VRWORKS AUDIO SDK

DU-08562-001_v01 | May 2017

Overview

DOCUMENT CHANGE HISTORY

DU-08562-001_v01

Version	Date	Authors	Description of Change
01	May 2017	AS	Initial release

TABLE OF CONTENTS

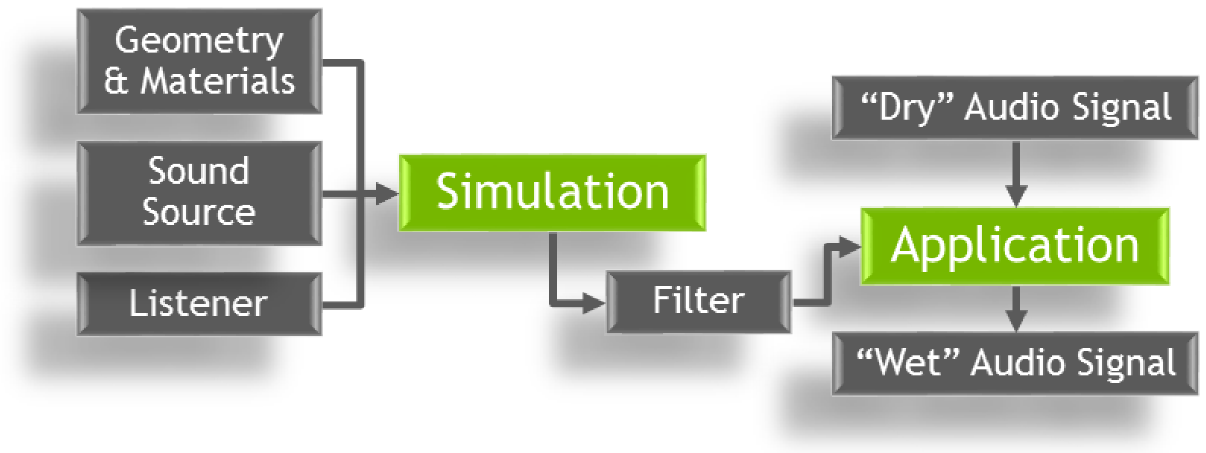
VRWORKS AUDIO SDK OVERVIEW

The NVIDIA VRWorks Audio Software Development Kit (SDK) provides the ability to add path-traced audio to applications, games, and virtual reality (VR) experiences. This guide provides a conceptual overview of the VRWorks Audio SDK API and explains how to use the functions in this API. For details of the functions and parameters in VRWorks Audio SDK API, refer to the VRWorks Audio SDK Reference.

INTRODUCTION TO PATH-TRACED AUDIO

Path-traced audio uses ray-tracing techniques to model the propagation of acoustic energy through a 3D space. Directional audio aims to give a listener information to allow him or her to locate the sound source. This information is most often computed only for the direct path: a straight line through 3D space between the source and the listener.

Path-traced audio follows the sound's direct path as well as many indirect paths which may carry acoustic energy from the source and listener. In many environments, such as cities and indoor environments, more acoustic energy may reach the listener from indirect paths than from the direct path. VRWorks Audio provides the tools to make these environments sound more realistic by following both direct and indirect acoustic paths.



Path-traced audio involves these steps:

1. **Simulation**, which is also called **tracing** in VRWorks Audio, is the process of finding paths through the specified 3D geometry between the sound source and the listener. The product of the simulation step is an auditory filter which affects an audio signal in the same manner as its propagation through the space.
2. **Application**, which is sometimes called **auralization**, is the application of that filter to the audio data. The audio data before the application of the filter is called the 'dry' signal in audio production terminology, and the audio which results from the application of the filter on the dry signal is referred to as the 'wet' signal. VRWorks Audio provides functions for both simulation and application.

PREREQUISITES

Hardware and system requirements for VRWorks Audio are as follows:

- **Hardware -** Windows-compatible PC with the following processors:
  - **GPU -** NVIDIA Maxwell or later
  - **CPU -** Intel core i7 or equivalent
- **Operating system -** 64-bit Windows 7, Windows 8, or Windows 10
- **Development environment -** Visual Studio 2013
- **Display driver -** NVIDIA display driver 378.*xx* or higher

SDK SETUP AND SAMPLE APPLICATIONS

To get started using the VRWorks Audio SDK, set up the include files, library files, and dynamic linked libraries (DLLs), and build the examples.

Include File

To compile the VRWorks Audio functionality into your project:

1. Add the `include/` directory where you installed the SDK to the include path list of your Visual C/C++ project or make file.
2. Add the following include directive to the relevant files within your project.
   ```
   #include <nvar.h>
   ```

Library File

Projects which use NVAR must include `nvar.lib` during the link step. You can meet this requirement in one of the following ways:

- Add the `lib/` directory where you installed the SDK to the library path list.
- Add `nvar.lib` to the linker input or add the following pragma directive: to one of the source files.
  ```
  #pragma comment(lib, "nvar.lib")
  ```

DLLs

When testing or deploying an application that uses VRWorks Audio, ensure that the following `.dll` files are present in the application directory or installed into a location within the system's `$PATH` variable:

- `nvar.dll`. – The NVAR library binary
- `optix.1.dll`. - NVIDIA's OptiX GPU ray tracing library

Sample Applications

The VRWorks Audio SDK includes these sample applications:

- `NvAudioBasic` is a simple demonstration of using VRWorks Audio to create a filter and apply it to an audio file. It renders a source and listener within a simple box geometry and applies the resulting filter to an input .wav file and writes the output to a .wav file.
- `NvAudioDemo` is an interactive, game-like demonstration which loads a geometry from a file and allows the user to move the source and listener location within this geometry and experience VRWorks Audio interactively. This sample serves to illustrate how VRWorks Audio fits within a typical interactive application structure.

The SDK contains a Visual Studio 2013 solution called `Samples_vs2013.sln` that builds the sample applications.

NVIDIA ACOUSTIC RAYTRACER (NVAR)

VRWorks Audio functionality is accessible to applications through the NVIDIA Acoustic Raytracer (NAVR) application programming interface (API). NVAR is a GPU-accelerated acoustic path-tracing solution which makes use of the ray tracing hardware capabilities of NVIDIA GPUs.

NVAR API Return Codes

NVAR functions always return an error code of type `nvarStatus_t` that indicates either the status of the call or an error state of the library.

Functions for Passing Scene Information

Most of the functions which pass scene information to NVAR have both a 'Set' and a 'Get' function. The exception to this is mesh data, which cannot be retrieved. In the interest of brevity, only the set routines are described in this document. For a full list of functions provided by VRWorks Audio, refer to the VRWorks Audio API Reference.

APPLICATION STRUCTURE

The functions of the NVAR API can be categorized according to what part of the application will likely use them. A subset of NVAR functions and where they are used are shown in the following table.

Application Phase	**Description**	**Functions**

Initialization	Application startup, initializing library, querying system configuration	`nvarInitialize()`
		`nvarCreate()`
		`nvarGetDevices()`

Scene Setup	Pass geometry and material information to NVAR	`nvarCreateMaterial()`
		`nvarCreateMesh()`
		`nvarCommitGeometry()`

Game Loop	Create sound sources, change listener location or orientation, change source location, start a new acoustic trace, apply the filter generated by the trace to an audio stream.	`nvarTraceAudio()`
		`nvarCreateSource()`
		`nvarSetListenerLocation()`
		`nvarSetListenerOrientation()`
		`nvarSetSourceLocation()`
		`nvarApplySourceFilters()`
		`nvarSetMeshTransform()`

Termination	End of application, clean up	`nvarDestroy()`

During the game loop phase, operations must happen quickly and interactively. Some setup functions are expensive and should be used sparingly, if ever, during the game loop. Because NVAR is designed for real-time acoustic tracing, many functions that reconfigure the scene can be called without restriction during the game loop.

INITIALIZATION

Applications must initialize the library and create an NVAR context.

Initializing the Library

The library is initialized by calling `nvarInitialize()`. This function must be called before any other library functions are called.

Creating an NVAR Context

Before any scene or source setup can be performed, an NVAR context must be created by using `nvarCreate()`. An NVAR context is created on one and only one GPU and contains all the necessary processing state for one scene and its geometry and sound sources. The application passes the GPU NVAR will use to carry out acoustic processing as a parameter to `nvarCreate()`. NVAR provides functions to query the topology of the system:

- `nvarGetDeviceCount()`
- `nvarGetDevices()`
- `nvarGetDeviceName()`
- `nvarGetPreferedDevice()`

`The nvarGetPreferedDevice()` function optionally accepts a `DXGIAdapter` argument and returns a preferred device which is not running DX graphics if available.

NVAR contexts are created with a name. If no name is provided, a default context is returned. If `nvarCreate()` is called for the same context name more than once, including the empty name, later calls will return the same context as the first call. This behavior enables code running in different areas of the application, such as the geometry system and the audio system, to both use the same context without the need to share memory.

SCENE SETUP

Scene setup conveys information about the 3D scene to the NVAR API. In some applications, setup may occur only once. In other applications, such as a multi-level game, setup may occur multiple times.

During scene setup, an application specifies acoustic materials and geometry to NVAR. The geometry consists of one or more 3D meshes. A single acoustic material is applied to an individual mesh. Different meshes may use different acoustic materials.

Specifying Acoustic Materials

Acoustic materials are created with the `nvarCreateMaterial()` or `nvarCreatePredefinedMaterial()` functions. `nvarCreatePredefinedMaterial()` creates a material with preset parameters for a set of common materials. The material for a mesh must be created before adding a mesh to the scene.

Material properties can be changed interactively within the game loop at very low cost:

- Call `nvarSetMaterialReflection()` to change reflection coefficient.
- Call `nvarSetMaterialTransmission()` to change the transmission coefficient.

To respect conservation of energy, the material reflection and material transmission coefficients must have a sum which is less than or equal to 1.0. NVAR does not enforce conservation of energy.

Specifying Geometry

Scene geometry is added by using `nvarCreateMesh()` and removed by using `nvarDestroyMesh()`. It is most efficient if scene geometry is added before tracing is started. However, meshes can also be added and deleted within the game loop with these calls.

When all of the geometry has been added to a scene, `nvarCommitGeometry()` can optionally be called to compute internal geometry data structures before time-critical processing in the game loop begins. Calling `nvarCommitGeometry()` is not necessary, but the first audio trace will be slow if it is not called first.

GAME LOOP

The game loop is the interactive phase of a game or VR application when the user is playing. During the game loop, the application must inform NVAR of changes to the state of the listener, sound sources, and 3D scene. Starting acoustic traces and applying the filters that a trace creates also occur during the game loop.

Listener

To change the location and orientation of the listener as the listener moves through the 3D scene, call `nvarSetListenerLocation()` and `nvarSetListenerOrientation()`.

Sound Sources

Sound sources may be created or destroyed any time after the NVAR context has been created, including within the game loop:

- Create a sound source by calling `nvarCreateSource()`.
- Destroy a sound source by calling `nvarDestroySource()`.

Sound source parameters other than the preset effect can be adjusted between traces. For more information about sound source parameters, see “Configuring a Sound Source,” on page 9.

Performance Note

The audio trace time increases with the number of sources. For maximum performance, sources which can be grouped should be treated as a single NVAR source. For example, in a first-person shooter game, all sounds originating from the player location can be combined into a submix, and this mixed signal used as the input to `nvarApplySourceFilters()`.

Changing Scene Setup

The scene geometry and materials can be changed at any time. Material reflection and transmission coefficients can be changed for existing materials by using `nvarSetMaterialReflection()` and `nvarSetMaterialTransmission()`. The material applied to a mesh may also be changed by using `nvarSetMeshMaterial()`. Meshes within the scene may also be moved by changing their transform matrix by using `nvarSetMeshTransform()`.

The functions described in the paragraph above are extremely lightweight and can be used as often as needed. Additionally, `nvarCreateMesh()` and `nvarDestroyMesh()` can be used to add new meshes to the scene at any time. Major additions or deletions may result in extensive changes to internal geometry representations, which may lead to an unusually long execution time for the `nvarTraceAudio()` function which follows calls to `nvarCreateMesh()` or `nvarDestroyMesh()` which add or delete large amounts of geometry.

Starting an Acoustic Trace

After the materials and geometry have been specified, the user location and orientation set, and at least one source has been created, NVAR is ready to trace acoustic energy through the scene. The function `nvarTraceAudio()` launches an acoustic trace. This function is asynchronous with respect to the calling thread. Therefore, it may safely be called from the rendering thread without stalling graphics draw calls or other work originating from that thread. `nvarTraceAudio()` schedule a trace and returns immediately.

`nvarTraceAudio()` accepts a Windows event handle which will be signaled when the trace has completed. When calling `nvarTraceAudio()`, applications should ensure that another trace is not already scheduled or running by checking the state of the Windows event passed to the previous call to `nvarTraceAudio()`. Calling `nvarTraceAudio()` faster than traces can be completed creates a backlog of traces that may interrupt real-time operation.

Using Trace Results

An NVAR sound source is not tied to a specific audio stream until the filter generated by the acoustic ray tracer is applied to an audio input stream. The result of a trace is a set of filters, one for each output channel, e.g. left and right. The input audio is assumed to be a single, mono, audio stream as physical sound sources are naturally monaural.

To apply the filter generated by NVAR to an audio stream, call `nvarApplySourceFilters()`. This function retrieves the latest filter set generated for the specified source and applies them to the audio stream. This function is expected to be called asynchronously with respect `nvarTraceAudio()` – there is no danger in calling `nvarApplySourceFilters()` while a trace is executing or scheduled. In typical use cases, `nvarApplySourceFilters()` is called from a signal processing chain driven by a thread created by the sound engine or sound output device.

If an application has another use for the filter or has its own convolution implementation, the filter data may be retrieved with `nvarGetSourceFilters()`.

The filters generated by NVAR are typically much larger than the individual source buffers passed to `nvarApplySourcefilters()`. NVAR maintains signal history and state within its internal convolution implementation. A single NVAR source should only be used for a single audio stream. Changing the audio stream or using the sharing the same NVAR source for multiple audio streams causes audio discontinuities.

TERMINATION

When a game level is unloaded, the NVAR state should be cleaned up by calling the corresponding `nvarDestroy()` function for all `nvarCreate()` functions called during setup or during the game loop. When the NVAR context is destroyed, all its contents are destroyed and any handles to its contents are invalid. However, if multiple locations have called `nvarCreate()`, the NVAR context will not be destroyed until the number of calls to `nvarDestroy()` equals the number of calls to `nvarCreate()` for a specific context. `nvarFinalize()` closes down the library. After calling `nvarFinalize()`, an application can no longer use any NVAR functionality.

CONFIGURING A SOUND SOURCE

At any point, an application can add a new sound source to NVAR by using the `nvarCreateSource()` function or remove a sound source by using the `nvarDestroySource()` function. When a source is created, the application must specify the effect preset to be used. NVAR has three presets for the level of effect desired on a sound source:

- `NVAR_EFFECT_LOW` – Subtle reverb, fast acoustic attenuation
- `NVAR_EFFECT_MEDIUM` – Realistic reverb and direct sound mixture
- `NVAR_EFFECT_HIGH` – Accentuated reverb, long-lived acoustic energy

Other than the effect preset, a source is created with default parameters. The application must call additional 'Set' functions to change these parameters. The following table summarizes these parameters and lists the functions for changing them.

Source Parameter	**Description**	**Functions**
Location	Position of a sound source in the 3D scene	`nvarSetSourceLocation()`
Direct Path Gain	A gain [0.0 – Inf) applied to the direct path between the listener and the specified sound source.	`nvarSetSourceDirectPathGain()`
Indirect Path Gain	A gain [0.0 – Inf) applied to all indirect paths between the specified sound source and the listener.	`nvarSetSourceIndirectPathGain()`

These parameters can be changed as often as needed. For example, in a typical usage scenario, the game loop changes the location of a sound source as often as every frame, but may set the direct and indirect path gains only once (and only if needed) when the source is created.

CONTEXT CONFIGURATION PARAMETERS

Some NVAR parameters apply to every source within a context.

Compute Preset

When an NVAR context is created, a compute preset must be passed to the create function. The compute preset dictates the amount of computational resources NVAR should use in calculating direct and indirect paths. There compute presets are as follows:

- `NVAR_COMPUTE_HIGH` uses more rays through more bounces to create a more dense and realistic filter. The use of more rays through more bounces incurs the cost of more GPU computations, higher memory consumption and longer trace times. `NVAR_COMPUTE_HIGH` is ideal for systems in which a GPU other than the GPU rendering graphics can be used for NVAR.
- `NVAR_COMPUTE_LOW` uses fewer rays through fewer bounces to create the filter. The priority of this compute mode is to complete the traces rapidly so they can be interspersed between frame renders on a single GPU.

After the compute preset for a context has been set, it cannot be changed.

Decay Factor

NVAR continuously refines the acoustic simulation across multiple calls to `nvarTraceAudio()`. Decay factor controls the interval over which the acoustic simulation is refined. A higher decay factor creates filters that incorporate a larger history of acoustic simulations, but reacts more slowly to changes in listener position, source position, or geometry changes. The decay factor can be in the range 0.0 to 1.0. A decay factor of 0.0 uses only the most recently discovered acoustic paths, while a decay factor of 1.0 uses all paths ever found. Decay factor should be less than 1.0 in gaming and VR applications.

Unit Length

For correct physical simulation, the scale of the geometry passed to NVAR must be established. The function `nvarSetUnitLength()` is used to give NVAR a reference distance. The argument to `nvarSetUnitLength()` is the length, in meters, of a unit vector, that is, a vector with a length of 1.0, within the geometry assets passed to `nvarCreateMesh()`. For example, if the geometry is created in units of centimeters, the value passed to `nvarSetUnitLength()` should be **0.01**, because there are 0.01 meters in a unit length of the geometry, a centimeter.

Reverb Length

NVAR generates reverbs that are finite impulse response (FIR) filters. The total length of the filter can be set with `nvarSetReverbLength()`. This function takes the reverb length in seconds. The length of the filters generated by NVAR is affected by this parameter, so changes cause reallocation of internal buffers, which may be expensive. This function should be called during application initialization or scene setup and the reverb length changed as infrequently as possible.

Sample Rate

To create effects correctly, NVAR requires the sample rate of the audio streams to which the filters will be applied. This can be set with `nvarSetSampleRate()`. The length of the filters generated by NVAR is affected by the sample rate, and changing it results in reallocation of internal buffers, which may be expensive. This function should be called during application initialization or scene setup and the sample rate changed as infrequently as possible.

Output Format

The output format specifies the type of audio device that will render the audio. NVAR creates a set of filters consisting of one filter one per channel of the output format for each sound source. The output format must be specified when a context is created and cannot be changed. Only `NVAR_OUTPUT_FORMAT_STEREO_HEADPHONES`, which has two output channels, is supported.

ASYNCHRONOUS EXECUTION

To avoid stalling the game loop and graphics dispatch, many NVAR functions enqueue the operation to be completed and return. The operation is added to a command queue and control returns to the calling thread. The requested operation is then carried out asynchronously with respect to the calling thread. Commands issued will be executed in the order that they were issued.

An application may synchronize the internal command queue which executes enqueued operations by using the function `nvarSynchronize()`, which blocks the calling thread until the command queue has finished executing. The application can also add Windows events to the command queue by using `nvarEventRecord()`. The Windows event passed in through this function will be signaled once all work added to the command queue before to the call to `nvarEventRecord()` has completed.

The following functions execute asynchronously to the calling thread:

- `nvarTraceAudio()`
- `nvarCreateMaterial()`
- `nvareCreatePredefinedMaterial()`
- `nvarDestroyMaterial()`
- `nvarSetMeshMaterial()`
- `nvarSetMaterialReflection()`
- `nvarSetMaterialTransmission()`
- `nvarCreateMesh()`
- `nvarDestroyMesh()`
- `nvarSetMeshTransform()`
- `nvarCommitGeometry()`