AUF DER HEFT-CD

Spiele-Programmierung unter Windows 95/98/NT

Klänge für den Weltraum

Für ein Weltraum-Ballerspiel kapseln Sie DirectSound-Aufrufe in einer eigenen Klasse und synthetisieren Soundeffekte.

CARSTEN DACHSBACHER/
NILS PIPENBRINCK

Das Interesse vieler Computerfreaks an Science Fiction erklärt die Existenz unzähliger Weltraumspiele für den PC. Sie müssen weder ein Freak sein noch jede Folge Raumschiff Enterprise auswendig kennen, um mit uns in den nächsten drei Ausgaben des *PC Magazin* ein kleines Weltraumspiel für Windows zu schreiben. Als Vorlage dient der Klassiker *Gravity Wars*, wie es ihn früher auf Videospielautomaten oder für PCs mit Hercules- und CGA-Karte gab.

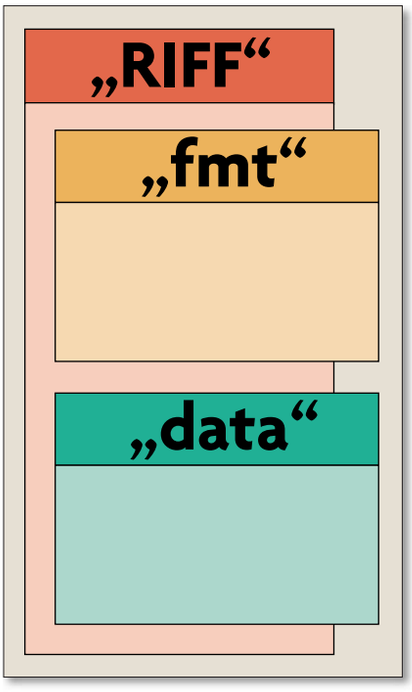
Bei diesem Spiel steuern zwei Spieler je einen Raumgleiter über ein zweidimensionales Spielfeld, das den Weltraum darstellt. Die Gleiter lassen sich drehen, beschleunigen und wieder abbremsen. Außerdem stehen jedem Spieler eine Laserkanone sowie Raketen zur Verfügung, mit denen er versucht, den Gegner abzuschießen (das alles ist natürlich nur ein Spiel, und die Raketen sind mit harmlosem Joghurt gefüllt).

Jeder Raumgleiter verfügt über Energie, die der Spieler auf die Waffen- und Schildsysteme verteilt. Optional können Sie in der Mitte des Spielfelds noch einen Planeten plazieren, der alle Objekte wie Raumgleiter und Raketen anzieht (daher der Name *Gravity Wars*). Je nach Lust und Laune sind zusätzliche Erweiterungen denkbar – einige werden Sie in den nächsten Ausgaben von PC Underground realisieren.

In diesem ersten Teil legen Sie den Grundstein für Ihr eigenes Spiel. Damit es die richtige Atmosphäre bietet, brauchen Sie schnelle, flüssige Grafikroutinen, ansprechende Soundeffekte und eine vernünftige Steuerung.

Die Grafikbibliothek

Als Grafikbibliothek verwenden Sie am besten eine modifizierte Version der Demobibliothek, die bereits allen früheren Programmen in dieser Rubrik als Grundlage diente. Doch keine Angst: Sie müssen jetzt keine alten Ausgaben des *PC Magazin* durchstöbern, um sich die Routinen des Grafiksystems wieder in Erinnerung zu rufen:



***WAV*-DATEIEN ENTHALTEN** je einen Format- und einen Daten-Chunk.

In der Textbox „Funktionen der Grafikbibliothek“ auf S. 229 stellen wir Ihnen alle Funktionen der Bibliothek noch einmal vor. Dieses leicht erneuerte Basissystem finden Sie komplett mit allen Quellcodes auf der Heft-CD in der selbstentpackenden Archiv-Datei *Basissystem.exe* und auf der Homepage des *PC Magazin*.

Der Einsatz in einem Spiel verlangte einige Änderungen an den bisherigen Routinen: Um Zugriff auf die Tastatur zu erhalten, fängt die Hauptschleife des Programms jetzt die Window-Messages der Tastatur ab. Der aktuelle Zustand der Tasten wird im Array *KeyState* gespeichert. In Ihren Anwendungen lesen Sie damit die Zustände der einzelnen Tasten aus.

Bitte beachten Sie, daß Ihr Programm nicht mehr automatisch bei jedem Tastendruck beendet wird. Ob die *[Esc]*- oder Leertaste gedrückt wurde, überprüfen Sie jetzt selbst. Die Hauptschleife Ihrer Demo bzw. Ihres Spiels sieht damit wie folgt aus:

```
void demomain (void)
{
  short * bild = new short
    [ SCREEN_X * SCREEN_Y ];

  while ( ( DemoRunning ) &&
    ( !KeyState[VK_ESCAPE] ) )
  {
    /* hier zeichnen Sie was */
    ...


    BlitGraphic( bild );
  }

  delete bild;
}
```

Grundlagen von DirectSound

Mit DirectSound wählen Sie einen modernen Weg, um einfach Soundeffekte unter Windows abzuspielen. Die DirectSound-API ist ein Bestandteil von DirectX. Haben Sie noch keine Erfahrung mit DirectX, ist dies gar kein Problem: Im Gegensatz zu DirectDraw ist die Soundprogrammierung erfreulich einfach.

Um das Abspielen von Soundeffekten so einfach und angenehm wie möglich zu gestalten, bauen Sie keine spezifischen Aufrufe von DirectSound in das Hauptprogramm ein. Statt dessen schreiben Sie eine C++-Klasse, die die genaue Implementierung versteckt.

Diese Technik bietet einige Vorteile: Zum einen müssen Sie sich bei der Spieleprogrammierung nicht mit den Sy-

stemaufrufen von DirectSound herumschlagen. Zum anderen können Sie eine gut durchdachte C++-Klasse auch in anderen Programmen wiederverwenden.

Mit DirectSound können Sie – im Gegensatz zum alten Multimedia-System – mehrere Soundeffekte gleichzeitig abspielen. Außerdem besitzen Sie auch während des Abspielens die volle Kontrolle über Lautstärke, Frequenz und Position im Stereoraum des Samples. Um das zu verwirklichen, gingen die Programmierer von DirectSound einen ungewöhnlichen Weg.

Nahezu alle herkömmlichen Soundbibliotheken unter Windows benutzen das Konzept der Soundkanäle. Bei der Initialisierung legen Sie fest, wie viele Kanäle für das gleichzeitige Abspielen von Samples bereitstehen. Dabei ist es egal, ob alle Kanäle einen Soundeffekt wiedergeben oder jeder Kanal einen anderen abspielt. Bei DirectSound sind Samples und Kanäle direkt miteinander verbunden. Wenn Sie einen Kanal (im DirectSound-Jargon heißen diese *DirectSoundBuffer*) anlegen, kann dieser immer nur einen einzigen Soundeffekt abspielen.

Das hat einige Folgen für Sie: Sie müssen schon während der Initialisierung des Soundsystems festlegen, wie viele Kanäle Sie zum Beispiel für Schußgeräusche verwenden möchten. Glücklicherweise ist DirectSound relativ intelligent. Soundeffekte, die nicht zu hören sind, brauchen zwar etwas Speicher, aber kaum Rechenzeit.

DirectSound ist ein sehr mächtiges Werkzeug. Deshalb kann dieser Artikel auch nur die Oberfläche dessen betrachten, was alles mit DirectSound machbar ist. Bitte beachten Sie, daß wir aus Platzgründen die Abfrage von Fehlern in den Beispielcodes weggelassen haben.

DirectSound im Detail

Zunächst legen Sie ein *IDirectSound*-Objekt an. Damit steuern Sie später die Soundkarte des PC:

```
IDirectSound
  *DSound;
HWND
  FensterHandle;

DirectSoundCreate(NULL,
  &DSound,0);
DSound->SetCooperativeLevel
  (FensterHandle,DSSCL_EXCLUSIVE)
```

DirectSoundCreate erzeugt das *IDirectSound*-Objekt. Durch den Aufruf von *SetCooperativeLevel* teilen Sie Windows mit, daß Sie von nun an direkten Zugriff auf die Soundkarte haben möchten. Das ist zwar nicht unbedingt nötig, aber Sie vermeiden so eine Menge Ärger. Programme, die im Hintergrund laufen und auch auf die Soundkarte zugreifen möchten, werden nun allerdings blockiert und können nicht mehr auf Ressourcen der Soundkarte zugreifen.

Fordern Sie einen Soundpuffer vom System an. Dafür füllen Sie – wie es allgemein bei DirectX üblich ist – einige Strukturen aus und definieren dabei genau, in welchem Format der Soundeffekt vorliegt. In diesem Beispiel erzeugen Sie ein zwei Sekunden langes 8-Bit-Mono-Sample mit einer Sampling-Frequenz von 22050 Hertz:

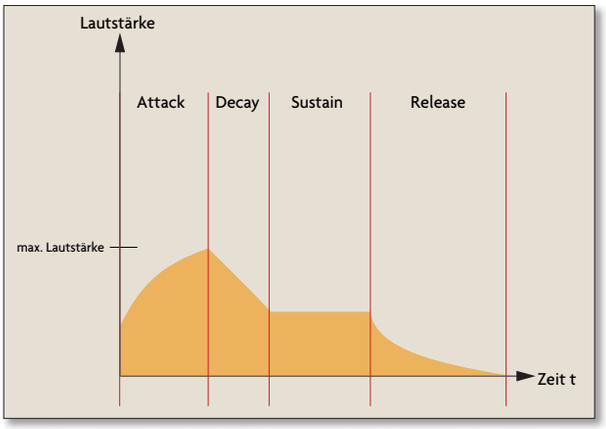
```
WAVEFORMATEX Format;
DSBUFFERDESC bd;
IDirectSoundBuffer Buffer;

// Strukturen löschen
memset(&Format, 0,
  sizeof (WAVEFORMATEX));
memset(&bd, 0, sizeof (bd));

// Benötigte Felder ausfüllen
Format.wFormatTag =
  WAVE_FORMAT_PCM;
Format.nChannels = 1;
Format.nSamplesPerSec = 22050;
Format.nAvgBytesPerSec = 22050;
Format.nBlockAlign = 1;
Format.wBitsPerSample = 8;

bd.dwSize = sizeof (bd);
bd.dwFlags =
  DSBCAPS_CTRLDEFAULT |
  DSBCAPS_STATIC;
bd.dwBufferBytes = 44100;
bd.lpwfxFormat = &Format;

// und den Buffer erstellen.
DSound->CreateSoundBuffer
  (&bd, &Buffer, NULL);
```

Interessant sind hier die Flags, die Sie im Feld *bd.dwFlags* übergeben. Dort bestimmen Sie, welche Fähigkeiten von DirectSound Sie benutzen. In diesem Beispiel setzen Sie die Flags *CTRLDEFAULT* und *STATIC*.

STATIC sagt DirectSound, daß Sie den Soundeffekt nicht ständig ändern möchten. Dadurch kann DirectSound bei einigen Modellen die Sample-Daten direkt auf die Soundkarte übertragen und dort von der Hardware abspielen lassen. Dies entlastet den Prozessor erheblich.

Das *CTRLDEFAULT*-Flag besagt nur, daß Sie Frequenz, Lautstärke und die Stereoposition des Samples kontrollieren möchten.

Jetzt füllen Sie den Puffer mit Sample-Daten. Dazu rufen Sie die Funktion *Lock* auf, die Ihnen einen Pointer auf ▶



DIE ADSR-HÜLLKURVE bestimmt mit ihren vier Parametern die Lautstärke des generierten Tons.

FUNKTIONEN DER GRAFIKBIBLIOTHEK

Um die Grafikbibliothek zu nutzen, implementieren Sie folgende drei Funktionen in Ihrem Programm:

- In der Funktion *BOOL demoinit() { ... }* erledigen Sie alle nur einmal benötigten Initialisierungen. Sind diese alle geglückt, gibt die Funktion den Wert *true* zurück. Hier belegen Sie auch die Variable *Fenster_Modus* mit dem gewünschten Bildschirmmodus (siehe Quellcode von *demosys.h*).
- Innerhalb von *void demomain() { ... }* läuft Ihr Hauptprogramm wie in einer gewöhnlichen *main*-Funktion.
- In der Funktion *void demoquit() { ... }* kann Ihr Programm die zum Programmende notwendigen Einzelschritte durchführen.

Sodann nutzen Sie folgende Funktionen:

- *void BlitGraphic(void *)* stellt den Inhalt des als Zeiger übergebenen HighColor-Bildschirmpuffers im Fenster oder im DirectX-Vollbildmodus dar. Im HighColor-Modus beansprucht jedes Pixel 16 Bit (5 für Rot, 6 für Grün und 5 für Blau).
- *unsigned short ColorCode(r,g,b)* übernimmt für Sie die Berechnung des HighColor-Farbwerts aus den RGB-Angaben. Liegen Rot, Grün und Blau jeweils im Bereich von *0* bis *255*, berechnen Sie aus Geschwindigkeitsgründen den Farbwert wie folgt:

```
color = Rtab[R]|Gtab[G]|Btab[B];
```

- *unsigned long GetDemoTime(void)* liefert Ihnen die seit dem Programmstart abgelaufene Zeit in Millisekunden.

den internen Speicher des Puffers zurückgibt. Die Funktion *Lock* ist so programmiert, daß Sie auch Zugriff auf die Samples bekommen, wenn der Sound gerade abgespielt wird. Das ist ein nützliches Feature, aber für unsere Zwecke gar nicht nötig.

```
void *Ptr1;
void *Ptr2;
unsigned long Size1;
unsigned long Size2;

Buffer->Lock(0,44100,&Ptr1,
  &Size1,&Ptr2,&Size2,0);
```

Die ersten beiden Parameter bezeichnen die Startposition und die Anzahl der Samples, auf die Sie Zugriff wünschen. Oben fordern Sie also Zugriff auf das gesamte Sample an.

Nachdem Sie die Funktion *Lock* aufgerufen haben, kopieren Sie Ihre Daten in den Speicherbereich, auf den *Ptr1* zeigt. Die beiden Variablen *Ptr2* und *Size2* brauchen Sie nur, wenn Sie einen Teil des Samples ändern möchten. Für unser Spiel ist dies aber nicht nötig. Jetzt beginnen Sie bereits mit der Ausgabe:

```
Buffer->Unlock (Ptr1, Size1,
  Ptr2, Size2);
Buffer->Play (0,0,0);
```

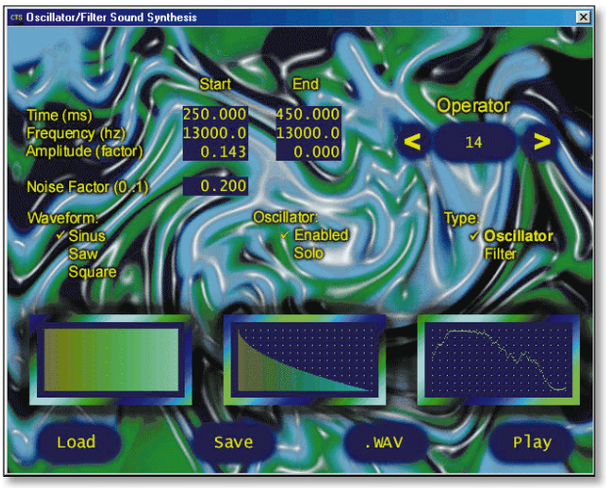
Über den Aufruf von *Unlock* teilen Sie DirectSound mit, daß Sie mit dem Schreiben fertig sind. *Play* startet das Abspielen des Samples. Wenn alles geklappt hat, hören Sie jetzt Ihren Soundeffekt.

Die Klasse SoundSystem

DirectSound ist zwar eine übersichtliche API, bereitet dem Programmierer aber trotzdem viel Arbeit. Deshalb haben wir Ihnen – wie bereits angekündigt – eine einfache Schnittstelle programmiert, mit der alles einfacher wird. Dieses sehr einfache Soundsystem finden Sie in den Dateien *soundsys.cpp* und *soundsys.h*. Ein kleines Beispiel zeigt die Anwendung:

```
SoundSystem Sound;
int Sample1, int Sample2;

Sound.initialize
  (Program_Fenster);

// Boing.WAV + Peng!.WAV laden
// jeweils 10 Soundkanäle für
// Samples vorbereiten
Sample1 = Sound.LoadSound
  ("Boing.WAV", 10);
Sample2 = Sound.LoadSound
  ("Peng!.WAV", 10);

// Zum Test Samples abspielen
Sound.PlaySound (Sample1);
Sound.PlaySound (Sample2);
```

Die Klasse *SoundSystem* kümmert sich um die interne Verwaltung von Soundpuffern, das Laden von Samples und die Fehlerbehandlung von DirectSound.



MIT UNSEREM SOUNDEDITOR mischen Sie per Mausklick tolle Effekte zusammen.

Bevor Sie jetzt loslegen und Ihre Nachbarn mit komischen Geräuschen unterhalten, noch ein Hinweis: Falls Sie einen anderen Compiler als Watcom C++ benutzen, fügen Sie Ihrem Projekt bzw. Makefile unbedingt die Bibliotheksdatei *dsound.lib* hinzu – sonst klappt's nicht mit dem Nachbarn. Das selbstentpackende *zip*-Archiv *SoundBeispiel.exe* enthält ein kleines Demoprogramm, das ein Donnergrollen und ein kurzes Ploppen zeitgleich abspielt.

Aufbau einer *wav*-Datei

In der Windows-Welt sind *wav*-Dateien das Standardformat für Sounds. Deshalb ist es sinnvoll, dieses Format für das Soundsystem zu benutzen, auch wenn es nicht ganz einfach zu lesen ist. Das *wav*-Format ist eine Unterform des *Resource Interchange File Format* (RIFF), welches Windows für die meisten Multimedia-Daten verwendet (neben Samples auch Bilder und Animationen).

Eine RIFF-Datei setzt sich aus mehreren kleinen Datenblöcken (sogenannten Chunks) zusammen. Den genauen Aufbau stellen Sie sich am besten wie eine Verzeichnisstruktur vor: Jeder Chunk kann mehrere Unter-Chunks besitzen (vergleichbar mit Unterverzeichnissen) und auch Daten enthalten.

Zur eindeutigen Identifikation tragen alle Chunks eine aus vier Buchstaben bestehende Kennung. Diese nutzen Sie, um Windows gezielt nach benötigten Daten-Chunks suchen zu lassen oder sich die Struktur eines unbekannten Dateityps anzusehen. Jede RIFF-Datei beginnt mit einem Chunk namens *RIFF*. Dieser Chunk enthält keine Daten, sondern nur Unter-Chunks. Er ist sozusagen das Hauptverzeichnis der RIFF-Datei. Für das *wav*-Format sind die Unter-Chunks genau definiert (siehe Bild S. 228).

Der erste ist der Format-Chunk mit der Kennung *fmt*. Dort gibt eine *WAVEFORMATEX*-Struktur genaue Auskunft über das Sample-Format. Sie informiert unter anderem über die Anzahl der verwendeten Kanäle (Mono oder Stereo), die Abtastrate und Bit-Tiefe:

```
struct WAVEFORMATEX
{
  WORD wFormatTag;
  WORD nChannels;
```

GLOSSAR ZUR KLANGSYNTHESE

Begriff	Bedeutung
ADSR	Abschnitte der Hüllkurve • *Attack* = Anschlag • *Decay* = Verzögerung nach dem Anschlag • *Sustain* = Haltezeit der Lautstärke nach dem Decay • *Release* = Abfallzeit der (Rest-)Lautstärke
Filter	schwächt oder verstärkt bestimmte Frequenzanteile in einem Klang
Hüllkurve	Lautstärkeverlauf für einen Oszillator
Oszillator	Schwingkreis, der durch Spannungswerte eine bestimmte Wellenform erzeugt
Phase	Position innerhalb einer Schwingungsperiode eines Oszillators
Wellenform	Art der Schwingung (zum Beispiel Sinus-, Rechteck- oder Sägezahnschwingung)

```
    DWORD nSamplesPerSec;
    DWORD nAvgBytesPerSec;
    WORD nBlockAlign;
    WORD wBitsPerSample;
    WORD cbSize;
};
```

Die Felder *nAvgBytesPerSec* und *nBlockAlign* benutzt Windows, um komprimierte Dateien abzuspielen. Da Sie unter DirectSound lediglich unkomprimierte Samples verwenden, können Sie diese Felder getrost ignorieren.

Direkt auf den Format-Chunk folgt der Daten-Chunk namens *data*. Er enthält die rohen Sample-Daten. Eine hervorragende Übersicht über den Aufbau verschiedenster Dateiformate erhalten Sie – wenn auch in englischer Sprache – auf der Web-Seite

```
www.wotsit.org
```

Die Multimedia-IO-Funktionen

Die Ein- und Ausgabe (IO = Input/Output) bei Multimedia-Dateien entspricht weitestgehend dem Umgang mit gewöhnlichen Dateien. Nachdem Sie eine Multimedia-Datei geöffnet haben, lassen Sie Windows den RIFF-Chunk suchen. Zwar steht dieser Chunk in der Regel am Anfang der Datei, aber es gibt auch einige wenige RIFF-Dateien, die davor noch zusätzliche Informationen bereithalten.

Rufen Sie zum Suchen die Funktion *mmioDescend* auf, die dem Wechseln in ein Unterverzeichnis entspricht. Nun sind Sie im Haupt-Chunk der *wav*-Datei. Von hier aus können Sie durch erneuten Einsatz von *mmioDescend* den Format- und Daten-Chunk suchen und lesen.

Um in den übergeordneten Chunk zurückzuwechseln, rufen Sie die Funktion *mmioAscend* auf. Im DOS-Dateisystem entspräche dieser Befehl dem Kommando

```
cd..
```

Das folgende Beispiel-Listing enthält aus Platzgründen keine Fehlerabfragen. Für einen sauberen Programmierstil sollten Sie dies aber auf keinen Fall versäumen. Eine leicht zu benutzende C++-Klasse zu *wav*-Dateien finden Sie in den Dateien *LoadWav.cpp* und *LoadWav.h*. Dort haben wir für Sie die Multimedia-IO-Funktionen soweit gekapselt, daß Sie sich keine Gedanken um Details machen müssen.

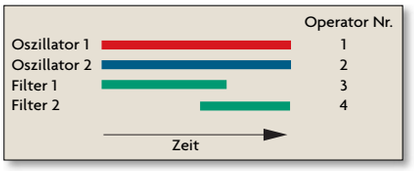
```
HMMIO ioHandle;
MMCKINFO ckInRIFF;
MMCKINFO fmtChunk;
MMCKINFO dataChunk;
WAVEFORMATEX format;

IoHandle = mmioOpen
  ("test.wav",0,MMIO_READ);
fmtChunk.ckid = mmioFOURCC
  ('f','m','t',' ');
dataChunk.ckid = mmioFOURCC
  ('D','A','T','A');

// In den RIFF-Chunk wechseln
mmioDescend(ioHandle,
  &ckInRIFF, NULL, 0);
// in den Format-Chunk wechseln
mmioDescend(ioHandle,
  &fmtChunk, &ckInRIFF,
  MMIO_FINDCHUNK);
// Wave-Format lesen
mmioRead(ioHandle, &format,
  sizeof (format));
// zurück in den RIFF-Chunk
mmioAscend(ioHandle,
  &fmtChunk, 0);

// In den Daten-Chunk wechseln
mmioDescend(ioHandle,
  &dataChunk, &ckInRIFF,
  MMIO_FINDCHUNK);
// Daten lesen
mmioRead(ioHandle, samples,
  dataChunk.cksize);
// und Datei wieder schließen
mmioClose (ioHandle, 0);
```

Wenn Sie sich intensiver mit der Idee der auf Chunks basierenden Dateien auseinandersetzen, werden Sie sicher viele Vorteile finden. RIFF-Dateien können Sie selbstverständlich nicht nur für die Windows-eigenen Datentypen benutzen. Entwerfen Sie Ihren eigenen Dateityp und schreiben Sie hinein, was immer Sie möchten.

Klangsynthese

Jetzt, wo Sie wissen, wie Sie *wav*-Dateien abspielen, möchten Sie sicherlich Ihre eigenen Soundeffekte für Ihr Spiel generieren und speichern. Falls Sie nicht mit einem Mikrofon losziehen und echte Geräusche aufnehmen (samplen), erzeugen Sie die Klänge künstlich. Diese synthetisierten Klänge können das Ergebnis einer elektronischen Schaltung sein oder aber von einem PC berechnet werden. Die wichtigsten hier verwendeten Begriffe zur Klangsynthese finden Sie im Glossar auf S. 230.

Wenn Sie einer etwas älteren Soundkarte lauschen, die noch nicht über Wavetable-Klänge verfügt, stammen die Töne wahrscheinlich von OPL2- bzw. OPL3-Chips. Diese Chips arbeiten nach der FM-Synthese und sind noch immer sehr verbreitet, etwa auf den Sound-Blaster-Karten und allen mehr oder weniger kompatiblen Konkurrenten.



IN DIESER SCHEMATISCHEN Beispielzeichnung verwendet jeder Operator das darüberliegende Signal als Eingang.

Die Töne, die physikalisch nichts anderes sind als Schwingungen, entstehen in diesen Chips durch mehrere Schwingkreise, auch Oszillatoren genannt. Diese arbeiten entweder unabhängig voneinander oder so gekoppelt, daß beim OPL2-Chip jeweils zwei (beim OPL3-Chips sogar vier) Oszillatoren ihre aktuelle Phasenposition in Wechselwirkung beeinflussen. Eine ADSR-Hüllkurve – die Abkürzung ADSR finden Sie im

PARAMETER DER OSZILLATOREN BEI DER KLANGSYNTHESE

Parameter	Bedeutung	Bemerkung
Start- und Endzeit	Zeitintervall, in dem der Oszillator einen Klang erzeugt	Angabe in Millisekunden
Start- und Endfrequenz	Frequenz, mit der der Oszillator am Beginn bzw. am Ende seiner Spielzeit schwingt	Graph im linken unteren Eck des Editorfensters zeigt Verlauf an
Start- und Endamplitude	Anfangs- und Endlautstärke des Oszillators	Graph unten in der Mitte zeigt Verlauf an
Noise Factor	Maß für die zufällige Änderung der Oszillatorphase	*0* = keine Phasenänderung, *1* = maximale Phasenänderung (mit dieser Einstellung können Sie Rauschen erzeugen)
Curve Tone	Exponentielle Umrechnung des Oszillatorausgangs	Bei einer Sägezahnschwingung sehen Sie die Auswirkungen am besten.
Waveform	Sinus-, Sägezahn- oder Rechteckschwingung	Im Graphen rechts unten im Editorfenster erkennen Sie auch die Auswirkungen des Noise-Faktors.

ENTWICKLUNGSSTUFEN DES SPIELEPROJEKTS

Teil 1:
- Entwicklung des Basissystems
- DirectSound-Programmierung
- Soundeffekt-Programmierung/ Klangsynthese

Teil 2:
- Sprite-Programmierung
- Partikel- und Effektsystem

Teil 3:
- Algorithmen zur Kollisionsabfrage
- Spielelogik
- Spielegrafik und Highscore-Routinen
- Musik

„Glossar zur Klangsynthese“, S. 230 – modifiziert dabei die Ausgabelautstärke der Schwingkreise (siehe Bild S. 229)

Im Gegensatz zur FM-Synthese mischt die Wavetable-Synthese zuvor aufgenommene Samples von echten Instrumenten, die entweder im ROM oder RAM der Soundkarte gespeichert sind, zu einem Klang zusammen. Damit Sie auch ohne Programmieraufwand Ihre individuellen Geräusche zusammenstellen können, haben wir außer den Routinen zum Laden und Berechnen der Klänge noch einen komfortablen Soundeditor geschrieben (siehe Bild S. 230).

Ein Beispielprogramm zur Berechnung der Sounds finden Sie in der Archivdatei *MakeSound.exe*, den Soundeditor in der Datei *FSSEditor.exe*.

Die Klangsynthese, die dieser Editor verwendet, arbeitet mit einer ganzen Reihe von (zeitweise) abhängigen Oszillatoren und Filtern. Um die Ausgabe der Oszillatoren zu beeinflussen, stehen Ihnen die Parameter aus der Tabelle „Parameter der Oszillatoren“ auf S. 233 zur Verfügung.

Filtern Sie die durch die Oszillatoren erzeugte Schwingung. Dieses Verfahren verwendet einen relativ einfachen Filteralgorithmus, dessen mathematische bzw. physikalische Funktionsweise uns hier nicht weiter beschäftigen soll. Die damit modellierten Filter besitzen zwei Parameter:

- Die sogenannte Cut-off-Frequenz gibt an, welche Frequenz verstärkt wird. Die Verstärkung betrifft allerdings nicht nur genau die eingestellte Frequenz, sondern einen Frequenzbereich.
- Der zweite Parameter ist die Resonanz, die gewissermaßen die Hervorhebung des gewählten Frequenzbereichs angibt. Die Werte der Resonanz bewegen sich sehr häufig im Bereich zwischen 0,99 und 1,0.

In unserem Soundeditor ist der linke, untere Kasten der Frequenz zugeordnet, der Kasten in der Mitte der Resonanz. Zusätzlich können Sie jeden der Operatoren, also Oszillator oder Filter, einzeln an- und ausschalten und Oszillatoren auch allein spielen lassen.

Den Editor bedienen Sie vollständig über die Maus. Die einzelnen Werte modifizieren Sie, indem Sie mit der Maus darauf klicken, den Knopf gedrückt halten und die Maus nach links oder rechts bewegen. Genauso ändern Sie auch die Frequenz-, Amplituden- und Resonanzkurven in den Kästen unten im Editorfenster. Sie wechseln bequem zwischen Oszillator und Filter, indem Sie einfach den entsprechenden Typ anwählen. Einstellungen gehen dadurch nicht verloren. Durch einfaches Anklicken legen Sie auch eine Wellenform fest und (de-)aktivieren Operatoren.

Wie bereits erwähnt, sind die Operatoren bedingt voneinander abhängig. Ein Filter beeinflußt die Ausgabe aller Operatoren mit einer niedrigeren Operatornummer als die eigene, sofern sich die Zeitbereiche überlappen. Ein Operator addiert seinen Ausgang einfach auf das (Zwischen-)Ergebnis aller Operatoren mit niedrigerer Nummer hinzu.

Am besten sehen Sie das an einem kleinen Beispiel (siehe Bild S. 233): Oszillator 2 addiert seinen Ausgang auf Oszillator 1, er mischt ihn also hinzu. Filter 1 filtert das Ergebnis der beiden Oszillatoren. Filter 2 filtert zuerst den Ausgang von Filter 1, und nachdem dieser nicht mehr aktiv ist, direkt den Ausgang der beiden Oszillatoren. Sie lernen die genaue Funktionsweise dieser Verfahren kennen, indem Sie die beigelegten Beispieleffekte durch Herumprobieren verändern. Mit dem Soundeditor generieren Sie statt Spieleffekte auch problemlos Instrument- und Schlagzeugsamples.

Blick hinter die Kulissen

Um aus den eingestellten Parametern ein Sample zu berechnen, gehen Sie folgenden Weg: Legen Sie für jeden Operator zusätzliche Variablen für die aktuelle Phasenposition und das Zwischenergebnis der Berechnung an. Diese setzen Sie natürlich vor jeder Berechnung eines Sample-Werts auf 0.

Nun behandeln Sie für jeden Sample-Wert alle Operatoren nacheinander, die zum aktuellen Zeitpunkt aktiv sind. Handelt es sich um einen Oszillator, dann berechnen Sie anhand der aktuellen Frequenz die Phasenveränderung seit dem letzten Sample-Wert und die aktuelle Amplitude. Diese beiden Werte ergeben sich aus der Start- und Endfrequenz sowie der exponentiellen Funktion des Frequenzgraphen.

Addieren Sie das Phasendelta auf die Phasenposition, und zählen Sie noch einen Zufallswert hinzu, den Sie vorher mit dem *Noise Factor* multiplizieren. Schließlich bestimmen Sie aus der Phasenposition den aktuellen Wert der Schwingung und modifizieren ihn mit dem Wert von *Curve Tone*.

Hier noch einmal das Prinzip des Oszillators in Pseudoquellcode:

```
Berechne Amplitude und
  Frequenzdelta
Phase+=Frequenzdelta+
  Zufallswert*Noise Factor
Ausgangswert=
  (Schwingfunktion(Phase)^
  Curve Tone)*Amplitude
Zwischenergebnis+=Ausgangswert
```

Handelt es sich beim aktuellen Operator hingegen um einen Filter, so schreiben Sie diesen Wert in den Filter und lesen das Ergebnis nach dem Filtern wieder aus. Das Ergebnis erhalten Sie durch in der Filterstruktur gespeicherte Werte sowie den neu hinzugekommenen Wert. Der Pseudocode sieht dann folgendermaßen aus:

```
Berechne Cutoff-Frequenz und
  Resonanz und setze diese Werte
Schreibe Zwischenergebnis in
  den Filter
Zwischenergebnis=Filterergebnis
```

Die genaue Implementierung sehen Sie in den Quelltexten ein. Aber auch ohne dieses Hintergrundwissen zur Klangsynthese können Sie sich nun die gewünschten Klänge für Ihr Spiel zurechtbasteln. In der nächsten Ausgabe legen Sie dann mit der Programmierung von Sprites und einem Partikel- und Effektsystem den Grundstein für Ihre Spielegrafik. Im übernächsten und letzten Teil fügen Sie daraus ein vollwertiges Arcade-Game zusammen. PEI/BM

Die Quelltexte und Kompilate der Klangsynthese sowie des kompletten Soundeditors finden Sie mit der zugrundeliegenden Grafikbibliothek auf unserer Heft-CD im Verzeichnis *praxis\underground* und im Internet-Angebot des PC Magazin unter
`www.pc-magazin.de/magazin/extras.htm`
Klicken Sie unter *Online Extras* im Menü *Praxis* auf das entsprechende *Download*-Feld.