7. Dźwięk

Niezbędna rzeczą w każdym projekcie multimedialnym, czy to w grze czy w prezentacji, jest dźwięk. Omówimy sobie teraz podstawy korzystania z biblioteki *DirectSound*.

7.1 Podstawy teoretyczne

Dźwięk to drgania, które ludzkie ucho odbiera jako zmianę ciśnienia powietrza wywieranego na bębenki. Mikrofon może również odbierać te drgania i zamieniać je na prąd elektryczny. Po wzmocnienu ten sam prąd możemy wysłać do głośników które odtworzą wibracje. Dźwięk zamieniony na prąd jest reprezentowany przez kształt fali powstały na skutek zmiany napięcia elektrycznego w czasie

Dźwięk w Windows zapisuje się w formacie *PCM* (*Pulse Code Modulation*). *PCM* wykorzystują także odtwarzacze *CD*. Modulacja kodowo-impulsowa ma dwa parametry: częstotliowość próbkowania - liczba pomiarów amplitudy na jedną sekundę czyli rozpiętość między maksymalnym a minimalnym wychyleniem fali, wielkość próbki czyli liczba bitów przeznaczonych na zapisanie wartości amplitudy jednej próbki. Wiadomo, że częstotliwość próbkowania na płytach *CD* wynosi *44.1 kHz* czyli *44100* próbek na sekundę. Jest tak ponieważ ludzkie ucho słyszy do *20 kHz*, tak więc, aby uchwycić całe pasmo słyszalne należy zastosować *40 kHz*. Ze względu na fakt, że filtry dolnoprzepustowe (wykorzystywane w czasie konwersji z zapisu cyfrowego na analogowy) dają spadek wzmocnienia, częstotliwość próbkowania powinna być o *10%* większa czyli wynosić *44 kHz*. Mając na uwadze fakt, że często zapisuje się dzwięk wraz z obrazem, liczba próbek powinna być całkowitą częstotliwością amerykańskich| i europejskich częstotliwości ramek w obrazie, czyli odpowiednio wynosić *30Hz* i *25Hz*. W ten sposób dochodzimy do częstotliwości *44,1 kHz*.

Wybierając rozmiar próbki należy pamiętać, że próbka musi być w stanie zapisać pełen zakres dynamiki dzwięku który chcemy przetworzyć na formę cyfrową. Zakres dynamiki (w decybelach) między dwoma dźwiękami można wyliczyć posługując się wzorem:

$$dB = 20 \cdot \log(A_1 / A_2)$$

gdzie A_1 i A_2 to amplitudy obu dźwięków. W prostym przypadku, kiedy próbka ma rozmiar 1 bita, zakres dynamiki jest zerowy, ponieważ możliwa jest tylko jedna amplituda. W przypadku próbki o rozmiarze *8* bitów możliwych jest *256* różnych amplitud, zatem zakres dynamiki wynosi:

$$dB = 20 \cdot \log(256)$$

co nam daje *48 dB*. Zakres dynamiki odpowiada różnicy natężenia dźwięku do cichego pomieszczenia do pracującej kosiarki do trawy. Podwojenie rozmiaru próbki do *16* bitów daje nam zakres:

$$dB = 20 \cdot \log(65535)$$

czyli 96 decybeli, a wiec bardzo blisko różnicy między dźwiękiem słyszalnym i granica bólu. Dlatego ten zakres jest idealny do reprodukcji muzyki, ponieważ używanie 16 bitowych próbek przy *44.1 kHz* daje nam *635* megabajtów co wystarcza aby zapisać *3600* sekund muzyki. Zmniejszenie częstotliwości próbkowania obniży nam znacząco jakość zapisnej muzyki, dlatego niezbędna jest kompresja. Zajmiemy się sposobem kompresji opracowanym przez *Xiph.org* o nazwie *OggVorbis*. *Ogg* jest formatem ogólnodostępnym dla każdego, nie podlega żadnym opłatom licencyjnym i ma dużo lepszą jakość skompresowanego dźwięku niż popularny format *mp3*. Trzeba też wspomnieć, że format *mp3* podlega opłatom licencyjnym (zobacz *Bibliografia* pozycja 17 i 18).

7.2 Podstawy *DirectSound*

Biblioteka *DirectSound* pozwala na komunikację z kartą dźwiękową. W głównej mierze zapewnia ona miksowanie dźwięku z wykorzystaniem sprzętu, czyli karty albo przy użyciu programowej emulacji zaimplementowanej w bibliotece. Podstawowymi pojęciami w *DSound* są bufory dźwięku. Zawsze istnieje jeden podstawowy bufor, który jest odgrywany przez kartę dźwiękową i bufory pochodne które trzymają źródła dźwięku. Na chwilę przed odegraniem podstawowego bufora przez kartę dźwiękową *DSound* miksuje dostępne bufory i wysyła je do bufora podstawowego.

Pierwszą czynnością jest przeprowadzenie enumeracji kart dźwiękowych do czego służy funkcja *DirectSoundEnumerate* przedstawiona na listingu 109.

```
HRESULT WINAPI DirectSoundEnumerate(
  LPDSENUMCALLBACK  lpDSEnumCallback,
  LPVOID  lpContext
);
```

Listing 1 – *DirectSoundEnumerate*, Źródło *DirectX SDK*

Pierwszy parametr jest to funkcja *CALLBACK*, która zostanie wywołana dla każdej znalezionej karty dźwiękowej. Drugi parametr to wskaźnik do struktury użytkownika. Będziemy jej używać do trzymania danych o kartach. Definicja funkcji *CALLBACK* podana jest na listingu 110.

```
BOOL CALLBACK DSEnumCallback(
  LPGUID  lpGuid,
  LPCSTR  lpcstrDescription,
  LPCSTR  lpcstrModule,
  LPVOID  lpContext
);
```

Listing 2 – *DirectSoundEnumerate*, Źródło *DirectX SDK*

lpGuid to identyfikator karty dźwiękowej (będziemy się nim posługiwać później dlatego należy go zapisać); *lpcstrDescription* to nazwa karty, trzeci parametr możemy pominąc a *lpContext* to wskaźnik do naszej struktury. Przykład znajduje się na listingu 111.

```
DirectSoundEnumerate((LPDSENUMCALLBACK)DSEnumProc,(VOID*)this );
```

Listing 3 – Użycie *DirectSoundEnumerate*, Źródło własne

Jako parametr przekazaliśmy *this* tak, abyśmy mieli dostęp do obiektu, który przeprowadza enumeracje wewnątrz funkcji enumerującej, co możemy zaobserwować na listingu 112.

```
BOOL CALLBACK DSEnumProc(LPGUID lpGUID,
LPCTSTR lpszDesc, LPCTSTR lpszDrvName, LPVOID lpContext )
{
      LTQSNDMusique* pTemp = (LTQSNDMusique*)lpContext;
      memcpy( pTemp->m_pDevices[ pTemp->m_dwNumberOfDevices].szName,
            lpszDesc, strlen( lpszDesc ) );
      memcpy( pTemp->m_pDevices[
            pTemp->m_dwNumberOfDevices].szDriverName, lpszDesc, strlen(
            lpszDrvName ) );
      LPGUID lpTemp = NULL;
      if (lpGUID != NULL)  //  NULL only for "Primary Sound Driver".
      {
            if ((lpTemp = (LPGUID)malloc(sizeof(GUID))) == NULL)
            {
                  return(TRUE);
            }
            memcpy(lpTemp, lpGUID, sizeof(GUID));
      }
      pTemp->m_pDevices[ pTemp->m_dwNumberOfDevices ].pGuid = lpTemp;
      ++pTemp->m_dwNumberOfDevices;
      return(TRUE);
}
```

Listing 4 – Funkcja *enumerująca*, Źródło własne

Po sprawdzeniu każdej karty powinniśmy sprawdzić, czy karta ma przynajmniej jeden podstawowy bufor. Możemy przyjąć że będziemy używać tylko podstawowej karty dźwiękowej która musi mieć podstawowy bufor. Tworzymy obiekt *DSound* dla konkretnej karty dźwiękowej. Funkcja tworząca ten obiekt podana jest na listingu 113

```
HRESULT WINAPI DirectSoundCreate8(
  LPCGUID  lpcGuidDevice,
  LPDIRECTSOUND8 * ppDS8,
  LPUNKNOWN  pUnkOuter
);
```

Listing 5 – *DirectSoundCreate8*, Źródło *DirectX SDK*

Pierwszy parametr to *GUID*, może być *NULL* dla podstawowej karty dźwiękowej. Drugi to wskaźnik który otrzyma obiekt *DSound* przy pomocy którego będziemy się komunikować z kartą dźwiękową. Przykład zastosowania znajduje się na listingu 114.

```
hr = DirectSoundCreate8( pGuid, &m_pDSound, NULL );
if ( FAILED( hr ) )
{
      return FALSE;
}
```

Listing 6 – Przykład użycia *DirectSoundCreate8*, Źródło własne

Następnie musimy uzyskać pierwszeństwo zapisu do karty dźwiękowej. *Windows* jest sytemem wielozadaniowym i może się zdarzyć, że nie tylko nasza aplikacja chce

odgrywać dźwięki. Do uzyskania pierwszeństwa służy metoda *IDirectSound8::SetCooperativeLevel* która przedstawiona jest na listingu 115.

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

Listing 7 – *IDirectSound8::SetCooperativeLevel*, Źródło *DirectX SDK*

Przykład zastosowania znajduje się na listingu 116.

```
hr = m_pDSound->SetCooperativeLevel( m_hWnd, DSSCL_EXCLUSIVE |
DSSCL_PRIORITY );
if ( FAILED( hr ) )
{
      return FALSE;
}
```

Listing 8 – Użycie *SetCooperativeLevel*, Źródło własne

Pierwszy parametr to uchwyt do okna naszej aplikacji. Flagi *DSSCL_EXCLUSIVE* i *DSSCL_PRIORITY* nadają naszej aplikacji wyłączność na pisanie do podstawowego bufora oraz możliwość ustalania formatu podstawowego bufora. Teraz możemy utworzyć bufor podstawowy posługując się metodą *IDirectSound8::CreateSoundBuffer* przedstawioną na listingu 117.

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

Listing 9 – *IDirectSound8::CreateSoundBuffer*, Źródło *DirectX SDK*

Pierwszy parametr to opis bufora, drugi to wskaźnik, gdzie zostanie zapisany uchwyt bufora a trzeci parametr musi być *NULL*. Struktura opisująca bufor *DSBUFFERDESC* znajduje się na listingu 118.

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

Listing 10 – *DSBUFFERDESC*, Źródło *DirectX SDK*

Przykład zastosowania metody *IDirectSound8::CreateSoundBuffer* oraz struktury *DSBUFFERDESC* znajduje się na listingu 119.

```
DSBUFFERDESC bufferDesc;
ZeroMemory( &bufferDesc, sizeof( DSBUFFERDESC ) );
bufferDesc.dwSize = sizeof( DSBUFFERDESC );
bufferDesc.dwFlags = DSBCAPS_PRIMARYBUFFER | DSBCAPS_STICKYFOCUS;
bufferDesc.dwBufferBytes = 0UL;
bufferDesc.lpwfxFormat = NULL;

hr = m_pDSound->CreateSoundBuffer( &bufferDesc, &m_pDSPrimaryBuffer,
NULL );

if ( FAILED( hr ) )
{
      return FALSE;
}
```

Listing 11 – Tworzenie podstawowego bufora, Źródło własne

Użycie flag *DSBCAPS_PRIMARYBUFFER* powoduje utworzenie podstawowego bufora. Flaga *DSBCAPS_STICKYFOCUS* powoduje, że nawet jeśli nasza aplikacja utraci *focus*, nadal dźwięk będzie odgrywany. Teraz należy ustawić format podstawowego bufora przy pomocy metody *IDirectSound8::SetFormat* przedstawionej na listingu 120.

```
HRESULT SetFormat(
  LPCWAVEFORMATEX pcfxFormat
);
```

Listing 12 – *IDirectSound8::SetFormat*, Źródło *DirectX SDK*

Parametrem jest wskaźnik do struktury *WAVEFORMATEX* która znajduje się na listingu 121.

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

Listing 13 – *WAVEFORMATEX*, Źródło *DirectX SDK*

Przykład zastosowania *SetFormat* i *WAVEFORMATEX* znajduje się na listingu 122.

```
ZeroMemory( &m_WaveFormat, sizeof( WAVEFORMATEX ) );
m_WaveFormat.wFormatTag = WAVE_FORMAT_PCM;
m_WaveFormat.nChannels = 2;   //stereo
m_WaveFormat.cbSize = 0;
m_WaveFormat.nSamplesPerSec = 44100;
m_WaveFormat.wBitsPerSample = 16;
m_WaveFormat.nAvgBytesPerSec = m_WaveFormat.nSamplesPerSec *
m_WaveFormat.nChannels * m_WaveFormat.wBitsPerSample/8;
m_WaveFormat.nBlockAlign     = m_WaveFormat.nChannels *
m_WaveFormat.wBitsPerSample/8;

hr = m_pDSPrimaryBuffer->SetFormat( &m_WaveFormat );

if ( FAILED( hr ) )
{
      return FALSE;
}
```

Listing 14 – Przykład ustawiania formatu tworzenia podstawowego bufora, Źródło własne

Zwróćmy uwagę na *nAvgBytesPerSec*, w jaki sposób obliczana jest liczba bajtów na sekundę.

Ostatnią rzeczą do zainicjalizowania jest wątek, który będzie dekompresował i odgrywał dźwięk. Pamiętajmy, że będziemy dekompresować dźwięk na tyle. na ile on będzie potrzebny. Dekompresja np. *3* minut w *44.1 kHz* przy *16* bitach daje ok. *30* megabajtów. Dlatego będziemy prowadzić dekompresje „w locie”. Tworzenie wątku pokazane jest na listingu 123.

```
if ( NULL == ( m_hThreadHandle = (HANDLE)CreateThread( NULL, 65536,
            (LPTHREAD_START_ROUTINE)ThreadRout, (void *)this, NULL,
            &m_dwThreadID ) ) )
      return FALSE;
```

Listing 15 – Tworzenia wątku, Źródło własne

Funkcja wątku znajduje się na listingu 124.

```
static DWORD WINAPI __stdcall ThreadRout(void *pObject)
{
      LTQSNDMusique *pMusique = (LTQSNDMusique *)pObject;
      if ( pMusique != NULL )
      {
            while ( pMusique->Update() )
                  Sleep(500);
      }
    return 0;
}
```

Listing 16 – Funckja wątka, Źródło własne

Uśpienie wątka jest zależne od rozmiarów buforów pochodnych, które odgrywają dźwięk. Przyjęcie małych rozmiarów wymaga częstego wykonywania wątka, przez co następuje zmniejszenie wartości *sleep*.

7.3 Odtwarzanie dźwięku

Przed rozpoczęciem dekompresji dźwięku musimy przygotować sobie pochodny bufor, który będzie trzymać dźwięk w formacie *PCM*. Przykład znajduje się na listingu 125.

```
DSBUFFERDESC desc;
ZeroMemory( &desc, sizeof( DSBUFFERDESC ) );
desc.dwSize = sizeof( DSBUFFERDESC );
desc.dwFlags = DSBCAPS_GETCURRENTPOSITION2 | DSBCAPS_GLOBALFOCUS;
desc.dwBufferBytes = dwBytesPerSecond;
desc.lpwfxFormat = pWaveFormat;

if ( FAILED( pDS->CreateSoundBuffer( &desc, &m_pDSBuffer, NULL ) ) )
{
      return;
}
```

Listing 17 – Tworzenie bufora pochodnego, Źródło własne

Utworzenie bufora pochodnego jest podobne jak utworzenie bufora podstawowego. Różnica to przedewszystkim inne flagi:

- *DSBCAPS_GETCURRENTPOSITION2* pozwala nam pobierać dokładną pozycje w buforze, gdzie możemy zapisać dźwięk
- *DSBCAPS_GLOBALFOCUS* pozwala buforowi być odgrywanym nawet wtedy, kiedy nasza aplikacja straci *focus*.

Następna różnicą jest podanie *WAVEFORMATEX* do struktury *DSBUFFERDESC*. W tym przypadku *WAVEFORMATEX* opisuje format dźwięku w buforze podstawowym. Jeżeli nie ma konwersji między buforem pochodnym a podstawowym, wtedy całość działa najszybciej. Teraz możemy rozpocząć dekompresję dźwięku. Tak przygotowujemy sobie plik *ogg* do dekodowania używając biblioteki dostarczonej przez *Xiph.org*. Przykład znajduje się na listingu 126.

```
ZeroMemory( &m_Vf, sizeof(OggVorbis_File) );
m_pBuffer = new BYTE[ dwBytesPerSecond * 2 ];

if ( NULL == ( m_pFile = fopen( szName, "rb" ) ) )
      return;

if ( ov_open( m_pFile, &m_Vf, NULL, 0 ) < 0 )
      return;
```

Listing 18 – Przygotowanie pliku *Ogg* do dekodowania, Źródło własne

Celowo nie podajemy składni *ov_open*, gdyż można zastosować tu dowolną kompresję. *OggVorbis* jest tylko przykładem.

W czasie dekodowania należy sobie przygotować bufor z rozkodowanym dźwiękiem do którego będziemy dekompresować kolejne kawałki dźwięku i z którego będziemy kopiować dane do bufora *DSound*. Bufor jest potrzebny, gdyż biblioteka *OggVorbis* zazwyczaj dekompresuje dźwięk w czterokilobajtowych kawałkach, a takie niekoniecznie w całości muszą się zmieścic do bufora *DSound*. Popatrzmy na przykład z listingu 127.

```
while ( m_dwBufferBytes < m_dwBytesPerSecond )
{
      DWORD dwTemp = Decode( m_pBuffer + m_dwBufferBytes, 32768 );
      m_dwBufferBytes += dwTemp;
      if ( dwTemp == 0 )
            break;
}
```

Listing 19 – Wypełnienie bufora, Źródło własne

Wypełniamy bufor tak długo, aż będzie całkowicie zapełniony zdekompresowanym dźwiękiem. Proszę zauważyć, że stosujemy wszędzie bufory rozmiaru sekundy. *m_dwBytesPerSecond* trzyma ilość bajtów na sekundę.

```
DWORD LTQSNDOggStream::Decode( BYTE* pBuffer, DWORD dwNumberOfBytes )
{
      if ( m_bIsPlayed == TRUE )
            return ov_read( &m_Vf, (char*)pBuffer, dwNumberOfBytes, 0,
2, 1, &m_iCurrentSection );

      else
      {
            memset( pBuffer, 0, dwNumberOfBytes );
            return dwNumberOfBytes;
      }
}
```

Listing 20 – Dekompresja, Źródło własne

Metoda dekompresująca dekompresuje przy pomocy funkcji biblioteki *OggVorbis* albo wypełnia ciszą, jeżeli muzyka nie została odegrana.

Kiedy już wypełnimy bufor zdekompresowanym dźwiękiem, musimy go wysłać do bufora *DSound*. W tym celu musimy *zalockować* bufor. Bufory w *DSound* są *kołowe*, to znaczy, że nie jest pewne, że dostaniemy jeden obszar pamięci. Możemy dostać dwa obszary, np. na końcu i na początku bufora. Metoda *IDirectSoundBuffer8::Lock* przedstawiona na listingu 129 *lockuje* bufor.

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

Listing 21 – *IDirectSoundBuffer8::Lock*, Źródło *DirectX SDK*

Pierwszy parametr to miejsce w bajtach, gdzie chcemy zalockować obszar. Drugi parametr to rozmiar w bajtach. Trzeci - to wskaźnik gdzie metoda zapisze pozycje pierwszego obszaru do wypełnienia. Czwartym parametrem jest rozmiar pierwszego obszaru. Piąty i szósty parametr są analogiczne do trzeciego i czwartego tyle, że dla drugiego obszaru do wypełnienia. Ostatni parametr zazwyczaj będzie *NULL*.

Do obliczenia pozycji do *zalockowania* w buforze potrzebna nam będzie inna metoda bufora. *IDirectSoundBuffer8::GetCurrentPosition*:

```
HRESULT GetCurrentPosition(
  LPDWORD pdwCurrentPlayCursor,
  LPDWORD pdwCurrentWriteCursor
);
```

Listing 22 – *IDirectSoundBuffer8::GetCurrentPosition*, Źródło *DirectX SDK*

Pierwszy parametr to wskaźnik, który otrzyma pozycję aktualnie odgrywaną przez *DSound*. Drugi parametr to wskaźnik, który otrzyma pozycję pod jaką można zapisywać. Aby rozpocząć odgrywanie bufora posłużymy się metodą *IDirectSoundBuffer8::Play* przedstawioną na listingu 131.

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

Listing 23 – *IDirectSoundBuffer8::Play*, Źródło *DirectX SDK*

Pierwszy parametr musi być *NULL*. Drugi to priorytet bufora, *0* – oznacza najniższy priorytet, *0xffffffff* – najwyższy. Priorytety są ustawiane aby poinformować *DSound* które bufory są ważniejsze. Może się zdażyć, że nie wystarczy zasobów (pamięci, kanałów w karcie dźwiękowej) aby utworzyć wymaganą przez użytkownika ilość strumieni dźwięku, wtedy *DSound* będzie pierw tworzył bufory o najwyższym priorytecie, a dla których nie starczy zasobów będą odgrywane w sposób mniej efektywny, na przykłdad będą miksowane przy użyciu procesora. Biblioteka *DSound* wybiera najefektywniejszą metodę obsługi buforów dla których nie starczyło zasobów. Trzecim parametrem metody *Play* są flagi. Jeżeli chcemy, żeby DSound potraktował bufor jako kołowy musimy przekazać flagę: *DSBPLAY_LOOPING*.

Na listingu 132 podamy przykład, który pokazuje dekompresje, wypełnianie bufora i sprawdzanie podstawowych błędów oraz przeliczanie pozycji w buforze.

```
BOOL LTQSNDOggStream::DecodeDSound()
{
	while ( m_dwBufferBytes < m_dwBytesPerSecond )
	{
		DWORD dwTemp = Decode(m_pBuffer + m_dwBufferBytes, 32768 );
		m_dwBufferBytes += dwTemp;
		if ( dwTemp == 0 )
			break;
	}

	if ( m_dwBufferBytes == 0 )
	{
		m_pDSBuffer->Stop();
		m_bIsPlayed = FALSE;
		return TRUE;
	}

	DWORD playPos,unusedWriteCursor;
	DWORD writeLen;
	LPVOID	p1,p2;
	DWORD l1,l2;

	HRESULT hRes;

	hRes =
	m_pDSBuffer->GetCurrentPosition( &playPos, &unusedWriteCursor );

	if ( hRes != DS_OK )
		playPos = 0;
```

```
	if ( m_dwWritePos < playPos)
		writeLen = playPos - m_dwWritePos;

	else
		writeLen = m_dwBytesPerSecond - ( m_dwWritePos - playPos );

	while (DS_OK != m_pDSBuffer->Lock( m_dwWritePos, writeLen, &p1,
		&l1, &p2, &l2 ,0 ) )
	{
		m_pDSBuffer->Restore();

		if ( m_bIsPlayed == TRUE )
			m_pDSBuffer->Play(0, 0, DSBPLAY_LOOPING);
	}

	if ( ( p1 ) && ( l1 > 0 ) && ( m_dwBufferBytes > 0 ) )
	{
		if ( l1 > m_dwBufferBytes )
			l1 = m_dwBufferBytes;

		memcpy( p1, m_pBuffer, l1 );
		m_dwBufferBytes -= l1;

		if ( m_dwBufferBytes > 0 )
			memcpy( m_pBuffer, m_pBuffer + l1, m_dwBufferBytes );
	}

	if ( ( p2 ) && ( l2 > 0 ) && ( m_dwBufferBytes > 0 ) )
	{
		if ( l2 > m_dwBufferBytes )
			l2 = m_dwBufferBytes;

		memcpy( p2, m_pBuffer, l2 );
		m_dwBufferBytes -= l2;

		if ( m_dwBufferBytes > 0 )
			memcpy( m_pBuffer, m_pBuffer + l2, m_dwBufferBytes );
	}

	if ( m_dwBufferBytes < 0 )
		m_dwBufferBytes = 0;

	m_pDSBuffer->Unlock(p1,l1,p2,l2);
	m_dwWritePos += l1 + l2;

	if ( m_dwWritePos >= m_dwBytesPerSecond )
		m_dwWritePos -= m_dwBytesPerSecond;

	return TRUE;
}
```

Listing 24 – Przykład dekompresji i wypełniania bufora *DSound*, Źródło własne

Do zatrzymania bufora posłużymy się metodą *IDirectSoundBuffer8::Play* przedstawioną na listingu 133.

```
HRESULT Stop();
```

Listing 25 – *IDirectSoundBuffer8::Stop*, Źródło *DirectX SDK*

Funkcja *OggVorbis* kończąca dekompresje znajduje się na listingu 134.

```
void LTQSNDOggStream::Release()
{
      ov_clear( &m_Vf );
}
```

Listing 26 – Kończenie dekompresji *OggVorbis*, Źródło własne