Game Engine Development II

Week3

Hooman Salamat

Audio

There is in souls a sympathy with sounds:
And as the mind is pitch'd the ear is pleased
With melting airs, or martial, brisk or grave;
Some chord in unison with what we hear
Is touch'd within us, and the heart replies.

--William Cowper

Objectives

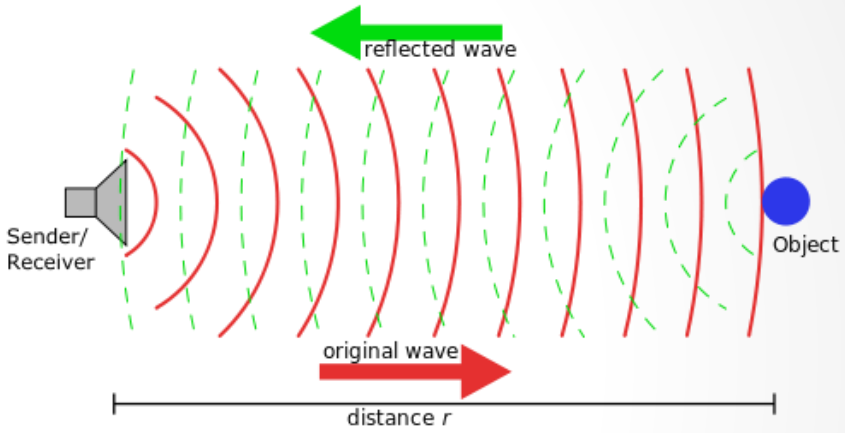
- we will learn how to use XAudio2 and the Windows Media Foundation to load both uncompressed and compressed audio files from the hard drive and how to play them back

Sound

- Sound is an important component of modern computer games, without sounds, games lose a lot of atmospheric detail.
- "Sound" is a vibration that propagates as an audible wave of pressure through a transmission medium such as a gas, liquid or solid.
- Obviously those mechanical waves travel with different velocities as the medium varies, for example sound travels a lot faster through water (1478 m/s) than through air (344 m/s) - which is why sonars work so well in water.

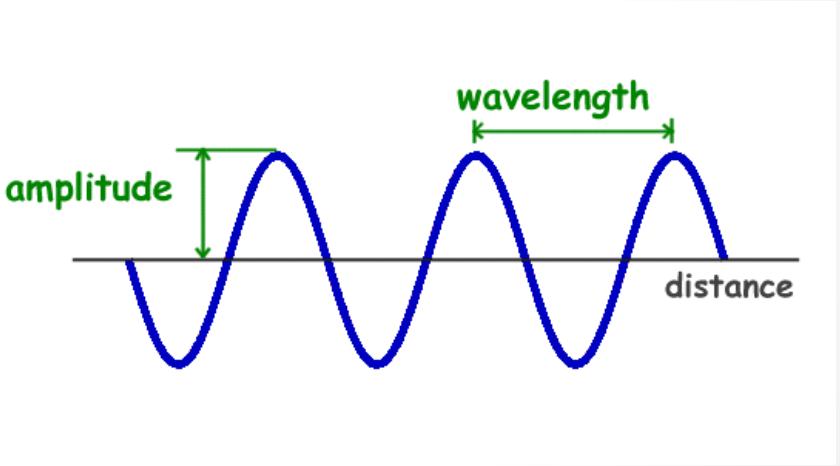
amplitude and its frequency

- Besides its velocity, there are two other parameters to a sound wave: its amplitude and its frequency.
- The amplitude is a measure of how much air volume is moved over a single period of time. Large speakers (and big-mouthed people) move more air, therefore the sound they emit is "stronger" or more "intense".



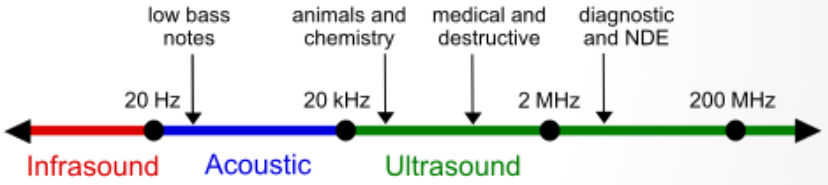
Herz

- A frequency, strictly speaking, is the number of occurrences of a repeating event per unit of time.
- The frequency, or wavelength, is how many complete waves are emitted per second by the sound source.
- The frequency is measured in Hertz (Hz).



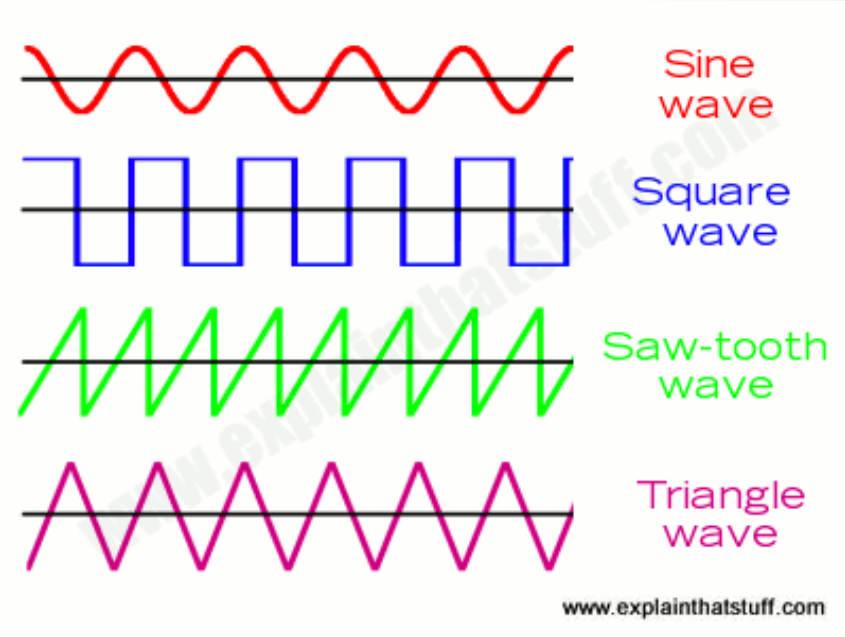
Sound Range

- Most human beings can hear sounds in the range of 20 and 20000 Hz.
- The average male has a voice that ranges from 20 to 2000 Hz, while the voice of an average female ranges from 70 to 3000 Hz.
- there are differences between men and women - men have more bass and women have more treble.



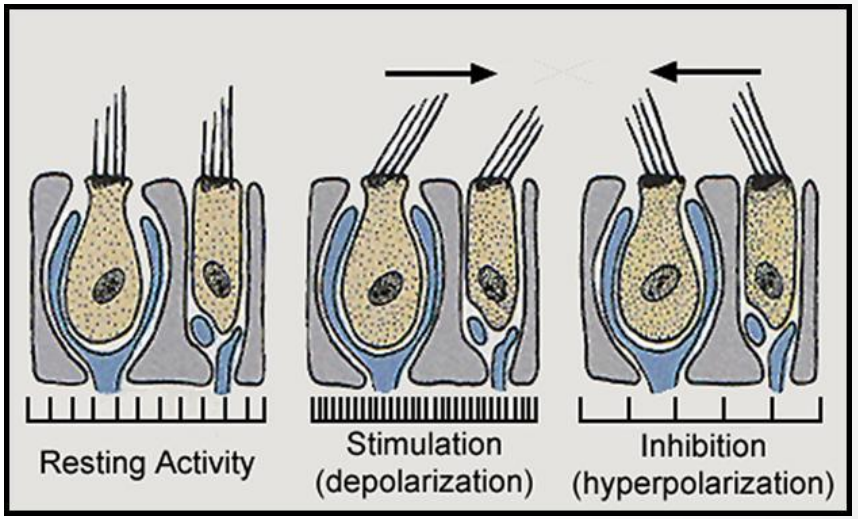
the velocity of sound

- Mathematically speaking, the velocity of sound, v, can be computed using its frequency f, and its amplitude λ,
- $v=f \cdot \lambda$
- sounds with the same frequency and the same amplitude might still sound differently, due to having different wave forms (think of sinus waves or sawtooth waves).



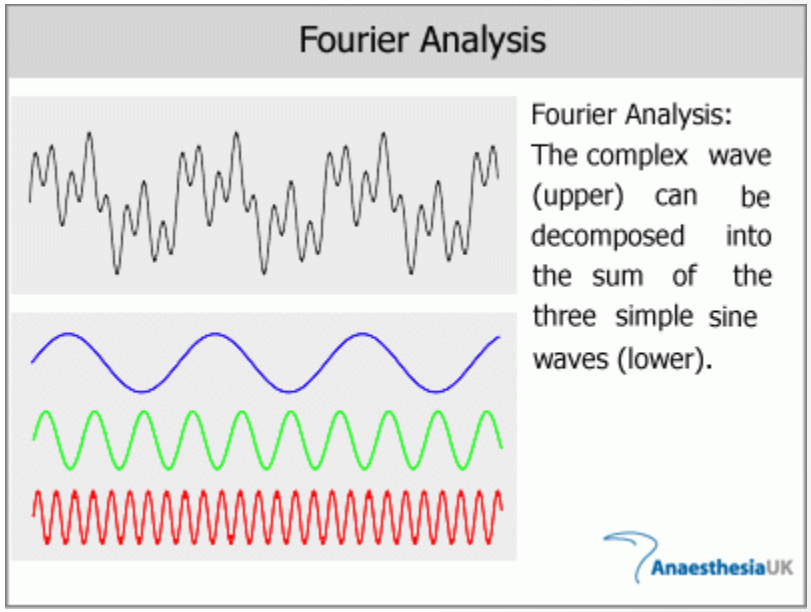
stereocilia

- Our ears have numerous little hair-like structures called *stereocilia*, which are responsible to catch sound waves.
- Each of these cilia can detect sound waves of different frequencies.
- Once a sound wave enters the ear, the cilia resonate and send the according signals to the brain, which then transforms those signals into the perception of sound.



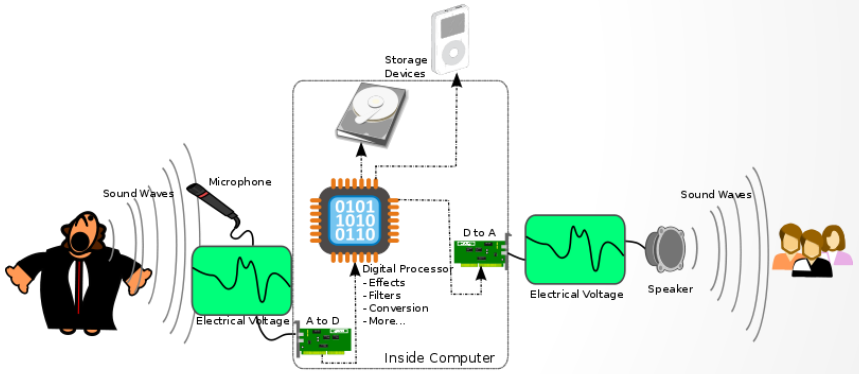
Fourier Analysis

- The form of a single pure tone will always be a sine wave, with an arbitrary frequency and amplitude.
- Mixed thousands of those pure tones together, we get the *spectrum* of a sound.
- The most basic waveform is the sine wave, as all other waveforms can be represented by linear combinations of several sine waves.



Digital versus MIDI

- There are two kinds of sounds that a computer can produce: *digital* and *synthesized*.
- Digital sounds are recordings of sounds, while synthesized sounds are programmed reproductions of sounds based on algorithms.



sample rate

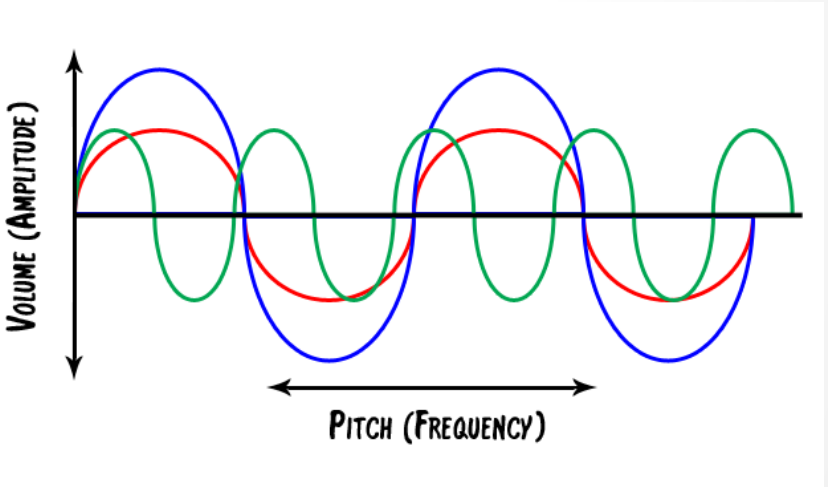
- The number of samples of a sound recorded per second is called the *sample rate*.
- If we want to reproduce a sound, the sample rate must be at least twice the frequency of the original sound.
- For example, if we want to reproduce a human voice, the sound must be sampled at 4000 Hz. The mathematical reasoning behind this is that if we can sample the highest frequency sine wave of a sound, we can sample all the lower ones as well.

amplitude resolution

- The second parameter of a sound, the amplitude, also plays a crucial role when sampling a sound.
- The *amplitude resolution* defines how many different values are available for the amplitude. For example, with a 16-bit resolution, there are 65536 possible values.
- Digital sound is a recording or sampling of sound converted into a digital form from an analog signal.

Synthesized Sound

- Synthesized sound isn't a "real" sound converted into a digital form;
- it is a mathematical reproduction of a sound.
- Playing back a single tone is easy, but real sound is made up of many frequencies, they have undertones, overtones and harmonics.
- To produce full sound, the hardware must be able to play back many "base" sounds simultaneously.



FM & MIDI

- One of the first attempts to create synthesized sounds was the Frequency modulation (FM) synthesis.
- FM synthesis feeds the output of a signal back to itself, therefore modulating the signal and creating harmonics from the original single sine wave.
- About the same time as FM synthesis, a technical standard for music synthesis was introduced, the Musical Instrument Digital Interface (MIDI) standard.
- Instead of digitalizing a sound, a MIDI document describes a sound as key, instruments and special codes:
  - Channel 1: B sharp
  - Channel 2: C flat

wave table synthesis

- A drawback to this format is, that the synthesis is up to the hardware, thus different hardware produces different sounds.
- The huge gain in memory size compared to digital music (a few kilobytes against a few megabytes) often made up for that drawback though.
- To further increase the quality of synthetic sound, a process called wave table synthesis was introduced.
- Basically wave table synthesis works like this: the wave table has a number of real, sampled digital sounds which can be played back at any desired frequency and amplitude by a digital signal processor.
- To take things even further, wave guide synthesis was introduced. By using special hardware, the sound synthesizer can generate a mathematical model of an instrument and then actually play it!

Sound Consideration

- a computer only has a finite number of *sound channels* it can use at the same time, therefore, sometimes, the game has to prioritize sounds, i.e. we have to play back those sounds that are important and ignore the less important ones.
- To be able to do this, our game engine will handle thru *sound events*. A sound event is a map between a game event and a sound (or multiple sounds). The game will not directly call for a sound to be played, but it will rather trigger a sound event.
- From how far away do we want the car crash to be heard? It would be quite useful if we could define a maximum distance for sound to travel - we will call this the *falloff* of a sound event.

Sound Event

```
struct SoundEvent {
unsigned int fallof;
unsigned int priority;
std::vector<SoundFiles*> sounds;
}
```

Sound Event

- The game world has different undergrounds, such as grass and stone, or even caves for the dog to explore.
- The pawsteps will sound differently in each scenario.
- We need to be able to switch between sounds based on game variables:

```
struct SoundEvent {
unsigned int fallof;
unsigned it priority;
std::map<std::string,std::vector<SoundFiles*> > sounds;
}
```

A Brief History of Windows Audio APIs

- MME, WASAPI, DirectSound, WDM/KS, and XAudio2. There are a lot different paths a developer could take. But which one makes the most sense?
- **1991 – Windows Multimedia Extensions (aka MME, aka WinMM)**
- MME was the very first standard audio API for Windows.
- Latency is a problem with MME. Dynamic, near-real time audio (e.g., game event sounds, software synthesizers, etc.) is a bit harder to do in a timely fashion. Anything that occurs 10ms later than the brains thinks it should, is perceived to be out of sync.
- MME can only handle a maximum of 96kHz and 16-bit audio.
- On Windows, MME is built on top of Core Audio.
- MME is still around.

DirectSound

- **1995 – DirectSound (aka DirectX Audio)**
- Windows 95. Besides giving us the infamous "Start" button and the music video for Weezer's "Buddy Holly", Windows 95 brought with it DirectX.
- DirectX was core to Microsoft's strategy for winning over game developers.
- DirectX was the umbrella name given to a collection of COM-based multimedia APIs, which included DirectSound.
- DirectSound distinguished itself from MME by providing things like on the fly sample rate conversion, effects, multi-stream mixing, alternate buffering strategies, and hardware acceleration
- Because DirectSound was implemented using VxDs, which were kernel mode drivers, it could work extremely close to the hardware.
- DirectX also brought pluggable, software based audio effects (DX effects) and instruments (DXi Instruments) to the platform.

Windows Driver Model

- **1998 – Windows Driver Model / Kernel Streaming (aka WDM/KS)**
- Windows NT had a very different driver model. This meant if a hardware vendor wanted to support both Windows NT and Windows 95, they needed to write two completely independent drivers – drivers for NT built using the the Windows NT Driver Model and VxDs for everything else.
- Microsoft decided to fix this problem and the Windows Driver Model (WDM) was born.
- KMixer was a kernel mode component responsible for mixing all of the system audio together. KMixer introduced latency!
- Cakewalk, a Boston-based company famous for their DAW software, began supporting a technique called WDM/KS.
- The KS stands for Kernel Streaming.
- Kernel streaming isn't an official audio API, per se. It's something a WDM audio driver supports as part of its infrastructure. The WDM/KS technique involves talking directly to the hardware's streaming driver, bypassing KMixer entirely.

XACT

- **2002 - Cross-platform Audio Creation Tool** (**XACT**)
- The "X" in Xbox comes from DirectX
- New APIs appeared that started with the letter "X", such as XInput and XACT3.
- XACT is an audio programming library and engine released by Microsoft as part of the DirectX SDK.
- It is a high-level audio library for authoring/playing audio that is written to use *Xaudio* on the *Xbox*, DirectSound on Windows XP, and the new audio stack on Windows Vista and Windows 7.
- Xaudio is an Xbox-only API designed for optimal digital signal processing.
- XACT also includes X3DAudio, a spatialization helper library available on both platforms, Windows and the Xbox.
- XACT was originally developed for Xbox development, and was later modified to work for Microsoft Windows development as well.
- The **XACT Audio Authoring Tool** is a companion application used to organize audio assets into *wave banks* (single files containing multiple WAV files) and *sound banks* (single files containing instructions for playing the WAV files in wave banks).
- Organization of sounds
  - Multiple audio files can be grouped together into Wave Banks (XWB extension)
  - Cues and settings can be bundled with the Waves in Sound Banks (XSB extension)

Windows Core Audio

- **2007 – Windows Core Audio**
- Windows Core Audio, not to be confused with OSX's similarly named Core Audio, was a complete redesign in the way audio is handled on Windows.
- KMixer was killed and buried.
- Most of the audio components were moved from kernel land to user land, which had an impact on application stability.
- Core Audio is actually 4 APIs in one – MMDevice, WASAPI, DeviceTopology, and EndpointVolume.
- The API for interacting with all of the software components that exist in the audio path is the DeviceTopology API.
- For interacting with volume control on the device itself, there's the EndpointVolume API.
- And then there's the audio session API – WASAPI. WASAPI is the workhorse API.
- Core Audio still serves as the foundation for platform audio for Windows 10.

XAudio2

- **2008 – XAudio2**
- was declared the official successor to DirectSound
- XAudio2 offers a number of features missing from DirectSound, including support for compressed formats like xWMA and ADPCM, as well as built-in, sophisticated DSP effects.
- XAudio2 provides a signal processing and mixing foundation for games. For example, it provides a flexible and powerful Digital Signal Processing (DSP) framework, with which, for example, cat meows can be turned into scary monster sounds.
- XAudio2 also facilitates combining different *voices* into single audio streams, called *submixing*, to, for example, create an engine sound made up of composite parts, all of which are playing simultaneously.
- The XAudio2 API is also "non-blocking", meaning that the game can safely make a set of method calls to XAudio2 at any time.

PCM vs DPCM vs ADPCM-Difference

- The short form of the Pulse Code Modulation is **PCM**. In PCM, the analog speech waveform is sampled and converted directly into a multibit digital code by an Analog to Digital converter. The digital code is stored in the memory and which is later re-called for the playback.
- The short form of Delta Pulse Code Modulation is DPCM. In **DPCM**, a multi-bit difference value is stored. A bipolar D/A converter is used for playback to convert the successive difference values to an analog waveform.
- In adaptive delta modulation, step size is chosen in accordance with message signal sampled value to overcome slope overload error and hunting.

ASIO

- There are plenty of third party audio APIs available for Windows that weren't invented by Microsoft.
- Some of them, like GSIF used by TASCAM's (formerly Nemesys) GigaStudio, are tied to specific hardware.
- Some of them, like PortAudio and JUCE (more than just an audio API), are open-source wrappers around platform specific APIs.
- Some of them like OpenAL are just specifications that have yet to gain widespread adoption.
- Steinberg, the same forward-thinking company that gave us VSTs and Cubase, introduced us to ASIO all the way back in 1997. ASIO was originally a pro-audio grade driver specification for Windows.

Conclusion

- games should go with XAudio2
- pro-audio should go with ASIO and/or Core Audio
- and everybody else should probably go with MME.

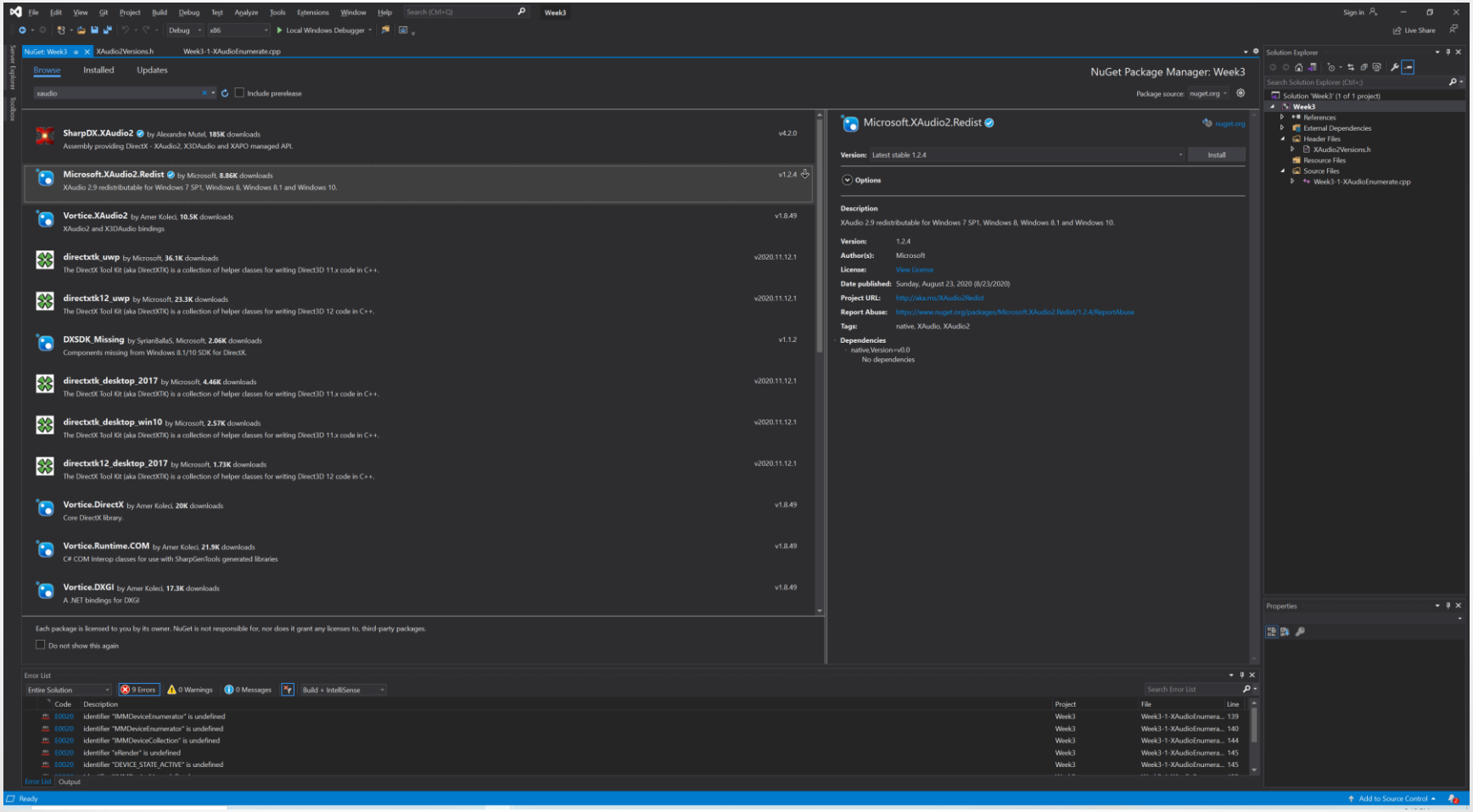
Xaudio2 versions

- **XAudio 2.7 and earlier (Windows 7)**
  - All previous versions of XAudio2 for use in apps have been provided as redistributable DLLs in the DirectX SDK.
- **XAudio 2.8 (Windows 8.x)**
  - XAudio2 version 2.8 ships as a system component in Windows 8, XAUDIO2_8.DLL. It is available "inbox" and does not require redistribution with an app.
- **XAudio 2.9 (Windows 10 and redistributable for Windows 7 and Windows 8.x)**
  - XAudio2 version 2.9 ships as part of Windows 10, XAUDIO2_9.DLL, alongside XAudio 2.8 to support older applications.

Redistributable version of XAudio 2.9

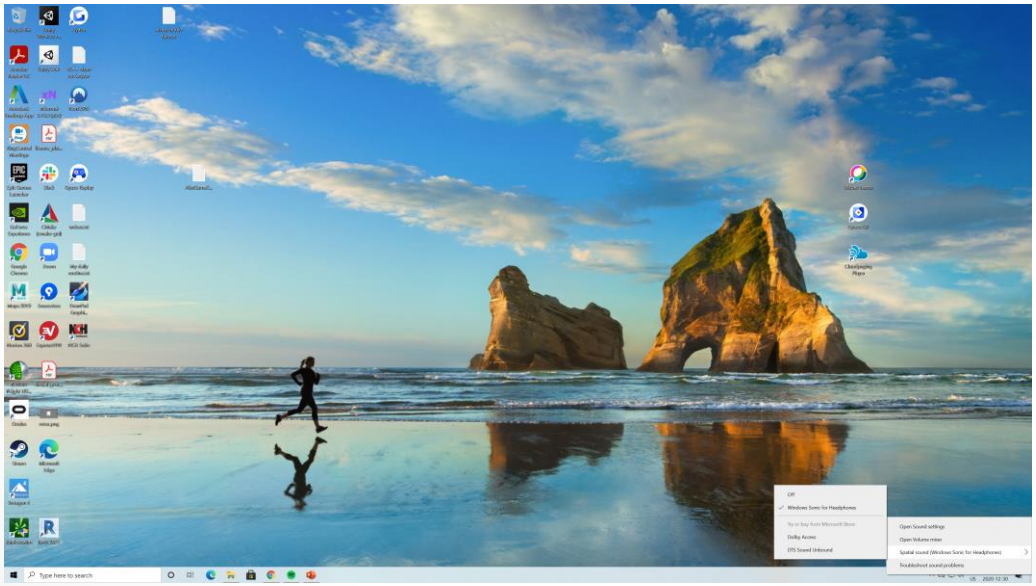
- A version of XAudio 2.9 is available as a NuGet package. Developers can redistribute this version of XAudio 2.9 with their apps. This allows an app to use XAudio 2.9 on older versions of Windows that do no not include XAudio 2.9 as part of the operating system image.
- The XAudio 2.9 NuGet package (*Microsoft.XAudio2.Redist.*.nupkg*) includes a 32-bit and a 64-bit version of a DLL that implements the XAudio 2.9 API.
- The DLL is called XAUDIO2_9REDIST.DLL. This DLL will work on Windows 7 SP1, Windows 8, Windows 8.1 and Windows 10.
- The easiest way to install the NuGet package is to use the NuGet Package Manager in Microsoft Visual Studio. If you do this, your Visual Studio project file will be automatically updated to include *Microsoft.XAudio2.Redist.targets*
- The *.targets* file will also make your .DLL or .EXE link with XAUDIO2REDIST.LIB and XAPOBASEREDIST.LIB.
- The library XAPOBASEREDIST.LIB is only needed if you intend to impement a custom XAudio Processing Object (XAPO)

Redistributable version of XAudio 2.9



Spatial sound in newer versions of Windows 10

- Starting with the Windows 10 1903 update, XAudio 2.9 automatically uses virtual surround sound, if certain conditions are satisfied.
- The user can enable a spatial sound format by right-clicking on the speaker icon in the Windows system tray.



Game Bar

- XAudio 2.9 will only use the user's selected spatial sound format if the process that is using the XAudio2 API is recognized as a game by the Windows Game Bar.
- During development, it is possible that the process is not yet recognized as a game by the Game Bar.
- To change this, use the Win+G keyboard short-cut to bring up the Game Bar while the game is running. Then click on the "Settings" icon and check the checkbox that says, "Remember that this is a game".

The XAudio2 Engine

- To initialize XAudio2, as with all things DirectX related, a pointer to an interface of an IXAudio2 object is required.
- With the IXAudio2 interface it is possible to enumerate the available audio devices, to configure global API properties, to create voices, and to monitor performance.
- The interface can be used to create a master voice.
- A mastering voice is used to represent the actual audio output device.
- Once a master voice is created, it can be used to create sound effects, bind them to the master voice and play them back.

XAudio2Enumerate

- This sample shows how to enumerate audio devices and initialize an XAudio2 mastering voice for a specific device.

```
UINT32 flags = 0;
HRESULT hr = CoInitializeEx(nullptr,
COINIT_MULTITHREADED);
ComPtr<IXAudio2> pXAudio2;
hr = XAudio2Create(pXAudio2.GetAddressOf(), flags);
std::vector<AudioDevice> list;
hr = EnumerateAudio(pXAudio2.Get(), list);
```

Enumerate audio end-points

```
HRESULT EnumerateAudio(_In_ IXAudio2* pXaudio2, _Inout_ std::vector<AudioDevice>& list)
{
RoInitializeWrapper initialize(RO_INIT_MULTITHREADED);
    HRESULT hr = initialize;
Microsoft::WRL::ComPtr<IDeviceInformationStatics> diFactory;
    hr = ABI::Windows::Foundation::GetActivationFactory(HStringReference(RuntimeClass_Windows_Devices_Enumeration_DeviceInformation).Get(), &diFactory);
ComPtr<IAsyncOperation<DeviceInformationCollection*>> operation;
    hr = diFactory->FindAllAsyncDeviceClass(DeviceClass_AudioRender, operation.GetAddressOf());
```

DeviceInformation

```
ComPtr<IVectorView<DeviceInformation*>> devices;
    operation->GetResults(devices.GetAddressOf());

    unsigned int count = 0;
    hr = devices->get_Size(&count);
    if (FAILED(hr))
        return hr;

    if (!count)
        return S_FALSE;

    for (unsigned int j = 0; j < count; ++j)
    {
        ComPtr<IDeviceInformation> deviceInfo;
        hr = devices->GetAt(j, deviceInfo.GetAddressOf());
        if (SUCCEEDED(hr))
        {
            HString id;
            deviceInfo->get_Id(id.GetAddressOf());

            HString name;
            deviceInfo->get_Name(name.GetAddressOf());

            AudioDevice device;
            device.deviceId = id.GetRawBuffer(nullptr);
            device.description = name.GetRawBuffer(nullptr);
            list.emplace_back(device);
        }
    }
```

Print Out Audio Devices

```
UINT32 devcount = 0;
    UINT32 devindex = -1;
    for (auto it = list.cbegin(); it != list.cend(); ++it,
++devcount)
    {
        wprintf(L"\nDevice %u\n\tID = \"%s\"\n\tDescription =
\"%s\"\n",
            devcount,
            it->deviceId.c_str(),
            it->description.c_str());

        // Simple selection criteria of just picking the first one
        if (devindex == -1)
        {
            devindex = devcount;
        }
    }
```

Example Output

- Device 0
- ID = "\\?\SWD#MMDEVAPI#{0.0.0.00000000}.{b90a659f-1d51-43e0-9561-fe1ccb670493}#{e6327cad-dcec-4949-ae8a-991e976a79d2}"
- Description = "EPSON PJ (NVIDIA High Definition Audio)"

- Device 1
- ID = "\\?\SWD#MMDEVAPI#{0.0.0.00000000}.{c9defaf4-b353-421e-856b-41deb9efae76}#{e6327cad-dcec-4949-ae8a-991e976a79d2}"
- Description = "Realtek Digital Output (Realtek(R) Audio)"

- Device 2
- ID = "\\?\SWD#MMDEVAPI#{0.0.0.00000000}.{2ac7029f-36e8-4e3f-a330-5948c9db1ddb}#{e6327cad-dcec-4949-ae8a-991e976a79d2}"
- Description = "Headphones (2- Rift S)"

- Device 3
- ID = "\\?\SWD#MMDEVAPI#{0.0.0.00000000}.{af5b8d96-83da-4a80-bcdf-0182e6de6ae2}#{e6327cad-dcec-4949-ae8a-991e976a79d2}"
- Description = "Headset Earphone (Logitech H570e Stereo)"

- Device 4
- ID = "\\?\SWD#MMDEVAPI#{0.0.0.00000000}.{1dcd31a2-e939-4840-af37-d9098ba658f9}#{e6327cad-dcec-4949-ae8a-991e976a79d2}"
- Description = "U32R59x (NVIDIA High Definition Audio)"

- Device 5
- ID = "\\?\SWD#MMDEVAPI#{0.0.0.00000000}.{0aeeabd7-d305-4abb-a3af-3021150eda5b}#{e6327cad-dcec-4949-ae8a-991e976a79d2}"
- Description = "Headphones (Oculus Virtual Audio Device)"

Play WAV files

- Initialize the XAudio2 engine by calling the XAudio2Create method. At this point, you can set some basic run-time parameters, and set the notification callback.
- Create a mastering voice using the IXAudio2::CreateMasteringVoice method. This method controls the final mix format used for all audio processing within the application.
- Create and play three WAV files.
  - Locate the WAV file to load.
  - Read in the WAV file and sample data using a sample helper class.
  - Create a source voice based on the format of the loaded WAV file. By default, the source voice will be linked to the first created mastering voice, so you don't need a voice send list.
  - To submit data to the source voice, create an XAUDIO2_BUFFER structure, specifying some play parameters.
  - Submit data to the source voice by means of the function IXAudio2SourceVoice::SubmitSourceBuffer. Note that the sample data is not duplicated by XAudio2, so the pAudioData buffer must remain available for the duration of play.
  - Start playback of the source voice, and perform a simple loop to detect when the playback has completed.
  - Clean up the source voice and associated sample data.
- Clean up by releasing the XAudio2 engine.

Basic Sound

```
    //
    // Play a PCM wave file
    //
    wprintf( L"Playing mono WAV PCM file..." );
    if( FAILED( hr = PlayWave( pXAudio2.Get(), L"Media\\Wavs\\MusicMono.wav" ) ) )
    {
        wprintf( L"Failed creating source voice: %#X\n", hr );
        pXAudio2.Reset();
        CoUninitialize();
        return 0;
    }

wprintf( L"\nFinished playing\n" );

    // All XAudio2 interfaces are released when the engine is destroyed, but being tidy
    pMasteringVoice->DestroyVoice();

    pXAudio2.Reset();


    CoUninitialize();
```

What Is a WMA File?

- WMA is an acronym for **Windows Media Audio**
- Initially created in 1999, Microsoft designed WMA to combat MP3 and Apple's AAC compression methods.
- Since then, WMA has expanded from its initial lossy format into a wide range of sub-formats including low-bandwidth voice audio to lossless multi-channel surround sound.
- When compared to MP3 format, WMA maintains a higher level of quality at a lower bitrate, especially when comparing bitrates less than 64 kbps.
- Because WMA is a proprietary format (Microsoft), very few programs support it compared to the widely used MP3. If you plan to open your WMA files on anything but Windows, you'll either have to download a third-party application or convert it into a different format.

Microsoft Media Foundation

- Microsoft Media Foundation enables the development of applications and components for using digital media
- You have to add Media Foundation Headers and Libraries
- Add mfuuid.lib to Linker --> Input --> Additional Dependencies
- Streaming from a media file, using Media Foundation to decompress the data
- We use Media Foundation to decode a media audio file (which could be compressed with any number of codecs) and streams it through an XAudio2 voice.
- This technique is most useful for XAudio 2.8 on Windows 8.x which only supports PCM and ADPCM, and not more aggressive lossy compressed schemes which are supported by Media Foundation

Use MF to Read WMA

```
// Start up Media Foundation
hr = MFStartup(MF_VERSION);
// Create MF reader for our media file
SourceReaderContext readerContext;
ComPtr<IMFSourceReader> reader;
WAVEFORMATEX wfx;
hr = CreateMFReader(mediaFile, &readerContext, reader.GetAddressOf(), &wfx, sizeof(wfx));
// Create the source voice
StreamingVoiceContext voiceContext;
IXAudio2SourceVoice* pSourceVoice;
hr = pXAudio2->CreateSourceVoice(&pSourceVoice, &wfx, 0, 1.0f, &voiceContext);
pSourceVoice->Start(0, 0);
hr = reader->ReadSample(MF_SOURCE_READER_FIRST_AUDIO_STREAM, 0, nullptr, nullptr, nullptr, nullptr);
ComPtr<IMFMediaBuffer> mediaBuffer;
hr = readerContext.sample->ConvertToContiguousBuffer(mediaBuffer.GetAddressOf());
```

XWB

- A file with the XWB file extension is an **XACT Wave Bank file**, a format that holds a collection of sound files for use in video games. They may include both sound effects and background music. The true source program for XWB files is the Microsoft Cross-Platform Audio Creation Tool (XACT), part of the Microsoft XNA Game Studio program.

XWBTool

- The XWBTool is a command-line utility for building XACT-style wave banks for use with the *DirectXTK for Audio* Wave Bank class.
- Open a Command Prompt, and change to the directory containing XWBTool.exe (i.e. ...\DirectXTK\XWBTool\bin\Release)
- Enter the following command-line after changing to the appropriate directory:
- xwbtool -o wavebank.xwb Sound.wav Explosion.wav Music.wav The file *wavebank.xwb* is generated from the three input .wav files.
- xwbtool -o test.xwb C:\Windows\Media*.wav

XAudio2WaveBank

- XACT wave banks are a binary file containing 1 or more .WAV files packaged together along with metadata information.
- We could play in-memory audio using an XACT-style wave bank.
- **WAVFileReader**.h/.cpp module is used to load a .WAV file for playback with XAudio2. It optionally returns loop-point information as well as supporting the standard PCM, MS-ADPCM, and WAVEFORMATEXTENSIBLE formats supported by XAudio2.
- **WaveBankReader**.h/.cpp module which is used by the XAudio2WaveBank samples. It is used to parse XACT3-style wave banks.
- The **XBWTool** command-line tool is a simple way to build XACT-style wave banks

XAudio2AsyncStream

- how to play streaming audio using asynchronous file I/O and the XAudio2 API
- First, you must decide on two important values:
  - The amount of data to request from the file system at a given time.
  - The number of data buffers to queue up for playback.
- Next, create the XAudio2 engine and a playback voice
- After everything is set up, the streaming process is relatively straightforward:
  - Call ReadFile to start an asynchronous read.
  - When the read finishes, check the number of buffers in the playback queue.
  - As soon as the number of buffers in the playback queue falls below your predetermined maximum value, submit the data you received from the last asynchronous read.
  - Call ReadFile again to start another read.

Example

- `HRESULT hr = CoInitializeEx(nullptr, COINIT_MULTITHREADED);`
- `ComPtr<IXAudio2> pXAudio2;`
- `hr = XAudio2Create(pXAudio2.GetAddressOf(), flags);`
- `IXAudio2MasteringVoice* pMasteringVoice = nullptr;`

- `hr = pXAudio2->CreateMasteringVoice(&pMasteringVoice);`
- `hr = FindMediaFileCch(wavebank, MAX_PATH, L"Media\\Banks\\wavebank.xwb");`

Extract wavebank data

- Extract wavebank data (entries, formats, offsets, and sizes)
- Note we are using Wave Banks to get sector-aligned streaming data to allow us to use async unbuffered I/O. Raw .WAV files do not meet these requirements.

```
WaveBankReader wb;
hr = wb.Open(wavebank);
if (!wb.IsStreamingBank())
    {
        pXAudio2.Reset();
        CoUninitialize();
        return 0;
    }
```

WAVEFORMATEX

```
for (DWORD i = 0; i < wb.Count(); ++i)
{
 wprintf(L"Now playing wave entry %u", i);
// Get the info we need to play back this wave (need enough space for PCM, ADPCM, and xWMA formats)
char formatBuff[64];
WAVEFORMATEX* wfx = reinterpret_cast<WAVEFORMATEX*>(&formatBuff);
hr = wb.GetFormat(i, wfx, 64);
WaveBankReader::Metadata metadata;
hr = wb.GetMetadata(i, metadata);
// Create an XAudio2 voice to stream this wave
StreamingVoiceContext voiceContext;
IXAudio2SourceVoice* pSourceVoice;
hr = pXAudio2->CreateSourceVoice(&pSourceVoice, wfx, 0, 1.0f, &voiceContext);
pSourceVoice->Start(0, 0);
```

Overlapped Structure

```
// Create an overlapped structure and buffers to handle
the async I/O
            //
OVERLAPPED ovlCurrentRequest = { 0 };
std::unique_ptr<uint8_t[]> buffers[MAX_BUFFER_COUNT];
            for (size_t j = 0; j < MAX_BUFFER_COUNT; ++j)
            {
                buffers[j].reset(new
uint8_t[STREAMING_BUFFER_SIZE]);
            }
            DWORD currentDiskReadBuffer = 0;
            DWORD currentPosition = 0;
HANDLE async = wb.GetAsyncHandle();
BOOL result = GetOverlappedResultEx(async,
&ovlCurrentRequest, &cb, 0, FALSE);
```

Submit the audio

- At this point we have a buffer full of audio and enough room to submit it, so
- let's submit it and get another read request going.

```
XAUDIO2_VOICE_STATE state;
                for (;; )
                {
                    pSourceVoice->GetState(&state);
                    if (state.BuffersQueued < MAX_BUFFER_COUNT - 1)
                        break;

                    WaitForSingleObject(voiceContext.hBufferEndEvent, INFINITE);
                }
XAUDIO2_BUFFER buf = { 0 };
                buf.AudioBytes = cbValid;
                buf.pAudioData = buffers[currentDiskReadBuffer].get();
                if (currentPosition >= metadata.lengthBytes)
                    buf.Flags = XAUDIO2_END_OF_STREAM;

                pSourceVoice->SubmitSourceBuffer(&buf);

                currentDiskReadBuffer++;
                currentDiskReadBuffer %= MAX_BUFFER_COUNT;
```