verificăm dacă liniile sunt paralele, adică dacă au aceeaşi pantă. Condiţia este:

$$D = (x_3-x_4)\cdot(y_1-y_2)-(x_1-x_2)\cdot(y_3-y_4) = 0$$

Dacă D=0 atunci nu avem intersecţie. (Ar fi mai indicată condiţia $|D|<\varepsilon$, cu ε dependent de precizia aritmetică.)

Dacă $|D|<\varepsilon$, efectuăm un al doilea test. Calculăm:

$$s = [(x_3-x_4)(y_1-y_2)-(x_1-x_3)(y_3-y_4)]/D$$
$$t = [(x_1-x_2)(y_1-y_3)-(x_1-x_3)(y_1-y_2)]/D$$

Dacă `0 < (s,t) < 1`, atunci segmentele au o intersecţie în punctul (x,y) situat între capetele lor:

$$x = x_1 + s(x_2 - x_1)$$
$$y = y_1 + s(y_2 - y_1)$$

(De obicei nu ne interesează coordonatele intersecţiei; este suficient să ştim că ele se intersectează.) Acest test nu ia în considerare cazul când un segment trece printr-unul din capetele celuilalt.

Dacă nici una dintre cele nouă perechi de laturi nu ne dă nici o intersecţie, avem două situaţii posibile: fie unul din triunghiuri îl conţine pe celălalt, fie ele nu se suprapun (Figura 1.22).

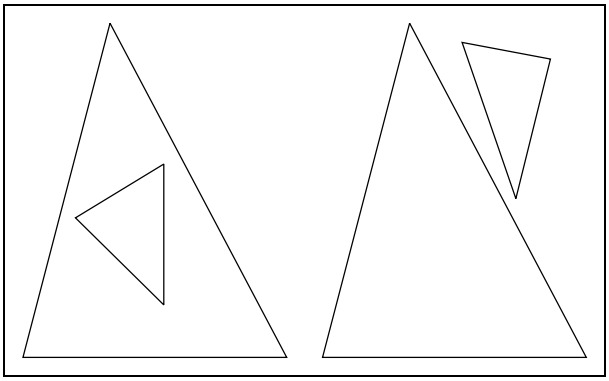

Figura 1.22 Nu există intersecţii de laturi

După testele de mai sus, ultimul test verifică dacă un triunghi îl conţine pe celălalt. Pentru aceasta, efectuăm un test de interior de triunghi pentru un vertex al unui triunghi faţă de celălalt triunghi.

Comparaţia în Adâncime În urma testelor de mai sus, ştim acum, pentru fiecare pereche de triunghiuri, dacă ele se suprapun. Dacă nu, relaţia de adâncime dintre ele nu ne interesează. Dacă da, atunci trebuie să vedem care dintre ele este deasupra. Şi acum avem de-a face cu test simple, care rezolvă majoritatea cazurilor, şi teste mai complexe, care trebuie aplicate dacă primele nu sunt concludente.

O metodă simplă este testul minimax pentru coordonata z. Dacă toate coordonatele z ale unui triunghi sunt mai mici decât ale celuilalt, atunci putem decide care dintre triunghiuri este deasupra. În Figura 1.23 avem două triunghiuri, A şi B, ale căror proiecţii se suprapun, văzute de pe axa y. De reţinut faptul că triunghiurile din figură au fost supuse unei transformări perspectivă în adâncime şi mai este necesară doar o proiecţie ortografică.

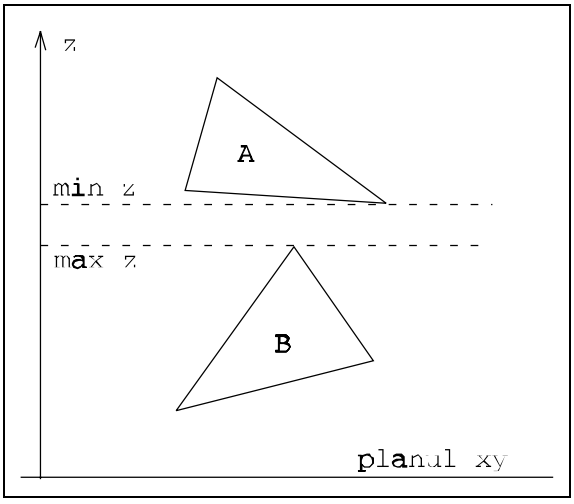

Figura 1.23

În Figura 1.23, cel mai mic z al lui A este mai mare decât cel mai mare z al lui B. Deci triunghiul B acoperă parţial sau total triunghiul A.

$\min(z_A) > \max(z_B) \Rightarrow$ B peste A
$\min(z_B) > \max(z_A) \Rightarrow$ A peste B

Totuşi, problema adâncimii nu se rezolvă întotdeauna atât de simplu. Un caz posibil este prezentat în Figura 1.24. Trebuie să găsim o pereche (x,y) care să reprezinte puncte care aparţin ambelor proiecţii şi apoi să comparăm coordonatele z. Relaţia dintre ele determină

relaţia de adâncime dintre A şi B. Această relaţie de adâncime s-ar putea să nu se aplice triunghiurilor în întregime, dar în mod sigur este valabilă pentru porţiunile care se suprapun.

Linia verticală care trece prin q şi p în Figura 1.24 indică puncte în care avem această relaţie. În acest caz, cele două puncte se află la intersecţia a două laturi în planul (x,y). Întrucât q, care aparţine triunghiului B, are z-ul mai mic, tragem concluzia că triunghiul A este în spatele triunghiului B.

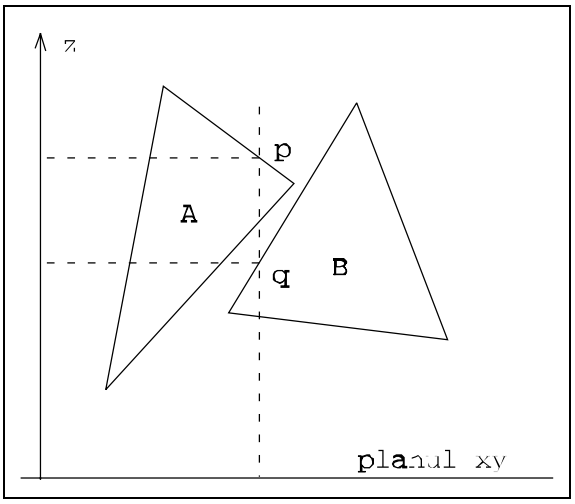

Figura 1.24 Testul minimax pentru z nu este concludent

Dacă testul minimax pentru z nu este concludent, căutăm puncte cu aceleaşi coordonate (x,y) care să aparţină câte unui triunghi. Întrucât ele se suprapun, în mod sigur există asemenea puncte. Deja am găsit dacă proiecţiile a două laturi se intersectează sau dacă un triunghi este în interiorul celuilalt. Dacă am găsit că este vorba de intersecţie de laturi, cunoaştem valorile s şi t; dacă un triunghi este în interiorul celuilalt, îi cunoaştem vârfurile -- fiecare dintre aceste puncte aparţine ambelor triunghiuri. Dacă un triunghi este conţinut în celălalt, comparăm coordonatele z pentru centru. Dacă avem o intersecţie de laturi, procedăm după cum urmează.

Fie (P_1,P_2) şi (P_3,P_4) capetele laturilor care se intersectează. Cunoscând s şi t, calculăm

$$z_A = z_1 + s(z_2 - z_1)$$
$$z_B = z_3 + s(z_4 - z_3)$$

Dacă $z_A<z_B$ atunci A îl acoperă pe B; dacă $z_B<z_A$ atunci B îl acoperă pe A.

Dar dacă $z_A=z_B$? În acest caz încă nu putem trage o concluzie şi trebuie efectuate alte teste. Mai căutăm altă intersecţie de laturi. Nu este sigur că o vom găsi pe una din laturile curente. Întrucât triunghiurile nu sunt coplanare, întotdeauna vom găsi un alt punct de intersecţie care să aibă coordonate z diferite.

Egalitatea $z_A=z_B$ trebuie interpretată ţinând cont de erorile de rotunjire. Egalitatea perfectă probabil că nu se întâlneşte niciodată, aşa că vom testa $|z_A-z_B|<\varepsilon$, cu ε mic. În absenţa lui ε, cele două triunghiuri s-ar putea să se atingă sau să fie foarte apropiate în acel punct din spaţiu, dar în rest să fie foarte îndepărtate. Erorile de precizie aritmetică ne-ar putea da $z_A<z_B$, când în realitate $z_A\geq z_B$ şi am putea ajunge la concluzii greşite.

Am spus că două triunghiuri nu pot fi coplanare, dar ce se întâmplă dacă din cauza preciziei de calcul ele rezultă ca fiind totuşi coplanare? În acest caz ele în realitate sunt *aproape* coplanare şi obiectele de care aparţin se ating în spaţiu. Obiectele reale sunt solide, deci interiorul obiectelor se află pe părţile opuse ale planelor respective. Deci, unul din plane este o faţetă ascunsă şi a fost deja eliminat. Cu alte cuvinte, nu putem întâlni această situaţie!

Recapitulând, dacă testul minimax pentru z nu este concludent, căutăm intersecţii de laturi. Dacă într-una din intersecţii avem egalitate pentru z, atunci căutăm altă intersecţie. Dacă z-urile diferă cu mai mult de ε, ştim care este situaţia, altfel căutăm o a treia intersecţie. Dacă ea există, z-urile trebuie să difere cu mai mult de ε.

Dacă a treia intersecţie nu există, înseamnă că un vârf al triunghiului se află în interiorul celuilalt. În acest caz (vezi Figura 1.25), trebuie să găsim acest vârf. Uneori este vorba de unul singur, alteori pot exista 2 vertex-uri. Trebuie efectuat un test minimax punct-triunghi, urmat, dacă este cazul, de un test de interior de triunghi pentru toate combinaţiile vertex-triunghi, până când găsim acest vertex. Pentru acest punct, p, calculăm coordonatele z ale planelor în care se află fiecare dintre triunghiuri.

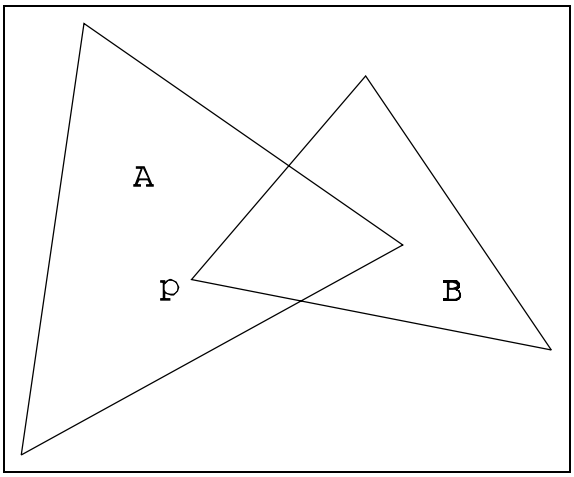

Figura 1.25 Numai două intersecţii de laturi

Fie A triunghiul a cărui proiecţie conţine punctul $p=(p_x,p_y)$ şi (x_1,y_1,z_1), (x_2,y_2,z_2) şi (x_3,y_3,z_3) vertex-urile sale. Fie B celălalt triunghi. Pentru punctul p:

$$z_A = -\frac{a}{c}(p_x-x_1) - \frac{b}{c}(p_y-y_1) + z_1$$

cu

$$a = (y_2-y_3)(z_1-z_3) - (z_2-z_3)(y_1-y_3)$$
$$b = (z_2-z_3)(x_1-x_3) - (x_2-x_3)(z_1-z_3)$$
$$c = (x_2-x_3)(y_1-y_3) - (y_2-y_3)(x_1-x_3)$$

$$z_B = p_z$$

Dacă $z_A<z_B$ atunci A îl acoperă pe B; dacă $z_B<z_A$, atunci B îl acoperă pe A.

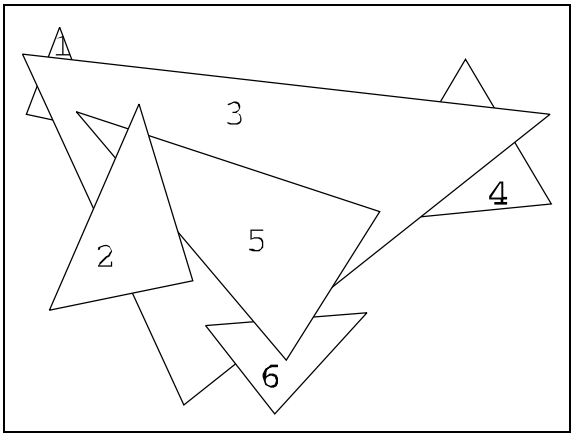

Figura 1.26

Stabilirea Ordinii în Adâncime În acest moment putem determina pentru fiecare pereche de triunghiuri dacă ele se suprapun şi dacă da, care dintre ele este mai aproape de noi. Pentru a exprima această *ordine de adâncime*, pentru fiecare triunghi se creează o listă care conţine pointeri la acele triunghiuri care îl acoperă. Mai păstrăm un contor al triunghiurilor care se află în spatele lui. Pentru exemplul din Figura 1.26 avem (vezi tabelul 1).

În pseudocod:

```
for i := 1 to n-1 do
 for j := i+1 to n do begin
  if △ i şi j se suprapun
   then begin
     găseşte relaţia de adâncime
      if j este în faţa lui i
      then begin
        adaugă j la lista lui i
        incrementează contor[j]
      end
      else begin
```

```
        adaugă i la lista lui j
        incrementează contor[i]
      end
    end
end
```

Tabelul 1

triunghi	contor	lista
1	0	3 5 2
2	3	
3	2	2 5 6
4	0	3 5
5	4	2
6	1	5

4.3 Îndepărtarea Suprafeţelor Ascunse (Algoritmul Pictorului)

În acest moment am îndepărtat faţetele ascunse, am descompus poligoanele rămase în triunghiuri şi le-am sortat în adâncime. Am completat structurile de date descrise şi suntem gata să afişăm triunghiurile ca suprafeţe solide sau sub formă de contururi. Pentru început, vom prezenta *algoritmul pictorului*. Pentru a afişa doar contururile este necesar să eliminăm liniile ascunse, procedeu prezentat mai târziu.

În comparaţie cu ce s-a făcut până acum, algoritmul pictorului este foarte simplu. Parcurgem tabloul şi desenăm triunghiurile care au contorul zero. Acestea nu acoperă alte triunghiuri. Când trasăm un astfel de triunghi, decrementăm contoarele triunghiurilor din lista respectivă (numărul de triunghiuri de sub ele s-a redus cu cel pe care îl desenăm). Marcăm contorul triunghiului desenat, pentru a nu-l mai lua în considerare. În urma acestor operaţii vor rezulta noi contoare nule. Continuăm acest proces, până când nu mai avem triunghiuri de afişat.

Secvenţa din Figura 1.27 prezintă evoluţia algoritmului pictorului. Datele iniţiale sunt cele din Figura 1.26. Operaţiile care se execută şi structurile de date sunt prezentate mai jos.

△		lista	cnt	cnt. după	nr.
1	trasare	3 5 2	0	*	
2			3	2	
3		2 5 6	2	1	
4		3 5	0	0	
5		2	4	3	
6		5	1	1	1
1		3 5 2	*	*	
2			2	2	
3		2 5 6	1	0	
4	trasare	3 5	0	*	
5		2	3	2	
6		5	1	1	2
1		3 5 2	*	*	
2			2	1	
3	trasare	2 5 6	0	*	
4		3 5	*	*	

```
5           2       2    1
6           5       1    0    3

1           3 5 2   *    *
2                   1    1
3           2 5 6   *    *
4           3 5     *    *
5           2       1    0
6 trasare   5       0    *    4

1           3 5 2   *    *
2                   1    0
3           2 5 6   *    *
4           3 5     *    *
5 trasare   2       0    *
6           5       *    *    5

1           3 5 2   *    *
2 trasare           0    *
3           2 5 6   *    *
4           3 5     *    *
5           2       *    *
6           5       *    *    6
```

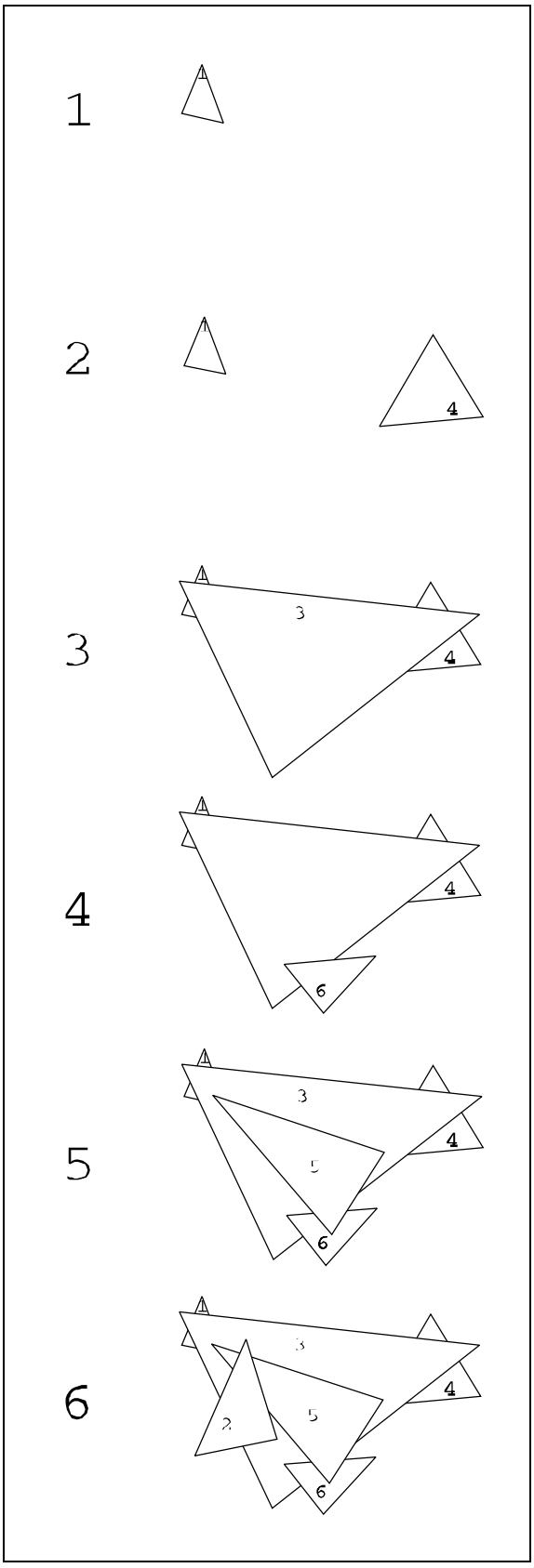

Figura 1.27

Algoritmul pictorului este valabil doar pentru dispozitive raster şi doar dacă avem de afişat obiecte solide, prin poligoane pline. În esenţă, se bazează pe faptul că o zonă dată, setată pe o anumită culoare, pe urmă poate fi setată pe altă culoare. Culoarea fiecărui pixel poate fi modificată în mod repetat. Aceasta înseamnă că atunci când se desenează un poligon, el se suprapune peste ceea ce era desenat anterior.

O Posibilă Metodă de Îndepărtare a Liniilor Ascunse Acest algoritm ar putea fi utilizat şi pentru afişarea unor obiecte solide prin trasarea laturilor. Scena poate fi desenată prin umplere de poligoane. Mai întâi umplem cu culoarea de fond, apoi trasăm conturul poligonului. Umplerea va şterge toate liniile afla în spatele poligonului. Ordinea în care se

efectuează operaţiile, umplere apoi trasarea conturului, este esenţială, pentru a evita ca umplerea să şteargă şi conturul poligonului în cauză.

După cum am mai spus, o scenă compusă nu numai din triunghiuri ci şi din poligoane mai mari, este bine să fie reprezentată intern doar prin triunghiuri. Patrulaterul abcd din Figura 1.28 este reprezentat ca două triunghiuri, abd şi bcd. Când umplem cele două triunghiuri cu aceeaşi culoare, ele se contopesc la conturul poligonului. Dar când dorim să reprezentăm doar laturile, va fi afişată şi linia care le desparte, bd. Cum putem evita acest lucru ? O soluţie simplă este să adăugăm fiecărui vertex al triunghiului (intern - un pointer) un operator M sau D, care va avea semnificaţia de mutare sau desenare. Cele două triunghiuri vor fi atunci DaDbMd şi MbDcDd. Trebuie să ţinem cont de aceşti operatori atunci când trasăm conturul triunghiurilor.

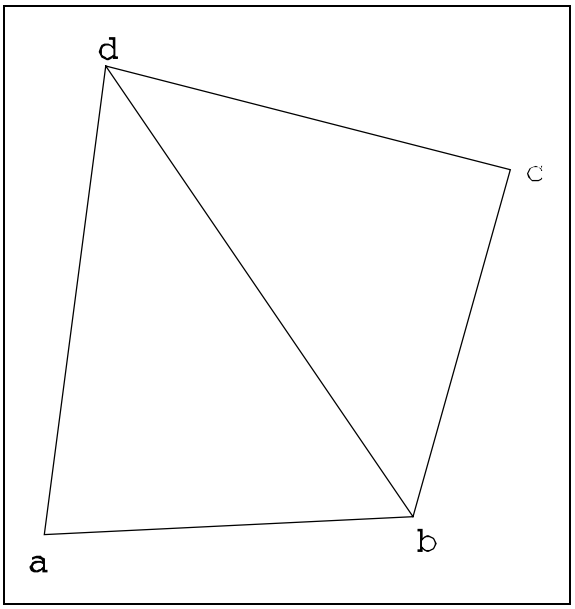

Figura 1.28

4.4 Îndepărtarea Liniilor Ascunse

Pe display-uri vectoriale nu avem posibilitatea să ştergem o linie prin scriere peste ea cu culoarea de fond, iar în cazul plotter-elor nu este posibilă nici un fel de ştergere. În ambele cazuri, o linie o dată trasată nu poate fi îndepărtată prin acoperire, deci nu putem utiliza algoritmul pictorului. Totuşi, încă se mai utilizează display-uri vectoriale, iar plotter-ele vor mai fi în uz încă multă vreme. Pentru astfel de dispozitive avem nevoie de un alt algoritm pentru îndepărtarea liniilor şi suprafeţelor ascunse. De fapt, nu se pune problema îndepărtării suprafeţelor, întrucât nu se practică umplerea pe astfel de dispozitive (deşi aceasta s-ar putea efectua pe un plotter). Ne vom îndrepta atenţia spre îndepărtarea liniilor ascunse.

Operaţiile pregătitoare pentru algoritmul pictorului, descompunerea în triunghiuri şi sortarea în adâncime, sunt necesare şi în acest caz. Va trebui doar să efectuăm operaţii suplimentare care să elimine trasarea părţilor ascunse ale contururilor poligonale.

Se porneşte cu tabloul de triunghiuri ordonate, ca şi la algoritmul pictorului, dar în loc să trasăm un poligon plin atunci când găsim un triunghi cu contor nul, vom efectua o "desenare cu linii ascunse" (DLA). O DLA este un proces îndelungat, care necesită calcule laborioase. În rest, algoritmul este identic. Atunci când găsim un triunghi cu contorul zero, îl DLA, parcurgem lista sa de triunghiuri care îl acoperă şi decrementăm contoarele acestor triunghiuri. De asemenea, marcăm contorul acestui triunghi, pentru a nu-l mai lua în considerare în continuare. Ca rezultat, apar noi contoare nule şi continuăm acest proces până când nu mai avem triunghiuri de afişat.

Înainte de a explica procesul de DLA trebuie să clarificăm relaţia dintre un segment şi un triunghi. În Figura 1.29 se prezintă un segment, ab, şi un triunghi. Există 5 modalităţi în care triunghiul acoperă segmentul:

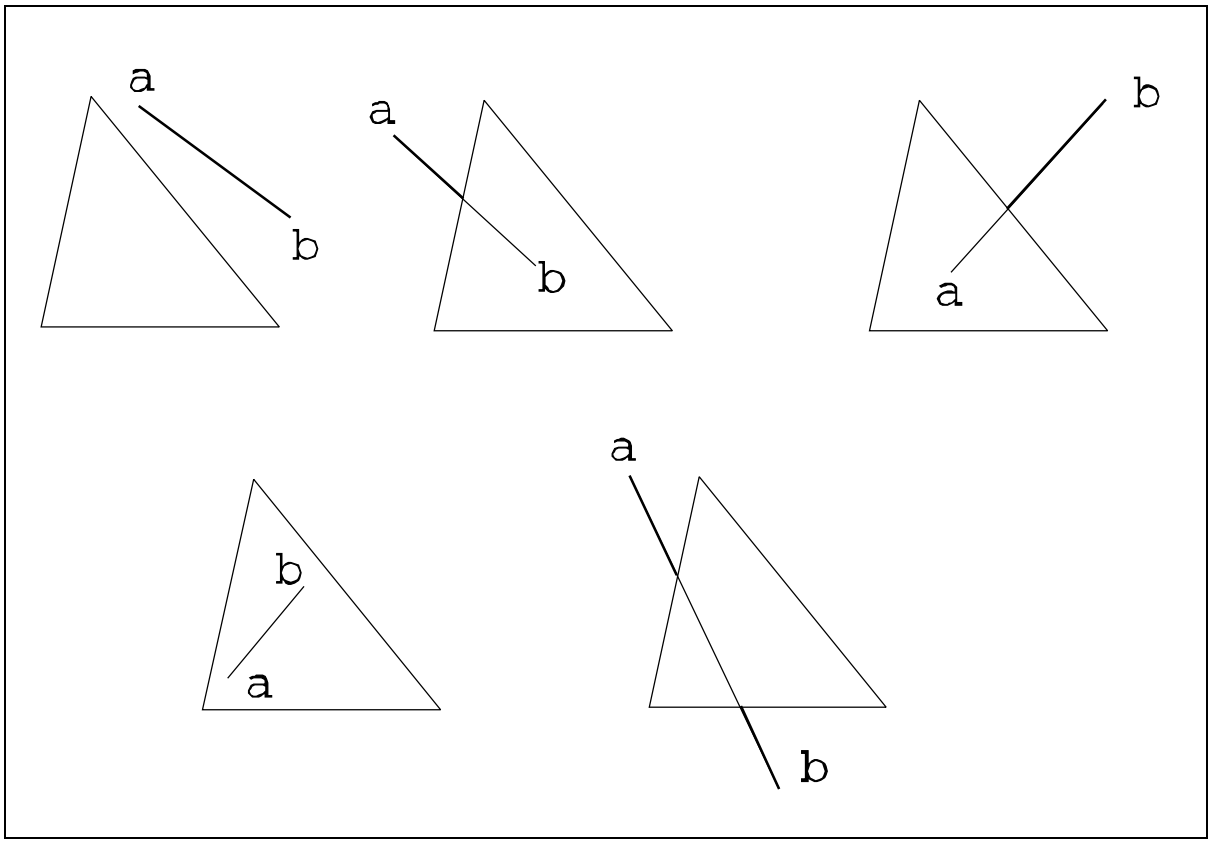

Figura 1.29 Poziţiile unui segment faţă de un triunghi

1. segmentul nu este ascuns;
2. a - ascuns, b - vizibil;
3. a - vizibil, b - ascuns;
4. segmentul este acoperit în întregime;
5. capetele - vizibile, o porţiune acoperită.

Pentru DLA avem nevoie de o rutină care să determine dacă un anumit triunghi acoperă un segment dat şi să calculeze punctul sau punctele de intersecţie. Deocamdată presupunem că dispunem de această rutină, pentru a ne concentra asupra principiului algoritmului.

Când trebuie să DLA un triunghi, îl descompunem în cele trei laturi şi prelucrăm fiecare latură în parte. Lista de triunghiuri conţine acele triunghiuri care pot să acopere oricare dintre cele trei laturi. Vom forma deci pentru fiecare latură o listă a triunghiurilor de deasupra.

Atunci când testăm o latură şi un triunghi este posibil să ajungem în cazul 5, când rezultă două segmente. Fiecare dintre aceste părţi trebuie comparată cu triunghiurile rămase în listă. Fiecare comparaţie poate să mărească numărul de segmente cu 1 şi trebuie reţinute toate. Pentru aceasta avem nevoie de o stivă în care să memorăm toate segmentele care trebuie testate. Iniţializăm stiva cu cele trei vertex-uri ale triunghiului care trebuie DLA. De asemenea, iniţializăm aşa-numitul "*punct curent*" cu al treilea vertex (punctul de start al primului segment). Segmentul testat este cel dintre punctul curent şi punctul memorat în vârful stivei. Împreună cu fiecare punct din stivă memorăm o comandă M sau D (iniţial toate

sunt D) şi lista triunghiurilor de deasupra. Fie triunghiul `abc`, cu triunghiurile 1, 2 şi 3 deasupra. Stiva va fi iniţializată cu

```
pct.crt.          comanda  punct  lista
   c      vârf →     D       a    1 2 3
                     D       b    1 2 3
                     D       c    1 2 3
```

Aceasta înseamnă că, înainte de a-l desena, segmentul de la punctul c (punctul curent) la punctul a (vârful stivei) trebuie comparat cu triunghiurile 1, 2 şi 3. (De fapt, stiva va conţine doar pointeri la listele triunghiurilor de deasupra, dar, pentru claritate, specificăm lista completă. Listele indicate trebuie să fie liste separate, chiar dacă iniţial ele sunt identice, întrucât vor fi modificate pe parcurs.)

În continuare vom preciza algoritmul de DLA în pseudocod. "comanda" şi "lista" se referă la conţinutul câmpurilor respective din vârful stivei.

DLA începe prin iniţializarea stivei cu cele trei vertex-uri ale triunghiului. Apoi verifică comanda şi lista; dacă comanda este M, nu se trasează nimic, se execută mutare. Dacă lista este vidă, nu avem nici un triunghi care să acopere segmentul şi deci se poate executa comanda, indiferent dacă ea este D sau M. Atunci când se efectuează o comandă, punctul curent ia valoarea celui din vârful stivei şi se extrage vârful stivei.

Dacă este o comandă D şi lista nu este vidă, se compară segmentul de la punctul curent la vârful stivei cu primul triunghi din listă. Aceasta se face prin apelul unei rutine pe care o vom descrie mai târziu. Indiferent de rezultat, nu mai avem nevoie de acest triunghi şi îl extragem din listă.

Acum, în funcţie de poziţia în care se află segmentul faţă de triunghi (vezi Figura 1.29), trebuie efectuate diferite operaţii:

cazul 1: nu se execută nimic. Urmează comparaţia cu următorul triunghi din listă.

cazul 2: o intersecţie cu triunghiul şi triunghiul acoperă punctul curent. Este necesară o mutare la punctul de intersecţie, urmată de o desenare până la punctul din vârful stivei. Comanda D este deja în vârful stivei, trebuie adăugată la stivă o mutare la punctul de intersecţie.

cazul 3: avem o intersecţie şi punctul curent este vizibil. Dorim trasare până în punctul de intersecţie, apoi mutare la vârful stivei (chiar dacă punctul din vârful stivei este acoperit, avem nevoie de el pentru următoarea latură a triunghiului). Deci se modifică comanda în M. Înaintea acestui M avem nevoie de D la intersecţie, care se adaugă la vârful stivei. Această trasare ar putea fi acoperită de triunghiurile rămase, deci primeşte aceeaşi listă.

cazul 4: tot segmentul este ascuns - comanda devine M.

cazul 5: două intersecţii. În loc să trasăm până la punctul din vârful stivei, putem trasa doar până la cea mai apropiată intersecţie, o mutare la intersecţia următoare şi apoi o trasare la vârful stivei. Ultima trasare se află deja în stivă; se adaugă la stivă un M la cea mai îndepărtată intersecţie şi un D la cea mai apropiată,

împreună cu lista rămasă.

Descriere în Pseudocod:

```
DLA pentru triunghiul a, b, c:
begin
   push(c şi lista);
   push(b şi lista);
   push(a şi lista);
   pct_crt := c;
   repeat
      if comanda=M sau lista vidă
      then begin
         efectuează comanda;
         pct_crt := pop();
      end
      else begin
         compară primul triunghi din listă cu
         segmentul dintre pct_crt şi vârful
         stivei;
         extrage primul triunghi din listă;
         case 1
            nimic;
         case 2
            push(M şi intersecţia);
         case 3
            schimbă comanda în M;
            push(D, intersecţia şi lista);
         case 4
            schimbă comanda în M;
         case 5
            push(M şi intersecţia mai depărtată);
            push(D, inters. apropiată şi lista)
      end {else}
   until stiva vidă
end {DLA};
```

Vom exemplifica pe cazul particular din Figura 1.30. Triunghiul 1 trebuie DLA; are deasupra triunghiurile 2 şi 3.

```
c  D a  2 3 comanda=D, lista nu e vidă
   D b  2 3 compară ca cu 2, şterge 2;
   D c  2 3 cazul 5, intersecţii d, e;
            push(M, e);
            push(D, d şi lista);

c  D d  3   comanda=D, lista nu e vidă:
   M e      compară cd cu 3, şterge 3;
   D a  3   cazul 3, intersecţie f:
   D b  2 3 comanda := M ;
   D c  2 3 push(D, f şi lista);
```

```
c  D f      lista vidă:
   M d      trasează la f;
   M e      pct_crt := f;
   D a  3   pop;
   D b  2 3
   D c  2 3

f  M d      comanda=M
   M e      mutare în d;
   D a  3   pct_crt := d;
   D b  2 3 pop
   D c  2 3

d  M e      comanda = M
   D a  3   mutare în e;
   D b  2 3 pct_crt := e;
   D c  2 3 pop;

e  D a  3   comanda=D, lista nu e vidă
   D b  2 3 compară ea cu 3, şterge 3;
   D c  2 3 cazul 1: nimic

e  D a      lista vidă:
   D b  2 3 trasează la a;
   D c  2 3 pct_crt := a; pop;

a  D b  2 3 comanda=D, lista nu e vidă
   D c  2 3 compară ab cu 2, şterge 2; cazul 1: nimic

a  D b  3   comanda=D, lista nu e vidă
   D c  2 3 compară ab cu 3, şterge 3; cazul 1: nimic

a  D b      lista vidă: trasează la b;
   D c  2 3 pct_crt := b; pop

b  D c  2 3 comanda=D, lista nu e vidă
            compară bc cu 2, şterge 2;
            cazul 5: intersecţii g, h;
            push(M, h);
            push(D, g şi lista);

b  D g  3   comanda=D, lista nu e vidă:
   M h      compară bg cu 3, şterge 3;
   D c  3   cazul 1: nimic

b  D g      lista vidă: trasează la g;
   M h      pct_crt := g;
   D c 3    pop;

g  M h      comanda=M: mutare h;
   D c  3   pct_crt := h; pop;
```

şi aşa mai departe.

Să vedem acum o rutină care determină în care din cele cinci situaţii ne încadrăm, iar

pentru cazurile 2, 3 şi 5 calculează intersecţiile. Atunci când am descompus poligoanele în triunghiuri am efectuat o serie de teste, pentru a determina dacă un punct este sau nu în interiorul unui triunghi. Mai întâi, am efectuat testul minimax punct-triunghi; dacă testul nu a dăt rezultatul "în exterior", a trebuit să efectuăm un test de interior de triunghi. Aici trebuie să facem acelaşi lucru. Trebuie să vedem dacă punctul curent şi cel din vârful stivei sunt sau nu în interiorul triunghiului. Nu repetăm algoritmii, dar vom descrie ce trebuie efectuat în funcţie de rezultatele celor două teste.

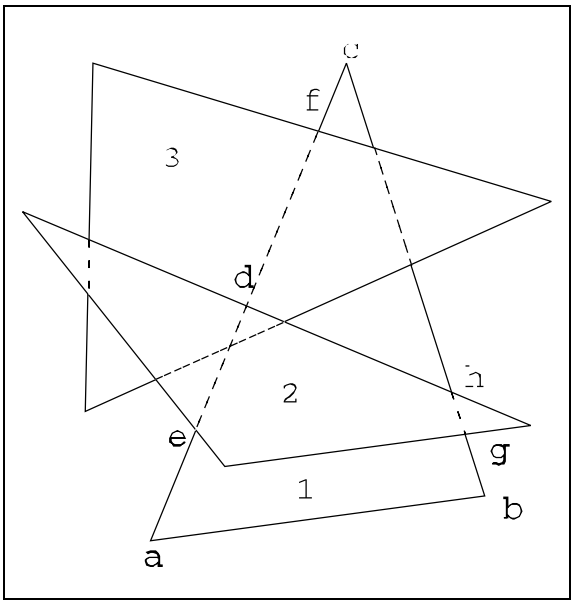

Figura 1.30

Rutina nu necesită ca argumente segmentul şi triunghiul care se testează, întrucât ele sunt aceleaşi. Codificarea trebuie întotdeauna să se refere la segmentul dintre punctul curent şi cel din vârful stivei şi la primul triunghi din listă. Rutina va returna trei elemente: un întreg - numărul cazului în care ne încadrăm şi două valori reale, pentru intersecţii (dacă ele există). Am văzut deja formulele pentru calculul intersecţiilor.

Atunci când testăm două puncte faţă de un triunghi putem întâlni următoarele situaţii:

a) ambele puncte sunt în interior. Rezultă că triunghiul acoperă segmentul în întregime: se returnează cazul 4.
b) un punct în interior, celălalt în exterior: se returnează cazul 2 sau 3, în funcţie de care din puncte este în interior. Apoi se caută intersecţia fiecărei laturi a triunghiului cu segmentul. Pentru una dintre laturi există în mod sigur o intersecţie, care se returnează.
c) ambele capete sunt în exterior. Pentru a vedea dacă suntem în cazul 1 sau 5, căutăm intersecţia fiecărei laturi cu segmentul. Dacă nu există, suntem în cazul 1, segmentul nu este acoperit de către triunghi. Dacă există intersecţii, ele vor fi două - cazul 5. Se caută care dintre ele este mai aproape de punctul curent şi se returnează ca primă intersecţie, iar cealaltă în al doilea rezultat. Intersecţia cea mai apropiată este uşor de determinat. Fie (p_x,p_y) punctul curent şi (a_x,a_y), (b_x,b_y) cele două intersecţii. Dacă

$$|p_x-a_x| < |p_x-b_x|$$

sau

$$|p_y-a_y| < |p_y-b_y|$$

atunci (a_x,a_y) este intersecţia cea mai apropiată.

Acestea sunt operaţiile necesare pentru îndepărtarea suprafeţelor şi liniilor ascunse. Reamintim faptul că nu sunt permise acoperirile ciclice şi nici poligoanele întrepătrunse. Astfel de situaţii ar putea fi tratate prin sortare în adâncime, dar algoritmul devine mult mai complex şi mare consumator de timp de calcul. Sunt mai indicate metoda Z-buffer sau cea a subdiviziunii, prezentate în capitolul următor, care sunt capabile să trateze aproape orice situaţie.