Als die Bilder laufen lernten

FLI-Player für Turbo Pascal **Stephan Brumme**

Kein kommerzielles Spiel kommt heute mehr ohne sie aus: die Animation. Doch die schnelle Wiedergabe selbst auf kleineren Rechnern ist gar nicht so schwer, wenn man weiß, wie's geht. Deshalb wird hier eine Turbo-Pascal-Unit für das weitverbreitete FLI-Format vorgestellt.

Der präsentatorische Effekt einer Animation ist unbestritten. So hat schon mancher Programmierer mit der Idee gespielt, eine solche in sein Projekt einzubinden. Oft scheiterte dies an der Geschwindigkeit oder dem Platzbedarf. Einen guten Kompromiß zwischen beiden Problemen stellt das FLI-Format dar. Es wurde ursprünglich für den Autodesk Animator entwickelt, findet jedoch auch in der Shareware-szene aufgrund der vielen Vorteile reichhaltig Anwendung. Selbst viele kommerzielle Programme benutzen dieses Format.

Voraussetzungen

Der Großteil der Unit wurde in optimiertem Assembler geschrieben. Daher ist die Geschwindigkeit sehr hoch. Selbst ein 286er/6 MHz genügte bei Tests den Ansprüchen von 18,2 Bildern pro Sekunde. Jedoch sollten zwei Dinge beachtet werden: 1. Das Programm benötigt mindestens einen 286er, überprüft aber nicht dessen Existenz. 2. Ebenfalls wird eine VGA-Grafikkarte vorausgesetzt, was aber auch nicht überprüft wird.

Die FLI-Unit wurde in Turbo Pascal 6.0 mit integriertem Assembler geschrieben. Der Quellcode wurde stark optimiert, er umfaßt nur 1.632 Byte, und braucht daher nicht verändert zu werden. Deshalb gehe ich nicht auf Einzelheiten des Quellcodes ein, sondern auf seine Funktion.

Komprimieralgorithmus

Ein großes Problem bei einer Animation ist der Platzbedarf. So benötigt man für die 10-Sekunden-Beispielanimation in der verwendeten Auflösung von 320 x 200 Pixel unkomprimiert satte 34,7 MB. Das ist jedoch entschieden zu viel. Also versucht man, „überflüssige" Daten zu entfernen. Dazu ein Beispiel: Bewegt sich eine Figur vor ständig gleichem Hintergrund, so braucht man nur einmal den gesamten Hintergrund abzuspeichern. In jedem neuen Bild werden bloß noch die Veränderungen der Figur festgehalten. Dieser Trick spart schon sehr viel Platz.

Doch das FLI-Format geht noch einen Schritt weiter. So enthält fast jedes Bild Informationen, die man kürzer darstellen kann. Dazu dient die sogenannte RLE-Codierung. Diese macht sich die häufig auftretende Eigenschaft zunutze, daß oft gleiche Farben mehrmals hintereinander vorkommen. Man speichert nun nur noch die Anzahl der Wiederholungen und einmal den Farbwert. So wird aus der Folge „22222" (5 Byte) die viel kürzere „5 mal 2" (2 Byte). Dieses Verfahren ist zwar bei weitem nicht so effektiv wie ARJ oder PKZIP, jedoch rasend schnell.

Das FLI-Format ist mit diesem sehr einfachen Algorithmus in der Lage, die Beispielanimation von 11,4 MB auf 1,0 MB zu verkürzen. Das entspricht einer Komprimierung auf nur 3 Prozent der Ursprungsgröße!

Im folgenden jetzt der genaue Aufbau des FLI-Formats. Bitte beachten Sie dazu die Tabellen und Textboxen.

Nägel mit Köpfen: Verwaltungsinformationen

Jede FLI-Datei beginnt mit einem Dateikopf. Der genaue Aufbau ist in der Tabelle beschrieben. Danach beginnen die einzelnen Bilder. Diese besitzen ebenfalls je einen Bildkopf. Auch dessen Aufbau entnehmen Sie bitte der Tabelle. Das ist aber nicht die niedrigste Verwaltungsebene. Jetzt folgen noch die sogenannten Chunks. Jeder beginnt auch mit einem Chunkkopf. Dieser ist auch tabellarisch dokumentiert.

Wozu dient der Grafikmodus 13hex ?

Das FLI-Format wurde für den Grafikmodus 13hex entwickelt. Dieser wurde erst auf der VGA-Grafikkarte implementiert. Er ist in der Lage, bei einer Bildschirmauflösung von 320 x 200 Punkten 256 Farben gleichzeitig darzustellen. Ein wichtiger Vorteil ist jedoch die Adressierung der einzelnen Punkte.

Das Segment des Videospeichers ist stets A000hex. Der entsprechende Offset berechnet sich nach der Formel 320*Y+X. Das Koordinatensystem der Grafikkarte weicht vom mathematischen Modell etwas ab.

Der Punkt (0/0) befindet sich in der oberen linken Ecke, der Punkt (319/199) in der unteren rechten Ecke.

Ganz kurz die wichtigsten Prozeduren für den Grafikmodus 13hex zusammengefaßt:

```
Procedure SetMode13;
Assembler;
asm mov ax,13h; int 10h end;

Procedure PutPixel(x,y: word;
farbe: byte);

Begin
Mem[$A000:320*y+x]:=farbe;End;
```

FLI-Player für Turbo Pascal

THeader-Record - der Dateikopf

Variable	Variablentyp	Bedeutung
Groesse	LongInt	Länge der kompletten FLI-Datei
Kennung	Word	ist bei FLI-Dateien stets AF11hex
Bilder	Word	Anzahl der Einzelbilder der Animation
MaxX	Word	Bildbreite, ist stets 320
MaxY	Word	Bildhöhe, ist stets 200
Farbbits	Word	Bits pro Farbe, ist stets 8, da 2^8=256
Flags	Word	sollte stets 0 sein
Geschwindigkeit	Word	Wiedergabegeschwindigkeit in 1/70 Sekunden
Fuell	Array[1..110] of Byte	füllt den Dateikopf auf 128 Byte auf

Farbspielerei: Der Color-Chunk

Der Color-Chunk setzt sich aus Einheiten zusammen. Jede dieser Einheiten enthält RGB-Werte. Das erste Wort des Chunks gibt die Anzahl der Einheiten an. Gleich darauf folgen diese. Jede Einheit beginnt mit einem Skip-Byte. Dieses gibt an, wieviel Farben zu überspringen sind, da sie unverändert bleiben. Danach folgt das Größenbyte, das die Anzahl der veränderten RGB-Werte darstellt. Erst jetzt folgen die eigentlichen RGB-Werte.

Einstellung der Farbtabelle

Die Standardfarbtabelle reicht für die meisten Animationen nicht aus. Eine Neudefinition ist unumgänglich. Das Programm bedient sich dabei nicht des BIOS, sondern programmiert die VGA-Grafikkarte direkt.
Den sogenannten DAC-Baustein findet man an zwei direkt aufeinanderfolgenden Ports. Der Indexport liegt dabei an Port 3C8hex. Ihm wird mitgeteilt, welcher Eintrag der Farbtabelle zu ändern ist. Die Daten werden am Port 3C9hex in der Reihenfolge Rot, Grün, Blau geschrieben.

Wie die Abkürzung RGB schon erkennen läßt, folgt zuerst der Rot-, dann der Grün- und zuletzt der Blauanteil. Sie sind jedoch auf den Bereich 0 bis 63 begrenzt. Also sind maximal 64^3= 262.144 Farbkombinationen möglich.

Diese Zahl reicht schon bei weitem aus, da das menschliche Auge nur in der Lage ist, 10.000 verschiedene Farben zu erkennen. Diese recht komplizierte Methode ist jedoch selten anzutreffen. Meist enthält das Größenbyte den Wert 0. Dann ist die komplette Farbpalette neu zu laden.

Gewisse Unterschiede: Der LC-Chunk

Am häufigsten wird der LC-Chunk verwendet. Er beschreibt Unterschiede zwischen zwei Bildern. Das verwendete Komprimierverfahren ist recht effektiv und reicht sogar für Double-Speed-CD-ROMs aus. Nur von Diskette ist der PC nicht in der Lage, volle Geschwindigkeit zu erreichen.

Das erste Wort enthält die Nummer der ersten Zeile, die sich vom vorherigen Bild unterscheidet. Dabei beginnt die Zählung bei 0. Gleich danach folgt wieder ein Wort, das allerdings die Anzahl der Zeilen angibt, die verändert werden. Diesmal beginnt die Zählung bei 1.

Am Anfang jeder Zeile steht ein Einheitenbyte. Es enthält die Anzahl der Einheiten je Zeile. Manchmal ist dieser Wert 0, in dieser Zeile wird also nichts verändert.

Jede einzelne Einheit wiederum beginnt mit einem Skip-Byte (skip, engl. überspringen). Dieses sagt dem Programm, wieviel Punkte unverändert bleiben, also übersprungen werden können. Jetzt folgt das Größenbyte: Wenn seine Bit 7 gleich 0 ist, werden soviel Bytes vom Puffer in den Videospeicher kopiert, wie das Größenbyte angibt. Ist das Bit 7 jedoch gleich 1, muß das Zweierkomplement gebildet werden. Es wird nach dem Term (NOT Größenbyte)+1 berechnet. Jetzt folgt nur ein Farbwert. Dieser wird der berechneten Anzahl entsprechend oft in den Videospeicher kopiert.

Diese Komprimierung ist in den meisten Fällen sehr gut. Bei komplexen Bildern kann jedoch der Fall eintreten, daß die komprimierten Informationen länger sind als das ursprüngliche Bild. Dann wird der Copy-Chunk verwendet. Der Black-Chunk wird fast nie verwendet. Der gesamte Videospeicher wird auf 0, das heißt schwarz gesetzt.

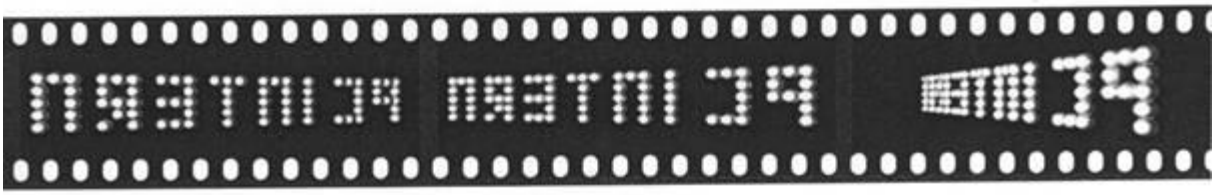
Voll im Bilde: der BRun-Chunk

Beim ersten Bild einer jeden Animation können keine Unterschiede zum Vorbild abgespeichert werden. Also muß das komplette Bild dargestellt werden. Das entsprechende Verfahren ähnelt sehr stark dem LC-Chunk.

Auch hier beginnt jede Zeile mit einem Einheitenbyte. Das erste Byte je Einheit ist diesmal das Größenbyte. Um die Verwirrung zu erhöhen, hat es jetzt die umgekehrte Bedeutung: Ist Bit 7 gleich 0, folgt nur ein Farbwert. Dieser wird, entsprechend dem Größenbyte, in den Videospeicher mehrfach kopiert. Ist Bit 7 jedoch 1, muß das Zweierkomplement gebildet werden. Der neue Wert gibt an, wie viel Farbwerte vom Puffer in den Videospeicher kopiert werden. Versagt diese Komprimierung, ist wie beim LC-Chunk der Copy-Chunk anzuwenden.

TBild-Record - der Bildkopf

Variable	Variablentyp	Bedeutung
Groesse	LongInt	Länge des Bildes inkl. Bildkopf
Kennung	Word	stets F1FAhex
Chunks	Word	Anzahl der Chunks pro Bild
Fuell	Array[1..8] of Byte	füllt den Bildkopf auf 16 Byte auf



PROGRAMMIEREN

FLI-Player für Turbo Pascal

TChunk-Record - der Chunkkopf

Variable	Variablentyp	Bedeutung
Groesse	LongInt	Länge des Chunks inkl. Chunkkopf
Typ	Word	Chunktyp

Dieser Chunk benötigt keinerlei Rechenoperationen. Es wird das komplette Bild einfach in den Videospeicher kopiert.

Lese-Kunst

Nachdem der Aufbau eines FLI-Filmes besprochen wurde, folgt jetzt die teilweise Lösung eines anderen Problems: Die Anzeige geschieht zwar unglaublich schnell, aber die Laufwerke sind nicht in der Lage, entsprechend viele Daten nachzuliefern. Hier muß also ein Mini-Cache her. Er liest die Daten eines kompletten Chunks ein. Diese Vorgehensweise verringert den Zeitaufwand für das Lesen von Daten erheblich.

Die programmtechnische Umsetzung des Caches äußert sich in der TPuffer-Struktur und der Lies-Prozedur. Da selbst in Zeiten vom 8 MB und mehr das Datensegment von Turbo Pascal auf 64 KB begrenzt ist, wäre ein statischer Cache problematisch. Also wird er dynamisch verwaltet. Dazu muß er zu Beginn angelegt (NEW) und am Ende zerstört (DISPOSE) werden.
Die Lies-Prozedur erfüllt im jetzigen Zustand lediglich die Funktion des Lesens vom Datenträger. Sie dient jedoch dem Zweck, später ausgebaut zu werden. So ist denkbar, den kompletten FLI-Film in den EMS-/XMS-Speicher zu laden und von dort abzuspielen.

Hauptprogramm

Nachdem die Prozeduren im einzelnen erklärt wurden, folgt nun das Hauptprogramm. Dieses versteckt sich in der Funktion Play_FLI. Sie ist auch als einzige im Interfaceteil der Unit aufgeführt.
Der erste zu übergebende Parameter ist „Name" vom Typ String. Dies ist der Name der FLI-Datei und kann auch einen kompletten Pfad enthalten. Der zweite Parameter heißt „Loop" und ist vom Typ Boolean. Damit kann gesteuert werden, ob die Animation ständig (normaler FLI-Player) oder nur einmal (Intro) gezeigt werden soll. Die Funktion gibt ebenfalls einen Wert vom Typ Boolean zurück. Er gibt Auskunft über das erfolgreiche Abspielen.

Das Hauptprogramm öffnet zuerst die FLI-Datei und liest den Dateikopf ein. Danach wird die Animation bis zum Ende durchlaufen und gegebenenfalls wiederholt. Dabei wird auch das Ringbild beachtet. Natürlich kann der Benutzer jederzeit abbrechen.

Sollte bei diesem Abspielvorgang ein Fehler auftreten, so wird das Hauptprogramm sofort beendet und eine entsprechende Rückmeldung gegeben. Da die Animation rechnerunabhängig dieselbe Geschwindigkeit haben soll, wird die Prozessoruhr benutzt. Die Bildwiederholrate wird so auf 18,2 fps (=frames per second, Bilder pro Sekunde) begrenzt. Dies entspricht in etwa der Geschwindigkeit von AVI-Dateien unter Windows.

Ringbild

Der BRun-Chunk ist der zeitaufwendigste Chunk im ganzen Programm. Man versucht, dieses Problem geschickt zu umgehen. Es ist bekannt, daß dieser Chunk nur im ersten Bild der Animation verwendet wird. Am Ende der FLI-Datei findet man ein Ringbild. Dieses enthält den Unterschied zwischen dem letzten und dem zweiten Bild der Animation. Somit kann hier der schnelle LC-Chunk verwendet werden. Im Dateikopf wird bei der Gesamtzahl der Bilder das Ringbild jedoch nicht mitgezählt.

Beispiel-Player

Um Ihnen die Benutzung der FLI-Unit zu verdeutlichen, habe ich auch einen FLI-Player entwickelt. Dieser zeigt, wie einfach der Einbau von FLI-Dateien in eigene Programme ist.
Der Player benötigt als Parameter den Namen der abzuspielenden Datei. Dabei ist es egal, ob Sie die FLI-Endung anhängen oder weglassen.

FLI-Dateien selbstgemacht

Sie haben jetzt einen FLI-Player, aber keine Animation? Dazu müssen Sie mehrere Einzelbilder berechnen oder zeichnen. Letzteres ist aber sehr aufwendig. Die gezeigte Beispielanimation wurde daher mittels POV 2.2 mit 180 Einzelbildern in drei Stunden berechnet.

Jetzt haben Sie die Einzelbilder, aber immer noch keine Animation. Dieses Problem läßt sich auf zwei verschiedene Arten lösen. Zum einen können Sie den Autodesk Animator kaufen. Dieser bietet zwar einen großen Funktionsumfang, schlägt aber auch mit mehreren Hundert Mark zu Buche.

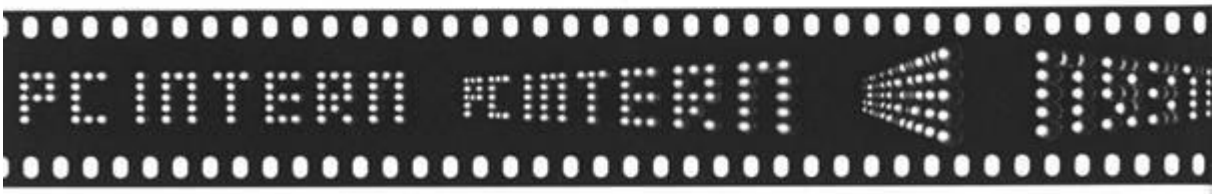
In der Sharewareszene hat man dieses Problem bereits erkannt. Für private Zwecke ist das Programm DTA daher völlig ausreichend. Sie können es 30 Tage kostenlos ausprobieren und werden erst bei Gefallen zum Registrieren aufgefordert.

Prozessoruhr

Jeder AT-Computer verfügt über die für das Programm wichtige Stelle 40hex:6Chex im Hauptspeicher. Sie enthält eine Variable vom Typ LongInt und wird 18,2 mal pro Sekunde um 1 erhöht und um 0 Uhr auf 0 gesetzt. Somit läßt sich sehr schnell die aktuelle Uhrzeit bestimmen, indem dieser Wert durch 18,2 dividiert wird. Das Ergebnis sind die vergangenen Sekunden seit Mitternacht.
Alle bekannten DOS-Betriebssysteme greifen auf diese Speicherstelle zurück, um die Uhrzeit zu bestimmen. Der Entwickler kann damit auch selbst Programmlaufzeiten bestimmen.

Variable - die Chunktypen

Nummer	Name	entsprechende Prozedur
11	Color	RGB_Decode
12	LC	LC_Decode
13	Black	FillScreen
15	BRun	BRun_Decode
16	Copy	Bitmap



FLI-Player für Turbo Pascal

Einen ganz anderen Weg zur Animationsherstellung beschreiten Morphprogramme. Sie lassen Bilder ineinander übergehen. So könnten Sie sich in Helmut Kohl morphen! Auch zu diesem Gebiet bietet der Sharewarebereich einen großen Fundus. Dieser Artikel behandelt nur das FLI-Format. Mittlerweile hat Autodesk eine Weiterentwicklung, das FLC-Format, herausgebracht. Dieses bietet zwar größere Auflösungen, ist aber auch komplizierter aufgebaut.

Vielleicht gelingt Ihnen mit Hilfe dieser Unit der nächste Knaller im Spielesektor. Viel Erfolg! (Stephan Brumme) ■

Vergleich FLI- und AVI-Format

	FLI	AVI
verfügbar für	DOS und Windows	nur Windows
Bilder pro Sekunde	18,2	15
Komprimierung	gut	sehr gut
Qualitätsverlust	nein	ja
Auflösung/Farben	320x200, 256	320x240, 65.536
Sound	nein	ja

```
{ FLI-Unit in Turbo-Pascal
 ab Version 6.0, PC Intern
 (C) Stephan Brumme
 Oktober 1995 }

{$A+,B-,D-,E-,F-,G+,I-,L-
,N-,O-,R-,S-,V-,X-}

unit fli;
interface
uses crt;
function play_fli(name:
 string; loop: boolean):
 boolean;
implementation
type theader = record
  groesse         : longint;
  kennung         : word;
  bilder          : word;
  maxx            : word;
  maxy            : word;
  farbbits        : word;
  flags           : word;
  geschwindigkeit : word;
  fuell           :
   array[1..110] of byte;
end;
tbild   = record
  groesse : longint;
  kennung : word;
  chunks  : word;
  fuell   : array[1..8] of
   byte;
end;
tchunk  = record
  groesse : longint;
  typ     : word;
end;
tpuffer = array[0..61000]
 of byte;
var datei  : file;
    header : theader;
    anfang : word;
    bild   : tbild;
    chunk  : tchunk;
    puffer : ^tpuffer;
    zeit   : longint
     absolute $40:$6C;
    tick   : longint;
procedure lies(var cache;
bytes: word);
begin
blockread(datei,cache,
 bytes);
end;
procedure rgb_decode;
 assembler;
asm
   cld
   push ds
   lds si,puffer
   lodsw
   mov cl,al
   mov al,0
@einheitschleife:
   mov bx,ds:[si]
   add si,2
   or bh,bh
   jz @allesneu
   add al,bl
   mov dx,3C8h
   out dx,al
   inc dl
@setzschleife:
   outsb
   outsb
   outsb
   inc al
   dec bh
   jnz @setzschleife
   dec cl
   jnz @einheitschleife
   jmp @ende
@allesneu:
   mov dx,3C8h
   mov al,0
   out dx,al
   inc dx
   mov cx,768
   rep outsb
@ende:
   pop ds
end;
procedure lc_decode;
  assembler;
asm
   cld
   push ds
   push bp
   lds si,puffer
   mov cx,320
   lodsw
   mul cx
   mov bx,ax
   lodsw
   mul cx
   add ax,bx
   mov bp,ax
   mov ax,0A000h
   mov es,ax
   mov ah,0
   mov cx,0
@yloop:
   mov di,bx
   mov dl,ds:[si]
   inc si
   or dl,dl
   js @yende
   jmp @xloop
@schreiben:
   neg cl
   lodsb
   rep stosb
@einheitentest:
   dec dl
   js @yende
@xloop:
   lodsw
   xchg ah,cl
   add di,ax
   or cl,cl
   js @schreiben
   rep movsb
   jmp @einheitentest
@yende:
   add bx,320
   cmp bx,bp
   jne @yloop
   pop bp
   pop ds
end;
procedure brun_decode;
  assembler;
asm
   cld
   push ds
   lds si,puffer
   mov ax,0A000h
   mov es,ax
   mov di,0
   mov ch,0
   mov ah,0
   mov dh,200
@yloop:
   lodsb
   or al,al
   js @yende
   mov dl,al
@xloop:
   lodsw
   xchg ah,al
   test ah,128
   jns @fuellen
   neg ah
   mov cl,ah
   stosb
   dec cl
   rep movsb
   dec dl
   jnz @xloop
   jmp @yende
@fuellen:
   mov cl,ah
   rep stosb
   dec dl
   jnz @xloop
@yende:
   dec dh
   jnz @yloop
   pop ds
end;
procedure fillscreen;
  assembler;
asm
   cld
   mov ax,0A000h
   mov es,ax
   mov di,0
   mov ah,0
   mov cx,32000
   rep stosw
end;
procedure bitmap;
begin
lies(mem[$A000:0],64000);
end;
function play_fli(name:
 string;loop: boolean):
 boolean;
var lauf : byte;
label raus;
begin
   play_fli:=true;
   new(puffer);
   assign(datei,name);
   reset(datei,1);
   if ioresult<>0 then
   begin play_fli:=false;
exit; end;
   lies(header,sizeof(
    header));
   anfang:=128;
   repeat
      seek(datei,anfang);
      while not eof(datei)
do
      begin
         tick:=zeit;
lies(bild,sizeof(bild));
        for lauf:=1 to
         bild.chunks do
        begin
           lies(chunk,
     sizeof(chunk));
      if (chunk.groesse>6)
       and (chunk.typ<16)
      then
       lies(puffer^,
        chunk.groesse-6);
        case chunk.typ of
          11 : rgb_decode;
          12 : lc_decode;
          13 : fillscreen;
          15 : brun_decode;
          16 : bitmap;
           else begin
play_fli:=false;
                goto
raus;
             end;
          end;
        end;
        if keypressed then
goto raus;
      if anfang=128 then
      anfang:=filepos(da-
tei);
      repeat until
tick<>zeit;
    end;
  until not loop;
raus:
  close(datei);
  dispose(puffer);
end;
end.
```