AUF DER HEFT-CD

download
www.pc-magazin.de

Demo-Programmierung unter Windows 95/NT

Schöne Töne

Nach den visuellen Effekten der letzten Ausgaben warten die Programme und Routinen diesmal mit akustischen Reizen auf.

CARSTEN DACHSBACHER

Ein PC mit Soundkarte kann Klänge über ein angeschlossenes Mikrofon aufnehmen oder elektrische Impulse von einer Hi-Fi-Komponente durch die Line-In-Schnittstelle empfangen. Dazu mißt er in regelmäßigen Zeitabständen die analoge Amplitude der Welle und rechnet die gemessene Spannung in einen digitalen Wert um. Beim Abspielen wandelt die Soundkarte die so erhaltenen Werte wieder in Spannungen zurück, die dann einen Lautsprecher betreiben.

Entscheidend für die Klangqualität ist zum einen die Anzahl der gemessenen Werte pro Sekunde (Sampling-Frequenz). Zum anderen spielt die Anzahl der Bits eine Rolle, die zur Repräsentation der abgetasteten Spannungswerte benutzt werden. Ein CD-Player arbeitet zum Beispiel mit einer Sampling-Frequenz von 44,1 kHz und 16 Bit pro Abtastwert. Multiplizieren Sie diese 44 100 Abtastwerte pro Sekunde mit dem Platzbedarf von 2 Byte (= 16 Bit) pro Abtastwert, so errechnen Sie einen Speicherbedarf von 88 200 Byte pro Sekunde für jeden der beiden Stereokanäle.

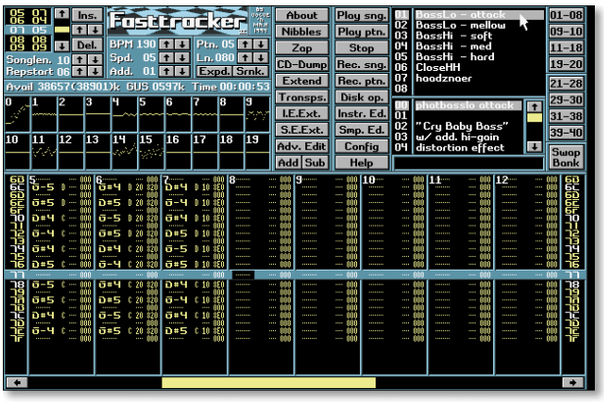
Um die aufgenommenen Musiksignale korrekt reproduzieren zu können, muß die Sampling-Frequenz mindestens doppelt so hoch sein wie die Frequenz des höchsten Tons. Aus dieser allgemeingültigen Regel (bekannt als Shannon-Theorem) resultiert auch der typische Frequenzgang eines CD-Players bis etwa 22 kHz.

Qual der Wahl

Um eine Demo mit Hintergrundmusik zu versehen, haben Sie die Wahl zwischen zwei verschiedenen Ansätzen.

• Sie können das ganze Musikstück aufnehmen, als *wav*-Datei speichern und parallel zur Demo abspielen. Der Vorteil hierbei ist, daß Sie nur einmal die Ausgabe starten müssen und sich um nichts weiteres zu kümmern brauchen. Außerdem benötigt die CPU zum Abspielen der Sample-Daten praktisch keine Rechenzeit. Negativ schlägt der relativ hohe Speicherverbrauch zu Buche. Wie Sie dem etwas entgegenwirken, erfahren Sie später.

• Bei der zweiten Variante werden einzelne Instrumente aufgenommen und die Musiknoten zu den Instrumentendaten gespeichert. Sie haben dann dafür zu sorgen, daß die Instrumente zu den gegebenen Zeitpunkten in der richtigen Tonhöhe abgespielt werden. Dieses Verfahren nutzen die noch vom Commodore Amiga stammenden *Modules* (MOD-Format) und die *Extended Modules* (XM-Format), welche eine erweiterte Funktionalität besitzen.



HERZSTÜCK DES FASTTRACKER II: Im Pattern-Editor weisen Sie jedem Instrument Tonhöhe, Lautstärke und Effekte zu.

Hauptsächlich fallen zwei verschiedene Berechnungen mit den Sample-Daten an. Zuerst möchten Sie sicherlich die Lautstärke eines Instruments beeinflussen. Dafür multiplizieren Sie jeden Sample-Wert mit einem Faktor und verändern somit die Amplitude, also die Lautstärke des Samples.

Die zweite wichtige Operation ist das Mischen der einzelnen Kanäle bzw. der Instrumente. Ohne Wavetable-Soundkarte muß die Software diese rechenintensive Aufgabe übernehmen. Moderne PCs sind schnell genug. Sie berechnen die Klänge in Echtzeit und fügen zusätzlich Effekte wie Chorus oder Reverb (Echo) hinzu. Das Mischen der Sample-Daten entspricht einer Addition.

Module komponieren

FastTracker II ist ein Musikprogramm, mit dem Sie selbst MOD- und XM-Musikstücke schreiben. Um Ihnen den Einstieg ins Komponieren mit FastTracker II (auf unserer Heft-CD) zu erleichtern, gibt Ihnen PC Magazin wertvolle Tips zur Hand.

Profi-Einstieg

Wie auch alle anderen Tracker auf dem Markt ist FastTracker II kein Programm, um schnell oder einfach Musik zu machen. Es erfordert eine gewisse Einarbeitung und Gewöhnung, um effektiv mit diesem mächtigen, aber unkonventionellen Tool umzugehen.

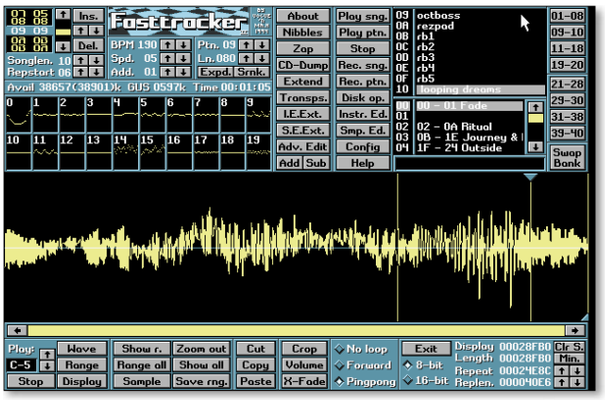
Generell ist ein sogenanntes Modul anders aufgebaut als MIDI-Dateien. Der Ansatz der Module ist – da sie ursprünglich vom Amiga stammen – sehr Hardware-nah und effizient gehalten. Generell gibt es eine Anzahl von Kanälen (normalerweise zwischen 4 und 32). Jeder Kanal entspricht einer Mono-Audiodatei, ähnlich einer *wav*-Datei. Im Gegensatz zu Harddisk-Recording-Software bieten Tracker jedoch die Möglichkeit, diese Klänge beliebig in Tonhöhe und Lautstärke zu modifizieren. So entstehen aus diesen einzelnen Samples komplexe polyphone (vielstimmige) Musikstücke.

Die Musikdaten sind deshalb in sogenannten Patterns zusammengefaßt. In einem Pattern gibt es für jeden Kanal eine Spalte, in der untereinander die Events stehen. Jedes Event besteht aus vier Komponenten:

• einer Note, die die Tonhöhe des abzuspielenden Instruments angibt;

• einer Instrumentennummer, die das Instrument bzw. das Sample spezifiziert;
• einer Lautstärkespalte, die die Lautstärke des Tons bestimmt
• und einer Effektspalte, in der Sie mittels einer dreistelligen Hexadezimalzahl bestimmte Modulationen auf den Ton anwenden.

Eine Zeile in einem Pattern entspricht einem bestimmten Zeitintervall. Im Gegensatz zu einer Event-Liste in üblichen Sequenzerprogrammen ist dieser Zeitablauf jedoch starr, das heißt, die Liste wird mit einer mehr oder weniger konstanten Geschwindigkeit von oben nach unten abgearbeitet.



SAMPLE-EDITOR: die virtuelle Stimmgabel zum Feintuning der Instrumente

In der sogenannten Order-Liste reihen Sie die Patterns aneinander und fügen so einen Song zusammen. Alle Patterns in der Orderliste werden nacheinander abgespielt. Wenn Sie sich in die – leider nicht optimale – Benutzerführung von FastTracker II eingearbeitet haben, sind Sie mit ein wenig Übung bald in der Lage, professionelle Musikstücke zu komponieren.

Im Editor von FastTracker II stellen Sie schnell fest, daß das ganze Programm mit Hexadezimalzahlen arbeitet. Der Grund dafür ist die kompaktere Darstellung der verwendeten Parameter für die Bildschirmausgabe. Das ist etwas gewöhnungsbedürftig, aber die hexadezimale Numerierung der Pattern-Zeilen erscheint bei der heutigen 4/4-Takt-dominierten Musik wesentlich intuitiver. Sie können damit Abstände von beispielsweise 4, 8 oder 16 Zeilen schneller erkennen.

Bevor Sie sich ans Werk machen, sollten Sie sich die Online-Hilfe und die mitgelieferte Dokumentation ansehen. Dort finden Sie einige Tastaturbefehle, die die Arbeit mit dem Editor wesentlich erleichtern. Da die bereits erwähnten Effektspalten in den Patterns mit Zahlen arbeiten, sollten Sie auch diese in der Hilfe nachschlagen.

Das Sound-Cockpit

Nach dem Programmstart sehen Sie den sogenannten Arbeitsbildschirm. Dieser ist in mehrere Bereiche unterteilt. In der unteren Hälfte befindet sich normalerweise die Arbeitsfläche, also der Pattern-Editor, während Sie im oberen Bildschirmteil nützliche Anzeigen oder Hilfsbildschirme sehen. Im Normalfall finden Sie links oben die Order-Liste, diverse Einstellungen, eine Speicheranzeige und einen Bereich mit Oszilloskop-Darstellungen. Diese zeigen während des Abspielens den Zustand der einzelnen Kanäle an. Im rechten oberen Teil befinden sich zwei Reihen mit Schaltflächen zum Aufrufen der verschiedenen Funktionen und Editoren, rechts daneben die Liste der Instrumente und ihrer Samples.

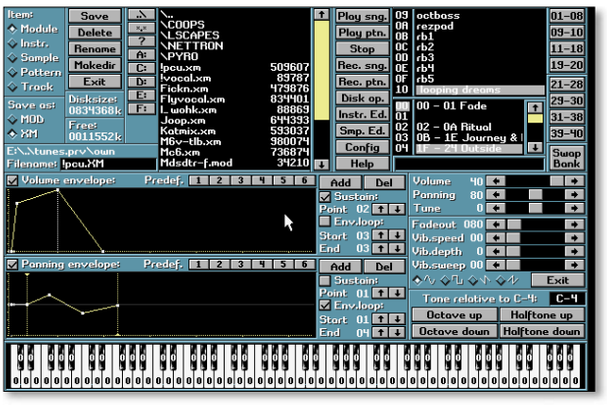
Da FastTracker II aus der DOS-Zeit stammt, steht zuerst die Konfiguration Ihrer Soundkarte an. Im *Config*-Menü wählen Sie in den Abschnitten *Output Device* und *Sampling Device* Ihre Soundkarte aus. Im Fall einer Sound-Blaster-kompatiblen Karte sollten Sie zusätzlich die IRQ- und DMA-Einstellungen überprüfen. Außerdem empfiehlt es sich für eine möglichst gute Klangqualität, die Optionen *Interpolation*, *16bit mixing* und *Stereo* zu aktivieren. Um die maximale Sampling-Frequenz für die Ausgabe zu wählen, klicken Sie rechts oben neben der Frequenztabelle auf *Max*.

Um ein Modul zu laden, klicken Sie oben in der Mitte auf *Disk op.* und wählen den entsprechenden Namen im Dateiauswahlfenster links oben. Mit der Schaltfläche *Save* speichern Sie ein Modul unter dem aktuellen Namen – dieser ist am unteren Fensterrand eingeblendet. Um diesen Namen zu ändern, klicken Sie ihn – wie bei allen Textfeldern in FastTracker II – mit der rechten Maustaste an. Möchten Sie einen Sample als Instrument laden, klicken Sie in der Instrumentenliste auf die Position, an der es eingetragen werden soll, und auf den Dateinamen in Dateiselektor.

Im unteren Teil des Bildschirms sehen Sie entweder den Pattern-, den Sample- oder den Instrumenteneditor. Der Pattern-Editor ist das Herzstück des Tracker. Hier geben Sie die Notendaten in die Patterns ein. Dazu benutzen Sie die Tastatur als virtuelles Keyboard: Die QWERTZ-Reihe stellt die weißen, die Zahlenreihe die schwarzen Tasten dar. QUERTZ ist ein nach der Anordnung der Tasten auf der Tastatur entstandenes Kunstwort zur Unterscheidung der englischen (QWERTY) und der deutschen (QWERTZ) Tastatur. Der Cursor befindet sich immer in der hervorgehobenen Mittelzeile des Pattern-Editors und ist mit einer Umrandung markiert.

Sie bewegen den Cursor mit den Pfeiltasten oder der *[Tab]*-Taste, um zwischen den Kanälen zu wechseln. Vorsicht: Auch wenn der Pattern-Editor nicht aktiv ist und Sie zum Beispiel den Sample-Editor sehen, können Sie den Inhalt des Pattern modifizieren. Mit der Leertaste wechseln Sie zwischen dem Anhörmodus (die Samples werden einfach nur gespielt, wenn Sie die Tasten drücken) und dem Editiermodus (die Noten werden in das Pattern eingetragen). Den Editiermodus erkennen Sie an der etwas helleren Farbe des Bildschirmrahmens.

Befindet sich der Cursor über der Notenspalte, können Sie Noten eingeben oder mit der Feststelltaste ein *Key-*



CYBER-KLAVIER: FastTracker II verwandelt die PC-Tastatur in ein virtuelles Keyboard.

off-Event setzen. Im letzteren Fall stoppt die Sample-Ausgabe des betreffenden Kanals an dieser Position. Bei den anderen Spalten geben Sie die Lautstärke und die Effekte ein, deren Zahlencodes Sie der Online-Hilfe entnehmen.

Im Sample-Editor schließlich können Sie die Sample-Daten der Instrumente betrachten und wie mit einem Wave-Bearbeitungsprogramm modifizieren.

Sound-Bibliothek benutzen

Um Ihre Kompositionen in Ihre Demo zu integrieren, bedienen Sie sich am besten einer fertigen Bibliothek wie der MIDAS Audio Library (auf unserer Heft-CD). Die Gründe, warum wir Ihnen hier eine fertige Bibliothek vorstellen, sind vielfältiger Natur: Zum einen würde die Entwicklung eines eigenen MOD-Players mit all seinen Feinheiten mehrere Ausgaben von PC Underground füllen – ganz zu schweigen von einem XM-Player.

Zum anderen ist es nicht immer sinnvoll, gleich das Rad neu zu erfinden. Deswegen kommt das MIDAS-Soundsystem auch in der Demoszene sehr oft zum Einsatz. Es ist zweifelsfrei eines der besten und bietet eine einfach Schnittstelle, um MOD- und XM-Dateien unter DOS, Windows oder Linux abzuspielen.

Aktuelle Updates und weitere Informationen bekommen Sie unter

```
www.s2.org/midas
```

Außer dem MIDAS-Paket, in dem eine Vielzahl von Beispielprogrammen und Dokumentationen enthalten sind, benötigen Sie nur noch eine kleine Änderung am Demo-Basissystem: Sie müssen sichergehen, daß MIDAS immer korrekt beendet wird. Deshalb fügen Sie dem Basissystem außer *demoinit()* und *demomain()* noch *demoquit()* hinzu. Diese Funktion wird beim Beenden der Demo automatisch aufgerufen.

Als Beispielprogramm hierzu finden Sie in dieser Ausgabe einen kleinen Module-Player. Dieses Programm verwendet die von MIDAS bereitgestellten Funktionen, um Informationen über die Musikwiedergabe zu erhalten:

```
MIDASmoduleInfo info;
MIDASplayStatus status;

MIDASgetModuleInfo(module,
  &info);
MIDASgetPlayStatus(playHandle,
  &status);
```

Die beiden Strukturen *&info* und *&status* werden von den aufgerufenen Funktionen mit Daten gefüllt und sind folgendermaßen definiert:

```
typedef struct {
  // Titel des Module
  char songName[32];
  // Länge der Order-List
  unsigned songLength;
  // Anzahl der ver-
  // schiedenen Patterns
  unsigned numPatterns;
  // Anzahl der Instrumente
  unsigned numInstruments;
  // Anzahl der Kanäle
  unsigned numChannels;
} MIDASmoduleInfo;

struct {
  // Position in der Order-List
  unsigned position;
  // Pattern an dieser Stelle
  unsigned pattern;
  // aktuelle Zeile im Pattern
  unsigned row;
  // Synchronisationsinfo
  int syncInfo;
  unsigned songLoopCount;
} MIDASplayStatus;
```

Der Module-Player stellt den Titel des Module in der Titelzeile dar, während er die momentane Position, das Pattern und die Zeile kontinuierlich im Fenster aktualisiert. Die Routinen, mit denen Sie die MIDAS-Bibliothek nutzen und Fehlermeldungen abfangen, finden Sie in der Datei *UseMIDAS.cpp*.

MACHT SPASS: Mit Rave-eJay schaffen Sie die Grundlagen für Ihre spätere D.J.-Karriere.

Musik im *wav*-Format

Das Komponieren mit FastTracker II benötigt ohne Zweifel einiges an Erfahrung und erscheint auf den ersten Blick sehr aufwendig. Einfacher geht es mit Musikprogrammen wie jenen aus der eJay-Reihe. Anstatt wie bei Tracker die Noten für einzelne Instrumente anzugeben, fügen Sie hier vorgefertigte Klangbausteine zu einem Musikstück zusammen. Dabei stehen Ihnen mehrere Kanäle zur Verfügung, in denen Sie mit automatischer Positionierungshilfe per Drag&Drop die Klänge aus einer Auswahl von mehreren hundert Samples aneinanderreihen können.

Abhängig vom gewählten eJay-Programm, das es für die Musikrichtungen Rave, Dance und Hip-Hop gibt, können Sie weitere Effekte und eigene Klangbausteine hinzufügen. Mit solch einem Programm erzielen Sie in kürzester Zeit beeindruckende Ergebnisse, die Sie als *wav*-Datei speichern.

Wie Sie bereits wissen, benötigen *wav*-Dateien im Vergleich zu *mid*-, *mod*- oder *xm*-Dateien relativ viel Spei-

FUNKTIONEN DER MIDAS-SOUND-BIBLIOTHEK

Wenn Sie die Datei *midasdll.lib* in Ihre Demo linken und die zugehörige Header-Datei einbinden, stehen Ihnen folgende Befehle und Datentypen zur Verfügung, mit denen Sie bereits das Abspielen starten können:

```
MIDASmodule module;
MIDASmodulePlayHandle
  playHandle;

//Startet MIDAS und ini-
//tialisiert die Soundausgabe
MIDASstartup();
MIDASinit();

//Legt fest, daß MIDAS auto-
//matisch im Hintergrund läuft
MIDASstartBackgroundPlay(0);

//Lädt die xm-Datei
module =
  MIDASloadModule("musik.xm");

//Beginnt das Abspielen
playHandle =
  MIDASplayModule(module,TRUE);

//Hier kann Ihr Programm wie
//bisher ablaufen

//Abspielen stoppen und
//Speicher freigeben
MIDASstopModule(playHandle);
MIDASfreeModule(module);

//Beendet MIDAS
MIDASclose();
```

cherplatz. Abhilfe könnte das MP3-Musikformat schaffen, das hervorragende Kompressionsraten erzielt. Allerdings ist es nicht so einfach, dieses Format wieder abzuspielen, und es gibt auch keine frei erhältlichen Bibliotheken.

Deshalb reduzieren Sie die Datenmenge mit anderen Mitteln. Der erste Schritt ist die Wahl einer niedrigeren Sample-Frequenz, da Sie für eine Demo nicht die volle CD-Qualität benötigen. Außerdem brauchen Sie für gute Ergebnisse nicht unbedingt 16 Bit je Sample-Wert. Oftmals sind 8 Bit aber zu wenig.

Um diesem Dilemma zu entgehen, nutzen Sie eine Eigenschaft des menschlichen Hörapparates aus: Das Ohr nimmt bei leisen Klangpassagen kleinere Unterschiede der Lautstärke wahr als bei lauten. Also verwenden Sie bei niedrigen Amplituden präzisere Sample-Werte, während Sie sich bei hohen Amplituden ungenaue Werte leisten können.

Wenn eine Soundkarte die Amplitude *a* in den Sample-Wert *s* überführt, wandelt sie *2*a* in *2*s* um. Es besteht also ein linearer Zusammenhang zwischen der Amplitude und dem Sample-Wert, das heißt die Genauigkeit eines Sample-Wert-Schrittes entspricht immer derselben Änderung der Amplitude. Um sich nun die Schwächen des Menschen beim Hören zunutze zu machen, verwenden Sie statt einer linearen Skala eine nichtlineare. So können Sie zum Beispiel 16-Bit-Werte in nichtlineare 8-Bit-Werte umwandeln und erhalten ein deutlich besseres Ergebnis als normalerweise bei 8 Bit. Ausgehend von einem unsigned 16-Bit-Sample-Wert errechnen Sie den neuen nichtlinearen Wert aus

```
nichtlinearer_wert =
➥ 255.0*(pow(2.0,
➥ samplewert/65536.0)-1.0);
```

Mit dieser einfachen Formel reduzieren Sie den Umfang der *wav*-Daten deutlich, ohne einen großen Verlust an Klangqualität hinzunehmen.

Bevor Sie die Daten wieder abspielen, müssen Sie diese natürlich zurückkonvertieren. Dies geschieht mit einer einfachen Formel:

```
samplewert =
➥ 65535.0*log10(1.0+wert/
➥ 256.0)/log10(2)-32767.0;
```

Die Variable *samplewert* ist hier 16 Bit signed.

Sind Sie mit dem bisherigen Ergebnis noch nicht zufrieden, wenden Sie einen sogenannten Low-Pass-Filter auf die zurückkonvertierten Sample-Daten an. Dieser Filter schwächt die Lautstärke von Klängen, deren Frequenzen über einer gewissen Grenzfrequenz liegen, deutlich ab.

Ein Klang ist – in der Realität – immer eine Mischung aus vielen verschiedenen Frequenzen. Das Rauschen allerdings, das Sie bei den Samples hören, besteht hauptsächlich aus hohen Frequenzen, die sie mit diesem Filter eliminieren.

Den denkbar einfachsten Low-Pass-Filter erhalten Sie, indem Sie den Mittelwert jeweils zweier Sample-Werte bilden:

```
for (i=1;i<anzahl_werte;i++)
➥wert[i] =
➥(wert[i]+wert
➥[i-1])/2;
```

Als letzter Punkt ist noch die Wiedergabe über Ihre Soundkarte zu klären. Für *wav*-Dateien steht Ihnen unter Windows die *PlaySound*-Funktion zur Verfügung, die Sie allerdings aus mehreren Gründen in Demos nicht verwenden sollten. Beispielsweise haben Sie damit nicht die Möglichkeit, an einer bestimmten Stelle die Soundausgabe anzuhalten und sie danach wieder fortzusetzen.

Außerdem können Sie nicht feststellen, an welcher Position innerhalb der Sample-Daten Sie sich befinden. Genau dies brauchen Sie aber, um die seit Abspielbeginn verstrichene Zeit zu berechnen und damit die Synchronisation mit Ihrer Demo zu gewährleisten.

Sie verwenden also besser das Waveform Audio Interface von Windows. Es stellt die nötigen Funktionen zur Verfügung, um beliebige *wav*-Daten abzuspielen.

Sie können auch bestimmte Teile eines Sample mit dem WaveOut-Device loopen, also mehrmals abspielen. So reduzieren Sie die Datenmenge erneut, da Sie identische Passagen eines Musikstücks – in der Praxis ein häufiger Fall – nur einmal speichern.

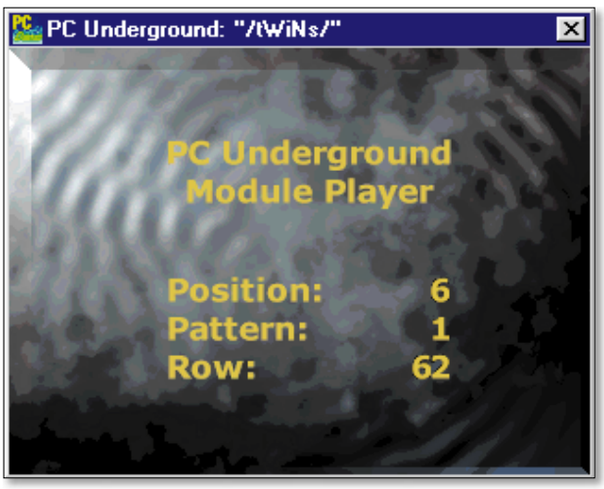
Um die Wiedergabe vorzubereiten, übergeben Sie der Funktion *waveOutOpen* eine *Waveformatex*-Struktur. Dort tragen Sie die von Ihnen gewünschten Parameter für die Soundausgabe ein. Das geeignete Multimedia-Gerät sucht sich *waveOutOpen* dann selbständig:

```
WAVEFORMATEX format;
HWAVEOUT wavehandle;
WAVEHDR wheader;

// ungepackte Sample-Daten
format.wFormatTag =
  WAVE_FORMAT_PCM;
// Monoausgabe
format.nChannels = 1;
// Sampling-Frequenz
format.nSamplesPerSec = 22050;
// Bytes pro Sekunde
format.nAvgBytesPerSec = 22050*2;
format.nBlockAlign = 2;

// 16 Bits pro Sample
format.wBitsPerSample = 16;
format.cbSize = 0;

waveOutOpen(&wavehandle,
  WAVE_MAPPER,&format,0,0,0);
```



UNSER MOD-PLAYER ist für die aktuelle Statusanzeige im Fenster verantwortlich.

Nun haben Sie mit *wavehandle* das richtige Ausgabegerät. Anschließend bereiten Sie den Datenblock vor, der an die Soundkarte geschickt werden soll:

```
// Zeiger auf die Sample-Daten
wheader.lpData =
  (char*)wave16bit;
// Größe des Datenblocks/Byte
wheader.dwBufferLength = size;
wheader.dwBytesRecorded = 0;
wheader.dwUser = 0;
wheader.dwFlags = 0;
wheader.dwLoops = 0;

waveOutPrepareHeader(
  wavehandle,&wheader,
  sizeof(wheader));
```

Damit ist die Soundausgabe initialisiert. Den Startschuß zum Abspielen geben Sie mit folgender Zeile:

```
waveOutWrite(wavehandle,
  &wheader,sizeof(wheader));
```

Um Ihre Demo zu jedem Zeitpunkt mit der gewünschten Musik zu synchronisieren, holen Sie mit der Zeile

```
MMTIME mmtime;
waveOutGetPosition(wavehandle,
  &mmtime,sizeof(mmtime));
```

die aktuelle Abspielposition ein, aus der Sie dann die verstrichene Zeit seit Sample-Beginn berechnen. Sie können die Zeit auch mit Hilfe der Funktion *GetDemoTime()* aus der Basisbibliothek bestimmen, allerdings sollten Sie diese Funktion nur bei kürzeren *wav*-Stücken verwenden. Auf der Heft-CD finden Sie eine vollständige Demo, die ein komprimiertes Sample lädt und dazu synchronisiert einige Effekte der letzten Ausgaben demonstriert.

Auch für dieses Programm benötigen Sie die *demoquit()*-Funktion zum Beenden der *wav*-Ausgabe:

```
waveOutReset(wavehandle);
waveOutUnprepareHeader(
  wavehandle,&wheader,
  sizeof(wheader));
waveOutClose(wavehandle);
```

Nun haben Sie alle Werkzeuge in der Hand, um Ihre eigene Demo mit allem Drum und Dran zu schreiben. Seien Sie kreativ: Wir sind gespannt auf Ihre Resultate! PEI/JR

Die Heft-CD enthält das Programm FastTracker II, die MIDAS-Sound-Bibliothek sowie die vollständige PC Underground Demo. Download-Angebote zum Artikel finden Sie auch in unserem Internet-Angebot unter

www.pc-magazin.de/magazin/➥extras.htm

Klicken Sie in der Tabelle *Online Extras* unter *Praxis* auf das entsprechende *Download*-Feld.

main.cpp

```
 1:
 2:
 3: #include "midasdll.h"
 4: #include "demo.h"
 5:
 6: unsigned short *screen;
 7:
 8: // Wird benötigt, um den Fenstertitel zu ändern
 9: extern HWND DemoHWND;
10:
11: // MIDAS-spezifische Deklarationen
12: extern void InitMIDAS(void);
13: extern void CloseMIDAS(void);
14: extern void StartupMIDAS(void);
15: extern MIDASmodulePlayHandle PlayModule(MIDASmodule
16:   module);
17: extern MIDASmodule LoadModule(char *fileName);
18: extern void StopFreeModule(MIDASmodulePlayHandle
19:   playHandle,MIDASmodule module);
20:
21: MIDASmodule module;
22: MIDASmodulePlayHandle playHandle;
23: MIDASmoduleInfo info;
24: MIDASplayStatus status;
25:
26: // Variablen für unseren kleinen Module-Player
27: short palette1[32][256];
28: bitmaptype background;
29: unsigned char *back;
30: unsigned short *back16;
31: short palette2[32][256];
32: bitmaptype zahlen;
33: unsigned char *zahl;
34:
35: BOOL demoinit (void)
36: {
37:   Fenster_Modus = FENSTER;
38:
39:   // Speicher für das Bild reservieren
40:   screen = (unsigned short *)malloc(SCREEN_X*SCREEN_Y*2);
41:   if (screen == NULL) return 0;
42:   memset(screen,0,SCREEN_X*SCREEN_Y*2);
43:
44:   // Hintergrundbild und Zahlen-Bild laden
45:   if (bmp_load("HINTERGRUND.BMP",background) !=
46:     BMP_NOERROR) return 0;
47:   if (bmp_load("ZAHLEN.BMP",zahlen) !=
48:     BMP_NOERROR) return 0;
49:
50:   bmp_make16bitpalette(background);
51:   bmp_make16bitpalette(zahlen);
52:
53:   back = (unsigned char *)background.cBitmap;
54:   zahl = (unsigned char *)zahlen.cBitmap;
55:
56:   // Hintergrundbild einmalig in Hicolor Umwandeln
57:   back16 = (unsigned short *)malloc(320*240*sizeof(short));
58:   if (back16 == NULL) return FALSE;
59:   for (int ofs = 0; ofs<SCREEN_X*SCREEN_Y; ofs ++)
60:     back16[ofs] = background.sColors[back[ofs]];
61:
62:   // MIDAS starten,initialisieren und Module laden
63:   StartupMIDAS();
64:   InitMIDAS();
65:   module = LoadModule("test.s3m");
66:
67:   return TRUE;
68: }
69:
70:
71: // Gibt eine Zahl aus
72: void write_single(int x,int y,int v)
73: {
74:   int i,j;
75:   int ofs = x+y*SCREEN_X;
76:
77:   for (j=0; j<14; j++)
78:   {
79:     for (i=0; i<10; i++)
80:     {
81:       int p = zahl[i+(j+v*14)*320];
82:       if (p != 16) screen[ofs] = zahlen.sColors[p];
83:       ofs++;
84:     }
85:     ofs += SCREEN_X-10;
86:   }
87: }
88:
89: // maximal 3-stellige Zahl ausgeben,
90: // x ist rechtsbundige Koordinate
91: void write_number(int x,int y,int v)
92: {
93:   write_single(x-10,y,v%10);
94:   v /= 10; if (v>0) write_single(x-20,y,v%10);
95:   v /= 10; if (v>0) write_single(x-30,y,v%10);
96: }
97:
98: void demomain(void)
99: {
100:   // Modulenamen in den Fenstertitel !
101:   MIDASgetModuleInfo(module,&info);
102:
103:   char text[128];
104:   strcpy(text,"PC Underground: \"");
105:   strcat(text,info.songName);
106:   strcat(text,"\"");
107:   SetWindowText(DemoHWND,text);
108:
109:   // Abspielen starten
110:   playHandle = PlayModule(module);
111:
112:   // Hintergrundbild komplett kopieren
113:   memcpy(screen,back16,320*240*2);
114:
115:   while (DemoRunning)
116:   {
117:     // Nur den Teil des Hintergrundbildes,
118:     // der überschrieben wird, kopieren
119:     for (int i=125; i<125+56; i++)
120:       memcpy(screen+i*320+208,back16+i*320+208,30*2);
121:
122:     // Informationen holen
123:     MIDASgetPlayStatus(playHandle,&status);
124:
125:     // Und darstellen
126:     write_number(238,125,status.position);
127:     write_number(238,146,status.pattern);
128:     write_number(238,167,status.row);
129:
130:     BlitGraphic(screen);
131:
132:     // Läßt den Demotask 50ms warten
133:     // Dies ermöglichst es, den Player
134:     // im Hintergrund laufen zu lassen
135:     Sleep(50);
136:   }
137: }
138:
139: void demoquit(void)
140: {
141:   // Ausgabe beenden und MIDAS beenden
142:   StopFreeModule(playHandle,module);
143:   CloseMIDAS();
144: }
```

Der einfache Module-Player beherrscht viele Formate, darunter MOD-, S3M- und XM-Dateien.