Grafik mit Format

Aufbau von PCX-Dateien

PCX ist ein weitverbreitetes Grafikformat. Fast alle Grafikprogramme kommen damit klar. Weil der Aufbau von PCX-Dateien vollständig offengelegt worden ist, können auch selbstgeschriebene Programme die Bildpunkte nach Herzenslust tanzen lassen. Am Beispiel eines Quick-Pascal-Programms zeigen wir, wie es gemacht wird.

Das PCX-Format wurde Anfang der achtziger Jahre von ZSoft Corporation zur Speicherung der mit PC-Paintbrush erzeugten Grafiken entwickelt. PCX wird von nahezu allen professionellen Systemen unterstützt. Der Grund dafür liegt neben der weltweiten Verbreitung des Paintbrush-Malprogramms in der vollständig veröffentlichten Dokumentation des Dateiformats. Die enge Kopplung des PCX-Formats an Paintbrush bringt auch einige Nachteile mit sich. Als das Format entwickelt wurde, hatte niemand damit gerechnet, daß PCX-Bilder auf verschiedenen Grafikkarten verarbeitet werden müssen. Es war vorgesehen, PCX-Bilder nur von der Hardware zu laden, auf der sie erzeugt wurden. Inzwischen sind PCX-Bilder grundsätzlich auf jeder Hardware darstellbar.
Jede PCX-Datei beginnt mit einem 128 Byte langen Header, von dem derzeit aber nur 70 verwendet werden (Tabelle: Header einer PCX-Datei).
Die Koordinaten des gespeicherten Bildes können von der Auflösung des Erzeugers abweichen. Spätesten dann ist für den Programmierer Panik angesagt. Während sich vertikale Abweichungen noch durch einfaches Verdoppeln oder Ignorieren von Zeilen korrigieren lassen, geben horizontale Abweichungen den Startschuß für umfangreiche Bitmanipulationen. Denn benachbarte Bits stehen für benachbarte Bildpunkte, die gemäß dem ursprünglichen Formatdesign einfach in den Bildschirmspeicher kopiert werden sollten. Findet eine Skalierung statt, müssen alle nachfolgenden Bits verschoben und zu neuen Bytes komprimiert werden, bevor das Byte in den Bildschirmspeicher kopiert werden kann. Spätestens hier wird klar, warum selbst Paintbrush für Windows gelegentlich eine ganze Weile benötigt, bis es ein PCX-Bild lädt und anzeigt.
Die PCX-Versionen 2.8 und 3.0 sind mit einer zusätzlichen Farbpalette ausgestattet (Versionsbyte = 2 bzw. 5) und offenbaren eine weitere Eigenheit des PCX-Formats. Da bei der Entwicklung des Formats an VGA-Karten mit 256 Farben noch gar nicht zu denken war, wurden lediglich 48 Byte für 16 Rot-, Grün- und Blauanteile eingeplant. Als dieser Speicherplatz nicht mehr ausreichte, wurde einfach an das Ende der PCX-Datei (hinter den Bilddaten) eine Zusatzpalette mit 768 Byte für die Farbwerte von VGA-Karten mit 256 Farben gepackt. Auf EGA-Grafikkarten müssen die Farbinformationen interpretiert werden. Denn in der PCX-Datei sind die Farbwerte im Bereich von 0-255 notiert. Die EGA-Karte kann aber nur Werte von 0-63 verarbeiten, dies jedoch in vier Stufen. Der korrekte Farbwert ergibt sich somit durch Division mit 64 (Tabelle 2).
Die Bilddaten selbst sind zeilenweise komprimiert abgelegt. Das PCX-Format nutzt dazu die Runlength-Komprimierung, die speziell für Byte-Wiederholungen geeignet ist. Denn Grafiken haben oft große einfarbige Flächen, die durch gleichlautende Bytes beschrieben werden. Das PCX-Format nutzt diese Tatsache und packt sich wiederholende Bytes in zwei Bytes. Eine Komprimierung lohnt sich erst, wenn mindestens drei aufeinanderfolgende Bytes identisch sind. Aus diesem Grund kommen in PCX-Dateien sowohl komprimierte als auch „nackte" Datenbytes vor. Erkennungszeichen für die Komprimierung sind die beiden höchstwertigen Bits eines Bytes, Bit 6 und 7. Haben beide Bits den Wert 1, dann beschreiben die Bit 0 bis 5 die Anzahl der Wiederholungen und das darauffolgende Byte den Farbwert.

Im Pseudocode lautet dieser Algorithmus:

```
temp := (gelesenes Byte);
IF (temp AND $C0) = $C0 THEN
BEGIN
  count := temp AND $3F;
  temp := (nächstes Byte);
END;
```

Da für die Kodierung der Anzahl der Wiederholungen nur 6 Bit zur Verfügung stehen, ist der höchste mögliche Wert 63. Dies ist besonders beim Erzeugen von PCX-Dateien zu beachten.
Die Komprimierung arbeitet zeilenübergreifend. Das heißt, sowohl ein Wechsel der Farbebene, als auch ein Wechsel der Bildschirmzeile sind innerhalb einer Wiederholung möglich. Das lesende Programm muß selbst herausfinden, wie die Daten zu interpretieren sind. Als Steuerung für die Anzeige der PCX-Bilder kann allerdings der Wert der Header-Variable BytePerLine (Byte 66 + 67) verwendet werden.

Diese Variable enthält immer die Anzahl der Bytes, die für eine komplette Farbebene einer Bildschirmzeile benötigt werden. An dieser Stelle ist jedoch große Vorsicht angebracht, da PCX-Bilder größer als der Bildschirm werden können. Es ist darauf zu achten, daß nur so viele Bytes in den Bildschirmspeicher kopiert werden wie tatsächlich darstellbar sind.
Die abgedruckten Listings enthalten die Grundroutinen zum Speichern und Lesen von Bildern im PCX-Format. Da die Ablage hardwarenah erfolgt, ist eine genaue Kenntnis der jeweiligen Grafikkarte nötig. Die vorgestellten Routinen, die sowohl unter Quick Pascal als auch unter Turbo Pascal lauffähig sind, dienen als Vorlage für andere Grafikkarten-Exoten.
Mit der Funktion SavePCX lassen sich Bilder von CGA-, EGA-, VGA-, MCGA- und Hercules-Grafikkarten speichern. Das Einlesen ist etwas komplizierter. Eventuell sind einige Bitoperationen nötig, um die Grafiken auf das rechte Format zu bringen. Die Prozedur PutPCX unterstützt daher nur die Original CGA-, EGA-, VGA- und Hercules-Grafikformate ohne Skalierung. Gestartet wird das Programm mit dem Befehl SHOWPCX *Dateiname*.

Dietmar Bückart

Header einer PCX-Datei		
Byte	**Bezeichner**	**Kommentar**
0	Creator	immer 10 = ZSoft
1	Version	PCX-Version: 0 = Version 2.5 2 = Version 2.8 mit Farbpalette 3 = Version 2.8 ohne Farbpalette 5 = Version 3.0 mit Farbpalette
2	Encoding	1 = Runlength-Kodierung
3	Bits	Anzahl der Bits pro Pixel und Farbplane; CGA LoRes + VGA mit 256 Farben: 2 Bit; alle anderen: 1 Bit pro Pixel
4	xmin, ymin xmax, ymax	logische Koordinaten des Bildes
12	HRes	horizontale Auflösung des Erzeugers
14	VRes	vertikale Auflösung des Erzeugers
16	Palette	Farbpalette 16 × 3 Bytes = 16 × (R, G, B)
64	VMode	nur für interne Paintbrush-Verwendung
65	Planes	Anzahl der Farbebenen EGA/VGA : 4 CGA/Hercules : 1
66	BytePerLine	Anzahl der gespeicherten Bytes pro Scanzeile, wobei nur eine Farbplane gerechnet wird
68	Paletteinfo	Interpretation der Farben: 1 = Farbe (auch schwarzweiß!) 2 = Graustufen
70	dummy	Füller auf 128 Bytes

Farbumsetzung auf der EGA	
Farbwert	EGA-Stufe
0 – 63	0
64 – 127	1
128 – 191	2
192 – 255	3

Listing 1. PCX.PAS: Unit für das Programm SHOWPCX.PAS

```
{ ********************************************************************* }
{ *  UNIT:   PCX.PAS  -  Routinen zum Bearbeiten von PCX-Dateien   * }
{ ********************************************************************* }
{$R-,S-,I-}

UNIT PCX;

INTERFACE

USES DOS;

CONST
  ActivePage    : WORD = 0;
  MAX_LINEBYTES        = 1023;
  HercBase             = $B000;
  EgaBase              = $A000;
  CgaBase              = $B800;


TYPE
  PlaneType = ARRAY[0..MAX_LINEBYTES] OF BYTE;
  plane     = ^Planetype;
  ScanLine  = ARRAY[0..3] OF plane;

VAR
  z : ScanLine;

TYPE
  PCX_HEADER = RECORD  { Header einer PCX-Datei:              }
    Creator    : BYTE; { Immer 10 für ZSoft                   }
    Version    : BYTE; { PCX-Version:                         }
                       { 0 = Version 2.5 ohne Palette-Info    }
                       { 2 = Version 2.8 mit Palette-Info     }
                       {     oder Version 3.0 ohne Palette    }
                       { 3 = Version 2.8/3.0 ohne Palette-Info}
                       { 5 = Version 3.0 mit Palette-Info     }
    Encoding   : BYTE; { 1 = Run-Length-Encoded               }
    Bits       : BYTE; { Anzahl der Pixel pro Bit; i.d.R. 1;  }
                       { für CGA 320x200 2 Bits, die aufein-  }
                       { ander folgen und die Farbe bilden    }
    xmin, ymin,        { die Dimension des Bildes             }
    xmax, ymax : INTEGER;
    HRes, VRes : INTEGER; { Auflösung des Erzeugers           }
    Palette    : ARRAY[0..15, 0..2] OF BYTE; { Farbpalette    }
    VMode      : BYTE;    { Reserviert                        }
    Planes     : BYTE;    { Anzahl der Farbebenen             }
    BytePerLine: INTEGER; { Bytes pro Scanzeile               }
    PaletteInfo: INTEGER; { 1 = Farbe/Schwarz-Weiß            }
                          { 2 = Grauwerte                     }
    dummy      : ARRAY[0..57] OF BYTE; { Füller auf 128 Bytes }
  END;

PROCEDURE SetEgaReadPlane ( Nr          : BYTE);
PROCEDURE SetEgaWritePlane( Nr          : BYTE);
PROCEDURE SetEgaReg       ( Nr, wert    : BYTE);

FUNCTION GetPCXHeader   (VAR PCXH       : PCX_Header;
                         VAR name       : STRING) : INTEGER;

FUNCTION PutPCXHeader   (VAR PCXH       : PCX_Header;
                         VAR name       : STRING) : INTEGER;

FUNCTION GetPCXByte     (VAR F          : FILE) : BYTE;

FUNCTION PutPCXByte     (VAR F          : FILE;
                         VAR wert,
                         VAR count      : BYTE) : INTEGER;

FUNCTION PutPCXLine     (VAR F          : FILE;
                          VAR buf       : plane;
                         VAR count      : BYTE) : INTEGER;

PROCEDURE DefPCXPalette (VAR PCXH       : PCX_Header;
                         VAR ColType    : BYTE);

FUNCTION SavePCX        (    gd, gm     : INTEGER;
                             XMin, YMin,
                             XMax, YMax : INTEGER;
                         VAR name       : STRING) : INTEGER;
```

```
IMPLEMENTATION


CONST
  BLOCKSIZE : WORD = 512;

  PCXDefaultPalette : ARRAY[0..15, 0..2] OF BYTE =
     ( (0, 0, 0), (0, 0, 170), (0, 170, 0), (0, 170, 170),
       (170, 0, 0), (170, 0, 170), (170, 170, 0), (170, 170, 170),
       (85, 85, 85), (85, 85, 255), (85, 255, 85), (85, 255, 255),
       (255, 85, 85), (255, 85, 255), (255, 255, 85), (255, 255, 255) );

VAR
  pcxbuf : ARRAY[1..512] OF BYTE;

PROCEDURE SetEgaReadPlane(Nr : BYTE);
BEGIN
  PORT[$3CE] := 4;
  PORT[$3CF] := Nr;
END;

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

PROCEDURE SetEgaReg (Nr, wert : BYTE);
BEGIN
  PORT[$3CE] := Nr;
  PORT[$3CF] := wert;
END;

FUNCTION GetPCXHeader(VAR PCXH : PCX_Header;
                          name : STRING) : INTEGER;
VAR
  F : FILE;
BEGIN
  FillChar(PCXH, 128, 0);
  ASSIGN(F, name);
  Reset(F, 1);
  DOSERROR := IOResult;
  IF DOSERROR <> 0 THEN
  BEGIN
    GetPCXHeader := DOSERROR;
    EXIT;
  END;

  Blockread(F, PCXH, 128);
  DOSERROR := IOResult;
  IF DOSERROR <> 0 THEN
  BEGIN
    GetPCXHeader := DOSERROR;
    Close(F);
    EXIT;
  END;

  Close(F);
  GetPCXHeader := IOResult;
  IF (PCXH.version > 5) OR (PCXH.encoding > 1) THEN GetPCXHeader := -1;
END;

FUNCTION PutPCXHeader(VAR PCXH : PCX_Header;
                          name : STRING) : INTEGER;
VAR
  F : FILE;
BEGIN
  ASSIGN(F, name);
  Rewrite(F, 1);
  DOSERROR := IOResult;
  IF DOSERROR <> 0 THEN
  BEGIN
    PutPCXHeader := DOSERROR;
    EXIT;
  END;

  Blockwrite(F, PCXH, 128);
  DOSERROR := IOResult;
  IF DOSERROR <> 0 THEN
  BEGIN
    PutPCXHeader := DOSERROR;
    Close(F);
    IF IOResult <> 0 THEN;
    EXIT;
  END;
```

```
  Close(F);
  PutPCXHeader := IOResult;
END;

FUNCTION GetPCXByte(VAR F : FILE) : BYTE;
CONST
  count   : BYTE = 0;
  wert    : BYTE = 0;
  p       : WORD = 512;
  endfile : BOOLEAN = FALSE;
VAR
  temp    :   BYTE;

  PROCEDURE Read_Block;
  VAR
    result : WORD;
  BEGIN
    IF EOF(F) THEN endfile := TRUE
              ELSE BEGIN
                BlockRead(F, pcxbuf, BLOCKSIZE, result);
                IF result < BLOCKSIZE THEN BLOCKSIZE := result;
                p := 1
              END;
  END;

  FUNCTION get_byte : BYTE;
  BEGIN
    IF endfile THEN get_byte := 0
               ELSE BEGIN
                 IF p = BLOCKSIZE THEN Read_Block
                                  ELSE INC(p);
                 get_byte := pcxbuf[p];
               END;
  END;

BEGIN
  IF count > 0 THEN
  BEGIN
    DEC(count);
    GetPcxByte := wert;
    EXIT;
  END;

  temp := Get_byte;

  IF temp AND $C0 = $C0 THEN
  BEGIN
    count := temp AND $3F-1;
    wert  := Get_Byte;
  END
  ELSE BEGIN
    count := 0;
    wert  := temp;
  END;
  GetPCXByte := wert;
END;

FUNCTION PutPCXByte(VAR F  : FILE;
                        wert,
                        count : BYTE) : INTEGER;
CONST
  total : LongInt = 0;
BEGIN
  IF (count = 1) AND ($C0 <> $C0 AND wert) THEN
  BEGIN
    Blockwrite(F, wert, 1);
    PutPCXByte := IOResult;
    total := total + 1;
  END
  ELSE BEGIN
    count := $C0 OR count;
    Blockwrite(F, count, 1);
    PutPCXByte := IOResult;
    Blockwrite(F, wert, 1);
    PutPCXByte := IOResult;
    total := total + 2;
  END;
END;

FUNCTION PutPCXLine(VAR F     : FILE;
                    VAR buf   : plane;
                        count : BYTE) : INTEGER;
VAR
  this, last,
```

```
  cptr, RunCount : BYTE;

BEGIN
  PutPCXLine := 0;
  last       := buf^[0];
  RunCount   := 1;

  FOR cptr := 1 TO count-1 DO
  BEGIN
    IF buf^[cptr] = last THEN
    BEGIN
      INC(RunCount);
      IF RunCount = 62 THEN
      BEGIN
        DOSERROR := PutPCXByte(F, last, RunCount);
        IF DOSERROR <> 0 THEN
        BEGIN
          PutPCXLine := DOSERROR;
          EXIT;
        END;
        RunCount := 0;
      END;
    END
    ELSE BEGIN
      DOSERROR := PutPCXByte(F, last, RunCount);
      IF DOSERROR <> 0 THEN
      BEGIN
        PutPCXLine := DOSERROR;
        EXIT;
      END;
      last := buf^[cptr];
      RunCount := 1;
    END;
  END;

  IF RunCount > 0 THEN
  BEGIN
    DOSERROR := PutPCXByte(F, last, RunCount);
    IF DOSERROR <> 0 THEN PutPCXLine := DOSERROR;
  END;
END;

PROCEDURE DefPCXPalette(VAR PCXH : PCX_Header; ColTYPE : BYTE);
VAR
  I, J : INTEGER;
BEGIN
  CASE ColType OF
    0 : BEGIN
          FillChar(PCXH.Palette, 48, 255);
          FillChar(PCXH.Palette,  3, 0);
        END;
    1 : FOR I := 0 TO 15 DO
        BEGIN
          IF ODD(I) THEN FOR J := 0 TO 2 DO PCXH.Palette[I, J] := 240
                    ELSE FOR J := 0 TO 2 DO PCXH.Palette[I, J] := 0;
        END;
    2 : MOVE(PCXDefaultPalette, PCXH.Palette, 48);
  END;
END;

FUNCTION SavePCX(gd, gm     : INTEGER;
                 XMin, YMin,
                 XMax, YMax : INTEGER;
                 name       : STRING) : INTEGER;
VAR
  F    : FILE;
  Page : BYTE;
  I, J : INTEGER;
  SPtr : POINTER;
  PCXH : PCX_Header;

  PROCEDURE ErrorCheck;
  BEGIN
    IF DOSERROR <> 0 THEN
    BEGIN
      SavePCX := DOSERROR;
      EXIT;
    END;
  END;

  PROCEDURE ReOpenFile;
  BEGIN
    Assign(F, name);
    Reset(f, 1);
    DOSERROR := IOResult;
```

```
    ErrorCheck;
    SEEK(F, 128);
    DOSERROR := IOResult;
    ErrorCheck;
  END;

BEGIN
  FillChar(PCXH, 128, 0);
  PCXH.creator     := 10;
  PCXH.version     := 3;
  PCXH.encoding    := 1;
  PCXH.bits        := 1;
  PCXH.xmin        := Xmin;
  PCXH.ymin        := Ymin;
  PCXH.xmax        := XMax;
  PCXH.ymax        := YMax;
  PCXH.HRes        := 75;
  PCXH.VRes        := 75;
  PCXH.PaletteInfo := 1;

  CASE gd OF
    3,4,5,9 : BEGIN
                CASE gm OF
                  0 : BEGIN
                        PCXH.Planes := 4;
                        PCXH.BytePerLine := 80;
                        DefPCXPalette(PCXH, 2);
                        DOSERROR := PutPCXHeader(PCXH, name);
                        ErrorCheck;
                        ReOpenFile;
                        FOR I := 0 TO 199 DO
                        BEGIN
                          SPTR := Ptr(EgaBase + $400 * ActivePage, I*80);
                          FOR Page := 0 TO 3 DO
                          BEGIN
                            SetEgaReadPlane(Page);
                            MOVE(SPtr^, Z[0]^, 80);
                            DOSERROR := PutPCXLine(F, Z[0], 80);
                            ErrorCheck;
                          END;
                        END;
                      END;
                  1 : BEGIN
                        PCXH.Planes := 4;
                        PCXH.BytePerLine := 80;
                        DefPCXPalette(PCXH, 2);
                        DOSERROR := PutPCXHeader(PCXH, name);
                        ErrorCheck;
                        ReOpenFile;
                        FOR I := 0 TO 349 DO
                        BEGIN
                          SPTR := Ptr(EgaBase + $800 * ActivePage, I*80);
                          FOR Page := 0 TO 3 DO
                          BEGIN
                            SetEgaReadPlane(Page);
                            MOVE(SPtr^, Z[0]^, 80);
                            DOSERROR := PutPCXLine(F, Z[0], 80);
                            ErrorCheck;
                          END;
                        END;
                      END;
                  2 : BEGIN
                        PCXH.Planes := 4;
                        PCXH.BytePerLine := 80;
                        DefPCXPalette(PCXH, 2);
                        DOSERROR := PutPCXHeader(PCXH, name);
                        ErrorCheck;
                        ReOpenFile;
                        FOR I := 0 TO 479 DO
                        BEGIN
                          SPTR := Ptr(EgaBase + $960 * ActivePage, I*80);
                          FOR Page := 0 TO 3 DO
                          BEGIN
                            SetEgaReadPlane(Page);
                            MOVE(SPtr^, Z[0]^, 80);
                            DOSERROR := PutPCXLine(F, Z[0], 80);
                            ErrorCheck;
                          END;
                        END;
                      END;
                  3 : BEGIN
                        PCXH.Planes := 1;
                        PCXH.BytePerLine := 80;
                        PCXH.Version := 2;
                        DefPCXPalette(PCXH, 0);
```

```
                        DOSERROR := PutPCXHeader(PCXH, name);
                        ErrorCheck;
                        ReOpenFile;
                        SetEgaReadPlane(0);
                        FOR I := 0 TO 349 DO
                        BEGIN
                          SPTR := Ptr(EgaBase + $800 * ActivePage, I*80);
                          MOVE(SPtr^, Z[0]^, 80);
                          BlockWrite(F, Z[0]^, 80);
                          DOSERROR := PutPCXLine(F, Z[0], 80);
                          ErrorCheck;
                        END;
                      END;
                END;
              END;
          7 : BEGIN { CASE gd OF 7 }
                PCXH.Planes := 1;
                PCXH.BytePerLine := 90;
                PCXH.Version := 2;
                DefPCXPalette(PCXH, 0);
                DOSERROR := PutPCXHeader(PCXH, name);
                ErrorCheck;
                ReOpenFile;
                FOR I := 0 TO 347 DO
                BEGIN
                  SPtr := Ptr(HercBase, WORD((I AND 3) SHL 13
                                                     + 90*(I SHR 2)));
                  MOVE(SPtr^, Z[0]^, 90);
                  DOSERROR := PutPCXLine(F, Z[0], 90);
                  ErrorCheck;
                END;
              END;
        1,2 : BEGIN { CASE gd OF 1, 2 }
                PCXH.Planes      := 1;
                PCXH.Bits        := 2;
                PCXH.BytePerLine := 80;
                IF (gd = 2) AND (gm = 3) THEN
                BEGIN
                  J := 479;
                  PCXH.Bits := 1;
                END
                ELSE J := 199;
                IF gm = 4 THEN PCXH.Bits := 1;
                PCXH.Version := 5;
                DefPCXPalette(PCXH, 1);
                DOSERROR := PutPCXHeader(PCXH, name);
                ErrorCheck;
                ReOpenFile;
                FOR I := 0 TO J DO
                BEGIN
                  SPtr := Ptr(CgaBase, WORD((I AND 1) SHL 13
                                                     + 80*(I SHR 1)));
                  MOVE(SPtr^, Z[0]^, 80);
                  DOSERROR := PutPCXLine(F, Z[0], 80);
                  ErrorCheck;
                END;
              END;
  END;
  CLOSE(F);
  IF IOResult <> 0 THEN;
END;

BEGIN
END.
```

Listing 2. SHOWPCX.PAS: Programm zum Zeigen von PCX-Dateien

```
{ ******************************************************* }
{ *  Programm: SHOWPCX - Anzeigen von PCX-Bilder         * }
{ ******************************************************* }
{SR-,S-,I-}

USES CRT, DOS, GRAPH, PCX;

CONST
{ Kartencodes: }
  PCX_VGA      = 0;
  PCX_EGA      = 1;
  PCX_CGA      = 2;
  PCX_HERCULES = 3;

{ Fehlercodes: }
  NoScanMem    = 1010;
  NoInitGraph  = 1011;
  NoCrtSupport = 1012;
  NoPcxFound   = 1013;
  WrongPcxType = 1014;

VAR
  Header        : PCX_HEADER; { PCX-Header                  }
  name          : STRING;     { Name der PCX-Datei          }
  gm, gd,                     { Grafik-Initialisierung      }
  screen,                     { Bildschirmtyp               }
  MaxScanLines,               { Anzahl der Bildzeilen       }
  MoveBytes     : INTEGER;    { Anzahl der Bytes pro Zeile }

PROCEDURE PutPCX(VAR Header : PCX_HEADER;
                     name   : STRING);
VAR
  F     : FILE;
  I,J,L : INTEGER;
  P     : POINTER;
BEGIN
  Assign(F, name);
  IF IOResult <> 0 THEN EXIT;

  Reset(F, 1);
  IF IOResult <> 0 THEN EXIT;

  Seek(f, 128);
  IF IOResult <> 0 THEN EXIT;

  IF (Screen = PCX_EGA) OR (Screen = PCX_VGA) THEN
  BEGIN
    SetEgaReg(5, 0);
    SetEgaReg(1, 0);
  END;

  FOR L := 0 TO MaxScanLines DO
  BEGIN
    FOR J := 0 TO Header.Planes-1 DO
    BEGIN
      FOR I:= 0 TO Header.BytePerLine-1 DO
      BEGIN
        z[J]^[I] := GetPCXByte(F);
      END;
    END;

    CASE screen OF
      PCX_EGA,
      PCX_VGA      : BEGIN   { alle Planes anzeigen }
                       P := Ptr(EgaBase, L*80);
                       FOR J := 0 TO Header.Planes-1 DO
                       BEGIN
                         SetEgaWritePlane(J);
                         MOVE(Z[J]^, P^, MoveBytes);
                       END;
                     END;
      PCX_HERCULES : BEGIN   { nur die erste Plane wird geschrieben }
                       P := Ptr(HercBase, WORD((L AND 3) SHL 13 + 90*(L SHR
2)));
                       MOVE(Z[0]^, P^, MoveBytes);
                     END;
      PCX_CGA      : BEGIN   { nur die erste Plane wird geschrieben }
                       P := Ptr(CgaBase, WORD((L AND 1) SHL 13 + 80*(L SHR
1)));
                       MOVE(Z[0]^, P^, MoveBytes);
                     END;
    END;
  END;
```

```
  Close(F);
  IF IOResult <> 0 THEN {};

  IF (Screen = PCX_EGA) OR (Screen = PCX_VGA) THEN SetEgaWritePlane($F);
END;

VAR
  I : INTEGER;
BEGIN
  IF paramcount < 1 THEN HALT(1);
  name := ParamStr(1);

  DetectGraph(gd, gm);

  CASE gd OF
    3 : Screen := PCX_EGA;
    1 : Screen := PCX_CGA;
    7 : Screen := PCX_HERCULES;
    9 : Screen := PCX_VGA;
  END;

  I := GetPCXHeader(Header, name);
  IF I <> 0 THEN HALT(NoPcxFound);

  IF (Header.Bits > 1) AND (gd <> 1) AND (gd <> 2) THEN HALT(WrongPcxType);

  IF (Header.Bits = 1) AND ((gd = 1) OR (gd = 2)) THEN HALT(WrongPcxType);

  FOR I := 0 TO Header.planes-1 DO GetMem(z[I], Header.BytePerLine);

  initgraph(gd, gm, 'C:\TP\BGI');

  IF GraphResult <> 0 THEN HALT(NoInitGraph);

  MoveBytes := GetmaxX DIV 8;
  IF MoveBytes > Header.BytePerLine THEN MoveBytes := Header.BytePerLine;

  MaxScanLines := Header.ymax - Header.ymin;

  IF MaxScanLines > GetMaxY + 1 THEN MaxScanLines := GetMaxY;

  PutPCX(Header, name);

  REPEAT UNTIL KEYPRESSED;

  RestoreCrtMode;
END.
```

aus mc 12/90 – Seite 104-109