AuSIM Engineering Solutions

CRE_TRON API

3D Sound API for Audio Simulation

Create Interactive Three-dimensional AuSIM3D™ Sound Spaces

OVERVIEW

CRE_TRON is a 3D audio application-programming interface (API) that was developed by Crystal River Engineering in the early 1990's to facilitate the creation of interactive AuSIM3D sound spaces with localized sound.

The goal of the CRE_TRON API is to allow a user or developer to build up a sound space using the concepts of physical reality without having to know about the underlying algorithms, implementation or audio hardware.

This API implements the concept of a sound space in the form of easy-to-understand objects. Objects include sound emitting sources, sound transfer mediums, and sound receiving listeners. Sounds get created by sources, such as a ringing phone, propagate through space, and finally reach a listener's ears, where they are received and interpreted.

AUDIO LOCALIZATION

The localization processing of two independent sound sources is illustrated in Figure 1. This schematic can be superposed to represent any number of sources. Each source is processed through a sequence of dynamic models, representing independent characteristics of wave propagation. First, a "Source Model" filter accounts for emission amplitude and the directivity of the emitter. A human voice, for instance, sounds dimmer when speaking away from the listener than towards. Next, a "Propagation Model" filter accounts for attenuation due to spreading and coloration due to friction in the medium. Finally, the signal is split due to the differences in travel time to each ear, and a Head-Related Transfer Function (or HRTF), a pair of filters each representing the directivity of an ear, is applied for the directional effect. Finally, the Left and Right Outputs from the respective source pipelines are summed together and output as a binaural mix for headphone presentation.

TECHNOLOGY

The audio simulation technology, *AuSIM3D™*, from AuSIM, Inc. uses physical modeling and empirical data to synthesize a sound space in a completely natural and realistic way. When listening to a system incorporating such technology, a user not only feels immersed by real-world, three-dimensional sounds, but also can use natural filtering to discern and comprehend any of several layered concurrent sound streams.

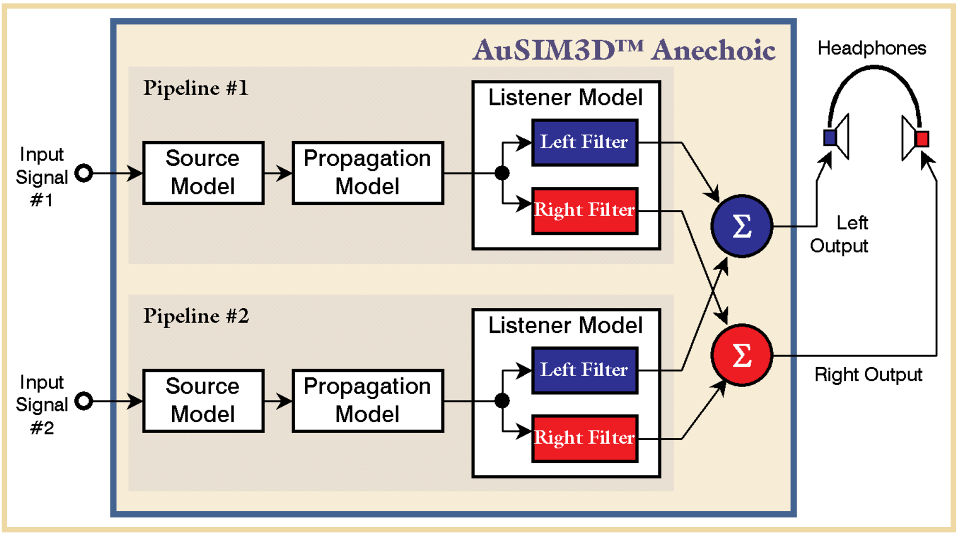
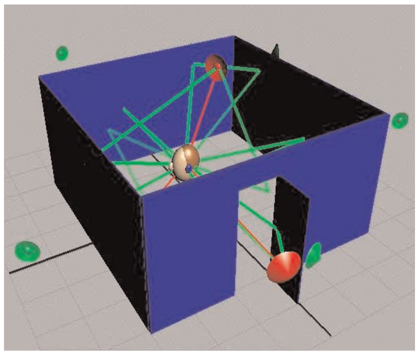


Figure 1
AuSIM3D™ Anechoic Processing Model

FEATURES	BENEFITS
■ **Elegant API**	■ Allows the widest range of simulation options of source, propagation, and listener characteristics
■ **High level programming**	■ Changes in simulation code require less time and resources
■ **Direct replacement**	■ Compatible with *CRE Acoustetron 2* which is supported by 3rd party simulation software
■ **Portable software**	■ Applications developed on one system can easily migrate to a non-identical system

For each audio source, the system produces a left and right output pair dependent on the direction of emission from the source, path of propagation, and direction of arrival to the listener. The output pairs corresponding to each source are mixed and played through conventional headphones or nearphones. The processing creates the perception that the source is positioned at any specified location in three-dimensional space.



SOFTWARE DISTRIBUTION

The *AuSIM3D™* server code comes pre-installed on the AuSIM system. Client-side code, including example programs with source, demo applications, and diagnostic tools, is supplied on a CD-ROM. Client-side code is ready-to-use for Win32 systems, and ready-to-recompile for a variety of Unix-based systems including Linux, Solaris, IRIX and others.

AuSIM Engineering Solutions

AuSIM, Incorporated
Mountain View, CA 94043
Voice: +1 (650) 526-3980
FAX: +1 (772) 325-0849
info@ausim3d.com

http://audiosimulation.com

SYSTEM FUNCTIONS

`cre_init(driver, hdid, sources, mode)`

Computes, detects, and allocates resources (i.e., processors and host memory) to provide the services specified by driver to listener head for the requested number of sources. The driver is a legacy CRE term. In the AuSIM implementation, all old anechoic CRE drivers are honored, with a competent emulation. New AuSIM drivers initialize a specific set of simulation parameters. All host objects are initialized with reasonable values. The listener's head is located at the origin. All sound sources are initially positioned at the full RESPONSE_DISTANCE directly in front of the listener.

- ▼ **driver**: Specifies the driver type. Used to determine sample rate, and to open live audio by default.
- ▼ **hdid:** Specifies which listener to initialize. Specify different listener numbers for multiple listener support.
- ▼ **sources:** Number of sources to be available
- ▼ **mode:** Units, resolution, shift bit, AHM selector, verbose, console mode

`cre_update_audio()`

Synchronizes frames and controls signal processing. This routine checks for any pending updates since the previous call, recomputes signal processing parameters with respect to all affected source-to-listener relationships, and passes the new values to the signal processing engine.

- ➽ Causes position calls to cre_locate_source and cre_locate_head to take effect.
- ➽ Also, the engine will be held until the first cre_update_audio call happens.

`cre_hold_audio(state)`

- ➽ If state is non-zero, then the engine stops, allowing for wavefile synchronization
- ➽ If the state is zero the engine resumes.

`cre_close(driver, head)`

De-allocates host resources for a given driver and listener (which may then be reallocated with another cre_init() call). cre_close() will gently shut off audio for the specified listener. When the last open listener is closed, all open wave files are also closed. Calling cre_close(ALL_DRIVERS, ALL_HEADS) will close all audio and wave files.

- ▼ **driver:** reset or close
- ▼ **head:** Which head to close or ALL_HEADS

`cre_end()`

This is an alias for cre_close(ALL_DRIVERS, ALL_HEADS) and is compatible with the ANSI C atexit() function

- ➽ Same as cre_close(Atron_CLOS, ALL_HEADS)

LISTENER FUNCTIONS

`cre_define_head(id, prm, pts, data)`

Allows the user to specify parameters defining the listener model (head size and pinnae characteristics) and the reference frame for location coordinates in subsequent calls to cre_locate_head()

- ▼ **id:** head id, as given to a previous cre_init call
- ▼ **prm:** ocular, pinnae, crown, offsets, interaural dist, HRTF model, HRTF file, near-field exaggeration, near distance scale, output gain in dB, path gain

PRM Types

AtrnAURALocular - AURAL OFFSET along X axis. Typically zero, or an offset from ocular axis (eye) coordinates. An offset from the ocular axis, which is in front of the aural axis, would be negative.

AtrnAURALpinnae - AURAL OFFSET along Y axis. One-half the positive distance between ear canal openings.

AtrnAURALcrown - AURAL OFFSET along Z axis. Typically, either the vertical separation of ocular and aural axes, or the vertical offset to the head tracking sensor, which is often placed on top of the head. An offset from the head crown, which is above the aural axis, would be negative.

AtrnAURALoffsets - The set of ordered AURAL OFFSETs. With a single call to cre_define_head(), this parameter can update one to all AURAL OFFSETs as an ordered array of floats pointed to by data, of pts items. The ordered sextuple is specified by the enumeration *ATRNspaceDef* (x,y,z,yaw,pitch,roll). Currently, only Cartesian translations (x,y,z) are supported. All of the single value AURAL OFFSET parameter rules stated above apply to AtrnAURALoffsets, including pts = 0 to reset the default values.

AtrnAHMname - Specify the name of an AHM subject to load. If it is not one of the pre-loaded subjects, the system will search the AHM files (located in the directory defined by the HRTF environment variable) for the specified AHM subject. An error is returned if the AHM subject is not found.

AtrnINTERAURAL - The AtrnAURALpinnae OFFSET doubled and time scaled. This parameter combines the spatial control of the AtrnAURALpinnae parameter above with its associated interaural delay scaling. AtrnINTERAURAL is the full ear to ear width (measured to ear canal opening) and thus is twice the AtrnAURALpinnae value. Using pts = 1 will set the pinnae offset to one-half the absolute float value pointed to by data. If pts = 0, data is ignored and the parameter is reset to the default value. In both cases, interaural delay values are not scaled to the ratio of the given interaural size with the default size. If pts = -1, the parameter is set according to data, but the delay scaling is not altered.

AtrnHRTFfile - Specify HRTF filename to be loaded. If pts = 0, data is ignored and the default HRTF map is reloaded. Otherwise the filename pointed to by data is loaded from the directory given by the HRTF environment variable. The pts value should be the string length of the filename for consistency.

AtrnDELAYtable - Redefine the interaural delay table. This parameter requires a properly formatted interaural delay table. This parameter is provided for psychoacoustic research, and is otherwise undocumented.

AtrnDELAYscale - Scale the current delay table. When pts = 1, this will scale all interaural delays by the float value pointed to by data. If pts = 0, data is ignored and the interaural delays are reset to their default values.

AtrnHRTFresolve - Sets the filter order trim.

AtrnHRTFmodel - Sets the model to use. This will be one of: NearField, FarField, or MixedField.

AtrnHEADgain - sets a single floating point dB level for the final conversion gain in preparing a listener's signal pair for output. The pts parameter is ignored.

AtrnDISPLAYtype - sets the output type for filtering per display device. The enum describing the particular device is given in the pts parameter. Choose one of the following: eqGenericHeadphone, eqGenericNearphone, eqSennheiserHD250 or eqSennheiserHD570. Any data is ignored.

AtrnEQleft - Downloads the display type for left EQ coefficients.

AtrnEQright - Downloads the display type for right EQ coefficients.

`cre_locate_head(id, hloc)`

Locates the head of a listener six dimensionally in world coordinates. It only updates changes from previous state, recalculating pinnae locations as needed. This function does not affect processing until a synchronization call to cre_update_audio() is successful.

- ▼ **id:** head id as passed to a previously called cre_init
- ▼ **hloc** = float[6] = { x, y, z, yaw, pitch, roll}

SOURCE FUNCTIONS

`cre_define_source(id, int prm, pts, data)`

Allows the user to specify parameters defining the source rendering model (directional radiation pattern, localization ON/OFF, and listener linkage). The function is a generic dispatcher that may be extended in future releases. See parameter descriptions below for specific behavior.

- ▼ **id:** source id, or ALL_SOURCES
- ▼ **prm:** channel input, specialization on/off, exponent factor, gain distance, time delay, radiation profile, link head to source, source model, model attributes, path gain

PRM Types

AtrnRADfields -Allows the user to control the directivity of the sound. data[] will contain two parameters (both in radians) describing a field of radiation and a field of intensity. These fields are cones centered on the source's boresight direction (principal direction of aural emission) in which ~90% (for the field of radiation) or ~45% (for the field of intensity) of the sound energy is dissipated.

AtrnRADprofile - Defines an audio source's radiation pattern about its boresight axis. The radiation profile of a sound source id is specified in data with an array of relative sound pressure levels in decibels, sampled equiangularly at pts points from the boresight direction to anti-boresight direction, inclusive. The boresight direction is coincident with the source's positive roll axis (the axis parallel to the world coordinate X-axis when source yaw and pitch are zero). All definable radiation patterns are symmetric about the roll axis (i.e., no rectangular horn speakers). For off-axis angles from source to listener that fall between sample points, the profile is linearly interpolated.

AtrnPROFILEpts - Specifies the number of profile points to use.

AtrnSPATIALoff - Disables localization for this source. The monaural sound is patched directly through and mixed with the left and right outputs with a gain (in dB) defined by data. This allows the user to implement simple panning. If pts = 0, the previous panning mixture values are used. If pts = 1, the given value is applied to a balanced mixture. If pts > 2, an error is returned. If pts < 0, localization is re-enabled, which is the default. The source is passed through a flat filter, so the latency remains the same as if it were localized.

AtrnSPATIALon - Enables source localization. This is the default.

AtrnHEADlink - By setting this parameter, the source maintains its relative position and orientation to the listener's head for all head positions and attitudes given. When AtrnHEADlink is enabled, all calls to cre_locate_source() establish a new relative position as a difference of the global coordinates of head and source locations given. Any data is ignored.

AtrnHEADunlink - Disables source to head linkage. This is the default.

AtrnSPRDrolloff - AtrnSPRDrolloff defines a single float value multiplier for that particular source of the global spreading-loss roll-off exponent defined in cre_define_medium(). The value must be positive and "reasonable". Reasonableness depends on the global rolloff value.

AtrnGAINdist - AtrnGAINdist defines the distance at which the gain is specified.

AtrnCHNLinput - AtrnCHNLinput allows zero, one, or more physical live audio input channels to be mapped to a particular source. The pts argument specifies how many and data is the array of integers specifying which channels to map.

AtrnCHNLmidi - AtrnCHNLmidi allows zero, one, or more MIDI audio input channels to be mapped to a particular source. pts specifies how many and data is the array of integers specifying which MIDI channels to map.

AtrnDPLRfactor - AtrnDPLRfactor sets the Doppler exaggeration. pts always specifies the listener id, which may be ALL_HEADS. If data is NULL, the Doppler factor is reset to its default of 1.0. Setting Doppler factor to 0.0 disables it completely.

`cre_locate_source(id, sloc)`

Locates an audio source id in (x, y, z, yaw, pitch, roll) world coordinates. This function does not affect processing until a synchronization call to cre_update_audio() is successful.

- ▼ **id:** source_id
- ▼ **sloc** = float[6] = { x, y, z, yaw, pitch, roll}

`cre_amplfy_source(id, dB)`

Sets the loudness of a source to 0 dB at 21/rollofexp units from the listener. This distance is ~1.96 inches from the head if units of inches (default) are being used. To alter this distance, scale the base 2 by the macro GAIN_RATIO. Distant sound sources may need to be set much higher (as much as +30 dB), in order to be audible at the listener's position.

- ▼ **id:** source id or ALL_SOURCES
- ▼ **dB:** input gain in dB

`cre_select_source(id, input)`

Selects from among the available hardware analog input channels for a given source id for all listeners. The implementation of this function is hardware specific and the command may not be desirable globally.

- ▼ **id:** source id or ALL_SOURCES
- ▼ **input:** none, or channel number

PROPIGATION MEDIUM FUNCTIONS

`cre_define_medium(volm, prm, pts, data)`

Allows the user to specify parameters to model the medium through which the sound propagates (absorption filter distance and spreading roll-off exponent).

- ▼ **volm:** medium id
- ▼ **prm:** roll off, model, attribute

PRM Types

AtrnLoadEnvModel - which model in DLL to install

AtrnSetEnvModelAttr - attribute in the model to set, and data for the attribute

AtrnROLLOFF - The roll-off exponent due to spreading power loss. The spreading roll-off exponent parameter sets the rate at which sound amplitude is attenuated over distance to yield cues in the third dimension. Currently, this parameter can only be set to apply to all sources. The pts argument must be set to one to have the float value pointed to by data set the spreading roll-off exponent, or may be zero or negative to reset the default value.

MISC. CLIENT FUNCTIONS

`cre_fetch_value(prm)`

- ▼ **prm:** number of sources and listeners initialized and number of units

PRM Types

AtrnLSTNRinit - number of listeners initialized

AtrnASRCinit - number of sources initialized

AtrnSYSunits - current units

`cre_query_head(id, prm, pts, data)`

- ▼ **id:** head id to query
- ▼ **prm:** AHM string

PRM Types

AtrnQRYHDahm - current AHM string

CLIENT TEST FUNCTIONS

`cre_test_atron(verbose)`

Tests the connection to the localization server via the ATRON protocol and gathers a text string from the server. If verbose is non-zero, it will print this string to stdout. Important: This function runs verbosely from a console window.

- ▼ **verbose:** 1 to print message to the screen

`cre_reset_atron(status)`

Resets the server

- ▼ **status:** ignored

`cre_client_nap(msecs)`

- ▼ **Sleep(msecs);**

WAVEFILE FUNCTIONS

`cre_open_wave(wavefile, mode)`

Opens a sound file referred to by the filename wavefile from the GoldServe's disk, returning a pointer to the allocated wavefile structure wavFt. Sound file control, such as playback through a particular source, is effected through cre_ctrl_wave(). Currently, the only formally recognized sound file format is RIFF (MS Windows .wav format). Note that, independent of which driver is being used, both 22.05 kHz and 44.1 kHz wavefiles can be opened and played back.

- ▼ **wavefile:** name of wavefile to load, either absolute or relative to the WAVEFORM environment variable
- ▼ **mode:** ignored
- ➽ Note: wavefiles opened BEFORE a cre_init will persist after a cre_close. Wavefiles opened AFTER a cre_init, will be cleaned up on a cre_close.

`cre_ctrl_wave(src_id, wave, cmd, data)`

Requests the host to control the waveform wave according to the command cmd, which may be related to source src. This function is a generic dispatcher that may be extended in future releases.

- ▼ **src_id:** id of the source to which a wavefile is linked
- ▼ **wave**: pointer to wave structure returned by a previous cre_open_wave
- ▼ **cmd:** wavefile position, rewind, looped, stop, clear flag, stats

`cre_close_wave(wave)`

Closes the wavefile, if open, and frees the signal and the wave structure. If wave is attached to a sound source and is playing, it will be stopped before the wavefile is closed. In order to properly de-allocate resources, each (successful) call to cre_open_wave() must be balanced with a call to this routine.

- ▼ **wave:** pointer to wave structure returned by a previous cre_open_wave

NOTE: Not all of the original **CRE_TRON** calls, as defined by CRE, are supported in the AuSIM version. Unsupported calls include:

```
cre_detect()
cre_direct_source()
cre_pmeter_source()
cre_set_rel_pos()
cre_get_polar()
cre_fetch_message()
cre_fetch_error()
cre_fetch_version()
cre_query_source()
cre_query_medium()
cre_query_router()
```

EXAMPLE CODE

```
#include <conio.h>
#include <stdio.h>
#pragma hdrstop

#include "cre_tron.h"
#include "cre_wave.h"

#define SourceID     0
#define HeadID       0
#define Sources      2
#define WaveFile  "TEST.WAV"
#define PanLimit   100.0f

void main(void)
{
  float step = -0.2f;
  float SrcLoc[6]  = { 10.0f, PanLimit, 0.0, 0.0, 0.0, 0.0 };
  float HeadLoc[6] = {  0.0, 0.0, -10.0f, 0.0, 0.0, 0.0 };
  wavFt *wave;

  /* initialize two Tron sources, with verbose report */
  if (cre_init(Atrn_ASM1, HeadID, Sources,
               _CONSOLE_|_VERBOSE_) < Ok)
     return;

  /* open WAV file and load wave form using all buffers */
  if (!(wave = cre_open_wave(WaveFile, 0))) {
     printf("\nwave load error.");
     return;
  }
  /* play open wave form as SourceID with repeat loop */
  cre_ctrl_wave (SourceID, wave, WaveCTRL_LOOP, NULL);
  /* locate listener once (not moving) */
  cre_locate_head (HeadID, HeadLoc);

  printf("\nPress Any Key to Exit ... ");
  /* enter simulation loop until key is pressed */
  while(!kbhit()) {
     SrcLoc[AtrnY] += step;    /* move source location */
     if ((SrcLoc[AtrnY]<-PanLimit) || (SrcLoc[AtrnY]>PanLimit))
        step = -step;          /* reverse panning direction */
     /* set new location as location of source 0 in space */
     cre_locate_source(SourceID, SrcLoc);
    cre_update_audio();       /* flush all changes */
  }
  /* stop wave form playback and detach from SourceID */
  cre_ctrl_wave(SourceID, wave, WaveCTRL_STOP, NULL);
  cre_close_wave(wave);       /* close waveform */

  cre_close(Atrn_CLOS, HeadID); /* close Tron */
  printf("\n");
}
```

EXAMPLE CODE

The code listed above is a "minimal" program which initializes the GoldServe™, loads and plays a wave file, turns on a single source to be used, positions a listener in space, moves a source between two points in space, and "displays" the sound space by updating the hardware.

The cre_init() call will locate and initialize sufficient hardware to localize two sources, locate the listener's head at the origin, and locate a sound source 50 inches directly in front of the head (by default). Since a zero was specified for the units argument, locations will be interpreted in inches, the default units. The HeadLoc variable therefore refers to a position 10 inches below the origin. A source is by default a uniform radiator.

After successful initialization, the call cre_amplfy_source() turns source #0 on. Note that all sources are initialized with no amplification. In order to hear anything from a source after initialization, cre_amplfy_source() must be used. The cre_open_wave() and cre_ctrl_wave() calls are used to load a waveform from disk and play it. The listener is located once to move from the default position to the one specified by HeadLoc.

Then the program moves the location of source #0 using cre_locate_source(), and uses cre_update_audio() to flush all changes to the hardware, in order to display the new sound space for the listener, until any character is typed, at which point the hardware is closed.

COORDINATE SYSTEM

The environment in which localized sounds can be experienced is described by a three-dimensional coordinate system. Within this coordinate system, six-dimensional vectors are used to specify the position and orientation of the listener's head and of all sound sources. The inputs to the GoldServe™ (external or wave forms) are mapped to the corresponding locations in the coordinate system relative to the listener's location.

The GoldServe™ software library represents a six-dimensional location vector as an array of six 32-bit floating-point numbers. In this array, the first three elements specify the x, y, and z position in space, in number of "units" (units are selected at initialization of the GoldServe™-see cre_init()). The second three vector elements specify the yaw, pitch, and roll, in radians. They define the orientation of the source or head at position (x,y,z).

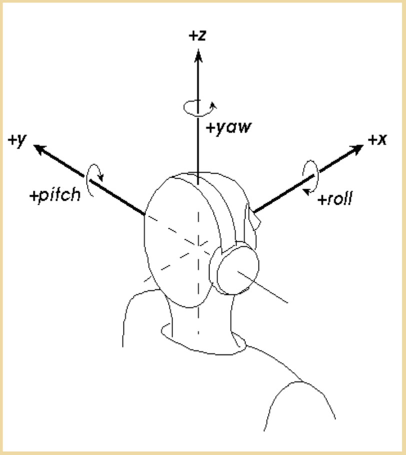


Figure 2
Six-dimensional coordinate system with listener located at (0, 0, 0, 0, 0, 0)

The coordinate system is adopted from the vehicle dynamic simulation world. As illustrated in Figure 2, the system is right-handed, with the positive x-axis straight ahead and the positive z-axis ascending vertically. Orientations are specified as right-handed radian Euler rotations, roll, pitch, and yaw, about respective x, y, and z axes.

The six-element vector employed in the GoldServe™ software (in using cre_locate_head() and cre_locate_source()) is ordered < x, y, z, yaw, pitch, roll>. The order of rotations depends upon the rotation basis. With respect to the global coordinate system, from a local coordinate system that initially coincides with the global one, an object is rolled, pitched, yawed, and finally translated.

AuSIM Engineering Solutions

CRE_TRON API

AuSIM, Incorporated
Mountain View, CA 94043   info@ausim3d.com
Voice: +1 (650) 526-3980   FAX: +1 (772) 325-0849

http://audiosimulation.com

AuSIM Engineering Solutions