Bits mit Image

Das IMG-Grafikformat entschlüsselt

Neben dem PCX-Grafikformat ist das IMG-Format von GEM weit verbreitet. Obwohl der überwältigende Erfolg von Windows 3.0 das Ende von GEM einzuläuten scheint, gibt es nach wie vor viele Grafikprogramme, die IMG-Dateien erzeugen. Damit Sie Ihre wertvollen Bilddaten auch weiterhin verwenden können, legen wir das IMG-Format offen.

Der generelle Aufbau einer IMG-Datei unterscheidet sich nur in wenigen Punkten von anderen Grafikformaten. Zu Beginn der Datei steht der Header mit den Eckdaten der Grafik. Alle Werte sind mit zwei Byte codiert, wobei aber zu beachten ist, daß das Low- und High-Byte des Wortes jeweils vertauscht ist. Es steht immer zuerst das High-Byte und erst danach folgt das Low-Byte.
Der Header besteht derzeit aus 8 Worten, also 16 Byte, deren Bedeutung in der Tabelle erläutert ist. Da die Pixelgröße in Mikrometer gespeichert ist, lassen sich IMG-Bilder mit den richtigen Proportionen auf unterschiedlichen Geräten ausgeben.
Der siebte Eintrag im Header (Wort 6) gibt die Länge einer Zeile an. Er wird für jeweils eine Farbebene angegeben. Im Gegensatz zu den meisten anderen Grafikformaten wird die Höhe des Bildes nicht im Header gespeichert. Daraus ergeben sich eine Reihe von Problemen bei der korrekten Skalierung des Bildes.

Header einer IMG-Datei

Wort	Bedeutung
0	Version der Datei (in der Regel 1)
1	Kopflänge in Worten
2	Anzahl der Farbebenen
3	Länge eines Musters
4	Pixelbreite in Mikrometer
5	Pixelhöhe in Mikrometer
6	Länge einer Zeile in Pixel = Bits
7	Anzahl der Elemente in der Datei

Wenn man ein und dasselbe Bild sowohl im IMG als auch im PCX-Format speichert, fällt sofort auf, daß die IMG-Datei in der Regel kleiner ist als die entsprechende PCX-Datei. Und das hat seinen Grund. Während PCX nur eine Lauflängen-Komprimierung kennt, werden in IMG-Dateien gleich vier verschiedene Komprimiermethoden angewandt: vertikale Wiederholung, Solid Run, Pattern Run und Bit String.
Bei der einfachsten Methode, der vertikalen Wiederholung, werden einfach identische Bildschirmzeilen zusammengefaßt. Dazu benötigt GEM insgesamt vier Byte. Die ersten drei Byte dienen als Flag zur Erkennung des Codes und haben immer die Werte 00h, 00h und FFh. Im vierten Byte steht schließlich die Anzahl der identischen Bildschirmzeilen. Damit lassen sich maximal 255 Zeilen in 4 Byte unterbringen, aber diese enorme Komprimierung bringt auch Nachteile mit sich. Da die maximale Anzahl der Bildschirmzeilen nicht bekannt ist, muß die Ausgaberoutine immer prüfen, ob nicht die letzte Bildschirmzeile erreicht worden ist. Anderenfalls könnte es zu unangenehmen Überraschungen kommen.
Innerhalb einer Zeile kennt GEM drei Methoden zum Codieren der Farbinformationen. Jede Zeile besteht aus einer (CGA und Hercules) oder mehreren Farbebenen (EGA und VGA). Alle Ebenen einer Zeile werden nacheinander codiert, wobei GEM versucht, die einzelnen Bits bestmöglich zu packen.
Wenn mehrere aufeinanderfolgende Bytes entweder nur den Wert 00h oder FFh haben, genügt es, die Anzahl der sich wiederholenden Bytes zu codieren. Diese Lauflängenkomprimierung wird Solid Run genannt. In nur einem Byte lassen sich bei Schwarz-weiß-Darstellungen bis zu 127 Byte codieren. Haben die aufeinanderfolgenden Bytes den Wert FFh (schwarz), wird das das erste Bit des codierten Byte auf 1 gesetzt. Die restlichen 7 Bit des codierten Byte repräsentieren die Länge der aufeinanderfolgenden Bytes mit dem Wert FFh. Das gleiche gilt für ein Feld von aufeinanderfolgen 00h-Bytes (weiß). In diesem Fall wird das erste Bit des codierten Byte auf 0 gesetzt.
Wenn ein Bild aus Mustern zusammengesetzt ist, lassen sich diese zu einem Pattern Run zusammenfassen. Allerdings offenbart sich dabei eine kleine Schwäche des Formats. Die Länge eines Musters ergibt sich aus Wort 3 des Headers und gilt für alle Muster der Datei. Ein Muster muß also immer dieselbe Länge haben. In der Regel wird der Wert 2 (Byte) verwendet.

Im Gegensatz zum Solid Run muß dieses Format über ein Flag zur Kennzeichnung der Komprimiermethode verfügen. GEM verwendet dazu den Wert 00h, dem die Anzahl der Muster und das Muster selbst folgen.
Handelt es sich um ein einmaliges Muster, so muß auch GEM kapitulieren. Solche Bytefolgen können lediglich unkomprimiert als sogenannter Bit String abgelegt werden.
Als Erkennungsflag verwendet GEM den Wert 80h, dem die Anzahl der Bytes und die Werte selbst folgen. Betrachtet man sich diese Codes, so ergibt sich eine zwangsläufige Reihenfolge bei der Decodierung. Denn die Solid-Run-Methode verfügt über kein Erkennungsflag. Deshalb müssen zuerst alle übrigen Flags abgefragt werden. Für die Praxis bedeutet dies:

- Test auf vertikale Wiederholung (mehrere identische Zeilen)
- Test auf Pattern Run und Bit String
- Alles übrige ist ein Solid Run

Im Gegensatz zum PCX-Format verfügen IMG-Dateien über keinerlei Farbpaletten-Informationen. Die Farben sind so zu setzen, wie sie im Speicher stehen. Diese Schwäche ist ein weiterer Grund dafür, daß das Format immer seltener eingesetzt wird. Allerdings gibt es noch viele Archive mit IMG-Dateien, so daß man nur wissen muß wie diese Dateien codiert sind.
Im folgenden stellen wir daher ein Programm (Listing) vor, das in der Lage ist, IMG-Dateien auf CGA-, EGA-, VGA- und Hercules-Bildschirmen darzustellen. Sie können es mit Turbo oder Quick Pascal kompilieren. Sie starten das Programm mit dem Befehl SHOWIMG *Dateiname*.
Den Namen der IMG-Datei können Sie mit oder ohne Dateiendung eingeben. Das Lesen von IMG-Dateien und das Darstellen der Bilder ist im allgemeinen recht einfach. Etwas schwieriger ist das Drucken und das Konvertieren zu einem anderen Format. Es ergeben sich nämlich eine Reihe von Problemen, die durch die mögliche vertikale Wiederholung ausgelöst werden. Es führt letztendlich kein Weg daran vorbei, einen temporären Puffer zu verwalten, der in der Lage ist, bis zu 255 Zeilen zu halten. In diesem Fall müssen Sie eine virtuelle Speicherverwaltung in Ihr Programm einfügen.

Dietmar Bückart

Pascal-Programm zur Bildschirmausgabe von IMG-Dateien

```
{$R-,S-,I-}
USES CRT, DOS, GRAPH;

VAR
  IMGFile      : FILE OF BYTE;
  IMGName      : STRING[79];
  Pattern      : ARRAY [1..16] OF BYTE;
  Muster       : ARRAY [0..4095] OF BYTE;
  MusterLen,
  Planes,
  Breite,
  Links,
  MaxBits,
  ActX,ActY    : WORD;
  gd, gm       : INTEGER;

FUNCTION GetIMGHeader : INTEGER;
VAR
  I, J    : BYTE;
  HeadLen : WORD;
BEGIN
  Assign (IMGFile, IMGName);
  Reset (IMGFile);
  DOSERROR := IOResult;
  IF DOSERROR <> 0 THEN
  BEGIN
    GetIMGHeader := DOSERROR;
    EXIT;
  END;
  Seek (IMGFile, 2);             { die Versionsnummer überspringen }
  Read (IMGFile, I, J);          { die Länge des Vorspanns lesen   }
  HeadLen := I*256+J;            { Bytes vertauschen!              }
  Read (IMGFile, I, J);          { Anzahl der Planes lesen         }
  Planes := PRED(I*256+J);       { Bytes vertauschen!              }
  Read (IMGFile, I, J);          { Musterlänge lesen               }
  MusterLen := I*256+J;          { Bytes vertauschen!              }
  Seek (IMGFile, 12);            { Erzeuger-Info überspringen      }
  Read (IMGFile, I, J);          { Bildbreite lesen                }
  Breite := I*256+J;             { Bytes vertauschen!              }
  Links := ((640-Breite) DIV 2) AND $FFF8;
  Seek (IMGFile, HeadLen*2);
  GetIMGHeader := 0;
END;

PROCEDURE SetEGAWritePlane(Nr : BYTE);
BEGIN
  PORT[$3C4] := 2;
  PORT[$3C5] := 1 SHL Nr;
END;

PROCEDURE Anzeigen (v,f: BYTE; Count: WORD);
VAR
  I       : BYTE;
  L, Adrs : WORD;
BEGIN
  L := ActY;                     { Bildschirmzeile übernehmen }

  IF (gd=3) OR (gd=9)            { richtige EGA-Plane setzen  }
    THEN SetEGAWritePlane(f);    { falls EGA oder VGA         }

  IF (ActX+Count*8) > MaxBits    { rechten Rand beachten!     }
    THEN Count := (MaxBits DIV 8) - (ActX DIV 8);

  FOR I := 0 TO PRED(v) DO
  BEGIN
    CASE gd OF
      3, 9 : BEGIN    { VGA      }
               Adrs := L*80 + (ActX SHR 3);
               Move (Muster, Mem[$A000:Adrs], Count);
```

```
              END;
      7   : BEGIN     { Hercules }
              Adrs := (L AND 3) SHL 13 + 90*(L SHR 2) + (ActX SHR 3);
              Move (Muster, Mem[$B000:Adrs], Count);
            END;
      1   : BEGIN     { CGA      }
              Adrs := (L AND 1) SHL 13 + 80*(L SHR 1) + (ActX SHR 3);
              Move (Muster, Mem[$B800:Adrs], Count);
            END;
    END;

    INC(L);
    IF L > GetMaxY THEN EXIT;
  END;
  INC(ActX, Count*8);
END;

PROCEDURE GetScanLine;
VAR
  I, J, C,
  Farbe,
  VertRep :  BYTE;
  Count   :  WORD;
BEGIN
  Read (IMGFile, I, J, C);
  IF (I=0) AND (J=0) AND (C=$FF)
    THEN Read (IMGFile, VertRep)
    ELSE IF (I=0) AND (J=0)
           THEN EXIT
           ELSE BEGIN
                  Seek (IMGFile, FilePos(IMGFile)-3);
                  VertRep := 1;
                END;
  FOR Farbe := 0 TO Planes DO
  BEGIN
    ActX := Links;
    WHILE (ActX < Breite+Links) DO
    BEGIN
      Read (IMGFile, I);
      CASE I OF
        $00: BEGIN       { Pattern Run }
               Read (IMGFile, I);
               FOR J := 1 TO MusterLen DO
                 Read (IMGFile, Pattern[J]);

               FOR J := 0 TO 1-1 DO
                 Move (Pattern, Muster[J*MusterLen], MusterLen);

               Count := I*MusterLen;
             END;
        $80: BEGIN        { BitString  }
               Read (IMGFile, I);
               FOR J := 0 TO PRED(I) DO
                 Read (IMGFile, Muster[J]);
                 Count := I;
               END;
        ELSE BEGIN
               Count := I AND $7F;
               IF (I AND $80)=0
                 THEN FillChar (Muster, Count, 0)
                 ELSE FillChar (Muster, Count, $FF);
             END;
      END;

      IF ActY <= GetMaxY
        THEN Anzeigen(VertRep, Farbe, Count);
    END;
  END;

  INC(ActY, VertRep);
```

```
END;

PROCEDURE EncodeIMG;
BEGIN
  WHILE   (NOT Eof(IMGFile))    { Dateiende noch nicht erreicht     }
      AND (NOT KeyPressed)      { und keine Taste gedrückt          }
      AND (ActY < GetMaxY)      { und Bildschirmende nicht erreicht }
    Do GetScanLine;             { Zeile bearbeiten                  }
END;

BEGIN
  IF (ParamCount = 0) THEN
  BEGIN
    Writeln(^J^M'Sie haben keinen Dateinamen angegeben.'^J^M);
    HALT;
  END ELSE IMGName := ParamStr(1);

  IF Pos('.', IMGName)=0
    THEN IMGName := IMGName+'.img';

  IF (GetIMGHeader <> 0) THEN
  BEGIN
    Writeln(^J^M'Datei ', IMGName, ' nichtgefunden!'^J^M);
    HALT;
  END;

  ActX := 0;
  ActY := 0;

  DetectGraph(gd, gm);
  InitGraph(gd, gm, '');

  CASE gd OF                    { für Test im Anzeigen() }
    1, 3, 9 : MaxBits := 640;
    7       : MaxBits := 720;
  END;

  EncodeIMG;

  REPEAT UNTIL KeyPressed;

  Close (IMGFile);

  RestoreCrtMode;
END.
```

aus mc 01/91 – Seite 116-119