Gut verpackt

Drucken von JPEG-Bildern mit PostScript Level 2

Thomas Merz

PostScript Level 2 unterstützt unter anderem die Bildkompression nach JPEG, wozu man die JPEG-Daten in eine PostScript-Datei verpacken muß. Dabei sind auch einige Bemerkungen über verschiedene JPEG-Dateiformate und die verschiedenen JPEG-Algorithmen angebracht. Sie sind häufig Ursache für Probleme beim Austausch von JPEG-Daten zwischen Grafikprogrammen.

JPEG bezeichnet kein Dateiformat, sondern eine ganze Familie von Algorithmen zur Kompression digitalisierter Standbilder in Echtfarbqualität. Die Abkürzung JPEG steht für 'Joint Photographic Experts Group'. Aus diesem ursprünglich losen Zusammenschluß von Entwicklern aus vielen Ländern entstand eine Arbeitsgruppe bei der internationalen Standardisierungsorganisation ISO, die mehrere Kompressionsmethoden verglich und weiter verbesserte. So entstand schließlich ein ganzer Werkzeugkasten sehr unterschiedlicher Verfahren, der vor einem Jahr unter der Bezeichnung ISO 10918 als Standard festgeschrieben wurde. Aus diesem Werkzeugkasten können sich Entwickler je nach gewünschtem Anwendungsbereich die benötigten Teile herausnehmen und in ihren Hard- und Softwareprodukten implementieren.

Die Grundlagen von JPEG waren bereits Gegenstand mehrerer c't-Artikel, daher möchte ich hier nur kurz die wichtigsten Schlüsselbegriffe ansprechen: Die diskrete Cosinus-Transformation (DCT) überführt Blöcke von 8×8 Pixeln in den Frequenzraum. Vorher werden die RGB-Daten meist noch in den Farbraum YC_bC_r transformiert. Im Gegensatz zum technisch orientierten Farbraum RGB trennt YC_bC_r Helligkeits- und Farbinformation und entspricht daher mehr der Wahrnehmung des menschlichen Auges. Dieses ist nämlich für Helligkeitsunterschiede empfindlicher als für Farbunterschiede, so daß für die Kodierung der Farbinformation eine geringere Genauigkeit als bei der Helligkeitsinformation ausreicht. YC_bC_r wird oft mit dem ähnlichen Farbraum YUV verwechselt, unterscheidet sich von diesem aber durch eine andere Skalierung der Farbkomponenten. Die DCT-transformierten Werte können im nächsten Schritt quantisiert werden. Dies bedeutet nichts anderes als ein gezieltes Reduzieren der Genauigkeit. Für die abschließende Entropie-Kodierung sieht JPEG alternativ eine Huffman- oder arithmetische Kodierung vor. Die arithmetische Kodierung komprimiert zwar besser als das Huffman-Verfahren, hat jedoch den Nachteil, mit verschiedenen Patenten belegt zu sein, so daß Lizenzgebühren für die Benutzung anfallen. Aus diesem Grund arbeiten viele Implementierungen ausschließlich mit der Huffman-Kodierung. Diese Verfahren können auch in einem hierarchischen Prozeß genutzt werden, bei dem das Bild nacheinander in unterschiedlichen Qualitätsstufen aufgebaut wird. Durch Kombination dieser Algorithmen entstehen insgesamt 13 Kompressionsverfahren, die in die folgenden vier Gruppen fallen:

– Basisprozeß (baseline)
– Erweiterte DCT-Prozesse (extended sequential)
– Verlustfreie Prozesse (lossless)
– Hierarchische Prozesse

Außer bei den verlustfreien Prozessen nimmt man bei allen Varianten einen gewissen Fehler durch die Quantisierung in Kauf. Bei diesen verlustbehafteten (lossy) Verfahren liefert die Dekompression der Daten nicht mehr exakt die Ausgangsdaten. Die Verluste sind jedoch so klein, daß selbst bei hohen Kompressionsraten mit bloßem Auge meist noch kein Unterschied zum Original auszumachen ist. Mit verschiedenen Parametern läßt sich die Darstellungsqualität auf Kosten der Kompressionsrate erhöhen. Eine anschauliche Maßeinheit für die Kompression ist die Anzahl der

benötigten Bits pro Pixel. Sind bei einem Farbbild die drei Grundfarben mit jeweils 8 Bit kodiert, so entspricht das 24 Bit pro Pixel bei den Ausgangsdaten. JPEG erreicht schon bei einer durchschnittlichen Bitrate von 0,75 Bit pro Pixel exzellente Bildqualität und selbst 0,2 Bit pro Pixel liefern noch brauchbare Bilder. Die verlustbehafteten JPEG-Prozesse sind auf fotografische Aufnahmen natürlicher Szenen mit fließenden Farbübergängen hin optimiert. Für andere Arten von Bildern sind sie nicht so gut geeignet, etwa computergenerierte Abbildungen oder Strichzeichnungen (z. B. Comics), die meist große Farbflächen und viele abrupte Farbwechsel enthalten.

Das JPEG-Komitee legte zwar auf mehreren hundert Seiten alle Einzelheiten der Algorithmen fest, definierte jedoch kein allgemein verwendbares Dateiformat für komprimierte Bilder. Es normierte nur das sogenannte *JPEG Interchange Format*, das zur Darstellung des eigentlichen JPEG-Datenstroms dient. Worin liegt der Unterschied? Dazu ein Beispiel: JPEG erlaubt Farbräume mit 1, 2, 3 oder 4 Komponenten. Die Kernalgorithmen kümmern sich nicht um die Bedeutung einzelner Farben, sondern komprimieren beziehungsweise dekomprimieren einen Datenstrom nach gewissen Regeln; man sagt daher, JPEG sei farbenblind. Die Anzahl der Komponenten legt den Farbraum aber noch nicht eindeutig fest, zum Beispiel verwenden RGB und YC_bC_r jeweils drei Komponenten. Das in der Norm festgelegte Format enthält aber keinerlei Informationen über den benutzten Farbraum, so daß diese zusätzlich zum eigentlichen Datenstrom kodiert werden müssen.

Der Inhalt …

Die vielen unterschiedlichen Kompressionsverfahren, die nicht alle realisiert sein müssen, und das nicht ganz vollständige Austauschformat hatten also zur Folge, daß verschiedene Hersteller ihre JPEG-Daten geringfügig anders auf Dateien ablegten. Dies wiederum bedeutete, daß nicht alle JPEG-Implementierungen zueinander kompatibel sind. Um diese häßliche Tatsache zu ändern, definierte der Hardwarehersteller C-Cube Microsystems das *JPEG File Interchange Format* (JFIF), das auf dem Interchange Format der ISO basiert und zum einen die vielen Freiheiten etwas einschränkt, zum anderen die Lücken des Formats schließt, um so den Datenaustausch zu ermöglichen. Dabei geht es wohlgemerkt nur um den Austausch von Daten zwischen verschiedenen Applikationen und Plattformen, innerhalb geschlossener Umgebungen gibt es natürlich keinerlei Einschränkungen. JFIF legt zum Beispiel

Marker im JPEG Interchange Format und in JFIF

Komprimierte Bilddaten gemäß JPEG Interchange Format werden durch sogenannte Marker strukturiert. Jedes Markersegment beginnt mit dem Byte FF, nachdem ein zweites Byte die Funktion des jeweiligen Markers angibt. Vor dem ersten FF dürfen optional noch weitere Füll-Bytes mit dem Wert FF stehen. FF-Bytes, die bei der Huffman-Kodierung zufällig entstehen, werden durch ein folgendes Null-Byte ‘entschärft’, um eine Verwechslung mit Markern auszuschließen. Die meisten Markersegmente haben variable Länge und enthalten daher am Anfang zwei Bytes, die die Länge des gesamten Segments angeben (einschließlich der beiden Längen-Bytes, aber ausschließlich der Marker-Bytes). Diese Struktur erlaubt es Programmen wie JPEG2PS, einen JPEG-Datenstrom auszuwerten, ohne sich mit der Huffman- oder arithmetischen Kodierung herumschlagen zu müssen. Die folgende Tabelle enthält alle in ISO 10918 definierten Marker. Auf die mit einem Stern versehenen Marker folgt keine Längenangabe, das Segment besteht nur aus dem Marker ohne weitere Daten. Die SOF-Marker legen den zur Kompression benutzten Algorithmus fest. PostScript unterstützt nur SOF_0 und SOF_1. Die differentiellen Verfahren sind Teil hierarchischer Bildkompression.

Wert	Kürzel	Beschreibung
Verfahren mit Huffman-Kodierung:		
FFC0	SOF_0	(Start of Frame) DCT-Basisprozeß (baseline)
FFC1	SOF_1	Erweiterter sequentieller DCT-Prozeß (extended sequential DCT)
FFC2	SOF_2	Progressiver DCT-Prozeß
FFC3	SOF_3	Sequentieller verlustfreier Prozeß
Differentielle Verfahren mit Huffman-Kodierung:		
FFC5	SOF_5	Differentieller sequentieller DCT-Prozeß
FFC6	SOF_6	Differentieller progressiver DCT-Prozeß
FFC7	SOF_7	Differentieller verlustfreier Prozeß
Verfahren mit arithmetischer Kodierung:		
FFC8	JPG	Reserviert für JPEG-Erweiterungen
FFC9	SOF_9	Erweiterter sequentieller DCT-Prozeß
FFCA	SOF_{10}	Progressiver DCT-Prozeß
FFCB	SOF_{11}	Sequentieller verlustfreier Prozeß
Differentielle Verfahren mit arithmetischer Kodierung:		
FFCD	SOF_{13}	Differentieller sequentieller DCT-Prozeß
FFCE	SOF_{14}	Differentieller progressiver DCT-Prozeß
FFCF	SOF_{15}	Differentieller verlustfreier Prozeß
Weitere Marker:		
FFC4	DHT	(Define Huffman table) Festlegung einer Huffman-Dekodierungstabelle
FFCC	DAC	(Define arithmetic coding conditionings) Festlegung einer Tabelle für die arithmetische Kodierung
FFD0 - FFD7	RST_m*	(Restart) Neustart des Dekodierungsprozesses (Synchronisierungspunkt)
FFD8	SOI*	(Start of image) Beginn des JPEG-Datenstroms, muß immer am Anfang stehen
FFD9	EOI*	(End of image) Ende des JPEG-Datenstroms
FFDA	SOS	(Start of scan) Dieses Segment enthält die komprimierten und kodierten Bilddaten
FFDB	DQT	(Define quantization table) Festlegung einer Quantisierungstabelle
FFDC	DNL	(Define number of lines) Nachträgliche Festlegung der Anzahl von Scanzeilen (wird in PostScript nicht ausgewertet)
FFDD	DRI	(Define restart interval) Festlegung der Länge eines Intervalls für die Synchronisierung durch RST-Marker
FFDE	DHP	(Define hierarchical progression) Festlegung der Parameter für das letzte Bild beim hierarchischen progressiven Bildaufbau
FFDF	EXP	(Expand reference image) Hochrechnen der räumlichen Auflösung um bestimmte horizontale und vertikale Faktoren
FFE0 - FFEF	APP_n	(Application segment) Markersegment für anwendungsspezifische Daten (wird z. B. von JFIF benutzt, siehe unten)
FFF0 - FFFD	JPG_n	Reserviert für JPEG-Erweiterungen
FFFE	COM	Kommentar (Informationen, die keine Auswirkung auf die Dekodierung haben)
FF01	TEM*	Wird temporär für die arithmetische Kodierung benutzt
FF02 - FFBF	RES	Reserviert

JPEG-Dateiformat JFIF

JFIF-Dateien werden durch ein APP_0-Markersegment gekennzeichnet, das direkt auf den SOI-Marker folgt. Das Segment beginnt mit FF, E0, dann folgen die Zeichenkette ‘JFIF’, ein Null-Byte sowie zwei Bytes für die JFIF-Version (z. Zt. ist Version 1.02 aktuell, also 01, 02). Das nächste Byte beschreibt die Einheit für die Pixeldichte (0 = Verhältnis der horizontalen zur vertikalen Pixelausdehnung, 1 = Pixel pro Zoll, 2 = Pixel pro cm), gefolgt von der x- und y-Pixeldichte mit je zwei Bytes. Optional kann der JFIF-Marker noch eine verkleinerte Version des Gesamtbilds (Thumbnail) enthalten. In diesem Fall beschreiben die nächsten beiden Bytes horizontale und vertikale Ausdehnung des Thumbnails in Pixeln, gefolgt von den Thumbnail-Farbwerten (pro Pixel drei Bytes mit RGB-Daten). Schließlich sind unter der APP-Kennung ‘JFXX’ noch verschiedene JFIF-Erweiterungen in zusätzlichen APP_0-Segmenten möglich. Eine der drei bisher definierten Erweiterungen erlaubt zum Beispiel die Integration einer Thumbnail-Version, die selbst mittels JPEG komprimiert ist.

als Farbraum Graustufen (bei einer Farbkomponente) oder YC_bC_r fest und empfiehlt den Basisprozeß als Kompressionsmethode. Der Basisprozeß muß bei allen DCT-Systemen implementiert sein und stellt sozusagen den kleinsten gemeinsamen JPEG-Nenner dar.

Was die benötigten Zusatzinformationen angeht, benutzt JFIF eine Möglichkeit, die schon im Standard vorgesehen ist: Über sogenannte Application Markers kann man Zusatzdaten integrieren, die Bestandteil des Bildes sein sollen, von den Kern-Algorithmen aber ignoriert werden. Syntaktisch entsprechen JFIF-Dateien also voll der Norm, sie bilden daher keine weitere Kompatibilitätshürde. Inhaltlich vereinfachen sie die Dekompression durch Einschränkung der ausufernden JPEG-Norm. Zusätzlich können JFIF-Dateien auch eine unkomprimierte Minidarstellung des Bildes (Thumbnail) enthalten. Dies ermöglicht einen schnellen Überblick, wenn man etwa eine Bilddatenbank durchsuchen möchte, ohne erst alle Bilder zu dekomprimieren.

Wie ist nun der Zusammenhang mit PostScript? Adobe wollte bei der Erweiterung von PostScript zu Level 2 alle wichtigen Kompressionsalgorithmen berücksichtigen. JPEG war zur Zeit der Freigabe von Level 2 im Jahre 1990 zwar schon ziemlich stabil, allerdings noch keine festgeschriebene Norm. Die einzelnen Versionen (Drafts) unterschieden sich aber nur noch in Kleinigkeiten, so daß Adobe die Implementierung auf Basis der damaligen Veröffentlichungen wagte. Zum Glück hat sich bis zur endgültigen Normierung nichts Wesentliches mehr geändert, so daß PostScript Level 2 kompatibel zur ISO-Norm ist und daher auch das JPEG-Interchange-Format unterstützt. Die Algorithmen wurden in sogenannten Filtern realisiert, die über beliebige Datenquellen (Dateien, Prozeduren, Standardeingabekanal) gelegt werden können und transparenten Zugriff auf die JPEG-komprimierten Daten ermöglichen. Diese Filter unterstützen (mit einer kleinen Einschränkung) den Basisprozeß sowie den erweiterten DCT-Prozeß mit Huffman-Backend und einer Genauigkeit von 8 Bit pro Komponente. Das bedeutet, daß der PostScript-Interpreter JFIF-Daten direkt dekomprimieren kann (vorausgesetzt, sie wurden mit dem Basisprozeß oder einem erweiterten DCT-Prozeß komprimiert).

Es stellt sich nur noch die Frage, wie die Daten zum Interpreter kommen und wie man ein JPEG-Bild in ein bestehendes Layout integrieren kann. Das hier gezeigte C-Programm (JPEG2PS) erledigt beides, indem es die Daten wahlweise in eine 7-Bit-Darstellung umwandelt und das ganze in eine EPS-Datei verpackt. EPS (Encapsulated PostScript) ist das Format der Wahl, wenn es darum geht, einzelne PostScript-Abbildungen in Dokumente einzubetten. Die 7-Bit-Darstellung ist auf den meisten DOS-üblichen Übertragungskanälen (seriell, parallel) nötig. Bei transparenter Datenübertragung zum Drucker (z.B. AppleTalk oder TCP/IP auf Ethernet) kann die EPS-Datei Binärdaten enthalten. Die Option -b beim Aufruf von JPEG2PS unterdrückt die Umwandlung nach ASCII und gibt die Daten binär aus. JPEG2PS wandelt die Daten

JPEG-Dilemma in der TIFF-Spezifikation 6.0

Das Tagged Image File Format (TIFF) ist ein Tausendsassa unter den Austauschformaten für Rasterdaten. Es ermöglicht die Speicherung digitalisierter Bilddaten von Schwarzweiß-, Graustufen-, Paletten- und Echtfarbbildern in einer Vielzahl von Varianten, Farbräumen und Kompressionsverfahren sowie die Ablage zusätzlicher Bildinformation neben den eigentlichen Pixel-Rohdaten. Um die Flut von Optionen für TIFF-fähige Software etwas einzudämmen, führte die TIFF-Spezifikation 5.0 im Jahre 1988 verschiedene Klassen von TIFF-Software ein. Das hatte jedoch zur Folge, daß manchmal selbst einfache Bilder nicht mehr zwischen zwei Programmen ausgetauscht werden konnten, da diese verschiedene TIFF-Klassen implementierten. Aus diesem Grund wurden die Klassen in der TIFF-Version 6.0 von 1992 wieder abgeschafft und statt dessen die Minimalanforderungen an TIFF-fähige Software unter der Bezeichnung ‘Baseline TIFF’ festgelegt. Zusätzlich gibt es eine Reihe von Erweiterungen, die jedoch nicht in jeder Applikation implementiert sein müssen. Das schafft zwar auch wieder Inkompatibilitäten, garantiert aber immerhin ein gemeinsames Austauschformat. TIFF 6.0 enthält als Erweiterung gegenüber früheren Versionen unter anderem die Farbräume CMYK und YC_bC_r, die Kachelung eines Bildes sowie die Kompression gemäß JPEG. Die Aufteilung eines großen Bildes in kleine rechteckige Teile (Kacheln) soll einen schnellen Zugriff auf Teilbereiche des Bildes ermöglichen, ohne daß das Gesamtbild dekomprimiert werden muß.

Die JPEG-Kompression in TIFF 6.0 ermöglicht wahlweise den verlustbehafteten Basisprozeß (JPEG-Marker SOF_0) oder den verlustfreien sequentiellen Prozeß mit Huffman-Kodierung (JPEG-Marker SOF_3). Die dazu benötigten Quantisierungs- und Huffman-Tabellen werden in eigenen TIFF-Feldern gespeichert. Die komprimierten Daten werden mittels der Synchronisierungslogik von JPEG (Restart-Marker) so organisiert, daß einzelne Kacheln des Gesamtbilds unabhängig voneinander und in beliebiger Reihenfolge dekomprimiert werden können.

Wie nun aber Dr. Tom Lane, unter dessen Führung die *Independent JPEG Group* Public-Domain-Sourcen zur Kompression und Dekompression von JPEG-Daten veröffentlicht, nachwies, enthält die JPEG-Integration in TIFF 6.0 schwerwiegende Mängel. Dazu gehören zum Beispiel einige unnötige Einschränkungen der Datenformate, die eine spätere Erweiterung auf 12 Bit Genauigkeit pro Farbkomponente verhindern. Zudem wurden die Marker des JPEG-Datenstroms so unglücklich in TIFF eingebettet, daß es nicht möglich ist, die Daten für einzelne Kacheln ohne Zusatzaufwand separat zu dekodieren (was eigentlich Ziel der Integration von JPEG war). Dies spielt vor allem beim Einsatz von Hardware-Codecs (Kodierer/Dekodierer) eine große Rolle, da diese meist einen vollständigen JPEG-Datenstrom einschließlich aller Marker erwarten. Die TIFF-Software kann diese Daten nicht der TIFF-Datei direkt entnehmen, sondern muß sie nach dem Einlesen erst zusammenstellen, bevor sie an den JPEG-Chip weitergereicht werden können. Daher kann die JPEG-Verarbeitung nicht vollständig durch Hardware erfolgen, sondern muß durch Software unterstützt werden. Ein anderes Problem besteht darin, daß bei den meisten JPEG-Chips die Restart-Logik durch Marker gesteuert wird und von außen nicht so beeinflußt werden kann, wie es bei TIFF 6.0 zur Dekompression einzelner Kacheln nötig ist.

Um diese und weitere Schwierigkeiten sowie Inkonsistenzen innerhalb der Spezifikation zu vermeiden, schlagen Tom Lane und das *TIFF Advisory Committee* vor, den JPEG-Teil von TIFF 6.0 aufzugeben und durch ein völlig neues Design zu ersetzen. Dieses beruht auf der Grundidee, auf den mißglückten Einsatz des Restart-Mechanismus ganz zu verzichten und statt dessen einzelne Kacheln durch einen vollständigen JPEG-Datenstrom (einschließlich aller benötigten Marker) zu beschreiben, der unverändert an einen eventuell eingesetzten JPEG-Chip weitergereicht werden kann. Um die Hilfstabellen nicht überflüssigerweise bei jeder Kachel zu laden, sieht der Entwurf ein eigenes TIFF-Feld zur Definition der Tabellen vor. Im Gegensatz zum Originalentwurf benutzt der alternative Vorschlag dazu allerdings kein neues Format, sondern das sogenannte abgekürzte Austauschformat, das bereits in der ISO-Norm definiert wird und daher auch in Hardwareimplementierungen zum Einsatz kommen kann.

Bei dem neuen Vorschlag sind zwar noch einige Detailfragen zu klären, er ermöglicht aber eine saubere und erweiterbare Integration von JPEG in TIFF. Der neue Ansatz hätte aber zur Folge, daß bereits implementierte Software gemäß TIFF 6.0 nicht mehr zu den Neuerungen kompatibel wäre. Allerdings stellt sich die Frage, ob es bereits TIFF/JPEG-Software gibt. Falls ja, müßten die Entwickler die Löcher in der Spezifikation auf ihre eigene Art gestopft haben, was an sich schon ein Kompatibilitätsproblem darstellt. Im NeXT-Betriebssystem gibt es die JPEG-Integration in TIFF zwar schon lange, sie existierte aber bereits vor Veröffentlichung von TIFF 6.0 und benutzt daher ‘private’ Tags, die sowieso nicht kompatibel zu 6.0 sind.

Es bleibt abzuwarten, wie Aldus, der offizielle ‘Hüter’ von TIFF, auf die Vorschläge reagiert. Bis zu einer Modifikation kann man TIFF 6.0 (zumindest den JPEG-Teil) wohl auf Eis legen …

```
 1  %!PS-Adobe-3.0 EPSF-3.0
 2  %%Creator: jpeg2ps
 3  %%Title: nesrin.jpg
 4  %%CreationDate: Mon Jan 24 12:36:33 1994
 5  %%BoundingBox: 40 20 575 822
 6  %%DocumentData: Clean7Bit
 7  %%LanguageLevel: 2
 8  %%EndComments
 9  %%BeginProlog
10  %%EndProlog
11  %%Page: 1 1
12  /languagelevel where {pop languagelevel 2 lt}{true} ifelse {
13    (JPEG-Bild Önesrin.jpgÖ braucht PostScript Level 2!\n) dup
14    print flush
15    /Helvetica findfont 20 scalefont setfont 100 100 moveto show
16    showpage stop
17  } if
18  save
19  /RawData currentfile /ASCIIHexDecode filter def
20  /Data RawData << >> /DCTDecode filter def
21  575 20 translate
22  90 rotate
23  802.00 534.67 scale
24  /DeviceGray setcolorspace
25  { << /ImageType 1
26       /Width 768
27       /Height 512
28       /ImageMatrix [ 768 0 0 -512 0 512 ]
29       /DataSource Data
30       /BitsPerComponent 8
31       /Decode [0 1]
32    >> image
33    Data closefile
34    RawData flushfile
35    showpage
36    restore
37  } exec
38  FFD8FFEE000E41646F626500648000000000FFDB00430009060707070606090909
39  090E0A0A0B0E110F0E0E0D121A1A151015181920201E1F1F1E22323829252425
40  2521272B2C2C2D2F343434332E343434343434343434343434FFC0000B08020000300
41  01011100FFC400D2000001050101010101010000000000000003000102040506
42  ...
43  7133B89E4A0D9D4AE77F6468F12DE4A83FA85C79BDC7E05577E4B9E61CF9F895
44  FFD9>
45  %%EOF
```

Diese EPS-Datei wurde vom JPEG-PostScript-Wrapper erzeugt. Der in den Daten hervorgehobene Teil ist das Segment des SOF_0-Markers. Es enthält unter anderem Breite und Höhe des Bildes, Anzahl der Farbkomponenten und die Anzahl der Bits pro Komponente.

nicht um, sondern verpackt sie nur in eine für den PostScript-Interpreter (und Layout-Programme) verdauliche Form und verdient daher die neudeutsche Bezeichnung *Wrapper*.

… und die Verpackung

Ab Zeile 159 erzeugt das Programm den EPS-Header mit einigen Pflichtübungen (siehe [3]), wie die DSC-Kommentare mit allgemeinen Angaben über das Bild. Wichtig ist hierbei der Kommentar *%%BoundingBox*, der die Ausmaße des Bildes festlegt. Die Druckgröße wird weiter unten durch eine scale-Anweisung bestimmt. JPEG2PS benutzt folgende Strategie zur Skalierung: Ziel ist die Ausgabe auf A4-Papier, wobei das Bild so groß wie möglich gedruckt werden soll. Falls die Bildbreite in Pixeln größer ist als die Höhe, fügt JPEG2PS automatisch eine Rotationsanweisung ein, um das Bild im Querformat zu drucken. Bei diesen geometrischen Operationen berechnet das C-Programm auch die BoundingBox. Wenn Sie die EPS-Datei in einer anderen Größe drucken wollen, so können Sie sie mit einer DTP-Software importieren und dann skalieren. Da die EPS-Datei keinen Preview-Teil enthält, wird sie am Bildschirm nur als grauer Kasten oder durch gekreuzte Linien dargestellt.

Der Kommentar %%LanguageLevel signalisiert einem eventuell benutzten Druck-Spooler (davon können DOS-Anwender natürlich nur träumen), daß die Datei PostScript Level 2 benötigt. Wird die Datei versehentlich auf einem Drucker ausgegeben, der nur PostScript Level 1 beherrscht, so wäre ein PostScript-Fehler die Folge. Um diesen Fall zu erkennen, überprüfen die ersten Anweisungen den Level des benutzten Interpreters. Auf einem Level-1-Gerät druckt die EPS-Datei eine Fehlermeldung und bricht dann den Druckauftrag ab.

Als nächstes wird der Standardeingabekanal (currentfile) zur Dekompression der JPEG-Daten eingerichtet. Dazu werden zwei Filter geschichtet. Da JPEG2PS die Daten in ASCII-Schreibweise darstellt, muß man zunächst den Filter ASCIIHex-Decode über den Standardeingabekanal legen, um an die Binärdaten zu kommen. Über die so erzeugte Datenquelle RawData macht sich der Filter DCTDecode her, der die Dekompression gemäß JPEG besorgt. Dieser Filter kann in einem Dictionary (zwischen doppelten spitzen Klammern) zusätzliche Parameter erhalten. Da der JPEG-Datenstrom jedoch meist schon alle zur Dekompression nötigen Informationen enthält, ist hier das Dictionary leer. Anders als in [2] beschrieben, kann das Dictionary bei Adobes Implementierungen von Level 2 ganz entfallen. Mit Rücksicht auf PostScript-Clones, die das 'Rote Buch' exakt implementieren, enthält das Programm trotzdem ein leeres Dictionary, um keinen Fehler zu riskieren. Jetzt stellt JPEG2PS die Grafikparameter so ein, daß das Bild die gewünschte Größe erreicht, und wählt den Farbraum aus, der der Anzahl der Komponenten entspricht. Ein JFIF-kompatibles Bild mit drei Komponenten ist zwar im Farbraum YC_bC_r kodiert, JPEG2PS stellt aber trotzdem den RGB-Farbraum ein: Der JPEG-Filter konvertiert die Daten bei der Dekompression automatisch zwischen den beiden Farbräumen. Damit sind alle Vorbereitungen für den Einsatz des Operators image getroffen, der die Rasterdaten auf Papier umsetzt. Der Wrapper benutzt eine neuere Form der Parameterübergabe, die es ebenfalls erst ab Level 2 gibt: Die Parameter für image werden in einem Dictionary übergeben. Breite und Höhe entsprechen den Angaben in der JPEG-Datei, als Datenquelle dient die zweifach gefilterte Standardeingabe. Im Anschluß an image erfolgen noch einige Aufräumarbeiten, und die Seite wird ausgegeben.

Aus Symmetriegründen enthält Level 2 auch das Filter DCTEncode, das Daten gemäß JPEG komprimiert. Dieses hat in einer normalen Seitenbeschreibung natürlich nichts zu suchen. Es ist beispielsweise dann nützlich, wenn Rohdaten vom Interpreter komprimiert und so auf der Festplatte des Druckers gespeichert werden sollen.

Kompatibel zum Mac

Die Implementierung von JPEG2PS ist ziemlich geradlinig: Die EPS-Verpackung und Auswertung der verschiedenen Varianten (Hoch-/Querformat, Farbräume) erledigt die Funktion JPEGtoPS, die sich die benötigten Bildparameter über die Funktion AnalyzeJPEG besorgt. Diese liest dazu die JPEG-Marker in der Datei und wertet sie aus (siehe Kasten 'Marker im JPEG Interchange Format und in JFIF'). Ein kleiner Trick hilft dabei, auch JPEG-Dateien vom Macintosh zu verarbeiten. Seit JPEG als Teil von Quick-Time im Apple-Betriebssystem implementiert ist, können PICT-Dateien auch JPEG-komprimierte Bilder enthalten. Solche PICT/JPEGs sind im Grunde JFIF-kompatibel, enthalten aber am Anfang einige hundert Bytes zusätzlicher Daten für die PICT-Strukturen. Der Wrapper umgeht dieses Problem, indem er ab dem Dateianfang nach dem ersten Marker sucht und davorliegende Daten einfach ignoriert.

Der Wrapper kann also Dateien nach dem JPEG Interchange Format (gemäß ISO-Norm), JFIF-Dateien und PICT/JPEGs vom Mac als EPS verpacken. Die entscheidenden Werte für räumliche Auflösung und Farbauflösung stehen im Markersegment SOF_0 oder SOF_1. Diese Werte benötigt übrigens nur der Operator image, die Dekompression erfolgt allein anhand des JPEG-Datenstroms. Wenn man die Bilddaten nur dekomprimieren und nicht drucken will oder die Bildkoordinaten bereits kennt, ist die Kenntnis der Markerstruktur nicht nötig. Anhand der ausgewerteten Marker erfolgen einige Überprüfungen, um sicherzustellen, daß nur solche Dateien als EPS verpackt werden, die der PostScript-Interpreter auch dekomprimieren kann. So werden zum Beispiel Kompressionsverfahren (SOF-Marker), die PostScript nicht unterstützt, ebenso abgelehnt wie eine Farbtiefe pro Komponente ungleich 8 Bit. (be)

Literatur

[1] William B. Pennebaker, Joan L. Mitchell, JPEG Still Image Data Compression Standard, Van Nostrand Reinhold, New York 1993

[2] Adobe Systems Incorporated, PostScript Language Reference Manual, Second Edition, Addison-Wesley, Reading, Mass. 1990

[3] Thomas Merz, Blick in die Kapsel, Das Grafikformat Encapsulated PostScript, c't 12/93, S. 240

[4] Thomas Merz, Stufe zwei, Erweiterung im großen Stil, PostScript Level 2, c't 8/93, S. 182

[5] Adobe Systems Incorporated, Supporting the DCT Filters in PostScript Level 2, Adobe Technical Note #5116, 1992

[6] Eric Hamilton, JPEG File Interchange Format, C-Cube Microsystems 1992

[7] Aldus Developers Desk, TIFF Revision 6.0, Seattle 1992

[8] Oliver Kesy, Bilder schrumpfen, Neue Methoden und Programme zur Bildkompression, c't 11/93, S. 120

```
/* ------------------------------------------------------------
 * jpeg2ps: Konvertierung von JPEG-Daten nach PostScript Level 2
 * (C) 1994 Thomas Merz
 * ------------------------------------------------------------*/
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>

#ifdef DOS
#include <dos.h>
#include <io.h>
#include <fcntl.h>
#else
#include <unistd.h>
#endif

#include "psimage.h"

#ifdef DOS
#define READMODE "rb"           /* JPEG-Daten binaer einlesen */
#else
#define READMODE "r"
#endif

int PageHeight = 842;           /* A4 Seite */
int PageWidth  = 595;           /* A4 Seite */
int Margin     = 20;            /* Sicherheitsabstand vom Rand */

BOOL AnalyzeJPEG(imagedata *image);
#ifdef USE_ASCII85      /* In der kurzen Fassung nicht implementrt */
int ASCII85Encode(FILE *in, FILE *out);
#endif

static void JPEGtoPS(imagedata *JPEG, FILE *PSfile);

static void usage(void) {
  fprintf(stderr, "jpeg2ps - Konvertierung von JPEG nach ");
  fprintf(stderr, "PostScript Level 2\n");
  fprintf(stderr, "(C) Thomas Merz 1994\n\n");
  fprintf(stderr, "Aufruf: jpeg2ps [-b] jpegfile > epsfile\n");
  fprintf(stderr, "-b\tBinaermodus: Ausgabe mit 8-Bit-Daten\n");
  exit(1);
}

int main(int argc, char **argv) {
  int opt;
  imagedata image;

  image.filename = NULL;
#ifdef USE_ASCII85
  image.mode     = ASCII85;
#else
  image.mode     = ASCIIHEX;
#endif
  image.startpos = 0L;
  image.landscape= FALSE;

  for (opt = 1; opt < argc; opt++) {
    if (strcmp(argv[opt], "-b") == 0)
      image.mode = BINARY;
    else if (image.filename) {
      fprintf(stderr, "Zu viele Dateinamen!\n\n");
      usage();
    } else
      image.filename = argv[opt];
  }

  if (!image.filename)
    usage();

  if ((image.fp = fopen(image.filename, READMODE)) == NULL) {
    fprintf(stderr, "Datei Õ%sÕ kann nicht gelesen werden!\n",
      image.filename);
    exit(1);
  }

  JPEGtoPS(&image, stdout);      /* Konvertierung der JPEG-Daten */
  return 0;
}

#define BUFFERSIZE 1024
static char buffer[BUFFERSIZE];

/* Hilfsfunktion: Hexadezimale ASCII-Darstellung von Binaerdaten */
static void ASCIIHexEncode(FILE *in, FILE *out) {
  static char BinToHex[] = "0123456789ABCDEF";
  int i, CharsPerLine;
  size_t n;
  unsigned char *p;

  CharsPerLine = 0;
  putc(Õ\nÕ, out);

  while ((n = fread(buffer, 1, sizeof(buffer), in)) != 0)
    for (i = 0, p = (unsigned char *) buffer; i < n; i++, p++) {
      putc(BinToHex[*p>>4], out);           /* Erstes Halbbyte  */
      putc(BinToHex[*p & 0x0F], out);       /* Zweites Halbbyte */
      if ((CharsPerLine += 2) >= 64) {
        putc(Õ\nÕ, out);
        CharsPerLine = 0;
      }
    }

  putc(Õ>Õ, out);         /* Abschlusszeichen eines HEX-Strings */
}

static char *ColorSpaceNames[] = {"", "Gray", "", "RGB", "CMYK" };

static void JPEGtoPS(imagedata *JPEG, FILE *PSfile) {
  int llx, lly, urx, ury;        /* Koordinaten der BoundingBox */
  size_t n;
  float scale, sx, sy;           /* Skalierungsfaktoren         */
  time_t t;
  int i;

  if (!AnalyzeJPEG(JPEG)) {      /* Bild-Parameter auslesen     */
    fprintf(stderr, "Õ%sÕ ist keine korrekte JPEG-Datei!\n",
                    JPEG->filename);
    return;
  }

  fprintf(stderr, "%s: %dx%d Pixel, %d Farbkomponente(n)\n",
    JPEG->filename, JPEG->width, JPEG->height, JPEG->components);

  /* Bei Bedarf auf Querformat umschalten */
  if (JPEG->width > JPEG->height) {
    JPEG->landscape = TRUE;
    fprintf(stderr, "Ausgabe erfolgt im Querformat!\n");
  }

  if (!JPEG->landscape) {       /* Skalierungsfaktoren berechnen */
    sx = (float) (PageWidth - 2*Margin) / JPEG->width;
    sy = (float) (PageHeight - 2*Margin) / JPEG->height;
  }else {
    sx = (float) (PageHeight - 2*Margin) / JPEG->width;
    sy = (float) (PageWidth - 2*Margin) / JPEG->height;
  }

  /* DIN A4 wird mindestens in einer Dimension voll genutzt */
  scale = min(sx, sy);

  if (JPEG->landscape) {
  /* Bei Querformat wird auf (urx, lly) positioniert und gedruckt */
    urx = PageWidth - Margin;
    lly = Margin;
    ury = (int) (Margin + scale*JPEG->width + 0.9);  /* aufrunden */
    llx = (int) (urx - scale * JPEG->height);        /* abrunden  */
  }else {
  /* Bei Hochformat wird auf (llx, lly) positioniert und gedruckt */
    llx = lly = Margin;
    urx = (int) (llx + scale * JPEG->width + 0.9);   /* aufrunden */
    ury = (int) (lly + scale * JPEG->height + 0.9);  /* aufrunden */
  }

  time(&t);

  /* EPS-Headerkommentare ausgeben */
  fprintf(PSfile, "%%!PS-Adobe-3.0 EPSF-3.0\n");
  fprintf(PSfile, "%%%%Creator: jpeg2ps\n");
  fprintf(PSfile, "%%%%Title: %s\n", JPEG->filename);
  fprintf(PSfile, "%%%%CreationDate: %s", ctime(&t));
  fprintf(PSfile, "%%%%BoundingBox: %d %d %d %d\n",
                  llx, lly, urx, ury);
  fprintf(PSfile, "%%%%DocumentData: %s\n",
                  JPEG->mode == BINARY ? "Binary" : "Clean7Bit");
  fprintf(PSfile, "%%%%LanguageLevel: 2\n");
  fprintf(PSfile, "%%%%EndComments\n");
```

```
169    fprintf(PSfile, "%%%%BeginProlog\n");
170    fprintf(PSfile, "%%%%EndProlog\n");
171    fprintf(PSfile, "%%%%Page: 1 1\n");
172
173    fprintf(PSfile, "/languagelevel where {pop languagelevel 2 lt}");
174    fprintf(PSfile, "{true} ifelse {\n");
175    fprintf(PSfile, "  (JPEG-Bild Õ%sÕ braucht PostScript Level 2!",
176                    JPEG->filename);
177    fprintf(PSfile, "\\n) dup print flush\n");
178    fprintf(PSfile, "  /Helvetica findfont 20 scalefont setfont ");
179    fprintf(PSfile, "100 100 moveto show showpage stop\n");
180    fprintf(PSfile, "} if\n");
181
182    fprintf(PSfile, "save\n");
183    fprintf(PSfile, "/RawData currentfile ");
184
185    if (JPEG->mode == ASCIIHEX)            /* HEX-Darstellung... */
186      fprintf(PSfile, "/ASCIIHexDecode filter ");
187    else if (JPEG->mode == ASCII85)        /* ...oder ASCII85    */
188      fprintf(PSfile, "/ASCII85Decode filter ");
189
190    fprintf(PSfile, "def\n");
191
192    fprintf(PSfile, "/Data RawData << ");
193    fprintf(PSfile, ">> /DCTDecode filter def\n");
194
195    /* Verschiebung zur linken unteren Ecke des Bildes */
196    fprintf(PSfile, "%d %d translate\n", (JPEG->landscape ?
197                     PageWidth - Margin : Margin), Margin);
198
199    if (JPEG->landscape)               /* Drehung bei Querformat */
200      fprintf(PSfile, "90 rotate\n");
201
202    fprintf(PSfile, "%.2f %.2f scale\n", /* Skalierung */
203                     JPEG->width * scale, JPEG->height * scale);
204    fprintf(PSfile, "/Device%s setcolorspace\n",
205                    ColorSpaceNames[JPEG->components]);
206    fprintf(PSfile, "{ << /ImageType 1\n");
207    fprintf(PSfile, "     /Width %d\n", JPEG->width);
208    fprintf(PSfile, "     /Height %d\n", JPEG->height);
209    fprintf(PSfile, "     /ImageMatrix [ %d 0 0 %d 0 %d ]\n",
210                    JPEG->width, -JPEG->height, JPEG->height);
211    fprintf(PSfile, "     /DataSource Data\n");
212    fprintf(PSfile, "     /BitsPerComponent %d\n",
213                    JPEG->bits_per_component);
214
215    fprintf(PSfile, "     /Decode [0 1");
216    for (i = 1; i < JPEG->components; i++)
217      fprintf(PSfile," 0 1");
218    fprintf(PSfile, "]\n");
219
220    fprintf(PSfile, "  >> image\n");
221    fprintf(PSfile, "  Data closefile\n");
222    fprintf(PSfile, "  RawData flushfile\n");
223    fprintf(PSfile, "  showpage\n");
224    fprintf(PSfile, "  restore\n");
225    fprintf(PSfile, "} exec");
226
227    /* Auf die Startposition der JPEG-Daten positionieren */
228    fseek(JPEG->fp, JPEG->startpos, SEEK_SET);
229
230    switch (JPEG->mode) {
231    case BINARY:
232      /* Wichtig: EIN Leerzeichen und KEIN Newline */
233      fprintf(PSfile, " ");
234  #ifdef DOS
235      setmode(fileno(PSfile), O_BINARY);    /* Binaermodus */
236  #endif
237      /* Daten unveraendert kopieren */
238      while ((n = fread(buffer, 1, sizeof(buffer), JPEG->fp)) != 0)
239        fwrite(buffer, 1, n, PSfile);
240      break;
241
242  #ifdef USE_ASCII85
243  case ASCII85:
244      fprintf(PSfile, "\n");
245
246      /* ASCII85-Darstellung der Daten */
247      if (ASCII85Encode(JPEG->fp, PSfile)) {
248        fprintf(stderr, "Problem bei ASCII85Encode!\n");
249        exit(1);
250      }
251      break;
252  #endif
253
254    case ASCIIHEX:
255      /* Hexadezimale ASCII-Darstellung der Daten */
256      ASCIIHexEncode(JPEG->fp, PSfile);
257      break;
258    }
259    fprintf(PSfile, "\n%%%%EOF\n");
260  }
```

Der JPEG-Wrapper verpackt Bilddaten in eine Datei gemäß PostScript Level 2. ð