The NASA Auralization Framework and plugin architecture

Aric R. Aumann[a]
Analytical Services and Materials, Inc.
Hampton, VA 23666-1340, USA

Brian C. Tuttle[b]
Analytical Mechanics Associates Inc.
Hampton, VA 23666-6143, USA

William L. Chapin[c]
AuSIM Inc.
Mountain View, CA 94043-1447, USA

Stephen A. Rizzi[d]
Aeroacoustics Branch, NASA Langley Research Center
Hampton, VA 23681-2199, USA

NASA has a long history of investigating human response to aircraft flyover noise and in recent years has developed a capability to fully auralize the noise of aircraft during their design. This capability is particularly useful for unconventional designs with noise signatures significantly different from the current fleet. To that end, a flexible software architecture has been developed to facilitate rapid integration of new simulation techniques for noise source synthesis and propagation, and to foster collaboration amongst researchers through a common releasable code base. The NASA Auralization Framework (NAF) is a skeletal framework written in C++ with basic functionalities and a plugin architecture that allows users to mix and match NAF capabilities with their own methods through the development and use of dynamically linked libraries. This paper presents the NAF software architecture and discusses several advanced auralization techniques that have been implemented as plugins to the framework.

[a] email: aric.r.aumann@nasa.gov
[b] email: brian.c.tuttle@nasa.gov
[c] email: wchapin@ausim3d.com
[d] email: stephen.a.rizzi@nasa.gov

1 INTRODUCTION

For over a decade, NASA-developed software has been used for the auralization of aircraft flyover noise. The software was built around a source-path-receiver paradigm. The Aircraft Source Noise Generator (ASoNG)[1] code was developed to synthesize source noise based on some prescribed definition. The Community Noise Test Environment (CNoTE)[1] suite of codes was developed to propagate the sound to a receiver based on physical models of the propagation path and receiver. The models produce a set of time-varying gains, time-delays, and filters (GTF). CNoTE utilizes the AuSIM3D[2] real-time digital signal processing (DSP) engine to apply the prescribed GTF. While this tool set was highly effective in generating sounds of present and future aircraft,[3] its lack of modularity inhibited its integration with NASA's Aircraft Noise Prediction Program (ANOPP2).[4] In addition, some compromises had to be made in the propagation modeling to operate in the real-time engine, even when real-time operation was not required.

This paper introduces the newly developed NASA Auralization Framework (NAF) which is intended to replace the synthesis and propagation path processes of ASoNG and CNoTE with a code base that is modular, extensible through a plugin system, and capable of operating in several host environments. Real-time listener simulation will still be realized with the combination of CNoTE and the AuSIM3D engine.

1.1 Framework Architecture

The NAF is a collection of software modules with functions and data common to auralization. Since the initial implementation of the NAF is intended for the Microsoft Windows operating system (OS), the NAF modules come in the form of dynamic-link libraries (DLLs). Future implementations of these modules will become available in other forms to support other operating systems. Users of the NAF write their own application code, using the NAF Application Programming Interface (API). Under the Windows OS, the NAF API allows users to access functions and/or data in one or more NAF DLLs by linking to the associated import libraries. The application code is written in the user's host environment, for example, a C++ executable or function call from MATLAB®. The NAF code base itself is written in C++.

The NAF consists of seven modules, as depicted in Figure 1. Inasmuch as possible, functions and data common to a particular set of operations are grouped with a module. There are two foundational modules; the NAFIPP and the NAFCore. The NAFIPP is a custom subset of Intel® Integrated Performance Primitives (IPP) redistributable libraries for common signal processing functions, e.g., Fast Fourier Transform related functions. The NAFIPP has no other dependencies. The NAFCore is dependent only on the NAFIPP, and contains common object definitions, containers, and functionality, e.g., sample interpolators needed for fractional delay lines.

The five remaining modules are all dependent on the NAFCore (see Section 6), and some also depend on the NAFIPP. The NAFGTF depends on the AuSIM AuGTF engine. Portions of the NAFScene, NAFPath, NAFSynth and NAFGTF modules are described in some detail in Sections 2-5, respectively. The NAFScheduler module is a placeholder at this time; however, in future releases, it will coordinate the activities of the other modules, through management of threads and order of operations, and by providing some "glue code" for moving data between modules.

Additionally, a system for developing, registering, and using plugins is implemented within the NAF. This allows users to create their own modules, which can act as 1) plugins to the NAF modules, e.g., a curved path finder DLL called by the NAFPath DLL, 2) complete replacements of NAF modules, e.g., a new NAFScene module, or 3) a completely new module, e.g., a NAFListener module for binaural simulation. It is through the NAF plugin system that users can

augment the skeletal framework of the NAF with more advanced capabilities. The plugin system is further described in Section 7. Finally, a typical use case is described in Section 8.

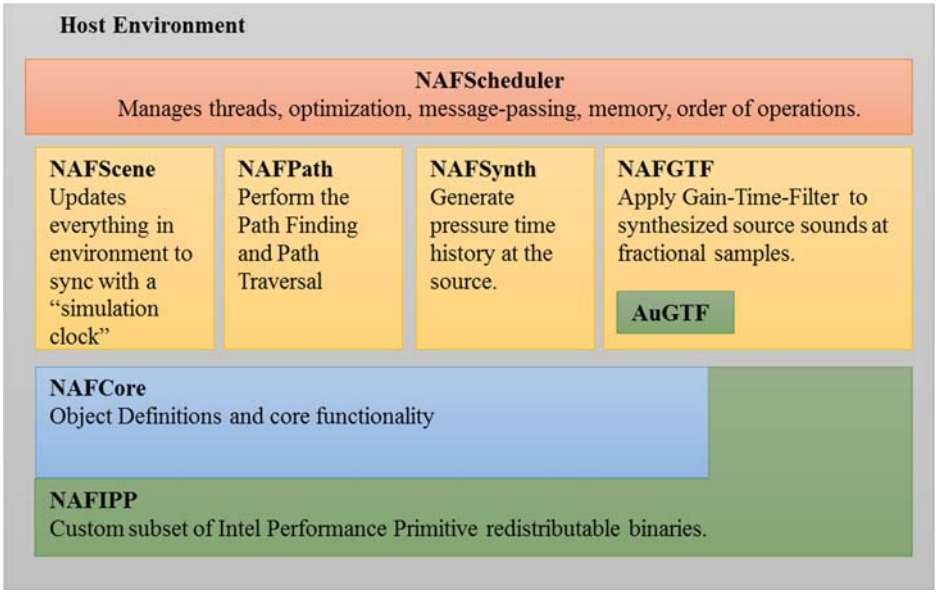

Figure 1 – Depiction of the NAF architecture.

2 SCENE MODULE

The NAF is intended to operate as a time simulation model using discretized information that generally includes time-stamped source and receiver position and orientation, source operating conditions (e.g., engine throttle setting), time-dependent path information, etc. It is the function of the NAFScene module to update these at each simulation time step, so that this information can be used in subsequent modules.

The built-in method for updating the parameters is via linear interpolation between the previous condition and the next condition. The current implementation assumes that source and receiver trajectories are known for all time at the start of the simulation. If either the source or receiver trajectories are not known a priori, then the built-in method must be replaced with a user-specified method using a model plugin.

3 PATH MODULE

The path module performs two primary functions: path finding and path traversal. While the path traversal operations rely on path finding, separating these two functions offers the potential for parallel computing, as will be described in Section 8, and for clean separation of plugins.

3.1 Path Finding

The path finding process produces essentially three pieces of information needed in subsequent processes. These are the source emission angles, the receiver angles, and the path itself. The source emission angles are used in source noise synthesis (see Section 4), the receiver angles are optionally used in listener simulation, and the path (defined by at least two time-stamped points) is used in the path traversal process (see Section 3.2).

Two approaches to path finding are possible: one that originates at the source and one that originates at the receiver. The approach that starts at the source may be acausal because the future

location of the receiver is not generally known, unless specified for all time at the start of the simulation. The approach that starts at the receiver is always causal because the past position of the source is known. Historically, NASA has used the first approach under the condition that the receiver location is fixed for all time. The NAF is intended to support a moving receiver. This is most readily accomplished by a path finding approach that starts at the receiver. If instead the approach starts at the source, it would necessitate either that the receiver position be known a priori, or that some other approach, e.g., predictor-corrector, be taken to establish the future receiver position.

At each scene update, all paths between the source and receiver are calculated using knowledge of the current source and future receiver positions (or past source and current receiver positions), and a definition of the environment (atmosphere, terrain, and any other relevant information). For a receiver near the ground, there are minimally two paths: a direct path and a ground reflected path. Depending on the environment definition, they may be additional paths, e.g., downward refracting atmosphere, or additional reflections off of non-flat terrain or building structures.

Built-in Path Finder Capabilities

The built-in path finder in the NAF is a very simple straight-line method that starts at the moving source and ends at a fixed receiver. The points returned by the built-in path finder for the direct path are the source and receiver positions. The points returned for a reflected path are the source position, the ground reflection point, and the receiver position. More sophisticated path finders, e.g., for a curved propagation path through a non-uniform atmosphere,[5] or even hardware accelerated methods[6] may be implemented through the NAF model plugin system.

3.2 Path Traverser

The function of the path traverser is to obtain a single gain, single time delay and list of filters (referred to as a "GTF-list") associated with each path segment, defined by the list of points that are determined by the path finder. Here, a segment is defined as a pair of consecutive path points. Hence, there are *n*-1 path segments for a path defined by *n* points. The gain, time delay and filters are obtained from functions in the NAFCore (see Section 6).

The complete traversal of a sound from the source emission position to the receiver position is thus represented by an ordered series of *n*-1 GTF-lists associated with the atmosphere segments and one GTF-list for each reflection. Together these are referred to as a "GTF-series." There is one GTF-series per path, and each carries with it a time-stamp for its time of emission according to the global clock. For computational efficiency, all GTF-lists within a particular GTF-series are collapsed (linear gains multiplied, time delay summed, and filters enumerated) to form a GTF-series data structure having a single GTF-list prior to submission to the GTF processor (see Section 5). The time of reception is thus determined by the summed delay in the collapsed GTF-series. This approach is consistent with that used in the AuSIM3D engine.

Finally, it is important not to overlook the fact that any change to the path, e.g., source or receiver movement, necessitates a reevaluation of the GTF-series. The GTF-series serves as input to the GTF engine.

Built-In Path Traverser Capabilities

Two possible sources of GTFs are considered in the built-in path traversal: the propagation medium and the ground. As an example, consider the direct and ground reflected paths obtained

with the built-in path finder. The direct path has a GTF-series composed of one GTF-list associated with the propagation medium alone, while the ground-reflected path has a GTF-series composed of three GTF-lists associated with the two propagation medium segments and one ground reflection. The propagation medium for the direct path includes a gain (attenuation) obtained via spherical spreading, a time delay obtained by the segment length divided by the speed of sound, and an atmospheric absorption filter (see Section 6.1). No GTF-series collapse is necessary for the direct path. For the ground-reflected path, the GTF-series is initially composed of three GTF-lists. The first GTF-list is associated with the propagation medium (gain, time delay, and atmospheric absorption) for the path segment between the source and the ground. The second GTF-list is associated with the ground plane and includes the filter and possible delay compensation (see Section 6.2). The third GTF-list is associated with the propagation medium (gain, time delay, and atmospheric absorption) for the path segment between the ground and the receiver. The collapsed GTF-series includes the summed gain, time delay and a filter list composed of the two atmosphere segments and the ground. Some additional efficiency can be gained by collapsing like-filters (e.g. the two atmospheric attenuation filters) in the frequency domain prior to conversion to individual finite impulse response (FIR) filters, since their sequential processing (see Section 5) effectively doubles the computational effort.

Additional sources of GTFs, for example, transmission through structures, can be accommodated via user-written model plugins.

4 SYNTHESIS MODULE

In describing the synthesis module, it is useful to first introduce some terminology to describe a hierarchy of noise generators. The lowest level noise generator is referred to as a noise component. In a turbofan engine, for example, there are multiple noise components including forward and aft radiated fan tones, forward and aft radiated fan broadband, core noise and jet noise. In the NAF, if a component definition is specified independent of other components, it is synthesized independently. For example, if the forward and aft radiated fan broadband components are combined, they are synthesized together, whereas if they are provided separately, they are synthesized separately. A compound source, or simply a source, is one in which one or more components can be considered as compact and originating from the same point. There are one or more unique paths between each source and receiver.

At each scene update, the synthesis engines query the component models at their instantaneous operational state (defined by NAFScene) and emission angle (defined by the NAFPath path finder). The component model returns its instantaneous noise definition. That definition is used by the component's synthesizer to generate a buffer of sound. This buffer typically serves as input to the GTF processor. The synthesis must be performed in such a way that the generated signal continuously evolves with changes in the noise definition. The manner in which that is accomplished is dependent on source type, i.e., broadband, narrowband, tonal, or time domain. If desired, components of the same source can be mixed together at the source position before being submitted for propagation processing.

Finally note that the number of component synthesis engines running at a time is equal to the number of paths times the number of independently specified components.

Built-In Synthesis Capabilities

NAFSynth provides three rudimentary component synthesizers: a test-tone synthesizer, a random noise synthesizer, and a wavetable synthesizer. The test-tone synthesis can generate a

single constant tone at a user-specified frequency. The random noise synthesis generates random samples out of a normal distribution with mean of 0 and standard deviation of 1. Wavetable synthesis loads a waveform from a user-specified file, and returns a buffer's worth of samples at each time step. All three synthesizers are omnidirectional.

NAFSynth provides two component models: a "null op" model and a directivity-file model. For some component synthesizers, like the rudimentary ones, the output does not depend on the operational state or emission angle, so an instantaneous noise definition is not necessary. In such instances, the built-in null op component can be used. To support the more complex component noise synthesizers incorporated in an advanced model plugin, NAFSynth provides a directivity-file database component. The directivity-file database component loads a series of directivity files and linearly interpolates over any specified independent variables as well as emission angles. This model currently supports directivities defined by 1/3-octave band levels (for broadband noise) or frequency-amplitude tables (for pure-tonal noise). If the loaded 1/3-octave band directivity file is Doppler shifted, the directivity-file component model will flag it for de-Dopplerization when it is later converted to a narrowband spectrum for synthesis, as indicated by Rizzi et al.[3] Tonal data is typically provided at the non-Doppler shifted blade passage frequencies.

Because users might want to load their own format directivity files, there is also plugin support specifically for directivity-file loaders. A sample project that builds a plugin to read NetCDF[7] directivity files is included as part of the NAF.

5 GTF MODULE

The NAFGTF module is composed of the DSP elements needed to apply the collapsed GTF-series associated with a path to the source signal. Most DSP theory is based on the assumption of linear time invariance (LTI). LTI systems allow operations to be commutative and invertible. As the scene is updated, any change in the path will violate this condition. Fortunately, LTI can be approximated by treating the system in a piecewise (in time) fashion. In practice, this entails sequential application of the buffered GTF-series over a number of source signal segments. At each segment transition, the change in time delay, filter (color), and gain is extracted from buffered GTS-series. The GTF processor applies these changes by first stretching/compressing the signal to accommodate the change in time, then smoothly adjusting the filter for change in coloration, and finally ramping the signal scale for the change in gain. Since LTI is assumed within each segment, the filters in each GTF list could be convolved into a single filter. However, the NAF keeps them as a succession of filters for the purpose of control.

5.1 GTF Engine

The GTF Engine, the top level object of the NAFGTF module, manages processing and memory resources for the digital signal processing core and serves as the interface to the rest of NAF. The GTF Engine creates an instance of the licensed AuSIM AuGTF engine. The AuGTF engine contains a dynamic list of GTF processors, each corresponding to a path in the simulation. When a GTF-series or synthesized signal block is submitted to the NAFGTF module, the GTF Engine ensures that a processor exists that corresponds to the path. The GTF Engine cleans-up unused processors if a path dissolves.

5.2 GTF Processing

Every path corresponds to a dedicated processor object to keep track of signal state from one block of signal to the next and to support parallel processing between data-independent paths. The

inputs to a GTF-processor for each processing time slice are a block of signal from the synthesis engine and a GTF-series from the path traverser. Upon submission, the block of signal is concatenated into a very long delay-line, while the GTF-series is submitted to a FIFO (first-in, first-out) queuing container. When asked to process, the first-out GTF-series in the FIFO queuing container is examined for the amount of signal required to produce a block of output. If enough signal exists in the delay-line, then the GTF-series is processed. A multi-algorithm sample interpolator extracts the required samples by the ratio of previous delay to the new delay. The samples are placed into a work buffer, where they are filtered in-place sequentially by the filter list of the collapsed GTF-series. The samples are scaled by the gain upon copy to an output buffer. The process is repeated until the end of the simulation. This processing scenario is depicted in Figure 2.

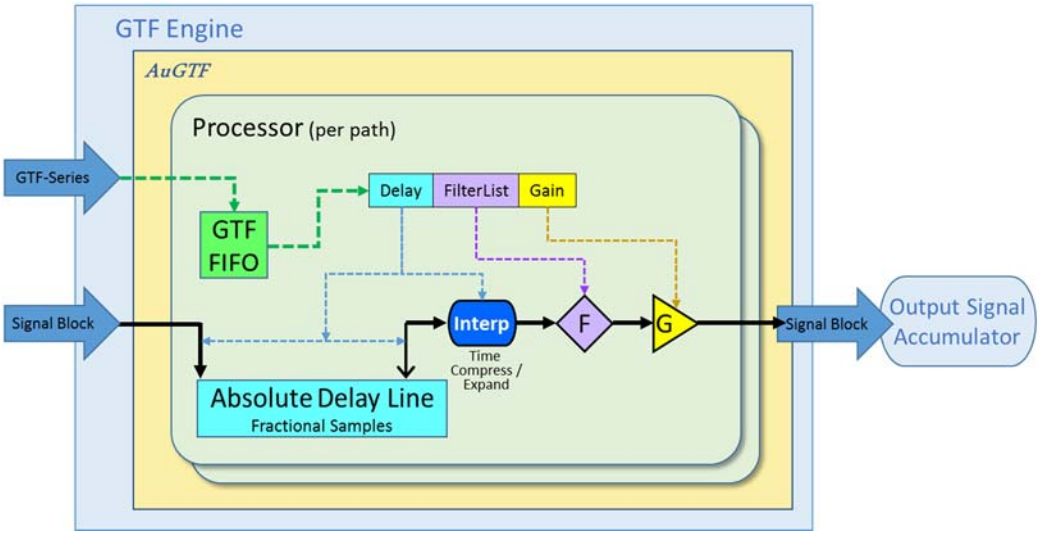

Figure 2 – Depiction of DSP operations in the NAF GTF engine.

6 CORE MODULE

As previously indicated, the NAFCore module is a container for common object definitions and functions. Not all are enumerated here; however, two of those objects which specify the built-in atmospheric absorption and ground plane impedance models are described below. As described in Section 3, the path finder and path traverser receive environment information, including the atmosphere model and ground model, as part of their inputs. Atmospheric models can independently return the speed of sound and the frequency response of the absorption along a path segment. Ground models return the frequency response of the ground impedance. The frequency responses are converted to FIR filters through a subsequent function call. The path finder and path traverser make use of these capabilities to generate the path and calculate the GTF-series.

6.1 Built-In Atmospheric Absorption

NAFCore provides a definition for a uniform atmosphere between the source and receiver. Included in that definition is a constant speed of sound and an atmospheric absorption frequency response specified in dB/m and defined at each of an extended set of standard 1/3-octave center frequencies. The speed of sound and absorption are pre-computed. By default, absorption is provided for standard uniform atmosphere having a nominal temperature of 15°C, atmospheric pressure of 1 atm., and relative humidity of 50%. This absorption was pre-computed using the methods embodied in the more versatile advanced model plugin, see Section 7.1. Alternatively,

users may define a custom atmosphere by supplying their own pre-computed speed of sound and atmospheric absorption. This adds some flexibility by allowing the user to define the absorption over an arbitrary set of frequencies. In either case, it is very expedient since it does not rely on a realization of a physical model during simulation.

The NAF path traverser uses the returned speed of sound to determine the time delay for sound to propagate over the path length segment. The NAF path traverser obtains the digital filter by multiplying the returned absorption frequency response by the propagation distance. This distance may be the entire propagation path from source to receiver or a smaller segment of that path depending on how the path is defined. The resulting total absorption over this distance is considered as the target frequency response for the filter design. This data is curve fit with a 2^n-point spline, evaluated at narrowband frequency increments, and converted to a minimum-phase FIR filter via a real cepstrum.[8]

6.2 Built-In Ground Plane Impedance

The NAF provides a simple ground impedance model for simulating a hard ground. A unity gain, zero time delay, and short "wire" FIR filter, e.g., [1, 0, 0, 0,], are returned to the path traverser for incorporation in its GTF-series.

7 MODEL PLUGIN SYSTEM

The NAF model plugin system allows users to extend the capabilities of the NAF by writing their code in the form of NAF plugins. The NAF model plugin system takes advantage of the "abstract factory" and "factory method" software design patterns as described by Gamma et al.,[9] with an extensible interface suggested by Reddy[10] for runtime registering of factory methods. This allows NAF plugins to be accessed at run time, provided the plugins register their factory methods with the appropriate abstract factory.

Users can create plugins to override default behavior for a number of NAF-defined classes. User plugins must provide a method that can create an object of their new class, i.e., their factory method, and register this method by name with an abstract factory. In order to create an instance, the abstract factory need only be supplied with the name of the registered factory method.

Abstract factories are provided for component noise models, synthesizers, atmospheric models, directivity-file loaders, one-dimensional interpolators, terrain models, as well as the full path finder and path traverser. Plugins can have more than one factory method, and should register all their factory methods on loading.

7.1 NASA Advanced Capability Plugins

In addition to developing the framework, NASA has converted previously developed advanced capabilities for synthesis, atmospheric absorption and ground attenuation to NAF plugins. These not only help verify the NAF implementation through comparison with the output from ASoNG and CNoTE, but also serve as a template for creating new techniques solely as NAF plugins.

Advanced Synthesis Plugin

Two different component synthesis engines are implemented in the NASA Advanced Synthesis Plugin: broadband and pure-tonal. The broadband synthesis engine uses the same subtractive synthesis technique, employing overlap-add, as used in ASoNG. It operates on the

instantaneous spectrum provided by the directivity-file component model. For brevity, the details are omitted (see Rizzi et al.[1]). At the end of each processing block, the resulting overlapped and added buffer is appended to the large output buffer of previously synthesized samples. Using this method, the broadband signal contained in that buffer can continuously evolve over time. The pure-tonal synthesis engine uses an additive synthesis method, wherein each harmonic is phase-tracked and interpolated in both frequency and amplitude on a per-sample basis to generate a signal that is contiguous across buffers and that continuously evolves over time to match the frequency and amplitudes specified by the directivity-file component model. This is also the same synthesis technique that has been used previously in ASoNG. A narrowband synthesis engine derived from the broadband engine is under development, as is the ability to apply low frequency oscillations (LFO) to any of the above. These methods were also previously implemented in ASoNG.

At this time, the directivity-file component model returns interpolations of a pre-computed directivity database to the synthesis engine. These databases are generated in advance by a prediction tool, e.g. ANOPP2, and loaded at run time. However, the NAF model plugin system provides the flexibility to either directly implement a component prediction model of its own, or query the ANOPP2 Observer object[4] at each simulation step to provide the instantaneous source spectrum corresponding to the interpolated operational state and emission angle.

Finally recall that the number of component synthesis engines running at a time is equal to the number of paths times the number of independently specified components. For example, a simulation of broadband nose-gear noise for a receiver near the ground would require two broadband synthesis engines: one for the direct path and one for the ground-reflected path. Further, all synthesis engines associated with the same component are (currently) synthesized with the same random attributes, i.e., the same broadband phase for broadband components, and the same tonal phase for tonal components. This works well in the above example when the two emission angles are very close together, and results in a direct path signal that (at the source) is coherent with the ground-reflected path signal. However, when the emission angles are far apart, some other approach may be necessary.

Advanced Atmosphere Plugin

The Advanced Atmosphere Plugin is derived from the atmospheric absorption model plugin in CNoTE,[8] which, in turn, is based on the ANSI standard atmospheric absorption model.[11] Operating in the same capacity as the built-in model (see Section 6.1), the plugin returns the speed of sound and absorption in dB/m evaluated at the 1/3-octave band center frequencies. These are dependent on user-specified parameters of temperature, relative humidity, and pressure.

A uniform atmosphere definition and two altitude-dependent atmosphere definitions, isothermal and lapse, are available. For the uniform atmosphere definition, the same user-specified parameters are used at all points along the path. For the isothermal atmosphere definition, the temperature and relative humidity remain constant, but the pressure varies hydrostatically with altitude. Thus, the isothermal atmosphere generates an altitude-dependent atmospheric absorption, but an altitude-independent speed of sound. For the lapse atmosphere definition, the lapse rate defines the variation of temperature with altitude. The pressure is taken to vary hydrostatically with altitude, and the relative humidity is held constant. In this case, both the atmospheric absorption and the speed of sound are altitude-dependent. Any speed of sound variation causes refraction and alters the sound propagation path,[5] however that effect is decoupled from the Advanced Atmosphere Plugin. A separate advanced plugin for the path finder is required to utilize that information for the generation of curved paths. For any of the atmosphere definitions, the generation of the resulting FIR absorption filter follows the process described in Section 6.1.

Advanced Ground Impedance Plugin

Work is underway to port an advanced ground impedance plugin[12] from CNoTE to a NAF plugin. This CNoTE plugin allows specification of two different ground impedance models[13, 14] and optionally applies a spherical wave correction[15] for low altitude sources.

The process for obtaining its digital filter entails an inverse FFT of the complex reflection coefficient, followed by a circular shift in the time domain. This process produces a linear phase FIR filter with its peak at the middle tap, so time delay compensation is required to synchronize the direct and ground reflected paths.[12]

8 PUTTING IT ALL TOGETHER

The NAF building blocks are assembled in a user-written code in a particular manner depending on its use. One time step of a typical use case is depicted in Figure 3.

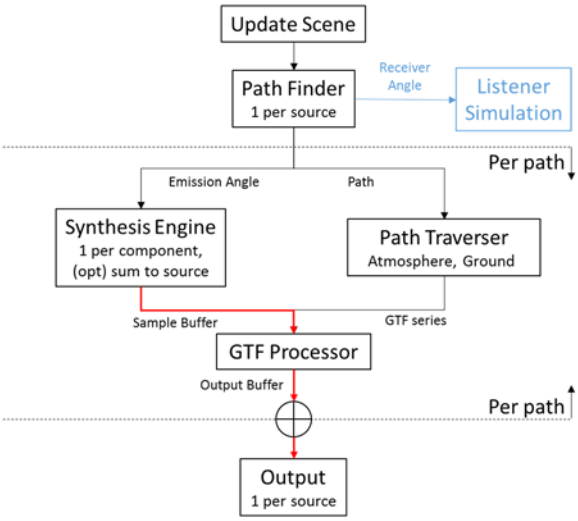

Figure 3 – Block diagram of a typical use case for the NAF. Black connecting lines indicate data and red connecting lines indicate samples (audio).

For each scene update, the path finder obtains the paths, emission angles and receiver angles for each source. Since the NAF does not have a listener model, listener simulation would be performed off-line using CNoTE. For each path, component synthesis engines generate sample buffers of pressure time history, which may optionally be summed back to the source level. In parallel, the path traversers generate the collapsed GTF-series for the specified atmosphere and ground. The GTF processor propagates the sound through application of the GTF-series to the sample buffer. The output of all the paths for a given source are shown summed to generate a pseudo-recording at the receiver. The summation of the direct and ground reflected paths create the comb-filtering effect previously noted.[1, 3, 8] The future role of the NAFScheduler in managing the operation of the threaded component synthesis engines and path traversers is not shown.

Note that the operations shown in Figure 3 are the same for both event-driven and clock-driven processes. In an event-driven process, the operations at each time step are processed for as long as they take to complete, and the result is pushed out at the end, presumably to a file. In a clock-driven process, the scene update rate is determined at a prescribed interval, presumably by a real-time clock on a sound card or other output channel. The operations are initiated, perhaps in a double-buffered scheme, and the output is pulled out when needed by the simulation, including gaps if processes are not finished. The NAFScheduler would adjust thread parameters, buffer

length, algorithm complexity, or other factors in real time to try to maintain throughput, but the operations are still completed in the same order for each scene update.

9 NAF RELEASES

There are three different NAF-related software releases: NAF-Dev, NAF-User, and NAF-Plugins. NAF-Dev is the developers' version of NAF, and includes source code for building all the built-in modules described herein. Developers are required to obtain licenses for two third-party dependencies to build NAF-Dev: the Intel® IPP library and the AuSIM GTF SDK. An optionally built NetCDF[7] directivity loader requires linking to the NetCDF link libraries. The NAF-Dev has been approved for general public release and is widely available through a NASA software usage agreement.

NAF-User is a planned binary release of NAF, and will include link libraries, dynamically linked libraries, and the C++ header files necessary for building applications that use NAF. NAF-Plugins is a planned binary release of the NASA advanced plugins.

10 DISCUSSION

The NAF is a new software API that replaces a significant portion of the auralization capabilities previously provided by ASoNG and CNoTE. It encompasses all processes required to generate a pseudo-recording given an acoustic definition of the source and description of the environment. The pseudo-recording can separately be used with the listener modeling capabilities in CNoTE to create a virtual environment. The NAF's basic palette of built-in capabilities can be augmented through the NAF plugin system. Advanced plugins for synthesis, atmospheric absorption and ground plane attenuation are NASA-written examples of this capability. Together, the NAF API and its plugins allow users to write more integrated application code than was previously possible. This will ultimately make auralization more readily available to users of noise prediction tools like ANOPP2 and the Advanced Acoustic Model.[16]

11 ACKNOWLEDGEMENTS

This research was supported by the National Aeronautics and Space Administration, Aeronautics Research Mission Directorate, Transformative Aeronautics Concepts Program, Transformational Tools and Technologies Project. The authors would like to acknowledge Dan Palumbo, NASA Langley Research Center, for his contributions to the early design of the NAF.

12 REFERENCES

1. Stephen A. Rizzi, Brenda M. Sullivan, and Aric R. Aumann, "Recent developments in aircraft flyover noise simulation at NASA Langley Research Center," *NATO Research and Technology Agency AVT-158 "Environmental Noise Issues Associated with Gas Turbine Powered Military Vehicles" Specialists' Meeting*, NATO RTA Applied Vehicle Technology Panel, Paper 17, Montreal, Canada, (2008).

2. "GoldServe, AuSIM3D Gold Series Audio Localizing Server System, User's Guide and Reference, Rev. 1d," AuSIM Inc., Mountain View, CA, October (2001).

3. Stephen A. Rizzi, Aric R. Aumann, Leonard V. Lopes, and Casey L. Burley, "Auralization of hybrid wing-body aircraft flyover noise from system noise predictions," *AIAA Journal of Aircraft*, **51**(6), 1914-1926, (2014).

4. Leonard V. Lopes and Casey L. Burley, "Design of the next generation aircraft noise prediction program: ANOPP2," *17th AIAA/CEAS Aeroacoustics Conference*, AIAA 2011-2854, Portland, Oregon, June 5-8, (2011).

5. Michael Arntzen, Stephen A. Rizzi, Hendrikus G. Visser, and Dick G. Simons, "A framework for simulation of aircraft flyover noise through a non-standard atmosphere," *AIAA Journal of Aircraft*, **51**(3), 956-966, (2014).

6. Michael Arntzen, Dick G. Simons, Jie Shen, Ana Lucia Varbanescu, and Henk Sips, "Acoustic ray tracing parallelization," *NoiseCon 2013*, NC-13-30, Denver, CO, (2013).

7. "NetCDF," http://www.unidata.ucar.edu/software/netcdf/, Unidata, Date Accessed April 30, (2013).

8. Stephen A. Rizzi and Brenda M. Sullivan, "Synthesis of virtual environments for aircraft community noise impact studies," *11th AIAA/CEAS Aeroacoustics Conference*, AIAA-2005-2983, Monterey, CA, May, (2005).

9. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design patterns: Elements of reusable object-oriented software*, Addison-Wesley, (1994).

10. Martin Reddy, *API design for C++*, Elsevier, Morgan Kaufmann Publishers, (2011).

11. "American national standard method for the calculation of the absorption of sound by the atmosphere," American Institute of Physics, ANSI S1.26-1995 (ASA 113-1995), (1995).

12. Stephen A. Rizzi and Andrew Christian, "A method for simulation of rotorcraft fly-in noise for human response studies," *InterNoise 2015*, San Francisco, CA, (2015).

13. M.E. Delany and E.N. Bazley, "Acoustical properties of fibrous absorbent materials," *Applied Acoustics*, **3**(2), 105-116, (1970).

14. Keith Attenborough, "Acoustic impedance models for outdoor ground surfaces," *Journal of Sound and Vibration*, **99**(4), 521-544, (1985).

15. Joseph E. Piercy, Tony F.W. Embleton, and Louis C. Sutherland, "Review of noise propagation in the atmosphere," *Journal of the Acoustical Society of America*, **61**(6), 1403-1418, (1977).

16. Juliet A. Page, Clif Wilmer, Troy Schultz, Kenneth J. Plotkin, and Joseph Czech, "Advanced Acoustic Model technical reference and user manual," Wyle Laboratories, Inc. (2009).