Ulrich Krinzner

OPL-Programmierung für Psion-Handhelds

z.B. Serie 5/mx/mxPro/, Serie 7, netBook und Revo

Ergänzung zum Basis-Buch, Thema: Sprites erstellen und anwenden

E-Mail: opl@krinzner.de
Web: http://www.krinzner.de Privatveröffentlichung, 1. Auflage, 08/2004

Bisher erschienen:

- Programmieren mit OPL
("Basis-Buch", Grundlagen für Einsteiger, 130 Seiten A4 als PDF, Bezug www.krinzner.de)

- Solide PSION-Programmierung unter OPL - am Beispiel eines Adventure-Games - ("Semiprofi-Buch", fortgeschrittene OPL-Programmierung, 110 Seiten A5 als PDF, Bezug www.krinzner.de)

Sprites erstellen und anwenden

1. Das Prinzip

Sprites sind Bitmap-Grafiken, die sehr spezielle Fähigkeiten ausspielen können. Sie sind

1. animierbar
2. freistellbar
3. bewegbar.

Damit erinnern sie einerseits an ein Daumenkino, dem Urvater der Animation und des Kinofilms, und andere- seits an eine einfache oder auch animierte GIF-Bilder im Webbrowser.

Für ein animiertes Sprite benötigt man mehrere Grafiken (Frames), die ähnlich wie bei einem Trickfilm die einzelnen Bewegungsschritte enthalten. Mit verschiedenen OPL-Befehlen werden sie zu einem Sprite zu- sammengeschmiedet und dann nacheinander angezeigt.

Die Bitmap-Grafiken ("Bilder") lassen sich wahlweise durch OPL-Techniken erzeugen (Grafikfenster öffnen, mit OPL-Befehlen zeichnen und das Ganze abspeichern) oder auch mit dem Skizzen-Programm anlegen und ins MBM-Format exportieren.

Das Freistellen einzelner Bildpartien geschieht durch eine Maskentechnik. Egal, ob das Sprite animiert ist oder nicht, jede einzelne Szene benötigt am Ende immer zwei Bilder: das eigentliche Bild und ein schwarz- weisses Maskenbild. Beide werden intern quasi übereinandergelegt. Dort, wo das Maskenbild schwarz ist, wird der Inhalt des richtigen Bildes auf dem Bildschirm dargestellt. An den weißen Stellen des Maskenbildes wird das richtige Bild durchsichtig. Man könnte so z.B. einen Mauszeiger basteln.

Obwohl sich auch andere Grafiken per OPL versetzen und bewegen lassen, geht das mit Sprites unvergleichlich einfacher. Wer also grafische Bewegungs-Probleme hat, sollte prüfen, ob nicht Sprites die Lösung dafür sein könnten. Noch dazu, wo selbst "Farbe" für Sprites kein Fremdwort ist.

Im Gegensatz zu SIBO/EPOC16 sind die Sprites-Anweisungen nicht mehr direkter Bestandteil von OPL, ste- hen aber trotzdem für jeden EPOC32-Computer als Bestandteil des Betriebssystemes zur Verfügung. Sie verstecken sich in einem Erweiterungsmodul, einer DLL (Dynamic Link Libray) mit der speziellen Endung OPX.

Damit man Funktionen eines Erweiterungsmoduls nutzen kann, muss es am Anfang des OPL-Quelltextes eingebunden werden. Man "inkludiert" dazu ein zugehöriges "header file" mit der Dateiendung OXH.

Die gesamte Spritefunktionalität und weitere Bitmap-Funktionen verbergen sich in der Datei BMP.OPX, die Einbindung erfolgt mit Hilfe von BMP.OXH. Die erste Zeile in einem Programm mit Sprites lautet also:

```
INCLUDE "BMP.OXH"
```

Der Aufruf einer Funktion aus BMP.OPX ähnelt sehr dem Aufruf eines OPL-Unterprogramms oder einer - Funktion, z.B.:

```
BITMAPLOAD&:(filename$,index&)
```

Die zur Verfügung stehenden Funktionen sind:

```
BITMAPLOAD&:(), BITMAPUNLOAD:(), BITMAPDISPLAYMODE&:(),

SPRITECREATE&:(), SPRITEAPPEND:(), SPRITECHANGE:(),

SPRITEDRAW:(), SPRITEPOS:(), SPRITEDELETE:() und SPRITEUSE:(id&)
```

Die Funktionen sind am Ende detailliert erläutert. Was hier nicht deutlich wird, zeigt vielleicht das Demo- Programm.

2. Demoprogramm

```
1   REM OPL: spritedemo1
2   INCLUDE "BMP.OXH"
4   PROC Main:
5   LOCAL verz$(128), bild1$(128), bild2$(128), maske$(128), bm%
6   LOCAL fr1&, fr2&, msk&, flash&, x&, y&, mask&, sprite&, n%, limit&
8   verz$= "c:\sprite\"
9   bild1$ = verz$ + "bild1.mbm"
10  bild2$ = verz$ + "bild2.mbm"
11  maske$ = verz$ + "maske.mbm"
12
13 REM *** Bitmaps erzeugen & speichern ***********
15  TRAP MKDIR verz$
16  bm%=gCREATEBIT(60,60,0)
17  gFILL 60,60,0
18  TRAP gSAVEBIT maske$
19  gAT 20,1 : gFILL 20,58,2
20  TRAP gSAVEBIT bild1$
21  gFILL 60,60,0 : gAT 1,20 : gFILL 58,20,2
22  TRAP gSAVEBIT bild2$
23  gCLOSE bm%
24
25 REM *** Bitmaps laden **************************
27  fr1&= BITMAPLOAD&:(bild1$,0)
28  fr2&= BITMAPLOAD&:(bild2$,0)
29  msk&= BITMAPLOAD&:(maske$,0)
30
31 REM *** Sprite anlegen ************************
33  x&=30 : y&=30
34  flash&=0
35  sprite&=SPRITECREATE&:(0,x&,y&,flash&)
36
37 REM *** zwei Frames einfügen ********************
39  mask&=1
40  SPRITEAPPEND:(250000,fr2&,msk&,mask&,0,0)
41  SPRITEAPPEND:(1000000,fr1&,fr1&,mask&,0,0)
42
43 REM *** Bitmap wurden zugewiesen, können entladen werden
44
45 BITMAPUNLOAD:(fr1&) : BITMAPUNLOAD:(fr2&) : BITMAPUNLOAD:(msk&) 46
47 REM *** Sprite zeichnen ************************
49  gGREY 1 : gAT 20,20 : gFILL 140,40,0 REM graue Kontrastfläche
50  SPRITEDRAW:
51
52  AT 1,1 : PRINT "Animiertes Sprite wird angezeigt, weiter mit Taste"
53  BEEP 1,333 : WHILE KEY : ENDWH : GET
54
55 REM *** Sprite verschieben **********************
57  AT 1,1 : PRINT "Sprite läuft ... "
58  n%= 1 : limit&= 130
60  WHILE x&<limit&
61  PAUSE -1
62  x&= x& + n%
63  SPRITEPOS:(x&,y&)
64  ENDWH
65
66  AT 1,1 : PRINT "Sprite ist auf Endposition, Programm mit ENTER beenden."
67  BEEP 1,333 : WHILE KEY : ENDWH : GET
68
69  SPRITEDELETE:(sprite&)
70  ENDP
```

3. Erläuterungen zum Demoprogramm

Zunächst: Die Zeilennummern im Demoprogramm dienen natürlich nur der besseren Beschreibungsmöglichkeit und sind imQuelltext wegzulassen ... Und: Um Platz zu sparen, wurde im Quelltext teilweise mehrere Anweisungen in einer Zeile notiert - also genau hinsehen!

Zeilen 1 bis 12 enthalten die notwendigen Vorbereitungen, z.B. die Variablen-Deklaration und erste Werte- zuweisungen.

In den **Zeilen 13 bis 24** werden drei quadratische Bitmaps in schwarz/weiss erzeugt: die völlig schwarze Maske, Bild 1 mit schwarzer Fläche und vertikalem weissen Balken sowie Bild 2 mit schwarzer Fläche mit horizontalem weissen Balken. Alle drei Bitmaps werden separat in je einer Grafikdatei abgelegt. Das Verfah- ren dient dem Experiment, es funktionieren im folgenden Verlauf natürlich auch fertige Grafiken im MBM- Format.

Zeile 25 .. 30: Die Bitmaps fürs Sprite werden geladen.

Das Sprite wird in den **Zeilen 31 .. 35** angelegt. Angegeben werden: das Fenster, in dem das Sprite agieren soll - hier "0" für das Basisfenster, die x,y-Position der linken oberen Ecke des Sprites im Fenster und schließ- lich, ob es blinken soll oder nicht (0 = blinkt nicht, 1 = blinkt).

In **Zeilen 37 .. 41** wird das Sprite mit zwei Frames gefüllt. Jeder Frame besteht aus einem Bild und einer Mas- ke. Beim ersten Frame (**Zeile 40**) wird ein separates Masken-Bitmap - eine völlig schwarze Fläche - verwen- det, im zweiten (**Zeile 41**) werden für Bild und Maske das gleiche Bitmap verwendet. Die Folge: der weisse vertikale Bereich des Bitmaps wird später durchsichtig erscheinen. Und das gilt allgemein: Was im Masken- Bitmap schwarz ist, wird dargestellt, alle weissen Pixel erscheinen durchsichtig.

Voraussetzung ist allerdings, das der Parameter mask& in der `SPRITEAPPEND`-Anweisung den Wert 1 hat. Exakt gesagt muss es heißen : das niedrigste Bit von `mask&` muss gesetzt sein, so dass `mask& AND 1 = 1` ist . Ist `mask&` Null, wird das Masken-Bitmap invertiert verwendet. In unserem Falle (**Zeile 40:** Kombination von Bild 2 `fr2&` mit der schwarzen Maske `msk&`) wäre die gesamte Fläche durchsichtig.

Die Anzeigedauer des ersten Frames beträgt 250.000 / 1.000.000 = 0,25 Sekunden, der zweite wird 1 Sekunde lang gezeigt. Die Animation läuft ohne unser Zutun in einer unendlichen Schleife.

Sind die Bitmaps dem Sprite zugeordnet, können sie entfernt werden - sie belasten unnötig den Speicher. In **Zeile 45** werden sie "entladen".

Das Sprite wird erst sichtbar, wenn die `DRAWSPRITE:`-Anweisung gegeben wird (**Zeile 50**). Es erscheint an der in `SPRITECREATE:` vorbestimmten Position und spielt die Animation ab, ohne sich vom Fleck zu bewegen. In der Demo erkennt man sehr schön die Wirkung der Maske.

Zeilen 55 .. 64 sorgen für die Verschiebung des Sprites in x-Richtung. Selbst während der Bewegung wird die Animation fortgesetzt.

4. Spezialitäten

Farbe verwenden

Sprites dürfen auch farbig sein! Man muss nur dafür sorgen, dass auf der Zeichenfläche farbig gemalt werden kann und jeweils die entsprechende Zeichenfarbe bei der Bitmap-Erstellung mittels gCOLOR aktiviert wird. Das könnte so aussehen:

```
LOCAL bm%, bild1$(128), verz$(128)
verz$ = "C:\sprite\"
bild1$= verz$ + "bild.mbm"
TRAP MKDIR verz$
bm%= gCREATEBIT(60,60,5)
gAT 0,0 : gCOLOR 255,0,0 : gFILL 60,60,0
gAT 10,20 : gCOLOR 255,255,255 : gPRINT "Bild"  REM weisse Schrift
TRAP gSAVEBIT bild1$
```

Die Maske kann und sollte weiterhin ein schwarz/weiss-Bitmap sein. Dieses muss aber nicht zwangsläufig auch in einem schwarz/weiss-Modus-Fenster erstellt werden. Es reicht aus, wenn ausschließlich schwarze und weisse Pixel verwendet werden!

Mit farbigen Masken erlebt man so seine Überraschungen. Weil sie gewissermaßen nicht "blickdicht" sind, decken farbige, nicht schwarze Masken nicht mehr vollständig. Unterliegende Bereiche scheinen durch, der dabei wirksame Farbenmix ist nur schwer vorausberechenbar und liefert oft unansehnliche Farben.

Diesem Effekt begegnet man oft, wenn man sich in Eile nicht die Mühe macht, eine getrennte Maske zu erstellen und das selbe Bipmap als Maske verwendet.

Fazit: Eigenständige schwarz/weiss-Masken verwenden und farbige Masken vermeiden!

Räumliche Anordnung von Sprites

Benutzt man mehrere Sprites, werden die Sprite-Anweisungen immer für das zuletzt erstellte ("aktive") Sprite ausgeführt. Dieses liegt gleichzeitig auch ganz oben auf, wie man bei Überlappungen feststellen kann. Das Verhalten gleicht damit dem von Fenstern.

Durch `SPRITEUSE:(id&)` kann man ein beliebiges anderes Sprite zum aktiven machen. Weitere Anweisun- gen wirken sich dann auf dieses aus, allerdings wird die räumliche Anordnung damit nicht verändert.

Leider verfügt das BMP.OPX-Modul nicht über eine Anweisung zur Tiefen-Staffelung von Sprites. Der `gORDER`–Befehl für Fenster kann nicht angewendet werden. Deshalb muss man ein Sprite, das in den Vordergrund soll, zuerst schließen und gleich wieder erzeugen und anzeigen. Die Geschwindigkeit des Vorgangs ist so groß, dass der Programmbenutzer von dieser Umständlichkeit nichts mitbekommt.

Bequemerweise legt man für solche Aktionen die Erzeugung jedes einzelnen Sprites in eine eigene Prozedur, das erleichtert den Überblick und erspart Tipparbeit.

5. Die Bitmap- und Sprite-Anweisungen von BMP.OPX

Allgemeiner Hinweis: Der Gebrauch sehr vieler Sprites zur gleichen Zeit kann unerwünschte Effekte hervorrufen. Einerseits können Animationen langsamer ablaufen, andererseits können Tastaturreaktionen u.a. stark verzögert ablaufen! Die Anzahl der möglichen Sprites ohne solche Störungen hängt ab von der Animationsgeschwindigkeit (je schneller, desto weniger Sprites). Richtwert: mehr als 20 Sprites mit einer Animationsrate von 0,2 Sekunden sollte man nicht gleichzeitig verwenden. Man experimentiere!

BITMAPLOAD&:

Verwendung: `id&=BITMAPLOAD&:(filename$,index&)`

Lädt ein Bitmap aus der Datei filename$ (aus dem aktuellen Verzeichnis, wenn nicht anders angegeben). Wenn die Datei mehr als ein Bitmap enthält, bezeichnet index&, welches Bitmap geladen werden soll. Das erste bzw. einzige Bitmap in der Datei wird mit dem index&-Wert 0 angesprochen. Der nach id& zurückge- gebene Wert ist die Identifikationsnummer (ID) der offenen Bitmapdatei, mit dessen Hilfe das Bitmap durch Sprite- oder entsprechende Grafik-Befehle angesprochen werden kann. angesprochen werden kann (z.B. mit `gBUTTON`).

Siehe auch `BITMAPUNLOAD:`

BITMAPUNLOAD:

Verwendung: `BITMAPUNLOAD:(id&)`

Deaktiviert das Bitmap mit der ID id& für die weitere Verwendung - die Datei wird "entladen". Sowie das Bit- map z.B. einem Sprite zugeordnet wurde und dieses angezeigt wird, kann es mit `BITMAPUNLOAD:` entfernt werden.

Siehe auch `BITMAPLOAD&:`

BITMAPDISPLAYMODE&:

Verwendung: `mode&=BITMAPDISPLAYMODE&:(id&)`

Gibt den Grafikmodus des Bitmaps bei der Erstellung mit z.B. `gCREATE` zurück. Der Wert ist wie dort: 0 = schwarz-weiss-Modus
1 = 4-Graustufen-Modus
2 = 16-Graustufen-Modus

3 = 256-Graustufen-Modus 4 = 16-Farbstufen-Modus 5 = 256-Farbstufen-Modus

SPRITECREATE&:

Verwendung: `id&=SPRITECREATE&:(winId%,x&,y&,flags&)`

Initialisiert ein Sprite im Fenster `winID%`, seine linke obere Ecke wird später an der Position `x&,y&` gezeichnet. Der Wert von `flags&` bestimmt, ob die einzelnen Spriteelemente blinken oder nicht. Bei `(flags& AND 1)=1` blinken sie, bei 0 nicht. Der an `id&` zurückgegebene Wert ist die ID des Sprites. Andere Sprite-Befehle benutzen diesen Wert, um das Sprite anzusprechen.

Siehe auch `SPRITEAPPEND:, SPRITECHANGE:, SPRITEDRAW:, SPRITEPOS:, SPRITEDELETE:, SPRITEUSE:`

SPRITEAPPEND:

Verwendung: `SPRITEAPPEND:(time&,bitmap&,maskBitmap&,invertMask&,dx&,dy&)`

Fügt zum aktuellen, mit `CREATESPRITE&:` erstellten Sprite einen einzelnen Frame hinzu.

`bitmap&` ist die ID des Bitmaps, das zum Sprite hinzugefügt werden soll; `maskBitmap&` ist die ID des Bitmaps, das als Maske benutzt werden soll. Ein Frame besteht also immer aus zwei Bitmaps. Beide müssen gleiche Abmessungen besitzen! Um eine ID zu erhalten, muss man zuvor eine existierende Bitmap-Datei "laden":

```
id&=BITMAPLOAD&:(filename$,index&).
```

`invertMask&` besitzt den Wert 1 oder 0. Der Normalfall ist 1 - die Maske wird normal auf das Bitmap angewendet: Vom darzustellenden Bitmap werden alle Pixel im Sprite dargestellt, an denen die zugehörige Maske schwarze Pixel besitzt. An den weißen (leeren) Stellen wird das Bitmap durchsichtig. Bei `invertMask&=0` werden die Maskenwerte invertiert, mit entsprechender Auswirkung auf die Anzeige. Um ein Bitmap vollständig anzuzeigen, ist als Maske ein völlig schwarz ausgefülltes Bitmap zu verwenden.

`time&` ist die Zeitdauer in Mikrosekunden, die ein Frame angezeigt wird. Wird nur ein einziger Frame im Sprite verwendet, ist der Wert unerheblich.

`dx&,dy&` ist eine Offsetposition - eine Verschiebung des Nullpunktes des Bitmaps relativ zur linken oberen Ecke des Sprites.

Siehe auch `SPRITECHANGE:, BITMAPLOAD&:`

SPRITECHANGE:

Verwendung:
`SPRITECHANGE:(id&,time&,bitmap&,maskBitmap&,invertMask&,dx&,dy&)`

Ändert die Parameter eines einzelnen Frames eines existierenden Sprites. `id&` ist die Nummer des Frames, der geändert werden soll. Diese Nummer ergibt sich aus der Reihenfolge beim Zufügen des Frames zum Sprite, der erste Frame hat die Nummer 0. Die anderen Parameter sind identisch zu handhaben wie bei `SPRITEAPPEND:` . Wenn dabei Bilder verwendet werden, die bereits "in dem Sprite stecken", müssen die Bitmaps nicht neu geladen werden. Andere Bilder bereitet man mit `BITMAPLOAD:` zur Verwendung vor.

Ein Sprite kann erst geändert werden, wenn es zuvor bereits mittels `SPRITEDRAW:` "gezeichnet" wurde.

Siehe auch `SPRITEAPPEND:, SPRITEDRAW:`

SPRITEDRAW:

Verwendung: `SPRITEDRAW:`

Stellt das Sprite am Bildschirm dar, nachdem es mittels `SPRITECREATE&:` angelegt und per `SPRITEAPPEND:` mit Inhalt gefüllt wurde. Muss nur einmal aufgerufen werden. Wendet man auf das sichtbare Sprite `SPRITECHANGE:` an, ist kein erneuter Aufruf erforderlich, um die Änderungen wirksam werden zu lassen. Das Sprite erscheint in dem Fenster, das beim Aufruf von `SPRITECREATE&:` übergeben wurde, das gleiche gilt für die Position im Fenster.

Siehe auch `SPRITECREATE&:, SPRITEAPPEND:, SPRITECHANGE:, SPRITEPOS:`

SPRITEPOS:

Verwendung: `SPRITEPOS:(x&,y&)`

Verschiebt das komplette aktuelle Sprite an die mit `x&,y&` benannte Position.

Siehe auch `SPRITECREATE&:,SPRITEDRAW:`

SPRITEDELETE:

Verwendung: `SPRITEDELETE:(id&)`

Löscht das Sprite mit der ID `id&`.

SPRITEUSE:

Verwendung: `SPRITEUSE:(id&)`

Macht das Sprite mit der ID `id&` zum aktuellen, so dass alle nachfolgen Sprite-Anweisungen sich auf dieses Sprite auswirken. Das Sprite kommt dadurch jedoch nicht in den Vordergrund! Dazu muss es gelöscht und neu aufgebaut werden. Im Vordergrund befindet sich immer das zuletzt erstellte.

Siehe auch `SPRITECHANGE:, SPRITEPOS:`