GRAFIKA I ROZPOZNAWANIE OBRAZÓW

Podstawy programowania graficznego w środowisku MS DOS

1. Tryb tekstowy a tryb graficzny

Ze względu na reprezentację wyświetlanych danych na ekranie można wyróżnić dwa tryby pracy monitorów w komputerach *PC*:

- **tekstowy** - ekran jest tu podzielony na pewną ilość komórek, w których wyświetlane są pojedyncze znaki (np. litery). W pamięci wideo pojedyncza komórka zajmuje dwa bajty: dolny, będący kodem *ASCII* znaku, i górny, stanowiący atrybut wyświetlania.
- **graficzny** - na obraz składa się tutaj siatka pikseli, z których każdy może być wyświetlany w jednym z dostępnych kolorów. Format zapisu piksela w buforze wideo zależy od konkretnego trybu graficznego oraz od wielkości pamięci *VIDEO RAM* i wynosi od 1 bita dla trybów czarno-białych do 24 dla pełnego odwzorowania kolorów.

W obu trybach współrzędne ekranowe wyznaczane są względem lewego górnego rogu, którego współrzędne w trybie tekstowym wynoszą (1, 1), a w graficznym (0, 0). Współrzędna pozioma rośnie więc w prawo, natomiast pionowa w dół. *VIDEO BIOS* każdego komputera *PC* standardowo po włączeniu systemu wybiera tryb tekstowy 80-kolumnowy.

Istnieją dwie podstawowe metody przeprowadzania operacji graficznych:

- bezpośrednie programowanie kart graficznych.
- procedury graficzne *BIOS*.

Ponadto kompilatory języków wysokiego poziomu jak *Pascal* czy *C* zawierają biblioteki graficzne wraz ze sterownikami kart graficznych wykorzystującymi pierwszy z przedstawionych sposobów. Przykładem może być biblioteka graficzna firmy *Borland* ze sterownikami *BGI*.

2. Programowanie kart graficznych

Karta graficzna w standardzie *VGA* składa się z następujących elementów funkcjonalnych:

- **kontroler CRT** (ang. *Cathode Ray Tube Controler*) - steruje wyświetlaniem obrazu na monitorze: wytwarza sygnały synchronizacji i odchylania, realizuje wyświetlanie kursora itp. Ustawianie trybów graficznych polega na odpowiednim programowaniu rejestrów kontrolera CRT. Umożliwia to uzyskanie niestandardowych trybów graficznych.
- **sekwencer** - generuje sygnały sterujące dostępem do pamięci obrazu.
- **kontroler graficzny** - determinuje sposób działania karty w trybie graficznym.
- **przetwornik cyfrowo - analogowy DAC** (ang. *Digital to Analog Converter*) - przekształca wartości pikseli na analogowe sygnały sterujące intensywnością RGB na ekranie monitora. Posiada 256 rejestrów kolorów, z których każdy reprezentuje kolor w postaci trzech 6-bitowych wartości RGB. Stąd rozmiar palety odcieni do wykorzystania wynosi 2^{18} = 262144. W trybach 256-kolorowych jednocześnie można wyświetlić 256 kolorów z każdego rejestru. Dostęp do rejestrów pozwala zmieniać paletę.
- **kontroler atrybutów** - steruje kolorem w trybach 16-kolorowych. Zawiera rejestry palety odwzorowujące zawarte w pamięci ekranu numery kolorów z zakresu 0 ÷ 15 na wartości RGB z rejestrów *DAC*. Programując kontroler atrybutów można ustalić, które 16 spośród 256 kolorów *DAC* ma być aktualnie wykorzystywanych.
- **rejestry ogólnego przeznaczenia**.
- **ROM BIOS** - zawiera zbiór podstawowych procedur graficznych oraz tablice wzorców bitowych znaków o szerokości 8 pikseli i wysokości najczęściej: 8, 14 i 16.
- **VIDEO RAM**.

Termin *SVGA* odnosi się do kart bazujących na przedstawionej architekturze *VGA*, jednak posiadające pewne rozszerzenia, które zostały ustandaryzowane przez stowarzyszenie *VESA* (*Video Electronics Standards Association*).

Ponieważ przetwornik *DAC* posiada 256 rejestrów, więc maksymalna liczba kolorów, jaką można jednocześnie wyświetlić wynosi 256 (8 bitów na piksel). Aby w standardzie *VESA* ominąć ten problem używa się dla pikseli większych wartości (więcej bitów na piksel), określających bezpośrednio składowe *RGB*, a nie pośrednio poprzez *DAC* lub kontroler atrybutów.

Tabela 1. Reprezentacja pikseli w trybach koloru bezpośredniego *SVGA*

bity na piksel	reprezentacja bitowa kolejnych składowych *RGB*	liczba kolorów
15	15 … 0: x | *5R* | *5G* | *5B*	32768 (32k)
16	15 … 0: *5R* | *6G* | *5B*	65536 (64k)
24	23 … 0: *8R* | *8G* | *8B*	16777216 (16M)

Karty graficzne *SVGA* ze względu na architekturę można podzielić na 3 grupy:

- **bierne bufory** - posiadają więcej pamięci *VRAM*, co zapewnia tryby o wyższej rozdzielczości i większej liczbie kolorów.
- **karty z przyspieszaczami (akceleratorami)** - umożliwiają szybkie przenoszenie fragmentów *VRAM* oraz pewne operacje na blokach *VRAM*.
- **karty z procesorami graficznymi** - obsługują sprzętowo różne funkcje graficzne: rysowanie odcinków, krótkich wektorów (krótkie linie: poziome, pionowe i ukośne), wypełnianie prostokątów, wycinanie i przenoszenie fragmentów obrazu.

Przedstawione układy zawierają programowalne rejestry sterujące dostępne do zapisu lub odczytu.

Karta graficzna z poziomu procesora jest widziana jako zbiór portów w przestrzeni wejścia - wyjścia oraz obszar pamięci obrazu (bufor wideo) położony w zakresie adresów A000:0000h a B000:FFFFh, czyli dwa segmenty po 64kB. Ponieważ wszystkie karty *SVGA* zawierają minimum 512 kB, dla zrealizowania komunikacji procesor - *VRAM* stosuje się techniki stronicowania, polegające na kojarzeniu z tym samym obszarem adresowym różnych fragmentów większego bloku pamięci zwanych stronami lub bankami. Przełączanie banków następuje przez wpisanie numeru strony do specjalnego rejestru.

Bezpośrednie programowanie polega więc na zapisie lub odczycie odpowiednich wartości do i z portów wejścia - wyjścia oraz pamięci.

Programowanie karty graficznej należy do metod najtrudniejszych, daje jednak największe możliwości i jest najbardziej efektywne. Programy obsługi kart graficznych nazywa się sterownikami.

3. Procedury graficzne BIOS

Procedury *ROM BIOS* karty graficznej dostarczają zbiór prostych narzędzi do wykonywania podstawowych zadań programowania graficznego: zmiana trybów pracy, ustawianie koloru piksela, wyprowadzanie ciągu znaków, czyszczenie ekranu, zmiana kolorów itp. Są one wolne i prymitywne jednak zapewniają zgodność z różnymi kartami graficznymi.

Dostęp do nich umożliwiają funkcje przerwania 10h. Przed ich wywołaniem (INT 10h) należy do rejestru AH wpisać numer funkcji, zaś do pozostałych rejestrów - ewentualne parametry.

Tabela 2. Niektóre tryby graficzne standardu *VGA BIOS*

numer trybu	rozdzielczość	liczba kolorów
3h (tekstowy)	80x25	16
0Eh	640x200	16
10h	640x350	16
11h	640x480	2
12h	640x480	16
13h	320x200	256

Tabela 3. Podstawowe funkcje graficzne przerwania 10h

numer	opis
00h	Przełączanie trybów pracy parametry wejściowe • AL - numer trybu
0Ch	Ustawienie koloru piksela parametry wejściowe: • AL - numer koloru • BH - numer strony pamięci ekranu • CX - współrzędna pozioma punktu • DX - współrzędna pionowa punktu
0Fh	Odczyt trybu pracy karty: zwracane wartości: • AH - szerokości ekranu w znakach dla trybów tekstowych • AL - numer trybu pracy • BH - numer aktualnie aktywnej strony ekranu
10h AL = 00h	Modyfikacja rejestru 16 - kolorowej palety kontrolera atrybutów parametry wejściowe: • BH - nowa zawartość rejestru (bity 5÷0) • BL - numer rejestru (wartość 0÷15)
10h AL = 10h	Wpis wartości do rejestru *RGB* przetwornika *DAC* parametry wejściowe: • BX - numer rejestru (0÷255) • DH - składowa R (0÷63) • CH - składowa G • CL - składowa B

Stosując *VGA BIOS* można jednocześnie wyświetlić maksymalnie 256 kolorów tryb 13h.

Wraz z kartami *SVGA* pojawił się *VESA BIOS*, będący rozszerzeniem *VGA BIOS* o nowe funkcje i tryby z 16-bitowymi numerami. Umożliwiają one pracę aż z 16 milionami kolorów jednocześnie.

Tabela 4. Niektóre tryby graficzne *VESA BIOS*

numer trybu	rozdzielczość	liczba kolorów
101h	640x480	256
10Dh	320x200	32k
111h	640x480	64k
115h	800x600	16M
11Bh	1280x1024	16M

Tabela 5. Niektóre dodatkowe funkcje przerwania 10h obsługiwane przez *VESA BIOS*

numer	opis
4F02h	Przełączanie trybów *VESA* (analogia do funkcji 00h) parametry wejściowe: • BX - bity 14÷0: właściwy numer trybu *VESA*. Bit 15 = 0: wyzerowanie pamięci obrazu, bit 15 = 1: brak wyzerowania wartości zwracane: • AH - wynik wykonania funkcji: 00h - wykonanie poprawne, 01h - błąd wykonania • AL - 4Fh - jeśli funkcja dostępna (*VESA BIOS* obecny)
4F08h	Ustawienie lub odczyt liczby bitów na kolor podstawowy w wideo *DAC* parametry wejściowe: • BL - 00h: ustawianie szerokości palety, 01h: odczyt szerokości palety • BH - liczba bitów na składową koloru (R, G, lub B) wartości zwracane: • AX - jak w funkcji 4F02h

Do posługiwania się funkcjami przerwania 10h można wykorzystać następującą procedurę:

```
procedure INT_10h(ax, bx, cx, dx: word; var Rejestry: Registers);
  begin
   Rejestry.AX := ax;
   Rejestry.BX := bx;
   Rejestry.CX := cx;
   Rejestry.DX := dx;
   Intr($10, Rejestry)
 end;
```

4. Biblioteka graficzna firmy Borland

Kompilator *Turbo Pascala* zawiera moduł *Graph* implementujący bibliotekę graficzną z ponad 50 procedurami zarówno wysokiego, jak i niskiego poziomu. Aby skompilować program odwołujący się do modułu *Graph* potrzebne jest ustawienie ścieżki dostępu do modułu *GRAPH.TPU.* Do uruchomienia niezbędne są sterowniki graficzne w plikach **.BGI* oraz pliki czcionek **.CHR* (tylko gdy program wykorzystuje czcionki skalowane).

Pliki *BGI* zawierają: nagłówek, procedurę wywołującą konkretne funkcje graficzne oraz ich kody wykorzystujące bezpośrednio elementy funkcjonalne danej karty. Procedury te mogą być wywoływane dzięki deklaracjom zawartym w module *Graph*.

Do uruchomienia trybu graficznego służy procedura `InitGraph`, która podczas pracy programu identyfikuje kartę graficzną, ładuje odpowiedni sterownik graficzny (znajdujący się w katalogu) i zwraca sterownie do wywoływanych procedur.

Istnieje również możliwość włączenia sterownika graficznego lub pliku z czcionkami do pliku wynikowego dzięki programowi *BGIOBJ.EXE* przekształcającemu pliki ze sterownikami lub z czcionkami do plików **.OBJ*, które można włączać do programu jako moduły lub procedury zewnętrzne używając dyrektywy kompilatora `{$L}`. Proces ten przebiega następująco:

- uruchomienie *BINOBJ* z danym plikiem **BGI* lub **CHR* jako parametrem.
- włączenie pliku **.OBJ* do programu jako procedury lub modułu zewnętrznego.
- zarejestrowanie włączonego pliku procedurą `RegisterBGIdriver` lub `RegisterBGIfont` przed wywołaniem `InitGraph`.

Procedura `CloseGraph` usuwa sterownik z pamięci i powraca do poprzedniego trybu pracy. Przełączanie między trybem graficznym a tekstowym realizuje się procedurami `RestoreCrtMode` i `SetGraphMode`.

Tabela 6. Dostępne tryby graficzne dla karty *VGA* (sterownik *EGAVGA.BGI*)

stała	wartość	rozdzielczość	liczba kolorów	ilość stron ekr.
VGALo	0	640x200	16	4
VGAMed	1	640x350	16	2
VGAHi	2	640x480	16	1

Standardowy sterownik dla karty *VGA* umożliwia stosowanie tylko trybów 16-kolorowych. Wykorzystując prosty format pliku *BGI* można dla danej karty *SVGA* napisać sterownik. Poniżej przedstawiono przykładową funkcję uruchomienia trybu graficznego.

```
function Tryb_Graficzny(sterownik, tryb: integer;
                        sciezka_dostępu_do_sterownikow: string):boolean;
var
  kod_bledu : integer;
begin
  InitGraph(sterownik, tryb, sciezka_dostępu_do_sterownikow);
  kod_bledu := GraphResult;
  if kod_bledu <> grOK then
    begin
      writeln('Błąd inicjacji trybu graficznego: ',
              GraphErrorMsg(ErrorCode));
      Tryb_Graficzny := false
    end
  else Tryb_Graficzny := true
end;
```

Funkcja `GraphResult` zwraca kod błędu ostatniej operacji graficznej, natomiast `GraphErrorMsg` zwraca komunikat o błędzie. Jeśli parametrowi `sterownik` procedury `InitGraph` przypisze się stałą `Detect`, to procedura ta wykryje typ karty graficznej i dobierze dla niej odpowiedni tryb graficzny o maksymalnej rozdzielczości. Aby wywołać dla karty *VGA* tryb *VGALo* należałoby przyjąć : `sterownik := VGA, tryb := VGALo`. Możliwe jest oczywiście również ustawienie innych sterowników i trybów.

Aby włączyć do programu źródłowego np. sterownik *EGAVGA.BGI* jako procedurę zewnętrzną o nazwie `ProcSterVga` należy:

- utworzyć deklarację postaci:
  ```
  procedure ProcSterVga; external;
  {$L EGAVGA.OBJ }
  end;
  ```
- przed `InitGraph` wywołać funkcję `RegisterBGIdriver(@ProcSterVga)`, która w przypadku błędu zwraca wartość ujemną. Parametr procedury `InitGraph`, będący ścieżką dostępu do sterownika jest pomijany.

Analogicznie jak przedstawiono powyżej można włączyć do pliku wynikowego również pliki z czcionkami.

Czcionki skalowane (wektorowe) są zdefiniowane jako ciąg linii i ewentualnie miejsc, od których należy rozpocząć wypełnianie. Wszystkie litery mają identyczną wysokość, ale mogą mieć różną szerokość. Ich przewaga w stosunku do czcionek bitowych (*ROM BIOS*) polega na tym, że jako obiekty wektorowe nie są zniekształcane przy zmianach rozmiarów. Firma *Borland* nie dostarcza niestety w swych produktach plików z polskimi literami.

Istotnym pojęciem w systemach graficznych jest aktualny wskaźnik *CP* `(xp,yp)` (kursor graficzny), będący analogią do koncepcji kursora tekstowego (nie jest on jednak widoczny). Chcąc

narysować - na przykład - linię łączącą punkty `(xp,yp)` i `(xk,yk)` należy przesunąć *CP* do punktu `(xp,yp)`, a następnie wywołać funkcję rysującą linię od *CP* do punktu `(xk,yk)`.

Rysowanie podstawowych obiektów graficznych umożliwiają następujące procedury:

- `PutPixel(x, y, kolor)` - zmienia kolor piksela o współrzędnych (`x,y`).
- `MoveTo(x, y)` - przesuwa *CP* do punktu (`x,y`).
- `MoveRel(dx, dy)` - przesuwa aktualny *CP* o wektor (`dx,dy`).
- `Line(x1, y1, x2, y2)` – rysuje odcinek między punktami (`x1,y1`) a (`x2,y2`).
- `LineTo(x, y)` - rysuje odcinek od *CP* do (`x,y`).
- `LineRel(dx, dy)` - rysuje odcinek od *CP* w kierunku i o długości wektora (`dx,dy`).
- `OutTextXY(x, y, tekst)` - wysyła na ekran `tekst`.
- `FloodFill(x, y, kolor_granicy)` - wypełnia ograniczony obszar aktualnym wzorem i kolorem wypełniania, przy czym (`x,y`) wskazuje punkt wnętrza obszaru wypełnianego, zaś `kolor_granicy` określa kolor brzegu.

Do ustawiania atrybutów rysowanych obiektów służą procedury:

- `SetColor(kolor)` - aktualny kolor rysowania (z palety).
- `SetBkColor(kolor)` - aktualny kolor tła.
- `SetLineStyle(styl_linii, wzor, grubosc)` - wzór i grubość linii.
- `SetFillStyle(wzor, kolor)` - wzór i kolor wypełniania.
- `SetTextStyle(czcionka, kierunek, rozmiar)` - parametry tekstu.

Idea wielu stron pamięci ekranu umożliwia kierownie graficznego wyjścia na strony nie wyświetlane aktualnie na ekranie, a następnie ich wyświetlanie przez zmianę aktywnej strony. Technika ta wykorzystywana jest na przykład w animacji. Procedury pozwalające operować stronami ekranu to:

- `SetVisualPage(numer_strony)` - aktualnie widoczna strona.
- `SetActivePage(numer_strony)` - aktualnie aktywna strona.

Oddzielną grupę stanowią procedury operujące na mapach bitowych:

- `PutImage(x, y, mapa, operacja)` - kopiuje na ekran graficzny mapę bitową reprezentowaną parametrem `mapa` w miejsce (`x,y`). Parametr `operacja` zadaje rodzaj operacji bitowej wykonywanej na bitach mapy i ekranu. Jego możliwe wartości określają stałe: `CopyPut, XORPut, OrPut, AndPut, NotPut`.
- `GetImage(x1, y1, x2, y2, mapa)` - zapisuje obraz bitowy o rozmiarach `x1, y1, x2, y2` do bufora `mapa`.
- `ImageSize(x1, y1, x2, y2)` - zwraca liczbę bajtów potrzebnych do zapamiętania prostokątnego obszaru na ekrania. Funkcja ta wykorzystywana jest do określania rozmiaru bufora dla procedury `GetImage`.

opracowali Robert Gwadera i Mariusz Szwoch