DirectSound und DirectMedia

Guter Ton

Statten Sie Ihre Programme mit Hilfe der DirectX-Komponente *DirectSound* so mit 3D-Klängen aus, dass sich ein Spieler am PC wie in einer Spielhölle fühlt.

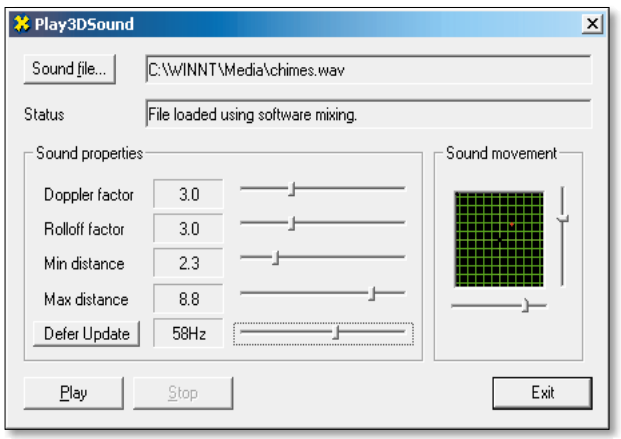
C. DACHSBACHER / O. KÄFERSTEIN

Als Programmierer können Sie optische Reize mit akustischen Signalen untermalen. Schöpfen Sie das Arsenal der Audiofunktionen in DirectX7 aus, um eine Demo, ein Spiel oder eine Multimedia-Anwendung zu bereichern. DirectX unterscheidet dabei zwei Komponenten: *DirectSound* und *DirectMusic*.

Mit der API (Application Programming Interface) *DirectSound* können Sie Wave-Daten abspielen und aufnehmen. *DirectSound* verschiebt Sound in den Ausgabepuffer der Soundkarte. *DirectSound* beschleunigt die Hardware und lässt sie auf die Sound-Hardware zugreifen. Bei diesem Zugriff bleibt die Kompatibilität zu den Gerätetreibern gewahrt.

Die zweite Audiokomponente, *DirectMusic*, ist der Teil des DirectX-Systems, der Musik ausgibt. *DirectMusic* arbeitet nicht mit Wave-Daten, sondern mit auf Nachrichten basierenden Musikdaten. Dazu gehören MIDI-Dateien (Musical Instrument Digital Interface), die die MIDI-Hardware oder ein Software-Synthesizer in hörbare Wave-Daten umwandeln muss.

DirectMusic kann nicht nur MIDI-Dateien abspielen, sondern Musik auch während der Laufzeit eines Programms komponieren, zumindest generieren. Das Verfahren basiert nicht auf einem Klang variierenden Algorithmus, sondern auf Elementen, die ein Musiker vorgibt. Dabei unterstützt ihn der *DirectMusic*-Producer, eine weitere Komponente von DirectX. *DirectMusic* greift auf Akkordfolgen, vorgegebene Musikstile und Klangmuster zurück, um dynamisch auf Ereignisse reagieren zu können. Damit beleben Sie ein Computerspiel mit einer satten Klangkulisse.



MIT DEM BEISPIELPROGRAMM drehen Sie an den Parametern, um Sound aus allen Richtungen zu hören.

Direct Sound

DirectX versucht, mit wenig Rechenzeit auszukommen. Die Komponente *DirectSound* nutzt dazu die Sound-Hardware, ohne dass Sie Details der Hardware kennen müssen. Programme, die Sie mit *DirectSound* ausstatten, klingen schon auf einfachen Soundkarten gut. Wenn eine Soundkarte mit ihren Treibern zusätzliche Effekte bietet, nutzt *DirectSound* diese ebenfalls. *DirectSound* kann sogar Effekte ansteuern, die beim Erscheinen einer derzeitigen *DirectSound*-Version noch nicht bekannt sind.

Ohne *DirectSound* einzusetzen, kann Windows mit seinen herkömmlichen Funktionen Wave-Dateien erklingen lassen. Dazu spielen *PlaySound* und *WaveOut* Sound oder Audiostream ab.

DirectSound sprechen Sie über das *IDirectSound*-Interface an. Sound Buffers und 3D-Sound-Effekte können Sie auch mit den Interfaces *IDirectSoundBuffer*, *IDirectSound3DBuffer* und *IDirectSound3DListener* verändern.

Mit dem *IDirectSoundNotify*-Interface versenden Sie Signale. Hat der Sound Buffer eine Playback-Position erreicht, können Sie automatisch ein Signal verschicken lassen.

So steuern Sie *DirectSound* an: Suchen Sie ein geeignetes Ausgabegerät, oder verwenden Sie das vom Benutzer gewählte Standardausgabegerät. Wenn Sie ein spezielles Gerät suchen oder mehrere Geräte verwenden wollen, müssen Sie – wie immer bei DirectX – die Funktion *EnumerationCallback* ansteuern:

```
// Callback Funktion
BOOL CALLBACK DSEnumProc
    (LPGUID lpGUID,
    LPCTSTR lpszDesc,
    LPCTSTR lpszDrvName,
    LPVOID lpContext )
{
// Wertvorgabe in lpContext

     LPGUID lpTemp = NULL;

     if ( lpGUID != NULL )
     {
// Gerät gefunden Info speichern
   lpTemp = malloc(sizeof(GUID));
   memcpy
   (lpTemp, lpGUID, sizeof(GUID));
     }

     return TRUE;
}
```

Um mehrere Geräte aufzuzählen, starten Sie mit:

```
if FAILED(DirectSoundEnumerate
((LPDSENUMCALLBACK)DSEnumProc,
          lpContext))
{
     // Fehler
}
```

Nachdem Sie sich für ein *DirectSound*-Gerät entschieden haben, initialisieren Sie es. Die Funktion

```
HRESULT WINAPI
 DirectSoundCreate
 (LPCGUID lpcGuid,
 LPDIRECTSOUND * ppDS,
 LPUNKNOWN  pUnkOuter);
```

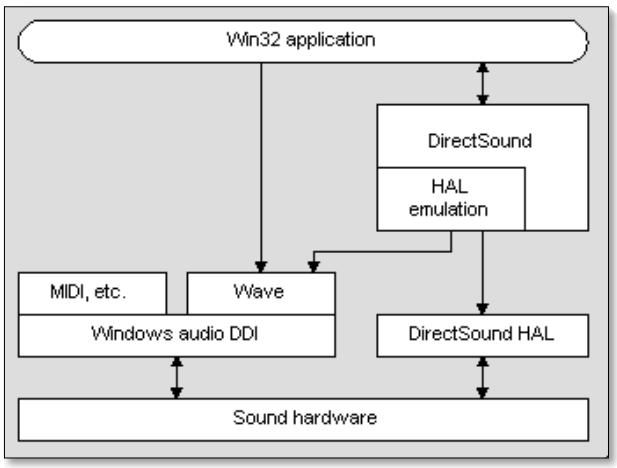
erzeugt und initialisiert das *IDirectSound*-Interface. Wenn Sie statt des Zeigers auf einen *Globally Unique Identifier* (GUID) einen Nullpointer angeben, wählen Sie das Standardausgabegerät. Dieses hat der Benutzer in der Systemsteuerung eingetragen. Im zweiten Para-

AUF CD

Die Quelltexte sowie die fertig übersetzten Routinen finden Sie auf unserer Heft-CD im Verzeichnis *Praxis/PC Underground*.

meter übergeben Sie die Adresse eines Zeigers auf ein *DirectSound*-Objekt, das Sie mit dem Funktionsaufruf erzeugen wollen. Der dritte Parameter muss *0* sein.

Nach der Initialisierung müssen Sie – wie stets bei DirectX-Geräten – den Kooperationslevel festlegen. Bei Spielen wollen Sie sicher sein, dass kein anderes Programm Soundeffekte im Hintergrund abspielt. Wählen Sie den *DSSCL_EXCLUSIVE*-Level. Damit kann nur die Anwendung das Direct-Sound-Gerät verwenden, die gerade aktiv im Vordergrund ist. Mit *DSSCL_PRIORITY* können Sie auch das Soundformat festlegen.



DIESE ARCHITEKTUR verbindet die Soundhardware mit einer Win32-Applikation.

Damit steht das Grundgerüst für unser Beispielprogramm. Den Kooperationslevel setzen Sie mit der Funktion

```
HRESULT SetCooperativeLevel
( HWND hwnd,  DSSCL_PRIORITY );
```

DirectSound legt von Haus aus einen primären Buffer an, in dem die Sounddaten aufgehoben und an die Soundkarte geschickt werden. Legen Sie dessen Format fest. Geben Sie folgende Codezeilen ein, um auf den Buffer zugreifen zu können:

```
LPDIRECTSOUND _ds;

if( FAILED(::DirectSoundCreate
( NULL, &_ds, NULL )))
    return false;
if( FAILED(_ds->
 SetCooperativeLevel(hwnd,
  DSSCL_PRIORITY)))
   return false;
if ( FAILED( hr = _ds->
CreateSoundBuffer( &dsbd,
&pDSBPrimary, NULL)))return hr;

// Primärer Buffer
WAVEFORMATEX wfx;
ZeroMemory( &wfx, sizeof
 (WAVEFORMATEX) );
wfx.wFormatTag  =
 WAVE_FORMAT_PCM;
wfx.nChannels = 2;
wfx.nSamplesPerSec  = 22050;
wfx.wBitsPerSample  = 16;
wfx.nBlockAlign     =
 wfx.wBitsPerSample
/8*wfx.nChannels;
wfx.nAvgBytesPerSec =
wfx.nSamplesPerSec *
 wfx.nBlockAlign;
if( FAILED( hr = pDSBPrimary->
SetFormat(&wfx)))return hr;
```

Danach geben Sie den primären Buffer wieder frei:

```
SAFE_RELEASE( primary );
```

Im nächsten Schritt fordern Sie einen *DirectSound*-Buffer an, in dem Sie Ihre Wave-Daten speichern und ausgeben. Ein solcher sekundärer Sound Buffer nimmt entweder einen einzelnen Soundeffekt auf, oder sein Inhalt ändert sich laufend. Sie können den Klang in einer Endlosschleife hören oder nur einmal abspielen.

Mit Buffers können Sie Daten auch stückweise einlesen (streamen). Das erledigt unser Beispielcode, der eine MP3-Datei über DirectMedia in *DirectSound* umleitet. Die Sounds von mehreren sekundären Sound Buffers können Sie mischen, indem Sie mehrere gleichzeitig abspielen. Sie können je nach Rechenzeit beliebig viele Soundeffekte gleichzeitig abspielen. Die Latenzzeit von *DirectSound* beginnt mit dem Zeitpunkt, zu dem das Abspielen startet, und endet, wenn Sie etwas hören (etwa 20 Millisekunden). Die Verzögerung ist so kurz, dass Sie in einem Spiel einen Soundeffekt starten und gleichzeitig die zugehörige Animation auf dem Bildschirm beginnen können. Sie wird nur größer (bis etwa 100 oder 150 Millisekunden), wenn *DirectSound* spezielle Hardware-Features mit Software-Routinen emulieren muss.

Zu diesem Zweck füllen Sie die *DSBUFFERDSC*-Struktur mit Informationen über den gewünschten *DirectSound*-Buffer aus und lassen sich einen Buffer erzeugen. Als Resultat erhalten Sie einen Zeiger auf das *IDirectSoundBuffer*-Interface, mit dem Sie den Buffer modifizieren und abspielen können:

```
 PCMWAVEFORMAT pcmwf;
DSBUFFERDESC dsbdesc;
HRESULT hr;
// Wave Format Struktur
memset(&pcmwf, 0, sizeof
     (PCMWAVEFORMAT));
pcmwf.wf.wFormatTag =
     WAVE_FORMAT_PCM;
pcmwf.wf.nChannels = 2;
pcmwf.wf.nSamplesPerSec = 22050;
pcmwf.wf.nBlockAlign = 4;
pcmwf.wf.nAvgBytesPerSec =
    pcmwf.wf.nSamplesPerSec
    * pcmwf.wf.nBlockAlign;
pcmwf.wBitsPerSample = 16;
// DSBUFFERDESC Struktur füllen
memset
(&dsbdesc, 0, sizeof(DSBUFFERDE-
SC));
dsbdesc.dwSize =
sizeof(DSBUFFERDESC);
//Default: Pan,Volume,Frequency
dsbdesc.dwFlags=DSBCAPS_CTRLPAN
| DSBCAPS_CTRLVOLUME |
DSBCAPS_CTRLFREQUENCY;
// 3 Sekunden langer Buffer
dsbdesc.dwBufferBytes =
3 * pcmwf.wf.nAvgBytesPerSec;
dsbdesc.lpwfxFormat =
(LPWAVEFORMATEX)&pcmwf;
// und Buffer erzeugen
hr = _ds->lpVtbl->
CreateSoundBuffer(lpDirectSound,
    &dsbdesc, _dsbuf, NULL);
    if SUCCEEDED(hr)
    { // Interface ist *_dsbuf
return TRUE; }
    else { // Fehler !
*_dsbuf= NULL; return FALSE;}
```

Nun haben Sie alle Strukturen angelegt und ausgefüllt (allokiert), um Sounds abzuspielen. Sie sind damit in der Lage, die Wave-Daten in den sekundären Buffer zu schreiben. Um auf den Buffer zuzugreifen und anderen Tasks den Zugriff zu verweigern, wenden Sie einen Lock-Befehl auf den Buffer an:

```
VOID *mem1, *mem2;
  DWORD sz1, sz2;
  if( FAILED( _dsbuf->Lock
(0,0,&mem1, &sz1, &mem2,
  &sz2, DSBLOCK_ENTIREBUFFER)))
   return false;
```

Diese Zeilen greifen ab der aktuellen Zeigerposition auf den Buffer zu. Als Rückgabe bekommen Sie zwei Zeiger (*mem1* und *mem2*) und zu jedem die Anzahl der Bytes über die Größe des korrespondierenden Speicherbereichs.

Warum bekommen Sie zwei Speicherbereiche bei diesem einen Buffer? Wenn Sie den Buffer *loopen*, also endlos abspielen, springt er jedesmal zum Anfang zurück. *DirectSound* erledigt das für Sie: Sie bekommen einen ersten Speicherbereich für den Teil des Sounds, der von der aktuellen Position bis zum Bufferende reicht, und einen eventuellen zweiten, der vom Anfang des Buffers bis zur aktuellen Position geht.

In den Speicher können Sie mit dem Format (8, 16 oder 32 Bit) hineinschreiben, das Sie für den Buffer festgelegt haben. Danach geben Sie den Buffer wieder frei:

```
_dsbuf->Unlock
 (mem1, sz1, mem2, sz2);
```

Nun können Sie den Buffer abspielen. Dazu verwenden die Interfaces die Methode *::Play()*:

```
HRESULT IDirectSoundBuffer::Play(
  DWORD dwReserved1,
  DWORD dwPriority,
  DWORD dwFlags
);
```

Der vorgegebene erste Parameter muss *0* sein, der zweite gibt die Prioriät an, wie Sie den Soundeffekt abspielen wollen. Dabei ist *0* die niedrigste und *0xffffffff* die höchste Priorität. Sie ist wichtig für den Direct-Sound-Voice-Manager, der bei hoher Systemauslastung Sounds höchster Priorität bevorzugt.

Die Flags wählen Sie in der Hilfefunktion des DirectX-SDK. Mit *DSBPLAY_LOOPING* lassen Sie Klänge aus dem Buffer endlos abspielen. Andere Flags verwenden Sie, um 3D-Sounds anzusteuern.

Mit *DirectSound* entlocken Sie Spielen, Demos und Multimedia-Anwendungen die richtigen Töne. Für einfachere Applikationen reichen *Play Sound*- und *WaveOut*-Befehle.

3D-Sound mit *DirectSound*

Wer eine entsprechende Soundkarte mit Verstärker und Lautsprechern besitzt, kann mit *DirectSound* Klänge im Raum frei positionieren: Der 3D-Sound platziert den Hörer (*Listener*) und die Sound-Effekte (Sound-Buffer) beliebig in den Raum. Für den Hörer können Sie weitere Eigenschaften, wie die Stärke des Dopplereffektes, festlegen. Für die Schallquellen können Sie die Abstrahlrichtung bestimmen.

Wenn Sie 3D-Sound verwenden wollen, legen Sie dies mit dem Format des primären Buffers mit *DSBCAPS_CTRL3D* fest:

```
ZeroMemory (&dsbdesc,
sizeof(DSBUFFERDESC));
dsbdesc.dwSize =
   sizeof(DSBUFFERDESC);
dsbdesc.dwFlags=DSBCAPS_CTRL3D |
        DSBCAPS_PRIMARYBUFFER;
if( FAILED( hr = _ds->
CreateSoundBuffer(
&dsbdesc, &pDSBPrimary, NULL)))
    return hr;
```

Greifen Sie auf das Interface zu, mit dem Sie die Attribute für den Hörer festlegen. Dazu rufen Sie die *QueryInterface*-Methode vom Objekt des primären Buffers auf:

```
// Listener
LPDIRECTSOUND3DLISTENER
 pDSListener = NULL;
if( FAILED (hr=pDSBPrimary->
QueryInterface(
IID_IDirectSound3DListener,
(VOID**)&pDSListener ) ) )
      return hr;
```

Das Listener-Interface stellt Ihnen die Methoden vor, mit denen Sie die entsprechenden Attribute setzen können. Diese Attribute sind zum Beispiel Faktoren wie Doppler-, Rolloff- oder Distanzeffekt, der Lautstärkenänderungen für verschiedene Entfernungen beschreibt, sowie Position, Geschwindigkeit und Blickrichtung der Hörers. Die Ortsangaben bestimmen Sie mit 3D-Koordinaten. Hier die Positionierung eines Listeners:

```
pDSListener->SetPosition
(0,0,1,DS3D_IMMEDIATE);
pDSListener->SetVelocity
(0,1,0,DS3D_IMMEDIATE);
pDSListener->SetOrientation
(0,1,0, 0,0,1, DS3D_IMMEDIATE);
```

Mit dem *DS3D_IMMEDIATE*-Flag legen Sie fest, dass die Zustandsänderungen sofort übernommen werden. Das kostet Rechenzeit. Wollen Sie mehrere Änderungen vornehmen, aber keine Rechenzeit vergeben, verwenden Sie das *DS3D_DEFERRED*-Flag und rufen am Ende aller Änderungen

```
HRESULT
IDirectSound3DListener::
CommitDeferredSettings()
```

auf, um sie wirksam werden zu lassen.

Nun fehlt Ihnen noch ein sekundärer Sound Buffer, der 3D-Sound unterstützt. Dazu legen Sie *DirectSound*-Buffer an und setzen wie beim primären Buffer das *DSBCAPS_CTRL3D*-Flag. Über einen Query-Interface-Aufruf bekommen Sie wieder ein *DirectSound 3DBuffer*-Objekt:

```
// sekundärer Buffer
LPDIRECTSOUNDBUFFER pDSBuffer
= NULL;
// 3D-Sound Buffer
LPDIRECTSOUND3DBUFFER
g_pDS3DBuffer = NULL;
if( FAILED(hr = _ds->
CreateSoundBuffer
(&dsbd, &pDSBuffer, NULL)))
return hr;

// Get the 3D buffer from the 2.
if( FAILED( hr = pDSBuffer->
QueryInterface(
IID_IDirect
Sound3DBuffer,
(VOID**)&pDS3D
Buffer ) ) )
        return hr;
```

Das *IDirectSound3DBuffer*- stellt Ihnen wie das Listener-Interface Methoden zur Verfügung, die Schallquelle zu positionieren oder die Abstrahlrichtung festzulegen. Sie können den Schall (Buffer) auch automatisch immer in gleicher Position zum Hörer platzieren. *DirectSound* interpretiert die Ortsangaben für den Buffer dann als relative Koordinaten. Dazu nehmen Sie folgenden Codeausschnitt:

```
// 3D Buffer Attribute
 DS3DBUFFER dsBufferParams;
dsBufferParams.dwSize =
sizeof(DS3DBUFFER);
pDS3DBuffer->GetAllParameters
( &dsBufferParams );
dsBufferParams.dwMode =
DS3DMODE_HEADRELATIVE;
pDS3DBuffer->SetAllParameters(
&dsBufferParams,
DS3D_IMMEDIATE );
```

Den Buffer können Sie wie die sekundären Buffer mit der *Play*-Methode abspielen, ohne sich weiter um den 3D-Sound kümmern zu müssen. *DirectSound* übernimmt die Arbeit für Sie. Ein Beispiel finden Sie im DirectX7-SDK, das Sie mit den Parametern experimentieren lässt.

Streaming von MP3 aus Direct Media

Unser Beispielprogramm auf der Heft-CD nutzt die Vorteile, die das DirectX-DirectMedia-Bundle bietet: Es bringt einen DirectMedia-AudioStream (*wav, mp3, wma*) dazu, seine *RAW*-Daten DirectSound zu liefern.

Diese können Sie in einen *DirectSound*-Buffer kopieren und damit alles anstellen, was das *DirectSound*-Buffer-Interface bietet. Da Sie viele Klassen aus dem *DirectSound*- und *DirectMedia*-SDK einsetzen, sollten Sie häufiger einen Blick in die Hilfedateien von DirectX-7 werfen. Die DirectMedia-Hierarchie zeigt die Verbindung von Win32-Applikationen mit der Sound-Hardware.

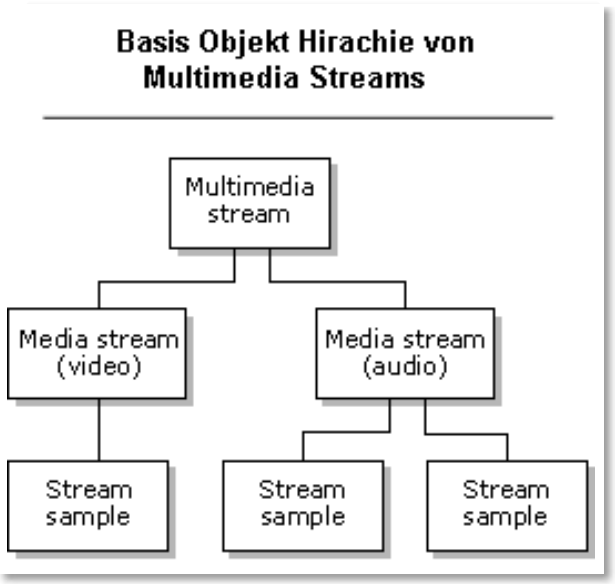
Um auf die DirectMedia-COM-Objekte zuzugreifen, aktivieren Sie die COM-Schnittstelle für die Applikation:

```
CoInitialize(NULL);
```

Anschließend benötigen Sie einen *IMultiMediaStream*, der in der Klassenhierarchie von DirectMedia den höchsten Level eines Streaming-Objekts darstellt. Dieses kann mehrere MediaStreams aufnehmen wie Audio und Video:

```
IMultiMediaStream
*pMMStream;
```

Als Nächstes brauchen Sie einen *IAMMultiMediaStream*, der die Schnittstelle zu den Stream-Funktionen darstellt. Der Vorteil dieses Interfaces: Es generiert die DirectMedia-Filter selbst. Diese benötigen Sie, um die Dateien abzuspielen oder anzuzeigen.



SIE SEHEN IN DIESER HIERARCHIE den Aufbau von Multimedia-Streams.

```
IAMMultiMediaStream *pAMStream;
```

Den DirectMedia Filter bekommen Sie auch wieder über die COM-Schnittstelle via *CoCreateInstance*. Dieses liefert Ihnen ein Object des *CLSID*-Eintrags.

```
CoCreateInstance
(CLSID_AMMultiMediaStream, NULL,
CLSCTX_INPROC_SERVER,
IID_IAMMultiMediaStream,
(void **)&pAMStream);
```

Initialisieren Sie den DirectMedia-Stream. Lassen Sie den Lesezugriff (*STREAMTYPE_READ*) und den zum Dateityp gehörigen Filter automatisch abrufen:

```
pAMStream->Initialize
(STREAMTYPE_READ,
AMMSF_NOGRAPHTHREAD, NULL);
```

Teilen Sie DirectMedia über die *AddMediaStream*-Methode mit, was Sie mit dem Stream vorhaben. Diese bekommt einen Pointer zu einem Media-Stream-Objekt als ersten Parameter. Um den Default-Renderer (die Standardfunktion, um die Daten aus dem Stream zu extrahieren) nutzen zu können, tragen Sie für den ersten Parameter den Wert *0* ein. Zudem brauchen Sie die *PurposeID* (MSPID) *Audio*. Den optionalen Parameter für *New Stream* lassen Sie auf dem Wert *0*.

```
pAMStream->AddMediaStream
(NULL, &MSPID_PrimaryAudio,
AMMSF_ADDDEFAULTRENDERER, NULL);
```

Nach den Einstellarbeiten für den MediaStream öffnen Sie die Klangdatei.

```
pAMStream->OpenFile
( pszFileName, AMMSF_RUN);
```

Übergeben Sie den COM-Objekten Unicode-Strings. Dazu wandeln Sie den eingehenden Dateinamen über *MultiByteToWideChar* in dieses Format um.

```
WCHAR
wszName[_MAX_PATH];
MultiByteToWideChar
(CP_ACP, 0,
argv[1], -1,
wszName,
sizeof(wszName) /
sizeof(wszNa-
me[0]));
```

Probieren Sie die Audiofunktionen des MediaStream aus. Dazu machen Sie aus dem *IAMMultiMediaStream* wieder einen *IMultiMediaStream*.

```
pMMStream = pAM-
Stream;
```

Jetzt können Sie den fertig initialisierten MediaStream bei DirectMedia abholen.

```
IMediaStream *pStream;
pMMStream->GetMediaStream
(MSPID_PrimaryAudio, &pStream);
```

Da der Multimedia-Stream auch ein *AVI*- oder *MPEG*-Video sein kann, fordern Sie nur dessen Audioteil an.

```
IAudioMediaStream
*pAudioStream;
pStream->QueryInterface
(IID_IAudioMediaStream,
(void **)&pAudioStream);
```

Den Audiotrack fragen Sie nach seinem Format und erhalten eine fertig ausgefüllte *WAVEFORMATX*-Struktur:

```
WAVEFORMATEX wfx;
pAudioStream->GetFormat(&wfx);
```

Jetzt fehlt Ihnen nur noch der Zugriff auf die Audiodaten. Diesen erlangen Sie wieder über die COM-Schnittstelle:

```
IAudioData *pAudioData;
CoCreateInstance
(CLSID_AMAudioData, NULL,
CLSCTX_INPROC_SERVER,
IID_IAudioData,
(void **)&pAudioData);
```

Somit haben Sie ein *IAudioData*-Interface, das von einem *IMemoryInterface* abgeleitet wurde. Dieses ist in der Lage, Daten in einem von Ihnen ansprechbaren Speicher abzulegen.

```
#define DATA_SIZE 8192
PBYTE pMyBuffer =
new PBYTE[DATA_SIZE];
pAudioData->SetBuffer
(DATA_SIZE, pMyBuffer, 0);
pAudioData->SetFormat(&wfx);
```

Teilen Sie Ihrem Audio-Stream per *CreateSample*-Methode mit, wo und wie Sie die entpackten Daten abholen wollen. Der erste Übergabeparameter ist Ihr erzeugtes Audiodata-Objekt, in dem die Samples abgelegt werden. Die Flags sind in der jetzigen Version nicht spezifiziert, deshalb setzen Sie sie auf den Wert *0*. Der dritte Parameter ist ein *IAudioStream Object*, über das Sie später auf die Update-Funktion zugreifen. Diese signalisiert, dass Daten abzuholen sind.

```
IAudioStreamSample *pSample;
pAudioStream->CreateSample
(pAudioData, 0, &pSample);
```

Nun brauchen Sie noch ein Event, über das das Audio-Interface Ihrem Programm mitteilen kann, dass neue Daten vorliegen. Sie können dann die Update-Funktion benutzen, die das *IAudioStreamSample* vom *IMemoryData*-Interface geerbt hat.

```
HANDLE hEvent =
CreateEvent(FALSE, NULL,
NULL, FALSE);
loop:
HRESULT hr = pSample->Update
(0, hEvent, NULL, 0);
if( FAILED(hr) ||
MS_S_ENDOFSTREAM == hr ) {
  break;
}
```

Wenn Ihr Event eintritt, können Sie die aktuellen Sound-Daten im Buffer abholen und an den *DirectSound*-Buffer schicken.

```
WaitForSingleObject
(hEvent, INFINITE);
DWORD dwLength;
pAudioData->GetInfo
(NULL, NULL, &dwLength);
ds_streambuffer.write
(dwLength, pBuffer );
goto loop
```

Die Vorteile dieser Methode für Streambuffer sind folgende: DirectMedia kümmert sich darum, die Daten aufzubereiten und zu entpacken. Außerdem extrahiert DirectMedia die Audiostreams aus allen ihm bekannten Videoformaten.

Ausblick auf Version 8

Unser Beispielprogramm und die des DirectX-SDKs bieten eine Grundlage für weitere Experimente und einen professionellen Einsatz von *DirectSound*. Wenn Sie sich für *DirectMusic* interessieren, sollten Sie einen Blick in den *DirectMusic*-Producer riskieren, den Sie im SDK etwas versteckt im Verzeichnis *essentls\dmusprod* finden. Die noch nicht veröffentlichte Version 8 von DirectX wird *DirectSound* und *DirectMusic* verschmelzen. Die dann entstehende Komponente soll *DirectAudio* heißen. ET

Die Quelltexte sowie die fertig übersetzten Routinen finden Sie auf unserer Website unter

www.pc-magazin.de/magazin/extras.htm

Klicken Sie unter *Online Extras* im Menü *Praxis* auf das entsprechende *Download*-Feld.