

Capitolul 1

Introducere în grafica pe calculator

Obiectul graficii pe calculator (GPC) îl constituie crearea, memorarea și manipularea modelelor și imaginilor obiectelor. Modelele pot proveni din matematică, fizică, științe inginerești, arhitectură, meteorologie, etc. .

Grafica pe calculator este, în mare măsură, *interactivă* : utilizatorul controlează conținutul, structura și modul de apariție al obiectelor și al imaginilor lor afișate, utilizând dispozitive de intrare (tastatură, mouse, ecrane sensibile la atingere¹).

Câteva etape în evoluția GPC :

- Anii '80 : hardware și software costisitoare, dificil de utilizat,
- PC-urile² având ecrane cu grafică rastru încorporată (XEROX STAR, APPLE MACINTOSH, IBM PC) au introdus conceptul de imagini grafice bazate pe *hărți de biți*³.

Definiția 1.1

O *hartă de biți* este o metodă de reprezentare a unei imagini grafice prin care aceasta este *divizată* într-o matrice de puncte⁴ pe ecran. Fiecare pixel este memorat independent de ceilalți pixeli și i se poate atribui o culoare și/sau intensitate (tot în mod independent de ceilalți pixeli). ■

- Următorul concept apărut în GPC este cel de *suprafață de lucru*⁵. Prin intermediul unui program gestionar de ferestre⁶ utilizatorul poate crea, poziționa, redimensiona, ferestre⁷. Ferestrele sunt terminale asociate unor aplicații. Acest concept a dus la modificarea interfețelor text ale sistemelor de operare.

¹ touch-sensitive panel în lb. engleză

² Personal Computers în lb. engleză

³ Bitmaps în lb. engleză

⁴ Pixeli sau pels în lb. engleză

⁵ Desktop în lb. engleză

⁶ Window manager în lb. engleză

⁷ suprafete dreptunghiulare pe ecran

1.1 Procesarea imaginii

Grafica pe calculator se referă la sinteza unor obiecte reale sau imaginare din modele computaționale⁸.

Procesarea imaginii este un domeniu înrudit dar *diferit* : tratează analiza scenelor sau reconstrucția unor obiecte 2D/3D din imagini. Procesarea imaginilor este utilă în diverse domenii : recunoaștere aeriană, explorarea spațiului cosmic, robotică industrială, tomografie, etc. .

Exemplul 1.1

Pe baza unor fotografii aeriene, să se reconstruiască obiectele 3D din zonă.

Subdomeniile procesării imaginii sunt :

1. îmbunătățirea imaginii⁹ : îmbunătățirea calității imaginii prin eliminarea zgomotului¹⁰¹¹ sau prin îmbunătățirea contrastului.
2. detectarea și recunoașterea modelelor¹²(sau recunoașterea formelor) : metode/tehnici prin care se poate realiza o clasificare în cadrul unei mulțimi de obiecte, procese, fenomene.

Exemplul 1.2

Tehnologia OCR¹³ : prelucrarea caracterelor alfanumerice scrise de mână sau la mașina de scris.

3. analiza scenelor¹⁴ și vederea computerizată¹⁵ : permit recunoașterea și reconstituirea modelului 3D al unei scene din mai multe imagini 2D.

Exemplul 1.3

Un robot industrial.

Deși GPC și procesarea imaginii au început ca discipline separate astăzi există mai multe domenii în care se suprapun :

1. procesarea digitală (interactivă) a imaginii : fotografii scanate sunt modificate și combinate cu altele (eventual cu imagini sintetice) înainte de publicare.
2. în GPC anumite operații simple de procesare a imaginii sunt utilizate pentru sinteza imaginii unui model.

⁸modele ce pot fi memorate în memoria unui calculator

⁹*Image Enhancement* în lb. engleză

¹⁰*Noise* în lb. engleză

¹¹pixeli care lipsesc sau nu au legătură cu imaginea respectivă

¹²*pattern detection/recognition* în lb. engleză

¹³*Optical Character Recognition* în lb. engleză

¹⁴*Scene Analysis* în lb. engleză

¹⁵*Computer Vision* în lb. engleză

1.2 Avantajele GPC interactive

1. Interfață naturală : GPC reprezintă un mijloc natural de comunicare om-mașină datorită capacitateii umane de procesare a datelor grafice 2D/3D în mod rapid și eficient.
2. Vizualizarea științifică : GPC a devenit un mijloc indispensabil de interpretare a datelor produse în supercomputere.
3. GPC interactivă a devenit cel mai important mod de producere a imaginilor¹⁶ (atât ale unor obiecte concrete dar mai ales ale unor obiecte abstractive sau ale unor date care nu au proprietăți geometrice, de ex. rezultate ale unor sondaje de opinie). Imaginile nu sunt numai statice ci pot fi și dinamice, de ex. ale unor obiecte/fenomene care variază în timp : modelarea transformărilor fizionomiei din copilărie și pînă la o vîrstă înaintată.
4. Utilizarea imaginilor dinamice este în mod deosebit efectivă atunci când utilizatorul poate controla viteza animației, poziunea din scenă care este vizualizată, numărul de detalii afișate, etc. Din aceste motive, în cadrul GPC interactive, s-au dezvoltat tehnologii hardware/software pentru controlul de către utilizator a dinamicii mișcării¹⁷ cât și a dinamicii actualizărilor de imagine¹⁸ :
 - (a) În cadrul *dinamicii mișcării* obiectele pot fi mutate sau răsturnate¹⁹ în raport cu un observator staționar. Putem avea și cazul invers : obiectele pot fi staționare și observatorul mobil (de ex. un simulator de zbor).
 - (b) *Actualizarea imaginii* se referă la schimbarea vizibilă a formei, culorii sau a altor proprietăți ale obiectelor vizualizate (de ex. afișarea deformării structurii unui avion în zbor).
5. GPC interactivă dinamică oferă numeroase posibilități de codificare și de comunicare a informației : forma 2D/3D a obiectelor dintr-o imagine, scala de gri/culoare a acestora precum și variația în timp a acestor proprietăți.

1.3 Utilizări reprezentative ale GPC

1. Interfețe utilizator.
2. Scheme grafice²⁰ utilizate în economie, știință, tehnologie : histograme, hărți grafice de tip *bar*, *pie*, ordonanțare a activităților, etc.
3. Publicații electronice, birotică.
4. CAD - design-ul industrial cu ajutorul calculatorului²¹ : în industria de automobile, avionică, procesoare, telefonie, etc.

¹⁶de la inventarea fotografiei și a televiziunii

¹⁷*motion dynamics* în lb. engleză

¹⁸*update dynamics* în lb. engleză

¹⁹*tumbled* în lb. engleză

²⁰*charts* în lb. engleză

²¹*Computer Aided Design* în lb. engleză

5. Simularea și animația pentru vizualizare științifică și divertisment.

Exemplul 1.4

Pentru crearea unor desene animate desenatorul creează doar câteva cadre de referință, lăsând pe seama calculatorului crearea cadrelor de tranziție.

6. Artă și comerț : utilizarea GPC pentru crearea de reclame sau ghiduri computerizate în muzeu.
7. Controloare de proces : GPC intervine în controlul proceselor industriale din rafinării, centrale electrice, etc. prin afișarea valorilor datelor preluate de la senzorii atașați diverselor componente critice ale respectivelor sisteme (de ex. controlul traficului aerian).
8. Cartografie : realizarea de hărți geografice, hărți de relief, hărți oceanografice, hărți meteorologice, etc.

1.4 Clasificarea aplicațiilor de GPC

Aplicațiile de GPC se pot clasifica după mai multe criterii :

1. După *tipul (dimensionalitatea) obiectului* ce trebuie reprezentat și *tipul de imagine* produsă : vezi tabelul 1.1, pag. 13.
2. După *tipul de interacțiune* ce determină gradul de control al utilizatorului asupra obiectului și imaginii sale :
 - (a) *Offline plotting* : afișarea imaginii pe baza unei baze de date apriorice,
 - (b) *Interactive plotting* : utilizatorul controlează afișarea prin furnizarea de parametri. Afișarea interactivă este astfel și un proces iterativ.
 - (c) Afișarea obiectului în *temp real*²² (de ex. în simulatoarele de zbor).
 - (d) *Design interactiv* : în care utilizatorul începe cu un ecran vid, definește noi obiecte apoi prelucrează imaginea obținută.
3. După *rolul imaginii* sau în ce grad imaginea reprezintă un scop în sine sau servește unui scop : de ex., în cartografie sau în animație imaginea este produsul final, pe când în aplicațiile gen CAD imaginea este cea a unui obiect construit sau analizat.
4. *Relațiile temporale și logice* între obiecte și imaginile lor :
 - Utilizatorul poate lucra cu o singură imagine la un moment dat (cazul plotterelor),
 - Utilizatorul poate lucra cu o secvență variind în timp de imagini înrudite (ca în *motion dynamics*),
 - Utilizatorul poate lucra cu o colecție structurală de obiecte (ca în cazul aplicațiilor CAD).

²²Real-time plotting în lb. engleză

tip obiect	reprezentare grafică
2D	desenare de linii
	imagini în nuanțe de gri
	imagini colorate
3D	desenare de linii
	desenare de linii cu diverse efecte
	imagini colorate cu diverse efecte

Tabelul 1.1: Clasificarea după dimensionalitate

1.5 Evoluția tehnologiilor hardware și software pentru GPC

1.5.1 Evoluția tehnologiilor hardware

1. Dispozitive periferice de ieșire similară mașinilor de scris (cca 1950).
2. Apariția tubului catodic CRT²³ (cca 1955).
3. Apariția tastaturii, a creioanelor optice (inventate de Ivan Sutherland) în *Sketchpad Drawing System* (1963).
4. Apariția domeniilor CAD (1964) și CAM²⁴ (1981).
5. Apariția calculatoarelor personale (PCs) cu interfețe grafice (Apple, IBM PC) a redus dramatic costurile tehnologiilor hard/software pentru GPC.

Evoluția tehnologiilor hardware de ieșire

1. *Ecranul vectorial*²⁵ are arhitectura din figura 1.1, pag. 15. Procesorul de ecran (PE)²⁶ este similar unui dispozitiv periferic de ieșire. În memoria tampon a ecranului (ME)²⁷ sunt memorate liste de comenzi²⁸ de genul :

- **move** x_0, y_0 , care deplasează cursorul de ecran în poziția (x_0, y_0) ,
- **line** x_1, y_1 , care trasează un segment de dreaptă având extremitatea inițială poziția curentă a cursorului iar extremitatea finală poziția (x_1, y_1) ,
- **char** ABC, care afișează sirul "ABC".

Imaginea pe ecran (CRT) este obținută prin execuția de către PE a comenziilor din ME. Acest mod de obținere a imaginii se numește baleiere aleatoare²⁹.

²³Cathode Ray Tube în lb. engleză

²⁴Computed-Aided Manufacturing în lb. engleză

²⁵vector, stroke, line drawing, calligraphic display în lb. engleză

²⁶Display processor în lb. engleză

²⁷Display buffer memory în lb. engleză

²⁸Display list, display program în lb. engleză

²⁹Random scan în lb. engleză

Exemplul 1.5

În figura 1.2, pag. 16, imaginea din partea stângă se obține, pe un ecran vectorial printr-o succesiune de comenzi `move` și `line`.

Limitele acestei tehnologii sunt date de capacitatea de memorare a ME precum și de viteza PE. În plus trebuie ținut cont și de limitările tehnologice care impun reluarea execuției tuturor comenzilor din buffer cel puțin de 30 ori pe secundă și aceasta deoarece fosforul utilizat în CRT își pierde luminozitatea în zecimi/sutimi de μs .

2. Ecranele DVST³⁰ sunt similare celor vectoriale dar nu au ME (și deci și procesul de reîmprospătare a imaginii este inutil în acest caz).
3. Apariția configurației *ecran-minicalculator* a fost următorul pas în dezvoltarea tehnologicilor hardware grafice de ieșire.
4. Apariția în 1968 a ecranelor cu reîmprospătare a imaginii³¹ prevăzute cu procesoare ce puteau efectua (fără a solicita CPU) diverse transformări geometrice : scalare, rotație, translarea punctelor și a segmentelor de dreaptă în timp real, decupări 2D/3D³² și proiecții paralele și perspective.
5. Dezvoltarea graficii rastru (cca 1970) a contribuit, mai mult decât orice altă tehnologie, la dezvoltarea domeniului GPC. *Ecranele rastru*³³ memorează *primitivele de ecran* (segmente de dreaptă, caractere, suprafețe colorate uniform sau cu diverse modele³⁴) într-o zonă de memorie tampon de reîmprospătare³⁵ sub forma *pixelilor* componenți. Arhitectura unui ecran rastru este prezentată în figura 1.3, pag. 16.

Imaginea pe un ecran rastru este formată pe baza *rastrului* (alcătuit din linii de rastru³⁶ ce reprezintă șiruri de pixeli; practic rastrul fiind memorat într-o matrice de pixeli ce reprezintă tot ecranul). Imaginea este baleiată (sau scanată) linie cu linie, secvențial de către controller-ul video în ordinea sus, jos și înapoi. Acest mod de baleiere este baleierea rastrului³⁷.

În momentul afișării unui pixel, intensitatea fasciculului de electroni din CRT va fi modificată astfel încât să corespundă intensității pixelului respectiv. În cazul unui ecran în culori avem trei fascicule de electroni ale căror intensități corespund cu intensitățile celor trei componente ale culorii pixelului.

Exemplul 1.6

În figura 1.4, pag. 17 este prezentată imaginea pe un ecran rastru a figurii din partea stângă.

³⁰Direct view storage tube în lb. engleză

³¹Refresh display hardware în lb. engleză

³²Clipping în lb. engleză

³³Raster displays în lb. engleză

³⁴patterns în lb. engleză

³⁵Refresh buffer în lb. engleză

³⁶raster lines în lb. engleză

³⁷Raster scan în lb. engleză

Dacă într-un ecran vectorial buffer-ul de reîmprospătare conținea codul operațiilor de trasare și coordonatele finale, într-un ecran rastru întreaga imagine (de ex. 1024×1024 pixeli) trebuie să fie memorată explicit în buffer-ul de reîmprospătare. Termenul de *bitmap* se referă atât la buffer-ul de reîmprospătare cât și la matricea valorilor pixelilor.

6. Apariția memoriilor RAM a contribuit decisiv la impunerea modelului de ecran rastru³⁸.

Utilizând memorii RAM au apărut ecrane rastru *monocrom*³⁹ (i.e., cu 2 culori : alb/negru, negru/verde, portocaliu/verde). Bitmap-ul acestui tip de ecran conține 1 bit/pixel și deci, dacă un ecran monocrom are o rezoluție de 1024×1024 pixeli atunci bitmap-ul corespunzător are 2^{20} biți ($= 2^0 \cdot 1024 \cdot 1024$).

Ulterior au apărut ecrane *policrome* cu 8 biți/pixel (permășând 256 culori pentru un pixel) sau 24 biți/pixel (permășând 16 milioane de culori pentru un pixel).

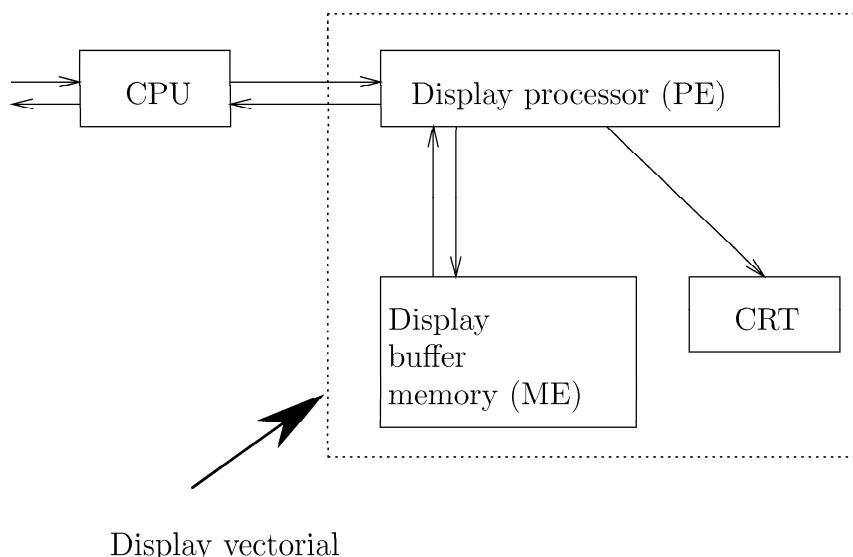


Figura 1.1: Arhitectura unui ecran vectorial

Observația 1.1

Termenul de *bitmap* se aplică doar ecranelor monocrome, în cazul ecranelor policrome utilizăm termenul de *pixmap*.

În cazul în care dorim să distingem între cele 2 semnificații ale termenului *bitmap* (*pixmap*) : conținutul bufferului de reîmprospătare și bufferul de reîmprospătare vom utiliza pentru bufferul de reîmprospătare termenul de *buffer cadru*⁴⁰.

Observația 1.2

Avantajele GPC utilizând ecrane rastru față de GPC utilizând ecrane vectoriale sunt :

³⁸în fața ecranului vectorial

³⁹Bilevel în lb. engleză

⁴⁰frame buffer în lb. engleză

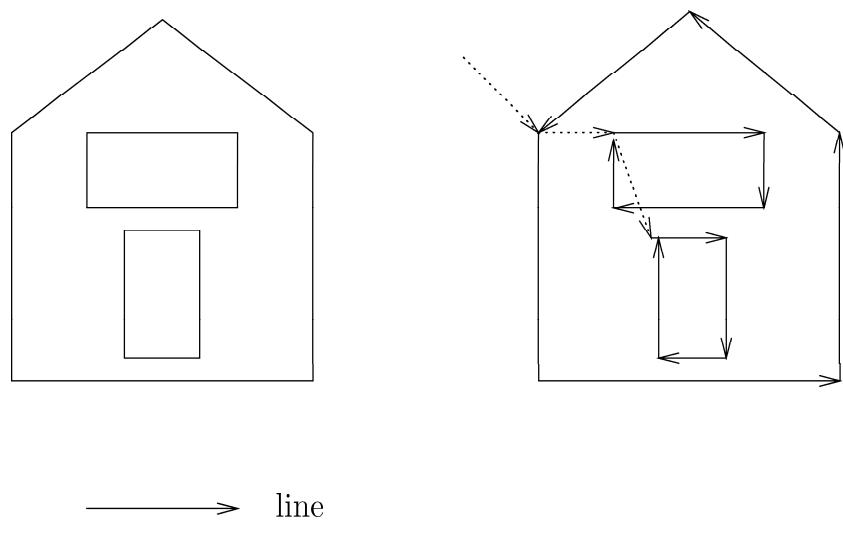


Figura 1.2: Obținerea unei imagini pe un ecran vectorial

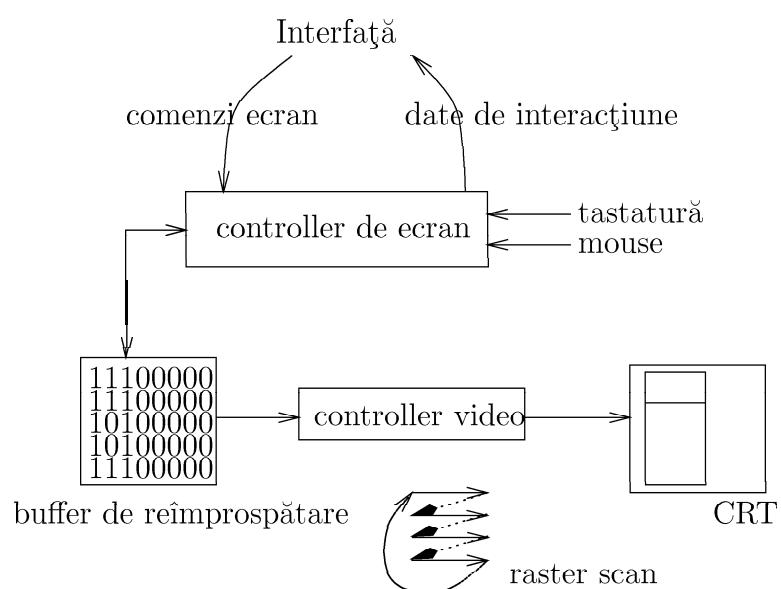


Figura 1.3: Arhitectura unui ecran rastru

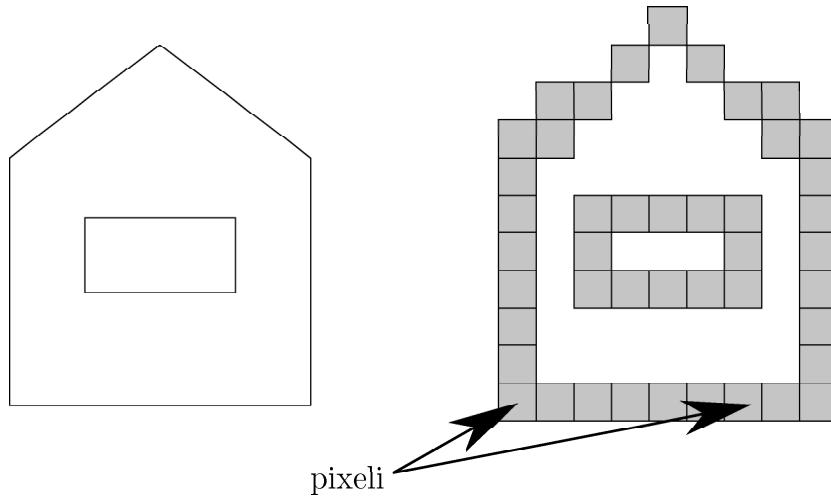


Figura 1.4: Obținerea unei imagini pe un ecran rastru

- Costuri scăzute (datorate costurilor scăzute ale memoriilor RAM).
- Posibilitatea de a colora uniform sau cu diferite modele zone compacte pe ecran.
- Procesul de reîmprospătare al imaginilor este independent de complexitatea imaginii (i.e., de numărul de segmente de dreaptă, de numărul de poligoane, etc.) iar tehnologiile hardware actuale permit citirea cel puțin o dată a fiecărui pixel din bufferul cadru într-un ciclu de reîmprospătare, lucru care nu este posibil întotdeauna în cazul unor imagini complicate afișate pe un ecran vectorial.

Totuși GPC utilizând ecrane rastru are și o serie de *dezavantaje* față de GPC utilizând ecrane vectoriale :

- Reprezentarea segmentelor de dreaptă sau a poligoanelor se face, în general, prin vârfurile lor și în concluzie pentru a fi afișate pe un ecran rastru trebuie să existe software specializat care să calculeze pixelii corespunzători unui segment de dreaptă sau poligon. În general această conversie se efectuează utilizând un procesor dedicat RIP⁴¹.
- Deoarece reprezentarea unei primitive grafice pe un ecran rastru necesită o conversie, reprezentarea grafică a unor fenomene dinamice în timp real este costisitoare pentru ecranele rastru.
- Reprezentarea unor curbe continue este aproximată prin pixeli pentru ecranele rastru. De aici rezultă aspectul zimțat, colțuros al acestor reprezentări.

⁴¹Raster Image Processor în lb. engleză

Evoluția tehnologiilor hardware de intrare

Primele dispozitive hardware grafice de intrare au fost *creioanele optice* pentru ecranele vectoriale. Ulterior au apărut : *mouse-ul* (Doug Engelbart 1968), *tablettele grafice*⁴², *ecranele sensibile la atingere*⁴³.

1.5.2 Evoluția tehnologiilor software grafice (și al standardelor grafice)

Tehnologiile software grafice s-au dezvoltat plecând de la software scris pentru un singur tip de ecran și într-un limbaj de programare de nivel scăzut (chiar limbaj de asamblare sau cod mașină) și ajungând la software independent de dispozitivul grafic de ieșire și dezvoltat în limbaje de programare de nivel înalt (C, Pascal, Fortran, etc.).

Standardele grafice au apărut începând cu anul 1975 (cca) :

- 1977, 1979 : 3D Core Graphics System.
- GKS (Graphical Kernel System) : este standard ANSI (1985).
- GKS 3D (1988).
- PHIGS (Programmer's Hierarchical Interactive Graphic System) : 1988.
- SRGP (Simple Raster Graphics Package).
- SPHIGS (Simple PHIGS).
- OPENGL.

1.6 Conceptele GPC interactive

Majoritatea sistemelor grafice interactive au structura din figura 1.5, pag. 19 :

1. *Modelul aplicației*⁴⁴ reprezintă datele sau obiectele ce vor fi reprezentate pe ecran.
2. *Programul aplicației*⁴⁵ creează, memorează în sau regăsește din modelul aplicației.

Programul aplicației gestionează intrările de la utilizator și produce *vizualizări*⁴⁶ ale modelului trimițând *sistemului grafic* o serie de comenzi grafice de ieșire ce conțin o descriere geometrică detaliată a imaginii care va fi afișată cât și atributele ce descriu modul în care va apărea respectiva imagine.

Sistemul grafic poate fi văzut realizând o *transformare de ieșire* ale obiectelor din modelul aplicației într-o priveliște a respectivului model sau realizând o *transformare de intrare* a acțiunilor utilizatorului în intrări pentru programul aplicației.

Designer-ul unui sistem grafic interactiv trebuie să specifice :

⁴² *Tablet data* în lb. engleză

⁴³ *Touch Sensitive Panels* în lb. engleză

⁴⁴ *Application model* în lb. engleză

⁴⁵ *Application program* în lb. engleză

⁴⁶ *Views* în lb. engleză

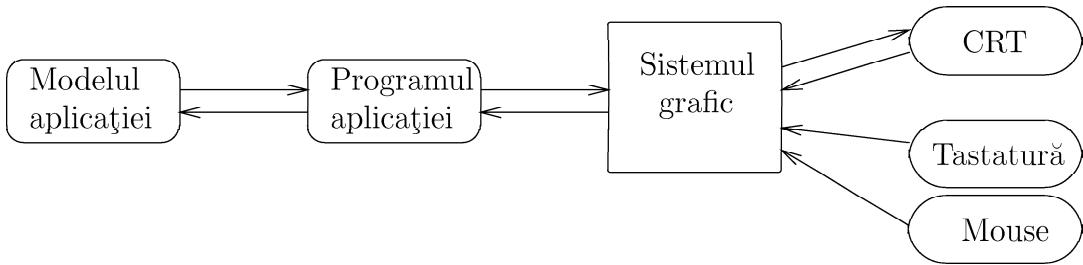


Figura 1.5: Structura unui sistem grafic interactiv

- Clasele de obiecte ce vor fi generate și reprezentate grafic. De exemplu, OpenGL permite să afișăm triunghiuri, poligoane, etc. (prin specificarea coordonatelor vârfurilor) dar nu și cercuri (aceasta realizându-se utilizând GLUT).
- Modalitățile prin care utilizatorul și programul aplicației vor interacționa pentru a crea și modifica modelul și reprezentarea sa vizuală.

1.6.1 Modelarea aplicației

Definiția 1.2

Modelul aplicației cuprinde datele, obiectele și relațiile dintre ele relevante pentru partea de afișare și de interacțiune a programului aplicației și pentru orice modul de postprocesare negrafiic (de ex. module care realizează simularea unui model populațional).

Exemplul 1.7

Programul PAINT (WINDOWSTM) este un program interactiv de realizare de imagini. Utilizatorul poate modifica în mod direct pixelii din bitmap/pixmap.

Modelul aplicației poate fi de mai multe feluri :

1. Model *inexistent* (de ex. în programul PAINT).
2. Model alcătuit din *combinări* de descrieri de date și de proceduri. Modelul datelor poate fi un tablou sau o listă înlanțuită sau chiar o bază de date relațională.

Modelul aplicației *memorează* de obicei :

- *Primitive* (punkte, segmente de dreaptă, poligoane în 2D/3D, poliedre, suprafețe 3D) care definesc *forma* componentelor obiectului.
- *Atribute* ale obiectului : culoare, stil (pentru segmente de dreaptă), textură (pentru suprafețe).
- *Relații de conexiune și date de poziționare* care descriu modul de ajustare al obiectelor.

Putem efectua o *clasificare* a modelelor aplicațiilor în funcție de *gradul de modelare geometrică* al obiectelor :

1. Modele în care obiectele sunt descrise în întregime în termeni geometrici (de ex. poliedrele componente).
2. Modele în care descrierea geometrică a obiectelor nu este necesară (de ex. foi de calcul⁴⁷).
3. Modele pentru care descrierea geometrică poate fi obținută *on-the-fly* (de ex. digrafurile se memorează prin liste de adiacență și doar când e nevoie se poate genera descrierea geometrică).

1.6.2 Programarea aplicației

Programul aplicației *creează modelul aplicației* fie *aprioric* (ca rezultat al unor calcule anterioare) sau *interactiv* (când utilizatorul ghidează procesul de construcție al modelului aplicației pas cu pas; în acest caz programul aplicației trebuie să poată furniza priveliști ale modelului parțial).

Deoarece modelele sunt specifice diverselor aplicații și sunt create independent de un anumit tip de ecran, programul aplicației trebuie să *convertească* o porțiune din modelul ce trebuie vizualizat din reprezentarea internă geometrică (fie explicit memorată în model, fie derivată *on-the-fly*) în apeluri de proceduri sau comenzi pe care sistemul grafic le utilizează pentru a crea o imagine.

Procesul de *conversie* are două etape :

- Programul aplicației extrage porțiunile ce trebuie vizualizate utilizând criterii de interogare sau selecție pentru baza de date în care este memorat modelul aplicației.
- Porțiunile extrase sunt convertite în formatul de intrare al sistemului grafic. Datele extrase în prima etapă trebuie să fie geometrice sau să fie convertite într-un format geometric înțeles de către sistemul grafic. Dacă modelul conține primitive geometrice care nu sunt suportate de către sistemul grafic atunci programul aplicației trebuie să le convertească în primitivele suportate de către sistemul grafic.

Sistemul grafic constă într-un set de proceduri corespunzând diverselor primitive și attribute. Acestea sunt colectate în biblioteci grafice (apelabile din limbaje de nivel înalt : C, Pascal). Sistemul grafic permite deasemenea și abstractizarea diverselor detaliilor de implementare a sistemului grafic prin crearea *dispozitivelor logice de ecran*⁴⁸. Pot fi abstractizate detaliile de implementare cum ar fi :

- care parte anume din generarea de imagini se efectuează cu ajutorul tehnologiilor hardware și care se efectuează de către sistemul grafic,
- mouse-ul, tabela de date, ecranul senzitiv, joystick-ul pot fi tratate ca dispozitive de intrare logice de localizare ce returnează o adresă (x, y) de pe ecran. În acest caz programul aplicației poate cere sistemului grafic să testeze dispozitivele de intrare sau să aștepte până când un eveniment este generat în momentul activării de către utilizator a unui dispozitiv de intrare. Cu aceste valori de intrare (obținute prin testare)⁴⁹

⁴⁷ Spreadsheets în lb. engleză

⁴⁸ Logical Display Device în lb. engleză

⁴⁹ sampling în lb. engleză

sau prin aşteptarea unui eveniment) programul aplicației poate gestiona interacțiuni cu utilizatorul care modifică modelul sau ecranul sau care îi schimbă modul de operare.

1.6.3 Tratarea interacțiunilor

Tratarea interacțiunilor în programul aplicației se face utilizând o buclă de tratare a evenimentelor⁵⁰ (reprezentată în tabelul 1.2, pag. 21). Aceasta este un automat finit determinist cu o singură stare (de aşteptare) și tranziții generate de evenimentele cauzate de intrările utilizator.

```
begin
    1. generarea ecranului inițial (derivată din modelul aplicației)
    while(!quit)
        2. trecerea în modul de selectare a comenziilor
        3. aşteptare selecție utilizator
        switch(selectie)
            4. selectarea unui proces pentru completarea (finalizarea)
                comenzi sau procesarea completă de către acest proces a comenzi
            5. actualizarea modelului și a ecranului
        endswitch
    endwhile
end
```

Tabelul 1.2: Buclă de tratare a evenimentelor

Observația 1.3

În bucla de tratare a evenimentelor trebuie reținut că *nu* pot exista modificări pe ecran fără a exista în prealabil în modelul aplicației (coerența dintre modelul aplicației și ecran trebuie menținută).

Această observație nu se referă și la cazul în care modelul aplicației și imaginea de pe ecran sunt identice (de ex. în cazul programul PAINT sau în aplicațiile de îmbunătățire a imaginilor).

Observația 1.4

Sistemul grafic nu are nici un fel de responsabilitate în construirea sau modificarea modelului (fie inițial, fie ca răspuns la interacțiunea cu utilizatorul). Singura sa responsabilitate constă în crearea imaginilor din descrieri geometrice și de a transmite datele de intrare de la utilizator.

1.7 Sumar

- Interfețele grafice au substituit pe cele în mod text ca standarde în interacțiunea utilizator-calculator.

⁵⁰event-driven loop în lb. engleză

- Exprimarea *grafică* a ideilor, datelor în cele mai diverse domenii este realizată prin intermediul GPC.
- În anii '80 au apărut calculatoarele personale (PCs) cu grafică bitmap și cca un deceniu mai târziu procesoarele care realizează în timp real animații 3D cu imagini policrome.

Capitolul 2

Grafică rastru 2D

În acest capitol vom prezenta diversi algoritmi de afişare a unor obiecte matematice simple (segmente de dreaptă, cercuri, elipse, poligoane) pe ecrane rastru (algoritmi de conversie prin scanare¹ a primitivelor geometrice în pixeli). Ulterior vom prezenta algoritmi care realizează o colorare uniformă a pixelilor interiori unei primitive geometrice (dreptunghi, poligon oarecare, cerc, elipsă).

Ecranele rastru vor fi modelate prin spațiul discret $\mathbf{Z} \times \mathbf{Z}$ și funcția $value : \mathbf{Z} \times \mathbf{Z} \rightarrow V$, unde, în cazul ecranelor rastru monocrome (cu 1 bit / pixel) $V = \mathbf{B} (= \{\text{true}, \text{false}\}$ sau $\{0, 1\}$) iar în cazul ecranelor multicrome (cu $n \geq 2$ biți / pixel) alegem $V \subseteq \mathbf{N}$ astfel încât $|V| \geq$ numărul nivelurilor de intensitate (sau numărul de culori).

Din punct de vedere grafic vom reprezenta ecranele rastru printr-o mulțime de drepte paralele (orizontale și verticale - linii și coloane) egal distanțate (sub forma unei grile) iar pixelii (aprinși cu diverse intensități sau stinși) vor fi discuri centrate în punctele de intersecție ale grilei (denumite vârfurile grilei). În figura 2.1, pag. 23 am reprezentat un ecran rastru.

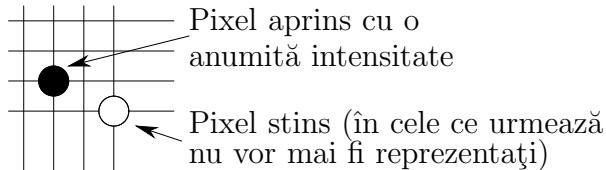


Figura 2.1: Reprezentarea unui ecran rastru

2.1 Afisarea segmentelor de dreaptă

Vom afișa segmente de dreaptă pe ecrane rastru monocrome. Segmentele de dreaptă vor fi approximate prin mulțimi de pixeli aprinși. Avem problema următoare :

Problema 2.1

Fie $[P_0P_n]$ un segment de dreaptă, unde $P_0(x_0, y_0)$ și $P_n(x_n, y_n)$. Mai presupunem că $(x_0, y_0) \in \mathbf{Z} \times \mathbf{Z}$ și $(x_n, y_n) \in \mathbf{Z} \times \mathbf{Z}$. Să se determine secvența de pixeli $M = (V_0, V_1, \dots, V_n)$ având proprietățile următoare :

¹scan converting în lb. engleză

1. $V_0(x_0, y_0)$ (i.e., pixelul V_0 este situat la intersecția dintre coloana $x = x_0$ și linia $y = y_0$), $V_n(x_n, y_n)$ (i.e., pixelul V_n este situat la intersecția dintre coloana $x = x_n$ și linia $y = y_n$),
2. Fie m panta dreptei P_0P_n (ecuația dreptei P_0P_n este $y = m \cdot x + n$), unde m și n se obțin din ecuația dreptei care trece prin punctele P_0 și P_n : $\frac{y-y_0}{y_n-y_0} = \frac{x-x_0}{x_n-x_0}$). Dacă $|m| \leq 1$ atunci $x_0 < x_1 < \dots < x_n$ și $(\forall 0 \leq i < n)(x_{i+1} = x_i + 1)$. În mod informal această condiție semnifică faptul că pentru segmente de dreaptă având panta $|m| \leq 1$ trebuie să existe un unic pixel aprins în fiecare coloană intersectată de segmentul de dreaptă.
3. Dacă $|m| > 1$ atunci $y_0 < y_1 < \dots < y_n$ și $(\forall 0 \leq i < n)(y_{i+1} = y_i + 1)$. În mod informal această condiție semnifică faptul că pentru segmente de dreaptă având panta $|m| > 1$ trebuie să existe un pixel aprins în fiecare linie intersectată de segmentul de dreaptă.
4. Distanța dintre pixelul $V_i(x_i, y_i)$ și punctul de intersecție dintre $[P_0P_n]$ și coloana $x = x_i$ (dacă $|m| \leq 1$) sau dintre $V_i(x_i, y_i)$ și punctul de intersecție dintre $[P_0P_n]$ și linia $y = y_i$ (dacă $|m| > 1$) trebuie să fie minimă, i.e., pixelul V_i este cel mai apropiat de punct dintre toți pixelii de pe coloana $x = x_i$ sau, similar, de pe linia $y = y_i$.

Observația 2.1

Cerințele 2 și 3 ale problemei 2.1, pag. 23 au rolul de a asigura o anumită "continuitate" în reprezentarea unui segment de dreaptă. Încălcarea acestor cerințe poate conduce la afișări incorecte, de exemplu reprezentarea unui segment de dreaptă continuu printr-un segment de dreaptă discontinuu.

În cele ce urmează vom prezenta o serie de algoritmi pentru rezolvarea problemei de mai sus. Vom pleca de la un algoritm inițial din care vom obține algoritmi superiori din punctul de vedere al complexității timp. Rafinarea algoritmului inițial constă în eliminarea operațiilor de înmulțire și împărțire și înlocuirea acestora cu operații de adunare și scădere. Vom încerca să utilizăm variabile de tip **int** în loc de variabile de tip **float** sau **double**.

Algoritm 1 AfișareSegmentDreaptă1

Date de intrare : $(x_0, y_0), (x_n, y_n) \in \mathbf{Z} \times \mathbf{Z}$

Date de ieșire : $M \in \text{list of } (\mathbf{Z} \times \mathbf{Z})$, secvența de pixeli cerută de problema 2.1, pag. 23. Presupunem că ne aflăm în cazul $|m| \leq 1$.

Metodă :

```

procedure AfisareSegmentDreapta1 (x0,y0,xn,yn : Z,
                                   var M : list of (Z x Z))
{
  double m = (yn - y0) / (xn - x0);
  double k = y0 - m * x0;
  // am calculat dreapta y = m * x + k care trece prin
  // punctele P0 si Pn
  M = \epsilon; // M este initial lista vida
}

```

```

for (int x = x0; x <= xn; x++)
{
    double y;
    y = m * x + k;
    // se crează M și se adaugă, prin concatenare, un
    // nou pixel V(x, floor(y + 0.5))
    M = M . (x, floor(y + 0.5));
}

```

Mai sus notația \backslashepsilon semnifică ϵ (notația pentru lista vidă) și $M = M . (x, floor(y + 0.5))$; semnifică concatenarea listei M cu un element : $M = M \cdot (x, floor(y + 0.5))$.

Exercițiul 2.1

Modificați algoritmul 1, pag. 24 astfel încât să funcționeze corect și în cazul în care $|m| > 1$. În cazul în care avem $|m| > 1$ procedura *AfisareSegmentDreaptă1* poate calcula o secvență de pixeli $M = ((x_0, y_0), \dots, (x, y), (x+1, y'), \dots, (x_n, y_n))$, cu $y' > y + 1$ ceea ce contravine cerinței 3 din problema 2.1, pag. 23.

Algoritmul următor provine din algoritmul 1, pag. 24. Este un algoritm *incremental* (aceasta însemnând că valorile (x', y') ale unui pixel sunt obținute din valorile (x, y) ale pixelului imediat precedent printr-o operație de adunare).

Algoritmul 2 AfisareSegmentDreaptă2

Date de intrare : $(x_0, y_0), (x_n, y_n) \in \mathbb{Z} \times \mathbb{Z}$

Date de ieșire : $M \in \text{list of } (\mathbb{Z} \times \mathbb{Z})$, secvență de pixeli cerută de problema 2.1, pag. 23. Presupunem că ne aflăm în cazul $|m| \leq 1$.

Metodă : În algoritmul 1, pag. 24 înlocuim instrucțiunea $y = m * x + k$; să presupunem că $M = (x_0, y_0) \cdot \dots \cdot (x_i, y_i)$. La pasul următor lui M i se adaugă pixelul (x_{i+1}, y_{i+1}) : $x_{i+1} = x_i + 1$ și $y_{i+1} = m \cdot x_{i+1} + k$ deci $y_{i+1} = m \cdot (x_i + 1) + k = (m \cdot x_i + k) + m$. Dar $y_i = m \cdot x_i + k$ și deci $y_{i+1} = y_i + m$.

```

procedure AfisareSegmentDreaptă2 (x0,y0,xn,yn : Z,
                                    var M : list of (Z x Z))
{
    double m = (yn - y0) / (xn - x0);
    double y = y0;
    M = \epsilon; // M este initial lista vida
    for (int x = x0; x <= xn; x++)
    {
        // se crează M și se adaugă, prin concatenare, un
        // nou pixel V(x, floor(y + 0.5))
        M = M . (x, floor(y + 0.5));
        y += m;
    }
}

```

Observația 2.2

Și în cazul algoritmului 2, pag. 25 cazul $|m| > 1$ este tratat incorrect. Modificați algoritm 2, pag. 25 astfel încât cazul $|m| > 1$ să fie tratat corect.

Algoritmul următor (propus de Pitteway în 1967 și de van Aken în 1984 și variantă a unui algoritm propus de Bresenham în 1965) calculează exact aceeași secvență de pixeli ca algoritmii anteriori dar utilizează doar aritmetică întregilor (astfel vom renunța la utilizarea variabilelor de tip **double** sau la utilizarea funcției parte întreagă *inferioră floor*).

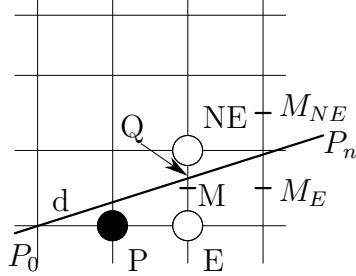


Figura 2.2: Alegerea unuia dintre pixelii E sau NE

Principiul acestui algoritm este următorul : să presupunem că pixelul P este pixelul curent selectat (i.e., ultimul pixel calculat al secvenței M).

Mai presupunem, fără a restrânge generalitatea, că dreapta d (determinată de segmentul $[P_0P_n]$) are panta $|m| \leq 1$ și poziția ca în figura 2.2, pag. 26 (i.e., $x_0 < x_n$, $y_0 < y_n$ și $\frac{y_n - y_0}{x_n - x_0} < 1$).

În acest caz următorul pixel din secvență poate fi fie pixelul E (*est*), fie pixelul NE (*nord-est*). Conform cerinței 4 din problema 2.1, pag. 23 va trebui ales acel pixel a cărui distanță până la dreapta d este $\min(\text{distanta}(E, d), \text{distanta}(NE, d))$.

Decizia alegerii unuia dintre cei doi pixeli este luată în funcție de poziția punctului M situat la mijlocul distanței dintre E și NE. Dacă punctul M este situat *sub* dreapta d (aceasta însemnând că punctul Q , care este intersecția dreptei d cu segmentul de dreaptă [E NE], este situat între M și NE) atunci este evident că NE este pixelul cel mai apropiat de d și va fi selectat. Altfel, dacă M este situat *deasupra* lui d , i.e., Q este situat între M și E atunci este selectat pixelul E.

Notația 2.1

În cele ce urmează folosim următoarele notații : $P(x_P, y_P)$, $E(x_P + 1, y_P)$, $NE(x_P + 1, y_P + 1)$, $Q = d \cap [E \text{ NE}]$. Ecuația dreptei d este $F(x, y) = 0$, unde $F(x, y) = a \cdot x + b \cdot y + c$. Coeficienții a , b , c se obțin plecând de la ecuația dreptei d (care trece prin P_0 și P_n) :

$$\frac{x - x_0}{x_n - x_0} = \frac{y - y_0}{y_n - y_0} \quad (2.1)$$

Utilizăm notațiile $d_x \doteq x_n - x_0$ și $d_y \doteq y_n - y_0$. Rescriem ecuația 2.1 :

$$\frac{x - x_0}{d_x} = \frac{y - y_0}{d_y} \quad (2.2)$$

și deci avem $y = \frac{d_y}{d_x} \cdot x + (y_0 - \frac{d_y}{d_x} \cdot x_0)$, și deci avem

$$d_y \cdot x - d_x \cdot y + (d_x \cdot y_0 - d_y \cdot x_0) = 0 \quad (2.3)$$

În concluzie coeficienții a, b, c sunt : $a = d_y, b = -d_x, c = d_x \cdot y_0 - d_y \cdot x_0$. Ecuația 2.3 se mai numește și ecuația *implicită* a dreptei d .

Proprietatea 2.1 (poziția punctului M relativ la dreapta d)

Fie punctul $M(x', y')$ și dreapta d dată prin ecuația implicită $F(x, y) = 0$. Atunci :

1. $M \in d \iff F(x', y') = 0$.
2. M este situat sub dreapta d dacă $F(x', y') > 0$.
3. M este situat deasupra dreptei d dacă $F(x', y') < 0$.

Demonstratie

1. Evident.
2. Să presupunem că M este situat sub dreapta d . Fie $V = d \cap \{x \mid x = x'\}$. Deci V are coordonatele $(x', \frac{-ax'-c}{b})$. Atunci $M(x', y')$ este situat sub d dacă $y' < \frac{-ax'-c}{b}$. Dacă $b < 0$ (și în acest caz este deoarece $b = -d_x$) atunci avem $M(x', y')$ este situat sub d dacă $a \cdot x' + b \cdot y' + c > 0 \iff F(x', y') > 0$
3. Se arată în mod similar cu cazul precedent.

□

Am văzut mai sus că alegerea unuia din pixelii E sau NE depinde de poziția punctului M față de dreapta d . Conform proprietății 2.1, pag. 27 putem acum să descriem *matematic* acest lucru : punctul M are coordonatele $x_P + 1$ și $y_P + \frac{1}{2}$ și notăm $d \doteq F(x_P + 1, y_P + \frac{1}{2})$. d este o variabilă de decizie utilizată în algoritm². Dacă $d > 0$ atunci alegem pixelul NE și dacă $d \leq 0$ alegem pixelul E.

Stim că primul pixel din secvența de pixeli este chiar extremitatea $P_0(x_0, y_0)$ a segmentului $[P_0 P_n]$. Valoarea de inițializare a variabilei de decizie d este atunci $F(x_0 + 1, y_0 + \frac{1}{2}) = a \cdot (x_0 + 1) + b \cdot (y_0 + \frac{1}{2}) + c = a \cdot x_0 + b \cdot y_0 + c + a + \frac{b}{2} = F(x_0, y_0) + a + \frac{b}{2}$ (deoarece $P_0 \in [P_0 P_n]$ avem $F(x_0, y_0) = 0$).

Deoarece există posibilitatea ca $\frac{b}{2} \notin \mathbf{Z}$ vom considera $F(x, y) = 2 \cdot (a \cdot x + b \cdot y + c)$ și se observă că această alegere nu schimbă semnul variabilei de decizie d .

În algoritmul care urmează, după alegerea unuia dintre pixelii E sau NE ca fiind pixel current, trebuie să modificăm și punctul de mijloc M (deci și valoarea variabilei de decizie d). Așa cum vom vedea această modificare este *incrementală*.

Să presupunem că pixelul E a fost cel selectat. Atunci punctul de mijloc M devine $M_E(x_P + 2, y_P + \frac{1}{2})$. Noua valoare a variabilei de decizie d este $d' = F(M_E) = F(x_P + 2, y_P + \frac{1}{2}) = 2 \cdot (a \cdot (x_P + 2) + b \cdot (y_P + \frac{1}{2}) + c) = 2 \cdot (a \cdot (x_P + 1) + b \cdot (y_P + \frac{1}{2}) + c) + 2 \cdot a = d + 2 \cdot a$.

²a nu se confunda cu dreapta d care trece prin punctele P_0 și P_n

Să presupunem că pixelul NE a fost cel selectat. Atunci punctul de mijloc M devine $M_{NE}(x_P + 2, y_P + \frac{3}{2})$. Noua valoare a variabilei de decizie d este $d' = F(M_{NE}) = F(x_P + 2, y_P + \frac{3}{2}) = 2 \cdot (a \cdot (x_P + 2) + b \cdot (y_P + \frac{3}{2}) + c) = 2 \cdot (a \cdot (x_P + 1) + b \cdot (y_P + \frac{1}{2}) + c) + 2 \cdot (a + b) = d + 2 \cdot (a + b)$.

Pe baza celor de mai sus putem descrie algoritmul de afişare a segmentelor de dreaptă care utilizează tehnica punctului de mijloc.

Algoritm 3 AfisareSegmentDreaptă3

Date de intrare : $(x_0, y_0), (x_n, y_n) \in \mathbf{Z} \times \mathbf{Z}$

Date de ieşire : $M \in \text{list of } (\mathbf{Z} \times \mathbf{Z})$, secvența de pixeli cerută de problema 2.1, pag. 23.
Presupunem că ne aflăm în cazul $|m| \leq 1$ și $x_0 < x_n$, $y_0 < y_n$.

Metodă :

```
procedure AfisareSegmentDreapta3 (x0,y0,xn,yn : Z,
                                   var M : list of (Z x Z))
{
    // valoarea initială a variabile de decizie
    // dx, dy sunt constante - a se vedea mai sus
    int d = 2 * dy - dx;
    int dE = 2 * dy;
    int dNE = 2 * (dy - dx);
    int x = x0, y = y0;
    M = (x,y);
    while (x < xn)
    {
        if (d <= 0) /* alegem E */      d += dE; x++; }
        else {           /* alegem NE */   d += dNE; x++; y++; }
        M = M . (x, y);
    }
}
```

Observația 2.3

Algoritmii 1, pag. 24, 2, pag. 25 și 3, pag. 28 de afişare a segmentelor de dreaptă pot fi folosiți pentru afişarea liniilor poligonale sau a contururilor poligoanelor.

2.2 Afisarea cercurilor

Ca și în cazul afișării segmentelor de dreaptă vom prezenta mai întâi algoritmi relativ simpli de afișare a unui cerc (dar care utilizează operații complexe în raport cu operația de adunare a întregilor). Ulterior vom prezenta algoritmi mai complicați dar având o complexitate timp mai bună (deoarece sunt incrementali și utilizează doar operații de adunare a întregilor).

Vom prezenta algoritmi de afișare a unui cerc având centrul în origine și rază $R \in \mathbf{Z}$. Ecuația unui astfel de cerc este $F(x, y) = 0$, unde $F(x, y) = x^2 + y^2 - R^2$.

Algoritmul 4 AfisareCerc1

Date de intrare : $R \in \mathbf{Z}$, raza unui cerc centrat în origine,

Date de ieșire : $M \in \text{list of } (\mathbf{Z} \times \mathbf{Z})$, o secvență de pixeli care aproximează cadranul I al cercului de rază R centrat în origine (cadran descris matematic astfel : $\{(x, y) \mid (x, y) \in \mathbf{R}^2 \wedge 0 \leq x \leq R \wedge y = \sqrt{R^2 - x^2}\}$).

Metodă :

```

procedure AfisareCerc1 (R : Z, var M : list of (Z x Z))
{
    M = \epsilon;
    for (int x = 0; x <= R; x++)
    {
        M = M . (x, floor(sqrt(R*R - x*x) + 0.5));
    }
}

```

Observația 2.4

Utilizând procedura AfisareCerc1 putem afișa un cerc astfel : dat un pixel $P(x, y) \in M$ putem obține pixelii simetrici din cadranele 2,3,4 prin simetrie față de axa Oy , față de origine și față de axa Ox : $P''(-x, y)$, $P'''(-x, -y)$ și $P^\dagger(x, -y)$.

Observația 2.5

Algoritmul 4, pag. 29 utilizează operații relativ costisitoare (*floor*, *sqrt*). În plus prezintă și dezavantajul de a genera pixeli în mod neuniform : e.g., în cazul unui cerc de rază 17 centrat în origine, primii doi pixeli generați sunt $(0, 17)$ și $(1, 17)$ iar ultimii doi pixeli generați sunt $(16, 6)$ și $(17, 0)$. Acest ultim dezavantaj nu apare în cazul algoritmului următor.

Algoritmul 5 AfisareCerc2

Date de intrare : $R \in \mathbf{Z}$, raza unui cerc centrat în origine,

Date de ieșire : $M \in \text{list of } (\mathbf{Z} \times \mathbf{Z})$, o secvență de pixeli care aproximează cadranul I al cercului de rază R centrat în origine.

Metodă : Vom folosi coordonatele polare ale pixelilor : $\begin{cases} x = R \cdot \cos t \\ y = R \cdot \sin t \end{cases}$, unde $0 \leq t \leq \frac{\pi}{2}$.

```

procedure AfisareCerc2 (R : Z, var M : list of (Z x Z))
{
    M = \epsilon;
    for (double t = 0; t <= PI/2; t+=PI/100)
    {
        M = M . (floor(R*cos(t) + 0.5), floor(R*sin(t) + 0.5));
    }
}

```

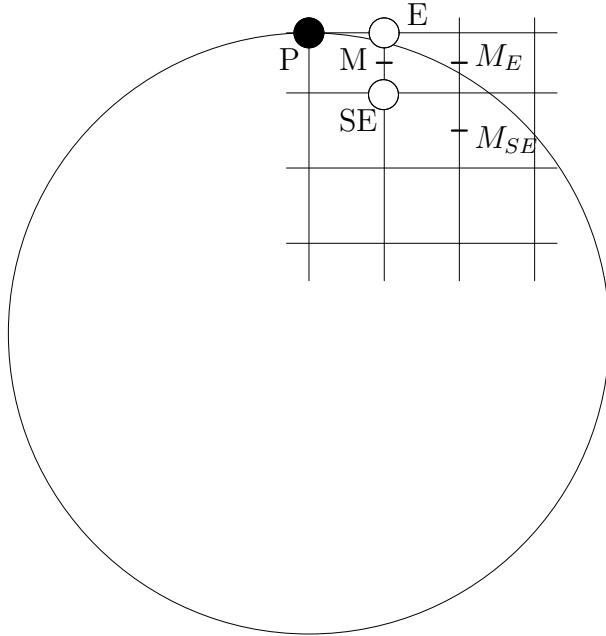


Figura 2.3: Alegerea unuia dintre pixelii E sau SE

Şi în cazul cercului putem aplica tehnica punctului de mijloc. Principiul este acelaşi ca în cazul unui segment de dreaptă : plecând de la pixelul curent, următorul pixel din secvenţă este ales dintre doi pixeli, în funcţie de poziţia faţă de cerc a punctului situat la jumătatea distanţei dintre cei doi pixeli. Vom genera prin această metodă doar pixelii din octantul al doilea al cercului de rază R centrat în origine (descriş matematic astfel : $\{(x, y) \mid (x, y) \in \mathbf{R}^2 \wedge x^2 + y^2 = R^2 \wedge 0 \leq x \leq \frac{R\sqrt{2}}{2} \wedge \frac{R\sqrt{2}}{2} \leq y \leq R\}$).

Presupunând că $P(x_P, y_P)$ este pixelul curent, alegerea se va face între pixelii $E(x_P + 1, y_P)$ (est) și $SE(x_P + 1, y_P - 1)$ (sud-est), conform figurii 2.3, pag. 30. Punctul de mijloc este $M(x_P + 1, y_P - \frac{1}{2})$. Dacă a fost ales E atunci noul punct de mijloc este $M_E(x_P + 2, y_P - \frac{1}{2})$ iar în cazul alegerii SE noul punct de mijloc este $M_{SE}(x_P + 2, y_P - \frac{3}{2})$.

Dacă punctul M este situat în interiorul cercului atunci pixelul E este mai apropiat de cerc și va fi selectat iar dacă M este situat în exteriorul cercului atunci pixelul SE este mai apropiat de cerc și va fi selectat.

Proprietatea 2.2

Fie $M(x', y')$. Atunci :

1. $M \in$ cercului de rază R centrat în origine ddacă $F(x', y') = 0$,
2. M este situat în interiorul cercului de rază R centrat în origine ddacă $F(x', y') < 0$,
3. M este situat în exteriorul cercului de rază R centrat în origine ddacă $F(x', y') > 0$,

unde $F(x, y) = x^2 + y^2 - R^2$.

Demonstrație

Ecuația cercului de rază R centrat în origine este $F(x, y) = 0$, unde $F(x, y) = x^2 + y^2 - R^2$.

1. Evident.
2. Originea este situată în interiorul cercului și deci un punct din interiorul cercului $M(x', y')$ are proprietatea că $\operatorname{sgn}(F(x', y')) = \operatorname{sgn}(F(0, 0)) = \operatorname{sgn}(-R^2) = -1$ și deci $F(x', y') < 0$.
3. Prin excluderea celor două cazuri anterioare.

□

Ca și în cazul segmentului de dreaptă considerăm o variabilă de decizie $d \doteq F(M) = F(x_P + 1, y_P - \frac{1}{2})$. Primul pixel din octantul al doilea care este generat de algoritm are coordonatele $(0, R)$ deci valoarea inițială a variabilei de decizie este $F(0+1, R-\frac{1}{2}) = F(1, R-\frac{1}{2}) = (1) + (R^2 - R + \frac{1}{4}) - R^2 = \frac{5}{4} - R$.

Dacă $d < 0$ atunci M este în interiorul cercului și deci alegem pixelul E. În acest caz noua valoare a variabilei de decizie este $d' = F(M_E) = F(x_P + 2, y_P - \frac{1}{2}) = (x_P + 2)^2 + (y_P - \frac{1}{2})^2 - R^2 = (x_P + 1)^2 + (y_P - \frac{1}{2})^2 - R^2 + 2 \cdot x_P + 3 = d + 2 \cdot x_P + 3$.

Dacă $d > 0$ atunci M este în exteriorul cercului și deci alegem pixelul SE. În acest caz noua valoare a variabilei de decizie este $d' = F(M_{SE}) = F(x_P + 2, y_P - \frac{3}{2}) = (x_P + 2)^2 + (y_P - \frac{3}{2})^2 - R^2 = (x_P + 1)^2 + (y_P - \frac{1}{2})^2 - R^2 + 2 \cdot x_P + 3 - 2 \cdot y_P + 2 = d + 2 \cdot x_P - 2 \cdot y_P + 5$.

Pe baza observațiilor de mai sus putem descrie o versiune preliminară a algoritmului de afișare a cercului care utilizează tehnica punctului de mijloc. Este o versiune care va fi rafinată prin eliminarea valorilor de tip **float** sau **double** (valori posibile pentru variabila de decizie d) precum și prin eliminarea operațiilor de înmulțire (e.g., $d' = d + 2 \cdot x_P + 3$ sau $d' = d + 2 \cdot x_P - 2 \cdot y_P + 5$).

Algoritm 6 AfisareCerc3

Date de intrare : $R \in \mathbf{Z}$, raza unui cerc centrat în origine,

Date de ieșire : $M \in \text{list of } (\mathbf{Z} \times \mathbf{Z})$, o secvență de pixeli care aproximează cel mai bine octantul al II-lea al cercului de rază R centrat în origine. Prin simetrie față de Oy, O, Ox și față de bisectoarea $y = x$ este afișat cercul în întregime.

Metodă :

```

procedure AfisarePuncteCerc3 (x,y : Z, var M : list of (Z x Z))
{
  M = M . (x,y) . (-x,-y) . (-x,y) . (x,-y);
  if (x != y)
    M = M . (y,x) . (-y,-x) . (-y,x) . (y,-x);
}
procedure AfisareCerc3 (R : Z, var M : list of (Z x Z))
{
  int x = 0, y = R;
  double d = 5.0 / 4 - R;
  M = \epsilon;
  AfisarePuncteCerc3(x, y, M);
  while (y > x)

```

```

{
    // conditia y > x caracterizeaza octantul al 2-lea
    if (d < 0)
    {
        // alegem pixelul E
        d += 2 * x + 3;
        x++;
    }
    else
    {
        // alegem pixelul SE
        d += 2 * (x - y) + 5;
        x++;
        y--;
    }
    AfisarePuncteCerc3(x, y, M);
}
}

```

În algoritmul precedent aducem următoarele modificări : mai întâi înlocuim variabila de decizie d cu o nouă variabilă de decizie h cu proprietatea că valorile lui h sunt doar de tip **int** (se observă că valorile variabilei de decizie d sunt de tip **double** : valoarea inițială $\frac{5}{4} - R$ este de tip **double** iar valorile ulterioare se obțin prin incrementarea cu valori de tip **int**).

Fie $h = d - \frac{1}{4}$. Atunci valoarea inițială a lui h este $\frac{5}{4} - R - \frac{1}{4} = 1 - R \in \mathbf{Z}$. Observăm că instrucțiunile de incrementare $d += 2 * x + 3$; și $d += 2 * (x - y) + 5$; nu se modifică (deoarece, de exemplu, putem rescrie instrucțiunea $d += 2 * x + 3$; sub forma $d = d + 2 * x + 3$; și efectuând înlocuirea în membrii stâng/drept a variabilei d cu $h + \frac{1}{4}$ vom obține $h + 1/4 = h + 1/4 + 2 * x + 3$; și deci noua instrucțiune este $h = h + 2 * x + 3$; deci $h += 2 * x + 3$;).

Avem deasemeni și faptul $d < 0 \iff h < 0$ ($d < 0 \iff h < -\frac{1}{4}$ dar valoarea inițială a lui h este din \mathbf{Z} și deasemeni și valorile ulterioare și deci rezultă că $h \in \mathbf{Z}$). Avem deci $h \in \mathbf{Z} \wedge h < -\frac{1}{4}$ și deci $h < 0$.

În concluzie, în algoritmul 6, pag. 31 este suficient să modificăm instrucțiunea de inițializare a variabilei de decizie d din **double** $d = 5.0 / 4 - R$; în **int** $d = 1 - R$; . Mai rămân operațiile de înmulțire. Acestea sunt eliminate în cele ce urmează. Utilizăm următoarele notații : $d_E \doteq 2 \cdot x_P + 3$, și $d_{SE} \doteq 2 \cdot (x_P - y_P) + 5$. Observăm că expresiile d_E și d_{SE} depind de pixelul curent $P(x_P, y_P)$. Valorile inițiale³ ale acestor expresii sunt 3 și $-2 \cdot R + 5$. În cazul alegerii pixelului E pixelul curent va avea coordonatele $(x_P + 1, y_P)$. Valorile expresiilor d_E și d_{SE} se modifică astfel : $d'_E = 2 \cdot (x_P + 1) + 3 = 2 \cdot x_P + 3 + 2 = d_E + 2$, $d'_{SE} = 2 \cdot (x_P + 1 - y_P) + 5 = 2 \cdot (x_P - y_P) + 5 + 2 = d_{SE} + 2$. În cazul alegerii pixelului SE pixelul curent va avea coordonatele $(x_P + 1, y_P - 1)$. Valorile expresiilor d_E și d_{SE} se modifică astfel : $d'_E = 2 \cdot (x_P + 1) + 3 = 2 \cdot x_P + 3 + 2 = d_E + 2$, $d'_{SE} = 2 \cdot (x_P + 1 - y_P + 1) + 5 = 2 \cdot (x_P - y_P) + 5 + 4 = d_{SE} + 4$.

³pixelul curent în acest caz este pixelul inițial $P(0, R)$

În concluzie, algoritmul de afișare al cercurilor care utilizează tehnica punctului de mijloc (și care utilizează doar operații de adunare și valori de tip **int**) este :

Algoritm 7 AfisareCerc4

Date de intrare : $R \in \mathbf{Z}$, raza unui cerc centrat în origine,

Date de ieșire : $M \in \text{list of } (\mathbf{Z} \times \mathbf{Z})$, o secvență de pixeli care aproximează cercul.

Metodă :

```
procedure AfisareCerc4 (R : Z, var M : list of (Z x Z))
{
    int x = 0, y = R;
    int d = 1 - R;
    int dE = 3, dSE = -2 * R + 5;
    M = \epsilon;
    AfisarePuncteCerc3(x, y, M);
    while (y > x)
    {
        if (d < 0)
        {
            d += dE;
            dE += 2;
            dSE += 2;
        }
        else
        {
            d += dSE;
            dE += 2;
            dSE += 4;
            y--;
        }
        x++;
        AfisarePuncteCerc3(x, y, M);
    }
}
```

2.3 Afișarea elipselor

Vom prezenta algoritmi de afișare a unor elipse având centrul în origine și semiaaxe $a, b \in \mathbf{Z}$. Ecuția unei astfel de elipse este $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$ și o rescriem sub forma $F(x, y) = 0$, unde $F(x, y) = b^2 \cdot x^2 + a^2 \cdot y^2 - a^2 \cdot b^2$.

Și în cazul elipselor putem aplica algoritmi de genul algoritmului 5, pag. 29 pentru cercuri, în acest caz utilizând coordonatele polare : $\begin{cases} x = a \cdot \cos t \\ y = b \cdot \sin t \end{cases}$, unde $0 \leq t < 2\pi$.

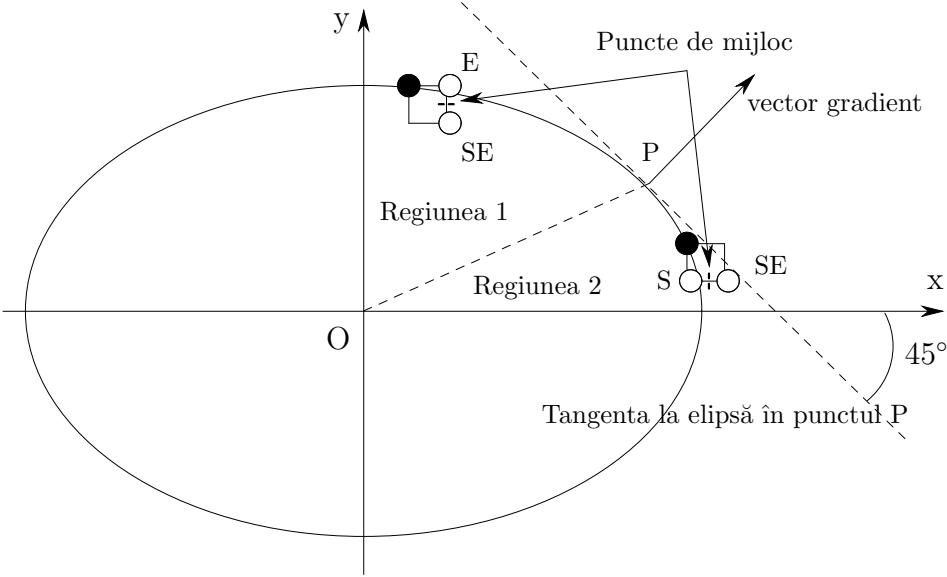


Figura 2.4: Alegerea unuia dintre pixelii E/SE sau S/SE

Spre deosebire de algoritmii de afișare ai cercurilor, în cazul elipsei nu există simetria față de bisectoarea $y = x$. Deci vom efectua afișarea pixelilor în cadranul I și, prin simetrie față de axele Ox , Oy , respectiv origine, afișarea pixelilor din celelalte cadrane.

Afișarea elipsei în cadranul I (vezi figura 2.4, pag. 34) se face în funcție de una din cele două regiuni. În regiunea 1 trebuie să alegem, la un moment dat, între pixelii E și SE iar în regiunea 2 între pixelii S și SE. Alegerea se face în funcție de poziția punctului M situat la mijlocul distanței dintre pixelii între care se face alegerea. Delimitarea celor două regiuni este realizată de către punctul P cu proprietatea că vectorul gradient al elipsei în P (care este perpendicular pe tangentă la elipsă în P) formează un unghi de 45° cu axa Ox (sau o formulare echivalentă : punctul P are proprietatea că tangentă la elipsă în P formează cu axa Ox un unghi de 45°).

Definiția 2.1

Fie curba dată prin ecuația implicită $F(x, y) = 0$. Vectorul gradient al curbei este : $\text{grad}F(x, y) = \frac{\partial F}{\partial x} \cdot \vec{i} + \frac{\partial F}{\partial y} \cdot \vec{j}$. ■

În cazul elipsei vectorul gradient este : $\text{grad}F(x, y) = 2 \cdot b^2 \cdot x \cdot \vec{i} + 2 \cdot a^2 \cdot y \cdot \vec{j}$. În regiunea 1 panta vectorului gradient este > 1 iar în regiunea 2 este < 1 sau, echivalent, în regiunea 1 componenta corespunzătoare vesorului \vec{j} este mai mare decât componenta corespunzătoare vesorului \vec{i} și în regiunea 2 viceversa. Deci un punct P care aparține elipsei se află în regiunea 1 dacă $a^2 \cdot y > b^2 \cdot x$ și se află în regiunea 2 dacă $a^2 \cdot y < b^2 \cdot x$.

În fiecare regiune evaluăm $F(x, y) = b^2 \cdot x^2 + a^2 \cdot y^2 - a^2 \cdot b^2$ în fiecare punct de mijloc $M(x, y)$ și determinăm poziția acestuia față de elipsă :

- dacă $F(x, y) = 0$ atunci M aparține elipsei,
- dacă $F(x, y) < 0$ atunci M se găsește (ca și originea de altfel : $F(0, 0) = -a^2 \cdot b^2 < 0$) în interiorul elipsei,

- dacă $F(x, y) > 0$ atunci M se găsește în exteriorul elipsei.

În regiunea 1 dacă punctul de mijloc M se găsește în interiorul elipsei atunci este selectat pixelul E (deoarece este cel mai apropiat de elipsă) iar dacă M se găsește în exteriorul elipsei atunci este selectat pixelul SE.

În regiunea 2 dacă punctul de mijloc M se găsește în interiorul elipsei atunci este selectat pixelul SE altfel este selectat pixelul S.

Trecerea într-o altă regiune (din 1 în 2) se face când punctul de mijloc $M(x, y)$ nu mai aparține regiunii, deci când $a^2 \cdot y \leq b^2 \cdot x$.

Vom folosi următoarele notații : pixelul curent (cel deja selectat) este $P(x_P, y_P)$, punctul de mijloc în regiunea 1 este $M(x_P + 1, y_P - \frac{1}{2})$ iar în regiunea 2 este $M(x_P + \frac{1}{2}, y_P - 1)$.

Variabila de decizie în regiunea 1 este $d_1 = F(x_P + 1, y_P - \frac{1}{2})$ iar în regiunea 2 este $d_2 = F(x_P + \frac{1}{2}, y_P - 1)$.

Valoarea inițială a variabilei de decizie în regiunea 1 se obține pentru $P(0, b)$ și $M(0 + 1, b - \frac{1}{2})$: $d_1^{\text{init}} = F(1, b - \frac{1}{2}) = b^2 - a^2 \cdot b + \frac{a^2}{4}$. În regiunea 2 valoarea inițială a variabilei de decizie este $d_2^{\text{init}} = F(x_P + \frac{1}{2}, y_P - 1)$ (pentru primul punct de mijloc care aparține regiunii 2).

Valorile variabilelor de decizie se calculează incremental astfel :

- în regiunea 1 dacă $d_1 < 0$ alegem E, altfel SE. Dacă am ales pixelul E atunci $d'_1 = F(x_P + 2, y_P - \frac{1}{2}) = d_1 + b^2 \cdot (2 \cdot x_P + 3)$. Dacă am ales pixelul SE atunci $d'_1 = F(x_P + 2, y_P - \frac{3}{2}) = d_1 + b^2 \cdot (2 \cdot x_P + 3) + a^2 \cdot (-2 \cdot y_P + 2)$.
- în regiunea 2 dacă $d_2 < 0$ alegem SE, altfel S. Dacă am ales pixelul SE atunci $d'_2 = F(x_P + \frac{3}{2}, y_P - 2) = d_2 + b^2 \cdot (2 \cdot x_P + 2) + a^2 \cdot (-2 \cdot y_P + 3)$. Dacă am ales pixelul S atunci $d'_2 = F(x_P + \frac{1}{2}, y_P - 2) = d_2 + a^2 \cdot (-2 \cdot y_P + 3)$.

Obținem astfel algoritmul de afișare utilizând tehnica punctului de mijloc (algoritm propus de Pitteway în 1967, van Aken în 1984 și Kappel în 1985) :

Algoritm 8 AfișareElipsă

Date de intrare : $a, b \in \mathbb{Z}$, semiaxele unei elipse centrate în origine.

Date de ieșire : $M \in \text{list of } (\mathbb{Z} \times \mathbb{Z})$, o secvență de pixeli care aproximează cel mai bine elipsa.

Metodă :

```

procedure AfisarePuncteElipsa (x,y : Z, var M : list of (Z x Z))
{
    M = M . (x,y) . (-x,-y) . (-x, y) . (x,-y);
}
procedure AfisareElipsa (a,b : Z, var M : list of (Z x Z))
{
    int x = 0, y = b;
    double d1 = b*b - a*a*b + a*a/4.0;
    double d2;
    M = \epsilon;
    AfisarePuncteElipsa(x, y, M);
    while (a*a*(y - 0.5) > b*b*(x+1))

```

```

{
    // atat timp cat punctul de mijloc apartine regiunii 1
    if (d1 < 0)
    {
        // este selectat E
        d1 += b*b*(2*x+3);
        x++;
    }
    else
    {
        // este selectat SE
        d1 += b*b*(2*x+3) + a*a*(-2*y+2);
        x++;
        y--;
    }
    AfisarePuncteElipsa(x, y, M);
}

// trecem in regiunea 2 si initializam variabila de decizie
d2 = b*b*(x+0.5)*(x+0.5) + a*a*(y-1)*(y-1) - a*a*b*b;
while (y > 0)
{
    if (d2 < 0)
    {
        // este selectat SE
        d2 += b*b*(2*x+2) + a*a*(-2*y+3);
        x++;
        y--;
    }
    else
    {
        // este selectat S
        d2 += a*a*(-2*y+3);
        y--;
    }
    AfisarePuncteElipsa(x, y, M);
}
}

```

.

2.4 Colorarea uniformă a dreptunghiurilor

Problema care se pune este următoarea : să presupunem că pe grila carteziană (a se vedea figura 2.5, pag. 37) avem suprapus un dreptunghi ale căruia vârfuri sunt date prin intersecția dreptelor $x = x_{\min}$, $x = x_{\max}$, $y = y_{\min}$, $y = y_{\max}$, unde $x_{\min}, x_{\max}, y_{\min}, y_{\max} \in \mathbf{Z}$. Se cere să se determine și să se coloreze uniform pixelii din interiorul dreptunghiului.

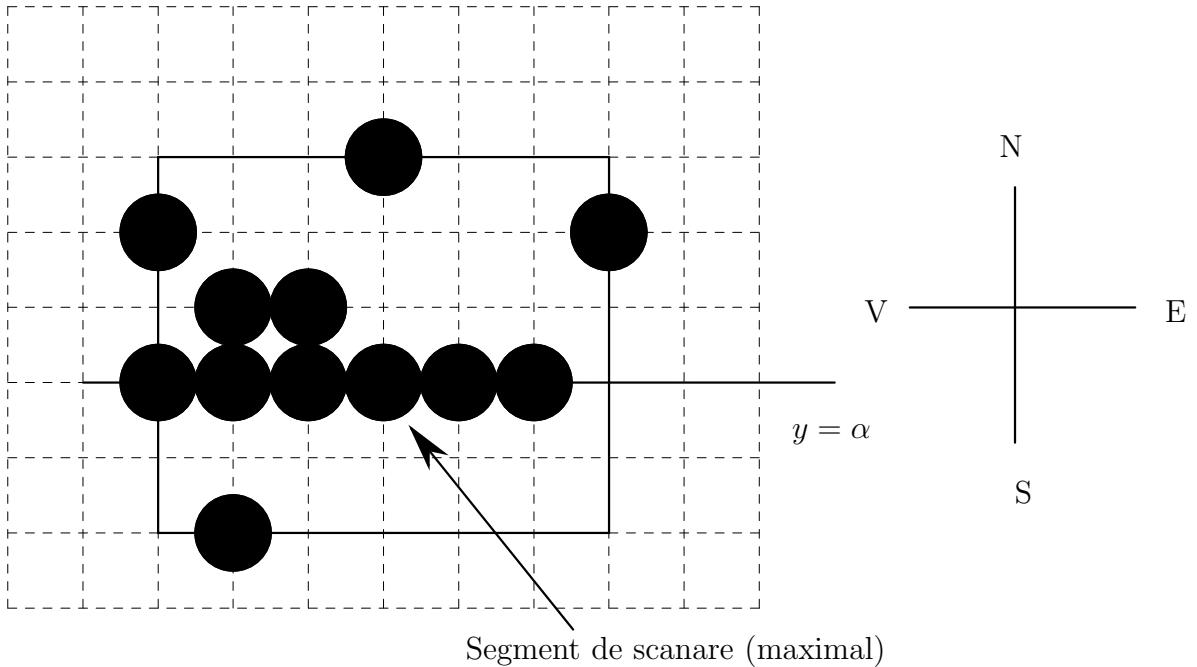


Figura 2.5: Pixelii interiori

Așa cum se poate observa în figura 2.5, pag. 37 determinarea pixelilor strict interiori este evidentă (sunt acei pixeli $P(x_P, y_P)$ cu proprietatea că $x_{\min} < x_P < x_{\max}$ și că $y_{\min} < y_P < y_{\max}$). În cazul pixelilor de pe frontieră dreptunghiului (cea de nord, sud, est sau vest) recurgem la următoarea convenție : pixelii de pe frontieră de nord și de est nu sunt interiori dreptunghiului iar cei de pe frontieră de vest și de sud sunt⁴.

Această convenție se impune deoarece colorarea pixelilor trebuie să se realizeze prin apelarea *o singură dată* a primitivei `writePixel(int x, int y, int valoare)` care realizează colorarea unui pixel având coordonatele (x, y) . Să presupunem că nu am fi respectat convenția sus-menționată. Atunci în cazul a două dreptunghiuri care partajează o muchie am fi în situația în care procedura `writePixel` ar fi apelată de două ori.

Având în vedere cele expuse mai sus algoritmul care efectuează colorarea uniformă a pixelilor interiori unui dreptunghi este :

Algoritm 9 Colorare uniformă dreptunghi

Date de intrare : $x_{\min}, x_{\max}, y_{\min}, y_{\max} \in \mathbf{Z}$ și $val \in \mathbf{Z}$ parametrul care determină culoarea pixelilor.

Date de ieșire : Algoritmul colorează pixelii interiori dreptunghiului cu culoarea val .

Metodă :

```
procedure ConvScanDreptunghi (xm,xM,ym,yM,val : Z)
{
    for (int x = xm; x < xM; x++)

```

⁴În general, regula este următoarea : un pixel situat pe o muchie nu este pixel interior dacă semiplanul definit de muchia respectivă și care conține obiectul se află în stânga sau sub muchie

```

    for (int y = ym; y < yM; y++)
        writePixel(x,y,val);
}

```

2.5 Colorarea uniformă a poligoanelor

Algoritmul care va fi prezentat poate fi aplicat nu numai poligoanelor convexe ci și celor concave sau celor care au găuri. Acest algoritm calculează pixelii interiori ai unui poligon utilizând intersecția unor drepte de scanare cu poligonul (determinând astfel segmente de scanare maximale⁵).

Un exemplu de segment de scanare maximal este prezentat în figura 2.5, pag. 37 : dreapta de scanare $y = \alpha$, $\alpha \in \mathbf{Z}$, intersectează dreptunghiul iar pixelii interiori sunt cei reprezentați în figură.

Un alt exemplu este cel din figura 2.6, pag. 39 : în acest caz dreapta de scanare $y = 8$ intersectează poligonul⁶, respectiv muchiile AF , EF , DE , CD în punctele $a(2, 8)$, $b(\frac{9}{2}, 8)$, $c(\frac{17}{2}, 8)$ și $d(13, 8)$. Se poate observa că pixelii strict interiori poligonului de pe dreapta de scanare $y = 8$ sunt cei marcați în figură, cu excepția pixelului $(13, 8)$ care nu este considerat interior conform convenției precizate anterior. Pentru același poligon $ABCDEF$, pixelii interiori sunt prezențați în figura 2.7, pag. 40. Pixelii interiori sunt marcați utilizând simbolul ●, prin simbolul ■ am marcat câțiva pixeli care sunt strict exteriori poligonului $ABCDEF$ iar prin simbolul ○ am marcat câțiva pixeli care, deși s-ar părea că sunt interiori poligonului, datorită convenției din secțiunea 2.4, pag. 36, nu sunt interiori poligonului $ABCDEF$.

Algoritmul are trei etape :

1. calculul intersecției unei drepte de scanare cu toate muchiile poligonului,
2. sortarea punctelor de intersecție crescător după coordonata x ,
3. colorarea tuturor pixelilor dintre două intersecții succesive, pixeli care sunt strict interiori poligonului. Având la dispoziție lista punctelor de intersecție dintre o dreaptă de scanare și poligon (listă ordonată crescător după coordonata x), pixelii interiori sunt cei dintre punctele de intersecție 1 și 2, dintre 3 și 4, dintre 5 și 6, etc. Trebuie clarificate următoarele aspecte :

- (a) Cum procedăm în cazul în care coordonata x a unui punct de intersecție nu are o valoare din \mathbf{Z} ? În acest caz acest punct are doi pixeli vecini. Pe care îl selectăm ? În acest caz⁷ procedăm astfel : în cazul punctelor de intersecție 1, 3, 5, etc. pixelul interior va avea coordonata $[x]$ iar în cazul punctelor de intersecție 2, 4, 6, etc. pixelul interior va avea coordonata $[x]$. De exemplu, în figura 2.6, pag. 39, cele 2 cazuri se referă la punctele de intersecție $b(\frac{9}{2}, 8)$ și $c(\frac{17}{2}, 8)$ (folosind notația noastră

⁵un segment de scanare maximal este acel segment de pe o dreaptă de scanare, complet inclus în interiorul obiectului și care este maximal cu această proprietate, i.e., nu există un alt segment de scanare care să-l includă

⁶ale cărui vârfuri sunt $A(2, 3)$, $B(7, 1)$, $C(13, 5)$, $D(13, 11)$, $E(7, 7)$ și $F(2, 9)$

⁷Vom presupune, fără a restrângere generalitatea, că ne aflăm în cadranul 1.

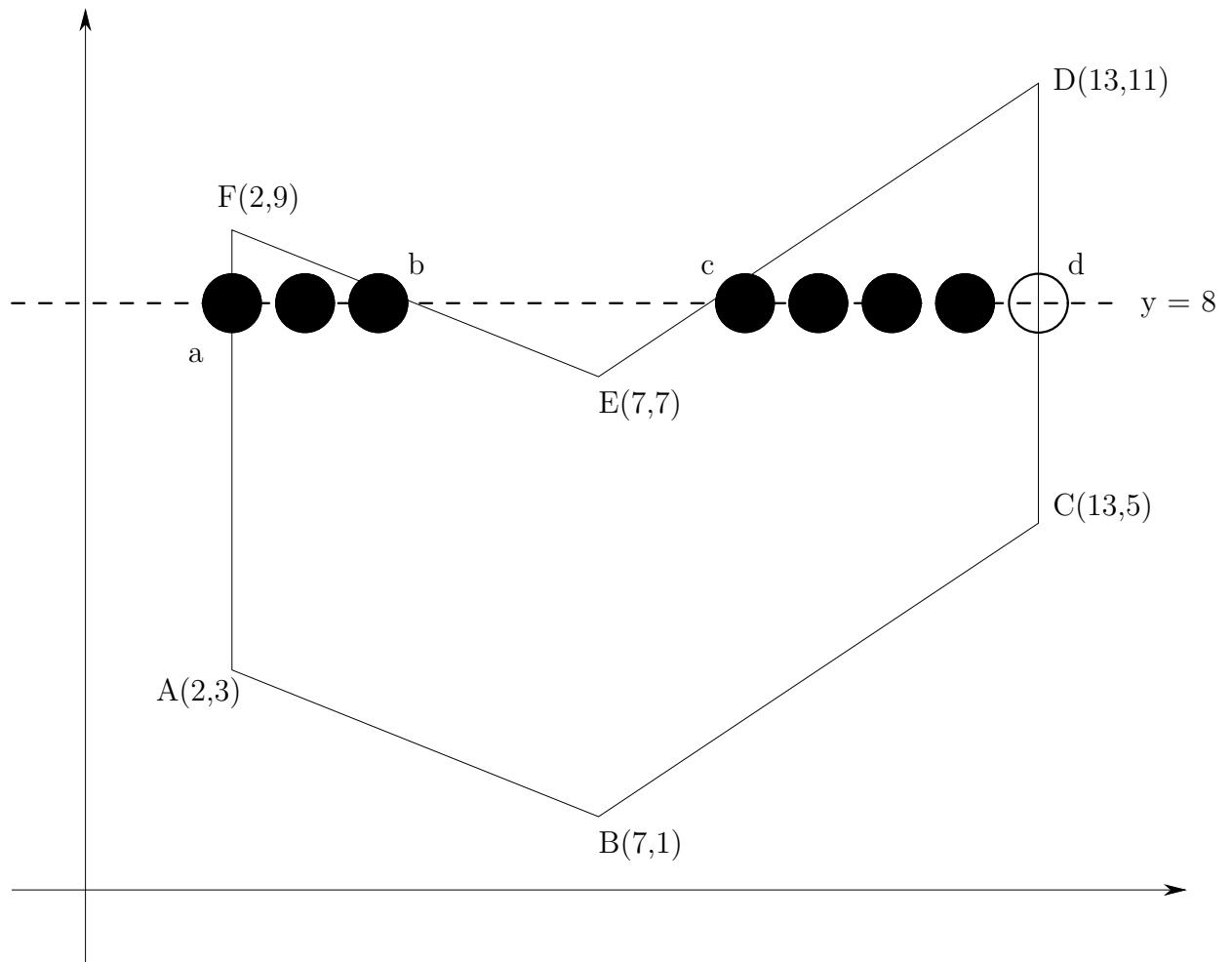


Figura 2.6: Colorarea uniformă a unui poligon. Un segment de scanare maximal.

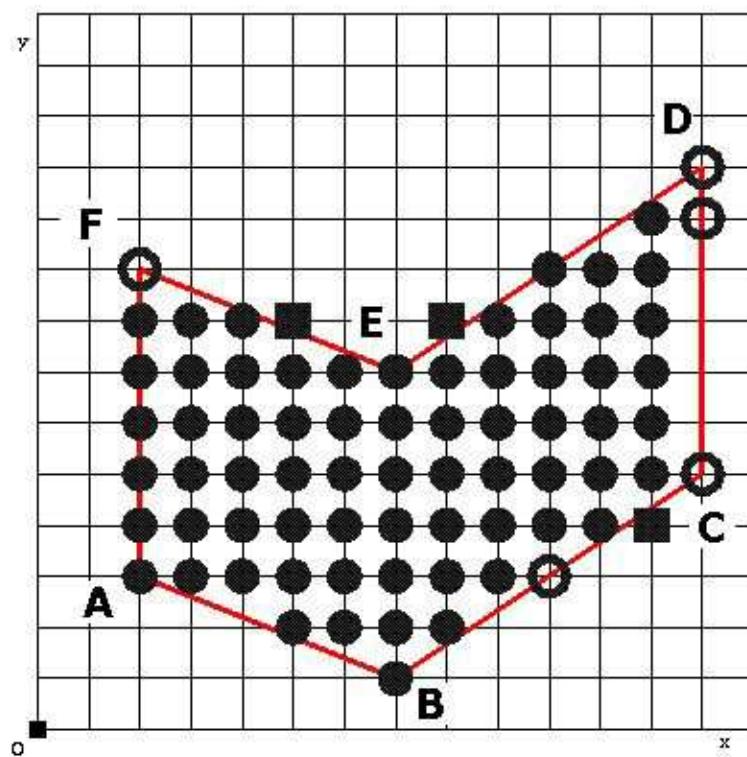


Figura 2.7: Pixeli interiori poligonului $ABCDEF$

ele sunt punctele 2 și 3). Conform regulii de mai sus, pixelul vecin al lui $b(\frac{9}{2}, 8)$ este $(4, 8)$ iar cel vecin al lui $c(\frac{17}{2}, 8)$ este $(9, 8)$ (așa cum de altfel se poate observa în figura 2.7, pag. 40).

- (b) Cum procedăm cu pixelii care sunt extremități ale unui segment de scanare maximal? Așa cum am precizat în secțiunea 2.4, pag. 36 pixelul din extremitatea stângă este considerat interior poligonului iar cel din extremitatea dreaptă nu este considerat interior.
- (c) Cum procedăm cu pixelii care sunt în același timp și vârfuri ale poligonului? De ce, de exemplu, în figura 2.7, pag. 40 pixelii A , B și E sunt interiori poligonului dar pixelii C , D , F nu? Vom numi un vârf al unei muchii y_{\min} dacă este vârful care are coordonata y minimă (dintre cele două vârfuri ale muchiei) iar y_{\max} altfel. Regula este: dacă un pixel corespunde unui vârf care este vârf y_{\min} al cel puțin uneia dintre cele două muchii care îl partajează atunci pixelul este considerat interior. În cazul poligonului $ABCDEF$ vârful A este y_{\min} pentru muchia AF și y_{\max} pentru AB , vârful B este y_{\min} pentru muchiile AB și BC , vârful C este y_{\min} pentru muchia CD și y_{\max} pentru muchia BC (dar, aplicând regula 3b, pag. 41 în prealabil, a fost etichetat extern), vârful D este y_{\max} pentru muchiile CD și DE , vârful E este y_{\min} pentru muchiile DE și EF , vârful F este y_{\max} pentru muchiile AF și EF . În concluzie pixelii A , B și E sunt interiori poligonului iar pixelii C , D , F nu sunt interiori poligonului.
- (d) Cum procedăm cu pixelii situați pe muchiile orizontale ale poligonului? De ce, de exemplu, în cazul figurii 2.9, pag. 42 pixelii de pe muchiile AB , respectiv CD sunt interiori iar cei de pe muchia HG nu? Sau în cazul figurii 2.8, pag. 42 de ce pixelii de pe muchia BC nu sunt interiori triunghiului? Vom proceda astfel: vom considera intersecțiile dreptelor de scanare doar cu muchiile care nu sunt orizontale. Vom elibera acele intersecții cu vârfuri care sunt doar y_{\max} . De exemplu, în cazul poligonului $ABCDEFGHI$ (figura 2.9, pag. 42):
 - i. dreapta de scanare $y = 10$ intersectează muchiile HI , FG , EF în $H(1, 10)$, $G(8, 10)$, $(10, 10)$. Remarcăm că vârful H este doar vârf de tip y_{\max} . Eliminăm acest vârf și vom obține doar punctele de intersecție $G(8, 10)$ și $(10, 10)$ și vom proceda ca mai sus.
 - ii. dreapta de scanare $y = 6$ intersectează muchiile HI , BC , DE în $(1, 6)$, $C(8, 6)$, $D(10, 6)$. Remarcăm că vârful C este doar vârf de tip y_{\max} . Eliminăm acest vârf și vom obține doar punctele de intersecție $(1, 6)$, și $D(10, 6)$ și vom proceda ca mai sus.

Regula pe care am aplicat-o în acest caz este conformă cu convenția stabilită în secțiunea 2.4, pag. 36.

În cazul unor poligoane cu unghiuri foarte ascuțite (de exemplu triunghiul din figura 2.8, pag. 42) se recomandă și afișarea pixelilor de pe muchii sau chiar din exteriorul poligonului dar cu intensități care variază în funcție de distanța dintre pixel și cea mai apropiată muchie.

Algoritm 10 Afisare poligon

Date de intrare : Un poligon specificat prin multimea muchiilor sale.

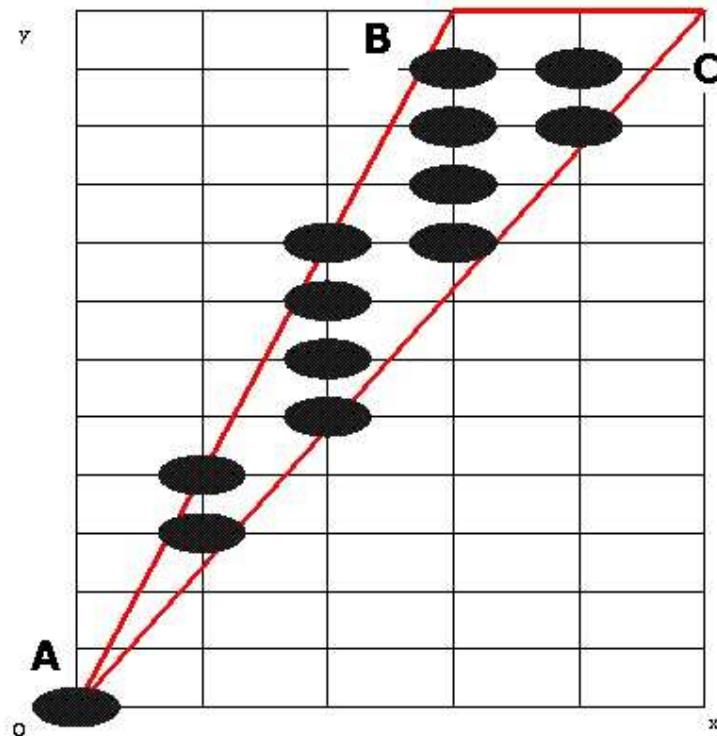


Figura 2.8: Pixelii interiori triunghiului ABC

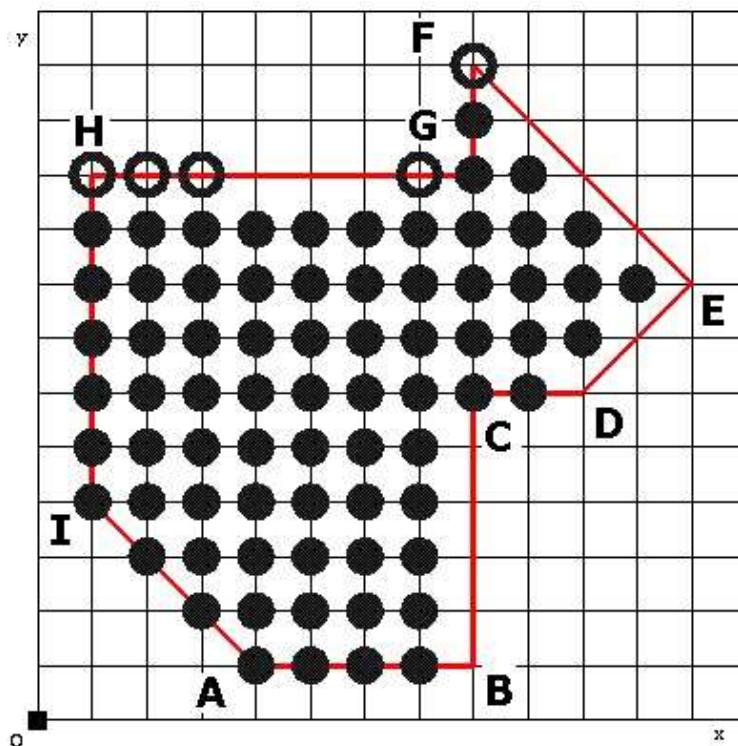


Figura 2.9: Pixelii interiori poligonului $ABCDEFGHI$

Date de ieșire : $M \in \text{list of } (\mathbf{Z} \times \mathbf{Z})$, mulțimea pixelilor interiori poligonului.

Metodă : Se aplică procedura *InitializareET* pentru inițializarea unei tabele de muchii, o structură de date ce conține, pentru fiecare dreaptă de scanare, punctele de intersecție ale acestei drepte cu muchiile neorizontale ale poligonului. Procedura *calculism* calculează incremental, plecând de la tabela de muchii, segmentele de scanare maximale, pentru fiecare dreaptă de scanare.

```
type
    VARF = struct S1 { int x, y; };
    MUCHIE = struct S2 { VARF vi, vf; };
    POLIGON = set of MUCHIE;
    INTERSECTIE = struct S3 { int ymax; double xmin; double ratio} ;
    INTERSECTII = list of INTERSECTIE;
    DOM_SCAN = 0 .. 100;
    ET = DOM_SCAN -> INTERSECTII;

procedure initializareET(p : POLIGON, var et : ET)
{
    int xm,ym,xM,yM;
    bool change;

    for (i in DOM_SCAN) et(i) = [ ];
    for (m in p)
    {
        // pentru fiecare muchie din poligon ...
        if (m.vi.y != m.vf.y)
        {
            // ... care nu este orizontală
            ym = min (m.vi.y, m.vf.y);
            yM = max (m.vi.y, m.vf.y);
            xm = (ym == m.vi.y) ? m.vi.x : m.vf.x;
            xM = (yM == m.vi.y) ? m.vi.x : m.vf.x;
            et(ym) = et(ym) . [(yM, xm, (xm - xM)/(ym - yM))];
        }
    }
    /*
        sortarea în ordine crescătoare conform cu
        xm a intersecțiilor dreptei de scanare cu
        muchiile poligonului
    */
    for (i in DOM_SCAN)
    {
        do
        {
            change = false;
            if (|et(i)| == 0) break;
            for (j in et(i))
                if (et(j).ratio < et(i).ratio)
                    et(i) = et(i) . et(j);
        }
        while (change);
    }
}
```

```

        for (j = 1; j < |et(i)|; j++)
        {
            if (et(i)(j).xmin > et(i)(j+1).xmin)
            {
                interschimba et(i)(j) <-> et(i)(j+1);
                change = true;
            }
        }
    } while (change)
}
}

procedure calculssm(p : POLIGON, et : ET, var ssm : ET)
{
    INTERSECTII aet;
    int y, k;

    for (i in DOM_SCAN) ssm(i) = [ ];

    // lui y i se atribuie o valoare care nu este din DOM_SCAN
    y = -1;
    // se determina y = min {y' | et(y') != \emptyset}
    for (i in DOM_SCAN)
    {
        if (! et(i).empty())
        {
            y = i;
            break;
        }
    }
    if (! y in DOM_SCAN) return;

    aet = [ ];
    do
    {
        aet = aet . et(y);

        // eliminarea varfurilor cu ymax == y
        for (i = 1; i < |aet|; i++)
        {
            if (aet(i).ymax == y)
                aet.delete(aet(i));
        }
    }

    // sortarea aet cf. cheii xmin
}

```

```

k = |aet|;
while (k >= 2)
{
    for (i = 1; i < k; i++)
    {
        if (aet(i).xmin > aet(i+1).xmin)
            interschimba aet(i) <-> aet(i+1);
    }
    k--;
}

ssm(y) = aet;

y++;

// reactualizarea punctelor de intersectie pentru noua dreapta de scanare
for (i = 1; i < |aet|; i++)
{
    if (aet(i).ratia != 0)
        aet(i).xmin += aet(i).ratia;
}
} while (!aet.empty() || !et(y).empty())
}

procedure main()
{
    POLIGON p;
    ET et, ssm;

    read(p);
    initializareET(p, et);
    calculSSM(p, et, ssm);
    coloreaza(ssm);
}

```

Segmentele de scanare maximale obținute la ieșirea algoritmului 10, pag. 41 sunt colorate utilizând regulile 3a, pag. 38 și 3b, pag. 41.

2.6 Colorarea uniformă a cercurilor și a elipselor

Atât în cazul cercurilor cât și în cazul elipselor vom ține cont de simetria acestor obiecte geometrice. În cazul unui cerc este suficient să colorăm pixelii din cel de-al doilea octant și apoi, utilizând simetria față de axele Ox, Oy, față de originea O cât și față de prima bisectoare,

putem colora pixelii interiori cercului. În cazul unei elipse este suficient să colorăm pixelii din primul cadran și apoi, utilizând simetria față de axele Ox, Oy, și față de originea O, putem colora pixelii interiori elipsei.

Fără a restrânge generalitatea vom presupune că cercurile sunt centrate în origine și au raza R iar elipsele sunt centrate în origine și au semiaxele a, b .

În ambele cazuri algoritmii propuși calculează, în mod incremental, extremitățile segmentelor de scanare maximale din cerc, respectiv elipsă. Atât în cazul unui cerc cât și în cazul unei elipse vom utiliza o structură de date care să ne permită să memorăm una din extremitățile unui segment de scanare maximal. Cealaltă extremitate se obține, în cazul cercului, prin intersecția dreptelor de scanare $y = 0, y = 1, \dots, y = R$ cu dreapta $x = 0$ iar în cazul elipsei, prin intersecția dreptelor de scanare $y = 0, y = 1, \dots, y = b$ cu dreapta $x = 0$.

Algoritmii sunt similari (sunt incrementali și utilizează doar aritmetică întregilor) celor de conversie prin scanare prezentați în secțiunile anterioare.

În cazul cercului, să presupunem că pixelul curent este pixelul $P(x_P, y_P)$ și deci extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P$ este P . În momentul în care se va trece la un nou pixel curent va trebui să luăm două decizii :

1. dacă să reactualizăm sau nu extremitatea segmentului de scanare maximal a cărui extremitate este $P(x_P, y_P)$ (sau, mai bine zis, dacă pixelul E poate deveni extremitate în locul lui P),
2. să reactualizăm extremitatea segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P - 1$.

Ecuația cercului centrat în origine, de rază R este $F(x, y) = 0$, unde $F(x, y) = x^2 + y^2 - R^2$. Deoarece punctul P este extremitatea unui segment de scanare maximal atunci el este în interiorul sau pe cerc, deci $F(P) \leq 0$, unde $F(P) \doteq F(x_P, y_P)$. Putem calcula valorile $F(E)$ și $F(SE)$ plecând de la valoarea lui $F(P)$: $F(E) \doteq F(x_P + 1, y_P) = (x_P + 1)^2 + y_P^2 - R^2 = F(P) + 2 \cdot x_P + 1$ și $F(SE) \doteq F(x_P + 1, y_P - 1) = (x_P + 1)^2 + (y_P - 1)^2 - R^2 = F(P) + 2 \cdot x_P - 2 \cdot y_P + 2$.

Tinând cont de cele de mai sus :

1. Dacă $F(E) \leq 0$ (decă pixelul E este interior, a se vedea figura 2.10, pag. 47) atunci reactualizăm segmentul de scanare maximal corespunzător dreptei de scanare $y = y_P$ cu extremitatea E în loc de P . În acest caz pixelul E devine pixelul curent P .
2. Dacă $F(SE) \leq 0$ atunci actualizăm segmentul de scanare maximal corespunzător dreptei de scanare $y = y_P - 1$ cu extremitatea SE (pixelul SE este interior, a se vedea figura 2.10, pag. 47). În acest caz pixelul SE devine pixelul curent P .
3. Altfel (i.e., dacă $F(E) > 0$ și $F(SE) > 0$) înseamnă că atât E cât și SE sunt în exteriorul cercului și decă vom alege S în loc de P (deoarece este evident că dacă P este în interiorul cercului atunci pixelul S este tot în interiorul cercului : avem $F(P) \leq 0$, deci $x_P^2 + y_P^2 - R^2 \leq 0$, $F(S) \doteq F(x_P, y_P - 1) = x_P^2 + (y_P - 1)^2 - R^2 = F(P) - 2 \cdot y_P + 1$ și cum $1 - 2 \cdot y_P < 0$ deoarece $y_P \geq 1 > \frac{1}{2}$ avem decă $F(S) \leq 0$).

În cazul elipsei problema este similară dar puțin mai complicată deoarece în acest caz raționamentul trebuie făcut pentru tot cadranul întâi pentru cele două regiuni identificate la algoritmul de conversie prin scanare al unei elipse.

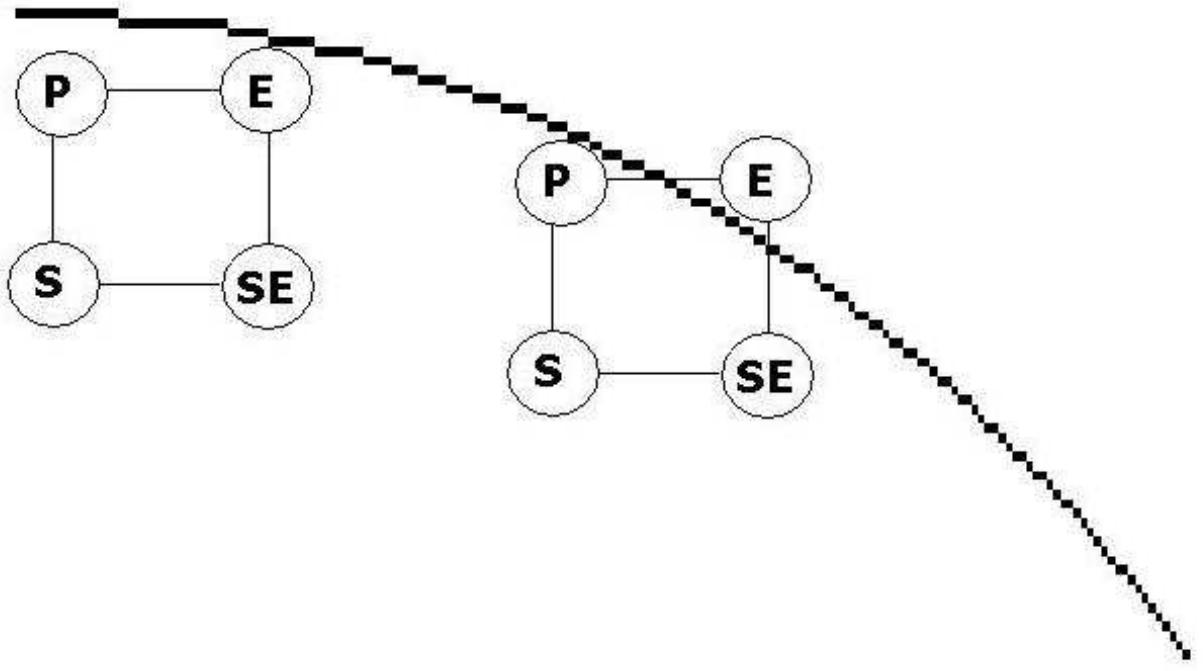


Figura 2.10: Cazuri posibile în al doilea octant al unui cerc

Ecuația elipsei de semiaxe a, b centrate în origine este $F(x, y) = 0$, unde $F(x, y) = b^2 \cdot x^2 + a^2 \cdot y^2 - a^2 \cdot b^2$. Regiunea 1 este acea regiune din primul cadran caracterizată de inecuația $a^2 \cdot y > b^2 \cdot x$ iar regiunea 2 de inecuația $a^2 \cdot y < b^2 \cdot x$. Un punct $A(x, y)$ este interior elipsei dacă $F(A) < 0$.

În prima regiune să presupunem că pixelul curent este pixelul $P(x_P, y_P)$ și deci extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P$ este P . Avem mai multe cazuri :

1. Dacă pixelul $E(x_P + 1, y_P)$ este interior elipsei atunci va trebui să reactualizăm extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P$. Ea devine E . În acest caz pixelul E devine următorul pixel curent.
2. Altfel, dacă pixelul $SE(x_P + 1, y_P - 1)$ este interior elipsei atunci va trebui să reactualizăm extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P - 1$. Ea devine SE . În acest caz pixelul SE devine următorul pixel curent.
3. Altfel, dacă pixelul $SE(x_P + 1, y_P - 1)$ este exterior elipsei atunci va trebui să reactualizăm extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P - 1$. Ea devine S . În acest caz pixelul S devine următorul pixel curent.

În a doua regiune să presupunem că pixelul curent este pixelul $P(x_P, y_P)$ și deci extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P$ este P . Avem mai multe cazuri :

1. Dacă pixelul $SE(x_P + 1, y_P - 1)$ este interior elipsei atunci va trebui să reactualizăm extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P - 1$. Ea devine SE . În acest caz pixelul SE devine următorul pixel curent.
2. Altfel, dacă pixelul $SE(x_P + 1, y_P - 1)$ este exterior elipsei atunci va trebui să reactualizăm extremitatea dreaptă a segmentului de scanare maximal corespunzător dreptei de scanare $y = y_P - 1$. Ea devine S (este evident că dacă P este în interiorul elipsei atunci S este și el în interiorul elipsei deoarece presupunând că $F(P) \leq 0$ atunci cum $F(S) = F(P) + a^2 \cdot (-2 \cdot y_P + 1)$ și cum $y_P \geq 1$ avem deci $F(S) \leq F(P) \leq 0$). În acest caz pixelul S devine următorul pixel curent.

Valorile $F(E)$, $F(SE)$, $F(S)$ sunt calculate în funcție de $F(P)$ astfel : în regiunea 1 $F(E) = F(P) + b^2 \cdot (2 \cdot x_P + 1)$ iar $F(SE) = F(P) + b^2 \cdot (2 \cdot x_P + 1) + a^2 \cdot (-2 \cdot y_P + 1)$. Valori similare se pot deduce și pentru a doua regiune.

Algoritmul care realizează umplerea unei elipse este :

Algoritmul 11 UmplereElipsa

Date de intrare : O elipsă centrată în origine de semiaxe întregi $a, b \in \mathbf{Z}$.

Date de ieșire : $M \in \text{list of } (\mathbf{Z} \times \mathbf{Z})$, multimea pixelilor interiori elipsei.

Metodă : O procedură C/C++ care implementează algoritmul descris succint mai sus este următoarea :

```
void umplereelipsa(int x0, int y0, int a, int b, double val,
                     void (*sablon) (int x, int y, double val,
                                      CGrilaCarteziana &cc))
{
    int xi = 0, x = 0, y = b;
    double fxpyp = 0.0;
    double deltaE, deltaSE, deltaS;

    ssm.vidare();
    ssm.adauga(y+y0, xi+x0, x+x0);

    // regiunea 1
    while (a*a*(y-0.5) > b*b*(x+1))
    {
        deltaE = b * b * ( 2 * x + 1 );
        deltaSE = b * b * ( 2 * x + 1 ) + a * a * (-2 * y + 1);
        if (fxpyp + deltaE <= 0.0)
        {
            // E este in interior
            fxpyp += deltaE;
            x++;
            ssm.setare(y+y0, xi+x0, x+x0);
        }
        else if (fxpyp + deltaSE <= 0.0)
```

```

{
    // SE este in interior
    fxpyp += deltaSE;
    x++;y--;
    ssm.adauga(y+y0,xi+x0, x+x0);
}
}

// regiunea 2
while (y > 0)
{
    deltaSE = b * b * ( 2 * x + 1 ) + a * a * (-2 * y + 1);
    deltaS = a*a*(-2 * y + 1);
    if (fxpyp + deltaSE <= 0.0)
    {
        // SE este in interior
        fxpyp += deltaSE;
        x++;y--;
    }
    else
    {
        // S este in interior
        fxpyp += deltaS;
        y--;
    }
    ssm.adauga(y+y0,xi+x0, x+x0);
}
ssm.desenare(val, *this, sablon);
}

```

După ce am determinat pixelii interiori unui cerc sau unei elipse în cadranul întâi, prin simetrie, putem determina pixelii interiori din celelalte cadrane.

Capitolul 3

Spații vectoriale

În cele ce urmează reamintim câteva noțiuni legate de spațiile vectoriale.

Definiția 3.1

Se numește *spațiu vectorial* tuplul $(K, V, +, \cdot)$, unde $+$ și \cdot sunt două funcții : $+ : V \times V \rightarrow V$ și $\cdot : K \times V \rightarrow V$.

Elementele mulțimii V se numesc *vectori* iar elementele mulțimii K se numesc *scalari*. Operațiile de adunare a vectorilor precum și de înmulțire a vectorilor cu scalari au următoarele proprietăți :

SV_1 adunarea vectorilor este asociativă : $(\forall \vec{u}, \vec{v}, \vec{w} \in V)((\vec{u} + \vec{v}) + \vec{w} = \vec{u} + (\vec{v} + \vec{w}))$,

SV_2 adunarea vectorilor este comutativă : $(\forall \vec{u}, \vec{v} \in V)(\vec{u} + \vec{v} = \vec{v} + \vec{u})$,

SV_3 în V există vectorul nul : $(\exists \vec{0} \in V)(\forall \vec{v} \in V)(\vec{v} + \vec{0} = \vec{v})$,

SV_4 existența vectorului opus : $(\forall \vec{v} \in V)(\exists -\vec{v} \in V)(\vec{v} + (-\vec{v}) = \vec{0})$,

SV_5 înmulțirea vectorilor cu scalari este asociativă : $(\forall a, b \in K)(\forall \vec{v} \in V)(a \cdot (b \cdot \vec{v}) = (a \cdot b) \cdot \vec{v})$,

SV_6 în K există scalarul unitate : $(\exists 1 \in K)(\forall \vec{v} \in V)(1 \cdot \vec{v} = \vec{v})$,

SV_7 adunarea vectorilor este distributivă : $(\forall a \in K)(\forall \vec{u}, \vec{v} \in V)(a \cdot (\vec{u} + \vec{v}) = a \cdot \vec{u} + a \cdot \vec{v})$,

SV_8 înmulțirea vectorilor cu scalari este distributivă : $(\forall a, b \in K)(\forall \vec{v} \in V)((a + b) \cdot \vec{v} = a \cdot \vec{v} + b \cdot \vec{v})$.

Exemplul 3.1

Un exemplu de spațiu vectorial este $(\mathbf{R}, \mathbf{R}^n, +, \cdot)$, unde $(a_1, a_2, \dots, a_n) + (b_1, b_2, \dots, b_n) = (a_1 + b_1, a_2 + b_2, \dots, a_n + b_n)$ și $a \cdot (a_1, a_2, \dots, a_n) = (a \cdot a_1, a \cdot a_2, \dots, a \cdot a_n)$. ■

În cele ce urmează definim un spațiu vectorial izomorf¹ cu $(\mathbf{R}, \mathbf{R}^3, +, \cdot)$: spațiul vectorial tridimensional (\mathbf{R}, V_3) .

¹vom arăta că există o bijecție între cele 2 spații vectoriale care păstrează operațiile : $f : \mathbf{R}^3 \rightarrow V_3$, $f(a, b, c) = a\vec{i} + b\vec{j} + c\vec{k}$, $f((a_1, a_2, a_3) + (b_1, b_2, b_3)) = f(a_1, a_2, a_3) + f(b_1, b_2, b_3)$, $f(a \cdot (a_1, a_2, a_3)) = a \cdot f(a_1, a_2, a_3)$.

Elementele multimii V_3 se definesc prin intermediul unei relații de echivalență R , definită pe mulțimea segmentelor orientate din plan (spațiu).

Un *segment orientat* este perechea (A, B) , unde A, B sunt puncte din plan (spațiu). Îl mai notăm și \overrightarrow{AB} . Dreapta AB este dreapta suport, A este originea iar B extremitatea segmentului orientat \overrightarrow{AB} .

Fie $\overrightarrow{AB}, \overrightarrow{CD}$ două segmente orientate. Atunci $(\overrightarrow{AB}, \overrightarrow{CD}) \in R$ dacă au aceeași direcție, sens și lungime, unde :

1. \overrightarrow{AB} și \overrightarrow{CD} au *aceeași direcție* dacă dreptele suport AB și CD coincid sau sunt paralele,
2. segmentele orientate \overrightarrow{AB} și \overrightarrow{CD} (care au aceeași direcție) au *același sens* dacă B și D sunt situate în același semiplan (al planului ABC) determinat de dreapta AC ,
3. *lungimea* (modulul, norma) segmentului orientat \overrightarrow{AB} (notată $|\overrightarrow{AB}|$) este distanța ($\in \mathbf{R}^+$) între punctele A și B .

Un *vector* \vec{v} este un reprezentant al unei clase de echivalență \bar{v} determinate de relația R . Se pot menționa câțiva vectori particulari :

1. vectorul nul $\vec{0}$ este reprezentantul clasei de echivalență $\{\overrightarrow{AA} | A \text{ punct din plan (spațiu)}\}$.
2. vectorul unitate (versor) este un vector \vec{v} cu proprietatea că $|\vec{v}| = 1$.

Definim în plan (spațiu) un punct O numit *origine*. Atunci, fiecare punct A din plan (spațiu) determină *vectorul de poziție* \overrightarrow{OA} . Se poate observa că putem defini o bijecție între mulțimea punctelor din plan (spațiu) și mulțimea vectorilor din V_3 .

Definim aplicațiile $+ : V_3 \times V_3 \rightarrow V_3$ și $\cdot : \mathbf{R} \times V_3 \rightarrow V_3$ după cum urmează :

1. Fie vectorii $\overrightarrow{PQ} \in \overrightarrow{AB}$ și $\overrightarrow{QR} \in \overrightarrow{CD}$. Atunci definim adunarea vectorilor (vezi și figura 3.1, pag. 53) astfel : $\overrightarrow{AB} + \overrightarrow{CD} = \overrightarrow{PR}$ (regula triunghiului). Fără a restrânge generalitatea spunem că $\overrightarrow{AB} + \overrightarrow{CD} = \overrightarrow{PR}$.

Proprietatea 3.1

Adunarea vectorilor are proprietățile $SV_{1,2,3,4}$ din definiția 3.1, pag. 51.

Fie $\overrightarrow{AB} \in \overrightarrow{AB}$. Inversul său (vectorul opus) este $\overrightarrow{BA} \in \overrightarrow{BA}$.

2. Fie $k \in \mathbf{R}$ și $\vec{v} \in V_3$. Atunci $k \cdot \vec{v}$ este :
 - (a) vectorul $\vec{0}$ dacă $k = 0$ sau $\vec{v} = \vec{0}$,
 - (b) vectorul care are aceeași direcție cu \vec{v} și același sens (dacă $k > 0$) sau sens diferit (dacă $k < 0$) și lungime $|k| \cdot |\vec{v}|$.

Proprietatea 3.2

Înmulțirea vectorilor cu scalari are proprietățile $SV_{5,6,7,8}$ din definiția 3.1, pag. 51.

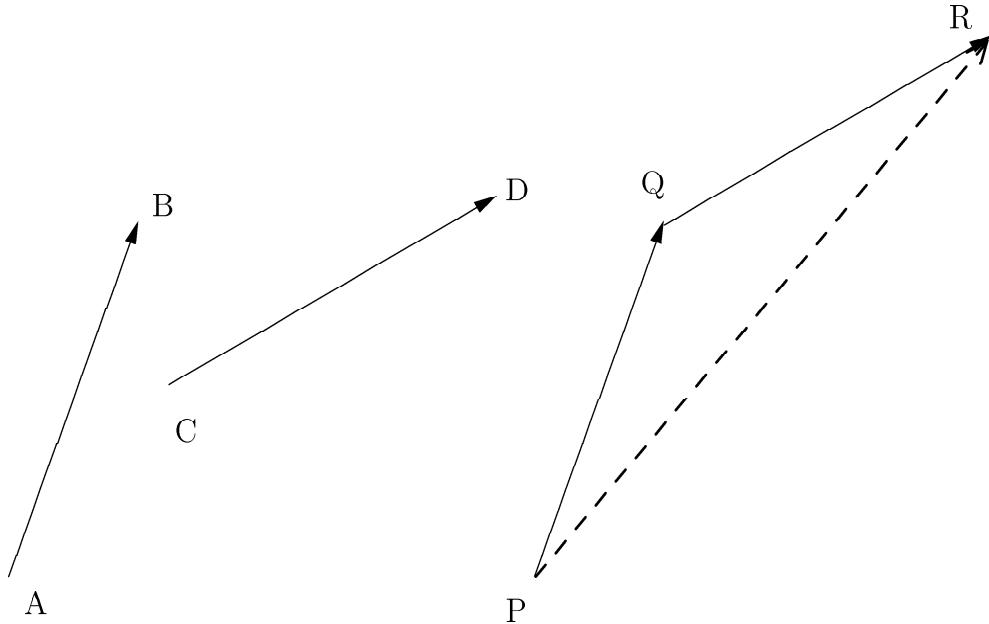


Figura 3.1: Adunarea vectorilor

Definiția 3.2

Doi vectori \vec{u} și \vec{v} sunt *coliniari* dacă au aceeași direcție.

Proprietatea 3.3

Vectorii \vec{u} și \vec{v} sunt coliniari dacă $(\exists \lambda \in \mathbf{R}^*)(\vec{u} = \lambda \cdot \vec{v})$.

Pentru evidențierea izomorfismului între spațiile vectoriale $(\mathbf{R}, \mathbf{R}^3, +, \cdot)$ și $(\mathbf{R}, V_3, +, \cdot)$ vom introduce un sistem de coordonate carteziene $Oxyz$ (astfel încât axele Ox , Oy și Oz sunt perpendiculare două câte două).

Proprietatea 3.4

$(\forall \vec{v} \in V_3)(\exists \vec{v}_x, \vec{v}_y, \vec{v}_z \in V_3, \text{perpendiculari doi căte doi})(\vec{v} = \vec{v}_x + \vec{v}_y + \vec{v}_z)$ și mai mult, \vec{v}_x , \vec{v}_y și \vec{v}_z sunt unici cu această proprietate. Direcțiile vectorilor \vec{v}_x , \vec{v}_y și \vec{v}_z sunt axele Ox , Oy și Oz . Dacă $\vec{v} = \overrightarrow{OA}$ atunci $\vec{v}_x = \overrightarrow{OA_x}$, $\vec{v}_y = \overrightarrow{OA_y}$ și $\vec{v}_z = \overrightarrow{OA_z}$, unde $AA_x \perp OA_x$, $AA_y \perp OA_y$ și $AA_z \perp OA_z$ iar $A_x \in Ox$, $A_y \in Oy$, $A_z \in Oz$.

Fie $\vec{i}, \vec{j}, \vec{k}$ versorii sistemului de coordonate (vectorii unitate ai direcțiilor Ox , Oy , Oz). Atunci $(\exists v_x, v_y, v_z \in \mathbf{R})(\vec{v}_x = v_x \cdot \vec{i} \wedge \vec{v}_y = v_y \cdot \vec{j} \wedge \vec{v}_z = v_z \cdot \vec{k})$. Numim v_x, v_y, v_z componentele vectorului \vec{v} în raport cu sistemul de coordonate $Oxyz$ iar v_x, v_y, v_z coordonatele vectorului \vec{v} în raport cu sistemul de coordonate $Oxyz$.

Fie $A_i(a_i, b_i, c_i)$, $1 \leq i \leq 2$ două puncte astfel încât $\vec{v} = \overrightarrow{A_1 A_2}$. Forma hipercomplexă a vectorului $\overrightarrow{A_1 A_2}$ este $\overrightarrow{A_1 A_2} = (a_2 - a_1) \cdot \vec{i} + (b_2 - b_1) \cdot \vec{j} + (c_2 - c_1) \cdot \vec{k}$ iar $|\overrightarrow{A_1 A_2}| = \sqrt{(a_2 - a_1)^2 + (b_2 - b_1)^2 + (c_2 - c_1)^2}$.

Deci există o corespondență bijectivă ϕ între spațiile vectoriale $(\mathbf{R}, V_3, +, \cdot)$ și $(\mathbf{R}, \mathbf{R}^3, +, \cdot)$ și dată de faptul că oricărui vector $\vec{v} \in V_3$ îi corespunde în mod unic tripletul coordonatelor

sale (v_x, v_y, v_z) și viceversa, oricărui triplet (a, b, c) din \mathbf{R}^3 îi corespunde în mod unic vectorul $a \cdot \vec{i} + b \cdot \vec{j} + c \cdot \vec{k}$ din V_3 .

Mai mult, această bijectie păstrează operațiile: $\phi(\vec{u} + \vec{v}) = \phi(\vec{u}) + \phi(\vec{v})$ și $\phi(a \cdot \vec{u}) = a \cdot \phi(\vec{u})$, $\forall \vec{u}, \vec{v} \in V_3, \forall a \in \mathbf{R}$.

Deci, dacă $\vec{a} = a_1 \cdot \vec{i} + a_2 \cdot \vec{j} + a_3 \cdot \vec{k}$ și $\vec{b} = b_1 \cdot \vec{i} + b_2 \cdot \vec{j} + b_3 \cdot \vec{k}$ și $p \in \mathbf{R}$ atunci $\vec{a} + \vec{b} = (a_1 + b_1) \cdot \vec{i} + (a_2 + b_2) \cdot \vec{j} + (a_3 + b_3) \cdot \vec{k}$ și $p \cdot \vec{a} = (p \cdot a_1) \cdot \vec{i} + (p \cdot a_2) \cdot \vec{j} + (p \cdot a_3) \cdot \vec{k}$.

În concluzie, spațiile vectoriale $(\mathbf{R}, V_3, +, \cdot)$ și $(\mathbf{R}, \mathbf{R}^3, +, \cdot)$ sunt izomorfe.

Exercițiu 3.1

Fie $\vec{a} = 2\vec{i} + (\frac{1}{2})\vec{j} - \vec{k}$ și $\vec{b} = -3\vec{i} + 2\vec{j} + 5\vec{k}$. Să se calculeze $\vec{a} + \vec{b}$, $-\vec{b}$, $\vec{a} - \vec{b}$, $d\vec{a}$.

În concluzie vom identifica un vector cu coordonatele sale: \vec{i} cu $(1, 0, 0)$, \vec{j} cu $(0, 1, 0)$ și \vec{k} cu $(0, 0, 1)$ iar \vec{v} cu (v_x, v_y, v_z) .

Definiția 3.3 (Produs scalar)

Produsul scalar este aplicația $\cdot : V_3 \times V_3 \rightarrow \mathbf{R}$ definită astfel:

$$\vec{u} \cdot \vec{v} = \begin{cases} 0 & , \text{ dacă } \vec{u} = \vec{0} \vee \vec{v} = \vec{0}, \\ |\vec{u}| |\vec{v}| \cos \widehat{\vec{u}, \vec{v}} & , \text{ dacă } \vec{u} \neq \vec{0} \wedge \vec{v} \neq \vec{0} \end{cases}$$

, unde $\vec{u}, \vec{v} \in V_3$ și $\widehat{\vec{u}, \vec{v}}$ este cel mai mic dintre unghiurile determinate de dreptele suport ale vectorilor \vec{u} și \vec{v} .

Proprietatea 3.5

Produsul scalar are următoarele proprietăți:

1. este comutativ,
2. $\vec{i} \cdot \vec{i} = \vec{j} \cdot \vec{j} = \vec{k} \cdot \vec{k} = 1$ (deoarece unghiul dintre vectori este 0°),
3. $\vec{i} \cdot \vec{j} = \vec{i} \cdot \vec{k} = \vec{j} \cdot \vec{k} = 0$ (deoarece unghiul dintre vectori este 90°),
4. dacă $\vec{a} = a_1 \cdot \vec{i} + a_2 \cdot \vec{j} + a_3 \cdot \vec{k}$ și $\vec{b} = b_1 \cdot \vec{i} + b_2 \cdot \vec{j} + b_3 \cdot \vec{k}$ atunci $\vec{a} \cdot \vec{b} = \sum_{i=1}^3 a_i b_i$,
5. $\cos \widehat{\vec{u}, \vec{v}} = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}| |\vec{v}|} = \frac{\sum_{i=1}^3 a_i b_i}{\sqrt{(\sum_{i=1}^3 a_i^2)(\sum_{i=1}^3 b_i^2)}}$.

Exercițiu 3.2

Fie $\vec{a} = 3\vec{i} - 4\vec{j}$ și $\vec{b} = \vec{i} + 2\vec{j} - 2\vec{k}$. Care este unghiul $\widehat{\vec{a}, \vec{b}}$?

Definiția 3.4 (Produs vectorial)

Produsul vectorial este aplicația $\times : V_3 \times V_3 \rightarrow V_3$ definită astfel: $\vec{u} \times \vec{v}$ este $\vec{0}$ dacă $(\vec{u} = \vec{0} \vee \vec{v} = \vec{0})$ și, în cazul în care $(\vec{u} \neq \vec{0} \wedge \vec{v} \neq \vec{0})$, este vectorul \vec{w} definit astfel încât:

1. $|\vec{w}| = |\vec{u}| |\vec{v}| \sin \widehat{\vec{u}, \vec{v}}$,

2. $\vec{w} \perp \vec{u}$ și $\vec{w} \perp \vec{v}$,
3. $(\vec{u}, \vec{v}, \vec{w})$ constituie un sistem orientat drept : dacă efectuăm cea mai scurtă rotație astfel încât să ducem \vec{u} peste \vec{v} atunci degetul mare arată sensul vectorului \vec{w} (sau dacă degetul mare al mâinii drepte arată sensul \vec{u} iar degetul arătător al mâinii drepte arată sensul \vec{v} atunci arată sensul \vec{w}).

Proprietatea 3.6

Produsul vectorial are următoarele proprietăți :

1. $(\forall \vec{u}, \vec{v} \in V_3)(\vec{u} \times \vec{v} = -\vec{v} \times \vec{u})$,
2. $\vec{i} \times \vec{i} = \vec{j} \times \vec{j} = \vec{k} \times \vec{k} = \vec{0}$ (deoarece unghiul dintre vectori este 0°),
3. $\vec{i} \times \vec{j} = \vec{k}$, $\vec{j} \times \vec{k} = \vec{i}$, $\vec{k} \times \vec{i} = \vec{j}$ (deoarece unghiul dintre vectori este 90°),
4. $\vec{u} \times \vec{v} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix}$,

Exercițiul 3.3

Fie $\vec{a} = 5\vec{i} - 3\vec{j} + \vec{k}$ și $\vec{b} = -\vec{i} - \vec{j} - 2\vec{k}$. Calculați $\vec{u} \times \vec{v}$.

Proprietatea 3.7

Doi vectori \vec{u} și \vec{v} sunt ortogonali (direcțiile lor sunt drepte perpendiculare) dacă produsul lor scalar este 0 : $\vec{u} \cdot \vec{v} = 0$.

Definiția 3.5 (Dependență liniară)

Fie $(\vec{v}_i)_{1 \leq i \leq n}$ vectori din spațiul vectorial (K, V) și $\vec{v} \in V$. Atunci vectorul \vec{v} este *liniar dependent* de vectorii $(\vec{v}_i)_{1 \leq i \leq n}$ dacă $(\exists (a_i \in K)_{1 \leq i \leq n})(\vec{v} = \sum_{i=1}^n a_i \vec{v}_i)$. Mai spunem că \vec{v} este o combinație liniară de vectorii $(\vec{v}_i)_{1 \leq i \leq n}$.

Definiția 3.6 ((In)dependență liniară)

Un sistem $(\vec{v}_i)_{1 \leq i \leq n}$ de vectori din spațiul vectorial (K, V) este *liniar dependent* dacă $(\exists$ scalarii $(a_i \in K)_{1 \leq i \leq n}$ nu toți nuli) $(\sum_{i=1}^n a_i \vec{v}_i = \vec{0})$ și este *liniar independent* dacă $(\forall (a_i \in K)_{1 \leq i \leq n})(\sum_{i=1}^n a_i \vec{v}_i = \vec{0} \implies (\forall 1 \leq i \leq n)(a_i = 0))$.

Proprietatea 3.8

Vectorii \vec{i} , \vec{j} , \vec{k} sunt liniar independenți.

Demonstrare

Fie $(a_i \in \mathbf{R})_{1 \leq i \leq 3}$ astfel încât $a_1 \vec{i} + a_2 \vec{j} + a_3 \vec{k} = \vec{0}$. Atunci, efectuând înmulțirea scalară la dreapta cu \vec{i} , rezultă $a_1 \vec{i} + a_2 \vec{j} + a_3 \vec{k} \vec{i} = \vec{0} \vec{i} = 0$ și deci $a_1 = 0$. Analog se arată $a_2 = a_3 = 0$.

□

Proprietatea 3.9

Orice vector din V_3 este liniar dependent de vectorii $\vec{i}, \vec{j}, \vec{k}$: $(\forall \vec{v} \in V_3)(\exists(a_i \in \mathbf{R})_{1 \leq i \leq 3})(\vec{v} = a_1\vec{i} + a_2\vec{j} + a_3\vec{k})$.

Demonstratie

$(a_i \in \mathbf{R})_{1 \leq i \leq 3}$ există și sunt unici cu proprietatea din enunț : $a_1 = \vec{v} \cdot \vec{i}$, $a_2 = \vec{v} \cdot \vec{j}$, $a_3 = \vec{v} \cdot \vec{k}$.

□

Definiția 3.7 (Bază)

O bază a unui spațiu vectorial (K, V) este un sistem $B \subseteq V$ de vectori astfel încât orice vector din V se exprimă în mod unic ca o combinație liniară de elemente din B .

D.p.d.v. formal : $B = \{\vec{e}_1, \vec{e}_2, \dots, \vec{e}_n\}$ și

$(\forall \vec{v} \in V)(\exists(a_i \in K)_{1 \leq i \leq n})$

$$\left(\vec{v} = \sum_{i=1}^n a_i \vec{e}_i \wedge \left((\forall(b_i \in K)_{1 \leq i \leq n})(\vec{v} = \sum_{i=1}^n b_i \vec{e}_i \Rightarrow (\forall 1 \leq i \leq n)(a_i = b_i)) \right) \right).$$

O definiție echivalentă cu definiția 3.7, pag. 56 este următoarea :

Definiția 3.8 (Bază)

O bază a unui spațiu vectorial (K, V) este un sistem $B \subseteq V$ liniar independent de vectori astfel încât orice vector din V să depindă liniar de B .

Definiția 3.9 (Dimensiune finită)

Dacă într-un spațiu vectorial (K, V) există o bază finită atunci spunem că V are dimensiune finită.

Observația 3.1

În cele ce urmează vom presupune că V are dimensiune finită.

Teoremă 3.1

Dacă spațiul vectorial V are dimensiune finită atunci două baze ale lui V au întotdeauna același număr de elemente. Acest număr este dimensiunea lui V : $\dim(V)$.

Observația 3.2

Sistemul $\{\vec{i}, \vec{j}, \vec{k}\}$ este o bază a spațiului vectorial (\mathbf{R}, V_3) .

Definiția 3.10 (Coordonate în raport cu o bază)

Fie $B = \{\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n\}$ o bază în spațiul vectorial (K, V) . Atunci $\forall \vec{v} \in V, \exists(a_i \in K)_{1 \leq i \leq n}$ unice astfel încât $\vec{v} = \sum_{i=1}^n a_i \vec{v}_i$. Numim $(a_i)_{1 \leq i \leq n}$ coordonatele lui \vec{v} în raport cu baza B .

Definiția 3.11 (Baze ortogonale, ortonormale)

1. O bază $B = \{\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n\}$ în spațiul vectorial (K, V) este ortogonală dacă vectorii bazei sunt ortogonali doi către doi : $(\forall 1 \leq i < j \leq n)(\vec{v}_i \cdot \vec{v}_j = 0)$.

2. Baza B este ortonormală dacă este ortogonală și vectorii bazei sunt vectori unitate : $(\forall 1 \leq i \leq n)(|\vec{v}_i| = 1)$.

Observația 3.3

Sistemul $\{\vec{i}, \vec{j}, \vec{k}\}$ este o bază ortonormală a spațiului vectorial (\mathbf{R}, V_3) .

Capitolul 4

Aplicații liniare

În cele ce urmează reamintim câteva noțiuni legate de aplicațiile liniare.

Fie V, V' două spații vectoriale¹.

Definiția 4.1

Se numește *aplicație liniară* o aplicație $A : V \rightarrow V'$ astfel încât $(\forall a \in \mathbf{R})(\forall \vec{u}, \vec{v} \in V)(A(\vec{u} + \vec{v}) = A(\vec{u}) + A(\vec{v}) \wedge A(a\vec{u}) = aA(\vec{u}))$.

Exemplul 4.1

Aplicația identitate $I : V \rightarrow V, I(\vec{v}) = \vec{v}, \forall \vec{v} \in V$ este o aplicație liniară : $I(\vec{u} + \vec{v}) = \vec{u} + \vec{v} = I(\vec{u}) + I(\vec{v})$ și $I(a\vec{u}) = a\vec{u} = aI(\vec{u})$.

Definiția 4.2 (Operații cu aplicații liniare)

Fie $AL(V, V') = \{A \mid A : V \rightarrow V' \text{ și } A \text{ este aplicație liniară}\}$. Atunci $AL(V, V')$ este un spațiu vectorial :

1. Fie $A, B \in AL(V, V')$, $(A + B)(\vec{v}) = A(\vec{v}) + B(\vec{v})$, $(aA)(\vec{v}) = aA(\vec{v})$. Se poate arăta că $A + B$ și aA sunt aplicații liniare ($\in AL(V, V')$).
2. Produsul a două aplicații liniare se definește astfel : fie $A \in AL(V, V')$ și $B \in AL(V', V'')$, unde V, V', V'' sunt spații vectoriale. Definim $B \cdot A \in AL(V, V'')$ prin $(B \cdot A)(\vec{v}) = B(A(\vec{v}))$.

Definiția 4.3

O aplicație liniară $A \in AL(V, V)$ este *ortogonală* dacă $(\forall \vec{u}, \vec{v} \in V)(\vec{u} \cdot \vec{v} = A(\vec{u}) \cdot A(\vec{v}))$ i.e., lasă neschimbă produsul scalar al vectorilor.

Proprietatea 4.1

Fie $A \in AL(V, V)$ o aplicație liniară ortogonală. Atunci avem :

1. $(\forall \vec{u} \in V)(|\vec{u}| = |A(\vec{u})|)$ i.e., lasă neschimbă lungimea vectorilor. Această proprietate rezultă din faptul că $1 = \cos \widehat{\vec{u}, \vec{u}} = \frac{\vec{u} \cdot \vec{u}}{|\vec{u}| |\vec{u}|} \iff \vec{u} \cdot \vec{u} = |\vec{u}| |\vec{u}| \iff |\vec{u}| = \sqrt{\vec{u} \cdot \vec{u}}$ și deci $|A(\vec{u})| = \sqrt{A(\vec{u}) \cdot A(\vec{u})} = \sqrt{\vec{u} \cdot \vec{u}} = |\vec{u}|$.

¹dacă mulțimea scalarilor este subînteleasă atunci o putem omite (în acest caz este \mathbf{R})

2. $(\forall \vec{u}, \vec{v} \in V)(\widehat{\vec{u}}, \widehat{\vec{v}} = A(\widehat{\vec{u}}), A(\widehat{\vec{v}}))$. Se arată ca mai sus.

Exemplul 4.2

Se poate arăta că rotațiile sunt aplicații liniare ortogonale.

Definiția 4.4

Fie $A \in AL(V, V)$ o aplicație liniară. Dacă A este și izomorfism atunci A se numește *aplicație liniară regulată*:

$$ALR_1 \quad (\forall \vec{u}, \vec{v} \in V)(A(\vec{u} + \vec{v}) = A(\vec{u}) + A(\vec{v})),$$

$$ALR_2 \quad (\forall a \in K)(\forall \vec{v} \in V)(A(a\vec{v}) = aA(\vec{v})) \text{ și}$$

$$ALR_3 \quad A : V \rightarrow V \text{ este bijectivă.}$$

Proprietatea 4.2

Fie $B \subseteq V$ o bază a spațiului vectorial V . Fie $A \in AL(V, V)$ o aplicație liniară regulată. Atunci $A(B) \subseteq V$ este tot o bază în V .

Demonstrație

Fie $B = \{\vec{e}_1, \vec{e}_2, \dots, \vec{e}_n\}$. Fie $\vec{v} \in V$. Atunci $(\exists v_i \in K)_{1 \leq i \leq n}$ scalari unici astfel încât $\vec{v} = \sum_{i=1}^n v_i \vec{e}_i$. Deci $A(\vec{v}) = A(\sum_{i=1}^n v_i \vec{e}_i) = \sum_{i=1}^n A(v_i \vec{e}_i) = \sum_{i=1}^n v_i A(\vec{e}_i)$. Dar $A(V) = V$ (avem $A(V) \subseteq V$ și $V \subseteq A(V)$ deoarece A este surjectivă). Deci $\forall \vec{w} \in A(V) (= V)$, $(\exists w_i \in K)_{1 \leq i \leq n}$ unice ($w_i = v_i, \forall 1 \leq i \leq n$) astfel încât $\vec{w} = \sum_{i=1}^n w_i \vec{A}(e_i)$. Avem $|A(B)| = |B|$ deoarece A este bijectivă. Deci $A(B)$ este o bază în $A(V)$ și deci în V .

□

Teorema 4.1

Fie $A \in AL(V, V)$ o aplicație liniară. Atunci A este ortogonală dacă $(\forall B \subseteq V$ bază ortonormală $) (A(B)$ este o bază ortonormală).

Demonstrație

(\Rightarrow) Arătăm că $A(B)$ este un sistem liniar independent de vectori din V și orice vector din V depinde liniar de $A(B)$ și în concluzie $A(B)$ este o bază în V . Ulterior arătăm că $A(B)$ este ortogonală și ortonormală.

Fie $B = \{\vec{e}_1, \vec{e}_2, \dots, \vec{e}_n\}$ o bază ortonormală oarecare din V și fie $\vec{e}_i, \vec{e}_j \in B$, $i \neq j$. Deoarece A este ortogonală avem $A(\vec{e}_i) \cdot A(\vec{e}_j) = \vec{e}_i \cdot \vec{e}_j$ și, deoarece B este ortonormală, avem $\vec{e}_i \cdot \vec{e}_j = 0$. Deci :

$$(\forall 1 \leq i \neq j \leq n)(A(\vec{e}_i) \cdot A(\vec{e}_j) = 0) \tag{4.1}$$

Similar, $|A(\vec{e}_i)|^2 = A(\vec{e}_i) \cdot A(\vec{e}_i) = \vec{e}_i \cdot \vec{e}_i = 1$ și deci :

$$(\forall 1 \leq i \leq n)(|A(\vec{e}_i)| = 1) \tag{4.2}$$

Fie $(a_i \in K)_{1 \leq i \leq n}$ astfel încât $\sum_{i=1}^n a_i A(\vec{e}_i) = \vec{0}$. Înmulțim scalar membrul stâng și drept cu $A(\vec{e}_i)$ și obținem $a_i \cdot A(\vec{e}_i) \cdot A(\vec{e}_i) + \sum_{j=1, j \neq i}^n a_j \cdot A(\vec{e}_j) \cdot A(\vec{e}_i) = 0$ și deci (ținând cont de 4.1, pag. 58) :

$$(\forall 1 \leq i \leq n)(a_i = 0) \quad (4.3)$$

Deci $A(B)$ este un sistem liniar independent de vectori.

Fie $\vec{e} \in B$. Atunci putem scrie pe \vec{e} ca o combinație liniară de elemente din $A(B)$: $\vec{e} = \sum_{i=1}^n c_i A(\vec{e}_i)$, unde $(\forall 1 \leq i \leq n)(c_i = \vec{e} \cdot A(\vec{e}_i))$.

Fie acum $\vec{v} \in V$. Deoarece B este o bază în V atunci \vec{v} depinde liniar de toți vectorii din B și deci, cum orice vector din B depinde liniar de toți vectorii din $A(B)$ rezultă că \vec{v} depinde liniar de toți vectorii din $A(B)$.

Deci $A(B)$ este o bază și, ținând cont de 4.1, pag. 58 și de 4.2, pag. 58 este o bază ortonormală.

(\Leftarrow) Aplicația A este ortogonală dacă, conform definiției 4.3, pag. 57 avem $(\forall \vec{u}, \vec{v} \in V)(\vec{u} \cdot \vec{v} = A(\vec{u}) \cdot A(\vec{v}))$. Arătăm acest lucru : fie $\vec{u} = \sum_{i=1}^n a_i \vec{e}_i$ și $\vec{v} = \sum_{i=1}^n b_i \vec{e}_i$. Atunci $\vec{u} \cdot \vec{v} = \sum_{i=1}^n a_i b_i$.

Deoarece A este o aplicație liniară avem $A(\vec{u}) = \sum_{i=1}^n a_i A(\vec{e}_i)$ și $A(\vec{v}) = \sum_{i=1}^n b_i A(\vec{e}_i)$ și deci, deoarece $A(B)$ este o bază ortonormală, avem : $A(\vec{u}) \cdot A(\vec{v}) = \sum_{i=1}^n a_i b_i = \vec{u} \cdot \vec{v}$.

□

Definiția 4.5

Fie matricea $A \in \mathcal{M}_{m \times m}(\mathbf{R})$. Numim A o matrice *ortogonală* dacă ${}^t A = A^{-1}$ i.e., $A \cdot {}^t A = I_m$.

Proprietatea 4.3

Fie matricea ortogonală $A \in \mathcal{M}_{m \times m}(\mathbf{R})$. Dacă interpretăm liniile (coloanele) matricii A ca vectori din \mathbf{R}^m atunci avem : $(\forall \vec{u}, \vec{v}$ liniile (coloanele) din $A)(\vec{u} \neq \vec{v} \Rightarrow \vec{u} \cdot \vec{v} = 0 \wedge \vec{u} = \vec{v} \Rightarrow \vec{u} \cdot \vec{v} = 1)$ i.e., liniile (coloanele) matricii A sunt o bază ortonormală în \mathbf{R}^m .

Definiția 4.6

Fie $A \in \mathcal{M}_{m \times m}(\mathbf{R})$ o matrice ortogonală. Dacă $\det A = 1$ atunci A se numește *matrice ortogonală proprie*.

Exemplul 4.3

Se poate arăta că matricile rotațiilor 3D sunt matrici ortogonale proprii.

Capitolul 5

Transformări geometrice

În acest capitol introducem transformările geometrice 2D/3D de bază : *translația, scalarea și rotația*.

5.1 Transformări geometrice 2D

Definiția 5.1 (Translație)

Se numește *translație* (notată $T(d_x, d_y)$) o transformare geometrică care asociază oricărui punct $P(x, y)$ un punct $P'(x', y')$ cu proprietatea că $x' = x + d_x$, $y' = y + d_y$.

Putem exprima matricial acest lucru :

$$P' = P + T(d_x, d_y), \quad (5.1)$$

$$\text{unde } P' = \begin{pmatrix} x' \\ y' \end{pmatrix}, P = \begin{pmatrix} x \\ y \end{pmatrix} \text{ și } T(d_x, d_y) = \begin{pmatrix} d_x \\ d_y \end{pmatrix}.$$

Translația unui obiect se face efectuând translațiile ale fiecărui punct al obiectului utilizând ecuația 5.1, pag. 61.

Translația unei drepte se face efectuând translațiile a două puncte distincte aparținând dreptei. ■

Exercițiu 5.1

Ce se obține prin translația $(T(2, 0.5))$ poligonului cu vârfurile $A(1, 1)$, $B(2, 1)$, $C(2, 2)$, $D(1.5, 3)$, $E(1, 2)$ și muchiile AB, BC, CD, DE, EA ? ■

Definiția 5.2 (Scalare)

Se numește *scalare* (notată $S(s_x, s_y)$) o transformare geometrică care asociază oricărui punct $P(x, y)$ un punct $P'(x', y')$ cu proprietatea că $x' = x \cdot s_x$, $y' = y \cdot s_y$.

Putem exprima matricial acest lucru :

$$P' = S(s_x, s_y) \cdot P, \quad (5.2)$$

$$\text{unde } P' = \begin{pmatrix} x' \\ y' \end{pmatrix}, P = \begin{pmatrix} x \\ y \end{pmatrix} \text{ și } S(s_x, s_y) = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}.$$

O scalare este *diferențiată* dacă $s_x \neq s_y$ și este *uniformă* dacă $s_x = s_y$. ■

Observația 5.1

O scalare uniformă păstrează proporțiile obiectului.

Exercițiu 5.2

Ce se obține prin scalarea $(S(\frac{1}{3}, \frac{1}{2}))$ poligonului cu vîrfurile $A(3, 2)$, $B(6, 2)$, $C(6, 4)$, $D(4, 6)$, $E(3, 4)$ și muchiile AB, BC, CD, DE, EA ?

Definiția 5.3 (Rotație)

Se numește *rotație* (notată $R(\theta)$) o transformare geometrică care asociază oricărui punct $P(x, y)$ un punct $P'(x', y')$ cu proprietatea că $x' = x \cdot \cos \theta - y \cdot \sin \theta$, $y' = x \cdot \sin \theta + y \cdot \cos \theta$.

Putem exprima matricial acest lucru :

$$P' = R(\theta) \cdot P, \quad (5.3)$$

unde $P' = \begin{pmatrix} x' \\ y' \end{pmatrix}$, $P = \begin{pmatrix} x \\ y \end{pmatrix}$ și $R(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$.

Unghiurile θ se măsoară în sens trigonometric.

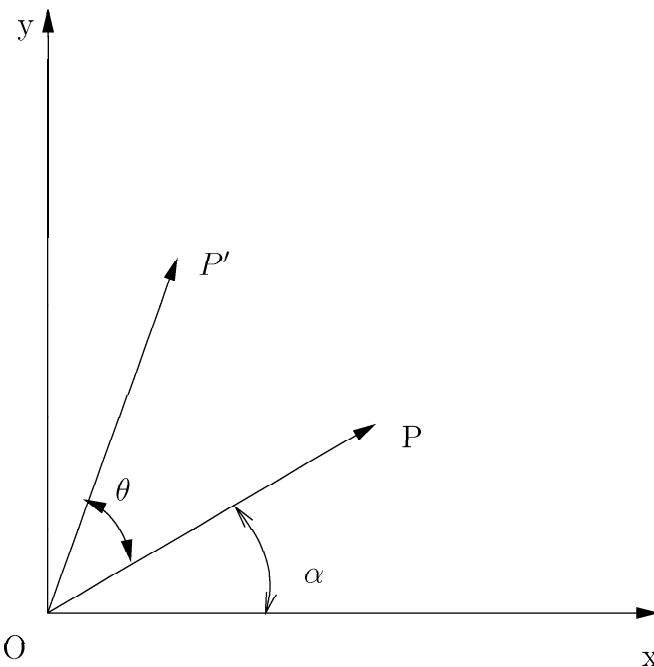


Figura 5.1: Transformarea de rotație este bine definită

Exercițiu 5.3

Să se arate că transformarea de rotație este bine definită: în figura 5.1, pag. 62 se aplică o transformare de rotație $R(\theta)$ punctului $P(x, y)$. Se obține punctul $P'(x', y')$. Avem $OP = OP' = r$. Să se arate că P' are coordonatele date de ecuația 5.3, pag. 62.

5.2 Coordonate omogene și reprezentarea matricială a transformărilor 2D

Așa cum se poate observa, ecuațiile matriciale ale transformărilor geometrice translație (5.1, pag. 61), scalare (5.2, pag. 61) și rotație (5.3, pag. 62) nu au aceeași formă (în cazul translației este vorba de adunare de matrici iar în celelalte două cazuri de înmulțire de matrici). Acest lucru face dificilă exprimarea unor transformări geometrice compuse (de exemplu transformarea $R_1T_1R_2S_1T_2R_3$, unde prin litere majuscule $\{T, S, R\}$ s-au notat tipurile de transformări). Pentru a ușura exprimarea matematică a unor astfel de transformări compuse vom trata în mod uniform transformările geometrice, mai precis acestea vor fi exprimate exclusiv prin produs de matrici.

Pentru aceasta vom exprima transformările geometrice nu în coordonate carteziene 2D ci în *coordonate omogene*.

Definiția 5.4 (Coordonate omogene)

Spațiul omogen 3D este spațiul cartezian 3D fără origine (punctul $(0, 0, 0)$) și deci tripletul (x, y, W) este un punct în *coordonate omogene 3D* dacă $x \neq 0 \vee y \neq 0 \vee W \neq 0$.

Unui punct cartezian 2D îi pot corespunde mai multe puncte omogene 3D și anume : punctele omogene 3D (x, y, W) și (x', y', W') reprezintă același punct cartezian 2D dacă $(\exists \alpha \neq 0)(x' = \alpha x \wedge y' = \alpha y \wedge W' = \alpha W)$.

Punctele $(x, y, 0)$ se numesc *puncte omogene la infinit*.

Numim transformare de *omogenizare* acea transformare care asociază unui punct omogen 3D (x, y, W) , $W \neq 0$, punctul omogen 3D $(\frac{x}{W}, \frac{y}{W}, 1)$. ■

Utilizând coordonatele omogene reformulăm definițiile transformărilor geometrice de mai sus.

Definiția 5.5 (Translație în coordonate omogene)

Se numește *translație* (notată $T(d_x, d_y)$) o transformare geometrică care asociază oricărui punct $P(x, y)$ un punct $P'(x', y')$ cu proprietatea că $x' = x + d_x$, $y' = y + d_y$.

Putem exprima matricial acest lucru :

$$P' = T(d_x, d_y) \cdot P, \quad (5.4)$$

$$\text{unde } P' = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}, P = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \text{ și } T(d_x, d_y) = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix}. \quad ■$$

Proprietatea 5.1

Componerea translațiilor este o operație aditivă : dacă P' se obține din P prin translația $T(d_{x_1}, d_{y_1})$ iar P'' se obține din P' prin translația $T(d_{x_2}, d_{y_2})$ atunci P'' se obține din P prin translația $T(d_{x_1} + d_{x_2}, d_{y_1} + d_{y_2})$. ■

Demonstrație
Ca exercițiu.

□

Definiția 5.6 (Scalare în coordonate omogene)

Se numește *scalare* (notată $S(s_x, s_y)$) o transformare geometrică care asociază oricărui punct $P(x, y)$ un punct $P'(x', y')$ cu proprietatea că $x' = x \cdot s_x$, $y' = y \cdot s_y$.

Putem exprima matricial acest lucru :

$$P' = S(s_x, s_y) \cdot P, \quad (5.5)$$

unde $P' = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}$, $P = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ și $S(s_x, s_y) = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$. □

Proprietatea 5.2

Componerea scalărilor este o operație multiplicativă : daca P' se obține din P prin scalarea $S(s_{x_1}, s_{y_1})$ iar P'' se obține din P' prin scalarea $S(s_{x_2}, s_{y_2})$ atunci P'' se obține din P prin scalarea $S(s_{x_1} \cdot s_{x_2}, s_{y_1} \cdot s_{y_2})$. □

Demonstrație

Ca exercițiu.

□

Definiția 5.7 (Rotație în coordonate omogene)

Se numește *rotație* (notată $R(\theta)$) o transformare geometrică care asociază oricărui punct $P(x, y)$ un punct $P'(x', y')$ cu proprietatea că $x' = x \cdot \cos \theta - y \cdot \sin \theta$, $y' = x \cdot \sin \theta + y \cdot \cos \theta$.

Putem exprima matricial acest lucru :

$$P' = R(\theta) \cdot P, \quad (5.6)$$

unde $P' = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}$, $P = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ și $R(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$. □

Proprietatea 5.3

Componerea rotațiilor este o operație aditivă : daca P' se obține din P prin rotația $R(\theta_1)$ iar P'' se obține din P' prin rotația $R(\theta_2)$ atunci P'' se obține din P prin rotația $R(\theta_1 + \theta_2)$. □

Demonstrație

Ca exercițiu.

□

Observația 5.2

Matricile transformărilor de rotație 2D/3D sunt matrici ortogonale proprii. □

Definiția 5.8

- O matrice de transformare de forma $\begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix}$, unde matricea $\begin{pmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{pmatrix}$ este ortogonală, păstrează unghiurile și distanțele. Transformările asociate se numesc *transformări de corp rigid*¹.

¹*rigid-body transformation* în lb. engleză

2. O secvență arbitrară de matrici de rotație și de translație creează o matrice de forma de mai sus.
3. Produsul unei secvențe arbitrară de matrici de rotație, translație și scalare se numește *transformare afină* și păstrează doar paralelismul dreptelor (și nu unghiurile și distanțele).

Exemplul 5.1

În figura 5.2, pag. 65 este prezentată o transformare afină, printr-o rotație de -45° , urmată de scalare neuniformă.

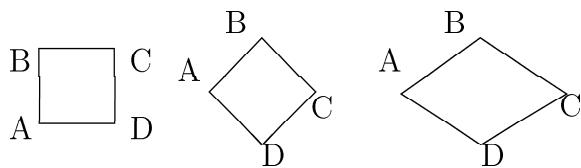


Figura 5.2: Transformare afină

Proprietatea 5.4

$R(\theta)$, $S(s_x, s_y)$, $T(d_x, d_y)$ sunt transformări affine.

Definiția 5.9 (Transformare forfecată)

Se numește *transformare forfecată* în lungul axei Ox (notată $SH_x(a)$) o transformare geometrică care asociază oricărui punct $P(x, y)$ un punct $P'(x', y')$ cu proprietatea că $x' = x + a \cdot y$, $y' = y$.

Putem exprima matricial acest lucru :

$$P' = SH_x(a) \cdot P, \quad (5.7)$$

$$\text{unde } P' = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}, P = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \text{ și } SH_x(a) = \begin{pmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

O transformare forfecată în lungul axei Oy (notată $SH_y(b)$) este caracterizată matricial cu ajutorul matricii $SH_y(b) = \begin{pmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

Exemplul 5.2

În figura 5.3, pag. 66 sunt prezentate transformări forfecate în direcția axelor Ox și Oy aplicate unui patrat de latură 1.

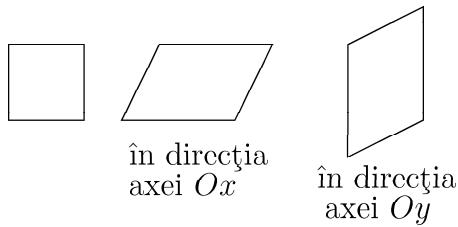


Figura 5.3: Transformări forfecate

5.3 Compunerea transformărilor 2D

Vom utiliza compunerea transformărilor 2D (înmulțirea matricilor aferente) pentru a combina transformările de translație, rotație, scalare într-o transformare generală pentru care dorim să-i calculăm matricea corespunzătoare.

Exemplul 5.3

În figura 5.4, pag. 66 este prezentată următoarea succesiune de transformări geometrice (care de fapt realizează rotația pentagonului în jurul punctului P_1) :

1. Translație obiect astfel încât P_1 să ajungă în originea sistemului de coordonate,
2. Rotație obiect,
3. Translație inversă astfel încât P_1 să ajungă în poziția inițială.

Pentru a obține matricea transformării generale compunem transformările 2D componente : $T(x_1, y_1) \cdot R(\theta) \cdot T(-x_1, -y_1)$ și obținem matricea $\begin{pmatrix} \cos \theta & -\sin \theta & x_1(1 - \cos \theta) + y_1 \sin \theta \\ \sin \theta & \cos \theta & y_1(1 - \cos \theta) + x_1 \sin \theta \\ 0 & 0 & 1 \end{pmatrix}$.

Se verifică că este o transformare de corp rigid. ■

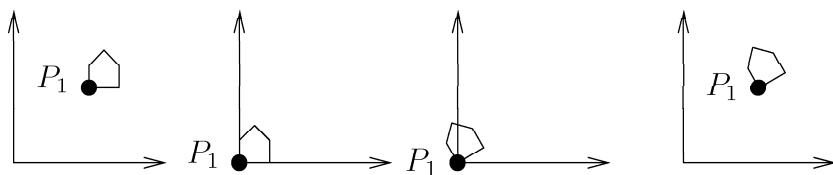


Figura 5.4: Rotația pentagonului în jurul P_1

Exemplul 5.4

În figura 5.5, pag. 67 este prezentată următoarea succesiune de transformări geometrice (care de fapt realizează scalarea pentagonului în raport cu punctul P_1) :

1. Translație obiect astfel încât P_1 să ajungă în originea sistemului de coordonate,
2. Scalare obiect,

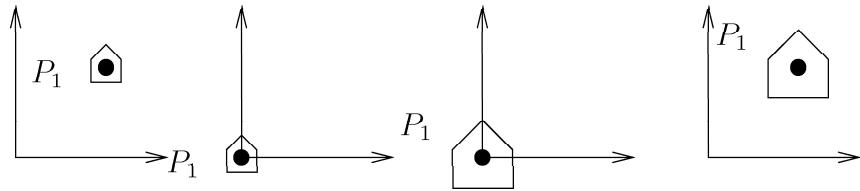


Figura 5.5: Scalarea pentagonului în raport cu P_1

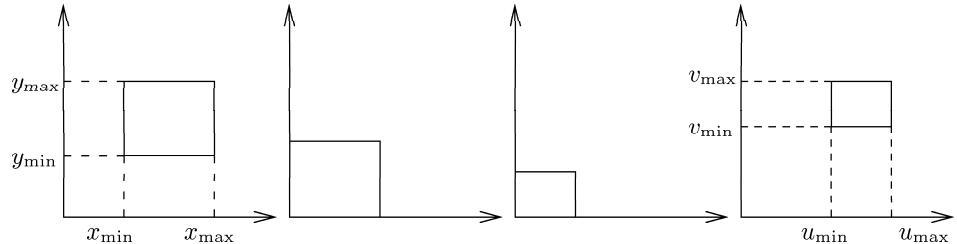


Figura 5.6: Transformarea window-viewport

3. Translație inversă astfel încât P_1 să ajungă în poziția inițială.

Pentru a obține matricea transformării generale compunem transformările 2D componente :

$$T(x_1, y_1) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1)$$

și obținem matricea $\begin{pmatrix} s_x & 0 & x_1(1-s_x) \\ 0 & s_y & y_1(1-s_y) \\ 0 & 0 & 1 \end{pmatrix}$. Se verifică că este o transformare afină dar nu și de corp rigid.

Proprietatea 5.5

Dacă avem unul din următoarele cazuri :

1. M_1 și M_2 sunt translații sau
2. M_1 și M_2 sunt scalări sau
3. M_1 și M_2 sunt rotații sau
4. M_1 este o scalare uniformă ($s_x = s_y$) și M_2 este o rotație

atunci $M_1 \cdot M_2 = M_2 \cdot M_1$.

Demonstratie

Ca exercițiu. Se ține cont că translațiile și rotațiile sunt aditive iar scalările sunt mălțivative.

■

5.4 Transformarea window-viewport

Unele pachete grafice permit specificarea coordonatelor primitive de ieșire în *sistemul de coordonate al domeniului (universului) aplicației*² utilizând unități de măsură care au o

²world-coordinate system în lb. engleză

semnificație în acest sistem de coordonate : microni, metri, ani lumină. *Universul (domeniul) aplicației* reprezintă o lume (univers) care este creată de către programul aplicației în mod interactiv sau este afișată (tot de către programul aplicației) pentru utilizator.

Apare astfel necesitatea *conversiei* coordonatelor din universul aplicației în coordonate ale dispozitivului de ieșire (ecran, etc.). Această conversie se efectuează astfel : programatorul poate specifica o regiune dreptunghiulară în sistemul de coordonate al universului aplicației (*fereastra universului aplicației*³) și o regiune dreptunghiulară corespondentă în coordonatele ecranului (dispozitiv de ieșire) denumită *viewport* în care fereastra universului aplicației va fi convertită. Această transformare este *universală* : se aplică tuturor primitelor de ieșire din fereastra universului aplicației.

Exemplul 5.5

În figura 5.7, pag. 68 am pus în evidență conversia ferestrei universului aplicației în viewport. Dacă fereastra universului aplicației și viewport-ul nu au același raport înălțime/lungime atunci apare o scalare neuniformă. ▀

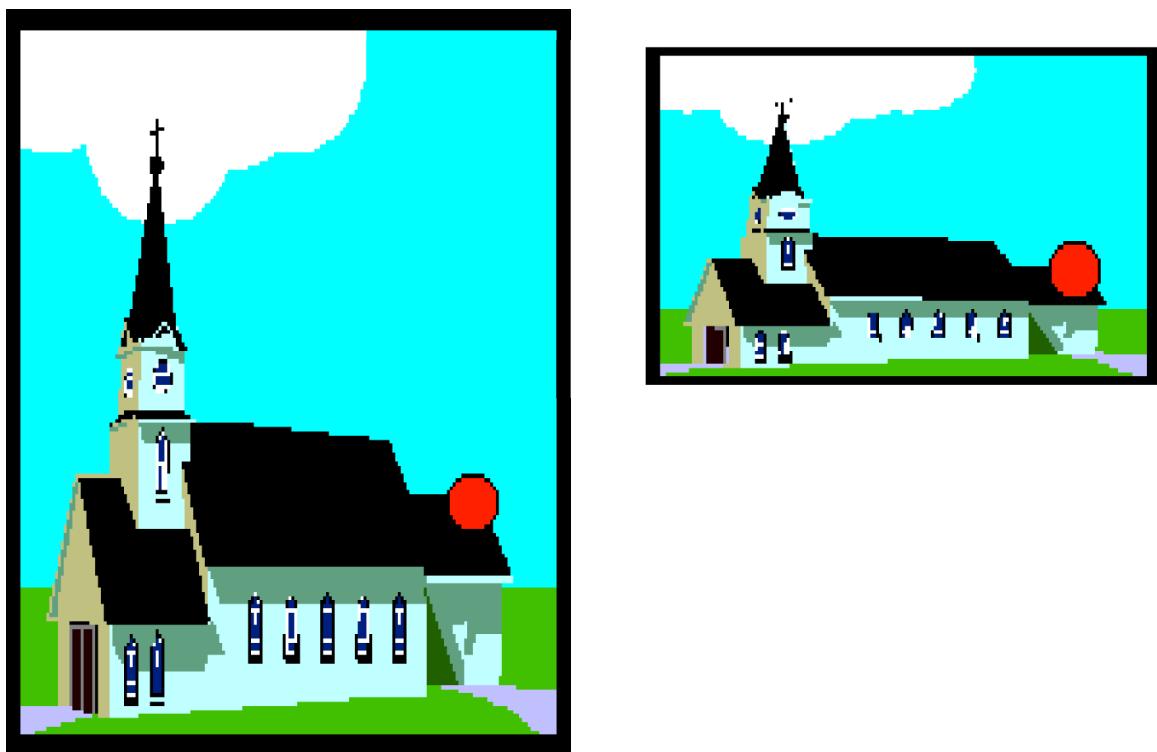


Figura 5.7: Conversia ferestrei universului aplicației în viewport

³*world coordinate window* în lb. engleză

Problema 5.1

Data o fereastră (a universului aplicației) și un viewport, care este matricea de transformare care atribuie un punct din sistemul de coordonate al universului aplicației unui punct din sistemul de coordonate ecran ?

Soluție

Din punct de vedere grafic sunt efectuate următoarele transformări (vezi figura 5.6, pag. 67) :

1. translarea ferestrei universului aplicației astfel încât colțul din stânga jos al ferestrei să ajungă în originea sistemului de coordonate al universului aplicației,
2. scalarea ferestrei universului aplicației pentru a căpăta dimensiunile viewport-ului,
3. translație pentru poziționarea viewport-ului.

Obținem astfel următoarea matrice : $M_{WV} = T(u_{\min}, v_{\min}) \cdot S\left(\frac{u_{\max}-u_{\min}}{x_{\max}-x_{\min}}, \frac{v_{\max}-v_{\min}}{y_{\max}-y_{\min}}\right) \cdot T(-x_{\min}, -y_{\min})$

$$T(-x_{\min}, -y_{\min}) = \begin{pmatrix} \frac{u_{\max}-u_{\min}}{x_{\max}-x_{\min}} & 0 & -x_{\min} \cdot \frac{u_{\max}-u_{\min}}{x_{\max}-x_{\min}} + u_{\min} \\ 0 & \frac{v_{\max}-v_{\min}}{y_{\max}-y_{\min}} & -y_{\min} \cdot \frac{v_{\max}-v_{\min}}{y_{\max}-y_{\min}} + v_{\min} \\ 0 & 0 & 1 \end{pmatrix}.$$

□

5.5 Eficiența transformărilor geometrice

O secvență $(R^*S^*T^*)^*$ de transformări geometrice produce o matrice M de forma : $\begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix}$, unde matricea $\begin{pmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{pmatrix}$ se obține prin compunerea rotațiilor R cu scalările S iar $t_{x,y}$ se obțin prin compunerea translațiilor T .

Înmulțirea $M \cdot P$, unde P reprezintă vectorul coordonatelor unui punct, necesită, în general, 9 operații de înmulțire și 6 operații de adunare. Deoarece ultimul rând al matricii M are forma $(0 \ 0 \ 1)$ atunci produsul $M \cdot P$ se realizează numai cu 4 operații de înmulțire și 4 operații de adunare : $x' = xr_{11} + yr_{12} + t_x$ și $y' = xr_{21} + yr_{22} + t_y$. Această reducere este utilă de exemplu în cazul în care transformarea M trebuie aplicată multor de puncte dintr-o anumită figură geometrică sau atunci când dorim să creem imagini succesive ale unui obiect (moleculară, avion) rotit doar cu câteva grade între 2 imagini succesive.

În acest din urmă caz (al rotațiilor succeseive doar cu câteva grade) putem ține cont de faptul că unghiul de rotație $\theta \approx 0^\circ$ și deci $\cos \theta \approx 1$. Obținem $x' = x - y \sin \theta$ și $y' = x \sin \theta + y$ și deci avem nevoie de 2 operații de înmulțire și 2 operații de adunare. Pentru a micșora erorile care apar după aplicarea acestor ecuații de un număr mare de ori, înlocuim în a doua ecuație pe x cu x' : $x' = x - y \sin \theta$, $y' = x' \sin \theta + y = x \sin \theta - y \sin^2 \theta + y = x \sin \theta + y(1 - \sin^2 \theta)$.

5.6 Reprezentarea matricială a transformărilor 3D

Așa cum am reprezentat transformările 2D prin matrici $\in \mathcal{M}_{3 \times 3}(\mathbf{R})$ utilizând coordonate omogene tot așa și transformările 3D pot fi reprezentate prin matrici $\in \mathcal{M}_{4 \times 4}(\mathbf{R})$ dacă utilizăm reprezentarea în coordonate omogene a punctelor din spațiul 3D.

Definiția 5.10 (Coordonate omogene)

Spațiul omogen 4D este spațiul cartezian 4D fără origine (punctul $(0, 0, 0, 0)$) și deci tripletul (x, y, z, W) este un punct în *coordonate omogene* 4D dacă $x \neq 0 \vee y \neq 0 \vee z \neq 0 \vee W \neq 0$.

Unui punct cartezian 3D îi pot corespunde mai multe puncte omogene 4D și anume : punctele omogene 4D (x, y, z, W) și (x', y', z', W') reprezintă același punct cartezian 3D dacă $(\exists \alpha \neq 0)(x' = \alpha x \wedge y' = \alpha y \wedge z' = \alpha z \wedge W' = \alpha W)$.

Punctele $(x, y, z, 0)$ se numesc *puncte omogene la infinit*.

Numim transformare de *omogenizare* acea transformare care asociază unui punct omogen 4D (x, y, z, W) , $W \neq 0$, punctul omogen 4D $(\frac{x}{W}, \frac{y}{W}, \frac{z}{W}, 1)$. ■

Observația 5.3

Un *sistem de coordonate cartezian* se definește prin originea O și trei axe perpendiculare Ox, Oy, Oz , orientate după regula mâinii drepte sau după regula mâinii stângi (vezi figura 5.8, pag. 70). ■

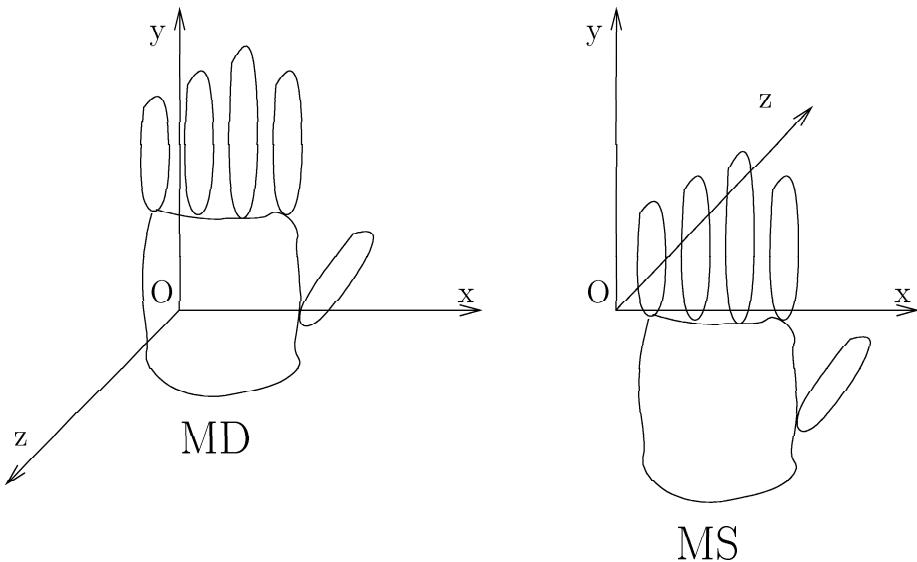


Figura 5.8: Regulile mâinii drepte și stângi

Definiția 5.11 (Translație 3D în coordonate omogene)

Se numește *translație* (notată $T(d_x, d_y, d_z)$) o transformare geometrică care asociază oricărui punct $P(x, y, z)$ un punct $P'(x', y', z')$ cu proprietatea că $x' = x + d_x$, $y' = y + d_y$ și $z' = z + d_z$.

Putem exprima matricial acest lucru :

$$P' = T(d_x, d_y, d_z) \cdot P, \quad (5.8)$$

$$\text{unde } P' = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}, \quad P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \text{ și } T(d_x, d_y, d_z) = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Definiția 5.12 (Scalare 3D în coordonate omogene)

Se numește *scalare* (notată $S(s_x, s_y, s_z)$) o transformare geometrică care asociază oricărui punct $P(x, y, z)$ un punct $P'(x', y', z')$ cu proprietatea că $x' = x \cdot s_x$, $y' = y \cdot s_y$ și $z' = z \cdot s_z$.

Putem exprima matricial acest lucru :

$$P' = S(s_x, s_y, s_z) \cdot P, \quad (5.9)$$

unde $P' = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$, $P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$ și $S(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.

Definiția 5.13 (Rotație 3D în coordonate omogene)

Transformarea de *rotație 3D* se obține prin trei tipuri de rotații :

1. Rotația în raport cu axa Oz , dată prin matricea $R_z(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
2. Rotația în raport cu axa Ox , dată prin matricea $R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
3. Rotația în raport cu axa Oy , dată prin matricea $R_y(\theta) = \begin{pmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Proprietatea 5.6

Matricile formate din primele trei linii și trei coloane ale matricilor $R_x(\theta)$, $R_y(\theta)$, $R_z(\theta)$ sunt matrici ortogonale proprii. Aceeași proprietate o are orice matrice corespunzătoare unei secvențe arbitrară de transformări de rotație. În concluzie, o secvență arbitrară de transformări de rotație păstrează unghiiurile și distanțele.

Demonstrație

Ca exercițiu. □

Proprietatea 5.7

Transformările geometrice inverse⁴ translației $T(d_x, d_y, d_z)$, scalării $S(s_x, s_y, s_z)$ sau rotației $R_{x,y,z}(\theta)$ sunt respectiv $T(-d_x, -d_y, -d_z)$, $S(\frac{1}{s_x}, \frac{1}{s_y}, \frac{1}{s_z})$ și $R_{x,y,z}(-\theta)$.

Demonstrație

Ca exercițiu. □

⁴produsul matricilor respective este matricea identitate I_4

Proprietatea 5.8

Fie B matricea corespunzătoare unei secvențe arbitrară de transformări de rotație. Atunci avem $B^{-1} = {}^t B$. ▀

Demonstrație

Ca exercițiu. □

Definiția 5.14 (Transformare forfecată 3D)

Se numește *transformare forfecată* în raport cu axele Ox, Oy (notată $SH_{xy}(sh_x, sh_y)$) o transformare geometrică care asociază oricărui punct $P(x, y, z)$ un punct $P'(x', y', z')$ cu proprietatea că $x' = x + sh_x \cdot z$, $y' = y + sh_y \cdot z$, $z' = z$.

Putem exprima matricial acest lucru :

$$P' = SH_{xy}(sh_x, sh_y) \cdot P, \quad (5.10)$$

$$\text{unde } P' = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}, P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \text{ și } SH_{xy}(sh_x, sh_y) = \begin{pmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

O transformare forfecată în raport cu axele Ox, Oz (notată $SH_{xz}(sh_x, sh_z)$) este caracterizată matricial cu ajutorul matricii $SH_{xz}(sh_x, sh_z) = \begin{pmatrix} 1 & sh_x & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & sh_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.

O transformare forfecată în raport cu axele Oy, Oz (notată $SH_{yz}(sh_y, sh_z)$) este caracterizată matricial cu ajutorul matricii $SH_{yz}(sh_y, sh_z) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ sh_x & 1 & 0 & 0 \\ sh_z & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$. ▀

Observația 5.4

1. Transformarea geometrică a unui segment de dreaptă se face efectuând transformarea extremităților segmentului și apoi trasând segmentul prin cele două puncte.
2. Transformarea geometrică a unui plan se face alegând trei puncte necoliniare din plan și aplicând transformarea geometrică acestor trei puncte.
3. Pentru transformarea geometrică a unui plan putem proceda și mai simplu : să presupunem că ecuația planului este $Ax + By + Cz + D = 0$. Putem reprezenta această ecuație în două moduri :

$$(a) \text{ ca un produs scalar : } N \cdot P = 0, \text{ unde } N = \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} \text{ și } P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix},$$

$$(b) \text{ ca un produs de matrici : } {}^t N \cdot P = 0.$$

Fie M matricea de transformare și să presupunem că dorim pentru planul transformat să avem o ecuație similară cu cea de mai sus : ${}^t N' \cdot P' = 0$, unde $P' = M \cdot P$. Să presupunem că N' este obținută prin aplicarea transformării Q : $N' = Q \cdot N$. Dacă am avea ${}^t Q M = I$ atunci ${}^t N' \cdot P' = {}^t (QN)MP = {}^t N {}^t QMP = {}^t N \cdot I \cdot P = {}^t N \cdot P = 0$. Deci dacă ${}^t Q M = I \iff {}^t Q = M^{-1}$ atunci ${}^t N' \cdot P' = 0$ iar $N' = {}^t (M^{-1})N$ (în cazul în care există matricea inversă M^{-1}). ■

5.7 Compunerea transformărilor 3D

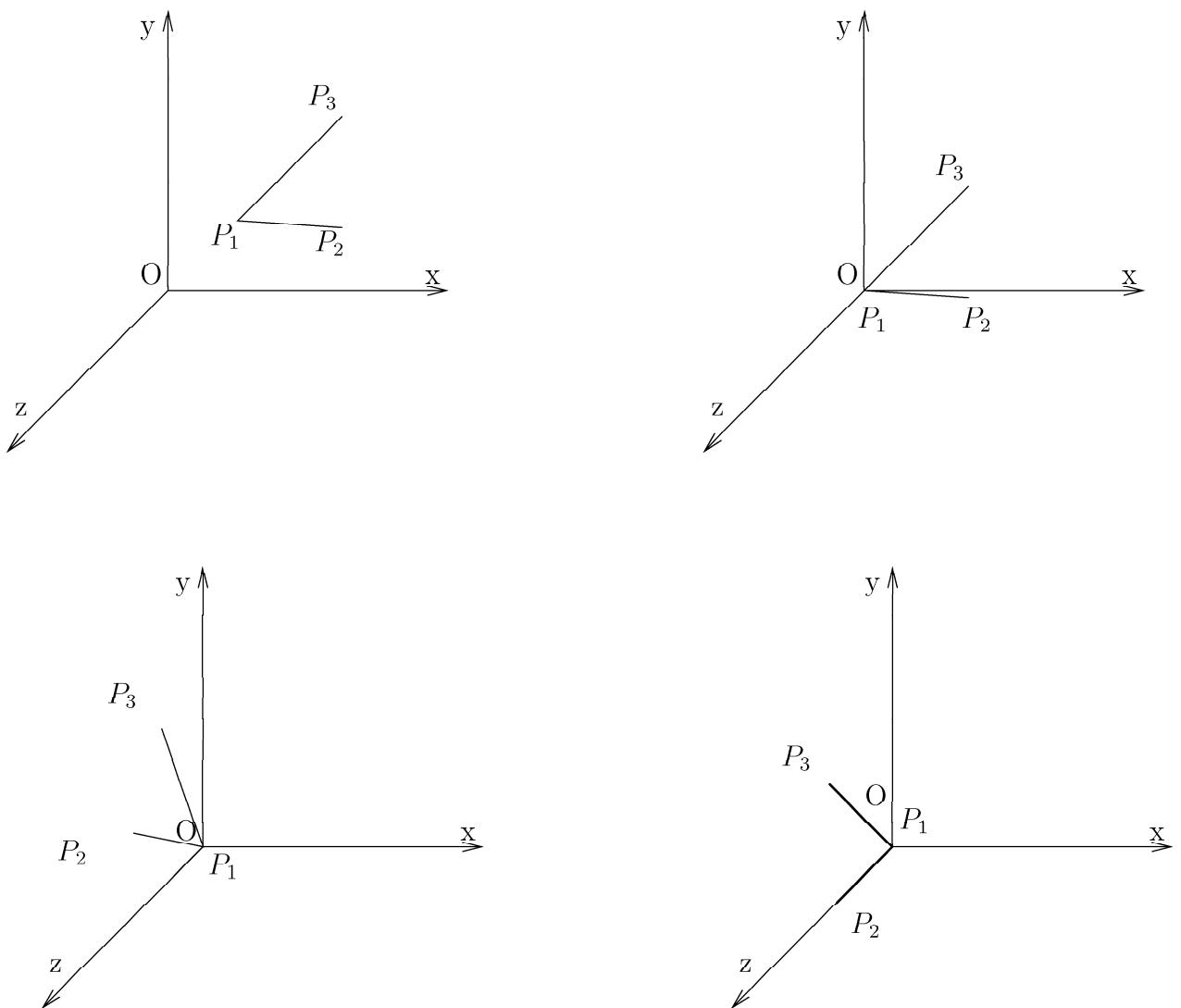


Figura 5.9: Transformare 3D compusă

Fie problema următoare :

Problema 5.2

Să se obțină o matrice de transformare astfel încât segmentele P_1P_2 și P_1P_3 (și planul determinat de aceste puncte) să se transforme aşa cum este arătat în figura 5.9, pag. 73 : în poziția finală segmentul P_1P_2 se află pe axa Oz iar P_1P_3 se află în planul Oyz (în cadranul I). În plus, dorim să nu afectăm, prin această transformare, lungimea segmentelor (i.e., să fie o transformare de corp rigid).

Soluție

Vom împărți această problemă în probleme mai simple :

1. Translarea punctului P_1 în originea sistemului de coordonate. Obținem astfel punctele $(P'_i)_{1 \leq i \leq 3}$.
2. Rotație în jurul axei Oy astfel încât $P'_1P'_2 \in Oyz$. Obținem astfel punctele $(P''_i)_{1 \leq i \leq 3}$.
3. Rotație în jurul axei Ox astfel încât $P''_1P''_2 \in Oz$. Obținem astfel punctele $(P'''_i)_{1 \leq i \leq 3}$.
4. Rotație în jurul axei Oz astfel încât $P'''_1P'''_3 \in Oyz$.

Problema 1, pag. 74 se rezolvă aplicând transformarea $T(-x_1, -y_1, -z_1)$. Punctele $(P_i)_{1 \leq i \leq 3}$ devin punctele $(P'_i)_{1 \leq i \leq 3}$, unde $P'_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$, $P'_2 = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ 1 \end{pmatrix}$ și $P'_3 = \begin{pmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \\ 1 \end{pmatrix}$.

Problema 2, pag. 74 se rezolvă astfel : fie $Q(x'_2, 0, z'_2)$ proiecția punctului P'_2 pe planul Oxz . Fie $D_1 = |P'_1Q| = \sqrt{x'^2_2 + z'^2_2} = \sqrt{(x_2 - x_1)^2 + (z_2 - z_1)^2}$. Efectuăm o rotație în jurul axei Oy astfel încât $P''_2 \in Oyz$ iar proiecția punctului P''_2 pe planul Oxz să se găsească pe axa Oz . Fie θ unghiul $\widehat{QP'_1, Ox}$. Vom aplica o rotație de unghi $-(90 - \theta)$: $R_y(\theta - 90)$ și vom obține

matricea $\begin{pmatrix} \cos(\theta - 90) & 0 & \sin(\theta - 90) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta - 90) & 0 & \cos(\theta - 90) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ și deci matricea $\begin{pmatrix} \frac{z_2 - z_1}{D_1} & 0 & -\frac{x_2 - x_1}{D_1} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{x_2 - x_1}{D_1} & 0 & \frac{z_2 - z_1}{D_1} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.

Punctele $(P'_i)_{1 \leq i \leq 3}$ devin punctele $(P''_i)_{1 \leq i \leq 3}$, unde, de exemplu $P''_2 = \begin{pmatrix} 0 \\ y_2 - y_1 \\ D_1 \\ 1 \end{pmatrix}$ (evident $P''_2 \in Oyz$).

Problema 3, pag. 74 se rezolvă aplicând o rotație în jurul axei Ox de unghi $\varphi = \widehat{P''_1P''_2, Oz} : R_x(\varphi)$. Unghiul φ se obține știind că $\cos \varphi = \frac{z''_2}{D_2}$ și $\sin \varphi = \frac{y''_2}{D_2}$, unde $D_2 = \sqrt{y''^2_2 + z''^2_2} = \sqrt{(y_2 - y_1)^2 + D_1^2} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$. Deci $\sin \varphi = \frac{y_2 - y_1}{D_2}$ și $\cos \varphi = \frac{D_1}{D_2}$.

Obținem deci matricea $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{D_1}{D_2} & -\frac{y_2 - y_1}{D_2} & 0 \\ 0 & \frac{y_2 - y_1}{D_2} & \frac{D_1}{D_2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$. În concluzie, punctele $(P''_i)_{1 \leq i \leq 3}$ devin

punctele $(P''_i)_{1 \leq i \leq 3}$, unde, de exemplu $P''_2 = \begin{pmatrix} 0 \\ 0 \\ D_2 \\ 1 \end{pmatrix}$, unde $D_2 = |P'_1 P'_2| = |P_1 P_2|$.

Problema 4, pag. 74 se rezolvă aplicând o rotație $R_z(\alpha)$, unde $\alpha = \widehat{OQ, Oy}$ iar Q este proiecția pe planul Oxy a punctului P'''_3 . Fie $D_3 = |OQ| = \sqrt{x'''_3^2 + y'''_3^2}$ și atunci avem $\cos \alpha = \frac{y'''_3}{D_3}$ și $\sin \alpha = \frac{x'''_3}{D_3}$.

În concluzie, matricea transformării este $R_z(\alpha)R_x(\varphi)R_y(\theta - 90)T(-x_1, -y_1, -z_1)$ și este

$$\text{matricea } \begin{pmatrix} \frac{z_2 - z_1}{D_1} & 0 & -\frac{x_2 - x_1}{D_1} & \frac{x_2 z_1 - x_1 z_2}{D_1} \\ -\frac{(x_2 - x_1)(y_2 - y_1)}{D_1 D_2} & \frac{D_1}{D_2} & -\frac{(y_2 - y_1)(z_2 - z_1)}{D_1 D_2} & \frac{(y_1 - y_2)(x_1^2 + z_1^2 - x_1 x_2 - y_1 z_2)}{D_1 D_2} - \frac{y_1 y_2}{D_2} \\ \frac{x_2 - x_1}{D_2} & \frac{y_2 - y_1}{D_2} & \frac{z_2 - z_1}{D_2} & \frac{x_1^2 + y_1^2 + z_1^2 - x_1 x_2 - y_1 y_2 - z_1 z_2}{D_2} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

□

5.8 Transformări ale sistemelor de coordonate

Putem vedea transformările geometrice în două moduri :

1. Ca o transformare a unei multimi de puncte în altă mulțime de puncte, în același sistem de coordonate.
2. Ca o schimbare a sistemului de coordonate. Această modalitate de a vedea transformările geometrice este utilă în cazul în care dorim să combinăm diverse obiecte, fiecare definite în propriul sistem de coordonate și se dorește exprimarea coordonatelor acestor obiecte într-un singur sistem global de coordonate.

Fie $M_{i \leftarrow j}$ transformarea care convertește reprezentarea unui punct din sistemul de coordonate j în reprezentarea sa în sistemul de coordonate i . Fie $P^{(i)}$, $P^{(j)}$, $P^{(k)}$ reprezentările unui punct în sistemele de coordonate i , j , k . Avem atunci relațiile $P^{(i)} = M_{i \leftarrow j} \cdot P^{(j)}$ și $P^{(j)} = M_{j \leftarrow k} \cdot P^{(k)}$ pe baza cărora putem obține $P^{(i)} = M_{i \leftarrow j} \cdot P^{(j)} = M_{i \leftarrow j} \cdot M_{j \leftarrow k} \cdot P^{(k)} = M_{i \leftarrow k} \cdot P^{(k)}$. Deci se obține $M_{i \leftarrow k} = M_{i \leftarrow j} \cdot M_{j \leftarrow k}$.

Exemplul 5.6

În figura 5.10, pag. 76 avem patru sisteme de coordonate diferite. Pentru a transforma sistemul de coordonate 1 în 2 ducem originea lui 1 în originea lui 2, pentru a transforma sistemul de coordonate 2 în 3 ducem originea lui 2 în originea lui 3 după care scalăm cu $\frac{1}{2}$, pentru a transforma sistemul de coordonate 3 în 4 ducem originea lui 3 în originea lui 4 după care rotim cu 45° . Avem relațiile : $M_{1 \leftarrow 2} = T(4, 2)$, $M_{2 \leftarrow 3} = T(2, 3) \cdot S(\frac{1}{2}, \frac{1}{2})$, $M_{3 \leftarrow 4} = T(8 - \sqrt{2}, 6 - 3\sqrt{2}) \cdot R(+45^\circ)$. Deci $M_{1 \leftarrow 3} = M_{1 \leftarrow 2} \cdot M_{2 \leftarrow 3} = T(4, 2) \cdot T(2, 3) \cdot S(\frac{1}{2}, \frac{1}{2})$ și $M_{1 \leftarrow 4} = M_{1 \leftarrow 2} \cdot M_{2 \leftarrow 3} \cdot M_{3 \leftarrow 4} = T(4, 2) \cdot T(2, 3) \cdot S(\frac{1}{2}, \frac{1}{2}) \cdot T(8 - \sqrt{2}, 6 - 3\sqrt{2}) \cdot R(+45^\circ)$. În plus avem $P^{(1)}(10, 8)$, $P^{(2)}(6, 6)$, $P^{(3)}(8, 6)$ și $P^{(4)}(4, 2)$.

Se mai poate observa că $M_{i \leftarrow j} = M_{j \leftarrow i}^{-1}$.

D.p.d.v. formal transformarea lui P din sistemul de coordonate 2 în sistemul de coordonate 1 se justifică astfel : fie O și O' originile sistemelor de coordonate 1 și 2 (a se vedea în figura 5.11, pag. 77). Avem $\overrightarrow{OP} = \overrightarrow{OO'} + \overrightarrow{O'P} = (4\vec{i} + 2\vec{j}) + (6\vec{i}' + 6\vec{j}') = (4\vec{i} + 2\vec{j}) + (6\vec{i}' + 6\vec{j})$ și deci

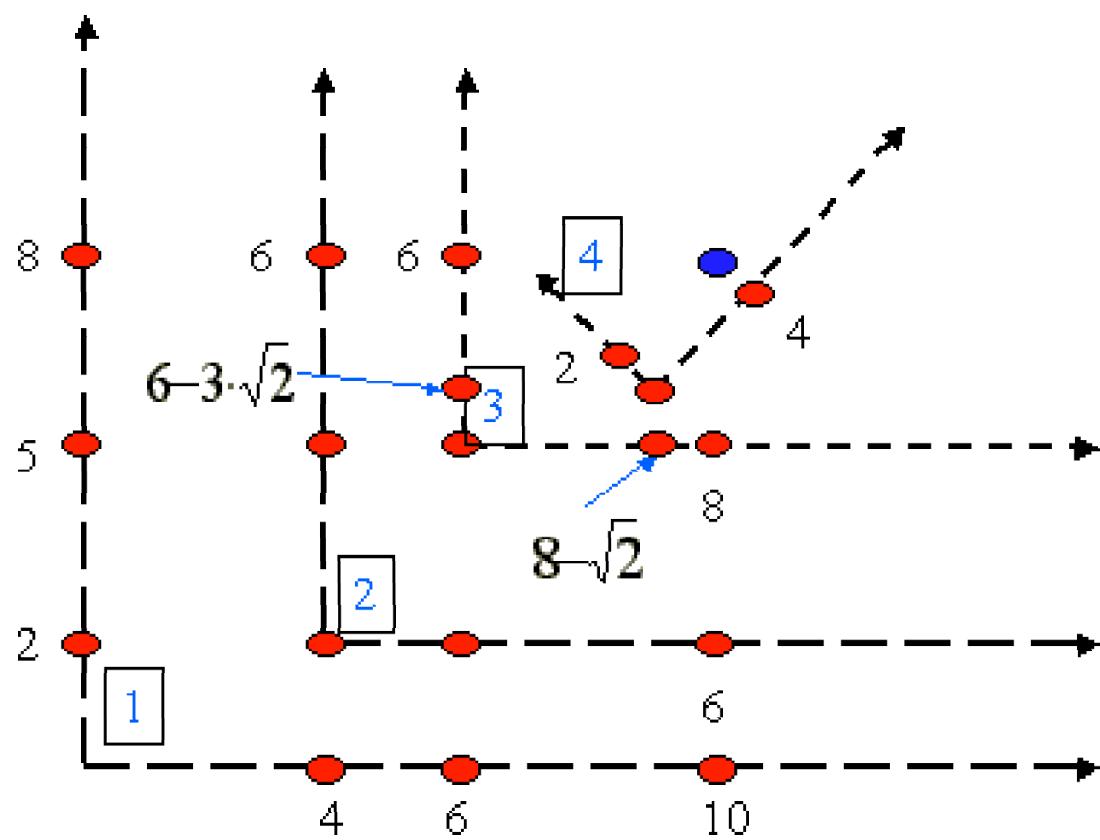


Figura 5.10: Coordonatele unui punct în mai multe sisteme de coordonate

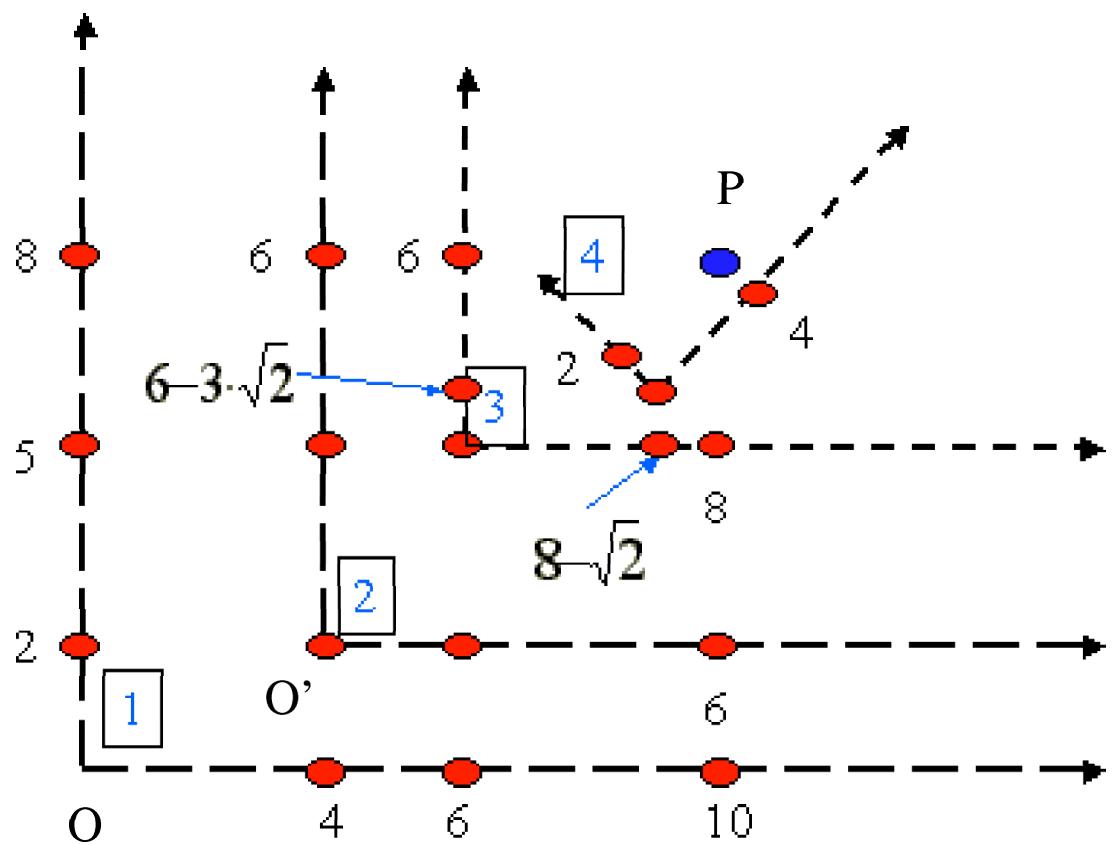


Figura 5.11: Transformarea coordonatelor lui P din 2 în 1

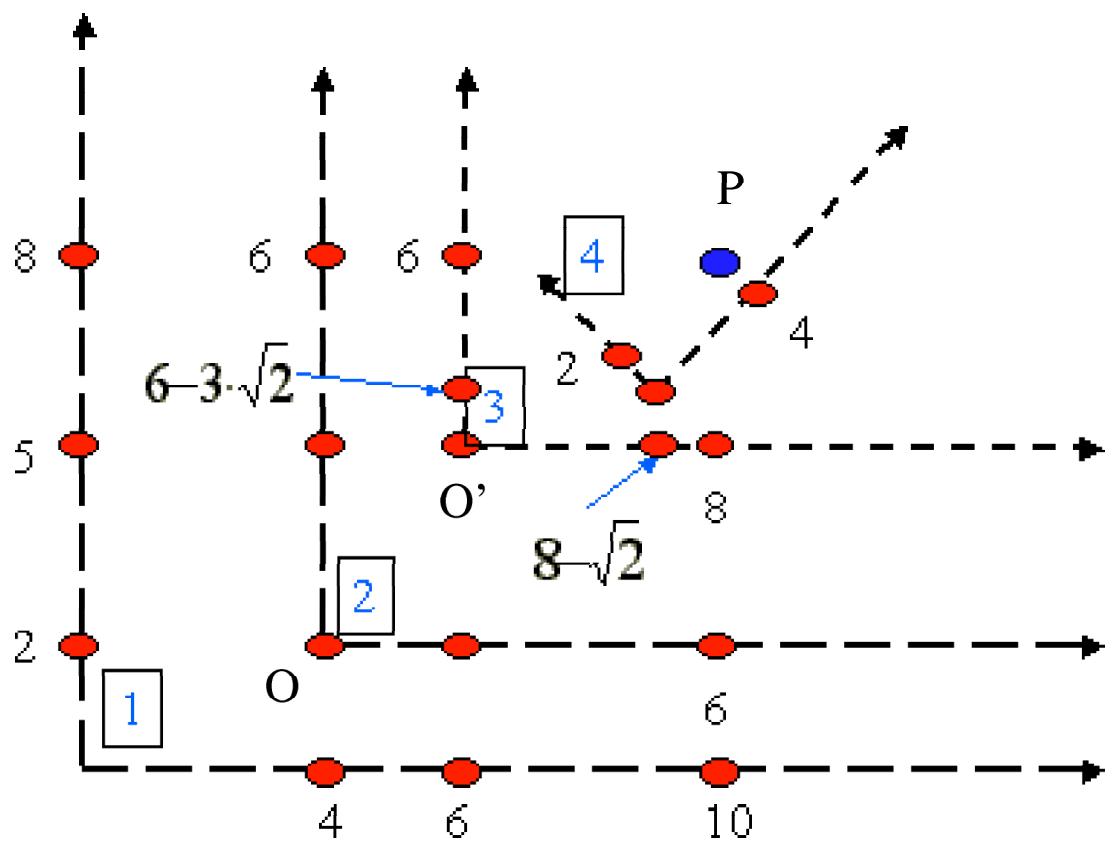


Figura 5.12: Transformarea coordonatelor lui P din 3 în 2

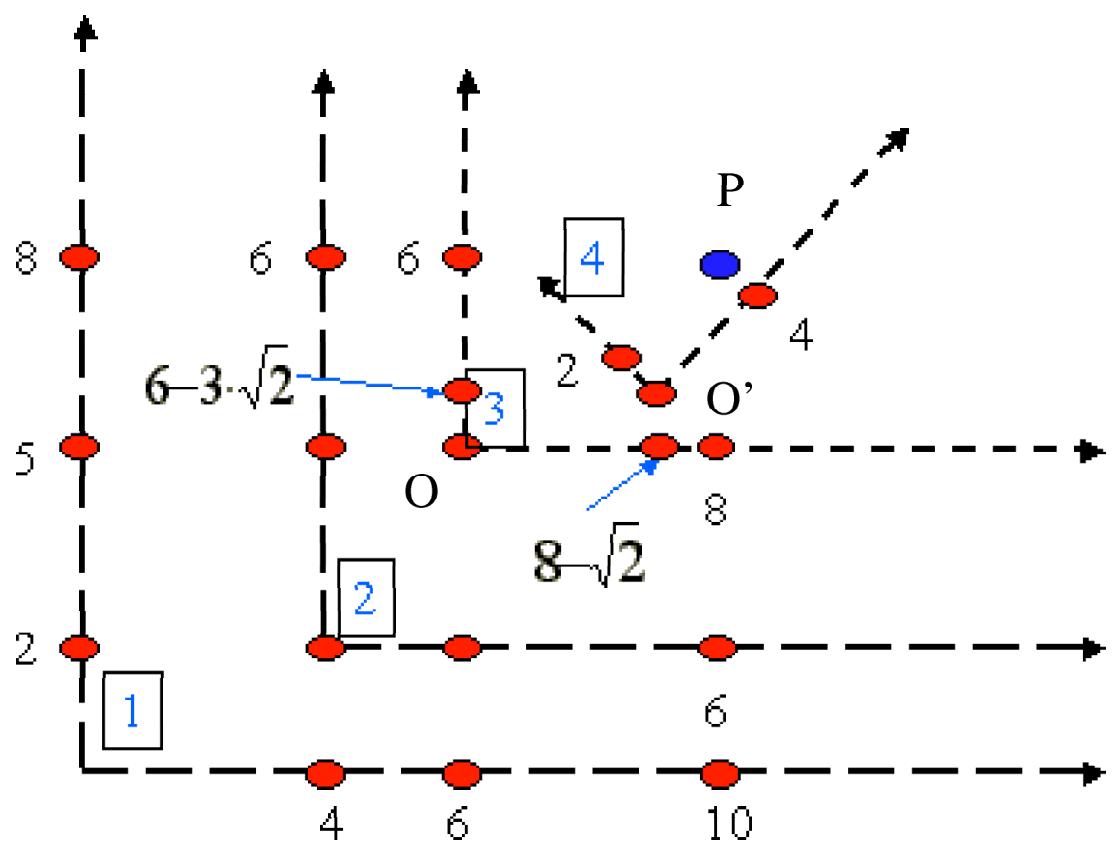


Figura 5.13: Transformarea coordonatelor lui P din 4 în 3

avem $x\vec{i} + y\vec{j} = (4+x')\vec{i} + (2+y')\vec{j}$ și deci se obține $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = T(4, 2) \cdot \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}$ și deci avem $M_{1 \leftarrow 2} = T(4, 2)$.

D.p.d.v. formal transformarea lui P din sistemul de coordonate 3 în sistemul de coordonate 2 se justifică astfel : fie O și O' originile sistemelor de coordonate 2 și 3 (a se vedea în figura 5.12, pag. 78). Avem $\overrightarrow{OP} = \overrightarrow{OO'} + \overrightarrow{O'P} = (2\vec{i} + 3\vec{j}) + (8\vec{i}' + 6\vec{j}') = (2\vec{i} + 3\vec{j}) + \frac{1}{2}(8\vec{i} + 6\vec{j})$ și deci avem $x\vec{i} + y\vec{j} = (2 + \frac{1}{2}x')\vec{i} + (3 + \frac{1}{2}y')\vec{j}$ și deci se obține $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = T(2, 3) \cdot S(\frac{1}{2}, \frac{1}{2}) \cdot \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}$ și deci avem $M_{2 \leftarrow 3} = T(2, 3) \cdot S(\frac{1}{2}, \frac{1}{2})$.

D.p.d.v. formal transformarea lui P din sistemul de coordonate 4 în sistemul de coordonate 3 se justifică astfel : fie O și O' originile sistemelor de coordonate 3 și 4 (a se vedea în figura 5.13, pag. 79). Avem $\overrightarrow{OP} = \overrightarrow{OO'} + \overrightarrow{O'P} = ((8 - \sqrt{2})\vec{i} + (6 - 3\sqrt{2})\vec{j}) + (4\vec{i}' + 2\vec{j}')$.

$$\text{În acest caz } \vec{i}' = R(+45^\circ) \cdot \vec{i} = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \\ 0 \end{pmatrix} \text{ și deci } \vec{i}' = \frac{\sqrt{2}}{2}(\vec{i} + \vec{j}).$$

$$\text{Similar avem } \vec{j}' = R(+45^\circ) \cdot \vec{j} = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \\ 0 \end{pmatrix} \text{ și deci } \vec{j}' = \frac{\sqrt{2}}{2}(-\vec{i} + \vec{j}).$$

Deci avem $\vec{i}' = \frac{\sqrt{2}}{2}(\vec{i} + \vec{j})$ și $\vec{j}' = \frac{\sqrt{2}}{2}(-\vec{i} + \vec{j})$ și deci $\overrightarrow{OP} = \overrightarrow{OO'} + \overrightarrow{O'P} = ((8 - \sqrt{2})\vec{i} + (6 - 3\sqrt{2})\vec{j}) + (4\vec{i}' + 2\vec{j}') = ((8 - \sqrt{2})\vec{i} + (6 - 3\sqrt{2})\vec{j}) + 4\frac{\sqrt{2}}{2}(\vec{i} + \vec{j}) + 2\frac{\sqrt{2}}{2}(-\vec{i} + \vec{j}) = ((8 - \sqrt{2}) + x'\frac{\sqrt{2}}{2} - y'\frac{\sqrt{2}}{2})\vec{i} + ((6 - 3\sqrt{2}) + x'\frac{\sqrt{2}}{2} + y'\frac{\sqrt{2}}{2})\vec{j}$.

Obținem deci $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = T(8 - \sqrt{2}, 6 - 3\sqrt{2}) \cdot R(+45^\circ) \cdot \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}$ și deci avem $M_{3 \leftarrow 4} = T(8 - \sqrt{2}, 6 - 3\sqrt{2}) \cdot R(+45^\circ)$.

Observația 5.5

Matricea transformării punctelor din sistemul de coordonate ce respectă regula mâinii drepte într-un sistem de coordonate ce respectă regula mâinii stângi (și viceversa) este

$$M_{L \leftarrow R} = M_{R \leftarrow L} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Am văzut în secțiunile precedente că putem aborda transformările geometrice ca având loc în același sistem de coordonate (ceea ce presupune că obiectele sunt definite inițial unul peste celălalt în același sistem de coordonate și de aici sunt transformate în poziția dorită). Mai putem avea și altă perspectivă : fiecare obiect este definit în propriul sistem de coordonate

și apoi este scalat, rotit, translatat prin redefinirea coordonatelor sale în noul sistem de coordonate.

Să considerăm exemplul 5.5, pag. 67. Acesta presupune aplicarea translației $T(-x_1, -y_1)$ astfel încât punctul $P(x_1, y_1)$ să ajungă în originea sistemului de coordonate. Putem vedea lucrurile și altfel : în figura 5.14, pag. 81 se observă că transformarea sistemului de coordonate 1 în sistemul de coordonate 2 este $M_{2 \leftarrow 1} = T(-x_1, -y_1)$ și deci $M_{1 \leftarrow 2} = T(x_1, y_1)$. Regula este următoarea : putem fie aplica o transformare geometrică M într-un unic sistem de coordonate fie schimba sistemul de coordonate inițial i într-un sistem de coordonate j și în acest caz avem $M_{i \leftarrow j} = M^{-1}$.

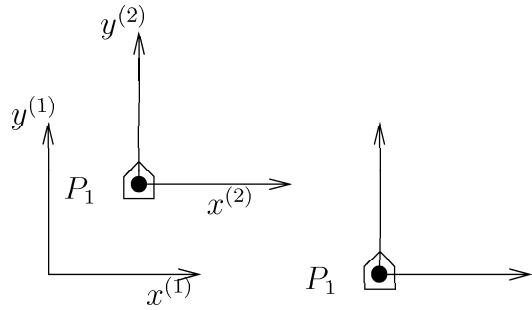


Figura 5.14: Transformarea sistemului de coordonate

Capitolul 6

Lumină (a)cromatică

Culoarea unui obiect depinde nu numai de obiectul în sine ci și de sursa de lumină care îl luminează, de culoarea zonelor adiacente și de sistemul vizual uman.

Unele obiecte reflectă lumina (de ex. un perete, o foaie de hârtie) iar altele o transmit (de ex. o folie de celofan, sticla).

6.1 Lumină acromatică

Senzatiile vizuale acromatice pot fi descrise utilizând doar culorile negru, gri și alb.

Definiția 6.1

Lumină acromatică este ceea ce se poate observa pe un ecran TV sau monitor A/N¹.

Lumină acromatică are un singur atribut : *cantitatea*.

Definiția 6.2

Cantitatea de lumină poate fi descrisă prin conceptul fizic de *energie* (caz în care utilizăm termenii *intensitate* și *luminanță*) sau prin conceptul psihologic de *intensitate percepță* (caz în care utilizăm termenul de *strălucire*²).

Diverselor niveluri de intensitate li se asociază scalari : 0 (reprezentând culoarea neagră) și 1 (reprezentând culoarea albă). Nivelurile între 0 și 1 reprezintă nuanțe de gri.

6.1.1 Selectarea intensităților

În lucrul cu intensitățile luminii acromatice se pun 2 probleme :

¹alb/negru

²*brightness* în engleză

Problema 6.1

Dacă dorim să avem $n + 1$ niveluri de intensitate ($n \in \mathbf{N}$), cum le plasăm în intervalul $[0, 1]$?

Problema 6.2

În mod practic, de câte niveluri de intensitate este nevoie ? Câte niveluri de intensitate sunt suficiente ?

În cazul problemei 6.1, pag. 86 să presupunem că dorim să avem 256 ($n = 255$) niveluri de intensitate :

- Dacă am plasa 128 niveluri în $[0, 0.1]$ și alte 128 niveluri în $[0.9, 1]$ atunci trecerea de la intensitatea 0.1 la 0.9 ar apărea discontinuă (spre deosebire de celelalte tranziții).
- Dacă le-am plasa uniform în intervalul $[0, 1]$ (la distanță de $\frac{1}{255}$) atunci această alegere nu ar fi conformă cu o anumită caracteristică a ochiului uman :

Fapt 6.1 *Ochiul uman percepce nu atât valori absolute ale intensității cât rapoarte între intensități.*

Astfel trecerea de la intensitățile 0.1 și 0.5 la 0.11 și 0.55 este resimțită ca identică. În concluzie vom amplasa nivelurile de intensitate la distanțe logaritmice (și nu liniare) pentru a obține diferențe egale în strălucire, așa cum rezultă din figura 6.1, pag. 87.

Pentru obținerea celor $n + 1$ niveluri de intensitate procedăm astfel : fie I_0 nivelul de intensitate cel mai de jos posibil (din motive practice, pentru un CRT avem $I_0 \in [0.005, 0.025]$).

Alegem $I_1 = r \cdot I_0$, $I_2 = r \cdot I_1 = r^2 \cdot I_0$, ..., $I_j = r^j \cdot I_0$ ($0 \leq j \leq n$), ..., $I_n = r^n \cdot I_0 = 1$. Deci $r = \left(\frac{1}{I_0}\right)^{\frac{1}{n}}$ și $I_j = r^j \cdot I_0 = I_0^{1-\frac{j}{n}} = I_0^{\frac{n-j}{n}}$, $0 \leq j \leq n$.

Exemplul 6.1

Dacă $n = 3$, $I_0 = \frac{1}{8}$ atunci $r = 2$ și avem nivelurile de intensitate : $\{\frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1\}$.

Definiția 6.3

Se numește *domeniu dinamic*³ valoarea $\frac{1}{I_0}$ (raportul dintre intensitatea maximă și minimă).

Observația 6.1

Din punct de vedere practic valoarea unui pixel și valoarea intensității sale calculată conform ecuației $I_j = r^j \cdot I_0$, $0 \leq j \leq n$ diferă. Se procedează astfel :

1. Intensitatea luminii pe ecranul unui CRT depinde de energia unei unde de electroni care stimulează niște particule de fosfor. Practic avem $I = k_1 \cdot N^\gamma$, k_1, γ constante și $\gamma \in [2.2, 2.5]$.

³dynamic range în engleză

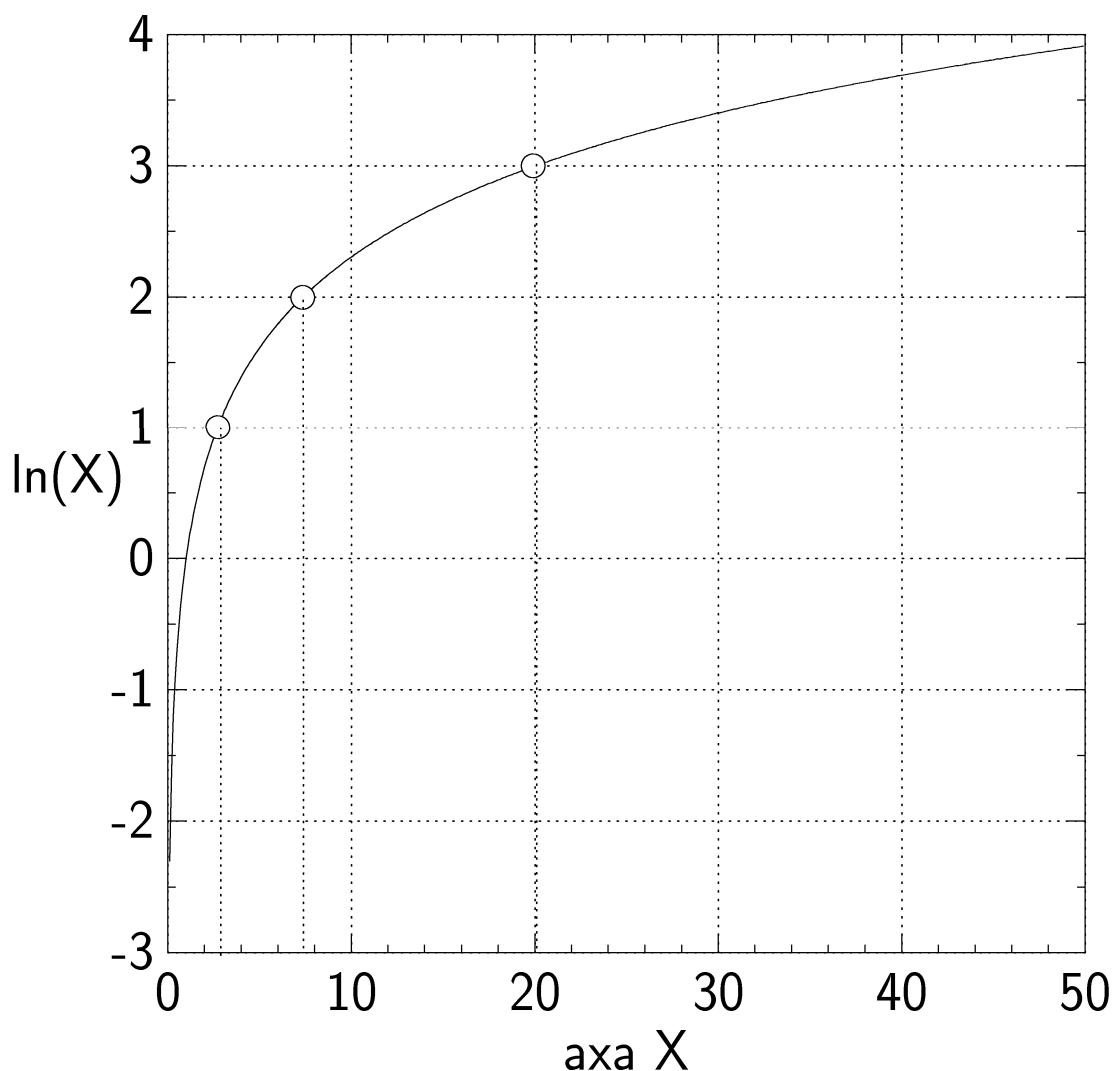


Figura 6.1: Amplasarea nivelurilor de intensitate

2. N este proporțională cu valoarea V a unui pixel și deci avem $I = K \cdot V^\gamma \Leftrightarrow V = (\frac{I}{K})^{\frac{1}{\gamma}}$.
3. Să presupunem că dorim să determinăm valoarea V a unui pixel astfel încât aceasta să creeze intensitatea I . Se procedează astfel :
- Determinăm j astfel încât intensitatea I_j este cea mai apropiată de I : $I = I_j = r^j \cdot I_0 \Rightarrow j = \left[\log_r \left(\frac{I}{I_0} \right) \right]$.
 - Deoarece $V = (\frac{I}{K})^{\frac{1}{\gamma}}$ vom avea $V_j = \left[\left(\frac{I_j}{K} \right)^{\frac{1}{\gamma}} \right]$.

În cazul problemei 6.2, pag. 86 prin termenul de *suficient* ne gândim la numărul de intensități necesar pentru ca reproducerea unei imagini A/N (de ex. o fotografie A/N) să aibă un aspect *continuu* (i.e., între 2 pixeli vecini să nu existe diferențe mari de intensitate). Aspectul continuu este posibil doar pentru $r = 1.01$. S-a putut observa că dacă $r < 1.01$ ochiul nu poate distinge între intensitățile I_j și I_{j+1} . Deci, din ecuația $r = \left(\frac{1}{I_0} \right)^{\frac{1}{n}}$ obținem $r^n = \frac{1}{I_0} \Rightarrow n = \log_r \left(\frac{1}{I_0} \right)$ și deci $n = \log_{1.01} \left(\frac{1}{I_0} \right)$.

Câteva din valorile $\frac{1}{I_0}$ și n se regăsesc în tabelul 6.1, pag. 88.

	$\frac{1}{I_0}$	n
CRT	50 – 200	400 – 530
hârtie fotografică	100	465
hârtie A/N	100	465
hârtie culori	50	400
ziar A/N	10	234

Tabelul 6.1: Valori pentru $\frac{1}{I_0}$ și n

6.1.2 Aproximarea intensităților prin autotipie

Problema 6.3

În cazul ecranelor (sau al dispozitivelor de tipărire) nu putem avea decât un număr limitat de intensități (2 pentru ecrane monocrome sau 4, 8 dacă avem ecrane rastru cu 2, 3 biți/pixel). Cum putem obține, în aceste cazuri, niveluri de intensitate suplimentare ?

Pentru rezolvarea problemei 6.3, pag. 88 ne folosim de faptul că *ochiul realizează o integrare spațială* : dacă privim o zonă foarte mică de la o distanță suficient de mare atunci ochiul percepă o intensitate medie (și nu intensitatea fiecărui pixel în parte).

Tehnica de *autotipie*⁴ constă în simularea intensității prin cercuri negre de arie direct proporțională cu $1 - I$, unde I reprezintă valoarea intensității din fotografia originală. Aceasta tehnică se folosește, în general, pentru tipărirea fotografiilor A/N.

Dispozitivele de ieșire grafice pot aproxima cercurile de arie variabilă folosite în autotipie astfel : o zonă de 2×2 pixeli monocromi poate simula 5 niveluri de intensități diferite, cu prețul reducerii la jumătate a rezoluției spațiale⁵ pe fiecare axă (vezi figura 6.2, pag. 89).

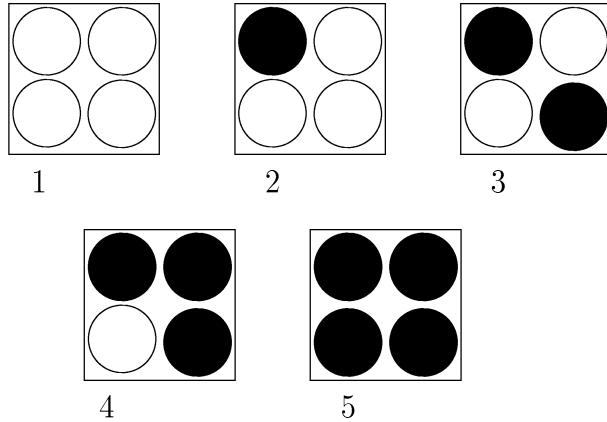


Figura 6.2: Simulare intensități

În general avem următoarea proprietate :

Proprietatea 6.1

Un grup de $n \times n$ pixeli monocromi poate furniza $n^2 + 1$ niveluri de intensitate. ▀

Exemplul 6.2

O zonă de 3×3 pixeli monocromi poate simula 10 intensități : vezi figura 6.3, pag. 90.

Putem sintetiza aceste niveluri de intensitate într-o matrice de oscilație⁶ : $\begin{pmatrix} 6 & 8 & 4 \\ 1 & 0 & 3 \\ 5 & 2 & 7 \end{pmatrix}$, unde nivelul de intensitate I se obține setând toți pixelii cu valori mai mici decât I . ▀

Putem avea mai multe modele pentru zonele de $n \times n$ pixeli utilizate în autotipie, dar aceste modele trebuie să satisfacă următoarele reguli :

- Un anumit model nu trebuie să introducă noi obiecte vizuale în zone de intensități egale. De exemplu, dacă în figura 6.3, pag. 90 în loc de modelul dat pentru 3 am fi utilizat modelul 3' am produce apariția unei linii orizontale în orice zonă a imaginii inițiale colorată uniform cu pixeli de intensitate 3.

⁴halftoning în engleză

⁵i.e., dacă o imagine era reprezentată pe $n \times n$ pixeli, fiecare pixel putând avea 5 intensități, în cazul când avem un ecran monocrom, pentru a avea o imagine la fel de detaliată avem nevoie de $2n \times 2n$ pixeli

⁶Dither matrix în engleză

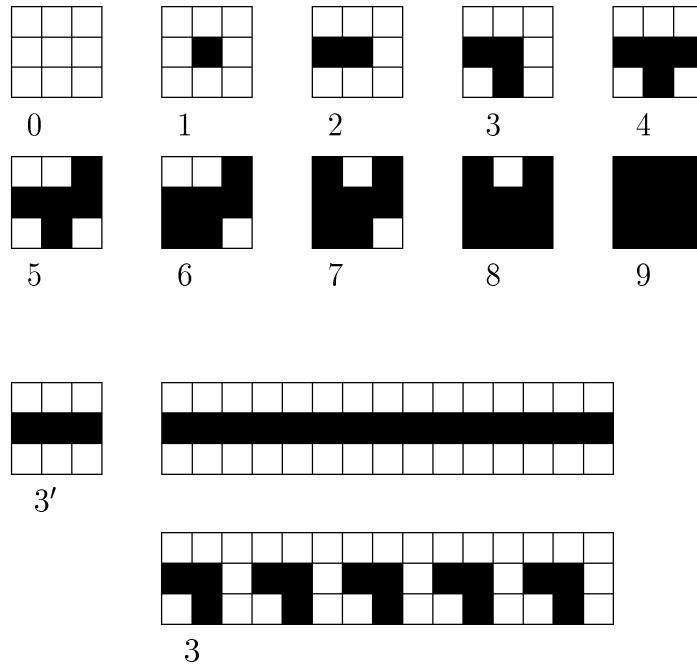


Figura 6.3: 10 niveluri de intensitate

- Modelele trebuie să alcătuiască o secvență crescătoare astfel încât dacă un pixel era intensificat pentru simularea nivelului de intensitate j atunci va rămâne intensificat și pentru toate nivelurile de intensitate $k > j$.
- Modelele trebuie să crească dinspre centru înspre exterior pentru a simula efectul de creștere a mărimi punctului.
- Pixelii setați (1) trebuie să fie grupați. Nu se admit pixeli 1 răsfirați.

Aproximarea diferitelor niveluri de intensitate nu se limitează doar la ecrane monocrome. În cazul unui ecran cu 2 biți pe pixel (i.e., 4 niveluri de intensitate pentru un pixel), dacă utilizăm zone de 2×2 pixeli, se pot simula 13 niveluri de intensitate (bineînțeles cu prețul reducerii rezoluției spațiale) : vezi figura 6.4, pag. 91.

6.2 Lumină cromatică

Percepția culorii implică 3 valori : *nuanță*⁷, *saturație*⁸, *luminozitate*⁹. Nuanța distinge familiile de culori : avem nuanțe de roșu, verde, albastru, etc. Saturația se referă la distanța culorii

⁷ *Hue* în engleză

⁸ *Saturation* în engleză

⁹ *Lightness* în engleză

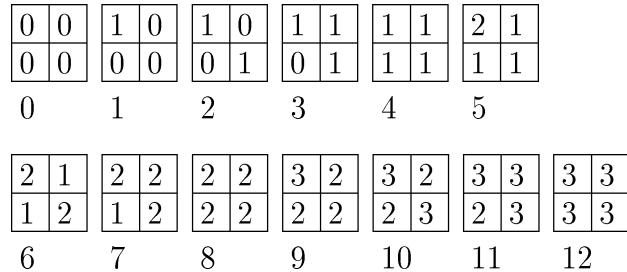


Figura 6.4: 13 niveluri de intensitate

respective față de culoarea gri care are aceeași intensitate (de ex., culoarea roșie este puternic saturată iar cea roz este relativ nesaturată, adică mai apropiată de culoarea gri de aceeași intensitate). Luminozitatea reprezintă intensitatea percepță a unui obiect care reflectă lumină. Strălucirea¹⁰ se referă la intensitatea percepță a unui obiect care emite lumină (de ex., un bec, soarele, CRT).

Există mai multe modalități de specificare și de măsurare a culorilor :

- prin compararea culorii necunoscute cu un set de culori standard (modelul de culori Munsell¹¹),
- în modelul artistic culoarea este specificată prin 3 parametri : nuanță¹², umbrire¹³, ton¹⁴, vezi figura 6.5, pag. 92. O *nuanță* rezultă prin adăugarea culorii alb unei culori pure, astfel scăzând saturăția. O *umbrire* provine din adăugarea de culoare neagră unei culori pure, astfel scăzând luminozitatea. Un *ton* este obținut adăugând atât culoarea alb cât și negru unei culori pure. Obținem astfel culori de aceeași nuanță (hue) dar cu saturatie și luminozitate variabile. Amestecarea culorilor alb/negru produce culori gri.

6.2.1 Modele de culori pentru grafica raster

Definiția 6.4

Un *model de culori* constă într-un sistem (cartezian - în cazul RGB, cilindric - în cazul HSV) de coordonate 3D și o submulțime a \mathbf{R}^3 care conține toate culorile dintr-o anumită gamă de culori. ▀

De exemplu, modelul RGB constă din cubul $[0, 1]^3$ definit într-un sistem de coordonate cartezian 3D.

¹⁰Brightness în engleză

¹¹Munsell space în engleză

¹²Tint în engleză

¹³Shade în engleză

¹⁴Tone în engleză

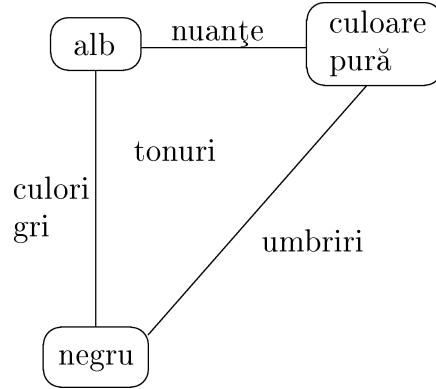


Figura 6.5: Modelul artistic

Modelul RGB

Modelul de culori RGB¹⁵ este utilizat pentru CRTS în culori precum și în grafica rastru. Spațiul de culori este cubul unitate ($[0, 1] \times [0, 1] \times [0, 1]$) (vezi figura 6.6, pag. 93). O anumită culoare este reprezentată printr-un triplet de culori primare : (roșu, verde, albastru). Aceste culori primare sunt și *aditive* : fiecarei culori $C(r, g, b)$ îi corespunde bijectiv un punct din spațiul de culori RGB (punct care este dat de vectorul $r \cdot \vec{R} + g \cdot \vec{G} + b \cdot \vec{B}$, unde $\vec{R}, \vec{G}, \vec{B}$ sunt versori). Pe diagonala principală regăsim diverse nuanțe de gri.

Modelul HSV

Spre deosebire de alte modele de culori orientate spre hardware (de ex. RGB), modelul HSV¹⁶ este orientat spre utilizator, fiind apropiat de modelul artistic. Spațiul de culori HSV este o piramidă hexagonală într-un sistem de coordonate polar (vezi figura 6.7, pag. 94). Culorile gri sunt situate pe dreapta *black-white*.

O culoare este specificată în modelul HSV prin tripletul (h, s, v) , unde $h \in [0^\circ, 360^\circ]$, $s \in [0, 1]$ și $v \in [0, 1]$. Parametrul h (hue) semnifică nuanța culorii, s (saturation) specifică saturarea culorii iar v (value) specifică strălucirea percepă a culorii. În modelul HSV, variind h alegem o culoare (dintr-o familie de culori), variind s modificăm saturarea culorii (făcând-o să varieze între culoarea gri cu aceeași intensitate și culoarea pură) iar variind v variem contribuția culorii negre (variind astfel luminozitatea).

Corespondența dintre modelele RGB și HSV

Conversia culorilor din modelul RGB în HSV se efectuează proiectând cubul unitate RGB pe un plan perpendicular pe diagonala principală în punctul *white*.

¹⁵Red, Green, Blue în engleză

¹⁶Hue, Saturation, Value în engleză

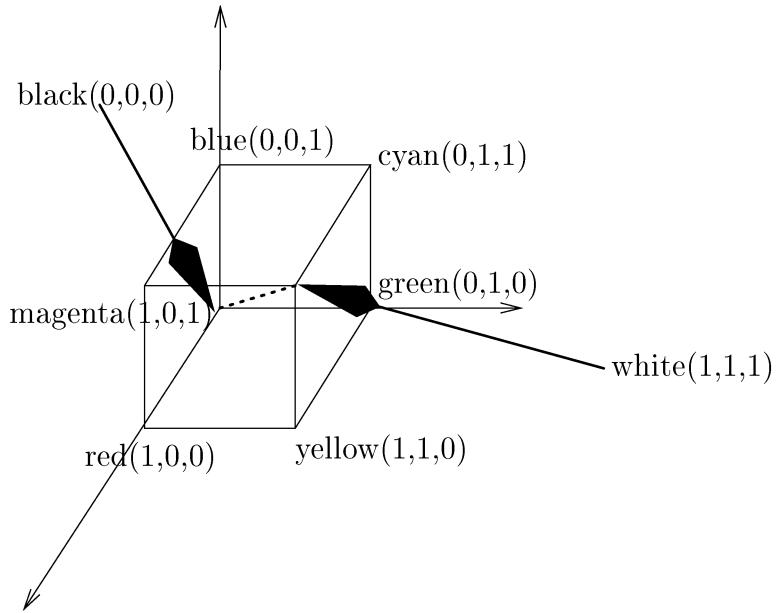


Figura 6.6: Modelul RGB

Algoritm 12 (Conversia RGB în HSV)

Date de intrare : r, g, b : real

Date de ieșire : h, s, v : real

Metodă :

```

procedure RGBtoHSV(var h,s,v : real, r,g,b : real) {
  const int UNDEF = -1;
  real max, min, delta;

  max = valoarea_maxima(r,g,b);
  min = valoarea_minima(r,g,b);
  v = max;
  s = (max != 0.0) ? ((max - min)/max) : 0;
  if (s == 0) h = UNDEF;
  else {
    delta = max - min;
    if (delta == 0.0) { h = UNDEF; s = 0; v = 1; }
    if (max == 0) { h = s = UNDEF; v = 0; }
    if (r == max) h = (g - b) / delta;
    else if (g == max) h = (b - r) / delta + 2.0;
    else if (b == max) h = (r - g) / delta + 4.0;
    else {}
  }
}

```

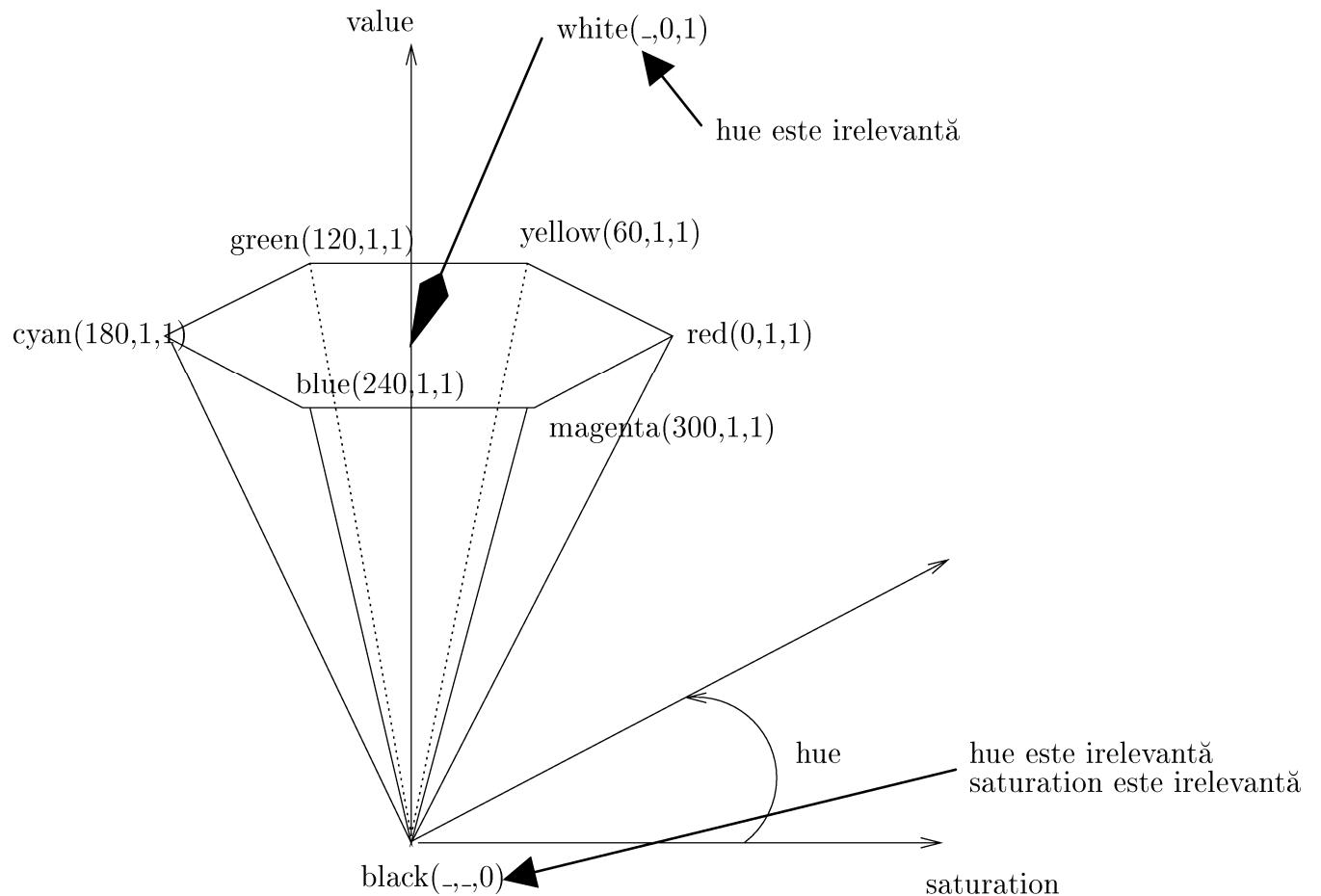


Figura 6.7: Modelul HSV

```

    h = h * 60;
    if (h < 0) h = h + 360;
}
}

```

Algoritm 13 (Conversia HSV în RGB)

Date de intrare : h, s, v : real

Date de ieșire : r, g, b : real

Metodă :

```

procedure HSVtoRGB(var r,g,b : real, h,s,v : real) {
    const int UNDEF = -1;
    real f,p,q,t;
    int i;

    if (s == 0) {
        if (h == UNDEF) r = g = b = v;
        else printf("Eroare");
    }
    else {
        if (h == 360) h = 0;
        h = h / 60;
        i = floor(h);
        f = h - i;
        p = v * (1 - s);
        q = v * (1 - s * f);
        t = v * (1 - (s * (1 - f)));
        switch (i) {
            case 0 : r = v; g = t; b = p; break;
            case 1 : r = q; g = v; b = p; break;
            case 2 : r = p; g = v; b = t; break;
            case 3 : r = p; g = q; b = v; break;
            case 4 : r = t; g = p; b = v; break;
            case 5 : r = v; g = p; b = q; break;
        }
    }
}

```

6.2.2 Specificarea interactivă a culorii

În aplicațiile care utilizează culori pentru drepte, text, suprafete, etc. există mai multe modalități (vezi figura 6.8, pag. 97) prin care se permite utilizatorului selectarea unei culori :

- Utilizatorul poate alege o culoare din mai multe mostre de culori (modelul indexat),
- Utilizatorul poate specifica coordonate R, G, B ,
- Utilizatorul poate specifica coordonate H, S, V .

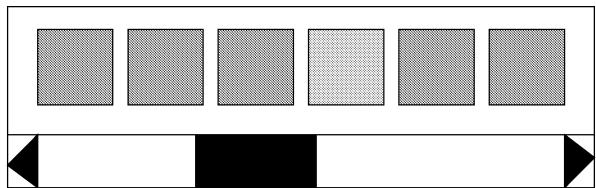
6.2.3 Utilizarea culorii în grafica pe calculator

Un *principiu fundamental* al utilizării culorii în diverse aplicații este următorul : utilizarea culorii trebuie să aibă un scop funcțional și să nu aibă un subînteleș.

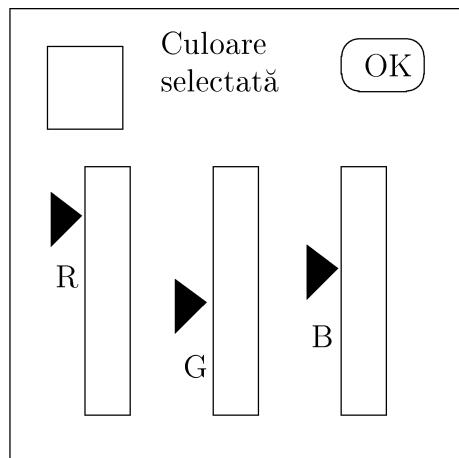
Reguli de estetică ale culorii

1. Culorile trebuie selectate conform unei anumite metode, de obicei prin traversarea unui drum continuu în modelul culorilor sau prin restricționarea culorilor la anumite planuri sau secțiuni hexagonale.
2. Nu se recomandă alegerea aleatoare a H, S .
3. Dacă pentru un grafic se utilizează câteva culori atunci se recomandă pentru fundal¹⁷ utilizarea unei culori care este complementară unei culori utilizate în grafic (culoarea complementara unei culori C este acea culoare care combinată cu C produce o culoare neutră : alb, gri sau negru. Pentru modelul RGB se obține prin simetrie față de centrul cubului iar pentru modelul HSV se obține prin simetrie față de centrul hexagonului). Dacă se utilizează mai multe culori atunci pentru fundal se recomandă culoarea gri.
4. Dacă alăturarea a 2 culori nu este armonioasă se recomandă utilizarea de chenare subțiri de culoare neagră.
5. Dacă dorim să asociem diverse înțelesuri culorilor atunci trebuie să luăm câteva precauții :
 - (a) Codurile culorilor pot avea înțelesuri implice : de ex., *roșu* sugerează atenție, *eroare*, *verde* corect.
 - (b) Culorile strălucitoare denotă o importanță crescută asociată unui element.
 - (c) Două obiecte cu aceeași culoare pot fi interpretate ca având același cod (ceea ce nu e valabil întotdeauna).

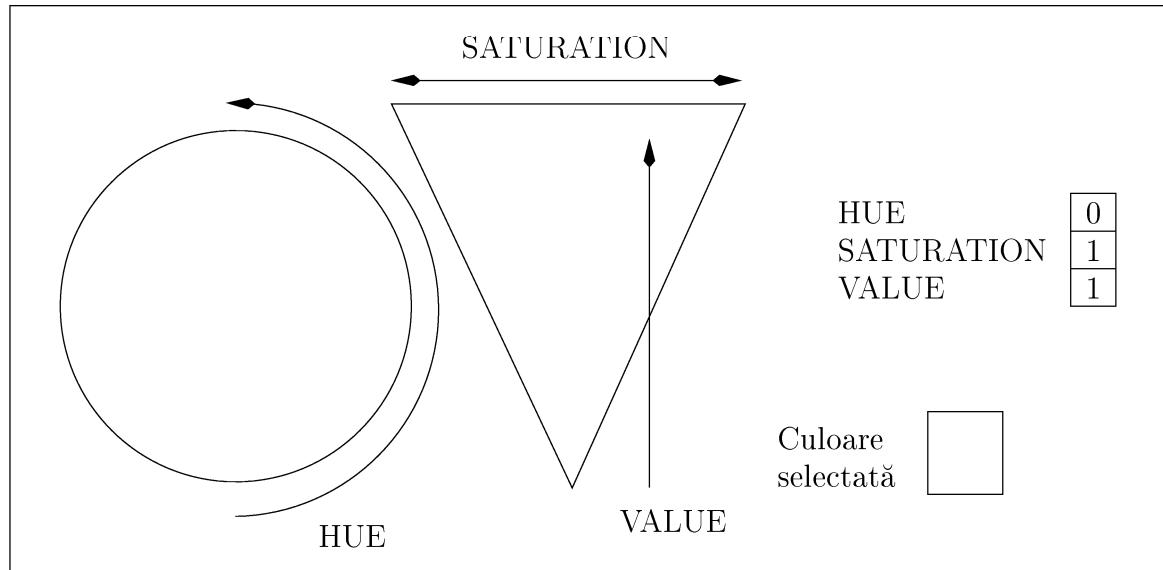
¹⁷Background în engleză



Modelul indexat



Modelul RGB



Modelul HSV

Figura 6.8: Specificarea interactivă a culorii

6. Din regula 5, pag. 96 rezultă că utilizarea culorilor în meniuri, ferestre de dialog, chenare de ferestre trebuie să fie restricționată.
7. Detaliile fine (segmente de dreaptă, text) trebuie să difere de fundal nu doar în cromaticitate ci și în strălucire (în mod deosebit când se utilizează culori conținând albastru).
8. Combinații greșite de culori : albastru și negru (deoarece diferă foarte puțin în strălucire), galben și alb (idem). Combinații bune de culori : galben (pentru fundal) și negru (pentru text), text alb pe fundal albastru. Alte combinații greșite sunt : roșu și verde (când nu sunt strălucitoare pot crea probleme persoanelor care nu percep diferențe între cele două culori).
9. Ochiul nu poate distinge culoarea obiectelor foarte mici, deci nu se recomandă, într-o imagine colorarea pixelilor vecini cu culori distincte.
10. Culoarea percepută de către ochi a unei suprafețe este afectată de suprafața înconjurătoare. Se recomandă deci culori gri sau nesaturate pentru suprafețele înconjurătoare.
11. Culoarea unei suprafețe poate modifica mărimea sub care este percepută de către ochi (de ex., suprafețele colorate roșu sunt percepute ca fiind mai mari decât cele colorate verde).
12. Nu se recomandă utilizarea de suprafețe mari colorate cu o singură culoare.
13. Nu se recomandă colorarea a două suprafețe cu două culori de la capetele opuse ale spectrului (ROGVAIV) : de ex., roșu pentru fundal și albastru pentru prim-plan¹⁸. În acest caz roșu apare mai aproape iar albastru mai departe, invers față de ceea ce dorim.

¹⁸ *Foreground* în engleză