분산 버퍼 구조의 실시간 오디오 믹서의 구현

임진영*, 송동호
한국항공대학교 컴퓨터 공학과

An Implementation of Real-time Audio Mixer on the Distributed Buffer Structure

Jin-Young Lim, Dong Ho Song
Hankuk Aviation University

요 약

멀티미디어 응용프로그램 환경에서 실시간 오디오 믹싱 기능은 자주 요구되는 지원 사항중에 하나이다. 지금까지는 단일시스템에서의 여러가지 스트림 조합에 대한 믹싱이 일반적인 방법이나 분산 응용프로그램이 일반화됨에 따라 분산 스트림에 대한 데이타 믹싱이 필요하게 되었다. 이러한 요구를 충족시킬 수 있는 방법으로써 분산 버퍼들로부터 스트림을 억세스하여 믹싱 모듈의 내부 버퍼를 통한 실시간 오디오 믹싱을 연구해 보았다.

1. 서론

멀티미디어 응용프로그램 환경에서 실시간 오디오 믹싱(mixing) 기능은 자주 요구되는 지원 사항중에 하나이다. 대표적으로 게임의 예를 들자면, 주인공 케릭터가 어느 도시의 거리를 활보한다고 할때 어느 상점에서는 거리를 향한 음악이 흘러 나온다. 이때 바로 옆 도로에서는 자동차가 지나갈때의 소음이 들려 온다. 차가 막히자 그 자동차는 돌발적인 경적을 울린다. 이렇듯 한 상황에 있어서 동시 다발적인 소리를 하나의 시스템에 추가적으로 요구할 경우가 있을 수 있다. 이러한 응용 프로그램을 개발시 모든 상황에 맞는 미리 믹스된 사운드를 인덱스화하여 표현할 수도 있겠지만, 그보다는 각각의 고유한 소리를 준비하고 발생한 이벤트에 해당하는 사운드를 불러 내어 현재 출력되는 사운드에 믹스하는 것이 많은 경우에 있어서 더 타당하다고 볼 수 있다. 이런 경우에는 각 플랫폼에서 제공하는 가장 적절한 솔루션을 사용하여 유연하게 오디오 믹서를 구현할 수 있다. 다음에 오디오 컨퍼런스를 생각해 보자. 실시간으로 각 회의장에 연결된 사람들의 목소리는 게임에서의 미리 준비된 고유의 소리와 믹싱 처리할 데이터의 확보 차원에서 다르다. 왜냐하면, 게임에서는 믹싱 프로세싱 보다 먼저 오디오 스트림을 버퍼화하여 대기 시킬 수 있지만, 오디오 컨퍼런스의 경우에는 버퍼의 대기라는 것이 없는 실시간 버퍼링이 주요 대상이기 때문이다. 이 경우에는 각 플랫폼에서 제공하는 범용적인 믹싱 솔루션을 이용하기는 적절하지 않다. 이와 같이 지금까지는 단일 시스템에서의 여러가지 스트림 조합에 대한 믹싱이 일반적인 방법이나 분산 응용프로그램이 일반화됨에 따라 분산 스트림에 대한 데이타 믹싱이 필요하게 되었다. 이러한 요구를 충족시킬 수 있는 방법으로써 분산 버퍼들로부터 직접 스트림을 엑세스하여 믹서 모듈의 내부 버퍼를 통한 소프트웨어적인 실시간 오디오 믹싱에 대한 연구를 해 보았다.

2. 본론

오디오 믹싱이란 두가지 의미가 될 수 있다. 한가지는, 하드웨어에서 지원하는 디지탈 웨이브 형태와 미디(MIDI), CD 오디오 등 여러가지 오디오 디바이스 타입이 있는데, 이것은 각각 다른 타입의 오디오 소스를 통합 볼륨 조절 하거나 선택적으로 동시에 오디오 출력하는 경우에 해당한다. 다른 하나는, 디지탈 웨이브 출력 장치나 미디에서 볼 수 있듯이 같은 오디오 타입에 속하는 멀티플 오디오 스트림을 하나의 오디오 스트림으로 혼합하는 경우이다. 본 논고에서는 후자의 경우로써 디지탈 웨이브 형태의 별개의 오디오 스트림들을 하나의 웨이브 스트림 형태로 믹싱하고 단일의 출력 장치로 전송(submit)하는 것으로, 오디오 스트림들이 각각의 독립된 출력 장치로 동시에 플레이한 효과를 갖게 하는 소프트웨어적인 '믹싱(mixing)' 또는 '믹서(mixer)'라는 용어를 사용했다.

1. 믹서(mixer) 모듈의 구성

디지탈 웨이브 오디오 스트림(줄여서 '웨이브 스트림'이라고 하겠다)들의 믹서 모듈 구현에 있어서 구성별로 블록 관리 모듈, 채널 관리 모듈, 믹싱(mixing) 모듈로서 크게 세가지 부분으로 나누어질 수 있는데, 우선 각각의 모듈의 기능에 대해 간략히 언급하고 구체적인 구현 방법에 대해서는 앞으로 설명하겠다.

1.1 블록 관리 모듈

믹싱을 위한 버퍼들은 다시 두가지로 나눌 수 있는데, 첫째로 화일들이나 외부 모듈로부터 전해진 웨이브 멀티플 스트림들을 받아 믹싱 모듈의 요구에 끊이지 않게 공급할 수 있는 버퍼들이 필요하다. 둘째, 믹싱 모듈 자신이 소리의 인터럽션이 느껴지지 않도록 출력 디바이스로 공급할 수 있는 버퍼들이 필요하다. 따라서 전자는 각 채널별 최소 더블 버퍼링이 가능하도록 블록 관리를 요구하는 반면 후자는 출력 디바이스에서 단위 시간에 처리할 수 있는 수준의 블록 크기 이하의 최소 다중 버퍼가 필요로 한다.

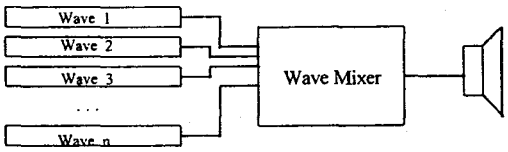
1.2 채널 관리 모듈

단일 프로세스 또는 복수개의 프로세스등에서 공급하는 멀티플 웨이브 스티림들은 각각의 스트림이 하나의 채널로 구성되어진다. 여기서 말하는 채널은 스테레오 2채널, 모노 1채널의 의미가 아닌 모노, 스테레오 구별 없는 서로 다른 웨이브 스트림들을 의미 한다. 이렇게 구성된 각 채널은 예를 들어, 몇개가 믹싱 되어야 하

는지 그리고 어느 시간에 새로운 웨이브 스트림이 추가되는지, 또한 각 스트림 블록은 믹싱시 어느 포지션까지 믹싱 되었는지에 대한 관리가 필요하다. 다시말해, 별개의 웨이브 스트림들 수에 대한 변동이 있을 때 마다 채널 할당 또는 삭제가 이루어지며 동시에 각 채널별 웨이브 스트림을 싣을 수 있는 블록 할당과 삭제가 이루어지며 믹싱 포지션 정보등을 관리한다.

1.3 믹싱 모듈

각 채널에 할당된 버퍼에 도착된 스트림 데이터는 음감의 인터럽션이 발생하지 않을 수준의 단위 크기로 믹싱 모듈의 내부 버퍼로 다시 읽어들인다. 이때 실시간 오디오 디바이스 출력이 전제되므로 이 내부 버퍼를 이용한 신속한 웨이브 믹스 알고리즘이 필요하다. 아울러 이 알고리즘에 따르는 내부의 멀티플 버퍼링이 필요하다. 다음의 [그림 1]은 웨이브 스트림과 믹서, 출력 장치와의 관계를 개념적으로 도시화하였다.



[그림 1] 웨이브 스트림과 믹서, 출력 장치와의 관계

2. 블록 관리 모듈의 구성

웨이브 스트림의 블록 관리 모듈 기능은 언급했다시피 오디오 출력의 끊임을 느끼지 못할 정도의 범위내에서 믹싱 모듈에 웨이브 스트림 블록을 공급하는데 있다. 이러한 기능을 구현함에 있어서 크게 두 가지 유형의 구현 방법을 볼 수 있다. 한 가지는 '단일 시스템 환경'과 '웨이브 화일'을 믹싱할 때에 적합한 방법으로써, 시스템 수준에서 제공하는 화일 I/O 메니저를 이용하여 웨이브 화일을 마치 메모리에 로드된 하나의 버퍼로 생각하여 포인터 스트럭쳐 지향 타입의 구현 방법이 있다. 두번째는 '네트워크 환경'에서 적합한, 웨이브 데이터의 배열(array) 버퍼화된 구조의 구현 방법이다. 먼저 단일 시스템에서의 믹싱 환경에서 가장 유효한 방법인 화일 I/O 메니저에 의한 블록 관리를 알아본다.

MS-WINDOWS에서는 웨이브 스트림을 출력하기 위한 여러가지 프로그래밍 기술을 제공하는데, 이것에 대해 알아보기 위해선 윈도우즈 표준 오디오 데이타 포맷 중에 하나인 'Wave Form'을 살펴봐야한다. 이것은 또한 MS-WINDOWS가 멀티미디어 화일 포맷을 RIFF(Resource Interchange File Format)형식으로 취하고 있는 만큼 웨이브 오디오 화일도 이에 준하고 있다. 따라서 RIFF형식에 따르는 웨이브 화일 포맷을 알아봐야 한다. 화일 포맷을 알아야 하는 것은 그것에서 블록 관리에 대한 접근법의 아이디어를 취할 수 있기 때문이다.

2.1 단일 시스템에서의 웨이브 믹싱

웨이브 화일의 스트림을 주기적으로 블록화하여 믹싱 모듈에 도달하게끔 하는데 있어서, 멀티미디어 화일 I/O를 담당하는 MMIO API군과 어떠한 크기의 화일이라도 메모리 할당을 가능케 하는 GlobalAlloc API의 조합으로 블록 관리 모듈의 개발을 쉽게 할 수 있다. 먼저 간단한 예제 코드를 보고 설명을 하겠다.

```
    if (!(hmmio = mmioOpen(szWaveFilename, NULL,
MMIO_READ | MMIO_ALLOCBUF)))
    goto ErrorCleanup;
    if (!lpData = GlobalAllocPtr(GMEM_MOVEABLE |
GMEM_SHARE, dwDataSize)))
    goto ErrorCleanup;
  if (mmioRead(hmmio, (HPSTR)lpData, dwDataSize) !=
(LONG) dwDataSize)
    goto ErrorCleanup;
```

mmioOpen 함수의 MMIO_ALLOCBUF라는 플래그는 이미 셋팅된 또는 디폴트 8KB의 화일 I/O의 내부 버퍼를 이용한 '버퍼화된 화일 구조'를 생성시키고, GlobalAlloc 함수를 이용하여 시스템 수준에서 적당한 크기의 메모리를 할당하여 스스로 해당되는 스트림 블록을 메모리에 로드 시킨다. 이렇게 메모리 할당된 화일은 매우 큰 화일이라도 하나의 버퍼화된 화일로써 모든 웨이브 스트림이 메모리에 로드된 것으로 간주하여 코딩을 하도록 지원한다. 웨이브 오디오 API군을 살펴보기 전에 마찬가지로 대표적인 출력 함수를 보겠다.

```
  HWAVEOUT  hWaveOut;
  HWAVEHDR  hWaveHdr;
  LPWAVEHDR lpWaveHdr;
  lpWaveHdr->lpData = lpData;
// 할당 받은 메모리 블록의 포인터
  lpWaveHdr->dwBufferLength = dwDataSize;
// 데이터 청크의 크기
  lpWaveHdr->dwFlag = 0L;
  lpWaveHdr->dwLoops = 0L;
waveOutPrepareHeader(hWaveOut, lpWaveHdr,
sizeof(WAVEHDR));
wResult = waveOutWrite(hWaveOut, lpWaveHdr,
sizeof(WAVEHDR));
```

윈도우즈 확장 서비스에서 제공하는 웨이브 스트림에 대한 구조체를 근간으로 화일 헤더 청크의 정보를 설정해 주고 그 스트럭쳐 포인터를 바로 API함수에 인자로 전달한다. waveOutPrepareHeader 함수는 웨이브 스트림이 할당 받은 메모리 블록과 크기를 오디오 출력 장치에 인식시키며 waveOutWrite 함수는 맵핑하여 준비된 블록을 오디오 디바이스에 보내 사운드 출력한다. 오디오 장치에 의해 스트림이 출력된 결과로 메시지를 어플리케이션에 발생 시킨다. 이어서 어플리케이션은 이 메시지를 체크하여 버퍼된 웨이브 화일의 현재 위치와 sizeof(WAVEHDR)만큼을 포인터 증가시켜 다시 화일 I/O 메니저로 하여금 웨이브 블록을 갱신하도록 한다. 정리하면, MS-WINDOWS의 웨이브 스트림 관련 API들은 커스텀 블록 콘트롤을 위한 방법도 제공되지만 전반적으로 구조체 포인터를 인수로 받는 패턴을 가지고 있다. 그것은 화일 I/O 메니저의 지원과 화일로서의 억세스를 기본으로 다루어지는 이유이며 단일 시스템에서 응용 프로그램 개발의 편리성을 배려한 이유이기도 하다. 웨이브 화일의 재생에 있어서 주목할 점은, 이와같은 패턴의 API군은 단일 시스템에서의 자동화된 화일 I/O 메니저를 이용하여 블록 관리에 관한한 개발자에게 숨겨져 있어서 보다 쉽게 응용 프로그램을 설계할 수 있다는 장점이 있다. 반면에 오디오 컨퍼런스 상황과 같은 네트워크 환경에서의 실시간 웨이브 스트림 재생은 구조체 포인터형을 배제한 배열(array) 구조의 버퍼를 구현해야 한다는 것이다. 이것은 가시적인 유저 버퍼 형태로써 안정적인 오디오 출력을 유도해야 한다는 점에서는 단점으로 지적된다. 따라서 블록 콘트롤

부분에서는 단일 시스템과 분산 시스템에서 서로 다른 차이가 있다.

2.2 네트워크 환경에서의 웨이브 믹싱

네트워크 환경에서의 분산된 웨이브 스트림의 블록 공급에 있어서 두가지를 생각할 수 있다. 준비된 웨이브 화일을 공급하는 경우와, 오디오 컨퍼런스와 같이 실시간으로 발생하는 웨이브 스트림을 공급하는 경우이다. 전자는 공급측의 스트림 전송 속도가 소비측의 오디오 출력 보다는 빠르다는 전제하에, 단일 시스템에서와 같이 믹싱 모듈이 블록을 요구하기 전에 전송되는 스트림 블록을 미리 적재시키는 방법을 취하면 된다. 이 경우에서는 단일 시스템에서의 구현 방법을 배열 버퍼구조에 주안점을 두면 된다. 반면에 후자에서는 미리 적재시킬 블록이 없으므로 블록의 크기를 추가로 고려해야 할 것이다. 왜냐하면 블록의 공급은 그 크기에 따라서 연속된 스트림의 공급 주기가 결정되는데, 크기가 커지면 오디오 출력의 공백 시간이 발생할 수 있고 너무 작으면 믹서 프로세스 입장에서 비효율적인 연산 적업이 추가될 수 있기 때문이다. 일반적으로 블록 공급 입장에서의 블록 크기는 믹싱 모듈의 처리 단위 크기에 의존하는 것이 효율적이다. 믹싱 모듈에서의 믹스 처리 단위 크기는 웨이브 스트림의 샘플 크기와 샘플링 레이트를 고려하여 최적화되는데, 전송되는 블록 크기는 이 믹스 처리 단위와 같이 하거나 근사한 정수배가 가장 효율적이다.

MS-WINDOWS멀티미디어 확장 서비스에서 제공되는 웨이브 API군을 이용한 배열(array) 버퍼 지향적인 구현 기법을 설펴 보겠다. 네트워크 환경의 분산된 웨이브 블록을 엑세스 하기위해서는 화일 I/O를 이용하지 않으므로 상기 기술한 MMIO API군은 이용하지 않는다. 기본적으로 제공되는 몇가지 API를 이용하여 비교적 용이하게 구현될 수 있다. 다음은 관련된 코드이다.

```
  LPWAVEFORMAT) pcmWaveFormat,
  hwndApp, 0L, CALLBACK_WINDOW));
 lpBlockPtr = GlobalAllocPtr( GMEM_FIXED,
dwBlockSize);
      lpWaveHdr->lpData = lpBlockPtr;
      lpWaveHdr->dwBufferLength = dwBlockSize;
      lpWaveHdr->dwFlag = 0L;
      lpWaveHdr->dwLoops = 0L;

      waveOutPrepareHeader (hWaveOut, lpWaveHdr,
sizeof (WAVEHDR));
      waveOutWrite (hWaveOut, lpWaveHdr,
sizeof (WAVEHDR));
```

3. 채널 관리 모듈의 구성

믹서의 입장에서 독립된 웨이브 스트림을 하나의 채널이라고 한다. 별개의 새로운 웨이브 스트림이 추가될때 구조체 타입의 링크드리스트 채널 노드 한개가 새롭게 추가 된다. 더불어 그 구조체의 맴버 변수와 배열등이 추가로 생성 되는데, 그것은 하나의 고정된 메모리 블록을 할당하고, 믹싱 모듈의 믹스 처리 단위 크기보다 근사한 정수배의 블록 관리를 할 경우에는 다음 샘플의 위치 포인터에 대한 정보를 설정하게 된다. 마찬가지로 종결되는 웨이브 스트림의 경우도 새롭게 생성되는 스트림 관리와 같은 방식으로 각각을 삭제한다. 각 채널에 최소 더블 버퍼링이 요구되는데, 이것은 믹싱 모듈에서 하나의 버퍼를 소비하고 있을 때 다른 하나는 계속해서 전해지는 데이타를 담아내는 역할을 한다. 다음은 상기 열거한 채널 관리 자료 구조.를 보인 것이다.

```
struct _CHANNELNODE
{
   struct _CHANNELNODE *next;
   MIXPLAYPARAMS PlayParams;
   LPMIXWAVE lpMixWave;
   DWORD dwNumSamples;
   DWORD dwStartPos;
   DWORD dwEndPos;
   LPSAMPLE lpPos;
   LPSAMPLE lpEnd;
} CHANNELNODE, *PCHANNELNODE;
```

4. 믹싱 모듈 구성

저수준의 유일한 웨이브 오디오 출력 함수인 waveOutWrite 는 동시에 멀티 채널 출력을 지원하지 않는다. 그래서 이것을 가능하기 위해서는 멀티 채널 출력 장치를 가장하는 믹싱 모듈이 요구된다. 이것은 각각의 채널에 할당된 웨이브 스트림 블록에서 믹싱 처리 단위로써 믹싱 모듈 내부의 다중 버퍼로 다시 읽어 온다. 읽어 온 데이터는 샘플 단위로 각 채널의 샘플 웨이브 벡터값을 연산하여 내부의 다중 버퍼중 한 곳에 그 결과값을 넣는다[그림2, 3]. 이 버퍼가 waveOutWrite함수에 전달되어 사운드로써 출력을 발생시킨다. waveOutWrite함수에 의해 발생된 사운드는 각각의 스트림 사운드가 동시에 모두 들려지는 효과를 나타내며, 일단 이 함수에 프로세싱이 들어간 즉시 출력하기위한 이어지는 데이타 블록을 요구하는 메시지를 응용프로그램에게 발생 시킨다. 응용프로그램에서는 이 메시지를 포착함에 따라 상기의 믹싱 루틴이 반복된다. 백터값 연산중에 새로운 채널이 생성되면 현재의 연산을 중지하고 새로운 채널의 데이타를 포함하여 다시 리믹스 작업을 한다.

믹싱 처리 단위의 크기란 믹싱 모듈의 내부 버퍼 크기를 의미하며, 프로세싱 측면에서의 최적화에 영향을 준다. 모두 8개까지 채널을 생성시킬 수 있는 모듈을 예로 들자면, 11Khz 8bit Mono의 한 웨이브 스트림은 초당 11025 byte로 구성될 것이다. 믹싱 모듈의 내부 버퍼는 실시간 결과를 얻기위해 전형적으로 작은 크기인데 이경우 전형적인 길이는 초당 1/8 (11025/8 = 1378 samples)이다.

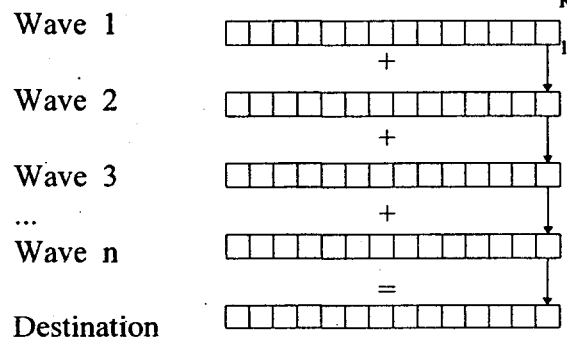
믹싱 모듈의 내부 버퍼는 최소 2개에서 약 6개 정도까지 다중 버퍼링을 하는데, 그 중 하나는 믹싱 연산의 데스티네이션 버퍼 역할을 하며, 또 하나는 이미 믹스되어 채워진 버퍼를 오디오 출력에서 소비시키는 역할을 한다. 그 이외의 버퍼는 각 역할 분담한 버퍼의 작업 시간이 서로 다른 까닭에 버퍼의 요구에 있어서 대기의 시간을 줄이는 효과를 준다. 각 버퍼의 역할에 대해 4중 버퍼를 예로써 [그림 3]에 나타내었다.

믹싱 알고리즘은 11Kh, 8bit, 모노의 샘플을 연산하는 예를 다음과 같이 구현해 볼 수 있다.

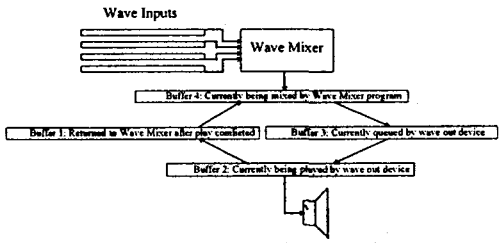
lpDest: 믹싱된 데이타가 들어가게될 데스티네이션버퍼

rgWaveSrc[] : 샘플 소스
iNumWaves : 웨이브 스트림의 갯수(즉, 채널 수)
wLen : 믹싱할 최소 단위 샘플 크기(slice length)

```
void mixit(LPSAMPLE lpDest, LPSAMPLE
rgWaveSrc[], int iNumWaves, WORD wLen)
    {
     int i, iSum;
     WORD ctr;
     ctr = 0;
     while(wLen)
     {
      iSum = 128; // 128 is the "normal" value (silence)
for 8bit 11KHz
      for (i=0; i<NumWaves; i++)
        iSum = iSum + *(rgWaveSrc[i]+ctr) - 128;
      PEG((int)0, iSum, (int)255); /
      *lpDest++=iSum;
      ctr++;
      wLen==;
     }
    }
```



[그림 2] 멀티 채널이 데스트네이션 버퍼로(믹싱 모듈의 내부 버퍼중 하나) 믹싱 처리 단위로 믹스하는 그림.



[그림 3] 믹싱 모듈의 내부 버퍼

3. 결론.

단일 시스템 환경과 실시간을 요구하는 분산 환경의 웨이브 스트림 버퍼는 그 주요한 믹서의 기능과 구현 방법은 웨이브 블록의 버퍼 기법을 제외하고 거의 동일하다. 이것은 미리 준비될 수 있는 데이타 블록이 있느냐 없느냐에 기인된 문제로서 실시간의 오디오 인터럽션을 야기시키는 차이만이 있는 것이다.

따라서 본론의 1절에서는, 믹서 모듈을 구성하는 서브 모듈에 대해 간략히 살펴 보았다.

2절에서는 믹싱 모듈에 공급되는 웨이브 블록의 관리에 대하여 살펴 보았는데, 단일 시스템 환경과 분산 환경에 있어서의 각 구현 기법을 코드와 함께 알아 보았다.

3절에서는 새롭게 추가되고 제거되는 채널의 관리 방법을 살펴 보았다.

4절에서는 믹싱 모듈의 기능에 대해 설명하고 내부의 다중 버퍼의 크기와 역할에 대해 살펴 보았다. 또한 믹싱 알고리즘을 한가지 예로써 설명했다.

이후의 연구 방향은 분산 환경에서의 보다 안정적인 웨이브 출력을 보장하는 최적화된 버퍼링 기법에 대해 검증해야 할 것이고, 또한 멀티 테스킹시 웨이브 출력의 인토럽션을 고려하여 프로세서의 부하를 최대한 줄일 수 있는 코드의 최적화에 대해 연구해야 할 것이다.

참고 문헌

[1] Microsoft Corporation. "Multimedia programmer's Worbook" 1991, Microsoft press.

[2] Microsoft Corporation. "Multimedia programmer's Reference" 1991, Microsoft press.

[3] Microsoft Coporation "How WaveMix Works" 1995.