Direct Sound

305890
Spring 2010
5/28/2010
Kyoung Shin Park
kpark@dankook.ac.kr

DirectSound

- DirectSound
  - Links d3d9.lib, d3dx9.lib, winmm.lib, dsound.lib, dxerr9.lib, dxguid.lib

DirectSound

- DirectSound
  - Provides a single Application Programming Interface (API) for the playback of sounds and music
- How does DirectSound work?
  - DirectSound manages the sound data through the use of ***buffers***
  - Possible to have multiple buffers that hold any sound data
  - The buffers can be manipulated or played by DirectSound or even mix them up to construct a new buffered data
- Sound buffers
  - The areas that hold the sound data
  - E.g., if a **WAV file** is loaded to a sound buffer, the sound data within the file would be placed into a sound buffer, which can be manipulated or played

Direct Sound

- Types of buffers DirectSound provides
  - **Primary buffer**
    - All sounds played are mixed into the primary buffer, which will be played by the sound card
  - **Secondary buffer**
    - The buffer that holds all the sound data that our application needs
    - DirectSound can play multiple sounds by accessing more than one secondary buffer simultaneously
  - Static buffer
    - When sound data is of limited size, use the static buffer
    - This buffer allows for the complete loading of a particular sound into memory
  - Streaming buffer
    - When sound data may be too large to fit into memory at one time
    - Allows for only a portion of a sound to be loaded into it before being send of to be played
    - As the sound within the streaming buffer is played, new sound data is loaded

Direct Sound Interface

- IDirectSound8
- IDirectSoundBuffer8
- IDirectSound3DBuffer8
- IDirectSound3DListener8
- IDirectSoundCapture8
- IDirectSoundCaptureBuffer8
- IDirectSoundNotify8
- IKsPropertySet8

DirectSound Setup

- DirectSound Setup
  1. Create a DirectSound device using DirectSoundCreate8()
  2. Set the cooperative level using IDirectSound8::SetCooperativeLevel()
  3. Create a secondary buffer using IDirectSound8::CreateSoundBuffer()
  4. Read the sound data into the secondary buffer
  5. Play/Pause/Stop sound in a buffer
  6. Release all the instances after uses

Using DirectSound

- DirectSound
  - Must be initialized to be used
  - The first step is to use the DirectSound device, which is represented by the IDirectSound8 interface
- DirectSound device
  - Represents an interface to a specific piece of sound hardware with a computer.
  - To make DirectSound work, select a sound card and create a DirectSound device.
  - Create a DirectSound device using DirectSoundCreate8

Using DirectSound

```
HRESULT DirectSoundCreate8(LPCGUID lpcGuidDevice,
        LPDIRECTSOUND8 *ppDS8, // LPDIRECTSOUND8 pointer
        LPUNKNOWN pUnkOuter); // always NULL
```

- lpcGuidDevice
  - The GUID that represents the sound device to use
  - Use either DSDEVID_DefaultPlayback or NULL
  - Use NULL to use the default sound device
- ppDS8
  - The address to the variable that will hold the newly created DirectSound device
- pUnkOuter
  - The controlling object's IUnknown interface, should be NULL

Using DirectSound

```
// variable that will hold the return code
HRESULT hr;

// variable that will hold the created DirectSound device
LPDIRECTSOUND8 m_pDS = NULL;

// Attempt to create the DirectSound device
hr = DirectSoundCreate8(NULL, &m_pDS, NULL);

// Check the return value to confirm that a valid device was created
if (FAILED(hr)) return false;
```

Setting the Cooperative Level

- Because DirectSound provides an access to a hardware device, it needs to have a cooperative level set.
- In DirectSound, it's not possible to gain exclusive access to the sound device
- But it's possible to ask OS to set the highest priority to our application, but other applications can still trigger sounds to be played

Setting the Cooperative Level

- 4 Cooperative Levels
  - DSSCL_NORMAL
    - This level works best with other applications that still allow other events
    - Cannot change the format of the primary buffer because the device is shared with other applications
  - **DSSCL_PRIORITY**
    - If an application requires more control over the primary buffer and your sounds, you should use this cooperative level
    - Most games should use this level
  - DSSCL_EXCLUSIVE
    - Exclusive access to sound device
  - DSSCL_WRITEPRIMARY
    - This level gives an application write access to the primary buffer

Setting the Cooperative Level

- The cooperative level is set using SetCooperativeLevel()

```
HRESULT SetCooperativeLevel(HWND hWnd, DWORD dwLevel);
```

- hWnd
  - The handle of the application window requesting the change in cooperative level
- dwLevel
  - The cooperative level

Setting the Cooperative Level

```
HRESULT hr;

// Create the DirectSound device
LPDIRECTSOUND8       g_pDS = NULL;
hr = DirectSoundCreate8(NULL, &g_pDS, NULL);

// Set the DirectSound cooperative level
hr = g_pDS->SetCooperativeLevel(hWnd, DSSCL_PRIORITY);

if (FAILED(hr)) return false;
```

Sound Files

- Should **load sound data** within DirectSound **into secondary buffers** before using it
- Sound data should be loaded into either a *static* or *streaming buffer*
- Static buffer
  - A fixed-length buffer that has full sound loaded into it
- Streaming buffer
  - Needed when the sound being loaded is larger than what the buffer can accommodate
  - A small buffer is used
  - Parts of the sound data are continuously loaded in and played

The Secondary Buffer

- Steps to play a sound data
  - DirectSound uses buffers to store the audio data that it needs
  - Should create a secondary buffer storing the audio data to play a sound
  - After the buffer is created, the sound is loaded into it fully (or partially for a streaming sound)
  - Then, play the sound
- DirectSound allows for **any number of secondary buffers to be played simultaneously all being mixed into the primary buffer.**
- Before creating a secondary buffer, needs to know the format of the sound that will reside in it
- DirectSound requires that the buffers you create are of the same format as the sound within them
  - E.g., if a 16-bit WAV file with two channels of sound, the secondary buffer must be created by this format

WAVEFORMATEX structure

- The formats of the buffers in DirectSound are described using the **WAVEFORMATEX** structure
  - Create a standard WAVEFORMATEX structure if the format of the WAV file data is known
  - If the file format is not known, create this structure and fill it in after opening the audio file

```
typedef struct {
    WORD wFormatTag;
    WORD nChannels;
    DWORD nSamplesPerSec;
    DWORD nAvgBytesPerSec;
    WORD nBlockAlign;
    WORD wBitsPerSample;
    WORD cbSize;
} WAVEFORMATEX;
```

WAVEFORMATEX structure

- wFormatTag
  - The type of waveform audio
  - For one- or two-channel PCM data, this value should be WAVE_FORMAT_PCM
- nChannels
  - The number of channels needed (1: MONO, 2: STEREO)
- nSamplesPerSec
  - Sampling rate (Mhz). 8.0 kHz, 11.025 kHz, 22.05 kHz, 44.1 kHz
- nAvgBytesPerSec
  - Average data-transfer rate (in bytes per second)
- nBlockAlign
  - Alignment (in bytes).
  - nChannels * wBitsPerSample / 8
- wBitsPerSample
  - The number of bits per sample (either 8 or 16)
- cbSize
  - Extra number of bytes to append to this structure. Always 0.

The Secondary Buffer

- Needs a second structure to finish describing the secondary buffer to DirectSound: DSBUFFERDESC

```
typedef struct {
    DWORD dwSize;
    DWORD dwFlags;
    DWORD dwBufferBytes;
    DWORD dwReserved;
    DLPWAVEFORMATEX lpwfxFormat;
    GUID guid3DAlgorithm;
} DSBUFFERDESC, *LPDSBUFFERDESC;
```

DSBUFFERDESC structure

- dwSize
  - The size of the DSBUFFERDESC structure (in bytes)
- dwFlags
  - Set of flags specifying the capabilities of the buffer
- dwBufferBytes
  - The size of the new buffer (in bytes)
  - Number of bytes of sound data that this buffer can hold
- dwReserved
  - Must be 0, Not used
- lpwfxFormat
  - An address to a WAVEFORMATEX structure
- guid3DAlgorithm
  - GUID identifier to the two-speaker virtualization algorithm to use

DSBUFFERDESC structure

- dwFlags
  - DSBCAPS_CTRLALL: 버퍼는 모든 제어 기능을 가진다.
  - DSBCAPS_CTRLDEFAULT: 버퍼는 기본 제어 옵션을 가진다. 이 값은 DSBCAPS_CTRLVOLUME, DSBCAPS_CTRLFREQUENCY를 지정하는 것과 동일하지만, DirectX6.0이후부터 없어졌다.
  - DSBCAPS_CTRLFREQUENCY: 버퍼가 주파수 제어 기능을 가진다.
  - DSBCAPS_CTRLPAN: 버퍼가 팬 (pan) 기능을 가진다.
  - DSBCAPS_CTRLVOLUME: 버퍼가 볼륨제어 기능을 가진다.
  - DSBCAPS_STATIC: 버퍼가 정적 사운드 데이터에 사용될 것임을 알린다. 대부분 하드웨어 (사운드카드) 메모리에 생성한다.
  - DSBCAPS_LOCHARDWARE: 메모리가 사용가능 하다면 하드웨어 메모리에 사운드 버퍼를 생성하며 하드웨어 믹싱을 사용한다.
  - DSBCAPS_LOCSOFTWARE: 시스템 메모리(RAM)에 사운드 버퍼를 생성하며 소프트웨어 믹싱을 사용한다.
  - DSBCAPS_PRIMARYBUFFER: 주 사운드 버퍼로 생성한다. 이 플래그를 주지 않으면 기본값으로 보조 사운드 버퍼로 생성된다.

Creating a Secondary Buffer

- After creating the DSBUFFERDESC structure, create the secondary buffer using CreateSoundBuffer().

```
HRESULT CreateSoundBuffer(LPCDSBUFFERDESC pcDSBufferDesc,
                 LPDIRECTSOUNDBUFFER *ppDSBuffer,
                          LPUNKNOWN pUnkOuter);
```

- pcDSBufferDesc
  - Address to an already-defined DSBUFFERDESC structure
- ppDSBuffer
  - Address to the variable that will hold the newly created buffer
- pUnkOuter
  - Address to the controlling object's IUnknown interface
  - Should be NULL

Creating a Secondary Buffer

```
// Define a WAVEFORMATEX structure
WAVEFORMATEX wfx;

// Clear the structure to all zeros
ZeroMemory(&wfx, sizeof(WAVEFORMATEX));

// Set the format to WAVE_FORMAT_PCM
wfx.wFormatTag = (WORD) WAVE_FORMAT_PCM;
wfx.nChannels = 2; // set channels by 2
wfx.nSamplesPerSec = 22050;
wfx.wBitsPerSample = 16;
wfx.nBlockAlign = (WORD) (wfx.wBitsPerSample / 8 * wfx.nChannels);
wfx.nAvgByPerSec = (DWORD) (wfx.nSamplesPerSec * wfxnBlockAlign);
```

Creating a Secondary Buffer

```
DSBUFFERDESC dsbd;
ZeroMemory(&dsbd, sizeof(DSBUFFERDESC));
dsbd.dwSize = sizeof(DSBUFFERDESC);
dsbd.dwFlags = 0;
dsbd.dwBufferBytes = 64000;
dsbd.guid3DAlgorithm = GUID_NULL;
dsbd.lpwfxFormat = &wfx;

LPDIRECTSOUNDBUFFER DSBuffer = NULL;
hr = g_pDS->CreateSoundBuffer(&dsbd, &DSBuffer, NULL);
if (FAILED(hr)) return NULL;
```

Locking the Sound Buffer

- Locking the sound buffer
  - Locking the sound buffer gives us a chance to manipulate and change the sound data within a buffer
  - After locking, sound data can be loaded into the buffer
  - Make sure to unlock the buffer after loading data

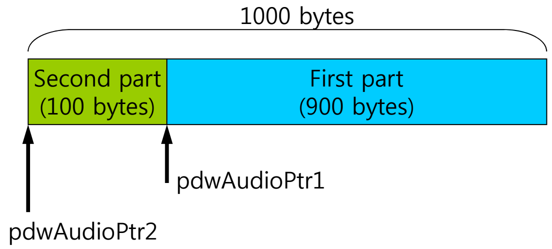
```
HRESULT Lock(
    DWORD dwOffset,
    DWORD dwBytes,
    LPVOID *ppvAudioPtr1,
    LPDWORD pdwAudioBytes1,
    LPVOID *ppvAudioPtr2,
    DPDWORD pdwAudioBytes2,
    DWORD dwFlags);
```

Locking the Sound Buffer

- dwOffset
  - Specifies where in the buffer the lock should begin
- dwBytes
  - The number of bytes within the buffer to lock (in bytes)
- ppAudioPtr1
  - Receives a pointer to the first part of the locked buffer
- pdwAudioBytes1
  - Receives the number of bytes in the block of bytes in the block pointer by ppvAudioPtr1 (in bytes)
- pdwAudioPtr2
  - Receives a pointer to the second part of the locked buffer
  - If filling the whole buffer with sound data, this must be NULL
- pdwAudioBytes2
  - Receives the number of bytes in the block pointed by ppvAudioPtr2 (in bytes)
  - Should be NULL if pdwAudioPtr2 is NULL.

Locking the Sound Buffer

- dwFlags
  - Specifies how the lock should occur
  - DSBLOCK_FROMWRITECURSOR: start the lock from the write cursor
  - DSBLOCK_ENTIREBUFFER: Lock the entire buffer. If this flag is set, the dwBytes variable is ignored



첫 번째 버퍼에 있는 사운드를 재생하는 동안에 두 번째 버퍼에는 나머지 사운드 부분을 저장한다. 그리고, 첫 번째 버퍼에 있는 사운드 재생이 끝나면 곧바로 두 번째 버퍼에 저장된 사운드를 재생하게 된다.
이렇게 해서, 사운드의 처리 속도를 빠르게 하고 사운드 재생이 끊기지 않고 반복될 수 있게 한다.

Unlocking the Sound Buffer

- Unlock the Sound Buffer
  - After loading sound data into the buffer, then unlock it

```
HRESULT Unlock(LPVOID pvAudioPtr1, DWORD dwAudioBytes1,
               LPVOID pvAudioPtr2, DWORD dwAudioBytes2);
```

- pvAudioPtr1
  - The address of the value from ppvAudioPtr1 used in Lock
- dwAudioBytes1
  - The number of bytes written to pvAudioPtr1 (in bytes)
- pvAudioPtr2
  - The address of the value from ppvAudioPtr2 used in Lock
- dwAudioBytes2
  - The number of bytes written to pvAudioPtr2 (in bytes)

Reading the Sound Data into the Buffer

- Loading sound data
  - Will use the sample file, dsutil.cpp, included in the DirectX Sound.
- Loading sound data process
  1. Create CWaveFile object
  2. Use Open() method of CWaveFile to gain access to the WAV file
  3. Create the secondary sound buffer to hold the WAV data
  4. Lock the buffer
  5. Read and copy sound data
  6. Unlock the buffer

Reading the Sound Data into the Buffer

1. Create a CWaveFile object

```
CWaveFile wavFile = new CWaveFile();
```

2. Use Open() of CWaveFild to gain access to the WAV file
   - The following example shows opening a file called test.wav for reading.
   - If the file doesn't have any data in it (size=0), then stop.

```
// open "test.wav"
wavFile->Open("test.wav", NULL, WAVEFILE_READ);
// Check to make sure that the size of data within the wave file is valid
if (wavFile->GetSize() == 0) return false;
```

3. Create the secondary sound buffer to hold the WAV data

Reading the Sound Data into the Buffer

4. Lock the buffer

```
HRESULT hr;
VOID *pDSLockedBuffer = NULL; // pointer to locked buffer memory
DWOR dwDSLockedBufferSize = 0; // size of the locked buffer
// Start the beginning of the buffer
hr = DSBuffer->Lock(0,
            // This assumes a buffer of 64000 bytes
            64000,
            // The variable holds a pointer to the start of the buffer
            &pDSLockedBuffer,
            // holds the size of the locked buffer
            &dwDSLockedBufferSize,
            NULL, // No secondary is needed
            NULL, // No secondary is needed
            DSBLOCK_ENTIREBUFFER); // Lock the entire buffer
if (FAILED(hr)) return NULL;
```

Reading the Sound Data into the Buffer

5. Read and copy sound data
   - Before reading data from the opened wave file, need to reset the WAV data to the beginning using ResetFile of CWaveFile
   - Then, read data using Read method

```
HRESULT hr; // variable to hold the return code
// the amount of data read from the wav file
DWORD dwWaveDataRead = 0;
 // reset the WAV file to the beginning
wavFile->ResetFile();
// read the WAV file
hr = wavFile->Read((BYTE *) pDSLockedBuffer,
                   dwDSLockedBufferSize, &dwWaveDataRead);
if (FAILED(hr)) return NULL;
```

Reading the Sound Data into the Buffer

6. Unlock the sound buffer

```
DSBuffer->Unlock(pDSLockedBuffer, dwDSLockedBufferSize, NULL,
NULL);
```

Playing Sound in a Buffer

- Playing sound in a buffer
  - After loading data into the DirectSoundBuffer, it is possible to play it using Play function

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

  - dwReserved1
    - Must be 0
  - dwPriority
    - The priority level to play the sound
    - Any value from 0 to 0xFFFFFFFF
    - Must set to 0 if the DSBCAPS_LOCDEFER flag was not set when the buffer was created.
  - dwFlags
    - Specifying the how the sound should be played, e.g. DSBPLAY_LOOPING

```
DSBuffer->Play(0, 0, DSBPLAY_LOOPING); // background loop sound
```

Stopping a Sound

- Stopping a sound

```
HRESULT Stop();
HRESULT hr;
hr = DSBuffer->Stop();
if (FAILED(hr)) return false;
```

Controling the Volume

- Changing the volume
  - Can adjust the volume of a sound through the buffer in which it resides
  - The volume must be in between DSBVOLUME_MIN (silence) and DSBVOLUME_MAX (original volume of the sound)

```
HRESULT SetVolume(LONG lVolume);
```

  - lVolume
    - Any value between 0 (DSBVOLUME_MAX) and -10000 (DSBVOLUME_MIN)

- Get the current volume at which a sound is playing

```
HRESULT GetVolume(LPLONG plVolume);
```

Panning the Sound

- Panning a sound between the left and right speakers
  - Lowering the volume of a sound in one speaker and increasing it in the opposite speaker
  - Sounds seem to move around

```
HRESULT SetPan(LONG lPan);
```

  - lPan
    - Takes any value between DSBPAN_LEFT and DSBPAN_RIGHT
    - DSBPAN_LEFT (-10000) increase the volume of sound in the left speaker to full while silencing the sound in the right speaker.
    - DSBPAN_RIGHT (10000) does the opposite.
    - DSBPAN_CENTER (0) defined as 0, resets both speakers to full volume.

Panning the Sound

- Get the current pan value

```
HRESULT GetPan(LPLONG plPan);
```

- Before using SetPan or GetPan functions, you must set the buffer to use these controls
- Need to set DSBCAPS_CTRLPAN flag in the DSBUFFERDESC structure when you create the secondary buffer

Reference

- DirectSound Overview
  http://telnet.or.kr/directx/htm/directsound.htm
- DirectSound C/C++ Reference
  http://telnet.or.kr/directx/htm/directsoundccreference.htm