MULTIMEDIADATENFORMATE
Seminar im WS 02/03

Freie Universität Berlin
Institut für Informatik
Dozent: Gerald Friedland

Die Grafikformate

GIF und PNG

Autor:
Annette Kaudel

Inhaltsverzeichnis

1 Einleitung 4

Historie 4

2 GIF 5

2.1 Eigenschaften 5

2.2 Dateiaufbau 5

2.2.1 Header 5

2.2.2 Bilddatenblock 6

2.2.3 Interlaced Modus 7

2.2.4 Bilddaten 8

2.2.5 LZW-Kompression 9

2.2.6 Terminator 10

2.2.7 Erweiterungsblöcke 10

2.3 Beispiele 12

3 PNG 13

3.1 Eigenschaften 13

3.2 Dateiaufbau 13

3.2.1 Signatur 14

Chunk Layout 14

3.2.3 Header Chunk IHDR 15

3.2.4 Palette Chunk PLTE 16

Data Chunk IDAT 16

3.2.6 Trailer Chunk IEND 17

3.2.7 ancillary Chunks 17

3.2.8 Interlaced Modus 18

Filter 19

3.2.10 LZ77-Algorithmus 20

Beispiele 21

3.2.1 Browser Kompatibilität 21

4 Zusammenfassung 23

5 Abkürzungen 23

6 Literatur 23

1 Einleitung

Die Grafikformat GIF wurde Ende der 80er Jahre entwickelt, als der Bedarf für ein plattformunabhängiges Rastergrafikformat bestand, um Grafiken über Datennetze auszutauschen. So sollte es auch eine gute Kompression bieten, die mit Hilfe des LZW-Algorithmus erreicht wurde. Besonders im Internet hat sich GIF dadurch sehr stark durchgesetzt.
Nachdem die Firma Unisys 1994 Gebührenforderungen für den LZW-Kodieralgorithmus stellt, dessen Patent sie besitzt, entsteht die PNG Gruppe. Daraus geht das keine Patentrechte verletzende neue Grafikformat PNG hervor.

Historie

Ein paar wichtige Daten zur Geschichte der Grafikformate GIF und PNG:

1977 Abrahem Lempel und Jacop Ziv erfinden den Kompressionsalgorithmus LZ7
1978 Der Algorithmus LZ78 wird als Variante von LZ77 entwickelt.
1981 In den USA ist es nun möglich Software patentieren zu lassen.
Abrahem Lempel und Jacop Ziv patentieren ihren LZ78 Algorithmus.
1983 (20. Juni) Terry A. Welch (Sperry Corporation - später Unisys) patentiert das LZW-Kompressionsverfahren unter dem Titel „High speed data compression and decompression apparatus and method“, welches eine Variante des LZ78 Algorithmus ist. Die Ähnlichkeit, wird aber vom Patentamt nicht erkannt.
(weitere Varianten von LZ77, LZW wurden entwickelt und teils auch patentiert)
1987 (15. Juni) CompuServe veröffentlicht GIF (Version 87a) als freie und offene Spezifikation.
1989 Die Version GIF 89a wird von CompuServe vorgestellt.
1993 Unisys informiert CompuServe über die Verwendung ihres patentierten LZW-Algorithmus in GIF.
1994 (29. Dez.)Unisys gibt öffentlich bekannt, Gebühren für die Verwendung des LZW-Algorithmus einzufordern.
1995 (4. Jan.) Die PNG Gruppe wird in einigen Diskussionsforen gegründet.
(6. Jan.) Der Name PNG steht fest.
(7 Feb.) CompuServe kündigt volle Unterstützung für PNG an.
(7. März) Die ersten PNG Bilder werden ins Netz gestellt..
(8.Dez.) Das *World Wide Web Consortium* (W3C) veröffentlicht die PNG-Spezifikation 0.92 als offizielles Arbeitsdokument.
1997 Die Internetbrowser Netscape 4.04 und Internet Explorer 4.0 erscheinen mit PNG Unterstützung.
2003 Das GIF-Patent von Unisys läuft aus.

2 GIF

Das Grafikformat GIF hat sich besonders im Bereich des Internets sehr stark durchgesetzt, wofür auch die folgenden Eigenschaften verantwortlich sind.

2.1 Eigenschaften

GIF ist ein Raster Grafikformat mit einer Farbtiefe 1 bis 8 Bit (256 Farben) pro Einzelbild. Zusätzlich kann eine Farbe als transparent angegeben werden. Es besteht die Möglichkeit mehrere Einzelbilder in eine Datei zu speichern, womit sich z.B. Animationen erzeugen lassen. Das verwendete LZW-Kompressionsverfahren bietet eine verlustfreie Kompression. Mit dem einstellbaren Interlaced Modus ist zudem möglich noch während der Übertragung ein grobe Vorschau des Bildes anzuzeigen.

2.2 Dateiaufbau

Eine GIF-Datei besteht aus 3 Blöcken, dem Header, dem Bilddatenblock, und dem Terminator. Der Bilddatenblock kann mehrfach vorkommen, falls die GIF Grafik mehrere Einzelbilder enthält, wie sie z.B. bei Animationen eingesetzt werden. Zusätzlich können auch noch Erweiterungsblöcke (ab Version 89a) vorkommen, die sich zwischen zwei Bilddatenblöcken oder hinter dem letzten Bilddatenblock befinden dürfen.

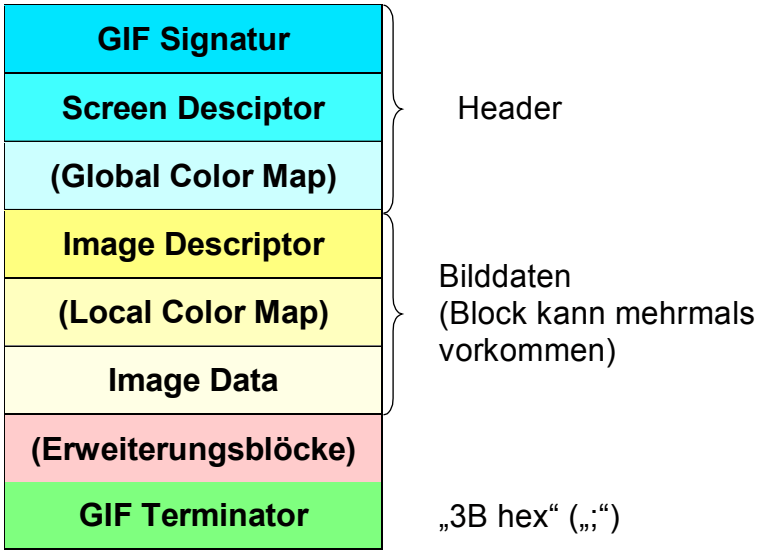


Abbildung 1: Dateiaufbau von GIF

2.2.1 Header

Der Header ist der Dateikopf. Hier werden alle grundsätzlichen Angaben zur Grafik gemacht, die für alle Teilbilder gelten sollen. Den Header besteht aus den Abschnitten: GIF-Signatur, Screen Descriptor und optionale globale Farbtabelle (Global Color Map).

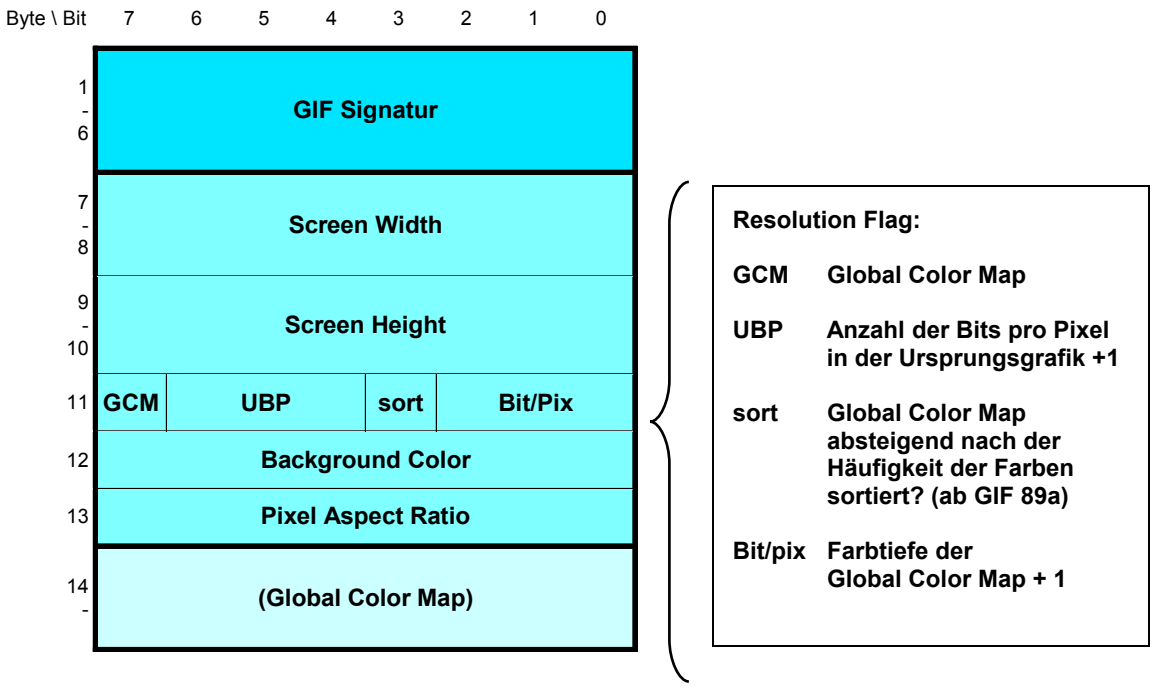


Abbildung 2: Aufbau des GIF Header

Jede GIF-Datei beginnt mit der **Signatur**. Diese besteht aus den ASCII Zeichen: „GIF87a“ oder „GIF89a“. die 87 bzw. 89 gibt dabei die verwendete GIF-Version an.

Nach der Signatur folgt der **Screen Descriptor**. Hier wird die Breite und Höhe des gesamten Bildschirms in Pixeln angegeben. Die Einzelbilder können innerhalb dieses definierten Bildschirms dargestellt werden.
Das nächste Byte ist das sogenannte Resolution Flag. Innerhalb dieses Bytes werden folgende Angaben gemacht:

1. ob eine globale Farbtabelle existiert
2. welche Farbtiefe die Ursprungsgrafik vor dem abspeichern als GIF hatte
   (Dieser Wert hat keine Wirkung auf die Darstellung.)
3. ob die globale Farbtabelle absteigend nach Häufigkeit der Farbe sortiert ist
   (Falls auf einem System weniger Farben als benötigt dargestellt werden können, ist es so möglich die am häufigsten benötigten Farben bevorzugt auszuwählen.)
4. Farbtiefe der globalen Farbtabelle in Bit/Pixel - 1

Anschließend wird der Index des Hintergrundfarbewertes für den Bildschirm in der globalen Farbtabelle angegeben. Falls keine globale Farbtabelle existiert, wird der Index in der Standard Farbtabelle verwendet.
Darauf folgt das Pixel-Abbildungs-Verhältnis (Pixel Aspect Ratio). Mit diesem Wert ist es möglich, das Verhältnis von Pixelhöhe und Pixelbreite nach folgender Formel anzugeben: Pixel-Abbildungs-Verhältnis = (Pixel Aspect Ratio + 15) / 64 . (Dieses Byte wird erst ab der Version 89a verwenden. Zuvor mussten alle Bits 0 sein.)

Falls vorhanden, folgt nun die **globale Fabtabelle**. Die einzelnen Farbwerte werden als 24-Bit-Farbwerte (RGB) hintereinander angegeben.

2.2.2 Bilddatenblock

Im Bilddatenblock werden die Angaben gespeichert, die nur das jeweilige Einzelbild betreffen. Auch dieser Block ist in drei Abschnitte gegliedert. Dem Image Descriptor, einer

optionalen lokalen Farbtabelle (Local Color Map) und den eigentlichen kodierten Pixelwerten.

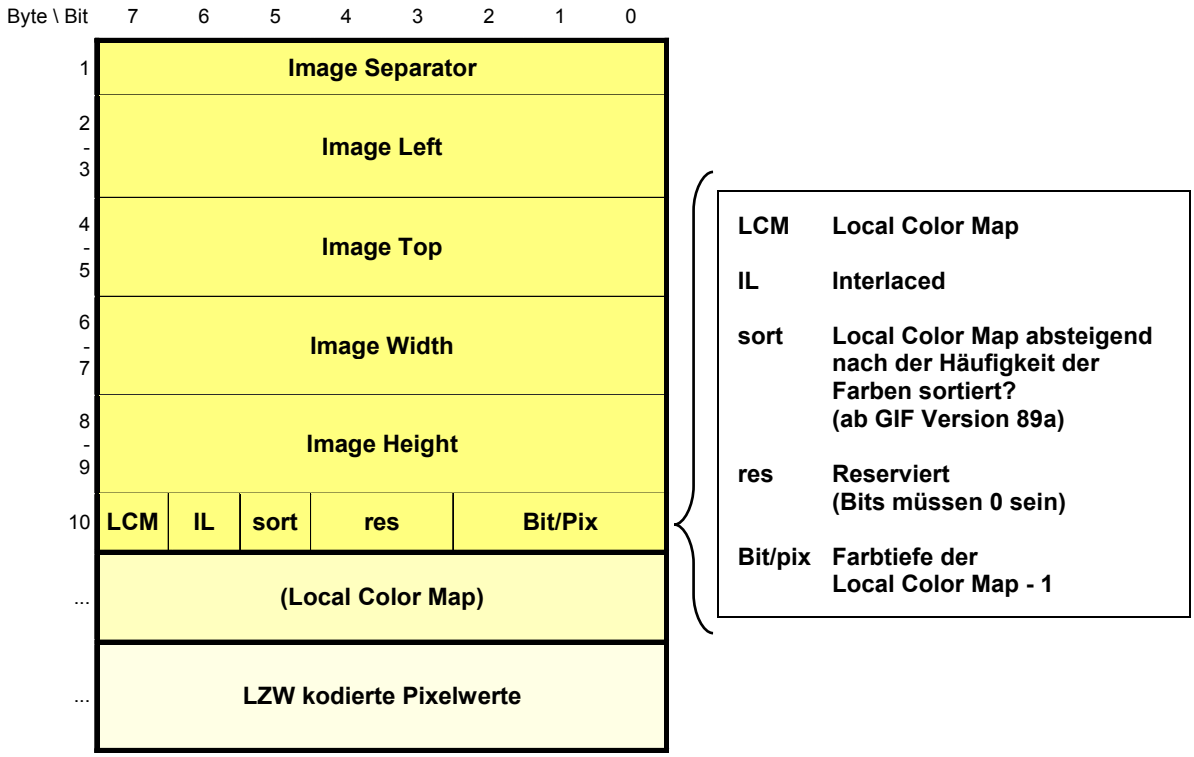


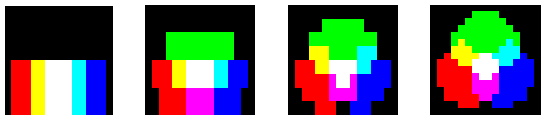
Abbildung 3: Aufbau des GIF Bilddatenblockes

Image Descriptor:
Jedes Einzelbild beginnt mit dem 1 Byte langen **Image Separator**, der aus dem hexadezimal Wert „2C“ (ASCII: „ , “) besteht. Dieses Byte gibt somit den Beginn eines neuen Einzelbildes an.
Danach folgen die Angaben Image Left und Image Top, die angeben, um wieviel Pixel das Einzelbild von der linken oberen Ecke auf dem GIF-Bildschirm verschoben angezeigt wird. Anschließend wird die Breite und Höhe des Einzelbildes definiert.
Das letzte Byte des Image Descriptors beinhaltet wiederum mehrere Parameter:
1. ob eine lokale Farbtabelle für dieses Einzelbild existiert
2. ob das Bild im Interlaced Modus abgespeichert wurde
3. ob die lokale Farbtabelle absteigend nach Häufigkeit der Farbe sortiert ist
4. zwei reservierte Bits (müssen 0 sein)
5. Farbtiefe der lokalen Farbtabelle in Bit/Pixel – 1

Falls vorhanden folgt nun eine **lokale Farbtabelle**, die nur für das jeweilige Einzelbild gilt, und eine eventuell vorhandene globale Farbtabelle ersetzt.

2.2.3 Interlaced Modus

Falls das Bild im Interlaced Modus abgespeichert wird, sind die Bildzeilen nach folgendem Schema umsortiert abgespeichert.



Zeile	Schritt 1	Schritt 2	Schritt 3	Schritt 4
0	1a			
1				4a
2			3a	
3				4b
4		2a		
5				4c
6			3b	
7				4d
8	1b			
9				4e
10			3c	
11				4f
12		2b		
13				4g
14			3d	
15				4h

Abbildung 4: die 4 Schritte des Interlaced Verfahrens bei GIF

Die Bildzeilen werden in 4 Schritten geladen:

1. jede 8. Zeile ab Zeile 0
2. jede 8. Zeile ab Zeile 4
3. jede 4. Zeile ab Zeile 2
4. jede 2. Zeile ab Zeile 1

So ist es möglich nach nur wenig übertragenen Daten schon eine grobe Vorschau des Bildes darstellen zu können.

2.2.4 Bilddaten

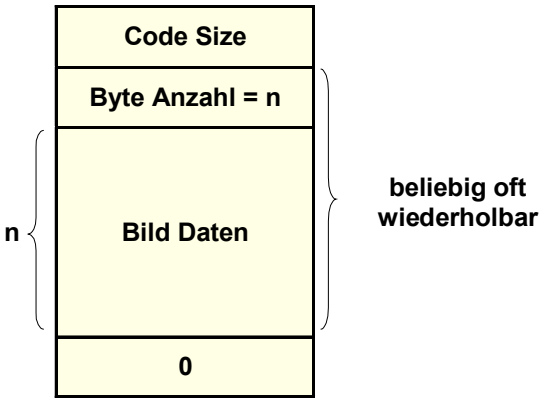


Abbildung 5: GIF Bilddaten (LZW komprimiert)

Das erste Byte der Bilddaten gibt die anfängliche Codelänge (Code Size) in Bits an. Normalerweise sind das die Anzahl der Bits pro Pixel (Farbtiefe) des Bildes. Außer bei Schwarz-Weiß Bildern, da muss der Wert 2 sein.
Danach folgen beliebig viele Datenblöcke mit den kodierten Pixelwerten beginnend von links oben. Jeder dieser Datenblöcke beginnt mit einem Byte welches die Länge der folgenden

Daten in Byte (1-255) ohne dieses Byte angibt. Sind die kodierten Daten länger als 155 Byte wird ein neuer Datenblock nach dem selben Schema angehängt.
Nach dem letzten Datenblock folgt ein Nullbyte welches das Ende der Datenblöcke und somit auch des gesamten Einzelbildes anzeigt.

2.2.5 LZW-Kompression

Die Pixelwerte werden mit Hilfe des LZW-Algorithmus komprimiert. Die Grundidee der LZW-Kompression ist wiederkehrende Zeichenketten durch kürzere Codes zu ersetzen.

Besonderheiten bei GIF:

Codevergabe:
0 ... 2^Code Size – 1 jeder Paletteneintrag erhält einen Code
2^Code Size „Clear Code“: Codetabelle wird auf den Anfangswert zurückgesetzt
2^Code Size + 1 „End of Information Code“: zeigt das Ende der LZW Daten an
2^Code Size + 2 ... neu eingetragene Codes

Die Codes haben eine variable Codelänge von 3 bis 12 Bits. Sind alle Codes innerhalb der aktuellen Codelänge vergeben muss ein Clear Code ausgeführt werden und die neue Codelänge wird um eins erhöht. Zwischen den Clear Codes haben alle Code die selbe Länge. Um Füllbits zu vermeiden sind die Codes nacheinander in Bytes gepackt (siehe Abbildung 6).

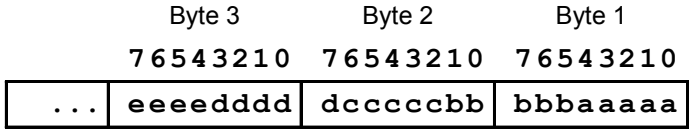


Abbildung 7: Codes in Bytes verpackt

Der eigentliche **LZW-Algorithmus** arbeitet wie folgt:

```
Encode:
Codetabelle initialisieren (jedes Zeichen erhält einen Code)
präfix = ” ”
while (Ende des Eingabedatenstroms noch nicht erreicht) {
     suffix := nächstes Zeichen aus dem Eingabedatenstrom
     muster := präfix + suffix
     if muster in Codetabelle then
          präfix := muster
     else
          muster in Codetabelle eintragen
          LZW-Code von präfix ausgeben
          präfix := suffix
}
if präfix nicht leer then
     LZW-Code von präfix ausgeben
```

	präfix	muster	suffix	Eingabedatenstrom	LZW Code	Ausgabe
0		A	A	ABCABCABCD		
1	A	AB	B	BCABCABCD	4:AB	0 (A)
2	B	BC	C	CABCABCD	5:BC	1 (B)
3	C	CA	A	ABCABCD	6:CA	2 (C)
4	A	AB	B	BCABCD		
5	AB	ABC	C	CABCD	7:ABC	4 (AB)
6	C	CA	A	ABCD		
7	CA	CAB	B	BCD	8:CAB	6 (CA)
8	B	BC	C	CD		
9	BC	BCD	D	D	9:BCD	5 (BC)
10	D					3 (D)

Initianlisierung:

Codetabelle: 0:A
1:B
2:C
3:D

Abbildung 8: Beispiel für den LZW Encode Vorgang

Decode:

```
Codetabelle initialisieren (jedes Zeichen erhält einen Code)
präfix = ” ”
while (Ende der Daten noch nicht erreicht) {
      lese LZW-Code
      muster := dekodiere (LZW-Code)
      gebe muster aus
      neuer LZW-Code := präfix + erstes Zeichen von muster
      präfix := muster
}
```

Code	präfix	muster	neuer Code	Ausgabe
0		A		A
1	A	B	4 = AB	B
2	B	C	5 = BC	C
4	C	AB	6 = CA	AB
6	AB	CA	7 = ABC	CA
5	CA	ABC	8 = CAB	BC
3	ABC	D	9 = BCD	D

Initianlisierung:

Codetabelle: 0:A
1:B
2:C
3:D

Abbildung 9: Beispiel für den LZW Decode Vorgang

Das Software-Patent der Firma Unisys beschränkt sich auf den Encode-Vorgang. Programme die diesen Algorithmus verwenden sind von den Gebührenforderungen betroffen. Der Decode-Vorgang bleibt Gebührenfrei.

2.2.6 Terminator

Dieses letzte Byte einer GIF Datei besteht aus dem hexadezimal Wert „3B“ (ASCII: „ ; “). Es zeigt das Ende der gesamten Datei an.

2.2.7 Erweiterungsblöcke

In der Version 89a wurden 4 verschiedene Erweiterungsblöcke definiert, die nach folgendem Schema aufgebaut sind (siehe Abbildung 10).

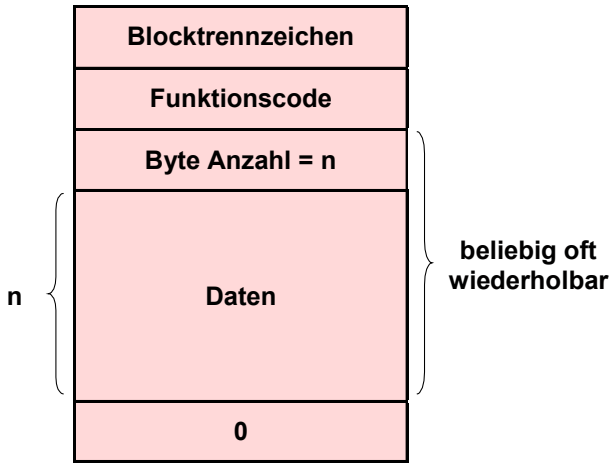


Abbildung 11: Aufbau der GIF Erweiterungsblöcke

Als Blocktrennzeichen wird der hexadezimal Wert „21“ (ASCII: „!“) verwendet. Anschließend folgt der Funktionscode, der die Art des jeweiligen Erweiterungsblockes bestimmt. Das nächste Byte gibt die Länge der folgenden Daten in Byte an. Falls diese länger als 255 Byte sind, wird ein weiterer Datenblock angehängt. Das Ende eines Erweiterungsblockes wird mit einem Nullbyte gekennzeichnet.

Folgende Erweiterungsblöcke wurden in der Version 89a definiert:

Plain Text Extension: (01 hex)	kann beliebigen ASCII Test als Grafik auf der Bildfläche darstellen
Comment Extension: (FE hex)	Kommentare (z.B.: Autor ...) als ASCII Text mitspeichern aber nicht im Bild anzeigen
Graphic Control Extension: (F9 hex)	gibt an wie nach der Anzeige des folgenden Bilddatenblocks weiter verfahren werden soll (z.B.: Animationsgeschwindigkeit)
Application Extension: (FF hex)	anwendungsspezifische Daten für eigene Erweiterungsblöcke

Innerhalb des Datenfeldes gibt es für die einzelnen Erweiterungsblöcke definierte Strukturen, die in der Spezifikation genauer nachgelesen werden können.

2.3 Beispiele

Abbildung 12: Beispiel einer GIF Datei

47 49 46 38 39 61	Gif Signatur = „GIF89a“
10 00 10 00 A2 00 00	Gesamt Breite (hier 16) Gesamt Höhe (hier 16) = 1 010 0 010 Background Color Pixel Aspect Ratio Farbtiefe (hier 3) Farbpalette sortiert (hier nein) Farbtiefe der Ursprungsgrafik (hier 3) globale Farbtabelle (hier ja)
00 00 00 00 00 FF 00 FF 00 00 FF FF FF 00 00 FF 00 FF FF FF 00 FF FF FF	globale Farbtabelle in RGB Werten
2C 00 00 00 00 10 00 10 00 00	Image Seperator Image Left (hier 0) Image Top (hier 0) Image Width (hier 16) Image Height (hier 16) = 0 0 0 00 000 Farbtiefe der LCM Farben sortiert (hier nein) Interlaced abgespeichert reservierte Bits lokale Farbtabelle (hier nein)
04 46 10 C8 49 AB 05 2A AB 4B B5 E6 98 F7 59 A2 48 2E A5 B3 55 4B EB 39 70 95 B4 ED F3 C0 30 33 25 7C 6D DF 0E 86 50 C2 2B FE 1E C2 24 B1 98 68 D8 1A C9 21 80 D9 6C 58 A3 BA 25 CF 7A 55 EE 98 5D 2F A5 28 11 83 38 11 00	LZW komprimierte Bilddaten
3B	Terminator

3 PNG

PNG wurde in erster Linie entwickelt um ein Grafikformat zu haben, welches nicht von Patentrechten und den damit verbundenen Lizenzgebühren betroffen ist. So soll es auch eine Ausweichmöglichkeit für GIF bieten. Es wurden aber noch zahlreiche Features spezifiziert, die über die Spezifikation von GIF hinausreichen und den mit der Zeit veränderten Bedürfnissen und Möglichkeiten angepasst sein sollten.

3.1 Eigenschaften

Auch PNG ist ein für Datennetze optimiertes Raster-Grafikformat. Es deckt alle Eigenschaften von GIF ab, bis auf die Möglichkeit mehrere Einzelbilder (für Animationen) in einer Datei zu speichern. Dafür wurde das spezielle Format MNG (ehemals PNF) entwickelt welches PNG sehr ähnlich ist, jedoch zahlreiche Animationsfähigkeiten bietet.
Mit einer Farbtiefe von bis zu 48 Bit (RGB) soll es echte True Color Bilder ermöglichen. Ebenso sind aber auch Palletten-Bilder mit bis zu 8 Bit (256 Farben), sowie Graustufen-Bilder mit bis zu 16 Bit möglich.
Neben der transparenten Farbe wie in GIF ist bei PNG möglich echte Alphakanäle für eine Stufenlose Transparenz mitzuspeichern. Für jedes Pixel kann so ein Transparenzwert von 16 Bit gespeichert werden, der die Angabe über die Transparenz von komplett deckend bis komplett transparent macht.
Für die Komprimierung wird der LZ77 Algorithmus verwendet, welcher ebenfalls eine verlustfreie Komprimierung ermöglicht, und nicht von Lizenzforderungen betroffen ist.
Auch ein Interlaced Modus ist möglich, wird aber etwas anders als in GIF realisiert.
Das frei definierte Chunk Layout macht PNG leicht erweiterbar für individuelle Zwecke.

3.2 Dateiaufbau

Eine PNG Datei besteht aus mehreren einzelnen Chunks, von denen einige alle Decoder interpretieren können müssen („critical chunks“) und weitere die optional vorhanden sein können („ancillary chunks“) und falls sie nicht ausgewertet werden können eine Bildausgabe trotzdem möglich ist.

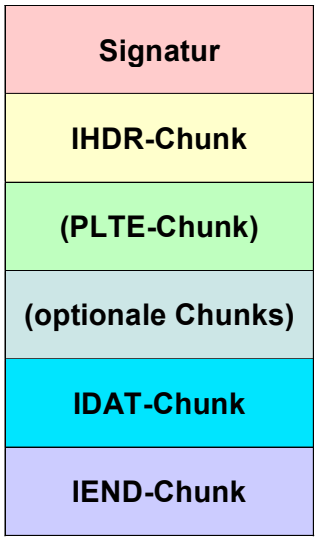


Abbildung 13: PNG Dateiaufbau

Auf jeden Fall vorhanden sein müssen, die Signatur, der Header Chunk (IHDR), der Data Chunk (IDAT) sowie der Trailer Chunk (IEND). Eventuell kann auch noch ein Palette Chunk (PLTE) sowie weitere optionale Chunks vorkommen. Diese optionalen Chunks müssen sich teilweise an bestimmten Stellen innerhalb der PNG-Datei befunden, in jedem Fall aber zwischen Header und Trailer.

3.2.1 Signatur

Jede PNG-Datei muss mit der 8 Byte langen Signatur beginnen. Anders als bei GIF dient diese nicht nur der Dateiart- und Versionsangabe, sondern hat ein paar Kontrollfunktionen integriert.

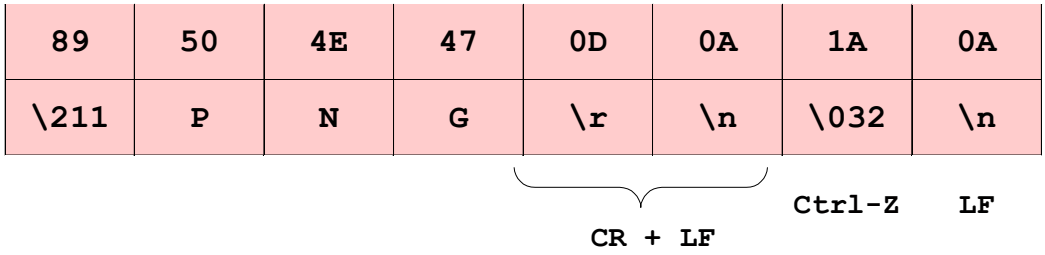


Abbildung 14: Aufbau der PNG Signatur

Das erste Byte testet, ob die Datei durch ein 8-Bit fähiges Datennetz gesendet wurde, ansonsten wäre schon dieses Byte falsch und die weitere Ausgabe kann abgebrochen werden. Anschließend folgen die drei ASCII Zeichen „PNG“, die die Dateiart klarstellen. Die nächsten beiden Bytes sind die Steuerzeichen Carriage Return (Wagenrücklaufsymbol) und Line Feed (Zeilenvorschubsymbol). Dieses Zeichen könnten auch in den Datenbytes vorkommen und müssen deshalb unverändert übertragen werden. Das 7. Byte stoppt die Ausgabe unter DOS, und das letzte Byte der Signatur überprüft nochmals ob auch einzelne Line Feed Symbole unverändert gesendet werden.

Chunk Layout

Bis auf die Signatur sind alle Blöcke (Chunks) nach der gleichen Grundstruktur aufgebaut.

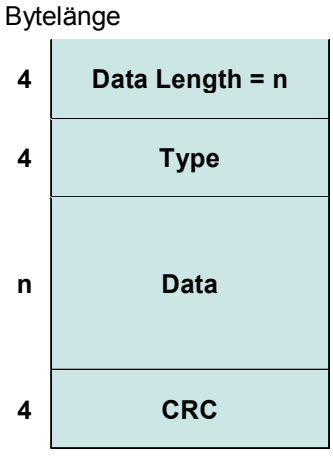


Abbildung 15: PNG Chunk Layout

Die ersten 4 Bytes geben die Länge des Chunks ohne Data Length, Typkennung und CRC-Prüfsumme in Bytes an. So kann ein eventuell für den Encoder unbekannter Block ohne Probleme übersprungen werden.
Anschließend folgt die ebenfalls 4 Byte lange Typkennung. Diese besteht in der Regel aus 4 ASCII Zeichen, wobei das 5. Bit jedes Zeichens für einige Zusatzparameter benutz wird, die nach folgender Konvention vergeben werden Dieses Bit entscheidet auch über die Groß-Kleinschreibung, was das Auswerten vereinfacht.

	Zeichen			
	1	**2**	**3**	**4**
Großbuchstabe	**critical**	**public**	**reserv.**	**not copy**
Kleinbuchstabe	**ancillary**	**privat**	**X**	**copy**

Abbildung 16: Konvention der Typkennung

Ist der erste Buchstabe ein Großbuchstabe, so handelt es sich um einen critical Chunk, ansonsten ist es ein anciallary Chunk. Der zweite Buchstabe gibt an ob der Block öffentlich oder privat ist. Das dritte Zeichen ist in der aktuellen Spezifikation noch unbenutzt und reserviert. Es sollte Momentan aber als Großbuchstabe vergeben werden. Der letzte Buchstabe gibt dem PNG Encoder darüber Auskunft, ob der Block vom Inhalt eines kritischen Blocks abhängt, und ohne Anpassung nicht einfach kopiert werden darf.
Nach der Typkennung folgen die eigentliche Daten, die in jedem Blocktyp anders definiert sein können.
Die letzten 4 Byte eines jeden Chunks, sind eine CRC-Prüfsumme über die Typkennung und die Datenbytes. Folgendes Prüfpolynom wird dabei verwendet:
$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

3.2.3 Header Chunk IHDR

Der Start Block (Header Chunk) muss direkt an die Signatur anschließen. Er beinhaltet mehrere grundlegende Informationen über das PNG-Bild.

Bytelänge	
4	**Data Length = 13**
4	**„IHDR“**
4	**Width**
4	**Height**
1	**Bit depth**
1	**Color Type**
1	**Compression method**
1	**Filter method**
1	**Interlace methode**
4	**CRC**

Abbildung 17: Aufbau des Header Chunks (IHDR)

Die Typkennung „IHDR“ zeigt an, dass es sich um einen kritischen, öffentlichen Block handelt, der nicht ohne eventuelle Anpassung an veränderte Bilddaten kopiert werden darf.
Nach der Typkennung folgen Angaben über Bildbreite, Bildhöhe, Länge der Bitmuster (1,2,4,8 oder16 je nach Farbtyp) und Farbtyp. Der Farbtyp wird nach folgendem Muster angegeben:

	0	1
Bit 1	ohne Palette	mit Palette
Bit 2	Graustufen	RGB Farbe
Bit 3	kein Alpha Kanal	mit Alpha Kanal

Abbildung 18: Farbtyp-Byte Belegung

Desweiteren werden auch noch Angaben über die verwendeten Kompressionsmethode (bisher spezifiziert: 0 = Deflate-Algorithmus), über den verwendeten Filter (0 = adaptive Filterung) und die Interlaced Methode (0 = unverschachtelt; 1 = Adam 7).

3.2.4 Palette Chunk PLTE

Optional kann auch in PNG-Bildern eine Palette angegeben werden. Dieser Paletten Block (Palette Chunk) kann auch angegeben werden, wenn es sich um ein True Color Bild handelt. Die Palette kann damit als Angabe zur bestmöglichen Farbreduktion dienen, falls auf dem System nicht die erforderliche Farbtiefe zu Verfügung steht.

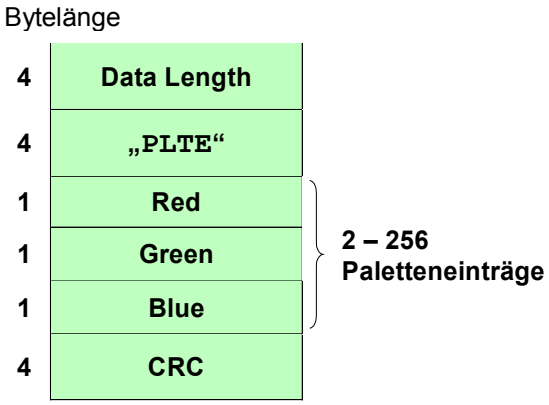


Abbildung 19: Aufbau des Palette Chunks (PLTE)

Innerhalb der Palette können 2 bis 256 RGB Farbwerte vorkommen. Reine Graustufenpaletten sind nicht möglich.

Data Chunk IDAT

Innerhalb des Daten Blocks (Data Chnuk) sind die eigentlichen Pixelwerte gespeichert. Bei Palettenbildern sind es Verweise auf den zugehörigen Paletteneintrag, ansonsten die tatsächlichen Farbwerte bzw. Graustufenwerte. Diese Pixelwerte sind mit dem LZ77-Algorithmus (siehe 3.2.10) gepackt.

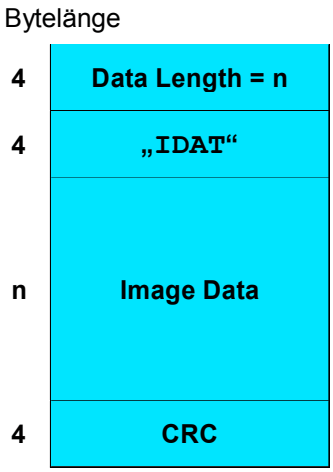


Abbildung 20: Aufbau des Data Chunks (IDAT)

Die einzelnen Pixelwerte werden in Bytes gepackt um keine Bits zu verschwenden.
Sind die Bilddaten länger als 2^32 Bytes werden die restlichen Bilddaten in einen weiteren Daten Block gepackt. Es können beliebig viele Daten Blöcke aufeinander folgen.

3.2.6 Trailer Chunk IEND

Der 12 Byte große Ende Block (Trailer Chunk) enthält keine Daten. Er dient nur dazu das Ende der PNG-Datei anzuzeigen.

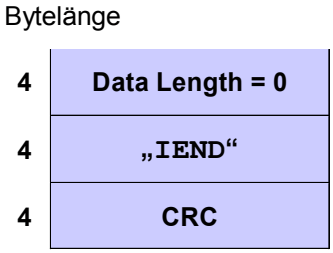


Abbildung 21: Aufbau des Trailer Chunks (IEND)

3.2.7 ancillary Chunks

Neben den vier kritischen Blöcken gibt es auch noch eine Reihe von Zusatzblöcken (ancillary Chunks). Einige spezielle sind Teil der Spezifikation, aber es können nach dem bereits beschriebenen allgemeinen Chunk Layout auch beliebige eigene Chunks definiert und benutzt werden.

Folgende Liste zeigt ein paar der definierten ancillary Chunks:

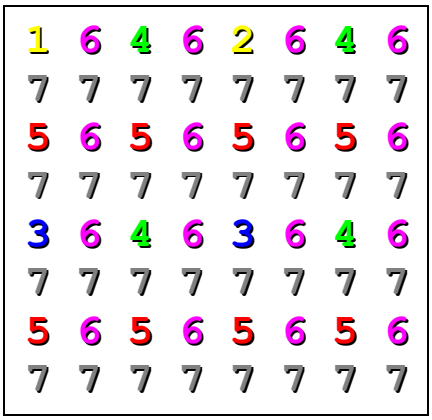
Typkennung	**Name**	**Beschreibung**
tIME	last modification	Datum der letzten Veränderung
bKGD	Background	Hintergrundfarbe (Paletteneintrag, Grau- oder RGB-Wert)
tRNS	Transparency	Alphawerte für einzelne Farben/Paletten
sBit	Significant Bits	Bittiefe vor dem Speichern und nötige Umrechnung

gAMA	Image gama	Heiligkeitswerte beim Erstellen
cHRM	chromaticities	Chromaticity und White Point (RGB Spezifikation)
hIST	histogram	Häufigkeit der Farbwerte (nur bei Palette)
pHYS	physical pixel	Abmessungen der Pixel
sCAL	Physical scale	Größe des abgebildeten Objekts (Karten, Medizin, ...)
tEXt	Textual Data	unkomprimierter Text (Autor, Copyright ...)
zTXt	compr. Text	komprimierter Text (für lange Texte)
gIFg	GIF Control Extension	GIF-Kontrollwerte
gIFx	GIF Aplication Extension	GIF-Erweiterungsblöcke

Weitere Zusatzblöcke wurden definiert und können genauer in der aktuellen Spezifikation nachgelesen werden.

3.2.8 Interlaced Modus

Für den Interlaced Modus ist momentan als einzige Methode das Adam 7 Verfahren spezifiziert.
Dieses Verfahren lädt das Bild in 7 Schritten nach folgenden Muster:



Das komplette Bild wird dazu in 8x8 Blöcke unterteilt. Innerhalb dieser Blöcke werden die Pixel in der nebenstehenden Reihenfolge geladen und gespeichert.
Das Adam 7 Verfahren liefert somit nach einer noch kürzeren Übertragungszeit wie bei dem Zeilenweisen Interlaced Modus aus GIF eine grobe Vorschau des Bildes.

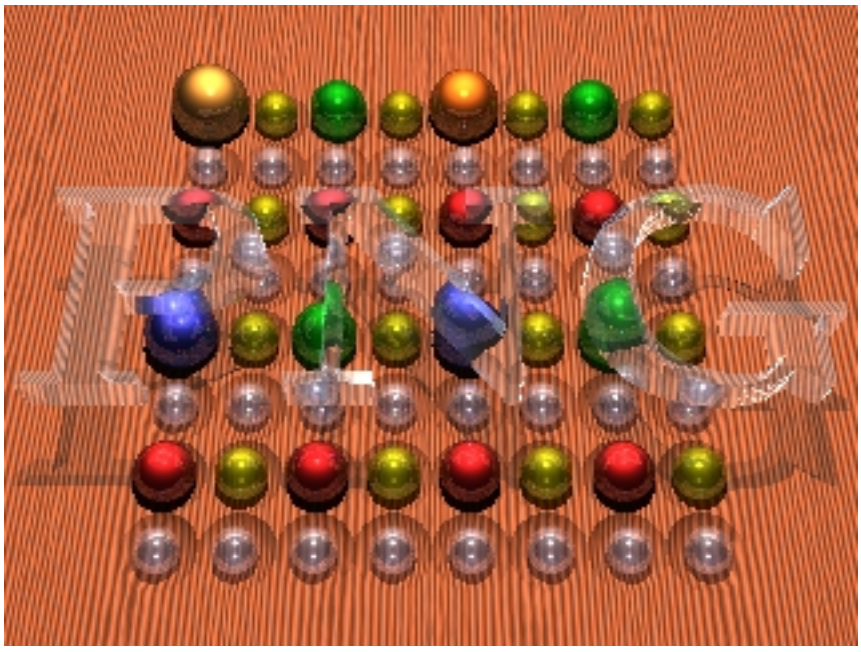


Abbildung 22: PNG Adam 7 Verfahren (Veranschaulichung aus der Spezifikation)

Filter

Noch vor dem Kodieren der Pixel mit dem LZ77-Algorithmus, können über die einzelnen Bildzeilen Filter angewendet werden, um die Kompressionsrate zu verbessern. Die Kompression wird effektiver wenn sie nicht über viele sehr unterschiedliche Pixelwerte angewendet wird, sondern statt dessen über die Differenz zu vorherigen Pixeln. So entstehen in vielen Fällen (z.B. bei Farbverläufen) viele gleiche Werte, In deren Folgen so auch größere wiederkehrende übereinstimmende Muster zu finden sind. Auch die Anwendung dieser Filter ist verlustfrei.
Welche Filtermethode in der Zeile verwendet wurde, wird in einem vor jeder Zeile mitgespeichertem Byte, dem Filtertypindikator angegeben.

In PNG sind folgende Filtermethoden definiert:

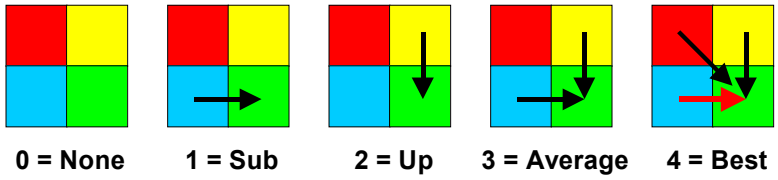


Abbildung 23: Filtermethoden

0 = keine Filterung
1 = Differenz zum links benachbarten Pixelwert
2 = Differenz zum darüber liegenden Pixelwert
3 = Durchschnitt aus dem linken und darüber liegenden Pixelwert
4 = kleinste Differenz zum linken, darüber oder links-darüber liegenden Pixelwert

3.2.10 LZ77-Algorithmus

In PNG wird der Deflate Algorithmus mit 32kBit Gleitfenster (Sliding Window) zur Komprimierung der Bilddaten verwendet. Dieser ist eine Variante des LZ77 Algorithmus, dem ältesten Kompressions Algorithmus der Lempel-Ziv Familie.
Er arbeitet ähnlich wie LZW nach dem Prinzip wiederkehrende Zeichenketten zu erkennen, ersetzt diese aber nicht durch Codes, sondern speichert einen Zeiger auf die zuletzt gefundenen Position dieser Übereinstimmung.

Encode:

```
Kodierungsposition := Anfang des Eingabedatenstroms
while (Ende des Eingabedatenstroms noch nicht erreicht) {

      finde die längste Übereinstimmung im Datenfenster mit
      der an der Kodierungsposition beginnenden Zeichenkette

      gebe den Zeiger auf die Übereinstimmung Datenfenster und
      das erste nicht passende Zeichen im Vorschaupuffer aus

      Kodierungsposition += Länge der Übereinstimmung + 1
}
```

Pos	Sliding Window	Vorschaupuffer	Pos, ÜS	Ausgabe
0		AABCAAABC		(0,0)A
1	A	ABCAAABC	0,A	(0,1)B
3	AAB	CAAABC		(0,0)C
4	AABC	AAABC	0,AA	(0,2)A
7	AABCAAA	BC	3,BC	(2,1)C

Abbildung 24: Beispiel für den LZ77 Algorithmus

Beispiele

Abbildung 25: Beispiel einer PNG Datei

89 50 4E 47 0D 0A 1A 0A		Signatur
00 00 00 0D 49 48 44 52 00 00 00 10 00 00 00 10 08 03 00 00 00 28 2D 0F 53	Data Length = 13 Byte "IHDR" Breite = 16 Pixel Höhe = 16 Pixel Länge der Bitmuster = 8 0011 → kein Alpha-Kanal, Farbbild, mit Palette 0: Deflate Komprimierung 0: adaptive Filterung 0: unverschachtelt CRC	IHDR
00 00 00 07 74 49 4D 45 07 D2 0C 0A 17 19 3B 0A 0D EF 28		tiME
00 00 00 09 70 48 59 73 00 00 0B 12 00 00 0B 12 01 D2 DD 7E FC		pHYs
00 00 00 04 67 41 4D 41 00 00 B1 8F 0B FC 61 05		gAMA
00 00 00 18 50 4C 54 45 00 00 00 FF 00 00 00 00 FF FF 00 FF FF FF FF FF FF 00 00 FF FF 00 FF 00 09 B4 2B E0		PLTE
00 00 00 4E 49 44 41 54 78 DA 6D CA 5B 0E 00 21 08 43 D1 82 3C F6 BF 63 51 E2 08 66 FA 77 4F 0A FC CF 63 6F 57 71 EF E2 FE 88 6B A6 1D 50 DD 62 66 D9 14 A0 22 D1 C6 BB 89 56 2F 61 E6 04 A2 05 CC 05 86 C8 E8 10 3B 80 0A C0 BD 7C 87 24 E0 66 DF 04 5A B2 02 38 83 63 D1 5B		IDAT
00 00 00 00 49 45 4E 44 AE 42 60 82		IEND

3.2.1 Browser Kompatibilität

PNG Bilder werden zwar von allen aktuellen Internet-Browsern dargestellt, aber diese Darstellung geschieht verschieden gut. Besonders der Alphakanal wird nicht immer richtig interpretiert.

Abbildung 26: MS Internet Explorer 5.5 und 6.0

Abbildung 27: Netscape Navigator 4.75

Abbildung 28: Netscape Navigator 6.1 und 7.0, Opera 6.05, Mozilla1.1

4 Zusammenfassung

Das Grafikformat PNG sollte ein Ersatz für das von Lizenzgebühren betroffene GIF sein. Ersetzen konnte es das sehr weit verbreitete GIF nicht, aber es ist ein sehr gute Alternative, durch die zahlreichen Funktionen die über die Möglichkeiten von GIF hinausgehen. Besonders die verlustfreie Kompression auch von True Color Bildern sowie die Alphakanäle bieten eine gute Möglichkeit Fotos ohne Kompressionsverluste wie sie z.B. bei JPG auftreten abzuspeichern. Die Dateigröße ist dabei meistens größer als bei JPG aber doch geringer als TIF.
Dass PNG nicht GIF einfach ersetzten sollte, sondern ein neues leistungsfähigeres Grafikformat sein sollte wird auch durch die interne Namenserklärung „PiNG is not GIF“. Die Aussprache als „PiNG“ steht aber mittlerweile offiziell in der Spezifikation.

5 Abkürzungen

GIF	Graphics Interchange Format
LZ77	Lempel-Ziv-(Kompression) von 1977
LZW	Lempel-Ziv-Welch-(Kompression)
MNG	Multiple Network Graphics (ehemals PNF)
PNF	Portable Network Frame (später MNG)
PNG	Portable Network Graphics („PiNG“)
RGB	Rot-Grün-Blau Farbwerte

6 Literatur

[1] CompuServe Inc.: GIF 87a Specification; 1987; http://www.wotsit.org

[2] CompuServe Inc.: GIF 89a Specification; 1990; http://www.w3.org/Graphics/GIF/spec-gif89a.txt

[3] Thomas W. Lipp: Das Graphics Interchange Format (GIF); in: Grafikformate; 1997, Deutschland; S. 277 ff. (ISBN: 3-86063-391-0)

[4] Andreas Kunz: Das Grafikdateiformat GIF; in: Proseminar Redundanz; 1998/1999; Universität Karlsruhe; http://goethe.ira.uka.de/seminare/redundanz/vortrag10/

[5] Maximilian Hrabowski: Lempel Ziv; in: Proseminar Redundanz; 1998/1999; Universität Karlsruhe; http://goethe.ira.uka.de/seminare/redundanz/vortrag05/

[6] World Wide Web Consortium: PNG Specification Version 1.0; 1996; http://www.w3.org/Graphics/PNG/

[7] Greg Roelofs: PNG home page; 2003; http://www.libpng.org/pub/png/

[8] Max Völkel: Das Grafikdateiformat PNG; in: Proseminar Redundanz; 1998/1999; Universität Karlsruhe; http://goethe.ira.uka.de/seminare/redundanz/vortrag12/