

6

REPREZENTAREA VIZUALĂ A DATELOR

Forma și formatul afișării datelor pe monitor (ca ecou al introducerii de la tastatură sau ca rezultat al scrierii) precum și modul în care se desfășoară conversația în timpul acestui proces constituie interfața programului cu utilizatorul. Realizarea unei interfețe eficiente și atractive trebuie să constituie pentru programator un obiectiv la fel de important ca și cel al atingerii performanțelor ridicate de eficiență a programului însuși.

6.1 Resursele și modurile de lucru video ale unui microcalculator

În procesul afișării video a datelor sunt implicate componente hardware și software, care interacționează pentru crearea imaginii pe ecranul monitorului (figura 6.1).

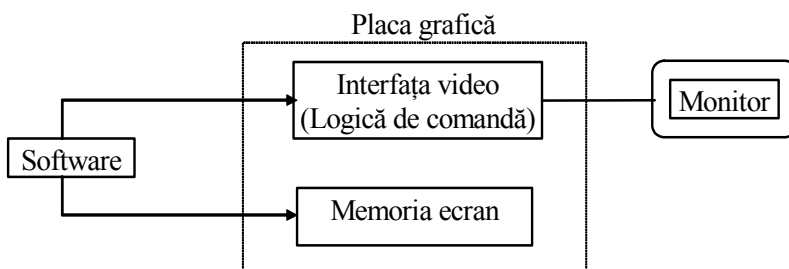


Fig. 6.1 Componente implicate în afișarea video a datelor

- *Memoria ecran*, de tipul RAM, are rolul de a stoca, sub formă binară, imaginea care se afișează pe ecran. Ea poate avea diferite capacități - uzual, până la 1

MB – și este împărțită în pagini de o anumită mărime, în care se pot memora concomitent una sau mai multe imagini de ecran. În plus, se pot memora definițiile binare ale unor seturi de caractere pentru afișarea textelor cu modele de caractere (fonturi) ale utilizatorului, altele decât cele definite de generatorul de caractere al interfeței video.

- *Interfața video* (adaptorul video) are rolul de a comanda și controla monitorul pentru crearea imaginii pe tub catodic. Ea comunică cu UC a microcalculatorului și cu memoria ecran prin magistrale de date și de adrese. UC și interfața pot utiliza simultan memoria ecran (prin porturi diferite), astfel încât microprocesorul poate înscrie date în memoria ecran în timp ce interfața citește memoria pentru a realiza afișarea (figura 6.2).

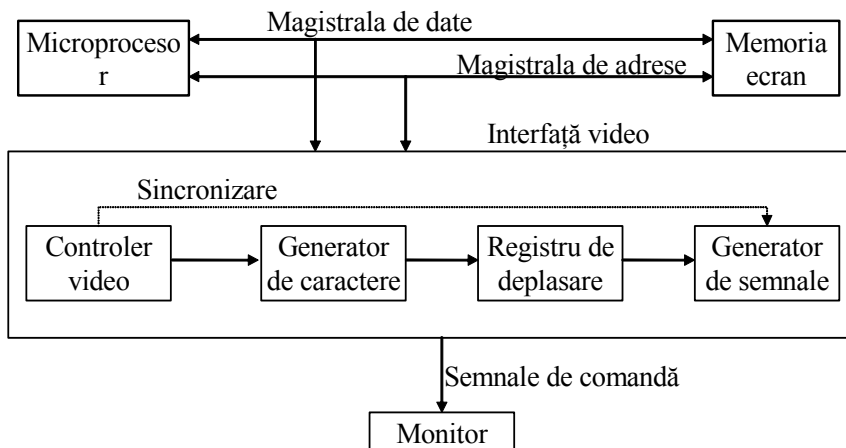


Fig. 6.2 Modul de interacțiune a interfeței video

Controlerul video (logica de comandă) al interfeței citește periodic, cu o anumită frecvență, memoria de ecran și formează semnalele de comandă pentru monitor în vederea creării și menținerii imaginii video. Dacă frecvența de refacere a imaginii pe ecran (frecvența de reîmprospătare) este de peste 50 Hz (de obicei 50-70 Hz), atunci, datorită inerției ochiului, imaginea apare ca fiind continuă și fără pâlپări.

Generatorul de caractere este dispozitivul interfeței care produce imaginile (matricele) punctiforme ale caracterelor ASCII de afișat. Prin comanda dată de controler și utilizând caracterele ASCII citite din memoria ecran, generatorul de caractere produce, la ieșire, matricele discrete care definesc caracterele. Pe baza lor și a atributelor de afișare asociate, aduse de controler din memoria de ecran, generatorul de semnale produce semnalele seriale care comandă monitorul. Serializarea în producerea semnalelor este asigurată prin intermediul unui registru de deplasare care furnizează, pe rând, câte un element de imagine.

- *Monitorul* utilizează un tub catodic monocrom sau color și creează imaginea pe ecran, de sus în jos. Fascicolul de electroni “aprinde”, pe rânduri sau linii, puncte elementare din suprafața fluorescentă a tubului catodic, în conformitate cu semnalele discrete de comandă trimise de sistemul de baleere a ecranului.

Punctele elementare de imagine sunt denumite *pixeli* (picture elements) și apar pe monitor organizate matriceal, definind *spațiul fizic* sau *spațiul ecran* (figura 6.3). În acest spațiu, pixelii pot fi adresați printr-o pereche de numere naturale (x, y), unde x este numărul coloanei și y este numărul liniei. De remarcat faptul că, în acest spațiu, abscisa crește normal, de la stânga spre dreapta, dar ordonata crește de sus în jos.

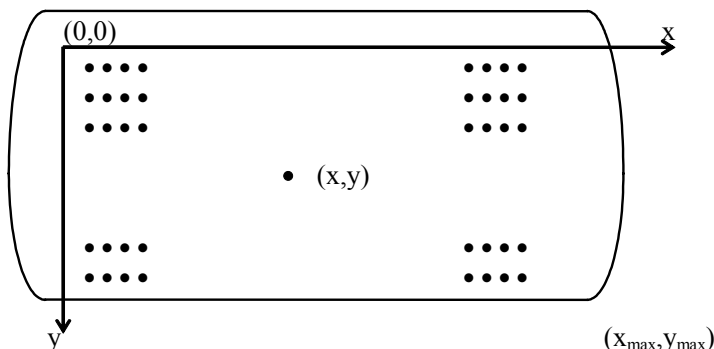


Fig. 6.3 Spațiul ecran

Numărul total de pixeli care pot fi tratați independent constituie o caracteristică a monitorului, denumită *rezoluția ecranului*. Rezoluția poate fi joasă (cca.320 x 200 pixeli), medie (cca.640 x 200 pixeli) și înaltă (peste 640 x 350 pixeli). Interfețele video sunt proiectate astfel încât pot lucra cu monitoare de diferite rezoluții, pentru un monitor simulând toate rezoluțiile inferioare celei date. Imaginea este cu atât mai clară cu cât rezoluția utilizată în realizarea ei este mai mare. De regulă, în construcția microcalculatoarelor, interfața video și memoria de ecran sunt realizate pe o placă unică, numită *placă grafică*. “Puterea” plăcii grafice este cea care dă, în principal, puterea de afișare a datelor pe ecran. În prezent, microcalculatoarele sunt echipate cu plăci grafice de mare putere, de tipul SVGA (Super Video Graphic Array), realizate după standardul firmei IBM pentru utilizatori profesionali. Ele sunt compatibile cu plăci realizate anterior (MDA, CGA și EGA), al căror mod de lucru îl pot realiza, oferind în plus o rezoluție înaltă și o paletă bogată de culori și nuanțe de gri (pentru monitoare monocrome).

- *Partea de software* implicată în afișarea video are rolul de a comanda interfața video în realizarea diferitelor operații necesare afișării: alegerea modului de afișare - text sau grafic - definirea poziției spațiului ecran pentru afișare, definirea atributelor de afișare etc. Ea este constituită din rutine BIOS multifuncționale (cu mai multe servicii) care se apelează prin intermediul tehnicii întreruperilor software. Tipică este INT 10h, care oferă o serie de servicii video standard. În program, înainte de cererea întreruperii, codul serviciului dorit trebuie încărcat într-un registru special

(AH), iar alți parametri, dacă sunt necesari, se încarcă în alte registre (AL, BH, BL, DH, DL).

6.2 Moduri video

Din punctul de vedere al constituirii imaginii, interfața video poate lucra în două moduri: text și grafic.

- În *modul text*, ecranul este văzut ca o matrice de linii și coloane de text. Locul de afișare a unui caracter în spațiul ecran este dat de o pereche (x,y) de numere naturale, cu x=numărul coloanei și y=numărul liniei (figura 6.4.a).

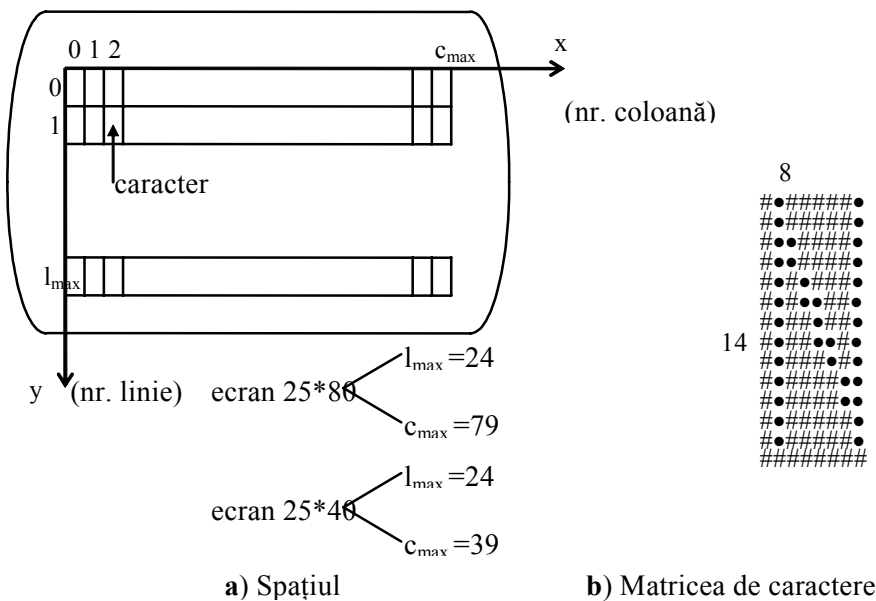


Fig. 6.4 Ecranul în modul text

Deoarece spațiul text rezultă prin gruparea corespunzătoare a liniilor și coloanelor de pixeli, afișarea caracterelor se face, în mod natural, prin matrice de caractere (figura 6.4.b). O astfel de matrice, de regulă 8x14 sau 8x8, definește forma caracterului prin pixelii aprinși, iar fondul prin pixelii stinși. În fondul caracterului se cuprind pixeli de separare a caracterelor pe coloane și linii. Caracterele realizate astfel se numesc *caractere matriceale* sau *rastru*. De regulă, în memoria ROM a calculatorului se definesc patru seturi de caractere, având forme (fonturi) tipice (scriere standard, caractere italice etc). La pornirea calculatorului sau la cererea

programatorului, ele sunt încărcate în memoria de ecran pentru a fi accesibile generatorului de caractere. Un anumit set, implicit sau cerut de utilizator (via INT 10h), va deveni *set curent* pentru generator. Există posibilitatea ca programatorul să-și definească în memoria calculatorului propriile seturi de caractere (cel mult patru odată), pe care să le încarce în memoria ecran și din care, apoi, să definească setul curent (ambele sunt servicii INT 10h).

Datorită adresabilității spațiului text, există posibilitatea de a comanda (INT 10h) poziția de afișare a unui caracter. În acest scop, interfața păstrează în registre speciale coordonatele locului de afișare, iar pe ecran acesta este marcat prin afișarea unui caracter special, denumit *cursor* (de regulă, un dreptunghi umplut sau luminos). Atunci când se afișează un caracter în poziția de la cursor (poziția curentă), acesta se suprascrie peste cursor, iar cursorul este mutat automat cu o poziție spre dreapta și în jos, la începutul noului rând, dacă este cazul. Prin servicii ale rutinei INT 10h se poate defini o formă proprie a cursorului în modul text și se poate afla locul acestuia pe ecran.

Fiecărui caracter de afișat i se asociază un *caracter de atribut* (figura 6.5.a).

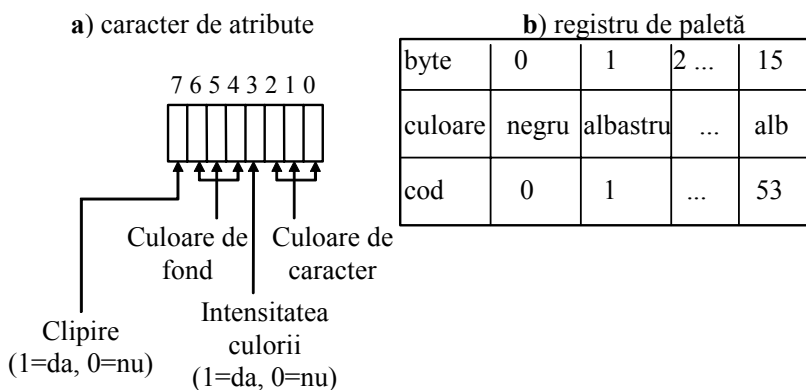


Fig. 6.5 Definirea atributelor de afișare

Caracterul de atribut, prin valorile biților săi, definește *culoarea de fond* a caracterului (background color), *culoarea de caracter* (foreground color) și faptul că imaginea caracterului se va “stinge intermitent” (clipire - blinking). Se observă că se pot defini cel mult 8 culori pentru fond și 16 culori de caracter, dacă se consideră că afișarea normală și intensă definesc două culori diferite. Dacă ecranul este monocrom, atunci se pot combina culorile alb și negru.

Pentru variabilitate în exprimarea culorilor și pentru asigurarea independenței față de convențiile de notare a lor la diverse plăci grafice, s-a adoptat ideea specificării indirecte a culorilor. Interfața posedă un registru, de regulă de 16 octeți, denumit *registru de paletă*, prin care se specifică culorile care pot fi utilizate la un moment dat (figura 6.5.b). Fiecare poziție din registrul de paletă definește, prin cod, o culoare și, în ansamblu, constituie *paleta curentă* de culori.

Astfel, în caracterele de atribute se specifică numărul intrării din paleta curentă (*index*), de unde trebuie extras codul culorii dorite. Pe baza acestei idei, în orice moment, poate fi modificat conținutul registrului de paletă (serviciu INT 10h), ceea ce conduce, în mod automat, la modificarea culorilor pentru întregul text afișat pe ecran. La pornirea calculatorului, registrul de paletă este încărcat cu o *paletă inițială* (implicită), definită în memoria ROM. Programatorul poate schimba, total sau parțial, culorile în registrul de paletă, depinzând de tipul plăcii grafice și aceasta este unica operație pentru care trebuie cunoscute codurile culorilor la placa respectivă. Trebuie remarcat că unele monitoare nu pot afișa toate culorile, deși interfața cu care sunt cuplate acceptă specificarea oricărui atribut.

Definirea caracterului de atribute se face prin serviciul INT 10h, care realizează scrierea caracterului text ca parametru într-un registru (registrul BL). Scrierea unui caracter poate fi însoțită de operația SAU EXCLUSIV (suma modulo-2, notată \oplus) între culoarea caracterului afișat într-o anumită poziție și culoarea noului caracter, destinat a fi afișat în aceeași poziție. Operația poate fi utilizată pentru scrierea și ștergerea caracterelor (ștergerea selectivă). De exemplu, pentru ștergere, dacă culoarea este 0101 și se aplică o suprascriere cu aceeași culoare, atunci, cum $0101 \oplus 0101 = 0000$, se obține un cod care desemnează culoarea fondului curent, adică, de fapt, “disparația” caracterului de pe ecran.

Având în vedere că necesarul de memorie ecran este în jur de $4KB \times 25 \times 80$, depinzând de rezoluția în modulul text, în memoria video se pot focaliza una sau mai multe pagini video (ecrane). Se creează posibilitatea constituirii textului într-o anumită pagină, denumită *pagină video*, selectată din mulțimea paginilor acceptate de interfața respectivă prin serviciul INT 10h. Orice poziție și deplasare a cursorului pe ecran își are echivalentul într-o poziție și o deplasare corespunzătoare în pagina video selectată. Din mulțimea paginilor definite, programatorul poate selecta una, ca pagină de afișare pe ecran, denumită *pagină activă*. Astfel crește viteza de lucru prin suprapunerea operației de afișare cu operația de pregătire a imaginii, într-o nouă pagină video. Interogarea, prin INT 10h, în legătură cu atributele unui anumit caracter al imaginii este, de fapt, o citire din pagina video a memoriei ecran, eventual pagina activă.

Modul text permite definirea, în pagina activă, a ferestrelor de text. O *fereastră text* este o zonă de afișare, formată din una sau mai multe linii și coloane consecutive. Ferestrele text se definesc și se selectează prin INT 10h. Definirea se realizează prin precizarea coordonatelor din spațiul text ale colțului stânga sus și respectiv ale colțului dreapta jos. În fereastră se poate realiza o operație de defilare în sus (jos) care deplasează textul rândurilor cu un număr de poziții în sus (jos) și umple rândurile eliberate, la partea de jos (sus), cu caracterul blank (ștergere). Se pot scrie rânduri noi de text în rândurile eliberate, eventual în toată fereastra activă. Rândurile de text care ies din fereastră, prin defilare, se pierd. La limită și în mod implicit, fereastra text este întreaga pagină activă.

În *modul grafic*, imaginea se constituie prin aprinderea mulțimilor de pixeli (desen prin puncte - *pixel mode*). Imaginea se creează în memoria ecran, în pagini, dacă placa grafică permite mai multe pagini și se afișează pe ecran, pagină cu pagină.

O pagină video utilizează un bit pentru a defini starea fiecărui pixel al ecranului (1=aprins, 0=stins) și un număr de biți suplimentari pentru a defini culoarea acestora. Deoarece culorile se definesc pe baza *registului de paletă*, dacă se utilizează concomitent 16 culori, sunt necesari 4 biți suplimentari. Rezultă că o pagină video trebuie să aibă cel puțin **4*rezoluție** biți. De exemplu, pentru o placă EGA, cu rezoluție înaltă, sunt necesari $4*640*350=896000$ biți ≈ 110 Kb și deci, într-o memorie ecran de 256 Kb, pot fi create două pagini video.

Modul grafic utilizează posibilitatea de adresare a pixelilor într-o pagină video și un *cursor grafic* invizibil. Cursorul grafic are coordonatele ultimului pixel adresat în pagina video selectată și poate fi deplasat în orice punct al acestui spațiu. Se mențin, de asemenea, facilitățile de interogare asupra poziției cursorului și asupra culorii pixelului curent.

Ca și la modul text, la definirea unui pixel există posibilitatea realizării operației SAU EXCLUSIV între biții de culoare ai unui pixel dintr-o pagină și valorile noi ale acestora. În acest fel, pixelii pot fi scriși sau șterși.

Toate aceste operații se realizează ca servicii ale rutinei INT 10h. În modul grafic, se selectează un anumit submod acceptat de placa grafică, submodurile diferind prin rezoluție, număr de culori și număr de pagini de ecran pe care le acceptă.

6.3 Facilități de scriere a textelor prin unitatea CRT

În Pascal, operațiile de afișare pe monitor, în modul text, au fost implementate în două unități de program: SYSTEM și CRT. Unitul SYSTEM definește operația de afișare pe monitor în mod standard, adică cu atribute de scriere implicite și începând din poziția curentă a cursorului. Unitul CRT (anexa 3) definește variabile, funcții și proceduri prin care programatorul poate defini și controla contextul de scriere (ferestre, culori, poziție de scriere etc.)

- *Alegerea modului text.* Programatorul are posibilitatea de a alege un anumit mod de scriere, caracterizat, în principal, prin rezoluția de afișare și tipul de monitor: monocrom sau color. Sunt acceptate modurile prezentate în anexa 3. Pentru alegerea modului se apelează procedura:

TextMode (Mode);

în care parametrul *Mode*, de tip *Word*, definește modul dorit, prin numărul asociat. Procedura salvează vechiul mod în variabila predefinită **LastMode**, de unde, prin aceeași procedură, poate fi restaurat. Procedura setează culoarea de fond zero (culoarea neagră). Dacă nu se apelează **TextMode**, atunci, implicit, se consideră modul de mare rezoluție al plăcii grafice.

Exemplu:

6.1.

```
. . .  
User CRT;  
. . .  
TextMode(C080);  
. . .  
TextMode(LastMode);  
. . .  
TextMode(BW40)
```

• *Stabilirea culorii de fond și de scriere* se face prin referire la paleta implicită. Intrările în paletă, desemnate prin constante întregi sau constante simbolice, sunt asociate cu diferitele culori (anexa 3). Definirea culorii fondului se face prin apelul procedurii:

TextBackGround(culoare);

în care parametrul *culoare* este de tipul Byte și poate avea valorile 0-7. Procedura tratează biții 4-6 ai unei variabile predefinite **TextAttr** de tip Byte, prin care se modifică corespunzător octetul de atribute din memoria video.

Culoarea de scriere se definește prin apelul procedurii:

TextColor (culoare);

în care parametrul *culoare* poate să aibă o valoare între 0 și 15 (anexa 2), eventual mărită cu 128. Procedura modifică biții 0-3 și 7 ai variabilei **TextAttr**, scriind în bitul 7, pentru definirea clipirii, 1 sau 0, după cum a fost adunată sau nu constanta 128 (constanta simbolică corespunzătoare este **Blink**).

Exemplu:

6.2.

```
. . .  
TextColor (Red+128);  
. . .  
TextColor (Green+Blink);  
. . .  
Culoare:=6  
. . .  
TextColor (Culoare);
```

Programatorul poate controla intensitatea culorii, prin modificarea bitului 3 din variabila **TextAttr** (șters de către **TextColor**) prin procedurile fără parametri:

HighVideo - intensitate mărită (bitul 3 are valoare 1);

LowVideo - intensitate mică (bitul 3 are valoare 0);

NormVideo - intensitate implicită.

Execuția procedurilor pentru stabilirea culorii de fond și de scriere afectează numai fereastra curentă. Atributele rămân valabile până la o nouă definire sau până la terminarea execuției programului.

• *Definirea unei ferestre de scriere.* Scrierea textului pe ecran se poate realiza într-o anumită zonă a acestuia, denumită fereastră curentă de scriere (text

window). O fereastră se definește ca un dreptunghi, prin coordonatele colțurilor stânga-sus, dreapta-jos. Ecranul întreg este o fereastră de coordonate (1, 1, 80, 25) sau (1, 1, 40, 25) etc., depinzând de modul text. Declararea ferestrei se face prin procedura:

Window(x1, y1, x2, y2);

în care parametrii (de tipul Byte) definesc, în ordine, cele două colțuri: (x1, y1) - colțul stânga-sus, (x2, y2) - colțul dreapta-jos (în exprimare matriceală, abscisele x1,x2 reprezintă numerele coloanelor, iar ordonatele y1, y2 sunt numerele de linii).

Exemplu:

6.3. Window (10, 5, 70, 20); definește o fereastră care se întinde între liniile 5-20 și coloanele 10-70.

Unitatea CRT definește variabilele **WindMin**, **WindMax**, de tipul Word, care conțin coordonatele colțurilor ferestrei curente. Octetul cel mai semnificativ conține numărul liniei (y), iar cel mai puțin semnificativ conține numărul coloanei (x). Coordonatele ferestrei curente pot fi scrise ca:

(Lo(WindMin), Hi(WindMin)) - colțul stânga-sus

(Lo(WindMax), Hi(WindMax)) - colțul dreapta-jos

Dacă nu se apelează procedura **Window**, fereastra implicită este întregul ecran, variabilele **WindMin** și **WindMax** fiind inițializate corespunzător modului text selectat sau implicit. Fereastra selectată rămâne activă până la un nou apel al procedurii **Window**, care poate defini o nouă fereastră curentă, eventual una utilizată anterior.

- *Poziționarea cursorului.* În fereastra curentă, programatorul poate să gestioneze poziția cursorului după necesitățile programului, impunând locul pe ecran începând cu care se va realiza scrierea următorului text. Poziția curentă a cursorului poate fi aflată prin apelul funcțiilor, fără parametri, cu rezultat de tipul Byte: **WhereX** (pentru abscisă) și **WhereY** (pentru ordonată). Mutarea cursorului într-o anumită poziție a ferestrei curente poate fi comandată prin apelul procedurii:

GotoXY(X, Y);

Exemplu:

6.4.

```
. . .
Window(20, 3, 100, 15); TextBackGround(Red); TextColor(Blue);
ClrScr;                      {sterge fereastra}
Writeln ('Text 1');          {Se scrie pe linia 1, coloana 1 a
ferestrei (coordonatele absolute: linia 3, coloana 20)}
GotoXY(5, 6); Write ('Text 2');
{Se scrie pe linia 6, coloana 5 a ferestrei (coordonatele
absolute: linia 9, coloana 25)}
X:=WhereX;
Y:=WhereY;
Write;
Writeln ('X=', X, 'Y=', Y); {Se scrie X=11, Y=6}
```

Dacă programatorul nu gestionează poziția cursorului, atunci acesta se deplasează, implicit, de sus în jos și de la stânga la dreapta, pe măsură ce se scrie

textul în fereastră. Când cursorul este la sfârșitul ultimului rând al ferestrei, se produce defilarea textului în fereastra curentă.

- *Ștergerea și inserarea în fereastra curentă se pot realiza utilizând următoarele proceduri fără parametri:*

ClrScr - șterge fereastra curentă și poziționează cursorul pe prima linie și coloană a acesteia;

ClrEol - șterge linia curentă, de la cursor până la sfârșitul ei, fără a muta cursorul;

DelLine - șterge rândul pe care se află cursorul și mută, în sus cu un rând, textul de sub linia ștersă; nu este afectată poziția cursorului;

InsLine - deplasează în jos, cu un rând, textul, începând cu cel pe care este cursorul și inserează o linie goală, în linia din fereastră pe care este cursorul. Se “pierde” ultimul rând din fereastră.

- *Citirea caracterelor fără ecou.* Prin intermediul a două funcții, fără parametri, programatorul poate citi fără ecou caracterul produs de ultima tastă apăsată sau poate verifica dacă a fost apăsată o tastă. Se utilizează funcțiile:

ReadKey - funcție de tipul CHAR care reîntoarce caracterul existent în buffer-ul tastaturii; dacă nu există un astfel de caracter, atunci așteaptă până la tastarea unui caracter;

KeyPressed - funcție de tipul BOOLEAN care reîntoarce valoarea True dacă a fost apăsată o tastă și False altminteri.

Funcția ReadKey întoarce codul caracterului rezultat prin apăsarea tastei, dacă aceasta produce un singur cod sau primul cod (valoare zero binar), dacă tasta apăsată produce două coduri. În acest ultim caz, se obține caracterul complet dacă se fac două apeluri succesive ale funcției Readkey.

În general, tastele **F1-F12**, tastele de deplasare cursor (săgeți), **Home**, **Ins**, **Del**, **PgUp**, **PgDwn** când sunt apăsate singure sau simultan cu **CTRL**, **Shift** ori **ALT**, produc două coduri. Pot, de asemenea, produce două coduri tastele de litere sau cifre, atunci când sunt apăsate simultan cu tasta **ALT**.

În legătură cu funcția KeyPressed, trebuie menționat că răspunsul este furnizat potrivit stării buffer-ului tastaturii de la acel moment (funcția nu așteaptă apăsarea pe o tastă). Dacă KeyPressed este TRUE, atunci ReadKey returnează imediat caracterul.

- *Pornirea și oprirea difuzorului.* Programul poate provoca producerea sunetelor de o anumită frecvență pe o durată controlabilă. În acest scop sunt prevăzute procedurile:

Sound(Hz) - pornirea difuzorului care emite un sunet continuu de frecvență dată de parametrul **H_z**, în Hertz;

NoSound - oprirea difuzorului;

Delay(Ms) - realizarea unei întârzieri de **M_s** milisecunde până la execuția următoarei instrucțiuni.

Controlând corespunzător frecvența **H_z** și intervalul de emisie **M_s** al difuzorului se poate realiza o anumită linie melodică.

Pentru cei care doresc să se inițieze în tehnica compunerii muzicii cu ajutorul calculatorului, sunt prezentate în continuare câteva elemente de bază.

În tabelul 6.1 sunt prezentate frecvențele notelor, în patru octave consecutive (în total sunt 7 octave numerotate 0÷6). Se poate constata că frecvența unei note într-o octavă se poate estima prin dublarea frecvenței acesteia din octava imediat inferioară. Similar, se poate deduce frecvența notelor dintr-o octavă inferioară prin înjumătățirea frecvențelor notelor din octava imediat superioară.

Tonul unei note poate fi alterat (mărit sau micșorat) cu o jumătate de ton în octava respectivă (note cu diez și bemol). Aceasta înseamnă că frecvența unei note oarecare **i** cu diez se calculează cu relația: $nota_i = 0.5(nota_{i+1} + nota_i)$; $nota_i = 1, 2, 4, 5, 6$.

Notele pot avea diferite durate: întreagă, doime, pătrime, optime etc. Dacă se stabilește durata notei întregi ca durată inițială, atunci duratele celorlalte note se calculează ca o fracție corespunzătoare din aceasta (1/2, 1/4, 1/8 etc.).

Tabelul 6.1 - Frecvențele notelor muzicale în diverse octave

Octava 1		Octava 2		Octava 3 (mijlocie)		Octava 4	
Nota	Frecvența	Nota	Frecvența	Nota	Frecvența	Nota	Frecvența
Do	130.81	Do	261.63	Do	523.25	Do	1046.5
Re	146.83	Re	293.66	Re	587.33	Re	1174.7
Mi	164.81	Mi	329.63	Mi	659.26	Mi	1318.5
Fa	174.61	Fa	349.23	Fa	698.46	Fa	1396.9
Sol	186.00	Sol	392.00	Sol	783.99	Sol	1568.0
La	220.00	La	440.00	La	880.00	La	1760.0
Si	246.94	Si	493.88	Si	987.77	Si	1976.5

Se știe că durata unitară este determinată de tempo-ul în care trebuie “cântate” notele. În tabelul 6.2, sunt redate câteva tempo-uri, exprimate în pătrimi/minut. Din tabel rezultă duratele diferitelor note, în minute, pentru un tempo, **T**, dat:

$$\begin{array}{ll} D_1=4/T; & D_8=1/2T; \\ D_2=2/T; & D_{16}=1/4T; \\ D_4=1/T; & D_{32}=1/8T \text{ etc.} \end{array}$$

Durata unei note poate fi modificată prin punct. Astfel, durata reală este 3/2 din durata notei fără punct. Pentru a obține stilul *legato* și *stacatto*, durata pe care se cântă o notă se modifică astfel:

normal: 7/8 din durată; 1/8 pauză;
legato: 1/1 din durată;
stacatto: 3/4 din durată; 1/4 pauză.

Trebuie observat că, datorită dependenței duratei notelor de ceasul intern, pentru unele tempo-uri nu se obțin rezultate satisfăcătoare. De aceea, este necesar ca durata notelor și tempo-urile să se stabilească prin experimentări.

Tabelul 6.2 Duratele notelor în diverse tempo-uri

Tempo	Denumire	Pătrimi/minut
foarte încet	Largo	40-60
	Larghetto	60-66
	Adagio	66-76
încet	Andante	76-108
mediu	Moderato	108-120
repede	Alegro	120-168
	Presto	168-208

6.4 Implementarea modului grafic în Pascal

Subprogramele care implementează modul grafic se bazează pe caracteristicile plăcilor existente și se împart în: primitive grafice și subprograme ajutătoare. Ele sunt declarate în unitatea Graph și sunt memorate în următoarele fișiere:

- GRAPH.TPU - care conține codul obiect al subprogramelor definite în unitatea Graph;
- *.BGI - câte un fișier pentru fiecare tip de placă grafică, care conține codul obiect al rutinelor de tratare a întreruperii pentru operații grafice (driver de placă), ca soft specific pentru comanda interfețelor grafice;

- *.CHR - fișiere care conțin seturi de caractere suplimentare cu diferite stiluri (fonturi).

Unitatea Graph definește, de asemenea, o serie de constante, variabile și structuri de date care să-i ușureze programatorului apelul acestor subprograme (anexa 4). Pentru utilizarea subprogramelor de grafică, programatorul trebuie să procedeze astfel:

- a) Să declare faptul că utilizează unitatea: **uses Graph**;
- b) Să selecteze modul (submodul) grafic dorit, prin utilizarea procedurii:

InitGraph (GraphDriver, GraphMode, PathToDriver)

Primii doi parametri sunt variabile de tipul INTEGER, iar ultimul este o constantă sau variabilă de tipul STRING. Parametrii au următoarea semnificație:

- *GraphDriver* precizează tipul plăcii grafice considerate. Variabila poate lua una din valorile întregi 0-10 sau constantele predefinite echivalente. Constanta *Detect* (valoare 0) cere subprogramului să identifice tipul de placă a calculatorului.
- *GraphMode* selectează unul din submodurile acceptate de placa grafică. Valoarea parametrului nu este luată în considerare dacă *GraphDriver* a fost *Detect*, caz în care se selectează automat submodul cu cea mai mare rezoluție.
- *PathToDriver* definește calea spre directorul de rezidență a driver-ului de placă (fișier de tip BGI), dacă acesta nu este în director curent. Șirul vid definește o căutare în directorul curent.

Exemple:

6.5.

```
GraphDriver:=EGA;  
GraphMode:=EGALo;  
InitGraph ('GraphDriver, GraphMode,');
```

6.6.

```
GraphDriver:=Detect;  
GraphMode:=0;  
InitGraph (GraphDriver, GraphMode, 'C:\TP\BIN');
```

c) Să verifice dacă inițializarea modului grafic dorit, din pasul anterior, s-a desfășurat cu succes. În acest sens, se utilizează funcția specială **GraphResult**, de tipul INTEGER și fără parametri, care returnează valoarea zero (GrOK) pentru succes. Funcția poate fi utilizată după execuția oricărei operații grafice.

d) Să apeleze subrutinele grafice necesare, dacă inițializarea s-a derulat cu succes.

e) Să revină în modul text, la terminarea operațiilor grafice, prin utilizarea procedurii fără parametri, **CloseGraph**. Inițializarea sistemului grafic alocă memoria dinamică necesară, iar închiderea o eliberează.

6.4.1 Fereastră și vizor

În grafica computerizată se consideră că înregistrarea pe ecran este o transpunere, micșorată sau mărită, a desenelor din spațiul real (spațiul utilizator). Partea imaginii reale care se desenează este cuprinsă într-un dreptunghi, numit *fereastră* (window), cu laturile paralele cu axele de coordonate ale acestui spațiu. Fereastra se proiectează pe spațiul ecran într-un dreptunghi, asemenea cu fereastra, denumit *vizor* (viewport). Astfel, pe ecran, pot coexista mai multe imagini independente. La limită, vizorul poate fi întregul ecran.

Relația între fereastră și vizor este evidențiată în figura 6.6. De aici se deduce o transformare de coordonate, care proiectează orice punct (x_r, y_r) din spațiul real, în punctul (x_e, y_e) al spațiului ecran. Dacă fereastra și vizorul se definesc prin coordonatele colțurilor stânga-sus și dreapta-jos, iar centrele celor două dreptunghiuri sunt notate (x_r^0, y_r^0) , (x_e^0, y_e^0) , din asemănarea imaginii din cele două spații se pot scrie relațiile:

$$\left. \begin{aligned} \frac{x_e - x_e^0}{x_r - x_r^0} &= \frac{v_2 - v_1}{w_2 - w_1} \\ \frac{y_e - y_e^0}{y_r^0 - y_r} &= \frac{v_4 - v_3}{w_3 - w_4} \end{aligned} \right\} \quad (1)$$

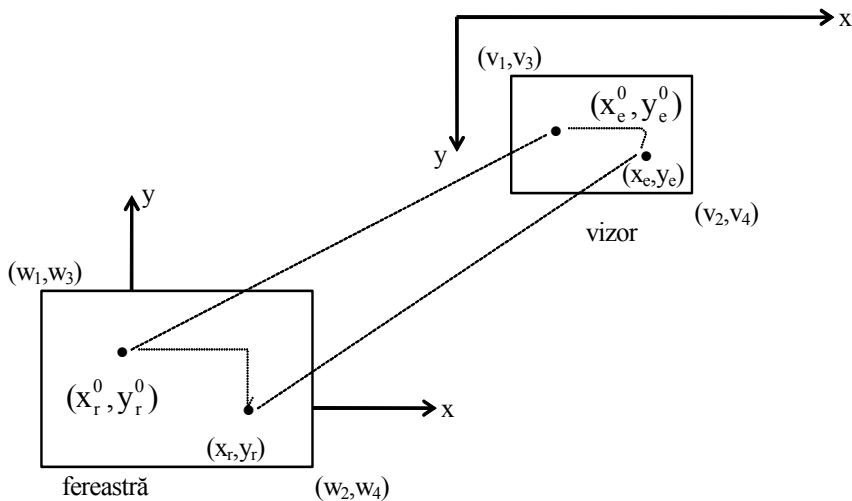


Fig. 6.6 Relația fereastră-vizor

Din (1) se explicitizează coordonatele (x_e, y_e) și se obțin relațiile:

$$\left. \begin{aligned} x_e &= x_e^0 + \frac{v_2 - v_1}{w_2 - w_1} x_r - \frac{v_2 - v_1}{w_2 - w_4} x_r^0 \\ y_e &= y_e^0 - \frac{v_4 - v_3}{w_3 - w_4} y_r + \frac{v_4 - v_3}{w_3 - w_4} y_r^0 \end{aligned} \right\} \quad (2)$$

din care, prin înlocuirea coordonatelor centrelor ferestrelor și vizorului, în funcție de coordonatele colțurilor acestora, se obțin relațiile:

$$\left. \begin{aligned} x_e &= \frac{v_1 + v_2}{2} + \frac{v_2 - v_1}{w_2 - w_1} x_r - \frac{v_2 - v_1}{w_2 - w_1} \cdot \frac{w_1 + w_2}{2} \\ y_e &= \frac{v_3 + v_4}{2} - \frac{v_4 - v_3}{w_3 - w_4} y_r + \frac{v_4 - v_3}{w_3 - w_4} \cdot \frac{w_3 + w_4}{2} \end{aligned} \right\} \quad (3)$$

Realizând calculul din (3) se obțin relațiile:

$$\left. \begin{aligned} x_e &= \frac{v_2 - v_1}{w_2 - w_1} x_r + \frac{v_1 w_2 - v_2 w_1}{w_2 - w_1} \\ y_e &= -\frac{v_4 - v_3}{w_3 - w_4} y_r + \frac{v_4 w_3 - v_3 w_4}{w_3 - w_4} \end{aligned} \right\} \quad (4)$$

Notând:

$$\left. \begin{aligned} a_1 &= \frac{v_2 - v_1}{w_2 - w_1} > \quad a_2 = -\frac{v_4 - v_3}{w_3 - w_4} \\ b_1 &= \frac{v_1 w_2 - v_2 w_1}{w_2 - w_1} > \quad b_2 = \frac{v_4 w_3 - v_3 w_4}{w_3 - w_4} \end{aligned} \right\} \quad (5)$$

se obțin relațiile de transformare:

$$\begin{aligned} x_e &= \text{Round}(a_1 x_r + b_1) \\ y_e &= \text{Round}(a_2 y_r + b_2) \end{aligned} \quad (6)$$

Relațiile (6) dau *coordodatele absolute*, în spațiul ecran, pentru orice punct (x_r, y_r) . Este, de multe ori, preferabil să se exprime punctele în spațiul ecran relativ la vizor, adică într-un subspațiu cu originea în colțul stânga-sus al acestuia. Dacă se notează (dx_e, dy_e) coordonatele absolute (x_e, y_e) , se obțin relațiile:

$$\left. \begin{aligned} x_e &= v_1 + dx_e \\ y_e &= v_3 + dy_e \end{aligned} \right\} \quad (7)$$

Utilizând relațiile (4), din (6) se obțin relațiile:

$$\left. \begin{aligned} dx_e &= \frac{v_2 - v_1}{w_2 - w_1} (x_r - w_1) \\ dy_e &= -\frac{v_4 - v_3}{w_3 - w_4} (y_r + w_3) \end{aligned} \right\} \quad (8)$$

Dacă se notează:

$$\begin{aligned} k_1 &= \frac{v_2 - v_1}{w_2 - w_1} \\ k_2 &= -\frac{v_4 - v_3}{w_3 - w_4} \end{aligned}$$

atunci se pot considera coordonatele relative sub forma:

$$\left. \begin{aligned} dx_e &= \text{round}(k_1(x_r - w_1)) \\ dy_e &= \text{round}(k_2(y_r - w_3)) \end{aligned} \right\} \quad (9)$$

Utilizarea ferestrei și/sau vizorului în tehnica realizării imaginilor grafice oferă, în plus, posibilitatea de verificare a apartenenței punctelor la desenul curent. Se pot implementa măsuri de eliminare a punctelor străine (tehnica clipping-ului sau tăierii), pentru a asigura protecția imaginilor vecine împotriva distrugerilor accidentale, desenându-se numai imaginea din fereastră.

În sistemul grafic Pascal, ideile tehnicii *fereastră-vizor* sunt implementate parțial, așa după cum rezultă din cele ce urmează.

- Există posibilitatea definirii vizorului ca spațiu curent pentru un anumit desen (vizor curent). Declararea lui se face prin apelul procedurii **SetViewport**, în forma:

SetViewport(v1,v3,v2,v4,Clip);

în care (v1,v3) și (v2,v4) sunt constante sau variabile de tipul INTEGER care dau coordonatele colțurilor vizorului, iar **Clip** este un parametru boolean care activează sau inhibă aplicarea clipping-ului.

Unitatea Graph definește funcțiile de tipul INTEGER fără parametri: **GetMaxX** și **GetMaxY** care returnează dimensiunile maxime ale spațiului ecran pentru placa grafică cu care este echipat calculatorul. Pentru a crea independență programului față de calculator, se recomandă utilizarea lor în definirea vizorului. De asemenea, unitatea Graph definește constantele simbolice **ClipOn** (True) și **ClipOff** (False) care pot fi folosite pentru parametrul **Clip**.

Exemplu:

6.7.

SetViewport(10,25,GetMaxX-150,GetMaxY-50,ClipOn);

Vizorul implicit, stabilit la inițializarea modului grafic, este întregul ecran, echivalent cu un apel de forma:

SetViewport (0,0,GetMaxX,GetMaxY,ClipOn) ;

- Vizorul curent poate fi șters, la culoarea de font curentă, prin apelul procedurii **ClearViewport**, care este fără parametri. Pentru a fi pus în evidență față de zonele vecine, vizorul poate fi șters cu o culoare de fond adecvată, stabilită în prealabil (vezi § 6.4.2.) și poate fi încadrat într-un dreptunghi.

Exemplu:

6.8. Pentru un ecran monocrom (dar nu numai) se poate utiliza o secvență de forma:

Rectangle(9,24,GetMaxX-149,GetMaxY-49);

SetViewport(10,25,GetMaxX-150,GetMaxY-50,ClipOn)

ClearViewport;

în care procedura *Rectangle* desenează un dreptunghi cu liniile în culoarea curentă de desen (alb), iar *SetViewport* îl șterge, la culoarea fondului.

Procedura **ClearDevice** șterge întregul ecran și determină revenirea la parametrii de lucru impliciti, stabiliți la inițializare.

- Declararea vizorului determină rutinele sistemului grafic să lucreze în coordonate relative. Aceasta înseamnă că orice procedură sau funcție care primește ca parametru de intrare coordonatele (dx_e, dy_e) 'înțelege' să le utilizeze pentru a calcula coordonatele absolute (x_e, y_e), prin relații de forma (6), iar cele care au aceste coordonate ca parametru de ieșire le determină prin relațiile:

$$\left. \begin{aligned} dx_e &= x_e - v_1 \\ dy_e &= y_e - v_3 \end{aligned} \right\}$$

- Sistemul nu prevede posibilitatea declarării ferestrelor în spațiul utilizator și, de aici, lipsa facilității de exprimare a coordonatelor, pentru rutinele de grafică, direct în acest spațiu și transformarea lor automată în coordonatele ecran. Programatorul trebuie să-și construiască proceduri proprii care să-i transforme coordonatele din spațiul utilizator în spațiul ecran, relative la vizorul curent. Deși pot fi aplicate procedee diverse, după natura desenului, pentru generalitate și uniformitate în realizarea programelor de grafică se aplică tehnica fereastră-vizor descrisă anterior.

6.4.2 Grafică în culori

Sistemul grafic Pascal implementează culorile pe principiul paletelor. Sistemul posedă o serie de proceduri, funcții și date prin care se pot accesa facilitățile de culoare ale plăcii grafice și modului grafic selectat.

- *Determinarea paletelor.* Deoarece caracteristicile de culoare depind de placa grafică și modul grafic selectat, se poate obține un program relativ portabil, dacă acesta se scrie astfel încât să-și determine singur facilitățile pe care le poate utiliza. Pentru aceasta pot fi utilizate rutinele:

- **GetMaxColor** - funcție fără parametri care întoarce o valoare de tipul WORD, reprezentând indexul maxim al paletelor;
- **GetPaletteSize** - funcție fără parametri care întoarce o valoare de tipul INTEGER reprezentând numărul maxim de culori care pot fi utilizate simultan;
- **GetDefaultPalette(palette)** - procedură care returnează, în parametrul *palette*, numărul și culorile paletelor implicate;
- **GetPalette(palette)** - procedură care returnează aceleași date ca procedura anterioară, dar pentru paleta curentă.

Parametrul *palette* trebuie să fie declarat ca o variabilă de tipul PaletteType, definit în unitatea Graph ca un articol cu câmpurile: *Size* (Byte) și *Colors* (Shortint).

- *Modificarea paletelor curente.* Programatorul poate modifica una sau toate intrările paletelor curente, dar pentru aceasta trebuie să cunoască codurile de culoare. Procedurile care se utilizează sunt:

- **SetPalette(Index,Color)** - care permite modificarea intrării date de parametrul *Index* (WORD), cu o culoare de cod *Color* (INTEGER). Atât pentru *Index*, cât și pentru *Color* pot fi utilizate constantele simbolice predefinite în unitatea Graph.
- **SetAllPalette(Palette)** - înlocuiește paleta curentă cu cea dată de parametrul *Palette*. Parametrul este de tipul PaletteType și trebuie să fi fost încărcat corect cu codurile de culori.

În ambele situații, procedurile modifică automat culorile desenului de pe ecran, în concordanță cu noua configurație a paletelor curente. Procedurile pot fi utilizate și cu scopul de a produce efecte de culoare deosebite sau pentru a releva treptat, prin modificări repetate, un desen 'ascuns'.

- *Determinarea și modificarea culorilor de fond și de desen.* Funcțiile și procedurile din această categorie sunt cel mai frecvent utilizate deoarece definesc fondul și culoarea de desen pe baza paletelor curente.

- **SetBkColor(Color)** - procedură pentru selecția culorii de fond. *Color* este parametrul de tip WORD care precizează indexul culorii dorite. Implicit, se utilizează indexul zero care corespunde, în general, culorii negre;
- **GetBkColor** - funcție fără parametri, de tipul WORD, care returnează indexul culorii curente pentru fond;

- **SetColor(Color)** - procedură pentru selecția culorii de desen. *Color* are aceeași semnificație ca la procedura anterioară. Implicit, se utilizează ultimul *Index* care, în general, corespunde culorii alb;
- **GetColor** - funcție similară cu *GetBkColor* pentru a obține culoarea curentă de desen.

Modificarea culorii de fond atrage după sine modificarea culorii pe ecran. Culorile selectate, denumite culori curente, rămân active până la o nouă selecție sau până la ieșirea din program.

Exemplu:

6.9. Presupunând o placă EGA, secvența care urmează stabilește un vizor cu fond albastru în care desenează un dreptunghi umplut. După 2000 ms se refac culorile anterioare.

```
Uses Graph,Crt;  
VAR  
    ColorBk,ColorFg:Word;  
    .....  
    ColorBk:=GetBkColor;  
    ColorFg:=GetColor;  
    SetViewport(20,50,250,150,ClipOn);  
    SetColor(EGARed);  
    SetBkColor(EGABlue);  
    ClearViewport;  
    Bar(20,10,60,50);  
    Delay(2000);  
    SetBkColor(ColorBk);  
    SetColor(ColorFg);  
    .....  
    .....
```

6.4.3 Desen prin puncte și vectori

Desenul prin puncte este modul 'natural' de realizare a imaginii video. Biblioteca de subprograme grafice Pascal conține pentru acest mod de lucru procedura **PutPixel(x,y,Color)** și funcția de tipul WORD **GetPixel(x, y)**.

Procedura *PutPixel* stabilește culoarea *Color* de aprindere a pixelului de coordonate (x,y) din vizorul curent, iar funcția *GetPixel* întoarce indexul culorii cu care este aprins un astfel de pixel. Pe baza subprogramelor, se pot construi desene complexe, dar cu dificultățile pe care le implică gestionarea mulțimii de pixeli care compun imaginea video. Pentru a facilita realizarea desenelor, sistemul grafic PASCAL implementează și alte moduri de lucru, cum este *desenul prin vectori*. În acest mod, pot fi trasate segmente de dreaptă, independente sau legate (linii frânte), utilizând pentru forma liniilor stilurile și culorile curente (setate). Culoarea de desen și culoarea fondului sunt curente, setate prin *SetBkColor* și *SetColor*. Stilul de linie este cel definit de procedura **SetLineStyle**, care se apelează astfel:

SetLineStyle(TipStil,Model,Grosime);

Stilul definit rămâne valabil până la o nouă setare. Stilul implicit este linia continuă normală. Parametrii, de tipul WORD, constante sau variabile, definesc atributele liniei (stilul liniei):

- *TipStil* - definește tipul de linie din punctul de vedere al continuității. Parametrul poate avea valorile 0-4 sau constanta simbolică corespunzătoare;
- *Grosime* - precizează grosimea, în număr de pixeli și poate avea valorile: 1 (linie normală - *NormWidth*) și 2 (linie groasă - *ThickWidth*);
- *Model* - definește un stil de utilizator printr-un fragment de 16 pixeli. Un pixel este aprins, dacă bitul corespunzător, din numărul dat de acest parametru, este unu. De exemplu, dacă *Model*=\$F053 (în binar 1111 0000 0101 0011) atunci se definește modelul ●●●● ●●●● ●●●● ●●●●, în care ● este pixel aprins.

Desenarea unui segment de dreaptă se face prin procedura **Line**(x1,y1,x2,y2). Prin parametri x1, y1, x2, y2, de tip INTEGER, se precizează coordonatele capetelor segmentului de dreaptă. Coordonatele sunt relative la vizorul curent și se pot exprima ca variabile sau constante. Se desemnează numai porțiunea de segment care se află în interiorul vizorului.

Exemplu:

6.10. Secvența care urmează:

```
SetViewport(20,20,150,100,ClipOn);  
Line(10,10,150,150);
```

desenează un segment cu originea absolută în (30,30) și extremitatea finală în (170,120) din care se vede numai porțiunea de coordonate absolute (30,30), (138,100) - figura 6.7.

Desenarea segmentelor de dreaptă cu procedura **Line** este incomodă, când trebuie desenate linii frânte, datorită redundanței de informații, având în vedere coincidența punctului final al unui segment cu originea segmentului următor.

Desenarea liniilor frânte este facilitată de introducerea în sistemul grafic a noțiunii de punct curent și de existența unor rutine care desenează linii relativ la acesta. *Punctul curent*, definit prin coordonatele sale în spațiul ecran, este înțeles de rutinele sistemului ca origine a următorului segment de dreaptă posibil de desenat. Sistemul prevede rutine pentru aflarea și modificarea poziției punctului curent, precum și reguli de comportare a rutinelor de trasare de linii, cu privire la actualizarea acestuia. Locul punctului curent poate fi aflat cu ajutorul funcțiilor, fără parametri, **GetX** și **GetY**, care returnează abscisa și ordonata absolută, ca numere întregi.

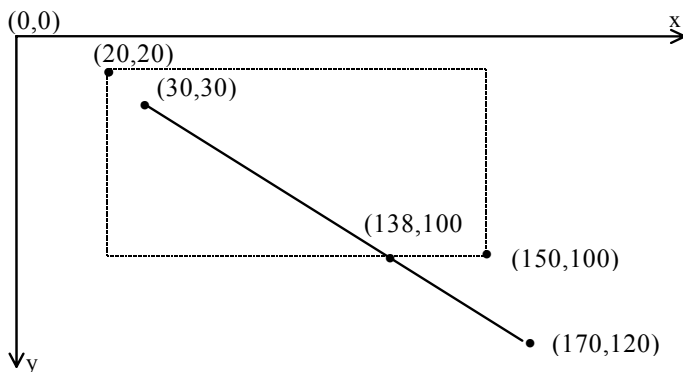


Fig. 6.7. Desenarea liniilor relativ la vizor

Punctul curent poate fi mutat într-un alt loc din spațiul fizic, prin procedurile **MoveTo** și **MoveRel** care se apelează sub forma: **MoveTo(x,y)**; **MoveRel(x,y)**. Procedura *MoveTo* consideră valorile (x,y) drept coordonate relative la vizorul curent și calculează coordonatele absolute (x_e, y_e) ale punctului curent prin relațiile: $x_e = v_1 + x$; $y_e = v_3 + y$, unde (v_1, v_3) sunt coordonatele originii vizorului. Procedura *MoveRel* deplasează punctul curent relativ la poziția sa anterioară, adică $x_e = \text{GetX} + x$; $y_e = \text{GetY} + y$, în care x,y apar ca deplasări. La intrarea în regim grafic, la definirea și/sau ștergerea vizoarelor, precum și la schimbarea modului grafic, punctul grafic curent este nedefinit. Este sarcina programului să inițializeze punctul curent prin procedura *MoveTo*. Toate procedurile de desenare de linii și scriere de text deplasează poziția punctului curent în ultimul punct utilizat.

Desenarea liniilor care își au originea în punctul curent se realizează cu ajutorul procedurilor **LineTo(x,y)**; **LineRel(x,y)**. Pentru *LineTo*, coordonatele extremității liniei sunt relative la vizorul curent și deci $x_e = v_1 + x$, $y_e = v_3 + y$, unde (x_e, y_e) sunt coordonatele absolute ale extremității liniei relativ la punctul curent $x_e = \text{GetX} + x$; $y_e = \text{GetY} + y$. După trasare, punctul curent este deplasat în punctul $x_e = x_e$, $y_e = y_e$. Dacă la trasare a fost necesară operația de clipping, punctul final al segmentului vizibil este recalculat ca (x_{\sim_e}, y_{\sim_e}) , însă punctul curent este mutat tot în (x_e, y_e). Se evită astfel deformarea desenelor.

Exemplu:

6.11. Dacă se consideră linia frântă P_1, P_2, P_3, P_4 din figura 6.8, atunci secvențele de instrucțiuni de trasare, având în vedere coordonate relative la vizor și relative la punctul curent, pot fi scrise astfel:

Relativ la vizor	Relativ la punctul curent
MoveTo(50,60)	MoveTo(50,60)
LineTo(70,40)	LineRel(20,-20)
LineTo(85,45)	LineRel(15,5)
LineTo(140,30)	LineRel(25,-15)

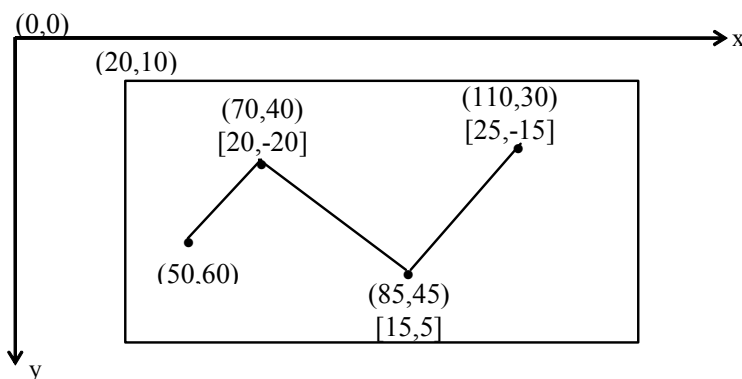


Fig. 6.8. Desenarea segmentelor relativ la vizor și la punctul curent

În plus, trebuie menționat faptul că desenarea unei linii simple sau frânte poate fi făcută astfel încât să distrugă sau să conserve, într-un anumit sens, culorile pixelilor care se suprapun. Comportamentul pixelilor depinde de modul de scriere a noii imagini în raport cu imaginea existentă pe ecran (setarea modului Write);

6.4.4 Raportul de aspect

În realizarea desenelor prin segmente apare un fenomen de deformare care poate să fie deranjant (un pătrat apare ca dreptunghi, un cerc ca o elipsă etc). Fenomenul se datorează formei pixelilor care, cu excepția unor plăci de foarte mare rezoluție, nu sunt de formă pătratică, ci dreptunghiulară. Aceasta face să se aplice, implicit, unități diferite de măsură pe cele două axe. Deformarea poate fi atenuată sau chiar eliminată, dacă în realizarea desenului se aplică un anumit raport între lungimile segmentelor de pe orizontală și ale celor de pe verticală.

Dacă se notează cu **Lpx** lungimea unui pixel, în sensul axei **ox** și cu **Ipy** înălțimea sa, în sensul axei **oy**, atunci raportul $Ra = Ipy / Lpx$ este denumit *raport de aspect*. El poate fi utilizat pentru a determina, în mod adecvat, lungimile segmentelor (în pixeli) care se desenează. Pentru a obține dimensiunile pixelului plăcii grafice se utilizează procedura:

GetAspectRatio(Lpx, Ipy);

în care cei doi parametri sunt variabile de tipul WORD.

Dacă se notează cu AB un segment vertical de lungime **n** pixeli și cu CD un segment orizontal, de lungime **m** pixeli, atunci, pentru ca raportul AB/CD să fie **k**, trebuie ca **m**, dacă se cunoaște **n** (respectiv **n**, dacă se cunoaște **m**), să se determine cu relațiile:

$$\left. \begin{aligned} m &= \text{Round}(Ra * n / k) \\ n &= \text{Round}(m * k / Ra) \end{aligned} \right\}$$

Exemplu:

6.12. Desenarea unui pătrat, cu latura orizontală CD de $m=50$ pixeli și vârful stânga sus în (30,60), pe un vizor egal cu întregul ecran. Deoarece $k=1$, se poate utiliza secvența:

```
GetAspectRatio(Lpx, Ipy);  
Ra:=Ipy/Lpx;  
n=Round(50/Ra);  
MoveTo(30,60);  
LineTo(80,60);  
LineTo(80,60+n);  
LineTo(30,60+n);  
LineTo(30,60);
```

Există, de asemenea, rutina *SetAspectRatio* care permite definirea software a raportului de aspect pentru a fi utilizat de rutinele sistemului grafic. Programatorul care utilizează rutinele sistemului pentru desenare de figuri poate să reducă fenomenul de deformare, modificând repetat, eventual interactiv, raportul de aspect. Apelul acestei proceduri este similar procedurii *GetAspectRatio*.

6.4.5 Scrierea textelor în modul grafic

În mod uzual, imaginile de pe ecran sunt însoțite de texte explicative. Având în vedere că, de multe ori, lucrul în modul text nu satisface cerințele de mărime, formă, culoare și direcție de scriere, în unit-ul Graph au fost introduse primitive specifice prelucrării textelor. Unitatea conține două grupe de rutine: pentru definirea atributelor textelor și pentru scrierea propriu-zisă.

- *Rutinele de declarare a atributelor textelor* trebuie apelate ori de câte ori este necesar să se definească alte caracteristici de scriere decât cele curente. Atributele astfel definite (setate) rămân active până la o nouă setare sau până la sfârșitul programului. Rutinele de declarare sunt proiectate pentru a putea trata următoarele attribute de scriere:

- a) Stilul de caractere, adică forma caracterelor imprimabile. Sistemul recunoaște cinci stiluri (fonturi), codificate 0-4, cărora li s-au asociat constante simbolice;

- b) Culoarea: caracterele se afișează pe fondul și culoarea de desen curentă;

- c) Mărimea se referă la lățimea și înălțimea caracterelor, caracteristici care se definesc pentru toate fonturile, cu excepția stilului *DefaultFont*, la care mărimea nu poate fi modificată. Fonturile cu mărime de caracter modificabilă definesc caracterele prin vectori (caractere vectoriale). Mărimea se poate referi la spațiul dreptunghiular pentru un caracter, în sensul micșorării sau creșterii proporționale a acestuia pentru ambele dimensiuni sau pentru fiecare dimensiune în parte (dimensiuni de utilizator).

d) Direcția de scriere poate fi: orizontală sau verticală. Pentru direcția orizontală (constanta simbolică *HorizDir*) scrierea se face de la stânga spre dreapta, iar pentru direcția verticală (*VertDir*), de jos în sus;

e) Alinierea: precizează, pentru fiecare din cele două direcții, modul de amplasare a textului în raport cu punctul de scriere (punct de referință).

Pentru definirea atributelor se pot utiliza procedurile:

SetTextStyle (Font, Directie, Marime);

SetTextJustify (FtHorizDir, FtVertDir);

SetUserCharSize (MultX, DivX, MultY, DivY);

Parametrul de mărime, la procedura **SetTextStyle**, poate lua valori naturale mai mari sau egale cu zero și precizează de câte ori să fie mărite, proporțional, caracterele fontului ales față de dimensiunea normală a acestora. Pentru definirea dimensiunilor de utilizator (independente pe cele două direcții), valoare *zero* a parametrului *Marime* declară intenția programatorului de a apela procedura *SetUserCharSize*. La această procedură, se utilizează ideea că dimensiunea se modifică prin înmulțire cu un raport. Raportul *MultX/DivX* va fi utilizat pentru a modifica lățimea caracterelor, iar *MultY/DivY* pentru modificarea înălțimii. Atunci când raportul este unu, dimensiunea pe direcția respectivă va rămâne nemodificată.

La procedura de definire a alinierii se cere să se precizeze modul de aliniere a textului pe ambele direcții, chiar dacă scrierea utilizează numai una din ele.

Exemple:

6.13. Se definește fontul *SmallFont*, cu mărime proporțională, de 3 ori și cu afișare centrată pe orizontală:

```
SetTextJustify(CenterText,CenterText);  
SetTextStyle(SmallFont,HorizDir,3);
```

6.14. Se stabilește o creștere a dimensiunii caracterelor fontului *SansSerifFont* numai în înălțime, de 1,5 ori; scrierea se face pe direcția verticală, cu aliniere la partea de jos:

```
SetTextStyle(SansSerifFont,VerDir,0);  
SetUserCharSize(1,1,3,2);  
SetTextJustify(LeftText,BottomText);
```

Dacă atributele de scriere au fost deja definite, adică au devenit atribute curente, programatorul poate utiliza funcțiile întregi (de tipul *WORD*): **TextHeight(Text)**, **TextWidth(Text)**, pentru a afla înălțimea, respectiv lățimea, în număr de pixeli, necesare pentru a afișa întregul text dat de parametrul *Text*. Pe baza acestor informații, se poate alege punctul de referință astfel încât scrierea textului să fie posibilă (totul să înceapă în vizorul de scriere) sau acesta să fie mai bine amplasat, pe direcția curentă.

- Pentru *scrierea efectivă a textelor* se utilizează una din următoarele două proceduri:

OutText(Text);
OutTextXY(x,y,Text);

Textul de scris (parametrul *Text*) se poate prezenta ca o constantă, variabilă sau expresie de tipul STRING. Scrierea se face în raport cu punctul de referință cu coordonate (x,y) sau "la cursor", dacă nu se definește explicit un astfel de punct. După scriere, cursorul se găsește pe poziția următoare celei utilizate pentru scrierea ultimului caracter al textului, dacă scrierea s-a făcut prin procedura *OutText* sau rămâne nemodificat, atunci când s-a utilizat procedura *OutTextXY*.

Dacă există definit vizor curent, cu acceptarea clipping-ului, textul este decupat la limitele vizorului. Nu se aplică procedeul de clipping dacă fontul este *DefaultFont* și comportarea rutinei de scriere este nedefinită când survine o depășire a vizorului. În plus, trebuie menționat că primitivele de afișare țin cont de setarea modului de scriere.

Exemplu:

6.15. Se scrie orizontal textul "TURBO PASCAL", de 4 ori, cu cele patru fonturi vectoriale. Fiecare rând, centrat pe linia verticală a centrului ecranului, se scrie cu o altă culoare.

```
Clear Device;
SetTextJustify(CenterText,CenterText);
y:=20;
For i:=1 to 4 do
  Begin
    SetColor (i);
    SetTextStyle(i, HorizDir, 2+i);
    Y:=y+TextHeight('TURBO PASCAL')+8;
    OutTextXY(GetMaxX Div 2,Y, 'TURBO PASCAL');
  End;
```

6.4.6 Primitive pentru figuri

Pentru a facilita realizarea imaginilor, sistemul grafic cuprinde câteva primitive pentru desenarea unor figuri, majoritatea în plan. Unele din aceste proceduri umplu figura desenată cu un anumit model de culoare și hașură. Toate rutinele au fost concepute să lucreze în condițiile existenței unei vizor curent, adică în coordonate relative la vizor și să aplice clipping-ul, dacă acesta a fost acceptat. Câteva din ele țin seama de modul de scriere selectat. Ca și rutinele prezentate anterior, primitivele pentru figuri utilizează, pentru contur și linii, culorile și atributele curente. Dacă o primitivă umple figura, atunci ea folosește un model de hașurare și o anumită culoare. Modelul de hașurare poate fi ales dintr-o mulțime predefinită sau poate fi definit de utilizator.

Pentru a preciza stilul și culoarea de umplere se apelează procedura:

SetFillStyle(TipStil,Culoare);

în care parametrul *Culoare* este un index în paleta curentă, iar *TipStil* o constantă între 0-12. Dacă *TipStil* este 12 (UserFill), atunci el trebuie să fie definit ca un model binar pe 8 byte (un vector) care este interpretat de rutinele respective ca o matrice de 8x8 biți. Matricea este traversată circular pe linii și pixelul curent este aprins, la culoarea definită prin parametrul culoare, dacă bitul din model este unu. În plus, tipul definit rămâne ca model curent.

Exemplu:

6.16.

```
CONST
MyPattern:Array[1..8]of Byte=($01,$01,$FF,$01,$01,$FF,$01,$01);
SetFillStyle(MyPattern,3);
```

În figura 6.9 se prezintă toate tipurile de hașuri considerând numai culorile alb și negru.

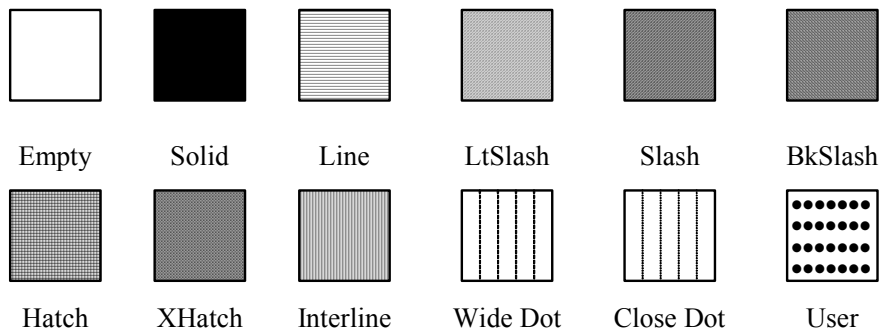


Fig. 6.9 Tipuri de hașuri

Principalele primitive pentru figuri sunt prezentate în continuare.

a) *Proceduri pentru dreptunghiuri:*

Rectangle (x1,y1,x2,y2);

Bar(x1,y1,x2,y2);

Prima procedură desenează un dreptunghi simplu, neumplut, iar a doua un dreptunghi umplut. Procedura *Rectangle* ține seama de setarea modului de scriere (WriteMode). Parametrii de apel sunt coordonatele întregi ale colțului stânga-sus (x1,y1) și dreapta jos (x2,y2) ale dreptunghiului.

b) *Proceduri pentru linii poligonale și poligoane oarecare:*

DrawPoly(NumarVarfuri,CoordonateVarfuri);

FillPoly(NumarVarfuri,CoordonateVarfuri);

Procedurile sunt similare perechii de proceduri pentru desenarea dreptunghiurilor. Coordonatele punctelor liniei poligonale trebuie să fie declarate ca o variabilă de tip masiv, la care tipul de bază trebuie să fie *PointType*, adică un articol cu două câmpuri: X și Y. Dacă se dorește trasarea unui poligon, atunci față de linia poligonală cu **n** vârfuri trebuie declarate **n+1** vârfuri și coordonatele ultimului vârf trebuie să coincidă cu cele ale primului. Procedura *FillPoly* se aplică numai pentru poligoane, iar interiorul acestora este umplut cu hașura și culoarea cerute.

Exemplu:

6.17.

```
CONST
    Pentagon: Array[1..6]of PointType=
        ((x:20;y:45),(x:50;y:25),(x:100;y:10),
         (x:120;y:40),(x:80;y:60),(x:20;y:45));
    FillPoly (6,Pentagon);
```

c) *Proceduri pentru arce, cercuri și elipse.* Procedurile din această categorie desenează marginile figurilor utilizând culoarea curentă, stabilită de *SetColor*. Cele care hașurează sau umplu figura desenată utilizează atributele date de *SetFillStyle*. Procedurile nu țin seama de setarea modului de scriere.

Procedurile care desenează arce utilizează, ca parametri, unghiuri în grade. Trebuie avut în vedere că acestea sunt considerate în sens trigonometric (sens invers acelor de ceasornic, cu 0 grade la ora 3, 90 grade la ora 12, 180 grade la ora 9 etc.). De asemenea, raza cercului și razele elipselor (raza orizontală - *RazaX* și raza verticală - *RazaY*) se dau în număr de pixeli. Coordonatele centrului figurii constituie, de asemenea, un parametru (x,y). Pentru asigurarea circularității pe ecran, procedurile utilizează raportul de aspect al plăcii, eventual setat soft, prin *SetAspectRatio*.

Procedurile de bază sunt:

(x,y,UnghiStart,UnghiFinal,Raza);

Arc

(x,y,UnghiStart,UnghiFinal,Raza);

PieSlice

((x,y,UnghiStart,UnghiFinal,RazaX,RazaY);

Circle (x,y,Raza);

Ellipse

(x,y,UnghiStart,UnghiFinal,Raza);

Sector

FillEllipse (x,y,RazaX,RazaY);

Procedura *Circle* desenează un cerc (neumplut), iar procedura *Arc* desenează un arc care, la limită, poate fi un cerc, depinzând de alegerea unghiurilor. Procedura *PieSlice* desenează un sector umplut care poate fi, la limită, un cerc umplut.

Aproape similar este cazul elipselor. Procedura *Ellipse* desenează un arc de elipsă sau o elipsă (neumplută), procedura *Sector* desenează un sector de elipsă sau o elipsă umplută, iar procedura *FillEllipse* desenează o elipsă umplută.

Exemplu:

6.18.

```
Arc (150,100,0,270,50);  
PieSlice (GetMaxX Div 2,GetMaxY Div 2,0,360,GetMaxX Div 4);  
FillEllipse (150,150,300,75).
```

Trebuie remarcat și faptul că sistemul grafic posedă o procedură de interogare asupra centrului punctului de start și punctului final care au fost utilizate în timpul execuției uneia din procedurile: *Arc*, *PieSlice*, *Ellipse*, *Sector*. Aceste informații sunt furnizate prin apelul procedurii:

GetArcCoords(ArcCoords);

în care parametrul *ArcCoords* are tipul *ArcCoordsType* definit public în unitatea *Graph* astfel:

Type

ArcCoords=Record

x,y:Integer;	{Punct de centru}
xStart,yStart:Integer;	{Punct de inceput}
xEnd,yEnd:Integer;	{Punct de sfarsit}

End;

Procedura este utilă pentru racordarea arcelor cu segmente de dreaptă sau alte arce.

d) *Procedura pentru paralelipiped dreptunghic*

Bar3D (x1,y1,x2,y2,Grosime,Top);

Procedura permite desenarea paralelipipedelor dreptunghice colorate (umplute). Parametrii au următoarele semnificații:

- $(x1,y1),(x2,y2)$ sunt coordonatele colțurilor stânga-sus, dreapta-jos ale feței (dreptunghiului din față);
- *Grosime*, de tipul *WORD*, indică grosimea paralelipipedului;
- *Top*, de tip *BOOLEAN*, arată că fața superioară trebuie să fie vizibilă (valoare *True* sau constanta *TopOn*) sau să fie ascunsă (valoare *False* sau *TopOff*). Parametrul permite realizarea paralelipipedelor suprapuse, caz în care trebuie să fie vizibilă numai fața superioară a ultimului paralelipiped.

Procedura se poate utiliza pentru desenarea histogramelor prin bare (grafice din bare) care prezintă, comparativ, valorile unei (unor) caracteristici (mărimi).

6.4.7 Elemente de animație

Animația poate fi considerată ca tehnică de realizare a mișcării obiectelor grafice pe ecran. Ea presupune refacerea imaginii video, de peste 30 de ori pe secundă, cu obiectele în diferite poziții, astfel încât să se asigure persistența imaginii pe retina ochiului. Reușita în realizarea animației este strâns condiționată de resursele calculatorului, cum sunt: viteza microprocesorului, existența coprocesorului matematic, numărul de pagini ale memoriei video etc.

1. Aspecte generale. Animația este o tehnică complexă prin multitudinea de probleme pe care le pune realizatorului de programe.

- În primul rând, programul trebuie să ia în considerare comportarea obiectului de animat în “mediu”. În cazurile simple, imaginea pe ecran se reduce la obiectul în mișcare, adică nu există mediu ambiant. Uzual, însă, pe ecran există o imagine “peste” care trebuie să se deplaseze obiectul. Această imagine este, pentru obiectul animat, un mediu ambiant, pe care acesta trebuie să îl conserve. Rezultă de aici că imaginea obiectului, în diferitele sale poziții, trebuie scrisă peste imaginea mediu într-un anumit mod. De regulă, regimul grafic al unui calculator acceptă cel puțin două moduri de scriere (*WriteMode*): unul care distruge imaginea veche (modul PUT) și altul care suprapune noua imagine peste cea veche (modul XOR). Trebuie remarcat că realizarea a două operații logice XOR (sau exclusiv) succesive ale unei imagini peste același mediu înseamnă, de fapt, ștergerea noii imagini din poziția respectivă.

- În al doilea rând, programul crește în complexitate dacă obiectul animat nu rămâne identic cu el însuși în toate pozițiile sale de mișcare. Este mai simplu de animat un obiect pentru care imaginea formei sale se conservă, deoarece această imagine poate fi realizată o singură dată, memorată și, apoi, afișată în diferite poziții. Dacă obiectul animat își schimbă forma de la o poziție la alta, așa cum este cazul general, atunci imaginea sa trebuie refăcută continuu. În acest caz, consumul de timp este mult mai mare și animația reușește numai în prezența unor resurse suplimentare ale calculatorului care să asigure viteza necesară de afișare.

- În al treilea rând, obiectul în mișcare poate să aibă un anumit comportament la atingerea limitelor vizorului de afișare. Problema este simplă atunci când obiectul poate ieși și intra în vizor, deoarece se rezolvă prin tehnica clipping-ului. Dacă obiectul nu poate părăsi vizorul sau atingerea limitelor acestuia îi imprimă o anumită traiectorie, atunci programul trebuie să prevadă teste pentru a sesiza situația și să includă relații adecvate de calcul.

- În sfârșit, probleme complexe ridică situațiile în care mediul însuși poate modifica forma și/sau traiectoria obiectului sau când trebuie animate mai multe obiecte concomitent. Din punct de vedere algoritmic, orice program de animație trebuie să aibă încorporați următorii pași:

- a) Se afișează pe ecran imaginea, cu obiectul în poziția **a**;
- b) Se construiește imaginea cu obiectul într-o poziție nouă, **b** (dacă este cazul);
- c) Se șterge obiectul din poziția **a** (eventual, prin afișarea a doua oară a imaginii acestuia în **a**, cu scriere XOR);

d) Se atribuie lui **a** valoarea lui **b** și se reia, dacă este cazul, de la punctul a.

2. Facilități pentru realizarea animației în PASCAL. Sistemul grafic Turbo Pascal prevede câteva rutine care facilitează realizarea animației.

- *Moduri de scriere.* Pot fi definite moduri de determinare a culorii unui pixel din noua imagine, în funcție de culoarea curentă de scriere și culoarea pe care același pixel o are în imaginea existentă pe ecran (culoarea veche). Având în vedere că noua imagine se constituie în memoria video, peste cea existentă, sunt posibile diferite operații logice între biții care determină culoarea unui pixel și noile valori dorite. Aceste operații sunt denumite moduri de scriere și se definesc prin procedura:

SetWriteMode(Mode);

unde parametrul *Mode* poate lua valorile întregi între 0-4. În mod implicit se aplică *NormalPut*, când pixelul capătă culoarea curentă, indiferent de culoarea pe care o avea (imaginea veche dispare). Cel mai frecvent se utilizează XOR, care determină culorile după o funcție "sau exclusiv". Se are în vedere că specificarea culorilor se face prin index în paletă și, deci, rezultatul este tot o astfel de valoare.

Din exemplul următor se poate urmări și afirmația că $(A \text{ XOR } B) \text{ XOR } B = A$.

Culoare veche (A)	Culoare curentă (B)	Culoare nouă (A XOR B)	(A XOR B) XOR B
0011 (3)	0101 (5)	0110 (6)	0011 (3)
1011 (11)	0010 (2)	1001 (9)	1011 (11)
1111 (15)	1101 (13)	0010 (2)	1111 (15)
0000 (0)	1111 (15)	1111 (15)	0000 (0)

Așa după cum s-a mai remarcat, nu toate procedurile grafice țin seama de setarea modului de scriere. Rutinele care determină culoarea nouă după regulile descrise sunt:

Line LineTo LineRel OutTextXY
DrawPoly Rectangle OutText

- *Salvarea și restaurarea unei imagini.* Sistemul prevede salvarea, din memoria ecran, a unei imagini încadrată într-o zonă dreptunghiulară, într-un buffer definit de utilizator în memoria internă, de regulă în yona *heap*. Se utilizează procedura:

GetImage(X1,Y1,X2,Y2,Buffer);

în care (X1,Y1), (X2,Y2) dau coordonatele colțurilor stânga-sus și dreapta-jos ale dreptunghiului care definește imaginea de salvat. Variabila *Buffer* trebuie să aibă un număr acoperitor de cuvinte pentru imagine, plus două cuvinte, (la începutul acesteia) în care procedura înscrie lățimea și lungimea dreptunghiului de imagine. Mărimea variabilei *Buffer* este returnată de funcția **ImageSize(X1,Y1,X2,Y2)** de tipul WORD și nu poate depăși 64KB. Imaginea salvată în buffer poate să fie afișată din nou utilizând procedura:

PutImage (X1,Y1,Buffer,WriteMode);

În acest apel, (X1,Y1) sunt coordonatele punctului în care se va suprapune colțul stânga-sus al imaginii, iar *WriteMode* definește modul de scriere a imaginii și are una din valorile definite pentru procedura *SetWriteMode*.

Exemplu:

6.19. Se desenează un cerc umplut, cu centrul în (50,50) și raza de 40 pixeli. Imaginea se salvează și apoi este afișată (restaurată) în punctul (200,50), fiind ștearsă din vechea sa poziție. Imaginea nouă conservă imaginea existentă pe ecran.

```
. . . . .
Uses CRT,Graph
. . . . .
p:Pointer;
Sz:Word;

. . . . .
PieSlice(50,50,0,360,40);
Sz:=ImageSize(0,0,50,100);
GetMem(p,Sz);
GetImage(0,0,50,100,p^);
Delay(2000);
PutImage(0,0,p^,XORPut);
PutImage(200,50,p^,XORPut);
. . . . .
```

- *Pagina activă și pagina video.* Pentru modurile grafice care suportă mai multe pagini (EGA, VGA etc.), imaginea se poate constitui în pagina activă, în timp ce pe ecran este afișată pagina video (vezi §6.2). Pentru selecția paginilor se utilizează procedurile:

SetActivePage(NumarPagina);

SetVisualPage(NumarPagina);

în care parametrul *NumarPagina*, de tipul WORD, poate lua valori naturale, începând cu zero. Facilitatea de paginare în memoria video asigură o creștere de viteză și ușurință în realizarea imaginilor, mai ales, în animație.