AUREAL

A3D 2.0 Software Development Kit Users' Guide

Official Users' Guide for A3D Version 2.0

Disclaimer

Copyright

Trademarks

Document Number: DO1011-030799

Contents

Chapter 1

Welcome to A3D 2.0

Welcome to A3D 2.0, the industry-leading 3D audio standard. The Aureal A3D 2.0 Software Development Kit (SDK) includes everything necessary to build a complete sound engine including 3D positional audio and other features into your next software application. Included in the kit are the following:

- A3D 2.0 API Reference Guide
- A3D 2.0 User's Guide
- A3D 2.0 Platform and Resource Manager Guide
- Source code and sample A3D 2.0 applications
- Code support files
- Release notes
- A3D drivers
- A3D logos
- Portable A3D demo
- Aureal Wavetracing™ technology demo
- Minerva test application
- Marketing presentation and co-marketing materials
- Vortex audio controller data sheets

The A3D 2.0 API Reference Guide

This is the heart of the SDK. The *A3D 2.0 API Reference Guide* contains a complete list of A3D 2.0 functions, their descriptions, structures, and error codes. This is where you will find the actual code that makes A3D 2.0 work. As new features are added to A3D 2.0, the release notes will be updated and you will find the actual code in the reference manual. An index of A3D 2.0 functions is included for easy reference.

The A3D 2.0 Users' Guide

This document contains all the information you need (aside from the actual API reference manual, which details specific code) to add support for A3D 2.0 into your application. It has been separated from the actual *A3D 2.0 API Reference Guide* for your convenience. This document is divided into the following chapters:

- Welcome to A3D 2.0: description of items included in the SDK.
- Starting Up: what you need to code, compile and run your application with A3D. It also includes a description of A3D, what's new, where to go for answers and information.
- Debug Viewer GL: information on this helpful debug tool that enables you to "see" what you are hearing.
- Tweaking Your A3D 2.0 Code: tips and tricks for writing the fastest and best sounding A3D code.
- Releasing Your Application: information about the steps necessary to ensure A3D 2.0 works on your customers' computers.

A3D 2.0 Platform and Resource Manager Guide

The *A3D 2.0 Platform and Resource Manager Guide* contains information about the new resource manager featured in A3D 2.0. Among other things it covers DS3D, A2D, and a platform matrix of the different features available under A3D 2.0 using different audio hardware setups.

Source Code and Sample A3D 2.0 Applications

Also known as tutorials, we have included some very simple applications which utilize A3D 2.0, along with the source code used to compile them. This enables you to see what complete A3D 2.0 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 2.0 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. These drivers are updated often and are also available on our website (www.a3d.com) for your convenience. To see exactly what the A3D code is doing at a given time, you can use the debug drivers to step through the code and receive valuable debug information.

A3D Logos

Included with the SDK are various versions (MAC, PC, PhotoShop, Illustrator, etc.) of the A3D logo. Although A3D 2.0 works 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 box and your web site.

Portable A3D Demo

We have put together an A3D Demo that runs on any PC platform. Using the output of an A3D sound card we recorded some audio files of A3D in action and encoded it in a linear format that plays on any PC. Although it is better to use A3D hardware in a interactive demonstration, this demo can still be used to demonstrate rather than explain the benefits of A3D. For a real-life A3D demo check the A3D control panel included with your A3D sound card or run the Wavetracing demo included in the SDK.

(Available only on the SDK CD)

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 2.0. Move sound sources, the listener, and walls around to dynamically affect the A3D experience.

(Available only on the SDK CD)

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.

(Available only on the SDK CD)

Marketing Presentation and Co-marketing Materials

A simple PowerPoint presentation file about marketing opportunities with Aureal and Aureal's partners.

(Available only on the SDK CD)

Vortex Product Briefs

These documents detail the specifications and features of Aureal's popular PCI audio controllers and A3D accelerators, the Vortex chipset.

(Available only on the SDK CD)

Technical Support

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

Aureal Software Partners Page: http://www.aureal.com/partners/pa_soft1.htm

Skip McIlvaine
Developer Relations Manager
510-252-4378
Skip@A3D.com

Suneil Mishra
Developer Support Manager
510-252-4236
Suneil@A3D.com

Chapter 2

Starting Up

What's New and Different About A3D 2.0?

- Complete sound engine

  A3D 1.x was an extension to DirectSound. A3D 2.0 is a complete sound engine, including the improved streaming resource manager, Wavetracing technology, DS3D hardware support, A2D software mixing support, and an improved distance model.

- Streaming Resource Manager

  This mode optimally manages available 3D hardware, available 2D hardware, and application-specified software mixed buffers. See "Chapter 2: The Resource Manager" on page 3 of the *A3D 2.0 Platform and Resource Manager Guide* for more detail.

- Wavetracing technology

  Geometry based reflection and occlusion effects. See "Chapter : Wavetracing Algorithms" on page 18 of the *A3D 2.0 API Reference Guide* for more detail.

- Geometry Debug Viewer

  A tool to visually display all the geometry, sources, reflections, and occlusions that are currently being rendered. See "Chapter 3: Debug Viewer GL" on page 13 of the *Users' Guide* for more detail.

- A2D

  Host-based 3D rendering of the A3D 2.0 engine, providing a tightly-optimized, feature-reduced version of A3D for sound cards that do not perform hardware audio acceleration.
- DS3D support

  For those without a Vortex-based sound card, A3D 2.0 fully supports all hardware accelerated audio cards.
- Enhanced distance model

  The A3D 2.0 distance model has a few differences from the A3D 1.x distance model. **IDirectSoundListener::SetRolloffFactor** has been replaced with a similar, per source function: **IA3dSource::SetDistanceModelScale**. Also, **IDirectSoundListener::SetDistanceFactor** has been replaced with **IA3d4::SetUnitsPerMeter**.

Starting to Code

This section describes the steps needed to create an A3D 2.0 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 2.0 API Reference Guide*.

For the latest drivers and information about A3D 2.0, visit the Aureal website at www.aureal.com.

A3D 2.0, in addition to the distance model and superb 3D audio quality, introduces the new Aureal Wavetracing technology. Wavetracing technology takes application geometry information, combines it with the source and listener positions, and renders the audio scene based on proven psychoacoustic principles.

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 2.0 application. You can refer to the *A3D 2.0 API Reference Guide* for specific information about each of these functions.

Initializing A3D

The first thing that must be done is to initialize A3D 2.0. 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 2.0 object by calling the standard Windows function **CoCreateInstance** with the **GUID_CLSID_A3dApi** as an argument.
4. Call **IA3d4::Init** with the desired features (reflections, direct path, etc.).
5. Call **IA3d4::SetCooperativeLevel**.

   Most applications only need a priority level of A3D_CL_NORMAL.

Now that you've initialized A3D and your application has an A3D 2.0 object, you can begin creating sources.

Creating a Source

To create a new source, simply call **IA3d4::NewSource**.

Loading Wave Data Into a Source

There are two ways to load wave file data into a source. The first is quite simple: a call to **IA3dSource::LoadWaveFile** with the specified wave file as argument, and you're ready to go. The other way to load data into a source is to copy the wave data in with the following steps:

1. **IA3d4::SetWaveFormat** — to set the appropriate information about the type of wave data.
2. **IA3d4::AllocateWaveData** — allocate memory and resources for the wave data.
3. **IA3d4::Lock** — get a pointer to the buffer's wave data.
4. **IA3d4::Unlock** — after copying in the data, you must release the buffer's wave data pointer.

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

Getting the Listener

Now, with an A3D 2.0 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:

IA3d4::QueryInterface, with **IID_IA3dListener** as the guid.

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 **IA3d4::Clear** and **IA3d4::Flush**. The following list shows some of the operations that may be performed inside the main loop to describe the audio scene:

1. Call **IA3d4::Clear**
1. Position the listener with **IA3dListener::SetPosition3f** or **IA3dListener::SetPosition3fv**.
2. Position all sources to be played with **IA3dSource::SetPosition3f** or **IA3dSource::SetPosition3fv**.
3. Call **IA3d4::Flush**.

Thread Safety

Once you have an understanding direct path implementation, you can move on to the other features in the A3D 2.0 engine. An important thing to remember is that typical applications will run their audio engine from a single thread. The A3D 2.0 library is not guaranteed to be thread safe; it should be accessed only from the thread that issued the **IA3d4::Init** 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 with your audio engine, bear this in mind.

Additional Information

For more information about tweaking the distance model, Doppler effect, and other audio effects, see “Chapter 4: Tweaking Your A3D 2.0 Code” on page 17 of the *A3D 2.0 Users’ Guide*.

For more information about the A3D 2.0 Resource Manager, what features of A3D work with what sound cards, and more about A2D support, see the *A3D 2.0 Platform and Resource Manager Guide*.

For more information about adding geometry-based effects, see “Chapter 5: Geometry Engine Reference Pages” on page 105 of the *A3D 2.0 API Reference Guide* for more detail.

Chapter 3

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.

   This prohibits you from using a 3Dfx Voodoo 1 or Voodoo 2 chipset for OpenGL graphics display.
3. The Debug Viewer version of a3dapi.dll must be in the application directory. The msvcrtd.dll must be in your Windows System folder.

Starting the Debug Viewer

To start the Debug Viewer, simply double-click the DebugViewerGL.exe file.

Start the Debug Viewer after your A3D title has been started and A3D has been initialized and **A3D::Flush()** has been called. Otherwise the Debug Viewer quits with an error message stating that no A3D process has been started.

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

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 to change the geometry output mode between wireframe (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 player 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.

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)

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:

- Always close the Debug Viewer before closing the title you are viewing. If you close the title first, the Debug Viewer may crash.
- The Debug Viewer has a limit of 2048 polygons. If your title uses more than 2048 polygons, the Debug Viewer only shows the first 2048. 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 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 4

Tweaking Your A3D 2.0 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 2.0, 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 they are expected 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 2.0 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 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 2.0.

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/off 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 6.0m, but louder, closer than that. Where to set that min distance that gets 6.0m sounding the way you want it, is what you can experiment with.

Additionally, your application and A3D 2.0 may not share the same units. To handle this case, there is a function call, **IA3d4::SetUnitsPerMeter,** that takes in a conversion value which specifies the number of application units per meter. The higher this number is, the closer all sources get to the listener. The smaller the number, the further everything gets. Use this to adjust the distance model to be more intuitive relative to your application.

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 2.0 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.

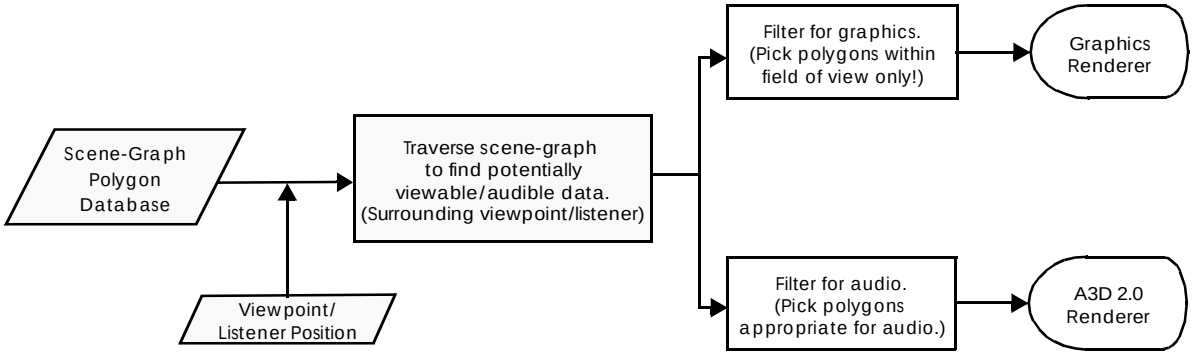

FIGURE 1.

As you can see from the diagram, there are many common processes for rendering your geometry for A3D 2.0, and for your visuals. The grey 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 2.0 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 2.0 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: **A3dGeom::Tag()** ***must be called for every individual polygon sent to the API for all polygons trhat you wish to use for reflections.***

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 retail a3dapi.dll, which is also the one to be shipped with the final title, should be used when performance testing. The DebugViewer a3dapi.dll should be used to test, tweak, and optimize geometry, once basic functionality has been successfully integrated.

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 2.0 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 2.0 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 2048 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 which is capable of reducing geometry data to a lower resolution, or a second copy of the geometry data which has already been reduced for A3D 2.0.

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 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 which 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. This will be calculated internally by the A3D engine. 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 **IA3d4::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 **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 **IA3d4::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 5

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 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 2

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 help 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 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 2.0 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 2.0 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 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_LATE_REFLECTIONS
      | A3D_DIRECT_PATH
      | A3D_DISTANCE_MODEL
      | 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 **A3dInitialize** function is 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 **Init()** function. We've created a flag that gets passed into **Init()** 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_pA3d->Init(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 a 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

By placing the A3D logo on the outside of your box, in the manual, on your website, etc., gamers 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. The logo artwork is located in a directory called \Logos in the root directory. The logos directory contains a subdirectory for Aureal and for A3D.

Index