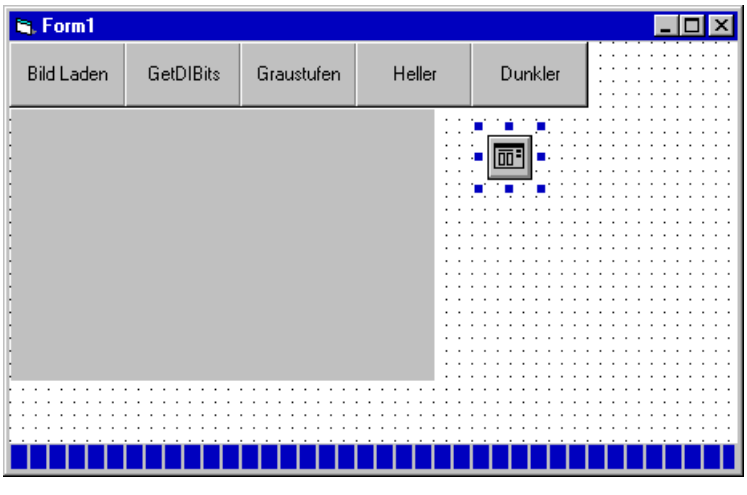
R109 ... eine RGB-Grafik manipulieren?

Viele Grafikprogramme warten mit einer fast schon unüberschaubaren Vielfalt an Filtern und Funktionen auf. Wer von Ihnen schon selbst Grafiken unter VB bearbeitet hat, wird wohl zunächst einmal mit der *PSet*-Methode an den einzelnen Pixeln herumgedoktert haben. Wenn damit auch die gewünschte Funktionalität realisierbar ist (z.B. Graufärbung einer Bitmap), wird man wohl spätestens bei der Geschwindigkeit das Handtuch werfen, denn quälend langsam baut sich das Bild pixelweise neu auf. Der Ansatz ist auch vollkommen falsch, schier endlose Funktionsaufrufe verbergen sich hinter der Methode *PSet*[8]. Abhilfe schafft nur der direkte Griff nach den Daten, d.h. die Arbeit mit der Speicherbitmap.

Vielleicht haben Sie sich schon gewundert, warum sich die Rezeptüberschrift ausgerechnet auf RGB-Grafiken bezieht? Die Antwort: Um den Quellcode überschaubar zu halten, versuchen wir hier, das Brett an der dünnsten Stelle zu bohren, denn bei RGB-Grafiken (24-Bit-Bitmaps) ist jedes einzelne Pixel durch drei Bytes im Speicher abgebildet, was in diesem Fall die Programmierung erheblich vereinfacht.

Oberfläche

Auf das Startformular plazieren Sie fünf Buttons, eine *PictureBox*-Komponente, einen *Progressbar* sowie einen *CommonDialog* zum Laden der Grafik. Setzen Sie *AutoSize* und *AutoRedraw* der *PictureBox* auf *True*.



Die Progressbar können Sie mit der *Align*-Eigenschaft am unteren Rand des Fensters verankern.

---

[8] Auch die Verwendung der entsprechenden GDI-Funktion bringt hier keine wesentliche Verbesserung.

Quelltext

Bevor wir zur Tat schreiten, müssen wir uns um das Einbinden diverser Konstanten, Typen und API-Funktionen kümmern:

```
Private Const DIB_RGB_COLORS As Long = 0

Private Type BITMAPFILEHEADER    ' 14 Bytes
   bfType As Integer
   bfSize As Long
   bfReserved1 As Integer
   bfReserved2 As Integer
   bfOffBits As Long
End Type

Private Type BITMAPINFOHEADER   ' 40 Bytes
   biSize As Long
   biWidth As Long
   biHeight As Long
   biPlanes As Integer
   biBitCount As Integer
   biCompression As Long
   biSizeImage As Long
   biXPelsPerMeter As Long
   biYPelsPerMeter As Long
   biClrUsed As Long
   biClrImportant As Long
End Type
```

Mit dieser Struktur wird später jeder einzelne Punkt der Bitmap beschrieben:

```
Private Type RGBQUAD
   rgbBlue As Byte
   rgbGreen As Byte
   rgbRed As Byte
   rgbReserved As Byte
End Type
```

RGB-Bilder brauchen keine Farbpalette, deshalb nur der Info-Header:

```
Private Type BITMAPINFO_24
   bmiHeader As BITMAPINFOHEADER
End Type

Private Declare Function GetDIBits Lib "gdi32" (ByVal aHDC As Long, ByVal hBitmap _
        As Long, ByVal nStartScan As Long, ByVal nNumScans As Long, lpBits As Any,_
        lpBI As BITMAPINFO_24, ByVal wUsage As Long) As Long
```

```
Private Declare Function SetDIBits Lib "gdi32" (ByVal hdc As Long, _
        ByVal hBitmap As Long, ByVal nStartScan As Long, ByVal nNumScans As Long, _
        lpBits As Any, lpBI As BITMAPINFO_24, ByVal wUsage As Long) As Long
```

Weiterhin brauchen wir einige Variablen bzw. ein Puffer-Array für die Bilddaten:

```
Dim buf() As RGBQUAD
Dim bufsize&, l&
Dim BInfo As BITMAPINFO_24
```

Erster Schritt des Programms ist, wie nicht anders zu erwarten, das Laden der Bitmap:

```
Private Sub Command1_Click()
Dim filename$
   CommonDialog1.Flags = cdlOFNFileMustExist
   CommonDialog1.ShowOpen
   filename = CommonDialog1.FileName
   Picture1.Picture = LoadPicture(filename)
End Sub
```

Nachfolgend laden wir die Bitmap in unser vorbereitetes Puffer-Array

```
Private Sub Command2_Click()
```

Nun die Größe des Puffers bestimmen (da jeder Punkt betroffen ist, fällt die Berechnung recht simpel aus):

```
   bufsize = Picture1.ScaleWidth * Picture1.ScaleHeight   ’  ScaleMode = Pixel !!!!!
   ReDim buf(0 To bufsize - 1)
```

Den Header initialisieren:

```
   With BInfo.bmiHeader
     .biSize = 40
     .biWidth = Picture1.ScaleWidth
     .biHeight = Picture1.ScaleHeight
     .biPlanes = 1
     .biBitCount = 32
     .biCompression = 0
     .biClrUsed = 0
     .biClrImportant = 0
     .biSizeImage = bufsize
   End With
```

Abrufen der gewünschten Daten:

```
   GetDIBits Picture1.hdc, Picture1.Image.Handle, 0, BInfo.bmiHeader.biHeight, _
             buf(0), BInfo, DIB_RGB_COLORS
   Command3.Enabled = True
   Command4.Enabled = True
   Command5.Enabled = True
End Sub
```

Hinweis: Über den Info-Header sowie die Parameter der Funktion *GetDIBits* können Sie die Größe des zurückgegebenen Ausschnitts der Bitmap steuern. Es ist also problemlos möglich, auch nur einen bestimmten Bereich zu bearbeiten.

Der Rest der Geschichte ist relativ schnell erzählt: Mit Hilfe des Typs *RGBQUAD* können Sie nachfolgend auf die Speicherbitmap zugreifen und natürlich auch Veränderungen vornehmen.

Beispiel: Umwandlung einer Farbgrafik in eine Graustufen-Bitmap. Dazu werden die einzelnen Farben entsprechend ihrer Leuchtkraft bewertet und daraus ein Graustufenwert berechnet, der jedem der drei Farbwerte zugewiesen wird.

```
Private Sub Command3_Click()
Dim max&
Dim grau As Byte

   Screen.MousePointer = 11
   max = (Picture1.ScaleWidth) * (Picture1.ScaleHeight) - 1
   ProgressBar1.max = max
   For l = 0 To max
```

Hier erfolgt die Neuberechnung der Farbwerte, wobei uns die Aufsplittung in die drei Grundfarbwerte sehr entgegenkommt:

```
      grau = (CLng(buf(l).rgbRed) * 77 + _
              CLng(buf(l).rgbGreen) * 151 + CLng(buf(l).rgbBlue) * 28) \ 256
```

Allen drei Grundfarben wird der Grauwert zugewiesen:

```
      buf(l).rgbRed = grau
      buf(l).rgbBlue = grau
      buf(l).rgbGreen = grau
      If l Mod 500 = 0 Then ProgressBar1.Value = l
   Next
```

Mit dem folgenden Aufruf von *SetDIBits* wird der veränderte Puffer in die Bitmap zurückgeschrieben, auf dem Bildschirm erscheint die neue Bitmap:

```
   SetDIBits Picture1.hdc, Picture1.Image.Handle, 0, BInfo.bmiHeader.biHeight, _
             buf(0), BInfo, DIB_RGB_COLORS
   Screen.MousePointer = 0
   ProgressBar1.Value = 0
End Sub
```

Hinweis: Bei größeren Bildern sollten Sie die Bitmap in mehreren einzelnen "Streifen" bearbeiten und zurückschreiben. Zum einen ist zu sehen, daß etwas passiert, zum anderen brauchen Sie nicht so große Pufferspeicher.

Beispiel: Aufhellen eines Bildes. Dazu wird zu jedem Farbwert einfach eine Konstante addiert. Wir müssen lediglich darauf achten, daß es nicht zu einem Werteüberlauf kommt.

```
Private Sub Command4_Click() ' Heller
Dim max&
Dim grau As Byte
   Screen.MousePointer = 11
   max = (Picture1.ScaleWidth) * (Picture1.ScaleHeight) - 1
   ProgressBar1.max = max
   For l = 0 To max
     If buf(l).rgbRed < 245 Then buf(l).rgbRed = buf(l).rgbRed + 10
     If buf(l).rgbBlue < 245 Then buf(l).rgbBlue = buf(l).rgbBlue + 10
     If buf(l).rgbGreen < 245 Then buf(l).rgbGreen = buf(l).rgbGreen + 10
     If l Mod 500 = 0 Then ProgressBar1.Value = l
   Next
   SetDIBits Picture1.hdc, Picture1.Image.Handle, 0, BInfo.bmiHeader.biHeight, _
             buf(0), BInfo, DIB_RGB_COLORS
   Screen.MousePointer = 0
   ProgressBar1.Value = 0
End Sub
```

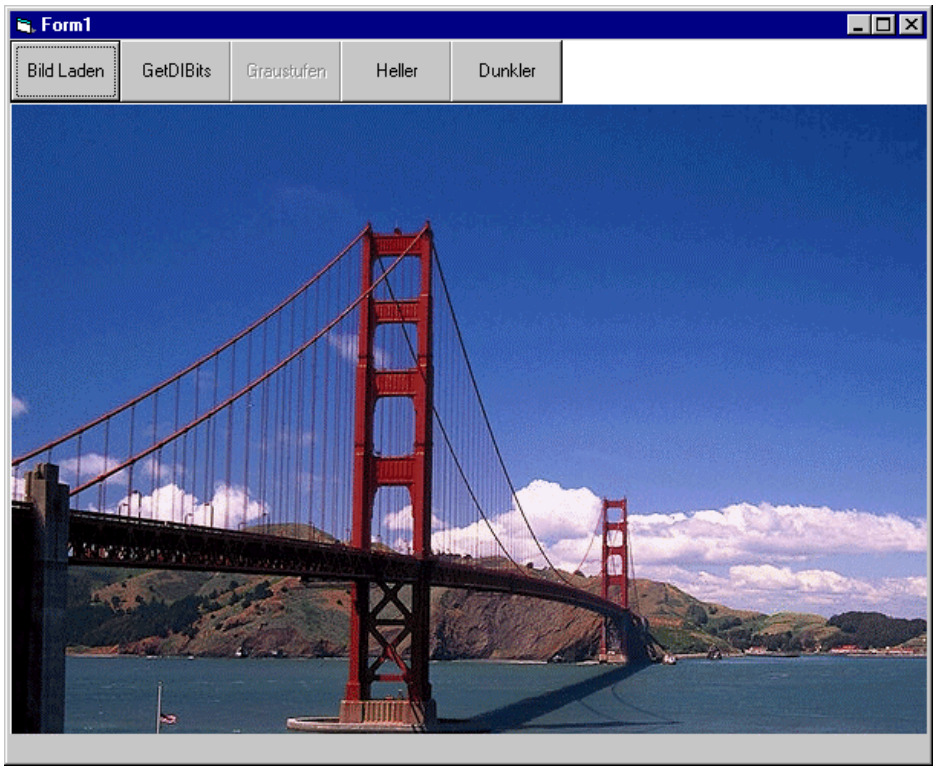
Beispiel: Ein Bild abdunkeln. Das Vorgehen entspricht dem vorherigen Beispiel, allerdings wird jetzt eine Konstante abgezogen.

```
Private Sub Command5_Click()
Dim max&


   Screen.MousePointer = 11
   max = (Picture1.ScaleWidth) * (Picture1.ScaleHeight) - 1
   ProgressBar1.max = max
   For l = 0 To max
     If buf(l).rgbRed > 10 Then buf(l).rgbRed = buf(l).rgbRed - 10
     If buf(l).rgbBlue > 10 Then buf(l).rgbBlue = buf(l).rgbBlue - 10
     If buf(l).rgbGreen > 10 Then buf(l).rgbGreen = buf(l).rgbGreen - 10
     If l Mod 500 = 0 Then ProgressBar1.Value = l
   Next
   SetDIBits Picture1.hdc, Picture1.Image.Handle, 0, BInfo.bmiHeader.biHeight, _
             buf(0), BInfo, DIB_RGB_COLORS
   Screen.MousePointer = 0
   ProgressBar1.Value = 0
End Sub
```

Test

Das Beispielprogramm (Buch-CD) dürfte eindrucksvoll demonstrieren, wie schnell man eine Bitmap bearbeiten kann.



Bemerkungen

- Wer an die Vorzüge der vorgestellten Lösung immer noch nicht glauben will, sollte einmal versuchen, die gleiche Funktionalität durch Setzen der Eigenschaft *Pixels[x,y]* zu realisieren!
- Wie eingangs schon erwähnt, ist das beschriebene Vorgehen nur bei RGB, d.h. 24-Bit-Grafiken, so einfach. Sie können sich sicher vorstellen, daß anderenfalls einiges mehr gerechnet werden muß, der Weg bleibt allerdings der gleiche.
- An dieser Stelle zeigen sich allerdings auch die Grenzen der "Programmiersprache" Visual Basic. Neben fehlenden Pointern etc. sind es vor allem Schleifen, die zu Geschwindigkeitsverlusten gegenüber echten Programmiersprachen führen (das gleiche Programm ist in Delphi ca. 4-5 mal schneller).

R110 ... mit Paletten arbeiten?

Wer seine Grafikkarte auf 256 Farben eingestellt hat, wird festgestellt haben, daß bei Farbverläufen nicht der gewünschte Effekt von gleitenden Farbübergängen auftritt. Der Grund hierfür ist recht schnell gefunden, die gerade aktive Farbpalette (256 Einträge) weist lediglich die vordefinierten Farben auf. Sollen Farbverläufe dennoch dargestellt werden, müssen diese