3

Linii şi Suprafeţe Ascunse: Cazuri Speciale

În acest al treilea capitol despre îndepărtarea liniilor şi suprafeţelor ascunse, vom trata două cazuri speciale; ambele afişează funcţii matematice de două variabile independente. Vor fi trei secţiuni.

1 Introducere furnizează o serie de informaţii generale.

2 Îndepărtarea Liniilor Ascunse pentru Funcţii Bi-Dimensionale explică modul în care se trasează curbe care reprezintă o suprafaţă prin trasare de linii la intervale constante, pentru fiecare din variabilele independente. Dificultatea apare la al doilea set de curbe.

3 Suprafeţe Grilă prezintă modul în care se afişează funcţii în care suprafaţa este o interpolare liniară a valorilor în punctele unei grile, uniform distribuite pe ambele direcţii.

1 Introducere

În aplicaţiile matematice şi inginereşti se obişnuieşte să se afişeze o funcţie de suprafaţă de forma

$$z = f(x,y)$$

În această notaţie, planul (x,y) este imaginat orizontal, iar valorile z sunt înălţimi; z defineşte o suprafaţă în spaţiu. In grafica pe calculator, aproape toate sistemele de coordonate plasează planul (x,y) în ecran, cu axa z perpendiculară pe acesta. Forma de mai sus ar putea intră în conflict fie cu imaginaţia noastră, fie cu convenţia coordonatelor în grafica pe calculator, aşa că vom exprima funcţia ca:

$$y = f(x,z)$$

Aici, funcţia este definită în planul orizontal (x,z). Esenţial este ca f să fie o funcţie cu

valoare unică pentru x şi z, altfel vom avea două suprafeţe. O astfel de funcţie poate fi afişată fără liniile ascunse utilizând algoritmul orizontului flotant (relativ simplu). Vom prezenta acest algoritm pentru diplay-uri raster.

2 Îndepărtarea Liniilor Ascunse pentru Funcţii Bi-Dimensionale

Vom afişa funcţia prin trasarea a două seturi de curbe în spaţiu. Primul set de curbe sunt intersecţii ale lui f(x,z) cu plane z=constant; celălalt set reprezintă intersecţii ale lui f(x,z) cu plane x=constant. Ca efect, suprafaţa este afişată sub forma unei reţele. Trasarea acestor curbe este simplă; dificultatea apare la îndepărtarea liniilor ascunse. Vom dezvolta algoritmul în mai mulţi paşi.

2.1 Trasarea Unidirecţională

Atunci când trasăm primul set de curbe, eliminarea părţilor ascunse ale curbelor este simplă. Vom presupune că primul set de curbe reprezintă intersecţii ale lui f(x,z) cu plane z=constant. Dacă poziţia observatorului este pe semiaxa z negativă, planele z=constant corespunzătoare la valori mai mici vor fi mai apropiate.

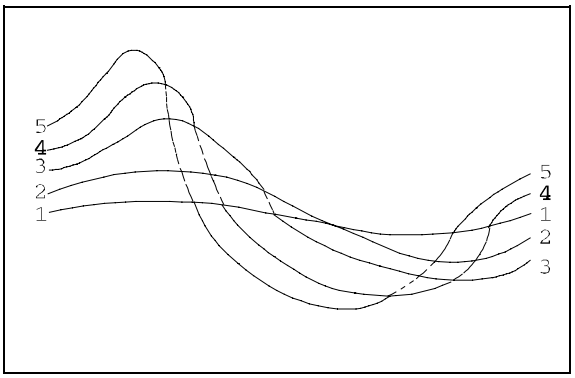

Figura 3.1

Ideea de bază a algoritmului este:

a. Trasează curbele mai apropiate înaintea celor mai îndepărtate.

b. Trasează numai acele părţi ale fiecărei curbe care sunt deasupra sau mai jos faţă de ceea ce s-a trasat până în acel moment.

Aceasta este situaţia din Figura 3.1. Curbele de la 1 la 5 se trasează în această ordine. Atunci când o curbă este sub orizontul superior sau peste orizontul inferior, nu se trasează; afişăm doar acele părţi care sunt deasupra orizontului superior sau sub orizontul inferior. Se actualizează fiecare orizont cu partea care îl depăşeşte. În partea din stânga vedem faţa superioară a suprafeţei; pe mijloc şi în partea dreaptă avem faţa inferioară.

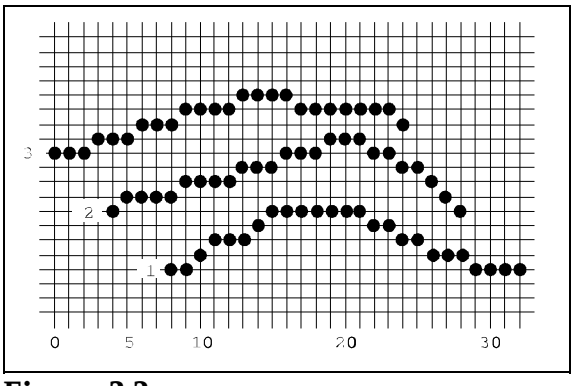

Figura 3.2

Bineînţeles, curbele care formează suprafaţa se trasează prin setarea pixelilor pe un display raster. Presupunem că display-ul are lăţimea de w pixeli şi înălţimea de h pixeli. Fiecare pixel care trebuie setat are coordonatele întregi ix şi iy; ix ∈ [1,w] şi iy ∈ [1,h]. Orizonturile superior şi inferior se păstrează în două

tablouri de numere întregi de dimensiune w, numite up_hor şi lo_hor. Up_hor va conţine valorile iy cele mai mari, iar lo_hor pe cele mai mici trasate până în acel moment.

Ne vom limita la trasarea unei curbe de la stânga la dreapta, în incrementi de (cel mult) un pixel pe orizontală. Aceasta se realizează prin calcularea lui y=f(x,z), z=constant, incrementând pe x astfel încât să avem paşi de cel mult un pixel pe diplay. Pasul vertical poate avea orice valoare.

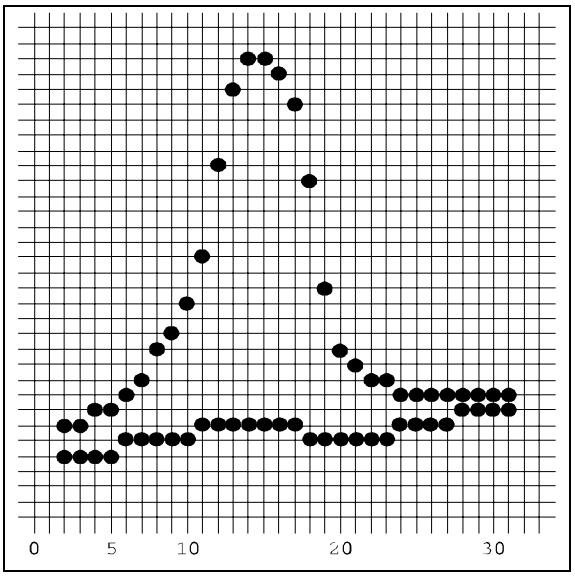

Figura 3.3

Fie y înălţimea curbei în poziţie (x,z). Mai întâi rotunjim pe x şi y: curr_x=round(x) şi curr_y=round(y), apoi comparăm pe curr_y cu up_hor[curr_y]. Dacă este mai mare setăm pixelul (curr_x,curr_y) şi actualizăm up_hor[curr_x]. Altfel, comparăm curr_y cu lo_hor[curr_y]. Dacă este mai mic, setăm pixelul (curr_x,curr_y) şi modificăm lo_hor[curr_x].

Pentru începerea procesului, trebuie să iniţializăm up_hor şi lo_hor. O variantă ar fi să le iniţializăm cu valorile primei curbe. Aceasta ar fi valabil dacă toate celelalte curbe au aceeaşi întindere pe orizontală ca şi prima curbă. Totuşi, dacă suprafaţa este privită oblic, curbele care urmează se pot întinde mai mult la stânga sau la dreapta. Un exemplu este prezentat în Figura 3.2. Curba 1 porneşte cu curr_x=8, dar curba 2 începe cu curr_x=5 şi ar găsi elemente neiniţializate în up_hor şi lo_hor. Similar pentru curba 3 şi următoarele.

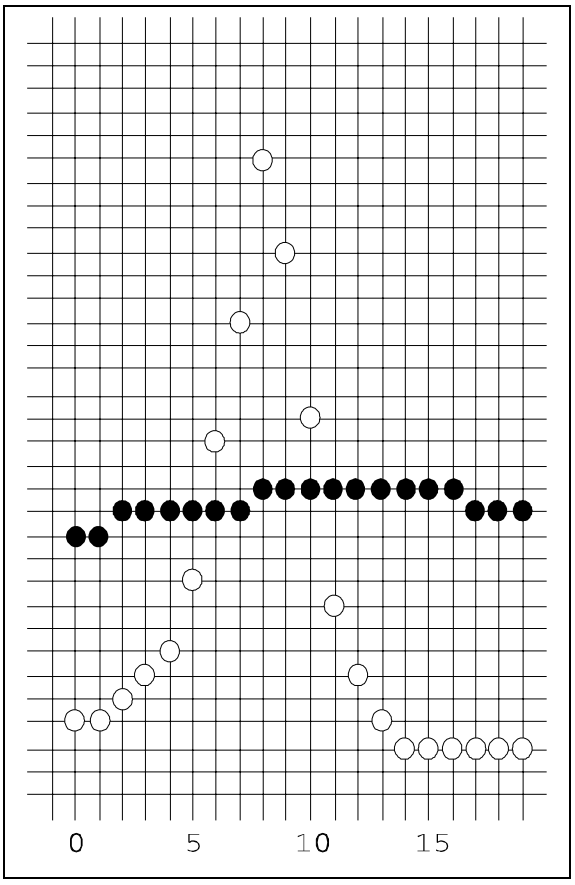

Figura 3.4

Avem nevoie de o regulă de iniţializare mai generală. Vom iniţializa cele două tablouri cu valori cu valori care nu vor fi întâlnite pe parcurs. Apoi, când se calculează valoarea curbei în curr_x, vom şti dacă up_hor[curr_x] şi lo_hor[curr_x] au fost modificate. Dacă nu, le setăm pe această valoare a primei curbe. Dacă au fost modificate, le utilizăm pentru comparaţii şi le actualizăm, dacă este cazul, după cum s-a descris mai sus.

Mai trebuie luată în considerare o altă problemă. Figura 3.3 prezintă o curbă foarte plată şi una foarte abruptă. Părţile abrupte arată foarte dispersate, deoarece se avansează câte un pixel la fiecare pas orizontal. Ce se poate face?

Dacă am utiliza paşi mai mici pe direcţia x când evaluăm funcţia, atunci pentru o porţiune foarte abruptă a funcţiei mai multe valori ale lui x vor fi rotunjite la acelaşi curr_x, deşi valorile y vor fi diferite. Pe măsură ce y creşte, se vor seta pixelii corespunzători la acelaşi curr_x, astfel încât curba va arăta ceva mai densă pe porţiunea ascendentă, însă, pe porţiunea descendentă se va seta un singur pixel, deoarece orizontul superior îi va elimina pe ceilalţi. De asemenea, prin mai multe evaluări ale funcţiei se consumă timp. De aceea, aceasta nu este o soluţie acceptabilă.

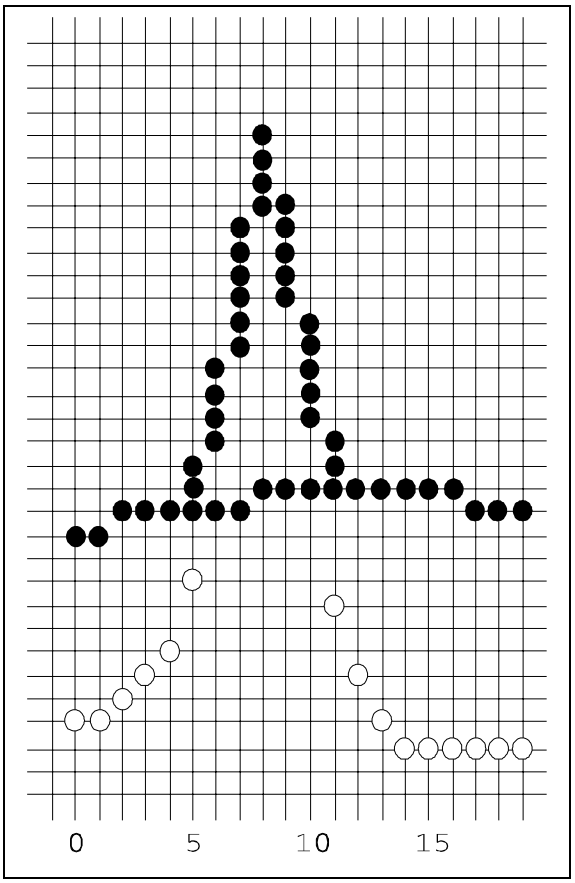

Figura 3.5

O soluţie mai bună este să setăm un singur pixel la fiecare poziţie evaluată a curbei şi să unim aceste poziţii prin segmente de dreaptă. Trebuie să cunoaştem punctul de start al segmentului - memorăm ultimul curr_x. (De reţinut că valorile succesive pentru x nu conduc neapărat la creşterea lui curr_x şi că, mai târziu, va trebui să evaluăm funcţia pentru perechi (x,z) care pot conduce la descreşterea lui curr_x.)

Atunci când cunoaştem poziţia curr_x precedentă, cunoaştem poziţia de start a segmentului; dacă noua valoare curr_x este deasupra orizontului superior, segmentul începe de la orizontul superior, la precedenta poziţie curr_x. Acest lucru este valabil chiar dacă poziţia curr_y precedentă este sub orizontul superior, întrucât părţile de sub orizont nu se trasează. O situaţie similară se aplică pentru un segment trasat până la un punct al curbei situat sub orizontul inferior. De aceea, vom trata în detaliu numai cazul cu orizontul inferior. În Figura 3.4, orizontul superior este reprezentat prin puncte negre. Noua curbă este foarte abruptă şi este indicată prin puncte albe. Vom preciza punctele albe cu notaţia curr_y[i], desi curr_y nu este un tablou în cadrul algoritmului; de exemplu, curr_y[5] este punctul alb din coloana 5.

Pentru simplificare, presupunem că valorile curr_x cresc cu 1 la fiecare evaluare a funcţiei. Pentru curr_x între 0 şi 5, valorile curr_y sub orizontul superior - ce se execută în acest caz va fi descris mai târziu. La curr_x=6 valoarea curr_y este deasupra orizontului superior: curr_y[6]>up_hor[6].

Valoarea precedentă a lui curr_x este 5. Se observă că trebuie să trasăm un segment până în punctul (6,curr_y[6]). Punctul de start nu este (5,curr_y[5]), ci trebuie să fie (5,up_hor[5]), ca să nu fie trasate şi porţiunile de sub orizont. De asemenea, noua valoare a lui up_hor[6]este curr_y[6]. Situaţia se păstrează până la curr_x=10.

La curr_x=11 întâlnim o situaţie nouă: curr_y este sub orizont, curr_y[11]<up_hor[11]. Trebuie să trasăm un segment de la (10,up_hor[10]) la (11,up_hor[11]), ca să nu trasăm şi partea de sub orizont. Pentru a formaliza: fie prec_x valoarea precedentă a lui curr_x. Atunci

putem întotdeauna să trasăm un segment de la (prec_x,up_hor[prec_x]) la (curr_x,up_hor[curr_x]). Aplicăm această regulă şi pentru porţiunea curbei cuprinsă între curr_x=0 şi curr_x=5. Aceasta va redesena orizontul superior. Rezultatele aplicării acestei metode este prezentat în Figura 3.5.

Generalizând pentru ambele orizonturi, rezultă un algoritm simplu. Iată acţiunea pentru un pas:

```
Fie prec_x şi curr_x poziţia orizontală precedentă, respectiv curentă;
fie curr_y valoarea funcţiei în curr_x;

pentru orizontul superior:
 up_hor[curr_x] = max(up_hor[curr_x],curr_y);
 trasează linie între (prec_x,up_hor[prec_x]) şi
 (curr_x,up_hor[curr_x]);

pentru orizontul inferior:
 lo_hor[curr_x] = min(lo_hor[curr_x],curr_y);
 trasează segment între (prec_x,lo_hor[prec_x]) şi
 (curr_x,lo_hor[curr_x]).
```

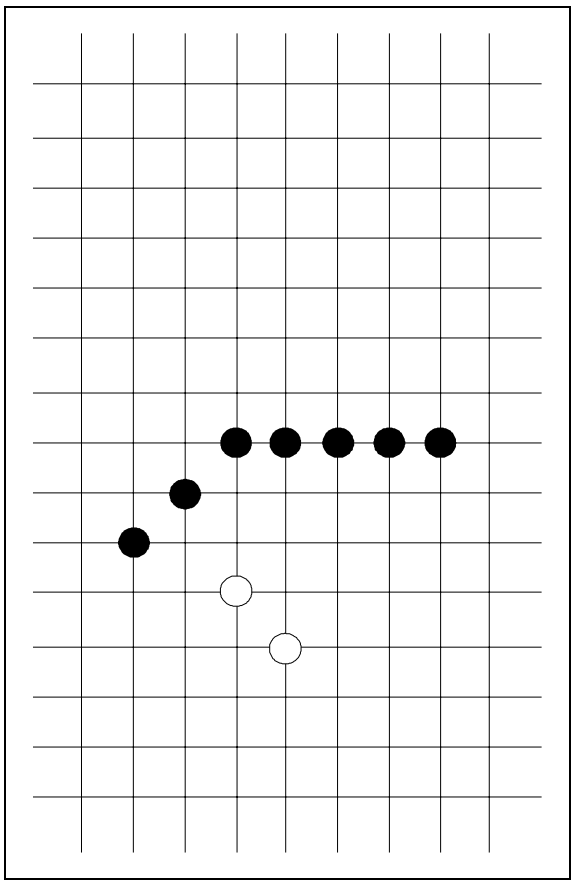

Figura 3.6
prec_y < up_hor[prec_x];
curr_y < up_hor[curr_x].

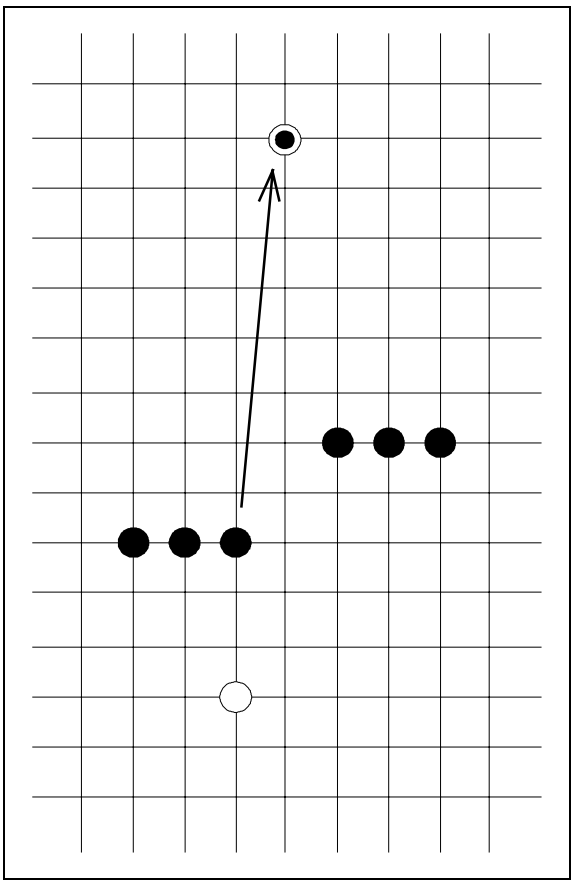

Figura 3.7
prec_y < up_hor[prec_x];
curr_y = up_hor[curr_x].

Repetând aceasta pentru valori succesive pentru x, ne rezultă o curbă. Trasând toate curbele

pentru valori succesive ale lui z, se afişează suprafaţa, dar cu marcare pe o direcţie.

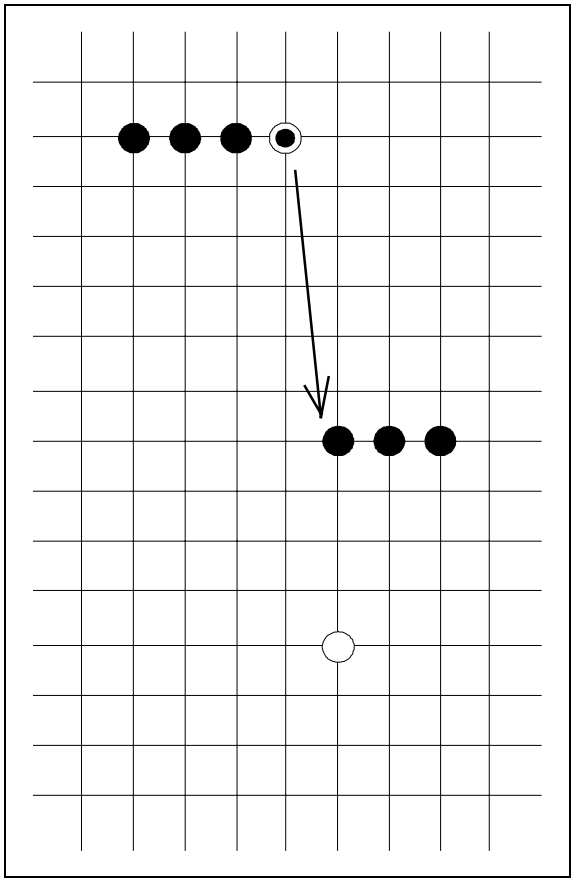

Figura 3.8
prec_y = up_hor[prec_x];
curr_y < up_hor[curr_x].

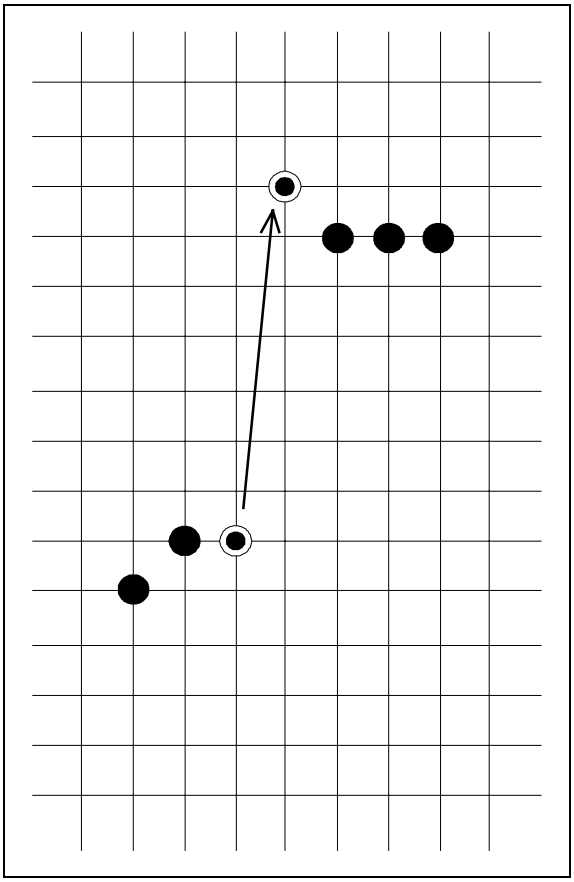

Figura 3.9
prec_y = up_hor[prec_x];
curr_y = up_hor[curr_x].

Această formă simplă a algoritmului este uşor de programat şi produce o afişare corectă a suprafeţei. Are dezavantajul că redesenează ambele orizonturi la fiecare pas, fie că se modifică, fie că nu. Acest lucru ar putea fi eliminat.

Pentru aceasta, ar trebui memorat nu numai prec_x, dar şi prec_y, valoarea precedentă a lui curr_y. După actualizarea orizonturilor, comparăm prec_y cu up_hor[prec_x] şi curr_y cu up_hor[curr_x]. În mod sigur nu trebuie să trasăm o linie dacă prec_y şi curr_y se află sub orizontul superior: prec_y < up_hor[prec_x] şi curr_x < up_hor[curr_x]. În toate celelalte cazuri trebuie efectuată trasarea. În Figurile 3.6..3.9 sunt prezentate cele patru cazuri posibile pentru orizontul superior. Direcţia de trasare este de la stânga la dreapta. Orizontul este indicat prin puncte negre, valorile prec_y şi curr_y - prin puncte albe. De reţinut că orizontul se actualizează *înaintea* testului.

Algoritmul îmbunătăţit este:

```
Fie prec_x şi curr_x, poziţiile orizontale precedentă, respectiv
curentă; fie prec_y şi curr_y valorile precedentă, respectiv curentă
ale funcţiei;

orizontul superior:
     up_hor[curr_x] := max(up_hor[curr_x],curr_y);
```

dacă prec_y = up_hor[prec_x] sau curr_y = up_hor[curr_x], atunci trasează segment între (prec_x,up_hor[prec_x]) şi (curr_x,up_hor[curr_x]);

orizontul inferior:
lo_hor[curr_x] := min(lo_hor[curr_x], curr_y); dacă prec_y = lo_hor[prec_x] sau curr_y = lo_hor[curr_x] atunci trasează segment între (prec_x,lo_hor[prec_x]) şi (curr_x,lo_hor[curr_x]).

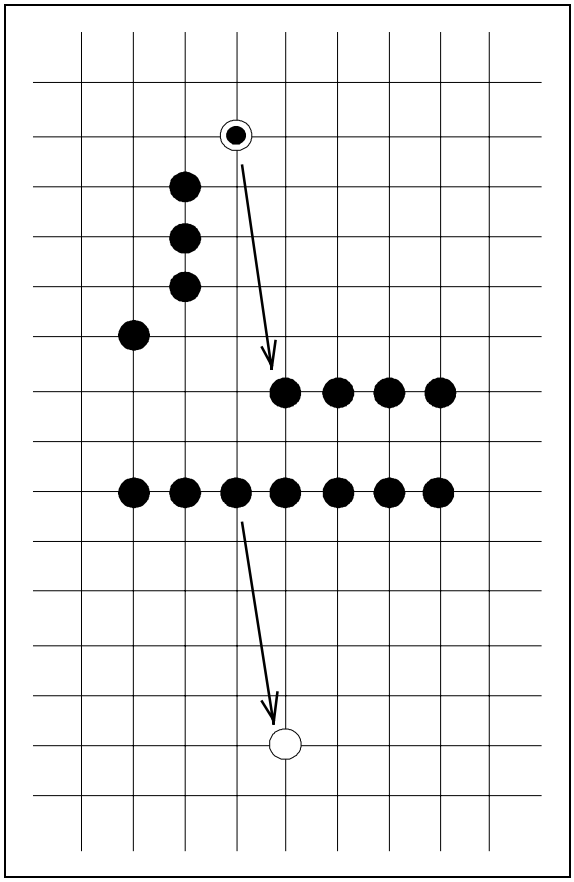

Figura 3.10

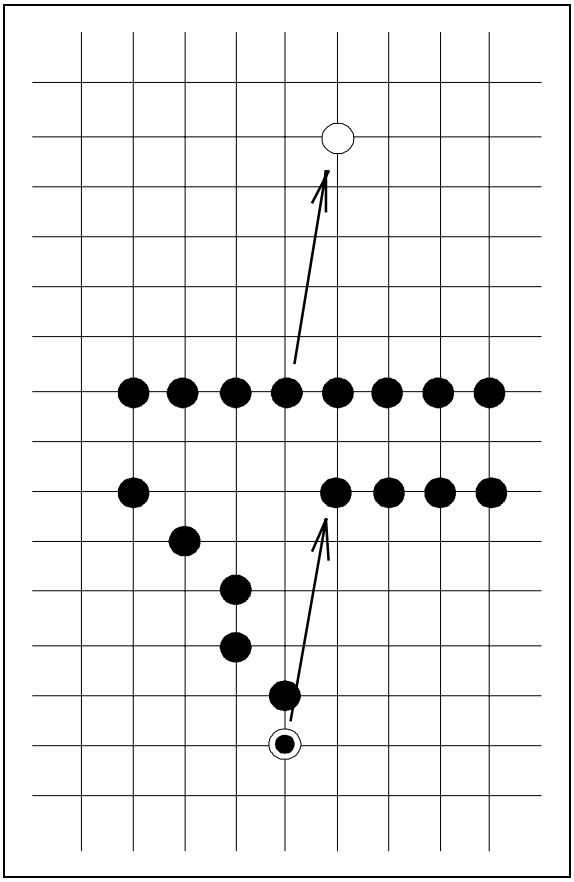

Figura 3.11

Trebuie efectuat testul pentru ambele orizonturi întrucât sunt cazuri în care trebuie trasate două segmente: dacă o curbă este atât de abruptă încât trece de deasupra orizontului superior, până sub orizontul inferior sau viceversa la un singur pas (vezi Figura 3.10, 3.11). Utilizând teste mai laborioase, am putea elimina mai multe situaţii în care nu trebuie efectuată trasarea.

De obicei, suprafaţa este afişată în poziţie înclinată. Aceasta se obţine prin efectuarea de rotaţii în spaţiu asupra fiecărui triplet calculat (x,f(x,y),z) al suprafeţei. Se execută o rotaţie în sens antiorar în jurul lui Oy, urmată de o rotaţie în sens antiorar în jurul lui Ox. Punctul rotit este apoi proiectat ortografic - se ignoră z - şi se utilizează aceste puncte în cadrul algoritmului orizontului flotant. Autorul a testat algoritmul pentru funcţia

$$f(x,z) = 18 \cdot \sin^9(.35*(x^2+z^2))*\exp(-x^2-z^2)+\sin(x+.3)*\cos(z-.3)$$

în domeniul $[-\pi, \pi]\times[-\pi, \pi]$. Funcţia a fost evaluată de-a lungul a 41 de linii z=constant şi pe

fiecare linie pe 380 de valori echidistante pentru x. După fiecare evaluare, tripletul (x,f(x,z),z) a fost rotit în spaţiu, mai întîi antiorar în jurul lui Oy cu 29°, apoi cu 40° antiorar în jurul lui Ox.

Descriere în pseudocod Se presupune că funcţia este definită pe domeniul [0,1]×[0,1].

```
f(x,z) este valoarea funcţiei în (x,z).
num_crv este numărul de curbe de afişat-1.
num_pix este numărul de pixeli pe orizontală.
hight este factorul de scalare pentru înălţimea curbelor pe ecran.
up_hor şi lo_hor sunt tablouri întregi de dimensiune suficientă.

for z:=0 to 1 step 1/num_crv do begin
   for x:=0 to 1 step 1/num_pix do begin
      calculează y=f(x,z);
      roteşte (x,y,z) în jurul lui y,
      apoi în jurul lui x; rezultă (x_r,y_r,z_r);

      prec_x := curr_x;
      prec_y := curr_y;
      curr_x:= x_r*width;
      curr_y:= y_r*hight;
      if up_hor[curr_x] nu a fost iniţializat then
      begin
         up_hor[curr_x] := curr_y;
         lo_hor[curr_x] := curr_y;
      end
      else begin
         up_hor[curr_x] := max(up_hor[curr_x], curr_y);
         lo_hor[curr_x] := min(lo_hor[curr_x], curr_y);
      end;

      { codul de mai jos nu se execută pentru
        primul punct al curbei }
      if x > 0 then begin
         if prec_y = up_hor[prec_x] or
            curr_y = up_hor[curr_x] then begin
             move(prec_x, up_hor[prec_x]);
             draw(curr_x, up_hor[curr_x])
         end;
         if prec_y = lo_hor[prec_x] or
            curr_y = lo_hor[curr_x] then begin
             move(prec_x, lo_hor[prec_x]);
             draw(curr_x, lo_hor[curr_x])
         end
      end
   end { for x }
end { for z };
```

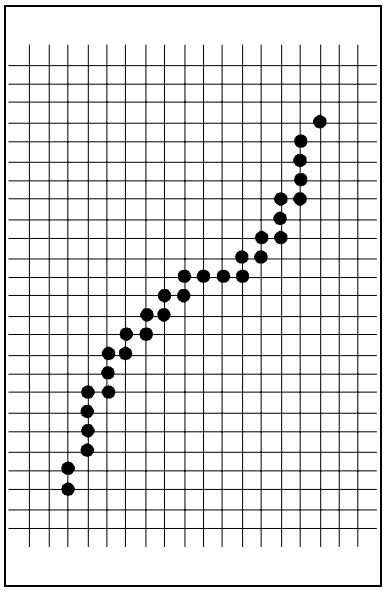

Figura 3.12

O problemă despre care încă nu s-a discutat şi nu a fost rezolvată în codul de mai sus este posibilitatea îngroşării unei linii când funcţia suprafeţei de-a lungul unei curbe este supraîncărcată (aliasing). Vom prezenta un exemplu.

Presupunem că întinderea maximă pe orizontală pe ecran a unei curbe z=constant, width, este de 300 de pixeli. Aceasta este lăţimea înainte ca

suprafaţa să fie rotită în jurul lui y. În acest caz vom calcula valoarea funcţiei în 300 de locaţii echidistante pe x, pentru a avea o valoare pe pixel.

După rotaţia cu 60° în jurul lui Oy, curba se va extinde pe 300*cos(60°)=150 pixeli pe orizontală. Calculând pentru 300 de locaţii pe Ox, ne rezultă două valori pentru un pixel pe orizontală. Atunci când rotaţia este mai mare de 45°, rezultă o linie îngrosată, întrucât se setează de două ori mai mulţi pixeli decât este necesar. De aceea, pentru a avea o singură valoare la un pas pe orizontală trebuie redus numărul de locaţii la width*cos(φ), cu φ = unghiul de rotaţie în jurul lui Oy. Rotaţia în jurul lui Ox nu influenţează aspectul liniei.

2.2 Reprezentarea Completă

Mai sus am afişat funcţia suprafaţă prin trasarea curbelor intersecţii ale lui f(x,z) cu plane z=constant. Dorim să dezvoltăm în continuare algoritmul, ca să se traseze şi curbele intersecţii ale lui f(x,z) cu plane x=constant. Adăugarea celui de-al doilea set de curbe nu este însă o sarcină simplă.

S-ar părea că este suficient să trasăm cele două seturi de curbe una peste alta, dar aceasta nu elimină liniile ascunse. În Figura 3.13 şi Figura 3.14 sunt prezentate două seturi individuale de curbe. Figura 3.15 prezintă suprapunerea lor, iar în Figura 3.16 avem afişarea corectă.

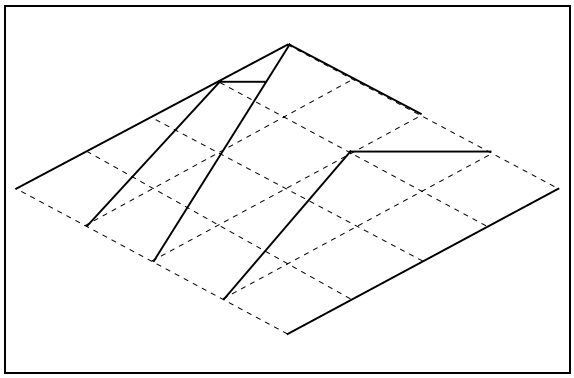

Figura 3.13 Curbe z=constant

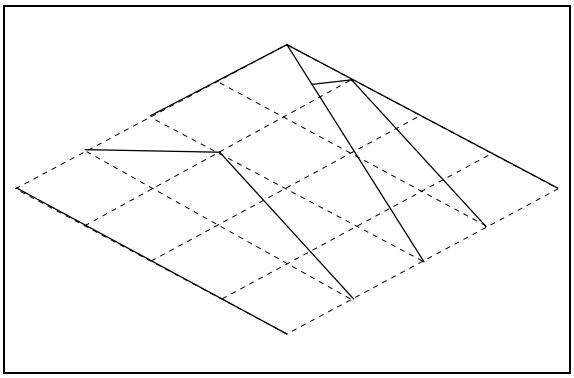

Figura 3.14 Curbe x=constant

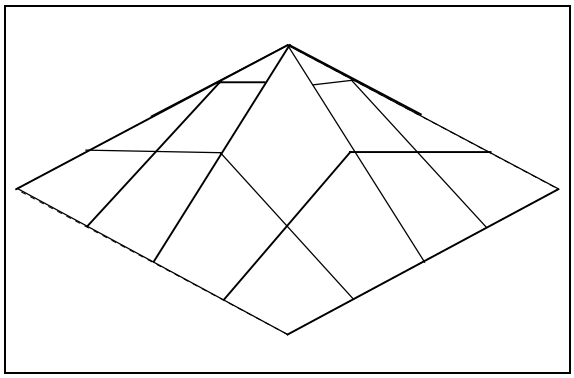

Figura 3.15 Suprapunere

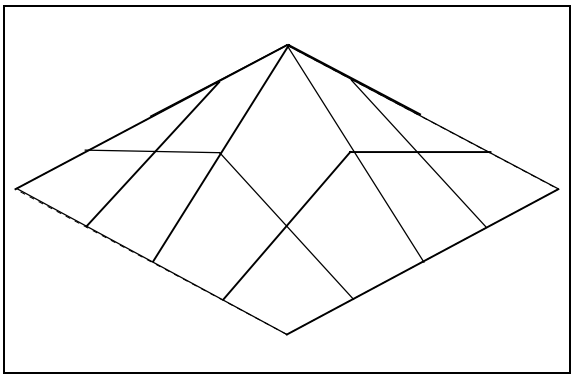

Figura 3.16 Afişarea corectă

Rezultatul corect se obţine trasând cele două seturi de curbe alternativ, utilizând acele orizonturi, inferior şi superior pentru amândouă. Ambele seturi trebuie trasate în ordinea valorii crescătoare a lui z. O modalitate ar fi să trasăm mai întîi o curbă z=constant şi apoi să trasăm

toate porţiunile curbelor x=constant dintre prima şi următoarea curbă z= constant. După aceasta, trasăm următoarea curbă z=constant. În Figura 3.17 este prezentată ordinea şi direcţia în care se desenează curbele şi segmentele de curbă. O altă posibilitate este să trasăm o curbă x=constant, apoi porţiuni de curbe z=constant, ca în Figura 3.18.

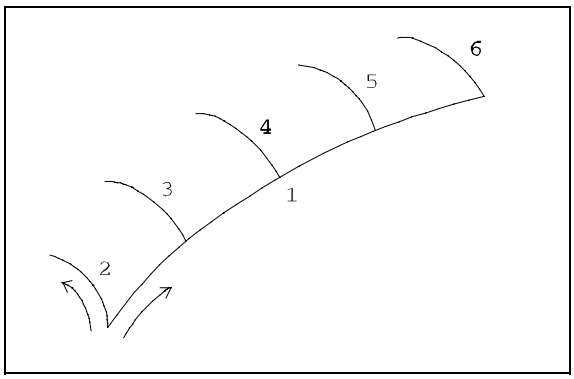

Figura 3.17

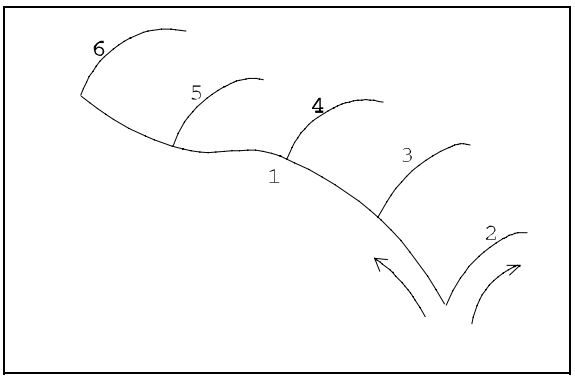

Figura 3.18

Acest procedeu ridică o serie de probleme. Una ar fi supraîncărcarea descrisă mai sus. Acum o vom trata mai în detaliu.

Indiferent de unghiul φ cu care se face rotaţia în jurul lui y, fie curbele z=constant, fie curbele x=constant vor rezulta supraîncărcate dacă nu reducem numărul de eşantioane. Numărul de pixeli pe orizontală va fi width*cos(φ) pentru curbele z=constant şi width*sin(φ) pentru curbele x=constant. Însă nu putem utiliza direct aceste valori ca număr de eşantioane pe direcţiile respective. Pentru un φ foarte mic, de exemplu 2° şi width=300, vom avea doar 11 eşantioane pentru curbele x=constant, iar pentru φ=0 rezultă 0 eşantioane.

Nu vom permite ca acest număr să devină prea mic. Atunci când avem o extindere mică pe orizontală, curba va fi abruptă. Chiar dacă utilizăm un număr mare de eşantioane, curba va avea aspectul corect, deoarece, chiar şi pentru un unghi de peste 45°, supraîncărcarea nu va îngroşa linia.

Pe de altă parte, presupunem din nou că avem un unghi de rotaţie, φ, mic, de exemplu 2°, ca mai sus. Dacă reprezentăm cu câte 25 de curbe pe fiecare direcţie, vom eşantiona suprafaţa în 25 de direcţii echidistante pe z în domeniul de definiţie, când trasăm curbe z=constant. Curbele x=constant trebuie trasate în câte 25 de segmente separate. Aceasta implică faptul că trebuie să eşantionăm funcţia cel puţin în acele valori z care au fost utilizate pentru curbele z=constant. În exemplul nostru vom utiliza 25 de valori z echidistante atunci când trasăm curbele x=constant, câte una pentru fiecare segment de curbă. Deşi aceasta face ca un segment de curbă să arate ca o dreaptă, nu este necesară o eşantionare mai densă, întrucât nu avem nici măcar o jumătate de pixel pe orizontală. Chiar şi cu o eşantionare mai densă, un segment de curbă va avea aspectul unei drepte.

Dacă avem, de exemplu, 35 de pixeli pe orizontală, vom utiliza 50 de locaţii pentru z, ca fiind următorul multiplu de 25. Aceasta va conduce la o supraîncărcare cu 15, dar vom folosi valorile z utilizate pentru curbele z=constant.

Din aceasta rezultă următoarea strategie. Numărul de eşantioane va fi multiplu întreg al numărului de curbe încrucişate şi va fi mai mare sau egal cu numărul de pixeli pe orizontală. Dacă aplicăm această regulă pentru ambele seturi de curbe, în mod sigur vom evalua funcţia de suprafaţă, pentru ambele seturi de curbe, în locaţia unde ele se întâlnesc. Altfel, de exemplu, segmentele x=constant s-ar putea să nu "atingă" punctul prin care trece

următoarea curbă z=constant.

O altă problemă apare atunci când suprafaţa este rotită în jurul lui O_y cu un unghi apropiat sau egal cu 90°. Dacă unghiul de rotaţie este exact 90°, ordinea de trasare din Figura 3.17 va suprima toate curbele z=constant cu excepţia primei curbe. Toate curbele z=constant vor fi linii verticale. Curbele x=constant sunt trasate corect. Segmentele x=constant se întind spre stânga exact până în punctul din care va începe următoarea linie verticală (curba z=constant). Aceste curbe nu vor mai fi trasate.

Pe de altă parte, o astfel de reprezentare nu prea are sens pentru unghiuri de rotaţie de 0° sau 90°, deoarece unul din seturile de curbe vor degenera în linii verticale, care nu furnizează nici o informaţie despre forma curbei.

Atunci când unghiul de rotaţie este apropiat de 0° sau 90°, aspectul curbelor abrupte ar putea lăsa de dorit. Problema se rezolvă simplu schimbând ordinea în care se trasează curbele şi segmentele de curbă. În general, putem spune că ordinea de trasare din Figura 3.17 va afişa corect pentru unghiuri de rotaţie începând cu 0°, dar nu şi aproape de 90°, iar cea din Figura 3.18 - pentru unghiuri de la 90° în jos, dar nu în apropiere de 0°. Un algoritm general de trasare va schimba ordinea de trasare la 45°. Totuşi, trebuie să evităm rotaţia suprafeţei în jurul lui O_y cu unghiuri negative sau peste 90°.

Descriere în Pseudocod Algoritmul este descris în parte în pseudocod, pentru a elimina amănuntele nesemnificative. Toţi termenii şi identificatorii corespund celor din algoritmul precedent. În plus, se calculează numărul de eşantioane conform unghiurilor de rotaţie şi a numărului de curbe de trasat, `stepx` şi `stepz`. Stepx este cel mai mic număr multiplu al numărului de curbe, mai mare decât width*cos(φ), iar stepz este analog pentru width*sin(φ). Rotaţia unui punct calculat, actualizarea orizonturilor şi trasarea unui segment până în acel punct reprezintă o porţiune de cod necesară de două ori în forme aproape identice, astfel încât le vom plasa în procedura `rotate_update_draw`.

```
procedure rotate_update_draw(x, y, z : real);
begin
   roteşte (x,y,z) în jurul lui Oy
   apoi în jurul lui Ox,
   rezultă (xr,yr,zr);

   prec_x := curr_x;
   prec_y := curr_y;
   curr_x := xr*width;
   curr_y := yr*hight;
   if up_hor[curr_x] nu este iniţializat then begin
      up_hor[curr_x] := curr_y;
      lo_hor[curr_x] := curr_y
   end
   else begin
      up_hor[curr_x] := max(up_hor[curr_x], curr_y);
      lo_hor[curr_x] := min(lo_hor[curr_x], curr_y);
   end;
   if x > 0 then begin
      if prec_y = up_hor[prec_x] or
         curr_y = up_hor[curr_x] then begin
          move(prec_x, up_hor[prec_x]);
          draw(curr_x, up_hor[curr_x])
      end;
      if prec_y = lo_hor[prec_x] or
```

```
         curr_y = lo_hor[curr_x] then begin
          move(prec_x, lo_hor[prec_x]);
          draw(curr_x, lo_hor[curr_x])
      end
   end
end {procedure rotate_update_draw};

begin
   stepx := trunc((round(width*cos(angy)) - 0.5) /
                num_crv) * num_crv + num_crv;
   stepz := trunc((round(width*sin(angy)) - 0.5) /
                num_crv) * num_crv + num_crv;

   for z:=0 to 1 step 1/num_crv do begin
   {trasează o curbă z=constant}
      for x:=0 to 1 step 1/stepx do
         rotate_update_draw(x,f(x,z),z);
      if z < 1 then
      {trasează num_crv segmente x=constant}
         for x:=0 to 1 step 1/num_crv do
            for zh:=z to z+1/num_crv step 1/stepz do
               rotate_update_draw(x,f(x,zh),zh)
   end {for z}
end.
```

3 Suprafeţe Definite pe o Grilă

Această secţiune prezintă încă un caz special al unui obiect 3D, o suprafaţă grilă. O *suprafaţă grilă* este o suprafaţă care interpolează valori definite numai în nodurile unei reţele rectangulare. Asemenea suprafeţe sunt utilizate la afişarea unei funcţii de două variabile, de exemplu

$$y = f(x,z)$$

Valoarea funcţiei pentru punctul (x,z) este văzută ca înălţimea deasupra planului (x,z). Mulţimea tuturor valorilor funcţiei dintr-o zonă a planului (x,z) în care funcţia este definită formează o suprafaţă în spaţiu. Pentru afişarea realistă a unei asemenea suprafeţe este necesară o reprezentare tridimensională. Un bun exemplu sunt suprafeţele descrise în secţiunea precedentă, dar modelarea ar presupune calculul unui număr mare de valori şi deci ar fi prea costisitoare şi ar consuma prea mult timp.

Pentru simplificare vom calcula valorile funcţiei numai într-un număr mai mic de puncte care formează o grilă în planul (x,z) şi apoi vom conecta punctele învecinate prin segmente de dreaptă. Va rezulta o suprafaţă grilă. De fapt, interpolăm liniar punctele calculate. Apoi se proiectează această suprafaţă grilă pe un mediu bidimensional (o foaie de hârtie sau ecranul). Dorim să obţinem astfel, cât mai simplu, rapid şi "ieftin", o imagine cât mai realistă.

Pentru a accentua aparenţa de tridimensionalitate, trebuie îndepărtate liniile ascunse. Pe un display raster, am putea să utilizăm algoritmul pictorului, pentru o pseudo-îndepărtare a liniilor ascunse, dar dorim să vedem cum se poate realiza aceasta în acest caz particular,

întrucât proprietăţile geometrice ale suprafeţelor grilă duc la o îndepărtare mai simplă decât în cazul general a liniilor ascunse.

Figura 3.19 prezintă o suprafaţă grilă tipică. Unele faţete sunt parţial sau total ascunse. Nu ne este greu să desenăm de mână o asemenea suprafaţă, însă calculatorul trebuie să determine dacă o faţetă este ascunsă. Pentru aceasta, descompunem suprafaţa grilă. Ea este formată din suma tuturor faţetelor, definite pe o reţea rectangulară care nu este în mod necesar uniform spaţiată. Fiecare faţetă reprezintă partea superioară a unei prisme cu baza un dreptunghi din planul (x,z). Desigur, în cazul general, faţetele suprafeţei grilă pot să se extindă şi dedesubtul planului (x,z), dar vom porni cu cazul mai simplu în care valorile funcţiei f(x,z) sunt toate pozitive. Ne va fi mai uşor de vizualizat şi nu se pierde din generalitate.

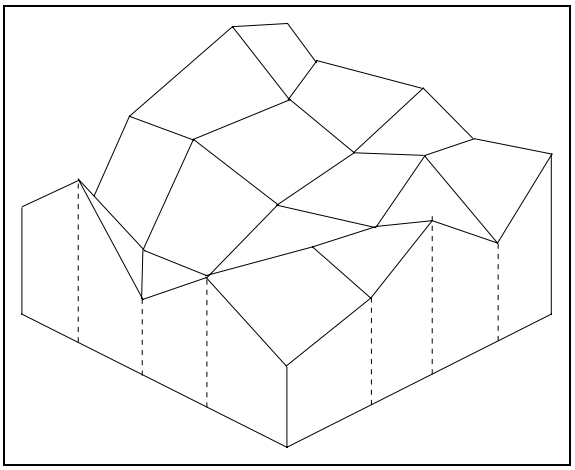

Figura 3.19 O suprafaţă grilă.

Figura 3.20 prezintă două prisme, izolate, de pe marginea dinspre noi a suprafeţei. În general, faţetele nu sunt dreptunghiuri plane, aşa cum se vede în desen. Vom trece în revistă condiţiile în care o faţetă poate să acopere o altă faţetă.

Aceasta depinde de punctul din spaţiu din care este privită suprafaţa. Vom porni cu cazul general în care punctul de observaţie este un punct variabil, deasupra suprafeţei, şi cu coordonatele x şi z în interiorul zonei dreptunghiulare de definiţie.

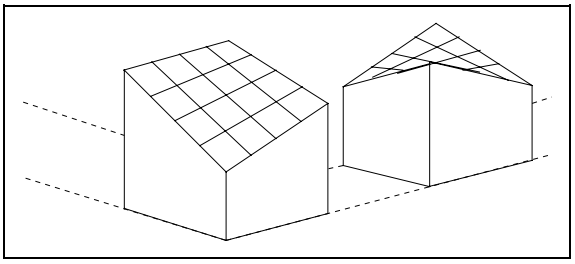

Figura 3.20 Două prisme ale suprafeţei grilă.

În Figura 3.21 este prezentat ce vedem când privim cele două prisme dintr-un punct situat exact deasupra unui punct dintre cele două prisme. Punctele marchează punctele grilei din planul (x,z), iar liniile ascunse sunt reprezentate prin linii întrerupte. Dacă prelungim muchiile verticale ale prismelor, ele converg într-un singur punct, *punctul de fugă*, VZ. De reţinut însă că faţetele suprioare ale prismelor pot fi înclinate şi non-planare.

Înaite de construirea suprafeţei grilă, trebuie să o definim precis în spaţiul 3D. Presupunem că $x_0,...,x_n$ şi $z_0,...,z_m$ sunt şiruri strict crescătoare de numere reale, nu neapărat la distanţe egale. Vom nota cu i indicii pentru x şi cu j indicii pentru z. Punctele din planul (x,z) cu coordonatele $(x_i,0,z_j)$ se numesc *punctele grilei*. Planele verticale faţă de planul (x,z), care îl intersectează după dreptele $x=x_i$ sau $z=z_j$ se numesc *planele grilei*. Dreptunghiul cu laturile $x=x_i$, $x=_{i+1}$, $z=z_j$, $z=z_{j+1}$ este *elementul (i,j) al grilei*. Avem n*m elemente de grilă.

În general, funcţia de afişat este definită în toate punctele planului (x,z). Valoarea funcţiei în punctul (x_i,z_j) este $y_{i,j}$. Pentru afişarea funcţiei, determinăm valorile $y_{i,j}$, le conectăm prin segmente de dreaptă pe cele corespunzătoare la două puncte ale grilei, învecinate pe orizontală, şi similar pentru punctele grilei pe verticală. În acest fel ne rezultă o grilă în spaţiul 3D. Ea este o aproximare a funcţiei în punctele grilei. De exemplu, dacă $y_{i,j}$, $y_{i,j+1}$, $y_{i+1,j}$ şi $y_{i+1,j+1}$ sunt valorile funcţiei în cele patru puncte ale grilei care definesc elementul (i,j),

atunci

$$\alpha\beta y_{i,j}+(1-\alpha)\beta y_{i,j+1}+\alpha(1-\beta)y_{i+1,j}+(1-\alpha)(1-\beta)y_{i+1,j+1},\ 0\le(\alpha,\beta)\le 1$$

este valoarea suprafeţei grilă în orice punct din element (*faţeta (i,j)*).

Vom proiecta laturile faţetei şi vom presupune că aceasta este chiar proiecţia faţetei. Acest lucru nu este întotdeauna corect, întrucât există situaţii în care proiecţia unei faţete nu este mărginită de proiecţiile laturilor sale. Facem însă această presupunere, pentru a simplifica algoritmul. În fond, faţetele sunt doar o aproximare a valorii reale a funcţiei. Vom obţine însă o imagine suficient de bună a suprafeţei grilă.

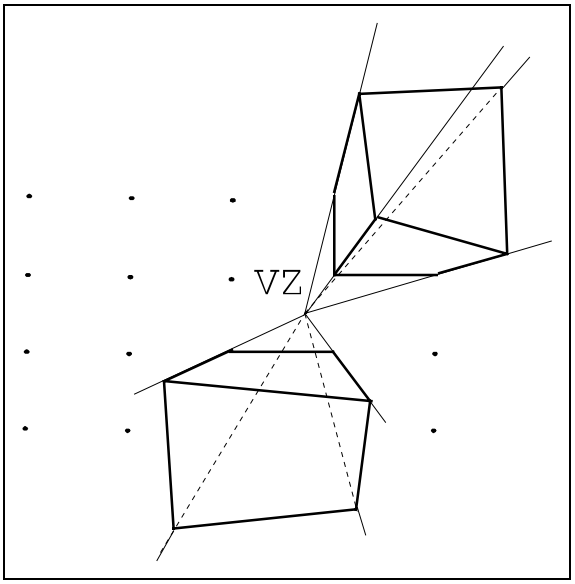

Figura 3.21 Vedere de sus a celor două prisme.

În vederea unei afişări realiste, trebuie să efectuăm o proiecţie perspectivă. Punctul de observaţie are coordonatele (PO_x,PO_y,PO_z), iar direcţia de observaţie este definita de vectorul (DO_x,DO_y,DO_z). Planul de vedere este întotdeauna normal la direcţia de observaţie. Dacă $x_0 \le PO_x \le x_n$ şi $z_0 \le PO_z \le z_m$, spunem că privim suprafaţa grilă *din faţă* (de sus). Imaginea pe care o vedem corespunde celei din Figura 3.22. Dacă numai una dintre aceste condiţii este îndeplinită, vedem suprafaţa grilă *din lateral*; dacă nici una dintre condiţii nu este satisfăcută, atunci o privim *din colţ*, ca în Figura 3.23. În toate aceste cazuri presupunem că $PO_y>0$, adică punctul de observaţie se află deasupra planului (x,z). (Pentru generalitate, trebuie sa putem avea şi $PO_y<0$). Este esenţial ca $PO_y \neq 0$.

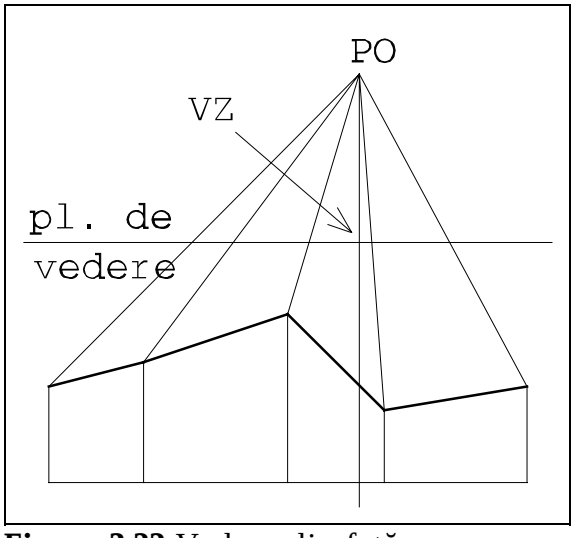

Figura 3.22 Vedere din faţă.

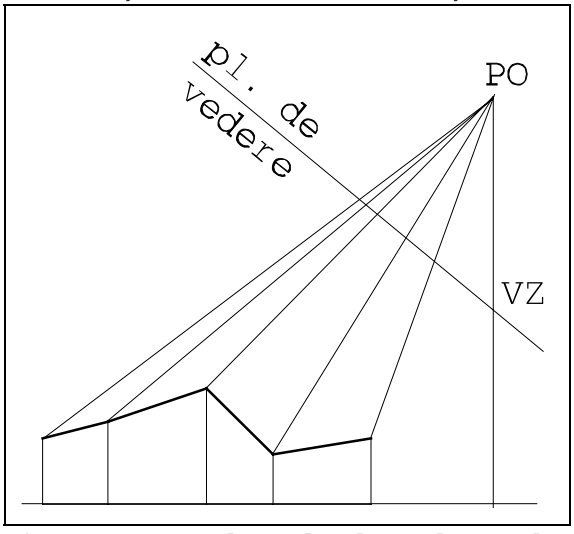

Figura 3.23 Vedere din lateral sau din colţ.

A privi dintr-un asemenea punct înseamnă a proiecta suprafaţa grilă pe planul de vedere (normal la direcţia de observaţie). Această proiecţie o transformă într-o imagine 2D. În planul de vedere, vom considera coordonatele x înspre dreapta, iar coordonatele y în sus.

În Figura 3.22 şi în Figura 3.23, planul de vedere este ecranul. Ochiul observatorului se află în PO - centrul proiecţiei.

La vederea din faţă, datorită proiecţiei perspectivă, proiecţiile muchiilor verticale ale prismelor converg toate într-un singur punct din planul de vedere, VZ, care este punctul în care o dreaptă paralelă cu axa y, dusă prin punctul de observaţie, intersectează planul de vedere, ca în Figura 3.22.

Şi la vederea din lateral sau din colţ, din nou toate proiecţiile muchiilor verticale converg într-un singur punct, având aceeaşi poziţie ca mai sus, dar în acest caz este situat înafara proiecţiei suprafeţei grilă, ca în Figura 3.23.

Ordinea de Acoperire Când trasăm proiecţia suprafeţei grilă, vom trasa numai laturile faţetelor, şi numai acelea care nu sunt acoperite de către o altă faţetă. Pentru aceasta, vom desena faţetele pe rând, într-o *ordine de acoperire*, adică, o faţetă care poate fi acoperită de altă faţetă este desenată numai după acea faţetă. O proprietate importantă a suprafeţelor grilă este aceea că, oricare ar fi modul în care sunt privite, întotdeauna există o singură ordine de acoperire în care se pot aranja faţetele. Aceasta este o consecinţă a modului regulat în care sunt amplasate elementele grilei.

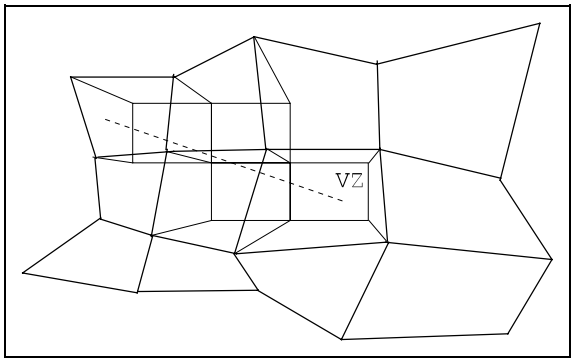

Figura 3.24 Raza de vedere, vedere de sus.

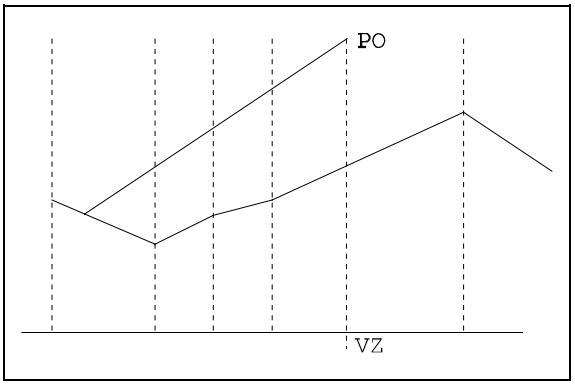

Figura 3.25 Raza de vedere, vedere în secţiune.

Pentru a explica modul în care se determină ordinea de acoperire, să ne întoarcem la suprafaţa grilă originală din spaţiul 3D şi să ne imaginăm că ne aflăm în punctul de observaţie, deasupra ei, şi că privim către un punct al unei faţete. Raza după care privim este indicată în Figura 3.24. Acum să ne imaginăm că prismele se prelungesc în sus la infinit, astfel încât această rază trebuie să traverseze prismele situate între noi şi punctul către care privim. În cazul nostru, ea traversează trei prisme înainte să ajungă în punctul de destinaţie. Acest lucru se vede mai bine în Figura 3.25.

Planul secţiunii din Figura 3.25 este vertical faţă de planul (x,z) şi conţine raza de vedere. Acesta intersectează planele grilei şi, probabil, faţete. Liniile verticale din figură sunt extensii ale planelor grilei. Ele ne arată locul în care raza de vedere traversează o prismă. Oricare dintre prismele întâlnite de raza de vedere ar putea acoperi vederii punctul către care privim, dacă acea faţetă este suficient de înaltă.

În imaginea proiectată, punctul de fugă VZ joacă rolul punctului de observaţie, iar o rază de vedere este orice rază care porneşte din VZ. Dacă trasăm o rază din VZ către oricare dintre faţetele din imaginea proiectată, această rază va traversa proiecţiile câtorva baze de prisme înainte să ajungă la faţeta destinaţie. Aceasta înseamnă că, în spaţiul 3D, această rază va traversa prismele corespunzătoare înainte să ajungă la prisma destinaţie. Într-o ordine de acoperire, faţetele acestor prisme trebuie să fie înaintea faţetei prismei destinaţie. Problema care se pune este dacă este sau nu posibil să aranjăm toate faţetele suprafeţei grilă într-o astfel de ordine încât prismele traversate de orice rază pornind din punctul de fugă să fie în ordinea corectă.

Desigur, această ordine va depinde de poziţia punctului VZ. Însă oricare ar fi VZ, un asemenea aranjament este uşor de găsit pentru suprafeţe grilă şi acest lucru stă la baza acestui algoritm de îndepărtare a liniilor ascunse. Pentru anumite poziţii ale lui VZ, sunt posibile mai multe ordonări de acoperire.

Pentru dezvoltarea algoritmului, setăm:

$$x_{-1} = z_{-1} = -\infty,$$
$$x_{n+1} = z_{m+1} = \infty$$

Fie K şi L astfel încât $x_{K-1} \le VZ_x \le x_K$ şi $z_{L-1} \le VZ_z \le z_L$, şi mai definim:

```
K1 = max(K,1)     KN = min(K-1,n)
L1 = max(L,1)     LM = min(L-1,m)
```

Atunci o ordine de acoperire este generată de următoarele cicluri (în Pascal):

```
for i := K1 to n do
     for j := L1 to m do
          faţetă(i, j);
for i := KN downto 1 do
     for j := L1 to m do
          faţetă(i, j);
for i := K1 to n do
     for j := LM downto 1 do
          faţetă(i, j);
for i := KN downto 1 do
     for j := LM downto 1 do
          faţetă(i, j);
```

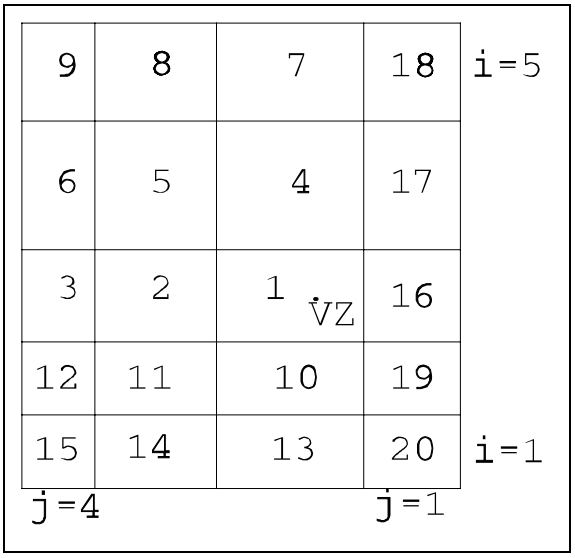

Figura 3.26

Pentru anumite poziţii ale lui VZ, unele dintre ciclurile de mai sus sunt vide. Figurile care urmează indică poziţia lui VZ şi ordinea de acoperire prin enumerarea elementelor de grilă proiectate. Proiectarea grilei pe planul de vedere ar putea s-o rotească în orice direcţie, de aceea se indică şi domeniile pentru i şi j.

În Figura 3.26 se vede ordonarea faţetelor într-o vedere din faţă. Formulele dau:

```
K  = 3      L  = 2
K1 = 3      KN = 2
L1 = 2      LM = 1
```

Nu avem cicluri vide.

În Figura 3.27 avem ordonarea faţetelor pentru o vedere din lateral, pentru care:

```
K  = 0      L  = 3
K1 = 1      KN =-1
L1 = 3      LM = 2
```

Două cicluri sunt vide.

În Figura 3.28 avem ordinea faţetelor pentru o vedere din colţ:

```
K  = 6      L  = 0
K1 = 6      KN = 5
L1 = 1      LM =-1
```

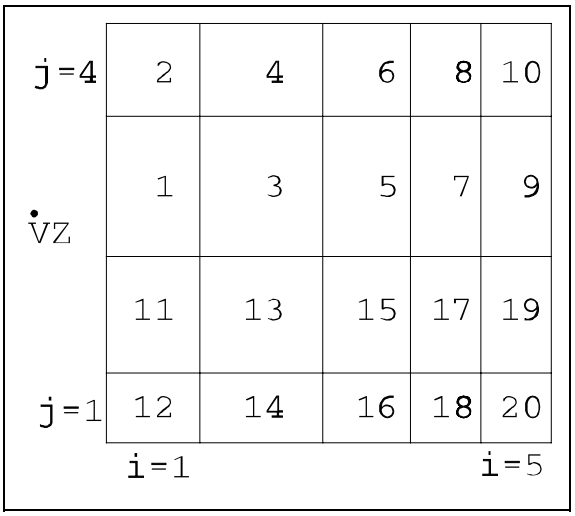

Figura 3.27

Trei cicluri sunt vide.

Ordinea de acoperire poate fi caracterizată ca o secvenţă ordonată a faţetelor de la faţetă$_1$ la faţetă$_{n*m}$, cu proprietatea că dacă faţetă$_i$ poate să acopere faţetă$_j$, atunci i<j. Dacă proiecţiile faţetelor sunt trasate într-o asemenea ordine, în momentul când se desenează o faţetă, toate faţetele care o pot acoperi au fost deja desenate. Vom trasa doar acea parte a unei faţete care nu este acoperită de către faţetele trasate până în acel moment.

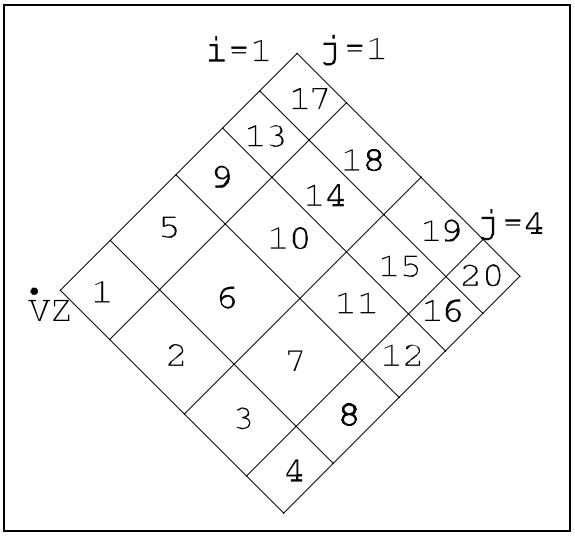

Figura 3.28

Perimetrul Dorim să trasăm proiecţiile perspectivă ale faţetelor pe planul de vedere. În continuare, vom numi aceste proiecţii "faţete". Algoritmul decurge după cum urmează. Atunci când trasăm faţetele în ordinea de acoperire, faţetele deja trasate formează un poligon care are un perimetru exterior, pe care îl vom numi în continuare pur şi simplu "perimetru". La trasarea unei noi faţete, perimetrul trebuie actualizat pentru a include noua faţetă. Importanţa perimetrului constă în aceea că atunci când dorim să trasăm un segment care reprezintă muchia unei faţete, un vom trasa acea parte din linie situată în interiorul perimetrului (acea parte din faţetă este ascunsă). Este evidentă diferenţa faţă de utilizarea orizonturilor inferior şi superior din secţiunea precedentă.

Iniţializarea perimetrului depinde de punctul de observaţie.

a. Dacă suprafaţa grilă este privită din faţă, perimetrul se iniţializează cu faţeta care conţine VZ, după cum se vede în Figura 3.29 (liniile subţiri indică porţiuni din perimetru care încă nu au fost trasate).
b. La vederea din lateral, perimetrul se iniţializează cu suma tuturor laturilor dinspre

observator ale faţetelor mai apropiate (vezi Figura 3.30).

c. La vederea din colţ, perimetrul se iniţializează cu laturile dinspre observator ale faţetelor frontale pe direcţiile x şi z.

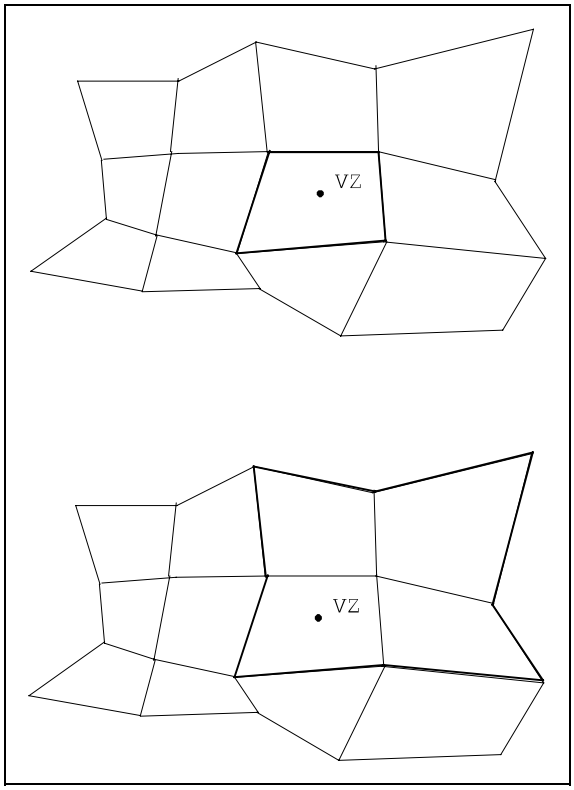

Figura 3.29 Generarea perimetrului la vederea din faţă.

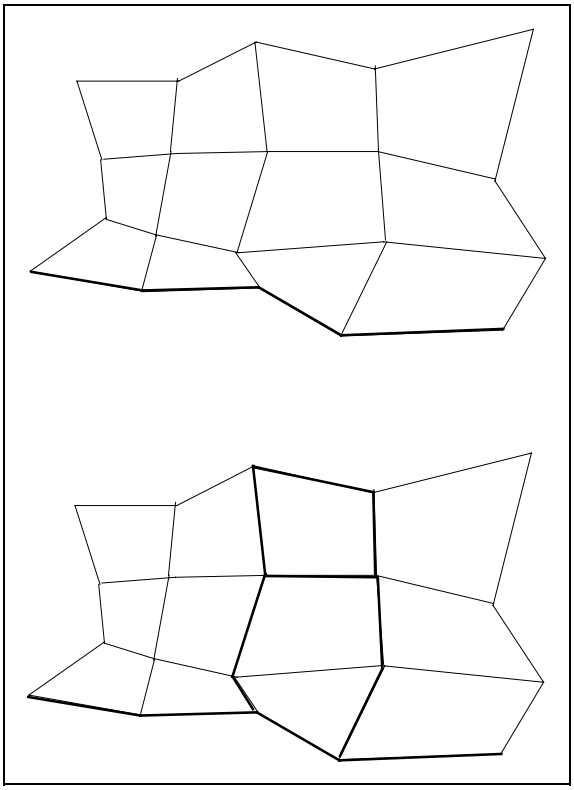

Figura 3.30 Generarea perimetrului la vederea din lateral.

Dacă $PO_y \neq 0$, după cum am presupus, imaginile trasate vor avea două proprietăţi importante: ele sunt întotdeauna coerente şi ele sunt *stea-convexe* în jurul lui VZ. Ultima proprietate înseamnă că dacă două puncte ale imaginii sunt situate pe o rază din VZ, atunci toate punctele de pe rază dintre cele două puncte aparţin de asemenea imaginii. Convexitatea în stea este o consecinţă directă a ordinii de acoperire în care se face trasarea.

Există însă o altă poziţie posibilă a punctului de observaţie, o versiune a vederii din lateral sau din colţ, numită *vedere orizontală*. Vederea este orizontală atunci când PO_x sau PO_z sau amândouă sunt înafara grilei, iar $PO_y=0$ iar direcţia de observaţie DO este în planul (x,z). Direcţia "în sus" a imaginii în planul de vedere este identică cu coordonata y din sistemul de coordonate al suprafeţei grilă. În acest caz nu există VZ (este la distanţă infinită) şi imaginea nu este stea-convexă. Noţiunea de "stea-convexitate" trebuie înlocuită cu ceva care s-ar putea numi "paralel-convexitate" faţă de verticală, adică dacă două puncte ale imaginii sunt pe o linie verticală, atunci toate punctele de pe linie situate între ele aparţin de asemenea imaginii. Ordinea de acoperire este fie cea pentru vederea din lateral, fie cea pentru vederea din colţ, în funcţie de poziţia punctului PO.

Forma perimetrului depinde de modul în care este privită suprafaţa grilă. În figurile care urmează se prezintă perimetrul în două cazuri diferite. Un vertex al perimetrului este determinat în mod unic de unghiul razei din VZ la acel vertex şi de distanţa faţă de VZ (datorită stea-convexităţii). Dacă unghiurile vertex-urilor îşi schimbă direcţia la parcurgerea

vârfurilor consecutive ale perimetrului, atunci aceste vârfuri descriu un contur care nu este stea-convex. Aceasta se numeşte *monotonia unghiurilor* şi se păstrează şi după actualizarea perimetrului. Ea are consecinţe importante asupra timpului de execuţie al algoritmului.

În cazul vederii din faţă perimetrul arată ca în Figura 3.31. Punctul de fugă VZ este în interiorul imaginii. Dacă se parcurge perimetrul în sens antiorar, atunci unghiurile vertex-urilor vor fi monoton crescătoare. Perimetrul se numeşte exterior (nu există perimetru interior).

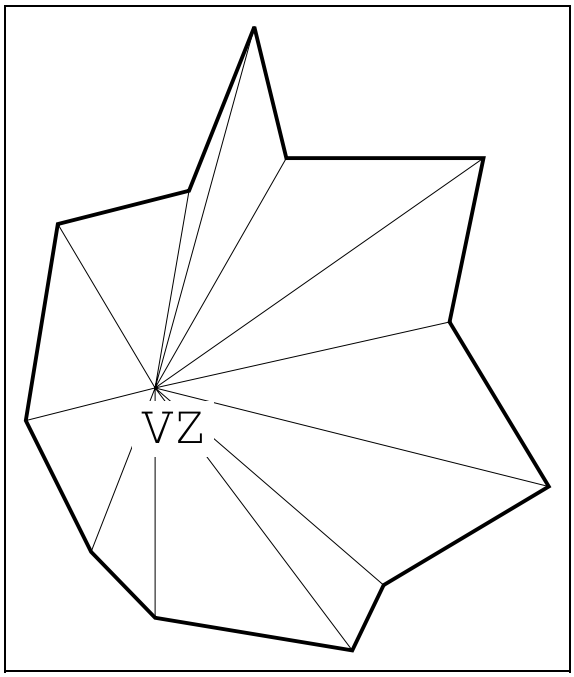

Figura 3.31 Perimetrul din faţă.

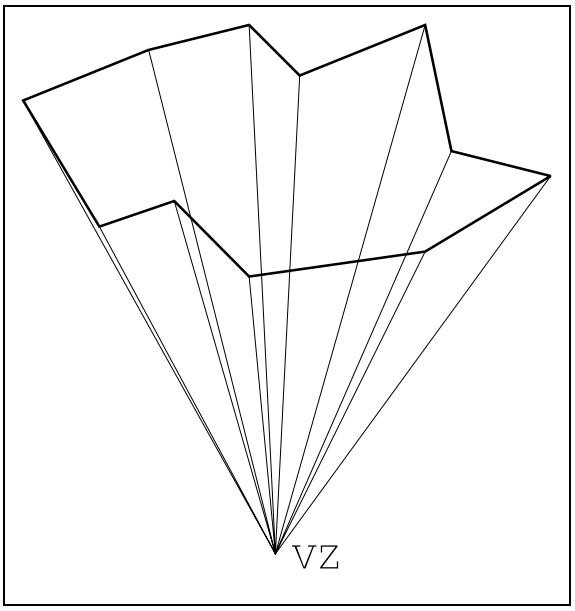

Figura 3.32 Perimetru din lateral.

În cazul vederii din lateral, perimetrul arată ca în Figura 3.32. Punctul VZ este înafara imaginii. Laturile mai apropiate de VZ formează perimetrul interior, iar celelalte - perimetrul exterior. La parcurgerea în sens antiorar a perimetrului, unghiurile corespunzătoare sunt monoton crescătoare pe perimetrul exterior şi monoton descrescătoare pentru perimetrul interior. Similar în cazul vederii din colţ.

Dacă o suprafaţă grilă este privită orizontal, proiecţiile muchiilor verticale ale prismelor sunt linii verticale (vezi Figura 3.33). Această vedere este asemănătoare cu vederea din lateral sau din colţ întrucât unghiurile în jurul lui VZ din aceste vederi au o valoare minimă şi una maximă. Vom înlocui "unghiul în jurul lui VZ" cu "stânga-dreapta" în imaginea vederii orizontale. În acest fel, algoritmii pentru procesarea vederii din lateral sau din colţ pot să trateze şi acest caz.

În cele ce urmează, "faţetă" înseamnă proiecţia unei faţete pe planul de vedere. La actualizarea perimetrului cu o faţetă, se determină unghiurile vertex-urilor faţetei din extremităţile stânga şi dreapta (în sens antiorar în jurul lui VZ); le vom numi fmin şi fmax. "Vârfurile perimetrului circumscris" sunt vertex-urile Pmin, cu Pmin≤fmin şi Pmax, cu Pmax≥fmax. Toate laturile perimetrului care pot intersecta laturile faţetei se pot determina parcurgând perimetrul în sens antiorar de la Pmin la Pmax, ca o consecinţă a monotoniei unghiurilor. În lipsa acestei proprietăţi, căutarea laturilor perimetrului care ar putea intersecta laturile faţetei ar fi necesitat parcurgerea întregului perimetru, ceea ce ar fi dus la un timp de

execuţie al algoritmului cel puţin de ordinul N_2.

Înainte de căutarea vârfurilor perimetrului se poate efectua o preprocesare a faţetei. Aceasta reduce numărul de intersecţii posibile. Preprocesarea constă în a verifica dacă o faţetă nu este cumva acoperită de acele faţete care sunt în imediata vecinătate a ei şi sunt mai apropiate de punctul de observaţie. Se poate întâmpla ca faţeta să se acopere pe ea însăşi, dacă este atât de înclinată faţă de punctul de observaţie încât să nu poată fi văzută suprafaţa ei superioară. În acest caz nu mai este necesar testul faţă de conturul perimetrului. Dacă numai o parte a faţetei se acoperă pe ea însăşi, atunci unel dintre laturile sale nu mai trebuie comparate cu conturul perimetrului. De remarcat însă că algoritmul de actualizare a perimetrului funcţionează şi fără această preprocesare.

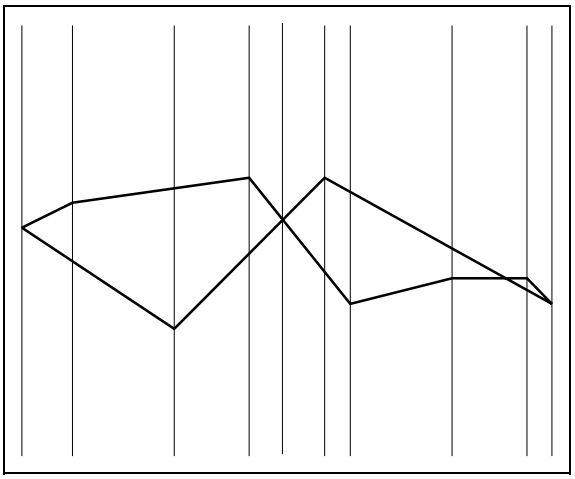

Figura 3.33 Vedere orizontală, perimetru din lateral.

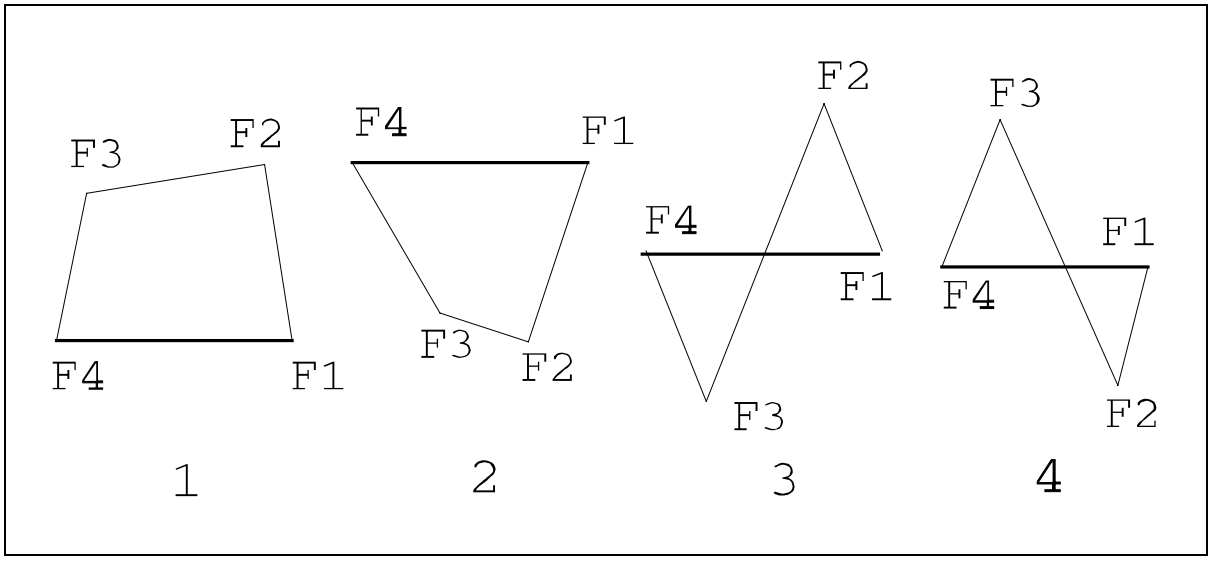

Figura 3.34 Cele patru poziţii diferite în cazul vederii din lateral.

Preprocesarea Faţetei Dacă o faţetă este văzută din lateral atunci vârfurile sale pot fi aşezate în patru moduri diferite. Vârfurile sunt numerotate în ordinea creşterii unghiurilor (vezi Figura 3.34). Latura îngroşată dintre F_1 şi F_4 este deja prezentă întrucât este latură a unei faţete care precede faţeta curentă în ordinea de acoperire. Prin calculul coordonatei z a produselor vectoriale $(F_4-F_1)x(F_2-F_1)$ şi $(F_4-F_1)x(F_3-F_1)$ se disting mai multe cazuri. Dacă F_2 se află la dreapta vectorului F_1F_4, atunci $(F_4-$

Tabela I

caz	1	2	3	4
$(F_4-F_1)x(F_2-F_1)_z$	-	+	-	+
$(F_4-F_1)x(F_3-F_1)_z$	-	+	+	-

F_1)x(F_2-F_1)$_z$ este negativ. La fel pentru poziţia lui F_3. Rezultă cele patru cazuri din Tabela I.

Latura deja prezentă este fie parte fie în interiorul perimetrului, dar celelalte trei laturi pot ieşi din perimetru. Vârfurile acestor laturi se văd în Tabela II (s-au precizat numai indicii lui F).

Tabela II

caz	1	2	3	4
perimetrul exterior	1234		123	234
perimetrul interior		1234	234	123

Dacă faţeta este văzută din colţ, atunci există 10 aranjamente diferite ale vârfurilor sale. Vârfurile sunt numerotate astfel încâ F_1 are unghiul cel mai mic iar F_4 are unghiul cel mai mare (vezi Figura 3.35).

Laturile îngroşate dintre F_1, F_2 şi F_4 sunt deja prezente întrucât ele sunt laturi ale celor două faţete adiacente care o preced pe cea curentă în ordinea de acoperire.

Tabela III

caz	1	2	3	4	5	6	7	8	9	10
(F_4-F_1)x(F_2-F_1)$_z$	-	-	-	-	-	+	+	+	+	+
(F_2-F_1)x(F_3-F_1)$_z$	+	-	+	-	+	-	+	-	+	-
(F_4-F_2)x(F_3-F_2)$_z$	+	-	+	+	-	-	+	-	-	+

În Tabela III se văd cele zece cazuri. Efectuând numai trei calcule, cazul 1 nu se poate distinge de cazul 3 şi nici 6 de 8. Însă aceasta nu are importanţă, deoarece cazul 1 este tratat identic cu cazul 3 şi 6 cu 8. Laturile deja procesate sunt fie parte a perimetrului, fie în interiorul său, dar celelalte două laturi pot să iasă din perimetru, după cum se vede în Tabela IV.

Tabela IV

caz	1&3	2	4	5	6&8	7	9	10
perimetrul exterior		134	134	134	134		34	13
perimetrul interior	134		34	13		134	134	134

Pseudounghiuri La căutarea celor două vârfuri ale perimetrului care conţin faţeta curentă, trebuie să comparăm unghiurile formate de vârfurile perimetrului şi ale faţetei în jurul lui VZ. Dacă am lucra cu unghiuri reale, timpul de calcul ar fi foarte mare, dar putem să evităm aceasta dacă nu utilizăm unghiul şi distanţa până la VZ ca definiţie a vertex-ului, ci utilizând coordonatele carteziene. În locul valorii reale a unghiului vom utiliza un

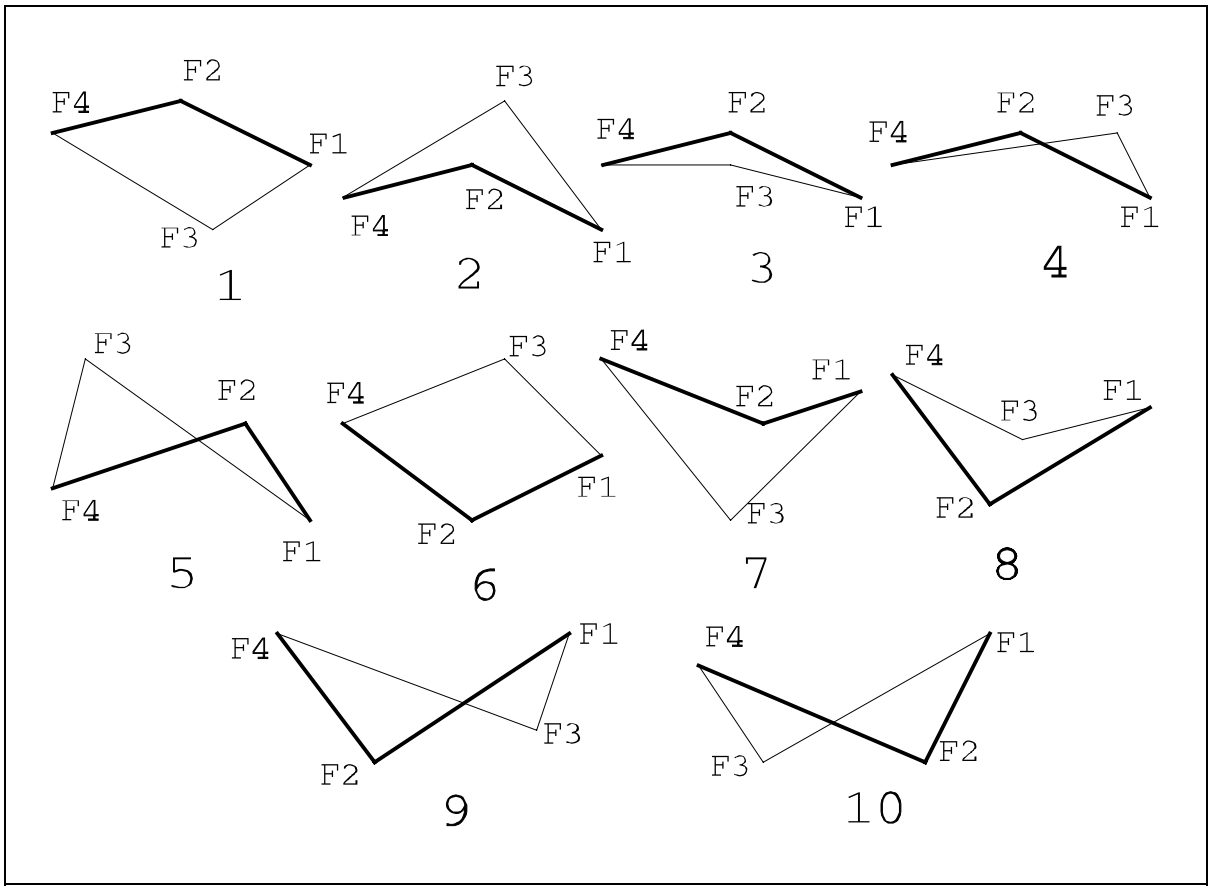

Figura 3.35 Cele zece poziţii diferite la vederea din colţ.

pseudounghi, care să păstreze numai ordinea vârfurilor în jurul lui VZ. Structura de date pentru perimetru este o listă înlănţuită a vârfurilor perimetrului. La parcurgerea acestei liste în una dintre direcţii, pseudounghiurile vor (des)creşte monoton şi le vom utiliza doar la determinarea vârfurilor perimetrului care conţin faţeta. Fiecare element al listei perimetrului va conţine coordonatele carteziene şi pseudounghiul vertex-ului, după cum este definit în Figura 3.36.

La 45° valoarea pseudounghiului este 1; la 135° este 3; la 225° este 5, iar la 315° este -1.

Codificarea Algoritmului În majoritatea aplicaţiilor practice (de exemplu harta unui teren) punctele grilei sunt uniform distribuite pe direcţiile x şi z. Asemenea grile necesită spaţiul minim de memorie: un tablou g bidimensional, pentru întreaga suprafaţă grilă. Coordonata x a grilei corespunde indicelui i al unui element al tabloului, iar coordonata z corespunde indicelui j. Înălţimea în punctul (i,j) este chiar valoarea elementului g[i,j].

Poziţia lui PO faţă de suprafaţa grilă determină ordinea de acoperire şi tipul perimetrului care va fi utilizat (din faţă, din lateral sau din colţ). Trebuie să efectuăm o transformare a planului de vedere şi o proiecţie perspectivă pentru fiecare element al grilei.

Pentru a obţine o bună transformare a planului de vedere, trebuie să aplicăm o serie de regului. Plasăm punctul de referinţă R aproximativ în mijlocul suprafeţei grilă. Ca urmare, normala planului de vedere este descrisă de vectorul de la PO la R. Fie 2d distanţa dintre PO

şi R. Plasăm planul de vedere (ecranul) pe care efectuăm proiecţia cam la distanţa d de R, adică la jumătatea distanţei dintre cele două puncte. În acest fel rezultă o perspectivă bună.

După transformarea planului de vedere, efectuăm o proiecţie perspectivă pe planul de vedere, cu PO centrul proiecţiei. PO are acum coordonatele (0,0,-d).

Nu este nevoie să efectuăm cele două transformări pentru *toate* faţetele, înainte de a începe trasarea. În acest fel, necesarul de memorie ar fi cam dublul tabloului original care memorează datele suprafeţei grilă. De fapt, trebuie să efectuăm aceste transformări numai asupra vârfurilor faţetei care se procesează. Totuşi, în funcţie de ordinea în care parcurgem faţetele, dacă am memora o linie sau o coloană de vertex-uri transformate, am evita repetarea de patru ori a aceleiaşi transformări asupra unui vertex. Acest lucru nu este prezentat în cod.

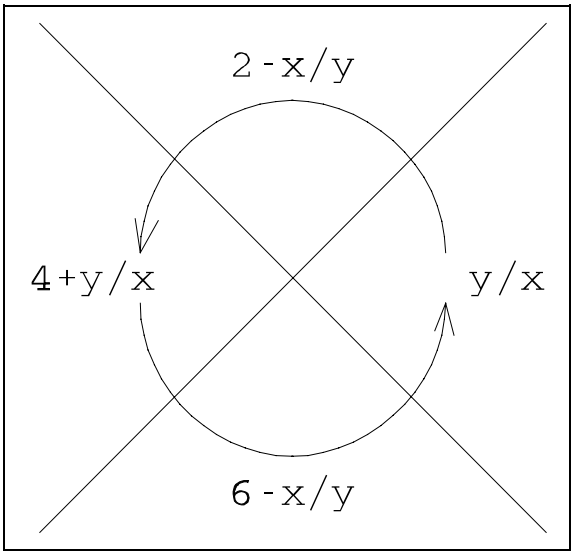

Figura 3.36 Pseudounghiuri.

Secvenţa de cod de mai jos actualizează perimetrul cu faţeta(i,j) în cazul unei vederi din faţă a suprafeţei grilă, cu faţeta privită din lateral, i=k sau j=l (vezi Figura 3.26). Se disting patru secvenţe diferite de parcurgere a vârfurilor faţetei, în funcţie de poziţia faţetei (i,j) faţă de faţeta (k,l). În cazul considerat aici, o parcurgere în sens antiorar a faţetei (i,j) este $F_{i+1,j}$, $F_{i+1,j+1}$, $F_{i,j+1}$ şi $F_{i,j}$; în cazul i>k şi j=l, această secvenţă este $F_{i,j}$, $F_{i+1,j}$, $F_{i+1,j+1}$ şi $F_{i,j+1}$, şi aşa mai departe. Pentru simplificare, nu se face o preprocesare a faţetei, deşi acest lucru ar reduce mult din numărul de vârfuri de parcurs. Nu vom încerca să tratăm orice caz posibil, ci dorim doar să ilustrăm ideea care stă la baza algoritmului.

Determinarea indicilor pentru ordinea de parcurgere arată oarecum bizar în cod, dar se bazează pe regulile de modificare a indicilor la parcurgerea în sens antiorar a celor patru vârfuri (vezi Figura 3.37).

Secvenţa de cod nu realizează şi afişarea suprafeţei grilă. Facem o serie de presupuneri:

- proiecţia perspectivă a suprafeţei grilă a fost efectuată;
- proiecţia punctului de pe suprafaţă $(x_i,y_j,z_{i,j})$ are coordonatele fx(i,j) şi fy(i,j) în planul de vedere;
- coor**do**natele lui VZ în planul de vedere sunt (vzx,vzy);
- perimetrul este o listă circulară înlănţuită.

```
type vertex = record x, y, a : real;
                     p       : ^vertex
              end;

var dpx, dpy, dfx, dfy, ix, iy,
    cp1, cp2, cp3, cp4           : real;
    firstp, faţeta, lastf        : ^vertex;
    p1, p2, f1, f2               : ^vertex;

procedure intersect(var ix, iy : real);
```

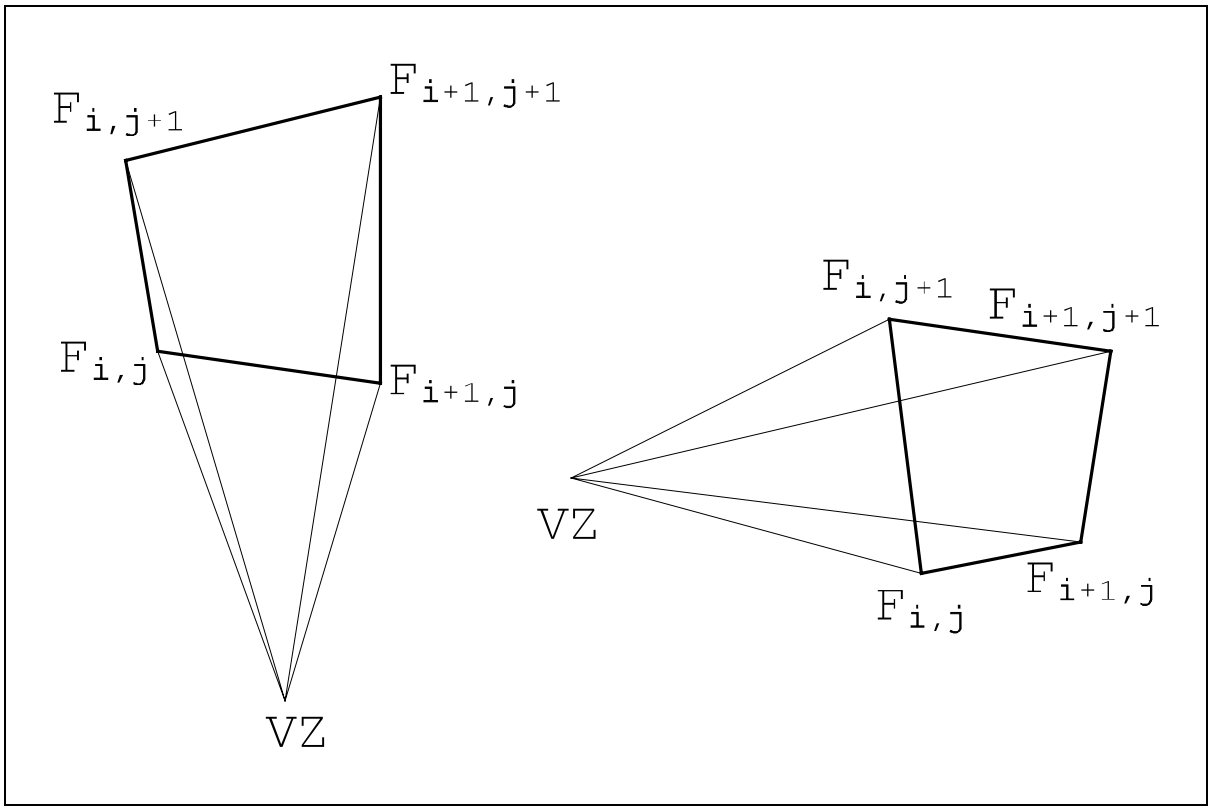

Figura 3.37 Parcurgerea în sens antiorar a vertex-urilor faţetei.

```
var d, s : real;
begin
   d := -dfx*dpy + dfy*dpx;
   s := (-(p1^.x-f1^.x)*dpy + (p1^.y-f1^.y)*dpx)/d;
   ix := f1^.x + s*dfx;
   iy := f1^.y + s*dfy;
end {procedure intersect};

function unghi(x, y : real) : real;
begin
   x := x-vzx; y := y-vzy;
   if abs(x) >= abs(y)
   then if x > 0
      then unghi := y/x
      else unghi := 4+y/x
   else if y > 0
      then unghi := 2-x/y
      else unghi := 6-x/y
end {function unghi};

function maiMic(a1, a2 : real) : boolean;
begin {discontinuitatea din cadranul 3}
   if (a1 > 5) and (a2 < 1)
      then maiMic := true
      else maiMic := a1 < a2
end {function maiMic};
```

```
procedure completeazăListaFaţetei(i, j : integer);
{ introduce vârfurile faţetei (i,j) în listă,
  în sens antiorar }
var di, dj : integer;
    f      : ^vertex;

begin
   {distinge cele patru cazuri}
   if i = k
   then if j > 1
      then begin di := 1; dj := 0
         end
      else begin di := 0; dj := 1
         end;
   if j = 1
   then if i > k
      then begin di := 0; dj := 0
         end
      else begin di := 1; dj := 1
         end;
   f := faţeta;
   repeat
      with f^ do begin
         {obţine (x,y) prin transformarea planului de vedere
          şi proiecţie perspectivă a lui
          (i+di,g[i+di,j+dj],j+dj) }
         a := unghi(x,y);
         f := p;
         if di = dj
         then di := (di+1) mod 2
         else dj := (dj+1) mod s
      end
   until f = nil
end {procedure completeazăListaFaţetei};

begin {determinarea intersecţiilor}
 {creează lista pentru patru vertex-uri; aceasta se
  execută o singură dată, înafara acestei secvenţe,
  de aceea s-a trecut ca şi comentariu
   new(faţeta);    lastf := faţeta;
   new(lastf^.p);  lastf := lastf^.p;
   new(lastf^.p);  lastf := lastf^.p;
   new(lastf^.p);  lastf := lastf^.p;
   lastf^.p := nil; }

   { introduce faţeta(i,j) în listă }
   completeazăListaFaţetei(i, j);

   {determină primul dintre cele două vârfuri care
    aparţine perimetrului: firstp }

   p1 := firstp; p2 := p1^.p;

   f1 := faţeta; f2 := f1^.p;
```

```
   repeat
      dpx := p2^.x-p1^.x;  dpy := p2^.y-p1^.y;
      dfx := f2^.x-f1^.x;  dfy := f2^.y-f1^.y;
      {calculează produsele vectoriale}
      cp1 := dpy*(f1^.x-p1^.x) - dpx*(f1^.y-p1^.y); {P1P2xP1F1}
      cp2 := dpy*(f2^.x-p1^.x) - dpx*(f2^.y-p1^.y); {P1P2xP1F2}
      cp3 := dfx*(p1^.y-f1^.y) - dfy*(p1^.x-f1^.x); {F1P1xF1F2}
      cp4 := dfx*(p2^.y-f1^.y) - dfy*(p2^.x-f1^.x); {F1P2xF1F2}

      {dacă există o intersecţie dinspre interior spre exterior
       atunci o calculează}
      if (cp1 >= 0) and (cp2 < 0) and (cp3 >= 0) and (cp4 < 0)
      then intersect(ix, iy);
      {dacă există o intersecţie dinspre exterior spre interior
       atunci o calculează}
      if (cp1 < 0) and (cp2 >= 0) and (cp3 < 0) and (cp4 >= 0)
      then intersect(ix, iy);

      {avansează pointerii}
      if maiMic(p2^.a, f2^.a)
      then begin
         p1 := p2; p2 := p2^.p
      end
      else begin
         f1 := f2; f2 := f2^.p
      end
   until f2 = nil
end {codul pentru găsirea intersecţiilor};
```

Această secvenţă de cod determină toate intersecţiile laturilor faţetei (i,j) cu perimetrul. Avansul la următoarele vârfuri ale perimetrului şi ale faţetei se vede în Figura 3.38. Termenii "mai mare" şi "mai mic" se referă la unghiurile vertex-urilor. În calculul intersecţiei intervine o pereche de vârfuri ale perimetrului, P_i şi P_{i+1} şi o pereche de vârfuri ale faţetei, F_1 şi F_2. Prima latură a faţetei începe din F_1, iar prima latură a perimetrului începe din P_i, deci următorul vertex F mai mare este F_2, iar următorul vertex P mai mare este P_{i+1}. Acestea sunt capetele laturilor. Se caută posibila intersecţie a acestor laturi prin metoda calculului produsului vectorial, şi dacă aceasta există, ea se calculează. Apoi trebuie trecut la una din următoarele laturi. Dacă cel mai mare dintre cele patru vertex-uri este din P, atunci se ia următoarea latură a faţetei. Dacă s-ar lua următoarea latură din perimetru, ambele capete ale ei ar fi "mai mari" decât capetele laturii faţetei, deci nu există intersecţie. De asemenea, dacă $F_2<P_{i+1}$, atunci şi următorul vertex al faţetei ar putea fi mai mic decât P_{i+1}, deci ar mai putea exista o intersecţie. Analog în cazul în care cel mai mare vertex este din F. În caz de egalitate, luăm următoarea latură a faţetei. În Figura 3.39 se vede un exemplu concret.

În figură se vede pentru care laturi se caută intersecţii:

început:	P_iP_{i+1}	cu F_1F_2
avans F:	P_iP_{i+1}	cu F_2F_3
avans P:	$P_{i+1}P_{i+2}$	cu F_2F_3
avans P:	$P_{i+2}P_{i+3}$	cu F_2F_3

Algoritmul are nevoie doar de primul vârf al perimetrului, P_{min}. Determinarea lui P_{min} constă doar într-o parcurgere a listei perimetrului şi nu este inclusă în cod. Ţinând cont de ordinea în care se adaugă faţetele la perimetru, nu este necesară parcurgerea întregului perimetru pentru determinarea lui P_{min}.

S-a prezentat codul pentru o vedere din faţă; în acest caz avem numai un perimetru exterior. În celelalte cazuri există şi un perimetru interior şi faţeta trebuie comparată şi cu acesta. Întotdeauna vom avea un număr par de intersecţii, prima este o intersecţie de ieşire, a doua - de intrare, şi aşa mai departe, în sens antiorar. Toate intersecţiile se adaugă la perimetru. De asemenea, vârfurile faţetei, situate între o intersecţie de ieşire şi una de intrare se adaugă la lista perimetrului, iar vârfurile existente anterior în lista perimetrului, între cele două intersecţii, se elimină. Se vor trasa toate latirule dintre vârfurile care figurează în listă. Pentru o vedere din faţă, trasarea suprafeţei grilă creşte dinspre interior spre exterior.

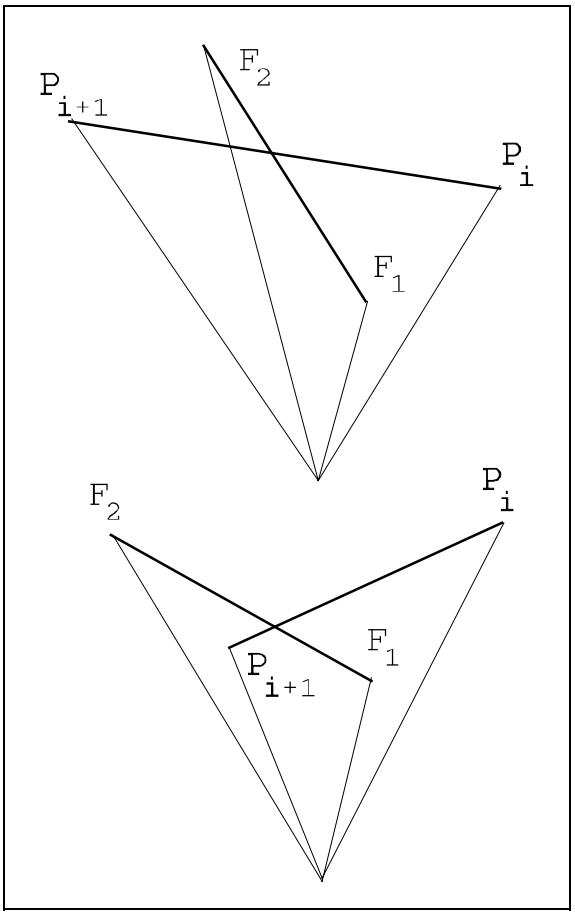

Figura 3.38

Cazuri Extreme Algoritmul de iniţializare şi actualizare a perimetrului trebuie modificat puţin pentru anumite cazuri extreme. Dacă punctul de observaţie PO *se află într-un plan al grilei*, atunci VZ va fi pe proiecţia unei linii $x=x_k$ sau $z=z_l$, sau pe amândouă. Ca urmare, vârfurile perimetrului nu vor avea unghiuri monoton crescătoare la parcurgerea în sens antiorar. Însă secvenţa de cod se bazează pe această monotonie. O rezolvare simplă a acestei probleme este să divizăm suprafaţa grilă în două sau patru porţiuni. Dacă PO se află pe linia $x=x_k$, atunci suprafaţa grilă se divide în porţiunile de la stânga şi de la dreapta ei. Similar, dacă PO se află pe linia $z=z_l$. Dacă PO este pe un punct al grilei, atunci avem două linii de demarcaţie şi patru zone ale suprafeţei grilă. Aceste porţiuni nu se pot acoperi una pe alta, deci ele pot fi procesate independent.

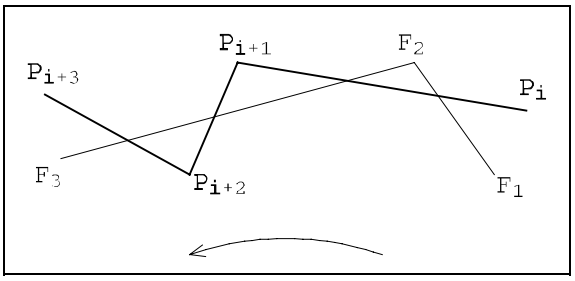

Figura 3.39 Intersecţii faţetă-perimetru.

Să luăm în consideraţie procesarea unui cadran al suprafeţei grilă şi în cadrul acestui cadran vom procesa doar o faţetă adiacentă marginii $x=x_k$. În cazul celorlalte situaţii se aplică simetria. Presupunem că PO se află pe punctul grilei (x_k,z_l) şi că procesăm cadranul din dreapta sus. Perimetrul se iniţializează cu faţeta (k,l) (vezi Figura 3.40). P_b este transformarea vertex-ului (k,g[k,l+1],l+1). P_a corespunde lui (k+1,g[k+1,l],l).

Cele două vârfuri din stânga ale unei faţete (k,j), cu j>l, se află pe o rază din VZ la

P_b. La procesarea unei asemenea faţete, vârfurile care se iau în consideraţie sunt:

F1 = transformarea lui (k+1,g[k+1,j],j)
F2 = transformarea lui (k+1,g[k+1,j+1],j+1)
F3 = transformarea lui (k,g[k,j+1],j+1)

în această ordine. Dacă y-ul lui F3 este mai mare decât y-ul lui P_b, atunci F3 se ia ca intersecţie. Din aceasta rezultă următorul algoritm. La procesarea unei faţete adiacente razei din VZ la P_b, vom utiliza acelaşi ciclu ca şi în secvenţa anterioară pentru trecerea la următoarele laturi din faţetă şi perimetru. Însă după terminarea ciclului efectuăm testul de mai sus şi putem obţine încă o intersecţie. La fel pentru faţetele adiacente razei de la VZ la P_a. Strategia de includere a intersecţiilor calculate în perimetru rămâne aceeaşi. Rezultă o suprafaţă în genul celei din Figura 3.41.

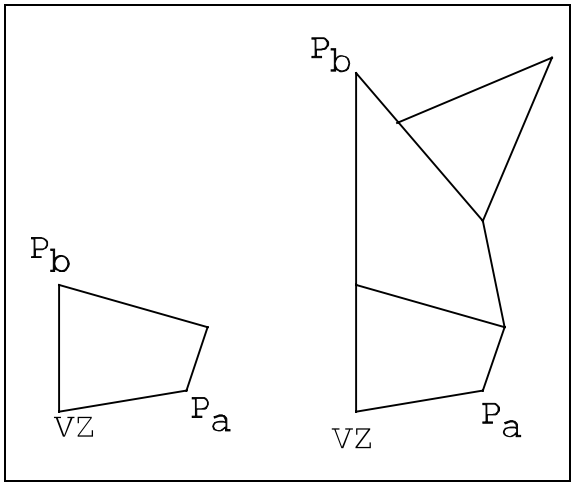

Figura 3.40 Caz special, perimetrul de start, la care se adaugă apoi încă două faţete.

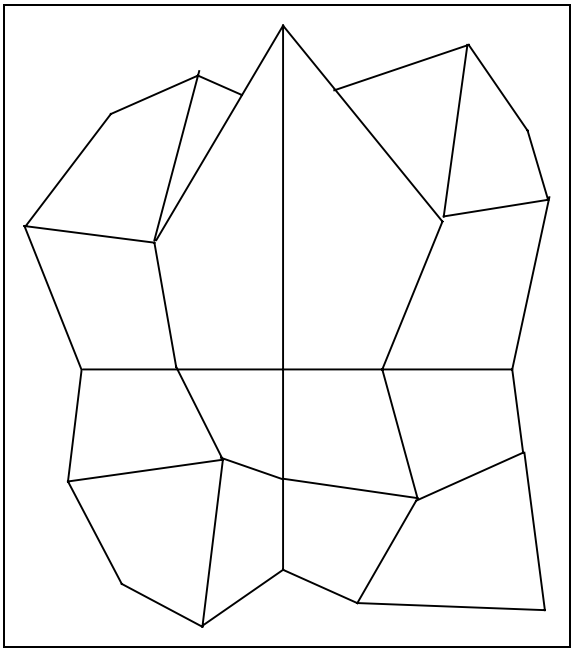

Figura 3.41 Vedere din faţă. PO este deasupra unui punct al grilei.