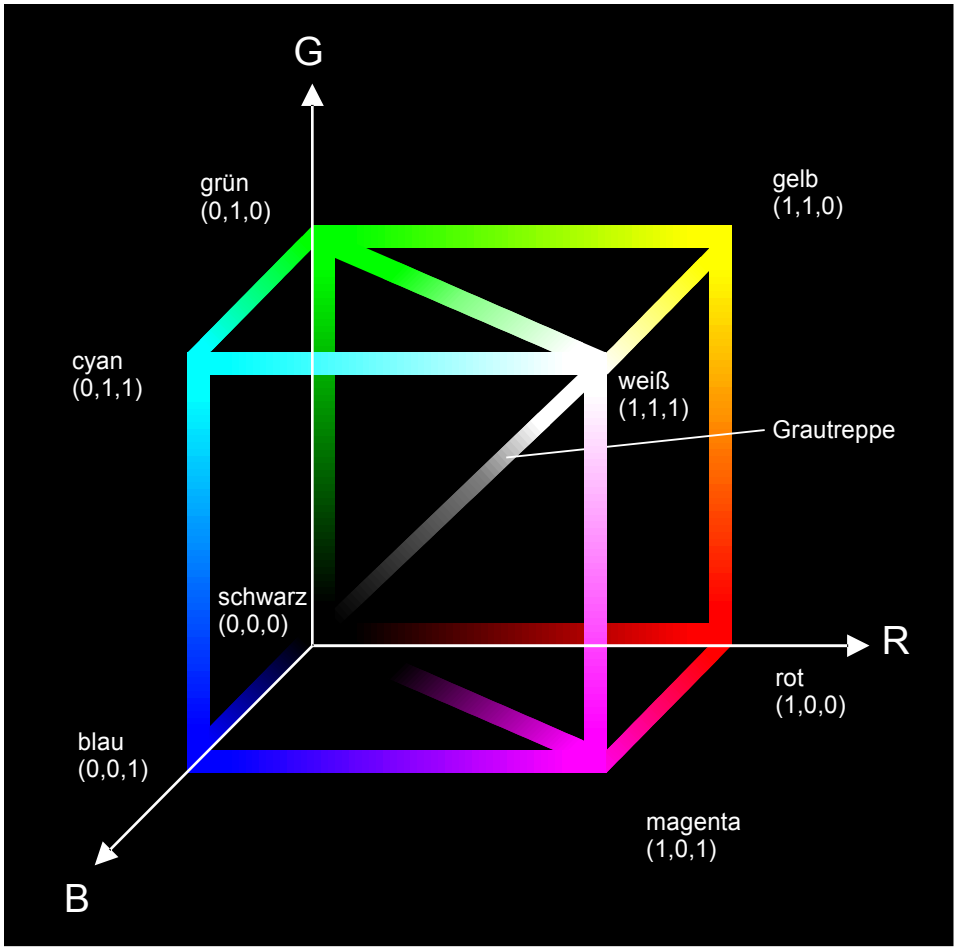
Farbige Tricks

Farbauflösung bildgerecht reduzieren

Farbenprächtige Bilder sind eine Freude – wenn die Grafikkarte sie anzeigen kann. Hochauflösende Grafiken erfordern jedoch mehr Speicherplatz, als eine Standard-VGA bieten kann. Dann gibt es nur noch zwei Möglichkeiten: Das Bild mit geringer Auflösung im 320x200-Modus mit 256 Farben ansehen oder bei hoher Auflösung die 256 Farben geschickt auf 16 reduzieren.



Wer einmal mit der Screen Machine von FAST-Electronic gearbeitet hat, der weiß, daß das Reduzieren von Farben eine komplizierte Angelegenheit ist. Die Screen Machine kennt zwei Modi: einen schnellen und einen langsameren, der dafür eine bessere Farbanpassung bietet. Allerdings muß die Screen Machine auch aus 16 Millionen Farben 256 Basisfarben herauspicken. Damit sieht sich die Software der Screen Machine mit den gleichen Problemen konfrontiert, die auch wir zu lösen haben, wenn es darum geht, 256 Farben auf 16 zu mappen.

Es gibt mehrere Ansätze, die jedoch alle mit Vorsicht zu genießen sind. Die einfachste Lösung wäre, alle 256 Farben auf die Standard-EGA/VGA-Farben zu mappen. Dazu sind zwei Schritte nötig:

– Zuerst stellt man die 16 Basisfarben zusammen, deren Werte man beispielsweise im Turbo-Pascal-Referenzhandbuch unter SetRGBPalette nachschlägt. Alternativ dazu ermittelt man die Werte auch aus der Unit COLOR.PAS (*Listing 2*) über die folgende Formel:

```
RGB-Werte Farbe I := EGAStandardPalette [ EGAStandardIndex[I] ]
```

– Wenn die RGB-Werte der 16 Basisfarben bekannt sind, müssen die übrigen Farben über die RGB-Werte auf diese Basisfarben reduziert werden. Dazu kann man die Farbe über die RGB-Farbwerte mappen oder über den korrespondierenden Grauwert. Üblicherweise verwendet man den RGB-Wert, obwohl der Grauwert teilweise bessere Ergebnisse liefert. Der Grauwert einer RGB-Farbe wird in der Regel über die folgende Umrechnung gewonnen:

$$0{,}30 \cdot R + 0{,}59 \cdot G + 0{,}11 \cdot B$$

Der so gewonnene Grauwert liegt abhängig von den Ausgangswerten zwischen 0 und 63, 0 und 255 oder zwischen 0 und 1.

Das Mappen selbst erfolgt durch Bestimmen der kleinen RGB-Differenz, also einer Vektordifferenz, die über die Quadrate der Einzeldifferenzen ermittelt wird:

$$(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2$$

oder

$$\left(\text{Grauwert}(R_1, G_1, B_1) - \text{Grauwert}(R_2, G_2, B_2) \right)^2$$

Für jede der 16 Basisfarben wird die Differenz aus Basisfarbe und RGB-Wert der zu mappenden Farbe ermittelt und der Index mit der kleinsten Differenz als neuer Farbindex gewählt. Folglich muß man ein Array [0..255] OF BYTE mitführen, das für alle 256 Komponenten den neuen Farbindex enthält. In diesem Array sind am Ende nur noch Werte zwischen 0 und 15 vorhanden. Die Farbauflösung wurde somit erfolgreich reduziert.

Aber ist diese Aktion wirklich so erfolgreich? Wir haben 16 Farben willkürlich ausgewählt und alle Farben darauf abgebildet. Dabei wurde nicht berücksichtigt, ob das Bild irgendwelche Farbschwerpunkte hat. Vielleicht stellt das Bild das Portrait eines Menschen dar und besteht deshalb zu über 80 Prozent aus Fleischfarben. In diesem Fall ergäbe sich ein Falschfarbenbild, da in der Standardpalette mal eben vier Farbtöne enthalten sind, die ungefähr paßten.

Der Fehler liegt also bereits bei der Auswahl der 16 Basisfarben. Um diesen zu vermeiden, muß allerdings die Verteilung der Farbindizes im Bild bekannt sein. Und genau hier gilt es aufzupassen! Wird die Verteilung über das gesamte aufgebaute Bild berechnet, fällt sehr viel Rechenzeit an. Ein Aufwand, der sich manchmal nicht vermeiden läßt – sehr oft aber auch überflüssig ist. Wenn ein Bild nämlich neu eingelesen werden muß, läßt sich besonders bei GIF-Bildern der Aufwand halbieren. Der GIF-Dekodierungs-Algorithmus liefert bereits die benötigten Farbindizes, die nur noch in einer Häufigkeitstabelle erfaßt werden müssen. Ähnlich läßt sich auch bei PCX-Bildern verfahren, da neuere Programme (beispielsweise die Windows-Screen-Shot-Utilities TIFFANY und WinShot) die Farben nicht mehr in Form von Planes ablegen, sondern als geblockte Zahlen. In anderen Fällen kann die Farbe immer noch etwas beschleunigt beim Einlesen der einzelnen Zeilen ermittelt werden.

Mit der Häufigkeit der Farbindizes ist die Verteilung der Farben bekannt. Bevor jedoch weitere Berechnungen erfolgen, sollte zunächst ein Test auf die Anzahl der Farben erfolgen. In der Regel enthält ein im 256-Farben-Modus gespeichertes Bild selten 256 Farben. Eine kleine Reduzierung ergibt sich also schon durch die Bestimmung der tatsächlich benutzten Farbindizes.

Farben auswählen

Aus den verbleibenden Farbindizes werden zunächst 16 Basisfarben gewählt. Dafür gibt es eine naheliegende Strategie: Man wählt die 16 häufigsten Farben aus. Diese Methode ist sicherlich die schnellste, aber nicht immer die beste. Wenn die am häufigsten verwendeten Farben nahe beieinander liegen, geht viel Farbkontrast verloren. Neben den Häufigkeiten muß folglich noch der Abstand der Farben voneinander berücksichtigt werden. Den gesuchten Wert für den durchschnittlichen Abstand der Farben erhält man mit zwei Schritten:

- Das RGB-Array der benutzten Indizes nach Grauwerten sortieren, so daß der Index 0 den größten Grauwert enthält.
- Die Differenz zwischen jeweils zwei benachbarten Grauwerten bilden und durch die Anzahl der Indizes dividieren.

Dieser Ansatz läßt die Häufigkeit der Farben völlig außer acht: er stellt lediglich den Durchschnittswert fest, mit dessen Hilfe die 16 Basisfarben leichter bestimmt werden. In der Praxis haben sich dabei zwei Ansätze als sinnvoll erwiesen: Das Top-Down-Prinzip und das wechselweise Setzen der Basisfarben.

Beim Top-Down-Prinzip wird die Farbe mit der größten Häufigkeit zuerst gesetzt. Es folgt die Farbe mit der nächst niedrigeren Häufigkeit, deren Grauwert sich mindestens um einen festzulegenden Delta-Wert von den bereits gesetzten Farben unterscheidet. Dies wiederholt man, bis 16 Farben gefunden wurden.

Paßt der Farbabstand?

Bleibt die Frage nach dem passenden Delta-Wert. Leider gibt es kein mathematisch berechenbares Optimum. Liegen die häufigsten Farben sehr eng beieinander, sollte der Delta-Wert etwas unter dem Durchschnittswert liegen. Wird er jedoch zu klein gewählt, geht wiederum zu viel Kontrast verloren.

Der Durchschnittswert ergibt sich aus

(durchschnittlicher Farbabstand) × (Anzahl der Farben) ⁄ 16

Dieser Durchschnittswert stellt sicher, daß in der Regel auch einige der weniger häufig vorkommenden Farben erhalten bleiben. Beim zweiten Verfahren, dem wechselweisen Setzen der Basisfarben wird, ausgehend von einem Delta-Wert und der Häufigkeitsverteilung, abwechselnd die häufigste und die am wenigsten häufigste Farbe ausgewählt, wobei immer der Delta-Wert als Schranke berücksichtigt wird. Auf diese Weise bewegt man sich von den Rändern der Verteilung zur Mitte. Dieser Weg liefert besonders gute Ergebnisse, wenn die Farben annähernd gleich verteilt sind.

Sind die Basisfarben gefunden, werden alle Farbindizes nach dem oben beschriebenen Differenzenverfahren auf die Basisfarben gemappt.

Ob die Basisfarben nach dem Top-Down-Prinzip oder wechselweise gesetzt werden, hängt letztendlich von dem Bildmaterial ab. Als Faustformel kann gelten: Sind die Farben annähernd gleich verteilt, ist das wechselweise Auswählen vorzuziehen, ansonsten das Top-Down-Prinzip. Beide Verfahren weisen jedoch eine jeweils typische Schwäche auf. Wird beim Top-Down-Prinzip der Delta-Wert zu klein gewählt, werden die häufig vorkommenden Farben überproportional berücksichtigt, so daß es zu Kontrastverlusten kommt; ist der Delta-Wert zu groß, werden die häufig vorkommenden Farben nicht genügend berücksichtigt.

Typisch ist jedoch, daß die Farbpalette in etwa linear von der häufigsten Farbe an absteigend zusammengestellt wird und gegen Ende der Palette, also bei Index 15 oder 16, in der Regel die verbleibenden Farben abschneidet. Die Anzahl der nicht berücksichtigten Farben kann folglich als Kriterium für die Qualität der Anpassung verwendet werden.

Das Verfahren des wechselweisen Auswählens der Basisfarben vermeidet das Abschneiden am Ende der Palette, birgt jedoch die Gefahr, daß in der Mitte des Farbspektrums zu wenig Farben vorhanden sind. Als Qualitätskriterium kann hier der Abstand der Indizes 15 und 16 gelten, da diese Farben in der Mitte der Ausgangspalette liegen sollten.

Da es kein optimales Verfahren gibt, sollte man die Original-Farbinformationen temporär speichern, damit der Vorgang eventuell wiederholt werden kann. Alternativ kann man jedoch auch einen Trick verwenden, um aus 16 Farben 32 zu machen. Zuerst bestimmt man 32 Basisfarben, anschließend erfolgt das Mapping auf 32 Farben.

Beim Setzen der Palette werden jedoch nur die geraden Indizes berücksichtigt. Erst beim Anzeigen der Farben kommen die Zwischenwerte zur Geltung, indem für alle ungeraden Indizes von 1 bis 31 ein simples Dithering erfolgt. Dazu ist eine 2×2-Dither-Matrix erforderlich.

$$D = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Für jede Pixelposition (x, y) wird [(x AND 1), (y AND 1)] berechnet. Ist das Ergebnis 0, wird der nächst kleinere gerade Farbindex verwendet, ansonsten der nächst höhere gerade Farbindex.

Auf diese Weise werden zwar nur 16 Farben verwendet, jedoch ergibt sich durch das Mischen von jeweils zwei Farben ein etwas weicherer Übergang, der jedoch nur zum Tragen kommt, wenn das Bild größere Flächen aufweist. Denn die Mischung erfordert mindestens vier benachbarte Pixel, um zur Geltung zu kommen.

Die folgende Unit (*Listing 2*) enthält eine Reihe von Basisroutinen und Datentypen, die beim Mappen von Farben nützlich sind. Darüber hinaus sind die Standard-Paletten der EGA- und VGA-Grafikkarte als Konstanten beigefügt.

Dietmar Bückart/ha

Listing 1. Ermittlung der RGB-Werte

```
USES COLOR;

VAR
  S_RGB          : SmallRGBPalette;
  B_RGB          : BigRGBPalette;
  IA             : BigIndexArray;
  RHPtr          : RGBHistoPtr;
  Maxindex, I, J : INTEGER;
  ActGray, rStep : REAL;

BEGIN
  Randomize;
  GetMem(RHPtr, SizeOf(RGBHistoArray) );
  FOR I := 0 TO 255 DO
    FOR J := 0 TO 2 DO
      B_RGB[I, J] := RANDOM(63);
  InitRGBHistoArray(RHPtr, @B_RGB);

  FOR I := 0 TO 255 DO
  BEGIN
    { Histogramm zufällig erzeugen und für NULLen sorgen: }
    RHPtr^.HistoCount[I] := -10000 + Random($FFFF);
    IF RHPtr^.HistoCount[I] < 0
      THEN RHPtr^.HistoCount[I] := 0;
  END;

  { Nur 16 Farben ?     }
  MaxIndex := GetUsedColors(RHPtr);
  IF MaxIndex <= 16 THEN
  BEGIN
    Writeln('Es wurden nur maximal 16 Farben verwendet.');
    HALT(0);
  END;

  { Array nach Anzahl sortieren }
  SortRGBHistoArray(RHPtr, 255, SortByCount);
  { Aus RGB-Werten Grauwerte erzeugen }
  SetRGBHistoArrayGray(RHPtr);
  { Array bis Index I nach Grauwerten sortieren }
  SortRGBHistoArray(RHPtr, I, SortByGray);
  { Durchschnittsabstand der Farben ermitteln }
  rStep := GetColorDifference(RHPtr);
  { 16 Basisfarben nach Durchschnittsmethode festlegen }
  FillChar(S_RGB, SizeOf(SmallRGBPalette), 0);
  rStep := MaxIndex / 16;
  I := 0;
  J := 0;
  WHILE I < 16 DO
  BEGIN
    S_RGB[I, 0] := RHPtr^.RGBValues[J, 0];
    S_RGB[I, 1] := RHPtr^.RGBValues[J, 1];
    S_RGB[I, 2] := RHPtr^.RGBValues[J, 2];
    INC(I);
    INC(J, ROUND(rStep) );
  END;

  { Farben über kleinste Differenz mappen }
  FOR I := 0 TO 255 DO
  BEGIN
```

```
    IA[I] := GetSmallRGBIndex(S_RGB, I, RHPtr);
    Write( I, ' RGB(', B_RGB[I][0], ',',
                       B_RGB[I][1], ',',
                       B_RGB[I][2], ') -> ');
    Writeln (IA[I], ' RGB(', S_RGB[IA[I],0], ',',
                             S_RGB[IA[I],1], ',',
                             S_RGB[IA[I],2], ')' );
  END;
END.
```

Listing 2. Diese Unit enthält Basisroutinen und -tabellen

```
UNIT COLOR;

INTERFACE

TYPE
  SmallRGBPtr     = ^SmallRGBPalette;
  SmallRGBPalette = ARRAY[0..15, 0..2] OF BYTE;
  BigRGBPtr       = ^BigRGBPalette;
  BigRGBPalette   = ARRAY[0..255, 0..2] OF BYTE;
  BigIndexArray   = ARRAY[0..255] OF BYTE;
  RGBHistoCount   = ARRAY[0..255] OF LongInt;
  RGBGrayArray    = ARRAY[0..255] OF REAL;
  RGBHistoPtr     = ^RGBHistoArray;
  RGBHistoArray   = RECORD
                      HistoCount : RGBHistoCount;
                      GrayValues : RGBGrayArray;
                      RGBValues  : BigRGBPalette;
                    END;
  RGBSortStyle    = (SortByGray, SortByCount);
  RGBLessProc     = FUNCTION (X, Y : INTEGER;
                              RH   : RGBHistoPtr) : BOOLEAN;

CONST
  EGAStandardIndex   : ARRAY[0..15] OF BYTE = (
    0, 1, 2, 3, 4, 5, 20, 7, 56, 57, 58, 59, 60, 61, 62, 63);
  EGAStandardPalette : ARRAY[0..63, 0..2] OF BYTE = (
   ($00,$00,$00), ($00,$00,$2b), ($00,$2b,$00), ($00,$2b,$2b),
   ($2b,$00,$00), ($2b,$00,$2b), ($2b,$2b,$00), ($2b,$2b,$2b),
   ($00,$00,$15), ($00,$00,$3F), ($00,$2b,$15), ($00,$2b,$3F),
   ($2b,$00,$15), ($2b,$00,$3F), ($2b,$2b,$15), ($2b,$2b,$3F),
   ($00,$15,$00), ($00,$15,$2b), ($00,$3F,$00), ($00,$3F,$2b),
   ($2b,$15,$00), ($2b,$15,$2b), ($2b,$3F,$00), ($2b,$3F,$2b),
   ($00,$15,$15), ($00,$15,$3F), ($00,$3F,$15), ($00,$3F,$3F),
   ($2b,$15,$15), ($2b,$15,$3F), ($2b,$3F,$15), ($2b,$3F,$3F),
   ($15,$00,$00), ($15,$00,$2b), ($15,$2b,$00), ($15,$2b,$2b),
   ($3F,$00,$00), ($3F,$00,$2b), ($3F,$2b,$00), ($3F,$2b,$2b),
   ($15,$00,$15), ($15,$00,$3F), ($15,$2b,$15), ($15,$2b,$3F),
   ($3F,$00,$15), ($3F,$00,$3F), ($3F,$2b,$15), ($3F,$2b,$3F),
   ($15,$15,$00), ($15,$15,$2b), ($15,$3F,$00), ($15,$3F,$2b),
   ($3F,$15,$00), ($3F,$15,$2b), ($3F,$3F,$00), ($3F,$3F,$2b),
   ($15,$15,$15), ($15,$15,$3F), ($15,$3F,$15), ($15,$3F,$3F),
   ($3F,$15,$15), ($3F,$15,$3F), ($3F,$3F,$15), ($3F,$3F,$3F) );

  VGAStandardPalette : ARRAY[0..255, 0..2] OF BYTE = (
   ($00,$00,$00), ($00,$00,$2A), ($00,$2A,$00), ($00,$2A,$2A),
   ($2A,$00,$00), ($2A,$00,$2A), ($2A,$15,$00), ($2A,$2A,$2A),
   ($15,$15,$15), ($15,$15,$3F), ($15,$3F,$15), ($15,$3F,$3F),
   ($3F,$15,$15), ($3F,$15,$3F), ($3F,$3F,$15), ($3F,$3F,$3F),
   ($3B,$3B,$3B), ($37,$37,$37), ($34,$34,$34), ($30,$30,$30),
   ($2D,$2D,$2D), ($2A,$2A,$2A), ($26,$26,$26), ($23,$23,$23),
   ($1F,$1F,$1F), ($1C,$1C,$1C), ($19,$19,$19), ($15,$15,$15),
   ($12,$12,$12), ($0E,$0E,$0E), ($0B,$0B,$0B), ($08,$08,$08),
   ($3F,$00,$00), ($3B,$00,$00), ($38,$00,$00), ($35,$00,$00),
   ($32,$00,$00), ($2F,$00,$00), ($2C,$00,$00), ($29,$00,$00),
   ($26,$00,$00), ($22,$00,$00), ($1F,$00,$00), ($1C,$00,$00),
   ($19,$00,$00), ($16,$00,$00), ($13,$00,$00), ($10,$00,$00),
   ($3F,$36,$36), ($3F,$2E,$2E), ($3F,$27,$27), ($3F,$1F,$1F),
   ($3F,$17,$17), ($3F,$10,$10), ($3F,$08,$08), ($3F,$00,$00),
   ($3F,$2A,$17), ($3F,$26,$10), ($3F,$22,$08), ($3F,$1E,$00),
   ($39,$1B,$00), ($33,$18,$00), ($2D,$15,$00), ($27,$13,$00),
   ($3F,$3F,$36), ($3F,$3F,$2E), ($3F,$3F,$27), ($3F,$3F,$1F),
```

```
   ($3F,$3E,$17), ($3F,$3D,$10), ($3F,$3D,$08), ($3F,$3D,$00),
   ($39,$36,$00), ($33,$31,$00), ($2D,$2B,$00), ($27,$27,$00),
   ($21,$21,$00), ($1C,$1B,$00), ($16,$15,$00), ($10,$10,$00),
   ($34,$3F,$17), ($31,$3F,$10), ($2D,$3F,$08), ($28,$3F,$00),
   ($24,$39,$00), ($20,$33,$00), ($1D,$2D,$00), ($18,$27,$00),
   ($36,$3F,$36), ($2F,$3F,$2E), ($27,$3F,$27), ($20,$3F,$1F),
   ($18,$3F,$17), ($10,$3F,$10), ($08,$3F,$08), ($00,$3F,$00),
   ($00,$3F,$00), ($00,$3B,$00), ($00,$38,$00), ($00,$35,$00),
   ($01,$32,$00), ($01,$2F,$00), ($01,$2C,$00), ($01,$29,$00),
   ($01,$26,$00), ($01,$22,$00), ($01,$1F,$00), ($01,$1C,$00),
   ($01,$19,$00), ($01,$16,$00), ($01,$13,$00), ($01,$10,$00),
   ($36,$3F,$3F), ($2E,$3F,$3F), ($27,$3F,$3F), ($1F,$3F,$3E),
   ($17,$3F,$3F), ($10,$3F,$3F), ($08,$3F,$3F), ($00,$3F,$3F),
   ($00,$39,$39), ($00,$33,$33), ($00,$2D,$2D), ($00,$27,$27),
   ($00,$21,$21), ($00,$1C,$1C), ($00,$16,$16), ($00,$10,$10),
   ($17,$2F,$3F), ($10,$2C,$3F), ($08,$2A,$3F), ($00,$27,$3F),
   ($00,$23,$39), ($00,$1F,$33), ($00,$1B,$2D), ($00,$17,$27),
   ($36,$36,$3F), ($2E,$2F,$3F), ($27,$27,$3F), ($1F,$20,$3F),
   ($17,$18,$3F), ($10,$10,$3F), ($08,$09,$3F), ($00,$01,$3F),
   ($00,$00,$3F), ($00,$00,$3B), ($00,$00,$38), ($00,$00,$35),
   ($00,$00,$32), ($00,$00,$2F), ($00,$00,$2C), ($00,$00,$29),
   ($00,$00,$26), ($00,$00,$22), ($00,$00,$1F), ($00,$00,$1C),
   ($00,$00,$19), ($00,$00,$16), ($00,$00,$13), ($00,$00,$10),
   ($3C,$36,$3F), ($39,$2E,$3F), ($36,$27,$3F), ($34,$1F,$3F),
   ($32,$17,$3F), ($2F,$10,$3F), ($2D,$08,$3F), ($2A,$00,$3F),
   ($26,$00,$39), ($20,$00,$33), ($1D,$00,$2D), ($18,$00,$27),
   ($14,$00,$21), ($11,$00,$1C), ($0D,$00,$16), ($0A,$00,$10),
   ($3F,$36,$3F), ($3F,$2E,$3F), ($3F,$27,$3F), ($3F,$1F,$3F),
   ($3F,$17,$3F), ($3F,$10,$3F), ($3F,$08,$3F), ($3F,$00,$3F),
   ($38,$00,$39), ($32,$00,$33), ($2D,$00,$2D), ($27,$00,$27),
   ($21,$00,$21), ($1B,$00,$1C), ($16,$00,$16), ($10,$00,$10),
   ($3F,$3A,$37), ($3F,$38,$34), ($3F,$36,$31), ($3F,$35,$2F),
   ($3F,$33,$2C), ($3F,$31,$29), ($3F,$2F,$27), ($3F,$2E,$24),
   ($3F,$2C,$20), ($3F,$29,$1C), ($3F,$27,$18), ($3C,$25,$17),
   ($3A,$23,$16), ($37,$22,$15), ($34,$20,$14), ($32,$1F,$13),
   ($2F,$1E,$12), ($2D,$1C,$11), ($2A,$1A,$10), ($28,$19,$0F),
   ($27,$18,$0E), ($24,$17,$0D), ($22,$16,$0C), ($20,$14,$0B),
   ($1D,$13,$0A), ($1B,$12,$09), ($17,$10,$08), ($15,$0F,$07),
   ($12,$0E,$06), ($10,$0C,$06), ($0E,$0B,$05), ($0A,$08,$03),
   ($00,$00,$00), ($00,$00,$00), ($00,$00,$00), ($00,$00,$00),
   ($00,$00,$00), ($00,$00,$00), ($00,$00,$00), ($00,$00,$00),
   ($31,$0A,$0A), ($31,$13,$0A), ($31,$1D,$0A), ($31,$27,$0A),
   ($31,$31,$0A), ($27,$31,$0A), ($1D,$31,$0A), ($13,$31,$0A),
   ($0A,$31,$0C), ($0A,$31,$17), ($0A,$31,$22), ($0A,$31,$2D),
   ($0A,$2A,$31), ($0A,$1F,$31), ($0A,$14,$31), ($0B,$0A,$31),
   ($16,$0A,$31), ($21,$0A,$31), ($2C,$0A,$31), ($31,$0A,$2B),
   ($31,$0A,$20), ($31,$0A,$15), ($31,$0A,$0A), ($3F,$3F,$3F) );

FUNCTION  TestArrayForGray  (    RP      :  BigRGBPtr;
                                 Max     :  INTEGER) : BOOLEAN;
FUNCTION  TestForGray       (    R, G, B : BYTE) : BOOLEAN;
PROCEDURE ColorToGray       (VAR R, G, B : BYTE);
FUNCTION  GetGray_BYTE_Value(    R, G, B : BYTE) : BYTE;
FUNCTION  GetGray_REAL_Value(    R, G, B : BYTE) : REAL;
PROCEDURE SetRGBHistoArrayGray (VAR RH      : RGBHistoPtr);
PROCEDURE InitRGBHistoArray (VAR RH      : RGBHistoPtr;
                                 RGBP    : BigRGBPtr);
PROCEDURE SortRGBHistoArray (VAR RH      : RGBHistoPtr;
                                 MaxIndex: INTEGER;
                                 SStyle  : RGBSortStyle);
FUNCTION  GetUsedColors     (    RH      : RGBHistoPtr) : INTEGER;
FUNCTION  GetColorDifference(    RH      : RGBHistoPtr) : REAL;
```

```
FUNCTION  GetSmallRGBIndex  (    SR      : SmallRGBPalette;
                                 Col     : INTEGER;
                                 RH      : RGBHistoPtr) : BYTE;
FUNCTION GetSmallGrayIndex  (    SR      : SmallRGBPalette;
                                 Col     : INTEGER;
                                 RH      : RGBHistoPtr) : BYTE;

IMPLEMENTATION

VAR
  SortRGBLess : RGBLessProc;

FUNCTION TestArrayForGray(RP  : BigRGBPtr;
                          Max : INTEGER) : BOOLEAN;
VAR
  I : INTEGER;
BEGIN
  TestArrayForGray := TRUE;

  FOR I := 0 TO Max DO
  BEGIN
    IF (RP^[I, 0] <> RP^[I, 1]) OR
       (RP^[I, 0] <> RP^[I, 2]) THEN
    BEGIN
      TestArrayForGray := FALSE;
      EXIT;
    END;
  END;
END;

FUNCTION TestForGray(R, G, B : BYTE) : BOOLEAN;
BEGIN
  TestForGray := ( (R = G) AND (R = B) );
END;

FUNCTION GetGray_Real_Value( R, G, B : BYTE) : REAL;
BEGIN
  GetGray_REAL_Value := 0.30 * R + 0.59 * G + 0.11 * B;
END;

FUNCTION GetGray_BYTE_Value( R, G, B : BYTE) : BYTE;
BEGIN
  GetGray_BYTE_Value := ROUND( 0.30 * R + 0.59 * G + 0.11 * B);
END;

PROCEDURE ColorToGray (VAR R, G, B : BYTE);
BEGIN
  R := ROUND( 0.30 * R + 0.59 * G + 0.11 * B);
  G := R;
  B := R;
END;

PROCEDURE InitRGBHistoArray (VAR RH      : RGBHistoPtr;
                                 RGBP    : BigRGBPtr);
BEGIN
  FillChar(RH^, SizeOf(RGBHistoArray), 0);
  MOVE(RGBP^, RH^.RGBValues, SizeOf(BigRGBPalette));
END;

{$F+}
FUNCTION GrayLess(X, Y : INTEGER;
                  RH   : RGBHistoPtr) : BOOLEAN;
```

```
BEGIN
  GrayLess := ( RH^.GrayValues[X] > RH^.GrayValues[Y] );
END;

FUNCTION CountLess(X, Y : INTEGER;
                   RH   : RGBHistoPtr) : BOOLEAN;
BEGIN
  CountLess := ( RH^.HistoCount[X] > RH^.HistoCount[Y] );
END;
{$F-}

PROCEDURE SortRGB(VAR RH   : RGBHistoPtr;
                      L, M : INTEGER);
VAR
  I, J, X, Y       : INTEGER;
  LVal             : LongInt;
  R                : REAL;
  RVal, GVal, BVal : BYTE;
BEGIN
  I := L;
  J := M;
  X := (L + M) SHR 1;
  REPEAT
    WHILE SortRGBLess(I, X, RH) DO Inc(I);
    WHILE SortRGBLess(X, J, RH) DO Dec(J);
    IF I <= J THEN
    BEGIN
      LVal   := RH^.HistoCount[I];
      R      := RH^.GrayValues[I];
      RVal   := RH^.RGBValues[I, 0];
      GVal   := RH^.RGBValues[I, 1];
      BVal   := RH^.RGBValues[I, 2];
      RH^.HistoCount[I]   := RH^.HistoCount[J];
      RH^.GrayValues[I]   := RH^.GrayValues[J];
      RH^.RGBValues[I, 0] := RH^.RGBValues[J, 0];
      RH^.RGBValues[I, 1] := RH^.RGBValues[J, 1];
      RH^.RGBValues[I, 2] := RH^.RGBValues[J, 2];
      RH^.HistoCount[J]   := LVal;
      RH^.GrayValues[J]   := R;
      RH^.RGBValues[J, 0] := RVal;
      RH^.RGBValues[J, 1] := GVal;
      RH^.RGBValues[J, 2] := BVal;
      INC(I);
      DEC(J);
    END;
  UNTIL I > J;
  IF L < J THEN SortRGB(RH, L, J);
  IF I < M THEN SortRGB(RH, I, M);
END;

PROCEDURE SetRGBHistoArrayGray (VAR RH : RGBHistoPtr);
VAR
  I : INTEGER;
BEGIN
  FOR I := 0 TO 255 DO
    RH^.GrayValues[ I ] := GetGray_REAL_Value(RH^.RGBValues[I, 0],
                                              RH^.RGBValues[I, 1],
                                              RH^.RGBValues[I, 2]);
END;

PROCEDURE SortRGBHistoArray (VAR RH      : RGBHistoPtr;
                                 MaxIndex: INTEGER;
```

```
                                  SStyle  : RGBSortStyle);
BEGIN
  CASE SStyle OF
    SortByGray  : SortRGBLess := GrayLess;
    SortByCount : SortRGBLess := CountLess;
  END;

  SortRGB(RH, 0, MaxIndex);
END;

FUNCTION GetUsedColors(RH : RGBHistoPtr) : INTEGER;
VAR
  I, C : INTEGER;
BEGIN
  C := 0;
  FOR I := 0 TO 255 DO
    IF RH^.HistoCount[I] > 0
      THEN INC(C);
  GetUsedColors := C;
END;

FUNCTION GetColorDifference( RH : RGBHistoPtr) : REAL;
VAR
  Count      : LongInt;
  I          : INTEGER;
  diff, rVal : REAL;
BEGIN
  rVal  := 0;
  I     := 0;
  Count := 0;

  WHILE (RH^.HistoCount[ I + 1 ] > 0) DO
  BEGIN
    diff  := (RH^.GrayValues[I] - RH^.GrayValues[ I + 1 ]);
    rVal  := rVal + diff;
    Count := Count + 1;
    INC(I);
  END;
  IF Count = 0 then Count:=1;

  GetColorDifference := rVal / Count;
END;

FUNCTION GetSmallRGBIndex(SR  : SmallRGBPalette;
                          Col : INTEGER;
                          RH  : RGBHistoPtr) : BYTE;
VAR
  I, MinIndex       : INTEGER;
  TempDiff, minDiff : REAL;
BEGIN
  minDiff  := MaxLongInt;
  MinIndex := 0;
  FOR I := 0 TO 15 DO
  BEGIN
    TempDiff := ( SQR(SR[I, 0] - RH^.RGBValues[Col, 0]) +
                  SQR(SR[I, 1] - RH^.RGBValues[Col, 1]) +
                  SQR(SR[I, 2] - RH^.RGBValues[Col, 2]) );
    IF TempDiff < MinDiff THEN
    BEGIN
      MinDiff  := TempDiff;
      MinIndex := I;
    END;
```

```
  END;
  GetSmallRGBIndex := MinIndex;
END;

FUNCTION GetSmallGrayIndex (  SR : SmallRGBPalette;
                            Col : INTEGER;
                             RH : RGBHistoPtr) : BYTE;
VAR
  I, MinIndex       : INTEGER;
  TempDiff, minDiff : REAL;
BEGIN
  minDiff  := MaxLongInt;
  MinIndex := 0;
  FOR I := 0 TO 15 DO
  BEGIN
    TempDiff := ( SQR(GetGray_REAL_Value(SR[I, 0],
                                         SR[I, 1],
                                         SR[I, 2]) -
                      RH^.GrayValues[Col])       );
    IF TempDiff < MinDiff THEN
    BEGIN
      MinDiff := TempDiff;
      MinIndex := I;
    END;
  END;
  GetSmallGrayIndex := MinIndex;
END;

END.
```

aus mc 12/91 – Seite 92-97