Laboratorium Grafiki Komputerowej i Animacji

Ćwiczenie II

Formaty plików graficznych

Sławomir Samolej

Rzeszów, 1999

1. Wstęp

Grafika jest podstawowym elementem multimedialnych aplikacji. Często w programach tego typu duży rysunek stanowi tło, na którym poruszają się mniejsze elementy graficzne, zwane duszkami (ang. sprites). Przykładowo, w grach komputerowych pojawia się wiele realistycznie wyglądających rysunków, wprowadzających gracza w nastrój gry. Gry i programy edukacyjne wyświetlają obrazy graficzne, aby przyciągnąć uwagę dziecka i zwiększyć w ten sposób skuteczność nauczania.

Aby można było wykorzystać w programie obraz graficzny, obraz ten musi być zapisany w sposób elektroniczny. Można przygotowywać grafikę przy pomocy programu graficznego, można też zeskanować gotowe, drukowane ilustracje. Niezależnie od sposobu przygotowania rysunku, jest on zapisany w pliku dyskowym w określonym formacie i musi być odpowiednio przetworzony przed wyświetleniem. Opracowanie opisuje jeden z najpopularniejszych formatów plików graficznych - bitmapę. Zawiera także opis sposobu konstruowania oprogramowania zdolnego do wyświetlania map bitowych. Do opracowania dołączona jest dyskietka zawierająca szkielet aplikacji Windows o nazwie BmpView, który po uzupełnieniu o omówione poniżej funkcje można przekształcić w prostą przeglądarkę plików typu bitmapa.

2. Formaty plików graficznych

Obraz graficzny jest dwuwymiarową tablicą pikseli, zwaną czasem rastrem. Kolor każdego piksela jest reprezentowany w jeden z poniższych sposobów:

- W przypadku obrazów monochromatycznych kolor piksela jest określony przez jeden bit, który może przybierać wartość 1 lub 0.

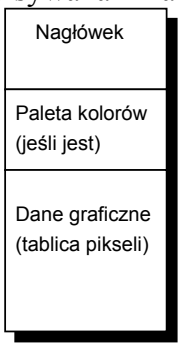
- W przypadku obrazów "true color" dla każdego piksela są określone trzy składowe: czerwona (R), zielona (G) i niebieska (B). Typowo wartość każdej składowej jest reprezentowana przez jeden bajt, co daje 256 poziomów jasności każdej barwy składowej. Takie podejście wymaga przeznaczenia na każdy piksel trzech bajtów i umożliwia wyświetlenie 256 ### 256 ### 256 = 16777216 (czyli ponad 16 milionów) różnych kolorów.

- W przypadku obrazu opartego na palecie, wartości poszczególnych pikseli są indeksami tablicy RGB, zwanej paletą kolorów. Liczba przypadających na jeden piksel bitów zależy od liczby kolorów w palecie. Typowa paleta może zawierać 16 (4 bity na piksel) lub 256 kolorów (8 bitów na piksel).

Plik graficzny, aby można było z niego odczytać dane i wyświetlić na ekranie musi zawierać następujące minimum danych:
- Rozmiary rysunku (szerokość i wysokość).
- Liczbę bitów na piksel
- Rodzaj grafiki - czy wartość pikseli określają składowe RGB, czy też są indeksami palety
- Paletę kolorów, jeżeli obraz jest opisany na palecie
- Dane graficzne, będące tablicą pikseli.

Prawie wszystkie pliki graficzne zawierają ten minimalny zestaw informacji, ale informacje te mogą być zapisane w różny sposób. Rysunek 2.1 pokazuje ogólną strukturę typowego pliku graficznego. Na początku znajduje się nagłówek. Zawiera on różne informacje, związane z

obrazem graficznym (oprócz samych danych graficznych i palety kolorów). Dalej znajduje się paleta kolorów, jeśli obraz opiera się na palecie. Dane graficzne - tablica pikseli - znajdują się za paletą. Zwykle tablica pikseli jest zapisywana linia po linii.



Rys 2.1 Typowa struktura pliku graficznego

Tablica pikseli jest największą częścią pliku. Przykładowo, dla rysunku o rozdzielczości 640 ### 480 w 256 kolorach konieczne jest 640 ### 480 = 307200 bajtów (każdy piksel zajmuje jeden bajt). Można oczywiście zastosować kompresję danych w celu zmniejszenia długości pliku graficznego. Chociaż większość plików ma ogólną strukturę podobną jak na rysunku 2.1 to możliwe jest wiele różnych modyfikacji:

- Kolejność umieszczonych w nagłówku informacji może być różna dla różnych formatów plików.
- Niektóre formaty, przystosowane do konkretnej karty graficznej, nie muszą zawierać palety, a jedynie tablice pikseli.
- Linie obrazu mogą być ułożone w kolejności od góry do dołu lub od dołu do góry.
- Jeżeli wartości pikseli są składowymi RGB, to kolejność zapisu tych składowych może być różna.
- Wartości pikseli mogą być zapisane w postaci pakietów lub planów obrazu. W przypadku pakietów bity, odpowiadające kolejnym pikselom, są zapisywane jeden za drugim. Jeżeli obraz składa się z planów, bity poszczególnych pikseli są zapisywane na tej samej pozycji w kolejnych planach. Najmniej znaczące bity pikseli obrazu znajdują się w jednym planie, a następnym planie są bity bardziej znaczące, itd.
- Dane graficzne mogą być skompresowane.

3. Mapy bitowe niezależne od sprzętu (DIB)

W Windows 3.0 zdefiniowany został nowy format map bitowych, tzw. map bitowych niezależnych od sprzętu, czyli DIB (ang. device-independent bitmap). DIB zawiera własną tabelę barw, która określa metodę przekodowywania bitów piksela na kolory systemu RGB. Obraz w formacie DIB można wyświetlić na każdym rastrowym urządzeniu wyjściowym. Jedyny problem to konieczność dopasowania kolorów mapy bitowej do kolorów dostępnych na danym urządzeniu. Format DIB służy głównie do wymiany obrazów między programami. Można go zapisać w pliku albo skopiować do schowka.

Drugim rodzajem map bitowych występujących w Windows są mapy bitowe zależne od sprzętu - DDB (ang. device-dependet bitmaps). Są one uzależnione od sprzętu, bo muszą być kompatybilne z konkretnym graficznym urządzeniem wyjściowym. Wyświetlanie bitmap w środowisku Windows polega zwykle na odczycie niezależnych od sprzętu map bitowych DIB, przekształceniu ich w mapy zależne sprzętowo DDB a następnie przesłanie ich do konkretnego urządzenia wyświetlającego, lub drukującego.

W plikach nagłówkowych Windows (np: windows.h) znajdują się gotowe struktury ułatwiające pracę z bitmapami. Odpowiednie struktury i pola struktur odzwierciedlają sposób organizacji danych zarówno w pliku DIB jak i mapie bitowej zależnej od sprzętu.

3.1 Plik DIB

Podstawowym typem plików graficznych wykorzystywanych w Windows są bitmapy. Zwykłym rozszerzeniem spotykanym dla tych plików jest .BMP lub .DIB. Poniżej przedstawiona zostanie struktura pliku DIB.

Początek nagłówka pliku DIB wypełnia struktura BITMAPINFOHEADER składa się ona z 5 pól:

```
typedef struct tagBITMAPFILEHEADER { // bmfh
        WORD    bfType;
        DWORD   bfSize;
        WORD    bfReserved1;
        WORD    bfReserved2;
        DWORD   bfOffBits;
} BITMAPFILEHEADER;
```

Pole	Wielkość	Opis
bfType	WORD	Dwa bajty "BM" (od słów: mapa bitowa)
bfSize	DWORD	Całkowita wielkość pliku
bfReserved1	WORD	Wartość 0
bfReserved2	WORD	Wartość 0
bfOffBits	DWORD	Odległość od początku pliku do miejsca, w którym zaczynają się bity mapy bitowej.

Potem następuje drugi nagłówek (BITMAPINFO) składający się z dwu struktur:

```
typedef struct tagBITMAPINFO { // bmi
   BITMAPINFOHEADER bmiHeader;
   RGBQUAD          bmiColors[1];
} BITMAPINFO;
```

Pierwsza ze struktur licząca 11 pól ma postać:

```
typedef struct tagBITMAPINFOHEADER{ // bmih
   DWORD  biSize;
   LONG   biWidth;
   LONG   biHeight;
   WORD   biPlanes;
   WORD   biBitCount;
   DWORD  biCompression;
   DWORD  biSizeImage;
   LONG   biXPelsPerMeter;
   LONG   biYPelsPerMeter;
   DWORD  biClrUsed;
   DWORD  biClrImportant;
} BITMAPINFOHEADER;
```

Pole	Wielkość	Opis
biSize	DWORD	Wielkość tej struktury w bajtach
biWidth	LONG	Szerokość mapy bitowej w pikselach
biHeight	LONG	Wysokość mapy bitowej w pikselach
biPlanes	WORD	Wartość 1

biBitCount	WORD	Liczba bitów koloru przypadająca na jeden piksel (1,4,8 lub 24)
biCompression	DWORD	Metoda kompresji (0, gdy nie ma kompresji)
biSizeImage	DWORD	Wielkość mapy bitowej w bajtach (potrzebne jedynie w przypadku map skompresowanych)
biXPelsPerMeter	LONG	Rozdzielczość pozioma w pikselach na metr
biYPelsPerMeter	LONG	Rozdzielczość pionowa w pikselach na metr
biClrUsed	WORD	Liczba kolorów użyta w obrazie
biClrImportant	DWORD	Liczba istotnych kolorów użyta w obrazie

Wszystkie pola występujące po biBitCount mogą zawierać domyślną wartość 0 (lub w ogóle mogą nie wystąpić w pliku). Struktura może więc liczyć jedynie 16 bajtów. Może jednak także zawierać pola dodatkowe, nie wymienione powyżej.

Jeśli biClrUsed równa się 0, a liczba bitów koloru pikseli wynosi 1, 4, lub 8, po strukturze BITMAPINFOHEADER następuje tablica barw, a w niej - co najmniej dwie struktury RGBQUAD. Struktura RGBQUAD zawiera definicję wartości kolorów w systemie RGB:

```
typedef struct tagRGBQUAD { // rgbq
    BYTE    rgbBlue;
    BYTE    rgbGreen;
    BYTE    rgbRed;
    BYTE    rgbReserved;
} RGBQUAD;
```

Pole	**Wielkość**	**Opis**
rgbBlue	BYTE	Intensywność barwy niebieskiej
rgbGreen	BYTE	Intensywność barwy zielonej
rgbRed	BYTE	Intensywność barwy czerwonej
rgbReserved	BYTE	Wartość 0

Liczba struktur RGBQUAD jest zwykle wyznaczona przez pole biBitCount: dla koloru 1-bitowego potrzebne są dwie struktury RGBQUAD, dla kolorów 4-bitowych - 16, a dla 8-bitowych - 256. Jednak w przypadku, gdy pole biClrUsed jest różne od zera, biClrUsed zawiera liczbę struktur RGBQUAD występujących w tablicy barw.

Po tablicy barw następuje tablica bitów definiujących obraz mapy bitowej. Tablica zaczyna się od dolnego wiersza pikseli. Każdy wiersz zaczyna się od piksela położonego z lewej strony. Każdy piksel zajmuje 1, 4, 8, lub 24 bity. W przypadku monochromatycznych map bitowych, w których 1 bit koloru przypada na piksel, pierwszy piksel w każdym wierszu odpowiada najbardziej znaczącemu bitowi pierwszego bajtu w każdym wierszu. Jeśli wartość bitu wynosi 0, kolor piksela należy odczytać z pierwszej struktury RGBQUAD znajdującej się w tablicy barw. Jeżeli wartość bitu wynosi 1, to kolor jest zakodowany w drugiej strukturze RGBQUAD.

W przypadku 16-kolorowych map bitowych z 4 bitami koloru na piksel, pierwszy piksel w każdym wierszu odpowiada czterem najbardziej znaczącym bitom pierwszego bajtu każdego wiersza. W celu odkodowania koloru piksela używamy pobranej 4-bitowej wartości jako indeksu do tablicy barw. Tablica barw liczy 16 elementów.

W 256-kolorowych mapach bitowych każdy bajt odpowiada jednemu pikselowi. Kolor piksela odczytujemy używając wartości 8-bitowej jako indeksu do tablicy barw, która liczy tym razem 256 elementów.

Jeśli obraz składa się z pikseli kodowanych na 24 bitach koloru, to każdy zbiór kolejnych trzech bajtów odpowiada wartości RGB piksela. W tym przypadku nie ma tablicy barw, chyba że pole biClrUsed struktury BITMAPINFOHEADER jest różne od 0.

W każdym przypadku, każdy wiersz danych mapy bitowej zawiera wielokrotność 4 bajtów. Jeżeli nie pasuje to do rzeczywistych wymiarów obrazu, to wiersz jest uzupełniany, aby ten warunek został spełniony.

Wraz z Windows 95 pojawił się trzeci nagłówek, o nazwie BITMAPV4HEADER. (System Windows 95 czasami jest nazywane Windows 4.0, stąd akronim V4, czyli "Version 4"). Znajdują się w nim dodatkowe informacje korygujące wyświetlanie kolorów na różnych urządzeniach:

```
typedef struct {
    DWORD        bV4Size;
    LONG         bV4Width;
    LONG         bV4Height;
    WORD         bV4Planes;
    WORD         bV4BitCount;
    DWORD        bV4V4Compression;
    DWORD        bV4SizeImage;
    LONG         bV4XPelsPerMeter;
    LONG         bV4YPelsPerMeter;
    DWORD        bV4ClrUsed;
    DWORD        bV4ClrImportant;
    DWORD        bV4RedMask;
    DWORD        bV4GreenMask;
    DWORD        bV4BlueMask;
    DWORD        bV4AlphaMask;
    DWORD        bV4CSType;
    CIEXYZTRIPLE bV4Endpoints;
    DWORD        bV4GammaRed;
    DWORD        bV4GammaGreen;
    DWORD        bV4GammaBlue;
} BITMAPV4HEADER, FAR *LPBITMAPV4HEADER, *PBITMAPV4HEADER;
```

4. Metody wyświetlania bitmap w środowisku Windows

Podstawowy sposób wyświetlania bitmap w środowisku Windows polega na wczytaniu pliku do pamięci w postaci DIB, przekształceniu danych w mapę DDB i przesłaniu tak skonstruowanego obiektu na ekran monitora. Drugim, sposobem wyświetlania jest zastosowanie odpowiednich funkcji bezpośrednio odczytujących dane do rysowania z wczytanych do pamięci DIB. Czas wyświetlania nie przekształconych plików DIB jest znacznie dłuższy od czasu wyświetlania przetworzonych map bitowych DDB.

4.1 Bezpośrednie wyświetlanie obrazów DIB

Plik DIB można bezpośrednio wczytać do zarezerwowanego bloku pamięci. Mówi się wówczas o formacie pamięciowym "upakowany DIB". Zawiera on wszystko, co znajduje się w pliku DIB, z wyjątkiem struktury BITMAPFILEHEADER. Tak więc blok pamięci zaczyna się od struktury nagłówka informacyjnego, po której występuje tablica barw (o ile w ogóle jest obecna), a następnie bity mapy bitowej. Format pamięciowy upakowany DIB służy do przesyłania obrazów w formacie DIB przez schowek. Windows ma dwie funkcje umożliwiające wyświetlanie map bitowych z bloku pamięci zawierającego upakowany DIB: StretchDIBits() oraz SetDIBitsToDevice(). Obydwie funkcje wymagają podania wskaźnika do struktury BITMAPINFOHEADER - początku pamięci zawierającego obraz DIB - oraz wskaźnika do bitów mapy bitowej.

4.2 Wyświetlanie obrazów DDB

Powszechniejszym sposobem wyświetlania bitmap w systemie Windows jest tworzenie map bitowych zależnych od sprzętu i umieszczanie ich w oknach programu. W przypadku, gdy zadaniem programu jest wczytywanie i wyświetlanie różnych plików graficznych (np. przeglądarka plików graficznych) drogę postępowania można podzielić na następujące etapy:

- *Wczytanie zawartości pliku do pamięci komputera*
- *Przekształcenie wczytanych danych w format DDB*
- *Przesłanie obrazu do okna programu*

Przyglądając się strukturze nagłówka pliku typu DIB łatwo zauważyć, że wczytywanie danych do pamięci może składać się z dwu kroków. W pierwszym należy odczytać z pliku zawartość struktury BITMAPFILEHEADER. Pole `bfSize` tej struktury zawiera całkowity rozmiar pliku w bajtach. Bazując na tej informacji można w kroku drugim zaalokować w pamięci komputera bufor o odpowiednim rozmiarze i wczytać do niego pozostałe dane. Pamiętając o budowie pliku typu DIB (rozdział 3.1) można do odpowiednich obszarów pamięci zajmowanych przez dane przyłączyć wskaźniki na odpowiednie struktury (BITMAPFILEHEADER, BITMAPINFOHEADER, BITMAPINFO). Przyłączone wskaźniki mogą być wykorzystywane w funkcjach przetwarzających wczytane dane graficzne.

Do utworzenia DDB w systemie Windows wykorzystuje się tak zwane uchwyty bitmap. Uchwyt bitmapy jest jednym z predefiniowanych typów Windows (HBITMAP), a sposób jego utworzenia prezentuje linia kodu:

```
static HBITMAP mHBmp=0;
```

Jeśli założymy, że ten sam uchwyt do bitmapy wykorzystywany będzie do wyświetlania wielu obiektów, to przed ponowną próbą użycia uchwytu do utworzenia nowego obiektu należy usunąć z pamięci (jeśli istnieje) poprzedni obiekt, na który wskazywał uchwyt. Zakładając, że uchwyt do bitmapy mHBmp (typu HBITMAP) wskazuje na istniejący obiekt w pamięci to fragment kodu:

```
if(mHBmp)
{ DeleteObject(mHBmp);
  mHBmp=0;
}
```

powoduje usunięcie obiektu z pamięci (poprzez wywołanie funkcji DeleteObject()) i ustawienie uchwytu na wartość 0.

Podstawową funkcją służącą do przetworzenia danych w formacie DIB na dane DDB jest CreateDIBitmap(). Prototyp funkcji jest następujący:

```
HBITMAP CreateDIBitmap
(
HDC hdc,                            // uchwyt do kontekstu urządzenia
CONST BITMAPINFOHEADER *lpbmih,     // wskaźnik do struktury
                                    // BITMAPINFOHEADER przetwarzanej DIB
DWORD fdwInit,                      // flaga inicjalizacji
CONST VOID *lpbInit,                // wskaźnik do początku tablicy pikseli
CONST BITMAPINFO *lpbmi,            // wskaźnik do struktury BITMAPINFO
                                    // przetwarzanej DIB
UINT fuUsage                        // tryb konwersji kolorów
```

```
);
```

Pierwszym parametrem funkcji jest uchwyt do kontekstu urządzenia (typ Windows: HDC) , dla którego tworzona ma być mapa bitowa (Pojęcie kontekstu urządzenia i jego rola w Windows omówione zostały we wprowadzeniu do ćwiczenia I z przedmiotu Grafika Komputerowa i Animacja, rozdział 2.3.3). Jeśli mapa bitowa ma być wyświetlana w zadanym oknie aplikacji, do funkcji podany powinien być uchwyt kontekstu urządzenia zgodny z uchwytem okna. Do uzyskania uchwytu kontekstu urządzenia zgodnego z uchwytem zadanego okna służy funkcja GetDC(). Jeśli przyjmiemy, że *hwnd* (typ Windows: HWND) jest uchwytem do okna, w którym chcemy wyświetlić bitmapę a *mHDC* (typ Windows: HDC) jest uchwytem kontekstu urządzenia, to wywołanie:

```
mHDC=GetDC(hwnd);
```

powoduje, że w zmiennej mHDC przechowywany jest kontekst urządzenia zgodnego z kontekstem urządzenia dla okna identyfikowanego przez uchwyt hwnd.

Drugim parametrem przekazywanym do funkcji CreateDIBitmap() jest wskaźnik do struktury typu BITMAPINFOHEADER wczytanego do pamięci pliku DIB, dla którego chcemy dokonać konwersji danych.

Parametr trzeci (flaga inicjalizacji) może przyjmować 2 wartości. Jeśli wynosi on **CBM_INIT** (predefiniowana stała Windows) do utworzenia bitmapy uwzględniane są dane wskazywane przez parametry *lpbInit* i *lpbmi*. W przypadku zaś, gdy parametr wynosi 0 do tworzenia bitmapy DDB nie są uwzględniane wymienione parametry. Do tworzenia map DDB na podstawie DIB parametr ten ustala się na CBM_INIT.

Parametr czwarty wskazywać ma na początek tablicy pikseli przetwarzanej DIB (wskaźnik ten powinien być typu **unsigned char***). Położenie początku tablicy pikseli można wyznaczyć na podstawie analizy odpowiednich pól struktur nagłówkowych pliku DIB.

Piąty parametr funkcji jest wskaźnikiem na strukturę BITMAPINFO przetwarzanej DIB.

Ostatni parametr wywołania funkcji ustala w jaki sposób interpretowana jest tablica kolorów zawarta w danych z pliku DIB. Jeśli parametr przyjmuje wartość **DIB_PAL_COLORS** (predefiniowana stała Windows) przetwarzany plik traktowany jest jako plik z paletą kolorów zawierającą 16 bitowe wskaźniki do palety opisujące kolory poszczególnych pikseli. Jeśli zaś parametr ustalony jest na **DIB_RGB_COLORS** (predefiniowana stała Windows) sugeruje to, że przetwarzany plik traktowany jest jako mapa bitowa bez palety kolorów. Kolor piksela określają wtedy trzy kolejne odczytane bajty reprezentujące składowe czerwoną, zieloną i niebieską koloru (model kolorów RGB). Do przetwarzania większości plików DIB wykorzystuje się parametr DIB_RGB_COLORS.

W przypadku pomyślnego utworzenia DDB na podstawie DIB funkcja CreateDIBitmap() zwraca uchwyt typu HBITMAP do nowo utworzonej mapy bitowej zależnej od urządzenia. W przeciwnym razie funkcja zwraca 0.

Ponieważ na czas wykowywania funkcji CreateDIBitmap() pobierany jest uchwyt kontekstu urządzenia do okna programu, po zakończeniu działania funkcji należy ten uchwyt zwolnić. Przykładowa linia kodu:

```
ReleaseDC(hwnd,mHDC);
```

przy pomocy funkcji ReleaseDC() zwalnia uchwyt kontekstu urządzenia przechowywany w zmiennej mHDC (typu HDC) (dodatkowym parametrem jest uchwyt okna - hwnd (typu HWND), dla którego tworzony był uchwyt kontekstu urządzenia).

Jeśli istnieje w pamięci programu utworzona bitmapa DDB (identyfikowana poprzez uchwyt typu HBITMAP), to możliwe jest przesłanie jej zawartości do okna. Specyfika programowania w środowisku Windows wymaga jednak ściśle określonego trybu postępowania przy wyświetlaniu obiektów typu DDB. Pierwszym etapem wyświetlania bitmapy jest utworzenie tak zwanego kontekstu urządzenia pamięciowego. Kontekst urządzenia pamięciowego to taki kontekst, który ma "powierzchnię wyświetlania" istniejącą tylko w pamięci komputera. Kontekst taki można uzyskać za pomocą funkcji:

```
mMemDC=CreateCompatibleDC(HDCPaint);
```

Uchwyt HDCPaint jest uchwytem istniejącego kontekstu urządzenia. Jak widać, kontekst urządzenia pamięciowego tworzony jest na podstawie kontekstu rzeczywistego urządzenia (Często "wzorem" do utworzenia kontekstu pamięciowego jest kontekst urządzenia, na którym zamierza się wyświetlić mapę bitową). Funkcja CreateCompatibleDC() zwraca uchwyt kontekstu urządzenia pamięciowego.

Następnie do utworzonego kontekstu urządzenia należy "przyłączyć" utworzoną mapę DDB. Makrorozwinięcie SelectBitmap() wybiera w kontekście pamięciowym bitmapę:

```
mHBitmapOld=SelectBitmap(mMemDC,mHBmp);
```

Wartością zwracaną przez makrorozwinięcie jest poprzednia bitmapa, która była wybrana w kontekście (Wartość zwracana przez makrorozwinięcie wykorzystywana może być do przełączenia stanu kontekstu urządzenia pamięciowego w początkowe ustawienia). Parametry SelectBitmap() to uchwyt kontekstu pamięciowego i uchwyt bitmapy, którą chcemy wybrać w kontekście.

Jeśli istnieje w pamięci kontekst urządzenia pamięciowego z wybraną bitmapą oraz kontekst urządzenia na którym ma być wyświetlona bitmapa to kolejny etap wyświetlania obrazu polega na przesłaniu bitmapy pomiędzy kontekstami. Podstawową funkcją wykorzystywaną w tego typu operacjach jest BitBlt(). Prototyp funkcji ma następującą postać:

```
BOOL BitBlt
(
HDC hdcDest, // uchwyt kontekstu urządzenia docelowego
int nXDest,  // współrzędna x lewego górnego rogu obrazu docelowego
int nYDest,  // współrzędna y lewego górnego rogu obrazu docelowego
int nWidth,  // szerokość obrazu docelowego
int nHeight, // wysokość obrazu docelowego
HDC hdcSrc,  // uchwyt kontekstu urządzenia źródłowego
int nXSrc,   // współrzędna x lewego górnego rogu obrazu źródłowego
int nYSrc,   // współrzędna y lewego górnego rogu obrazu źródłowego
DWORD dwRop  // kod operacji rastrowej
);
```

Parametr hdcDest, w przypadku wyświetlania bitmapy w oknie, powinien być uchwytem kontekstu urządzenia danego okna. Parametry nXDest, nYDest, nWidth i nHeight określają wymiary prostokąta w którym ma być wyświetlana bitmapa (Jeśli chcemy wyświetlić całą bitmapę należy w tym miejscu podać wymiary bitmapy odczytane wcześniej z nagłówków pliku DIB). Parametr hdcSrc jest kontekstem urządzenia skąd chcemy przesłać bitmapę (W tym miejscu powinien znajdować się kontekst urządzenia pamięciowego w którym wybrana została bitmapa). Parametry nXSrc i nYSrc wyznaczają położenie lewego górnego rogu obrazu źródłowego. Od tego punktu obrazu rozpoczynać się będzie kopiowanie danych.

Ostatni parametr wywołania funkcji pozwala określić operację logiczną na pikselach obrazu, jaka wykonywana będzie w urządzeniu docelowym. W przypadku, gdy obraz ma być jedynie przekopiowany na urządzenie docelowe, należy wstawić jako ostatni parametr wywołania funkcji stałą **SRCCOPY** (predefiniowana stała Windows).

Po przesłaniu obrazu do okna (odpowiedniego kontekstu urządzenia) należy odłączyć bitmapę od pamięciowego kontekstu urządzenia (do tego celu wykorzystać można znane makrorozwinięcie SelectBitmap):

```
SelectBitmap(mMemDC,mHBitmapOld);
```

A następnie przy pomocy funkcji:

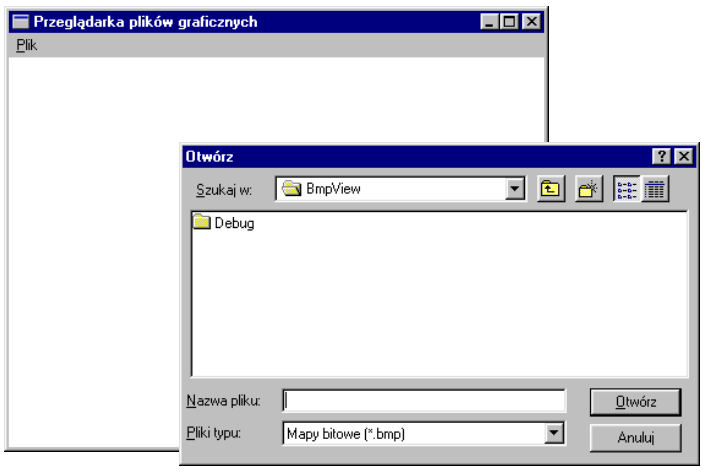
```
DeleteDC(mMemDC);
```

usunąć z pamięci kontekst urządzenia pamięciowego.

5. Program BmpView

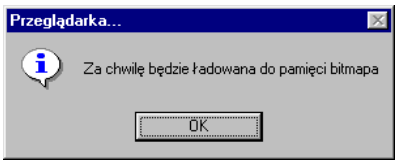
Program BmpView umieszczony jest na dyskietce dołączonej do opracowania. Jest on szkieletem aplikacji przeznaczonej do przeglądania plików graficznych. Ogólne założenia tworzenia aplikacji w języku C dla środowiska Windows omówione zostały w opracowaniu do ćwiczenia I z przedmiotu Grafika Komputerowa i Animacja. Poniżej omówione zostaną nowe elementy i modyfikacje wprowadzone do programu w celu realizacji postawionego przed aplikacją zadania.

Po uruchomieniu program BmpView zgłasza się w systemie jako okno jak na rysunku 5.1. Program wyposażony jest w menu. Dostępne są dwie opcje menu: Plik###Otwórz i Plik ###Koniec. Uruchomienie opcji Koniec powoduje zakończenie działania programu. Uruchomienie opcji Otwórz powoduje otworzenie standardowego okna Windows "Otwórz" (rys. 5.1).



Rys. 5.1 Okono programu BmpView.

Przy pomocy okna istnieje możliwość otwarcia plików o rozszerzeniu *.bmp. Próba otwarcia pliku kończy się wyświetleniem okna komunikatu jak na rysunku 5.2.



Rys. 5.2 Okno komunikatu programu BmpView

Do kodu źródłowego programu dołączono dodatkowe pliki nagłówkowe: *stdio.h*, *stdlib.h, windowsx.h*. Plik *stdio.h* zawiera definicje funkcji służących do operacji na plikach dyskowych, plik *stdlib.h* zawiera prototypy funkcji do dynamicznego zarządzania pamięcią komputera, zaś plik *windowsx.h* zawiera makra służące do przyłączania bitmap do kontekstu urządzenia.

Ciało funkcji obsługi komunikatów rozpoczyna się od deklaracji pomocniczych zmiennych lokalnych:

```
LRESULT CALLBACK WndProc(HWND hwnd,
                                    UINT iMsg,
                                    WPARAM wParam,
                                    LPARAM lParam)
{
      HDC HDCPaint;            // uchwyt obszaru kontekstu
                               // urządzenia klienta
      PAINTSTRUCT PaintStruct; // informacja o malowaniu

      // Zestaw danych do obsługi standardowego okna "Otwórz plik";
      char Filter[]="Mapy bitowe (*.bmp)\0*.bmp\0"; // filtr wyboru plików
      char FileName[256]={'\0'};                    // tablica na nazwę
                                                    // otworzonego pliku
      OPENFILENAME OFName;                          // struktura obsługująca
                                                    // okno "Otwórz plik"
      FILE *plik;                                   // uchwyt pliku

      // Zestaw danych, które mogą być przydatne do przetwarzania
      // i wczytywania pliku DIB
      static BITMAPFILEHEADER      *BmpFilHeadPtr;
      static BITMAPINFOHEADER      *BmpInfHeadPtr;
      static BITMAPINFO            *BmpInfPtr;
      static unsigned char         *BmpDataPtr;
      HDC mHDC;   // uchwyt do kontekstu urządzenia
      HDC mMemDC; // uchwyt do pamięciowego kontekstu urządzenia
      static int mBmp_Width;
      static int mBmp_Height;

      // Uchwyty do bitmap DDB:
      HBITMAP mHBitmapOld;
      static HBITMAP mHBmp=0;
      ...
```

Tablice tekstowe *Filter* i *FileName* przeznaczone są do obsługi standardowego okna dialogowego Windows "Otwórz". Pierwsza z nich zawiera tekst interpretowany przez okno jako maska wyświetlania plików. W tablicy *FileName* przechowywana będzie nazwa pliku wybranego przez użytkownika do otwarcia. Struktura *OFName* typu OPENFILENAME (predefiniowany typ Windows) służyć będzie do przechowywania pozostałych danych niezbędnych do obsługi okna "Otwórz". Wskaźnik na strukturę typu FILE (*plik*) w programie służy do identyfikacji pliku dyskowego. Następujące po zmiennej *plik* definicje wskaźników mogą posłużyć do identyfikacji poszczególnych porcji danych wczytywanych z pliku BMP.

Pomocne w pisaniu kodu programu mogą być także dwa uchwyty do kontekstu urządzenia (*mHDC* i *mMemDC*) oraz dwa uchwyty do obiektów typu bitmapa (*mHBitmapOld* i *mHBmp*).

Główna instrukcja obsługująca komunikaty systemowe ma postać:

```
// Obsługa komunikatów Windows:
switch(iMsg)
{

      case WM_COMMAND:
            switch(LOWORD(wParam))
            {
                  case ID_PLIK_OTWORZ:
                        // Załaduj do pamięci bitmapę
                        memset(&OFName,0,sizeof(OPENFILENAME));
                        OFName.lStructSize=sizeof(OPENFILENAME);
                        OFName.hwndOwner=hwnd;
                        OFName.lpstrFilter=Filter;
                        OFName.lpstrFile=FileName;
                        OFName.nMaxFile=sizeof(FileName);
                        OFName.Flags=OFN_FILEMUSTEXIST|OFN_HIDEREADONLY;
                        OFName.lpstrDefExt="bmp";
                        if(GetOpenFileName(&OFName)==FALSE)
                              return 0;

                        // Okno komunikatu:
                        MessageBox(hwnd,
                        "Za chwilę będzie ładowana do pamięci bitmapa",
                        "Przeglądarka...",
                        MB_OK|MB_ICONINFORMATION);
                              if(mHBmp)
                        {     DeleteObject(mHBmp);
                              mHBmp=0;
                        }
                  // W tym miejscu programu należy:
                  // 1. Otworzyć plik z danymi
                  //    (nazwa pliku znajduje się w tablicy FileName)
                  // 2. Załadować nagłówek pliku DIB do pamięci
                  // 3. Na podstawie danych z nagłówka zarezerwować
                  //    odpowiedni obszar pamięci i wczytać do niego
                  //    bitmapę
                  // 4. Np. przy pomocy funkcji CreateDIBitmap()
                  //    utworzyć DDB na podstawie DIB
                  //    (uchwyt kontekstu okna można uzyskać przy pomocy
                  //    funkcji GetDC(), a zwolnić go przy pomocy
                  //      funkcji ReleaseDC())
                  // 5. Zamknąć otwarty plik
                  // 6. Aby zmusić Windows do "odmalowania" okna można
                  //    wywołać funkcję: InvalidateRect(hwnd,NULL,TRUE);

                              return 0;

                  case ID_PLIK_KONIEC:
                        DestroyWindow(hwnd);
                        return 0;
            }
            return 0;

            case WM_DESTROY:
                  // Jeśli istnieje usuń obiekt DDB z pamięci
                  if(mHBmp)DeleteObject(mHBmp);
                  PostQuitMessage(0);
                  return 0;
```

```
            case WM_PAINT:

            // inicjalizuje malowanie i utrzymuje kontekst urządzenia:
                  HDCPaint=BeginPaint(hwnd,&PaintStruct);

            // W tym miejscu programu należy:
            // Jeśli utworzony został obiekt DDB (to znaczy, jeśli
            // uchwyt bitmapy mHBmp jest różny od zera) to:
            // 1. Utwórz pamięciowy kontekst urządzenia na podstawie
            //    kontekstu urządzenia związanego z oknem na którym ma być
            //    wyświetlona bitmapa
            //      (funkcja: CreateCopatibleDC())
            // 2. Przyłącz do pamięciowego kontekstu urządzenia bitmapę
            //    DDB
            //    (makro SelectBitmap())
            // 3. Dokonaj przekopiowania bitmapy z pamięciowego kontekstu
            //    urządzenia do kontekstu urządzenia związanego z oknem
            //    , na którym ma być wyświetlona bitmapa
            //    (funkcja BitBlt())
            // 4. Odłącz od pamięciowego kontekstu urządzenia bitmapę
            //    DDB
            //    (makro SelectBitmap())
            // 5. Usuń pamięciowy kontekst urządzenia
            //    (funkcja DeleteDC())

            // przerwanie malowania i uwolnienie kontekstu urządzenia:
                  EndPaint(hwnd, &PaintStruct);
                  return 0;
            default:
                  return DefWindowProc(hwnd,iMsg,wParam,lParam);

      }
}
```

Program obsługuje trzy komunikaty systemowe: WM_DESTROY, WM_COMMAND i WM_PAINT. Podczas obsługi komunikatu WM_DESTROY usuwana jest z pamięci komputera, jeśli istnieje, załadowana bitmapa a następnie wywoływana jest funkcja PostQuitMessage().

W obsłudze komunikatu WM_COMMAND, jeśli użytkownik wybrał opcję menu Koniec, następuje wywołanie funkcji PostQuitMessage(). Natomiast jeśli użytkownik wybrał opcję Otwórz, program wypełnia odpowiednimi wartościami pola struktury typu OPENFILENAME a następnie, przy pomocy funkcji GetOpenFileName(), uruchamia standardowe okno Windows "Otwórz" i pobiera od systemu nazwę pliku, który użytkownik chce przetwarzać. W tym miejscu obsługi komunikatu wyświetlane jest okno komunikatu (funkcja MessageBox()) informujące, że w następnej kolejności odbywać się będzie w programie proces otwarcia pliku z danymi. Umieszczony w pliku komentarz sugeruje w jaki sposób uzupełnić obsługę komunikatu, aby wczytać do pamięci wybrany plik z danymi, a następnie przekształcić go w bitmapę DDB (typ Windows: BITMAP) przystosowaną do wyświetlania w głównym oknie programu. W kodzie programu przeznaczonego dla systemu Windows z powodzeniem można stosować (a nawet jest to zalecane przez autorów wielu publikacji o programowaniu) znane z bibliotek standardowych ANSI C funkcje służące do otwierania plików tekstowych i do alokacji pamięci takie jak: fopen(), fclose(), fseek(), fread(), malloc(), calloc(), free() (Między innymi dlatego do pliku tekstowego programu dołączono pliki nagłówkowe stdio.h i stdlib.h). Sugeruje się aby po utworzeniu w pamięci komputera bitmapy DDB identyfikowanej przez uchwyt typu HBITMAP, na zakończenie obsługi komunikatu WM_COMMAND dla opcji menu Plik###Otwórz wywołać funkcję Windows:

```
InvalidateRect(hwnd,NULL,TRUE);
```

Wywołanie tej funkcji z parametrami jak powyżej, gdzie *hwnd* powinno być uchwytem okna programu powoduje automatyczne przesłanie do programu komunikatu WM_PAINT.

Obsługa komunikatu WM_PAINT w programie BmpView ogranicza się do wywołania funkcji BeginPaint() i EndPaint() z odpowiednimi parametrami (**Uwaga: Jeśli tworzymy program dla systemu Windows i obsługujemy komunikat WM_PAINT, to obowiązkowe jest wywołanie wtedy funkcji BeginPaint() a po niej EndPaint()**). Podobnie jak w przypadku obsługi komunikatu WM_COMMAND w liniach programu znajdują się sugestie dotyczące sposobu uzupełnienia kodu. Obsługa komunikatu powinna wyświetlać na ekranie utworzoną bitmapę, jeśli w pamięci komputera taki obiekt istnieje.

Program BmpView sugeruje pewien sposób postępowania w przypadku tworzenia oprogramowania przeznaczonego do wyświetlania plików graficznych. Rozsądnym wydaje się umieszczenie funkcji dekodujących zawartość pliku graficznego w obsłudze komunikatu służącego do otwarcia pliku. Wskazane także jest umieszczenie funkcji bezpośrednio wyświetlających obraz w oknie w obsłudze komunikatu odpowiedzialnego za prawidłowe wyświetlanie zawartości okna programu. Program BmpView w prosty sposób można rozbudować uzupełniając go o możliwość wyświetlania zawartości plików o innych formatach niż bitmapa (tga, tiff, gif, pcx itp.). Najprostszym sposobem wyświetlania tego rodzaju plików wydaje się wstępne przekształcenie ich do formatu DIB, po czym posłużenie się gotowym kodem wyświetlającym zawartość pliku DIB naszkicowanym w opracowaniu.

Bibliografia:

[1] *Programowanie Windows 95,* C. Petzolt, 1997 Oficyna Wydawnicza READ ME
[2] *Grafika i Animacja w Windows*, N. Barkakati, 1994 INTERSOFTLAND
[3] *Win32 Programming*, B. E. Rector, J.M. Newcomer, 1997 Addison Wesley Longman, Inc
[4] *Windows Animation Programming*, M. J. Young, 1994 AP PROFESSIONAL
[5] Opracowania do wykładu *Grafika Komputerowa i Animacja*, R. Leniowski

Dodatek A:
Pełny kod źródłowy programu BmpView.

```
#define STRICT    // żądanie ścisłego przestrzegania
                        // zgodności typów w programie
#include <windows.h>  // plik nagłówkowy funkcji API
                              // Windows
#include <windowsx.h> // makro SelectBitmap()
#include <stdio.h>    // operacje na plikach
#include <stdlib.h>   // alokacja pamięci
#include "resource.h" // zasoby aplikacji - menu


LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
                                    // Prototyp głównej funkcji
                                    // obsługi głównego okna programu

// Główna funkcja programu:
int WINAPI WinMain(     HINSTANCE hInstance,    // uchwyt instancji
                                                // programu
                        HINSTANCE hPrevInstance,// 0
                        PSTR szCmdLine,              // linia komend
```

```
                        int iCmdShow)                // początkowy stan
                                                     // programu
{
      static char szAppName[]="BmpView"; // unikalna nazwa aplikacji
      HWND hwnd;         // uchwyt głównego okna
      MSG  msg;          // struktura przechowująca messages
      WNDCLASS wndclass; // struktura do konstruowania klasy okna

      // Tworzenie klasy głównego okna:
      wndclass.style=CS_HREDRAW|CS_VREDRAW;
      wndclass.lpfnWndProc=WndProc;
      wndclass.cbClsExtra=0;
      wndclass.cbWndExtra=0;
      wndclass.hInstance=hInstance;
      wndclass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
      wndclass.hCursor=LoadCursor(NULL,IDC_ARROW);
      wndclass.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
      wndclass.lpszMenuName=MAKEINTRESOURCE(IDR_MENU1);
      wndclass.lpszClassName=szAppName;

      RegisterClass(&wndclass);

      hwnd=CreateWindow(
                                  szAppName,
                                  "Przeglądarka plików graficznych",
                                  WS_OVERLAPPEDWINDOW,
                                  CW_USEDEFAULT,
                                  CW_USEDEFAULT,
                                  CW_USEDEFAULT,
                                  CW_USEDEFAULT,
                                  NULL,
                                  NULL,
                                  hInstance,
                                  NULL
                               );
      ShowWindow(hwnd,iCmdShow);
      UpdateWindow(hwnd);

      while(GetMessage(&msg,NULL,0,0))
      {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
      }
      return msg.wParam;
}

LRESULT CALLBACK WndProc(HWND hwnd,
                                   UINT iMsg,
                                   WPARAM wParam,
                                   LPARAM lParam)
{
HDC HDCPaint;                // uchwyt obszaru kontekstu
                             // urządzenia klienta
PAINTSTRUCT PaintStruct; // informacja o malowaniu

// Zestaw danych do obsługi standardowego okna "Otwórz plik";
char Filter[]="Mapy bitowe (*.bmp)\0*.bmp\0"; // filtr wyboru plików
char FileName[256]={'\0'};                    // tablica na nazwę
                                              // otworzonego pliku
OPENFILENAME OFName;         // struktura obsługująca
                             // okno "Otwórz plik"
FILE *plik;                  // uchwyt pliku

// Zestaw danych, które mogą być przydatne do przetwarzania
// i wczytywania pliku DIB
```

```
static BITMAPFILEHEADER *BmpFilHeadPtr;
static BITMAPINFOHEADER *BmpInfHeadPtr;
static BITMAPINFO       *BmpInfPtr;
static      unsigned char    *BmpDataPtr;
HDC mHDC;   // uchwyt do kontekstu urządzenia
HDC mMemDC; // uchwyt do pamięciowego kontekstu urządzenia
static int mBmp_Width;
static int mBmp_Height;

// Uchwyty do bitmap DDB:
HBITMAP mHBitmapOld;
static HBITMAP mHBmp=0;

// Obsługa komunikatów Windows:
switch(iMsg)
{

      case WM_COMMAND:
            switch(LOWORD(wParam))
            {
                  case ID_PLIK_OTWORZ:
                        // Załaduj do pamięci bitmapę
                        memset(&OFName,0,sizeof(OPENFILENAME));
                        OFName.lStructSize=sizeof(OPENFILENAME);
                        OFName.hwndOwner=hwnd;
                        OFName.lpstrFilter=Filter;
                        OFName.lpstrFile=FileName;
                        OFName.nMaxFile=sizeof(FileName);
                        OFName.Flags=OFN_FILEMUSTEXIST|OFN_HIDEREADONLY;
                        OFName.lpstrDefExt="bmp";
                        if(GetOpenFileName(&OFName)==FALSE)
                              return 0;

                        // Okno komunikatu:
                        MessageBox(hwnd,
                        "Za chwilę będzie ładowana do pamięci bitmapa",
                        "Przeglądarka...",
                        MB_OK|MB_ICONINFORMATION);

                        if(mHBmp)
                        {     DeleteObject(mHBmp);
                              mHBmp=0;
                        }

                  // W tym miejscu programu należy:
                  // 1. Otworzyć plik z danymi
                  //    (nazwa pliku znajduje się w tablicy FileName)
                  // 2. Załadować nagłówek pliku DIB do pamięci
                  // 3. Na podstawie danych z nagłówka zarezerwować
                  //    odpowiedni obszar pamięci i wczytać do niego
                  //    bitmapę
                  // 4. Np. przy pomocy funkcji CreateDIBitmap()
                  //    utworzyć DDB na podstawie DIB
                  //    (uchwyt kontekstu okna można uzyskać przy pomocy
                  //    funkcji GetDC(), a zwolnić go przy pomocy
                  //      funkcji ReleaseDC())
                  // 5. Zamknąć otwarty plik
                  // 6. Aby zmusić Windows do "odmalowania" okna można
                  //    wywołać funkcję: InvalidateRect(hwnd,NULL,TRUE);

                        return 0;

                  case ID_PLIK_KONIEC:
                        DestroyWindow(hwnd);
                        return 0;
```

```
            }
            return 0;

      case WM_DESTROY:
            // Jeśli istnieje usuń obiekt DDB z pamięci
            if(mHBmp)DeleteObject(mHBmp);
            PostQuitMessage(0);
            return 0;

      case WM_PAINT:

            // inicjalizuje malowanie i utrzymuje kontekst urządzenia:
            HDCPaint=BeginPaint(hwnd,&PaintStruct);

      // W tym miejscu programu należy:
      // Jeśli utworzony został obiekt DDB (to znaczy, jeśli
      // uchwyt bitmapy mHBmp jest różny od zera) to:
      // 1. Utwórz pamięciowy kontekst urządzenia na podstawie
      //    kontekstu urządzenia związanego z oknem na którym ma być
      //    wyświetlona bitmapa
      //      (funkcja: CreateCopatibleDC())
      // 2. Przyłącz do pamięciowego kontekstu urządzenia bitmapę
      //    DDB
      //    (makro SelectBitmap())
      // 3. Dokonaj przekopiowania bitmapy z pamięciowego kontekstu
      //    urządzenia do kontekstu urządzenia związanego z oknem
      //    , na którym ma być wyświetlona bitmapa
      //    (funkcja BitBlt())
      // 4. Odłącz od pamięciowego kontekstu urządzenia bitmapę
      //    DDB
      //    (makro SelectBitmap())
      // 5. Usuń pamięciowy kontekst urządzenia
      //    (funkcja DeleteDC())

      // przerwanie malowania i uwolnienie kontekstu urządzenia:
            EndPaint(hwnd, &PaintStruct);
            return 0;
      default:
            return DefWindowProc(hwnd,iMsg,wParam,lParam);

}
}
```