AUREAL

A3D 2.0 Platform and Resource Manager Guide

3D

Document Number: DO1012-030799

Contents

Chapter 1 Introduction

One of the key features of A3D 2.0 is the new streaming Resource Manager. A3D 2.0 enables the application developer to specify an arbitrarily complex audio scene, with virtually any number of sources and wall surfaces. The end-user’s PC system, however, is limited to a finite number of 3D audio channels, and a finite number of surface reflections. The task of optimally mapping the sound sources to hardware can be a difficult task, and greatly affects the believability of the rendered audio scene.

For most developers, using the Resource Manager will be a transparent process. Others however, will want to take a look at how exactly it operates to get the most from it. This document covers various aspects of the Resource Manager, its functions, and how it relates to DS3D and A2D rendering. The document will also provide more detail about A2D. Applications written to the A3D 2.0 API can use Aureal Vortex or hardware DirectSound3D resources, if available, to avoid host processing whenever possible. In the absence of hardware resources, A2D runs stand-alone on any host CPU to emulate A3D in a software-only environment. A2D is based on a heavily speed-optimized, feature-reduced version of A3D.

Written in tightly optimized x86 assembly code, A2D offers great speed (lower CPU usage than other host-based solutions) and great audio quality (saturated mixing, linear interpolation, sample rate conversion, interaural time delay-based positioning).

Finally, this document opens with a platform guide which details what features of A3D 2.0 will work with each sound card.

Platform Matrix

This table serves to detail the capabilities of several A3D and non-A3D sound cards/audio chipsets currently on the market, and which features of A3D 2.0 will function on them.

Chipset[a]	**Aureal Vortex2**	**Aureal Vortex1**	**Diamond Freedom PCI**	**EMU 10k1**	**Ensoniq ES137x**	**ESS Maestro-2**	**EMU 8000**	**Yamaha YMF724**
HW 3D channels	76	8	8	32	?	5	0	8
Rendering Level	A3D	A3D	A3D	DS3D	A2D	DS3D	A2D	DS3D
Direct Path	Yes	Yes	Yes	Yes	Yes	Yes	Yes[b]	Yes
Reflections	Yes	No	No	No	No	No	No	No
Occlusions	Yes	Yes	Yes	Yes	Yes[b]	Yes[b]	Yes[b]	Yes[b]
Example Cards	Diamond MX300 Xitel Storm Platinum Turtle Beach Montego II Videologic SonicVortex2	Diamond Sonic Impact S90 Xitel Storm VX Turtle Beach Montego Aztech PCI338 Orchid NuSound PCI	Diamond Monster Sound Diamond Monster Sound M80 Diamond Monster Sound MX200	Sound Blaster Live! Sound Blaster Live! Value	Sound Blaster PCI64 Sound Blaster PCI 128 Ensoniq AudioPCI	Diamond Sonic Impact S70	Sound Blaster AWE32 Sound Blaster AWE 64	Yamaha Wave-Force 192XG

a. All specifications are believed to be accurate based on manufacturer data sheets, advertisements, reviews, and audio testing with applications such as DSShow3D.

b. Through A2D software rendering.

Chapter 2 The Resource Manager

Introduction

A3D 2.0 enables the application developer to specify an arbitrarily complex audio scene, with virtually any number of sources and wall surfaces. The end user's PC system, however, is limited to a finite number of 3D audio channels, and a finite number of surface reflections. The task of optimally mapping the sound sources to hardware can be a difficult task, and greatly affects the believability of the rendered audio scene. Therefore, A3D 2.0 includes a powerful audio hardware resource manager (RM) that enables the application developer to easily optimize sound rendering for an arbitrary audio scene.

RM functionality includes:

- Sophisticated management of looping buffers.

  Looping buffers are prioritized and managed just like single-shot buffers.
- Continuation of single-shot buffers

  Single-shot buffers aren't thrown away when "swapped out," but continue playing if resources become available. The restart sample point is where it should have been if the buffer was never swapped out.

- Virtualization of buffers

  When a buffer is swapped out, A3D 2.0 simulates the sample read pointer movement. Therefore, status/pointer queries to swapped buffers respond as though the buffer was still playing. Buffer swap status is also available to the application, if needed.
- Advanced prioritization heuristic

  Ensures that the most important sounds get played.

All of these features are provided transparently to the application developer — no special function calls are necessary to utilize any of these baseline features.

The advanced resource manager also provides a simple, powerful extension to the original resource manager: application-specific priority. By specifying a priority value for each buffer, the developer retains a high degree of control over the relative importance of each sound, while still leaving the mechanics of resource management to A3D. Priority can also be modified on the fly, allowing the importance of a sound to change as the acoustic scene changes over time. This small addition to the baseline API results in a more intelligent resource management, and hence a more realistic acoustic environment.

The resource manager also includes A2D, Aureal’s fallback 3D sound engine. A2D is a highly optimized, software-based, 3D sound rendering and mixing engine which sits on top of the sound card driver. The net effect is that A3D 2.0 provides 3D functionality for all generic sound cards. If an A3D application is run on a system without 3D sound hardware, A2D is automatically enabled. Therefore, the application developer can write to a single API for all sound card hardware. A2D is described in greater detail later in this document.

Methods of Direct Path Source Rendering

The RM optimizes direct path and reflection audio resources for audio scene rendering. Direct path resources are called *channels*. A channel represents an individual source rendering pipe for real-time 3D spatialization. Three different channels are available: *hardware*, *software,* and *virtualized*.

Table 1. Rendering Channel Overview

	Hardware 3D	**Software "A2D"**	**Virtualized**
Audible[a]	yes	yes	no
Preference[b]	highest	mid	lowest
No. Available[c]	system-specific	application-controlled	infinite
CPU Overhead[d]	system-specific	determinant	negligible

a. Audible means the channel actually renders the audio input to the speaker outputs.

b. Preference is the relative measure of channel value vs. the other channels

c. No. Available is the number of channels available for concurrent rendering.

d. CPU Overhead is a general gauge of MIPS utilization.

Hardware (A3D/DS3D)

Hardware channels are channels that are rendered and mixed by the 3D sound card. Hardware is considered the premium, or most preferred, rendering channel type, therefore the resource manager optimizes the audio scene by allocating hardware channels to the most *important* sources (described below).

The resource manager can only use as many hardware channels as the target sound card supports. If the developer prefers to limit the total number of hardware channels used, regardless of hardware availability, the application can call **Root::SetNumHWBuffers()** to set the maximum number of open hardware channels**.**

DS3D Support

A3D 2.0 can utilize DS3D-enabled sound cards for hardware rendering, though advanced A3D features such as reflections and atmospheric effects are not available. A3D 2.0 automatically utilizes any available DS3D hardware if the end user system does not have an A3D sound card installed.

Aureal is committed to providing an optimal audio engine for all sound cards, in the form of A3D 2.0. A3D 2.0, however, can’t gauge the performance or audio quality of third-party DirectSound3D sound cards. We are committed to making A3D 2.0 perform well with all DS3D sound cards; however, just like any other DS3D audio application, A3D 2.0 cannot directly control the algorithms used for spatialization, Doppler, and distance modeling. Therefore, problems may arise utilizing some DS3D sound cards with A3D 2.0. These problems include:

- Drivers that take significant CPU resources, dramatically reducing overall performance of system.
- Audio effect is not as expected — distance model is incorrect, or audio is clicky, etc.
- Hardware not matched well with A2D.

Considering the unpredictable differences between generic DS3D hardware products, the end result of the DS3D support can’t be guaranteed and diligence through application testing is recommended. This testing should be nothing more or less than what would be necessary had you used DS3D directly. The A2D rendering engine can provide much more predictable and constant results and serves as an alternative should problems arise.

A2D

A2D is Aureal's real-time software 3D audio and mixing engine for use with A3D 2.0 titles. It is embedded in the A3D 2.0 API DLL, and is transparently used in lieu of, or in conjunction with, 3D sound hardware. The developer can easily utilize A2D for multiple purposes:

- A3D 2.0 is a 3D audio scene rendering system. End user systems, however, may not support 3D sound — either the sound card is a legacy ISA card, or one of the limited number of PCI sound cards that do not support A3D or hardware DirectSound3D.
- A3D 2.0 can be used to "fill out" the number of 3D channels supported in hardware. For example, sound cards that only support five 3D sources could be increased to sixteen or more, via A2D.
- As an option, A2D can be used instead of actual 3D sound card hardware, where either performance or audio quality of the sound card is unknown.

The A2D effect includes direct path spatialization, Doppler, occlusions, and distance (gain) modeling. It does not include reflections or atmospheric EQ effects. A2D is highly optimized for efficient 3D rendering on host systems, and generally utilizes less CPU resources than standard software mixers.

The resource manager always utilizes any available hardware channels before falling back to A2D.

Usage

A2D is, for the most part, transparent to the application developer. A2D is automatically utilized to "fill out" the number of 3D hardware channels. The number of A2D channels can also be controlled via **Root::SetNumFallbackBuffers()**, where 12 is the default setting and 64 is the maximum. Setting this to a value larger than 64 will cause the call to fail. Valid A2D channels only consume CPU bandwidth when they are playing audio. Though the consumption is minimal, the number of A2D should be limited to only render audibly-important sources. A good benchmark is 16-32 A3D and A2D sources, total.

Virtual Channels

Playing sources which are not allocated to hardware or A2D (i.e., the least important playing sounds, when more sources are playing than hardware/A2D channels are available) are attached to *virtual channels*. Virtual channels are inaudible, however the sample read pointer is still updated as though the source is playing. Since static sources are always attached to hardware or A2D, only managed sources are virtualized. The CPU cost of virtual buffers is negligible.

Determining a Source's Render Method

Applications can call **Source::GetStatusEx()** to determine a source's current render mode. Rendering models differ for static buffers and dynamic buffers. Static buffers do not change render modes after allocation. Dynamic buffers, however, are completely controlled by the resource manager, where render mode is determined by source importance and the rendering resources available: hardware, software A2D, and virtualized. Since relative importance changes over time (due to audibility and priority changes, and sound sources starting and finishing), the resource manager bumps managed buffers from one particular render mode to another, as needed, to ensure the most important sources are rendered using the most optimal renderers.

The Resource Manager Software Architecture

The A3D 2.0 resource manager aids the application developer by mapping any arbitrary sound-scene to the specific, and limited, resources of any given end user's platform. The application may open, and concurrently play, any number of managed buffers. The resource manager dynamically maps the most important managed sounds to hardware or software rendering engines. The RM performs this dynamic mapping by streaming the audio source data into the hardware channels, in the background of the application. Alternatively, static, or unmanaged, buffers can be created that bypass the streamer engine, and take direct advantage of available hardware. These can be used when the limitations of the streaming engine are undesirable (for example, when an application developer prefers to use a custom resource management engine). FIGURE 1. illustrates the basic block functionality of the resource manager.

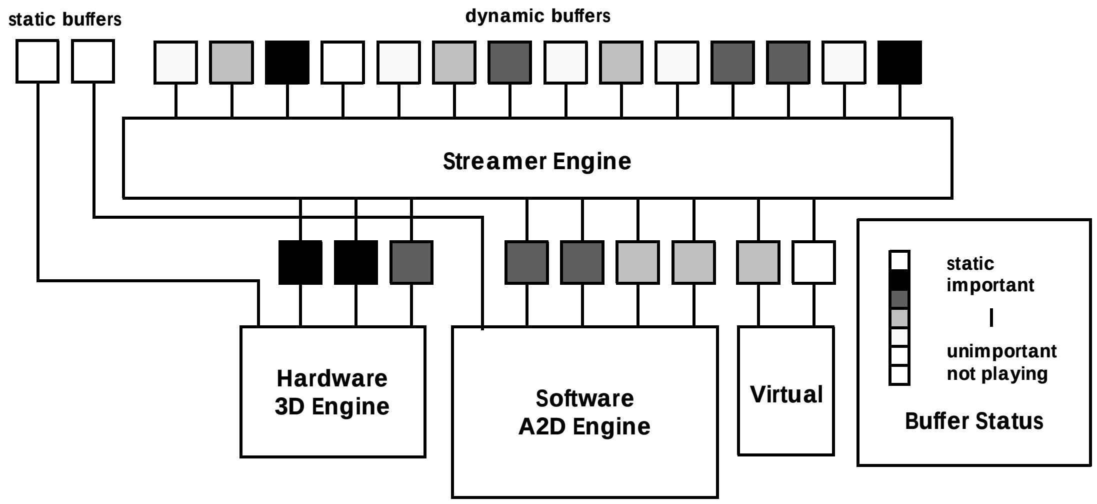

FIGURE 1. **Resource Manager Block Functionality**

Every 100 milliseconds, the resource manager performs the following tasks to optimize the audio scene:

1. Calculate the importance of each buffer (see below).
2. Build list of playing buffers that are sorted by importance.

   (Where T = the Total number of sources in this list and where T = M + N +V).
3. Determine number of hardware (M) and A2D channels (N) available.
4. Attach first M sources to the M available hardware channels, detaching less important sources as needed. This leaves T - M sources unattached.
5. Attach next N sources to the N available A2D channels, detaching less important sources as needed. This leaves T - (M + N) sources unattached.
6. Attach rest (V) of playing sources to V virtual channels. Virtual channels are inaudible, though the buffer read pointer changes as though the buffer was still playing. Thus all T sources have been attached.

Every 10 milliseconds, the streamer copies the appropriate audio data to the corresponding hardware/ A2D buffer, update read pointers, etc.

Importance

Importance is a measure of the merit of hearing a particular sound in the sound scene. For example, a loud explosion near the listener would have a high importance, but a bird chirping in the distance would probably have a low importance. Importance of each buffer is periodically calculated by the resource manager, and used to determine which buffers are rendered.

The formula for source importance is:

Equation 1a: Source is not Playing

$$\text{Importance} = -1 \times \infty$$

Equation 1b: Source is Playing

$$\text{Importance} = \text{Audibility} \times \text{Weight} + \text{Priority} \times (1.0 - \text{Weight})$$

Audibility, priority, and weight are described below.

Audibility

Audibility is the time-varying loudness, based on three parameters:

- Source volume
- Distance from source to listener
- Amount of occlusion between source and listener

Audibility is calculated internally by the A3D 2.0 engine, and indirectly controlled by the application (by varying source volume, position, etc.). Audibility ranges from 0.0 – 1.0, where 1.0 is a full-gain, unoccluded source within minimum-distance of the listener.

Priority

Priority is the application-specified “relative importance” of a sound. For example, in a “shooter” game, an attacking monster’s growl generally has a higher priority than a background waterfall effect, since it is more apt to fixate the player’s attention, even if the waterfall is louder.

The application sets the priority of each source by calling **Source::SetPriority()**. The new priority value takes effect on the next importance calculation, within 100 ms. Priority ranges from 0.0 – 1.0. By default, priority is set to 0.0.

Weight

In the Equation 1b, importance is the weighted-average of priority and audibility; this *weight* is directly specified by the developer’s application, by calling **Root::SetWeight()**.

When calculating importance, weight is defined as the weighted-average term for all source priority values.   Conversely, (1.0 – weight) is the weighting-average term for all source audibility values. For example, a weight of 0.5 would apply priority and audibility equally.

The application sets the source-global weight by calling **Root::SetWeight()**. The new weight value takes effect on the next importance calculation, within 100 ms. Weight ranges from 0.0 – 1.0. By default, weight is set to 0.5.

Source Types

The resource manager, or RM, supports two RM-specific modes that qualify level of resource management: *managed* and *static*. The application specifies the RM mode of a particular source via **NewSource()**, and it remains constant throughout the lifetime of the source. RM modes apply only to the direct path of the source; reflections are handled separately, described below.

Managed Sources

Sources created with the RM_MODE_STATIC flag **cleared** are *managed sources*. All managed sources share the available hardware and A2D resources, as described above. Resources are allocated, in real-time, to the most important sources. Source importance is based on both the audibility, or loudness, of the source, and the source priority, which is specified directly by the application.

Creating Managed Sources

Applications create a managed source by calling **NewSource()** with the RM_MODE_STATIC field of *dwFlags* clear.   Since a managed source is not attached at creation, to a particular software or hardware channel, both ALLOCATE_SW and ALLOCATE_HW fields must be clear.

Managed sources won't fail creation, or wave allocation, due to sound card or A2D limitations. Therefore, applications may open many more RM-managed sources than an end user's sound card, and/or A2D renderer, supports.

Static (Unmanaged) Sources

Sources created with the RM_MODE_STATIC flag set are RM-static sources. The direct path of a RM-static source is entirely unmanaged by the resource manager, giving complete control to the application. This mode should be used when playback of a particular wave file must be guaranteed, or when the overhead of resource-management is undesirable.

Creating Static Sources

An RM-static source is created during **NewSource()** by ORing the *dwFlags* field with RM_MODE_STATIC. ALLOCATE_SW and ALLOCATE_HW flags can also be set (mutually exclusive) to force allocation to hardware or software. Without these additional flags, a source is created in hardware, if it is available, otherwise in software.

Static sources can fail allocation if the hardware and/or A2D resources are not available.

Note: Static source allocation failure is reported via **IA3dSource::LoadWave()** or **IA3dSource::AllocateWave()**, not **IA3d4::NewSource()**! Make sure the static buffer succeeds the **Source::LoadWave() Source::AllocateWave()** call before assuming hardware is allocated.

A3D 2.0 uses the following heuristic to determine if a static source fails creation:

In **Source::LoadWave()** and **Source::AllocateWave()**:

```
if ((dwFlags & ALLOCATE_SW) && (dwFlags && ALLOCATE_HW))
   return fail

if !(dwFlags & ALLOCATE_SW)

{
   attempt to allocate hardware source
   if success
      return success
   else if fail and (dwFlags & ALLOCATE_HW)
      return fail

}

attempt to allocate A2D source

if success
   return success

else return fail
```

Once a static buffer succeeds the **LoadWave()** or **AllocateWave()** call, the hardware or A2D resources remain allocated until the wave is released.

Reflection Resource Management

The resource manager optimizes the hardware utilization of reflections for both static and dynamic sources. When the end user system supports reflections, the geometry engine performs the calculations necessary to specify and control the reflections in real-time. These reflection controls are sent to the resource manager, where they are mapped to hardware. Since the application may specify more audible reflections than the hardware supports, the resource manager allocates reflection channels to the most important reflections. Reflection importance is defined as its individual audibility.

For example, Source A is playing with four reflections. The corresponding reflection audibilities are: 0.25, 0.5, 0.75, and 1.0. Source B is playing with 3 reflections with audibility values of 0.33, 0.66, 0.99. Assuming the hardware only supports 4 reflections, the reflections from each source would be: A-1.0, B-0.99, A-0.75, and B-0.66. These reflections are rendered by the sound card until next 100 ms timer tick, where audibility-based reflection optimization happens again.