## Cuprins

# 1. Noţiuni fundamentale

*Grafica pe calculator se ocupă de sinteza picturală a unor obiecte imaginare sau reale pe baza unor modele construite cu ajutorul calculatorului.* Pentru rezolvarea problemelor specifice se utilizează adesea tehnici binecunoscute din geometrie (pentru descrierea spaţiilor bi– şi tri-dimensional, a obiectelor şi a operaţiilor cu acestea), algebră (pentru definirea şi evaluarea expresiilor şi ecuaţiilor matematice), analiză matematică (eşantionare, filtre), analiză numerică (rezolvareaaproximativă a ecuaţiilor), optică (în construirea de modele pentru descrierea comportării luminii), electronică (generarea semnalelor, tehnologii a dispozitivelor), sau psihologie umană (oferă modele pentru percepţia culorii şi pentru realitatea virtuală).

### Sistem grafic

Un *sistem grafic* este un asamblu format din echipamente şi programe specializate în tratarea şi reprezentarea grafică a informaţiei.

Sisteme grafice pot fi clasificate după prelucrările pe care le efectuează în:

1. sisteme de sinteză a imaginilor;
2. sisteme de prelucrare şi de analiză a imaginilor.

Programele din componenţa unui sistem grafic de sinteză sunt structurate în două nivele:

(a) nivelul dependent de echipamente, format prin programele `driver`;
(b) nivelul independent de echipamente, biblioteca grafică de subprograme apelabile din programele de aplicaţie.

### Relaţia grafică – prelucrarea imaginilor

Domeniul alăturat al *prelucrării de imagini* (numit şi *procesare picturală*) tratează procesul invers: analiza scenei sau reconstituirea modelelor unor obiecte bi- sau tri-dimensionale pornind de la imaginea lor. Analiza picturală este importantă în numeroase domenii precum: astronautică, medicină, meteorologie etc. Procesarea de imagini are o serie de subdomenii precum îmbunătăţirea imaginilor, detectarea şi recunoaşterea şabloanelor (recunoaşterea formelor), precum şi analiza scenei. Imbunătăţirea scenei se ocupă, de exemplu, de eliminarea zgomotelor sau de îmbunătăţirea contrastului. Un exemplu practic pentru detectarea şi recunoaşterea şabloanelor este tehnologia `OCR` – recunoaşterea optică a caracterelor. Analiza scenei permite oamenilor de ştiinţă să recunoască şi reconstituie un model al unei scene din mai multe imagini bidimensionale (cu aplicaţii, de exemplu, în robotică). Grafica pe calculator şi procesarea de imagini sunt considerate două discipline separate, însă suprapunerea între cele două este în continuă creştere. Astfel imagini fotografice preluate cu ajutorul unui scanner pot fi combinate cu imagini produse sintetic, sau pot ajuta la crearea unei imaginea unui model.

### Subdomenii ale graficii

Conform standardului (`ACM - Association for Computing Machinery`), subdomeniile graficii sunt: (a) probleme generale; (b) arhitectura hardware (dispozitive I/O, procesoare grafice); (c) sisteme grafice (sisteme dependente/independente de gazdă, sisteme distribuite); (d) *generarea desenelor şi imaginilor* (*primitive*, simboluri, instanţe, *reprezentări grafice, transformări geometrice şi geometrie computaţională, algoritmi de vizualizare*); (e) programe de/cu grafică pe calculator (aplicaţii care utilizează grafica, pachete de grafică, drivere, limbaje de descriere a imaginilor, suport `software`); (f) *modelarea obiectelor* (*curbe şi suprafeţe, ierarhie în modele*, pachete de modelare); (g) *metode, tehnici şi procedee în grafica pe calculator* (independenţa de dispozitiv, *eficienţa în execuţie*, tehnici de interacţiune); (h) *grafica 3D şi sinteza imaginilor realiste* (*eliminarea liniilor şi suprafeţelor ascunsse, texturi, umbre şi culori, animaţie*, algoritmi avansaţi); (i) diverse (sisteme de comunicare vizuală, componente de sinteză în aplicaţii multimedia, realitate virtuală). Subiectele ce vor fi abordate în acest material sunt marcate prin înclinare.

### Aplicaţii ale graficii

Grafica pe calculator, până în anii 1980, a fost un domeniu îngust datorită tehnicii şi programelor extrem de scumpe. Calculatoarele personale au popularizat însă utilizarea graficii rastru (în special în interacţiunea utilizator-calculator). Sistemele grafice au devenit după anii 1980, obiect de lucru al inginerilor, oamenilor de ştiinţă, artiştilor plastici şi arhitecţilor. S-au construit o serie de aplicaţii grafice precum biblioteci de subrutine grafice şi interfeţe cu utilizatorul uşor de manipulat, cu facilităţi de acces la diverse resurse prin intermediul reţelelor. Astfel, *domeniile actuale de aplicaţie* ale graficii pe calculator sunt foarte variate: educaţie şi învăţământ, cercetare ştiinţifică, inginerie (cu ajutorul pachetelor de programe `CAD – Computer Aided Design`, proiectare asistată, `CAM – Computer Aided Machinning`, fabricaţie asistată, `CAE – Computer Aided Engineering`, inginerie asistată), economie, conducere, simulare, cartografie, meteorologie, prospectarea resurselor, artă, comerţ, reclamă, comanda şi urmărirea proceselor, recreere Scopurile aplicaţiilor grafice în domeniile mai sus menţionate sunt de asemenea diverse: afişarea informaţiilor, proiectare, simulare, sau doar interfaţă cu utilizatorul.

### Interactivitate

Grafica se întâlneşte sub formele: secvenţială (generarea automată a imaginilor fără posibilitate de intervenţie din partea observatorului), interactivă sau distribuită (cu aplicaţie în sistemele de realitate virtuală).

Un sistem grafic care permite interacţiunea cu utilizatorul prin echipamente specializate (`mouse, joystick`, tabletă grafică) se numeşte sistem *grafic interactiv*.

Interactivitatea este caracteristica principală de astăzi a graficii: utilizatorul controlează conţinutul, structura şi apariţia obiectelor utilizând dispozitive de intrare. Datorită relaţiei strânse între dispozitivele de intrare şi ecran, studiul unor asemenea dispozitive este considerat subdomeniu al graficii pe calculator.

### Sinteza imaginilor

Crearea unei imagini pe dispozitivul de ieşire a unui sistem grafic presupune două procese. Primul este *procesul de formare al imaginii*. In această etapă, sunt procesate comenzile utilizatorului. Imaginea este formată din elemente (puncte, linii, texte) care sunt disponibile în sistem cu anumite atribute (culoare, font). Procesorul fizic utilizat în această etapă este de obicei procesorul calculatorului gazdă. Al doilea proces este *afişarea imaginii*. In sistemele grafice performante se utilizează, pentru executarea acestei operaţii, *procesorul de terminal*, hardware specializat pentru a asista convertirea primitivelor grafice într-o "hartă" de biţi (de exemplu, un `bitmap`, o reprezentare prin 0 şi 1 a unei matrice rectangulare de puncte a ecranului) şi pentru a executa operaţii de mutare, copiere şi modificare a biţilor. Un procesor de terminal (controlor grafic, coprocesor de `display`, `display processor`, procesorul de imagine rastru, `RIP`) are un set limitat de instrucţiuni. In sistemele grafice simple, precum calculatoarele personale, sarcina controlorului grafic este preluată de o componentă `software` a bibliotecii grafice.

### Grafica rastru (`bitmap graphics`, `raster graphics`)

Există două modalităţi principale de memorare a imaginii: (a) punct cu punct — în grafica rastru (în sistemele precum cel din figura 1.1 cu `raster image display`); (b) ca o mulţime de segmente de dreaptă pentru care se memo-

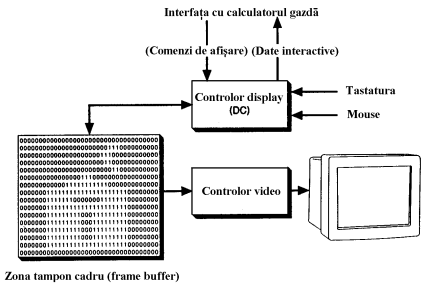



Figura 1.1: Arhitectura unui sistem cu display rastru

rează coordonatele capetelor segmentelor (figura 1.2.b) în sistemul propriu al dispozitivului (`vector, calligraphic display` sau `plotter`). `Display`-urile tip rastru se caracterizează printr-un spaţiu de afişare alcătuit dintr-o ma-

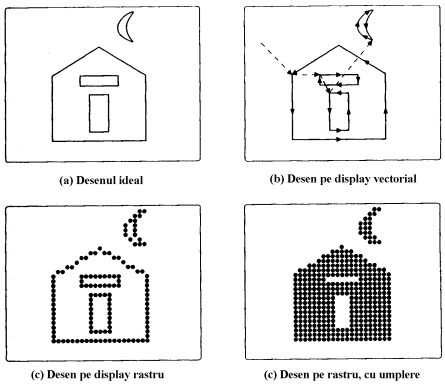



Figura 1.2: Rasterizare şi baleiere vectorială

trice rectangulară de elemente grafice indivizibile. Pentru reprezentarea unei imagini pe ecran, calculatorul sau procesorul dispozitivului de afişare trebuie să genereze o aproximare discretă cât mai fidelă pentru imaginea ideală şi să selecteze în rastru punctele corespunzătoare imaginii discretizate. Definirea unei imagini se realizează printr-un set de valori de intensitate pentru fiecare punct-ecran şi aceste valori sunt vizualizate pe ecran câte o linie la un moment dat.

### Pixel

Elementul cel mai fin vizibil pe ecran este *spotul* (punct, `pixel`, `picture--element`). Dimensiunea spotului este diametrul punctului luminos pe care îl creează pe ecran fascicolul focalizat al tunului tubului de electroni. Ca ordin de mărime, spotul are în jur de 0.02 inch = 0.508 mm (1 inch = 2.54 cm). Distanţa dintre două puncte adiacente trebuie să fie mai mică decât diametrul unui punct pentru a asigura trasarea unei imagini continue.

### Rezoluţie

Un parametru foarte important al unui `display` este rezoluţia. Ea se defineşte prin dimensiuni relative la cel mai mic detaliu ce poate fi distins pe ecran, respectiv *rezoluţia* unui ecran este indicată prin produsul dintre numărul de pixeli pe orizontală şi pe verticală. Rezoluţia poate fi definită şi prin numărul maxim de puncte care pot fi afişate fără suprapunere sau numărul de linii distincte care pot fi create pe ecran într-un `inch` (dacă 40 de linii negre intercalate cu 40 de linii albe pot fi distinse într-un `inch`, rezoluţia este de 80 linii/`inch`).

### Imagine digitizată

*O imagine digitizată este reprezentată printr-o matrice în care fiecare element este o colecţie de numere ce descriu atributele unui pixel al imaginii* (sau o funcţie de variabilă discretă: unei perechi de numere naturale i se asociază o valoare). Adesea aceste numere sunt reprezentări discrete ale unui interval de numere reale. De exemplu, întregii de la 0 la 255 pot fi utilizaţi pentru a reprezenta o diviziune echidistantă a intervalului [0,1] (numerele reprezentând intensităţi ale punctelor imaginii în scara de gri sau intensităţi ale unor componente ale culorilor respectivelor puncte). Pentru o imagine colorată, valoarea asociată unui pixel este un ansamblu de trei numere reprezentând intensităţile componentelor de roşu, verde şi albastru ale culorii pixelului sau trei numere reprezentând indexi în tabele de intensităţi de roşu, verde şi albastru sau un singur număr care este un index într-o tabelă de triplete de culoare. Dimensiunile matricii sunt numite *lăţime*, respectiv *înălţime* a imaginii, iar numărul de biţi asociaţi cu fiecare pixel al matricei este *adâncimea* imaginii.

### Rastru

*Rastrul* este matricea de pixeli ce reprezintă întreaga arie a ecranului. Fiecare linie de pixeli este referită ca linie de *baleiere* sau *scanare*.

### Zona tampon cadru (`frame-buffer`).

Pixelii unei imagini sunt stocaţi într-o zonă de memorie specială, numită *zona tampon cadru*. Motivul introducerii acestei memorii speciale este următorul: dacă memoria video poate fi citită din program, atunci imaginea poate fi construită chiar în această memorie; din păcate, nu toate terminalele oferă posibilitatea de citire prin program a stării bitului asociat cu un punct de pe ecran. Astfel, pentru a oferi posibilitatea prelucrării ulterioare a reprezentărilor simple ale imaginii, se poate crea un "ecran virtual" în care se construieşte imaginea, o zonă de memorie care este identică ca mărime cu memoria video. După definitivare, imaginea se transpune de aici în memoria video, deci pe ecran. Zona tampon cadru se întâlneşte şi sub denumirile de zonă (`buffer`) de împrospătare, `frame buffer, refresh buffer` sau `bitmap/pixmap` (`bitmap` pentru sisteme cu 1 bit pentru reprezentarea unui pixel, `pixmap` sau `pixel-map` pentru sisteme cu mai mulţi biţi pentru un pixel), fiind folosită şi din considerente de creştere a vitezei atunci când există mai multe plane de memorie video. In calculatoarele personale, `frame-buffer`-ul face parte din memoria calculatorului, iar

memoria video este conţinută în spaţiul propriu zis de adresare al procesorului standard. Imaginea este formată în memoria video direct, fără a mai fi nevoie de o transmisie, obţinându-se astfel o mare viteză de execuţie.

### Rasterizare

Primitivele grafice, precum segmentele de linii sau textele, sunt afişate prin excluderea sau includerea unor pixeli în zona tampon cadru. Acest proces este numit *conversie de scanare*, *conversie de baleiere* sau *rasterizare*.

### Reîmprospătare

Zona tampon cadru este scanată (baleiată) secvenţial de *adaptorul grafic* (o linie de rastru la un moment dat) cu o rată de 50 la 70 de ori pe secundă şi imaginea este împrospătată linie cu linie, în modul în care se produc imaginile de televiziune (figura 1.3).

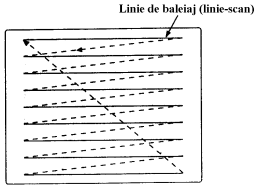



Figura 1.3: Baleierea ecranului de tip rastru

### Cuplor grafic

Pentru afişarea propriu-zisă a imaginilor memoria video este citită secvenţial de un bloc logic, independent de procesor, care realizează semnalele pentru monitor, conform standardului acestuia. Această funcţie este realizată de *adaptorul grafic*, numit şi *cuplor grafic* sau *cuplor video* (*controlor video* în figura 1.1). Performanţele unui cuplor grafic sunt măsurate prin viteza de execuţie, rezoluţia (dimensiunea în pixeli, puncte) şi numărul de culori simultane. Standardele pentru cuploare grafice sunt: `HGC` (`Hercules Graphics Card`), `CGA` (`Color Graphics Adapter`), `EGA` (`Enhanced Graphics Adapter`), `VGA` (`Video Graphics Array`), `SVGA` (`Super Virtual Graphics Adapter`), `AGA` (`Advanced Graphics Adapter`).

### Clasificarea aplicaţiilor grafice

O aplicaţie grafică poate fi inclusă într-una din următoarele categorii:

1. editor grafic (de exemplu `AutoCAD, Windows Paint`),
2. bibliotecă grafică (asociate cu limbaje de programare precum `C`),
3. program specializat cu facilităţi grafice (de exemplu, produsele `CAS - Computer Algebra System`, precum `Maple` sau `Mathematica`),
4. produs de birotică (de exemplu `Word`, `Excel`).

### Sisteme carteziene de referinţă

In aplicaţiile grafice se disting următoarele trei sisteme carteziene de referinţă:

(1) sistemul de coordonate ale lumii reale sau înconjurătoare (sistemul de coordonate logice / universale / utilizator) în care observatorul este centrul acestuia (`WCS` – `World Coordonate System`);

(2) sistemul de coordonate ale unui dispozitiv (`DCS` – `Device Coordonate System`, sistemul de coordonate fizice / dispozitiv);

(3) sistemul de coordonate normalizate ale unui dispozitiv (`NDCS`– `Normalized Device Coordonate System`).

Transformarea coordonatelor din sistemul (1) în sistemul (2) cade în sarcina sistemelor grafice (figura 1.4). In general, un sistem grafic are o varietate de intrări şi ieşiri, astfel încât pachetul grafic ar trebui să gestioneze mai multe sisteme (2). In majoritatea aplicaţiilor se utilizează un sistem de coordonate intermediar, desemnat prin (3). Se consideră astfel un dispozitiv virtual în termenii căruia se scrie `soft`-ul necesar. Dispozitivul virtual uzual este pătratul unitate din sistemul (3), care are colţul din stânga jos în originea sistemului. Valorile exprimate în unităţi de pe dispozitivul virtual sunt transformate în valori în sistemul (2) ca ultimă etapă de procesare. Această transformare este realizată de *driver-urile de dispozitiv* (componente `software` ale sistemelor grafice). Se permite astfel adăugarea unui nou dispozitiv în sistemul grafic, dacă este însoţit de un anumit `driver`, fără a fi necesară alterarea pachetelor grafice.

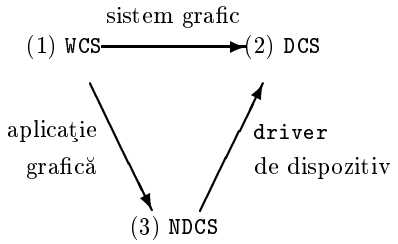



Figura 1.4: Transformarea unei imagini din lumea reală în imagine pe dispozitivul de ieşire al unui sistem grafic

# 2. Geometria vizualizării obiectelor tridimensionale

## 2.1 Transformări bidimensionale

Se consideră un reper cartezian, un obiect bidimensional descris în respectivul sistem şi $P(x, y)$ un punct oarecare al obiectului.

### Translaţie

Operaţia este definită relativ la cele două axe de coordonate carteziene prin deplasamentele specifice pe fiecare axă (figura 2.1).

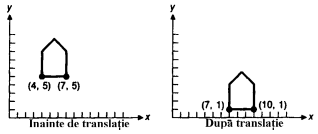



Figura 2.1: Translatarea unei case

Punctul $P(x, y)$ este translatat în punctul $P(x', y')$ dacă

$$\begin{cases} x' = x + d_x, \\ y' = y + d_y. \end{cases}$$

unde $d_x$, $d_y$ sunt *deplasamentele specifice*. Vectorial, operaţia poate fi scrisă

$$P' = P + T$$

unde

$$P = \begin{pmatrix} x \\ y \end{pmatrix}, \qquad P' = \begin{pmatrix} x' \\ y' \end{pmatrix}, \qquad T = \begin{pmatrix} d_x \\ d_y \end{pmatrix}.$$

### Scalare

Scalarea poate fi descrisă prin ecuaţiile

$$\begin{cases} x' = s_x x, \\ y' = s_y y, \end{cases}$$

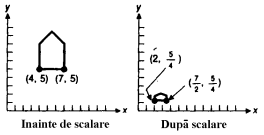



Figura 2.2: Scalarea unei case

unde $s_x$ şi $s_y$ sunt *constantele de scalare*, $P(x, y)$, punctul iniţial, iar $P'(x', y')$ punctul transformat (figura 2.2). Un factor de scalare negativ produce ca efect secundar reflecţia faţă de axa corespunzătoare. Ecuaţia vectorială este

$$P' = SP$$

unde

$$S = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}.$$

**Rotaţie**

Prin rotaţia cu un unghi $\theta$ faţă de origine, punctele unui obiect rămân la aceeaşi distanţă faţă de originea sistemului (figura 2.3). Un unghi pozitiv respectă

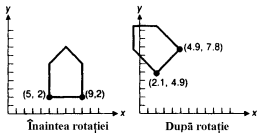



Figura 2.3: Rotirea unei case în jurul originii sistemului

sensul de rotaţie trigonometric (contrar fusului orar). Utilizând transformarea din coordonate carteziene în coordonate polare

$$\begin{cases} x = r\cos\phi, \\ y = r\sin\phi, \end{cases}$$

după rotaţie (figura 2.4)

$$\begin{cases} x' = r\cos(\phi + \theta), \\ y' = r\sin(\phi + \theta), \end{cases}$$

Prin dezvoltarea funcţiilor `cos` şi `sin` din expresia anterioară se obţin ecuaţiile

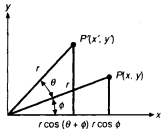



Figura 2.4: Derivarea ecuaţiilor rotaţiei

transformării în coordonate carteziene:

$$\begin{cases} x' = x\cos\theta - y\sin\theta, \\ y' = x\sin\theta + y\cos\theta. \end{cases}$$

Vectorial, operaţia poate fi exprimată prin

$$P' = RP,$$

unde

$$R = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

### Coordonate omogene

Pentru transformările bidimensionale amintite mai sus, ecuaţiile vectoriale sunt următoarele:

$$P' = T + P, \quad P' = SP, \quad P' = RP.$$

Se observă că translaţia este tratată în mod distinct, printr-o adunare.

Reprezentarea punctelor în coordonate omogene permite tratarea tuturor transformărilor prin multiplicări matriceale. Pe lângă cele două coordonate carteziene standard apare o a treia: un punct $(x, y)$ în coordonate carteziene este reprezentat prin tripletul $(x, y, w)$ în coordonate omogene. Reprezentarea unui punct nu este unică. De exemplu, (2, 4, 8) şi (1, 2, 4) reprezintă acelaşi punct din plan. Dacă $w \neq 0$, atunci $(x, y, w)$ reprezintă acelaşi punct ca şi $(x/w, y/w, 1)$, iar coordonatele $(x/w,\ y/w)$ sunt coordonatele carteziene asociate punctului omogen. Punctele cu $w = 0$ se numesc *puncte la infinit*.

Un triplet reprezintă de obicei coordonatele unui punct din spaţiu tridimensional. Dacă considerăm toate tripletele ce reprezintă acelaşi punct în coordonate carteziene, adică toate tripletele de forma $(tx, ty, tw)$, cu $t \neq 0$ obţinem o linie în spaţiul tridimensional (figura 2.5). Astfel fiecare punct în coordonate omogene reprezintă o linie în spaţiul tridimensional. Astfel punctele omogenizate formează un plan definit de ecuaţia $w = 1$ în spaţiul $(x, y, w)$.

Matricile transformărilor mai sus amintite, se rescriu în coordonate omogene astfel:

$$T(d_x, d_y) = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix}, \; S(s_x, s_y) = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

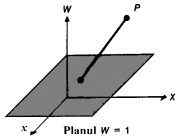



Figura 2.5: Spaţiul coordonatelor omogene, cu planul $w = 1$ şi un punct $P(x, y, w)$ proiectat în planul $w = 1$

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

iar noile relaţii vectoriale sunt:

$$P' = T(d_x, d_y)P, \text{ sau } P' = S(s_x, s_y)P, \text{ sau } P' = R(\theta)P.$$

### 2.1.1 Transformări afine

Dacă un punct este translatat cu $T(d_x^1, d_y^1)$ în $P_1$ şi apoi cu $T(d_x^2, d_y^2)$ în $P_2$, atunci

$$P_2 = T(d_x^2, d_y^2)P_1 = T(d_x^2, d_y^2)T(d_x^1, d_y^1)P = T(d_x^1 + d_x^2, d_y^1 + d_y^2)P.$$

Produsul matriceal este adesea referit drept *compunere* sau *concatenare*. In mod similar, pentru scalare $S(s_x^1, s_y^1) \cdot S(s_x^2, s_y^2) = S(s_x^1 s_x^2, s_y^1 s_y^2)$, iar pentru rotaţie $R(\theta_1) \cdot R(\theta_2) = R(\theta_1 + \theta_2)$. Se observă că matricea rotaţiei este o matrice ortogonală (fiecare vector-linie a matricii este unitar şi perpendicular pe celelalte). In plus cele două direcţii definite de primii doi vectori-linie a unei matrice de rotaţie sunt direcţiile în care vor fi reorientaţi vectori aflaţi pe axele $x$ respectiv $y$ (figura 2.6). Această proprietate poate conduce la construcţia cu

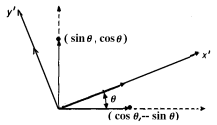



Figura 2.6: Cordonatele versorii axelor sistemului $xOy$, faţă de sistemul $x'Oy'$ obţinut prin rotaţie rotit, corespund cu liniile din matricea de rotaţie

uşurinţă a matricei de rotaţie dacă efectul rotaţiei este cunoscut.

Majoritatea transformărilor efectuate în lumea reală păstrează liniile drepte. Asemenea transformări includ rotaţia, reflecţia, scalarea şi translaţia. O matrice de forma

$$\begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

unde submatricea $2 \times 2$ superioră este ortogonală, păstrează unghiurile şi lungimile. O transformare asociată unei asemenea matrice este numită *transformare de corp rigid* deoarece orice obiect transformat nu este distorsionat. O secvenţă arbitrară de matrice de rotaţii şi translaţii este reprezentabilă printr-o matrice de forma de mai sus.

Produsul unui secvenţe arbitrare de matrice de rotaţii, translaţii şi scalări (transformări elementare sau simple) păstrează paralelismul liniilor, dar nu şi lungimile şi unghiurile (figura 2.7). Transformarea corespunzătoare unui

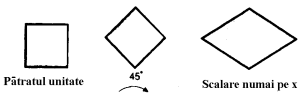



Figura 2.7: Un pătrat unitate este rotit şi scalat neuniform; transformarea fiind afină, paralelismul liniilor se menţine, dar nu şi unghiurile şi lungimile

asemenea produs se numeşte *transformare afină*.

O secvenţă de transformări afine poate fi reprezentată printr-o matrice unică. De exemplu, rotaţia în jurul unui punct $P_1(x_1, y_1)$ poate fi descrisă

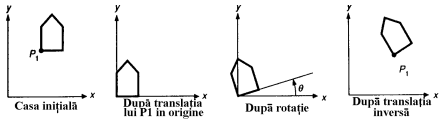



Figura 2.8: Rotirea unei case în jurul unui punct $P_1$

prin produsul matriceal $CBA$, unde

$$A = T(-x_1, -y_1), \qquad B = R(\theta), \qquad C = T(x_1, y_1).$$

O clasă importantă de operaţii o constituie transformările inverse. Dacă transformarea $A$ produce $P' = AP$, transformarea inversă produce $P = A^{-1}P'$.

Un exemplu concludent pentru faptul că transformările afine păstrează paralelismul dintre linii dar nu şi lungimile şi unghiurile, este *înclinarea* sau *forfecarea* faţă de o anumită axă (figura 2.9). Corespunzător axei $x$ şi unghiului de


Figura 2.9: Aplicarea operaţiei de înclinare faţă de axe la pătratul unitate – laturile oblice obţinute au lungimea mai mare decât unitatea

înclinare $\phi$, ecuaţiile transformării sunt

$$\begin{cases} x' = x + y \operatorname{ctg} \phi, \\ y' = y, \end{cases}$$

ecuaţii care conduc la matricea de transformare

$$H_x = \begin{pmatrix} 1 & \operatorname{ctg} \phi \\ 0 & 1 \end{pmatrix}.$$

Operaţia poate fi obţinută prin concatenarea a trei transformări elementare. In cazul general, forfecarea este descrisă prin ecuaţiile $x' = x + F_x y,\ y' = y + F_y x$ (forfecarea pe axa $x$ : $F_y = 0$).

## 2.2 Ferestre şi zone de lucru

Anumite aplicaţii grafice permit specificarea primitivelor grafice în sistemul de coordonate ale lumii reale (`WCS`) utilizând unităţi de lungime. Pachetul de subrutine grafice trebuie să efectueze transformarea din coordonatele lumii reale în coordonatele ecranului (`DCS`).

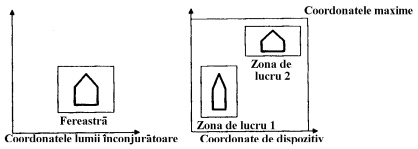



Figura 2.10: Efectul trasării primitivelor în două zone de lucru

Termenul de *fereastră* (`window`) a fost introdus spre a desemna o zonă dreptunghiulară a unui plan situat în spaţiul de coordonate ale utilizatorului (`WCS`), pe care se proiectează informaţia dorită la un moment dat (figura 2.10). Unei

asemenea ferestre i se asociază o regiune dreptunghiulară în coordonatele normalizate de terminal (`NDCS`), numită *zonă de lucru* (`viewport`, *poartă de afişare* sau *poartă*), în care imaginea din fereastră este transformată (digitizată).

Fie fereastra specificată prin $(x_{min}, y_{min})$, $(x_{max}, y_{max})$, coordonatele a două vârfuri extreme ale zonei dreptunghiulare din `WCS`, iar zona de lucru, prin $(u_{min}, v_{min})$, $(u_{max}, v_{max})$, unde $0 \leq u_{min} < u_{max} \leq 1$, $0 \leq v_{min} < v_{max} \leq 1$, coordonatele vârfurilor corespunzătoare ale zonei dreptunghiulare din `NDCS`. Matricea transformării fereastră – zonă de lucru poate fi construită prin concatenarea celor trei transformări sugerate în figura 2.11:

$$P' = MP,$$

unde

$$M = T(u_{min}, v_{min}) S \left( \frac{u_{max} - u_{min}}{x_{max} - x_{min}}, \frac{v_{max} - v_{min}}{y_{max} - y_{min}} \right) T(-x_{min}, -y_{min}).$$

Astfel, ecuaţiile de transformare a unei ferestre într-o zonă de lucru sunt următoarele:

$$\begin{cases} u = u_{min} + \dfrac{u_{max} - u_{min}}{x_{max} - x_{min}}(x - x_{min}), \\ v = v_{min} + \dfrac{v_{max} - v_{min}}{y_{max} - y_{min}}(y - y_{min}). \end{cases}$$

Un triplet similar de transformări se aplică pentru trecerea de la `NDCS` la `DCS`.

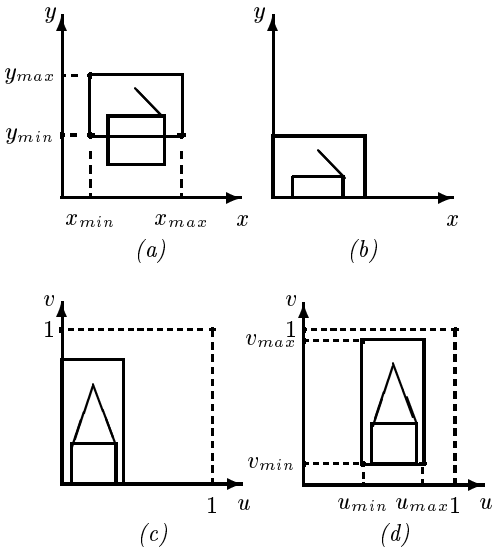



Figura 2.11: Transformarea unei ferestre într-o zonă de lucru (a) Fereastra în coordonatele lumii reale (b) Fereastra translatată în originea sistemului (c) Fereastra scalată la dimensiunea zonei de lucru (d) Translatarea imaginii în zona de lucru

In cazul construirii unei imagini sub limitele spaţiului vizibil, pot apărea o serie de primitive care vor intersecta frontiera ferestrei (figura 2.11.a). Este necesară eliminarea din aceste primitive a porţiunilor care ies din zona observabilă (din fereastră). Această operaţie este cunoscută sub numele de `clipping` (*decupare, retezare, tăiere*).

## 2.3 Transformări tridimensionale

Aşa cum unei transformări bidimensionale exprimate în coordonate omogene îi corespunde o matrice $3 \times 3$, o transformare tridimensională este reprezentată printr-o matrice $4 \times 4$.

Precum în cazul bidimensional, se asociază unui punct $(x, y, z)$ în coordonate carteziene, un punct $(x, y, z, w)$ în coordonate omogene, cu proprietatea că două cvadruple reprezintă acelaşi punct dacă coordonatele lor sunt proporţionale. Reprezentarea standard a unui punct omogen $(x, y, z, w)$ cu $w \neq 0$ este $(x/w, y/w, z/w, 1)$.

**Translaţia** tridimensională este o simplă extensie a celei bidimensionale. Matricea transformării este

$$T(d_x, d_y, d_z) = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

**Rotaţia** oarecare a unui sistem de puncte se descompune în cel mult trei rotaţii, maxim câte una după fiecare axă a unui reper triortogonal. Pentru rotaţiile în jurul axelor de coordonate s-au convenit anumite direcţii ale rotaţiilor pozitive (figura 2.12).

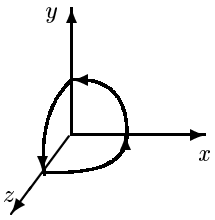
Ţinând seama că *rotaţiile în jurul axelor* pot fi exprimate prin rotaţii în spaţiul bidimnesional al planelor perpendiculare pe axe, ele pot fi reprezentate prin matricele de transformare următoare:

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \; R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

*Rotaţia unui obiect în jurul unei drepte arbitrare*, dar paralelă cu o anumită axă de coordonate presupune:

1. translatarea obiectului astfel încât axa de rotaţie să coincidă cu axa paralelă de coordonate;
2. efectuarea rotaţiei;
3. translatarea obiectului astfel încât axa de rotaţie să fie mutată în poziţia originală.




— pentru axa $x$, de la axa $y$ înspre axa $z$;

— pentru axa $y$, de la axa $z$ înspre axa $x$;

— pentru axa $z$, de la axa $x$ înspre axa $y$.

Figura 2.12: Sensul rotaţiilor de unghi pozitiv

Dacă axa de rotaţie nu este paralelă cu nici una din axele de coordonate este necesară rotaţia în prealabil a acestei axe astfel încât să fie paralelă cu una din axele de coordonate. In final se aplică transformarea inversă pentru a aduce axa de rotaţie la orientarea iniţială.

Spre deosebire de rotaţie şi translaţie, înmulţirea cu factori de scală afectează distanţele dintre puncte. Sistemul de puncte poate fi modificat diferit după fiecare din cele trei axe ale sistemului de referinţă, caz în care se definesc factorii de scară direcţionali $s_x,\ s_y,\ s_z$. Astfel **scalarea** tridimensională este descrisă prin matricea

$$S(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Cea mai simplă transformare de acest gen este *asemănarea*, care foloseşte un factor de scală global, $s$, cu care se înmulţesc coordonatele corpului. Alte cazuri particulare de scalări sunt *simetriile* pentru care $s_x,\ s_y,\ s_y$ sunt $-1$ sau 1. Astfel, de exemplu, pentru

(a) simetria faţă de planul $xOy$: $s_x = s_y = 1,\ s_z = -1$;
(b) simetria faţă de axa $y$: $s_x = s_z = -1,\ s_y = 1$;
(c) simetria faţă de origine: $s_x = s_y = s_z = -1$.

*Simetria faţă de un plan oarecare* se poate rezolva aplicând sistemului de referinţă o translaţie şi o rotaţie astfel încât un plan de coordonate să se suprapună peste planul dat şi apoi se efectuează simetria faţă de acel plan de coordonate. După aceasta se face rototranslaţia inversă. Analog se procedează în cazul unei drepte oarecare. In cazul unui punct este suficientă o translaţie aplicată sistemului de coordonate astfel încât originea să ajungă în punctul dat, apoi se aplică relaţiile de simetrie faţă de origine; după calcularea coordonatelor punctului simetric, se aplică translaţia inversă. Simetriile se pot exprima ca produs de simetrii faţă de planele de coordonate. *Simetria faţă de o axă* se obţine prin concatenarea simetriilor faţă de cele două plane a căror intersecţie este axa, iar simetria faţă de origine rezultă prin aplicarea succesivă a celor trei simetrii faţă de planele de coordonate. La rândul lor, aceste simetrii faţă de planele de coordonate se pot obţine folosind simetria faţă de un singur plan de coordonate şi rotaţiile care suprapun planele de coordonate unul peste celălalt.

*Inclinarea* sau *forfecarea* faţă de planul $xOy$ este descrisă prin matricea

$$H_{xy}(h_x, h_y) = \begin{pmatrix} 1 & 0 & h_x & 0 \\ 0 & 1 & h_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

In cazul general, forfecarea poate fi descrisă printr-o matrice cu elementele de pe diagonală egale cu 1.

Compunerea transformărilor tridimensionale fundamentale (rotaţia, translaţia şi scalarea) este utilizată pentru derivarea unor transformări mai complexe. Fie transformarea din figura 2.13. Această transformare poate fi realizată în 4 paşi:

1. translatarea lui $P_1$ în origine;
2. rotirea în jurul axei $y$ astfel încât $P_1P_2$ să fie aşezat în planul $(y, z)$;
3. rotirea în jurul axei $x$ astfel încât $P_1P_2$ să fie aşezat pe axa $z$;
4. rotire în jurul axei $z$ astfel încât $P_1P_3$ să fie aşezat în planul $(y, z)$.

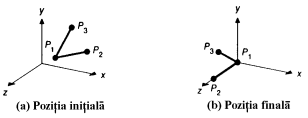



Figura 2.13: Vectorii $P_1P_2$ şi $P_1P_3$ sunt transformaţi din poziţia iniţială (a) în poziţia finală (b)

Calculele referitoare la rotire nu sunt simple. Pentru a le evita putem folosi proprietatea, menţionată anterior, a matricelor de rotaţie: vectorii unitari descrişi de liniile matricei corespund direcţiilor obţinute prin rotirea versorilor axelor. Fie compunerea rotaţiilor de la paşii 2-4 reprezentată prin matricea

$$R = \begin{pmatrix} r_{1_x} & r_{2_x} & r_{3_x} & 0 \\ r_{1_y} & r_{2_y} & r_{3_y} & 0 \\ r_{1_z} & r_{2_z} & r_{3_z} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

In cazul considerat, $R_z$ corespunde vectorului unitate de-a lungul lui $P_1P_2$ care va fi rotit pentru a se sprijini pe axa $z$; astfel

$$R_z = (r_{1_z}, r_{2_z}, r_{3_z})^T = \frac{P_1P_2}{\|P_1P_2\|}.$$

Vectorul unitate perpendicular pe planul definit de $P_1$, $P_2$ şi $P_3$, este rotit pentru a se sprijini pe axa $x$, deci

$$R_x = (r_{1_x}, r_{2_x}, r_{3_x})^T = \frac{P_1P_3 \times P_1P_2}{\|P_1P_3 \times P_1P_2\|}.$$

Vectorul $R_y$,

$$R_y = (r_{1_y}, r_{2_y}, r_{3_y})^T = R_z \times R_x$$

este rotit pentru a se sprijini pe axa $y$. Matricea finală a transformării este $R \cdot T(-x_1, -y_1, -z_1)$.

Considerăm un alt exemplu. Avionul din figura 2.14.(a) definit în sistemul de coordonate $x_P, y_P, z_P$ şi centrat în origine trebuie transformat astfel încât să fie orientat în direcţia $D$ (direcţia de zbor) din figura 2.14.(b), neînclinat şi centrat într-un punct $P$. Transformarea constă într-o rotaţie pentru a orienta avionul în direcţia dorită, urmată de o translaţie a originii în $P$. Presupunem că

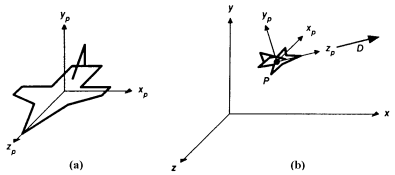



Figura 2.14: Avionul în sistemul de coordonate $(x_P, y_P, z_P)$

vectorii direcţionali $x_p$, $y_p$, $z_p$, $D$ sunt normalizaţi. Precum în exemplul anterior construim matricea de rotaţie cunoscând transformarea axelor: $z_p$ trebuie transformat în $D$, $x_p$ trebuie transformat într-un vector orizontal perpendicular pe $D$, adică în direcţia $y \times D$, $y_p$ este dat de $z_p \times x_p$. Notând cu $|\cdot|$ vectorii normalizaţi, matricea de rotaţie are forma

$$R = \begin{pmatrix} & & & 0 \\ |y \times D| & |D \times (y \times D)| & |D| & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^T .$$

## 2.4 Proiecţii

Complexitatea vizualizării obiectelor tridimensionale este cauzată de a treia dimensiune şi de faptul că terminalele prezintă imagini bidimensionale. Soluţia este utilizarea proiecţiilor care transformă obiectele tridimensionale în proiecţii plane bidimensionale.

In general, proiecţiile transformă puncte dintr-un sistem de coordonate de dimensiune $n$ în puncte din sistemul de coordonate de dimensiune mai mică

decât $n$. In grafica pe calculator, obiecte $n$-dimensionale sunt proiectate în spaţiu bidimensional pentru vizualizare.

### 2.4.1 Clasificare

Proiecţia unui obiect tridimensional este definită astfel: *dreptele (razele) de proiecţie*, numite şi *proiectori*, trec printr-un punct dat al spaţiului, numit *centru de proiecţie (punct de vedere)*, şi prin fiecare punct al corpului, intersectând *planul de proiecţie* pentru a forma *proiecţia*.

Deoarece suprafaţa ecranului este plană, ne rezumăm la proiecţiile geometrice plane. Acestea pot fi împărţite în două clase:

1. *proiecţii paralele*,
2. *proiecţii perspective*.

Distincţia se face funcţie de distanţa dintre centrul de proiecţie şi planul de proiecţie. Dacă această distanţă este finită, proiecţia este perspectivă; dacă distanţa este infinită, proiecţia este paralelă (figura 2.15).

In proiecţia perspectivă se precizează poziţia centrului de proiecţie şi planul de proiecţie.

In cazul proiecţiei paralele, centrul de proiecţie se află la infinit, ceea ce face ca dreptele de proiecţie să fie paralele (în figura 2.15.b proiectorii $AA'$ şi $BB'$ sunt paraleli). Pentru a defini o proiecţie paralelă se precizează direcţia de proiecţie (un vector paralel cu proiectorii).

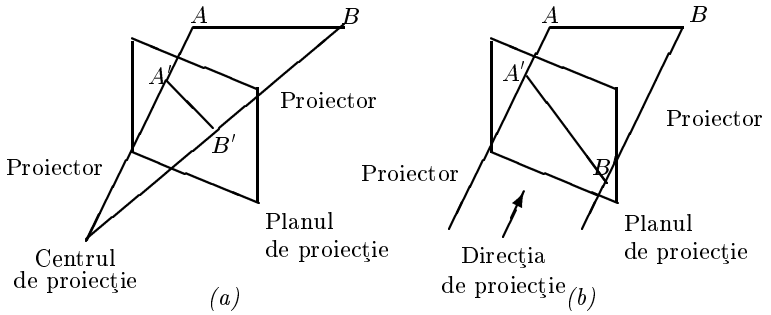



Figura 2.15: Proiecţiile unui segment de dreaptă (a) $A'B'$ este proiecţia perspectivă a lui $AB$ (b) $A'B'$ este proiecţia paralelă a lui $AB$.

*Observaţii:*

(a) Proiecţia paralelă păstrează paralelismul liniilor, dar nu păstrează unghiurile (mai puţin cele aflate în planuri paralele cu planul de proiecţie).

(b) Efectul vizual al proiecţiei perspective este asemănător cu cel realizat de tehnica fotografică şi de sistemul vizual uman. Principala caracteristică este aceea că dimensiunea proiecţiei perspective a unui obiect variază invers proporţional cu distanţa de la obiect la centrul de proiecţie (figura 2.16). Distanţele nu sunt cele reale, unghiurile se păstrează numai dacă aparţin unor feţe ale obiectului paralele cu planul de proiecţie, iar liniile

paralele nu sunt proiectate, în general, în linii paralele. Lungimile unor segmente egale în spaţiu pot apărea în imagine diferite, depinzând de apropierea de centrul de proiecţie.

(c) Direcţia de proiecţie este un vector a cărui componente pot fi calculate printr-o diferenţă de tipul $(x, y, z, 1) - (x', y', z', 1) = (a, b, c, 0)$. Astfel există o corespondenţă biunivocă între direcţii şi puncte la infinit. O proiecţie perspectivă a cărui centru este un punct la infinit devine o proiecţie paralelă.

Există o serie de cazuri particulare atât pentru proiecţiile paralele cât şi pentru cele perspective:

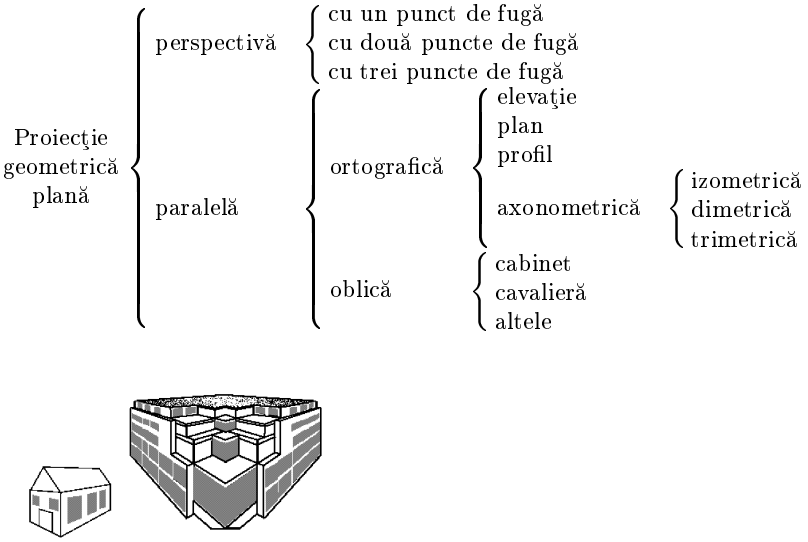



Figura 2.16: Variaţia dimensiunii obiectelor cu distanţa de la observator

### 2.4.2 Proiecţia perspectivă

In proiecţia perspectivă, liniile paralele între ele şi neparalele cu planul de proiecţie converg spre un anumit punct numit **punct de fugă** (*punct de convergenţă*). Dacă liniile sunt paralele cu una dintre axele principale, punctul de fugă este numit *punct de fugă axial* (al axei). Există cel mult trei asemenea puncte într-o imagine, câte unul pentru fiecare axă. De exemplu, dacă planul de proiecţie taie doar axa $z$, numai axa $z$ are punct de fugă deoarece liniile paralele cu axele $y$ şi $x$ sunt paralele şi în planul de proiecţie şi nu au puncte de fugă (nu se intersectează în proiecţie). In figura 2.17 se prezintă două proiecţii

perspective, cu un punct de fugă, ale unui cub. Deoarece în 3D liniile paralele se întâlnesc la infinit, fiecare punct de fugă poate fi considerat proiecţia unui punct la infinit.

Proiecţiile perspective sunt clasificate funcţie de numărul punctelor de fugă axiale.

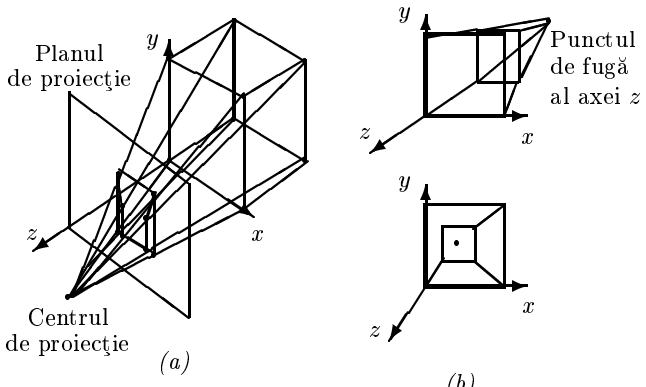



Figura 2.17: Proiecţia perspectivă a unui cub pe un plan care taie numai axa $z$ (a) Construcţia proiecţiei (b) Două perspective cu un punct de fugă

Proiecţiile perspective cu două puncte de fugă sunt utilizate des în arhitectură şi design industrial (figura 2.18). In figura 2.19 se prezintă modul de construcţie al proiecţiei perspective a unui cub cu două, respectiv trei, puncte de fugă.

Presupunem că $P(x, y, z)$ se proiectează în $P'(x', y')$. Ecuaţiile proiecţiei perspective pe planul $z = 0$, cu un punct de fugă şi centrul de proiecţie $C_p$ se obţin din paralelismul liniilor $C_pP$ şi $C_pP'$ şi apartenenţa lui $P'$ la planul $z = 0$:

$$x' = \frac{xz_p - zx_p}{z_p - z}, \quad y' = \frac{yz_p - zy_p}{y_p - z}.$$

Proiecţia perspectivă cu două puncte de fugă se poate obţine printr-o rotaţie în jurul unei axe urmată de o proiecţie perspectivă cu un singur punct de fugă. Transformarea perspectivă cu trei puncte de fugă se obţine prin rotaţii în jurul a două sau mai multe axe principale urmate de transformarea perspectivă cu un singur punct de fugă.

In general, pentru a găsi coordonatele într-o proiecţia perspectivă oarecare a unui punct material se rezolvă sistemul format din ecuaţia planului de proiecţie şi ecuaţia razei vizuale. Reprezentarea perspectivei pe planul de proiecţie prin această modalitate este greoaie. De aceea, se determină, în primul rând, transformarea tridimensională care transpune reperul cartezian al lumii reale (în care este descris obiectul) în reperul cartezian asociat planului de proiecţie şi normalei la acesta, dusă prin centrul de proiecţie. Se aplică această transformare obiectului, apoi se proiectează.

Funcţie de înclinarea planului de proiecţie faţă de centrul de proiecţie şi obiect se deosebesc:

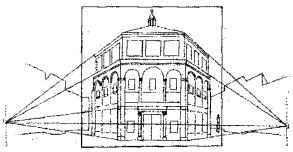


Figura 2.18: Imaginea în perspectivă a unui clădiri

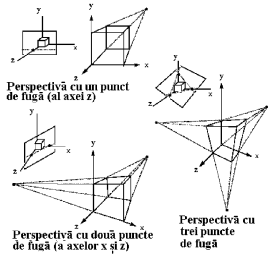



Figura 2.19: Proiecţii perspective ale cubului

1. *perspectiva ascendentă* (cu punctul de fugă al verticalelor situat deasupra obiectului)
2. *perspectiva descendentă* (cu punctul de fugă al verticalelor situat sub obiect).

Perspectiva ascendentă a unui obiect apare în situaţia în care observatorul priveşte obiectul de jos şi destul de aproape. Perspectiva descendentă a unui obiect apare în situaţia în care observatorul priveşte obiectul de sus şi de aproape.

### 2.4.3 Proiecţia paralelă

Funcţie de unghiul dintre direcţia de proiecţie şi normala la planul de proiecţie, proiecţia paralelă poate fi:

1. *proiecţie ortografică (ortogonală)*, în cazul în care direcţia de proiecţie coincide cu normala la planul de proiecţie, adică direcţia de proiecţie este perpendiculară pe planul de proiecţie;
2. *proiecţie oblică*, în cazul în care direcţia de proiecţie diferă de normala la planul de proiecţie.

### Proiecţia ortografică

Cele mai comune proiecţii ortogonale sunt proiecţiile care utilizează, ca plane de proiecţie, anumite plane perpendiculare pe axele de coordonate. Denumirea dată în desenul tehnic unor asemenea proiecţii ale obiectelor sunt *plan (vedere de sus), profil (vedere laterală)* şi *elevaţie (vedere frontală)*.

In figura 2.20 se prezintă cele trei asemenea proiecţii ale unei case (originea sistemului de coordonate se află la intersecţia celor trei plane de proiecţie). Aceste proiecţii au proprietatea de a păstra distanţele şi unghiurile, astfel încât sunt des utilizate în inginerie şi construcţii. Natura tridimensională a obiectului este însă greu de înţeles, chiar dacă se studiază simultan cele trei proiecţii ale respectivului obiect.

Fie punctul de coordonate omogene $(x, y, z, 1)$. Vederea frontală (proiecţia pe planul $xOy$) are matricea caracteristică

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

astfel încât ecuaţiile transformării sunt $x' = x,\ y' = y,\ z' = 0$. Pentru vederea de sus (proiecţia pe planul $xOz$) se efectuează rotaţia de unghi $-90°$ în jurul axei $x$ şi proiecţia pe planul $xOy$:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

astfel încât $x' = x,\ y' = z,\ z' = 0$. Pentru vederea laterală (proiecţia pe planul $yOz$) se efectuează rotaţia de unghi $90°$ în jurul axei $y$ şi proiecţia pe planul $xOy$:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

astfel încât $x' = z,\ y' = y,\ z' = 0$.

Pentru a oferi cât mai multe informaţii despre obiectele reprezentate în proiecţie, cele trei proiecţii ortografice menţionate mai sus sunt adesea detaliate astfel:

1. vedere din faţă: în planul $z = 0$, cu centrul de proiecţie la infinit pe semiaxa $+z$;
2. vedere din spate: în planul $z = 0$, cu centrul de proiecţie la infinit pe semiaxa $-z$;

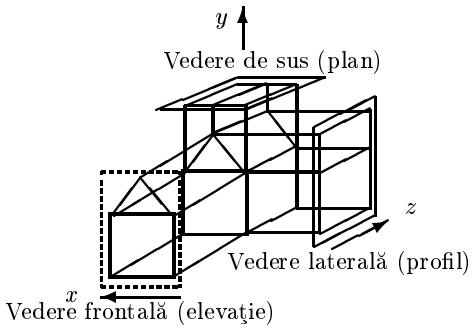


Figura 2.20: Planul, profilul şi elevaţia unei case

3. vedere din stânga: în planul $x = 0$, cu centrul de proiecţie la infinit pe semiaxa $+x$;
4. vedere din dreapta: în planul $x = 0$, cu centrul de proiecţie la infinit pe semiaxa $-x$;
5. vedere de sus: în planul $y = 0$, cu centrul de proiecţie la infinit pe semiaxa $+y$;
6. vedere de jos: în planul $y = 0$, cu centrul de proiecţie la infinit pe semiaxa $-y$.

Proiecţiile *axonometrice* sunt de asemenea cazuri particulare de proiecţii ortogonale, pentru care planul de proiecţie nu mai este perpendicular pe nici o axă a sistemului de referinţă. Acest tip de proiecţii păstrează paralelismul liniilor, dar nu şi unghiurile. Distanţele se măsoară de-a lungul fiecărei axe de coordonate, în general, cu factori de scală diferiţi.

Cea mai utilizată proiecţie ortografică axonometrică este *proiecţia izometrică*, pentru care direcţia de proiecţie (care coincide cu normala la planul de proiecţie) face unghiuri egale cu cele trei axe ale sistemului de referinţă. In cazul acestei proiecţii cei trei factori de scară pentru măsurarea lungimilor sunt egali (`iso` semnifică egal, iar `metric` indică o măsură), iar axele principale se proiectează pe plan în trei drepte care fac unghiuri egale (figura 2.21.b). Există doar opt direcţii de proiecţie care permit proiecţii izometrice. In figura 2.21.a este prezentată o astfel de proiecţie a unui cub.

Intr-o proiecţie *dimetrică* (figura 2.22), vectorul normal la planul de proiecţie (direcţia de proiecţie) face unghiuri egale cu două axe dintre axele sistemului de coordonate, spre deosebire de o proiecţie izometrică pentru care toate cele trei unghiurile sunt egale. Intr-o proiecţie *trimetrică* unghiurile dintre vectorul normal şi cele trei axe sunt diferite. Astfel proiecţia izometrică este un caz particular de proiecţie dimetrică, iar proiecţia dimetrică este un caz particular al proiecţiei trimetrice.

O proiecţie axonometrică se poate obţine prin transformarea obiectelor folosind rotaţii şi translaţii urmată de o proiecţie ortografică. Astfel, o proiecţie trimetrică se obţine prin una sau mai multe rotaţii, într-o ordine arbitrară, în jurul axelor sistemului de coordonate, urmate de o proiecţie în planul $z$. Dacă se consideră rotaţia de unghi $\varphi$ în jurul axei $y$, rotaţia de unghi $\theta$ în jurul axei $x$ şi apoi proiecţia ortografică în planul $z$, proiecţiile izometrice se obţin pentru

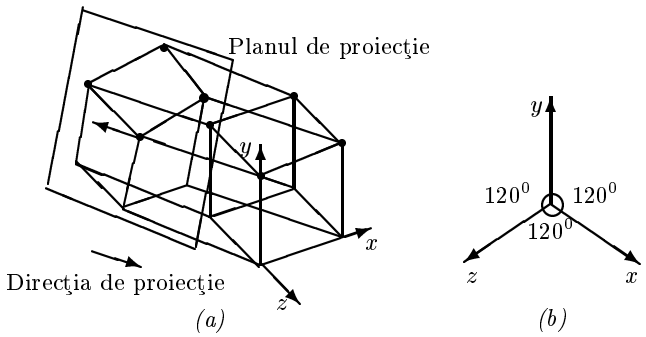



Figura 2.21: Proiecţii izometrice (a) Construirea proiecţiei unui cub în direcţia $(1,-1,-1)$ (b) Proiecţia izometrică a versorilor în direcţia $(1,1,1)$

$\varphi = \arcsin(1/\sqrt{3}) = \pm 35.26°$, $\theta = \arcsin(1/2) = \pm 45°$.

### Proiecţia oblică

Este obţinută prin proiectarea unui obiect de-a lungul unor linii paralele care nu sunt perpendiculare pe planul de proiecţie. Proiecţia unei feţe a obiectului, paralelă cu planul de proiecţie, permite măsurarea corectă a unghiurilor şi distanţelor, celelalte proiecţii ale feţelor permiţând doar măsurarea distanţelor.

O proiecţie oblică pe planul $xOy$ este caracterizată prin punctul în care este proiectat $M(0,0,1)$, distanţa $r$ de la origine a noului punct şi unghiul $\alpha$ dintre raza $r$ şi $Ox$ (figura 2.23.a). Funcţie de aceste valori se pot exprima noile coordonate $(x', y', z')$ ale proiecţiei oblice pe planul $xOy$ a unui punct oarecare $(x, y, z)$ (vezi figura 2.23.b):

$$\begin{cases} x' = x + zr\cos\alpha, \\ y' = y + zr\sin\alpha, \\ z' = 0, \end{cases}$$

adică în coordonate omogene,

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & r\cos\alpha & 0 \\ 0 & 1 & r\sin\alpha & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

Proiecţia ortogonală pe $xOy$ se obţine când $r = 0$.

Cele mai frecvente proiecţii oblice sunt:

(a) proiecţia cavalieră, pentru care $r = 1$;

(b) proiecţia cabinet, pentru care $r = 1/2$.

In *proiecţia cavalieră* direcţia de proiecţie face un unghi de $45^0$ cu planul de proiecţie, astfel încât proiecţia unui segment de dreaptă perpendicular pe planul de proiecţie are aceeaşi lungime ca şi segmentul însuşi (figura 2.24.a).

In *proiecţia cabinet* direcţia de proiecţie face un unghi de $\text{arctg}(2) \approx 63.4^0$ cu planul de proiecţie, astfel încât segmentele perpendiculare pe planul de proiecţie se proiectează la $1/2$ din lungimea lor reală (figura 2.24.b).

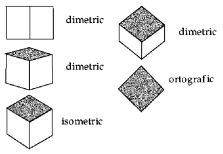



Figura 2.22: Proiecţii dimetrice ale unui cub

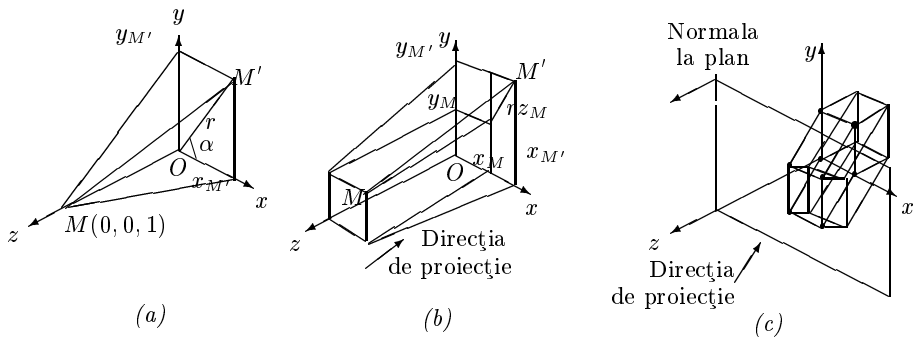



Figura 2.23: Proiecţia oblică (a) Construirea proiecţiei punctului $M(0,0,1)$ pe planul $xOy$ (b) Proiecţia unui punct oarecare pe planul $xOy$ (c) Proiecţia unui cub pe un plan paralel cu axa $y$

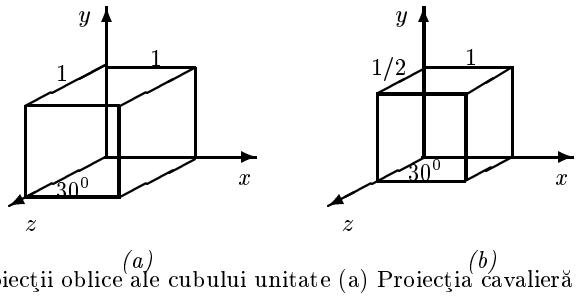



Figura 2.24: Proiecţii oblice ale cubului unitate (a) Proiecţia cavalieră cu direcţia de proiecţie $(\sqrt{3}/2, 1/2, -1)$ (b) Proiecţia cabinet cu direcţia de proiecţie $(\sqrt{3}/4, 1/4, -1)$

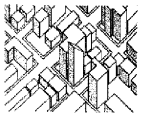


Figura 2.25: Plan oblic a unui oraş

Dintre proiecţiile oblice, proiecţia cabinet oferă imaginile cele mai realiste (figura 2.25).

### 2.4.4 Transformări proiective

Se determină în cele ce urmează matricea $4 \times 4$ caracteristică unei transformări proiective oarecare. Presupunem că în proiecţia perspectivă planul de proiecţie este perpendicular pe axa $z$, iar în proiecţia paralelă, planul de proiecţie este $z = 0$ (planul $xOy$). Fie $P(x, y, z)$ un punct oarecare al spaţiului şi $P'(x', y', z')$ proiecţia acestuia.

Se consideră următoarele trei cazuri:

1. *Centrul de proiecţie perspectivă se află în originea sistemului, iar planul de proiecţie este* $z = d$.

Din asemănarea triunghiurilor din figura 2.26 (b) şi (c) rezultă

$$\frac{x'}{d} = \frac{x}{z}, \quad \frac{y'}{d} = \frac{y}{z},$$

adică $x' = xd/z,\ y' = yd/z,\ z' = d$. Astfel, divizarea cu $z$ a coordonatelor în proiecţia perspectivă explică faptul că obiectele mai îndepărtate apar mai mici decât obiectele mai apropiate de centrul de proiecţie. Matricea transformării este

$$M_{pers} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}.$$

Multiplicând $(x, y, z, 1)^T$ cu $M_{pers}$ se obţine punctul omogen $(X, Y, Z, W)^T = (x, y, z, z/d)^T$. Coordonatele carteziene asociate punctului omogen se obţin prin diviziune cu $W$, adică $(x', y', z') = (X/W, Y/W, Z/W) = (xd/z, yd/z, d)$.

2. *Centrul de proiecţie se află la* $z = -d$, *iar planul de proiecţie este* $z = 0$.

Din asemănarea triunghiurilor din figura 2.27 se obţine

$$\frac{x'}{d} = \frac{x}{z+d}, \quad \frac{y'}{d} = \frac{y}{z+d},$$

adică $x' = x/(z/d + 1),\ y' = y/(z/d + 1),\ z' = 0$. Matricea transformării este

$$M'_{pers} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix}.$$

Spre deosebire de cazul anterior, această formulare permite includerea proiecţiilor paralele: dacă $1/d \to 0$, se obţine proiecţia ortografică pe planul $z = 0$, de ecuaţii $x' = x,\ y' = y,\ z' = 0$.

3. *Planul de proiecţie este* $z = z_p$, *iar centrul de proiecţie se află la distanţa* $Q$ *de punctul* $(0, 0, z_p)$ *în direcţia vectorului normalizat* $(dx, dy, dz)$ (figura 2.28). Dreapta care uneşte centrul de proiecţie $C_p$ şi punctul $P$

poate fi descrisă parametric: $C_p + t(P - C_p),\ t \in R$. Punctul $P'$ se află pe această dreaptă. Cum $C_p = (0, 0, z_p) + Q(dx, dy, dz)$,

$$\begin{cases} x' = Qdx + t(x - Qdx), \\ y' = Qdy + t(y - Qdy), \\ z' = (z_p + Qdz) + t[z - (z_p + Qdz)]. \end{cases}$$

Cum $P'$ se află în planul $z' = z_p$, se obţine $t$:

$$t = \frac{z_p - (z_p + Qdz)}{z - (z_p + Qdz)} = \frac{1}{\frac{z_p - z}{Qdz} + 1}.$$

Astfel

$$x' = Qdx + \frac{x - Qdx}{\frac{z_p - z}{Qdz} + 1} = \frac{x - z\frac{dx}{dz} + z_p\frac{dx}{dz}}{\frac{z_p - z}{Qdz} + 1},$$

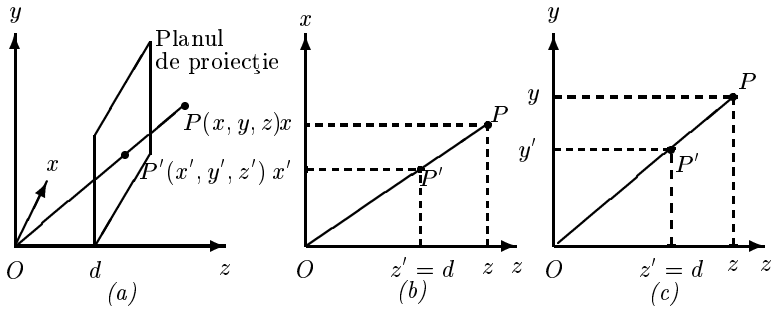



Figura 2.26: Cazul centrului de proiecţie in originea sistemului şi al planului de proiecţie $z = d$ (a) Construcţia proiecţiei (b) Proiecţie pe planul $xOz$ (c) Proiecţie pe planul $yOz$

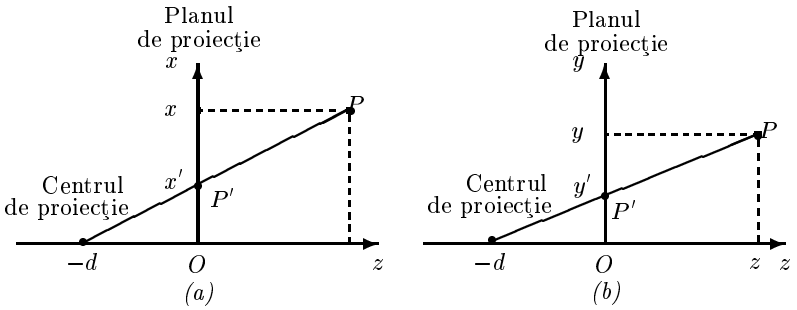



Figura 2.27: Cazul centrului de proiecţie la $z = -d$ şi al planului de proiecţie $z = 0$ (a) Proiecţie pe planul $xOz$ (c) Proiecţie pe planul $yOz$

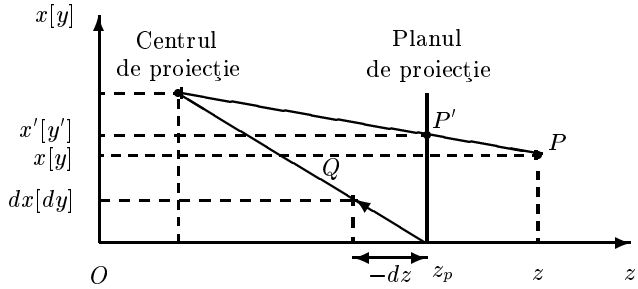


Figura 2.28: Cazul centrului de proiecţie la distanţa $Q$ de $(0,0,z_p)$ în direcţia $(dx,dy,dz)$ şi planul de proiecţie $z=z_p$

$$y'=\frac{y-z\frac{dy}{dz}+z_p\frac{dy}{dz}}{\frac{z_p-z}{Qdz}+1},\ z'=z_p=z_p\frac{\frac{z_p-z}{Qdz}+1}{\frac{z_p-z}{Qdz}+1}=\frac{-z\frac{z_p}{Qdz}+\frac{z_p^2+z_pQdz}{Qdz}}{\frac{z_p-z}{Qdz}+1}.$$

Matricea transformării este constituită astfel încât ultima linie înmulţită cu $(x,y,z,1)^T$ să producă coordonata omogenă $w$ egală cu numitorul comun al fracţiilor de mai sus:

$$M_{general}=\begin{pmatrix}1&0&-\frac{dx}{dz}&z_p\frac{dx}{dz}\\0&1&-\frac{dy}{dz}&z_p\frac{dy}{dz}\\0&0&-\frac{z_p}{Qdz}&\frac{z_p^2}{Qdz}+z_p\\0&0&-\frac{1}{Qdz}&\frac{z_p}{Qdz}+1\end{pmatrix}.$$

Matricele cazurilor anterioare se pot regăsi în această reprezentare. Astfel,

- pentru $M_{pers}$, $z_p=d,\ Q=d,\ (dx,dy,dz)=(0,0,-1)$;
- pentru $M'_{pers}$, $z_p=0,\ Q=d,\ (dx,dy,dz)=(0,0,-1)$;
- pentru proiecţia ortogonală pe $z=0,\ z_p=0,\ Q=\infty,\ (dx,dy,dz)=(0,0,-1)$;
- pentru proiecţia cavalieră pe $z=0,\ z_p=0,\ Q=\infty,\ (dx,dy,dz)=(\cos\alpha,\sin\alpha,-1)$;
- pentru proiecţia cabinet pe $z=0,\ z_p=0,\ Q=\infty,\ (dx,dy,dz)=(1/2\cos\alpha,1/2\sin\alpha,-1)$.

Când $Q\neq\infty$, $M_{general}$ defineşte o proiecţie perspectivă cu un punct de fugă. Punctul de fugă se obţine multiplicând punctul de la infinit de pe axa $z$, reprezentat în coordonate omogene prin $(0,0,1,0)^T$, cu $M_{general}$. Se obţine $x'=Qdx,\ y=Qdy,\ z'=z_p$.

### 2.4.5 Descrierea sistemului de vizualizare

Pentru definirea unei proiecţii s-au convenit o serie de notaţii, specificate în cele ce urmează.

Planul de proiecţie este definit printr-un punct de referinţă al planului $P_r$ şi normala la plan, vectorul $n$.

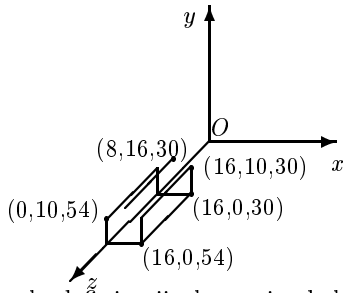



Figura 2.29: Coordonatele definitorii ale unui schelet de casă ce urmează a fi proiectat

Se construieşte sistemul de vedere de referinţă (asociat planului de proiecţie şi normalei la acesta) astfel:

(a) originea sistemului este $P_r$;
(b) una din axe este $n$;
(c) o altă axă este vectorul $v$, care este proiecţia pe planul de proiecţie a verticalei imaginii (de obicei, proiecţia axei $y$ din sistemul lumii reale);
(d) a treia axă, $u$, este determinată astfel încât $u \times v = n$.

Planul de proiecţie este astfel $uP_rv$.

Fereastra dreptunghiulară din planul de proiecţie este definită relativ la axele de coordonate $u$ şi $v$, nu neapărat simetric faţă de $P_r$. Se notează centrul ferestrei cu $C_f$.

Centrul de proiecţie $C_p$ şi direcţia de proiecţie $D_p$ sunt definite printr-un punct de referinţă al proiecţiei $P_d$ şi un indicator al tipului de proiecţie. Dacă proiecţia este perspectivă, $C_p = P_d$. Dacă proiecţia este paralelă, $D_p$ este definită de vectorul cu originea în $P_d$ şi capătul în $C_f$.

Specificările *clasice* ale proiecţiilor în notaţiile de mai sus sunt următoarele:

1. $P_r$= originea sistemului de coordonate ale lumii reale, $xyz$;
2. $n$= axa $z$;
3. $v$= axa $y$;
4. fereastra: $[0,1] \times [0,1]$ în planul $uP_rv$;
5. $P_d$=$(0.5, 0.5, 1)$ în sistemul $uvn$.

De exemplu, parametrii proiecţiei paralele pe $xOy$ sunt

$$P_r = (0,0,0)\ (xyz), \quad n = (0,0,1)\ (xyz), \quad v = (0,1,0)\ (xyz),$$

$$P_d = (0.5, 0.5, 1)\ (uvn),\ \text{Fereastra} = (0,1,0,1)\ (uvn),\ \text{Proiecţie: paralelă}$$

Se consideră exemplul casei din figura 2.29. O proiecţie perspectivă pe un plan ce taie axa $z$ este prezentată în figura 2.30. Proiecţia paralelă pe planul $z = 0$ produce imaginea din figura 2.31. In figura 2.32 sunt trasate alte trei proiecţii ale casei.

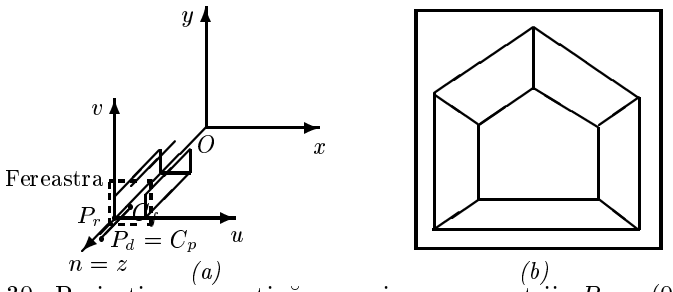



Figura 2.30: Proiecţia perspectivă a casei cu parametrii: $P_r = (0, 0, 54)$, $n = (0, 0, 1)$, $v = (0, 1, 0)$, $P_d = (8, 6, 30)$, Fereastra=$(-1, 17, -1, 17)$, Proiecţie: perspectivă (a) Elementele caracteristice proiecţiei (b) Imaginea rezultată

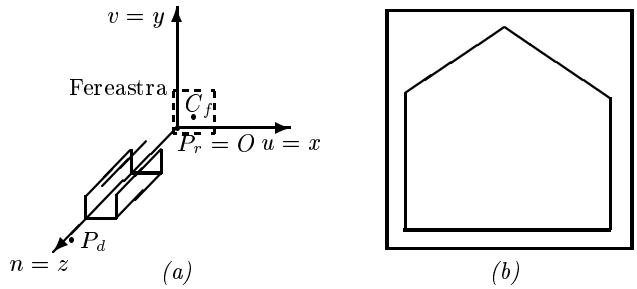



Figura 2.31: Proiecţia paralelă a casei cu parametrii: $P_r = (0, 0, 0)$, $n = (0, 0, 1)$, $v = (0, 1, 0)$, $P_d = (8, 8, 100)$, Fereastra=$(-1, 17, -1, 17)$, Proiecţie: paralelă (a) Elementele caracteristice proiecţiei (b) Imaginea rezultată

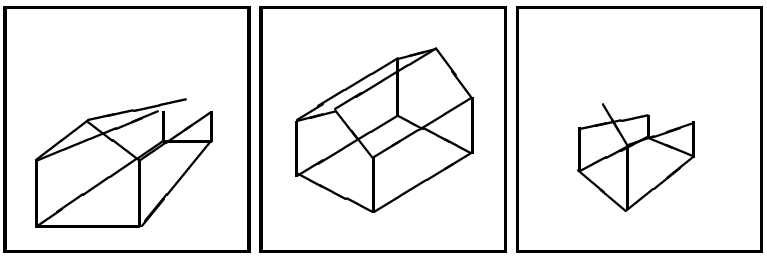



Figura 2.32: Proiecţii cu parametrii (a) $P_r = (16, 0, 54)$, $n = (0, 0, 1)$, $v = (0, 1, 0)$, $P_d = (20, 25, 20)$, Fereastra=$(-20, 20, -5, 35)$, Proiecţie: perspectivă (cu un punct de fugă) (b) $P_r = (8, 8, 42)$, $n = (1, 1, 1)$, $v = (0, 1, 0)$, $P_d = (0, 0, 10)$, Fereastra=$(-20, 20, -20, 20)$, Proiecţie: paralelă (izometrică) (c) $P_r = (16, 0, 54)$, $n = (1, 0, 1)$, $v = (0, 1, 0)$, $P_d$ =(0,25,20$\sqrt{2}$), Fereastra=$(-20, 20, -5, 35)$, Proiecţie: perspectivă (cu două puncte de fugă)

## 2.5 Volumul de vedere

Ochiul uman poate observa toate obiectele situate în interiorul unui con de vedere. Directoarea acestui con situată într-un plan normal pe direcţia de observare este eliptică. In aplicaţiile grafice se înlocuieşte conul de vedere cu o piramidă de vedere (figura 2.33).

Astfel, se poate imagina că observatorul priveşte lumea printr-o fereastră dreptunghiulară decupată într-un plan opac, situată la o anumită distanţă de observator.

In vizualizarea 3D trebuie specificate un *volum de vedere*, o *proiecţie* pe un *plan*, o *fereastră* în acel plan şi o *zonă de lucru* pe suprafaţa de vizualizare. Conţinutul volumului de vedere este proiectat în fereastra din planul de proiecţie şi apoi este transferat în zona de lucru. Figura 2.34 schiţează acest model conceptual de vizualizare 3D.

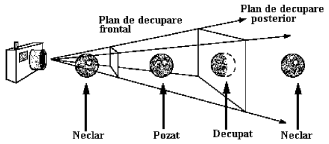



Figura 2.33: Volumul de vizualizare în cazul unui aparat de fotografiat

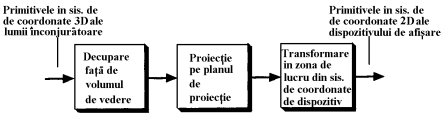



Figura 2.34: Modelul conceptual al procesului de vizualizare tridimensională

*Volumul de vedere* (sau *volum de vizualizare*) mărgineşte acea porţiune din spaţiul lumii reale care va fi proiectat pe planul de proiecţie. Este definit astfel:

1. în proiecţia perspectivă, ca o piramidă semi-infinită cu vârful în centrul de proiecţie şi cu laturile trecând prin colţurile ferestrei din planul de proiecţie;
2. în proiecţia paralelă, ca un paralelipiped infinit cu laturile paralele cu direcţia de proiecţie.

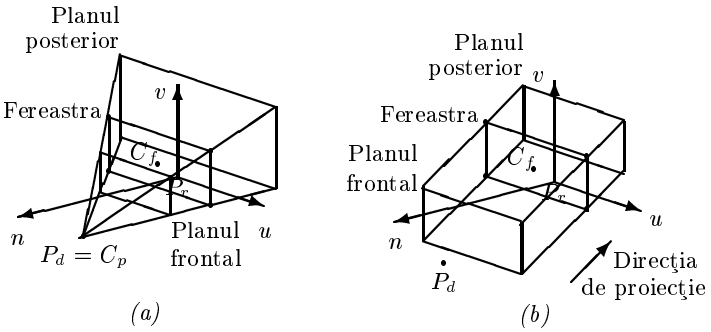



Figura 2.35: Volumul de vedere (a) Cazul proiecţiei perspective (b) Cazul proiecţiei paralele

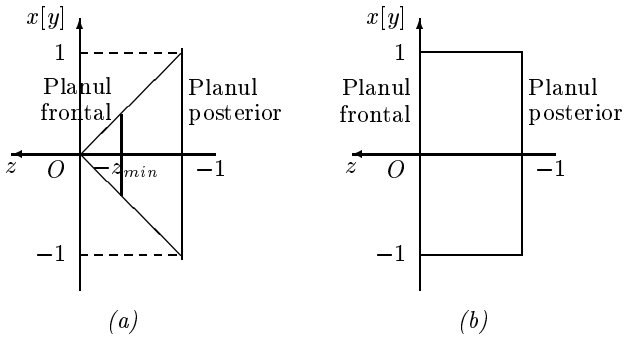



Figura 2.36: Secţiuni ale volumului de vedere canonic (a) Cazul proiecţiei perspective (b) Cazul proiecţiei paralele

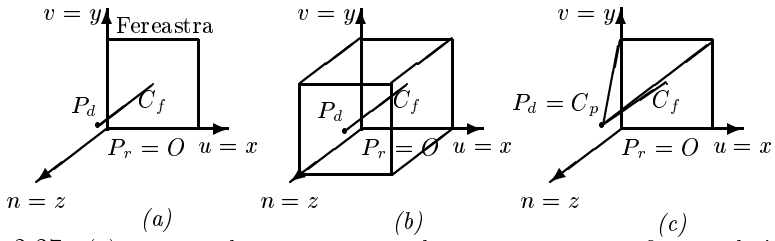



Figura 2.37: (a) Elementele caracteristice ale proiecţiei în specificarea clasică (b) Volumul de vedere în proiecţia paralelă (c) Volumul de vedere în proiecţia perspectivă

Pentru a limita numărul de primitive grafice proiectate în planul de proiecţie, aceste volume sunt transformate în corpuri „finite" prin introducerea a două plane de limitare, paralele cu planul de proiecţie, în faţa şi în spatele acestuia, aflate la distanţe bine definite de-a lungul normalei la planul de proiecţie (figura 2.35).

Se obţine un trunchi de piramidă, respectiv un paralelipiped.

In urma transformării de *normalizare* a volumului de vedere, planele ce definesc frontiera noului volum, numit *volum de vedere canonic*, sunt:

1. pentru proiecţia paralelă, $x = -1,\ x = 1,\ y = -1,\ y = 1,\ z = 0,\ z = -1$ (figura 2.36.a);
2. pentru proiecţia perspectivă, $x = z,\ x = -z,\ y = z,\ y = -z,\ z = -z_{min},\ z = -1$ (figura 2.36.b).

Volumele de vedere asociate specificării clasice sunt prezentate în figura 2.37.

**Transformarea unui volum de vedere al unei în proiecţii paralele în volum de vedere canonic**

Seria de transformări este următoarea (figura 2.38):

1. translatarea punctului de referinţă $P_r$ în origine;
2. rotirea sistemului de referinţă astfel încât axa $n$ devine axa $z$, $u$ devine $x$ şi axa $v$ devine $y$;
3. înclinare astfel încât direcţia de proiecţie să devină paralelă cu axa $z$;
4. translaţie şi scalare pentru obţinerea volumului canonic.

Pasul 1 corespunde unei translaţii $T = T(-P_r)$. Pentru pasul 2 se utilizează proprietatea matricelor ortogonale discutată în secţiunile anterioare. Fie $R$ matricea de rotaţie. Cum $n$ este rotit pe $z$,

$$R_z = \frac{n}{\|n\|}.$$

Vectorul $v$ este proiecţia pe planul de proiecţie a verticalei imaginii, notată $V$ (de obicei axa $y$). Axa $u$ este perpendiculară pe $V$ şi $n$ şi deci poate fi obţinută din $V \times R_z$; fiind rotită pe axa $x$,

$$R_x = \frac{V \times R_z}{\|V \times R_z\|}.$$

Axa $v$ este perpendiculară pe $R_z$ şi $R_x$ şi este rotită pe $y$, astfel încât

$$R_y = R_z \times R_x.$$

Matricea de rotaţie din pasul 2 este astfel

$$R = \begin{pmatrix} R_x^T & 0 \\ R_y^T & 0 \\ R_z^T & 0 \\ 0_3 & 1 \end{pmatrix}, \quad 0_3 = (0,0,0).$$

Al treilea pas presupune înclinarea volumului de vedere de-a lungul axei $z$ astfel încât planele sale să fie normale la una din axele sistemului de coordonate. Direcţia de proiecţie $D_p$, în urma acestei transformări, trebuie să coincidă cu axa $z$. $D_p$ este definită de vectorul de la $P_d$ la centrul ferestrei $C_f$, iar $P_d$ (transformat) este specificat în coordonatele sistemului $uvn$. Cum primele două transformări au pus în corespondenţă sistemul $uvn$ cu sistemul de coordonate

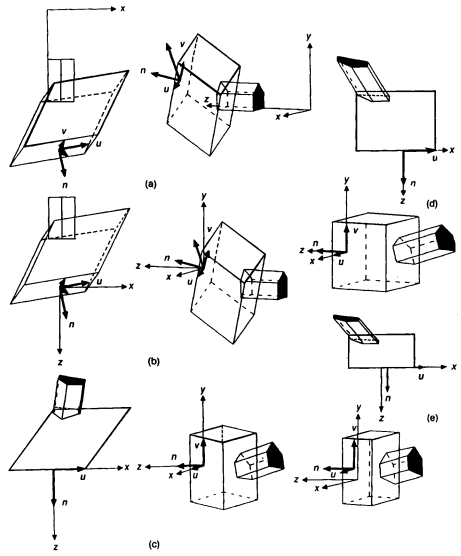


Figura 2.38: Rezultate ale etapelor transformării în volum canonic al proiecţiei paralele (a) Sistemul de vizualizare iniţial (b) $P_r$ translatat în origine (c) Rotirea sistemului de referinţă $uvn$ (d) Inclinarea volumului de vedere (e) Translaţie şi scalare

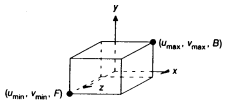



Figura 2.39: Volumul de vedere după transformările de la paşii 1, 2 şi 3

a lumii înconjurătoare $xyz$, $P_d$ este de asemenea în sistemul de coordonate a lumii înconjurătoare. Atunci $D_p = C_f - P_d$. Fie $D_p = (d_x, d_y, d_z, 0)$, $C_f = ((u_{max} + u_{min})/2, (v_{max} + v_{min})/2, 0, 1)$, $P_d = (p_u, p_v, p_n, 1)$. Se caută în pasul 3 matricea transformării de înclinare

$$H = \begin{pmatrix} 1 & 0 & h_x & 0 \\ 0 & 1 & h_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

pentru care noua direcţie de proiecţie coincide cu axa $z$, adică $(0, 0, d_z, 0)^T = H \cdot D_p$. Ţinând seama de forma matricei $H$, obţinem

$$h_x = -\frac{d_x}{d_z}, \quad h_y = -\frac{d_y}{d_z}.$$

Dacă proiecţia este ortografică, atunci $d_x = d_y = 0$ astfel încât şi $h_x = h_y$, iar $H$ este matricea identitate. Volumul de vedere după aceste trei transformări este delimitat de

$$u_{min} \leq x \leq u_{max}, \quad v_{min} \leq y \leq v_{max}, \; B \leq z \leq F$$

unde $F$ şi $B$ sunt distanţe de la $P_r$ de-a lungul lui $n$ la planele de decupare frontal şi posterior (figura 2.39). Ultimul pas constă în transformarea acetui volum în volum canonic. Acest pas presupune translatarea centrului planului frontal al volumului de vedere în origine şi scalarea la dimensiunile $2 \times 2 \times 1$ a volumului canonic:

$$T_{par} = T\left(-\frac{u_{max} + u_{min}}{2}, -\frac{v_{max} + v_{min}}{2}, -F\right)$$

$$S_{par} = S\left(\frac{2}{u_{max} - u_{min}}, \frac{2}{v_{max} - v_{min}}, \frac{1}{F - B}\right)$$

Transformarea căutată este deci

$$N_{par} = S_{par} \cdot T_{par} \cdot H \cdot R \cdot T.$$

$N_{par}$ transformă un volum arbitrar de vedere al unei proiecţii paralele, în volum de vedere canonic, permiţând astfel decuparea primitivelor faţă de volumul canonic.

**Transformarea unui volum de vedere al unei proiecţii perspective în volum de vedere canonic**

Seria de transformări este următoarea (figura 2.40):

1. translatarea punctului de referinţă $P_r$ în origine;
2. rotirea sistemul de referinţă $uvn$ astfel încât axa $n$ să devină axa $z$, axa $u$ să devină $x$, iar $v$ să devină $y$;
3. translatarea centrul de proiecţie $C_p = P_d$ în origine;
4. înclinarea conform căreia linia centrală a volumului de vedere devine axa $z$;
5. scalare astfel încât volumul de vedere devine volum canonic.

Paşii 1 şi 2 sunt identici cu cei din cazul proiecţiei paralele: $R \cdot T$. Pasul 3 presupune o translatare $T_{per} = T(-P_d)$. Linia centrală a volumului de vedere, care trece prin origine şi centrul ferestrei nu coincide cu axa $z$. Scopul pasului 4 este transformarea acestei linii în axa $z$ (figura 2.41). Linia centrală trece de la $P_d$ la $C_f$, direcţia fiind aceeaşi ca în proiecţia paralelă. Astfel matricea de înclinare este aceeaşi ca în cazul proiecţiei paralele. După ce înclinarea este aplicată, fereastra (şi deci şi volumul de vedere) este centrată pe axa $z$. Marginile ferestrei sunt definite de

$$-\frac{u_{max} - u_{min}}{2} \leq x \leq \frac{u_{max} - u_{min}}{2}, \quad -\frac{v_{max} - v_{min}}{2} \leq y \leq \frac{v_{max} - v_{min}}{2}.$$

$P_r$ care înainte de pasul 3 a fost în origine, este translatat prin pasul 3 şi înclinat prin pasul 4, astfel încât după transformare se află la $H \cdot T(-P_d) \cdot (0,0,0,1)^T$, componenta $z$ a acestuia fiind $-p_n$, a 3-a coordonată a lui $-P_d$, deoarece înclinarea nu afectează coordonatele $z$. Pasul final constă în scalarea de-a lungul celor trei axe pentru a crea volumul canonic (figura 2.42). Primul subpas presupune scalarea pe $x$ şi $y$, diferenţiat, pentru a obţine pantele corespunzătoare volumului de vedere canonic; în plus se doreşte ca volumul rezultat să aibă semi-înălţimea şi semi-lăţimea egale cu $p_n$. Factorii de scală sunt $2p_n/(u_{max} - u_{min})$ şi $2p_n/(v_{max} - v_{min})$. In al doilea subpas, se aplică o scalare uniformă de-a lungul celor trei axe (pentru menţimerea pantelor) astfel incât planul $z = -p_n + B$ să devină planul $z = -1$. Factorul de scală este $-1/(-p_n + B)$. Astfel

$$S_{per} = S\left(\frac{-2p_n}{(u_{max} - u_{min})(-p_n + B)}, \frac{-2p_n}{(v_{max} - v_{min})(-p_n + B)}, \frac{-1}{-p_n + B}\right).$$

Transformarea căutată este

$$N_{per} = S_{per} \cdot H \cdot T_{per} \cdot R \cdot T.$$

*Observaţie:* $N_{per}$ şi $N_{par}$ nu afectează coordonata omogenă $w$ a punctelor transformate, astfel încât împărţirea cu $w$ nu este în mod normal necesară. Astfel coordonatele omogene obţinute prin trecerea la volumul de vedere canonic pot fi uşor transformate din nou în spaţiul tridimensional, după care se poate aplica decuparea faţă de volumul de vedere canonic şi în final matricea de proiecţie $M_{general}$.

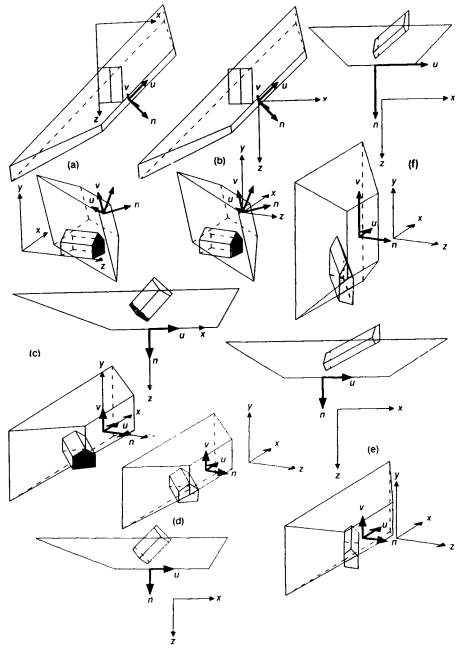



Figura 2.40: Transformarea în volum canonic al proiecţiei perspective: (a) sist. vizualizare iniţial (b) $P_r$ translatat în origine (c) $uvn$ este aliniat la $xyz$ (d) $C_p$ translatat în origine (e) înclinare volum (f) scalare volum

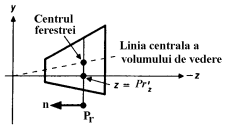



Figura 2.41: Secţiune prin volumul de vedere după transformările paşilor 1, 2 şi 3

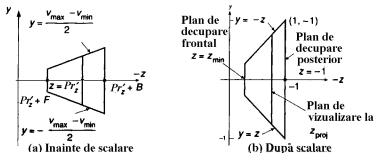



Figura 2.42: Secţiune a volumului de vedere înainte şi după scalare

### Transformarea volumului canonic al proiecţiei perspective în volumul canonic al proiecţiei paralele

Volumul de vedere canonic din proiecţia perspectivă poate fi transformat în volumul de vedere canonic din proiecţia paralelă astfel: unui punct $(x, y, z)$ din interiorul trunchiului de piramidă îi corespunde un punct $(x', y', z')$ din interiorul paralelipipedului prin relaţiile

$$x' = -\frac{x}{z}, \quad y' = -\frac{y}{z}, \quad z' = \frac{z_{min} - z}{z_{min} + 1}.$$

Matricea corespunzătoare

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1+z_{min}} & \frac{-z_{min}}{1+z_{min}} \\ 0 & 0 & -1 & 0 \end{pmatrix}, \quad z_{min} = -1,$$

indică o transformare perspectivă. Astfel, pentru construirea imaginii prin proiecţie perspectivă a unui corp 3D se poate proceda în două moduri:

(a) se proiectează fiecare punct component, noile componente fiind stocate separat;

(b) se aplică corpului transformarea perspectivă care îl deformează astfel încât proiecţia paralelă a corpului obţinut să coincidă cu proiecţia perspectivă a obiectului iniţial. In acest fel imaginea va fi stocată în memorie

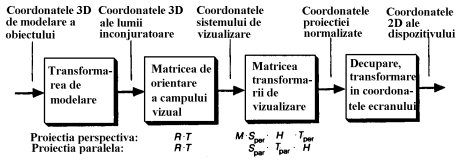



Figura 2.43: Sistemele de coordonate utilizate la vizualizare şi relaţiile între ele

prin chiar coordonatele $x$ şi $y$ ale punctelor implicate. Se proiectează ortogonal corpul obţinut.

### Vizualizare 3D

Setul de transformări, în cazul general, care se aplică obiectelor pentru *vizualizare* este următorul:

1. extinderea coordonatelor 3D la coordonate omogene;
2. aplicarea transformării de normalizare $N_{par}$ sau $N_{per}$;
3. revenire la coordonate carteziene în 3D;
4. decupare în 3D faţă de volumul de vedere canonic;
5. extinderea coordonatelor 3D la coordonate omogene;
6. aplicarea proiecţiei paralele (în urma transformărilor aplicate mai sus este vorba de o proiecţie ortografică pe planul $xOy$) sau aplicarea proiecţiei perspective (cazul 1 în care centrul de proiecţie este în originea sistemului, iar planul de proiecţie este $z = 1$);
7. translatarea şi scalarea pentru trecerea la coordonatele dispozitivului;
8. trecerea la coordonate carteziene în 2D.

Obiectele sunt definite în *sistemul de coordonate ale obiectului* (numit şi *sistemul de coordonate de modelare*, sau *sistemul de coordonate locale*). Obiectele sunt transformate în *sistemul de coordonate a lumii înconjurătoare* (sau sistemul de coordonate a problemei, sau sistemul de coordonate a aplicaţiei), sistemul în care o scenă cu mai multe obiecte este reprezentată pe calculator, prin *transformarea de modelare*. Sistemul de referinţă a coordonatelor de vizualizare, sistemul *uvn*, este utilizat pentru definirea unui *volum de vedere*. Când originea acestui sistem este plasată în centrul de proiecţie şi planul de proiecţie este normal pe axa $z$, sistemul se mai numeşte şi *sistem de coordonate a ochiului* sau *sistem de coordonate ale camerei* (cu subînţeles de cameră foto sau de filmare). Primii trei paşi din transformarea de normalizare a volumului de vedere realizează conversia dintre sistemul de coordnate ale lumii înconjurătoare în sistemul de coordonate ale camerei (figura 2.43). Din sistemul de coordonate ale camerei se trece la sistemul normalizat de coordonate a proiecţiei (sistemul de coordonate ecran 3D, sau sistemul de coordonate 3D a unui dispozitiv logic), adică *sistemul de coordonate a volumului de vedere*

*canonic*. Termenul de normalizat se referă la faptul că toate valorile coordonatelor sunt în intervalul [0,1] sau [-1,1], pe când termenul logic se referă la valori de coordonate într-un interval mai larg, de exemplu [0,1023] (în acest caz sistemul nu este normalizat). Prin aplicarea proiecţiei de la 3D la 2D se crează *sistemul normalizat de coordonate 2D ale dispozitivului* (sau sistemul de coordonate a imaginii, sistem de coordonate rastru, sistemul de coordonate ecran, *sistem de coordonate ale dispozitivului fizic* – în contrast cu termenul de logic utilizat anterior).

# 3. Trasarea primitivelor grafice

## 3.1 Primitive grafice

*Se numesc primitive grafice acele elemente de bază pe care programatorul le poate folosi pentru a realiza desenele necesare unei anumite aplicaţii.* Primitivele grafice ale unei aplicaţii grafice pot fi extrem de variate: puncte, segmente de dreaptă, caractere, dreptunghiuri, linii poligonale, poligoane, conice, curbe cubice, cuadrice, suprafeţe bicubice, cub, paralelipiped, etc.

Se consideră un dispozitiv de afişare grafică cu tub catodic cu rastru dreptunghiular, cu originea (0,0) în punctul din stânga jos al ecranului, cu valorile absciselor crescând la dreapta şi valorile ordonatelor crescând în sus. Unităţile axelor $u$, $v$ sunt egale cu distanţele dintre doi pixeli alăturaţi pe orizontală, respectiv verticală. Se presupune de asemenea că fiecare pixel de pe ecran are două stări posibile: aprins sau stins (imagine monocromă). Se neglijează, în acest capitol, celelalte atribute posibile pentru reprezentarea unei imagini pe ecran, cum sunt culoarea, nuanţa, intensitatea, etc.

## 3.2 Trasarea incrementală

O atenţie deosebită s-a acordat optimizării algoritmilor de trasare a primitivelor în ceea ce priveşte eficienţa de execuţie (viteza de generare a unei primitive). In acest scop se urmăreşte *reducerea numărului de operaţii în virgulă mobilă, în favoarea celor cu numere întregi şi minimizarea numărului de înmulţiri şi împărţiri* (microprocesoarele actuale nu conţin instrucţiuni pentru aritmetică în virgulă mobilă, astfel încât instrucţiunile de înmulţire şi împărţire sunt mai lente decât operaţiile de adunare şi scădere).

Metodele de trasare incrementală se bazează pe *scheme cu diferenţe* între mărimi asociate punctelor succesive, care conduc la relaţii liniare de transformare iterativă a unor variabile. Aceste relaţii liniare pot fi implementate în general prin operaţii simple de adunare şi scădere de numere întregi. Metodele incrementale îşi dovedesc eficienţa în special la trasarea curbelelor de grad unu (linii drepte) şi doi (cercuri, elipse, hiperbole, parabole).

Algoritmii de trasare incrementală a curbelor de grad cel mult doi pornesc de la anumiţi parametri care caracterizează analitic curba şi construiesc cea mai bună reprezentare discretizată a curbei respective. Se efectuează *deplasări elementare* între două puncte vecine pe orizontală, verticală sau diagonală, care se traduc prin operaţii simple de incrementare sau decrementare a *variabilelor de adresare fizică*. Algoritmul de trasare incrementală păstrează în permanenţă informaţii cu privire la starea curentă a procesului iterativ, prin intermediul uneia sau mai multor *variabile de stare*. Acestea permit realizarea cât mai simplă a funcţiilor algoritmului şi sunt uşor actualizabile în vederea trecerii la următorul pas iterativ.

Ţinând seama de faptul că modelul reprezentării este o curbă continuă

pentru care tangenta variază continuu, pe când spaţiul de afişare este discret, în dezvoltarea algoritmilor s-au adoptat următoarele conventii metodologice de reprezentare:

1. pentru fiecare punct generat la un capăt al curbei, sau la o margine a ferestrei de vizualizare, există exact un punct vecin pe orizontală, verticală sau diagonală, generat pe curbă.
2. orice alt punct generat pe curbă are exact două puncte vecine pe orizontală, verticală sau diagonală, generate pe curbă, cu care formează un unghi de $180^0$ sau $135^0$ (se evită unghiurile ascuţite în reprezentările discrete ale curbelor) pentru un raport aspectual unitar.

Aceste reguli sunt referite sub numele de *reguli de conexiune discretă*.

## 3.3 Segmente de linii

Pentru a trasa un segment de dreaptă trebuie cunoscute coordonatele capetelor acestuia. Deoarece un segment se trasează în timp, cele două puncte pot fi denumite extremitatea de început şi extremitatea de sfârşit a segmentului.

Pentru trasarea segmentului pe ecran, numită *conversie prin baleiaj* (baleiere, scanare) sau *conversie scan*, se determină care sunt pixelii rastrului cei mai apropiaţi de imaginea ideală a segmentului (care vor fi aprinşi). Segmentul obţinut va arăta ca o scăriţă (figura 3.1). Un pixel este reprezentat printr-un disc centrat în punctul de coordonate întregi $(u, v)$ (în sistemele actuale diametrul unui pixel circular este mai mare decât spaţiul interpixeli, deci reprezentarea din figură este exagerată). Segmentul de linie trasat are "lăţimea" de un pixel.

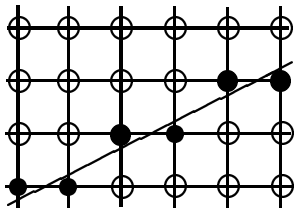


Figura 3.1: Un segment de dreaptă convertit prin baleiaj, pentru care pixelii aprinşi sunt reprezentaţi prin cercuri negre

### 3.3.1 Algoritmul incremental de bază

Algoritmul incremental de bază sau algoritmul `DDA (Digital Differential Analyzer)` determină poziţia pixelilor cei mai apropiaţi de un segment de linie, parcurgând paşi unitari pe o axă de coordonate şi calculând valorile corespunzătoare pe cealaltă axă. Fie $xOy$ sistemul de coordonate ale dispozitivului de afişare.

In planul ecranului, ecuaţia dreptei care trece prin $A$ şi $B$ este

$$y = y_A + m(x - x_A), \quad m = \frac{y_B - y_A}{x_B - x_A},$$

unde $m$ reprezintă panta dreptei. Se presupune că, coordonatele punctelor extreme, $x_i,\ y_i,\ i = A, B$, sunt numere întregi. Altfel, se rotunjesc la cel mai apropiat întreg.

Dacă $x_B > x_A$, cea mai simplă strategie de discretizare a liniilor este calcularea pantei $m$, incrementarea lui $x$ cu 1 pornind de la punctul $A$ din partea stângă, calcularea lui $y_i = mx_i + y_A - mx_A$ pentru fiecare $x_i$ şi aprinderea pixelului ($x_i$,`Round`($y_i$)), unde `Round`$(y_i) = [y_i + 0.5]$; se selectează astfel cel mai apropiat pixel de linia ideală, anume pixelul a cărui distanţă până la linia ideală este cea mai mică (figura 3.2.a). Se trece la următoarea valoare a lui $x$, până când $x$ depăşeşte pe $x_B$. Această strategie este ineficientă deoarece la fiecare iteraţie este necesară o multiplicare în virgulă mobilă, o adunare şi invocarea unei funcţii de tip `Floor` pentru partea întreagă. Putem elimina multiplicarea dacă incrementăm şi pe $y_i$ nu numai pe $x_i$.

Dacă la un moment dat intersecţia cu o verticală a grilei este $(x_i, y_i)$ şi se aprinde pixelul $(x_i, \texttt{Round}(y_i))$, atunci la pasul următor intersecţia cu următoarea verticală va fi $(x_{i+1}, y_{i+1})$, unde $x_{i+1} = x_i + 1$, iar

$$y_{i+1} = \texttt{Round}(y_A + m(x_{i+1} - x_A)) = \texttt{Round}(y_A + m(x_i + 1 - x_A)) = \texttt{Round}(y_i + m).$$

Se va aprinde pixelul $(x_{i+1}, y_{i+1})$.

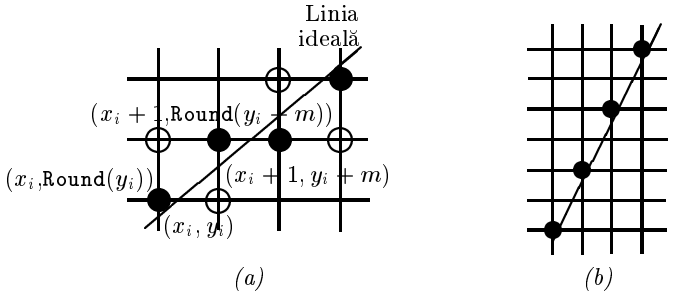



Figura 3.2: Algoritmul `DDA` (a) Calculul incremental pentru o pantă subunitară (b) Cazul incrementării unitare pe axa $x$ cu $|m| > 1$

In următoarea secvenţă de cod, `WritePixel` este o procedură de nivel scăzut oferită de software-ul de dispozitiv.

```
Procedure Line(        {Presupune că −1 ≤ m ≤ 1, x0 < x1}
  x0, y0,              {Punct stânga}
  x1, y1,              {Punct dreapta}
  value : integer)     {Valoarea dată pixelilor liniei}
var
   x:  integer;                {x variază între x0 şi x1}
   dy, dx, x, m : real;
begin
   dy := y1 − y0;
   dx := x1 − x0;
   y := y0;
   for x := x0 to x1 do
       begin
```

```
            {Setează val.pixel}
            WritePixel(x, Round(y), value);
            y := y + m
        end
    end; {Line}
```

Pentru a proceda uniform indiferent de relaţia de ordine dintre $x_A$ şi $x_B$, se calculează $s = \text{sgn}(x_B - x_A)$ şi formula de calcul a lui $y$ rămâne neschimbată, doar că $x$ se incrementează cu $s$ în loc de 1.

Folosind acest raţionament, segmentele mai apropiate de verticală decât de orizontală, ar apărea punctate (figura 3.2.b), deoarece panta $|m| > 1$, iar segmentele verticale nu ar putea fi reprezentate deoarece $x_B - x_A = 0$. Astfel se face distincţie în algoritm între segmentele apropiate de orizontală şi cele apropiate de verticală, pentru acestea din urmă calculându-se $x$ în funcţie de $y$ şi realizând incrementarea după $y$ cu $s = \text{sgn}(y_B - y_A)$ între $y_A$ şi $y_B$.

### 3.3.2 Algoritmul punctului de mijloc

In algoritmul `DDA` ponderea cea mai mare o ocupă în timp calculele în virgulă flotantă (funcţia `Round`). Din punct de vedere al timpului, mai eficienţi sunt algoritmii care elimină operaţiile în virgulă mobilă. Un exemplu tipic de algoritm în aritmetica numerelor întregi pentru trasarea segmentelor de dreaptă este algoritmul `Bresenham`.

Pentru exemplificare, se consideră cazul unui segment al cărui înclinare este cuprinsă între $0^0$ şi $45^0$ (panta $m \in (0, 1)$). Conform regulilor de conexiune discretă, cunoscând poziţia curentă a unui punct oarecare de pe segment, există două puncte posibile pentru următoarea poziţie (figura 3.3).

Se consideră că la pasul $i$ s-a determinat pixelul care trebuie aprins ca fiind $P(x_i, y_i)$. La pasul $i+1$ este necesară alegerea unui punct dintre $E(x_i+1, y_i+1)$ şi $F(x_i + 1, y_i)$. Criteriul de decizie pentru algoritm este apropierea punctului faţă de segmentul real. Practic se compară distanţele $e$ şi $f$ din figura 3.3. Dacă $f < e$, se alege $F$, altfel se alege $E$.

Pentru stabilirea criteriului decizional se parcurg următoarele etape:

Pas 0: Se compară $x_A$ cu $x_B$ şi eventual se face o schimbare între capetele segmentului astfel încât $x_A < x_B$. Se realizează o translaţie cu deplasamentul $(-x_A, -y_A)$ astfel încât ecuaţia dreptei modificate să fie $y = (dy/dx)x$, unde $dx := x_B - x_A,\ dy := y_B - y_A$. Se consideră $x_0 = 0,\ y_0 = 0$.

Pasul $i$: Punctul $M$ de intersecţie al dreptei $x = x_i + 1$ cu dreapta ideală are coordonatele $x_M = x_i + 1, y_M = (dy/dx)x_M$. Se determină distanţele la punctele din rastru cele mai apropiate pe verticala $x = x_i + 1$: $f = y_M - y_i,\ e = y_i + 1 - y_M$. Se determină cantitatea $dx(f - e)$:

$$dx(f - e) = dx(2y_M - 2y_i - 1) =$$

$$= dx\left[2\frac{dy}{dx}(x_i + 1) - 2y_i - 1\right] = 2dy(x_i + 1) - 2y_i dx - dx.$$

Deoarece $dx > 0$ (din pasul 0) cantitatea $f - e$ are acelaşi semn ca $dx(f - e)$. Se notează $d_{i+1} = dx(f - e)$ şi se caută o relaţie recursivă pentru

acest *factor de decizie*. Cum

$$d_{i+1} = 2(x_i dy - y_i dx) + 2dy - dx, \quad d_i = 2(x_{i-1} dy - y_{i-1} dx) + 2dy - dx,$$

rezultă

$$d_{i+1} - d_i = 2[dy(x_i - x_{i-1}) - dx(y_i - y_{i-1})]$$

şi fiindcă $x_i - x_{i-1} = 1$ se obţine relaţia recursivă

$$d_{i+1} = d_i + 2dy - 2dx(y_i - y_{i-1}),$$

cu valoarea de start

$$d_1 = 2(x_0 dy - y_0 dx) + 2dy - dx = 2dy - dx.$$

In concluzie, variabilele de stare sunt $(x_i, y_i, d_i)$, iar criteriul decizional este următorul:

dacă $d_{i+1} \geq 0$ se alege $E(x_i + 1, y_i + 1)$ şi $d_{i+2} = d_{i+1} + 2(dy - dx)$

dacă $d_{i+1} < 0$, atunci se alege $F(x_i + 1, y_i)$ şi $d_{i+2} = d_{i+1} + 2dy$.

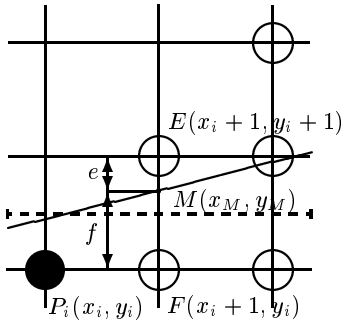



Figura 3.3: Selecţia pixelilor în algoritmul `Bresenham`

Algoritmul general presupune mai multe cazuri, funcţie de înclinarea dreptei care trebuie trasată. O procedură ce modelează cazul analizat este următoare:

```
Procedure MidpointLine (x0, y0, x1, y1, value: integer)
var
   dx, dy, incrE, incrF, d, x, y : integer;
begin
   dx := x1 − x0;
   dy := y1 − y0;
   d := 2 ∗ dy − dx;          {Val.iniţială variabilă decizie}
   incrE := 2 ∗ (dy − dx);    {Increment pentru mutarea la E}
   incrF := 2 ∗ dy;           {Increment pentru mutarea la F}
   x := x0;
   y := y0;
   WritePixel(x, y, value); {Pixel de start}
```

```
while x < x1 do
      begin
      if d < 0 then
         begin              {Alege F}
              d := d + incrF;
              x := x + 1
         end
      else
         begin              {Alege E}
              d := d + incrE;
              x := x + 1;
              y := y + 1;
         end;
      WritePixel (x, y, value) {Cel mai apropiat pixel}
      end {while}
end; {MidpointLine}
```

La fiecare pas, prin algoritm se alege un pixel din 2 posibili, alegerea bazându-se pe semnul unei variabile de decizie calculate în iteraţia anterioară, apoi se actualizează variabila de decizie adăugând o valoare care depinde de alegerea pixelului. La fiecare pas se efectuează o adunare pentru modificarea variabilei de decizie (fără vreo multiplicare consumatoare de timp).

Tehnica `Bresenham` poate fi aplicată şi pentru trasarea cercurilor dar nu poate fi generalizată pentru orice conică. O formulare diferită datorată lui `Pitteway` şi `Van Aken` (algoritmul punctului de mijloc pentru trasarea unei curbe plane) se reduce în cazul liniilor şi cercurilor la formularea `Bresenham`.

Variabila decizională din algoritmul `Bresenham` poate fi exprimată liniar funcţie de ecuaţia dreptei evaluată în punctul de mijloc al segmentului $EF$. Fie $D(x,y) = ax + by + c,\ a > 0$ ecuaţia dreptei. Variabila de decizie în *algoritmul punctului de mijloc* este $d_{i+1} = 2D(x_i + 1, y_i + \frac{1}{2})$. Cum $D(x,y)$ este negativ pentru un punct de deasupra liniei şi pozitiv pentru un punct sub linie, se alege $E$ dacă $d_{i+1} \geq 0$ şi $F$ dacă $d_{i+1} < 0$. Dacă este ales $F$, atunci

$$d_{i+2} = 2D(x_i + 2, y_i + 1/2) = 2[a(x_i + 2) + b(y_i + 1/2) + c] = d_{i+1} + 2a =$$

$$= d_{i+1} + 2dy,$$

iar dacă se alege $E$, atunci

$$d_{i+2} = 2D(x_i + 2, y_i + 3/2) = d_{i+1} + 2(a + b) = d_{i+1} + 2(dy - dx).$$

Valoarea de start este

$$d_1 = 2D(x_A + 1, y_A + 1/2) = 2D(x_A, y_A) + 2a + b = 2a + b = 2dy - dx,$$

deoarece se presupune că $(x_A, y_A)$ se află pe dreaptă.

## 3.4 Cercuri

Se consideră un cerc centrat în origine, de ecuaţie

$$x^2 + y^2 = R^2,$$

unde raza $R$ este un număr întreg de unităţi de rastru. Cercurile care nu sunt centrate în origine se translatează înspre origine, iar la convertirea scan se ţine seama de deplasament.

Există o serie de variante pentru conversia scan a cercurilor. Rezolvând ecuaţia implicită se obţine $y = \pm\sqrt{R^2 - x^2}$. Pentru a trasa un sfert de cerc (celelalte trasându-se prin simetrie, vezi figura 3.4.a) se poate incrementa $x$ de la 0 la $R$ cu paşi unitari, rezolvând ecuaţia $y = \sqrt{R^2 - x^2}$ pentru fiecare pas. Viteza de conversie a cercului în acest caz este extrem de scăzută datorită numeroaselor operaţii de multiplicare şi extragere a rădăcinii pătrate. In plus cercul va avea „găuri" pentru valori ale lui $x$ apropiate de $R$ (figura 3.5).

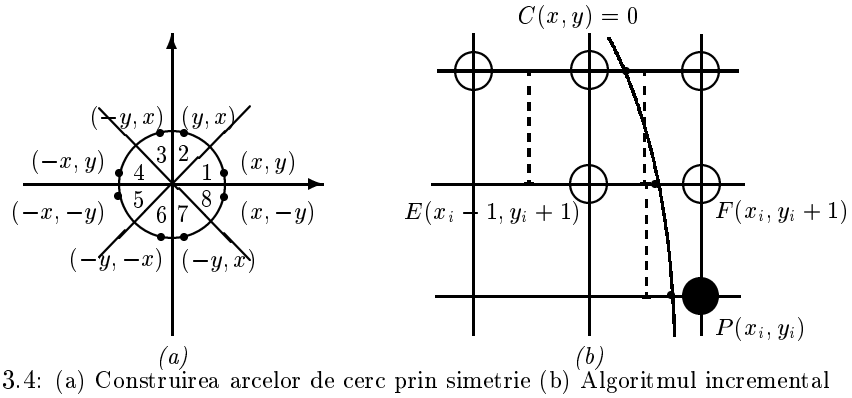



Figura 3.4: (a) Construirea arcelor de cerc prin simetrie (b) Algoritmul incremental în octantul unu

O altă variantă este afişarea punctelor apropriate de $(R\cos\theta, R\sin\theta)$, cu $\theta$ parcurgând pas cu pas intervalul $[0^0, 45^0]$ (calculele sunt de asemenea numeroase).

Ca şi în algoritmul de generare a liniilor în rastru, determinarea poziţiilor de coordonate întregi ce formează o imagine circulară pot fi obţinute printr-un algoritm incremental ce determină, la fiecare pas, pixelul cel mai apropiat de cerc.

Algoritmul de generare incrementală se bazează în mod esenţial pe ipoteza de echidistanţă orizontală şi verticală a punctelor rastrului. Dacă această condiţie nu este îndeplinită, algoritmul următor va trasa elipse în loc de cercuri.

Se consideră sensul trigonometric de parcurs al cercului. Problema este examinată separat pentru fiecare octant. In interiorul unui octant sunt posibile numai două deplasări elementare între două puncte generate consecutiv: o deplasare perpendiculară pe axa mai apropiată, sau o deplasare diagonală. De exemplu, în octantul 1, deplasările elementare sunt în sus sau spre stânga-sus, iar în octantul 2, la stânga sau spre stânga-sus.

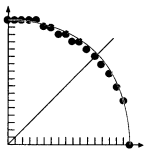


Figura 3.5: Rezultatul trasării incrementale pe baza ecuaţiei cercului

Se consideră, de exemplu, cazul primului octant şi a unui cerc de ecuaţie $C(x,y) = 0$ cu $C(x,y) = x^2 + y^2 - R^2$. Se consideră generate la un moment dat (la un pas anterior) coordonatele $(x_i, y_i)$ relative la centrul cercului. Se pot alege două posibilităţi la pasul următor: $(x_i, y_i + 1)$ sau $(x_i - 1, y_i + 1)$ (figura 3.4.b). Va fi generat punctul care este cel mai apropiat de cercul ideal. Primul se află în exteriorul cercului, deci $C(x_i, y_i + 1) > 0$, iar celălat în interior, deci $C(x_i - 1, y_i + 1) < 0$. Punctul corespunzător valorii celei mai mici în modul este cel selectat. In formularea `Bresenham` a algoritmului se consideră *factorul de decizie*

$$\Delta_i = C(x_i, y_i + 1) + C(x_i - 1, y_i + 1) = x_i^2 + (x_i - 1)^2 + 2(y_i + 1)^2 - 2R^2.$$

Dacă $\Delta_i < 0$, adică $|C(x_i, y_i+1)| < |C(x_i-1, y_i+1)|$, atunci se aprinde primul punct, altfel cel de-al doilea. Pentru evitarea multiplelor operaţii de adunare, scădere şi înmulţire, factorul de decizie se poate obţine printr-o formulă recursivă. Astfel, dacă $\Delta_i < 0$, atunci

$$\Delta_{i+1} = \Delta_i + 4y_{i+1} + 2,$$

altfel

$$\Delta_{i+1} = \Delta_i - 4x_{i+1} + 4y_{i+1} + 2.$$

Cele trei puncte $(x_i, y_i, \Delta_i)$ definesc *starea* curentă a algoritmului pentru fiecare punct generat. Valorile iniţiale pentru cele trei *variabile de stare* sunt

$$x_0 = R, \qquad y_0 = 0, \qquad \Delta_0 = 3 - 2R.$$

In concluzie, la fiecare iteraţie se fac doi paşi: (1) se alege pixelul pe baza semnului variabilei decizionale calculate în iteraţia anterioară şi (2) se actualizează variabila decizională cu o valoare ce corespunde alegerii pixelului. Singura diferenţă faţă de algoritmul pentru linii este aceea că, în actualizarea variabilei decizionale, se evaluează o funcţie liniară în coordonatele pixelului.

Odată rezolvată problema generării cercului în primul octant, pentru trasarea în ceilalţi octanţi se poate proceda prin simetrie. Există însă situaţii când ordinea generării punctelor cercului are importanţă, de exemplu, dacă trasarea trebuie să se efectueze nu cu linie continuă, ci cu linie întreruptă. Atunci se continuă generarea incrementală şi în ceilalţi octanţi.

In octantul 2, după generarea punctului $(x_i, y_i)$, următorul punct generat trebuie ales între $(x_i - 1, y_i + 1)$ şi $(x_i - 1, y_i)$. Factorul de decizie este

$$\Delta_i = y_i^2 + (y_i + 1)^2 + 2(x_i - 1)^2 - 2R^2,$$

cu relaţia recursivă

$$\Delta_{i+1} = \begin{cases} \Delta_i - 4x_{i+1} + 2, & \Delta_i < 0 \\ \Delta_i - 4x_{i+1} - 4y_{i+1} + 2, & \text{altfel} \end{cases}.$$

Prin calcule simple se pot stabili formulele de trecere de la octantul 1 la octantul 2 pentru factorul decizional, funcţie de ultimele valori alese şi ultima direcţie de deplasare, respectând regulile de conexiune discretă.

*Algoritmul punctului mijlociu* pentru trasarea dreptelor poate fi extins la cazul trasării cercurilor. Pentru exemplificare se consideră octantul de la $x = 0$ la $x = y = R/\sqrt{2}$ şi sensul de parcurs al curbei conform acelor ceasornicului. Precum în algoritmul punctului de mijloc pentru linii, strategia constă în a selecta un pixel dintre doi posibili, respectiv cel care este mai apropiat de cercul ideal, prin evaluarea unui funcţii în punctul de mijlocul dintre cei doi pixeli (în cazul octantului secund, între $E$ şi $F$ din figura 3.6).

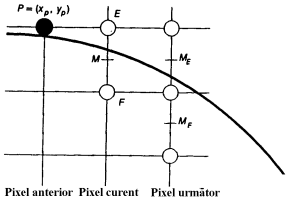



Figura 3.6: Grila cu pixeli în cazul octantului secund

Funcţia $C$ este 0 pe cerc, pozitivă în afara cercului, negativă în interiorul cercului. Dacă punctul de mijloc dintre $E$ şi $F$ este în afara cercului, $F$ este mai apropiat de cerc, iar dacă este în interior, $E$ este mai apropiat de cerc. Se consideră $\Delta_{i+1} = C(x_i + 1, y_i - \frac{1}{2})$. Dacă $\Delta_{i+1} < 0$ se alege $E$ şi numai $x$ va fi incrementat, astfel încât $\Delta_{i+2} = C(x_i + 2, y_i - \frac{1}{2}) = \delta_{i+1} + \Delta^E_{i+1}$, $\delta^E_{i+1} = 2x_i + 3$; în caz contrar, se alege $F$ şi $\Delta_{i+2} = C(x_i + 2, y_i - \frac{3}{2}) = \Delta_{i+1} + \delta^F_{i+1}$, $\Delta^F_{i+1} = 2x_i - 2y_i + 5$. Considerăm punctul iniţial de start a procedurii iterative ca fiind $(0, R)$. Următorul punct de mijloc este $(1, R - \frac{1}{2})$, iar $C(1, R - \frac{1}{2}) = \frac{5}{4} - R$. Datorită fracţiei iniţiale în expresia lui $d$ este necesară aplicarea aritmeticii în numere reale. Astfel, se defineşte variabila decizională $\Delta_i = d_i - \frac{1}{4}$, pentru ca iniţializarea să fie $\Delta = 1 - R$, iar comparaţia $d < 0$ devine $\Delta < -1/4$. Deoarece

$\Delta$ porneşte de la o valoare întreagă şi este incrementat cu valori întregi această ultimă comparaţie poate fi înlocuită cu $\Delta < 0$.

Performanţele acestui algoritm pot fi îmbunătăţite dacă tehnica incrementală este utilizată şi la evaluarea funcţiilor liniare ce constituie incrementele variabilei decizionale (orice polinom poate fi calculat incremental). Stategia utilizată constă în evaluarea funcţiilor direct în două puncte adiacente, calcularea diferenţei şi aplicarea diferenţei la fiecare iteraţie. Dacă se alege $E(x_i + 1, y_i)$ în iteraţia curentă, atunci $\Delta^E_{i+2} = 2(x_i + 1) + 3 = \Delta^E_{i+1} + 2$, iar $\Delta^F_{i+2} = 2(x_i + 1) - 2y_i + 5 = \delta^F_{i+1} + 2$. Dacă se alege $F(x_i + 1, y_i - 1)$ în iteraţia curentă, atunci $\Delta^E_{i+2} = 2(x_i + 1) + 3 = \Delta^E_{i+1} + 2$, iar $\Delta^F_{i+2} = 2(x_i + 1) - 2(y_i - 1) + 5 = \Delta^F_{i+1} + 4$. Astfel, algoritmul revizuit constă în următorii paşi:

1. calculul pixelului bazat pe semnul variabilei $\Delta$ determinată în iteraţia anterioară;
2. actualizarea variabilei de decizie utilizând valorile $\Delta^E$ şi $\Delta^F$ calculate în iteraţia anterioară;
3. actualizarea variabilelor $\Delta^E$ şi $\Delta^F$ ţinând cont de poziţia noului pixel şi utilizând diferenţele constante calculate anterior.

```
Procedure MidpointCircle(radius, value: integer);
{Trasează semicercul de la (0, R) la (R√2, R√2)}
{Se presupune că cercul este centrat în origine}
var
   x, y, d, deltaE, deltaF: integer;
begin
   x := 0;
   y := radius;
   d := 1 − radius;
   deltaE := 3;
   deltaF := −2 ∗ radius + 5;
   WritePixel(x, y, value);
   while y > x do
         begin
         if d < 0 then {Selectează E}
            begin
                  d := d + deltaE;
                  deltaE := deltaE + 2;
                  deltaF := deltaF + 2;
                  x := x + 1
            end
         else
            begin
                  d := d + deltaF;
                  deltaE := deltaE + 2;
                  deltaF := deltaF + 4;
                  x := x + 1;
                  y := y − 1
            end;
```

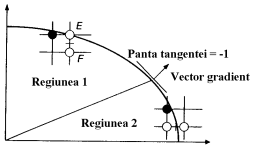



Figura 3.7: (a) Primul cuadrant în trasarea incrementală a elipsei centrate în origine şi cu semiaxele paralele cu axele de coordonate

```
            WritePixel(x, y, value);
            end {while}
end; {MidpointCircle}
```

## 3.5 Elipse

Algoritmul punctului mijlociu a fost extins şi la cazul conversiei `scan` a elipselor drepte (algoritmul `Da Silva`). Pentru simplificarea calculelor se trasează arcul de elipsă care se află în primul cuadrant, iar celelalte trei, prin simetrie.

Se divide primul cuadrant în două regiuni (figura 3.7), în urma determinării punctului în care tangenta la elipsă are panta $-1$ (punctul satisface ecuaţia elipsei $E(x,y) = b^2x^2 + a^2y^2 - a^2b^2 = 0$ şi componentele vectorului gradient trebuie să fie egale, $2b^2x = 2a^2y$). Ca şi în algoritmul punctului mijlociu pentru drepte, se evaluează funcţia $E$ în punctul de mijloc dintre cei doi pixeli în discuţie în regiunea curentă pentru a determina poziţia punctului de mijloc faţă de elipsă şi, deci, cel mai apropiat pixel. Variabila de decizie este aleasă proporţional cu această valoare, multiplicată cu un factor ce elimină împărţirile prin constante, şi care poate fi evaluată prin diferenţe.

Se consideră sensul de parcurs al elipsei conform sensului orar. In regiunea 1, dacă pixelul curent are coordonatele $(x_i, y_i)$, atunci variabila de decizie este $d_{i+1} = 4E(x_i + 1, y_i - \frac{1}{2})$. Dacă următoarea mutare se face la $E$, atunci $d_{i+2} = E(x_i + 2, y_i - \frac{1}{2}) = d_{i+1} + 4b^2(2x_i + 3)$. Pentru o mutare la $F$, $d_{i+2} = E(x_i + 2, y_i - \frac{3}{2}) = d_{i+1} + 4a^2(-2x_i + 2)$. Presupunem că punctul de start este $(0, b)$; atunci $d_1 = 4E(1, b - \frac{1}{2}) = 4b^2 + a^2(-4b + 1)$. Vectorul gradient este definit prin $\text{grad}\, E(x,y) = \frac{\partial E}{\partial x}\mathrm{i} + \frac{\partial E}{\partial y}\mathrm{j} = 2b^2x\mathrm{i} + 2a^2y\mathrm{j}$. Frontiera dintre cele două regiuni este dat de punctul în care panta curbei este -1, adică când vectorul gradient are panta 1. Dacă $a^2(y_i - \frac{1}{2}) \leq b^2(x_i + 1)$ se trece din regiunea 1 în regiunea 2 (se va proceda în mod similar cu cazul cercului).

Algoritmul descris mai sus poate fi extins pentru cazul unei elipse oarecare descrisă prin ecuaţia $G(x,y) := ax^2 + bxy + cy^2 + dx + ey + f = 0$ (se include astfel şi cazul cercului). Planul se divide în opt octanţi, funcţie de direcţiile de trasare incrementală (figura 3.8 şi tabelul 3.1). Astfel, în octantul 1 se poate alege

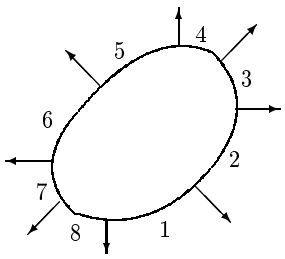


Figura 3.8: Octanţii în cazul unei elipse oarecare

între mutarea la dreapta sau pe diagonală sus-dreapta. Se consideră aprins pixelul $(x_i, y_i)$ din octantul 1. Variabila de decizie este $d_i = G(x_i + 1, y_i + \frac{1}{2})$. Următoarea valoare poate fi $d_{i+1} = G(x_i+2, y_i+\frac{1}{2})$ sau $d_{i+1} = G(x_i+2, y_i+\frac{3}{2})$. In primul caz, $d_{i+1} = d_i + u_{i+1}$, iar în al doilea, $d_{i+1} = d_i + v_{i+1}$, unde

$$u_i = a(2x_i + 1) + b(y_i + 1/2) + d + 2a,$$

$$v_i = (2a + b)x_i + (b + 2c)y_i + a + b/2 + d + e + 2(a + b + c).$$

Pentru $u_i$ şi $v_i$ se pot stabili relaţii recursive ce depind de direcţia de mişcare. Pentru mişcarea orizontală,

$$u_{i+1} = u_i + 2a, \quad v_{i+1} = v_i + 2a + b,$$

iar pentru mişcarea diagonală,

$$u_{i+1} = u_i + 2a + b,\ v_{i+1} = v_i + 2a + 2b + 2c.$$

Trecerea din octantul 1 în octantul 2 se realizează când expresia următoare îşi schimbă semnul:

$$\left(\frac{\partial G}{\partial x} + \frac{\partial G}{\partial y}\right)(x_i, y_i) = (2ax_i + by_i + d) + (bx_i + 3cy_i + e) = v_i - \left(a + \frac{b}{2}\right)$$

iar trecerea din octantul 2 în octantul 3, când expresia următoare îşi schimbă semnul:

$$\frac{\partial G}{\partial x}(x_i, y_i) = 2ax_i + by_i + d = u_i - \left(a + \frac{b}{2}\right).$$

Dacă se notează cu $k_1 = 2a,\ k_2 = 2a + b,\ k_3 = 2a + 2b + 2c$, atunci valorile de actulizare sunt:

1. mişcarea pe orizontală sau verticală: $u_{i+1} = u_i + k_1,\ v_{i+1} = v_i + k_2$;
2. mişcarea pe diagonală: $u_{i+1} = u_i + k_2,\ v_{i+1} = v_i + k_3$.

Componentele vectorului gradient se pot exprima astfel:

$$\frac{\partial G}{\partial x} = u_i - \frac{k_2}{2},\ \frac{\partial G}{\partial x} + \frac{\partial G}{\partial y} = v_i - \frac{k_2}{2}.$$

Algoritmul constă din mai mulţi paşi (un pas pentru un pixel), la un pas efectuându-se următoarele etape:

| Octant | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| $\Delta x$ | 1 | 0,1 | 0,-1 | -1 | -1 | 0,-1 | 0,1 | 1 |
| $\Delta y$ | 0,1 | 1 | 1 | 0,1 | 0,-1 | -1 | -1 | 0,-1 |

Tabelul 3.1: Mutări în octanţi

1. trasare pixel;
2. alegere pixel nou pe baza valorii $d_i$;
3. calculare $u_{i+1}$ şi $v_{i+1}$ funcţie de alegerea efectuată în etapa anterioară;
4. calculare $d_{i+1}$ prin adăugare $u_{i+1}$ sau $v_{i+1}$ funcţie de alegerea efectuată în etapa a doua;
5. verificarea schimbării octantului.

```
function GetOctant(D, E :integer):  integer;
begin
  if D > 0 and E < 0 then
    if D < −E then GetOctant:=1
      else GetOctant:=2;
  if ... {celelalte şase cazuri}
end;
procedure Conic(xs, ys, xe, ye, {Punctele de start şi stop}
A, B, C, D, E, F : integer); {Coeficeinţii}
var
   x, y, octant,            {Punctul şi octantul curent}
   dxp, dyp, dxd, dyd,      {Schimbări în (x, y)}
   d, u, v, k1, k2, k3,     {Var.  decizie şi incrementare}
   dSdx, dSdy,              {Componentele gradientului}
   octantContor,            {Numărul de octanţi de trasat}
   tmp : integer;
begin
   octant :=GetOctant(D, E);
   case octant of
   1:  begin
         d :=round(A + B/2 + C/4 + D + E/2 + F);
         u :=round(A + B/2 + D); v :=round(A + B/2 + D + E);
         k1 := 2 * A;  k2 := 2 * A + B;  k3 := k2 + B + 2 * C;
         dxp := 1;  dyp := 0;  dxd := 1;  dyd := 1
       end;
   2:  begin
         d :=round(A/4 + B/2 + C + D/2 + E + F);
         u :=round(B/2 + C + E); v :=round(B/2 + C + D + E);
         k1 := 2 * C;  k2 := B + 2 * C;  k3 := k2 + B + 2 * A;
         dxp := 0;  dyp := 1;  dxd := 1;  dyd := 1
       end;
   {... celelelate 6 cazuri}
   end;
   {Translatează (xs, ys) în origine}
   x := xe − xs;  y := ye − ys;
```

```
    dSdx := 2 * A * x + B * y + D;  dSdy := 2 * C * y + B * x + E;
    octantContor :=GetOctant(dSdx, dSdy) - octant;
    {determină octantul terminal}
    if octantContor <= 0 then
         octantContor := octantContor + 8;
    x := xs;  y := ys;
    while octantContor > 0 do
      begin
          if odd(octant) then
            begin
             while v <= k2/2 do
               begin
                   DrawPixel(x, y);
                   if d < 0
                      then begin x := x + dxp;  y := y + dyp;
                           u := u + k1;  v := v + k2;  d := d + u end
                      else begin x := x + dxd;  y := y + dyd;
                           u := u + k2;  v := v + k3;  d := d + v end
               end;
             {schimbă octantul}
             d :=round(d - u - v/2 - k2/2 + 3 * k3/8);
             u :=round(-u + v - k2/2 + k3/2);
             v :=round(v - k2 + k3/2);
             k1 := k1 - 2 * k2 + k3;  k2 := k3 - k2;
             tmp := dxp;  dxp := -dyp;  dyp := tmp
            end
          else
            begin
             while u <= k2/2 do
               begin
                   DrawPixel(x, y);
                   if d < 0
                      then begin x := x + dxd;  y := y + dyd;
                           u := u + k2;  v := v + k3;  d := d + v end
                      else begin x := x + dxp;  y := y + dyp;
                           u := u + k1;  v := v + k2;  d := d + u end
               end;
             {schimbă octantul}
             d := d + u - v + k1 - k2;  v := 2 * u - v + k1 - k2;
             u := u + k1 - k2;
             k3 := 4 * (k1 - k2) + k3;  k2 := 2 * k1 - k2;
             tmp := dxd;  dxp := -dyd;  dypd := tmp
            end;
          octant := octant + 1;
          if octant > 8 then octant := octant - 8;
          octantContor := octantContor - 1
      end
end;
```

## 3.6 Curbe de grad doi

Fie o curbă de grad doi în forma generală

$$f(x,y) = a_1x^2 + a_2xy + a_3y^2 + a_4x + a_5y + a_6 = 0,$$

unde coeficienţii $a_1, a_2, \ldots, a_6$ sunt în general numere reale, cu $a_1 > 0$. Curba reprezentată de această ecuaţie este o conică. Se consideră cantităţile

$$\delta = a_2^2 - 4a_1a_3, \qquad \Delta = a_1a_5^2 + a_3a_4^2 - a_2a_4a_5 + (a_2^2 - 4a_1a_3)a_6$$

(dacă determinăm pe $x$ funcţie de $y$ din ecuaţia $f(x,y) = 0$, discriminantul ecuaţiei de grad doi, $\Delta_1(y)$, polinom de grad doi în $y$, are la rândul său ca discriminant pe $4a_1\Delta$).

Ecuaţia conicei reprezintă o curbă degenerată dacă funcţia de grad doi $f(x,y)$ se poate scrie sub forma

$$f(x,y) = (b_1x + b_2y + b_3)(c_1x + c_2y + c_3),$$

unde coeficienţii $b_i, c_i$ sunt în general numere complexe. Dacă aceştia sunt numere reale, conica degenerează în două drepte. Condiţia de degenerare este $\Delta = 0$.

Dacă curba este nedegenerată, ecuaţia conicei reprezintă o curbă propriu-zisă, reală sau imaginară. Tipul conicei depinde de semnul discriminantului $\delta$:

$$\delta \begin{cases} < 0: & \text{elipsă (sau cerc),} \\ = 0: & \text{parabolă,} \\ > 0: & \text{hiperbolă.} \end{cases}$$

Dacă curba este reală, atunci funcţia $f(x,y)$ ia valori negative în interiorul concavităţilor curbei $f(x,y) = 0$ şi valori pozitive în exteriorul acestora.

Se presupune că $a_i$ sunt numere întregi fără divizori comuni. Se utilizează ipoteza de echidistanţă orizontală şi verticală a punctelor rastrului. Dacă dimensiunile celulei elementare de rastru sunt în raportul $p/q \neq 1$, cu $p, q$ întregi, pentru o trasare corectă se poate proceda în modul următor: se înlocuieşte ecuaţia curbei $f(x,y) = 0$ cu

$$g(x,y) = p^2 f\left(x, \frac{q}{p}y\right) = 0,$$

care are toţi coeficienţii întregi şi permite redarea corectă a proporţiilor.

Rezolvarea exactă a problemei plasării punctelor în toate cazurile implică, în general, rezolvarea unor ecuaţii de grad doi pentru fiecare punct generat. Aceste operaţii sunt prea complicate pentru un algoritm eficient. De aceea se caută o procedură incrementală de trasare a curbei, care să funcţioneze prin deplasări elementare pe orizontală, verticală sau în diagonală între punctele succesiv generate (8 direcţii de deplasare).

Se numerotează de la 0 la 7 direcţiile posibile de deplasare de la un punct la cele 8 puncte vecine din rastru, numerotarea făcându-se în sens trigonometric direct. Punctele vecine lui $P$ se notează cu $P_i$, $i = 0, \ldots, 7$ (figura 3.9.a).

*Observaţii:*

(a) pentru două puncte $(x_1, y_1)$ şi $(x_2, y_2)$ relativ apropiate de curba ideală, se poate decide care dintre ele este cel mai apropiat prin compararea cantităţilor $|f(x_1, y_1)|$ şi $|f(x_2, y_2)|$. Punctul cel mai apropiat de curbă este acela în care funcţia $f(x, y)$ este minimă în valoare absolută.

(b) dacă algoritmul de generare a efectuat o deplasare în direcţia $i$ între două puncte succesive, atunci pentru următorul punct generat trebuie să se efectueze o deplasare elementară în una din direcţiile $i-1$, $i$ sau $i+1$ (considerate `modulo` 8). Această regulă este o altă exprimare pentru regula de conexiune discretă (figura 3.9.b). Cercurile minime care îndeplinesc această condiţie au diametrul de cel puţin 3 unităţi de rastru.

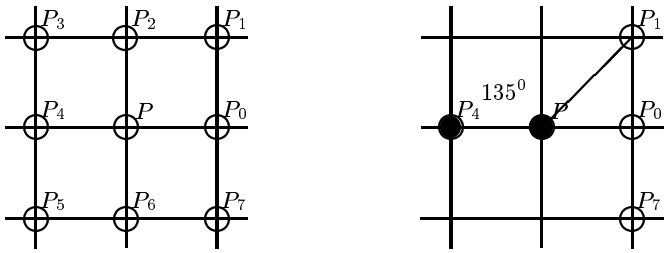



Figura 3.9: (a) Direcţiile posibile de deplasare în algoritmul incremental (b) Aplicarea regulii conexiunii discrete

Ideea algoritmului este următoarea. Se alege sensul de parcurs al curbei astfel încât $f(x, y) < 0$ ($> 0$) la stânga (respectiv dreapta) direcţiei de înaintare. Dacă punctul $P$ a fost generat în urma deplasării elementare în direcţia $i$, atunci punctele candidat sunt $P_{i-1}$, $P_i$, $P_{i+1}$ (considerate `modulo` 8). Conform presupunerii anterioare $P_{i+1}$ se află în interior ($f(P_{i+1}) < 0$), iar $P_{i-1}$ în exterior ($f(P_{i-1}) > 0$). Despre punctul $P_i$ nu se poate afirma dacă este în exterior sau interior. Pentru alegerea unuia dintre cele trei puncte sunt necesare următoarele comparaţii:

$$\begin{cases} f(P_{i-1}) + f(P_i) < 0: & \text{se alege punctul } P_{i-1} \\ f(P_{i-1}) + f(P_i) \geq 0 \text{ şi } f(P_i) + f(P_{i+1}) \leq 0: & \text{se alege punctul } P_i \\ f(P_i) + f(P_{i+1}) > 0: & \text{se alege punctul } P_{i+1} \end{cases}$$

*Observaţie.* Aplicând acest criteriu decizional pentru ecuaţia unei drepte, se obţin relaţiile ce stau la baza algoritmului `Bresenham`. De asemenea, acest criteriu general este particularizat de relaţiile care guvernează trasarea incrementală a cercurilor.

Dacă se exprimă toate valorile funcţiei $f$ în punctele din imediata apropiere a lui $P$, de coordonate $(x, y)$, rezultă:

$$f(P_0) = f(x+1, y) = f(P) + (2a_1x + a_2y + a_4) + a_1;$$
$$f(P_1) = f(P) + (2a_1x + a_2y + a_4) + (a_2x + 2a_3y + a_5) + a_1 + a_2 + a_3;$$
$$\ldots$$

şi se poate dezvolta algoritmul de trasare incrementală pe baza criteriului de decizie şi a acestor relaţii de recurenţă. Sunt efectuate doar operaţii de

adunare, scădere şi înmulţire între numere întregi pentru fiecare punct generat. Operaţiile de înmulţire pot fi evitate în cazul unei alegeri adecvate a variabilelor de stare. Relaţiile de recurenţă pot fi exprimate simplu în funcţie de derivatele parţiale $f'_x$ şi $f'_y$ ale lui $f$ în punctul $P$, notate cu $fx$ şi $fy$:

$$fx(x,y) = f'_x(x,y) = 2a_1x + a_2y + a_4,$$

$$fy(x,y) = f'_y(x,y) = a_2x + 2a_3y + a_5.$$

Funcţiile $fx, fy$ fiind liniare, actualizările lor incrementale se exprimă numai prin operaţii de adunare şi scădere. In cazul unei deplasări elementare între două puncte vecine:

$$fx(P_0) = fx(P) + 2a_1, \qquad fy(P_0) = fy(P) + a_2,$$

$$fx(P_1) = fx(P) + 2a_1 + a_2, \qquad fy(P_1) = fy(P) + a_2 + 2a_3,$$

$$\ldots$$

Variabilele de stare sunt valorile funcţiilor $f, fx, fy$ în punctul curent generat.

## 3.7 Curbe plane

### 3.7.1 Descrierea curbelor plane

In raport cu un reper cartezian ortogonal, curbele plane pot fi descrise prin:

1. *ecuaţii explicite:* un punct oarecare $(x, y)$ de pe curbă este descris în forma explicită $y = f(x)$;
2. *ecuaţii polare:* un punct oarecare $(x, y)$ de pe curbă este descris în coordonate polare prin cuplul $(r, t)$, unde $t$ parcurge un interval $[a, b]$ şi $r = f(t)$, corespondenţa dintre cele două sisteme de coodonate, carteziene şi polare, presupunând ecuaţii de tipul $x = r\cos t,\ y = r\sin t$.
3. *ecuaţii parametrice:* un punct oarecare $(x, y)$ de pe curbă este descris prin $x = f(t),\ y = g(t)$, unde $t$ parcurge un interval $[a, b]$ din $\mathbb{R}$, adică $F(t) = (f(t), g(t))$ reprezintă un drum în planul $\mathbb{R}\times\mathbb{R}$;
4. *ecuaţii implicite:* un punct oarecare $(x, y)$ de pe curbă este descris de ecuaţia $F(x, y) = 0$. In anumite condiţii impuse lui $F$, folosind teorema funcţiilor implicite din calculul diferenţial, se poate ajunge la studiul unei curbe descrise prin ecuaţii explicite. In secţiunea anterioară a fost descris un algoritm incremental pentru cazul în care $F(x, y)$ este funcţie polinomială de grad maxim doi în $x$ şi $y$.

### 3.7.2 Trasarea curbelor plane

Pentru trasarea curbelor definite *explicit* trebuie să se ţină seama de dispozitivul de afişare grafică de care se dispune. Este necesară scrierea unui proceduri care operează cu coordonate (în mod grafic) ce reprezintă numere întregi şi care ţine seama de scara la care se execută desenul. Scalarea care trebuie efectuată se obţine în urma analizei valorilor extreme ale funcţiei pe domeniul pe care se reprezintă.

Pentru reprezentarea grafică a unei curbe descrisă prin ecuaţii *polare* este necesară o transformare în coordonate carteziene. Această transformare este $x = f(t)\cos t,\ y = f(t)\sin t$ (curbă plană descrisă prin ecuaţii parametrice).

Se abordează aici doar problema trasării curbei cu reprezentarea *parametrică*

$$x = x(t), \quad y = y(t), \qquad t \in [0, 1],$$

unde $x(\cdot)$ şi $y(\cdot)$ sunt funcţii continue.

O primă idee pentru dezvoltarea unui algoritm de trasare constă în alegerea unei *rate constante* $\Delta$ de creştere a variabilei independente $t$ şi selectarea, în mediul de afişare, a punctelor care corespund cel mai bine adreselor calculate $(x(k\Delta), y(k\Delta))$, unde $k = 0, 1, \ldots, [1/\Delta]$. Principala dificultate constă în alegerea parametrului $\Delta$, deoarece nu se ştie de la început câte puncte vor fi efectiv generate. Pe de altă parte, însăşi reprezentarea parametrică dată, care nu este unică, poate fi defavorabilă, astfel încât orice alegere a unui constante de creştere $\Delta$ este total nepotrivită: pe unele porţiuni ale curbei, parametrul $\Delta$ poate fi prea mic, ceea ce are ca rezultat selectarea de mai multe ori a aceluiaşi punct; pe alte porţiuni ale curbei, acelaşi parametru $\Delta$ poate fi prea mare, generând puncte consecutive neadiacente în mediu de afişare, care, dacă sunt unite prin linii drepte, nu mai aproximează suficient de exact curba dată. Astfel, orice metodă bazată pe creşterea constantă a variabilei independente nu poate produce algoritmi eficienţi în toate cazurile. Este necesar să se găsească o metodă bazată pe variaţia neconstantă a variabilei $t$, care să urmărească pe cât posibil variaţia curbei.

Principiul metodei trasării prin *înjumătăţirea intervalului* este următorul: se calculează mai întâi adresele punctelor de la capetele curbei, corespunzând extremităţilor intervalului de variaţie a parametrului $t$. Dacă aceste două puncte sunt suficient de apropiate în raport cu rezoluţia mediului de afişare, problema este rezolvată prin aprinderea acestor puncte. In caz contrar, intervalul curent de variaţie a lui $t$ se împarte în două subintervale de lungimi egale, subintervalul din dreapta se memorează într-o structură de date de tip stivă pentru necesităţi ulterioare şi se reia problema cu subintervalul din stânga. Injumătăţirea intervalului continuă până când lungimea acestuia este suficient de mică pentru ca punctele de pe curbă corespunzătoare extremităţilor intervalului să fie adiacente în mediu de afişare. In acest caz se descarcă stiva şi se reia procedura cu cel mai recent subinterval memorat în stivă. Trasarea curbei se termină când stiva se goleşte complet.

Metoda de trasare prin înjumătăţirea intervalului poate fi concretizată şi într-o formă mai concisă cu ajutorul unei proceduri recursive, care să nu facă apel în mod explicit la o structură de date de tip stivă.

Corectitudinea procedurii de principiu se bazează implicit pe faptul că funcţiile de parametrizare sunt continue. In caz contrar, nu este neapărat necesar ca, între două puncte corespunzătoare valorilor $t_1 < t_2$ ale parametrului, toate punctele intermediare (în sensul valorilor absciselor şi ordonatelor) să corespundă la valori ale lui $t$ din intervalul $[t_1, t_2]$.

Există situaţii când ordinea generării punctelor curbei are importanţă. De exemplu, dacă trasarea curbei trebuie să se efectueze nu cu linie continuă, ci cu linie întreruptă corespunzând unui anumit şablon (configuraţie liniară de biţi, `pattern`) predefinit, atunci algoritmul de trasare trebuie să permită calcularea punctelor succesive ale curbei în paralel cu parcurgerea configuraţiei binare a

şablonului, care se repetă în mod circular. Punctul curent de pe curbă este efectiv selectat în mediu de afişare numai dacă bitul corespunzător din şablon este 1. Trasarea cu şablon nu este pe deplin reuşită dacă algoritmul de trasare generează de două ori acelaşi punct pe curbă. Este necesară inhibarea generării efective a unui punct dacă coordonatele acestuia coincid cu cele ale unui punct generat anterior.

## 3.8 Curbe spaţiale

### 3.8.1 Descrierea curbelor spaţiale

Curbele spaţiale pot fi descrise:

1. explicit prin ecuaţii de tipul $y = f(x),\ z = g(x)$;
2. implicit printr-o ecuaţie de tipul $F(x, y, z) = 0$;
3. parametric prin ecuaţii de tipul $x = x(t),\ y = y(t),\ z = z(t)$.

Prima descriere nu este valabilă pentru toate curbele. De exemplu, un cerc într-un plan cu $z$ constant are două valori pentru un $x$. A doua descriere oferă mai multe soluţii, dar, de exemplu, semicercul $x^2 + y^2 = 1,\ x \geq 0$ nu poate fi descris printr-o ecuaţie de tipul cerut. De aceea, reprezentarea parametrică este des utilizată.

*Curbele polinomiale parametrice* sunt curbe 3D definite prin trei polinoame într-un parametru $t$, câte una pentru coordonatele $x$, $y$, $z$. *Curbele cubice* folosesc polinoame cubice în $t$. Curbele polinomiale de grad mai mare ca patru sunt rar utilizate deoarece prezintă o multitudine de ondulaţii.

### 3.8.2 Curbe parametrice cubice

Cubica parametrică spaţială poate fi exprimată parametric prin trei funcţii cubice şi este definită prin 12 coeficienţi:

$$\begin{cases} x(t) = a_x t^3 + b_x t^2 + c_x t + d_x \\ y(t) = a_y t^3 + b_y t^2 + c_y t + d_y, \\ z(t) = a_z t^3 + b_z t^2 + c_z t + d_z, \end{cases}$$

unde $t \in [0, 1]$. Dacă se notează

$$Q(t) = (x(t), y(t), z(t)), \quad T = (t^3, t^2, t, 1), \quad C = \begin{pmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{pmatrix}$$

atunci $Q(t) = TC$. Vectorul tangent la curbă în punctul corespunzător unui anumit $t$ este

$$Q'(t) = (3t^2, 2t, 1, 0)C$$

Fiecare polinom cubic are 4 coeficienţi şi deci sunt necesare 12 condiţii pentru determinarea necunoscutelor.

Un segment de curbă este definit prin constrângeri privind:

(a) punctele de capăt;
(b) vectorii tangenţi;
(c) continuitatea între mai multe segmente de curbă.

In cazul general, matricea $C$ este rescrisă sub forma $C = MG$, unde $M$ este o matrice $4 \times 4$, numită *matrice de bază* (ce defineşte tipul constrângerilor), iar $G$ o matrice $4 \times 3$, numită *vectorul geometric* (care defineşte constrângerile geometrice). Se notează $G_x$ prima coloană a lui $G$. Atunci $x(t) = TMG_x$, adică

$$x(t) = b_1(t)g_{1x} + b_2(t)g_{2x} + b_3(t)g_{3x} + b_4(t)g_{4x},$$

unde polinoamele `blending` (amestec, combinare) $b_i$ sunt polinoamele cubice obţinute din ecuaţia matriceală $B = TM$ (de exemplu, $b_1(t) = t^3m_{11} + t^2m_{21} + tm_{31} + m_{41}$). Se poate observa o similitudine cu modalitatea în care poate fi descrisă parametric o linie în 3D care are punctele de capăt $G_1$ şi $G_2$:

$$x(t) = (1-t)g_{1x} + tg_{2x},\ y(t) = (1-t)g_{1y} + tg_{2y},\ z(t) = (1-t)g_{1z} + tg_{2z}.$$

Dacă două curbe au un capăt comun spunem că ele sunt continue de clasă $G^0$ (*continuitate geometrică*). Dacă direcţiile (nu neapărat dimensiunile) celor doi vectori tangenţi la două curbe cu un punct comun coincid, se consideră continuitate geometrică de clasă $G^1$. Dacă vectorii tangenţi sunt egali (în direcţie şi dimensiune) în punctul comun a două curbe, este vorba de *continuitate parametrică* sau continuitate $C^1$ (figura 3.10). Dacă direcţiile şi mărimile derivatelor de ordin $n$, $d^n/dt^n[Q(t)]$, sunt egale în punctul comun curba rezultată este numită continuă de clasă $C^n$. Continuitatea $C^1$ implică $G^1$, exceptând cazul când vectorii tangenţi sunt nuli în punctul comun, dar reciproca nu este adevărată.

Inchiderea convexă în cazul curbelor bidimensionale este poligonul convex format de punctele de control ce definesc constrângerile. Pentru curbe 3D, închiderea convexă este poliedrul convex format de punctele de control. Proprietatea de închidere convexă este utilă în cazul decupării segmentelor de curbă: înainte de a aplica decuparea fiecărui segment de curbă se poate aplica algoritmul de decupare a închiderii convexe – numai dacă închiderea convexă se intersectează cu regiunea de decupare, segmentul de curbă în cauză trebuie examinat.

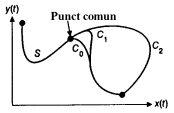



Figura 3.10: Segmentul de curbă $S$ are un punct comun cu segmentele de curbă $C_0$, $C_1$ şi $C_2$ cu gradele 0, 1 şi 2 de continuitate parametrică

### Curbe Hermite (curbe Coons)

Curbele `Hermite` sunt descrise prin funcţiile parametrice cubice ce au gradul minim necesar pentru a îndeplini patru condiţii: (figurile 3.11 şi 3.12):

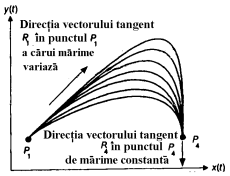



Figura 3.11: Familie de curbe `Hermite` pentru care vectorul tangent $R_1$ în $P_1$ variază pentru fiecare curbă, crescând în mărime pentru curbele mai înalte

(a) să treacă prin două puncte date ($P_1,\ P_4$);
(b) să aibă tangente date în aceste puncte ($R_1,\ R_4$).

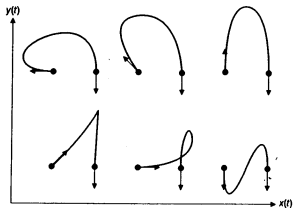



Figura 3.12: Familie de curbe `Hermite` pentru care direcţia vectorului tangent $R_1$ în $P_1$ variază, dar mărimea vectorului rămâne aceeaşi

Se defineşte vectorul de geometrie

$$G_H = \begin{pmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{pmatrix}$$

unde cele trei elemente ale unei linii reprezintă coordonatele punctelor, respectiv lungimile tangentelor descompuse pe axele de coordonate. Fie $G_{Hx}$ prima

coloană a acestei matrici şi $M_H$ matricea de bază. Atunci $x(t) = TM_HG_{Hx}$. Pentru determinarea matricei de bază $M_H$ se utilizează condiţiile din ipoteză. Se ştie că

$$\begin{aligned} x(0) &= x_{P_1} = (0,0,0,1)M_HG_{Hx}, \\ x(1) &= x_{P_4} = (1,1,1,1)M_HG_{Hx}, \\ x'(0) &= x_{R_1} = (0,0,1,0)M_HG_{Hx}, \\ x'(1) &= x_{R_4} = (3,2,1,0)M_HG_{Hx} \end{aligned}$$

adică

$$G_{Hx} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} M_HG_{H_x}.$$

Astfel

$$M_H = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}^{-1} = \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Aceeaşi matrice se obţine utilizând condiţiile în coordonata $y$ sau $z$. Generic,

$$Q(t) = TM_HG_H.$$

Polinoamele `Hermite` $B_H$ (figura 3.13) se obţin din $B_H = TM_H$. Funcţie de

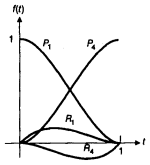



Figura 3.13: Funcţiile `Hermite` de amestecare (`blending`)

acestea, curba Hermite este descrisă astfel:

$$Q(t) = B_HG_H = (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_4 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4.$$

Pentru continuitatea geometrică $G^1$ a două segmente de curbă `Hermite` este necesar ca vectorii de geometrie să fie de tipul $(P_1, P_4, R_1, R_4)^T$ şi $(P_4, P_7, kR_4, R_7)^T$, unde $k \geq 0$. Pentru continuitatea de clasă $C^1$ a segmentului rezultat este necesar ca $k = 1$ (figura 3.14).

*Observaţie.* Curbele `Hermite` pot fi uşor transformate modificând doar vectorul de geometrie (matricea de bază este invariantă la rotaţii, scalări sau translaţii).

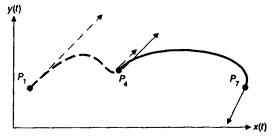


Figura 3.14: Două curbe Hermite au un punct comun $P_4$, vectorii tangent în $P_4$ au aceeaşi direcţie, dar nu şi aceeaşi mărime (continuitate $G^1$, dar nu şi continuitate $C^1$)

### Curbe Bézier

Se obţin cu ajutorul a patru puncte de control. Primul şi ultimul servesc la precizarea capetelor intervalului (în figura 3.15, $P_1$ şi $P_4$), iar punctele supli-

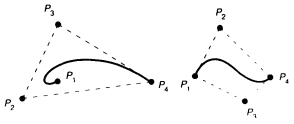


Figura 3.15: Două curbe `Bézier` şi punctele lor de control

mentare determină, împreună cu capetele, direcţia tangentelor ($P_1P_2$, $P_3P_4$). Dacă $R_1$ şi $R_4$ sunt tangentele în punctele $P_1$, respectiv $P_4$, atunci

$$R_1 = Q'(0) = m(P_2 - P_1), \quad R_4 = Q'(1) = m(P_4 - P_3).$$

Parametrul $m$ se numeşte factor de formă. Din considerentul includerii curbei în închiderea convexă a punctelor $P_1$, $P_2$, $P_3$, $P_4$ se consideră $m = 3$. Vectorul de geometrie `Bézier` este

$$G_B = \begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{pmatrix}.$$

Atunci

$$G_H = \begin{pmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -m & m & 0 & 0 \\ 0 & 0 & -m & m \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{pmatrix} = M_{HB} G_B.$$

Cum $Q(t) = TM_H G_H = TM_B G_B$ se obţine matricea de bază `Bézier` pentru $m = 3$

$$M_B = M_H M_{HB} = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

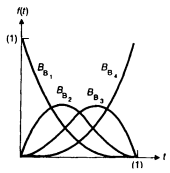


Figura 3.16: Polinoamele `Bernstein`

Polinoamele $B_B = TM_B$ sunt polinoamele `Bernstein` (figura 3.16). Funcţie de acestea curba `Bézier` este descrisă prin

$$Q(t) = (1-t)^3 P_1 + 3t(t-1)^2 P_2 + 3t^2(1-t)P_3 + t^3 P_4.$$

Pentru continuitatea geometrică $G^1$ a două segmente de curbă `Bézier` este necesar ca vectorii de geometrie să fie de tipul $(P_1, P_2, P_3, P_4)^T$ şi $(P_4, P_5, P_6, P_7)$, cu $P_3 - P_4 = k(P_4 - P_5)$ unde $k \geq 0$ (figura 3.17). Pentru continuitatea de clasă $C^1$ a segmentului rezultat este necesar ca $k = 1$

*Observaţie.* Deoarece suma ponderilor polinoamelor `Bernstein` este unu, segmentul de curbă se află în interiorul închiderii convexe a celor patru puncte de control.

### Curbe spline

Sunt constituite din mai multe segmente ce prezintă continuitate de clasă $C^1$ sau $C^2$ în punctele de cuplare. Există mai multe tipuri de curbe de acest tip,

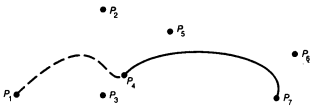


Figura 3.17: Două curbe `Bézier` cu un punct comun $P_4$; punctele $P_3$, $P_4$ şi $P_5$ sunt coliniare

cele mai utilizate fiind curbele B-`spline` uniforme sau neuniforme şi curbele $\beta$-`spline`. `Spline` desemnează un instrument folosit în desenul tehnic pentru trasarea curbelor netede (o bandă elastică care este fixată, cu ajutorul unor greutăţi, în punctele prin care trebuie să treacă curba).

In cazul concatenării unor segmente de curbe de tip `Hermite` sau `Bézier`, schimbarea unui punct de control afectează întreaga curbă. Curbele B-`spline` (figura 3.18) consistă din mai multe segmente de curbă a căror coeficienţi polinomiali depind de un număr mic de puncte de control (control local), astfel încât modificarea unui punct afectează numai o mică parte a curbei (figura 3.19).

O curbă cubică B-`spline` presupune precizarea a $m+1$ puncte de control notate $P_0,\ P_1, \ldots,\ P_m$, unde $m \geq 3$, şi constă din $m-2$ segmente de curbă cubică, notate $Q_3,\ Q_4, \ldots,\ Q_m$. Fiecare segment de curbă este definit doar pe propriul interval $0 \leq t < 1$. Parametrul $t$ este ajustat astfel încât $Q_i$ să fie definit de $t_i \leq t \leq t_{i+1}$, pentru $3 \leq i \leq m$. Dacă $m = 3$, există un singur segment de curbă, $Q_3$, definit pe intervalul $t_3 \leq t < t_4$ prin punctele de control $P_0, \ldots, P_3$. Pentru $m \geq i \geq 4$, există un punct comun (nod) între segmentele $Q_{i-1}$ şi $Q_i$.

O curbă B-`spline` uniformă presupune ca nodurile să fie spaţiate la intervale egale ale parametrului $t$. Se presupune astfel că $t_3 = 0,\ t_{i+1} - t_i = 1,\ i \geq 3$. Termenul "B" provine de la "bază", deoarece acest tip de curbe pot fi reprezentate ca sume ponderate ale unor funcţii polinomiale de bază (funcţii `blending`).

Segmentul de curbă $Q_i$ este definit de punctele $P_{i-3},\ P_{i-2},\ P_{i-1},\ P_i$ astfel încât vectorul de geometrie `B-spline` pentru segmentul $Q_i$ este $G_{Bs_i} = (P_{i-3}, P_{i-2}, P_{i-1}, P_i)^T,\ 3 \leq i \leq m$. Fiecare punct de control influenţează patru segmente de curbă. Dacă $T_i = ((t-t_i)^3, (t-t_i)^2, (t-t_i), 1)$, atunci $Q_i(t) = T_i M_{Bs} G_{Bs\,i},\ t_i \leq t < t_{i+1},\ 3 \leq i \leq m$. Pentru determinarea matricei de bază se exprimă segmentele de curbă funcţie de polinoamele `blending`:

$$Q_i(t-t_i) = b_1(t)P_{i-3} + b_2(t)P_{i-2} + b_3(t)P_{i-1} + b_4(t)P_i, \quad 0 \leq t < 1.$$

Dacă se impune condiţia de continuitate de clasă $C^2$ a segmentelor de dreaptă ($Q_i$ şi $Q_{i+1}$ să se cupleze în $t_{i+1}$ prin continuitate de clasă $C^2$, pentru oricare puncte de control), şi condiţia ca ponderile polinomiale să aibă suma 1 (curba se află în închiderea convexă a mulţimii punctelor de control,

$$\begin{cases} Q_i(t_{i+1}-t_i) = Q_{i+1}(t_{i+1}-t_{i+1}), & i = 3, \ldots, m \\ Q'_i(t_{i+1}-t_i) = Q'_{i+1}(t_{i+1}-t_{i+1}), & i = 3, \ldots, m \\ Q"_i(t_{i+1}-t_i) = Q"_{i+1}(t_{i+1}-t_{i+1}), & i = 3, \ldots, m \\ b_1(t) + b_2(t) + b_3(t) + b_4(t) = 1, & \forall t \in [0,1], \end{cases}$$

se obţin polinoamele `blending` astfel încât

$$Q_i(t-t_i) = \frac{(1-t)^3}{6}P_{i-3} + \frac{3t^3-6t^2+4}{6}P_{i-2} + \frac{-3t^3+3t^2+3t+1}{6}P_{i-1} + \frac{t^3}{6}P_i,$$

iar matricea de bază B-`spline` este

$$M_{Bs} = \frac{1}{6}\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix}.$$

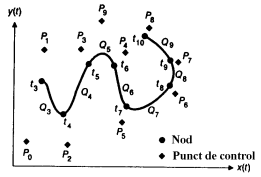



Figura 3.18: O curbă `B-spline` constituită din segmentele de la $Q_1$ la $Q_9$

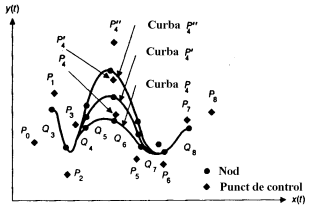



Figura 3.19: Curbele `B-spline` cu punct de control $P_4$ variabil

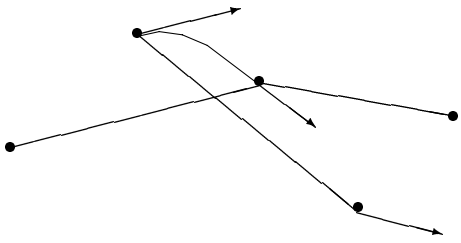


Figura 3.20: Curba `spline Catmull-Rom`

Curbele `B-spline` *neuniforme* diferă de cele uniforme prin faptul că intervalele parametrice dintre noduri succesive nu sunt uniforme (astfel încât funcţiile `blending` diferă de la un segment de curbă la altul). Continuitatea de clasă $C^2$ este redusă la $C^1$ sau $C^0$ (în ultimul caz fiind posibilă interpolarea unui punct de control. Funcţiile blending sunt definite recursiv în termenii unor funcţii de ordin mai mic decât 3.

*Curbele cubice raţionale* sunt definite prin raport de polinoame:

$$x(t) = \frac{X(t)}{W(t)},\ y(t) = \frac{Y(t)}{W(t)},\ z(t) = \frac{Z(t)}{W(t)},$$

unde $X(t),\ Y(t),\ Z(t)$ şi $W(t)$ sunt polinoame cubice a căror puncte de control sunt definite în coordonate omogene. Orice curbă neraţională poate fi transformată într-o curbă raţională adăugând $W(t) = 1$. Polinoamele pot fi de tip `Hermite, Bézier` etc. Dacă polinoamele sunt de tip `B-spline`, curbele raţionale sunt numite curbe `B-spline` raţionale neuniforme (`NURBS`). Curbele raţionale se utilizează din două motive. In primul rând sunt invariante la rotaţii, scalări, translaţii şi transformări perspective a punctelor de control (curbele neraţionale sunt invariante numai la rotaţie, scalare şi translaţie). Ca urmare transformarea perspectivă trebuie aplicată numai punctelor de control care pot fi apoi utilizate pentru generarea transformării perspective a curbei iniţiale (în cazul curbelor neraţionale se generează mai întâi punctele curbbei şi apoi se aplică transformarea perspectivă la fiecare punct generat). Al doilea avantaj este posibilitatea de definire precisă a secţiunilor conice (proprietate utilă în aplicaţii în care este posibilă trasarea unor curbe şi suprafeţe generală, dar şi a conicelor).

In cazul în care se doreşte interpolarea unor puncte 3D se pot utiliza curbele `Catmull-Rom` (sau `Overhauser-spline`). Un membru al acestei familii de curbe `spline` (figura 3.20) este capabil să înterpoleze punctele de la $P_1$ la $P_{m-1}$ dacă se furnizează o secvenţă de puncte de la $P_0$ la $P_m$. In plus, vectorul tangent la punctul $P_i$ este paralel cu linia ce uneşte punctele $P_{i-1}$ şi $P_{i+1}$. Aceste curbe spline nu posedă însă proprietatea închiderii convexe. Reprezentarea curbelor `Catmull-Rom` este realizată prin

$$Q^i(t) = T \cdot M_{CR} \cdot G_{B_i} = \frac{1}{2} T \begin{pmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{pmatrix} \begin{pmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{pmatrix}.$$

### Caracteristici ale curbelor cubice

Caracteristicile curbelor prezentate în secţiunile anterioare sunt sumarizate în tabelul 3.2. Numărul de parametrii de control se referă la constrângerile geometrice plus alţi parametrii precum spaţiul dintre noduri la `spline`-urile neuniforme. Continuitatea care poate fi atinsă se referă la constrângeri precum condiţia ca punctele de control să fie coliniare pentru a permite continuitate $G^1$.

| Proprietate | Hermite | Bézier | B-spline uniforme | B-spline neuniforme | Catmull--Rom |
|---|---|---|---|---|---|
| In închiderea convexă a punctelor de control | Nu | Da | Da | Da | Nu |
| Interpolează anumite puncte de control | Da | Da | Nu | Nu | Da |
| Interpolează toate punctele de control | Da | Nu | Nu | Nu | Da |
| Uşurinţa divizării | Bună | Cea mai bună | Medie | Complicată | Medie |
| Continuitate inerentă a reprezentării | $C^0$ $G^0$ | $C^0$ $G^0$ | $C^2$ $G^2$ | $C^2$ $G^2$ | $C^1$ $G^1$ |
| Continuitate care poate fi atinsă | $C^1$ $G^1$ | $C^1$ $G^1$ | $C^2$ $G^2$ | $C^2$ $G^2$ | $C^1$ $G^1$ |
| Numărul parametrilor care controlează curba | 4 | 4 | 4 | 5 | 4 |

Tabelul 3.2: Proprietăţile curbelor cubice

### 3.8.3 Trasarea curbelor parametrice cubice

Există două metode principale de trasare a curbelor parametrice:

- evaluarea polinoamelor $x(t),\ y(t),\ z(t)$ pentru o diviziune a intervalului [0,1] parcurs de $t$, în variantele:
  (a) evaluare directă (eventual utilizând schema `Horner` pentru determinarea valorilor polinoamelor);
  (b) evaluare iterativă;
- subdivizarea recursivă.

*Metoda evaluării directe* presupune următoarele etape:

1. se citesc coordonatele punctelor de control;
2. pentru fiecare pereche de puncte de control succesive:
   – se parcurge intervalul [0,1] cu un pas constant $\delta = 1/n,\ n > 1$;
   – pentru fiecare $t_i = i\delta,\ i = 1, \ldots, n$ se calculează
     (a) coordonatele punctului de pe curbă $x(t_i),\ y(t_i),\ z(t_i)$;
     (b) coordonatele proiecţiei acestui punct intermediar;
3. se trasează pe ecran o linie poligonală care trece prin proiecţiile punctelor intermediare.

Se observă că pentru evaluarea polinoamelor cubice sunt necesare cel puţin 11 multiplicări şi 10 adunări pentru un punct (vezi procedura ce urmează).

```
type CoefficientArray=array[1..4] of real;
Procedure DrawCurve(
 cx : CoefficientArray, {coeficienţi pt.x(t) : CxM · Gx}
 cy : CoefficientArray, {coeficienţi pt.y(t) : CyM · Gy}
 cz : CoefficientArray, {coeficienţi pt.z(t) : CzM · Gz}
 n :integer)            {număr de paşi}
var
   t, t2, t3 : real;
begin
   MoveTo(cx[4],cy[4],cz[4]); {t=0}
```

```
    d := 1/n;  t := 0;
    for i := 1 to n do
        begin
            t := t + d;  t2 := t * t;  t3 := t2 * t;
            x := cx[1] * t3 + cx[2] * t2 + cx[3] * t + cx[4];
            y := cy[1] * t3 + cy[2] * t2 + cy[3] * t + cy[4];
            z := cz[1] * t3 + cz[2] * t2 + cz[3] * t + cz[4];
            LineTo(x, y, z)
        end
end;
```

Utilizând schema `Horner` pentru evaluarea polinoamelor sunt necesare doar 9 multiplicări şi 10 adunări.

Multiplicările pot fi evitate prin *evaluarea iterativă*. Fie $f(t) := at^3 + bt^2 + ct + d$, $t_n := n\delta$, $f_n := f(t_n)$, $\Delta f(t) := f(t+\delta) - f(t)$, $\Delta^2 f(t) := \Delta(\Delta f(t))$ (diferenţe finite). Se observă că

$$\Delta f(t) = f(t+\delta) - f(t) = 3at^2\delta + t(3a\delta^2 + 2b\delta) + a\delta^3 + b\delta^2 + c\delta,$$
$$\Delta^2 f(t) = \Delta f(t+\delta) - \Delta f(t) = 6a\delta^2 t + 6a\delta^3 + 2b\delta^2,$$
$$\Delta^3 f(t) = \Delta^2 f(t+\delta) - \Delta^2 f(t) = 6a\delta^3$$

şi

$$f_{n+1} = f_n + \Delta f_n, \quad \Delta f_n = \Delta f_{n-1} + \Delta^2 f_{n-1}, \quad \Delta^2 f_{n-1} = \Delta^2 f_{n-2} + \Delta^3 f_{n-2}.$$

Astfel, pornind de la valorile de start

$$f_0 = d, \quad \Delta f_0 = a\delta^3 + b\delta^2 + c\delta, \quad \Delta^2 f_0 = 6a\delta^3 + 2b\delta^2, \quad \Delta^3 f_0 = 6a\delta^3,$$

se pot calcula recursiv valorile $f_n$. Se defineşte vectorul diferenţelor iniţiale, $D$, ca fiind

$$D = \begin{pmatrix} f_0 \\ \Delta f_0 \\ \Delta^2 f_0 \\ \Delta^3 f_0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ \delta^3 & \delta^2 & \delta & 0 \\ 6\delta^3 & 2\delta^2 & 0 & 0 \\ 6\delta^3 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = E(\delta)C.$$

In această variantă, algoritmul pentru trasarea unei curbe parametrice cubice presupune efectuarea unor multiplicări doar în faza de iniţializare a valorilor recursive. La pasul $n$ al algoritmului se efectuează următoarele calcule:

$$\begin{cases} x_{n+1} = x_n + \Delta x_n, & \Delta x_{n+1} = \Delta x_n + \Delta^2 x_n, & \Delta^2 x_{n+1} = \Delta^2 x_n + \Delta^3 x_n; \\ y_{n+1} = y_n + \Delta y_n, & \Delta y_{n+1} = \Delta y_n + \Delta^2 y_n, & \Delta^2 y_{n+1} = \Delta^2 y_n + \Delta^3 y_n; \\ z_{n+1} = z_n + \Delta z_n, & \Delta z_{n+1} = \Delta z_n + \Delta^2 z_n, & \Delta^2 z_{n+1} = \Delta^2 z_n + \Delta^3 z_n; \end{cases}$$

precum şi trasarea dreptei ce uneşte proiecţia punctului $(x_n, y_n, z_n)$ cu proiecţia punctului $(x_{n+1}, y_{n+1}, z_{n+1})$. Algoritmul presupune 9 adunări per pas şi nici o multiplicare.

```
Procedure DrawCurbeFwdDif(
  n, x, dx, d2x, d3x, y, dy, d2y, d3y, z, dz, d2z, d3z)
{ n este numărul de paşi de trasare a curbei,
```

```
  pasul δ fiind 1/n, x, dx, d2x, d3x sunt
  valorile iniţiale pentru x(t) la t = 0, calculate
  prin C_x = E(δ)C_x}
begin
    MoveTo(x, y, z);
    for i := 1 to n do
        begin
            x := x + dx;  dx := dx + d2x;  d2x := d2x + d3x;
            y := y + dy;  dy := dy + d2y;  d2y := d2y + d3y;
            z := z + dz;  dz := dz + d2z;  d2z := d2z + d3z
            LineTo(x, y, z)
        end
end {DrawCurveFwdDif}
```

Dezavantajul utilizării acestui algoritm constă în faptul că pentru valori mari ale numărului de paşi $n$ rezultatele pot fi afectate de cumularea erorilor de trunchiere.

Trasarea unei curbe 3D cu pas constant nu conduce întotdeauna la imaginile dorite (vezi trasarea curbelor plane). Există două modalităţi de eliminare a acestui efect nedorit:

1. creşterea gradului funcţiilor parametrice (se produc o serie de inflexiuni adiţionale);
2. creşterea numărului punctelor de control cu divizarea segmentelor de curbă în mai multe segmente.

Se consideră exemplul unui segment de curbă `Bézier` determinat prin patru puncte de control. Acesta poate fi subdivizat în două segmente având în total şapte puncte de control (cele două segmente noi au un punct comun şi se suprapun pe segmentul original). Dată curba $Q(t)$ definită de punctele $P_1,\ P_2,\ P_3,\ P_4$, se caută punctele $L_1,\ L_2,\ L_3,\ L_4$ care definesc un segment de curbă ce coincide cu $Q(t)$ pentru $0 \le t \le 1/2$ (analog $R_1,\ R_2,\ R_3,\ R_4$ pentru $1/2 \le t \le 1$). Punctul de pe curba `Bézier` care corespunde unui parametru $t$ se poate obţine astfel (figura 3.21.a): se determină punctele $L_2 \in P_1P_2,\ H \in P_2P_3,\ R_3 \in P_3P_4$ care împart segmentele $P_1P_2$, $P_2P_3$ şi $P_3P_4$ în raportul $t/(1-t)$; analog se determină $L_3 \in L_2H$ şi $R_2 \in HR_3$ care împart segmentele menţionate în acelaşi raport şi $L_4 \in L_3R_2$ tot în acelaşi raport. Punctul $L_4$ este punctul căutat. In cazul $t = 1/2$ noile puncte de control pot fi uşor calculate: $L_2 = (P_1 + P_2)/2,\ H = (P_2 + P_3)/2,\ L_3 = (L_2 + H)/2,\ R_3 = (P_3 + P_4)/2,\ R_2 = (H + R_3)/2,\ L_4 = R_1 = (L_3 + R_2)/2$. Noile puncte pot fi rescrise astfel:

$$G_B^L = \frac{1}{8}\begin{pmatrix} 8 & 0 & 0 & 0 \\ 4 & 4 & 0 & 0 \\ 2 & 4 & 2 & 0 \\ 1 & 3 & 3 & 1 \end{pmatrix}\begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{pmatrix}, \quad G_B^R = \frac{1}{8}\begin{pmatrix} 1 & 3 & 3 & 1 \\ 0 & 2 & 4 & 2 \\ 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 8 \end{pmatrix}\begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{pmatrix}.$$

Se observă că noile puncte de control se află în închiderea convexă a setului original de puncte de control. De aceea, în general, aceste puncte sunt mai apropiate de curbă decât punctele de control iniţiale. Această proprietate este valabilă şi pentru curbele `spline` care respectă proprietatea închiderii convexe. In cazul curbelor `B-spline`, se pot construi puncte de control în mod similar,

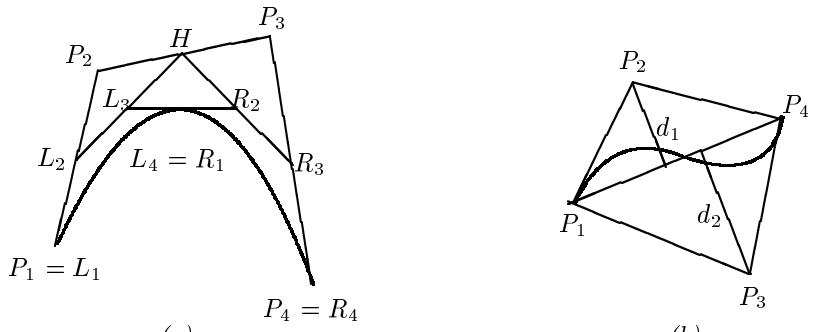



Figura 3.21: Subdivizarea recursivă (a) Subdivizarea unui segment de curbă `Bézier` (b) Testul de liniaritate a segmentului de curbă `Bézier`

rezultând următorii vectori geometrici:

$$G^L_{Bs} = \frac{1}{8}\begin{pmatrix} 4 & 4 & 0 & 0 \\ 1 & 6 & 1 & 0 \\ 0 & 4 & 4 & 0 \\ 0 & 1 & 6 & 1 \end{pmatrix}\begin{pmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{pmatrix},\ G^R_{Bs} = \frac{1}{8}\begin{pmatrix} 1 & 6 & 1 & 0 \\ 0 & 4 & 4 & 0 \\ 0 & 1 & 6 & 1 \\ 0 & 0 & 4 & 4 \end{pmatrix}\begin{pmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{pmatrix}.$$

Cele patru puncte de control sunt înlocuite cu un total de cinci puncte. Noile segmente de curbă depind parţial de vechile puncte de control. Astfel, dacă modificarea unui punct de control din cele patru iniţiale are loc după ce divizarea a fost realizată, este posibil obţinerea unor noi segmente de curbă neconectate (această problemă poate fi evitată utilizând curbe `B-spline` neuniforme).

Algoritmul bazat pe *subdivizarea recursivă* este indicat, în special, în cazul curbelor `Bézier`, deoarece subdivizarea este uşor de realizat şi testul de oprire a recursivităţii este simplu: dacă $P_1$ şi $P_4$ sunt punctele prin care trece segmentul de curbă, iar $P_2$, $P_3$ punctele de control pentru determinarea derivatei, iar distanţa de la $P_2$ la $P_1P_4$ şi distanţa de la $P_3$ la $P_1P_4$ sunt sub un anumit nivel $\varepsilon$ (figura 3.21b), atunci segmentul de curbă $P_1P_4$ este suficient de plat (din proprietatea de incluziune în închiderea convexă a mulţimii punctelor de control) pentru a fi aproximat cu o dreaptă (altfel se continuă subdivizarea).

Dacă se utilizează alt tip de curbe decât `Bézier`, testul de liniaritate a segmentului de curbă este mult mai complicat. Astfel trecerea la forma `Bézier` este recomandată pentru subdivizarea recursivă.

Dată fiind o curbă reprezentată prin vectorul geometric $G_1$ şi matricea de bază $M_1$, căutăm vectorul geometric $G_2$ pentru matricea de bază $M_2$ astfel încât curbele să fie identice: $T \cdot M_2 \cdot G_2 = T \cdot M_1 \cdot G_1$. Astfel $G_2 = M_2^{-1} \cdot M_1 \cdot G_1$. De exemplu, matricea de trecere de la `B-spline` la forma `Bézier` este

$$M_{B_sB} = M_B^{-1} \cdot M_{Bs} = \frac{1}{6}\begin{pmatrix} 1 & 4 & 1 & 0 \\ 0 & 4 & 2 & 0 \\ 0 & 2 & 4 & 0 \\ 0 & 1 & 4 & 1 \end{pmatrix}$$

## 3.9 Suprafeţe curbe

### 3.9.1 Descrierea suprafeţelor curbe

Suprafeţele curbe pot fi descrise:

1. *explicit*, prin ecuaţii de tipul $z = f(x, y)$;
2. *implicit*, printr-o ecuaţie de tipul $F(x, y, z) = 0$;
3. *parametric*, prin ecuaţii de tipul $x = x(s, t),\ y = y(s, t),\ z = z(s, t)$.

*Suprafeţele de rotaţie* sunt acele suprafeţe care se obţin prin rotaţia unui obiect plan în jurul unei drepte. Dacă obiectul generator are ecuaţia parametrică $(x(t), y(t), z(t)),\ t \in [0, 1]$, se află în planul $z = 0$, iar rotaţia se face în jurul axei $y$, atunci suprafaţa de rotaţie poate fi descrisă parametric prin $(x(t)\cos\theta, y(t), -x(t)\sin\theta)$. Suprafeţele de rotaţie sunt cazuri particulare de suprafeţe obţinute prin baleiere spaţială (translatarea unui obiect de-a lungul unei traiectorii).

In grafica pe calculator sunt des utilizate suprafeţele determinate polinomial prin funcţia de două variabile $x$ şi $y$:

$$z = \sum_{j=0}^{n} \sum_{i=0}^{m} a_{ij} x^i y^i.$$

Suprafaţa se notează $P_{m,n}$. Matricea $A$ poartă numele de *amprentă* a suprafeţei. Se disting următoarele cazuri speciale:

(a) $P_{0,0},\ P_{0,1},\ P_{1,0}$ sunt plane;
(b) $P_{1,1}$ este un paraboloid hiperbolic;
(c) $P_{0,n},\ P_{m,0}$ pentru $m, n \geq 2$ sunt suprafeţe cilindrice cu generatoarele paralele cu axa $Ox$ respectiv $Oy$, iar curbele directoare ale suprafeţelor cilindrice sunt conţinute în planele $yOz$, respectiv $xOz$;
(d) $P_{1,n},\ P_{m,1}$ pentru $m, n \geq 1$ sunt suprafeţele riglate, ale căror generatoare sunt conţinute în planele $y = const$, respectiv $x = const$.
(e) $P_{2,2}$ sunt suprafeţe cuadrice;
(f) $P_{3,3}$ sunt suprafeţe bicubice.

Un exemplu concludent de suprafeţe determinate prin polinoame sunt suprafeţele de interpolare, definite prin puncte şi prin curbe marginale. Suprafaţa de interpolare definită de patru puncte de colţ este un patrulater strâmb, o *cuadrică riglată*. Modelarea suprafeţelor de interpolare (sau de aproximare) se bazează pe noţiunea de petic (colecţie de puncte mărginită de patru curbe). Un petic bicubic este descris parametric prin polinoame de grad 3.

Suprafeţele *cuadrice* (hiperboloidul, paraboloidul, sfera, elipsoidul, cilindrul, conul) sunt definite implicit printr-o ecuaţie $F(x, y, z) = 0$ cu $F$ polinom cuadric în $x,\ y$ şi $z$:

$$F(x, y, z) = a_{11}x^2 + a_{22}y^2 + a_{33}z^2 +$$
$$+2a_{21}xy + 2a_{23}yz + 2a_{13}xz + 2a_{14}x + 2a_{24}y + 2a_{34}z + a_{44}.$$

Suprafeţele *parametrice polinomiale bivariate* definesc coordonatele unui punct de pe o suprafaţă utilizând trei polinoame bivariate, câte unul pentru fiecare dimensiune.

O problemă importantă este *cuplarea suprafeţelor*. Aceasta poate fi:

(a) de tip alipire, când se cere identitatea anumitor curbe marginale;
(b) cuplare netedă, ce impune suplimentar identitatea tangentelor (derivate parţiale de ordinul întâi) şi a curburilor (derivate mixte) de-a lungul curbelor marginale.

### 3.9.2 Suprafeţe parametrice bicubice

Folosind două familii de curbe cubice 3D se poate defini o suprafaţă curbă în spaţiu. O familie de curbe cubice 3D se poate obţine introducând, în ecuaţiile parametrice ale unei curbe 3D, a unui parametru nou ce variază de asemenea în intervalul $[0,1]$. Fie curba $Q(s) = SMG$. Dacă $G$ variază funcţie de un parametru $t$ se obţine o familie de curbe cubice:

$$Q(s,t) = SMG(t) = SM \begin{pmatrix} G_1(t) \\ G_2(t) \\ G_3(t) \\ G_4(t) \end{pmatrix}.$$

Dacă $G_i(t)$ sunt polinoame cubice de acelaţi tip ca şi cele în $s$, atunci $G_i(t) = TMG_i$, unde $G_i$ este vectorul de geometrie caracteristic, $G_i = (g_{i1}, g_{i2}, g_{i3}, g_{i4})^T$. Notând $G_x$, $G_y$, $G_z$ matricile $4 \times 4$ ce conţin coordonatele carteziene ale elementelor de geometrie $g_{ij}$, se obţin expresiile parametrice

$$\begin{cases} x(s,t) = SMG_xM^TT^T, \\ y(s,t) = SMG_yM^TT^T, \\ z(s,t) = SMG_zM^TT^T. \end{cases}$$

Astfel, ecuaţia matriceală este

$$Q(s,t) = SM \begin{pmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{pmatrix} M^TT^T = SMGM^TT^T, \ 0 \le s,t \le 1$$

unde $M$ este *matricea de bază*, iar $G$ este *matricea geometrică*.

Vectorul tangent la suprafaţă în direcţia $s$ este

$$\frac{\partial}{\partial s}Q(s,t) = \frac{\partial}{\partial s}(S \cdot M \cdot G \cdot M^T \cdot T^T) =$$

$$= \frac{\partial}{\partial s}(S) \cdot M \cdot G \cdot M^T \cdot T^T = (3s^2, 2s, 1, 0) \cdot M \cdot G \cdot M^T \cdot T^T,$$

şi vectorul tangent în direcţia $t$ este

$$\frac{\partial}{\partial t}Q(s,t) = S \cdot M \cdot G \cdot M^T \cdot (3t^2, 2t, 1, 0)^T.$$

Notăm cu $(x_s, y_s, z_s)$ respectiv $(x_t, y_t, z_t)$ componentele acestor vectori. Atunci normala este

$$\frac{\partial}{\partial s}Q(s,t) \times \frac{\partial}{\partial t}Q(s,t) = (y_sz_t - y_tz_s, z_sx_t - z_tx_s, x_sy_t - x_ty_s).$$

### Suprafeţe Hermite

O suprafaţă bicubică `Hermite` se defineşte pe baza a patru puncte în spaţiu corespunzătoare valorilor extreme 0 şi 1 pentru $s$ şi $t$, precum şi prin câte trei tangente la suprafaţă în fiecare dintre aceste puncte. Fiecare vector tangent are componentele determinate de derivatele de ordin 1 şi 2 ale funcţiilor $x(t,s),\ y(s,t),\ z(t,s)$.

In cazul special când $Q(0,t),\ Q(1,t),\ Q(s,0)$ şi $Q(s,1)$ sunt linii drepte, rezultatul este o suprafaţă riglată. Dacă aceste linii sunt coplanare, atunci suprafaţa `Hermite` se reduce la un poligon planar.

Matricea de geometrie poate fi construită pornind de la familia de curbe `Hermite`:

$$x(s,t) = SM_H G_{Hx}(t) = SM_H \begin{pmatrix} P_1(t) \\ P_4(t) \\ R_1(t) \\ R_4(t) \end{pmatrix}_x$$

In figura 3.22.a sunt trasate cubicele în $s$ definite la $t = 0,\ 0.2,\ 0.4,\ 0.6,\ 0.8, 1$.

Fiecare element al vectorului de geometrie poate fi reprezentat drept o curbă `Hermite`:

$$(P_1(t), P_4(t), R_1(t), R_4(t)) = TM_H \begin{pmatrix} g_{11} & g_{21} & g_{31} & g_{41} \\ g_{12} & g_{22} & g_{32} & g_{42} \\ g_{13} & g_{23} & g_{33} & g_{43} \\ g_{14} & g_{24} & g_{34} & g_{44} \end{pmatrix},$$

unde $g_{ij}$ rezultă din forma `Hermite` pentru curbe. De exemplu, $g_{33_x} = (\partial^2 x)/(\partial s \partial t)(0,0)$, deoarece este vectorul tangent la curba $R_{1x}(t)$ în $t = 0$, care este la rândul său vectorul tangent la curba $x(s,0)$ în $s = 0$. Astfel

$$G_{Hx} = \begin{pmatrix} x(0,0) & x(0,1) & \frac{\partial}{\partial t}x(0,0) & \frac{\partial}{\partial t}x(0,1) \\ x(1,0) & x(1,1) & \frac{\partial}{\partial t}x(1,0) & \frac{\partial}{\partial t}x(1,1) \\ \frac{\partial}{\partial s}x(0,0) & \frac{\partial}{\partial s}x(0,1) & \frac{\partial^2}{\partial s \partial t}x(0,0) & \frac{\partial^2}{\partial s \partial t}x(0,1) \\ \frac{\partial}{\partial s}x(1,0) & \frac{\partial}{\partial s}x(1,1) & \frac{\partial^2}{\partial s \partial t}x(1,0) & \frac{\partial^2}{\partial s \partial t}x(1,1) \end{pmatrix}$$

Elementele matricei geometrice sunt reprezentate în figura 3.22.b.

Condiţiile de joncţiune a două suprafeţe `Hermite` se pun relativ simplu: capetele curbei de margine trebuie să coincidă, iar tangentele la suprafeţe în aceste capete trebuie să fie proporţionale. Astfel matricele de geometrie ale suprafeţelor trebuie să concorde: linia a doua a primei matrice să coincidă cu prima linia a celei de a doua, iar linia a patra a primei matrice să fie proporţională, cu un factor unitar pozitiv, faţă de linia a treia a celei de a doua matrice.

### Suprafeţe Bézier

Pentru a defini o suprafaţă `Bézier` se folosesc cele 4 puncte de control corespunzătoare valorilor $(0,0), (0,1), (1,0), (1,1)$ ale parametrilor $s$ şi $t$ şi în plus alte 12 puncte de control prin intermediul cărora se precizează tangentele la suprafaţă. Astfel geometria unei suprafeţe `Bézier` este caracterizată prin

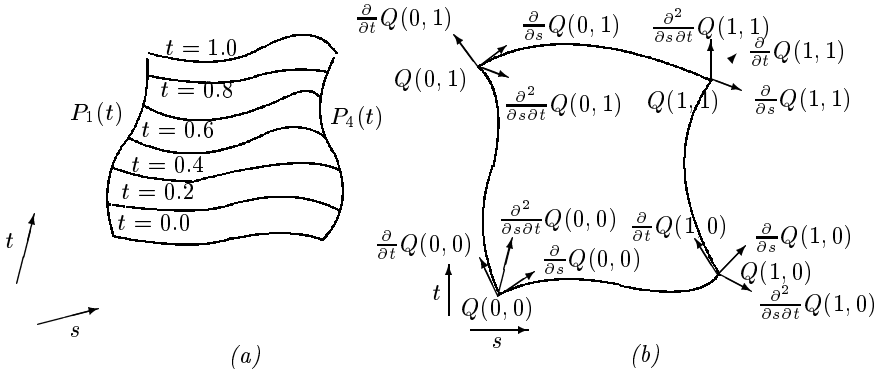



Figura 3.22: Suprafeţe `Hermite` (a) Interpolarea cubică între $P_1(t)$ şi $P_4(t)$ (b) Tangentele la suprafaţă

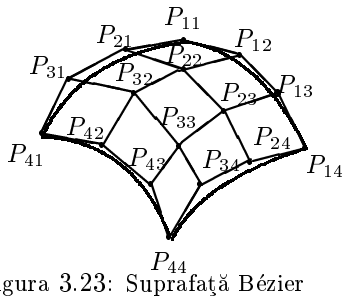



Figura 3.23: Suprafaţă Bézier

coordonatele a 16 puncte de control (figura 3.23). Ecuaţia parametrică a suprafeţelor Bézier este:

$$Q(s,t) = \begin{pmatrix} (1-s)^3 \\ 3s(1-s)^2 \\ 3s^2(1-s) \\ s^3 \end{pmatrix}^T \begin{pmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \\ P_{41} & P_{42} & P_{43} & P_{44} \end{pmatrix} \begin{pmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{pmatrix}.$$

Proprietăţile cele mai importante ale acestei suprafeţe sunt includerea în închiderea convexă a punctelor de control şi posibilitatea simplă de subdivizare (se divizează suprafaţa de-a lungul unui parametru, apoi cele două noi suprafeţe de-a lungul celuilalt parametru). In figura 3.24 sunt prezentate două suprafeţe `Bézier` ce au o latură comună.

### 3.9.3 Trasarea suprafeţelor parametrice bicubice

Pentru generarea imaginii unei suprafeţe bicubice parametrice prin *evaluare directă*, se procedează în modul următor:

1. se parcurg valorile $s$ între 0 şi 1 cu anumit pas $\delta_s = 1/m$;
2. pentru fiecare valoare $s_i = i\delta_s,\ i = 1, \ldots, m$
   - se parcurg valorile lui $t$ între 0 şi 1 cu anumit pas $\delta_t = 1/n$;

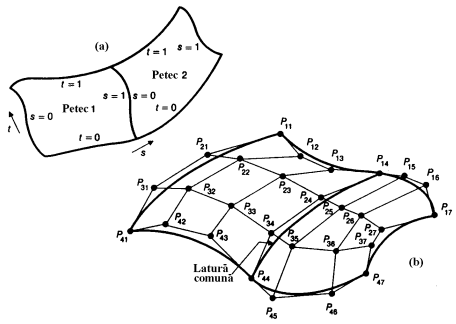



Figura 3.24: (a) Două petece oarecare de suprafaţă cu o latură comună (b) Două suprafeţe `Bézier` cu patru punct de control identice

- pentru fiecare valoare $t_j = j\delta_t,\ j = 1, \ldots, n$ se calculează
  (a) coordonatele punctului $x(s_i, t_j),\ y(s_i, t_j),\ z(s_i, t_j)$;
  (b) proiecţia punctului calculat anterior;

3. se trasează curbele din cele două familii (pentru fiecare $s$, respectiv pentru fiecare $t$) prin punctele calculate, ca linii poligonale.

Acurateţea redării suprafeţei depinde de alegerea paşilor incrementali (figura 3.25).

Evaluarea directă pentru suprafeţe este mult mai costisitoare decât în cazul curbelor deoarece ecuaţiile suprafeţei trebuie evaluate de $2/\delta^2$ ori. Pentru $\delta = 0.1$ această valoare este 200, acceptabil, dar pentru $\delta = 0.01$ este 20000. Astfel este de preferat să fie utilizată evaluarea iterativă.

Valorile funcţiilor polinomiale se pot calcula *iterativ*, folosind diferenţe finite şi urmărind algoritmul propus la trasarea curbelor spaţiale. Dacă în cazul curbelor s-a considerat matricea de start $D = E(\delta) \cdot C$, în cazul suprafeţelor se consideră

$$DD_x = E(\delta_s) \cdot C_x \cdot E(\delta_t)^T$$

unde $\delta_s$ este mărimea pasului pe $s$, $\delta_t$ este mărimea pasului pe $t$, iar $C_x$ este matricea $4 \times 4$ a coeficienţilor lui $x(s,t)$. Matricea $DD_x$ are în prima linie valorile $x(0,0),\ \Delta_t x(0,0),\ \Delta_t^2 x(0,0),\ \Delta_t^3 x(0,0)$. Astfel, această linie poate fi folosită pentru trasarea curbei cu proiecţia $x(0,t)$ pe axa $x$, cu increment $\delta_t$. Cum celelalte linii ale lui $DD_x$ corespund diferenţelor de ordin 1, 2 şi 3 în $s$ a

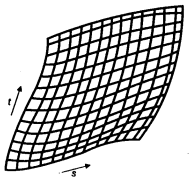



Figura 3.25: Petec de suprafaţă trasat prin curbe cu $s$ sau $t$ constant

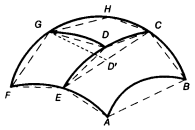



Figura 3.26: Probleme în subdivizarea recursivă: urmărind algoritmul vor fi afişate patrulaterele $ABCE$, $EFGD$ şi $GDCH$; corectă este afişarea patrulaterelor $ABCE$, $EFGD'$ şi $GD'CH$

primei linii, aplicând următoarea transformare liniilor lui $DD_x$:

linia1:=linia1+linia2, linia2:=linia2+linia3, linia3:=linia3+linia4,

în prima linie a lui $DD_x$ se va obţine $x(\delta_s, 0)$, $\Delta_t x(\delta_s, 0)$, $\Delta_t^2 x(\delta_s, 0)$ şi $\Delta_t^3 x(\delta_s, 0)$. Aceste valori pot fi utilizate pentru trasarea curbei cu proiecţia $x(\delta_s, t)$ pe $x$. Pasul este repetat pentru a obţine pe prima linie a lui $DD_x$ valoarea $x(2\delta_s, t)$ etc. Trasarea curbelor cu $t$ constant se face în mod similar. Pentru a calcula $x(s, 0)$ se transpune matricea $DD_x$ şi se aplică procedeul de mai sus.

Pentru algoritmul *subdivizării recursive* testul de suprafaţă plată pentru o suprafaţă `Bézier` este realizat prin determinarea planurilor, care trec prin câte trei puncte de control ce aparţin suprafeţei, şi determinarea distanţelor de la celelalte 13 puncte de control la aceste plane: distanţa maximă trebuie să fie sub anumit nivel $\varepsilon$ pentru ca subdivizarea în patru noi suprafeţe să nu fie necesară. O suprafaţă aproape plată, specificată, de exemplu, prin colţurile $A$, $B$, $C$, $D$, poate fi trasată fie prin muchiile cuadrilaterului $ABCD$, fie prin cele patru triunghiuri determinate de muchii şi punctul interior $P$, definit ca medie a punctelor de colţ (figura 3.26).

## 3.10 Decupare

In cazul construirii unei imagini ce depăşeşte limitele zonei de lucru de pe ecran sau al spaţiului vizibil, vor apărea o serie de primitive care vor intersecta frontiera (ecranului sau spaţiului vizibil). Este necesară eliminarea din aceste primitive a porţiunilor care ies din zona observabilă. Această operaţie este cunoscută sub numele de *decupare* (`clipping`, retezare, tăiere).

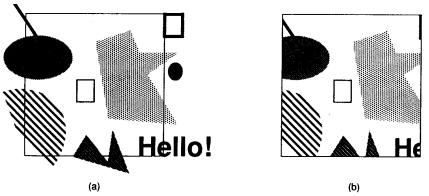



Figura 3.27: Decupare 2D (a) Primitivele şi dreptunghiul de decupare (b) Rezultatul decupării

*Procesul de decupare 2D decide care parte, dacă există, a unei primitive se află în interiorul unei ferestre sau a unei zone de lucru* (figura 3.27).

Decuparea poate fi realizată:

1. analitic;
2. în timpul conversiei de scanare (combinarea primelor două tehnici fiind numită *forfecare*) ;
3. prin copierea pixelilor unui poligon dintr-o regiune de memorie ce stochează primitivele nedecupate într-o altă regiune de memorie.

In cazul tehnicii de forfecare fiecare pixel generat pentru o primitivă este testat faţă de regiunea de decupare. Complicaţii apar dacă regiunea de decupare nu este dreptunghiulară. Deoarece decuparea analitică este uşor de realizat în cazul liniilor şi poligoanelor, decuparea acestora este realizată de obicei înanintea convertirii prin scanare, pe când alte primitive sunt decupate în timpul conversiei.

Sarcina procesului clasic este eliminarea acelor părţi ale primitivelor care sunt în afara ferestrei din `WCS` (sistemul coordonatelor lumii reale) şi nu din `NDCS` (sistemul coordonatelor terminalului normalizate). Se evită, în acest caz, costul transformării de normalizare pentru orice primitivă care nu apare pe ecran. Primitivele care se află într-o fereastră sunt transformate şi transferate într-o zonă de lucru, iar celor care sunt parţial sau complet în afara ferestrei li se aplică un algoritm de decupare.

Algoritmii de decupare sunt aplicaţi şi în cazul selectării unor părţi a primitivelor dintr-o zonă de lucru în vederea copierii, mutării sau ştergerii.

Dificultatea procesului depinde de tipul primitivei căreia i se aplică

(deoarece o curbă generală satisface o ecuaţie neliniară, calcularea punctelor de intersecţie cu marginea ferestrei necesită utilizarea unor metode numerice).

Decuparea 3D este efectuată asupra corpurilor 3D pentru încadrarea în volumul de vedere. Se elimină prin acest procedeu proiectarea unor primitive care nu se încadrează în fereastră (şi ulterior în zona de lucru).

Se consideră în cele ce urmează problema decupării bidimensionale a unei primitive plane faţă de o fereastră convexă.

### 3.10.1 Decuparea unui punct

Se presupune că fereastra este dreptunghiulară şi are limitele: superioară $y = y_{max}$, inferioară $y = y_{min}$, laterală stângă $x = x_{min}$, laterală dreaptă $x = x_{max}$. Pentru ca un punct $P(x, y)$ să fie vizibil în fereastră trebuie satisfăcute relaţiile

$$x_{min} \leq x \leq x_{max}, \qquad y_{min} \leq y \leq y_{max}.$$

### 3.10.2 Decuparea segmentelor de linii

In cazul segmentelor, problema se complică. O soluţie pentru cazul când desenăm pe întreg ecranul o reprezintă folosirea unei instrucţiuni de eroare imediat după apelarea instrucţiunii de trasare a segmentului în afara spaţiului vizibil. Această soluţie nu rezolvă problema în cazul desenării într-o fereastră mai mică decât ecranul.

O altă variantă este utilizarea unei subrutine de trasare a segmentelor de dreaptă care testează înainte de aprinderea unui pixel, vizibilitatea acestuia faţă de fereastră.

Un segment de dreaptă se află faţă de o fereastră într-una din următoarele patru situaţii (figura 3.28.a):

1. complet în interior (segmentul $AB$),
2. complet în exterior (segmentele $EF$ şi $IJ$),
3. un capăt în interior, celălalt în exterior (segmentul $CD$),
4. ambele capete în exterior şi o parte a segmentului, în interior (segmentul $GH$).

Un algoritmul simplu este următorul:

- dacă capetele segmentului sunt în interior, trasează segmentul;
- dacă numai un capăt este în interior, determină punctul de intersecţie cu marginea ferestrei şi trasează subsegmentul de interior;
- altfel, calculează intersecţiile cu liniile ce definesc marginea ferestrei şi dacă punctele de intersecţie sunt pe segmentele de dreaptă ale marginii ferestrei, trasează subsegmentul dintre acestea.

#### Teste de acceptare/respingere

Se consideră cazul unei ferestre dreptunghiulare. Algoritmul de mai sus poate fi perfecţionat prin efectuarea unor teste simple asupra capetelor segmentului. Segmentul care are ambele capete vizibile, este vizibil în întregime (fereastra este o mulţime convexă). Testul care verifică dacă ambele capete ale segmentului sunt în fereastră, poartă numele de *test de acceptare*. Orice segment

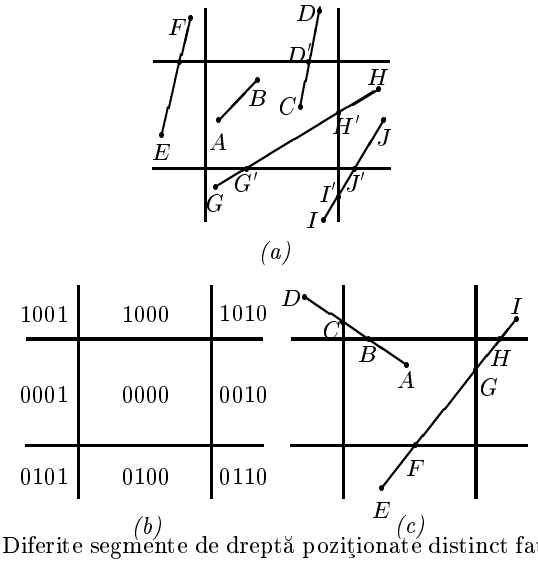



Figura 3.28: (a) Diferite segmente de dreptă poziţionate distinct faţă de fereastră (b) Cele nouă regiuni din algoritmul `Cohen-Sutherland` (c) Exemplu pentru algoritmul `Cohen-Sutherland`

care are ambele capete de aceeaşi parte a ferestrei (sus, jos, stânga, dreapta) este complet invizibil. Testul care verifică această condiţie se numeşte *test de respingere*. Prin adăugarea testelor de acceptare şi respingere la algoritmul de trasare a unui segment de dreaptă se obţine un algoritm a cărui eficienţă se manifestă cu precădere în 2 situaţii:

(a) dacă majoritatea segmentelor de reprezentat se află în interiorul ferestrei;
(b) dacă majoritatea segmentelor de reprezentat se află în afara ferestrei.

### Algoritmul Cohen-Sutherland

Dacă o linie nu este nici respinsă, nici acceptată, atunci se calculează într-o ordine prestabilită intersecţiile cu dreptele: $y = y_{max}$, $x = x_{max}$, $y = y_{min}$, $x = x_{min}$ îndepărtând porţiunile de segment care se situează în afara ferestrei (dreptunghiulare) . Rezultă astfel un nou segment, iar procedura se bucleză asupra ei însăşi până când un segment rezultat este acceptat sau respins.

Se divide `WCS` în nouă regiuni determinate de marginile ferestrei (figura 3.28.b). In prezentarea clasică, algoritmul `Cohen-Sutherland` asociază fiecărui capăt al unui segment un cod de 4 cifre:

(a) prima cifră este 1 dacă punctul este deasupra ferestrei, 0 altfel;
(b) a doua cifră este 1 dacă punctul este sub fereastră, 0 altfel;
(c) a treia cifră este 1 dacă punctul este în dreapta ferestrei, 0 altfel;
(d) a patra cifră este 1 dacă punctul este în stânga ferestrei, 0 altfel.

Puterea algoritmului constă în faptul că odată determinate codurile asociate capetelor segmentului, testele de acceptare sau respingere se realizează prin

operaţii logice rapide (de exemplu, în limbajul C: testul de acceptare are succes dacă $cod_1|cod_2$ este zero, iar testul de respingere, dacă $cod_1 \& cod_2$ nu este zero). Astfel, segmentul $AB$ din figura 3.28 (a) este acceptat, iar $EF$ respins. Rămân de analizat cazurile în care capetele segmentului sunt în regiuni diferite.

Se consideră că primul capăt al segmentului $(x_1, y_1)$ este în afara ferestrei (eventual printr-o inversare). Atunci codul asociat este nenul. Se determină primul bit nenul, aflând astfel de care parte a ferestrei se află primul capăt. Presupunem că primul bit este nenul, deci $y_1 > y_{max}$. Se elimină porţiunea din segment ce se află deasupra liniei $y = y_{max}$. Fie $(x_1', y_1')$ punctul de intersecţie cu $y = y_{max}$. Se determină codul punctului nou, necesar următorului pas. Analog se procedează pentru partea de jos, stânga şi dreapta ferestrei.

Se consideră cele două segmente din figura 3.28 (c). Punctul $A$ are codul 0000, iar punctul $D$, 1001. Segmentul nu poate fi acceptat sau respins. Codul lui $D$ indică faptul că segmentul trece peste latura de sus şi peste latura stângă a ferestrei. Urmărind ordinea biţilor din codul lui $D$ se determină, prima dată, intersecţia segmentului cu latura de sus. Codul punctului $B$ de intersecţie este 0000, astfel încât segmentul $AB$ este acceptat. Segmentul $EI$ necesită iteraţii multiple. Cum $E$ are codul 0100<1010, codul lui $I$, se consideră primul punct exterior. Testând codul lui $E$, se determină intersecţia $F$ a lui $EI$ cu dreapta suport a laturii de jos. Deoarece codul lui $F$ este 0000, iar al lui $I$ este 1010, testul de acceptare/respingere nu are succes. Punctul exterior este $I$. Testând codul lui $I$ de la stânga la dreapta se determină laturile ferestrei pe care $FI$ le poate intersecta. Prima este latura de sus. Se determină $H$, punctul de intersecţie. Codul lui $H$ este 0010. Se face o diviziune a segmentului obţinând $FG$ care este acceptat.

Algoritmul complet este descris în procedura următoare.

```
Procedure CohenSutherlandLineClipAndDraw(
  x0, y0, x1, y1, xmin, xmax, ymin, ymax:real; value:integer);
  {Algoritmul CS pentru decuparea segmentul de linie de
  la P0 = (x0, y0) la P1 = (x1, y1) faţă de dreptunghiul cu
  diagonala de la (xmin, ymin) la (xmax, ymax)}
type
  edge=(Left,Right,Bottom, Top);
  outcode= set of edge;
var
   accept, done : boolean;
   outcode0, outcode1, outcodeOut : outcode;
   {Coduri pentru P0, P1 şi punctul de intersecţie
   cu latura ferestrei}
   x, y : real;
   Procedure CompOutCode(x, y :real; var code : outcode)
   {Calculează codul pentru punctul (x, y)}
   begin
        code:=[];
        if y > ymax then code:=[Top]
        else if y < ymin then code:=[Bottom];
        if x > xmax then code := code+[Right]
        else if x < xmin then code := code+[Left]
```

```
   end;
begin
   accept:=false; done:=false;
   CompOutCode(x0, y0, outcode0);
   CompOutCode(x1, y1, outcode1);
   repeat
     if (outcode0 =[]) and (outcode1 =[]) then {Accept.triv.}
        begin accept :=true; done :=true end
     else if (outcode0 * outcode1) <>[] then
        done :=true {Resp.triv.}
     else
        {Calcul de intersecţii}
        begin
             {Determină capătul exterior}
             ifoutcode0 <>[]
               then outcodeOut := outcode0
               else outcodeOut := outcode1;
             {Determină intersecţia}
             if Top in outcodeOut then
               begin{Intersecţie cu latura de sus}
                     x := x0 + (x1 - x0) * (ymax - y0)/(y1 - y0);
                     y := ymax
               end
             else if Bottom in outcodeOut then
               begin {Intersecţie cu latura de jos}
                     x := x0 + (x1 - x0) * (ymin - y0)/(y1 - y0);
                     y := ymin
               end
             else if Right in outcodeOut then
               begin {Intersecţie cu latura din dreapta}
                     y := y0 + (y1 - y0) * (xmax - x0)/(x1 - x0);
                     x := xmax
               end
             else if Left in outcodeOut then
               begin {Intersecţie cu latura din stânga}
                     y := y0 + (y1 - y0) * (xmax - x0)/(x1 - x0);
                     x := xmax
               end;
             if (outcodeOut = outcode0) then
               begin
                 x0 := x; y0 := y; CompOutCode(x0, y0, outcode0)
               end
             else
               begin
                 x1 := x; y1 := y; CompOutCode(x1, y1, outcode1)
               end
        end
   until done;
   if accept then Trasare(x0, y0, x1, y1, value)
```

```
end; {CohenSutherlandLineClipAndDraw}
```

Algoritmul, în această variantă comună, nu este eficient: deoarece testele şi decuparea se efectuează într-o anumită ordine, se fac adesea operaţii suplimentare (de exemplu, determinarea punctului $H$ de pe $EI$). Algoritmul `Nicholl-Lee-Nicholl` ocoleşte calculul intersecţiilor externe ferestrei.

### Algoritmul bisecţiei

Algoritmul elimină calculele necesare pentru determinarea intersecţiilor. La început se folosesc testele de respingere şi acceptare trivilă. Dacă un segment nu verifică nici unul din aceste teste, el este împărţit în două jumătăţi egale. Cele două jumătăţi sunt analizate din nou şi procesul continuă până la acceptare sau respingere. De obicei, la fiecare iteraţie se poate accepta sau respinge jumătate din segmentul analizat. Acest algoritm ajunge la un rezultat după maximum $\log_2 N$ paşi, unde $N$ este dimensiunea maximă a segmentului (pe orizontală sau verticală), măsurată în pixeli.

### Algoritmul de decupare a liniilor parametrice (algoritmul Cyrus-Beck)

Metoda este utilizată pentru a tăia o linie bidimensională printr-un dreptunghi sau un poligon convex, sau o linie tridimensională printr-un poliedru. Algoritmul `Cohen-Sutherland` determină intersecţiile segmentului de linie cu liniile suport ale laturilor poligonului-fereastră substituind valorile coordonatelor astfel obţinute în ecuaţia segmentului de dreaptă. Algoritmul liniei parametrice determină valoarea parametrului $t$ din reprezentarea parametrică a segmentului de linie pentru punctele în care segmentul intersectează liniile infinite pe care se află laturile ferestrei. Dacă poligonul este un dreptunghi, se calculează patru valori ale parametrului $t$, care apoi sunt comparate pentru a determina acelea care reprezintă puncte interioare segmentului. Se reduce timpul faţă de algoritmul `Cohen-Sutherland` deoarece se evită saltul repetitiv necesar pentru a tăia segmentul prin laturile multiple ale dreptunghiului.

Se consideră că parcurgerea laturilor poligonului se efectuează în sens trigonometric. Semiplanul interior asociat unei laturi este semiplanul determinat de aceasta şi care conţine fereastra. Normala la o latură se consideră, în acest caz, orientată spre exteriorul ferestrei.

In figura 3.29 (a) s-au notat cu $P_0$, $P_1$ capetele segmentului şi $E_i$ o latură a ferestrei. Linia suport a segmentului este reprezentată parametric

$$P(t) = P_0 + (P_1 - P_0)t.$$

Se presupune că $x_0 < x_1$. Se consideră $P_{E_i}$ un punct de pe latura $E_i$, iar $N_i$ vectorul normal la latură. Datorită orientării normalei înspre exterior şi datorită sensului trigonometric de parcurgere a laturilor, produsul scalar $N_i \cdot [P(t) - P_{E_i}]$ este negativ pentru un punct din interiorul ferestrei, pozitiv pentru un punct din exteriorul ferestrei şi zero pentru un punct de pe latura ferestrei. Atunci punctul de intersecţie corespunde unui $t_i$ ce satisface

$$N_i \cdot [P(t_i) - P_{E_i}] = 0$$

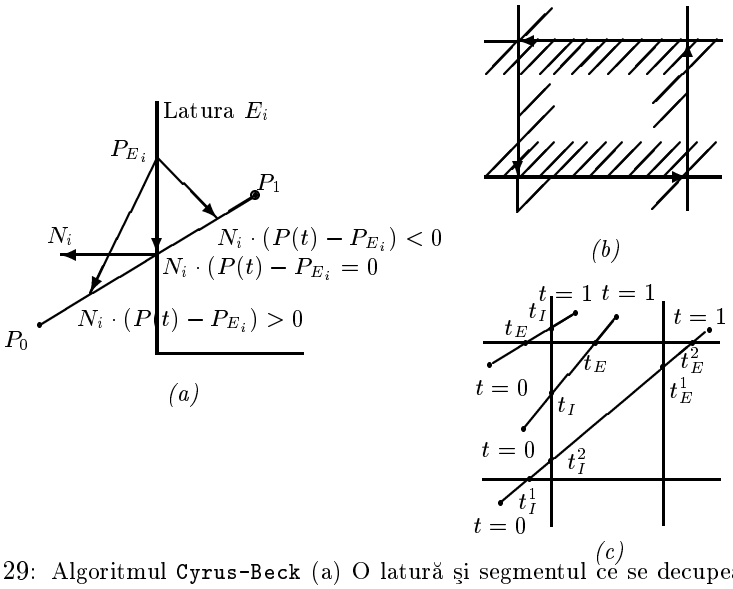



Figura 3.29: Algoritmul `Cyrus-Beck` (a) O latură şi segmentul ce se decupează (b) Semiplanele interioare definite de laturi (c) Exemplu

unde $\cdot$ reprezintă produsul scalar. Deoarece

$$N_i \cdot [P_0 + (P_1 - P_0)t_i - P_{E_i}] = N_i \cdot (P_0 - P_{E_i}) + N_i \cdot (P_1 - P_0)t_i = 0,$$

rezultă

$$t_i = \frac{N_i \cdot (P_0 - P_{E_i})}{-N_i \cdot (P_1 - P_0)}.$$

Pentru fiecare segment se determină cele patru valori $t_i$ corespunzătoare liniilor suport ale laturilor ferestrei, $E_i$. Pasul următor constă în determinarea valorilor ce corespund intersecţiilor cu laturile. Se adaugă la listă $t = 0$ şi $t = 1$. Valorile $t \notin [0, 1]$ sunt eliminate deoarece nu conduc la puncte din interiorul segmentului $P_0P_1$. Valorile $t$ rămase sunt sortate (maxim şase puncte). Se determină punctele, $t_I$, în care segmentul "intră" în interiorul unui semiplan, prin $N_i \cdot (P_1 - P_0) < 0$ (unghi mai mare decât $\pi/2$) şi punctele $t_E$ unde segmentul iese în exterior, prin $N_i \cdot (P_1 - P_0) > 0$ (unghi mai mic decât $\pi/2$). Se consideră $t = 0$ cel mai mic $t_I$, iar $t = 1$ cel mai mare $t_E$. Segmentul interior ferestrei este definit de cel mai mare $t_I$ şi cel mai mic $t_E$. In cazul primei linii din figura 3.29 (c), cel mai mic $t_I$ este mai mare decât cel mai mare $t_E$, astfel încât nici o porţiune din segment nu se află în interiorul dreptunghiului.

In calcule $P_{E_i}$ poate fi considerat unul dintre capetele laturii. Tabelul 3.3 indică pentru fiecare latură valorile lui $N_i$, coordonatele unui punct arbitrar a laturii, $P_{E_i}$, vectorul $P_0 - P_{E_i}$ şi parametrul $t$. Se observă că alegerea lui $P_{E_i}$ pe latură nu influenţează valoarea lui $t$. Produsul $N_i \cdot (P_1 - P_0)$, care determină dacă intersecţia este o intrare sau ieşire din semiplan, se reduce la $\pm(x_1 - x_0)$ sau $\pm(y_1 - y_0)$; astfel, dacă $dx = (x_1 - x_0)$ este pozitiv, linia trasată de la

| Latura de decupare | $N_i$ | $P_{E_i}$ | $P_0 - P_{E_i}$ | $t = \frac{N_i \cdot (P_0 - P_{E_i})}{-N_i \cdot (P_1 - P_0)}$ |
|---|---|---|---|---|
| stânga: $x = x_{min}$ | (-1,0) | $(x_{min}, y)$ | $(x_0 - x_{min}, y_0 - y)$ | $\frac{-(x_0 - x_{min})}{(x_1 - x_0)}$ |
| dreapta: $x = x_{max}$ | (1,0) | $(x_{max}, y)$ | $(x_0 - x_{max}, y_0 - y)$ | $\frac{-(x_0 - x_{max})}{(x_1 - x_0)}$ |
| jos: $y = y_{min}$ | (0,-1) | $(x, y_{min})$ | $(x_0 - x, y_0 - y_{min})$ | $\frac{-(y_0 - y_{min})}{(y_1 - y_0)}$ |
| sus: $y = y_{max}$ | (0,1) | $(x, y_{max})$ | $(x_0 - x, y_0 - y_{max})$ | $\frac{-(y_0 - y_{max})}{(y_1 - y_0)}$ |

Tabelul 3.3: Calculele pentru algoritmul de decupare a liniilor parametrice

stânga la dreapta are o intersecţie de tip $I$ cu latura stângă şi de tip $E$ cu latura dreaptă a ferestrei.

Algoritmul este descris în procedura ce urmează.

```
Procedure Clip2D(var x0, y0, x1, y1 :real; var visible :boolean);
{ Decupează segmentul de linie cu capetele (x0, y0) şi (x1, y1)
 faţă de dreptunghiul cu colţurile (xmin, ymin) şi (xmax, ymax);
 visible va fi adevărat dacă segmentul decupat va fi returnat
 prin parametrii declaraţi cu var; dacă linia este respinsă,
 coordonatele vârfurilor nu se schimbă, iar visible este
 setat pe fals}
var
   tI, tE, dx, dy:  real;
   function Clipt(denom, num :real; var tI, tE:real):boolean;
   var
      t : real;
      accept : boolean;
   begin
      accept :=true;
      if denom > 0 then {Intersectie tip I}
         begin
              t := num/denom; {Valoarea lui t al intersecţiei}
              if t > tE then
                 accept :=false {Segmentul va fi respins}
              else if t > tI then
                 tI := t {A fost găsit un nou tI}
         end
      else if denom < 0 then
         begin
              t := num/denom; {Valoarea lui t al intersecţiei}
              if t < tI then accept :=false
              else if t < tE then {A fost găsit un nou tE}
                                  tE := t
         end
      else
```

```
            if num > 0 then {Linia în afara laturii}
                 accept :=false;
            Clipt := accept
      end; {Clipt}
   begin
      dx := x1 − x0; dy := y1 − y0;
      visible :=false;
      if (dx = 0) and (dy = 0) and
         ClipPoint(x0, y0) {Adevărat dacă (x0, y0) interior}
      then
         visible :=true
      else
         begin
            tI := 0; tE := 1;
            if Clipt(dx, xmin − x0, tI, tE) then
               if Clipt(−dx, x0 − xmax, tI, tE) then
                  if Clipt(dy, ymin − y0, tI, tE) then
                     if Clipt(−dy, y0 − ymax, tI, tE) then
                      begin
                        visible :=true;
                        if tE < 1 then
                         begin {Calculează intersecţia de tip E}
                           x1 := x0 + tE ∗ dx;
                           y1 := y0 + tE ∗ dy;
                         end;
                        if tI > 0 then
                         begin {Calculează intersecţia de tip I}
                           x0 := x0 + tI ∗ dx;
                           y0 := y0 + tI ∗ dy;
                         end
                      end
         end
   end; {Clip2D}
```

Algoritmul Cohen-Sutherland este eficient când testarea codurilor capetelor de linii poate fi efectuat uşor (de exemplu prin operaţii pe bit) şi testele de acceptare sau respingere sunt aplicabile la majoritatea segmentelor. Decuparea liniilor parametrice este mai rapidă când numărul segmentelor de decupat este mare.

### Algoritmul Nicholl-Lee-Nicholl

Se dă un segment de dreaptă $PQ$ şi un dreptunghi de decupare, ca în figura 3.30. Determinând poziţia relativă a lui $Q$ faţă de liniile ce trec prin $P$ şi colţurile ferestrei, se află care laturi ale ferestrei intersectează $PQ$.

Se studiază poziţia lui $P$, determinând sectorul (din cele nouă posibile) în care se află. Funcţie de această poziţie se efectuează diferite teste asupra lui $Q$. Presupunem că $P$ se află în regiunea de stânga-jos (figura 3.30). Dacă $Q$ se

află sub $y_{min}$ sau la stânga lui $x_{min}$, atunci $PQ$ nu poate intersecta fereastra. Acelaşi lucru se poate afirma şi dacă $Q$ se află la stânga liniei de la $P$ la colţul de stânga sus sau dacă $Q$ se află la dreapta liniei de la $P$ la colţul din dreapta jos. Se verifică de asemenea poziţia lui $Q$ relativ la vectorul de la $P$ la colţul din stânga-jos. Fie cazul în care $Q$ este deasupra. Dacă $Q$ se află sub $y_{max}$, $Q$ este în interiorul dreptunghiului sau la stânga: $PQ$ intersectează latura $x = x_{min}$ sau/şi latura $x = x_{max}$. Dacă $Q$ se află peste $y_{max}$, $Q$ este în regiunea de sus sau stânga-sus. Dacă $Q$ este deasupra liniei dintre $P$ şi colţul dreapta-sus, $PQ$ intersectează $x = x_{min}$ şi $y = y_{max}$, altfel $x = x_{min}$ şi $x = x_{max}$.

De exemplu, pentru a testa dacă $Q$ este în stânga segmentului de la $P$ la colţul de dreapta-sus, $(x_{max}, y_{max})$, se verifică dacă

$$(y_Q - y_P)(x_{max} - x_P) > (y_{max} - y_P)(x_Q - x_P).$$

Dacă rezultatul este pozitiv, se calculează intersecţia cu latura de sus a ferestrei. In caz contrar, se determină intersecţia cu latura din dreapta: ordonata $y$ a intersecţiei este

$$y_P + (y_Q - y_P)\frac{x_{max} - x_P}{x_Q - x_P}$$

Calcule similare se efectuează pentru celelalte colţuri. Valorile $y_Q - y_P$, $x_Q - x_P$ se repetă şi pot fi calculate o singură dată.

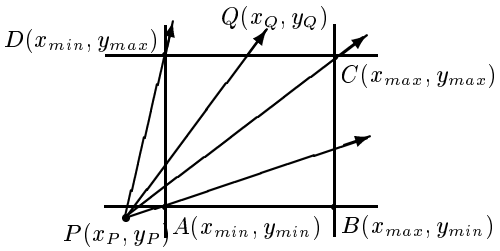


Figura 3.30: Algoritmul Nicholl-Lee-Nicholl

```
Procedure PartialNLNclip(var P,Q : point);
{ funcţionează în cazul în care P este în regiunea de
  jos-stânga; celelalte cazuri sunt similare}
var
   vizibil :boolean; {adevărat dacă segmentul decupat este nenul}
begin
   if Q.y < yMin then vizibil :=false
      else if Q.x < xMin then vizibil :=false
           else if LeftSide(Q,raza de la P la colţul jos-stânga) then
                begin
                     if Q.y <= yMax then
                       begin
                        vizibil :=true;
                        P :=Intersection(PQ,latura stânga);
                        if Q.x > xMax then
```

```
                        Q :=Intersection(PQ,latura dreapta);
                    end
                  else
                    begin
                     if LeftSide(Q,raza de la P la colţul sus-stânga)
                     then vizibil :=false;
                     else if Q.x < xMax then
                       begin
                        vizible :=true;
                        P :=Intersection(PQ, latura stânga);
                        Q :=Intersection(PQ, latura sus);
                       end
                     else if LeftSide(Q,raza P - colţ sus-dreapta) then
                       begin
                        vizible :=true;
                        P :=Intersection(PQ, latura stânga);
                        Q :=Intersection(PQ, latura sus);
                       end
                     else
                       begin
                        vizible :=true;
                        P :=Intersection(PQ, latura stânga);
                        Q :=Intersection(PQ, latura dreapta);
                       end
                    end
              end
    else {caz Q în dreapta liniei P - colţ jos-stânga}
    ...
end;
```

Algoritmul presupune mai puţine împărţiri şi doar o treime din numărul de comparaţii necesare algoritmului `Cohen-Sutherland`.

### 3.10.3 Decuparea poligoanelor

Rezultatul decupării faţă de o fereastră poligonală convexă este: niciunul, unul (cazul poligon decupat convex) sau mai multe poligoane (cazul poligon decupat concav).

#### Teste de acceptare/respingere

Considerăm o linie poligonală cu un număr mare de segmente. Abordarea clasică a clippingului presupune aplicarea algoritmului la fiecare segment de dreaptă în parte. Efortul de calcul este mare, mai ales dacă linia poligonală se află în exteriorul ferestrei. *Metoda ariei mărginite* (`bounding boxes`) determină în prealabil cel mai mic dreptunghi care încadrează linia poligonală – se determină colţurile acestuia ca valori minime şi maxime ale vârfurilor liniei

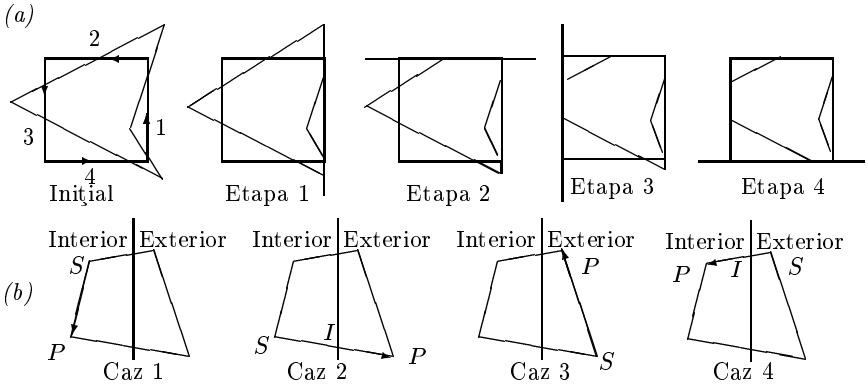



Figura 3.31: Algoritmul `Sutherland-Hodgman` (a) Etapele algoritmului (b) Modificarea listei vârfurilor

poligonale. Dacă dreptunghiul este în afara ferestrei, linia poligonală nu este trasată. Dacă este complet în interior, linia poligonală este trasată în întregime. Dacă dreptunghiul şi fereastra au doar o porţiune comună se trece la decupare pe fiecare segment de dreaptă în parte.

### Algoritmul Sutherland-Hodgman

Algoritmul utilizează o strategie `divide - et - impera` (`divide - and - conquer`) pentru a decupa un poligon convex sau concav relativ la un poligon convex care defineşte fereastra. Intrarea în algoritm este o listă a vârfurilor, iar ieşirea are o structură similară. Algoritmul comportă un număr de etape egal cu numărul laturilor poligonului-fereastră (patru în cazul unui dreptunghi). Intr-o etapă, poligonului curent i se aplică tehnica de tăiere relativă la una din laturile poligonului-fereastră, rezultând o listă modificată de vârfuri (figura 3.31.a). La fiecare etapă se parcurg un număr de paşi egali cu numărul laturilor poligonului care se taie. In fiecare pas se adaugă la lista de vârfuri 0, 1 sau 2 vârfuri noi ce aparţin laturii curente a poligonului-fereastră.

Se consideră latura poligonului de decupat ce porneşte de la vârful $S$ la vârful $P$ (figura 3.31.b). In primul caz, când latura poligonului este complet în interiorul ferestrei, se adaugă $P$ la lista de ieşire. In al doilea caz, punctul de intersecţie $I$ este introdus ca nou vârf. In cazul al treilea, ambele vârfuri sunt în afara ferestrei, astfel încât nu se produc modificări în lista de ieşire. In cazul al patrulea, atât punctul de intersecţie $I$, cât şi $P$ sunt adăugate la lista de ieşire.

### Algoritmul Liang-Barsky

Algoritmul presupune decuparea faţă de o fereastră dreptunghiulară. Se consideră poligonul care va fi tăiat reprezentat prin secvenţa vârfurilor $P_1$, $P_2, \ldots, P_n$, parcurse într-un anumit sens (trigonometric sau conform fusului orar). Fiecare latură este considerată ca un vector (pornind, de exemplu,

de la un $P_i$ la $P_{i+1}$). Planul este divizat în nouă regiuni determinate de laturile ferestrei. Fiecare latură a ferestrei împarte planul în două semiplane. Semiplanul care conţine fereastra este *semiplan de interior*. Cele nouă regiuni se identifică funcţie de numărul de semiplane interioare care sunt incluse în interiorul lor (figura 3.32.a). Regiunile notate "interior 2" se numesc *regiuni de colţ*, iar cele notate "interior 3", *regiuni de latură*.

Dacă ultima latură ce intersectează fereastra generează un vârf la marginea de sus a ferestrei şi următoarea latură ce intersectează fereastra va crea un vârf la dreapta ferestrei, poligonul de ieşire va trebuie să conţină colţul din dreapta sus a ferestrei (figura 3.32.b). In general, o latură care intră într-o regiune de colţ adaugă un colţ de fereastră ca vârf de ieşire. Un asemenea vârf este numit *vârf turnant*.

Se determină valorile parametrului $t$ ale intersecţiilor liniei ce conţine $P_iP_{i+1}$ cu laturile ferestrei. Două intersecţii sunt potenţial de intrare, $t_{in_1}$, $t_{in_2}$, două de ieşire $t_{out_1}$, $t_{out_2}$. Valoarea $t_{in_1}$ este cea mai mică, iar $t_{out_2}$ este cea mai mare deoarece orice linie neorizontală şi neverticală porneşte dintr-o regiune de colţ şi sfârşeşte într-o altă regiune de colţ. Celelalte valori se află între ele şi pot fi în orice ordine. Dacă $t_{in_2} < t_{out_1}$, linia intersectează fereastra, altfel linia trece printr-o regiune de colţ (figura 3.32.c).

Relaţiile dintre valorile $t = 0$, $t = 1$ şi $t_{in_1}$, $t_{in_2}$, $t_{out_1}$, $t_{out_2}$ caracterizează contribuţiile laturii la poligonul de ieşire. Dacă $0 \leq t_{in_2} < t_{out_1} \leq 1$, segmentul de dreaptă determinat de valorile $t_{in_2}$ şi $t_{out_1}$ este latură a poligonului decupat. In caz contrar, linia nu intersectează fereastra, ci porneşte dintr-o regiune de colţ, trece prin alta şi se termină în a treia. Intrarea segmentului în regiunea de colţ intermediară (ce necesită adăugarea unui colţ al ferestrei în lista vârfurilor poligonului decupat) este caracterizată prin $0 \leq t_{out_1} < t_{in_2} \leq 1$.

Algoritmul este schiţat în următoarea procedură.

```
Procedure Assign(var x,y :real; a,b :real);
begin x := a; y := b; end;
Procedure LiangBarskyPolygonClip(
  n : integer;
  x,y : tablou;                    {vârf.poligonului de intrare}
  var u,v : tablou;                {vârf.poligonului de ieşire}
  xMax,xMin,yMax,yMin :real;       {laturile ferestrei}
  var outContor : integer);        {contor a lat.ieşire}
var
   xIn,xOut,yIn,yOut : real; {Punct.de intrare şi ieşire}
   tOut1,tIn2,tOut2,tInX,tOutX,tInY,tOutY : real;
   deltaX,deltaY : real;       {Direcţia laturii}
   i :integer;
begin
   x[n+1] := x[1]; y[n+1] := y[1]; {închide poligonul}
   fori := 1 to n do
     begin
      deltaX := x[i+1] - x[i]; deltaY := y[i+1] - y[i];
      {determină primele laturi ale ferestrei intersectate}
      if (deltaX > 0) or (deltaX = 0) and (x[i] > xMax))
         then Assign(xIn,xOut,xMin,xMax)
```

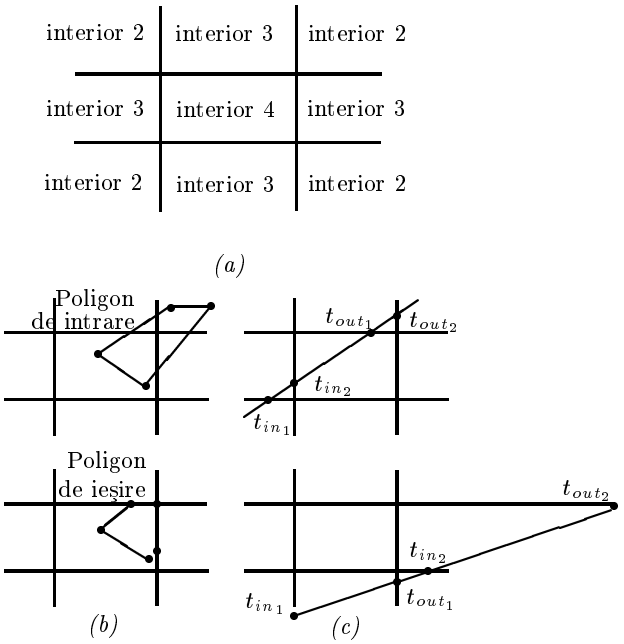



Figura 3.32: Algoritmul `Liang-Barsky` (a) Definirea regiunilor (b) Vârf turnant (c) Intersecţiile cu laturile ferestrei

```
   else Assign(xIn, xOut, xMax, xMin);
if (deltaY > 0) or (deltaY = 0) and (y[i] > yMax))
   then Assign(yIn, yOut, yMin, yMax)
   else Assign(yIn, yOut, yMax, yMin);
{determină valorile t pentru punctele de ieşire}
if deltaX <> 0 then tOutX := (xOut − x[i])/deltaX
   else if (x[i] <= xMax) and (xMin <= x[i])
           then tOutX := ∞ else tOutX := −∞;
if deltaY <> 0 then tOutY := (yOut − y[i])/deltaY
   else if (y[i] <= yMax) and (yMin <= y[i])
           then tOutY := ∞ else tOutY := −∞;
{ordonează punctele de ieşire}
if tOutX < tOutY
   then Assign(tOut1, tOut2, tOutX, tOutY)
   then Assign(tOut1, tOut2, tOutY, tOutX);
if (tOut2 > 0) then {poate exista o ieşire}
   begin
           if deltaX <> 0 then tInX := (xIn − x[i])/deltaX
           else tInX := −∞;
           if deltaY <> 0 then tInY := (yIn − y[i])/deltaY
           else tInY := −∞;
           if tInX < tInY then tIn2 := tInY
```

```
                    else tIn2 := tInX;
                    if (tOut1 < tIn2) then {segmentul invizibil}
                      begin
                       if(0 < tOut1) and (tOut1 <= 1) then
                         {linia trece printr-o regiune de colţ}
                         if tInX < tInY
                         then OutputVert(u, v, outContor, xOut, yIn)
                         else OutputVert(u, v, outContor, xIn, yOut)
                      end
                    else {Linie trece prin fereastră}
                      if (0 < tOut) and (tIn2 <= 1) then
                         begin
                           if 0 < tIn2 then {segment vizibil}
                              if tInX > tInY
                                 then OutputVert(u, v, outContor,
                                   xIn, y[i] + tInX * deltaY)
                                 else OutputVert(u, v, outContor,
                                   x[i] + tInY * deltaX, yIn);
                           if 1 > tOut then
                              if tIOutX < tOutY
                                 then OutputVert(u, v, outContor,
                                   xOut, y[i] + tOutX * deltaY)
                                 else OutputVert(u, v, outContor,
                                   x[i] + tOutY * deltaX, yOut);
                           else
                            OutputVert(u, v, outContor, x[i + 1], y[i + 1]);
                         end;
                    if (0 < tOut2) and (tOut2 <= 1) then
                         OutputVert(u, v, outContor, xOut, yOut);
            end
        end
    end;
```

### 3.10.4 Decuparea curbelor plane

Cercurile şi elipsele pot fi aproximate cu secvenţe de linii scurte, astfel încât pot fi tratate ca linii poligonale cu număr mare de segmente.

Conicele, în general, sunt construite prin metode incrementale care permit includerea testelor de vizibilitate în fereastră.

In cazul cercurilor sau elipselor se poate efectua un test preliminar de acceptare/respingere asupra dreptunghiului minim ce încadrează respectiva conică.

Structura algoritmului incremental de trasare a curbei generale de grad doi $f(x, y) = 0$ cuprinde trei faze:

1. se calculează $\delta$, $\Delta$ şi se separă cazul trasării conicelor degenerate;
2. se calculează toate intersecţiile curbei cu laturile ferestrei de vizualizare. In total pot exista maximum 8 puncte de intersecţie. Pentru fiecare punct de intersecţie se cercetează semnul funcţiei $f$ la dreapta arcului de curbă care porneşte din acest punct şi este orientat spre interiorul ferestrei.

Dacă $f < 0$ la dreapta curbei, atunci punctul de intersecţie este neglijat, deoarece arcul de curbă trebuie trasat începând de la celălalt capăt, care corespunde altui punct de intersecţie (algoritmul incremental porneşte de la presupunerea că $f < 0$ la stânga direcţiei de înaintare). In caz contrar se trasează arcul. Pentru calculul intersecţiei cu laturile ferestrei, de exemplu cu $y = y_{min}$, se recomandă modificarea ecuaţiei $f(x, y_{min}) = 0$. Se observă că $f(x, y_{min}) + f(x, y_{min} - 1) < 0$ (respectiv $> 0$) pentru toate punctele $(x, y_{min})$ mai apropiate (respectiv mai depărtate) de curbă decât $(x, y_{min} - 1)$. In cazul limită, de egalitate cu 0, punctele menţionate sunt egal depărtate de curba ideală. Astfel pentru determinarea intersecţiei cu $y = y_{min}$ se recomandă rezolvarea ecuaţiei (analog factorului de decizie de la trasarea cercurilor)

$$f(x, y) + f(x, y - 1) = 0,$$

cu $y = y_{min}$.

3. se trasează curbele care nu intersectează marginile ferestrei de vizualizare (cercuri sau elipse complet incluse în fereastră): se calculează centrul curbei $(x_c, y_c)$ şi se apelează procedura de trasare incrementală începând cu punctul din dreapta de intersecţie cu orizontala $y = y_c$.

### 3.10.5 Decuparea 3D

Pentru corpurile descrise prin mai multe puncte, dintre care doar o parte se află în volumul de observare, se preferă retezarea segmentelor în spaţiu, păstrând numai porţiunile interioare volumului observabil, şi abia după aceea proiecţia în 2D. Procesul este cunoscut sub denumirea de decupare 3D (3D `clipping`).

Pot fi generalizaţi atât algoritmul `Cohen-Sutherland` cât şi cel al bisecţiei.

In algoritmul `Cohen-Sutherland` 3D, spaţiul este împărţit în 27 de zone, pentru care se alcătuieşte câte un cod de 6 cifre (tabelul 3.4). Primele patru sunt aceleaşi ca şi în plan; a 5-a valoare este 1 dacă punctul se află în faţa volumului de observare şi 0 altfel, a 6-a valoare este 1 dacă punctul se află în spatele volumului de observare (figura 3.33).

Pentru calculul intersecţiilor cu planele limitatoare, în cazul algoritmului `Cohen-Sutherland` 3D, este comod să se transforme mai întâi volumul de observare în volum de observare canonic. In acest caz calculele şi comparaţiile se simplifică deoarece planele limitatoare înclinate au panta 1 sau -1.

Coordonatele intersecţiei se pot calcula folosind forma parametrică a ecuaţiei unei drepte ce trece prin două puncte. Fie segmentul $P_1P_2$ cu extremităţile $P_1(x_1, y_1, z_1)$, $P_2(x_2, y_2, z_2)$. Se calculează intersecţia cu planul $y = z$. Forma parametrică a dreptei este

$$\begin{cases} x = (x_2 - x_1)t + x_1, \\ y = (y_2 - y_1)t + y_1, \\ z = (z_2 - z_1)t + z_1, \end{cases} \qquad 0 \leq t \leq 1.$$

Deoarece $y = z$, are loc $(y_2 - y_1)t + y_1 = (z_2 - z_1)t + z_1$, de unde se obţine $t$ corespunzător punctului de intersecţie:

$$t_{plan(y=z)} = \frac{z_1 - y_1}{(y_2 - y_1) - (z_2 - z_1)}.$$

| Bit setat | Volum canonic al proiecţiei paralele | Volum canonic al al proiecţiei perspective |
|---|---|---|
| bit 1 | $y > 1$ | $y > -z$ |
| bit 2 | $y < -1$ | $y < z$ |
| bit 3 | $x > 1$ | $x > -z$ |
| bit 4 | $x < -1$ | $x < z$ |
| bit 5 | $z > 0$ | $z > z_{min}$ |
| bit 6 | $z < -1$ | $z < -1$ |

Tabelul 3.4: Constituirea codului în algoritmul `Cohen-Sutherland` 3D

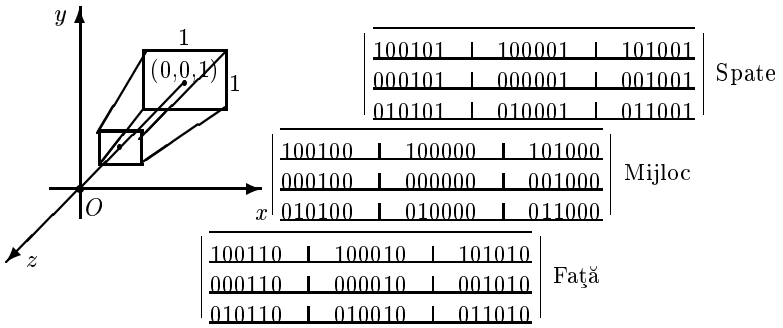



Figura 3.33: Algoritmul `Cohen-Sutherland` pentru decupare 3D

Segmentul $P_1P_2$ intersectează planul $y = z$, dacă $0 \leq t_{plan(y=z)} \leq 1$.

Algoritmul `Cyrus-Beck` poate fi reformulat pentru decupare relativ la un poliedru. Dacă în 2D cel mult 4 valori $t$ trebuiau calculate pentru fiecare latură a ferestrei, în 3D, sunt necesare 6 valori. Valorile noi pentru $N_i$, $P_{E_i}$, $P_0 - P_{E_i}$, în cazul decupării faţă de un trunchi de piramidă, sunt prezentate în tabelul 3.5, unde $dx = x_1 - x_0$, $dy = y_1 - y_0$, $dz = z_1 - z_0$. Semnul lui $-N_i \cdot (P_1 - P_0)$ determină tipul intersecţiei: de ieşire sau de intrare (în tabel, numitorul fracţiilor).

Algoritmul `Cyrus-Beck` 3D este schiţat în următoarea procedură. Procedura adiţională `Clipt` determină dacă intersecţia este de tip intrare sau ieşire.

```
Procedure Clip3D(var x0, y0, z0, x1, y1, z1, zmin : real;
 var accept : boolean);
var tmin, tmax : real; dx, dy, dz : real;
begin
   accept :=false;
   tmin := 0; tmax := 1;
   dx := x1 − x0; dy := y1 − y0; dz := z1 − z0;
   if Clipt(−dx − dz, x0 + z0, tmin, tmax) then {Parte dreaptă}
      if Clipt(dx − dz, −x0 + z0, tmin, tmax) then {Parte stângă}
       begin
         dy := y1 − y0;
         if Clipt(dy − dz, −y0 + z0, tmin, tmax) then {jos}
```

| Latura de decupare | Normala $N_i$ | $P_{E_i}$ | $t = \frac{N_i \cdot (P_0 - P_{E_i})}{-N_i \cdot (P_1 - P_0)}$ |
|---|---|---|---|
| dreapta: $x = -z$ | (1,0,1) | $(x, y, -x)$ | $\frac{x_0 + z_0}{-dx - dz}$ |
| stânga: $x = z$ | (-1,0,1) | $(x, y, x)$ | $\frac{-x_0 + z_0}{dx - dz}$ |
| jos: $y = z$ | (0,-1,1) | $(x, y, y)$ | $\frac{-y_0 + z_0}{dy - dz}$ |
| sus: $y = -z$ | (0,1,1) | $(x, y, -y)$ | $\frac{y_0 + z_0}{-dy - dz}$ |
| faţă: $z = z_{min}$ | (0,0,1) | $(x, y, z_{min})$ | $\frac{z_0 - z_{min}}{-dz}$ |
| spate: $z = -1$ | (0,0,-1) | $(x, y, -1)$ | $\frac{-z_0 - 1}{dz}$ |

Tabelul 3.5: Calculele ce intervin în algoritmul `Cyrus-Beck` 3D

```
            if Clipt(−dy − dz, y0 + z0, tmin, tmax) then {sus}
               if Clipt(−dz, z0 − zmin, tmin, tmax) then {faţă}
                  if Clipt(dz, −z0 − 1, tmin, tmax) then {spate}
                   begin
                     accept :=true; {Parte linie vizibilă}
                     if tmax < 1 then
                      begin
                        x1 := x0 + tmax ∗ dx;
                        y1 := y0 + tmax ∗ dy;
                        z1 := z0 + tmax ∗ dz
                      end;
                     if tmin > 0 then
                      begin
                        x0 := x0 + tmin ∗ dx;
                        y0 := y0 + tmin ∗ dy;
                        z0 := z0 + tmin ∗ dz
                      end
                   end
       end
end; {Clip3D}
```

# 4. Modelarea corpurilor tridimensionale

## 4.1 Scheme de reprezentare ale solidelor rigide

Atunci când dorim reprezentarea pe un ecran a unui solid din spaţiul 3D apar o serie de probleme a căror soluţie trebuie găsită:

(a) cum modelăm corpul în spaţiul 3D;
(b) cum construim o reprezentare a modelului;
(c) cum transpunem reprezentarea 3D în 2D.

Reprezentările simple care permit recunoaşterea unor suprafeţe şi efectuarea de calcule relative la aceste suprafeţe se obţin printr-unul din următoarele procedee de modelare a corpurilor:

- *modelarea suprafeţelor;*
- *modelarea solidului.*

In primul caz, un corp este modelat prin precizarea frontierei sale, descrisă ca suprafaţă strâmbă în spaţiu sau compusă din porţiuni de suprafeţe curbe. In cazul modelării solidului, corpul este construit prin alăturarea unor volume elementare, cuburi, piramide, tetraedre, sfere, cilindri care să aproximeze cât mai bine forma dorită. Ambele procedee se încadrează în clasa mai generală a modelării geometrice (numită şi *modelare a formelor*).

Reprezentarea obiectelor prin colecţii de linii drepte, curbe, poligoane este adesea insuficientă pentru reconstituirea obiectului reprezentat. De exemplu, despre obiectul din figura 4.1.a (reprezentarea prin cadru de sârmă, numai din

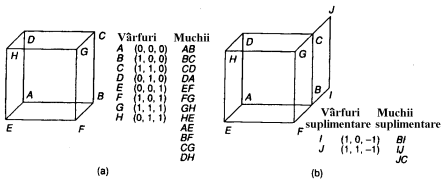



Figura 4.1: (a) Reprezentare prin cadru de sârmă a unui cub; (b) Faţă suplimentară

informaţii despre vârfuri şi muchii) nu se poate spune că este un cub (spunem că reprezentarea este ambiguă) până când nu includem informaţia conform căreia fiecare poligon defineşte o faţă poligonală ce mărgineşte un volum – în aceeaşi modalitate stocăm informaţia referitoare la obiect şi dacă una din feţele

cubului lipseşte (precum o cutie deschisă). In cazul în care mai adăugăm un poligon (figura 4.1.b) obiectul construit nu mărgineşte un volum.

Astfel în reprezentarea solidelor trebuie respectate o serie de proprietăţi precum următoarele:

(a) *domeniul* reprezentării trebuie să fie suficient de larg pentru a permite reprezentarea unor clase largi de obiecte fizice;

(b) reprezentarea trebuie să fie *neambiguă* (să nu existe dubii referitoare la obiectul reprezentat): o reprezentare trebuie să corespundă unui şi numai unui solid; în aceste condiţii testul de egalitate a două obiecte este simplu;

(c) reprezentarea să fie făcută cu *acurateţe* (reprezentarea obiectului fără aproximare);

(d) o schemă de reprezentare trebuie să facă *imposibilă crearea unei reprezentări invalide* (a unei reprezentări care nu corespunde unui solid, precum cea din figura 4.1.b);

(e) schema de reprezentare trebuie să permită *crearea cu uşurinţă a unei reprezentări valide* (în mod tipic, cu ajutorul unui sistem interactiv de modelare a solidelor);

(f) *închiderea la transformările afine* (aplicarea acestor operaţii asupra unor solide valide trebuie să conducă tot la solide valide);

(g) o reprezentare trebuie să fie *compactă* (din punct de vedere al spaţiului de memomare);

(h) o reprezentare trebuie să permită utilizarea unor *algoritmi eficienţi* pentru calculul proprietăţilor fizice şi pentru crearea imaginilor.

Construcţia unei scheme de reprezentare care respectă aceste proprietăţi este dificilă şi adesea sunt necesare o serie de compromisuri.

In literatura de specialitate sunt considerate următoarele scheme de reprezentare a solidelor rigide:

1. *Reprezentarea prin frontiere*. In timp ce toate metodele ce urmează reprezintă solidele direct, reprezentările prin frontiere sunt indirecte prin aceea că ele reprezintă direct (explicit) frontierele topologice ale solidului şi nu solidul însuşi. Această metodă înlocuieşte problema reprezentării unei anumite mulţimi prin reprezentarea unei mulţimi de dimensionalitate redusă.
2. *Instanţierea primitivelor pure*. Se definesc familii de entităţi, numite *primitive generice* sau pure, ai căror membrii se numesc *instanţe* (exemplare) ale primitivelor. O primitivă generică poate fi privită ca un prototip parametrizat al unei piese mecanice. Schema se pretează la tehnologiile de grup când se modelează familii de obiecte, în care membrii au aceeaşi structură, dar diferă prin parametrii, un set dat de parametrii identificând în mod unic un membru al familiei. Schema promovează standardizarea prototipurilor şi este cel mai des utilizată în industrie. Dezavantajul este imposibilitatea de combinare a primitivelor generice pentru a crea o nouă primitivă mai complexă.
3. *Măturare*. Un solid sau o suprafaţă mărginită care se deplasează de-a lungul unei traiectorii "mătură" un volum. In această schemă un solid poate fi reprezentat printr-un cuplu: (corp în mişcare, traiectorie). Metoda foloseşte noţiunile matematice de produs de mulţimi sau operaţii cu mulţimi asupra unui număr infinit de mulţimi.

4. *Enumerarea ocupării spaţiale.* Spaţiul este împărţit într-o reţea fină de celule tridimensionale, de obicei cuburi, numite *voxel* (`volume element`) prin analogie cu pixel (`picture element`). Un solid este reprezentat prin lista celulelor pe care le ocupă, adică a celulelor care conţin material.
5. *Descompunerea în celule.* Un solid este descompus în celule solide elementare, fără găuri, ale căror interioare sunt două câte două disjuncte. Un solid este rezultatul lipirii celulelor componente care satisfac anumite condiţii de potrivire a frontierelor. Schema constituie o generalizare a celei de a doua scheme, eliminând restricţiile referitoare la amplasarea celulelor în poziţii fixe şi la dimensiune şi forme fixe pentru celule. Se bazează pe teoria triangulaţiei.
6. *Geometrie solidă constructivă.* O schemă `CSG` (`Constructive Solid Geometry`) defineşte un solid ca o combinaţie de blocuri constructive solide, combinaţie realizată prin intermediul operaţiilor de tip adunare sau scădere volumetrică. Este o generalizare a celei de a treia scheme, înlocuind operatorul de lipire cu operatori care nu necesită satisfacerea unor condiţii de frontieră.

## 4.2 Metode de construcţie a reprezentărilor

Există trei metode de a construi modele geometrice ale solidelor rigide:

(a) *Descrierea procedurală.* Această metodă constă în reprezentarea unui obiect printr- un program a cărui execuţie produce un exemplar din respectivul obiect. O procedură obiect poate avea parametrii, devenind un obiect generic. Descrierea procedurală este convertită într-un model prin compilarea şi executarea procedurilor obiect în contextul în care aceste proceduri sunt activate. Practic, limbajele de descriere oferă facilităţi de apelare ale unor proceduri standard de descriere: pentru linie, cerc, curbe spline, cilindri. Procedurile se pot apela unele pe altele, permiţând descrierea unor obiecte complexe. Metoda este anevoioasă deoarece utilizatorii trebuie să aibă cunoştiinţe de programare. Se pretează în cazul instanţierii primitivelor pure.

(b) *Conversia bazei de date.* Constă în construirea modelului unui obiect prin vârfuri şi muchii – tip cadru de sârmă (`wire frame`) – pornind de la mai multe proiecţii în 2D ale obiectului. Proiecţiile obiectului în 2D se obţin fie prin metoda descrierii procedurale, fie prin grafică interactivă.

(c) *Grafică interactivă.* Se bazează pe organizarea tuturor comenzilor necesare realizării şi manipulării desenelor în grupuri relaţionate logic, numite *liste de meniu.* Utilizatorul transmite comanda dorită sistemului prin selectarea acesteia din lista de meniu cu ajutorul unui dispozitiv de interacţiune. Această modalitate de a dirija un sistem grafic se numeşte *tehnica meniului.* Listele de meniuri sunt organizate ierarhic într-o structură de arbore. Componenţa listelor de meniuri şi structura ierarhică se obţin în urma unui proces de generare.

## 4.3 Operaţii booleene

O metodă intuitivă şi populară pentru combinarea obiectelor este utilizarea operaţiilor booleene pe mulţimi (figura 4.2).

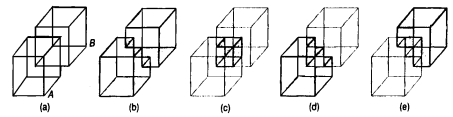



Figura 4.2: Operaţii booleene (a) Obiectele $A$ şi $B$ (b) $A \cup B$ (c) $A \cap B$ (d) $A - B$ (e) $B - A$

Din păcate aplicarea unei operaţii booleene asupra a două obiecte solide nu conduce întotdeauna la un obiect solid (de exemplu, intersecţia a două cuburi poate conduce la un solid, un plan, o linie, un punct sau la mulţimea vidă – figura 4.3). Se pot însă utiliza *operatori booleeni regularizaţi*, notaţi $\cup^*$, $\cap^*$ şi

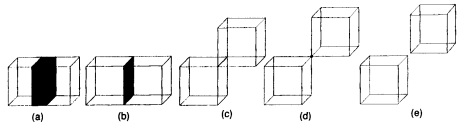



Figura 4.3: Intersecţia booleană a două cuburi poate fi (a) un solid (b) un plan (c) o linie (d) un punct sau (e) mulţimea vidă

$-^*$, definiţi astfel încât aplicarea lor la solide conduce întotdeauna la solide (de exemplu, intersecţia celor două cuburi din figura 4.3 poate fi numai de tip (a) sau (e) sau mulţimea vidă în cazurile (b), (c), (d)). Matematic acest deziderat este realizat prin

$$A \ \mathtt{op}^* \ B = \overline{\mathrm{Int}(A \ \mathtt{op} \ B)},$$

unde op este $\cup$, $\cap$ sau $-$. Un operator boolean regularizat aplicat la două mulţimi închise (în acest context numite regularizate) este tot o mulţime închisă.

Operatorii booleeni regularizaţi sunt utilizaţi ca tehnică de interfaţă cu utilizatorul la construirea unor obiecte complexe din obiecte simple şi sunt incluşi în majoritatea schemelor de reprezentare a solidelor (explicit în cazul geometriei solide constructive).

## 4.4 Reprezentarea corpurilor prin frontiere

Este numită şi `b-rep` de la `boundary representation`. Obiectul este descris prin vârfuri, laturi şi feţe. Feţele curbe sunt aproximate de obicei prin poligoane; se pot însă construi sisteme de modelare bazate pe porţiuni mici de suprafeţe standard.

Variantele cele mai des utilizate sunt: *reprezentarea prin reţea de poligoane* şi *reprezentarea poliedrală*.

Frontiera unui obiect tridimensional poate fi reprezentată şi prin procedee de reducere şi mai drastică a dimensionalităţii obiectului: numai prin puncte sau numai prin linii. Astfel, dacă este necesară doar o reprezentare schematizată a obiectului, se pot utiliza: *reprezentarea prin puncte*, *reprezentarea prin secţiuni transversale*, *reprezentarea tip cadru-de-sârmă* (`wire-frame`).

### 4.4.1 Reprezentarea prin puncte sau secţiuni transversale

Pentru mulţimea punctelor care modelează un corp (în general suprafaţa acestuia) se pot pune două tipuri de condiţii:

i) suprafaţa corpului să treacă efectiv prin punctele date;
ii) distanţa dintre suprafaţa reală şi punctele care definesc modelul să fie mică.

*Reprezentarea prin puncte* se utilizează în domeniul medicinei şi chimiei. In medicină, corpul se reprezintă printr-o reţea de puncte dispuse după secţiuni transversale (figura 4.4). In chimie, atomii sunt modelaţi prin sfere, reprezen-

Figura 4.4: Reprezentare prin puncte a corpului uman

tate prin puncte situate în nodurile unui reţele geodezice.

Elementul de bază al structurilor de date folosite pentru modelare este lista de puncte. O listă completă de puncte conţine coordonatele tuturor punctelor care descriu corpul. O listă de puncte trunchiată conţine numai informaţiile referitoare la o submulţime de puncte, precum şi alte informaţii necesare pentru determinarea coordonatelor celorlalte puncte folosind simetrii, rotaţii, translaţii şi scalări. Formele trunchiate sunt folosite pentru depozitarea informaţiei necesare regenerării periodice a corpului.

Reprezentările prin *secţiuni transversale* decurg direct din reprezentările prin puncte situate în secţiuni transversale paralele între ele. Punctele situate

în aceeaşi secţiune se unesc între ele astfel încât să pară reprezentat conturul secţiunii respective. Uneori, pentru o mai bună precizare a formei obiectului modelat se folosesc două curbe sau linii poligonale denumite *ecuator* şi *meridian*. Adăugând curbe longitudinale unei reprezentări prin secţiuni transversale se obţin reprezentările de tip `wire-frame`.

### 4.4.2 Reprezentarea prin cadru de sârmă

Intr-o asemenea schemă, un corp este reprezentat ca o mulţime de segmente de dreaptă sau porţiuni de curbe (figura 4.5).

Figura 4.5: Reprezentare `wire-frame` a unei mâini

Datele de intrare pot fi organizate astfel:

(a) *segmente explicite:* pentru fiecare segment se precizează un indice (opţional) şi coordonatele extremităţilor segmentului. Nu se utilizează lista de vârfuri.

(b) *segmente implicite:* fiecare segment este precizat printr-o pereche de indici care identifică capetele acestuia într-o listă de vârfuri (necesarul de memorie faţă de forma precedentă scade);

(c) *linii date prin indici:* atunci când o linie poligonală poate fi descrisă prin concatenarea unui şir de segmente, este mult mai indicată utilizarea informaţiilor: indice linie, număr de puncte pe linie, indici care reprezintă capetele segmentelor ce compun linia în lista de vârfuri;

(d) *secţiuni transversale şi linii longitudinale:* se procedează precum în cazul liniilor date prin indici, dar curbele transversale sunt în marea lor majoritate chiar reprezentările secţiunilor transversale.

Reprezentarea `wire-frame` a unui obiect nu permite definirea unor suprafeţe şi, deci, calcularea ariilor, volumelor, maselor, centrelor de greutate sau afişarea pe ecran a porţiunii vizibile a obiectului analizat.

Deşi reprezentarea wire-frame este simplistă şi nu furnizează informaţii complete asupra geometriei corpului, datorită uşurinţei de utilizare şi a rapidităţii de afişare a reprezentării pe terminal, este mult utilizată. Folosind această tehnică se pot obţine viteze mari de lucru şi chiar efecte interesante de animaţie.

### 4.4.3 Reprezentarea prin reţea de poligoane

O *reţea de poligoane* este o colecţie de laturi, vârfuri şi poligoane conectate astfel încât fiecare latură este partajată de cel mult două poligoane (o latură conectează exact două vârfuri, iar un poligon este o secvenţă închisă de laturi). O latură poate fi comună pentru două poligoane, iar un vârf, la cel puţin două laturi.

Dacă obiectul real are suprafeţe curbe, modelul poligonal este aproximativ (figura 4.6). Aproximarea poate fi îmbunătăţită prin mărirea numărului de

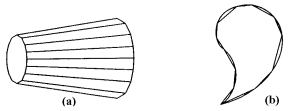



Figura 4.6: (a) Obiect tridimensional reprezentat prin reţea de poligoane (b) O secţiune printr-un obiect curbat şi reprezentarea sa poligonală

faţete poligonale plane care modelează o suprafaţă curbă. Dezavantajul constă în necesarul sporit de memorie, dar este de reţinut faptul că algoritmii care procesează suprafeţele poligonale plane sunt mult mai simpli decât cei pentru suprafeţe curbe.

#### Stocarea informaţiei

Elementul de bază pentru modelarea prin reţea de poligoane este *lista de vârfuri*. Punctele ale căror coordonate sunt înscrise în listă sunt vârfurile poligoanelor. Acestea împreună cu laturile poligoanelor şi cu faţetele poligonale constituie elementele definitorii ale reţelei poligonale.

Stocarea informaţiilor necesare reprezentării vârfurilor, muchiilor şi faţetelor poligonale se face sub diferite forme. Criteriile de alegere a formei de stocare privesc două caracteristici ale programului: memoria necesară şi viteza de lucru.

Reprezentarea reţelelor poligonale se face astfel:

(a) In reprezentarea explicită, fiecare poligon este reprezentat ca listă de coordonate a vârfurilor:

$$P = \{(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_n, y_n, z_n)\}.$$

Vârfurile sunt stocate în ordinea în care sunt întâlnite în parcurgerea poligonului. Laturile se construiesc prin unirea vârfurilor succesive din listă. Pentru un singur poligon, această reprezentare este eficientă din punct de vedere al spaţiului; pentru o reţea poligonală însă o cantitate considerabilă de spaţiu de memorare se pierde prin duplicarea coordonatelor vârfurilor comune. In plus, nu există o reprezentare explicită a

vârfurilor şi laturilor comune. Pentru trasarea unui vârf împreună cu laturile incidente în acesta, trebuie găsite toate poligoanele care conţin vârful – triplete de coordonate trebuie comparate pentru fiecare vârf (eventual toate coordonatele vor fi sortate). La trasarea reţelei, fiecare latură va fi trasată de două ori (în cazul ecranelor tip rastru, dacă laturile sunt trasate în direcţii opuse, e posibil ca anumiţi extra-pixeli să fie aprinşi).

(b) Dacă poligoanele sunt definite ca *pointeri la o listă de vârfuri*

$$V = \{(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_n, y_n, z_n)\},$$

fiecare vârf este stocat o singură dată (figura 4.7). In plus coordonatele

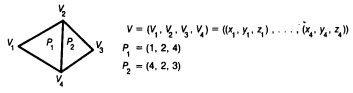



Figura 4.7: Reţea de poligoane definită prin indexi la o listă de vârfuri

vârfurilor pot fi uşor schimbate fără a afecta descrierea poligoanelor. Pe de altă parte, laturile comune sunt, şi în acest caz, trasate de două ori.

(c) Poligoanele sunt definite prin *pointeri la liste de laturi*. Structura de date asociată unei laturi din lista de laturi conţine pointeri (sau indexi) spre două vârfuri din lista de vârfuri, precum şi spre unul sau două poligoane la care aparţine (figura 4.8). Astfel, un poligon este definit prin $P =$

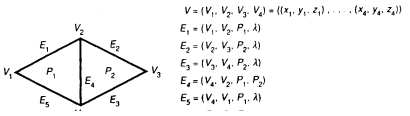



Figura 4.8: Reţea de poligoane defintă prin listă de laturi

$(E_1, \ldots, E_n)$, iar laturile prin $M = (V_1, V_2, P_1, P_2)$ (dacă latura aparţine numai unui poligon, $P_2$ este nul).

In mod curent se utilizează reţele de faţete patrulatere plane sau triunghiulare. Cele mai bune aproximări ale unor suprafeţe strâmbe prin număr dat de vârfuri se obţin folosind faţete triunghiulare.

Când se dispune de o memorie redusă şi se modelează corpuri de complexitate relativ mare printr-un număr mare de puncte este recomandată utilizarea unui liste de vârfuri şi un procedeu de parcurgere a acesteia pentru a determina

rând pe rând indicii corespunzători vârfurilor fiecărei faţete. Se consideră cazul generării unui corp prin faţete triunghiulare. Pentru a genera reţeaua de faţete sunt necesare:

1. o *listă de vârfuri* în care un punct este menţionat printr-un indice şi coordonatele sale;
2. o *listă de distribuţie a punctelor pe secţiuni* în care pentru fiecare secţiune transversală se menţionează numărul şi indicii punctelor care o descriu;
3. o *listă de ramificaţie* în care pentru fiecare secţiune a unui corp, cu excepţia ultimei secţiuni, se precizează pentru fiecare punct, în sensul creşterii indicilor, numărul de triunghiuri distincte care nu au mai fost considerate şi au un vârf în acesta. Pentru primul punct dintr-o secţiune, se numără numai triunghiurile formate cu puncte de pe secţiunea următoare, în sensul creşterii indicilor, pornind de la primul punct al secţiunii următoare.

Pentru cazurile în care este necesar a deosebi la un poligon o faţă exterioară şi una interioară se stochează vârfurile sale într-o ordine convenită în baza de date. Ordinea normală este aceea pentru care vârfurile se stochează astfel încât laturile poligonului privite din exterior să fie aşezate în sens trigonometric. In acest fel, direcţia normalei exterioare la poligonul dat se poate obţine ca produs vectorial a două laturi consecutive neconfundate şi nedegenerate pentru orice poligon convex. Această tehnică permite evitarea trasării unei laturi date de două ori, după regula: o latură nu se trasează decât atunci când este parcursă de la un vârf de indice inferior celui final (figura 4.9).

In nici una dintre aceste reprezentări nu se poate determina cu uşurinţă care laturi sunt incidente unui vârf (toate laturile trebuie cercetate). Informaţii suplimentare pot fi adăugate la structurile de mai sus pentru a rezolva acest inconvenient. De exemplu, la descrierea laturilor se pot include pointeri la cele două laturi adiacente a fiecărui poligon, iar în descrierea vârfurilor se pot include pointeri la fiecare latură incidentă în vârf.

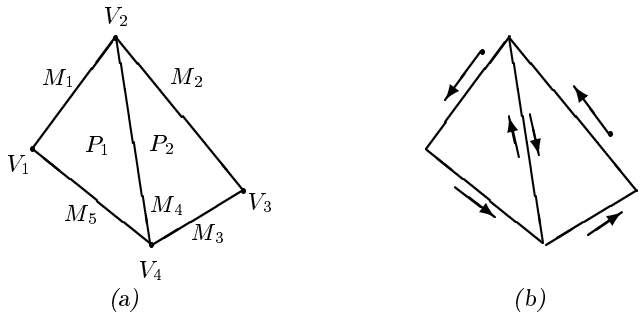



Figura 4.9: Parcurgerea în sens trigonometric a laturilor

### Consistenţa reprezentării prin reţea de poligoane

Dacă reţeaua este generată interactiv se pot produce numeroare erori. O verificare de *consistenţă* presupune:

1. fiecare latură aparţine cel puţin unui poligon şi maximum la un număr fixat de poligoane;
2. fiecare latură este descrisă o singură dată într-un poligon;
3. fiecare vârf aparţine cel puţin unui poligon;
4. fiecare vârf aparţine la cel puţin două laturi;
5. vârfurile unei laturi sunt distincte;
6. poligoanele sunt închise;
7. fiecare poligon care pointează spre o latură este referit prin pointer în structura asociată laturii;
8. laturile unui poligon se află în acelaşi plan;
9. reţeaua are o singură componentă conexă (relaţia de împărţire a unei laturi între poligoane este o relaţie binară de echivalenţă şi se poate efectua o partiţionare a reţelei în clase de echivenţă numite componente conexe), etc.

### 4.4.4 Reprezentarea poliedrală

Un *poliedru* este un solid care este mărginit de un set de poligoane a căror laturi sunt membrii a unui număr par de poligoane şi satisfac anumite condiţii adiţionale. Un poliedru simplu este un poliedru care poate fi deformat într-o sferă (nu are găuri). O condiţie adiţională pentru poliedre simple este formula lui `Euler`: $V - E + F = 2$, unde $V$ este numărul de vârfuri, $E$ este numărul de laturi, iar $F$ este numărul de feţe. Pentru poliedre cu găuri, o generalizare a formulei lui `Euler` este $V_E + F - H = 2(C - G)$, unde $H$ este numărul de găuri în feţe, $G$ este numărul de găuri care trec prin obiect, iar $C$ este numărul de părţi componente, ale obiectului, separate (figura 4.10). Condiţii suplimentare trebuie impuse pentru a garanta că obiectele reprezentate sunt solide.

Pentru a reprezenta direcţia în care poligoanele constituie feţe ale poliedrului, vârfurile poligoanelor sunt stocate în sensul acelor de ceasornic, aşa cum sunt văzute din exteriorul solidului. Pentru evitarea duplicării în reprezentare a coordonatelor vârfurilor comune feţelor, fiecare vârf a unei feţe va fi reprezentat printr-un index (sau pointer) în lista de coordonate. Muchiile sunt reprezentate implicit prin perechi de vârfuri adiacente în lista de vârfuri a poligoanelor.

### 4.4.5 Reprezentarea prin laturi mobile (laturi înaripate)

Reprezentările simple fac ca anumite calcule să fie extrem de costisitoare. De exemplu, descoperirea a două feţe care împart o muchie (test cerut de exemplu în validarea solidului) necesită o căutare în lista de laturi. Reprezentarea prin laturi mobile este o soluţie la această problemă (figura 4.11). Fiecare muchie într-o asemenea structură este reprezentată prin pointeri la cele două vârfuri, la cele două feţe care împart latura, şi la patru muchii adiţionale care pornesc din vârfurile ei. Fiecare vârf are un pointer spre una din muchiile care pornesc din el, iar fiecare faţă spre una din laturile sale. Vârfurile unei muchii sunt parcurse în ordine inversă când se urmăresc vârfurile fiecărei din cele două feţe cărora le aparţine muchia. Dacă vârfurile unei muchii sunt $N$ şi $P$, faţa din dreapta este referită ca faţă $P$, dacă se traversează muchia de la $N$ la $P$ şi ca faţă $N$ când se traversează muchia de la $P$ la $N$. De exemplu, in figura 4.11,

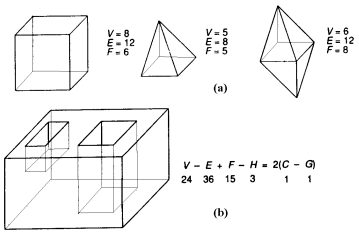



Figura 4.10: (a) Poliedre simple şi valorile lor $V$, $E$ şi $F$ (b) Poliedru cu două găuri în faţa de sus şi o gaură în faţa de jos

dacă $N = V_1$, $P = V_2$, atunci faţa $F_1$ este faţa $V_2$ a muchiei $E_1$, iar $F_2$ este faţa $V_1$ a muchiei $E_1$. Cele patru laturi spre care fiecare punct al muchiei pointează sunt:

$N$ : două laturi care împart vârful $N$ al muchiei pe faţa $N$ şi muchiile anterioare (în sensul acelor de ceasornic) din faţa $P$, $E_3$ şi $E_2$ în exemplu;

$P$ : două laturi care împart vârful $P$ al muchiei pe faţa $P$ şi muchiile anterioare (în sensul acelor de ceasornic) din faţa $N$, $E_4$ şi $E_5$ în exemplu.

Aceste patru muchii sunt „aripile" de unde structura de acest tip a preluat numele.

Acestă reprezentare este posibilă numai pentru feţe fără găuri. Structurarea permite determinarea în timp constant a vârfurilor sau muchiilor asociate cu o muchie.

## 4.5 Instanţerea primitivelor

In acest caz sistemul de modelare defineşte un set de primitive de solide tridimensionale care sunt relevante în domeniul aplicaţiei. Aceste primitive sunt parametrizate (de exemplu, una dintre primitive poate fi o piramidă regulată cu un număr de feţe dependent de utilizator). O astfel de primitivă (figura 4.12) poate fi privită ca o familie cu membrii care variază într-un număr mic de parametrii (*tehnologie de grup* este termenul utilizat în sistemele `CAD` pentru această tehnică). Instanţierea primitivelor este utilizată pentru crearea unor obiecte complexe, reprezentate prin ierarhii. Singura modalitate de a crea un nou tip de obiect este aceea de a scrie un cod care îl defineşte (incluzând şi procedurile de trasare a obiectului, sau de determinare a proprietăţilor fizice).

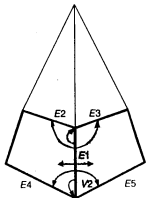



Figura 4.11: Structură de date pentru $E_1$: $V_1$, $V_2$, $F_1$ şi $F_2$ au pointeri la una din laturile lor

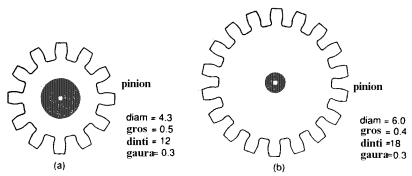



Figura 4.12: Două pinioane definite prin instanţierea primitivelor

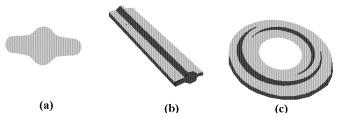



Figura 4.13: (a) Obiect 2D care se mişcă (b) Obiect 3D obţinut prin translatare (c) Obiect 3D obţinut prin rotire

## 4.6 Reprezentarea prin mişcare (măturare)

Mişcând un obiect de-a lungul unei traiectorii, urma lăsată de acesta defineşte un nou obiect. Cel mai simplu obiect construit astfel este definit de o suprafaţă bidimensională care se mişcă de-a lungul unui drum liniar normal la planul ariei, creându-se un *volum translaţional*. *Volumele rotaţionale* sunt definite prin rotirea unei arii în jurul unei axe (figura 4.13). Cazul general corespunde volumelor generate prin arii sau volume care se schimbă în formă, aspect şi orientare, urmând o traiectorie curbă oarecare (modelarea acestora este extrem de dificilă şi nu întotdeauna conduce la un solid – de exemplu mişcând o arie bidimensională în propriul plan se generează tot o arie bidimensională –; în plus obiectul în sine se poate auto-intersecta astfel încât calculul de volum este complicat). In aplicaţiile grafice se întâlneşte noţiunea de *cilindru generalizat* ca referinţă la un volum de măturare a unei secţiuni bidimensionale.

Aplicarea operaţiilor booleene regularizate este dificilă fără convertirea obiectelor într-o altă reprezentare. In plus volumele simple de măturare nu respectă proprietatea de închidere faţă de operaţiile booleene regularizate. De aceea numeroare sisteme de modelare a solidelor permit utilizatorilor construirea unor obiecte ca volume de măturare, dar stocarea informaţiilor despre aceste obiecte se realizează în alte reprezentări.

## 4.7 Reprezentarea prin partiţionare spaţială

In această reprezentare un solid este descompus într-o colecţie de solide (reprezentate prin primitive) adjuncte, care nu se intersectează şi sunt mult mai simple decât solidul reprezentat. Primitivele pot varia în tip, mărime, poziţie, parametrizare şi orientare.

### 4.7.1 Descompunerea celulară

Fiecare sistem de reprezentare prin celule defineşte un set de celule-primitive care sunt parametrizate. Descompunerea celulară diferă de instanţerea primitivelor prin aceea că putem compune obiecte complexe într-o manieră asemănătoare lipirii. Deşi reprezentarea obiectului este neambiguă, ea poate să nu fie unică (figura 4.14). Validarea descompunerii presupune testarea la intersecţie a perechilor componente, test dificil de realizat.

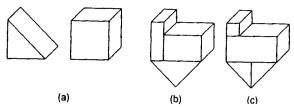



Figura 4.14: Celule prezentate la (a) pot fi transformate pentru a construi acelaşi obiect în diferite moduri, cel puţin două fiind prezentate la (b) şi (c)

### 4.7.2 Enumerarea ocupării spaţiale

Este un caz special de descompunere celulară în celule identice aranjate într-o grilă fixă şi regulată. Aceste celule sunt adesea numite `voxels` (`volume elements`) în analogie cu termenul de `pixel`. Tipul cel mai comun de celulă este cubul. Reprezentarea unui obiectului obiect presupune controlarea prezenţei sau absenţei unui celule la fiecare poziţie a grilei (se decide numai care celule sunt ocupate şi care nu). Astfel obiectul poate fi descris printr-o listă unică şi neambiguă de celule ocupate. Testul de apartenenţă a unei celule la un obiect se realizează foarte simplu. Acest tip de reprezentare este des utilizat în aplicaţiile biomedicale, în analiza datelor volumetrice obţinute din surse precum scanarea tomografică axială computerizată (`CAT`). Unul dintre dezavantajele acestei reprezentări este faptul că numai obiectele a căror feţe sunt paralele cu feţele cuburilor elementare şi a căror vârfuri se potrivesc exact pe grilă vor fi reprezentate cu exactitate. In celelalte cazuri, obiectele vor fi aproximate (figura 4.15). Acurateţea reprezentării poate fi îmbunătăţită prin reducerea dimensiunii celulelor, însă apar probleme relative la stocarea informaţiei relative la obiecte într-un spaţiu de memorie rezonabil.

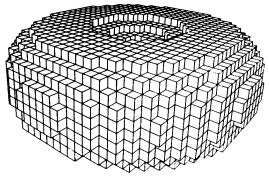


Figura 4.15: Tor reprezentat prin enumerarea ocupării spaţiale

### 4.7.3 Arbori octali

Prin utilizarea arborilor ca variantă ierarhică de enumerare a ocupării spaţiale se pot evita problemele de stocare a informaţiei. In cazul unei figuri bidimensionale, aria de reprezentare este divizată în patru părţi egale care pot aparţine obiectului, să conţină parţial obiectul sau să nu conţină părţi ale obiectului. In cazul al treilea este necesară subdivizarea în alte patru părţi egale despre care se poate afirma acelaşi lucru ca şi în cazul anterior (figura 4.16). Informaţia referitoare la obiect este stocată sub forma unui arbore (figura 4.17). Generalizarea la cazul 3D presupune înlocuirea pătratelor cu cuburi; în fiecare nod al arborelui reprezentării vor exista nu 4 ci 8 legături la noduri fii (figura 4.18). Operaţiile booleene pot fi realizate cu uşurinţă, iar transformările simple precum rotaţia cu 90° sau scalarea cu un factor 2 pot fi efectuate rapid. Una

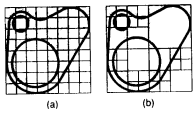



Figura 4.16: Un obiect definit prin (a) enumerarea ocupării spaţiale (b) prin arbori

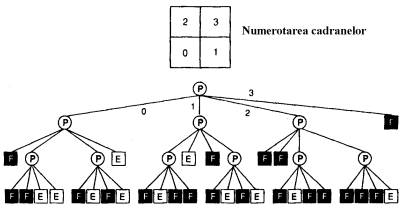



Figura 4.17: Structura de date pentru obiectul din figure 4.16; F=plin, P=parţial plin, E=gol

dintre operaţiile complicate în această reprezentare este găsirea vecinului unui nod din reprezentare (cu care împarte o faţă, muchie sau vârf).

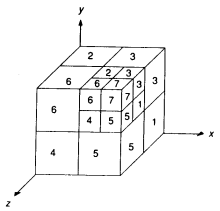



Figura 4.18: Enumerarea cuburilor

### 4.7.4 Arbori binari pentru partiţionare spaţială (arbori BSP)

Spaţiul este divizat în perechi de subspaţii, fiecare separat printr-un plan de orientare şi poziţie arbitrare. Fiecare nod intern a unui arbore BSP este asociat cu un plan şi are doi pointeri spre doi fii, unul de fiecare parte a planului. Presupunând că se cunosc normalele la suprafeţe, ce pointează spre exteriorul obiectului, fiul stâng va fi în interiorul obiectului, iar fiul drept în exterior. Dacă semispaţiul de o parte a planului este divizat în continuare, atunci fiul va fi rădăcina unui subarbore (figura 4.19). Dacă semispaţiul este omogen, atunci fiul este reprezentat printr-un nod terminal (regiune complet în interiorul poliedrului – celulă ,,in" –, sau complet în afara poliedrului – celulă ,,out"). Teste, precum clasificarea punctuală (determinarea poziţiei unui punct faţă de un solid), pot fi cu uşurinţă realizate utilizând această reprezentare.

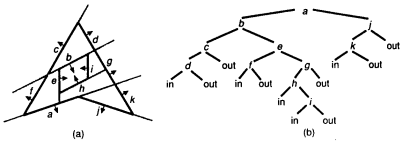



Figura 4.19: Reprezentare BSP în 2D (a) Poligon concav (b) Arborele BSP

## 4.8 Geometria constructivă a solidelor (CSG)

In CSG, primitivele simple sunt combinate utilizând operatorii booleeni regularizaţi care sunt direct incluşi în reprezentare. Un obiect este stocat ca un arbore cu operatori ca noduri interne şi primitive ca noduri terminale (figura 4.20). In anumite implementări primitivele sunt solide simple precum cuburi sau sfere, asigurându-se astfel cerinţa ca obiectele obţinute prin combinare regularizată să fie solide valide. Descompunerea celulară şi enumerarea ocupării spaţiale sunt cazuri speciale de CSG în care unicul operator este operatorul de lipire – reuniunea unor obiecte ce trebuie să aibă intersecţia booleană regularizată nulă.

CSG nu oferă o reprezentare unică. Aplicând aceeaşi operaţie la două obiecte care iniţial erau la fel se poate ajunge la rezultate diferite, precum este arătat în figura 4.21 (a), (b), (c). La (d) şi (e) faţa obiectului (b) respectiv (c) este ridicată rezultând două obiecte diferite. Cu toate acestea, datorită abilităţii în editarea modelelor, la ştergere, adăugare, înlocuire, cuplată cu forma relativ compactă în care modelele sunt stocate, CSG este reprezentarea dominantă în modelarea solidelor.

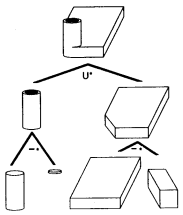


Figura 4.20: Un obiect definit prin CSG şi arborele său

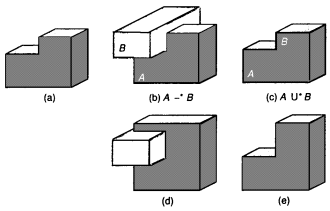



Figura 4.21: Obiectul din figura (a) poate fi definit prin operaţii CSG diferite, precum se arată în (b) şi (c); ridicând faţa superioară a lui (b) şi (c) se obţin obiecte diferite după cum se observă în (d) şi (c)

## 4.9 Compararea reprezentărilor

Au fost discutate cinci tipuri de reprezentare: instanţierea primitivelor, modelarea prin mişcare, reprezentarea prin frontiere, partiţionarea spaţială (incluzând şi descompunerea în celule), şi geometria constructivă. Pe baza criteriilor expuse în prima secţiune a capitolului curent, aceste reprezentări pot fi comparate.

In majoritatea sistemelor de modelare a solidelor se utilizează reprezentări multiple deoarece anumite operaţii sunt mai eficiente într-o reprezentare decât în alta. Convertirea dintr-o reprezentare în alta se face cu aproximaţie, astfel încât procesul este neinvertibil (se pierde din informaţie).

**Acurateţe.** Reprezentarea spaţială şi `b-rep` prin poligoane produc doar aproximări în cazul a numeroare obiecte. In anumite aplicaţii, precum găsirea unui drum pentru un robot, aceasta nu este o problemă dacă aproximarea este realizată la o rezoluţie adecvată. Pe de altă parte, pentru a crea imagini realiste sau pentru calcularea intersecţiei obiectelor rezoluţia trebuie să fie destul de înaltă, fapt irealizabil în practică. De aceea sistemele ce produc imagini de calitate înaltă utilizează adesea `CSG` cu primitive nepoliedrale şi `b-rep` cu suprafeţe curbe. Instanţierea primitivelor poate produce imagini de calitate însă nu permite ca două obiecte simple să fie combinate prin operatori booleeni.

**Domeniu.** Domeniul obiectelor care pot fi reprezentate prin instanţierea primitivelor sau mişcare este limitat. In tehnica enumerării ocupării spaţiale poate fi reprezentat orice solid, deşi adesea este necesară o aproximare. Permiţând unei reprezentări `b-rep` folosirea şi altor tipuri de feţe şi muchii decât poligoane mărginite de linii drepte, aceasta poate fi utilizată cu succes pentru reprezentarea unei clase foarte variate de obiecte.

**Unicitate.** Numai reprezentarea prin arbori octali şi enumerarea ocupării spaţiale garantează unicitatea reprezentării. Instanţierea primitivelor nu garantează, în general, unicitatea: de exemplu, o sferă poate fi reprezentată printr-o primitivă sferă sau ca un elipsoid (dacă setul de primitive este ales cu atenţie, unicitatea poate fi asigurată).

**Validitate.** Dintre toate reprezentările, `b-rep` este cea mai dificil a fi validată (nu numai vârfurile, muchiile şi structura de date a unei feţe pot fi inconsistente, dar se poate ca feţele şi muchiile să se intersecteze). Pe de altă parte un arbore `BSP` defineşte în mod implicit o reprezentare validă, dar nu în mod necesar un solid mărginit. In ceea ce priveşte un arbore `CSG` (întotdeauna mărginit dacă primitivele sale sunt mărginite) sau un arbore octal verificarea de validitate este simplă; în cazul enumerării ocupării spaţiale un asemenea test nu este necesar.

**Inchidere.** Prin instanţierea primitivelor obiectele nu pot fi combinate, iar reprezentarea prin mişcare nu satisface condiţia de închidere la operaţiile booleene. In `b-rep` apar adesea probleme cu închiderea la operaţiile booleene, însă aceste cazuri pot fi depistate şi evitate.

**Compacitate.** Din acest punct de vedere `CSG` este recomandat datorită abilităţii de înregistrare rapidă a operaţiilor booleene şi a transformărilor.

# 5. Realism vizual

## 5.1 Prelucrarea reprezentărilor simple ale imaginii

*Prelucrarea reprezentărilor simple* este o etapă de procesare grafică de complexitate superioră transformărilor simple ale imaginii. *Scopul este mărirea cantităţii de informaţie disponibilă pe ecran pentru a permite utilizatorului să înţeleagă relaţiile 3D existente între mai multe corpuri sau între componentele unui corp.* Crearea unor imagini realiste este un scop important în domenii precum simularea, design-ul, jocuri şi divertisment, cercetare şi educaţie, control şi comandă. Dificultatea fundamentală în atingerea realismului vizual este complexitatea lumii reale.

Termenul de *imagine realistă* este utilizat pentru a referi o imagine care capturează o cantitate apreciabilă de efecte de interacţiune a luminii cu obiectele fizice reale. Crearea unor imagini realiste implică un număr de etape. In primul rând sunt generate modele ale obiectelor. Apoi se specifică condiţiile de vizualizare şi condiţiile de luminare. Sunt determinate suprafeţele vizibile de către observator. Culoarea fiecărui punct în proiecţia unei suprafeţe vizibile este o funcţie de lumina reflectată şi transmisă de obiecte. Procesul de creare a imaginii din modele se numeşte *interpretare* (`rendering`).

Pentru ca utilizatorul să înţeleagă relaţiile spaţiale tridimensionale între mai multe obiecte, o simplă reprezentare poate fi adesea suficientă (în figura 5.1 informaţia este suficientă pentru ca utilizatorul să declare obiectele ca fiind

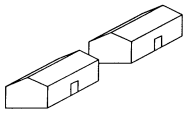


Figura 5.1: Reprezentare prin poligoane a două case

două case, una în spatele celeilalte); în anumite contexte, adăugarea unor detalii pot distrage atenţia privitorului de la informaţii mult mai importante care se furnizează.

Prin proiectare se pierd informaţii importante creându-se adesea ambiguităţi în imagine. Un exemplu concludent este reprezentarea prin cadru de sârmă a unui cub. Astfel, nu putem spune dacă cubul (a) din figura 5.2 reprezintă cubul solid (b) sau (c). In figura 5.3 se prezintă efectul numit iluzia scărilor: în imagine sunt reprezentate scări care sunt văzute de sus, de jos, sau este descris un alt obiect decât o scară? Orice informaţia suplimentară precum un obiect aşezat pe scară poate rezolva ambiguitatea.

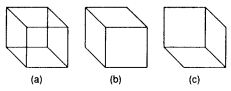



Figura 5.2: Iluzia cubului

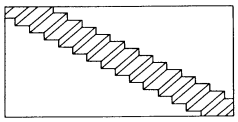


Figura 5.3: Iluzia scărilor

*Proiecţiile* cele mai uşor de realizat sunt proiecţiile ortografice. Obiectul reprezentat în figura 5.4 este uşor de înţeles, însă dacă obiectul reprezentat

Figura 5.4: Proiecţii ortografice ale literei L tridimensionale

este compus din numeroase părţi înţelegerea lui poate necesita ore de studiu. In proiecţiile axonometrice şi oblice, coordonata $z$ influenţează coordonatele $x$ şi $y$ din proiecţie, însă obiectele de aceeaşi mărime aflate la distanţe diferite de planul de proiecţie apar în imagine la aceeaşi mărime. Proiecţia perspectivă poate produce adesea imagini ambigue: în figura 5.5.a poate fi o proiecţie paralelă a unui trunchi de piramidă sau o proiecţie perspectivă a unui paralelipiped dreptunghic. Interpretarea propiecţiei perspective este adesea bazată pe presupunerea că obiectele mici sunt obiecte îndepărtate. Astfel, urmărind figura 5.5.b privitorul poate presupune că cea mai mare casă este cea mai apropiată de observator; în realitate, acea casă poate fi mai departe de observator decât casa mică în proiecţie, iar dimensiunile reale ale ei pot fi mari comparativ cu dimensiunile casei mici (informaţiile de detaliu precum trasarea ferestrelor pot rezolva această ambiguitate; dacă utilizatorul este informat că în imagine sunt prezentate linii paralele, punctul de fugă poate oferi informaţia lipsă în stabilirea poziţiei relative a celor două case).

In efectul numit *iluzia adâncimii* părţi ale obiectului care sunt mai îndepărtate de observator sunt afişate cu o intensitate mai scăzută decât cele mai apropiate. Această tehnică este eficientă în prezentarea unor scene cu


Figura 5.5: (a) Proiecţia perspectivă a unui cub (b) Proiecţia perspectivă a două case

distanţe mari în adâncime între obiecte.

In *decuparea în adâncime* se utilizează un plan de secţionare ce se deplasează (dinamic) odată cu creşterea adâncimii – la un moment dat trebuie determinate punctele obiectelor care intersectează planul de secţionare (aceste puncte vor constitui noua informaţie care trebuie adăugată la imaginea precedentă).

*Texturile* pot conduce adesea la rezolvarea ambiguităţilor produse prin proiecţie. In mod ideal texturile trebuie să respecte forma şi orientarea suprefeţelor pe care sunt aplicate. Pe ecranele monocrome, metoda texturii poate fi folosită pentru simularea umbrelor prin densitate de puncte.

*Culoarea* poate fi de asemenea utilizată cu succes în relaţia dintre obiecte.

Deoarece *înlăturarea liniilor şi suprafeţelor invizibile* din punctul observatorului distruge structura internă a obiectelor opace, aceasta nu trebuie să fie unica soluţie într-o imagine pentru stabilirea relaţiilor de adâncime dintre obiectele scenei.

In majoritatea tehnicilor utilizate în mediile grafice aceeaşi imagine este prezentată ambilor ochi ai utilizatorului. Creierul uman însă combină cele două imagini diferite preluate prin ochi (pereche stereo) într-una singură şi o interpretează ca fiind tridimensională. Există o serie de tehnici pentru a oferi diferite imagini pentru fiecare ochi, incluzând holografia şi ochelarii cu filtre polarizate.

In concluzie, reprezentările realiste ale corpurilor 3D se pot obţine astfel prin combinarea cel puţin a următoarelor patru metode:

1. eliminarea tuturor elementelor şi părţilor corpului care sunt *mascate* în anumite condiţii de privire;
2. simularea *iluminării* corpului cu un sistem de surse de lumină bine precizat (punctiforme, surse distribuite, lumină paralelă, lumină ambiantă);
3. reconstituirea *detaliilor* existente pe suprafaţă (a texturii materialului).
4. reconstituirea *culorilor* şi nuanţelor în care obiectul (corpul) original ar apărea, luminat în condiţiile precizate.

Metodele de asistare a procesului cognitiv prin mijloace grafice utilizate în sistemele `CAD-CAM` propun următoarele tehnici:

(a) utilizarea imaginii *perspective*, care încorporează informaţii de profunzime;

(b) efectul cinetic de profunzime (*proiecţii dinamice*): această tehnică se bazează pe mişcarea obiectului relativ la poziţia observatorului. O mişcare edificatoare este rotaţia în jurul unei axe verticale: liniile apropi-

ate se mişcă mai rapid decât cele îndepărtate, liniile din părţi opuse ale axei de rotaţie se mişcă în sensuri contrare.

(c) indicii de *intensitate*: această metodă implică variaţia intensităţii liniilor. Liniile îndepărtate vor apărea mai şterse decât cele mai apropiate de utilizatorul sistemului;

(d) *secţionarea în adâncime*: informaţii referitoare la a 3-a dimensiune pot fi oferite prin deplasarea unui plan perpendicular pe axa $Oz$ de-a lungul acesteia. Planul numit limitator posterior dezvăluie observatorului, pe măsura deplasării sale, obiectul, oferind astfel informaţii relative la profunzime;

(e) *secţionarea frontală*: planul de secţionare este frontal, iar partea eliminată de acesta este cea mai apropiată de observator. Metoda este mai utilă decât cea anterioră, deoarece poate furniza, printr-o poziţionare adecvată a obiectului, orice secţiune plană prin corpul respectiv;

(f) *umbrirea*: cunoscând direcţia luminii (pentru iluminarea paralelă) sau poziţia sursei (pentru iluminarea cu o sursă punctiformă), umbrele pot furniza informaţii utile pentru sugerarea poziţiei în spaţiu.

(g) *variaţia de culoare*: de-a lungul axei $Oz$ se poate introduce o variaţie spectrală de culoare, de exemplu, de la roşu (punctele apropiate) la violet (punctele îndepărtate), care să furnizeze informaţii cu privire la relaţiile de profunzime între elementele corpului.

## 5.2 Determinarea liniilor şi suprafeţelor vizibile

Date fiind un set de obiecte 3D şi o specificare a vizualizării, se cere determinarea liniilor şi suprafeţelor obiectelor care sunt vizibile, fie din centrul de proiecţie pentru proiecţiile în perspectivă, fie de-a lungul direcţiei de proiecţie în cazul proiecţiilor paralele (figura 5.6).

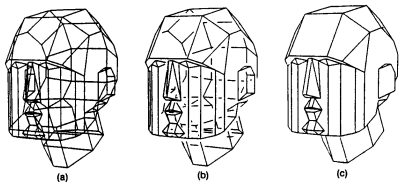



Figura 5.6: Reprezentarea unui cap (a) fără ascunderea liniilor invizibile (b) cu linii ascunse trasate ca linii întrerupte (c) cu ascunderea liniilor invizibile

Se pot imagina următoarele două tipuri de coduri pentru rezolvarea problemei:

```
for fiecare obiect din mediu do
    begin
        determină acele părţi ale obiectului care nu sunt
          obstrucţionate de alte părţi ale oricărui alt obiect
        trasează aceste părţi în culoarea corespunzătoare
    end
```

sau

```
for fiecare pixel din imagine do
    begin
        determină obiectul cel mai apropiat de observator
            care are o parte proiectată pe pixel
        trasează pixelul în culoarea corespunzătoare
    end
```

Există, corespunzător acestor modalităţi de abordare a problemei, două tipuri de algoritmi:

1. *algoritmi spaţiu-obiect* (precizie obiect): depind de precizia cu care obiectele sunt definite şi determină vizibilitatea fiecărui obiect;
2. *algoritmi spaţiu-imagine* (precizie imagine): depind de rezoluţia mediului de afişare şi determină vizibilitatea pe pixel.

Pentru $n$ obiecte şi $p$ pixeli efortul de calcul este proporţional, în primul caz, cu $n^2$ şi, în al doilea caz, cu $np$. Deşi în majoritatea situaţiilor $n^2 \ll np$, algoritmii spaţiu-imagine sunt mai rapizi deoarece subrutinele de comparare individuală sunt mult mai simple decât în cazul algoritmilor spaţiu- obiect şi deci consumă mai puţin timp.

Algoritmii spaţiu-obiect oferă o acurateţe deosebită a imaginii. Algoritmii spaţiu-imagine oferă rapiditate în construcţia imaginii. Calculul în spaţiu-obiect se realizează cu precizie arbitrară. Imaginea finală va fi corectă chiar şi mărită de mai multe ori. Soluţiile în spaţiu-imagine sunt calculate cu o rezoluţie mai mică, de obicei cea a dispozitivului de afişare.

### 5.2.1 Simplificarea calculelor

#### Coerenţă

Algoritmii de vizibilitate ţin seama ades de avantajele *coerenţei – gradul în care fiecare parte a unui mediu sau proiecţia acestuia prezintă similarităţi locale*. Există numeroase tipuri de coherenţă. Printre acestea se remarcă:

1. *coerenţa obiectelor*: dacă un obiect este complet separat de altul, comparaţiile trebuie efectuate numai între cele două obiecte nu şi între feţele sau muchiile componente;
2. *coerenţa feţelor*: proprietăţile suprafeţelor variază de obicei puţin de-a lungul unei feţe, permiţând astfel ca rezultatele calculelor corespunzătoare unei părţi ale feţei să fie modificate incremental pentru a fi utilizate la părţile adiacente;
3. *coerenţa muchiilor*: o muchie îşi schimbă vizibilitatea numai dacă trece prin spatele unei laturi vizibile sau străpunge o faţă vizibilă;

4. *coerenţa liniilor scanate*: setul de obiecte vizibile care se intersectează cu linia de scanare diferă puţin de setul corespunzător liniei precedente;
5. *coerenţa ariilor*: un grup de pixeli adiacenţi este adesea acoperit de aceeaşi faţă vizibilă;
6. *coerenţa adâncimii*: părţi adiacente ale unei aceleiaşi suprafeţe sunt apropiate în adâncine, pe când suprafeţe diferite la aceeaşi locaţie de ecran sunt de obicei mult separate în adâncime; astfel dacă este calculată adâncimea unui punct al suprafeţei, adâncimile punctelor din restul suprafeţei pot fi determinate prin ecuaţii simple cu diferenţe;
7. *coerenţa cadrelor*: imaginile aceluiaşi mediu la două puncte succesive în timp sunt aproape similare, excluzând mici schimbări în obiecte şi puncte de vedere; astfel, calculele realizate pentru o imagine pot fi reutilizate pentru următoarea imagine din secvenţa de cadre.

### Poziţia observatorului

Se consideră că planul de proiecţie este $xOy$ al sistemului de vizualizare, iar
(a) în cazul proiecţiei perspective, observatorul se află pe axa $z$ la o cotă pozitivă,
(b) în cazul proiecţiei paralele, direcţia de proiecţie este paralelă cu axa $z$ şi în sens invers acesteia.

Dacă condiţiile de vizualizare nu respectă aceste reguli, se impune o fază de preprocesare în care obiectul este rotit şi translatat pentru a se încadra în condiţiile standard.

### Teste de adâncime

Se pune problema dacă două puncte se acoperă unul pe celălalt în proiecţiile pe planul $xOy$.

In cazul proiecţiei paralele, două puncte $A(x_1, y_1, z_1)$, $B(x_2, y_2, z_2)$ sunt pe aceeaşi linie de proiecţie dacă şi numai dacă $x_1 = x_2$, $y_1 = y_2$.

In cazul proiecţiei perspective, pentru simplificarea calculelor, comparaţiile individuale privind relaţiile de profunzime se fac după aplicarea transformării care duce centrul de proiecţie în origine. Relaţiile de verificat pentru ca $A$ şi $B$ să fie pe aceeaşi linie de proiecţie sunt: $x_1/z_1 = x_2/z_2$, $y_1/z_1 = y_2/z_2$ (figura 5.7). Pentru fiecare comparare a două puncte trebuie efectuate patru împărţiri. O soluţie pentru evitarea acestor împărţiri este aplicarea unei transformări 3D obiectului real astfel încât obiectul transformat obţinut să aibă o proiecţie paralelă identică cu proiecţia perspectivă a obiectului real (figura 5.8). Această transformare mută centrul de proiecţie la $-\infty$ pe axa $z$ şi transpune liniile de proiecţie în linii paralele cu axa $z$, deci paralele între ele (vezi capitolul 2).

### Teste de interioritate

Se pune problema interiorităţii unui punct faţă de un contur poligonal. Fie $F = (P_1, P_2, \ldots, P_n)$ un poligon în planul de proiecţie cu vârfurile $P_i(x_i, y_i)$, $i = 1, \ldots, n$ şi $P_n = P_1$. Fie $P(x, y)$ punctul pentru care trebuie să se efectueze testul de interioritate. In geometria elementară există două proceduri bine

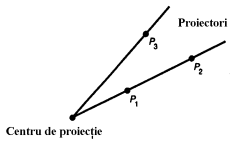



Figura 5.7: Punctele $P_1$ şi $P_2$ sunt pe acelaşi proiector

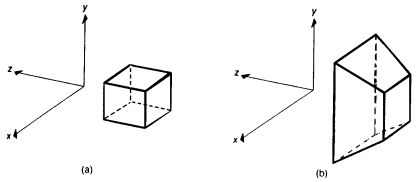



Figura 5.8: Un cub (a) înainte şi (b) după transformarea perspectivă

cunoscute pentru realizarea testului de interioritate a unui punct faţă de un contur poligonal în planul de proiecţie:

1. *calcularea sumei unghiurilor.* Se notează cu $\alpha_i$ măsura unghiului făcut de segmentele $PP_i$ şi $PP_{i+1}$, $(i = 1, \ldots, n-1)$. Atunci, $P$ este în exteriorul conturului poligonal dacă $\sum_{i=1}^{n-1} \alpha_i = 0$ şi este în interiorul conturului dacă $\sum_{i=1}^{n-1} \alpha_i = 2\pi$ (figura 5.9.a);
2. *calcularea numărului de intersecţii (testul par-impar).* Se consideră o semidreaptă, care porneşte din punctul $P$ şi nu trece prin niciunul din vârfurile poligonului. Punctul este exterior poligonului dacă numărul de intersecţii ale semidreptei cu laturile lui $F$ este par şi este interior lui $F$ dacă acest număr este impar (figura 5.9.b).

### Teste minimax

Testele minimax sunt utile în diverse momente ale algoritmilor: la calculul suprapunerii a două poligoane în planul de proiecţie, la calculul interiorităţii proiecţiei unui punct faţă de proiecţia unui poligon, la calculul intersecţiei a două segmente, la calculul de adâncime a unui punct faţă de un poligon sau chiar a două poligoane între ele în spaţiul obiect.

Aceste teste presupun doar *comparaţii*. Ele pot duce la respingerea a numeroase cazuri nefavorabile, dacă sunt utilizate înaintea calculelor propriu-zise.

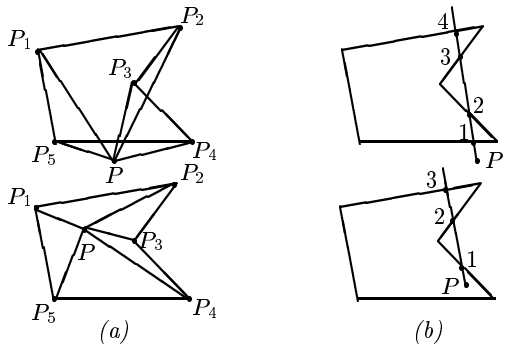



Figura 5.9: Teste de interioritate prin (a) suma unghiurilor (b) număr de intersecţii

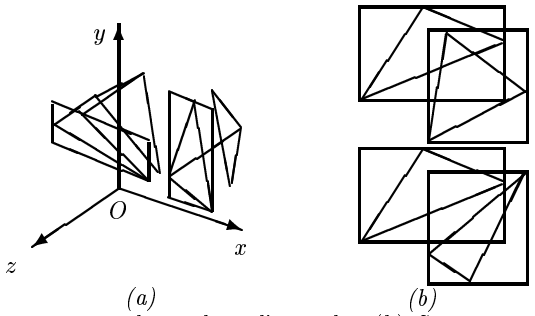



Figura 5.10: (a) Extinderile plane ale poligoanelor (b) Suprapunerea extinderilor plane

De exemplu, două poligoane din planul de proiecţie nu au nici un punct de intersecţie dacă valoarea minimă a proiecţiei pe axa $x$ a celui de al doilea este mai mare decât valoarea maximă a proiecţiei pe aceeaşi axă a primului.

Se numeşte *extindere plană* (extindere ecran) a unui poligon dat din 3D, dreptunghiul de arie minimă circumscris proiecţiei poligonului şi care are o latură orizontală (figura 5.10.a). Pentru determinarea extinderii ecran se calculează maximul şi minimul coordonatelor proiecţiilor pe ecran ale vârfurilor poligonului (patru valori ce definesc colţurile dreptunghiului).

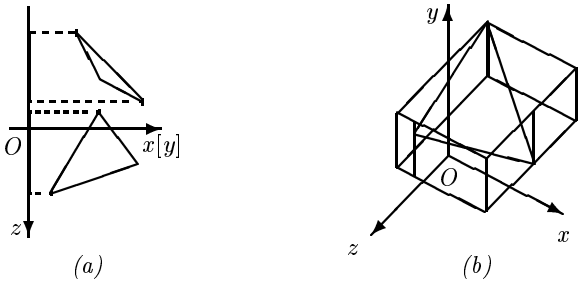
Două poligoane ale căror extinderi ecran sunt disjuncte (nu se suprapun) nu se pot acoperi unul pe altul. Dacă extinderile se suprapun, cu privire la suprapunerea poligoanelor nu se poate afirma nimic. In acest caz există două situaţii (figura 5.10.b):

(a) atât extinderile, cât şi proiecţiile poligoanelor, se suprapun;

(b) extinderile se suprapun, dar proiecţiile poligoanelor sunt disjuncte.

Analog se pot defini extinderile unidimensionale şi spaţiale.

O *extindere unidimensională* se defineşte, relativ la un plan de coordonate, ca fiind zona bandă de lăţime minimă în care se înscrie proiecţia poligonului în acel plan, situată între două drepte paralele şi paralele cu una din axele de coordonate ale planului. Cazuri particulare sunt extinderile după $x$, $y$, $z$ ale poligoanelor (figura 5.11.a).




(a) (b)

Figura 5.11: (a) Extinderea unidimensională după $z$ (b) Extinderea spaţială

Când extinderile după $z$ a două poligoane nu se suprapun, atunci poligonul cu extinderea după $z$ mai apropiată de observator va acoperi celălalt poligon (cel puţin parţial) dacă extinderile ecran se intersectează. Determinarea extinderii după $z$ se face căutând maximul şi minimul valorilor lui $z$ pentru vârfurile poligonului considerat.

*Extinderile spaţiale* ale poligoanelor sunt paralelipipedele de volum minim în care se înscriu poligoanele respective şi care au feţele paralele cu planele triedrului de referinţă (figura 5.11.b).

Inainte de a compara două poligoane, se compară extinderile acestora. Se elimină astfel o bună parte din calculele de intersecţii, simplificând şi accelerând tratarea imaginii.

### Alunecare spate-faţă

Dacă un obiect este aproximat printr-un poliedru solid, feţele lui poligonale închid volumul. Dacă toate feţele sunt definite astfel încât normalele la suprafeţe sunt orientate în exterior, atunci poligoanele a căror normală este orientată în direcţia opusă observatorului, nu vor fi vizibile de către observator. O asemenea faţă poate fi identificată prin $D \cdot N < 0$, unde $D$ este raza de la observator la suprafaţă, iar $N$ este normala la suprafaţă. Feţele astfel determinate pot fi eliminate din calculele de vizibilitate, dacă obiectul nu are găuri (figura 5.12).

### Partiţionarea spaţială

Permite împărţirea unei probleme mari într-un număr de probleme mai mici. Ca pas de preprocesare, se asignează obiecte sau proiecţiile lor la grupuri coerente spaţial. De exemplu, planul de proiecţie se poate împărţi cu o grilă rectangulară şi se determină în fiecare spaţiu al grilei care proiecţii ale obiectelor se află în ele. Astfel proiecţia unui obiect trebuie comparată, pentru testul de suprapunere, numai cu acele proiecţii care apar în spaţiile grilei pe care le ocupă. *Partiţionarea spaţială* presupune utilizarea unei grile tridimensionale. Procesul de determinare a obiectelor intersectate de un proiector poate fi accelerat prin determinarea în primul rând a partiţiilor pe care proiectorul le intersectează, şi numai apoi testarea intersecţiei cu obiectele asociate cu aceste partiţii. Dacă obiectele sunt distribuite inegal în spaţiu, este mult mai eficientă o *partiţionare adaptivă*, în care mărimea partiţiei variază. O posibilitate pen-

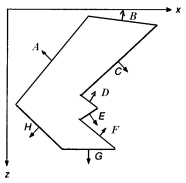



Figura 5.12: Secţiune printr-un poliedru: poligoanele $A$, $B$, $D$, $F$ sunt poligoane de spate, iar poligoanele $C$, $E$, $G$, $H$ sunt poligoane de faţă

tru a realiza acest deziderat este subdivizarea recursivă a spaţiului până când un anumit criteriu de terminare este împlinit (de exemplu, subdivizarea poate fi oprită când în fiecare partiţie există un număr de obiecte mai mic decât un număr prestabilit).

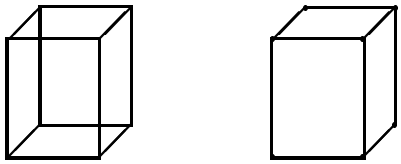
### 5.2.2 Algoritmi spaţiu-obiect pentru determinarea liniilor vizibile

#### Algoritmul `Robertson`

Este primul algoritm care a fost propus pentru determinarea liniilor vizibile ale unui poliedru. Se presupune că fiecare segment de linie este muchie pentru un poligon convex. Iniţial se utilizează alunecarea faţă-spate pentru a înlătura muchiile comune unei perechi de poligoane de spate. Apoi fiecare muchie rămasă este comparată cu fiecare poliedru. Se aplică testele minimax pentru eliminarea poliedrelor a căror extinderi nu se intersectează cu cele ale segmentului. Un poliedru poate ascunde segmentul de linie sau poate determina ca întregul segment, o parte sau două a segmentului să rămână vizibile. Segmentele rămase sunt comparate cu următorul poliedru. Testul de vizibilitate se bazează pe o versiune parametrică a proiectorului pornit de la observator la un punct al muchiei testate. Se determină acele valori ale ecuaţiei liniei care fac ca proiectorul să treacă printr-un poliedru, rezultând astfel invizibilitatea punctului de capăt. Proiectorul trece printr-un poliedru dacă conţine anumite puncte care sunt în interiorul tuturor feţelor frontale ale poliedrului.

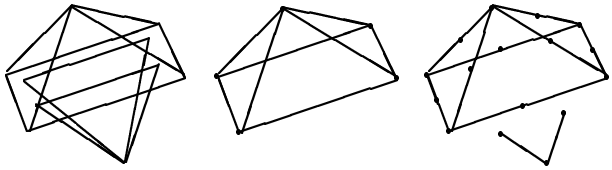
#### Algoritmul reprezentării incomplete

Se consideră o reprezentare poliedrală a unui corp convex şi o mulţime de puncte care definesc această reprezentare (mulţimea include cel puţin vârfurile faţetelor poligonale). Imaginea se reconstituie pe principiul: *un segment este vizibil atunci când capetele sale sunt vizibile*. Se alege ca obiectiv un anumit punct de pe suprafaţa corpului şi se depistează eventualele obstacole ce se interpun între punct şi observator.



*(a)* *(b)*

Figura 5.13: (a) Reprezentarea poliedrală a unui paralelipiped (b) Imagine prin algoritmul reprezentării incomplete



*(a)* *(b)* *(c)*

Figura 5.14: (a) Reprezentarea poliedrală a unui corp concav (b) Imagine prin algoritmul reprezentării incomplete (c) Imagine îmbunătăţită prin creşterea numărului de puncte ce descriu suprafaţa

Pentru determinarea vizibilităţii punctelor reprezentării poliedrale se parcurg următoarele etape:

(a) se iniţializează cu valoarea zero un vector de dimensiune egală cu numărul de puncte prin care este reprezentat corpul. Acest vector este numit *zona atributelor de vizibilitate*. Simbolul 0 marchează punctele vizibile. Valoarea 1 indică un punct invizibil.

(b) pentru fiecare faţă poligonală se parcurg toate punctele reprezentării căutând acelea a căror proiecţie pe ecran se află în interiorul sau pe frontiera proiecţiei faţetei, numite *puncte obiectiv*. Pentru fiecare punct obiectiv,

   1. se calculează intersecţia dintre dreapta de la punctul obiectiv la observator şi suprafaţa poligonală curentă, rezultând un punct numit *punct mască potenţial*;
   2. se compară distanţa de la observator la punctul obiectiv cu cea de la observator la punctul mască potenţial. Dacă prima este mai mare atunci se modifică atributul corespunzător punctului din 0 în 1 (punctul este invizibil).

In final zona atributelor de vizibilitate stochează informaţii complete cu privire la vizibilitatea punctelor prin care corpul este aproximat.

In figura 5.13.(b) se prezintă rezultatul aplicării algoritmului la paralelipipedul (a).

Algoritmul spaţiu-obiect incomplet, ce reprezintă numai curbele sau segmentele cu ambele capete vizibile, se poate aplica cu succes dacă corpul descris este convex, are un număr mic de faţete sau dacă imaginea ocupă o zonă mare pe ecran. Pentru corpurile concave, problema se complică (figura 5.14).

### Algoritmul invizibilităţii cantitative (algoritmul `Appel`)

Algoritmul invizibilităţii cantitative (algoritmul `Appel`) operează asupra solidelor formate din feţe plane mărginite de poligoane. Metoda folosită este aceea de a testa vizibilitatea segmentelor din care sunt alcătuite feţele solidului.

*Invizibilitatea cantitativă* a unui punct al unei linii este numărul de poligoane care maschează respectivul punct. Când o linie trece în spatele unui poligon, invizibilitatea cantitativă a punctelor sale este incrementată cu 1; când iese, este decrementată cu 1. O linie este vizibilă numai dacă invizibilitatea punctelor sale este 0 (figura 5.15).

Se presupune că poligoanele din reprezentare nu se întrepătrund. Atunci o linie îşi schimbă invizibilitatea cantitativă dacă trece prin spatele unei linii de contur.

O *linie de contur* este o muchie comună unei feţe de spate şi a uneia frontale. Se presupune că toate poligoanele au normala la suprafaţă orientată spre exteriorul corpului. Un *poligon de spate* este identificat printr-un produs scalar pozitiv între normala la suprafaţă şi vectorul ce porneşte din centrul proiecţiei la un punct oarecare al poligonului. In figura 5.15.a $AB$, $CD$, $DF$, $KL$ sunt linii de contur, iar $CE$, $EF$, $JK$ nu sunt linii de contur.

O linie de contur trece prin faţa unei muchii dacă străpunge triunghiul format de capetele muchiei şi ochiul observatorului (figura 5.15.b). Se aplică testul punct interior unui triunghi. schimbarea invizibilităţii cantitative este determinat de semnul produsului vectorial dintre muchie şi linia de contur.

Intr-un prim pas al algoritmului, se elimină muchiile care aparţin unor feţe total invizibile (poligoane de spate). In pasul al doilea se determină liniile de contur. Apoi se determină invizibilitatea cantitativă a vârfurilor.

Pentru determinarea vizibilităţii vârfurilor se consideră un vârf iniţial care se proiectează pe toate feţele poligonale ale reprezentării. Se reţin doar proiecţiile interioare feţelor. Numărul proiecţiilor mai apropiate de observator decât vârful studiat indică invizibilitatea cantitativă a acestuia. Această valoare se propagă de-a lungul unei muchii, fiind modificată la trecerea prin spatele unei linii de contur. Când se atinge capătul final al unei muchii, invizibilitatea cantitativă asociată (emanată prin muchie) devine invizibilitatea cantitativă a tuturor liniilor care pornesc din respectivul punct.

In propagarea invizibilităţii cantitative apar probleme la vârfurile liniilor de contur care sunt comune mai multor feţe frontale (de exemplu $JK$ are invizibilitatea cantitativă 0, iar $KL$, 1). Această schimbare poate rezulta dintr-un test suplimentar al muchiei faţă de laturile care împart vârful.

Algoritmul `Appel` presupune că toate laturile poligoanelor sunt trasate într-o direcţie consistentă relativ la poligon, astfel încât semnul schimbării în invizibilitatea cantitativă este determinat de semnul dintre produsul vectorial a laturii cu linia de contur.

Algoritmul poate beneficia de avantajele partiţionării spaţiale. Fiecare muchie trebuie astfel comparată numai cu muchiile sau obiectele din spaţiile grilei care conţin proiecţia sa.

*Aplicaţie.* Se consideră o reprezentare poliedrală prin triunghiuri. Mulţimea liniilor de contur (conturul aparent) este mulţimea segmentelor care au ambele capete vizibile şi sunt laturi ale unor faţete triunghiulare cu două vârfuri vizibile şi unul invizibil. Segmentele din conturul aparent se pot depista uşor după

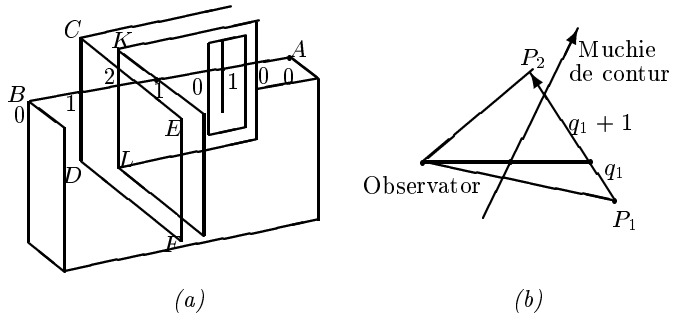



Figura 5.15: (a) Definirea liniilor de contur în algoritmul Appel (b) Schimbarea invizibilităţii cantitative

parcurgerea algoritmului spaţiu-obiect incomplet pe baza consultării zonei atributelor de vizibilitate. Se observă că, în cazul când ambele capete ale unui segment oarecare au acelaşi atribut de vizibilitate, numărul intersecţiilor cu conturul aparent este par, iar atunci când atributele sunt diferite, impar. Fiecare faţă este descrisă prin coordonatele celor trei vârfuri şi proiecţiile lor. Pentru fiecare faţetă se parcurg laturile acesteia şi pentru fiecare latură, numită *latură curentă* se parcurge din nou corpul, faţetă cu faţetă. Pentru fiecare faţetă parcursă, dacă are două vârfuri vizibile şi unul invizibil, atunci latura faţetei cu două capete vizibile face parte din conturul aparent şi este numită *segment curent*. Se proiectează latura curentă şi segmentul curent pe planul de proiecţie. Se calculează intersecţia dintre dreapta suport a laturii curente proiectate şi dreapta suport a segmentului curent proiectat. Dacă intersecţia se află atât între extremităţile laturii curente proiectate cât şi ale segmentului curent proiectat, atunci se stochează coordonatele intersecţiei. Se ordonează intersecţiile obţinute, pornind de la un capăt (ales arbitrar ca origine a laturii curente) către celălalt. Se obţin astfel anumite intervale pe latura curentă. Se consideră intervalul dintre prima extremitate a laturii curente şi prima intersecţie. Pe acest interval latura se consideră vizibilă dacă extremitatea laturii este vizibilă şi invizibilă în caz contrar. Această informaţie se reprezintă printr-o variabilă numită *atribut de vizibilitate al intervalului* (care are două valori, 0 şi 1, la fel ca atributul punctului). Până la epuizarea intervalelor, se trasează sau nu intervalul după cum precizează atributul de vizibilitate al intervalului, apoi se complementează atributul de vizibilitate faţă de 1.

### 5.2.3 Algoritmi spaţiu-imagine

#### Vizualizarea suprafeţelor descrise explicit

Se consideră cazul special al unui suprafeţe descrise explicit printr-o ecuaţie de tipul

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

Prin discretizarea funcţiei $f$ pe o grilă regulată în $x$ şi $z$ se obţine o matrice de valori $y$ reprezentând înălţimi faţă de grila în $x$ şi $z$ aflată în planul $y = 0$. Presupunem că fiecare coloană a matricei corespunde unei singure valori $x$, iar

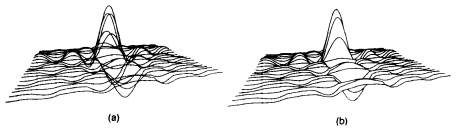



Figura 5.16: Reprezentarea unei suprafeţe explicite (a) fără sau (b)cu ascunderea liniilor

fiecare linie, unei singure valori $z$. Indicii şi valoarea elementului matriceal specifică în mod unic un punct din spaţiu.

O trasare tip cadru de sârmă poate fi construită prin aproximarea liniară (figura 5.16). Se trasează un set de linii poligonale ce trec prin punctele definite de fiecare linie a matricii (linii poligonale de $z$ constant) şi un set de linii poligonale ortogonale care trec prin punctele definite de fiecare coloană (linii poligonale cu $x$ constant).

Se consideră în primul rând problema trasării liniilor poligonale de $z$ constant. Presupunem că cea mai apropiată linie poligonală este o latură a suprafeţei. Liniile poligonale ce urmează a fi trasate se află în plane paralele de $z$ constant. Se trasează liniile în ordinea crescătoare a distanţei de la observator.

Dacă se trasează o nouă linie poligonală, ea este vizibilă numai acolo unde proiecţia sa se ridică deasupra celei mai înalte sau se află sub cea mai joasă siluetă. Deoarece noua linie poligonală se află într-un plan mai îndepărtat decât precedentele, nu poate acoperi liniile poligonale anterioare. Se consideră „frontiera siluetă" a liniilor poligonale care au fost trasate până la un moment dat (figura 5.17). Când o linie poligonală nouă este trasată, ea este vizibilă numai acolo unde proiecţia sa se ridică deasupra acestei siluete sau coboară sub siluetă. De aceea pentru a determina care părţi vor fi trasate, este necesară compararea valorilor $y$ ale proiecţiei cu cele corespunzătoare ale siluetei calculate până în acel moment.

Dacă se utilizează numai informaţia referitoare la valorile $y$ minime şi maxime ale siluetei este vorba de *algoritmul liniei orizontului* (algoritmul `Wright`). Din punct de vedere al clasificării algoritmilor de determinare a liniilor vizibile, acesta este un exemplu de algoritm spaţiu-imagine.

Pentru reprezentarea siluetelor se utilizează două tablouri unidimensionale, $Y_{min}$ şi $Y_{max}$, care conţin valorile minime şi maxime curente ale valorilor proiectate $y$ pentru o mulţime finită de valori $x$ (valorile sunt actualizate după calcularea valorilor $y$ ale fiecărei noi linii). Tablourile $Y_{min}$ şi $Y_{max}$ sunt iniţializate cu valori $y$ care sunt sub, respectiv deasupra, tuturor valorilor $y$ proiectate ale suprafeţei. Când se trasează o nouă linie, valorile $y$ proiectate ale fiecărei perechi de vârfuri adiacente sunt comparate cu valorile locaţiilor corespunzătoare în tablourile unidimensionale (figura 5.18). Dacă ambele capete ale segmentului de linie sunt invizibile, tablourile siluetelor rămân neschimbate (segmentul $CD$). Dacă ambele capete ale segmentului sunt vizibile rela-

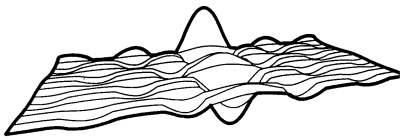


Figura 5.17: Siluete

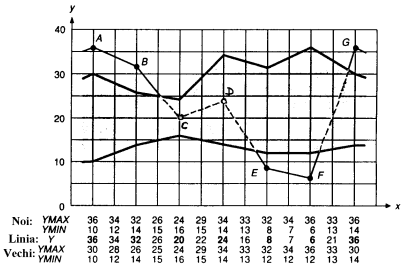



Figura 5.18: Valorile $y$ pentru poziţiile de pe linie între punctele de capăt trebuie interpolate şi comparate cu cele din matricele siluete

tiv la una dintre siluete, atunci segmentul de linie este vizibil în întregime şi tablourile siluetelor trebuie actualizate (cazul $AB$ sau $EF$). Coordonatele $x$ a două vârfuri adiacente într-o linie poligonală de $z$ constant de obicei corespund cu locaţii neadiacente în tablourile siluetelor. In această situaţie, valorile $y$ care sunt inserate în locaţii intermediare ale siluetelor pot fi obţinute prin interpolare liniară între valorile $y$ proiectate a două elemente adiacente a lui $Y$. In cazurile în care fie doar un vârf este vizibil şi celelalt este invizibil (cazul $BC$ sau $DE$), fie ambele vârfuri sunt vizibile, dar faţă de siluete diferite (cazul $FG$), valorile $y$ interpolate trebuie comparate cu cele ale locaţiilor intermediare din tablourile siluetelor pentru a determina punctele de intersecţie.

Calitatea imaginii produse de algoritm depinde de relaţia dintre numărul de puncte considerate pentru o siluetă şi rezoluţia dispozitivului de afişare In figura 5.19 a treia linie nu va fi trasată deşi trece peste intersecţia celorlalte două. (este necesară utilizarea unor tablouri-siluete cu rezoluţie mai înaltă).

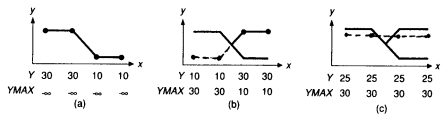



Figura 5.19: Problemă ce intervine datorită utilizării preciziei imagine pentru siluetă

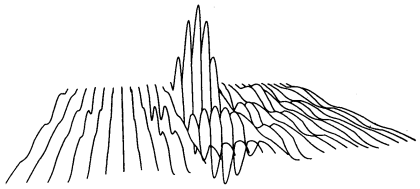


Figura 5.20: Suprafaţă trasată utilizând polilinii de $x$ constant

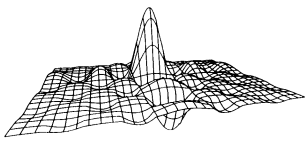


Figura 5.21: Suprafaţă trasată prin linii poligonale cu $x$ şi $z$ constant

Dacă se trasează liniile poligonale cu $x$ constant, linia poligonală cu $x$ constant cea mai apropiată de observator (dreaptă în figura 5.20) nu formează o latură a suprafeţei. Pentru a trasa suprafaţa corect, se trasează liniile poligonale de la dreapta celei mai apropiate în ordinea de la stânga la dreapta, iar cele la stânga celei mai apropiate, de la dreapta la stânga. In ambele cazuri, trasarea liniei poligonale se face în ordinea crescătoare a distanţei de la observator.

Suprapunerea liniilor poligonale cu $x$ constant şi $z$ constant (figura 5.21) nu permite întotdeauna o reprezentare corectă a suprafeţei. Un exemplu concludent este prezentat în figura 5.22. Soluţia corectă presupune trasarea concomi-

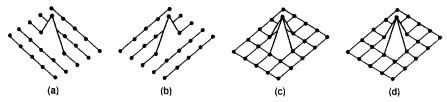



Figura 5.22: (a) Linii cu $z$ constant; (b) Linii cu $x$ constant; (c) Suprapunerea figurilor (a) şi (b); (d) Soluţia corectă

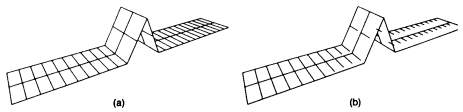



Figura 5.23: Liniile poligonale cu $x$ constant trebuie să fie procesate în ordinea corectă: (a) linii procesate corect (b) linii procesate incorect

tentă a liniilor poligonale. Setul de linii poligonale care sunt aproape paralele cu planul de vizualizare (în figura dată se consideră cele de $z$ constant) sunt procesate în ordinea descrisă mai sus. După ce o linie poligonală de $z$ constant este trasată, se construiesc segmentele de linie poligonală cu $x$ constant dintre linia poligonală nou trasată şi următoarea linie poligonală. Segmentele de linie cu $x$ constant se trasează utilizând aceeaşi structură de date-siluete care a fost utilizată pentru trasarea liniei poligonale de $z$ constant. Procesarea are loc în ordinea crescătoare a distanţei faţă de observator. In figura 5.23.b liniile cu $x$ constant au fost procesate în ordine incorectă, de la dreapta la stânga (linii succesive sunt umbrite în matricea $Y_{max}$ de liniile trasate anterior).

### Algoritmul tamponului de adâncime

Algoritmul $z$-`buffer` sau al tamponului de adâncime (algoritmul `Catmull`) este destinat eliminării suprafeţelor ascunse şi este unul dintre cei mai simplii algoritmi de vizibilitate.

Este necesar nu numai `frame-buffer`-ul $F$ în care sunt stocate culorile pixelilor, ci şi un $z$-`buffer` $Z$ cu acelaşi număr de intrări în care o valoare $z$ este stocată pentru fiecare pixel. $z$-`buffer`ul este iniţializat la valoarea $z$ a planului de decupare din spate, iar `frame buffer`ul este iniţializat cu culoarea fondului. Cea mai mare valoare care va fi reprezentată în $z$-`buffer` corespunde planului de decupare frontal, cel mai apropiat de observator. Primitivele sunt convertite prin baleiere în `frame-buffer` într-o ordine aleatoare. Dacă un punct al primitivei este proiectat şi este mai apropiat de observator decât punctul curent înregistrat în `buffer`e, atunci culoarea şi adâncimea acestuia vor înlocui valori vechi.

Etapele *algoritmului z-buffer-ecran* pentru o reprezentare poliedrală sunt următoarele:

(a) se creează o matrice, fiecărui pixel corespunzându-i un element din matrice, numită $z$- `buffer`-ecran; se iniţializează $z$-`buffer`-ul la o valoare foarte mică pe care suntem siguri că $z$-tul punctelor corpului nu o pot atinge şi se iniţializează ecranul la valoarea de intensitate corespunzătoare fondului.

(b) pentru fiecare faţă poligonală se transpune poligonul astfel: pentru fiecare punct-pixel de coordonate $(q, w)$ din interiorul sau de pe frontiera proiecţiei poligonului:

1. se reţine cota $z$ a punctului care are pe ecran proiecţia în pixelul $(q, w)$;
2. se compară $z$ cu $z(q, w)$, valoarea existentă în $z$-`buffer` la linia $w$, coloana $q$. Dacă este mai mare (punctul este mai apropiat de observator) scrie $z$ în $z$-`buffer` la linia $w$ şi coloana $q$ şi aprinde pixelul $(q, w)$ conform informaţiei de culoare sau umplere a poligonului curent.

Algoritmul oferă avantajul de a putea trata suprafeţe strâmbe (cuadrice, suprafeţe parametrice bicubice). Dezavantajul constă în necesarul de memorie pentru funcţionarea eficientă. Pentru o rezoluţie de $512 \times 512$ şi o reprezentare a valorilor $z$ pe 4 octeţi, necesarul de memorie depăşeşte 1Mo.

Procedurile `WritePixel` şi `ReadPixel` dintr-o procedură de trasare a unei primitive trebuie suplimentate cu procedurile `WriteZ` şi `ReadZ` care citesc şi scriu în $z$ buffer.

```
Procedure zBuffer()
var
   pz : integer;
begin
   for y := 0 to Ymax do
       for x := 0 to Xmax do
           begin
                WritePixel(x, y,Background_value);
                WriteZ(x, y, 0)
           end
   for each polygon do
       for fiecare pixel din proiecţia poligonului do
         begin
           pz :=valoarea z a poligonului proiectat pe pixelul (x, y);
           if pz >=ReadZ(x, y) then
            begin
              WriteZ(x, y, pz);
              WritePixel(x, y,cul.poligon proiectat pe pixel (x, y))
            end
         end
end;
```

Figura 5.24 arată ce este posibil a se întâmpla la adunarea a două poligoane ($z$-ul este indicat prin număr, culoarea prin nuanţe de gri).

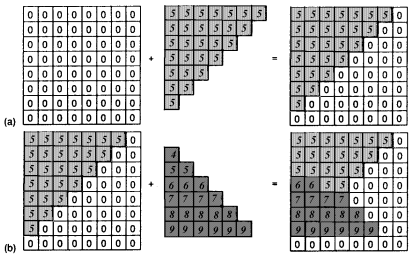


Figura 5.24: $z$-`buffer`: culoarea unui pixel este indicată prin nuanţă de gri, iar valoarea $z$ este prezentată ca număr (a) adăugarea unui poligon cu $z$ constant în $z$-`buffer`ul gol (b) adăugarea unui alt poligon care-l intersectează pe primul

In mod normal $z$ poate fi recuperat din ecuaţia planului poligonului. Fie $ax + by + cz + d = 0$ ecuaţia acestui plan. Atunci $z = -(ax + by + d)/c$. Dacă valoarea $z_1$ este determinată la $(x, y)$, putem determina valoarea $z$ la $(x+\Delta x, y)$ printr-o simplă incrementare a valorii $z_1$ cu $-a\Delta x/c$ (coerenţă în adâncime). In convertirea prin scanare $\Delta x = 1$ iar incrementul rămâne constant. Un calcul similar se poate efectua pentru a determina prima valoare $z$ de pe următoarea linie de scanare (incrementul fiind $-b/c$).

### Algoritmul liniei de baleiaj

Algoritmul `scan-line` operează în spaţiul-imagine creând imaginea linie cu linie. Diferenţa faţă de algoritmul de conversie `scan` constă în faptul că se tratează o mulţime de primitive, nu numai una singură.

Se consideră o reprezentare poliedrală. Se presupune că nu există poligoane care se străpung. Astfel calculul de adâncime necesare la compararea a două poligoane se face o singură dată. Se ţine o gestiune a intersecţiilor şi a proiecţiilor muchiilor şi poligoanelor pe $xOy$ prin:

1. o *listă a muchiilor* `ET` (`edge table`). Intrările în tabel sunt sortate în grupuri pe baza coordonatei $y$ cea mai mică a laturii şi, intrările unui grup, pe baza coordonatelor $x$ ale punctului de căpăt cu $y$ minim, iar intrările cu acelaşi punct de minim, în ordinea crescătoare a $x$-ilor corespunzător celuilalt capăt (cu $y$ maxim). Fiecare intrare conţine:
   (a) coordonata $x$ a extremităţii cu cel mai mic $y$,
   (b) coordonata $y$ a celuilalt capăt (maximă),
   (c) incrementul în $x$, inversul pantei muchiei, utilizat în saltul de la o linie de baleiaj la alta,

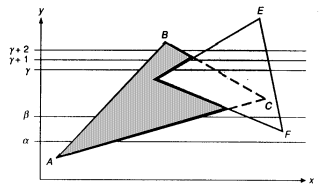



Figura 5.25: Două poligoane procesate cu algoritmul `scan-line`

(d) codul de identificare al poligonului care conţine muchia (o muchie apare de mai multe ori în listă, corespunzător fiecărui poligon);

2. o *listă a poligoanelor* `PT` (`poligon table`). Intrarea corespunzătoare unui poligon conţine informaţiile:
   (a) codul de identificare,
   (b) coeficienţii ecuaţiei planului suport,
   (c) informaţii privind culoarea sau umplerea poligonului,
   (d) o variabilă de stare, iniţializată de-a lungul procesului de baleiaj, care indică poziţia poligonului relativă la linia de baleiaj (aceasta se află în interior sau exterior poligon);
3. o *listă a muchiilor active* `AET` (`active edge table`) ţine evidenţa muchiilor intersectate de linia de baleiaj în ordinea crescătoare a $x$-ilor corespunzători intersecţiilor.

Primul pas al algoritmului constă în crearea primelor două liste. Laturile sunt procesate de la stânga la dreapta. Apoi se baleiază ecranul linie cu linie. Pentru fiecare linie, se crează lista muchiilor active şi se ordonează intersecţiile în aceasta. Se parcurge pixel cu pixel linia curentă. Pentru fiecare pixel se compară abscisa curentă cu intersecţiile din al treilea tabel. Dacă abscisa corespunde unei intersecţii, atunci:

1. se identifică poligonul, se modifică variabila de stare prin complementare (presupunem 1=în intersecţie, 0=afară);
2. se testează variabilele de stare:
   (a) dacă o singură variabilă are valoarea 1, linia de baleiaj taie în acel punct proiecţia unui singur poligon şi valorile corespunzătoare umplerii poligonului se trec în frame-buffer;
   (b) altfel, se calculează pentru fiecare poligon cota $z$ din ecuaţia planului (din `PT`). Se aprinde pixelul conform informaţiilor corespunzătoare poligonului celui mai apropiat ($z$-ul cel mai mare în ipoteza observatorului la $z$ pozitiv).

Se consideră cazul celor două poligoane din figura 5.25, pentru care

$$\texttt{ET} = (AB, AC, FD, FE, DE, CB),$$

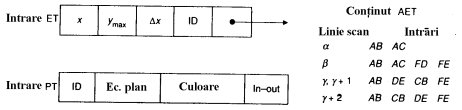



Figura 5.26: `ET`, `PT` şi `AET` pentru algoritmului liniei de baleiaj

$$\texttt{PT} = (ABC, DEF).$$

Variaţia listei muchiilor active este prezentată în figura 5.26. Când linia de baleiaj cu $y = \alpha$ intersectează $AB$, variabila de stare a poligonului $ABC$ se schimbă. Fiind o unică intersecţie, intersecţia dintre poligon şi linia de baleiaj este vizibilă. La intersecţia cu $AC$, variabila de stare este complementată. Inainte de trecerea la noua linie de baleiaj se actualizează `AET`. La $y = \beta$ există două poligoane care se intersectează cu linia de baleiaj, dar la un moment dat doar un poligon este vizibil. La $y = \gamma$ şi intersecţia cu $DE$, două poligoane au variabila de stare setată. Se evaluează $z$ din ecuaţia planelor pentru $y = \gamma$ şi $x$ corespunzător intersecţiei. In cazul dat, $z$-ul poligonului $DEF$ este mai mare, deci punctul este vizibil şi determină caracteristicile pixelului. Dacă poligoanele se întrepătrund, algoritmul este modificat pentru a determina punctele de penetraţie în linia de baleiaj (liste noi `ET`, `PT` şi `AET`).

Algoritmul poate fi utilizat şi pentru suprafeţe curbe – `ET` şi `AET` sunt înlocuite cu o tabelă de suprafeţe, repectiv o tabelă de suprafeţe active.

### Algoritmi de subdivizare a ariilor în arii elementare

Algoritmii de subdivizare a ariilor exploatează coerenţa ariilor.

Se consideră o reprezentare poliedrală. Se presupune că poligoanele nu se întrepătrund. O *arie elementară* este o zonă a imaginii în care se poate determina uşor care poligon dintr-o mulţime dată este vizibil.

Algoritmii bazaţi pe subdivizarea ariilor utilizează o strategie `divide-et-impera`. Prin împărţirea imaginii în zone de arie din ce în ce mai mică se caută obţinerea de zone în care să se situeze cel mult o parte a unui poligon din reţea. Divizarea se opreşte cel târziu în stadiul în care elementul de arie devine un pixel.

**Algoritmul Warnock.** Fiecare arie este divizată în patru arii de mărimi egale. In fiecare etapă a procesului recursiv, *proiecţia* fiecărui poligon se află, relativ la aria de interes, în una din următoarele situaţii (figura 5.27):

(a) conţine aria de interes;
(b) intersectează aria;
(c) este inclus în arie;
(d) este disjunct ariei.

In procesul recursiv, aria de interes este tratată astfel:

1. dacă aria nu conţine nici un poligon, atunci este afişată culoarea de fond;

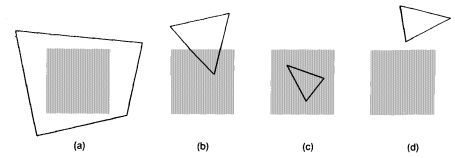



Figura 5.27: Relaţii ale proiecţiei unui poligon cu elementul de arie: (a) înconjurător (b) intersectează (c) conţinut sau (d) disjunct

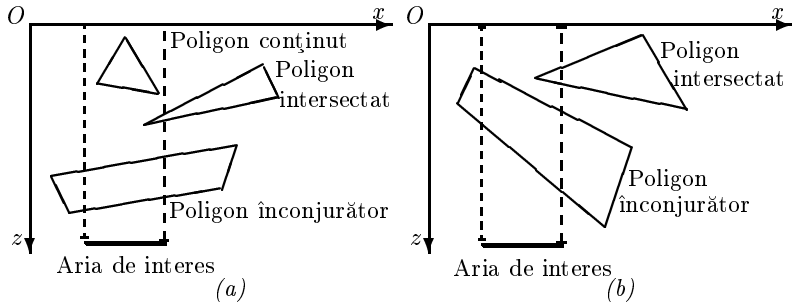



Figura 5.28: Două situaţii de înconjurare a ariei de interes în algoritmul `Warnock`

2. dacă există un singur poligon cu care se intersectează sau care este conţinut în arie, atunci aria este colorată iniţial cu culoarea fondului şi apoi partea poligonului conţinută în arie este convertită prin baleiere;
3. dacă există un singur poligon înconjurător, dar nu şi poligoane intersectate sau poligoane conţinute, atunci aria este umplută cu culoarea poligonului;
4. cazul contrar este cel al mai multor poligoane intersectate, conţinute sau înconjurătoare. Algoritmul impune divizarea ariei până când există un poligon înconjurător care se află în faţa celorlalte poligoane, întreaga arie fiind umplută cu culoarea acestuia.

Testul de poligon frontal (cazul 4) se realizează prin calcularea coordonatelor $z$ ale planelor tuturor poligoenelor înconjurătoare, care intersectează sau sunt conţinute la cele colţuri ale ariei; dacă există un poligon înconjurător a cărui valori $z$ ale colţurilor sunt mai mari decât ale celorlalte poligoane, atunci întreaga arie este umplută cu culoarea poligonului înconjurător. In figura 5.28.a, cele patru intersecţii ale poligonului înconjurător sunt toate mai apropiate de punctul de vedere (în cazul aceestei figuri, punctul de vedere se află la infinit pe axa $z$) decât sunt intersecţiile celorlalte poligoane (se va umple aria cu culoarea acestui poligon). In situaţia din figura 5.28.b, deşi poligonul intersectat se află de partea opusă planului poligonului înconjurător, algoritmul cere divizarea ariei până când poligonul înconjurător are valorile $z$ cele mai

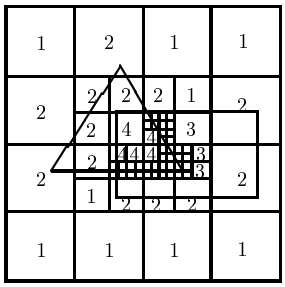


Figura 5.29: Exemplu pentru algoritmul `Warnock`

mari sau se întâlnesc cazurile 1, 2 sau 3.

In figura 5.29 se consideră un exemplu de subdivizare (triunghiul se află în faţa dreptunghiului). Numerele înscrise în pătrate se referă la cazurile mai sus menţionate.

Algoritmul operează în spaţiul obiect, cu excepţia conversiei `scan`. Cea mai mică arie este pixelul. La o rezoluţie de $1024 \times 1024$ sunt necesare cel mult zece nivele de subdivizare.

**Algoritmul Weiler-Atherton.** Subdivizarea ecranului este realizată de-a lungul frontierei poligoanelor (în locul dreptunghiurilor din algoritmul `Warnock`). Algoritmul este asociat cu o metodă de clipping. Fiind un algoritm de subdivizare a ariilor, a fost inclus în această secţiune, deşi este un algoritm hibrid.

Primul pas constă în sortarea poligoanelor funcţie de valoarea $z$ cea mai mare. Poligonul cel mai apropiat de observator este utilizat în procesul de decupare al celorlalte poligoane, rezultând două liste conţinând părţi ale acestora situate în interiorul, respectiv exteriorul proiecţiei primului poligon. Poligoanele din lista de interior se află, teoretic, în spatele poligonului şi, deci, trebuie şterse, fiind invizibile. Dacă un poligon din lista de interior este mai apropiat decât poligonul de decupare, sortarea iniţială nu oferă o ordine corectă a priorităţilor. Un asemenea poligon frontal este procesat recursiv pentru decupare faţă de părţile aflate în lista de interior. Când divizarea recursivă este terminată, algoritmul continuă procesarea poligoanelor din lista de exterior. Decuparea este realizată cu o copie a poligonului iniţial şi nu cu fragmentele acestuia, procesul de decupare fiind mai puţin costisitor.

Algoritmul utilizează o stivă pentru a trata cazurile de suprapunere, ca cel din figura 5.30.c. Stiva conţine o listă a poligoanelor care sunt poligoane curente de decupare, dar a căror utilizare a fost întreruptă datorită subdivizării recursive. Dacă un poligon din lista de interior se află în faţa poligonului de decupare, el este căutat în stivă. Dacă este găsit în stivă, nu mai sunt necesare alte recursii, deoarece toate bucăţile de poligon din interiorul poligonului aflate în spate au fost deja eliminate.

Algoritmul este descris în următoarea procedură:

```
Procedure WA_visibleSurface()
begin
```

```
      polyList :=lista unor copii a tuturor poligoanelor;
      sortează polyList în ordine descrescătoare după valoarea z
      clear stiva;
      while polyList <>nil do
            WA_subdivide(primul poligon din polyList, polyList)
end;
procedure WA_subdivide(
 clipPolygon : polygon; var polyList : ListOfPolygons)
var
   inList, outList : ListOfPolygons;
begin
   inList :=nil; outList :=nil;
   for fiecare poligon în polyList do
       begin
       decupează poligonul relativ la clipPolygon,
       piesele interioare se plasează în inList,
       piesele exterioare se plaseză în outList
       înlătură poligoanele din spatele clipPolygon din inList
       end;
   for fiecare poligon din inList care nu este în stiva
         şi nu este parte a lui clipPolygon do
       begin
            push clipPolygon în stiva;
            WA_subdivide(polygon, inList);
            pop stiva
       end;
   for fiecare poligon din inList do
       afişează poligon;
   polyList := outList
end;
```

**Algoritmul Encarnacao al reţelei de explorare.** Se aplică la suprafeţele curbe care sunt definite ca reţele de linii. Fragmentele (careurile) suprafeţelor curbe sunt specificate numai prin cele 4 vârfuri ale lor, iar liniile drepte sunt folosite pentru a defini muchiile lor. Numele algoritmului derivă din faptul că o reţea rectangulară bidimensională de linii, numită *reţea de explorare*, este suprapusă pe planul-imagine al suprafeţelor. Se constituie o *listă de fragmente de suprafaţă* pentru fiecare arie a reţelei de explorare. Dacă în careu se află prea multe elemente de suprafaţă se descompune careul de rastru în patru subcareuri. Testele mimimax sunt folosite pentru a stabili care arii de explorare vor fi suprapuse de un fragment specific de suprafaţă. Imediat ce s-a terminat preselectarea (constituirea listei), se execută algoritmul principal de vizibilitate. Un segment de linie este rupt într-un şir de puncte de test, spaţiate în mod egal între vârfuri. Fiecare punct este testat pentru vizibilitate faţă de fragmentele de suprafaţă care se află în interiorul aceleiaşi arii de explorare cu punctul de testare. Ori nici un fragment din suprafaţă nu acoperă punctul de testare, în care caz punctul de test este vizibil, ori cel puţin un fragment acoperă punctul de testare, în care caz este invizibil (se compară coordonata $z$ a punctului de

test cu coordonatele $z$ ale planurilor asociate fiecărui fragment de suprafaţă din lista asociată ariei curente de explorare). Dacă toate punctele care se află de-a lungul muchiei fragmentului sunt vizibile, întreaga muchie se trasează. Dacă se constată că unele puncte sunt invizibile, sunt trasate numai acele secţiuni ale muchiei fragmentului care se află între punctele vizibile de test.

### 5.2.4 Algoritmi hibrizi: algoritmi cu listă de prioritate

Algoritmii cu listă de prioritate determină ordinea vizibilităţii obiectelor. De exemplu, dacă nici un obiect nu se suprapune cu altul în extensiile după $z$, atunci este necesară doar o sortare a obiectului în ordinea crescătoare a lui $z$ şi trasarea lor în ordine. Poligoanele mai apropiate acoperă ulterior poligoanele mai îndepărtate deja trasate. Dacă obiectele se suprapun în extensiile lor după axa $z$ este necesară împărţirea lor în mai multe obiecte astfel încât ordonarea menţionată să fie posibilă.

Un algoritm de acest tip este *hibrid*, deoarece combină operaţii în spaţiul obiect cu operaţii în spaţiul imagine. Comparările şi împărţirea obiectelor se realizează cu precizie obiect. Conversia scan care permite rescrierea pixelilor este realizată cu precizie imagine. Deoarece lista obiectelor sortate este creată cu precizie obiect, se poate realiza o retrasare corectă la orice rezoluţie.

Algoritmii cu listă de prioritate diferă în modalitatea în care se determină ordinea obiectelor.

#### Algoritmul de sortare în adâncime

Algoritmul de sortare în adâncime (`depth-sort`), cunoscut şi sub numele de *algoritmul pictural* sau algoritmul `Newell-Newell-Sancha`, converteşte poligoanele unei reprezentări poliedrale în `frame-buffer` în ordinea descrescătoare a distanţei de la observator. Se parcurg trei etape:

- se sortează poligoanele conform ordinii crescătoare a valorii $z$ celei mai mici a vârfurilor (sortare în spaţiu-obiect);
- rezolvă orice ambiguitate cauzată de suprapunerea extensiilor $z$ (comparare în spaţiu- imagine) prin descompunerea poligoanelor la nevoie;
- se transpun prin baleiaj, pe rând, poligoanele pe ecran, în ordinea crescătoare a celei mai mici coordonate $z$. Poligoanele îndepărtate sunt primele transpuse pe ecran. Cele mai apropiate le maschează prin suprascriere.

Compararea dintre două poligoane presupune cinci nivele de testare (eliminatorii). Dacă extinderile $z$ a două poligoane consecutive $P_1$ şi $P_2$ din lista sortată se suprapun, atunci:

(1) Se compară extinderile după $x$. Dacă sunt disjuncte, poligoanele nu se suprapun;

(2) Se compară extinderile după $y$. Dacă sunt disjuncte, poligoanele nu se suprapun;

(3) Se testează dacă $P_1$ este în partea opusă observatorului faţă de planul lui $P_2$. Se determină ecuaţia planului lui $P_2$. Planul împarte spaţiul în 2 zone. Dacă $P_1$ este în întregime în zona mai îndepărtată, atunci este transpus primul pe ecran;

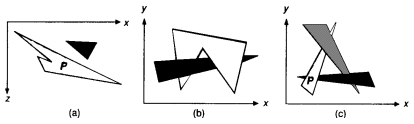



Figura 5.30: Cazuri în care extinderile după $z$ a poligoanelor se suprapun

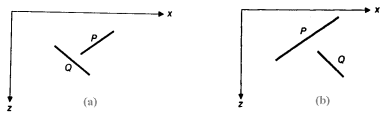



Figura 5.31: (a) Testul 3 este fals, dar testul 4 este adevărat (b) Testul 3 este adevărat

(4) Se testează dacă $P_2$ este de aceeaşi parte cu observatorul faţă de planul lui $P_1$. Se determină ecuaţia planului lui $P_1$. Dacă $P_2$ se află în întregime în zona mai apropiată de observator, $P_1$ este transpus pe ecran;
(5) Se testează dacă proiecţiile poligoanelor în planul $(x, y)$ nu se suprapun. Dacă nu se suprapun, $P_1$ poate fi trasat înaintea lui $P_2$ (figurile 5.30 şi 5.31).

Dacă cele 5 teste relativ la faptul că $P_1$ nu acoperă pe $P_2$ se termină cu succes, trebuie testat următorul poligon care se suprapune în extinderea $z$ cu $P_1$. Dacă unul din teste eşuează, se presupune pe moment că $P$ ascunde pe $Q$ şi de aceea trebuie testat dacă $Q$ poate fi trasat înaintea lui $P$. Testele 1, 2 şi 5 nu trebuie repetate, se repetă numai testele 3 şi 4 în variantă modificată. In cazul figurii 5.30.a, testul 3 modificat va avea succes. In cazul figurii 5.30.b, testele sunt neconcludente; unul dintre poligoane trebuie divizate de planul celuilalt, astfel încât poligonul ce se divizează se şterge din listă, fiind înlocuit cu părţile sale. In cazul figurii 5.30.c se va crea o buclă în algoritm deoarece între fiecare două poligoane se poate stabili o ordine. Pentru a evita ciclarea poligonul introdus ultimul în listă va fi marcat. Dacă primele cinci teste au eşuat şi poligonul curent $P_2$ este marcat, atunci nu se aplică testele 3 şi 4 modificate, ci se divide $P_1$ sau $P_2$. Dacă cele toate testele eşuează, se compară proiecţiile celor două poligoane. Se calculează intersecţiile latură cu latură ale celor 2 proiecţii. Dacă cel puţin una din intersecţii este simultan inclusă în laturile ambelor poligoane, atunci poligoanele se suprapun, se determină intersecţia celor două poligoane şi fiecare poligon este înlocuit în listă prin poligoanele componente. In caz contrar, poligoanele nu se maschează unul pe altul şi se transpun pe ecran în oricare ordine.

### Algoritmul arborelui binar de partiţionare spaţială (BSP)

Este o metodă extrem de eficientă pentru determinarea relaţiilor de vizibilitate între grupuri statice de poligoane tridimensionale. Este indicată utilizarea în special în cazul în care poziţia observatorului se schimbă, nu însă şi poziţia obiectelor. Se bazează pe următoarea observaţie. Dacă un plan separă o grup de poligoane de alte grup, iar grupul este de aceeaşi parte a planului cu observatorul, atunci grupul poate ascunde, dar nu poate fi ascuns de celelalte grupuri. Fiecare grup poate fi subdivizat dacă se aleg plane corespunzătoare de separare. Partiţionarea mediului poate fi reprezentată printr-un arbore binar cu rădăcina în primul plan de partiţionare ales. Nodurile interne sunt de asemenea plane de partiţionare; nodurile terminale sunt regiuni în spaţiu.

Algoritmul permite determinarea ordinii corecte pentru convertirea poligoanelor prin construirea unui arbore binar de poligoane, numit arborele BSP. Rădăcina acestui arbore este un poligon care este utilizat pentru a împărţi mediul în două semispaţii. Un semispaţiu va conţine toate poligoanele rămase în faţa poligonului rădăcină, relativ la normala la suprafaţă; celălalt subspaţiu conţine poligoanele din spatele poligonului rădăcină. Orice poligon care are părţi în ambele subspaţii va fi divizat relativ la planul poligonului rădăcină. Câte unul dintre poligoanele frontale şi de spate vor deveni fii nodului rădăcină şi fiecare fiu este utilizat recursiv pentru a determina subgrupuri de poligoane relative la planele fiilor. Construcţia arborelui se termină când fiecare nod conţine numai un singur poligon (figura 5.32).

Odată construit arborele, acesta poate fi traversat într-o manieră asemănătoare inordinii pentru a stabili prioritatea de afişare a poligoanelor dintr-un punct arbitrar de vedere. Dacă observatorul este în semispaţiul frontal al poligonului rădăcină, atunci algoritmul trebuie să afişeze prima dată poligoanele din spatele poligonului rădăcină (care pot fi acoperite de acesta), apoi poligonul rădăcină şi în final poligoanele din semiplanul frontal (care pot acoperi poligonul rădăcină). In mod similar se procedează în cazul când observatorul este în semispaţiul din spatele rădăcinii (ordinea de afişare a subarborilor acesteia este inversată).

Dacă poligonul este văzut numai ca o muchie în proiecţie ordinea de afişare a subarborilor stânga-dreapta nu contează (figura 5.33).

```
type
    BSP_tree=record
                root : polygon
                backChild, frontChild :^ BSP_tree
                end;
function BSP_makeTree(poly : listOfPolygons) : ^ BSP_tree;
var
   root : polygon;
   backList, frontList : listOfPolygons;
   p, backPart, frontPart : polygon; {poligoane convexe}
begin
   if polyList este vidă then
      BSP_makeTree:=nil
   else
```

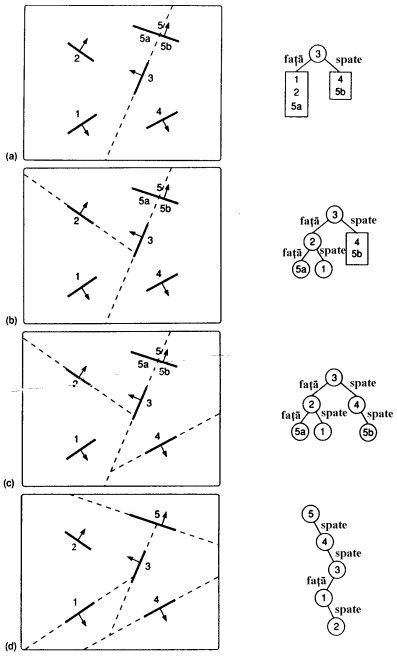



Figura 5.32: Arbore BSP (a) înaintea recursiei cu poligon 3 ca rădăcină (b) construire subarbore stâng (c) complet (d) arbore cu polig.5 ca rădăcină

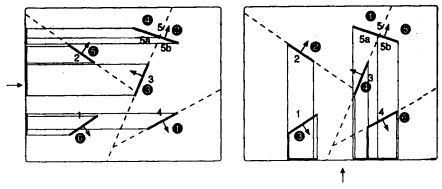


Figura 5.33: Două traversări ale arborelui BSP corespunzătoare la două proiecţii diferite

```
          begin
           root :=BSP_selectAndRemovePoly(polyList);
           backList :=nil; frontList :=nil;
           for fiecare poligon rămas în polyList
            begin
               if poligon p în faţa rădăcinii then
                  BSP_addToList(p, frontList)
               else if poligon p în spatele rădăcinii then
                  BSP_addToList(p, backList)
               else { Poligonul p trebuie divizat}
                begin
                  BSP_splitPoly(p, root, frontPart, backPart);
                  BSP_addToList(frontPart, FrontList);
                  BSP_addToList(backPart, backList)
                end
            end;
           BSP_makeTree:=BSP_combineTree(BSP_makeTree(
             frontList), root,BSP_makeTree(backList))
          end
end;
procedure BSP_displayTree(tree : ^BSP_tree);
begin
   if arbore nu este vid then
      if observator în faţa rădăcină then
       begin
         BSP_displayTree(tree^.backChild);
         displayPolygon(tree^.root);
         BSP_displayTree(tree^.frontChild)
       end
      else
       begin
         BSP_displayTree(tree^.frontChild);
```

```
            displayPolygon(tree^.root);
            BSP_displayTree(tree^.backChild)
          end
end;
```

### 5.2.5 Algoritmul drumului optic (`ray-tracing`)

Algoritmul `ray-tracing` determină vizibilitatea suprafeţelor trasând o rază imaginară de lumină de la ochiul observatorului la obiectele din scenă. Sunt date un centru al proiecţiei (ochiul observatorului) şi o fereastră într-un plan arbitrar.

Fereastra este divizată într-o grilă dreptunghiulară ale cărei elemente corespund pixelilor din imaginea finală. Pentru fiecare element al grilei din fereastră, raza vizuală este trasată de la centrul proiecţiei prin centrul elementului spre scenă (figura 5.34). Pixelul corespunzător elementului se aprinde conform informaţiilor obiectului care este intersectat primul (prototipul algoritmilor spaţiu-imagine). Raza este definită parametric. Pentru fiecare obiect este necesară o reprezentare care permite determinarea cu uşurinţă a intersecţiei cu raza.

Algoritmul `ray-tracing` poate fi sintetizat astfel:

```
selectează centrul proiecţiei şi fereastra din planul de proiecţie
for fiecare linie de baleiaj în imagine do
    for fiecare pixel din linia de baleiaj do
      begin
        determină raza de la centrul de proiecţie până la pixel;
        for fiecare obiect din scenă do
            if obiectul este intersectat şi este cel mai apropiat
              până în acest moment then
                reţine intersecţia şi numele obiectului
        setează culoarea pixelului la cea a obiectului cel mai apropiat
      end;
```

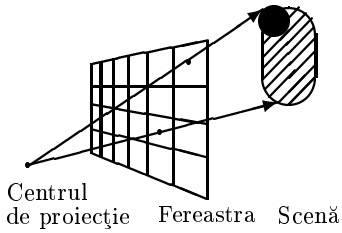



Figura 5.34: Algoritmul drumului optic

Problema principală asociată cu acest algoritm este timpul de calcul al intersecţiilor. Pentru o imagine $1024 \times 1024$ a 100 de obiecte este necesară calcularea a 100M de intersecţii. In consecinţă, îmbunătăţirea eficienţei algoritmului presupune fie accelerarea, fie evitarea calculelor de intersecţii.

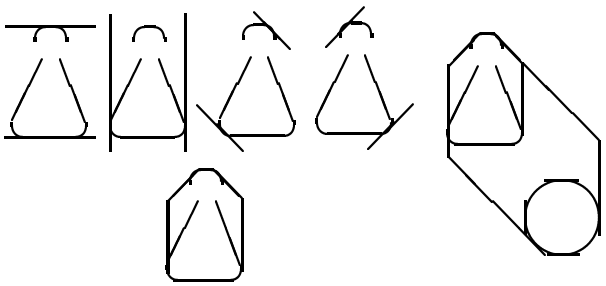


Figura 5.35: (a) Plăcile infinite ce încadrează un obiect bidimensional (b) Ierarhia volumelor

Dacă algoritmul *z*-`buffer` calculează informaţia numai pentru acele obiecte care se proiectează pe pixel, algoritmul `ray-tracing` intersectează (teoretic) fiecare rază cu fiecare obiect al scenei. Algoritmul *z*-`buffer` aproximează obiectele ca mulţimi de valori $z$ în lungul liniilor de proiecţie ce intersectează obiectul, pe când algoritmul `ray-tracing` aproximează obiectele ca mulţimi de intersecţii în lungul fiecărei linii de proiecţie ce intersectează scena.

### Optimizarea calculelor de intersecţii

*Volumele mărginite* oferă o modalitate atractivă de descreştere a timpul necesar calculelor de intersecţii. Un obiect care este relativ greu de testat la intersecţie cu o rază poate fi inclus într-un volum a cărui intersecţii se pot calcula cu mai multă uşurinţă (precum o sferă, un elipsoid, un solid rectangular). Obiectul nu trebuie testate pentru intersecţie dacă raza nu intersectează volumul său de mărginire. Un volum de mărginire poate fi, de exemplu, şi un poliedru convex definit prin perechi de plane paralele ce mărginesc obiectul (figura 5.35.a).

### Evitarea calculelor de intersecţii

Se aplică o tehnică de preprocesare pentru partiţionarea razelor şi obiectelor în clase pentru a limita numărul de intersecţii cerute, pe bază de: *ierarhie*, *partiţionare* sau *modelarea obiectelor prin* `CSG`.

**Ierarhie.** Volumele mărginite pot fi organizate într-o *ierarhie* cu obiectele scenei la nivelul inferior şi volume mărginite ca noduri interne (figura 5.35.b). Un volum-fiu nu va intersecta o rază dacă volumul-tată nu o intersectează. Astfel testul de intersecţie începe cu rădăcina şi multe ramuri ale ierarhiei sunt respinse uşor. Procedura poate fi îmbunătăţită printr-o evidenţă a intersecţiilor corespunzătoare unei raze. Schematizat, parcurgerea ierarhiei se face astfel:

```
procedure HIER_traverse(r : ray;  n : node)
begin
     if r intersectează volumul mărginit a lui n then
        if n este nod terminal then
           intersectează r cu obiectul n
```

```
        else
           for fiecare fiu c a lui n do
               HIER_traverse(r, c)
end;
```

Această procedură poate fi modificată ţinând seama că scopul traversării este găsirea celei mai apropiate intersecţii a razei cu obiectele scenei

```
Procedure KayKajiva;
var
   p:^obiect;
   t : float;
   c : ^nod;
begin
   t := ∞;  p :=nil;
   if raza intersectează volumul mărginit a rădăcinii then
      inserează rădăcina în coadă;
   while coada este nevidă şi distanţa la nodul top < t do
     begin
      c := nod top eliminat din coadă;
      if c este nod terminal then
        begin
         intersectează raza cu obiectul c;
         if raza se intersectează cu obiectul
              iar distanţa până la intersecţie < t then
          begin
            t :=distanţa;
            p := obiect
          end
        end
      else
         for fiecare fiu c do
             begin
                  intersectează raza cu volumul de mărginire;
                  if raza intersectează volumul then
                     inserează fiul în coadă
             end
     end
end;
```

**Partiţionare.** Ierarhia volumelor mărginite organizează obiectele de jos în sus, iar *partiţionarea* divide spaţiul de sus în jos. Se determină prima dată volumul care mărgineşte scena. Acest volum este divizat în subvolume egale, fiecărei partiţii fiindu-i asociată o listă de obiecte pe care le conţine parţial sau în întregime. Se vor căuta intersecţiile numai cu acele obiecte care se află în volumele prin care trece raza. In cazul intersecţiei unui corp cu raza este necesar testul de apartenenţă a punctului de intersecţie la partiţie. De exemplu, în figura 5.36, obiectul $B$ este intersectat este intersectat în partiţia 3 deşi este întâlnit în partiţia 2. Pentru a nu efectua intersecţiile unui corp cu raza de mai

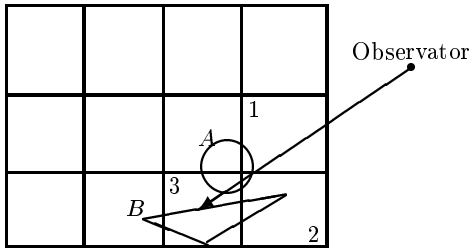



Figura 5.36: Divizarea volumului care mărgineşte scena

multe ori este necesară întreţinerea unei liste a intersecţiilor funcţie de obiect. Dacă obiectele nu sunt uniform distribuite în spaţiu este de preferat utilizarea unei subdivizări adaptive a scenei care produce partiţii inegale (de exemplu, utilizând arbori octali). Partiţionarea spaţială şi ierarhia pot fi foliste împreună pentru a combina avantajele (de exemplu, ierarhii cu noduri interne tip listă sau grile tridimensionale).

**Modelare prin `CSG`.** Dacă obiectele sunt modelate prin `CSG`, intersecţia fiecărei raze cu obiectele primitive conduce la o mulţime de valori $t$, fiecare specificând un punct în care raza intră sau iese din obiect. Astfel fiecare $t$ defineşte începutul unui interval în care raza este fie în interiorul, fie în exteriorul obiectului. Dacă obiectul este obţinut prin operaţii booleene din mai multe primitive, intervalele de intersecţie ale razei cu obiectul compus se pot obţine aplicând aceleaşi operaţii booleene asupra intervalelor de intersecţie cu primitivele din care este compus obiectul (figura 5.37). Fiecare obiect compus este descris printr-un arbore în care nodurile terminale sunt primitivele, iar nodurile intermediare, operaţiile booleene. Dacă nu există o intersecţie a subarborelui stâng cu raza, atunci obiectul nu se va intersecta cu raza în condiţiile în care operaţia din nodul rădăcină este o diferenţă sau o intersecţie.

Funcţia `CSG_combine`, descrisă mai jos în pseudocod, preia două liste de puncte de intersecţii şi le combină într-o unică listă care este ordonată în funcţie de $t$ crescător.

```
function CSG_intersect
 (var raza : ^Ray; var nod : ^CSG_node):  intervList
var
   leftIntersect, rightIntersect : intervList;
begin
   if nod este compus then
     begin
      leftIntersect :=CSG_intersect(raza, nod^.leftChild);
      if (leftIntersect =nil) and (nod^.op <> union) then
         CSG_intersect:=nil
      else
        begin
         rightIntersect :=CSG_intersect(raza, nod^.rightChild);
         CSG_intersect:=CSG_combine(nod^.op,
           leftIntersect, rightIntersect)
```

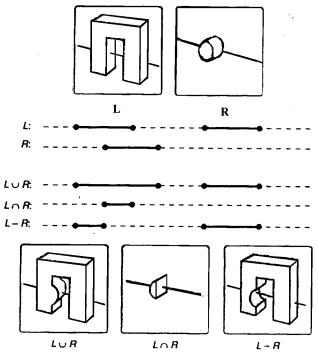



Figura 5.37: Combinarea intervalelor de intersecţie rază-obiect

```
        end
      end
    else
       CSG_intersect:=intersecţiile obiectului cu raza
end;
```

### Supra-eşantionarea

Fenomenul de `aliasing` spaţial poate apărea datorită rezoluţiei insuficiente a reţelei de eşantionare. Problema este rezolvabilă prin mărirea rezoluţiei reţelei doar în mod relativ: dacă se măreşte dimensiunea unui ochi al reţelei, deci şi a imaginii, efectul reapare.

Obiectele prea mici relative la dimensiunea unui ochi (al reţelei) pot adesea „scapă" reţelei de eşantionare (figura 5.38). Oricât de multe raze (oricât de dese) de eşantionare se folosesc, ar putea exista obiecte mai mici decât distanţa dintre raze. Pe de altă parte, s-ar putea afirma că, dacă un obiect este atât de mic încât nu poate fi depistat de razele de eşantionare, atunci nu are importanţă dacă apare în imagine sau nu.

O soluţie posibilă, simplă din punct de vedere conceptual, pentru problema de mai sus, constă în utilizarea mai multor raze care trec prin acelaşi pixel, culoarea finală fiind o medie a culorilor acestor raze. Supra-eşantionarea este

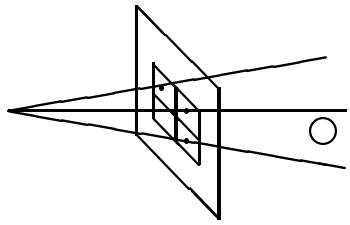


Figura 5.38: Obiect prea mic pentru a fi intersectat de razele de eşantionare

însă extrem de costisitoare: dacă factorul de multiplicare al razelor pentru un pixel este $n$, atunci timpul de trasare creşte de cel puţin $n$ ori. Pentru a evita ,,scăparea" unor obiecte printre razele ce trec prin poziţii fixe, se poate utiliza o distribuţie aleatoare a razelor (dar uniformă) peste regiunea corespunzătoare pixelului, prin *supra-eşantionare stohastică*.

*Algoritmul de supra-eşantionare adaptivă* porneşte de la o reprezentare prin cinci raze (figura 5.39). Dacă fiecare din razele de start au culori apropiate, se va utiliza media lor. Dacă razele de start diferă în culoare, atunci se subdivide pixelul în regiuni mai mici şi se repetă pasul pentru fiecare nouă regiune. Astfel, multiplicarea razelor nu se face pe regiuni de culoare constantă, ci *adaptiv*, acolo unde apar variaţii. Se consideră că, în figura 5.39 (a), razele care trec prin $A$, $D$ şi $E$ au culori similare, ceea ce nu poate fi afirmat şi despre $B$, $C$ şi $E$. Regiunea mărginită de $B$ şi $E$ este studiată mai în amănunţime. Se trasează noi raze prin $F$, $G$, $H$ (figura 5.39.b). Se compară cele cinci culori. Presupunem că razele ce trec prin $F$, $B$, $H$ şi $G$ au culori similare, mai puţin $E$ cu $G$. Regiunea mărginită de $G$ şi $E$ este studiată mai amănunţit. Se trasează trei noi raze, prin $J$, $K$ şi $L$ din figura 5.39.c. Presupunem că toate razele noii regiuni au culori similare. Se revine la regiunea iniţială, pentru a studia în amănunţime perechea de raze ce trec prin $C$ şi $E$. Se trasează două noi raze, prin $M$ şi $N$ din figura 5.39.d. Presupunem că razele specifice noii regiuni au culori similare, mai puţin cele care trec prin $C$ şi $M$. Regiunea mărginită de aceste două puncte este analizată în amănunţime (figura 5.39.e). Presupunem că în noua subregiune nu există variaţii semnificative de culoare. Culoarea finală este determinată printr-o medie ponderată:

$$\frac{1}{4}\left(\frac{A+E}{2}+\frac{D+E}{2}+\frac{1}{4}\left[\frac{F+G}{2}+\frac{B+G}{2}+\frac{H+G}{2}+\right.\right.$$

$$+\frac{1}{4}\left\{\frac{J+K}{2}+\frac{G+K}{2}+\frac{L+K}{2}+\frac{E+K}{2}\right\}\Bigg]+\frac{1}{4}\left[\frac{E+M}{2}+\right.$$

$$\left.\left.+\frac{H+M}{2}+\frac{N+M}{2}+\frac{1}{4}\left\{\frac{M+Q}{2}+\frac{P+Q}{2}+\frac{C+Q}{2}+\frac{R+Q}{2}\right\}\right]\right)$$

### Calcule de intersecţii

Partea principală a oricărui pachet de programe pentru `ray-tracing` ("inima") o constituie un set de rutine de intersecţie rază–obiect.

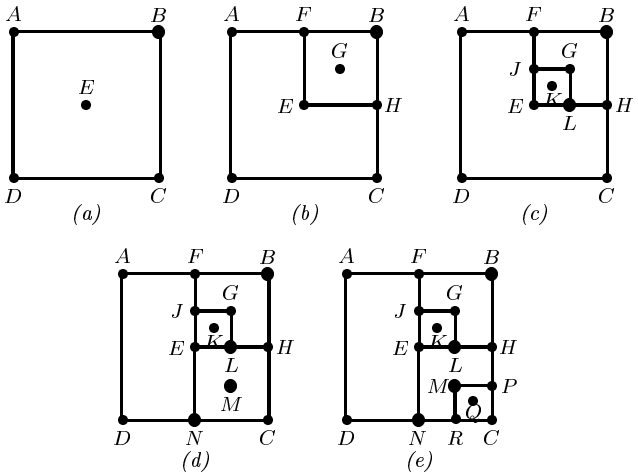


Figura 5.39: Paşi într-un algoritm de super-eşantionare adaptivă

**Intersecţia razei cu o sferă – soluţia algebrică.** Fie raza definită prin $R_0(x_0, y_0, z_0)$ şi direcţie $D(x_d, y_d, z_d)$, vector normalizat, astfel încât ecuaţia *parametrică* (explicită) a razei este $R(t) = R_0 + t \cdot D,\ t > 0$. Sfera se defineşte prin centru $C(x_c, y_c, z_c)$ şi rază $r$. Atunci ecuaţia intersecţiile corepund unui $t$ ce verifică ecuaţia:

$$(x_0 + tx_d - x_c)^2 + (y_0 + ty_d - y_c)^2 + (z_0 + tz_d - z_c)^2 = r^2.$$

Soluţiile ecuaţiei de grad doi sunt:

$$t_0 = \frac{-b - \sqrt{b^2 - 4c}}{2a}, \qquad t_1 = \frac{-b + \sqrt{b^2 - 4c}}{2a},$$

unde $a = x_d^2 + y_d^2 + z_d^2 = 1$, $b = 2[x_d(x_0 - x_c) + y_d(y_0 - y_c) + z_d(z_0 - z_c)]$, $c = (x_0 - x_c)^2 + (y_0 - y_c)^2 + (z_0 - z_c)^2 - r^2$. Algoritmul de intersecţie decurge astfel:

*Pas 1:* se calculează $a,\ b,\ c$ (8 adunări/scăderi şi 7 înmulţiri);

*Pas 2:* se calculează discriminantul ecuaţiei în $t$ (1 scădere, 2 înmulţiri şi 1 comparaţie);

*Pas 3:* se calculează $t_0$ şi se decide dacă soluţia căutată, adică $t_0 > 0$ (1 scădere, 1 înmulţire, 1 radical şi 1 comparaţie);

*Pas 4:* dacă $t_0 < 0$, atunci se calculează $t_1$ şi se decide dacă e soluţia căutată, adică $t_1 > 0$ (1 scădere, 1 înmulţire şi 1 comparaţie);

*Pas 5:* se calculează componentele punctului de intersecţie (3 adunări, 3 înmulţiri);

*Pas 6:* se calculează normala la suprafaţă în punctul de intersecţie (3 scăderi, 3 înmulţiri).

Pentru cazul cel mai defavorabil se efectuaează 17 adunări/scăderi, 17 înmulţiri, 1 radical, 3 comparaţii. Numărul de operaţii se poate reduce prin abordarea soluţiei geometrice.

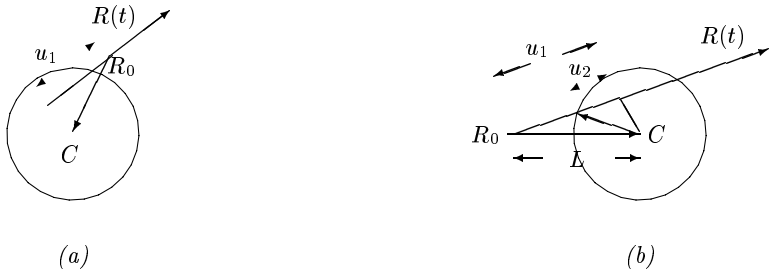



Figura 5.40: (a) O singură intersecţie (b) Calculul valorilor $u_i$

**Intersecţia razei cu o sferă – soluţia geometrică.** Fie $L = |C - R_0|$. Dacă $L^2 < r^2$, originea razei este în interiorul sferei, iar dacă $L^2 \geq r^2$, originea este pe sferă sau exterioară sferei, astfel încât raza poate să nu intersecteze sfera. Problema determinării distanţei minime de la rază la centrul sferei este echivalentă cu a găsi intersecţia razei cu planul perpendicular pe rază şi care trece prin $C$. Se notează cu $u_1$ produsul scalar $t_1 = (C - R_0) \cdot D$. Dacă $u_1 < 0$, atunci centrul sferei este "în spatele" lui $R_0$ (figura 5.40.a). Distanţa minimă este $L^2 - u_1^2$. Se determină cantitatea $u_2 = r^2 - L^2 + u_1^2$. Dacă $u_2 < 0$, raza nu intersectează sfera (figura 5.40.b). Altfel, valoarea parametrului $t$ corespunzător intersecţiei este

$$t = \begin{cases} u_1 - \sqrt{u_2}, & \text{rază cu origine exterioară sferei,} \\ u_1 + \sqrt{u_2}, & \text{rază cu origine interioară sferei.} \end{cases}$$

Algoritmul de intersecţie bazat pe aceste calcule constă în următorii paşi:
*Pas 1:* determină $L^2$ (5 adunări/scăderi, 3 înmulţiri);
*Pas 2:* calculează $u_1$ (2 adunări, 3 înmulţiri);
*Pas 3:* test de exterioritate şi îndepărtare: $u_1 < 0$ şi $|L| \geq r$ (2 comparaţii);
*Pas 4:* determină $u_2$ (2 adunări/scăderi, 1 înmulţire);
*Pas 5:* test dacă $u_2 < 0$ (1 comparaţie);
*Pas 6:* calculează $t$ corespunzător intersecţiei (1 adunare/scădere, 1 radical);
*Pas 7:* calculează coordonatele punctului de intersecţie (3 adunări, 3 înmulţiri);
*Pas 8:* calculează normala în punctul de intersecţie (3 adunări, 3 înmulţiri).
In cel mai defavorabil caz se efectuează 16 adunări/scăderi, 13 înmulţiri, 1 radical, 3 comparaţii. Se observă reducerea numărului de operaţii faţă de soluţia algebrică, fapt ce permite o procesare mai rapidă.

**Intersecţia razei cu un plan.** Fie vectorul unitar normal la plan $P_n(a, b, c)$, iar $d$, distanţa de la originea sistemului de coordonate la plan. Din ecuaţia planului $P_n \cdot (R_0 + tD) + d = 0$ rezultă $t$-ul intersecţiei

$$t = -\frac{P_n \cdot R_0 + d}{P_n \cdot D}.$$

Numitorul expresiei, $v_d = P_n \cdot D$, este o valoare decisivă pentru intersecţie. Dacă $v_d = 0$, atunci raza este paralelă cu planul (deci nu intersectează planul).

Dacă $v_d > 0$, normala la plan se "îndepărtează" de rază şi, dacă planele sunt considerate cu o singură faţă (exterior), atunci algoritmul se termină. Altfel, se calculează numărătorul expresiei lui $t$, $v_0 = -(P_n \cdot R_0 + d)$ şi apoi $t = v_0/v_d$. Dacă $t < 0$, atunci linia definită de rază intersectează planul în spatele originii razei şi, deci, nu există intersecţia căutată. In mod obişnuit, normala la plan de care este nevoie în algoritm este cea aflată de aceeaşi parte a planului ca şi raza. De aceea este necesară ajustarea semnului normalei $P_n$ în funcţie de relaţia sa cu vectorul de direcţie $D$ al razei. Astfel, $P_n$ este înlocuit în ecuaţii cu $-P_n$ dacă $v_d < 0$. Algoritmul de intersecţie constă în următorii paşi:

*Pas 1:* se calculează $v_d$ şi se compară cu 0 (2 adunări, 3 înmulţiri, 1 comparare);

*Pas 2:* se calculează $v_0$ şi $t$ şi se compară $t$ cu 0 (3 adunări, 3 înmulţiri, 1 comparare);

*Pas 3:* se calculează coordonatele punctului de intersecţie (3 adunări, 3 înmulţiri);

In total, în cel mai defavorabil caz, se efectuează 8 adunări/scăderi, 9 înmulţiri, 2 comparări.

**Intersecţia cu un paralelipiped.** Acest tip de intersecţie este util atât în cazul obiectelor paralelipipedice, cât şi în cazul când obiecte complicate (complexe) sunt încadrate într-un paralelipiped (volume de mărginire) cu care se testează intersecţia şi, în caz afirmativ, se încearcă determinarea intersecţiei cu obiectul în cauză. Ideea de bază constă în a considera un paralelipiped ca volumul obţinut prin intersecţia a trei perechi de plane paralele. Raza intersectează fiecare pereche de plane paralele într-un punct mai apropiat şi într-un punct mai îndepărtat. Dacă cea mai mare dintre valorile apropiate este mai mare decât cea mai mică dintre valorile îndepărtate, atunci raza nu intersectează paralelipipedul (altfel, îl intersectează).

Fie un paralelipiped cu laturi paralele cu axele de coordonate, definit prin: punctul cel mai apropiat de origine, $B_1(x_1, y_1, z_1)$ şi punctul cel mai îndepărtat de origine, $B_h(x_h, y_h, z_h)$.

Algoritmul de test al intersecţiei este următorul:

1. fie $t_{min} = -\infty$, $t_{max} = \infty$ (arbitrar de mari);
2. pentru fiecare pereche de plane paralele delimitatoare ale paralelipipedului (paralele cu axele $x$, $y$, $z$) se efectuează următoarele (exemplificare pentru cazul planelor paralele cu axa $x$):
   (a) dacă $X_d = 0$, raza este paralelă cu planele şi dacă $X_0$ nu se află între planele parelele ($X_0 < X_1$ sau $X_0 > X_h$), atunci nu există intersecţie;
   (b) altfel, raza nu este paralelă cu planele şi
      - calculează distanţele până la intersecţiile cu planele: $t_1 = (X_1 - X_0)/X_d$, $t_2 = (X_h - X_0)/X_d$;
      - dacă $t_1 > t_2$, interschimbă $t_1$ cu $t_2$;
      - dacă $t_1 > t_{min}$, atunci $t_1 = t_{min}$;
      - dacă $t_2 < t_{max}$, atunci $t_2 = t_{max}$;
      - dacă $t_{min} > t_{max}$, atunci paralelipipedul nu este intersectat;
      - dacă $t_{max} < 0$, atunci paralelipipedul este în spatele razei;
   (c) continuă cu alte perechi de plane

3. dacă testele de respingere de mai sus nu au succes, raza intersectează paralelipipedul în două puncte corespunzătoare lui $t_{min}$ şi $t_{max}$.

### 5.2.6 Comparaţii între algoritmii de vizibilitate

Algoritmii de vizibilitate pot fi clasificaţi după ordinea în care sortează informaţia. Algoritmul de sortare în adâncime efectuează o sortare după $z$, apoi după $x$ şi $y$ (testele 1 şi 2); astfel este numit un algoritm $zxy$. Algoritmul liniei de baleiaj sortează după $y$ (o sortare în grup), apoi după $x$ şi în final se face o căutare în $z$ a poligoanelor celor mai apropiate de observator; este un algoritm $yxz$. Algoritmul `Warnock` realizează consecutiv sortarea în $x$ şi $y$, apoi se face o căutare după $z$; este un algoritm $(xy)z$. Algoritmul $z$-`buffer` nu realizează o sortare explicită după o anumită direcţie; este un algoritm de tip $(xyz)$. Deoarece obiectele sunt, în general, egale în complexitatea în cele trei dimensiuni, algoritmii nu se pot ierarhiza, ca eficienţă, pe baza sortării.

Algoritmii diferă de asemenea în modalitatea de utilizare a coerenţelor (de exemplu, algoritmul liniei de baleiaj utilizează coerenţa liniilor de baleiaj).

Algoritmul de sortare în adâncime este eficient în cazul unui număr mic de poligoane (cu creşterea numărului de poligoane, testele de suprapunere sunt mai frecvente şi subdivizarea poligoanelor este mai des cerută). Pe de altă parte, algoritmul $z$-`buffer` are o performanţă constantă relativ la creşterea numărului de poligoane din scenă, deoarece numărul de pixeli acoperiţi de un singur poligon descreşte. Testele şi calculele din metoda `Warnock` sunt relativ complexe, astfel încât în general implementările acestui algoritm sunt mai lente decât cele corespunzătoare metodelor de mai sus.

Alegerea unui algoritm de vizibilitate este de asemenea influenţată de condiţiile de vizualizare. Dacă se va aplica o procedură complicată de umbrire este de preferat un algoritm care trasează doar părţi ale obiectelor (precum algoritmul liniei de baleiaj); în acest caz alegerea algoritmului de sortare în adâncime nu este prea fericită, deoarece obiectele se trasează în întregime. Dacă sistemul de vizualizare presupune interactivitate, atunci se preferă algoritmul $z$-`buffer`. Algoritmul arborelui `BSP`, pe de altă parte, poate genera foarte rapid imagini ale mediului static, dar necesită procesări adiţionale dacă mediul se schimbă. Acomodarea la noi primitive este un avantaj al algoritmului liniei de baleiaj.

## 5.3 Iluminare şi umbre

Pentru ca imaginea să fie cât mai realistă este utilă simularea iluminării obiectului precizând:

1. *sistemul de surse luminoase* cu care se face iluminarea (tipul surselor, poziţia acestora în sistemul de referinţă adoptat),
2. *caracteristicile optice ale suprafeţelor* – există astfel:
   (a) zone mate, care dispersează lumina reflectând-o în mod egal în toate direcţiile (de exemplu, o bucată de cretă);
   (b) zone lucitoare, care reflectă lumina numai după anumite direcţii relative la observator şi sursa de lumină (de exemplu, o oglindă);

(c) zone transparente sau translucide cu fenomene de refracţie şi atenuare;

3. *poziţia relativă a suprafeţelor* corpului şi a sistemului de surse luminoase.

Procesele de eliminare a suprafeţelor ascunse şi simulare a iluminării se pot desfăşura simultan.

### 5.3.1 Surse de lumină

Sursele de lumină pot fi *punctiforme* (punctuale) sau *distribuite*. Exemple de surse punctiforme sunt becurile cu incandescenţă, flăcările de dimensiuni mici. Ca exemplu de sursă distribuită se poate menţiona tubul fluorescent. Sursele punctiforme produc un efect realist deoarece iluminarea unei suprafeţe depide de orientarea ei, mai precis de cosinusul unghiului dintre normala la suprafaţă în fiecare punct considerat pe aceasta şi dreapta care uneşte punctul respectiv cu sursa. Iluminarea este mai puternică pentru suprafeţe orientate perpendicular faţă de razele de lumină provenite de la sursă şi scade odată cu înclinarea suprafeţelor faţă de direcţia luminii.

Pe lângă aceste tipuri de surse se pot întâlni situaţii în care obiectele sunt scăldate într-o lumină de intensitate în general moderată şi care pare să vină din toate părţile - aceasta poartă numele de *lumină ambiantă* (produs al reflexiilor multiple de la suprafeţele aflate în scenă). Este cea mai uşor de modelat deoarece iluminarea suprafeţelor este constantă şi independentă de orientarea lor. Imaginea formată depinde astfel numai de intensitatea luminii ambiante şi de proprietăţile suprafeţei corpului modelat. Acest tip de iluminare nu produce o imagine realistă (muchiile de joncţiune ale poligoanelor cu aceleaşi proprietăţi superficiale nu se pot distinge). In realitate este puţin probabil să se întâlnească situaţii în care un anumit corp să se afle numai în lumina ambiantă.

Dacă sursa se află la o distanţă suficient de mare de obiectul iluminat pentru ca razele incidente să fie considerate paralele între ele, iluminarea este *paralelă* sau *direcţională*. Prin acest procedeu poate fi modelată mai bine lumina solară. Pentru precizarea unei surse de lumină paralelă sunt suficiente intensitatea iniţială, direcţia şi sensul de iluminare. Dacă în cazul iluminării cu surse punctiforme, distanţa la sursă este o informaţie importantă, în cazul iluminării paralele influenţa ei devine nesemnificativă, intensitatea fiind practic invariantă cu distanţa de la sursă.

Variaţiile intensităţii luminoase pe suprafaţa unui obiect oferă informaţii importante cu privire la forma suprafeţei respective. Informaţii suplimentare pot fi obţinute prin analiza umbrelor pe care elemente ale corpului modelat le aruncă asupra unui fundal al imaginii sau asupra altor elemente ale aceluiaşi corp.

### 5.3.2 Modele de iluminare

Ecuaţia iluminării variază în funcţie de tipul sursei luminoaase.

**Lumina ambiantă.** *Lumina mediului înconjurător (lumina ambiantă)* se răsfrânge egal pe toate suprafeţele şi în toate direcţiile, astfel încât ecuaţia

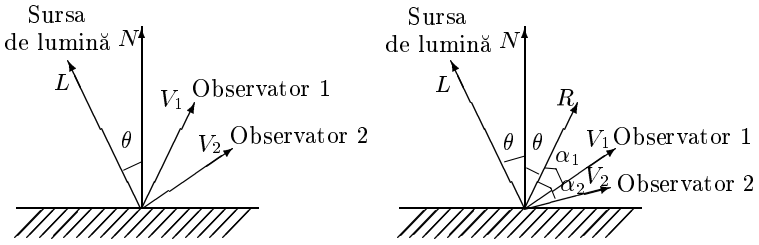


Figura 5.41: Iluminare (a) reflexie difuză (b) reflexie specular\u0103

iluminării este

$$I = I_a k_a$$

unde $I$ este intensitatea rezultată, $I_a$ este intensitatea luminii ambiante, constantă pentru fiecare obiect, iar $k_a$ este *coeficientul de reflexie ambiantă*, procentul în care suprafaţa reflectă lumina ambiantă (între 0 şi 1), caracteristică a materialului.

**Relexie difuză (lambertiană).** Se consideră obiectul luminat de o sursă punctuală de lumină a cărui raze emană uniform în toate direcţiile ce pornesc de la un singur punct. Suprafeţele mate au proprietatea de a împrăştia lumina incidentă, prin reflecţie difuză, în mod aproximativ egal după toate direcţiile posibile. Oriunde s-ar afla observatorul, o aceeaşi placă va avea o imagine de strălucire constantă. Intensitatea luminii reflectate de o anumită faţetă din reprezentarea poliedrală a unui corp este independentă în acest caz de poziţia observatorului. Ea depinde numai de unghiul $\theta$ dintre direcţia $L$ de la sursa de lumină şi normala la suprafaţă $N$ (figura 5.41.a). Faţetele luminate după o direcţie mai apropiată de normală vor reflecta lumina cu intensitate mai mare.

Intensitatea luminii reflectate se exprimă după *legea reflexiei cosinus*:

$$I = I_p k_d \cos\theta,$$

unde $I$ este intensitatea luminii reflectate, $I_p$ este intensitatea sursei de lumină punctuală, iar $k_d$ este un coeficient care depinde de natura şi proprietăţile optice ale materialului ce produce reflexia difuză, numit *coeficient de reflexie difuză* (între 0 şi 1). Dacă $L$ şi $N$ sunt versorul direcţiei de propagare a luminii (raza incidentă) şi, respectiv, versorul normalei la faţetă, atunci are loc egalitatea $I = I_p k_d (L \cdot N)$. Această relaţie nu ţine seama de variaţia intensităţii luminoase în raport cu distanţa la sursă. Astfel, proiecţiile a două suprafeţe paralele de material identic se pot suprapune într-o imagine uniformă. Pentru o sursă punctiformă ideală, intensitatea luminoasă este invers proporţională cu pătratul distanţei $R$ dintre sursă şi punctul în care se măsoară această intensitate. In majoritatea situaţiilor reale, obiectele observate sunt scăldate şi în lumină ambiantă, astfel încât ecuaţia iluminării este:

$$I = I_a k_a + (1/R^2) I_p k_d \cos\theta.$$

Sunt situaţii în care observatorul se află faţă de sursă la o distanţă mult mai mică decât sursa luminoasă, ceea ce face ca influenţa lui $R^2$ să fie exagerată şi imaginea nerealistă. Astfel, reprezentările cele mai realiste se obţin folosind formula

$$I = I_a k_a + I_p k_d \cos\theta/(d + D),$$

unde $d$ este o cotă care dă depărtarea relativă a faţetei considerate de sursă (pentru cea mai apropiată faţetă faţă de sursă $d$ se poate lua 0). $D$ este termenul de atenuare şi este un număr suficient de mare, astfel ales încât variaţia lui $d$ pe domeniul său să provoace variaţia lui $I$ în limitele dorite. Mai general,

$$I = I_a k_a + f_{at} I_p k_d \cos\theta,$$

unde $f_{at}$ este *factorul de atenuare*. Lumina colorată este tratată scriind ecuaţiile separat pentru fiecare componentă a modelului de culoare. Cele trei componente primare ale luminii $I_{pR}$, $I_{pG}$, $I_{pB}$ sunt reflectate în proporţiile $k_d O_{dR}$, $k_d O_{dG}$, $k_d O_{dB}$. Tripletul $(O_{dR}, O_{dG}, O_{dB})$ defineşte componentele culorii obiectului în sistemul RGB. $O_d$ este culoarea difuză a obiectului. De exemplu, pentru componenta roşie ecuaţia iluminării este

$$I_R = I_{aR} k_a O_{dR} + f_{at} I_{pR} k_d O_{dR} \cos\theta.$$

**Reflexia speculară (selectivă).** Este asociată cu suprafeţele lucioase. Se consideră, de exemplu, un glob roşu metalizat din pomul de iarnă, sau un măr lustruit, iluminat cu o sursă punctiformă. Pe suprafaţa acestora se pot distinge zone de roşu închis, care reflectă lumina mai puţin intens, zone care reflectă lumina cu un roşu intens, precum şi o mică zonă care apare albă şi pare o sursă de lumină de aceeaşi natură cu lumina incidentă aflată pe suprafaţa obiectului. La schimbarea poziţiei observatorului pata de lumină îşi schimbă şi ea poziţia pe suprafaţa corpului. Această pată este produsul reflexiei selective sau speculare, în timp ce intensităţile provenind de la celelelte puncte de pe suprafaţa corpului sunt rezultatul reflexiei difuze.

Reflexia selectivă se produce deoarece suprafeţele netede şi lustruite reflectă lumina în mod inegal. O suprafaţă perfect lucioasă (o oglindă perfectă) reflectă lumina numai după direcţia din planul de incidenţă pentru care unghiul de reflexie este egal cu cel de incidenţă (în direcţia $R$, simetrica lui $L$ faţă de $N$ din figura 5.41.b). Se consideră $\alpha$ unghiul dintre direcţia de observare $V$ şi direcţia de reflexie speculară $R$.

Suprafeţele pot fi şi oglinzi imperfecte (de exemplu, obiectele din vinilin). Pentru astfel de suprafeţe intensitatea reflexiei speculare scade brusc odată cu creşterea diferenţei dintre unghiul de reflexie real şi unghiul de incidenţă. Modelul empiric de simulare a reflexiei speculare presupune descreşterea intensităţii luminoase proporţional cu $\cos^m |\alpha|$ unde $m$ este un număr natural nenul, care depinde de natura suprafeţei iluminate, exponentul de reflexie speculară al materialului (de la 1 la ordinul sutelor). Pentru o oglindă perfectă, $m \to \infty$, astfel încât intensitatea reflectată este maximă pentru $\alpha = 0$. Se introduce noţiunea de factor de reflexie speculară, care este o funcţie de unghiul de incidenţă $\theta$, $S : [0, 90^o] \to [0, 1]$ şi care arată ce fracţie din intensitatea luminoasă incidentă este reflectată specular în cazul unei incidenţe $\theta$. Ecuaţia iluminării este

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{at} I_{p\lambda} [k_d O_{d\lambda} \cos\theta + S(\theta) \cos^m |\alpha|],$$

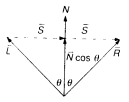



Figura 5.42: Calcularea vectorului de reflexie

unde $\lambda = R, G, B$. In modelul de iluminare `Phong`, $S(\theta)$ este setat pe $k_s O_{s\lambda}$, unde $k_s$ este coeficientul de reflexie specular cu valori între 0 şi 1, iar $O_{s\lambda}$ este culoarea specular a obiectului. Ecuaţia iluminării este, în acest caz,

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{at} I_{p\lambda} [k_d O_{d\lambda} (N \cdot L) + k_s O_{s\lambda} (R \cdot V)^n].$$

Vectorul $R$ se calculează pe baza lui $L$ şi $N$ (figura 5.42): $R = N \cos\theta + S$, iar $S = N \cos\theta - L$, deci $R = 2N(N \cdot L) - L$, iar $R \cdot V = (2N(N \cdot L) - L) \cdot V)$. Dacă sursa de lumină este la infinit, $N \cdot L$ este constant pentru un poligon dat, pe când $R \cdot V$ variază de-a lungul poligonului. Pentru suprafeţe curbe sau pentru o sursă de lumină la o poziţie finită, ambele produse $N \cdot L$ şi $R \cdot L$ variază de-a lungul suprafeţei.

Dacă există $m$ surse de lumină, atunci

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_{i=1}^{m} f_{at_i} I_{p\lambda_i} [k_d O_{d\lambda} (N \cdot L_i) + k_s O_{s\lambda} (R_i \cdot V)^n].$$

**Transparenţă.** Modelele simple de transparenţă nu includ refracţia (amestecarea) luminii printr-un solid transparent. Pentru simularea *transparenţei nerefractate* se utilizează două metode:

1. *transparenţă interpolată*: dacă poligonul transparent 1 se intercalează între observator şi poligonul opac 2, ecuaţia iluminării este

   $$I_\lambda = (1 - k_{t1}) I_{\lambda 1} + k_{t1} I_{\lambda 2},$$

   unde $k_{t1}$ este coeficientul de trasparenţă al poligonului 1 (între 0 şi 1). Dacă $k_{t1} = 0$, poligonul este opac şi nu transmite lumină. Dacă $k_{t1} = 1$, poligonul este perfect transparent. Valoarea $1 - k_{t1}$ desemnează opacitatea poligonului;
2. *transparenţă filtrată*: poligonul 1 este un filtru transparent, astfel încât

   $$I_\lambda = I_{\lambda 1} + k_{t1} O_{t\lambda} I_{\lambda 2},$$

   unde $O_{t\lambda}$ este transparenţa culorii $\lambda$ a poligonului 1.

Modele complexe includ refracţia, transluciditatea difuză, atenuarea luminii cu distanţa. Transparenţa cu refracţie este un fenomen des întâlnit, dar greu de modelat (figura 5.43). Relaţia dintre unghiul de incidenţă $\theta_i$ şi unghiul de refracţie $\theta_t$ este dată de

$$\frac{\sin\theta_i}{\sin\theta_t} = \frac{\eta_{t\lambda}}{\eta_{i\lambda}}$$

unde $\eta_{i\lambda}$ şi $\eta_{t\lambda}$ sunt indicii de refracţie a materialelor prin care trece lumina.

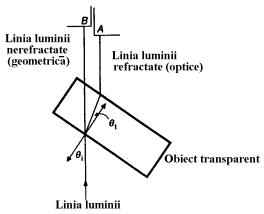



Figura 5.43: Refracţie

### 5.3.3 Iluminarea reprezentărilor poliedrale

Metodele de modelare a iluminării reprezentărilor poliedrale cele mai des utilizate sunt următoarele:

1. metoda iluminării constante;
2. metoda interpolării intensităţii luminoase;
3. metoda interpolării normalei la suprafaţă.

**Metoda iluminării constante.** Această metodă presupune calcularea intensităţii o singură dată pentru fiecare poligon, deci unui poligon îi este caracteristică în condiţiile precizate de iluminare şi observare o anumită intensitate, constantă pe toată suprafaţa sa. Există situaţii pentru care calculele de iluminare se pot face cu viteză foarte mare şi anume cele în care direcţiile normalelor la faţetele corpului sunt în număr mic, faţetele fiind în număr relativ mare, dar grupate în *clase de faţete paralele* (situaţie des întâlnită în scene arhitecturale). Pentru fiecare direcţie din scenă se calculează intensitatea corespunzătoare. Faţetele se sortează în clase după direcţiile normalelor. Dacă nu se ţine cont de variaţia iluminării cu distanţa la sursă, atunci fiecare clasă se transpune pe ecran cu intensitatea calculată pentru direcţia reprezentativă clasei respective. Dacă se ţine seama de variaţia iluminării cu distanţa, atunci faţetele din fiecare clasă se sortează în subclase după distanţa la sursă, intensitatea corespunzătoare fiecărei subclase calculându-se prin incrementare.

Pentru ca o situaţie de observare reală să dea o imagine de tip iluminare constantă, e nevoie să fie îndeplinite trei condiţii:

(a) suprafaţa poliedrală este chiar suprafaţa corpului reprezentat şi nu o aproximare a acestuia;
(b) observatorul este la infinit astfel încât $\alpha$ să fie constant pentru fiecare suprafaţă poligonală;
(c) sursa de lumină se află la infinit, ($\theta$ constant pentru fiecare suprafaţă poligonală).

Datorită acestor condiţii, mai ales a doua, situaţia este rar întâlnită în practică.

Totuşi, datorită simplităţii de calcul, metoda este cea mai folosită. Calculule pentru fiecare poligon se fac într-un singur punct interior privilegiat al acestuia, de obicei centrul de greutate.

Principalul dezavantaj al iluminării constante este acela că muchiile faţetelor poligonale sunt accentuate artificial datorită efectului de bandă `Mach`. Astfel, dacă faţeta $A$ este luminată, iar faţeta $B$ umbrită, zona de muchie a faţetei $A$ va apărea puternic luminată mărind contrastul cu faţeta $B$ (deci scoţând muchia în evidenţă), iar zona de frontieră a faţetei $B$ apare mai umbrită. In cazul imaginilor monocrome, când umbra şi lumina sunt simulate prin densitatea de puncte, efectul este mai puţin sesizabil.

**Metoda interpolării intensităţii luminoase (algoritmul `Gouraud`)** . Dă rezultate calitativ superioare, eliminând discontinuităţile de intensitate. Interpolarea vectorului de intensitate se face astfel:

(a) se calculează normalele la fiecare poligon;

(b) se calculează vectorii normali medii în fiecare vârf al poliedrului, adică se calculează în fiecare vârf media versorilor direcţiilor normale la toate faţetele alăturate vârfului (figura 5.44);

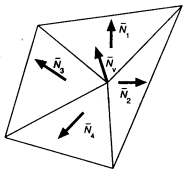



Figura 5.44: Media normalele la suprafeţe constituie normala în vârf

(c) se calculează în fiecare vârf intensităţile locale, folosind normalele medii anterior calculate;

(d) pentru fiecare linie de baleiaj şi pentru fiecare poligon se parcurg etapele următoare:
- se calculează intensităţile locale în intersecţiile liniei de baleiaj cu laturile poligonului (extremităţile segmentului de linie situat în poligon) prin interpolare liniară între valorile corespunzătoare vârfurilor ce mărginesc laturile respective;
- pentru fiecare pixel de pe segmentul de linie de baleiaj inclus în poligon se calculează intensitatea corespunzătoare prin interpolare liniară între valorile calculate la extremităţile segmentului, valorile obţinute transpunându-se pe ecran.

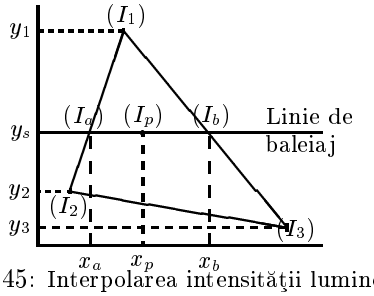


Figura 5.45: Interpolarea intensităţii luminoase

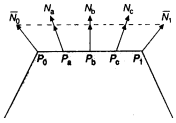


Figura 5.46: Interpolarea normalelor

De exemplu, pentru triunghiul din figura 5.45 intensităţile punctelor de pe linia de baleiaj sunt:

$$I_a = I_1 - (I_1 - I_2)\frac{y_1 - y_p}{y_1 - y_2},\ I_b = I_1 - (I_1 - I_3)\frac{y_1 - y_p}{y_1 - y_3},\ I_p = I_b - (I_b - I_a)\frac{x_b - x_p}{x_b - x_a}.$$

Se poate utiliza pentru creşterea eficienţei algoritmului ecuaţii cu diferenţe pentru diferite intensităţi de pe linii de scanare succesive.

**Metoda interpolării normalei la suprafaţă (algoritmul `Phong`).** Cele mai bune rezultate se obţin prin această metodă. După ce se calculează normalele medii locale în toate punctele care modelează corpul, se calculează, pentru fiecare segment de linie de baleiaj conţinut într-o faţetă, normalele în extremităţi prin interpolare între normalele medii locale calculate în capetele laturii pe care se află fiecare extremitate (figura 5.46). Apoi se interpolează normala într-un pixel de pe linia de baleiaj, între normalele calculate în extremităţi (relaţiile sunt analoage celor din metoda descrisă anterior, intensităţile fiind înlocuite cu vectori). Calculele sunt complicate căci normalele calculate sunt mărimi vectoriale şi deci interpolarea se face după cele 3 componente ale vectorului considerat în sistemul de referinţă $xyz$. Intensitatea este calculată numai după determinarea în fiecare pixel a normalei locale. Metoda nu este indicată atunci când `display`-ul este monocrom (când este necesară simularea intensităţii prin densitate de puncte).

Interpolarea în algoritmii de iluminare se face după aplicarea transformării perspective, în coordonate 3D ale ecranului şi nu în sistemul de coordonate a lumii înconjurătoare.

### 5.3.4 Umbre

Dacă iluminarea nu este frontală, atunci devin importante poziţiile surselor punctiforme şi direcţiile după care se face iluminarea paralelă. Există posibilitatea ca părţi vizibile ale corpului să nu fie iluminate, fiind umbrite de alte părţi. Pentru reprezentarea corectă a umbrelor în cazul iluminării cu surse punctiforme, calculele de vizibilitate pentru corpul considerat se fac de două ori: din poziţia observatorului şi din poziţia sursei. Pentru fiecare punct al unei suprafeţe se urmăreşte:

(a) dacă este văzut atât de sursă cât şi de observator, atunci pixelul este vizibil şi luminat de sursă;
(b) dacă nu se vede din poziţia observatorului, atunci pixelul este invizibil;
(c) dacă se vede din poziţia observatorului, dar nu se vede din poziţia sursei, atunci este vizibil, dar umbrit.

Dacă un punct al unei suprafeţe nu poate fi văzut de sursa de lumină, calculele de iluminare trebuie să fie ajustate pentru ca să ţină seama de acest fapt. Astfel ecuaţia iluminării devine:

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_{i=1}^{m} s_i f_{at_i} I_{p\lambda_i} [k_d O_{d\lambda} (N \cdot L_i) + k_s O_{s\lambda} (R_i \cdot V)^n].$$

unde

$$s = \begin{cases} 0 & \text{dacă lumina } i \text{ este blocată în acest punct} \\ 1 & \text{dacă lumina } i \text{ nu este blocată în acest punct} \end{cases}.$$

Algoritmii pentru determinarea umbrelor, respectiv a vizibilităţii, sunt de fapt aceiaşi.

Cele mai des utilizate metode sunt următoarele:

1. generarea la conversia scan a umbrelor;
2. utilizarea algoritmilor de vizibilitate cu două parcurgeri.

**Generarea scan a umbrelor.** Pentru o reprezentare poliedrală, utilizând sursa de lumină drept centru al proiecţiei, laturile poligoanelor care potenţial produc umbre sunt proiectate pe poligoanele care intersectează linia curentă de baleiaj. Când linia de baleiaj intersectează una din laturile acestor poligoane de umbră, culoarea imaginii este modificată (figura 5.47). O implementare brută a acestui algoritm presupune calculul proiecţiilor fiecărui poligon pe oricare alt poligon. Se poate introduce o fază de preprocesare în care toate poligoanele reprezentării sunt proiectate pe o sferă centrată în sursa de lumină. Perechile de proiecţii ale căror extensii nu se suprapun pot fi eliminate.

**Utilizarea algoritmilor spaţiu-obiect cu două parcurgeri.** Se determină în primul rând suprafeţele vizibile din punctul de vedere al sursei de lumină, ieşirea fiind o listă de poligoane luminate, fiecare asociat cu numărul de identificare a poligonului din care provine. Se aplică apoi algoritmul din punctul de vedere al observatorului, utilizând o copie a bazei de date anterior create, rezultând în final o nouă listă de poligoane. Algoritmul `Weiler-Atherton` poate fi utilizat cu succes.

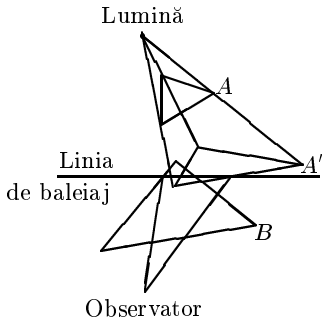


Figura 5.47: Poligonul $A$ produce umbra $A'$ în planul poligonului $B$

**Utilizarea algoritmilor spaţiu-imagine cu două parcurgeri.** Se consideră, de exemplu, algoritmul $z$-`buffer`. Algoritmul porneşte cu calcularea şi stocarea $z$-`buffer`-ului pentru imaginea văzută din punctul de vedere al sursei de lumină. Apoi $z$-`buffer`-ul şi imaginea sunt calculate din punctul de vedere al observatorului utilizând un algoritm $z$-`buffer` cu următoarele modificări: dacă un pixel este clasificat ca fiind vizibil, coordonatele sale în sistemul observatorului $(x_0, y_0, z_0)$ sunt transformate în coordonate în sistemul sursei de luminii $(x'_0, y'_0, z'_0)$. Coordonatele $x'_0$ şi $y'_0$ sunt utilizate pentru a selecta valoarea $z_L$ din $z$-`buffer`-ul sursei de lumină cu scopul de a fi comparată cu valoarea transformată $z'_0$. Dacă $z_L$ este mai apropiat de lumină decât $z'_0$, atunci există ceva care blochează lumina de la punct, iar pixelul este umbrit; altfel punctul este vizibil de către sursa de lumină. $z$-`buffer`-ul sursei de lumină poate fi privit ca un `buffer` de umbrire. Algoritmul poate fi uşor adaptat la cazul surselor de lumină multiple.

### 5.3.5 Implementarea transparenţei

Adăugarea efectelor de transparenţă în momentul în care se utilizează un algoritm de tip $z$-`buffer` este destul de dificilă, deoarece poligoanele sunt transpuse în ordinea în care sunt întâlnite. $z$-`buffer`-ul nu conţine informaţia care poligoane transparent se află în faţa unor poligoanelor opace, şi informaţia relativă la ordinea poligoanelor. Pentru a rezolva această problemă se propune următorul algoritm (a lui `Mannen`). In primul rând sunt procesate poligoanele opace cu un $z$-`buffer` convenţional. Apoi obiectele transparente sunt procesate într-o mulţime separată de `buffer`e care conţin, pentru fiecare pixel, o valoare a transparenţei şi un bit special, în completare la culoarea pixelului şi valoarea $z$. Bitul este iniţializat pe "off" şi fiecare valoare $z$ este setată pe o valoare corespunzătoare poziţiei celei mai apropiat posibilă. Dacă valoarea $z$ a obiectului transparent este mai apropiată decât valoarea $z$-`buffer`-ului opac, dar este mai departe decât valoarea din $z$-`buffer`-ul de transparenţă, atunci culoarea, valoarea $z$, şi transparenţa sunt salvate în `buffer`ele de transparenţă, iar bitul este setat pe "on". După ce toate obiectele au fost procesate, `buffer`ele obiectelor transparente conţin informaţia celui mai depărtat obiect transparent pentru fiecare pixel a cărui bit este setat. Informaţia pentru pixelii cu bitul setat este amestecată cu cea a `frame-buffer`-ului şi $z$-`buffer`-ul original. Valoarea $z$ de

transparenţă a pixelului cu bitul setat va înlocui valoarea $z$ din $z$-buffer-ul opac şi bitul este resetat. Acest proces se repetă pentru a transpune succesiv obiecte mai apropiate pe fiecare pixel.

### 5.3.6 Metoda recursivă a drumului optic

Pentru a determina culoarea unui pixel, trebuie constituită o listă a tuturor razelor de lumină care pleacă din punctul de pe primul obiect intersectat de dreapta de la ochi la pixel, culoarea pixelului fiind o combinaţie a culorilor acestor raze. De exemplu, o rază de lumină roşie şi o rază verde care cad amândouă într-un punct al unui obiect, pot constitui împreună o singură rază galbenă ce pleacă din respectivul punct înspre ochi.

Razele de lumină pot fi clasificate astfel:

(a) raze directe, care duc lumina prin pixel la ochi;
(b) raze de iluminare sau umbre, care duc lumina de la sursă la un obiect;
(c) raze reflectate de obiecte;
(d) raze de transparenţă, care trec printr-un obiect transparent.

Culoarea luminii radiate de un punct de pe suprafaţa unui obiect (într-o anumită direcţie) este, în esenţă, funcţie de combinaţia luminii ce provine de la:

1. surse de lumină,
2. alte obiecte ce reflectă lumina,
3. lumina transmisă prin obiecte.

Pentru a afla culoarea luminii reflectate şi a celei transmise trebuie determinate obiectele de la care provine această lumină. Pe de altă parte, culoarea luminii care pleacă de la oricare dintre aceste obiecte se determină în acelaşi mod, de unde rezultă caracterul *recursiv* al calculelor.

Algoritmul `ray-tracing` de bază poate fi extins pentru a trata umbrele, reflexie şi refracţie. Algoritmul de bază determină culoarea pixelului la cea mai apropiată intersecţie a razei vizuale. Pentru a calcula umbrele se trasează raze adiţionale de la punctul de intersecţie la fiecare din sursele de lumină. Dacă una dintre aceste raze intersectează un obiect în calea sa, atunci obiectul iniţial este umbrit în acel punct de intersecţie cu raza vizuală, iar algoritmul de iluminare ignoră contribuţia sursei de lumină spre care raza a fost blocată. Fiecare rază de relecţie şi refracţie pot produce recursiv alte raze reflectate şi refractate (figura 5.48). Astfel razele formează un arbore, precum cel din figura 5.49. O ramură a acestui arbore este terminală dacă razele reflectate sau refractate nu intersectează un alt obiect, sau o anumită adâncime prestabilită a fost atinsă.

Modelul de iluminare poate fi extins pentru a include şi reflexia speculară şi transparenţa refractată:

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_{i=1}^{m} s_i f_{at_i} I_{p\lambda_i} [k_d O_{d\lambda} (N \cdot L_i) + k_s (N \cdot H_i)^n] + k_s I_{r\lambda} + k_t I_{t\lambda}$$

unde $I_{r\lambda}$ este intensitatea razei reflectate, $k_t$ este coeficientul de transmisie, cu valori între 0 şi 1, iar $I_{t\lambda}$ este intensitatea razei transmise prin refracţie.

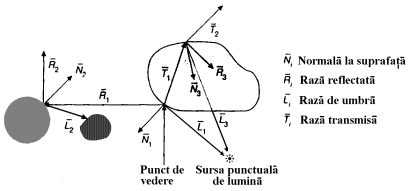



Figura 5.48: Recursivitatea razei

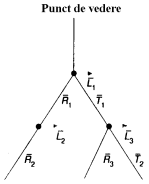



Figura 5.49: Arborele razei

Valorile $I_{r\lambda}$ şi $I_{t\lambda}$ sunt determinate prin evaluarea recursivă a ecuaţiei de mai sus la cea mai apropiată suprafaţă care este intersectată de razele reflectate şi transmise.

In cele ce urmează sunt prezentate, în pseudocod, elementele principale ale unui algoritm `ray-tracing` recursiv. `RT_trace` determină cea mai apropiată intersecţie pe care raza vizuală o realizează cu un obiect şi apelează `RT_shade` pentru a determina culoarea punctului. In primul rând `RT_shade` determină culoarea ambientală a intersecţiei. Apoi o rază este schiţată până la fiecare sursă de lumină pentru a determina contribuţia sa la culoare. Un obiect opac blochează total lumina, pe când un obiect transparent scalează contribuţia sursei de lumină. Apelurile recursive la `RT_shade` tratează razele de reflexie pentru obiecte reflective şi razele de refracţie pentru obiectele transparente. `RT_trace` reţine arborele razei numai atâta timp cât este necesar pentru determinarea culorii pixelului curent.

*selectează centrul de proiecţie şi fereastra în planul de proiecţie*
`for` *fiecare linie de scanare* `do`

```
    for fiecare pixel din linia de scanare do
        begin
              determină raza de la centrul de proiecţie la pixel;
              pixel := RT_trace(raza, 1)
        end;
Procedure RT_trace(raza:RT_ray;adâncimea:integer):RT_color;
begin
     determină cea mai apropiată intersecţie a razei cu un obiect;
     if obiect atins then
       begin
        calculează normala la intersecţie
        RT_trace:=RT_shade(cel mai apropiat obiectul atins, raza,
          intersecţia, normala, adâncimea)
       end
     else
        RT_trace:=Background_value;
end;
procedure RT_shade(
 obiect : RT_object;     {obiectul intersectat}
 raza : RT_ray;          {raza incidentă}
 punct : RT_point;       {punctul intersecţiei}
 normala : RT_normal;{normala în punct}
 adâncime:integer;    {adâncimea în arborele razei}
):RT_color;
var
   culoare : RT_color; {culoarea razei}
   rRaza, tRaza, sRaza : RT_ray; {raze reflect.,refract.,lumină}
   rCuloare, tCuloare : RT_color; {cul.raze reflect.,refract.}
begin
   culoare := termenul luminii ambiente;
   for fiecare sursă de lumină do
     begin
       sRaza :=raza de la lumină la punct;
       if produsul scalar dintre normală şi direcţia luminii este pozitiv
        then begin
          calculează câtă lumină este blocată de suprafaţă;
          scalează corespunzător termenii luminii speculare şi difuze;
          adună termenii la culoare;
        end
     end
   if adâncimea<maxDepth then
       begin
         if obiectul este reflectiv then
          begin
            rRaza :=raza în direcţia reflexiei de la punct;
            rCuloare :=RT_trace(rRay, depth + 1);
            scalează rCuloare cu coeficientul specular;
            adună rCuloare la culoare
          end;
```

```
            if obiectul este transparent then
             begin
               tRaza :=raza în direcţia de refracţie de la punct;
               if nu apare reflexie internă then
               begin
                  tCuloare :=RT_trace(tRaza,adâncime+1);
                  scalează tCuloare cu coeficientul de transmitere;
                  adună tCuloare la culoare
               end
             end
          end
       RT_trace:=culoare
    end;
```

Numărul de raze care trebuie procesate creşte exponenţial cu adâncimea până la care sunt trasate razele. In cel mai rău caz, arborele unei raze (în condiţiile unei singure surse de lumină) va conţine $2^n - 1$ raze, unde $n$ este adâncimea arborelui. Dacă există $m$ surse de lumină numărul maxim de raze devine $m(2^n - 1)$. In plus, deoarece razele pot veni din orice direcţie, nu se pot utiliza tehnici de decupare faţă de volumul de vedere şi nici alunecarea spate-faţă relativ la observator.

Numeroase tehnici de reducere a efortului de calcul au fost elaborate pentru algoritmul recursiv (de exemplu, prin *control adaptiv* a adâncimii arborelui – razele nu mai sunt urmărite dacă contribuţia pixelului la culoare este estimată a fi sub un anumit nivel –, prin `buffere` de lumină – un cub centrat în sursa de lumină şi aliniat cu axele sistemului lumii înconjurătoare, cu fiecare faţă împărţită într-o grilă rectangulară, fiecărui pătrat fiindu-i asociată o listă de suprafeţe sortate în adâncime în ordinea în care sunt văzute de la lumină, iar o rază de lumină este procesată prin determinarea pătratului prin care trece, astfel încât raza este testată la intersecţie numai cu lista de suprafeţe asociată pătratului –, sau prin *clasificarea razelor*).

Performaţe deosebite se obţin prin combinarea metodei drumului cu metoda radiosităţii. Opus modalităţii de abordare a algoritmilor clasici, *metoda radiosităţii* determină în primul rând interacţiunile legate de luminarea scenei, independent de sistemul de vizualizare, şi doar apoi se aplică determinarea suprafeţelor vizibile şi interpolarea iluminării conform unui sistem oarecare de vizualizare. Presupunerea de bază este conservarea energiei luminii într-un mediu închis. Radianţa este rata în care energia părăseşte suprafaţa.

## 5.4 Atributele primitivelor

Imaginea unui primitive poate fi controlată prin atributele sale. Fiecărei primitive îi sunt specifice anumite atribute.

Atributul *punctului* este culoarea.

Atributele unei *linii* (drepte sau poligonale) sunt:

(a) stilul liniei (continuă sau solidă, întreruptă, punctată sau corespunzătoare unui şablon)

(b) grosimea liniei;

(c) culoarea;
(d) stilul peniţei (pentru liniile groase, umplerea cu texturi a zonelor dreptunghiulare asociate)

Un *şablon* (model, `pattern`) este desemnat printr-o matrice. Inhibarea aprinderii sau aprinderea, conform unei anumite culori, a unui pixel $(x, y)$ de pe o primitivă se realizează în urma testării valorii înscrise în matricea şablon $m \times n$ la poziţia ($x$ mod $m$, $y$ mod $n$). De exemplu considerăm că şablonul poate fi reprezentat printr-un şir *string* 16 biţi (dacă şablonul se repetă după fiecare 16 biţi). Funcţia `WritePixel` din algoritmii incrementali este înlocuită cu

`if string[`$i$ `mod 16] then WritePixel(`$x, y, value$`);`

unde $i$ este variabila ciclului incremental. In această situaţie lungimea liniuţelor întrerupte în cazul primitivei linii variază cu unghiul liniei (mai lungă în cazul liniilor înclinate decât la liniile orizontale sau verticale). O soluţie este convertirea directă a liniuţelor ca segmente de linii individuale, de lungime invariantă cu unghiul de înclinare (vârfuri sunt calculate exact, ca funcţie de stilul liniei selectate). Liniile groase sunt create ca secvenţe de dreptunghiuri solide şi transparente a căror vârfuri sunt calculate exact ca funcţiei de stilul liniei.

Grosimea unei primitive este distanţa dintre marginile primitivei calculată pe perpendiculara pe tangenta sa. Primitivele ce se trasează cu o anumită grosime se bazează pe convertirea prin scanare a unei primitive cu grosimea de un pixel. Ingroşarea unei linii se poate efectua prin următoarele procedee:

1. *duplicarea pixelilor pe coloane* pentru pante subunitare şi pe linii, în caz contrar (figura 5.50). Această tehnică are mai multe dezavantaje:
   (a) liniile se termină întotdeauna vertical sau orizontal;
   (b) grosimea liniilor depinde de înclinare: dacă pentru o linie orizontală grosimea este $t$, la acelaşi număr de pixeli, o linie cu panta 1 are grosimea $t/\sqrt{2}$, obţinându-se un efect negativ asupra strălucirii (contrast între liniile orizontale sau verticale şi cele înclinate);
   (c) problema centrării primitivei duplicată în cazul unei grosimi număr par (în mod obişnuit se va trece la următorul număr impar);
   (d) grosime redusă la schimbarea octanţilor unei elipse (figura 5.51)

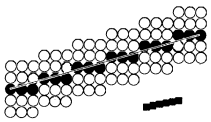


Figura 5.50: Segment trasat prin metoda duplicărea pixelilor pe verticală

2. alegerea unei *peniţe dreptunghiulare* al cărui centru străbate primitiva trasată la un pixel (figura 5.52). Grosimea variază cu înclinaţia: segmentele orizontale par mai subţiri decât cele înclinate. In cazul unui arc de cerc în dreptul bisectoarei cadranului 1, primitiva are un număr

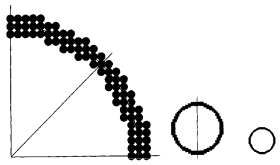


Figura 5.51: Semicerc gros trasat prin duplicarea pixelilor pe coloană

de pixeli de approximativ $\sqrt{2}$ mai mare decât în dreptul axei $x$ (figura 5.53). Acest fenomen poate fi eliminat dacă se utilizează o peniţă circu-

Figura 5.52: Segment de linie trasat cu o peniţă dreptunghiulară

lară. In ceea ce priveşte implementarea trebuie evitată rescrierea pixelilor (utilizând o tehnică incrementală).

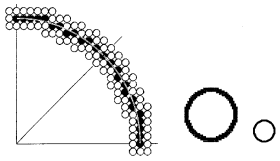


Figura 5.53: Semicerc gros trasat cu o peniţă dreptunghiulară

3. *umplerea ariei dintre margini*: se construiesc două primitive aflate la distanţa de $t/2$ de traiectoria primitivei ideale (cu un singur pixel). Pentru primitivele cu aria definită, frontiera originală poate fi considerată ca fiind marginea exterioară, iar cea interioară va fi construită în interiorul primitivei. Avantajul acestei metode constă în faptul că pot fi tratate atât primitive cu grosimi impare cât şi primitive cu grosimi pare. Grosimea efectivă a unei linii este independentă de unghiul de înclinare faţă de axe. In cazul cercurilor se converteşte cercul exterior de rază $R + t/2$ şi cercul

interior de rază $R - t/2$, apoi spaţiul dintre aceste primitive este umplut (se poate observa cum primitiva ideală, de un pixel, apare deplasată). In cazul elipselor se trasează elipsa cu semidiametrele $a - t/2$ şi $b - t/2$ şi cea cu semidiametrele $a + t/2$ şi $b + t/2$, apoi se umple spaţiul dintre ele (figura 5.54).

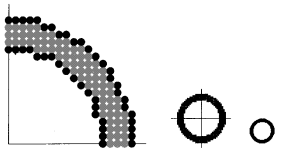


Figura 5.54: Semicerc gros trasat prin umplerea spaţiului dintre două cercuri concentrice

Atributele *poligoanelor, cercurilor şi elipselor* sunt cele ale liniei de frontieră plus stilul de umplere (textura interiorului).

Atributele *textului* sunt multiple:

(a) stil sau font (Roman, Helvica etc);
(b) mod de tipărire (drept, îngroşat, înclinat);
(c) dimensiune (măsurată în puncte tipografice - un punct aproximativ 1/72 inch);
(d) lăţime;
(e) spaţiu între caractere;
(f) spaţiu între linii consecutive;
(g) direcţia de scriere (orizontal, vertical sau sub un anumit unghi).

## 5.5 Atenuarea efectelor datorate discretizării imaginii

Se consideră disponibil un `display` care are cel puţin două nivele de intensitate. Liniile apar în rastru ca nişte scăriţe. Primitivele curbe precum cercul şi elipsele sunt de asemenea aproximate (eşantionate), ajungându-se la un acelaşi efect vizual, numit `alias` (pornind de la teoria procesării de semnale, unde `alias` este un semnalul discret transmis prin legături de telecomunicaţie ca înlocuitor al semnalului continuu).

Fenomenul `aliasing` (în acest context, *asperitate*) este rezultatul conversiei `scan` de tipul tot sau nimic (un pixel este aprins sau stins). Aplicarea tehnicilor

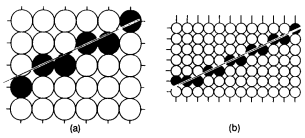



Figura 5.55: Segment de linie discret (a) pe un *display* monocrom (b) aceeaşi linie pe un *display* cu rezoluţie dublă

care reduc sau elimină fenomenul sunt referite ca `antialiasing` (*netezire* , *compensare*).

O primă metodă este *creşterea rezoluţiei*, respectiv utilizarea unui `display` cu o rezoluţie superioară (figura 5.55). O altă tehnică este cea a *pixelilor în diferite stadii*. Salturile în liniile de pe ecran sunt ajustate prin mutarea cu o micropoziţie a unor pixeli. Sistemele care permit o asemenea tehnică sunt construite astfel încât poziţii individuale de pixeli pot fi deplasate cu o fracţiune din latura pixelului, de obicei 1/4, 1/2 sau 3/4. Aceste două metode diminuează efectul de scăriţă, dar nu îl elimină.

O metodă des utilizată pe `display`-urile care permit mai multe nivele de intensitate este *metoda ariilor*. Ideea de bază este aceea că punctele şi liniile de pe un ecran au dimensiuni finite. Se consideră că un pixel este aproximativ un pătrat, iar o linie are grosimea cel puţin egală cu latura pătratului-pixel (figura 5.56a). In loc de a trasa linia cu un singur pixel corespunzător fiecărei poziţii pe axa $Ox$, toţi pixelii peste care se suprapune o parte din "aria" liniei, sunt aprinşi cu o intensitate proporţională cu aria de suprapunere (*metoda ariilor fără greutate*). Această tehnică are trei proprietăţi:

1. intensitatea unui pixel care intersectează o latură descreşte proporţional cu creşterea distanţei dintre centrul pixelului şi segmentul de linie;
2. primitiva de trasare a segmentului nu influenţează intensitatea unui pixel dacă nu intersectează aria acestuia;
3. arii egale contribuie egal la intensitate (figura 5.57).

*Metoda ariilor cu greutate* păstrează primele două proprietăţi, dar nu şi a treia: arii egale pot contribui inegal la intensitate, deoarece o arie mică apropiată de centrul pixelului are o mai mare influenţă decât o arie egală aflată la o distanţă mai mare. Aria de influenţă a unui pixel corespunde cercului determinat de centrele celor patru pixeli vecini mai apropiaţi.

Pentru a face distincţia între cele două metode a ariilor se consideră cele două cazuri din figurile 5.58 şi 5.59. In primul caz, se consideră o funcţie de greutate, $W(x, y)$, reprezentând subvolume ale unui cub a cărui bază este pixelul curent. Considerăm că intensitatea pixelului va fi proporţională cu aria acoperită multiplicată cu greutatea, $I_{max} \cdot W_S$. Pentru metoda ariilor fără greutate, înălţimea cubului este 1, iar $W_S \in [0, 1]$. Astfel, dacă linia acoperă întreg pixelul, intensitatea acestuia va fi maximă $I_{max} \cdot 1 = I_{max}$. In metoda

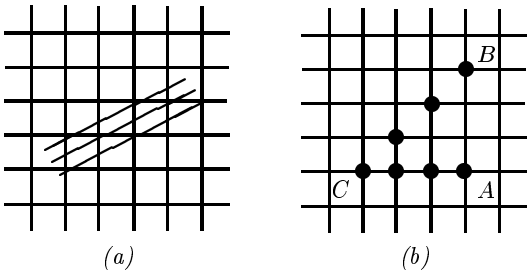



Figura 5.56: (a) Primitivele transpuse pe ecran au dimensiuni finite (b) Variaţia intensităţii funcţie de pantă

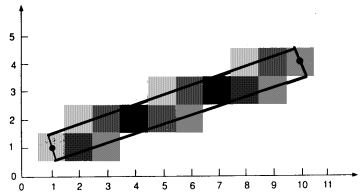



Figura 5.57: Intensitate proporţională cu aria acoperită

ariilor cu greutate, $W(x, y)$ reprezintă subvolume ale unui con a cărui bază este un cerc definit de centrele celor patru pixeli învecinaţi cu cel curent. Această funcţie are un maxim în centrul pixelului şi descreşte liniar cu distanţa de la centrul pixelului. Prin utilizarea acestei metode se elimină efectul nedorit de contrast între pixeli adiacenţi.

Algoritmul `Gupta-Sproull` de trasare a segmentelor de linii presupune precalcularea subvolumelor funcţiei de greutate definite de linii aflate la diferite distanţe de centrul pixelului şi stocarea valorilor într-un tabel. Pixelii sunt reprezentaţi prin cercuri care se suprapun (figura 5.60). Fie `Filter`$(D, t)$ funcţia discretă stocată, unde $D$ este distanţa (dependentă de unghiul de înclinare) între pixel şi centrul liniei, $t$ este este o constantă pentru linii de o anumită grosime, iar `Filter` este o funcţie dependentă de funcţia de greutate. De exemplu, pentru $t = 1$, valori ale funcţiei trebuie furnizate pentru $D$ între 0 şi 1.5. Numărul de valori din acest interval depinde de numărul de niveluri de intensitate. Reluăm algoritmul incremental pentru cazul segmentului de linie cu panta între 0 şi 1 (figura 5.61). Din figură se observă că

$$D = v \cos \phi = \frac{vdx}{\sqrt{dx^2 + dy^2}}.$$

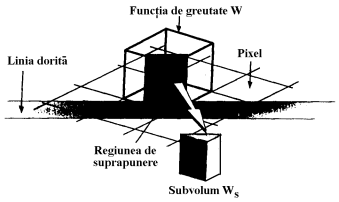



Figura 5.58: Filtru cubic pentru un pixel

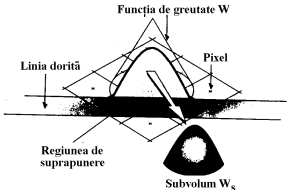



Figura 5.59: Filtru conic pentru un pixel circular cu diametrul a două unităţi

Distanţa $v$ pe verticală între punctul de pe linia ideală şi centrul pixelului ales este doar o diferenţă între coordonatele pe axa $y$. Dacă linia trece sub pixelul ales, $v$ este negativ; dacă trece pe deasupra, $v$ este pozitiv. Doar valoarea absolută a lui $v$ este utilizată de funcţia `Filter`. Pixelul selectat este în mijlocul a 3 pixeli a căror intensitate este influenţată la trasarea liniei. Pixelul de deasupra celui selectat se află la o distanţă de $1 - v$ pe verticală de la linia ideală, iar pixelul de sub cel selectat se află la $1 + v$. $v$ poate fi calculat direct sau incremental. De preferat este varianta a doua. Considerăm variabila decizională $d = 2F(x_P + 1, y_P + \frac{1}{2})$ unde $F(x, y) = ax + by + c$ cu $b = -dx$. Dacă se selectează $E$

$$2vdx = -2(y - y_P)b = 2[a(x_P + 1) + by_P + c] = 2F(x_P + 1, y_P) =$$

$$= 2a(x_P + 1) + 2b\left(y_P + \frac{1}{2}\right) - b + 2c = d + dx,$$

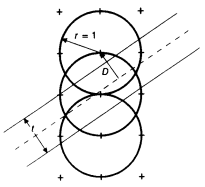



Figura 5.60: O linie de grosime 1 intersectează aria a 3 pixeli

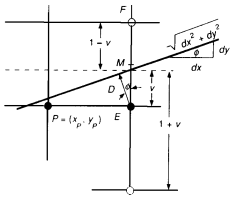



Figura 5.61: Calcularea distanţelor până la linie în algoritmul punctului de mijloc

astfel încât

$$D = \frac{d + dx}{2\sqrt{dx^2 + dy^2}}.$$

cu numitorul constant. Numărătorii fracţiilor $D$ corespunzătoare pixelii de pe aceeaşi abscisă, dar la $y_P + 1$ respectiv $y_P - 1$ sunt obţinuţi din $2(1 - v)dx = 2dx - 2vdx$, respectiv $2(1 + v)dx = 2dx + 2vdx$. In mod similar, în cazul în care se selectează pixelul $(x_P + 1, y_P + 1)$,

$$2vdx = 2F(x_P + 1, y_P + 1) = 2a(x_P + 1) + 2b(y_P + \frac{1}{2} + b + 2c = d - dx$$

iar numărătorii corespunzători pixelilor $y_P+2$ şi $y_P$ sunt $2(1-v)dx = 2dx-2vdx$ şi $2(1 + v)dx = 2dx + 2vdx$.

In algoritmul punctului de mijloc în varianta `Gupta-Sproull`, reprezentat prin procedura ce urmează, funcţia `WritePixel` este înlocuită cu `IntesifyPixel` pentru pixelul aprins şi vecinii săi pe verticală, care accesează

tabelul de de corespondenţă între valoarea absolută a distanţei şi numere ce reprezintă fracţii din intensitatea maximă.

```
Procedure AntiAliasedLineMidpoint(x1, y1, x2, y2 :integer);
var
   dx, dy, incrE, incrF, d, x, y, v :integer;
   fr, dxfr :real;
   procedure IntensifyPixel(x, y :integer, distance :real);
   var
      intensity :real;
   begin
      {Tabel cu index întreg}
      intensity :=Filter(Round(Abs(distance)));
      WritePixel(x, y, intensity);
   end;
begin
   dx := x2 − x1; dy := y2 − y1;
   d := 2 ∗ dy − dx;
   incrE := 2 ∗ dy;  incrF := 2 ∗ (dy − dx);
   v := 0;
   fr := 1/(2∗Sqrt(dx ∗ dx + dy ∗ dy));
   dxfr := 2 ∗ dx ∗ fr;
   x := x1;  y := y1;
   IntensifyPixel(x, y, 0);
   IntensifyPixel(x, y + 1, dxfr);
   IntensifyPixel(x, y − 1, dxfr);
   while x < x2 do
      begin
       if d < 0 then
          begin {Alege E}
                fr := d + dx;
                d := d + incrE;
                x := x + 1
          end
       else
          begin
                fr := d − dx;
                d := d + incrF;
                x := x + 1;
                y := y + 1
          end;
       IntensifyPixel(x, y, v ∗ fr);
       IntensifyPixel(x, y + 1, dxfr − v ∗ fr);
       IntensifyPixel(x, y − 1, dxfr + v ∗ fr)
      end {while}
end;
```

*Corectarea intensităţii pixelilor* se utilizează de asemenea pentru anularea şi altor efecte a trasării liniilor pe rastru. Astfel, o linie orizontală de aceeaşi lungime cu una înclinată va apărea mai luminată decât cea din urmă, chiar dacă

numărul de pixeli este identic. Explicaţia constă în faptul că intensitatea per unitate de lungime (geometrică) este mai redusă în cazul segmentului înclinat. In figura 5.56 (b) segmentul $CB$ are panta 1 şi este de $\sqrt{2}$ ori mai lung decât segmentul orizontal $CA$, deşi au acelaşi număr de pixeli în reprezentarea discretă. Dacă intensitatea per pixel este $I$, intensitatea per unitatea de lungime a segmentului $CA$ este $I$, iar a segmentului $CB$ este $I/\sqrt{2}$. Pentru compensarea acestui efect nedorit, se ajustează intensitatea de trasare a fiecărei linii în funcţie de panta acesteia. Liniile verticale şi orizontale sunt trasate cu intensitate redusă, în timp ce liniile înclinate la $45^0$, cu intensitate maximă.

Tehnica `antialiasing` este utilizată pentru obţinerea unor imagini de calitate. Datorită efortului de calcul implicat, tehnica este utilizată de obicei doar în ultimul pas al creării unei imagini.

## 5.6 Umplerea poligoanelor

Umplerea poligoanelor presupune, în primul rând, un algoritm pentru determinarea poziţiei unui punct relativ la un poligon (se testează dacă punct este sau nu în interiorul poligonului şi în caz afirmativ, se aprinde, eventual cu o anumită culoare sau intensitate).

Fiecare algoritm de umplere poate fi logic descompus în următoarele:

1. metoda de *propagare* care determină următorul punct ce va fi tratat;
2. procedura de *start* care iniţializează algoritmul;
3. procedura de determinare a *interiorului* poligonului;
4. procedura de *setare* care schimbă starea pixelului.

Algoritmii de umplere se diferenţiază funcţie de următoarele două situaţii:

(a) umplerea se realizează odată cu trasarea poligonului;
(b) umplerea se realizează după ce poligonul a fost trasat.

### 5.6.1 Primitiva poligon plin

In primul caz, una din problemele principale este aceea de a defini interiorul şi exteriorul. Problema nu este simplă dacă poligonul nu este convex.

*Regula par-impar*, menţionată la algoritmii de determinare a liniilor şi suprafeţelor ascunse, este des utilizată pentru determinarea interiorului unui poligon. Se alege un punct oarecare din interiorul regiunii ce urmează a fi testată şi o linie dreaptă ce pleacă din acel punct spre o direcţie arbitrară şi nu trece prin nici un vârf. Fie pe această dreaptă un segment delimitat de punct şi altul oarecare ce nu aparţine interiorului dreptunghiului ce încadrează figura. Dacă segmentul intersectează laturile poligonului de un număr impar de ori, regiunea este considerată de interior. In figura 5.62 se prezintă regiunile interioare ale unui poligon concav.

Tehnica par-impar este utilizată şi în procesul de selectare a unui obiect cu ajutorul cursorului grafic.

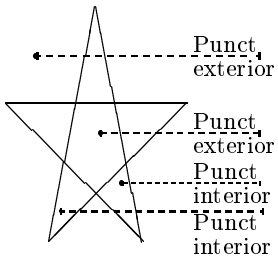



Figura 5.62: Determinarea regiunilor interioare prin regula par-impar

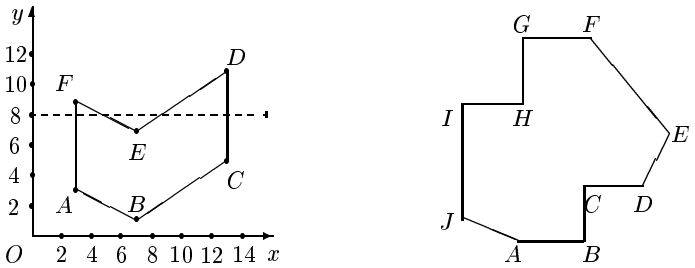



Figura 5.63: Conversia scan a poligoanelor (a) exemplu poligon concav (b) poligon cu laturi orizontale

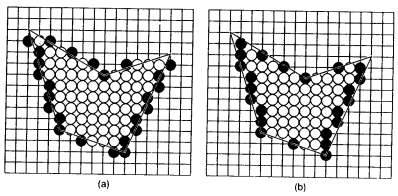



Figura 5.64: Intervale de umplere cu pixelii marginali marcaţi cu negru (a) aplicând algoritmul punctului mijlociu apar pixeli în exterior (b) pixeli corect aleşi

**Conversia scan a poligoanelor pline**

Se consideră poligonul din figura 5.63.a. Trebuie să determinăm care pixeli de pe fiecare linie de scanare sunt în interioriul poligonului şi să setăm pixelii corespunzători la valorile corespunzătoare. Acest proces trebuie repetat pentru fiecare linie de scanare care intersectează poligonul.

Cea mai simplă variantă de trasare constă în utilizarea algoritmilor de trasare a segmentelor de linii ce formează frontiera poligonului şi ţinerea evidenţei pixelilor intersectaţi de fiecare linie de baleiaj. In această manieră se vor aprinde anumiţi pixeli din afara poligonului (fiind mai apropiaţie de o latură), fără a ţine seama dacă sunt în interiorul sau exteriorul poligonului. Acest fenomen poate avea consecinţe negative în momentul când se trasează mai multe poligoane de culori diferite, pixelii menţionaţi fiind susceptibili a ocupa regiuni ale poligoanelor învecinate (sau în momentul în care are loc ştergerea unui poligon din colecţie). Este de preferat să se traseze numai pixelii care sunt strict în interiorul regiunii. Algoritmul de scanare trebuie ajustat în consecinţă (figura 5.64).

Construim un algoritm incremental pentru calcularea pixelilor care mărginesc zonele pline. Distincţia dintre interior şi exterior este realizată la trecerea liniei de baleiaj. Se presupune că iniţial se porneşte din afara oricărui poligon. Astfel după prima intersecţie cu o latură, se trece de la exterior la interior. La a doua intersecţie cu o latură a unui poligon se trece de la interior la exterior. Repetând acest procedeu, se observă că după un număr impar de intersecţii, pixelii sunt în interior, iar după un număr par, în exterior (consecinţă a regulii par- impar). Prin convenţie, laturile se consideră interioare poligonului.

Pentru fiecare linie de baleiaj, algoritmul de umplere presupune:

1. determinarea intersecţiilor liniei de baleiaj cu fiecare din laturile poligonului;
2. sortarea intersecţiilor în ordinea crescătoare a absciselor;
3. aprinderea pixelilor între perechi de intersecţii care se află în interiorul poligonului pe baza parităţii în curs. Paritatea iniţială este pară. Aprinderea pixelilor are loc numai dacă paritatea este impară. Paritatea se schimbă la intersecţia laturilor poligonului.

In pasul 3 apar următoarele probleme:

(a) Dată fiind o intersecţie cu un $x$, fracţionar, arbitrar, cum determinăm care dintre pixelii aflaţi de o parte şi de alta a intersecţiei sunt de interior: dacă ne apropiem de o intersecţie fracţionară din dreapta şi din interiorul poligonului, atunci punctul de interior va avea abscisa egală cu partea întreagă a coordonatei $x$ a intersecţiei; dacă ne aflăm în afara poligonului, se adaugă o unitate la partea întreagă.

(b) Cum tratăm cazul special al intersecţiei cu $x$ întreg: dacă ne apropiem de intersecţie din exterior, pixelul va fi considerat de interior; dacă ne apropiem de intersecţie din interior, pixelul va fi considerat de exterior.

(c) Cum tratăm cazul special al vârfurilor poligonului: se trasează şi ţine seama în calcul de paritate numai de un vârf, al unei laturi, a cărui $y$ este minim relativ la acea latură (în cazul în care un vârf are $y$ maxim pentru o latură, el va fi trasat numai dacă are $y$ minim relativ la o latură

adiacentă). In figura 5.63.a, la scanarea liniei $y = 3$ vârful $A$ este numărat o singură dată la paritate şi este trasat fiind vârf cu $y$ minim pentru latura $FA$ şi maxim pentru latura $AB$. Linia de scanare $y = 1$ întâlneşte doar pe $B$; deoarece liniile $AB$ şi CD au ambele $y$ minim în $B$, acesta va fi numărat de două ori la paritate (intrare în intervalul de pixeli, trasare pixel, ieşire din intervalul de pixeli). La intersecţia liniei $y = 9$ cu vârful $F$ nu se va trasa pixelul ($F$ are $y$ maxim pentru ambele laturi $FA$ şi $EF$, trecerea prin acest vârf neafectând paritatea).

(d) Cum tratăm cazul în care avem linii orizontale: de exemplu, se consideră cazul poligonului din figura 5.63.b. Se porneşte de la latura de jos. Cum la $A$, $y$ este minim, iar $AB$ coincide cu linia de baleiaj, paritatea este impară şi segmentul $AB$ este trasat. Cum $BC$ are $B$ ca minim, paritatea devine, după $B$, pară. La vârful $J$, latura $IJ$ are $J$ ca minim pe axa $y$, dar nu şi latura $JA$, astfel încât paritatea devine impară şi intervalul de umplere începe cu $J$. Intervalul de umplere care porneşte de la latura $IJ$ şi atinge $C$ continuă, deoarece $C$ este maxim local pe axa $y$ (pentru $BC$ şi $CD$). La $D$, minim local, paritatea se schimbă şi intervalul de umplere este închis. La $I$, maxim local, paritatea rămâne şi segmentul $IH$ nu este umplut. La $H$, minim local, paritatea se schimbă şi intervalul de umplere este deschis la $H$ şi închis la intersecţia cu $EF$. In mod similar, $GF$ nu este trasat.

La calcularea intersecţiilor trebuie evitată tehnica brută de testare a fiecărei laturi a poligonului relativ la intersecţiile cu fiecare linie nouă de scanare. De obicei, doar un număr redus de laturi intersectează linia de baleiaj. De asemenea, majoritatea laturilor care intersectează linia $i$ vor intersecta şi linia $i + 1$ (fenomen numit coerenţa laturilor). Când trecem de la o linie de scanare la alta, putem calcula noua valoare $x$ a intersecţiei a laturii pe baza valorii $x$ vechi a intersecţiei (precum în cazul trasării liniilor), utilizând $x_{i+1} = x_i + 1/m$, unde $m$ este panta laturii. In algoritmii incrementali anteriori aritmetica fracţionară este evitată prin calcularea unei variabile de decizie întreagă şi verificarea semnului ei în vederea alegerii pixelului celui mai apropiat de curbă.

Se consideră o latură cu panta mai mare decât $+1$ ca fiind latura stângă a unui poligon (cazurile contrare sunt tratate în mod similar). Presupunem că se porneşte de la coordonatele întregi $(x_{min}, y_{min})$. Când $y$ este incrementat, coordonata $x$ a punctului de pe linia ideală va creşte cu $1/m$, unde $m = (y_{max} - y_{min})/(x_{max} - x_{min})$ este panta liniei. $x$ va avea o parte întreagă şi una fracţionară, care poate fi exprimată ca o fracţie având numitorul $y_{max} - y_{min}$. De exemplu, dacă $x_{min}$ este 3, iar panta este $\frac{5}{2}$, secvenţa de valori a lui $x$ va fi 3, $3\frac{2}{5}$, $3\frac{4}{5}$, $3\frac{6}{5} = 4\frac{1}{5}$ etc. Dacă partea fracţionară a lui $x$ este nenulă, aceasta va trebui rotunjită pentru a determina pixelul de interior al poligonului. Când partea fracţionară a lui $x$ devine mai mare decât 1, $x$ trebuie incrementat şi din partea fracţionară se scade 1. Utilizarea fracţiilor poate fi evitată urmărind doar numitorul fracţiei $m$: se consideră o variabilă increment care contorizează adunările ce se fac la numitor până când acesta depăşeşte numitorul, moment în care numărătorul trebuie decrementat cu numitorul şi $x$ trebuie incrementat.

Construim un algoritm care preia avantajele coerenţei laturilor. Evidenţa mulţimii de laturi ce intersectează o linie de baleiaj se realizează printr-o structură numită *tabelul laturilor active* (`AET`). Laturile din acest tabel sunt sortate

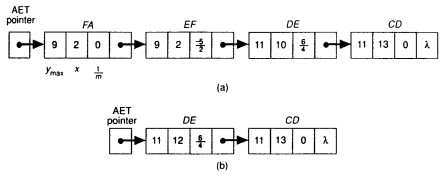



Figura 5.65: AET la (a) linia-`scan` 9 (b) linia-`scan` 10

după abscisele $x$ ale intersecţiilor (figura 5.65). La trecerea la o nouă linie de baleiaj, se actualizează tabelul: se elimină laturile care nu mai intersectează linia de baleiaj, se adaugă noile laturi intersectate şi se calculează valorile absciselor pentru laturile care au rămas tabel, printr-un algoritm incremental. Pentru simplificarea operaţiei de adăugare de laturi se creează un tabel al laturilor (`ET`) care conţine toate laturile sortate după coordonatele $y$ cele mai mici. Tabelul de laturi este constituit din mai multe subtabele corespunzătoare în număr cu numărul liniilor de baleiaj. Fiecare subtabel conţine descrierea laturilor în ordinea crescătoare a absciselor $x$ a punctelor ce se află pe linia de baleiaj (figura 5.66). Fiecare intrare a subtabelelor conţine coordonata $y$ maximă a laturii, coordonata $x$ a capătului laturii cu $y$ minim şi incrementul lui $x$ utilizat la trecerea de la o linie de baleiaj la alta (1/pantă).

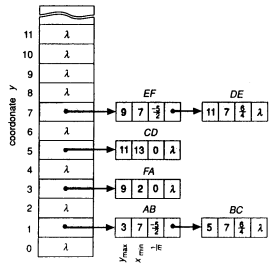



Figura 5.66: Tabelul `ET`

Algoritmul este următorul:

1. setează $y$ pe cea mai mică valoare coordonată $y$ din `ET` cu $y$-ul primei liste înlănţuite nevide;
2. iniţializează `AET` pe lista vidă;
3. repetă până când `AET` şi `ET` sunt vide:
   (a) mută din lista corespunzătoare lui $y$ din `ET` în `AET` acele laturi pentru care $y_{min} = y$ (laturi de intrare), apoi sortează `AET` pe baza lui $x$ (sortare rapidă deoarece `ET` este presortat)
   (b) aprinde pixelii de pe linia de scanare corespunzătoare lui $y$ utilizând perechi de coordonate din `AET`;
   (c) elimină din `AET` acele intrări pentru care $y = y_{max}$ (laturile nu vor fi utilizate la următoarea linie de scanare);
   (d) incrementează $y$ cu 1 (coordonatele următoarei linii de scanare);
   (e) pentru fiecare latură care rămâne în `AET`, actualizează $x$ pentru noul $y$.

Tringhiurile şi patrulaterele pot fi tratate precum cazuri speciale de poligoane deoarece au doar două laturi care se intersectează cu linia de scanare (excluzând laturile orizontale). Cum un poligon arbitrar poate fi descompus într-o reţea de triunghiuri, trasarea lui se poate efectua pe baza acestei descompuneri (trasarea triunghiurilor componente). Triangularizarea este o problemă clasică în geometrie şi este uşor de realizat pentru poligoane convexe; este însă dificilă în cazul poligoanelor concave.

Strategia descrisă în algoritmul de mai sus poate fi utilizată şi în cazul cercurilor şi elipselor.

### 5.6.2 Umplerea regiunilor din rastru

In cazul umplerii unui poligon deja trasat, intervin o serie de probleme adiţionale în implementarea algoritmilor. Aceste probleme sunt generate de structura discretă a `frame buffer`-ului. Una dintre acestea este determinarea laturilor unui poligon după ce acesta a fost deja figurat pe ecran. O altă problemă este determinarea interiorului.

O *regiune* din rastru este o colecţie de pixeli. Există două tipuri de regiuni conexe. O regiune este 4-conexă dacă oricare doi pixeli ai regiunii pot fi uniţi printr-o secvenţă de pixeli utilizând doar mutări în direcţiile sus, jos, stânga, dreapta. O regiune este 8-conexă dacă oricare doi pixeli ai regiunii pot fi uniţi printr-o secvenţă de pixeli prin mişcările anterior menţionate, plus cele pe diagonale. O regiune 4-conexă este 8-conexă, dar nu şi reciproc (vezi colecţia de pixeli aprinşi din figura 5.67).

O regiune poate fi definită în două moduri:

1. o *regiune definită prin interior* relativ la un pixel $P$ este cea mai mare regiune conexă de pixeli a căror valoare din `frame-buffer` este aceeaşi cu a lui $P$ şi care conţine pe $P$;
2. o *regiune definită prin frontieră* relativ la un pixel de interior $P$ şi unul de frontieră $Q$ este cea mai mare regiune conexă de pixeli a căror valoare nu coincide cu valoarea frontierei (valoarea lui $Q$) şi care conţine pe $P$ (interiorul poate conţine pixeli de culori diferite).

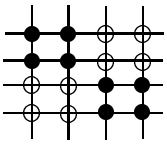


Figura 5.67: Regiune 8-conexă în rastru

### Algoritmi recursivi

In cazul în care poligonul este deja trasat în rastru, se pot utiliza tehnicile recursive de umplere. Se presupune că se cunoaşte un punct din interiorul poligonului. Dacă este aprins, atunci este pe o latură a poligonului. Se aplică tehnica par-impar pentru depistarea unui punct de interior care nu este pe nici o latură a poligonului. Dacă punctul interior considerat nu e aprins, se aprinde şi se continuă algoritmul pentru cei patru (sau opt) pixeli vecini. Dacă pixelul curent este aprins, se renunţă la apelul recursiv. Anumiţi pixeli pot fi vizitaţi însă de mai multe ori.

Algoritmii care umplu regiunile definite prin interior sunt numiţi algoritmi `flood-fill`, iar cei pentru regiunile definite prin frontieră, `boundary-fill`.

Procedurile recursive care implementează aceste tehnici sunt următoarele:

```
Procedure FloodFill4(
  x, y :integer; {punct de start}
  valVeche, {valoare ce defineşte interiorul}
  valNou : color); {valoare de înlocuire}
begin
  if ReadPixel(x, y) = valVeche then
   begin
     WritePixel(x, y, valNou);
     FloodFill4(x, y − 1, valVeche, valNou);
     FloodFill4(x, y + 1, valVeche, valNou);
     FloodFill4(x − 1, y, valVeche, valNou);
     FloodFill4(x + 1, y, valVeche, valNou)
   end
end;

procedure BoundaryFill4( x, y :integer; {punct de start}
valMargine, {valoare ce defineşte marginea}
valNou : color); {valoare de înlocuire}
var
   c : color;
begin
   c :=readPixel(x, y);
   if c <> valMargine and {Nu s-a atins marginea}
      c <> valNou then {Nu s-a mai trecut pe aici}
    begin
      WritePixel(x, y, valNou);
      BoundaryFill4(x, y − 1, valMargine, valNou);
      BoundaryFill4(x, y + 1, valMargine, valNou);
```

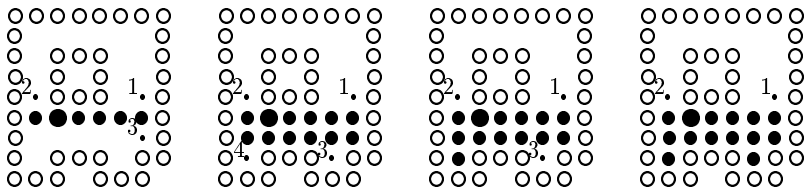


Figura 5.68: Paşii algoritmului de umplere pe bază de stivă

```
        BoundaryFill4(x + 1, y, valMargine, valNou);
        BoundaryFill4(x − 1, y, valMargine, valNou)
    end
end;
```

Pentru suprafeţe mari, pot interveni probleme de memorie – numărul mare de proceduri neterminate poate necesita o stivă sistem de dimensiune mare.

**Algoritmi pe bază de stivă**

Un algoritm care evită problemele algoritmilor recursivi este cel care lucrează cu *intervale de umplere*. Aceste intervale sunt mărginite la ambele capete de pixeli de valoarea frontierei. Un interval este identificat prin pixelul său cel mai din dreapta. Se consideră un punct de interior cunoscut. Se umple intervalul continuu de pixeli ce conţine punctul de start. Apoi linia de deasupra este examinată de la dreapta la stânga pentru a determina cel mai din dreapta pixel al intervalului. Adresa acestui pixel este memorată în stivă. La fel se procedează cu linia de sub cea umplută. In figura 5.68 se dă un exemplu: intervalul de umplere ce conţine punctul de start este figurat în (a) (punctul de start este îngroşat). Adresele pixelilor numerotaţi sunt introduşi în stivă. Numerele indică ordinea din stivă, 1 fiind procesat ultimul. Algoritmul se opreşte când stiva este goală.

## 5.7 Texturi

Textura se defineşte ca o structură compusă din elemente primitive (tipare) care se repetă mai mult sau mai puţin regulat.

### 5.7.1 Clasificare

Texturile sau detaliile ce apar pe suprafeţele obiectelor de modelat pot fi:

1. constante ca mărime şi orientare pe suprafaţa obiectului;
2. de mărime şi orientare variabilă, corespunzător poziţiei faţetelor a căror textură o reprezintă;
3. neregulate.

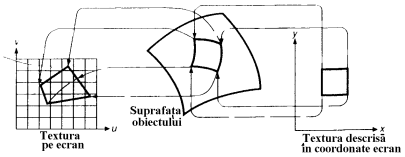



Figura 5.69: Textură transpusă pe suprafaţă

Texturile de primul tip sunt folosite pentru aplicaţiile 2D, iar în 3D se preferă texturile de al doilea tip.

Texturile de primul tip se codifică în baza de date prin matrice. Elementele matricei caracteristice texturi se numesc `texeli`. Se alege un element repetabil prin translaţie pe orizontală şi verticală. Funcţie de precizia dorită şi rezoluţia disponibilă se digitizează acest element printr-o partiţie $m \times n$ a ariei pe care este reprezentat. Fiecărui pixel astfel obţinut i se asociază un element al unei matrice $E$ cu $m$ linii şi $n$ coloane, ocupând poziţia corespunzătoare şi având valoarea 1 dacă simbolul conţine pixelul şi 0 dacă pixelul este neutilizat. Dacă se foloseşte un monitor care admite mai multe niveluri de intensitate pe pixel, atunci valoarea elementului $E_{ij}$ poate fi chiar intensitatea corespunzătoare pixelului din poziţia $i, j$.

Detaliile superficiale, cum sunt textele sau simbolurile marcate pe suprafaţa obiectelor, pot fi tratate ca texturi de categoria a doua.

Transpunerea pe ecran a texturii se poate face şi la o anumită scară. Dacă textura apare mărită, nu se pun probleme deosebite, pur şi simplu se construieşte o altă matrice pe baza lui $E$ dar având $s \cdot m$ linii şi $s \cdot n$ coloane, unde $s$ este factorul de scară. Dacă $s \cdot m$ şi $s \cdot n$ nu sunt numere întregi, ele se aproximează prin adaus, iar intensităţile în pixeli se aleg din gama disponibilă ca fiind cele mai apropiate de intensităţile medii calculate.

Texturile tridimensionale, deşi mai puţin utilizat, pot conduce la imagini mult mai realiste decât dacă se utilizează texturi bidimensionale proiectate pe suprefeţele obiectelor.

### 5.7.2 Aplicarea texturilor pe suprafeţe

O metodă de a reda texturile de tipul doi constă în a le introduce în baza de date prin puncte caracteristice, pentru ca apoi să li se aplice aceleaşi transformări precum tuturor punctelor corpului. Se marchează însă apartenenţa texturii la faţetele pe care apare efectiv, iar transpunerea pe ecran se face numai pentru punctele vizibile ale poligoanelor respective (figura 5.69).

Se consideră exemplul unei texturi ce trebuie aplicată pe o suprafaţă plană. Se caută o funcţie de mapare a unui punct $P(x, y, z)$ de pe suprafaţa plană la un punct $T(u, v)$ din spaţiul $uv$ în care se defineşte textura. Datele de intrare necesare mapării sunt: $N$, vectorul unitate normal la suprafaţa plană, $S$, vector

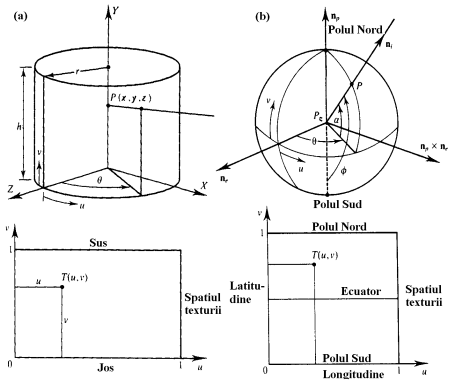



Figura 5.70: Maparea texturii (a) pe un cilindru (b) pe o sferă

unitate ortogonal pe $N$, din suprafaţa plană, care aliniază axa absciselor $u$, un punct $P_0(x_0, y_0, z_0)$ care va corespunde cu $T(0,0)$ al texturii, $k$, factor de scalare pentru textură. In aceste condiţii, maparea este definită prin

$$u = V \cdot S/k, \; v = V \cdot T/k,$$

unde $V = (x - x_0, y - y_0, z - z_0)^T$, iar $T = N \times S$.

In cazul mapării pe un cilindru de rază $r$ şi înălţime $h$ (figura 5.70.a), se deduce transformarea

$$v = y/h, \quad u = \begin{cases} \arccos(z/r)/(2\pi), & x \geq 0 \\ 1 - \arccos(z/r)/(2\pi), & \text{altfel} \end{cases}.$$

In cazul mapării pe o sferă (figura 5.70.b) sunt necesare următoarele informaţii: $n_e$, vectorul unitate normal la sferă într-un punct de la ecuator, $n_p$, vectorul unitate ortogonal pe $n_e$ şi normal la polul nord, $P_c(x_c, y_c, z_c)$, centrul sferei, $r$, raza sferei, $P(x, y, z)$ punctul care va fi mapat la $T(u, v)$. Atunci

$$v = \varphi/\pi, \quad v = \begin{cases} \theta/(2\pi), & \text{dacă } (\mathrm{n}_p \times n_e) \cdot n > 0 \\ 1 - \theta/(2\pi) & \text{altfel} \end{cases},$$

unde

$$n = ((x - x_c)/r, (x - x_c)/r, (x - x_c)/r)^T, \quad \varphi = \arccos(-n_p \cdot n),$$

$$\theta = \begin{cases} \arccos((n_e \cdot n)/\sin(\varphi)) & \varphi \neq 0 \\ 0, & \text{altfel} \end{cases}$$

### 5.7.3 Generarea texturilor tip fractal

Termenul de fractal din matematică a fost generalizat de comunitatea graficii pe calculator. In grafică, *un fractal se referă la orice obiect care are o măsură substanţială a similarităţii în sine, statistică sau exactă.* Astfel, numai fractalii generaţi prin procese recursive infinite sunt obiecte fractale adevărate. Pe de altă parte, obiectele generate prin procese finite pot conduce la schimbări nevizibile în detaliu după o anumită etapă, astfel încât sunt aproximări adecvate ale modelului ideal.

Ceea ce se înţelege prin *similaritate în sine* este sugerat de exemplul curbei lui Koch. trei. Pornind de la o linie cu un salt (figura 5.71), se înlocuieşte fiecare segment al liniei cu o figură identică cu cea originală, redusă cu factorul trei. Acest proces este repetat: fiecare segment al figurii (b) este înlocuit cu figura asemănătoare întregii figuri (b). Dacă procesul este repetat la infinit, rezultatul este similar în sine, adică întregul obiect este similar (prin translaţie, rotire, scalare) cu o porţiune a sa.

Un obiect care arată la fel dacă este scalat se spune că este substanţial similar în sine.

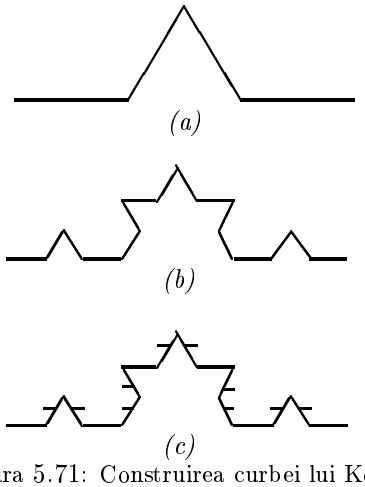



Figura 5.71: Construirea curbei lui Koch

Asociată cu noţiunea de similaritate în sine este cea de *dimensiune fractală*. Pentru a defini această noţiune se consideră următoarele obiecte:

1. Un segment de linie, care este unidimensional, divizat în $n$ părţi egale. Ficare parte a sa arată precum originalul scalat cu factorul $n = n^{1/1}$.

2. Un pătrat (figură bidimensională) divizat în $n$ subpătrate egale. Fiecare arată la fel precum originalul redus cu factorul $\sqrt{n} = n^{1/2}$ (de exemplu, considerînd nouă subpătrate, factorul este de reducere este 3).

Dacă divizăm imaginea rezultată prin aplicarea procedeului din figura 5.71, repetat la infinit, în patru părţi (analog bucăţilor asociate cu cele patru segmente originale din figura 5.71.a), fiecare bucată rezultată arată ca şi originalul redus cu factorul trei. Spunem că fractalul construit are dimensiunea $d$, dacă $d$ verifică ecuaţia $4^{1/d} = 3$, adică $d = \log_2 4 / \log_2 3 \approx 1.26$.

Cele mai cunoscute obiecte fractale sunt

(a) mulţimea `Julia-Fatou`,

(b) mulţimea `Mandelbrot`.

Aceste obiecte sunt generate urmărind regula $x \to x^2 + c$, unde $x$ este un număr complex.

Se reaminteşte faptul că dacă un număr complex are modulul subunitar, şirul recursiv al pătratelor converge la zero, iar dacă modulul este supraunitar, pătratele devin din ce în ce mai mari. Dacă modulul este unitar, şirul pătratelor are tot modulul unu. Numerele de modul unitar formează o frontieră între cele atrase spre zero şi cele atrase spre infinit.

Dacă se aplică repetat regula $x \to x^2 + c$ fiecărui număr complex, pentru o anumită valoare nenulă a lui $c$, anumite numere complexe vor fi atrase la infinit, altele spre numere finite. Trasând frontiera dintre cele două submulţimi se obţine mulţimea `Julia-Fatou`, care depinde de valoarea numărului complex $c$. Două exemple sunt prezentate în figura 5.72.

*(a)* *(b)*

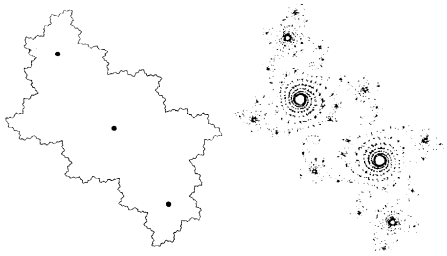


Figura 5.72: Mulţimea `Julia-Fatou` pentru (a) $c = -0,12375 + 0,056805i$ (b) $c = -0,012 + 0,74i$

Dacă se calculează mulţimile `Julia-Fatou` pentru toate valorile $c$ com-

plexe şi se colorează cu negru punctele $c$ pentru care mulţimea `Julia-Fatou` este conexă şi cu alb punctele pentru care mulţimea nu este conexă, se obţine mulţimea `Mandelbrot` din figura 5.73.

Din fericire, există o cale mai uşoară de a genera aproximaţii ale mulţimii `Mandelbrot`. Pentru fiecare valoare $c$ se aplică regula $x \to x^2 + c$ cu $x = 0 + 0i$ de un număr finit de ori (de exemplu, de 1000 de ori). Dacă după aceste iteraţii numărul complex obţinut este în afara discului centrat în zero şi de rază dată (de exemplu, 100), atunci $c$ va fi colorat alb, altfel negru. Cu cât numărul de iteraţii şi raza discului cresc, imaginea rezultată este o aproximare mai bună a mulţimii `Mandelbrot`.

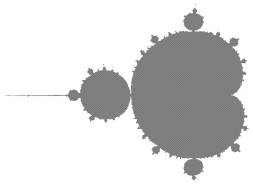


Figura 5.73: Mulţimea `Mandelbrot`

Imaginea pe ecran poate fi generată recursiv prin mai mulţi paşi astfel încît schimbările sub dimensiunile pixelului nu mai trebuie efectuate.

Formele autoasemenea (curba `Koch` de exemplu) corespund unui mod liniar de adăugare a detaliilor (sunt replici exacte ale formei iniţiale, la o scară mai mică). Detaliile pot fi adăugate şi într-un mod neliniar, obţinându-se astfel fractali autopătratici (mulţimile fractale invariante la o transformare pătratică – de exemplu mulţimea `Mandelbrot`), autoinverşi (mulţimi fractale invariante la o transformare de inversiune) etc.

Metodele de generare a fractalilor descrise mai sus se bazează pe recursivitate (care la nivelul limbajului de programare implementează proprietatea de autoasemănare) sau pe calcul iterativ. In modelarea sistemelor naturale, transformările deterministe aplicate în generarea unei figuri pot fi combinate în mod aleator. Rezultatele privind fractalii sunt extrem de sugestive pentru modelarea formelor naturale ca munţii, arborii sau coasta mării. De exemplu, se consideră următorul algoritm de generare a unui munte fractal, algoritm bazat pe subdivizarea recursivă.

Fie cazul unidimensional. Presupunem că pornim de la un segment de linie de pe axa $x$. Impărţim segmentul în două bucăţi şi mutăm mijlocul la o distanţă arbitrară în direcţia $y$, obţinând figura 5.74.b. Se continuă subdivizarea fiecărui segment şi se calculează poziţia nouă a punctului de mijloc al

segmentului de la $(x_i, y_i)$ la $(x_{i+1}, y_{i+1})$:

$$x_{nou} = \frac{x_i + x_{i+1}}{2},\ y_{nou} = \frac{y_i + y_{i+1}}{2} + P(x_{i+1} - x_i)R(x_{nou}),$$

unde $P(\cdot)$ este o funcţie ce determină extinderea perturbaţiei în termenii dimensiunii segmentului care se perturbă şi $R(\cdot)$ este un număr aleator între 0 şi 1 selectat pe baza lui $x_{nou}$. Dacă $P(s) = s$, atunci primul punct nu poate fi afişat mai sus de 1, fiecare din următoarele două puncte, nu mai sus de 1/2 ş.a.m.d., încât toate punctele rezultate se încadrează în pătratul unitate. Dacă $P(s) = s^a$ forma liniei frânte depinde de parametrul $a$. Pot fi utilizate şi alte funcţii, ca de exemplu $P(s) = 2^{-s}$.

Algoritmul poate fi uşor generalizat la două dimensiuni. Se porneşte de la un triunghi precum cel din figura 5.75. Se marchează mijloacele laturilor. Coordonatele $y$ ale fiecărui punct de mijloc sunt modificate în maniera descrisă în cazul unidimensional, astfel încît rezultatul să fie un set de mai multe triunghiuri. Cele patru triunghiuri înclinate se procesează în mod analog. Repetarea recursivă a acestei divizări conduce la o imagine realistă a unui munte.

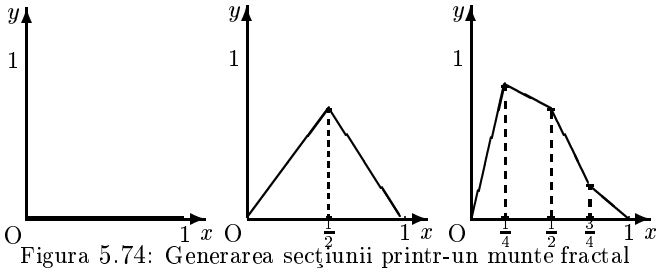


Figura 5.74: Generarea secţiunii printr-un munte fractal

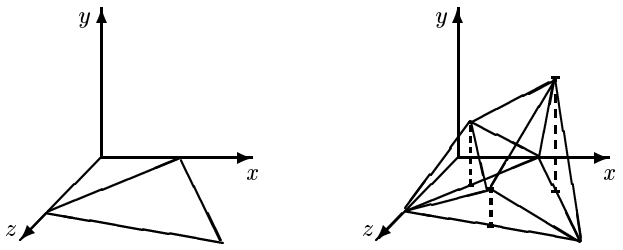


Figura 5.75: Generarea unui munte fractal

## 5.8 Culoare

Culoarea unui obiect nu depinde numai de obiectul în sine ci şi de sursa de lumină care-l luminează, de culoarea suprefeţei înconjurătoare şi de sistemul de vizualizare uman. In plus anumire obiecte reflectă lumina (obiecte compacte), altele transmit lumina (obiecte transparente).

### 5.8.1 Lumina acromatică

*Lumina acromatică* este ceea ce se distinge pe ecranul unui televizor alb-negru. Cantitatea luminii este singurul atribut al luminii acromatice. Această cantitate poate fi discutată în sensul fizic al energiei, în care termenii de *intensitate* şi *luminozitate* sunt utilizaţi, sau în sensul de percepţie a intensităţii, în care termenul de *strălucire* este utilizat. Diferitelor niveluri de intensitate li se asociază anumiţi scalari (0, negru, 1, alb, între 0 şi 1, diferite scale de gri).

Considerăm un dispozitiv cu $n+1$ nivele de intensitate (tipic $n = 255$). Se pune o primă problemă a corespondenţei intensităţilor din domeniul [0,1] cu nivele de intensitate a dispozitivului. Impărţirea intervalului [0,1] în subintervale egale este soluţia cea mai simplă, dar nu cea mai bună, datorită proprietătii ochiului de perceptare diferită a intensităţilor în raporturi de nivele de intensitate şi nu în valori absolute a intensităţii (valori absolute doar pe scara de strălucire – definită ca intensitatea percepţiei). De exemplu, diferenţa dintre intensităţile 0.10 şi 0.11 este percepută la fel ca diferenţa dintre 0.50 şi 0.55. Astfel nivelele de intensitate din [0,1] trebuie spaţiate logaritmic şi nu liniar. Dacă se consideră $I_0$ intensitatea minimă care poate fi atinsă, atunci

$$I_0 = I_0,\ I_1 = rI_0,\ I_2 = rI_1 = r^2 I_0,\ \ldots\ ,\ I_n = r^n I_0 = 1,$$

deci $r = (1/I_0)^{1/n}$, $I_j = I_0^{(n-j)/n}$, $0 \leq j \leq n$. De exemplu, pentru 4 nivele, adică $n = 3$, şi $I_0 = 1/8$, se obţine $r = 2$ şi intensităţile $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$, 1. $I_0$ în cazul unui display este între $1/200$ şi $1/40$. Minimul nu este 0 deoarece fosforul reflectă lumina incidentă pe display.

O altă problemă este numărul de nivele de intensitate necesare pentru a reproduce o imagine cu trecere continuă de la negru la alb astfel încât reproducerea să pară continuă. O asemenea situaţie apare când $r \leq 1.01$ (sub rata de 1.01 ochiul uman nu distinge diferenţa dintre două intensităţi succesive). Astfel,

$$n = \log_r(1/I_0) \geq \log_{1.01}(1/I_0),$$

cu mentţinuea că $1/I_0$ este o caracteristică dependentă de dispozitiv. In cazul `display`-urilor $n$ minim variază în intervalul [400,530], pe când în cazul dispozitivelor de tipărire a ziarelor, pentru care $I_0 = 0.1$, $n$ minim este 234. Acestea sunt valorile teoretice; în practică însă datorită efectului de amestecare a cernelii sau apariţia/dispariţia aleatoare a unor pixeli suplimentari în reproducere, $n$ descreşte considerabil pentru anumite medii de afişare. Astfel pentru tipărirea ziarelor sau cărţilor sunt suficiente 64 de nivele pentru a crea impresia unei imagini continue.

### 5.8.2 Simularea intensităţii pe monitoarele monocrome

Se consideră că alb şi negru sunt cele două niveluri de intensitate de care se dispune la nivelul unui pixel pe `display`.

Tehnica folosită este cea a *texturilor constante*. In principiu, se asociază fiecărui nivel de intensitate o textură cu o densitate corespunzătoare de puncte deschise. Simularea nivelurilor de intensitate luminoasă prin texturi de tip constant se bazează pe proprietatea ochiului uman de a acţiona ca integrator spaţial (la distanţe mari faţă de obiectul privit sau atunci când zona observată are dimensiuni mici). Metoda reduce evident din rezoluţie, căci un element de textură va fi reprezentat pe o arie din ecran de câţiva pixeli.

Tehnica aproximării nivelurilor de intensitate luminoasă prin densitate de puncte albe şi negre pe o anumită suprafaţă elementară (`half-toning`) este folosită în tipărirea fotografiilor alb-negru. Fiecare element de imagine are o finete precizată prin numărul de niveluri de intensitate care pot fi simulate.

Cel mai simplu element de imagine este o arie de $2 \times 2$ pixeli pe un ecran monocrom, pe care se pot produce 5 niveluri diferite de intensitate (figura 5.76.a). In general, o arie de $n \times n$ pixeli poate reproduce $n^2 + 1$ niveluri de intensitate. Pentru a folosi arii mai mari de $4 \times 4$ pixeli (peste 17 niveluri de intensitate) este necesar un monitor cu o rezoluţie mai mare de $512 \times 512$ pentru a obţine rezultate satisfăcătoare. Peste 32 de niveluri pe pixeli fac ca tranziţiile să fie abia percetibile. Imaginile realizate prin discretizarea intensităţii luminoase în 64 de clase au practic acelaşi aspect cu originalul.

Pentru o matrice $3 \times 3$ de pixeli se recomandă setul din figura 5.76 (b). Funcţie de ordinea în care se aprind pixelii se poate construi o matrice caracteristică. In cazul dat,

$$D = \begin{pmatrix} 6 & 8 & 4 \\ 1 & 0 & 3 \\ 5 & 2 & 7 \end{pmatrix}.$$

Pentru a aprinde o porţiune de pixeli cu intensitatea $I$ între 0 şi 9 se aprind pixelii a căror valoare corespunzătoare în matrice este mai mică decât $I$. Dacă imaginea are dimensiunea matricii-rastru, problema aprinderii unui pixel $(x, y)$ depinde de intensitatea dorită $I(x, y)$ în acel punct şi de matricea $D$. Fie $n$ dimensiunea acesteia din urmă. Se calculează $i = x \bmod n$ şi $j = y \bmod n$. Atunci, dacă $I(x, y) > D_{ij}$, pixelul $(x, y)$ se aprinde.

In construirea setului de texturi (`pattern`-uri) trebuie să se ţină seama de anumite reguli:

1. Texturile trebuie astfel proiectate încât să nu producă efecte vizuale artificiale în arii mari de intensitate identică. De exemplu, texturile din figura 5.77.a pot produce senzaţia de haşurare.
2. Texturile trebuie să constituie o secvenţă crescătoare, astfel încât orice pixel aprins pentru nivelul de intensitate $j$ să rămână aprins şi pentru nivelurile de intensitate $k > j$. Se minimizează astfel diferenţele dintre texturi succesive şi efectele de contur.
3. Texturile trebuie să crească de la centru înspre exterior pentru a crea efectul de creştere a mărimii punctului.
4. Pentru uşurarea trasării imaginilor pe imprimante, pixelii aprinşi trebuie să fie asociaţi cu pixeli vecini aprinşi. De exemplu, texturile din figura 5.77.b nu pot fi acceptate.

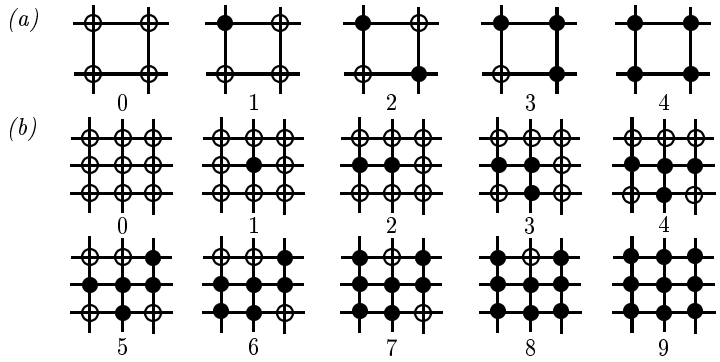


Figura 5.76: Simularea intensităţii – şabloane pentru tehnica `half-toning` cu (a) cinci niveluri de intensitate (b) nouă niveluri de intensitate

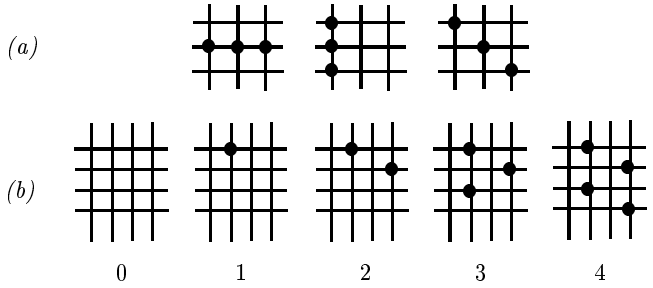


Figura 5.77: Texturi contraindicate

Tehnica `half-toning` este utilizată şi la `display`-uri binivel. Considerând un `display` cu doi biţi per pixel şi deci patru niveluri de intensitate, prin înjumătăţirea rezoluţiei se creşte numărul nivelurilor de intensitate la 13 (se consideră texturi de dimensiune $2 \times 2$: avem patru pixeli la dispoziţie, fiecare având posibilitatea de a lua trei valori în afara negrului, adică se obţin $4 \times 3+1=13$ niveluri de intensitate).

Utilizând un `display` color cu trei biţi per pixel, câte unul pentru fiecare culoare (roşu, verde, albastru) se pot utiliza texturi $2 \times 2$ pentru a obţine 125 culori diferite: fiecare textură poate afişa cinci intensităţi pentru roşu, verde sau albastru, rezultând $5^3$ combinaţii de culori.

### 5.8.3 Lumina cromatică

Pentru descrierea *luminii cromatice* se utilizează trei măsuri:

1. *nuanţa*, care face distincţie între culorile fundamentale;
2. *saturaţia*: se referă la cât de departe este culoarea faţă de un gri de intentitate egală. Culorile pastelate sunt relativ nesaturate. Culorile nesaturate conţin mai multă lumină albă decât culorile vii, saturate De exemplu, roşu este saturat, roz este nesaturat relativ la roşu;

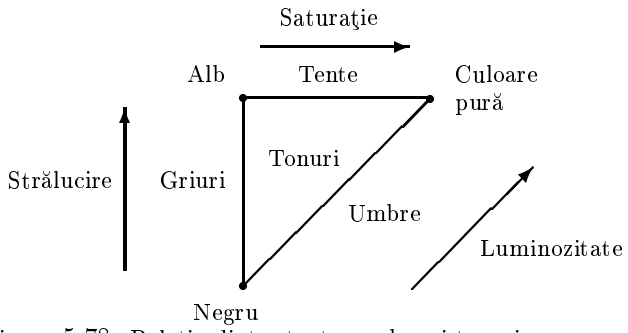


Figura 5.78: Relaţia dintre tente, umbre şi tonuri

3. *luminozitatea* include noţiunea acromatică de percepere a intensităţii unui obiect reflectant.

*Strălucirea* este utilizată pentru a referi intensitatea percepută de obiectele care emit lumină.

Culoarea poate fi specificată prin *tente, tonuri şi umbre*. O tentă rezultă prin adăugarea unui pigment alb la un pigment de culoare pură, descrescând saturaţia. O umbră se obţine adăugând un pigment negru la culoarea pură, descrescând luminozitatea. Un ton este rezultatul adăugării atât a pigmentului negru cât şi a unuia alb la pigmentul pur. In figura 5.78 se prezintă relaţia dintre tente, umbre şi tonuri.

### 5.8.4 Modele de culoare

Un model de culoare constă într-o specificare a unui *sistem de coordonate tridimensionale* şi a unui *domeniu vizibil* în sistem în care toate culorile particulare unei game pot fi descrise.

Modelele orientate hardware cele mai utilizate sunt modelul `RGB` pentru monitoare color, `YIQ` pentru sisteme TV color şi `CMY` pentru anumite printere color. Din păcate aceste sisteme nu permit exprimarea directă a noţiunilor de nunaţă, saturaţie şi strălucire. Pentru înlăturarea acestui inconvenient au fost construite o serie de alta modele precum `HSV` (sau `HSB`), `HLS` şi `HVC`.

#### Modelul `RGB` (`Red-Green-Blue`)

Sistemul `RGB` utilizează sistemul de coordonate cartezian, iar ca domeniu, un cub. Vârfurile sistemului de coordonate carteziene au semnificaţiile din figura 5.79.a. Diagonala principală reprezintă nivelurile de gri. Acest model este utilizat la `CRT` color. Este un sistem de tip aditiv (contribuţii individuale sunt adunate pentru a obţine rezultatul). Gama de culori reprezentată depinde de dispozitiv.

#### Modelul `CMY` (`Cyan-Magenta-Yellow`)

Este specificat printr-un cub asemănător şi este utilizat la anumite imprimante color.

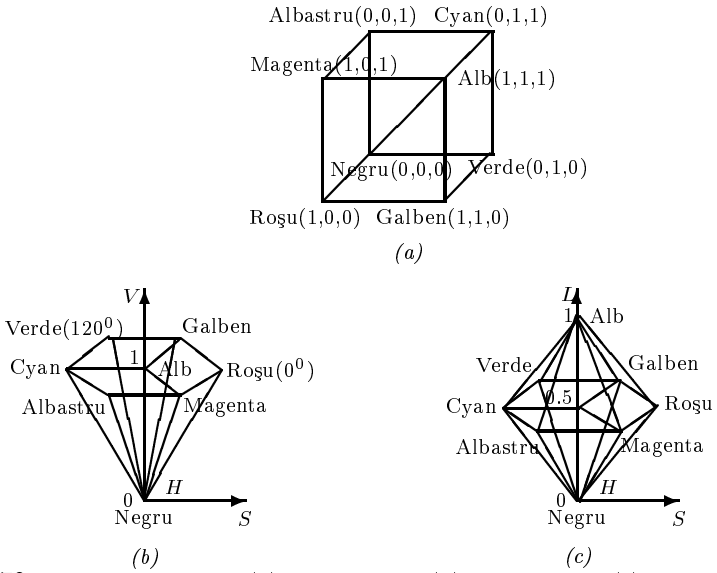



Figura 5.79: Modele de culoare (a) Modelul `RGB` (b) Modelul `HSB` (c) Modelul `HLS`

Sistemul `CMY` este un sistem substractiv (se folosesc filtre pentru a substrage culoarea din lumina albă). Se utilizează acelaşi sistem de coordonate şi domeniu ca în cazul modelului RGB, însă în acest caz alb-ul este în originea sistemului. Culorile sunt specificate prin ceea ce este înlăturat sau sustras din lumina albă (contrar adăugării la negru de la sistemul `RGB`). De exemplu, galbenul absoarbe albastru din lumina albă reflectată (care este sumă de roşu, verde şi albastru). Relaţia dintre modelul `CMY` şi `RGB` este exprimată prin schimbarea coordonatelor $(c, m, y) = (1, 1, 1) - (r, g, b)$. Un alt model, `CMYK`, utilizează negrul (notat `K`) ca a patra culoare; acest sistem este utilizat la anumite dispozitive de copiere.

### Modelul `YIQ`

Sistemul `YIQ` este utilizat în transmiterea televiziunii color. Componenta `Y` se releră la luminozitate şi este singura comnponentă recunoscută de televizoarele alb-negru; cromacitatea este codată în componentele `I` şi `Q`. Se utilizează tot sistemul cartezian de coordonate şi domeniul este definit de un poliedru convex care poate fi transformat într-un cub. Matricea de transformare din `RGB` în `YIQ` are elemente nenule şi distincte pentru a rezolva o problemă des întâlnită în televiziune: două culori diferite aşezate alăturat pe un monitor color apar diferite, dar pe un monitor monocrom, pot părea uneori identice – dacă cele două culori au valori `Y` diferite în modelul `YIQ` problema este evitată.

### Modelul `HSB` şi modelul `HLS`

Modelul `HSB` (`Hue-Saturation-Brightness`, nuanţă-saturaţie-strălucire) este bazat pe intuiţia artistică pentru tente, umbre şi tonuri (figura 5.78). Sistemul de coordonate este cilindric şi subsetul din spaţiu în care modelul este definit

este o piramidă cu şase feţe (figura 5.79.b). Nuanţa $H$ este unghiul din jurul axei verticale, pornind de la roşu considerat $0^0$. Culorile complementare formează un unghi de $180^0$. Saturaţia $S$ se măsoară de la axa verticală la laturile hexagonului. Valorile intermediare ale strălucirii $V$ de la 0 la 1, pentru $S = 0$, corespund nivelurilor de gri. Adăugarea pigmentului alb corespunde descreşterii lui $S$ fără schimbarea lui $V$. Umbrele sunt create păstrând $S = 1$ şi descrescând $V$. Tonurile sunt create descrescând atât $S$ cât şi $V$.

Hexagonul de bază a modelului corespunde cu proiecţia modelului `RGB` de-a lungul diagonalei principale privit de la alb la negru. Orice subcub a modelului `RGB` corespunde unui hexagon dintr-un plan cu $V$ constant. Se poate realiza astfel o corespondenţă între modelele `HSB` şi `RGB`, diagonala principală din modelul `RGB` definind axa $V$ din modelul `HSB`.

Modelul `HLS` (`Hue-Lightness-Saturation`, nuanţă-luminozitate-saturaţie) este definit prin corpul din figura 5.79.c. Poate fi privit ca o deformare a modelului `HSB`.

### 5.8.5 Metoda tabelului de culori

Intr-un sistem cu rastru ce permite afişarea mai multor culori, numărul de biţi necesari pentru reprezentarea unui pixel în frame buffer depinde de:

- numărul de culori disponibile,
- metoda de stocare a valorilor-culoare.

*Tabelele de culoare* pot fi structurate pentru a permite utilizatorului folosirea unor combinaţii de culori fără a cere o dimensiune mare a zonei tampon. Un exemplu concludent este tabelul 5.1.

| Cod culoare | Roşu | Verde | Albastru | Culoare afişată |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Negru |
| 1 | 0 | 0 | 1 | Albastru |
| 2 | 0 | 1 | 0 | Verde |
| 3 | 0 | 1 | 1 | Cyan |
| 4 | 1 | 0 | 0 | Roşu |
| 5 | 1 | 0 | 1 | Magenta |
| 6 | 1 | 1 | 0 | Galben |
| 7 | 1 | 1 | 1 | Alb |

Tabelul 5.1: Exemplu de tabel de culori

Când o culoare particulară este specificată într-un program aplicativ, valoarea binară corespunzătoare este stocată în `frame buffer` pentru fiecare pixel ce va fi afişat în acea culoare.

O altă metodă pentru stocarea codurilor de culoare a pixelilor (ce necesită o zonă tampon mai redusă ca dimensiune decât pentru metoda anterioară) este cea a *tabelului de celule-culori* (`color lookup table, lut`). Valorile din zona tampon cadru reprezintă indici în tabelul de celule-culori. Presupunem că se rezervă câte 4 biţi pentru informaţiile de intensitate (adică avem 16 niveluri de intensitate) pentru fiecare dintre cele trei tunuri de electroni (roşu, verde, albastru). Spre deosebire de cazul tabelului 5.1, fiecare intrare în tabel utilizează 12 biţi pentru specificarea unei culori, deci un total de $16^3$=4096 culori diferite

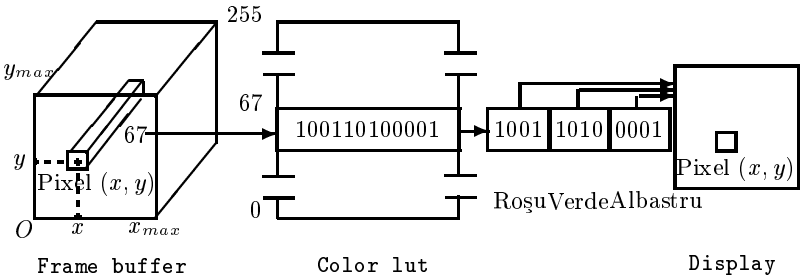



Figura 5.80: Tabelul de celule-culori pentru un sistem cu 8 biţi per pixel

| Intensitate | Valorile în frame buffer | Culoarea afişată |
|---|---|---|
| 0.0 | 0 (00) | Negru |
| 0.33 | 1 (01) | Gri închis |
| 0.67 | 2 (10) | Gri deschis |
| 1.0 | 3 (11) | Alb |

Tabelul 5.2: Codificarea nivelurilor de gri

sunt disponibile. O referinţă de 6 biţi din `frame-buffer` (limitare datorată dimensiunii memoriei video) permite accesarea a $2^6$=64 de poziţii în tabelul de celule.

Sistemele ce utilizează această tehnică permit utilizatorului să selecteze orice combinaţie de 64 de culori dintr-o paletă de 4096 de culori. Intrările în tabelul de celule pot fi schimbate oricând pentru a permite noi combinaţii de culori (schimbarea paletei de culori).

In figura 5.80 este schiţată organizarea tabelului de celule pentru o referinţă de 8 biţi. Pixelul de coordonate $(x, y)$ ce are valoarea 67 în `frame buffer` este afişat pe ecran cu o culoare ce corespunde tunului roşu de electroni la 9/15 din maxim, cel verde la 10/15, iar cel albastru la 1/15.

Utilizarea tabelului de celule poate creşte puternic numărul opţiunilor de culori fără creşterea mărimii imaginii în memorie.

Sistemele cu culori de calitate înaltă utilizează 24 de biţi pentru fiecare poziţie în tabelul de celule şi 9 biţi per pixel în frame buffer. Astfel se permite selectarea de către utilizator a 512 culori dintr-o gamă de aproximativ 16 milioane de culori-intrări ale tabelului de celule.

Pentru monitoarele care nu au capabilităţi privind culoarea, comanda de selectare a culorii poate fi utilizată în programele aplicative pentru a seta nivelul de intensitate, sau scala de gri. Numeroase aplicaţii utilizează valori numerice între 0 şi 1 pentru a seta nivelurile scalei de gri. Un exemplu este prezentat în tabelul 5.2.

Stocarea nivelurilor de intensitate este similară cu stocarea codurilor de culori. Dacă se alocă un bit pentru un pixel, sunt posibile doar două valori din scala de gri (negru şi alb). Dacă se alocă trei biţi per pixel, se permit 8 nivele diferite de intensitate. Sistemele cu calitate înaltă a imaginii permit 24 sau mai mulţi biţi pentru stabilirea nivelului de intensitate per pixel.

## 5.9 Animaţie

Animaţia, arta de a da viaţă unor imagini, presupune nu numai mişcare, ci şi o serie de efecte vizuale. Tehnica animaţiei constă în variaţia în timp a poziţiei, umbre, culori, transparenţă, texturi, schimbări de lumină şi puncte de vedere, apropiere şi îndepărtare (`zoom`).

*Animaţia presupune o reprezentare a corpurilor din 4D.*

Dacă anumite obiecte sau aspecte din imaginile animate se schimbă prea repede relativ la numărul de imagini afişate într-o secundă, apare fenomenul de clipire (`temporal aliasing`). Animaţia imaginilor video presupune o schimbare de cel puţin 30 de imagini per secundă (`fps = frame per second`).

### Animaţie convenţională

Se realizează pe baza standardului următor: se concepe povestea, se înregistrează sunetele, se desenează imaginile cheie (`key frames`), se construiesc imaginile intermediare (`inbetweening`). Rezultatul este filmul de test (`pencil test`) care ulterior este colorat. Acest model de construcţie a imaginilor animate poartă denumirea de *animaţie pe bază de imagini cheie* (`key-frame animation`). Modelul se aplică şi în animaţia pe calculator.

Anumite etape ale animaţiei convenţionale se pretează la asistenţă computerizată, în special fazele de construire a imaginilor intermediare şi colorare. Inainte de aceste două faze, imaginile cheie sunt digitizate (prin scanare sau trasare cu ajutorul unor echipamente grafice specifice) şi sunt postprocesate (filtrate pentru eliminarea zgomotelor şi netezirea contururilor). In faza de compunere a imaginilor, în care figurile principale şi imaginile de fond sunt combinate pentru a genera imagini distincte, se pot utiliza de asemenea tehnici automate.

### Interpolare

In procesul de creare a imaginilor intermediare apar o serie de probleme. Calculatorul primeşte o poziţie iniţială şi una finală. *Animaţia bazată pe imagini cheie* dezvoltă secvenţe animate continue prin interpolarea atributelor scenei. Imaginile cheie trebuie să aibă un nivel înalt de coerenţă a imaginii, altfel imaginile intermediare nu vor sugera o mişcare reală.

Ochiul uman percepe circumstanţele în care are loc interpolarea: de exemplu, dacă este vorba de o aruncare la un anumit unghi a unei mingi (figura 5.81) sau o alunecare pe o suprafaţă. Interpolarea liniară (adecvată celei de a doua situaţii), care este uşor de realizat, are numeroase limitări. Date valorile $v_i$ şi $v_f$ ale unor atribute (poziţie, culoare, mărime) din imaginea iniţială, respectiv cea finală, valoarea $v_t$ a unei imagini intermediare este $v_t = (1-t)v_i + tv_f$, cu variabila temporală $t \in [0,1]$. Valoarea $v_t$ variază neted de la $v_i$ la $v_f$. Dându-se trei imagini-cheie în aruncarea unei mingi sub anumit unghi, respectiv poziţia iniţială, cea finală şi cea corespunzătoare înălţimii maxime, prin interpolare liniară se creează o mişcare continuă, dar cu o schimbare bruscă a vitezei în punctul de maxim (interpolarea liniară generează discontinuităţi ale derivatelor). Astfel, pentru interpolarea imaginilor cheie consecutive se utilizează adesea funcţiile `spline` (vezi capitolul anterior, secţiunea curbe cubice).

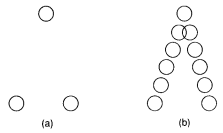



Figura 5.81: Interpolare liniară a mişcării unei mingi (a) Poziţii cheie (b) Interpolare liniară

O figură reprezentată printr-o linie poligonală poate fi interpolată între imaginile cheie prin interpolarea fiecărui vârf al liniei poligonale, de la poziţia sa iniţială la cea finală. Această idee este fundamentală în *tehnica de animaţie pe bază de schelet*, în care obiectelor în mişcare le sunt asociate "scheletele" (acestea sunt interpolate, şi nu obiectele în totalitate). Fie exemplul mişcării unui picior (figura 5.82): scheletul poate fi constituit dintr-o linie poligonală

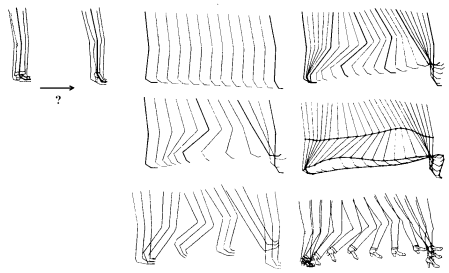



Figura 5.82: Utilizarea tehnicii scheletului

cu patru vârfuri, respectiv vârful tălpii, mijlocul acesteia, centrul genunchiului şi un punct ce defineşte legătura cu corpul. Trasarea a 5-6 polilinii-schelet corespunzătoare, interpolarea cu funcţii `spline` a poziţiilor consecutive ale vârfurilor şi reconstituirea piciorului pe bază de schelet pot conduce la o imagine realistă a efectuării unui pas.

*Animaţia bazată pe deformarea liberă a formelor* utilizează seturi de puncte de control care definesc un volum în spaţiu. Când aceste puncte sunt mutate, orice obiect conţinut este deformat cu o cantitate dependentă de relaţia sa cu punctele setului (figura 5.83). In cazul bidimensional se utilizează o matrice de

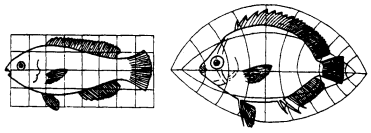


Figura 5.83: Deformarea liberă a formelor

puncte de control în aceeaşi modalitate în care se definesc suprafeţele cubice. Pentru o interpolare cuadratică se folosesc matrice $3 \times 3$, iar pentru interpolare cubică, matrice $4 \times 4$. In figura 5.84 se consideră cazul deformării unei grilei rectangulare definită de 9 puncte de control (interpolare cuadratică). Pentru

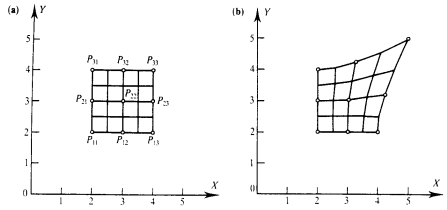


Figura 5.84: Deformarea liberă a formelor

controlul spaţiului bidimensional se utilizează 2 parametrii, $u$ şi $v$, un punct oarecare fiind definit astfel:

$$P'(x', y') = ((1-u)^2, 2u(1-u), u^2) \begin{pmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{pmatrix} \begin{pmatrix} (1-v)^2 \\ 2v(1-v) \\ v^2 \end{pmatrix},$$

unde $u = (x - x_{min})/(x_{max} - x_{min})$ şi $v = (y - y_{min})/(y_{max} - y_{min})$. Dacă nu există deformare $P'(x', y') = P(x, y)$. Generalizarea la 3D este imediată: sunt neecsari 3 parametrii, $u$, $v$, $w$, iar în cazul interpolării cuadratice, 27 puncte de control.

*Animaţia deplasamentelor* consideră mai multe stări ale obiectelor din care se obţin obiectele intermediare care vor deveni parte a secvenţei animate. Ideea de bază este asocierea unor vectori de deplasare (figura 5.85) unor puncte ce definesc suprafaţa unui obiect (fiecare vector presupune două sau mai multe deplasamente).

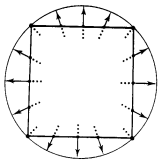


Figura 5.85: Transformarea unui pătrat într-un cerc prin deplasamente

In animaţia unor obiecte tridimensionale trebuie interpolate atât poziţia cât şi orientarea acestora. Poziţia poate fi interpolată ca şi în cazul 2D prin specificarea poziţiei centrului unui obiect în fiecare din imaginile cheie. Problema principală este interpolarea orientării corpurilor. De obicei, o rotaţie este specificată prin trei rotaţii după axele principale. Ordinea aplicării acestor rotaţii de bază nu poate fi aleatoare. Mulţimea tuturor rotaţiilor posibile corespunde unei structuri algebrice: cuaternionii. Rotaţiile sunt *cuaternionii unitari*, care sunt simboluri de forma $a + b\overline{v_1} + c\overline{v_2} + d\overline{v_3}$, unde $a, b, c, d$ sunt numere reale ce satisfac relaţia $a^2 + b^2 + c^2 + d^2 = 1$; cuaternionii se multiplică pe baza legii de distributivitate şi pe baza următoarelor reguli: $\overline{v_1}^2 = \overline{v_2}^2 = \overline{v_3}^2 = -1$, $\overline{v_1 v_2} = \overline{v_3} = -\overline{v_2 v_1}$, $\overline{v_2 v_3} = \overline{v_1} = -\overline{v_3 v_2}$, $\overline{v_3 v_1} = \overline{v_2} = -\overline{v_1 v_3}$. Rotaţia de unghi $\varphi$ în jurul vectorului unitate $(b, c, d)^T$ corespunde cuaternionului $\cos(\varphi/2) + b\sin(\varphi/2)\overline{v_1} + c\sin(\varphi/2)\overline{v_2} + d\sin(\varphi/2)\overline{v_3}$. Efectuarea unor rotaţii succesive presupune multiplicarea cuaternionilor corespunzători. Deoarece $a^2 + b^2 + c^2 + d^2 = 1$, cuaternionii pot fi priviţi ca puncte ale unei sfere în 4D. Pentru interpolarea dintre doi cuaternioni, se poate considera cel mai scurt drum de pe sferă dintre aceştia (arc). Denumirea acestui procedeu este acela de *interpolare sferică liniară*.

### Efecte simple de animaţie

Dacă se construiesc imagini color cu 8 biţi per pixel într-un `frame-buffer` de 640×512, numai o singură imagine conţine 320 Kb de informaţie. Transferarea unei noi imagini înspre `frame buffer` la fiecare 1/30 parte dintr-o secundă necesită o rată de transfer de 9Mb informaţie per secundă, astfel încât realizarea animaţiei este practic imposibilă pentru majoritatea calculatoarelor mici. Pentru evitarea stocării unor asemenea cantităţi de informaţii se recurge la metoda tabelului de celule-culori (`look-up tables` sau `lut animation`) şi compunerea imaginilor prin utilizarea tabelului de culori. De exemplu, scurgerea printr-o conductă poate fi simulată prin ciclarea unei secvenţe de culori din `lut`. Schimbările bruşte de culoare pot fi evitate prin schimbarea gradată a indicelui din lut în mai multe imagini intermediare, de la culoarea iniţială, la cea finală.

In cazul animaţiei unor obiecte de dimensiuni mici pe un fond fix se utilizează tehnica `sprite`-urilor. Un `sprite` este o regiune dreptunghiulară mică care este suprapusă peste `frame-buffer` la nivel video. Locaţia `sprite`-ului la fiecare moment în `frame buffer` este specificată în anumiţi regiştrii, modificarea valorilor acestora cauzând deplasarea `sprite`-ului. `Sprite`-ul poate ascunde valori din `frame-buffer` sau poate fi ascuns de acestea. Se utilizează la implementarea cursoarelor în `frame-buffer` şi generarea animaţiei prin mutarea `sprite`-ului deasupra unei imagini de fond. Mai multe `sprite`-uri presupun definirea unei liste de priorităţi (cine pe cine acoperă).

O aplicaţie a `sprite`-urilor, extrem de populară, este cea a jocurilor video, în care animaţia constă aproape în întregime din `sprite`-uri care se mişcă pe un fundal fix. Deoarece locaţia şi mărimea fiecărui `sprite` este stocat în regiştrii, este uşor de verificat coliziunea dintre `sprite`-uri.

Traiectoriile obiectelor pot fi generate prin urmărirea unui film real. O asemenea tehnică este `rotoscope`: se realizează un film în care oameni sau animale interpretează roluri din scenariul animaţiei, apoi animatorii, pe fondul existent, înlocuiesc personajele reale cu echivalentul lor desenat. Mişcarea personajelor animate poate fi realizată urmărind mişcarea personajelor pe care le înlocuiesc. Această tehnică produce o animaţie cu mişcări realistice. O altă tehnică este cea care asociază o serie de indicatori punctelor cheie ale corpului unei persoane. De exemplu, mici luminiţe pot fi ataşate punctelor cheie din arhitectura umană şi poziţiile acestor luminiţe sunt înregistrate din direcţii diferite pentru a furniza o poziţie 3D pentru fiecare punct cheie în orice moment (tehnica *marionetei grafice*).

### Atenuarea efectelor datorate discretizării temporale

Fenomenul de clipire a imaginii, ce pare în cazul unei mişcări foarte rapide `temporal aliasing`), poate fi parţial rezolvat prin creşterea rezoluţiei temporale. Dacă, de exemplu, se urmăreşte rotirea spiţei unei roţi (figura 5.86), la o viteză de rotaţie mare, adică la un anumit raport cu rata „expunerii", spiţa pare să se învârtească în sens invers, sau chiar să stea pe loc. Fenomenul apare deoarece mişcarea este prea rapidă pentru a fi înregistrată corect.

Imaginile consecutive sunt similare, aşa cum în 2D o linie de scan nu diferă prea mult de următoarea. Se poate ţine seama, pentru simplificare, de o anumită coerenţă în construirea imaginilor. Fiecare obiect descrie o regiune din 4D. De exemplu, o sferă care nu se mişcă descrie în 4D un cilindru. In mod similar, un cerc din 2D ce se deplasează dintr-un colţ al ecranului în cel opus descrie un cilindru în spaţiul 2D+timp. Aplicând divizarea hiperspaţiului 4D şi utilizarea unor tehnici de tip `ray-tracing` se poate economisi timp considerabil în construirea de imagini consecutive.

O altă tehnică pentru `antialiasing` temporal este animaţia pe câmpuri. O imagine video convenţională este trasată în două etape: prima dată liniile scan pare, apoi cele impare. Fiecare linie scan este retrasată, aproximativ, la fiecare 1/30 dintr-o secundă (presupunem că raza tubului de electroni parcurge ecranul de 60 de ori pe secundă). Dacă culorile pixelilor liniilor pare sunt calculate la timpul $t$, iar liniile impare la $t+60^{-1}(s)$, apoi imaginea este compusă într-o singură hartă de biţi, iar acest proces se repetă pentru fiecare imagine,

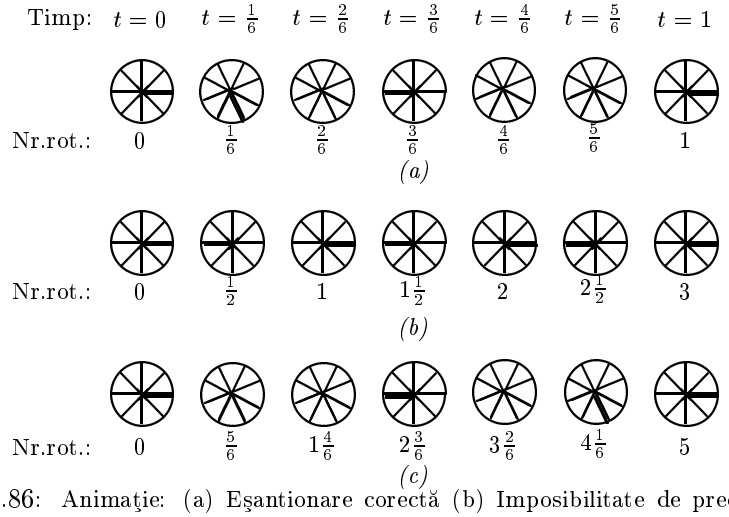



Figura 5.86: Animaţie: (a) Eşantionare corectă (b) Imposibilitate de precizare a direcţiei de înaintare (c) Eşantionare incorectă: impresia deplasării în sens invers

atunci animaţia este realizată cu un efect de 60fps, chiar dacă fiecare line scan este reactualizată la un interval de 1/30 dintr-o secundă. Dezavantajul acestei tehnici constă în faptul că imaginile de pe ecran constau din linii ale unor imagini construite la timp diferiţi şi poate apărea un fenomen de clipire.

### Metode de control a animaţiei

*Controlul explicit* este forma cea mai simplă de control al animaţiei. Animatorul descrie tot ceea ce se realizează în animaţie, specificând atât schimbări simple precum scalarea, translaţia, rotaţia, cât şi informaţiile despre imaginile cheie şi metodele de interpolare. Această interpolare poate fi dată explicit (prin descriere matematică) sau, într-un sistem interactiv, prin directa manipulare a unui `mouse, joystick` sau alt dispozitiv de intrare.

*Animaţia procedurală* introduce proceduri algoritmice pentru a controla atributele unui obiect, sursă de lumină sau observator (de exemplu, pentru animarea unor mecanisme fizice). In cazul unui *control procedural*, elementele distincte ale unui model comunică în scopul determinării proprietăţilor lor. Acest tip de control se utilizează pentru generarea mişcărilor greu de specificat prin control explicit.

Anumite obiecte din lumea fizică se mişcă descriind linii drepte, dar mişcările majorităţi obiectelor sunt determinate de alte obiecte cu care vin în contact, astfel încât mişcările rezultate nu mai sunt liniare. Mişcările acestora pot fi descrise prin constrângeri. Animaţia se poate realiza doar ţinând seama de aceste constrângeri, prin *control bazat pe constrângeri* (*animaţie comportamentală*). Se generează părţile unui ansamblu (linii, cercuri, etc), apoi se definesc constrângerile de tip punct ("linia are un capăt fix"), legătură ("linii unite la un capăt"), unghiuri ("linii paralele"). Mişcarea ansamblului se realizează respectând regulile de constrângere.

Utilizarea *actorilor* este o formă înaltă de control procedural. Un actor, în

animaţie, este un program mic, invocat odată pentru o imagine, pentru determinarea caracteristicilor unui obiect (actorul este un obiect în sensul programării orientate obiect). Un actor, poate trimite mesaje altor actori pentru a controla comportarea lor.

### Limbaje de animaţie

Limbajele de animaţie pot fi clasificate astfe: (a) limbaje bazate pe notaţii tip listă liniară, (b) limbaje cu scop general care înglobează şi directive de animaţie, (c) limbaje grafice.

Intr-un *limbaj bazate pe notaţii tip listă liniară*, fiecare eveniment din animaţie este descris printr-un număr de imagine iniţială şi unul de imagine finală, precum şi o acţiune care are loc. Acţiunile presupun anumiţi parametrii ( de exemplu, "roteşte obiectul Casa între imaginile 20 şi 34 în jurul axei 1 cu 45 de grade, determinând creşterea rotaţiei la fiecare imagine din tabelul C".

*Limbajele generale ce includ directive de animaţie* se bazează pe generarea procedurală a obiectelor şi a animaţiei. Valorile variabilelor limbajului pot fi utilizate ca parametrii pentru rutinele ce generează animaţie, astfel încât asemenea limbaje sunt indicate pentru simulări. Entităţile primitive ale unui asemenea limbaj includ vectori, culori, poligoane, colecţii de poligoane (solide), colecţii de obiecte (grupuri), puncte de vedere, lumini, volum de vedere. Obiectele pot fi transformate geometric (exemple: deplasare sus, jos, stânga, dreapta, înainte, înapoi, mărire, micşorare). O asemenea transformare primeşte ca argument un obiect şi returnează o copie transformată a obiectului.

Problema principală a limbajelor textuale este dificultatea animatorului de a-şi imagina rezultatul execuţiei unui şir de comenzi. In locul unor descrieri explicite ale acţiunii, în *limbajele grafice* de animaţie, animatorul construieşte o imagine a acţiunii. Un prim pas a fost realizat prin introducerea curbelor $P$. O curbă $P$ este o *reprezentare parametrică* a mişcării (sau a unui alt atribut) a unui obiect sau ansamblu de obiecte din interiorul unei scene. Animatorul descrie mişcarea unui obiect prin specificarea grafică a coordonatelor sale ca funcţie de timp.

# 6. Interfeţe utilizator

Deşi nu există o definiţie unanim acceptată şi general valabilă pentru *interfaţa utilizator*, se poate spune că aceasta reprezintă *ansamblul dispozitivelor logice* (de interacţiune între utilizator şi aplicaţie) *şi a sarcinilor* (`task`-uri) *elementare de interacţiune* (implementate cu diverse tehnici de interacţiune) *care asigură introducerea interactivă a datelor de intrare, controlul interactiv al execuţiei programului şi obţinerea rezultatelor intermediare sau finale.*

## 6.1 Elemente de bază ale interfeţelor utilizator

In contextul general al graficii interactive — şi în particular, al interfeţelor grafice — avantajele şi dezavantajele dispozitivelor de interacţiune sunt abordate la unul dintre următoarele trei nivele:

- dispozitiv
- `task`
- dialog

La nivelul dispozitivelor implicate, se analizează caracteristicile hardware ale acestora fără a se trata aspectele utilizării prin intermediul software-ului. De exemplu, la acest nivel se analizează ergonomia formei unui mouse prin comparaţie cu a altuia, posibilitatea organizării eficiente a spaţiului (pe birou!) în cazul utilizării unui dispozitiv în locul altuia, etc.

La nivelul task (sarcină) se compară tehnicile de interacţiune implicate în realizarea aceleiaşi sarcini utilizând diverse dispozitive. De exemplu, în urma unei astfel de analize se poate concluziona că un utilizator experimentat poate fi mai eficient utilizând tastatura decât selecţia din meniuri cu un mouse sau, că utilizarea unui mouse este de preferat pentru selectarea unui obiect afişat pe ecran decât prin utilizarea tastelor de deplasare a cursorului, etc.

La nivelul dialogului se analizează modul de înlănţuire în secvenţe a sarcinilor de interacţiune sub aspectul eficienţei, ergonomiei şi fiabilităţii. De exemplu, deşi poziţionarea cursorului este mai rapidă cu un mouse, dacă mână utilizatorului este deja pe tastatură şi sarcina care urmează poziţionării presupune utilizarea tastaturii, atunci utilizarea tastelor de deplasare ar putea fi mai convenabilă si, deci, trebuie oferită ca alternativă.

## 6.2 Dispozitive de intrare

Dispozitivele de intrare pot fi analizate din două puncte de vedere: fizic şi logic. *Dispozitivele fizice de intrare* se referă la acele dispozitive `hardware` prin care utilizatorul introduce informaţii într-un sistem de calcul. *Dispozitivele logice de intrare* reprezintă tipurile de interacţiune posibile într-o aplicaţie.

Dezvoltarea unor aplicaţii de avangardă a exercitat presiuni asupra proiectanţilor de dispozitive care au oferit noi tipuri de dispozitive. Pe de altă

parte (`feedback`!), noile dispozitive fizice disponibile reprezintă o provocare pentru proiectanţii de aplicaţii pentru a dezvolta noi tehnici de interacţiune.

Dispozitive specializate de intrare mai puţin comune sunt:

(a) dispozitive pentru recunoaşterea vocii;
(b) diverse tipuri de tablete grafice, sensibile la presiune sau orientare;
(c) dispozitive de interacţiune tridimensională.

Astfel, unele dintre dispozitivele de interacţiune 2D au fost extinse pentru a permite interacţiuni 3D (de exemplu: `joystick`-ul, `trackball`-ul, `spaceball`-ul, `data glove`, etc. Astfel de dispozitive se utilizeaza frecvent în aplicaţii din domeniul *realităţii virtuale* (`Virtual Reality`).

In cele ce urmează ne vom limita la discutarea aspectelor legate de dispozitivele logice. Dispozitive logice de bază sunt:

1. locatorul,
2. tastatura,
3. valuatorul,
4. selectorul.

### 6.2.1 Locatorul

Locatorul este un dispozitiv utilizat pentru a indica o poziţie şi/sau o orientare în spaţiul de lucru.

Dispozitivele fizice utilizabile ca locator pot fi clasificate funcţie de trei caracteristici independente:

1. după sistemul de coordonate propriu:
   (a) *dispozitivele absolute* (de exemplu, digitizoarele) au o origine şi precizează poziţiile în raport cu acea origine;
   (b) *dispozitivele relative* (de exemplu `mouse, trackball, joystick`) nu au o origine absolută, ele precizând doar modificări faţă de poziţia anterioară;
2. după modul de interacţiune:
   (a) *dispozitivele directe* (de exemplu, `lightpen` sau `touch screen`) la care poziţia curentă este indicată prin poziţionarea dispozitivului pe ecran;
   (b) *dispozitivele indirecte* (de exemplu, `mouse` sau `joystick`) la care mutarea cursorului (poziţiei curente) pe ecran se face fără a atinge ecranul. Astfel de dispozitive presupun realizarea unei coordonări între ochiul şi mâna utilizatorului.
3. după tipul poziţionării:
   (a) *dispozitivele continue* (de exemplu, `joystick` sau `mouse`) sunt dispozitive care transformă o deplasare lină şi continuă a mâini într-o deplasare similară a cursorului pe ecran.
   (b) *dispozitivele discrete* (de exemplu, săgeţile de pe tastaură) permit deplasarea cursorului doar pe direcţii prestabilite, cu incrementi de deplasare prestabilită, eventual în poziţii prestabilite.

*Observaţie.* Viteza de deplasare a cursorului la dispozitivele continue este afectată de rata $C/D$ (`Control - to - Display`), adică de viteza de deplasare a mâini raportată la deplasarea cursorului.

### 6.2.2 Tastatura

Reprezintă unul dintre dispozitivele de intrare cele mai comune. In principal, este utilizat pentru introducerea şirurilor de caractere. Poate fi utilizată însă şi pentru a suplini locatorul (poziţionare), valuatorul (precizarea unei valori) sau selectorul (alegerea unei variante din mai multe posibile).

Cel mai des întâlnit tip de tastatură este tastatura `Qwerty`. Există însă şi alte tipuri de tastaturi, cum ar fi tastatura `Dvorak`, la care literele frecvent utilizate sunt plasate în poziţii de bază ale degetelor, sau tastatura pentru stenodactilografie.

### 6.2.3 Valuatorul

Valuatorul este un dispozitiv logic care permite introducerea unui număr ce reprezintă o valoare numerică a unui parametru.

Din punct de vedere fizic, valuatoarele pot fi de tipul potenţiometrelor radio, rotative sau liniare. Poziţia potenţiometrului faţă de origine precizează valoarea numerică.

Valuatorul poate fi cu plajă fixă de valori, valorile obţinute de la un asemenea dispozitiv fiind absolute (de exemplu, butoane gen potenţiometru de volum), sau cu plajă nemărginită de valori, valorile obţinute fiind relative (potenţiometrii fără poziţie maximă şi minimă).

### 6.2.4 Selectorul

Selectorul sau întrerupătorul este un dispozitiv logic care permite selecţia unei variante din mai multe alternative posibile (listă de stări sau acţiuni posibile).

Dispozitivele fizice tipice pentru selectare sunt tastele funcţionale dar şi dispozitive specializate de tip întrerupător (de exemplu, butoanele `mouse`-ului sau a `trackball`-ului).

## 6.3 Sarcini de interacţiune

O *sarcină de interacţiune* (tip de interacţiune, unitate de interacţiune, `interaction task`) reprezintă unitatea de informaţie semnificativă în contextul aplicaţiei (interactive) pe care utilizatorul o poate produce cu ajutorul unui dispozitiv de interacţiune.

*Observaţie.* O sarcină de interacţiune diferă de un dispozitiv logic de intrare prin aceea că defineşte *ce* realizează utilizatorul, în timp ce dispozitivele logice precizează *cum* se realizează o anumită interacţiune. Sarcinile de interactiune sunt centrate utilizator, în timp ce dispozitivele logice de intrare reprezintă un concept orientat către programator şi pachetele grafice.

Se pune întrebarea: cât de mică sau cât de mare este o astfel de unitate de informaţie? De exemplu, deplasarea unui dispozitiv de poziţionare pe o anumită distanţă, determină o unitate de informaţie dacă noua poziţie semnifică repoziţionarea unui obiect sau specificarea unui capăt de segment. In schimb mutarea cursorului peste un element de meniu nu reprezintă o unitate de informaţie.

Sarcinile de interacţiune pot fi clasificate astfel:

1. *sarcini de interacţiune de bază* (`Basic Interaction Task`, prescurtate `BIT`), care sunt unităţi de informaţie indivizibile (pe baza acestei definiţii şi analogiei cu chimia, le putem numi *atomi de interacţiune*);
2. *sarcini de interacţiune compuse* (`Composite Interaction Task`, prescurtate `CIT`), care se obţin prin agregarea (compunerea) mai multor `BIT` (*molecule de interacţiune*).

O *tehnică de interacţiune* reprezintă modul în care se realizează o sarcină de interacţiune. De exemplu, o selecţie poate fi realizată printr-un "clic" cu `mouse`-ul pe un element de meniu, prin introducerea numelui (identificatorului) elementului de selectat de la tastatură, prin apăsarea unei taste funcţionale asociată cu varianta, sau utilizând un dispozitiv de recunoaştere a vocii. Un alt exemplu este cel al unui dispozitiv de intrare folosit în mai multe tipuri de interacţiune.

### 6.3.1 Sarcini de interacţiune de bază

Un set complet de `BIT`-uri pentru grafica interactivă este constituit din:

(a) poziţionare,
(b) selecţie,
(c) introducere de text,
(d) cuantificare (introducere de valori numerice).

Fiecare dintre aceste sarcini de interacţiune de bază poate fi realizat prin mai multe tehnici de interacţiune.

#### Poziţionarea

Poziţionarea necesită specificarea unui set de coordonate bidimensionale sau tridimensionale. Tehnica folosită de obicei pentru poziţionare implică fie mutarea cursorului pe ecran în poziţia dorită şi apoi apăsarea unui buton, fie precizarea coordonatelor prin intermediul unei tastaturi reale sau virtuale.

Indiferent de tehnica utilizată, apar o serie de probleme generale pentru toate aceste tehnici:

1. sistemele de coordonate,
2. rezoluţia,
3. grila,
4. `feedback`-ul,
5. timpul de acomodare.

**Sisteme de coordonate.** Sistemul de coordonate devine o problemă importantă atunci când, în cursul poziţionării, se primeşte reacţia de la sistemul asupra căruia se acţionează.

De exemplu, dacă un dispozitiv de tip locator este mutat spre dreapta pentru a "trage" un obiect, apare problema direcţiei de mutare a obiectului. Există cel puţin trei posibilităţi: obiectul se mută

(a) în sensul pozitiv al axei $x$ din sistemul de coordonate al ecranului (`DCS`);

(b) în sensul pozitiv al axei $x$ din sistemul de coordonate ale lumii înconjurător (`WCS`);

(c) în sensul pozitiv al axei $x$ din sistemul de coordonate propriu al obiectului.

Prima soluţie este cea mai viabilă întrucât respectă *principiul compatibilităţii stimul – reacţie*, care afirmă că răspunsul sau reacţia sistemului la acţiunea utilizatorului trebuie să fie în aceeaşi direcţie şi/sau cu aceeaşi orientare, iar magnitudinea răspunsului trebuie să fie proporţională cu acţiunea.

**Rezoluţia.** Rezoluţia necesară pentru poziţionare poate varia de la aplicaţie la aplicaţie. Dacă poziţia se introduce numeric (de la tastatură) rezoluţia poate fi oricât de mare. Dacă poziţionarea se face prin tehnici de mutare a cursorului (utilizând un locator) apar limitări de ordin fizic. In mod obişnuit, rezoluţia dispozitivelor de tip `mouse`, tabletă grafică etc, este de la a 500-a până la a 2000-a parte din unitatea de rezoluţie a dispozitivului de afişare.

Prin utilizarea transformărilor de tip `window - to - viewport`, de exemplu, pentru a mări o regiune din `WCS`, este posibil să se realizeze o corespondenţă între unitatea de rezoluţie a ecranului şi o unitate de rezoluţie a sistemului de coordonate ale lumii înconjurătoare, arbitrar de mică, corespondenţă care, de fapt, corespund unei măriri de rezoluţie.

**Grila.** Un ajutor important în realizarea sarcinii de poziţionare îl poate oferi o grilă (`grid`, reţea rectangulară) suprapusă peste aria de lucru pentru a uşura alinierea poziţiilor sau obiectelor.

De asemenea, grila poate fi utilă pentru a forţa ca punctele de capăt ale primitivelor grafice să cadă pe grilă (prin rotunjirea coordonatelor locatorului la coordonatele celui mai apropiat nod al grilei).

**Feedback.** Există două tipuri de poziţionare: spaţială şi lingvistică. In poziţionarea spaţială, utilizatorul cunoaşte poziţia dorită prin raport cu elementele spaţiale din apropiere. In poziţionarea lingvistică, utilizatorul cunoaşte valoarea numerică a coordonatelor poziţiei dorite. `Feedback`-ul reprezintă marcarea poziţiei curente prin cursor sau prin coordonatele numerice.

**Timpul de acomodare.** Din punct de vedere al factorului uman se pune problema reducerii timpului în care se ajunge la o bună coordonare ochi – mână în procesul de poziţionare. Un caz specific de poziţionare, care presupune un timp de acomodare mai lung, îl reprezintă poziţionarea continuă (o succesiune de poziţii care definesc o curbă).

### Selecţia

Selecţia, ca tip de interacţiune, are scopul de a alege un element dintr-un set de alternative. In mod tipic, seturile de alternative constau din

1. comenzi
2. seturi de valori ale unor atribute
3. clase de obiecte
4. instanţe de obiecte

De exemplu meniul `line style` (dintr-un program de desenat) constă dintr-un set de valori pentru atributul `line style`, iar meniul `object type` (linie, cerc, dreptunghi, text, etc.) este un set de clase de obiecte. Pe de altă parte, mulţimea tuturor obiectelor din suprafaţa de desen constituie un set de instanţe de obiecte.

Anumite *tehnici de interacţiune* pot fi folosite pentru a selecta din oricare din aceste seturi de alternative. Alte tehnici, însă, sunt utile numai pentru selecţii din anumite clase. De exemplu, prin poziţionarea pe reprezentarea vizuală a unui element dintr-un set de alternative se poate face o selecţie indiferent de tipul setului, pe când selecţia dintr-un set de instanţe de obiecte nu se poate face prin utilizarea tastelor funcţionale.

Ceea ce diferenţiază tehnicile de interacţiune care se pot utiliza pentru selecţie este caracterul fix sau variabil al numărului de elemente ale setului de alternative.

Din acest punct de vedere, un *set de alternative* poate fi clasificat ca fiind

- cu dimensiunea (relativ) fixă
- cu dimensiunea variabilă

In prima clasă intră seturile de comenzi, atribute, clase de obiecte, care în mod obişnuit au un număr fix de elemente, fără a se exclude însă posibilitatea lărgirii sau reducerii acestuia (nu foarte des şi nu foarte mult!). In a doua clasă, setul de alternative poate fi relativ mare şi se poate schimba frecvent (la fiecare nouă actualizare).

**Selecţia din seturi de alternative cu dimensiune variabilă.** Două tehnici sunt considerate ca fiind potrivite pentru acest tip de interacţiune: *numirea* (`naming`) şi *indicarea* (`pointing`).

**Selecţia obiectelor prin numire.** In cazul acestei tehnici, utilizatorul tastează numele obiectului pe care vrea să-l selecteze. Evident, există situaţii când această tehnică nu este eficientă (de exemplu, când utilizatorul nu cunoaşte numele obiectelor din diverse motive: sunt prea multe, nu apar explicit pe suprafaţa grafică, etc.). Dacă însă utilizatorul cunoaşte numele obiectelor din setul de alternative, selecţia prin numire poate fi mai eficientă decât selecţia prin indicare (de exemplu, această tehnică permite selecţii multiple prin utilizarea de `wildcards`).

Dacă se apelează la selecţia prin numire este indicat ca după fiecare tastă apăsată să se afişeze, ca `feedback`, lista obiectelor al căror nume se potriveşte cu numele parţial introdus. In momentul în care această listă conţine un singur element, numele parţial introdus poate fi completat de către program în mod automat (`auto completion`).

**Selecţia prin indicare.** Această tehnică presupune poziţionarea cursorului pe obiectul dorit şi apoi selecţia propriu zisă, fie cu ajutorul unei taste, fie cu un clic pe un buton de `mouse`. Şi în cazul acestei tehnici pot apare probleme atunci când obiectele sunt dispuse pe mai multe nivele (se acoperă unele pe altele). Soluţia constă de regulă în precizarea nivelului (`layer`) pe care se face selecţia. Exemple: `AutoCAD`, Meniuri ierarhice, etc (figura 6.1).

**Selecţia din seturi de alternative cu dimensiune (relativ) fixă.** In cazul acestui tip de interacţiune, tehnica cea mai frecvent utilizată o constituie

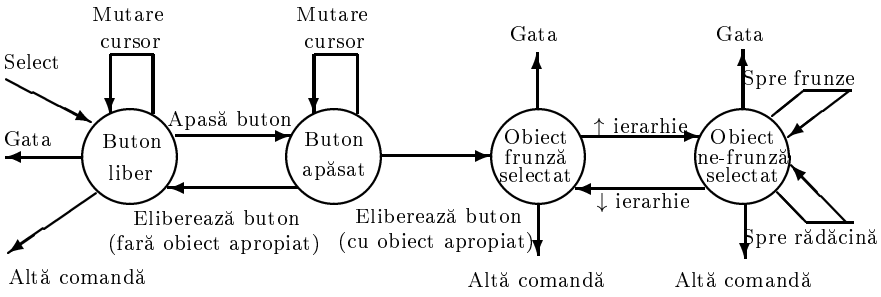



Figura 6.1: Diagrama de stare a tehnicii de selecţie prin indicare a unui obiect dintr-o ierarhie de obiecte

selecţia din meniuri. Problematica proiectării meniurilor include următoarele:

- ordinea elementelor dintr-un meniu;
- meniuri cu un singur nivel sau meniuri multinivel;
- plasarea meniurilor;
- reprezentare vizuală;
- dimensiunea şi forma meniurilor;
- taste funcţionale.

**Ordinea elementelor dintr-un meniu.** Elementele de meniu pot fi organizate după diverse criterii de ordonare:

1. alfabetică,
2. grupare logică după scopul funcţional,
3. după frecvenţa utilizării (cele mai frecvent utilizate la început),
4. în ordinea importanţei (cele mai importante la început).

Modul în care se ordonează elementele meniurilor trebuie să fie consistent în toate meniurile aplicaţiei.

**Meniuri cu un singur nivel sau meniuri multinivel.** In cazul în care numărul elementelor de meniu este prea mare pentru a fi afişate simultan, meniul respectiv poate fi divizat într-o ierarhie structurată logic sau într-o secvenţă liniară de elemente care se afişează succesiv (utilizând `scroll bars`).

In cazul meniurilor ierarhice utilizatorul face o selecţie din setul de alternative de la nivelul cel mai de sus, ceea ce îi pune la dispoziţie un alt set de alternative de pe nivelul imediat inferior. *Ierarhiile de meniuri* pot fi prezentate în diverse moduri:

1. cu reacoperire
2. în cascadă
3. panouri de ierarhie

**Plasarea meniurilor.** Meniurile pot fi statice şi vizibile permanent, sau pot apare dinamic, la cerere (`pop-up, pull-down`, etc.).

Pentru anumite aplicaţii, mai ales cele care presupun existenţa unui `display` specializat pentru "obiectul" propriu-zis, se poate plasa meniul pe un al doilea monitor.

*Meniurile statice* sunt indicate în cazul utilizării unui `display` (fereastră) doar pentru afişarea meniurilor şi precizarea comenzilor.

*Meniurile* `pop-up` apar pe ecran pentru a oferi posibilitatea unei selecţii fie:

(a) ca urmare a unei acţiuni explicite a utilizatorului (apăsarea unei taste funcţionale, selecţia unei icoane, etc);

(b) automat, când desfăşurarea dialogului (utilizator–aplicaţie) presupune o selecţie.

Observaţii:

1. Meniul `pop-up` apare de obicei în poziţia curentă a cursorului (spre care e focalizată atenţia utilizatorului).
2. La apariţia meniului poate fi prezentată ca selecţie curentă (care trebuie confirmată):
   (a) alternativa implicită (cea mai frecvent folosită, cea mai "generală", etc);
   (b) ultima alternativă utilizată.
3. Meniurile `pop-up` pot fi dependente de context (sensibile la context).

*Meniurile* `pull-down` (`pulled-out`) sunt ancorate într-o bază de meniu (care conţine "titlul" meniurilor `pull-down`). Activarea acestor meniuri poate fi:

- explicită (selecţie făcută cu cursorul pe titlu);
- implicită (meniul "apare" când cursorul ajunge deasupra titlului).

**Reprezentarea vizuală.** Problema principală este dacă meniul va conţine nume ale elementelor din setul de alternative sau reprezentări grafice (icoane) ale acestora. Reprezentarea textuală (prin nume) a alternativelor are avantajul clarităţii dar conferă un aspect "încărcat" meniului, în timp ce reprezentarea grafică (prin icoane) poate mări randamentul utilizării meniului dacă icoanele sunt sugetive. Uneori se dublează alternativele textuale cu icoane care vor fi din ce în ce mai des folosite, pe măsură ce utilizatorul se acomodează cu ele (exemplu: meniul din `MS Word`).

**Selecţia curentă.** In cele mai multe situaţii este de preferat ca elementul curent selectat din meniu să fie marcat pentru a se deosebi într-un fel sau altul de celelalte elemente. Selecţia curentă poate fi marcată în diferite moduri:

1. afişare în video invers sau cu altă culoare,
2. marcare cu diverse semne grafice,
3. afişare sub titlul meniului.

**Dimensiunea şi forma meniurilor.** La proiectarea meniurilor trebuie realizat un compromis între dimensiunea mai mare a elementelor de meniu — care permite o selecţie mai uşoară — şi dimensiunea mai mică — care are avantajul că ocupă mai puţin spaţiu din suprafaţa utilă a ecranului.

**Taste funcţionale.** In cazul în care numărul elementelor de meniu este relativ mic, se pot asocia unora sau tuturor elementelor câte o tastă funcţională ceea ce permite o selecţie mult mai rapidă (în cazul unui utilizator versat).

### Interacţiunea de tip text

Sarcina de interacţiune text permite introducerea de şiruri de caractere cărora aplicaţia nu le asociază o semnificaţie aparte. De exemplu introducerea numelui unei comenzi nu intră în această categorie de interacţiune, spre deosebire

de etichetarea obiectelor unui desen sau introducerea de text în cadrul unui editor de texte, care sunt interacţiuni de tip text. In mod obişnuit tehnica de interacţiune pentru introducerea de text presupune utilizarea tastaturii.

O alternativă la folosirea tastaturii pentru introducerea textului o reprezintă utilizarea unui scanner. In principiu, această tehnică presupune utilizarea ulterioară (după scanarea textului) a unei componente de recunoaştere optică a caracterelor (`Optical Character Recognition - OCR`) care are sarcina de a transforma `bitmap`-ul obţinut după scanare în succesiunea de coduri `ASCII` corespunzătoare textului din imagine. Tehnica `OCR` apelează intens la algoritmi de recunoaştere a formelor pentru identificarea caracterelor din imaginea cu text. Problemele care apar la utilizarea acestei tehnici de introducerea textului includ:

1. îmbunătăţirea imaginii scanate;
2. eliminarea fondului;
3. decuparea unităţilor de recunoscut (litere);
4. scalare;
5. orientare;
6. poziţionare.

O altă alternativă, relativ recent apărută, o reprezintă utilizarea dispozitivelor de recunoaşterea vocii, prin intermediul cărora textul vorbit este transformat în succesiune de coduri `ASCII`.

#### Cuantificarea

Acest tip de interacţiune presupune specificarea unei valori numerice, de regulă între o valoare minimă şi una maximă.

Tehnicile de interacţiune utilizate includ:

1. tastarea valorii;
2. poziţionarea unui "potenţiometru";
3. modificarea unui contor pentru incrementare/decrementare.

### 6.3.2 Sarcini de interacţiune compuse

Tipurile de interacţiune compozite (`CIT`) sunt combinaţii de tipuri de interacţiune de bază. Există trei tipuri de bază de `CIT`:

1. celule de dialog (`dialog boxes`),
2. tehnici de construcţie,
3. manipulare dinamică.

#### Celule de dialog

De multă ori este necesară o selecţie multiplă dintr-un set de alternative. De exemplu, atribute de text, ca *italic*, **bold**, underline, etc. nu se exclud reciproc şi utilizatorul ar dori să selecteze mai multe simultan. In plus pot să existe mai multe seturi de alternative care trebuie precizate simultan. De exemplu, tipul obiectului de desenat, grosimea liniei, aspectul liniei fac parte din seturi de alternative diferite care se pot preciza într-un pas de selecţie, anterior acţiunii de desenare a obiectului respectiv.

Tipurile de meniuri din care se poate face o singură selecţie la un moment dat (`pull-down, pop-up`) nu sunt satisfăcătoare pentru selecţii multiple fiindcă dispar după o selecţie făcută, fiind necesară activarea lor pentru o a doua selecţie. Această problemă poate fi depăşită prin utilizarea celulelor de dialog care reprezintă o formă de meniu ce rămâne vizibilă până când este părăsită în mod explicit (cu acceptarea selecţiilor făcute sau cu anularea lor). Selecţiile făcute într-o celulă de dialog pot fi corectate pe loc. De asemenea, atributele sau valorile specificate într-o celulă de dialog pot fi aplicate imediat, pentru ca utilizatorul să vadă efectul selecţiei făcute înainte de a părăsi celula de dialog.

### Tehnici de construcţie

Un mod de a desena o linie impune utilizatorului să precizeze punctul de start şi punctul de sosire, după care linia este desenată. Cu această tehnică însă, utilizatorul nu are posibilitatea de a vedea rezultatul înainte ca acţiunea să rămână definitivă.

O tehnică superioară constă în utilizarea benzilor elastice (`rubberband`). După precizarea poziţiei de start, orice nouă poziţie a cursorului este interpretată ca punct de sosire temporar şi linia este desenată corespunzător. De abia în momentul selectării punctului de sosire definitiv linia devine permanentă. Starea de bandă elastică este activă atâta timp cât un buton al dispozitivului de poziţionare şi de selecţie este apăsat. In această poziţie deplasarea cursorului provoacă deplasarea liniei. Pornind de la această tehnică de desenare a liniilor se derivează tehnica desenării dreptunghiului elastic, cercului elastic, elipsei elastice.

O altă tehnică de construcţie priveşte desenarea de linii frânte care presupune o succesiune de selecţii (ale capetelor segmentelor care alcătuiesc linia poligonală) intercalate cu poziţionări (care precizează poziţia capetelor).

In oricare dintre aceste tehnici pot fi aplicate asupra poziţiei cursorului anumite constrângeri. De exemplu, cursorul poate fi obligat să se deplaseze pe anumite direcţii sau la distanţe limitate de poziţia de start.

Un alt tip de constrângere îl reprezintă utilizarea unui, aşa numit, câmp de gravitate, pentru a uşura suprapunerea a doua poziţii: în momentul în care cursorul intră în câmpul de gravitate al unui punct de capăt al unui obiect existent pe suprafaţa de desen, poziţia care va fi selectată va coincide cu (va fi atrasă spre) acest punct.

### Manipulare dinamică

In multe situaţii nu este suficientă desenarea de obiecte geometrice, ci este necesară şi modificarea acestora. Modificările posibile includ schimbarea poziţiei, rotirea şi scalarea.

Schimbarea poziţiei se realizează prin tehnica de `click-and-drag`, care implementează succesiunea buton apăsat-deplasare-buton eliberat, corespunzând selecţiei obiectului, deplasării în noua poziţie şi selecţiei noii poziţii.

Tehnici similare se utilizează pentru rotirea sau scalarea unui obiect şi care presupun selecţia obiectului, precizarea noii orientări (prin definirea unghiului de rotaţie) sau a noii dimensiuni (prin definirea factorul de scală) şi, apoi, selecţia noii situaţii.

Deplasarea, rotirea sau scalarea afectează un obiect în întregime. Uneori, este de dorit însă mutarea unui punct individual dintr-un obiect. Această operaţie se poate realiza de asemenea prin tehnica `click-and-drag`.

## 6.4 Principii în proiectarea interfeţelor utilizator

Pentru a asigura calitatea unei interfeţe din punct de vedere utilizator trebuie avute în vedere o serie de principii de proiectare:

1. consistenţă,
2. asigurarea `feedback`-ului,
3. minimizarea posibilităţilor de eroare (la utilizare),
4. asigurarea posibilităţii de revenire din eroare,
5. posibilitatea utilizării pe nivele de îndemânare,
6. reducerea necesităţii de memorizare.

### 6.4.1 Consistenţa

Un *sistem este consistent* dacă modelul conceptual, funcţionalitatea, secvenţialitatea şi legătura cu dispozitivele `hard` sunt uniforme şi respectă câteva reguli simple şi deci nu prezintă excepţii şi condiţii speciale.

Principalul scop urmărit prin consistenţa unui sistem este de a permite utilizatorului să *generalizeze* cunoştinţe despre un aspect al sistemului la alte aspecte.

De asemenea consistenţa permite evitarea frustărilor care apar când un sistem nu se comportă într-un mod logic.

*Metoda principală* de a asigura consistenţa este *proiectarea* `top-down` *a sistemului.*

*Exemple de consistenţă:*

La ieşiri,

(a) se vor utiliza întotdeauna aceleaşi codificări (de exemplu roşu → stop, verde → porneşte);
(b) mesajele despre starea sistemului apar întotdeauna într-o poziţie logică fixată;
(c) elementele de meniu îşi vor păstra întotdeauna aceiaşi poziţie relativă în meniu.

La intrări,

(a) tastele (`CR, TAB, BS`, etc) au întotdeauna aceleaşi funcţiuni;
(b) comenzile globale (`Help, Status, Cancel`) pot fi inversate oricând;
(c) comenzile generice (`Move, Copy, Delete`) sunt disponibile şi pot fi aplicate, cu rezultate previzibile, asupra oricărui tip de obiect din sistem.

### 6.4.2 Asigurarea feedbackului

Are ca scop permanenta informare a utilizatorului asupra stării sistemului, asupra efectelor acţiunilor întreprinse şi asupra acţiunilor posibile ce urmează a fi întreprinse.

Reacţia sistemului se poate asigura pe trei nivele, corespunzătoare nivelelor de proiectare:

- *semantic:* funcţională,
- *sintactic:* a secvenţializării,
- *lexical:* a legăturii cu dispozitivele (`hard`) fizice.

Nivelul cel mai de jos este cel `hardware`: fiecare acţiune a utilizatorului cu un dispozitiv de intrare ar trebui să provoace o reacţie imediată şi evidentă (de ex. caracterele tastate vor fi afişate pe ecran, mutarea `mouse`-ului este fidel urmată de mişcarea cursorului).

La nivel sintactic acceptarea/neacceptarea fiecărei unităţi (cuvânt) a limbajului de intrare (comandă, poziţie, obiect, etc), trebuie semnalată de sistem. De exemplu, un obiect selectat sau un element de meniu selectat va fi "iluminat", astfel încât utilizatorul să ştie că acţiunea sa a fost acceptată.

La nivel funcţional, o formă de `feedback` o constituie notificarea faptului că acţiunea comandată este în curs de efectuare (dacă durează mai mult de câteva secunde). Altă formă de `feedback` la nivel funcţional presupune anunţarea terminării operaţiei fie printr-un mesaj, fie prin afişarea rezultatelor.

Trebuie făcută o distincţie între `feedback`-ul din domeniul problemei şi respectiv cel din domeniul controlului. `Feedback`-ul în domeniul problemei se referă la obiectele de manipulat: existenţa, poziţia, apariţia. `Feedback`-ul în domeniul controlului se referă la mecanisme de control ale sistemului interactiv: stare, valori curente şi implicite, meniuri, etc.

### 6.4.3 Minimizarea posibilităţilor de eroare

Ideea este aceea de a respecta următoarea regulă: "Nu pregăti capcane pentru utilizator!"

*Exemple:*

1. Nu oferi posibilitatea alegerii unui element de meniu care va genera mesaje de genul: selecţie ilegală, comandă invalidă, etc.!
2. Nu lăsa utilizatorul să selecteze `Move, Copy, Delete` când nimic nu este selectat!
3. Nu lăsa utilizatorul să selecteze `Paste` dacă `clipboard`-ul este gol!

In general, "oferta" care se face utilizatorului pentru a selecta din ea trebuie să fie senzitivă la context: sistemul ghidează utilizatorul în funcţie de contextul respectiv, făcând imposibilă selectarea comenzilor nepermise.

### 6.4.4 Asigurarea posibilităţii de revenire din eroare

De regulă patru tipuri de corectare a erorilor sunt avute în vedere la proiectarea interfeţelor utilizator: `Undo, Abort, Cancel, Correct`.

Dacă s-a lansat o operaţie din greşeală, este recomandat `Undo`. Acesta se poate asigna pe un nivel sau pe mai multe nivele (caz în care se păstrează

comenzile şi starea sau parametri într-o stivă). In acest context (`Undo` multi-nivel) este utilă o comandă `Redo`.

In cursul execuţiei operaţiei în cauză, dacă utilizatorul îşi dă seama că a comis o eroare, trebuie să i se asigure posibilitatea întreruperii operaţiei şi revenirea la starea anterioară selecţiei respectivei operaţii printr-o comandă `Abort`.

`Undo` şi `Abort` pot fi comasate într-o singură comandă.

Dacă utilizatorul decide, în cursul specificării parametrilor pentru o operaţie, că a greşit, el poate renunţa prin comanda `Cancel` la terminarea specificării şi lansării operaţiei.

De asemenea, dacă nu operaţia, ci doar anumiţi parametri sunt eronaţi, utilizatorul poate dispune de facilitatea de a corecta ceea ce a greşit.

### 6.4.5 Posibilitatea operării pe mai multe nivele

Diverse aplicaţii trebuie proiectate astfel încât să permită utilizarea lor pe diverse nivele de complexitate: de la nivelul unui utilizator fără experienţă, care de abia învaţă comenzile de bază şi are nevoie de sprijin din partea sistemului, până la nivelul unui utilizator cu experienţă, care este dispus să renunţe la anumite informaţii ajutătoare în schimbul unei viteze sporite de lucru. Utilizatorul cu experienţă trebuie să poată face uz de tehnici de interacţiune mai rapide.

Metodele utilizate pentru a permite utilizarea interfeţei pe mai multe nivele de îndemânare sunt:

1. acceleratori (taste funcţionale în loc de meniu),
2. prompteri (ghidarea acţiunii următoare),
3. `help`,
4. extensibilitate (posibilitatea combinării unor comenzi elementare într-una mai complexă),
5. ascunderea complexităţii (set minimal de comenzi).

### 6.4.6 Minimizarea memorizării

Reducerea necesităţii de memorizare a comenzilor disponibile în sistem se realizează de obicei prin apelul la capacitatea utilizatorului de a recunoaşte (intui) semnificaţia comenzilor reprezentate fie prin numele lor fie printr-o icoană. Pe de altă parte, necesitatea memorizării comenzilor poate fi redusă prin asigurarea unor servicii de tip `Help` ca cele menţionate mai sus.

## 6.5 Software pentru interfeţe utilizator

Software pentru interfeţe (grafice) utilizator există pe diverse nivele (figura 6.2).

Dispozitivele logice de intrare (locator, valuator, `pick, choice, string, stroke - GKS, PHIGS`) pot opera într-unul din modurile:

1. `request;`
2. `sample;`
3. `event.`

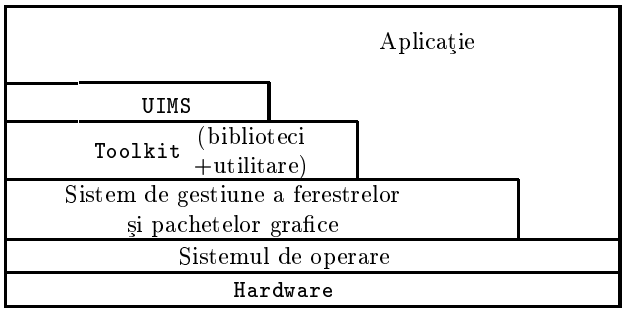



Figura 6.2: Software pentru interfeţe utilizator

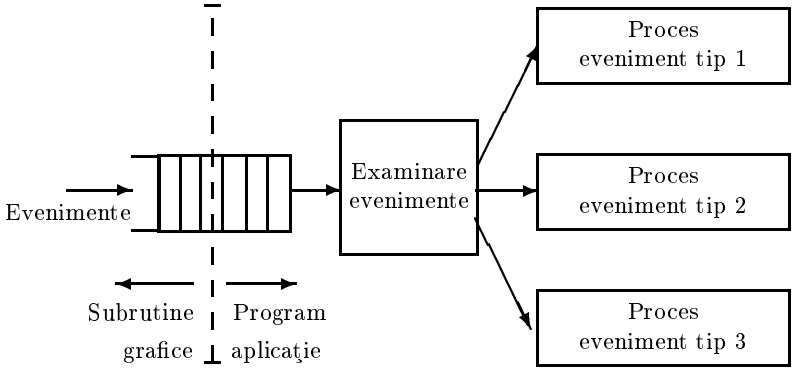



Figura 6.3: Tratarea evenimentele din coadă

Fiecărui dispozitiv îi este asociată o *măsură* (tipul informaţiei returnate de dispozitiv).

Modurile de operare indică modalitatea prin care programul de aplicaţie recepţionează o schimbare a măsurii dispozitivelor lor.

In modul `request` (la cerere) programul de aplicaţie solicită o intrare de la dispozitiv care întoarce odată cu controlul şi măsura asociată acţiunii utilizatorului (de exemplu, deplasare) asupra dispozitivului. Acţiunea se numeşte `trigger`. Dezavantajele sunt următoarele:

(a) intrarea se face în mod sincron;

(b) nu există posibilitatea de `feedback` dinamic pentru că aplicaţia nu recapătă controlul decât după terminarea acţiunii.

In modul `sample` (`polling`) dispozitivele sunt cercetate (baleiate) unul după altul în speranţa surprinderii unei modificări a măsurii. Dezavantajul constă în faptul că, în timp ce se procesează intrarea de la un dispozitiv, se pot pierde intrările de la alte dispozitive.

In modul `event` intrările (acţionările asupra dispozitivelor) sunt asincrone în sensul că orice eveniment asociat cu un dispozitiv generează o intrare într-o coadă de evenimente (figura 6.3) de regulă `FIFO` (se poate avea însă în vedere şi asocierea de priorităţi diferitelor clase de dispozitive).

### 6.5.1 Sisteme de gestiune a ferestrelor

Un sistem de gestiune a ferestrelor (`window-management system`) asigură majoritatea caracteristicilor importante ale interfeţelor utilizator moderne: rularea aplicaţiilor în ferestre distincte pe ecran, abilitatea de a redimensiona şi muta ferestrele, meniuri `pop-up` şi `pull-down`, ferestre de dialog (`dialog boxes`).

Un sistem de gestiune a ferestrelor este, în principal, un gestionar de resurse. El alocă spaţiu pe ecran diverselor programe care încearcă să folosească ecranul şi apoi gestionează utilizarea zonelor de ecran alocate, astfel încât programele să nu interfereze (pe ecran) unele cu altele. De asemenea, sistemul de gestiune a ferestrelor alocă programelor care solicită intrări resursele dispozitivelor de interacţiune şi, apoi, direcţionează informaţia de intrare de la dispozitiv către coada de evenimente a programului destinaţie.

Un sistem de gestiune a ferestrelor are două părţi:

1. gestionarul de ferestre (`window manager`) cu care interacţionează utilizatorul atunci când solicită crearea, redimensionarea, mutarea, deschiderea, închiderea, etc. unei ferestre;
2. sistemul de ferestre (`window system`), care reprezintă componenta funcţională responsabilă cu crearea, redimensionarea, mutarea, deschiderea, închiderea, etc. efectivă a unei ferestre.

Gestionarul de ferestre este construit deasupra sistemului de ferestre şi utilizează serviciile pe care acesta i le pune la dispoziţie, pentru a satisface cererile utilizatorului.

Programele construite deasupra sistemului de ferestre sunt denumite uneori *programe client*, iar sistemul de ferestre este denumit *program server*.

In anumite sisteme de gestiune a ferestrelor construite pe o arhitectură client-server, precum `X-Windows`, gestionarul de ferestre este el însuşi un client al sistemului de ferestre (figura 6.4).

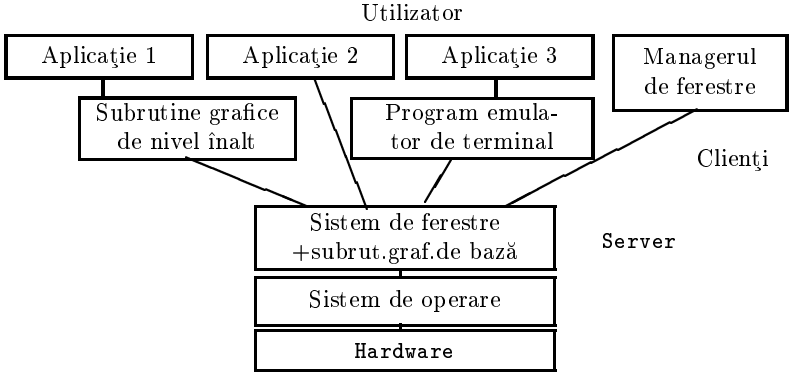



Figura 6.4: Relaţia dintre sistemul de ferestre, sistemul de operare şi programul-aplicaţie

Anumite sisteme de ferestre sunt astfel proiectate încât să suporte mai mulţi manageri de ferestre, fiecare cu aspect şi reacţie (`look and feel`) distincte (`X-Windows`).

*Observaţie.* Managerul de ferestre, şi nu sistemul de ferestre, este cel care determină aspectul unei ferestre şi modul în care utilizatorul interacţionează cu aceasta.

De obicei se integrează un pachet de subrutine grafice în sistemul de ferestre care asigură o parte din funcţionalitatea sistemului de ferestre, dar care pot fi apelate şi direct.

### 6.5.2 Tratarea ieşirilor în sistemele de ferestre

Resursa de ieşire alocată de sistemul de ferestre unui program client este *spaţiul ecran*, care trebuie astfel gestionat încât clienţii să nu interfereze unii cu alţii în utilizarea spaţiului ecran.

Strategiile de alocare a spaţiului ecran pot să varieze în mod considerabil de la un sistem de ferestre la altul, dar, în principiu, se încadrează în trei categorii mari. Principala diferenţă constă în modul de afişare a zonei din fereastră devenită vizibilă în urma măririi ferestrei, în urma descoperirii ferestrei sau în timpul defilării (figura 6.5).

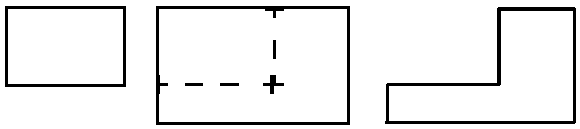



Figura 6.5: Lărgirea ferestrei (a) la dimensiunea ferestrei (b) necesită decuparea primitivelor faţă de regiunea (c)

**Strategii:**

- Un sistem de ferestre *minimal* nu-şi asumă nici o responsabilitate în afişarea suprafeţei nou expuse a ferestrei, ci transmite un eveniment de tipul *fereastră expusă* clientului asociat ferestrei . Un astfel de sistem de ferestre nu salvează partea acoperită a ferestrei. Când un client afişează o informaţie în fereastră, primitivele de ieşire sunt decupate relativ la partea vizibilă a ferestrei. La mărirea suprafeţei vizibile a ferestrei, clientul asociat ferestrei recepţionează un mesaj de tip fereastră expusă şi este responsabil cu afişarea primitivelor corespunzătoare zonei proaspăt expuse a ferestrei.
- Sistemele de ferestre care dispun de mai multă memorie salvează partea acoperită a ferestrei, astfel încât clientul este degrevat de sarcina afişării porţiunii proaspăt expuse. Problema care se pune este cât din fereastra acoperită trebuie salvat. In mod tipic, se salvează dimensiunea maximă posibilă a ferestrei (egală cu dimensiunea ecranului). Anumite sisteme de ferestre salvează chiar porţiuni mai mari decât ecranul, soluţie care, deşi costisitoare, devine din ceea ce mai atractivă datorită tendinţei de scădere a preţului memoriei calculatoarelor. Clientul este implicat în reafişare doar în cazul în care se face o defilare în fereastră, dincolo de porţiunea salvată, sau în cazul unei scalări.

O stategie puţin diferită constă în păstrarea de către managerul de ferestre a unei copii complete pentru fiecare fereastră. De câte ori o parte a unei ferestre este descoperită, porţiunea corespunzătoare din copia ferestrei este afişată pe ecran. Actualizarea ferestrei conform acestei strategii este lentă întrucât clientul asociat poate să scrie în porţiunea acoperită a ferestrei şi este necesară refacerea copiei. Actualizarea unei ferestre complet descoperite se face mai rapid întrucât copia ei trebuie actualizată doar în momentul dinaintea acoperirii ferestrei. O alternativă a acestei strategii constă în partiţionarea fiecărei ferestre în regiuni rectangulare, doar acelea care nu sunt vizibile fiind salvate.

- O strategie diferită este utilizată de sistemele de ferestre care menţin o listă a primitivelor (vizibile sau nu) afişate în fiecare fereastră. La fiecare reafişare a ferestrei, lista asociată este parcursă şi se fac decupările necesare. Această strategie presupune utilizarea unui hardware pentru conversie de scanare rapidă.

O altă problemă care apare frecvent este efectul operaţiei de redimensionare a unei ferestre asupra informaţiei vizibile în fereastră. Există două posibilităţi şi programul client ar trebui să fie capabil să le trateze pe amândouă.

In primul caz, în momentul redimensionării ferestrei de către utilizator, fereastra `WCS` îşi modifică dimensiunea în mod corespunzător, utilizatorul "văzând" mai mult sau mai puţin din "lume", în funcţie de operaţia de redimensionare (mărire sau micşorare) efectuată (figura 6.6.c).

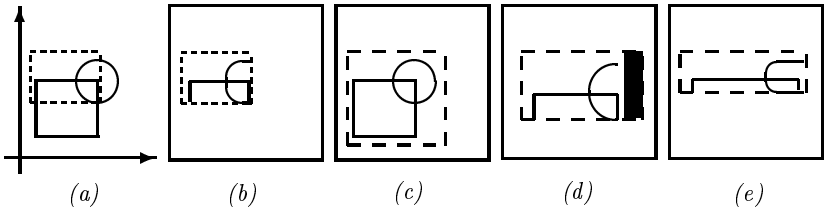



Figura 6.6: Relaţiile dintre fereastra `WCS` şi fereastra managerului (de ferestre) (a) Imagine în `WCS` (b) Imaginea printr-o fereastră (c) Lărgirea ferestrei în `WCS` (d) Lărgirea ferestrei managerului cu scalare uniformă (e) Lărgirea ferestrei managerului cu scalare neuniformă

In al doilea caz (figura 6.6.d), dimensiunea ferestrei `WCS` rămâne fixă astfel încât prin mărirea ferestrei se vede aceeaşi scenă, dar la o scară mai mare. In acest caz se mai pune şi problema scalării neuniforme pe cele două axe (figura 6.6.e) .

Anumite sisteme de ferestre permit crearea de ferestre ierarhice, adică ferestre care conţin subferestre (figura 6.7). Acestea pot fi utilizate de exemplu la implementarea ferestrelor de dialog: întreaga fereastră de dialog este definită ca o fereastră şi fiecare câmp, buton, `scroll-bar`, este definit ca o subfereastră şi evenimente de tipul buton - `mouse` apăsat sunt disponibile pentru fiecare subfereastră. Aceasta inseamnă că fiecare eveniment este însoţit de numele subferestrei în care a apărut.

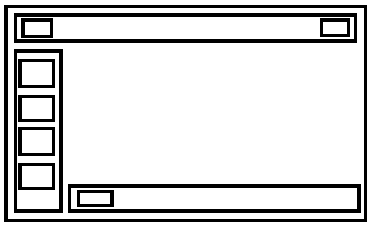


Figura 6.7: Divizarea unei ferestre în ferestre ierarhice

In mod tipic un sistem de ferestre ar trebui să dispună de următoarele funcţiuni:

1. crează o fereastră nouă (care devine fereastră curentă);
2. setează poziţia ferestrei curente;
3. setează dimensiunea ferestrei curente;
4. selectează o fereastră (care devine fereastră curentă);
5. plasează o fereastră invizibilă deasupra tuturor celorlalte (aceasta este noua fereastră curentă);
6. ascunde fereastra curentă şi expune toate ferestrele acoperite de aceasta;
7. setează titlul ferestrei curente;
8. citeşte poziţia ferestrei curente;
9. citeşte dimensiunea ferestrei curente;
10. plasează fereastra curentă deasupra tuturor celorlalte;
11. plasează fereastra curentă sub toate celelalte;
12. şterge fereastra.

Pe lângă spaţiul ecran, sistemul de ferestre este responsabil şi de alocarea unei alte resurse: intrări ale tabelei de culori (`look-up` table). In principiu, două strategii de alocare a intrărilor în tabela de culori sunt utilizate:

- se partajează intrările între toţi clienţii, alocându-se fiecăruia un număr fix. Dezavantajele metodei sunt:
  (a) în cazul în care sunt puţini clienţi, rămân intrări neutilizate în tabele;
  (b) pot exista clienţi cărora li se vor aloca mai puţine intrări decât au nevoie, ceea ce implică degradarea calităţii modului de afişare al ferestrei (din punct de vedere al culorilor).
- se alocă toate intrările necesare clientului a cărui fereastră conţine cursorul. Dezavantajul constă în faptul că aspectul coloristic al ferestrelor se poate modifica dramatic (chiar dacă pentru scurt timp!), în momentul trecerii cu cursorul dintr-o fereastră într-alta.

O altă soluţie posibilă constă în alocarea de culori în loc de intrări în `look-up` table. Toţi clienţii care utilizează aceleaşi culori vor lucra cu aceeaşi indecşi în tabela de culori. In cazul în care un client solicită o culoare pentru care nu există intrare în tabelă, şi aceasta nu este plină, se creează o nouă intrare. Dacă tabela este plină, atunci se va folosi indexul culorii celei mai apropiate de cea solicitată. Dezavantajul evident este acela că cea mai apropiată culoare ar putea fi inacceptabil de departe de cea solicitată.

### 6.5.3 Tratarea intrărilor în sistemele de ferestre

Resursa de intrare alocată şi gestionată de către sistemul de ferestre pentru clienţii săi reprezintă setul de dispozitive de intrare şi evenimentele generate de acestea. Sistemul de ferestre trebuie să dispună de informaţiile necesare pentru a dirija evenimentele spre clientul căruia îi sunt destinate.

Procesul de dirijare a evenimentelor către clientul adecvat poartă denumirea de *multiplexare*.

Pe lângă evenimentele generate de diverse dispozitive, un sistem de ferestre poate lua în considerare şi generarea unor evenimente speciale, ca, de exemplu, *intrarea într-o nouă fereastră (*`window enter`), respectiv *ieşirea din fereastra curentă (*`window leave`), care permit evidenţierea (`highlighting`) ferestrei care conţine cursorul.

Un alt tip de eveniment special – `window damage` – ar putea fi utilizat pentru a preciza sistemului de ferestre că fereastra clientului căruia îi este destinat evenimentul trebuie reafişată.

In cazul în care există ferestre ierarhice, evenimentele pot fi dirijate şi către subferestre.

Două modalităţi sunt utilizate, în principal, de către sistemele de ferestre pentru dirijarea evenimentelor către clienţi:

1. evenimentul este dirijat către clientul a cărui fereastră conţine cursorul (`real-estate-based event routing`);
2. evenimentul este dirijat către un client precizat de către un alt client, posibil, dar nu obligatoriu, acelaşi (`listener event routing`). De exemplu, managerul de ferestre poate avea o comandă prin care utilizatorul poate dirija intrările de la tastatură către clientul care posedă o anumită fereastră.

## 6.6 Utilitare pentru tehnici de interacţiune

Utilitarele pentru tehnici de interacţiune (`interaction-technique toolkits`) sunt subrutine care implementează legătura cu `hardware`-ul în cadrul unei interfeţe utilizator. Acestea determină caracteristica interfeţelor utilizator descrisă de termenul `look and feel"` (aspect şi reacţie).

Tehnicile de interacţiune pot fi implementate direct în aplicaţie, dar această soluţie are două dezavantaje majore:

1. este consumatoare de timp;
2. nu răspunde necesităţii utilizatorului de a întâlni interfeţe similare la aplicaţii distincte.

Utilitarele pentru tehnici de interacţiune, de fapt, colecţii de subrutine de bibliotecă accesibile programelor de aplicaţie, înlătură ambele dezavantaje menţionate.

Utilizarea aceleiaşi colecţii în toate aplicaţiile şi chiar în sistemul de ferestre reprezintă o metodă des folosită în scopul unificării aspectului interfeţelor.

Modalităţile de implementare sunt următoarele: peste sistemul de gestiune a ferestrelor, sau, în lipsa lui, peste pachetul de subrutine grafice.

*Exemple:* colecţia de subrutine care însoţeşte sistemul de gestiune a fere-

strelor `Andrew, OSF/Motil, InterViews,` implementare `OpenLook`.

In sistemul de gestiune a ferestrelor `X-Windows` tehnicile de interacţiune sunt denumite `widgets (WIndows gaDGETS)` şi o listă tipică de astfel de `widgets` include:

- fereastră de dialog,
- fereastră de selecţie fişier,
- fereastră de avertizare,
- fereastră de ajutor (`help`),
- fereastră de mesaje,
- butoane radio,
- banc de butoane radio,
- butoane de selecţie (marcare),
- bancuri de selecţie,
- butoane basculante,
- bancuri de butoane basculante,
- meniu fix,
- meniu `pop-up`,
- bară de defilare,
- fereastră de introducere text,
- fereastră de aplicaţie.

Fiecare `widget` este implementat ca o fereastră care poate conţine la rândul ei subferestre.

Utilitarele pentru tehnici de interacţiune dispun, în mod tipic, de notificatori (pentru intrare/ieşire în/din ferestre) pentru a permite apelul de proceduri de reîntoarcere atunci când anumite evenimente se produc în subferestrele lor.

In lista de `widgets` de mai sus apar atât elemente de bază cât şi altele compuse. De obicei, colecţiile de subrutine dispun de mijloace de compunere a `widget`-urilor. De exemplu, fereastra de dialog conţine atât butoane radio, cât şi butoane de selecţie binară.

Crearea de ierarhii de ferestre este o activitate migăloasă pentru orice programator (chiar dacă se dispune de o colecţie de `widgets`-uri). De aceea s-au realizat editoare interactive pentru proiectarea (crearea şi modificarea) aspectului ferestrelor de dialog (care, de regulă, conţin celelalte tipuri de widgets). Ieşirea din astfel de editoare este o reprezentare a ferestrei compuse, fie ca structură de date care poate fi translatată în cod sursă, fie sub formă de cod compilat. In toate situaţiile sunt însă prevăzute mecanisme pentru editarea legăturilor cu aplicaţia.

O soluţie alternativă pentru crearea de meniuri şi ferestre de dialog o constituie utilizarea unui limbaj de nivel înalt pentru descrierea acestora.

In fine, proiectantul de interfeţe poate crea un `widget` sau poate compune mai multe `widgets` în mod interaciv, prin exemple (sistemul `Peridot`).

### 6.6.1 Sisteme de gestiune a interfeţelor utilizator

Sistemele de gestiune a interfeţelor utilizator (`User-Interface Management System - UIMS`) asistă utilizatorul în proiectarea şi implementarea aspectului interfeţei dar şi, în anumite cazuri, a semanticii acesteia. Orice `UIMS` oferă

mijloace pentru definirea secvenţelor de acţiuni utilizator admisibile, pentru specificarea aspectului interfeţei şi a conţinutului mesajelor de help si eroare. Un UIMS poate mări considerabil productivitatea programatorului, facilitând în acelaşi timp rafinarea iterativă a interfeţei pe măsură ce se câştigă experienţă.

Aplicaţiile dezvoltate peste un UIMS constau, în mod tipic, dintr-un set de subrutine, denumite uneori *rutine de acţiune* sau *rutine de acţiune semantică*. UIMS-ul apelează rutina de acţiune adecvată ca răspuns la o intrare (input) produsă de utilizatorul aplicaţiei. In schimb, rutina de acţiune influenţează dialogul – de exemplu, modificând acţiunile utilizator posibile în continuare. Se poate spune că UIMS şi rutinele de acţiune partajează controlul dialogului. Acest model este cunoscut sub denumirea de model cu *control partajat* (shared-control model). Prin opoziţie cu acest model, există UIMS-uri în care rutinele de acţiune nu au nici o influenţă asupra dialogului. Aceste UIMS-uri se încadrează în modelul cu *control extern* (external-control model).

Serviciile specifice pe care diverse UIMS-uri le oferă proiectantului de interfeţe utilizator pot varia dar componenta esenţială, care nu lipseşte din nici un UIMS, o reprezintă specificarea secvenţei dialogului. Această componentă controlează ordinea în care tehnicile de interacţiune sunt puse la dispoziţia utilizatorului aplicaţiei.

### Secvenţializarea dialogului

Secvenţele admisibile de acţiuni ale utilizatorului pot fi definite într-o varietate de moduri: via reţele de tranziţie (numite şi diagrame de stări), reţele de tranziţie recursive, limbaje bazate pe evenimente sau prin exemple, (unde proiectantul demonstrează sistemului succesiunile de acţiuni permise şi sistemul "învaţă" care secvenţe sunt posibile). Comun tuturor acestor metode este conceptul unei stări a interfeţei utilizator şi acţiuni ale utilizatorului asociate care pot fi executate din această stare. Fiecare din metodele de specificare codifică starea interfeţei utilizator într-un mod specific,care generalizează folosirea uneia sau a mai multor variabile de stare. Dacă se crează o interfaţă utilizator senzitivă la context, răspunsul sistemului la acţiunile utilizatorului trebuie să depindă de starea curentă a interfeţei.

Cea mai simplă şi cea mai puţin puternică, dar totuşi utilă, metodă de specificare a succesiunii dialogului o reprezintă *reţeaua de tranziţie* sau *diagrama de stări*. Reţelele de tranziţie au o singură variabilă de stare, un întreg ce indică starea curentă. Acţiunile utilizatorului cauzează tranziţii de la o stare la alta; fiecare tranziţie are asociată cu ea zero sau mai multe rutine de acţiune care sunt apelate când tranziţia se întâmplă. Pe lângă aceasta, stările pot avea rutine de acţiune asociate care sunt executate oricând se intră în această stare. Reţelele de tranziţie sunt utile, în special, pentru găsirea incosistenţelor în secvenţializarea dialogului, şi pot fi folosite fără probleme pentru a determina numărul necesar de paşi pentru a completa o secvenţă de sarcini de interacţiune.

Reţelele de tranziţie au totuşi dezavantaje. In primul rând, starea interfeţei utilizator este, în mod tipic, bazată pe un număr de variabile de stare, şi necesitatea de a găsi toate combinaţiile posibile de valori ale acestor variabile specifice unei stări este dificilă şi non-intuitivă pentru proiectantul interfeţei utilizator. Reţeaua de tranziţie din figura 6.8 descrie o aplicaţie cu următoarele comenzi:

- selectează obiect (stabileşte obiectul curent - CSO);

- deselectează obiect (nu mai există `CSO`!);
- crează obiect (stabileşte un `CSO`);
- şterge `CSO` (nu mai există `CSO`!);
- copiază `CSO` în `clipboard` (trebuie să existe un `CSO`; umple `clipboard`-ul);
- preia (`paste`) din `clipboard` (trebuie sa existe ceva în `clipboard`; crează un `CSO`);
- goleşte `clipboard`-ul (`clipboard`-ul trebuie să conţină ceva şi va rămâne gol).

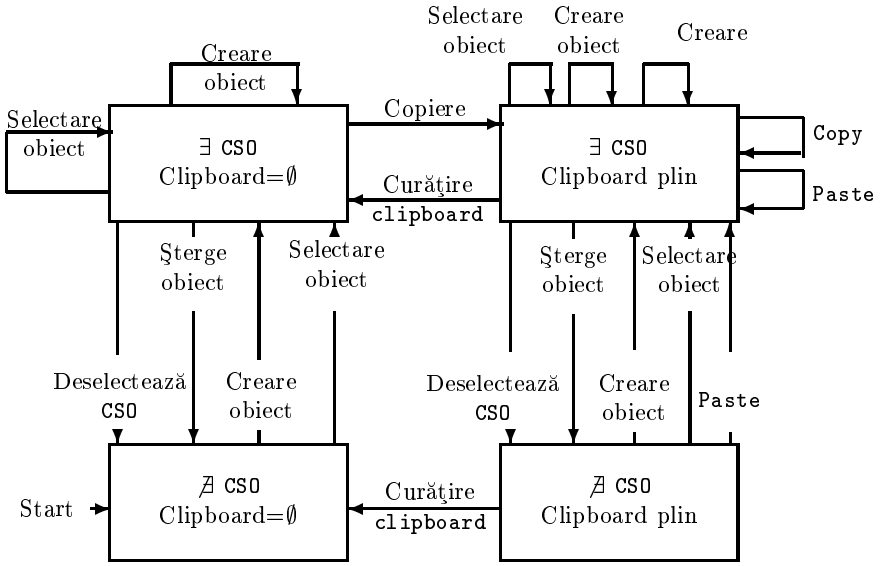



Figura 6.8: Reţea de tranziţie cu patru stări

Diferite construcţii specializate au fost dezvoltate pentru a simplifica reţelele de tranziţie. De exemplu, putem alinia problema de `help` prin folosirea subreţelelor într-un mod analog subrutinelor, pentru a ascunde un detaliu repetitiv localizat. Reţelele de tranziţie care pot apela recursiv subreţele sunt numite *reţele de tranziţie recursive*. Variabilele de stare în acest caz sunt întreaga stivă de stări salvate plus starea reţelei de tranziţie curent active.

Notaţia `Backus - Naur (BNF)` poate de asemenea fi folosită pentru a defini secvenţializarea, şi este echivalentă ca putere de reprezentare cu reţelele de tranziţie recursive (ambele sunt echivalente cu automatele `push-down`). Mai multe `UIMS` mai vechi au fost bazate pe specificaţii `BNF`.

Când reţelele de tranziţie devin mai complicate, cu expresii logice la tranziţii şi apeluri de subrutine, se ajunge la specificaţii asemănătoare unor programe (`program-like`). In definitiv, limbajele de programare reprezintă cel mai puternic mod de a specifica secvenţializările şi condiţiile multiple asociate, de multe ori, cu tranziţiile.

Mai multe *limbaje bazate pe evenimente* au fost dezvoltate special pentru specificarea interfeţelor utilizator. Este de remarcat că limbajele bazate pe evenimente, spre deosebire de limbajele de programare tradiţionale, nu au un

flux explicit de control. In schimb, oricând o condiţie `if` devine adevărată, acţiunile asociate sunt executate. Aşadar, limbajul bazat pe evenimente este un sistem de reguli de producţie. Limbajele bazate pe evenimente sunt mai puternice decât reţelele de tranziţie, reţelele de tranziţie recursive şi gramaticile, dar prea complicate pentru cazurile simple.

Un mod diferit de a defini sintaxa (secvenţializarea) dialogului este *prin exemplu*. In acest caz, `UIMS`-ul se plasează într-un mod "de învăţare", şi apoi se trece prin toate secvenţele acceptabile de acţiuni. Proiectantul poate începe cu un meniu principal, selectează un câmp din acest meniu şi merge printr-un director pentru a localiza submeniul, celula de dialog sau obiecte specifice aplicaţiei, pentru a fi prezentate utilizatorului ca răspuns la selectarea meniului principal. Obiectul apare pe ecran şi proiectantul poate să indice poziţia, mărimea sau alte atribute pe care obiectul trebuie să le aibă atunci când aplicaţia este într-adevăr executată. Proiectantul continuă să execute câteva operaţii asupra obiectului vizualizat şi apoi arată care obiect urmează, sau cum răspunde obiectul vizualizat la operatie; proiectantul repetă acest proces până când toate acţiunile asupra tuturor obiectelor sunt definite. Această tehnică funcţionează pentru secvenţializări prin `item`-uri (elemente) care au fost deja definite de proiectant.