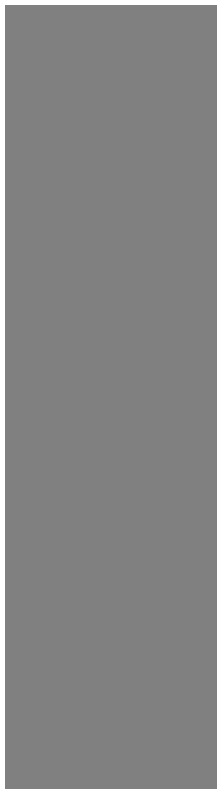
A3D 3.0 Software Development Kit User's Guide



AUREAL

Disclaimer

Copyright

Trademarks

Document Number: DO3011-030100

A3D version 3.11

Contents

Chapter 1: The A3D Developer Kit

Welcome to A3D 3.0, the powerful next generation sound engine – and the next level of industry-leading 3D audio. The Aureal A3D 3.0 Software Development Kit (SDK) includes everything necessary to integrate Aureal's complete sound engine including 3D positional audio, reverb, MP3 support, and other features into your next software application. Included in the kit are the following:

- A3D 3.0 API Reference Guide
- A3D 3.0 User's Guide
- A3D 3.0 Platform and Resource Manager Guide
- Source code and sample A3D 3.0 applications
- Debug Viewer GL
- Code support files
- Release notes
- A3D drivers
- A3D logos
- Aureal Wavetracing™ technology demo
- Minerva test application

The A3D 3.0 API Reference Guide

This is the heart of the SDK. The A3D 3.0 API Reference Guide contains a complete list of A3D 3.0 functions, their descriptions, structures, and error codes. This is where you will find the actual code that makes A3D 3.0 work. As new features are added to A3D 3.0, the release notes will be updated and you will find the actual code in the reference manual. An index of A3D 3.0 functions is included for easy reference.

The A3D 3.0 User's Guide

This document contains all the information you need (aside from the actual API reference manual, which details specific code) to implement A3D 3.0 into your application. This document is divided into the following chapters:

- **The A3D Developer Kit**: description of items included in the SDK.
- **Welcome to A3D 3.0**: introduction to A3D and its key features.
- **Starting up**: what you need to code, compile and run your application with A3D.
- **Debug Viewer GL**: information on this helpful debug tool that enables you to "see" what you are hearing.
- **Tweaking Your A3D 3.0 Code**: tips and tricks for writing the fastest and best sounding A3D code.
- **Releasing Your Application**: information about the steps necessary to ensure A3D 3.0 works on your customers' computers.

A3D 3.0 Platform and Resource Manager Guide

The A3D 3.0 Platform and Resource Manager Guide contains information about the new resource manager featured in A3D 3.0. Among other things it covers DS3D, A2D, and a platform matrix of the different features available under A3D 3.0 using different audio hardware setups.

Source Code and Sample A3D 3.0 Applications

Also known as tutorials, we have included some very simple applications that utilize A3D 3.0, along with the source code used to compile them. This enables you to see what complete A3D 3.0 code looks like up front. Starting from the sample code, you can make changes and recompile to experiment with settings and other parts of the code.

Code Support Files

Included with the SDK are the necessary static library files (with source) and header files to compile with your application.

Release Notes

This is where you will find the latest information regarding the status of the A3D 3.0 SDK and its contents.

A3D Drivers

Included with the SDK are the latest available drivers (debug and release) that A3D uses to interface with your sound card. As these drivers are updated, debug and release versions will be posted on our A3D Developer website. Release versions will be posted on the main A3D website.

A3D Logos

Included with the SDK are various versions (MAC, PC, PhotoShop, Illustrator, etc.) of the Aureal and A3D logos. Although A3D 3.0 works on any audio hardware, it works best with A3D-enabled hardware. Let your customers know that you support their hardware to the fullest by displaying this logo on the app packaging, advertisements, and your web site.

Aureal Wavetracing™ Technology Demo

Optimally supported by Vortex2-based A3D cards, this demo works on any A3D sound card. It very clearly demonstrates the benefits of Aureal Wavetracing technology, a key feature of A3D 3.0. Move the listener, sound sources, and walls around to dynamically affect the A3D experience. The Wavetrcing demo is not included in the downloadable SDK, though it can be downloaded from the A3D Developer website.

Minerva Test Application

Minerva is a testing and profiling tool for audio drivers and cards. Minerva tests for compatibility with, and features of, DirectSound, DirectSound3D, and A3D. Minerva is not included in the downloadable SDK, though it can be downloaded from the A3D Developer website.

Technical Support

For more information or technical assistance please feel free to contact us.

Aureal Developers Page: http://developer.a3d.com

Aureal Email Contacts:

Technical Contact:	devsupport@A3D.com
(API coding assistance, bug reports, etc)	
Marketing Contact:	evangelist@A3D.com
(Hardware requests, co-marketing issue , general inquries, etc.)	

Chapter 2: Welcome to A3D 3.0

What's New and Different About A3D 3.0?

- Full-featured, robust sound engine
  A3D 3.0 is a complete sound engine guaranteed to work on any sound card. It features standardized 3D positional audio, Wavetracing technology (geometric reverb, reflections, and occlusions), reverb functionality compatible with both the I3DL2 and EAX™ standards, an improved streaming resource manager, DS3D hardware support, A2D rendering engine, an improved distance model, complete Windows 9x sound card compatibility, and a host of new features listed below.
- Dolby Digital™ support
  A3D 3.0 is currently the only sound engine which features support for Dolby's AC-3 format. Complex, multi-channel music tracks can now be easily and seamlessly integrated into games and applications. The A3D 3.0 engine will detect the abilities of the host system and utilize the best possible playback mechanism, including hardware AC-3 decoding if available.
- MP3 support
  A3D 3.0 now supports one of the most popular audio formats in existence, Fraunhofer's MP3. Since MP3 files can easily occupy 1/10th the space of a WAV file with minimal difference in quality, developers can use MP3 to increase the number of sound effects in the game, or just reduce the space needed to store existing audio files, saving on media costs. The A3D 3.0 engine even features its own built-in MP3 decoder to ensure complete compatibility.
- Streaming Audio support
  Up until now, playback of large audio files (such as the popular MP3 format) either involved extra programming to implement, or the use of incredible amounts of memory to load the files in their entirety. But with A3D 3.0, the task is simplified; the audio engine transparently streams audio from files, saving both development time and memory. Streaming support has been implemented for all supported audio formats: WAV, MP3, and AC-3.

- Volumetric sound sources
  Ever tried to simulate a row of bleachers in a crowded stadium or a tall, rushing waterfall, all with just a point source? Volumetric sound sources are the answer – they add depth and volume to any source with little effort and no overhead.
- Manual reflections
  A3D 3.0 extends the existing Wavetracing technology by allowing the developer to place arbitrary reflections anywhere in 3D space. This opens the door for creating a host of new effects (such as a specific echo) and enhanced audio environments.
- Reverb
  A3D 3.0 now features A3Dverb, a reverb engine designed to enhance the environmental effects created by the Wavetracing engine. The engine features geometric reverb, which instantly adds reverb to any game or application that utilizes Wavetracing, as well as A3Dverb presets which can be set for each room or area and then customized. A3Dverb exposes the controls to tweak volume, decay time, and damping. The engine also supports both the IASIG's I3DL2 and Creative Labs' EAX™ specifications, and will work seamlessly with any sound card hardware that supports either of these or Aureal's own A3Dverb engine.
- Property Set Extensibility
  DirectSound property sets are accessible through the A3D interface. Programmers who know the specific features of a sound card can use the property set interface to "talk" directly to the sound card's driver.

Introduction to New Features

As explained above, A3D 3.0 introduces several key new features to the audio engine. These are explained in more depth here.

Reverb

Aureal's new A3Dverb builds upon, and extends, the existing Wavetracing technology by adding reverb. A3D 3.0 supports both I3DL2 and EAX 1.0 specifications, and will work seamlessly with any reverb-enabled hardware on the system.

In its simplest form, reverb is easy to implement through presets. Reverb presets are a quick way to add reverb to 3D sound sources in situations where the reverb amount will not be changing greatly – for example, in games where the player does not move among environments that are significantly different, or in titles where reverb will be used solely to accentuate the audio sources, instead of rendering a "realistic" reverb setting. Customized reverb is also available, as the underlying reverb controls are exposed to the developer.

The first thing to do is to enable reverb in your call to **IA3d5::InitEx**, by adding the **A3D_REVERB** flag to the features field. Then create a new **IA3dReverb** object using **IA3d5::NewReverb**, select a reverb preset with **IA3dReverb::SetReverbPreset**, and call **IA3d5::BindReverb**. The next time **IA3d5::Flush** is called, all of your sources will have reverberation. The reverb amount for each source can be specified individually by calling **IA3dSource2::SetReverbMix**, or even disabled by setting the reverb mix to 0.

The reverb preset effects are very customizable, and can be tweaked to perfection with controls for volume, decay time, and damping. If there is no preset to match the needs of a particular space, the presets can be bypassed completely, allowing all reverb properties to be set manually. The combination of presets and custom reverbs is an extremely flexible solution, allowing for the best reverb for each unique application.

Multiple reverb objects can exist at the same time, but only one can be bound at any time. To stop the current reverb effect, call **IA3d5::BindSource** with a NULL pointer. Alternately, bind a different reverb to stop the current reverb and start the new one. Whichever you choose, the old reverb will stop the next time **IA3d5::Flush** is called.

Any existing reverb objects can be tuned and changed at any time. If the reverb object is not bound then its changes will not take effect until it is. If the reverb object is currently bound (the reverb which is current being applied), then any changes to it won't take effect until the next call to **IA3d5::Flush**. These changes can be minor adjustments to the custom properties or even a new preset.

Implementing Reverb

1. First, call **IA3d5::NewReverb** to create an IA3dReverb object.

2. To specify a reverb, choose a preset and call **IA3dReverb::SetReverbPreset**. Or, create an A3DREVERB_PROPERTIES structure, set the desired values, and call **IA3dReverb:: SetAllProperties.**

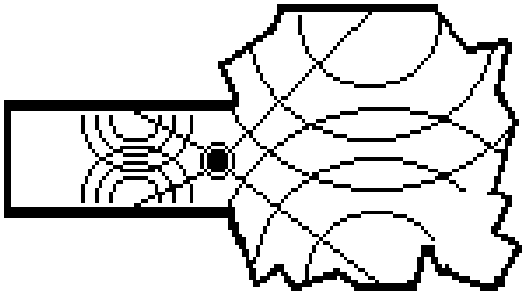
3. Call **IA3d5::BindReverb** to enable the current reverb object.

Repeat the previous two steps whenever a new reverb is needed. The old reverb will immediately and smoothly fade into the new reverb.

See the included “Reverb” sample application for example code.

Geometric Reverb

The great benefit of reverb is the intensity it adds to the audio environment. It’s one thing to hear an explosion – it’s another to hear it echo around the walls of the room you’re watching it from. Mixing reverb with Wavetracing creates an intense audio experience; the goal of geometric reverb is to make that experience powerful, while still being very easy to create.



A source generating two different types of reverb, depending on the listener’s location.

FIGURE 1.

Geometric Reverb is the automated process by which the A3D 3.0 engine picks a reverb based on relevant audio geometry. The idea is to add reverb to an A3D 3.0 title with as little effort as possible. By automating reverb selection, the process of individually tagging rooms and areas is avoided while the end result is maintained. Bursting out of narrow corridors into large rooms generates a sudden expansion of the reverb, while moving in and out of rooms, corridors, and the outdoors allows for more gradual changes in reverb. All these effects are achieved at the same time through the use of geometric reverb.

Enabling Geometric Reverb

Enabling geometric reverb depends on which version of A3D is being used. In all cases, it is critical that geometry is passed into the API by the application. If Wavetracing is not used, then geometric reverb will have no polygons to look at; in this case, it would be necessary to use reverb presets.

For interfaces 4 (IA3d4) and earlier, geometric reverb will automatically be enabled for any title that uses reflections. Calling **IA3dX::InitEx** with the **A3D_1ST_REFLECTIONS** flag enables reflections. Aureal Vortex2-based cards are currently the only audio hardware that can render reflections. Note that this only applies to older A3D 2.0 titles that use an earlier interface then **IA3d5**.

For interface 5 (IA3d5), geometry has to be requested. Calling **IA3d5::InitEx** with the **A3D_GEOMETRIC_REVERB** flag does this. Additionally, the effect can be tweaked with calls to **A3dGeom2::GeomReverbParam**.

Initializing and Tweaking the Geometric Reverb Effect

Tweaking the geometric reverb effect can only be done in the **IA3d5** interface, through the **A3dGeom2::GeomReverbParam** call, which takes an **A3DGEOMREVERBPARAM** structure. This structure has 5 members. The three that are relevant are:

fGeomScaling – This variable describes the scale factor which is used to determine just how big the surrounding polygons are. By default the scale factor is 1.0, and all units are assumed to be meters. So a corridor 1 meter wide and 2 meters tall is considered very tight. However, the application may not be describing its polygons in meters. For example: the application specifies a 3x3x3 cube, but without scaling it is impossible to determine if this is a gigantic auditorium or a tiny box. By default the cube is assumed to be 3 meters along each side, which makes it a small room. Since the size of the polygons is divided by the scale factor, a scale factor of 2 (which means 2 units per meter) makes the cube effectively 1.5 meters a side, which is tiny. A factor of 0.5 doubles the size of the cube to 6 meters a side.

fEffectScaling – This variable helps determine the sensitivity of the effect. By default this value is 1.0, and the default effect is calibrated accordingly. Increasing this number increases the effect, so reverb changes will occur more often and more drastically. Decreasing this number has the opposite effect. The minimum value is 0.0f, which decreases the effect to the point where it stops. Higher values can be specified, but the actual results will depend on the application. If the value is too high it will cause the reverb to change too often, and will muddle the effect. This value does not affect performance.

nVerticalAxis – This variable determines which coordinate axis points towards the sky. This is not critical, but it is helpful to have the A3D 3.0 coordinate system aligned with the application's coordinate system. This is completely independent of the listener orientation. The possible values are:

A3DAXIS_X
A3DAXIS_Y

A3DAXIS_Z

Manual Reflections

In A3D 2.0, reflections could only be rendered using the geometry-based Wavetracing engine. In A3D 3.0, we have opened up the underlying technology and exposed the reflection controls, allowing applications to manually create and place reflections. This allows for all sorts of additional effects, such as echoes.

Manual reflections are essentially an effect applied to a playing sound. Create an audio source, play it, then start creating and positioning reflections. These reflections can be created with a specified attributes for precisely controlled results – gain, delay, and position.

Using Manual Reflections

Manual reflections require an A3D source before they can be created. Once a source is made, follow these steps to generate and render manual reflections.

1. Create a new **IA3dReflection** object by calling **IA3dSource2::NewManualReflection**.

2. Once the reflection is created, it will start to play almost immediately. Use the **IA3dReflection** member functions to adjust the gain, delay, and position.

3. Once created, a reflection will play even after the source has stopped. Reflections only stop when:

    - The non-looping source reaches the end, and its reflection reaches its own end. This is the only case in which the reflection will not stop immediately when the source stops. Instead, it will continue to play until all of the sources data has been played.

    - Stop is called on the source.

    - The reflection is released.

    - The source is released.

When a source is finished with all its reflections, call **FreeManualReflections** to quickly stop and free them.

See the included “Manual Reflections” sample application for an implementation example.

Dolby Digital and MP3

In addition to high quality WAV support, A3D 3.0 has added seamless support for two new popular audio formats: Mpeg layer 3 (MP3) and Dolby Digital (AC-3). For both formats, the A3D 3.0 engine supports loading the files into memory as static sources and streaming them from disk, using the same **IA3dSource2** interface. (Currently, they can only be loaded from files; loading AC-3 or MP3 sources from a memory block is not possible.) MP3 and WAV sources work almost identically, but there are several important differences between them and AC-3 sources.

As with WAV files, MP3 files can be loaded into memory or streamed from disk. Both can be spatialized, occluded, reflected, and rendered as volumetric sources. Because of the nature of their compression, MP3 files may take more processor time to decode, and streamed MP3 sources cannot handle calls to **IA3dSource2::SetPlayTime**. Static MP3 sources will take a one-time CPU hit when they are created, since the MP3 data is decompressed into main memory. In all other respects, they work exactly like WAV sources.

Streamed MP3 sources take a constant amount of CPU as they are decompressed in real time. Also, calls to **IA3dSource2::SetPlayTime** will not work since the compressed formats make this very difficult to do. Calls to **IA3dSource2::SetPlayPosition** will work, but it will set the position to the compressed position, not the decompressed one.

AC-3 sources are a bit different. By nature, AC-3 is a multi-channel format, pre-rendered. Up to 6 streams will be compressed into a single AC3 stream. Thus, AC-3 files cannot be moved around in space the way WAV and MP3 sources are; AC-3 files will be rendered to the best of the system’s ability, using available hardware or the engine’s fallback decoder. A 5.1 channel AC-3 stream will always be played from each of the discreet channels. Reverb cannot be applied to AC-3 streams.

A key benefit of A3D 3.0's AC-3 support is its ability to automatically detect and use the best decoder on the system, falling back to its own internal decoder if no other is found. If the sound card is able to decode AC-3 in hardware, the user will get hardware decoding. If hardware is not available, DirectShow filters will be detected, and the best one is used. Note that some decoders have trouble playing all the various Dolby Digital bitrates; for maximum compatibility, we recommend 384kbps. There is absolutely no extra effort required on the part of the developer to enable any of these decoding options.

Its important to note that use of the Dolby Digital fallback decoder is pursuant to receiving an authentication code from Dolby. In most cases, this is free and can be arranged by contacting us at devsupport@a3d.com for information about Dolby's license agreement. Dolby will supply a key that is passed into **IA3d5::UnlockFallbackAC3Decoder**.

Using MP3 and AC-3 files

To create either MP3 or AC-3 sources, follow these steps:

1. If you are planning to use AC-3 file, you must unlock the A3D software fallback decoder by calling **IA3d5::UnlockFallbackAC3Decoder**.

2. First, create the source as either Native or A3D. AC-3 files can only play as Native.

3. Once the source is created, call **IA3dSource2::LoadFile** with the appropriate format flags - **A3DSOURCE_FORMAT_AC3** or **A3DSOURCE_FORMAT_MP3**.

The source is now loaded and ready to go. Call Play to start, Stop to stop, and set the volume and (for MP3 source) the position. All other effects work to. When the source is no longer needed, release it to free its resources.

For example code see the included "Loading" sample application.

An Overview of Dolby Digital

Dolby Digital is a multichannel audio codec that delivers up to 5.1 channels of 48kHz, 20 bit audio in approximately the bandwidth occupied by one-half of one channel of CD audio. As of this writing, there are over 15 million DVD-ROM drives equipped with DVD video players that can play back Dolby Digital. Aureal's A3D 3.0 is designed to harness the Direct Show-compatible decoders that are part of these DVD-equipped PCs to provide Dolby Digital playback in games.

The Dolby Digital logo is supplied at no charge to developers who produce content in Dolby Digital. To request a trademark agreement, please contact Dolby directly:

Dolby Laboratories, Inc.
Game Developer Relations
100 Potrero Ave.
San Francisco, CA 94103-4813
Tel: (415) 558-0200
Fax: (415) 863-1373

For more information on producing in Dolby Digital, please visit the Dolby web site at http://www.dolby.com/multi/.

For technical questions on producing content in Dolby Digital, please write to multimedia@dolby.com.

To request a licensing agreement via email, please contact us at tsa@dolby.com.

"Dolby" and the Double-D symbol are trademarks of Dolby Laboratories.

An Overview of MP3

In 1987, the IIS started to work on perceptual audio coding in the framework of the EUREKA project EU147, Digital Audio Broadcasting (DAB). In a joint cooperation with the University of Erlangen (Prof. Dieter Seitzer), the IIS finally devised a very powerful algorithm that is standardized as ISO-MPEG Audio Layer-3 (IS 11172-3 and IS 13818-3).

Without data reduction, digital audio signals typically consist of 16 bit samples recorded at a sampling rate more than twice the actual audio bandwidth (e.g. 44.1 kHz for Compact Disks). So you end up with more than 1.400 Mbit to represent just one second of stereo music in CD quality. By using MPEG audio coding, you may shrink down the original sound data from a CD by a factor of 12, without losing sound quality. Factors of 24 and even more still maintain a sound quality that is significantly better than what you get by just reducing the sampling rate and the resolution of your samples. Basically, this is realized by perceptual coding techniques addressing the perception of sound waves by the human ear.

Using MPEG audio, one may achieve a typical data reduction of

1:4	by **Layer 1** (corresponds with 384 kbps for a stereo signal)
1:6 ... 1:8	by **Layer 2** (corresponds with 256..192 kbps for a stereo signal)
1:10 ... 1:12	by **Layer 3** (corresponds with 128..112 kbps for a stereo signal)

still maintaining the original CD sound quality.

By exploiting stereo effects and by limiting the audio bandwidth, the coding schemes may achieve an acceptable sound quality at even lower bitrates. MPEG Layer-3 is the most powerful member of the MPEG audio coding family. For a given sound quality level, it requires the lowest bitrate - or for a given bitrate, it achieves the highest sound quality.

Some typical performance data of MPEG Layer-3 are:

Sound Quality	Bandwidth	Mode	Bitrate	Reduction Ratio
better than AM radio	7.5 kHz	mono	32 kbps	24:1
Similar to FM radio	11 kHz	stereo	56...64 kbps	26...24:1
Near CD	15 kHz	stereo	96 kbps	16:1
CD	>15 kHz	stereo	112..128kbps	14..12:1

In all international listening tests, MPEG Layer-3 impressively proved its superior performance, maintaining the original sound quality at a data reduction of 1:12 (around 64 kbit/s per audio channel). If applications may tolerate a limited bandwidth of around 10 kHz, a reasonable sound quality for stereo signals can be achieved even at a reduction of 1:24.

Streaming Audio

Up until now, playback of large files (such as the popular MP3 music files) resulted in either a lot of extra programming or the use of several megabytes of resident memory. A3D 3.0 solves the problem by transparently streaming audio from files, saving both development time and memory. Streaming support has been implemented for all supported audio formats: WAV, MP3, and AC3.

Streaming is implemented transparently, though the use of a separate thread. In most cases, the only programming effort needed is to add the **A3DSOURCE_STREAMING** flag to the second argument of **IA3dSource2::LoadFile**. Treat the source as any other source from then on. There may, however, be occasions that call for some fine-tuning. The **SetStreamingProperties** function allows the specification of the amount of data to buffer, and the priority of the streaming thread.

There are a few limitations on streaming sources. They must be created as unmanaged sources - because of the overhead in streaming they must reserve an output buffer. A related property of streaming sources is that they always play – the resource manager will never swap it out, because the resource manager does not manage the streaming sources. Streaming sources also cannot be duplicated. Finally, streaming compressed data has an additional restriction with the **IA3dSource2::SetWaveTime** function – it will not work for compressed audio formats such as MP3 and AC-3.

See the included “Loading” sample app for a demonstration of how to create and play streaming audio files.

Volumetric Sources

By default, sources are created as point sources. This means that they emanate only from the location that is specified for them. A volumetric source, on the other hand, can appear to emanate from an entire region, specified by a bounding box. This is useful for attaching sources to large objects that need to have a uniform sound over their entire surface. Examples of this are stadium crowds, rivers, and waterfalls.

In all other regards volumetric sources behave like ordinary point sources. They can be streamed from files, they can take either MP3 or WAV data, and they can be reflected and occluded. In fact, they can be partially occluded as well as fully occluded. And of course, they are very easy to create and use. Plus they are very efficient - the amount of system resources they use is slightly higher then point sources, and in virtually all cases unnoticeable.

Using Volumetric Sources

Specifying a volumetric source is quite simple. After the source has been created, call the function **IA3dSource2::SetVolumetricBounds** with the dimensions of the bounding box you wish to fill with sound. This bounding box will then be created around the location of the source. So the position of the source is the exact center of the volume.

1. First, create the source using **IA3d5::NewSource** or **IA3d5::DuplicateSource**.

2. Once the source is created and the appropriate data or file is loaded, call **SetVolumetricBounds** to make the source a volumetric source.

3. Now, position the source like normally, taking into account that the sources position now represents the center of a volume.

If desired, various tweaking functions can be applied by calling **IA3dSource2::SetVolumetricDamping**. A more detailed description of the tweaks can be found under the definition of the **A3DVOLSRCDAMPINFO** structure.

When the source is no longer needed, release it and its resources by calling **IA3dSource2::Release**.

Chapter 3: Starting up

This section describes the steps needed to create an A3D 3.0 object and to begin developing with Wavetracing. If there are any functions referred to in this document that you would like more information on, please refer to the A3D 3.0 API Reference Guide.

To look at some very simple A3D code, we've included several test applications with the SDK. The build environments we provide with the SDK include support for the following compilers:

Watcom C
Microsoft Visual C++ 5.0
Microsoft Visual C++ 6.0

Let's create a simple A3D 3.0 application. You can refer to the A3D 3.0 API Reference Guide for specific information about each of these functions.

Initializing A3D

The first thing that must be done is to initialize A3D 3.0. For convenience the SDK includes the ia3dutil.lib library with full source, to perform this operation easily.

1. Register the a3dapi.dll file.

   Use the **A3dRegister** function to do this.
2. Initialize COM.

   This can be done with the **A3dInitialize** function or the standard Windows CoInitialize function.
3. Create an A3D 3.0 object by calling the standard Windows function CoCreateInstance with the **GUID_CLSID_A3dApi** as an argument.
4. Call **`IA3d5::InitEx`** with the desired features (reflections, direct path, reverb, etc.)

Note that steps 3 and 4 can also be accomplished with the A3dCreate function.

Now that you've initialized A3D and your application has an A3D 3.0 object, you can begin creating sources.

Creating a Source

To create a new source, simply call **IA3d5::NewSource**.

Loading Wave Data Into a Source

There are two ways to load audio data into a source. The first is quite simple: a call to **IA3dSource2::LoadFile** with the specified audio file as argument, and you're ready to go. The other way to load data into a source is to copy the audio data in with the following steps:

1. **IA3d5::SetAudioFormat** — to set the appropriate information about the type of audio data.
2. **IA3d5::AllocateAudioData** — allocate memory and resources for the audio data.
3. **IA3d5::Lock** — get a pointer to the buffer's audio data.
4. **IA3d5::Unlock** — after copying in the data, you must release the buffer's audio data pointer.

Note that the lock/unlock method can also be used to dynamically stream data into a source. **AllocateAudioData** must be called before you can lock and copy memory.

Getting the Listener

Now, with an A3D 3.0 object and your sources, you have all but one of the blocks in place to start building a simple direct path application. The final block for direct path is the Listener; it has already been created for you. All you need to do is get a pointer to it. This involves one step:

IA3d5::QueryInterface, with IID_IA3dListener as the guid.

An important difference to note about A3D 3.0 in this case is how it differs from A3D 2.0. With A3D 3.0, you no longer need to query for the listener if you do not need to control it directly. If 3D sources are created without querying for a listener, the sounds will play with the listener initialized at 0, 0, 0. Native sources will be played back as usual. You can even obtain the Geom2 interface without first obtaining a listener. This mirrors what Microsoft has established with DirectSound. It is still recommended in most 3D gaming titles, however, that you query for the listener.

The Main Audio Loop

This sets up everything necessary for direct path audio. These commands only need to be issued once. The next stage of your application will be the main audio loop inside which you will update the listener and source positions. The main audio loop is delimited by calls to **IA3d5::Clear** and **IA3d5::Flush**. The following list shows some of the operations that may be performed inside the main loop to describe the audio scene:

1. Call **IA3d5::Clear**.
2. Position the listener with **IA3dListener::SetPosition3f** or **IA3dListener::SetPosition3fv**.
3. Position all sources to be played with **IA3dSource2::SetPosition3f** or **IA3dSource2::SetPosition3fv**.
4. Call **IA3d5::Flush**.

Thread Safety

Once you have an understanding of direct path implementation, you can move on to the other features in the A3D 3.0 engine. An important thing to remember is that typical applications will run their audio engine from a single thread. The A3D 3.0 library is not guaranteed to be thread safe; it should be accessed only from the thread that issued the **IA3d5::InitEx** call. Attempting to access the same set of A3D interfaces simultaneously from multiple threads could cause the application to crash. This doesn't mean that only one A3D application can run at a time; multiple applications can run simultaneously. If you are thinking about running multiple threads within your audio engine, you must make sure that no two A3D objects are accessed concurrently.

Additional Information

For more information about tweaking the distance model, Doppler effect, and other audio effects, see “Chapter 5: Tweaking Your A3D 3.0 Code” on page 27 of the A3D 3.0 User’s Guide.

For more information about the A3D 3.0 Resource Manager, what features of A3D work with what sound cards, and more about A2D support, see the A3D 3.0 Platform and Resource Manager Guide.

For more information about adding geometry-based effects, see “Chapter 6: Geometry Engine Reference Pages" on page 197 of the A3D 3.0 API Reference Guide for more detail.

Chapter 4: Debug Viewer GL

Debug Viewer GL

The Debug Viewer is a helpful tool for tweaking your title. The Debug Viewer allows you to view your audio much like you view graphics. You can follow the paths of sound sources, both direct path to the listener and those that get occluded. You can also view which polygons are reflecting and occluding your sound sources.

Debug Viewer Requirements

In order to run the debug viewer, you must ensure that the following things are in order:

1. You must have a valid OpenGL driver installed on your system.

   If you don't, Microsoft's software OpenGL driver has been provided on the SDK CD in the \OpenGL folder.
2. You must run the application in a window or on a monitor separate from the main windows monitor.

   This prohibits you from using a 3D only accelerator (such as those based on the 3dfx Voodoo 1 or Voodoo 2 chipset) for OpenGL graphics display on a single monitor setup.
3. The Debug Viewer version of a3dapi.dll must be in the application directory.
4. To run the viewer in network mode, you must have 2 connected computers configured with TCP/IP.

Starting the Debug Viewer

1. Double-click the DebugViewerGL.exe file.

   A red message appears indicating that no A3D process has been detected. The background behind the message is black.

   Make sure the Debug Viewer build of the a3dapi.dll is used.
2. Start your A3D title.

   After the A3D title calls **IA3d5::Flush**, the red message disappears and the background color changes from black to dark blue indicating a connection to the A3D title.

By default, when the Debug Viewer starts, the point of view is that of the listener.

Debugging Remotely Using Debug Viewer

If you cannot run your A3D title in windowed mode, you can still use the debug viewer by running the viewer remotely on a second computer connected by a TCP/IP network.

1. Start the Debug Viewer GL with the command line option -server on the computer on which you plan to run the full screen A3D title.

   Use the a3dapi.dll debug viewer build. The viewer starts up and displays a message that the viewer is searching for a connection.
2. Start Debug Viewer GL with the command line option –client on the second computer.

   A dialog box opens requesting the IP address of the server machine.
3. Enter the IP address

   The status changes to connected. The client display still shows that no A3D process is detected.
4. Start the full screen A3D title on the server machine.

   The client Debug Viewer screen turns dark blue and starts displaying geometry information.

The speed of the update depends on the speed of your network. The server Net Viewer by default searches port 1066 for a TCP connection. If your network reserves port 1066, you can change the port number by the registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Aureal\A3D\DebugViewer\PortNo
```

on both the client and server machine. Note that the Debug Viewer is not an optimized network application. It will use a lot of bandwidth, but it is an excellent tool for debugging and should only be used for debugging.

Commands

There are several keyboard commands you can use in the Debug Viewer:

S	Disable/enable staggering flicker. By default, when a title uses staggering to skip frame rendering, the Debug Viewer flickers when the audio system is not rendering a frame to let you know which frames are not being rendered. If you want a smooth playback instead, toggle the staggering of the Debug Viewer.
R	Toggle between wireframe and solid graphics modes. This allows you to change the geometry output mode between wire frame (default) and solid modes.
T	Toggles through the 6 views: top, bottom, left, right, face, and back. This enables you to change the camera location during a scene, in the order specified: top, bottom, left, right, face, and back. Note that depending on the coordinate system of the application, it may not be in this order. For example, the Quake II engine has a coordinate system is rotated 90 degrees on the X axis, so the fifth toggle is top.
E	Returns camera to listener point of view (Eye mode). This returns the camera to the listener position — viewing the application through the listener's point of view.
L	Toggles line displays on and off. This allows you to turn off drawing the lines for sound sources, viewing only the rendered polygons.
ESC	Closes the Debug Viewer.

Interpreting the Debug Viewer

The Debug Viewer uses colors to differentiate the properties of each polygon, box, and line.

Polygons

Orange:	Polygon is not occluding or reflecting.
Yellow:	Polygon is reflecting.
Blue:	Polygon is occluding.
White:	Polygons used in Geometric Reverb calculation

Note that if a polygon is both reflecting and occluding, it appears only as yellow.

Boxes

Yellow:	Sound source
Purple:	Listener
White:	Image (reflected sounds)

Note that volumetric sources appear as boxes with the dimensions dx, dy, dz applied to them by **IA3dSource2::SetVolumetricBounds(dx, dy, dz)**.

Lines

Green:	Direct path from source to listener.
Blue:	Direct path occluded from source to listener.
Orange:	Source to image.
Yellow:	Image to listener.

Limitations

There are a few limitations of the Debug Viewer that you should be aware of:

- The Debug Viewer has a limit of 1024 polygons. If your title uses more than 1024 polygons, the Debug Viewer only shows the first 1024. However, the remaining polygons are still rendered in your title.
- You can only debug one title or one instance of a title at a time.
- The Debug Viewer only displays 16 sources and 16 reflections.
- The Debug Viewer uses a right-handed coordinate system. If you have defined the coordinate system as left-handed, the Debug Viewer only displays geometry correctly when set to Eye mode. All other views have the Z-axis reversed.
- The Debug Viewer and the accompanying a3dapi.dll file must be the same version.

Chapter 5: Tweaking Your A3D 3.0 Code

Just like with graphics, how you code your audio has an immense effect on the user's experience, both in audio quality and system performance. This section includes some tips on how to tweak the effects of A3D 3.0, followed by some performance enhancing techniques. Like you, our ultimate goal is for your title to create an immersive, powerful audio experience.

The key concepts behind direct path rendering are 3D positioning, the distance model, and Doppler shifting.

3D Positioning

3D positioning is important because 3D sounds don't follow the same guidelines as 2D sounds. 3D sources have to be positioned where the listener expects them to be heard. For example, a first person shooter may position models according to where the models contact the floor, but any sounds attached to that model need to play from where the source is located. So if the model shoots, the sound should come from the muzzle of the gun, and not the floor.

These small positional differences are noticeable up close, and in certain edge cases. Take the case of an enemy with associated sounds positioned at the base of his model; if it walks behind a bench, its sources are now occluded.

If the listener is positioned next to the model's head, any sources emanating from that model sound as if they're coming from below.

Distance Model

The distance model is more complex to control accurately than simple positioning in xyz-coordinates. A3D 3.0 uses a 1/distance scale, so a source's gain drops off more slowly the farther it is from the listener. At a great enough distance, it is effectively silent, however.

As a source moves away from the listener, it first reaches the minimum distance. Before this distance, the source will be at full gain. Once it passes this value, its gain starts to drop and its high frequencies begin to get filtered. As it moves further away, the gain continues dropping and the high frequencies are even more filtered. Once it reaches the maximum distance, it either mutes or its gain and high frequency roll-off stop changing. This latter behavior is controlled by a flag to the **SetMinMaxDistance** call on the source.

The issue here is appropriately defining the minimum and maximum distances and the **DistanceModelScale** of each source. How the source behaves as it moves away from the listener depends on both the minimum distance and the distance model scale. The distance model scale is essentially a distance multiplier. The source's absolute distance from the minimum distance is scaled by the distance model scale value.

Example: A source is 5.0 meters beyond its minimum distance, with a **DistanceModelScale** of 2.0, so it behaves as if it were 5.0 * 2.0 = 10.0 meters beyond the minimum distance.

The minimum distance, apart from determining when distance effects are applied to a source, also determines how fast those effects take place. A source with a minimum distance of 5 has roll-off effects that occur 5 times slower then a source with a minimum distance of 1. Thus, the minimum distance value can be considered the base unit with which distance attenuation is measured.

Example: Source A has a minimum distance of 1.0m. Source B has a minimum distance of 5.0m. Both have a **DistanceModelScale** of 1.0. The gain of Source A at 10 meters is equivalent to that of Source B at 50 meters.

Here is a specific example, to help define how you can use these controls to adjust the distance attenuation effects in A3D 3.0.

Say you want to have a sound source that you only want to be heard up to 30.0m away, is silent beyond that point, and gets louder as you get right up to it, with a good, loud volume at 6.0m. Set the Max distance to 30.0m, and set **A3D_MUTE** as the flag in the **SetMinMaxDistance** call. This gives you an audible source until 30.0m and then the source will be muted as you want. Now you need to deal with it being too loud when it is at 30.0m, so you use the call **SetDistanceModelScale**, with a higher than default value, let's say 4.0, and that makes the sound roll off faster, so you're closer to silence by the time you hit 30.0m. Now you have the last problem of making your sound still appear loud at 8.0m, but get louder still as you get closer. This is the part where tweaking really makes the difference. Set the minimum distance to around 3.0m. Then, you should hopefully still get a decent volume at 6.0m, but as you get towards 3.0m, the sound will still get louder, until it hits a maximum at 3.0m and within. What you are doing is setting a minimum distance somewhere between 1.0m (the default) and 6.0m to get things to sound “reasonably loud” at 6.0 m, but louder, closer than that. Experiment with the min distance to tweak the effect.

Additionally, your application and A3D 3.0 may not share the same units. If this is the case, then a distance of 10 in your application will not correspond to 10 meters. The easiest way to solve this is to scale your min and max distances so that they correspond to your own units, and not to meters.

Doppler Shifting

Doppler is the effect caused by rapidly moving sources as they move in relation to the listener. As a source approaches the listener its pitch increases. As it moves away from the listener its pitch decreases. Sources moving parallel to the listener do not exhibit Doppler effects. The tweaking necessary to smooth out Doppler is to pick an appropriate DopplerScale value. Also, proper 3D positioning of sources is critical for smooth Doppler effects. In addition, you need to specify velocities on your sources and/or listener to generate Doppler effects, and it is important to make sure these velocity values are smooth and accurate.

Geometry Optimizations

Providing your scene geometry for A3D 3.0 rendering is not as daunting a task as it may seem. The processes in aurally displaying your scene-graph is not dissimilar to the ones you are most likely already using when rendering your graphical world.

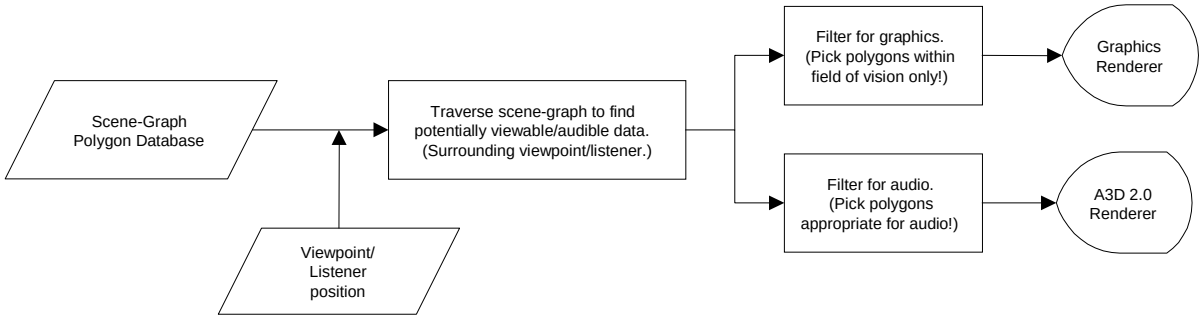


FIGURE 2.

As you can see from the diagram of Figure 2, there are many common processes for rendering your geometry for A3D 3.0, and for your visuals. The blocks and connector-lines illustrate processes that you will most likely already be doing in your graphics rendering pipeline. The standard pipeline dictates that you access your scene-graph at a location appropriate for the viewpoint position within the world. From there, a potentially viewable set (PVS) of polygons is extracted, and this is then filtered for field of view. This filtered polygon set is then sent down to the graphical renderer via appropriate API calls. Similarly, for A3D 3.0 rendering, you would take that same set of polygons, now considered a potentially audible set (PAS), and filter them for audio. That filtered set is then sent down to the Wavetracing renderer via appropriate A3D 3.0 API calls. The audio filtering process is the crucial added element here, and that is what the optimizations described below seek to address.

A few basic concepts before we get into details:

- The fewer polygons used, the less CPU overhead required to process them.
- Reflection polygons are considerably more expensive than occlusion polygons.
- Use of lists is beneficial.
- Global state changes should be done as sparingly as possible.

The number of polygons required to create a realistic and immersive effect depends on the nature of your title. Typical applications will require only 50 to 100 reflection polygons and 400 to 600 occlusion polygons to be rendered per frame. You will want to experiment with the number of audio polygons to render in your title to create a balance between effect and CPU overhead.

Note: **A3dGeom2::Tag** must be called for every individual polygon sent to the API for all polygons that you wish to use for reflections. This is used to identify unique polygons, so it is important that the same polygon has the same tag every time it is sent.

Integrating Wavetracing into your application has several stages, and we have provided customized a3dapi.dlls to target these processes. The debug a3dapi.dll should be used for general coding and debugging, when first integrating geometry. The Debug Viewer a3dapi.dll should be used to test, tweak, and optimize geometry, once basic functionality has been successfully integrated. The retail a3dapi.dll, which is also the one to be shipped with the final title, should be used when performance testing.

The Debug Viewer application that is provided with the SDK is an invaluable tool for successfully integrating optimized geometry into your application. It is an excellent way to visualize what A3D 3.0 is rendering, as well as helping tremendously to tweak the size of reflective polygons, the distance with which polygons no longer have to be rendered, and the quality of both the reflection and occlusion rendering. You can use the viewer in concert with your application, to visually inspect if the polygons being sent to the A3D 3.0 engine are appropriate, and consistent with your graphical world. Note that the viewer application will only attach to the special debug-viewer version of the a3dapi.dll. The viewer also displays reflection and occlusion processing on polygons, so you can quickly determine if each of these effects has been successfully enabled. Finally, the debug viewer can effectively show the volume/number of polygons being rendered in a given frame through simple visual inspection. You can quickly determine if you are not sending enough polygons down to the renderer to give a fair representation of your scene. The viewer does have a limit of 1024 polygons concurrently, but this should be well beyond the number you would want to send in a given application.

The ideal scenario for A3D is to drastically reduce the graphical geometry in your application to large, smooth audio polygons. A detailed room with arches, doorways, rafters, and other architectural details can often be acoustically rendered with a handful of polygons (perhaps 10 versus 500, for instance). To do this either requires an engine that is capable of reducing geometry data to a lower resolution, or a second copy of the geometry data that has already been reduced for A3D 3.0.

In the case that your application cannot produce a small number of large polygons, A3D is quite capable of rendering larger numbers of polygons per frame. Staying within 10% of the total CPU on a Pentium II 233MHz machine, A3D can render approximately 100 reflection polygons and 1000 occlusion polygons. The issue is then one of which polygons to choose, and what we call the "Swiss Cheese Effect." This is when the listener passes by what visually appears to be a solid surface, but a source on the opposite side is randomly occluded as it passes by polygons and holes. Due to this noticeable aural artifact, it is important to render almost all polygons in the vicinity of the listener.

Fortunately, reflections are much less prone to inconsistent geometry. What most people experience in terms of reflections is actually an intensity. A greater number of reflections leads to a more intense, or lively environmental effect. So for optimum effect and efficiency, small and large polygons should occlude, but only larger polygons should reflect. Determining what is a large polygon is a tweak value that should be experimented with to find the perfect size.

Now we address the geometry list interface. A3D has the IA3dList object, which represents and stores a collection of polygons. These polygons retain their individual properties, such as material characteristics, and whether they occlude and/or reflect. We recommend breaking up your polygons into a number of smaller lists; this concept is referred to as "list caching."

There are several advantages to using list caching: it is better to build all your lists at load time then to render individual polygons between frames; only the nearby, relevant lists need to be rendered at any given time; and there are a number of internal optimizations based on lists. Good uses of list caching are as follows:

- Build the lists of polygons using an appropriate level of detail for audio rendering.
- Call only those lists which are near the listener. Use the Debug Viewer to tweak this distance.

The internal optimizations done on lists include the following:

- The ability to occlude an entire list based on a pre-computed bounding box.
  This must be explicitly enabled by the application, via the appropriate API call on the list interface. It is not necessary to provide a bounding-box; the A3D engine will calculate this internally. However, if you have this information as a side effect of some other process, you can supply it and save computation time.
- Caching of the last occluding polygon per list.
  Same concept as a disk or memory cache: if this polygon was occluding a source in the last frame update, it is most likely occluding that same source now, and gets checked first.
- Future list caching optimizations will be seamlessly integrated into the API, improving performance and effect.

Finally, there are a number of miscellaneous optimizations. We support a mode where reflections and/or occlusions are not rendered every frame. Reflection staggering does not have much of an impact on audio quality but has a noticeable impact on performance. Occlusion staggering may not demonstrate enough performance gains for the loss in quality, and for most cases, is not recommended. However, your application may benefit from a reduced rate of reflection and/or occlusion rendering, and it is worthwhile testing and tweaking these values for effect and performance.

Example:

Reflection staggering = 2

Occlusion staggering = 1

Frame 1:

Reflection state = Reflect (frame 1)

Occlusion state = Occlude (frame 1)

Frame 2:

Reflection state = Reflect (frame 1) Cached and not recalculated.

Occlusion state = Occlude (frame 2)

Frame 3:

Reflection state = Reflect (frame 3)

Occlusion state = Occlude (frame 3)

Other optimizations are:

- Only call **IA3d5::Clear** when rendering new geometry.

If the geometry for the current frame is the same as the last frame, don't resend it or call **IA3d5::Clear**.

- Global state calls, such as setting the current material, require extra computation within the engine.
  For example, it is better to sort polygons so that those with the same material are rendered during the same state without the need for a transition to a different material in between. We recommend working on optimizations such as list caching before addressing these smaller benefits to performance.
- Only call **IA3d5::Flush** once per frame. Repeated calls slow performance dramatically, and may have negative side effects.

Conclusion

We welcome your feedback on this section. You, the developer, are an extremely valuable resource to us. In time, we will continue updating this section with new tips and tricks for getting the most from the audio engine.

Chapter 6: Releasing Your Title

Now that your new A3D-enabled title is ready to ship, let's go over a few things you need to do to ensure the your product's A3D implementation will work correctly for your users.

Install the a3dapi.dll File

The a3dapi.dll file is necessary for successful initialization of A3D and A2D, regardless of the sound card in the system. You have 3 options:

Option 1

In your setup program, install a3dapi.dll into the Windows System directory. If you do this, you should use the Win32 setup API call VerInstallFile to prevent installing an older version of the file over another application's installation of a more recent version. Using this approach, your application benefits from bug fixes and increased performance through customer updates.

Option 2

In your setup program, install a3dapi.dll in the application directory along with your application. This gives you greater control over compatibility and performance but also leaves you with greater responsibility for updating your customers' version of this file if and when that is required to address defects, add functionality, improve performance, etc.

Option 3

Rely on the user having a3dapi.dll already installed on his or her machine. If the file does not exist you will fail initialization of A3D and will need to check for this failure and fall back to your own sound engine. The a3dapi.dll file is necessary for the A2D sound engine to operate. If you are using A3D 3.0 as your sole audio engine, you must ship the a3dapi.dll and follow the directions from one of the aforementioned options.

Register the a3dapi.dll File

You must register the a3dapi.dll and include it in the application directory in order for A3D 3.0 to initialize.

The a3dapi.dll must be registered in the Windows Registry. A3D initialization will fail if a3dapi.dll is not registered. The safest method of ensuring the DLL file is registered is to register it each time the application launches. Link the ia3dutil.lib file into your application code and make one call at the start.

Call A3dInitialize on starting up your application and A3dCreate to initialize A3D. Both the A3dInitialize and A3dCreate calls register the a3dapi.dll file with the system registry. Here's some sample code:

```
IA3d *gpA3D;


 Initialize_A3D()
 {
    DWORD dwFeatures;


    dwFeatures = A3D_1ST_REFLECTIONS
      | A3D_DIRECT_PATH_A3D
      | DISABLE_FOCUS_MUTE
      | A3D_REVERB
      | A3D_GEOMETRIC_REVERB
      | A3D_OCCLUSIONS;


    A3DInitialize();
A3dCreate (NULL, &gpA3D, NULL, dwFeatures);
}
```

If your code doesn't call either of these functions, you'll need your code to call the A3dRegister function at the beginning of the application code.

Here's some more sample code:

```
InitAudio()
 {
    A3dRegister();
    ...
    // Create A3D via your method...
    ...
 }
```

A3D Splash Screen

Aureal provides a splash screen for all true-A3D cards to let the user know when A3D has been initialized. This occurs when the **NewSource** function is first called in your title. Sometimes certain video modes can cause a problem. In this case, disable the splash screen to keep it from appearing.

When you query for the A3D interface and receive a pointer to it, you must call the InitEx function. We've created a flag that gets passed into InitEx that can be used to disable the splash screen for your application. The flag is: A3D_DISABLE_SPLASHSCREEN.

Here is some sample code:

```
// m_pA3d is a pointer to the IA3d interface...
m_pA3d5->InitEx(NULL, A3D_1ST_REFLECTIONS |
A3D_DISABLE_SPLASHSCREEN, A3DRENDERMODEPREFS_DEFAULT);
```

We don't recommend disabling the splash screen unless absolutely necessary. Gamers rely on the splash screen to know that A3D has been initialized and that they will be getting 3D positional audio. If gamers fail to see this, they may question it and call your company's technical support lines.

End User Configurability

One of the best features you can add to your title is the ability for the end user to tweak the audio values for themselves. As we all know, audio is very subjective; what sounds great to one person may sound only mediocre to another. Giving your user the ability to adjust the audio to his or her liking can only add to the aural experience they'll enjoy in your title.

Games such as *Quake*, *Unreal*, and *Heretic II* provide extensive configurability for the user. Suggested tweak options you may wish to make available to your users:

- Reflections disable/enable
- Occlusions disable/enable
- Geometry processing disable/enable
- Reflections gain value
- Reflections delay value
- Occlusions transmission factor
- Distance model (min/max distances for roll-off)
- Number of polygons rendered

Whether through console variables or an application .ini file, exposing these to the end user will demonstrate the flexibility of your title's audio engine.

Aureal Provides A3D Compatibility Testing

There are many A3D cards on the market; most of those sound cards use our Vortex chipsets, while others have licensed A3D into their own design. Testing your title on all these sound cards is a lot of work for your QA department. Part of our commitment to you, the developer, is that if you provide us with copies of your application before release, we can run it on the various true A3D platforms to make sure the title sounds the best it can. We'll provide valuable feedback about audio quality and compatibility to you and your team. While our testing won't (and shouldn't) replace your own testing, hopefully it can help to lighten the load on your QA team.

A3D Logo Placement

While we don't require it, we ask that you acknowledge that your title uses the A3D engine by placing the A3D logo on the outside of your box, in the manual, on your website, etc. We've dedicated a lot of time and resources to create a full-featured sound engine that easily rivals the $5000 commercial packages, only to give it away royalty-free. Your support and promotion of A3D helps us to continue creating new levels of high-quality positional audio and further improve an already top-notch sound engine.

Displaying the A3D logo also allows gamers to instantly know that 3D positional audio is a feature of the title. This inevitably helps your product in both sales and reviews. Logo artwork is available on the SDK CD, located in a directory called \Logos in the root directory. The logos directory contains a subdirectory for Aureal and for A3D. Normally, the A3D logo (and not the Aureal logo) should be placed on the outside of boxes.

Addendum

Credits

The A3D 3.0 team:

Suneil Mishra
Terry Blanchard
Nam Do
Scott Etherton
David Gasior
Joe Longworth
Micah Mason
Sherman Mui

Thanks

Special thanks to everyone else who has helped in their own ways to make the A3D 3.0 audio engine a reality:

Brendan O'Flaherty
Rob Bishop
Alice Hwang
Kirsten Kuhail
Cameron Lewis
Caroline Lie
Skip McIlvaine
Andrew Wheeler

All the folks at Dolby Labs, Mediamatics, and the Fraunhofer Institute for their support in integrating Dolby Digital (AC-3) and MP3 support

The Vortex Driver and Algorithm teams for implementing all the cool features we asked them to

All the developers on A3D 1.x and A3D 2.0

Everyone at Aureal for their support

And finally, to all the developers who've supported A3D over the years and truly made us sound good

Index