

5 Sound and Music

5.1 Aims

To understand the full potential of DirectX Audio, you need to know lots about Digital Signal Processing, music, MIDI and loads of other stuff. On the other hand, you can use DirectX Audio to do some basic game stuff. In this Module, we'll look at the use of DirectSound to do simple sound effects in your game. We'll also explore the use of DirectMusic to play MIDI files for some mood music during your game.

5.2 DirectSound Overview

DirectSound supports audio devices in the same way that DirectDraw supports graphics devices; it gives you raw, quick access to all kinds of hardware in a uniform fashion.

5.2.1 Get the Interface

Use `DirectSoundCreate()` to get the `IDirectSound` interface. This is getting familiar by now, surely ...

5.2.2 Set the Cooperative Level

Use `SetCooperativeLevel()` in a similar fashion to DirectDraw to let Windows know how much control (if any) you are leaving for it (I'm having *de je vous* here). In general, you'll want to use the normal level. Use priority level if you want change the wave playing format. Use exclusive level, if you don't want other applications to be able to play sounds while your playing.

5.2.3 Primary and Secondary Buffers

The primary buffer is the buffer that actually plays the current sound. This is where your sound has to go to get played. The secondary buffers are used to hold the sounds in RAM until you're ready to play them. The analogy between primary and secondary surfaces in DirectDraw is obvious.

Buffers are created using `CreateSoundBuffer()`. When you create a buffer you need to specify its size (typically the size of the sound you are storing), and set which flags you require for buffer operations such as panning, frequency shifting and volume control. When setting the flags, you need to be aware of possible CPU

overhead, if the sound card doesn't support panning, frequency shifting and volume control in hardware. When you're finished with a buffer, don't forget to `Release()` it.

5.2.4 Writing Data to the Buffer

Calling `Lock()` gives you one, or possibly two pointers, to the buffer. Sound buffers are circular. If your lock doesn't wrap around the circle, you will get one pointer pointing to the appropriate bit of the buffer. If you wrap around the circle, you will get a pointer to the data, and a pointer to the start of the buffer. These pointers are only for writing, reading can produce unpredictable results.

5.2.5 Rendering the Buffer

Once you have stored the data in the buffer, you can play it using `Play()`. The `Play()` method takes the sound in the secondary buffer and mixes it into primary buffer and plays it. If this is the first buffer to play, it will implicitly create a primary buffer and start playing that buffer; the application need not explicitly direct the primary buffer to play. You can stop it using `Stop()`, and fiddle the volume, frequency and stereo planning if you've got the hardware (or can spare the CPU).

5.2.6 Reading Sounds from Disk

The DirectSound API does not include methods for handling wave files. The text describes his loader and his other utility functions. You should also check out the `Dsutil.cpp` file used by many of the SDK sample applications. Look under DirectX Audio in the SDK DirectX C++ Help file.



Use DirectSound to play some tennis noises when you hit the ball back and forth in your tennis game.

5.3 Overview of DirectMusic

It's reasonably straightforward to get DirectMusic to play some music while your game is playing. DirectMusic is pure COM, so there are no easy APIs to get you started. This also means you don't need to import any lib files when you link it.

5.3.1 Initialise COM

The first thing you must do before any other direct COM calls is initialise COM using `CoInitialize()`.

5.3.2 Creating a Performance Interface

To use DirectMusic, you will need an `IDirectMusicPerformance` interface. In the process, you'll create a `IDirectMusic` interface that you don't need to use. The interface is created using the COM method `CoCreateInstance()`, as we did in the COM module. Once you have the interface it must be initialised using a call to `Init()`. If you are using DirectSound and DirectMusic together, you must create the `IDirectSound` object first, and pass that in the `Init()` call.

5.3.3 Add a Port

The standard software synthesiser can be loaded simply enough by calling `AddPort(NULL)` from the `IDirectMusicPerformance` interface.

5.3.4 Load a MIDI file

Unlike WAV files, there is a `DirectMusic` loader for MIDI files. You can create a `IDirectMusicLoader` interface using the COM method `CoCreateInstance()`. The process of loading MIDI files into segments is still a little tricky so check the book for the details.

5.3.5 Play the MIDI file

Once you have loaded your MIDI files into the segments, you can play it using `PlaySegment()`, stop it using `Stop()`, and when your finished release it using `Release()`. Make sure you unload any loaded instruments before releasing the segment.



Using the code from DEMO10_4, add a little background music to your tennis game.