Sound Design Toolkit

078

Generated by Doxygen 1.8.10

Tue May 22 2018 12:31:19

Contents

Chapter 1

Main Page

The 'Sound Design Toolkit' (SDT) is a framework a for education and research in Sonic Interaction Design. It includes a collection of physically informed models, post-processing algorithms and sound analysis routines for interactive sound synthesis. It can be considered as a virtual Foley box of sound synthesis algorithms, each representing a specific sound-producing event.

Developed with the contribution of the following EU-projects: 2001-2003 'SOb' http://www.soundobject.org/ 2006-2009 'CLOSED' http://closed.ircam.fr/ 2008-2011 'NIW' http://www.niwproject.eu/ 2014-2017 'SkAT-VG' http://www.skatvg.eu/

Contacts: stefano.papetti@zhdk.ch stefano.dellemonache@gmail.com stefanobaldan@iuav.it

Complete list of authors (either programmers or designers): Federico Avanzini (avanzini@dei.unipd.it) Nicola Bernardini (nicb@sme-ccppd.org) Gianpaolo Borin (gianpaolo.borin@tin.it) Carlo Drioli (carlo.drioli@univr.it) Stefano Delle Monache (stefano.dellemonache@gmail.com) Delphine Devallez Federico Fontana (federico.fontana@uniud.it) Laura Ottaviani Stefano Papetti (stefano.papetti@zhdk.ch) Pietro Polotti (pietro.polotti@univr.it) Matthias Rath Davide Rocchesso (roc@iuav.it) Stefania Serafin (sts@media.aau.dk) Stefano Baldan (stefanobaldan@iuav.it)

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

Module Documentation

4.1 SDTAnalysis.h: Sound analysis tools

Modules

- Zero crossing rate
- Myoelastic features extractor
- Spectral audio descriptors
- Fundamental frequency estimator

4.1.1 Detailed Description

Tools for the extraction of low level audio descriptors, specifically tailored for the analysis of vocal imitations and the vocal control of SDT models in the SkAT-VG project.

4.2 Zero crossing rate

Typedefs

- typedef struct SDTZeroCrossing SDTZeroCrossing

  Opaque data structure for a zero crossing rate detector object.
- typedef struct SDTZeroCrossing SDTZeroCrossing

  Opaque data structure for a zero crossing rate detector object.

Functions

- SDTZeroCrossing * SDTZeroCrossing_new (unsigned int size)

  Instantiates a zero crossing rate detector.
- void SDTZeroCrossing_free (SDTZeroCrossing *x)

  Destroys a zero crossing rate detector.
- void SDTZeroCrossing_setOverlap (SDTZeroCrossing *x, double f)

  Sets the analysis window overlapping ratio. Accepted values go from 0.0 to 1.0, with 0.0 meaning no overlap and 1.0 meaning total overlap.
- int SDTZeroCrossing_dsp (SDTZeroCrossing *x, double *out, double in)

  Signal processing routine. Call this function at sample rate to perform signal analysis.

4.2.1 Detailed Description

Zero crossing rate signal analyzer.

4.2.2 Function Documentation

4.2.2.1 int SDTZeroCrossing_dsp (SDTZeroCrossing * *x*, double * *out*, double *in*)

Signal processing routine. Call this function at sample rate to perform signal analysis.

Parameters

`in`	*x*	Pointer to the instance
`out`	*out*	Pointer to a double containing the algorithm output
`in`	*in*	Input sample

Returns

1 if output available (analysis window full), 0 otherwise

4.2.2.2 void SDTZeroCrossing_free (SDTZeroCrossing * *x*)

Destroys a zero crossing rate detector.

Parameters

`in`	*x*	Pointer to the instance to destroy

4.2.2.3 SDTZeroCrossing * SDTZeroCrossing_new (unsigned int *size*)

Instantiates a zero crossing rate detector.

Parameters

in	*size*	Size of the analysis window, in samples

Returns

Pointer to the new instance

4.2.2.4 void SDTZeroCrossing_setOverlap (SDTZeroCrossing * *x*, double *f*)

Sets the analysis window overlapping ratio. Accepted values go from 0.0 to 1.0, with 0.0 meaning no overlap and 1.0 meaning total overlap.

Parameters

in	*x*	Pointer to the instance
in	*f*	Overlap ratio [0.0, 1.0]

4.3 Myoelastic features extractor

Typedefs

- typedef struct SDTMyoelastic SDTMyoelastic

  Opaque data structure for a myoelastic feature extractor object.
- typedef struct SDTMyoelastic SDTMyoelastic

  Opaque data structure for a myoelastic feature extractor object.

Functions

- SDTMyoelastic * SDTMyoelastic_new (int size)

  Instantiates a myoelastic feature extractor.
- void SDTMyoelastic_free (SDTMyoelastic *x)

  Destroys a myoelastic feature extractor.
- void SDTMyoelastic_setDcFrequency (SDTMyoelastic *x, double f)

  Sets the DC offset cutoff.
- void SDTMyoelastic_setLowFrequency (SDTMyoelastic *x, double f)

  Sets the low frequency cutoff.
- void SDTMyoelastic_setHighFrequency (SDTMyoelastic *x, double f)

  Sets the high frequency cutoff.
- void SDTMyoelastic_setThreshold (SDTMyoelastic *x, double f)

  Sets the amplitude threshold of the input gate. Myoelastic activity is not computed for signals whose amplitude is below this thresold.
- int SDTMyoelastic_dsp (SDTMyoelastic *x, double *outs, double in)

  Signal processing routine. Call this function at sample rate to perform signal analysis.

4.3.1 Detailed Description

Extracts amount and frequency of slow amplitude variations in the signal. Specifically designed for the detection of myoelastic activity in vocal input.

4.3.2 Function Documentation

4.3.2.1 int SDTMyoelastic_dsp (SDTMyoelastic * *x*, double * *outs*, double *in*)

Signal processing routine. Call this function at sample rate to perform signal analysis.

Parameters

in	*x*	Pointer to the instance
out	*outs*	Pointer to an array of four doubles containing the algorithm output (slow myoelastic amount and frequency, fast myoelastic amount and frequency)
in	*in*	Input sample

Returns

1 if output available, 0 otherwise

4.3.2.2 void SDTMyoelastic_free (SDTMyoelastic * *x*)

Destroys a myoelastic feature extractor.

Parameters

in	*x*	Pointer to the instance to destroy

4.3.2.3 SDTMyoelastic * SDTMyoelastic_new (int *size*)

Instantiates a myoelastic feature extractor.

Returns

Pointer to the new instance

4.3.2.4 void SDTMyoelastic_setDcFrequency (SDTMyoelastic * *x*, double *f*)

Sets the DC offset cutoff.

Parameters

in	*x*	Pointer to the instance
in	*f*	DC offset cutoff, in Hz

4.3.2.5 void SDTMyoelastic_setHighFrequency (SDTMyoelastic * *x*, double *f*)

Sets the high frequency cutoff.

Parameters

in	*x*	Pointer to the instance
in	*f*	High frequency cutoff, in Hz

4.3.2.6 void SDTMyoelastic_setLowFrequency (SDTMyoelastic * *x*, double *f*)

Sets the low frequency cutoff.

Parameters

in	*x*	Pointer to the instance
in	*f*	Low frequency cutoff, in Hz

4.3.2.7 void SDTMyoelastic_setThreshold (SDTMyoelastic * *x*, double *f*)

Sets the amplitude threshold of the input gate. Myoelastic activity is not computed for signals whose amplitude is below this thresold.

Parameters

in	*x*	Pointer to the instance
in	*f*	Amplitude threshold

4.4 Spectral audio descriptors

Typedefs

- typedef struct SDTSpectralFeats SDTSpectralFeats

  Opaque data structure for a spectral features extractor.
- typedef struct SDTSpectralFeats SDTSpectralFeats

  Opaque data structure for a spectral features extractor.

Functions

- SDTSpectralFeats ∗ SDTSpectralFeats_new (unsigned int size)

  Instantiates a spectral features extractor.
- void SDTSpectralFeats_free (SDTSpectralFeats ∗x)

  Destroys a spectral features extractor.
- void SDTSpectralFeats_setOverlap (SDTSpectralFeats ∗x, double f)

  Sets the analysis window overlapping ratio. Accepted values go from 0.0 to 1.0, with 0.0 meaning no overlap and 1.0 meaning total overlap.
- void SDTSpectralFeats_setMinFreq (SDTSpectralFeats ∗x, double f)

  Sets the lower frequency bound for spectral analysis. Spectral bins below this frequency are ignored in the audio descriptors computation.
- void SDTSpectralFeats_setMaxFreq (SDTSpectralFeats ∗x, double f)

  Sets the upper frequency bound for spectral analysis. Spectral bins above this frequency are ignored in the audio descriptors computation.
- int SDTSpectralFeats_dsp (SDTSpectralFeats ∗x, double ∗outs, double in)

  Signal processing routine. Call this function for each sample to perform signal analysis.

4.4.1 Detailed Description

Spectral features extractor: statistical moments (centroid, spread, skewness, kurtosis), spectral flatness, spectral flux and an onset detection function based on rectified, whitened spectral flux.

4.4.2 Function Documentation

4.4.2.1 int SDTSpectralFeats_dsp (SDTSpectralFeats ∗ *x*, double ∗ *outs*, double *in*)

Signal processing routine. Call this function for each sample to perform signal analysis.

Parameters

in	*x*	Pointer to the instance
out	*outs*	Pointer to an array of seven doubles, containing the algorithm outputs. Array members represent the following information respectively: 1. Spectral centroid, 2. Spectral spread, 3. Spectral skewness, 4. Spectral kurtosis, 5. Spectral flatness, 6. Spectral flux, 7. Onset detection function (rectified and whitened spectral flux).
in	*in*	Input sample

Returns

1 if output available (analysis window full), 0 otherwise

4.4.2.2 void SDTSpectralFeats_free (SDTSpectralFeats * *x*)

Destroys a spectral features extractor.

Parameters

in	*x*	Pointer to the instance to destroy

4.4.2.3 SDTSpectralFeats * SDTSpectralFeats_new (unsigned int *size*)

Instantiates a spectral features extractor.

Parameters

in	*size*	Size of the analysis window, in samples

Returns

Pointer to the new instance

4.4.2.4 void SDTSpectralFeats_setMaxFreq (SDTSpectralFeats * *x*, double *f*)

Sets the upper frequency bound for spectral analysis. Spectral bins above this frequency are ignored in the audio descriptors computation.

Parameters

in	*x*	Pointer to the instance
in	*f*	Maximum analyzed frequency, in Hz

4.4.2.5 void SDTSpectralFeats_setMinFreq (SDTSpectralFeats * x, double f)

Sets the lower frequency bound for spectral analysis. Spectral bins below this frequency are ignored in the audio descriptors computation.

Parameters

in	*x*	Pointer to the instance
in	*f*	Minimum analyzed frequency, in Hz

4.4.2.6 void SDTSpectralFeats_setOverlap (SDTSpectralFeats ∗ *x*, double *f*)

Sets the analysis window overlapping ratio. Accepted values go from 0.0 to 1.0, with 0.0 meaning no overlap and 1.0 meaning total overlap.

Parameters

in	*x*	Pointer to the instance
in	*f*	Overlap ratio [0.0, 1.0]

4.5 Fundamental frequency estimator

Typedefs

- typedef struct SDTPitch SDTPitch

  Opaque data structure for a fundamental frequency estimator.
- typedef struct SDTPitch SDTPitch

  Opaque data structure for a fundamental frequency estimator.

Functions

- SDTPitch ∗ SDTPitch_new (unsigned int size)

  Instantiates a fundamental frequency estimator object.
- void SDTPitch_free (SDTPitch ∗x)

  Destroys a fundamental frequency estimator instance.
- void SDTPitch_setOverlap (SDTPitch ∗x, double f)

  Sets the analysis window overlapping ratio. Accepted values go from 0.0 to 1.0, with 0.0 meaning no overlap and 1.0 meaning total overlap.
- void SDTPitch_setTolerance (SDTPitch ∗x, double f)

  Sets the peak detection tolerance. Always choosing the greatest NSDF peak as pitch estimation sometimes leads to wrong octave detection errors. To overcome this problem, some tolerance is introduced in the peak detection algorithm. The chosen NSDF peak is the one with lowest frequency among those with value close enough to the global maximum. A value of 0.0 always selects the global maximum, while a value of 1.0 always selects the last NSDF peak.
- int SDTPitch_dsp (SDTPitch ∗x, double ∗outs, double in)

  Signal processing routine. Call this function for each sample to perform signal analysis.

4.5.1 Detailed Description

The pitch detection algorithm implemented in this object is discussed in the paper "A smarter way to find pitch" by Philip McLeod and Geoff Wyvill (2005) and it is based on the NSDF (Normalized Squared Differences Function), a close relative of the autocorrelation function.

4.5.2 Function Documentation

4.5.2.1 int SDTPitch_dsp (**SDTPitch** ∗ *x*, double ∗ *outs*, double *in*)

Signal processing routine. Call this function for each sample to perform signal analysis.

Parameters

`in`	*x*	Pointer to the instance
`out`	*outs*	Pointer to an array of two doubles, containing the algorithm outputs. Array members represent the following information respectively: 1. Estimated pitch (Hz), 2. Pitch clarity [0.0, 1.0].

in	*in*	Input sample

Returns

1 if output available (analysis window full), 0 otherwise

4.5.2.2 void SDTPitch_free (SDTPitch * *x*)

Destroys a fundamental frequency estimator instance.

Parameters

in	*x*	Pointer to the instance to destroy

4.5.2.3 SDTPitch * SDTPitch_new (unsigned int *size*)

Instantiates a fundamental frequency estimator object.

Parameters

in	*size*	Size of the analysis window, in samples

Returns

Pointer to the new instance

4.5.2.4 void SDTPitch_setOverlap (SDTPitch * *x*, double *f*)

Sets the analysis window overlapping ratio. Accepted values go from 0.0 to 1.0, with 0.0 meaning no overlap and 1.0 meaning total overlap.

Parameters

in	*x*	Pointer to the instance
in	*f*	Overlap ratio [0.0, 1.0]

4.5.2.5 void SDTPitch_setTolerance (SDTPitch * *x*, double *f*)

Sets the peak detection tolerance. Always choosing the greatest NSDF peak as pitch estimation sometimes leads to wrong octave detection errors. To overcome this problem, some tolerance is introduced in the peak detection algorithm. The chosen NSDF peak is the one with lowest frequency among those with value close enough to the global maximum. A value of 0.0 always selects the global maximum, while a value of 1.0 always selects the last NSDF peak.

Parameters

in	*x*	Pointer to the instance
in	*f*	Pitch estimation tolerance [0.0, 1.0]

4.6 SDTCommon.h: Common variables and functions

Macros

- #define SDT_ver 078

  SDT version number.
- #define SDT_ver_str "078"

  SDT version string.
- #define SDT_PI 3.141592653589793

  Value of Pi.
- #define SDT_TWOPI 6.283185307179586

  Value of 2 ∗ Pi.
- #define SDT_EULER 2.718281828459045

  Euler number.
- #define SDT_SQRT2 1.4142135623730951

  Square root of 2.
- #define SDT_MACH1 340.29

  Mach 1, speed of sound in air under normal atmospheric conditions (m/s)
- #define SDT_EARTH 9.81

  Earth gravity (N/Kg)
- #define SDT_MICRO 0.000001

  One millionth, small value often used instead of 0 to avoid division errors.
- #define SDT_QUIET 0.00003

  Gain factor roughly corresponding to a -90dB attenuation.

- #define SDT_ver 078

  SDT version number.
- #define SDT_ver_str "078"

  SDT version string.
- #define SDT_PI 3.141592653589793

  Value of Pi.
- #define SDT_TWOPI 6.283185307179586

  Value of 2 ∗ Pi.
- #define SDT_EULER 2.718281828459045

  Euler number.
- #define SDT_SQRT2 1.4142135623730951

  Square root of 2.
- #define SDT_MACH1 340.29

  Mach 1, speed of sound in air under normal atmospheric conditions (m/s)
- #define SDT_EARTH 9.81

  Earth gravity (N/Kg)
- #define SDT_MICRO 0.000001

  One millionth, small value often used instead of 0 to avoid division errors.
- #define SDT_QUIET 0.00003

  Gain factor roughly corresponding to a -90dB attenuation.

Functions

- void SDT_setSampleRate (double sampleRate)

  Sets the sample rate.
- void SDT_blackman (double ∗sig, int n)

  Applies a Blackman window to a chunk of samples. Applies a Blackman window to a chunk of samples.
- unsigned int SDT_bitReverse (unsigned int u, unsigned int bits)

  Reverses the bit order of an unsigned integer of given bit length.
- long SDT_clip (long x, long min, long max)

  Clips an integer value. Limits the range of an integer value between a given lower bound and upper bound.
- double SDT_expRand (double lambda)

  Exponential random number generator. Generates random numbers, following an exponential distribution.
- double SDT_fclip (double x, double min, double max)

  Clips a floating point value. Limits the range of a floating point value between a given lower bound and upper bound.
- double SDT_frand ()

  Uniform random number generator. Generates random numbers, following a uniform distribution.
- void SDT_gaussian1D (double ∗x, double sigma, int n)

  One-dimensional Gaussian kernel. One-dimensional Gaussian kernel. The Gaussian function is computed in the [-1,1] interval with 0 mean and the given standard deviation. The output is normalized so that the sum of all samples is equal to 1.
- double SDT_gravity (double mass)

  Computes earth gravity force. Computes the earth gravity force acting on an object of a given mass.
- void SDT_hanning (double ∗sig, int n)

  Applies a Hanning window to a chunk of samples. Applies a Hanning window to a chunk of samples.
- void SDT_haar (double ∗sig, long n)

  Computes a direct Haar Wavelet Transform of the incoming signal (in place).
- void SDT_ihaar (double ∗sig, long n)

  Computes an inverse Haar Wavelet Transform of the incoming signal (in place).
- double SDT_kinetic (double mass, double velocity)

  Computes kinetic energy. Computes the kinetic energy of an object, given its mass and velocity.
- unsigned int SDT_nextPow2 (unsigned int u)

  Returns the smallest power of 2 greater or equal than u.
- double SDT_normalize (double x, double min, double max)

  Rescales a value of known range into the [0.0, 1.0] interval. Rescales a value of known range into the [0.0, 1.0] interval.
- void SDT_normalizeWindow (double ∗sig, int n)

  Normalizes samples in a window so that their sum is equal to 1.
- void SDT_ones (double ∗sig, int n)

  Fills a buffer with ones. Fills a buffer with ones.
- double SDT_rank (double ∗x, int n, int k)

  Finds the kth smallest value in the input array. Finds the kth smallest value in the input array.
- void SDT_removeDC (double ∗sig, int n)

  Removes the global average from samples in a window.
- int SDT_roi (double ∗sig, int ∗peaks, int ∗bounds, int d, int n)

  Finds regions of influence (local maxima and minima) in a buffer. Finds regions of influence (local maxima and minima) in a buffer.
- double SDT_samplesInAir (double length)

  Time needed to travel the given distance at Mach 1. Computes the amount of time, in samples, needed by a sound wave propagating in air to travel a given distance. Particularly useful to set the delay times of comb filters and/or digital waveguides representing hollow cavities.
- double SDT_scale (double x, double srcMin, double srcMax, double dstMin, double dstMax, double gamma)

  Rescales a value from a source range to a target range. Rescales a value from a source range to a target range.

- int SDT_signum (double x)

  Computes the signum function. Computes the signum function.
- void SDT_sinc (double ∗sig, double w, int n)

  Applies a sinc window (sin(wt)/(wt)) to a chunk of samples. Applies a sinc window (sin(wt)/(wt)) to a chunk of samples.
- double SDT_truePeakPos (double ∗sig, int peak)

  Performs quadratic interpolation to estimate the true position of a peak. Performs quadratic interpolation to estimate the true position of a peak.
- double SDT_truePeakValue (double ∗sig, int peak)

  Performs quadratic interpolation to estimate the true amplitude value of a peak. Performs quadratic interpolation to estimate the true amplitude value of a peak.
- double SDT_wrap (double x)

  Wraps a phase in the range -pi/pi. Wraps a phase in the range -pi/pi.
- void SDT_zeros (double ∗sig, int n)

  Fills a buffer with zeros. Fills a buffer with zeros.

Variables

- double SDT_sampleRate

  Sampling frequency (Hz)
- double SDT_timeStep

  Sampling period (s)
- double SDT_sampleRate

  Sampling frequency (Hz)
- double SDT_timeStep

  Sampling period (s)

4.6.1 Detailed Description

Macros, variables and functions commonly used by all the SDT objects. SDTCommon.h should always be included when using other SDT modules.

4.6.2 Function Documentation

4.6.2.1 unsigned int SDT_bitReverse (unsigned int *u*, unsigned int *bits*)

Reverses the bit order of an unsigned integer of given bit length.

Parameters

in	*u*	Input value
in	*bits*	Number of bits to reverse

Returns

Unsigned integer with reversed bits

4.6.2.2 void SDT_blackman (double ∗ *sig*, int *n*)

Applies a Blackman window to a chunk of samples. Applies a Blackman window to a chunk of samples.

Parameters

in,out	*sig*	samples to window
in	*n*	window size

4.6.2.3 long SDT_clip (long *x*, long *min*, long *max*)

Clips an integer value. Limits the range of an integer value between a given lower bound and upper bound.

Parameters

in	*x*	Integer value to clip
in	*min*	Lower limit
in	*max*	Upper limit

Returns

Clipped integer value

4.6.2.4 double SDT_expRand (double *lambda*)

Exponential random number generator. Generates random numbers, following an exponential distribution.

Parameters

in	*lambda*	Rate of the exponential distribution.

Returns

Randomly generated value [0.0, +inf]

4.6.2.5 double SDT_fclip (double *x*, double *min*, double *max*)

Clips a floating point value. Limits the range of a floating point value between a given lower bound and upper bound.

Parameters

in	*x*	Floating point value to clip
in	*min*	Lower limit
in	*max*	Upper limit

Returns

Clipped floating point value

4.6.2.6 double SDT_frand ( )

Uniform random number generator. Generates random numbers, following a uniform distribution.

Returns

Randomly generated value [0.0, 1.0]

4.6.2.7 void SDT_gaussian1D (double * *x*, double *sigma*, int *n*)

One-dimensional Gaussian kernel. One-dimensional Gaussian kernel. The Gaussian function is computed in the [-1,1] interval with 0 mean and the given standard deviation. The output is normalized so that the sum of all samples is equal to 1.

Parameters

out	*x*	pointer to the kernel samples
in	*sigma*	standard deviation of the Gaussian function
in	*n*	kernel size

4.6.2.8 double SDT_gravity (double *mass*)

Computes earth gravity force. Computes the earth gravity force acting on an object of a given mass.

Parameters

in	*mass*	Mass of the object (Kg)

Returns

Earth gravity force (N)

4.6.2.9 void SDT_haar (double $*$ *sig,* long *n*)

Computes a direct Haar Wavelet Transform of the incoming signal (in place).

Parameters

in,out	*sig*	incoming signals
in	*n*	window size

4.6.2.10 void SDT_hanning (double $*$ *sig,* int *n*)

Applies a Hanning window to a chunk of samples. Applies a Hanning window to a chunk of samples.

Parameters

in,out	*sig*	samples to window
in	*n*	window size

4.6.2.11 void SDT_ihaar (double $*$ *sig,* long *n*)

Computes an inverse Haar Wavelet Transform of the incoming signal (in place).

Parameters

in,out	*sig*	incoming signals
in	*n*	window size

4.6.2.12 double SDT_kinetic (double *mass,* double *velocity*)

Computes kinetic energy. Computes the kinetic energy of an object, given its mass and velocity.

Parameters

in	*mass*	Mass of the object (Kg)
in	*velocity*	Velocity of the object (m/s)

Returns

Kinetic energy (J)

4.6.2.13 unsigned int SDT_nextPow2 (unsigned int *u*)

Returns the smallest power of 2 greater or equal than u.

Parameters

in	*u*	Input value

Returns

Smallest power of 2 greater or equal than u

4.6.2.14 double SDT_normalize (double *x*, double *min*, double *max*)

Rescales a value of known range into the [0.0, 1.0] interval. Rescales a value of known range into the [0.0, 1.0] interval.

Parameters

in	*x*	Value to normalize
in	*min*	Lower bound
in	*max*	Upper bound

Returns

Value rescaled from [min, max] to [0.0, 1.0]

4.6.2.15 void SDT_normalizeWindow (double ∗ *sig*, int *n*)

Normalizes samples in a window so that their sum is equal to 1.

Parameters

in,out	*sig*	window to normalize
in	*n*	window size

4.6.2.16 void SDT_ones (double ∗ *sig*, int *n*)

Fills a buffer with ones. Fills a buffer with ones.

Parameters

in,out	*sig*	pointer to the buffer
in	*n*	buffer size

4.6.2.17 double SDT_rank (double ∗ *x*, int *n*, int *k*)

Finds the kth smallest value in the input array. Finds the kth smallest value in the input array.

Parameters

in	*x*	input array
in	*n*	array size
in	*k*	item rank

Returns

kth smallest value in the array

4.6.2.18 void SDT_removeDC (double * *sig*, int *n*)

Removes the global average from samples in a window.

Parameters

in,out	*sig*	window to remove the average from
in	*n*	window size

4.6.2.19 int SDT_roi (double * *sig*, int * *peaks*, int * *bounds*, int *d*, int *n*)

Finds regions of influence (local maxima and minima) in a buffer. Finds regions of influence (local maxima and minima) in a buffer.

Parameters

in	*sig*	pointer to the buffer
out	*peaks*	indexes of the local maxima in the buffer
out	*bounds*	indexes of the local minima in the buffer

4.6.2.20 double SDT_samplesInAir (double *length*)

Time needed to travel the given distance at Mach 1. Computes the amount of time, in samples, needed by a sound wave propagating in air to travel a given distance. Particularly useful to set the delay times of comb filters and/or digital waveguides representing hollow cavities.

Parameters

in	*length*	Distance (m)

Returns

Amount of samples to travel the distance at Mach 1

4.6.2.21 double SDT_scale (double *x*, double *srcMin*, double *srcMax*, double *dstMin*, double *dstMax*, double *gamma*)

Rescales a value from a source range to a target range. Rescales a value from a source range to a target range.

Parameters

in	*x*	Value to rescale

in	*srcMin*	Lower bound of source value
in	*srcMax*	Upper bound of source value
in	*dstMin*	Lower bound of rescaled value
in	*dstMax*	Upper bound of rescaled value
in	*gamma*	Gamma factor

Returns

Value rescaled from [srcMin, srcMax] to [dstMin, dstMax] with gamma factor `gamma`

4.6.2.22 void SDT_setSampleRate (double *sampleRate*)

Sets the sample rate.

Parameters

in	*sampleRate*	Sample rate (Hz).

4.6.2.23 int SDT_signum (double *x*)

Computes the signum function. Computes the signum function.

Parameters

in	*x*	Input value

Returns

Signum of x

4.6.2.24 void SDT_sinc (double ∗ *sig,* double *w,* int *n*)

Applies a sinc window (sin(wt)/(wt)) to a chunk of samples. Applies a sinc window (sin(wt)/(wt)) to a chunk of samples.

Parameters

in,out	*sig*	samples to window
in	*w*	sinc parameter
in	*n*	window size

4.6.2.25 double SDT_truePeakPos (double ∗ *sig,* int *peak*)

Performs quadratic interpolation to estimate the true position of a peak. Performs quadratic interpolation to estimate the true position of a peak.

Parameters

in	*sig*	signal buffer
in	*peak*	index of a local maximum

Returns

true peak position

4.6.2.26 double SDT_truePeakValue (double * *sig,* int *peak*)

Performs quadratic interpolation to estimate the true amplitude value of a peak. Performs quadratic interpolation to estimate the true amplitude value of a peak.

Parameters

in	*sig*	signal buffer
in	*peak*	index of a local maximum

Returns

true peak value

4.6.2.27 double SDT_wrap (double *x*)

Wraps a phase in the range -pi/pi. Wraps a phase in the range -pi/pi.

Parameters

4.6.2.28 void SDT_zeros (double * *sig*, int *n*)

Fills a buffer with zeros. Fills a buffer with zeros.

Parameters

in,out	*sig*	pointer to the buffer
in	*n*	buffer size

4.7 SDTComplex.h: Handling complex numbers

Data Structures

- struct SDTComplex

  Data structure containing the real and imaginary part of a complex number.

Typedefs

- typedef struct SDTComplex SDTComplex

  Data structure containing the real and imaginary part of a complex number.
- typedef struct SDTComplex SDTComplex

  Data structure containing the real and imaginary part of a complex number.

Functions

- SDTComplex SDTComplex_car (double real, double imag)

  Returns a complex number with the given real and imaginary parts.
- SDTComplex SDTComplex_exp (double phase)

  Returns a complex exponential with base e and given phase.
- double SDTComplex_abs (SDTComplex a)

  Returns the absolute value (magnitude) of a complex number.
- double SDTComplex_angle (SDTComplex a)

  Returns the angle (phase) of a complex number.
- SDTComplex SDTComplex_conj (SDTComplex a)

  Returns the complex conjugate of a complex number.
- SDTComplex SDTComplex_add (SDTComplex a, SDTComplex b)

  Returns the sum of two complex numbers.
- SDTComplex SDTComplex_sub (SDTComplex a, SDTComplex b)

  Returns the difference of two complex numbers.
- SDTComplex SDTComplex_mult (SDTComplex a, SDTComplex b)

  Returns the multiplication between two complex numbers.
- SDTComplex SDTComplex_div (SDTComplex a, SDTComplex b)

  Returns the division between two complex numbers.
- SDTComplex SDTComplex_addReal (SDTComplex a, double b)

  Returns the sum of a complex number and a real number.
- SDTComplex SDTComplex_subReal (SDTComplex a, double b)

  Returns the difference of a complex number and a real number.
- SDTComplex SDTComplex_realSub (double a, SDTComplex b)

  Returns the difference of a real number and a complex number.
- SDTComplex SDTComplex_multReal (SDTComplex a, double b)

  Returns the multiplication between a complex number and a real number.
- SDTComplex SDTComplex_divReal (SDTComplex a, double b)

  Returns the division between a complex number and a real number.
- SDTComplex SDTComplex_realDiv (double a, SDTComplex b)

  Returns the division between a real number and a complex number.

4.7.1 Detailed Description

This module contains data structures and functions to perform basic operations with complex numbers.

4.7.2 Function Documentation

4.7.2.1 double SDTComplex_abs (SDTComplex *a*)

Returns the absolute value (magnitude) of a complex number.

Parameters

in	*a*	Input value

Returns

Absolute value of input

4.7.2.2 SDTComplex SDTComplex_add (SDTComplex *a*, SDTComplex *b*)

Returns the sum of two complex numbers.

Parameters

in	*a*	First operand
in	*b*	Second operand

Returns

a plus b

4.7.2.3 SDTComplex SDTComplex_addReal (SDTComplex *a*, double *b*)

Returns the sum of a complex number and a real number.

Parameters

in	*a*	Complex operand
in	*b*	Real operand

Returns

a plus b

4.7.2.4 double SDTComplex_angle (SDTComplex *a*)

Returns the angle (phase) of a complex number.

Parameters

in	*a*	Input value

Returns

Angle of input

4.7.2.5 SDTComplex SDTComplex_car (double *real*, double *imag*)

Returns a complex number with the given real and imaginary parts.

Parameters

`in`	*real*	Real part
`in`	*imag*	Imaginary part

Returns

Complex number

4.7.2.6 SDTComplex SDTComplex_conj (SDTComplex *a*)

Returns the complex conjugate of a complex number.

Parameters

`in`	*a*	Input value

Returns

Complex conjugate of input

4.7.2.7 SDTComplex SDTComplex_div (SDTComplex *a*, SDTComplex *b*)

Returns the division between two complex numbers.

Parameters

`in`	*a*	First operand
`in`	*b*	Second operand

Returns

a divided by b

4.7.2.8 SDTComplex SDTComplex_divReal (SDTComplex *a*, double *b*)

Returns the division between a complex number and a real number.

Parameters

`in`	*a*	Complex operand
`in`	*b*	Real operand

Returns

a divided by b

4.7.2.9 SDTComplex SDTComplex_exp (double *phase*)

Returns a complex exponential with base e and given phase.

Parameters

in	*phase*	Phase

Returns

Complex exponential

4.7.2.10 SDTComplex SDTComplex_mult (SDTComplex *a*, SDTComplex *b*)

Returns the multiplication between two complex numbers.

Parameters

in	*a*	First operand
in	*b*	Second operand

Returns

a times b

4.7.2.11 SDTComplex SDTComplex_multReal (SDTComplex *a*, double *b*)

Returns the multiplication between a complex number and a real number.

Parameters

in	*a*	Complex operand
in	*b*	Real operand

Returns

a times b

4.7.2.12 SDTComplex SDTComplex_realDiv (double *a*, SDTComplex *b*)

Returns the division between a real number and a complex number.

Parameters

in	*a*	Real operand
in	*b*	Complex operand

Returns

a divided by b

4.7.2.13 SDTComplex SDTComplex_realSub (double *a*, SDTComplex *b*)

Returns the difference of a real number and a complex number.

Parameters

in	*a*	Real operand
in	*b*	Complex operand

Returns

a minus b

4.7.2.14 SDTComplex SDTComplex_sub (SDTComplex *a*, SDTComplex *b*)

Returns the difference of two complex numbers.

Parameters

in	*a*	First operand
in	*b*	Second operand

Returns

a minus b

4.7.2.15 SDTComplex SDTComplex_subReal (SDTComplex *a*, double *b*)

Returns the difference of a complex number and a real number.

Parameters

in	*a*	Complex operand
in	*b*	Real operand

Returns

a minus b

4.8 SDTControl.h: Compound solid interactions

Modules

- Bouncing
- Breaking
- Crumpling
- Rolling
- Scraping

4.8.1 Detailed Description

Objects designed to provide a temporal control layer over basic mechanical interactions, to simulate complex textures, evolving patterns and compound sound events.

4.9 Bouncing

Typedefs

- typedef struct SDTBouncing SDTBouncing

    Opaque data structure for the crumpling object.
- typedef struct SDTBouncing SDTBouncing

    Opaque data structure for the crumpling object.

Functions

- SDTBouncing ∗ SDTBouncing_new ()

    Object constructor.
- void SDTBouncing_free (SDTBouncing ∗x)

    Object destructor.
- void SDTBouncing_setRestitution (SDTBouncing ∗x, double f)

    Sets the coefficient of restitution.
- void SDTBouncing_setHeight (SDTBouncing ∗x, double f)

    Sets the initial height of the falling object.
- void SDTBouncing_setIrregularity (SDTBouncing ∗x, double f)

    Sets the irregularity of the shape of the object.
- void SDTBouncing_reset (SDTBouncing ∗x)

    Resets the bouncing process, restoring its initial energy.
- double SDTBouncing_dsp (SDTBouncing ∗x)

    Single iteration of the whole buncing process. Call this routine in a loop to simulate the bouncing process. The loop should end when SDTBouncing_hasFinished() returns true.
- int SDTBouncing_hasFinished (SDTBouncing ∗x)

    Checks if the bouncing process is finished, i.e. if the remaining energy is 0.

4.9.1 Detailed Description

Control layer for the impact model, generating (irregular) bouncing sonic textures. The output should be used to control the impact velocity between two resonators.

4.9.2 Function Documentation

4.9.2.1 double SDTBouncing_dsp (SDTBouncing ∗ *x*)

Single iteration of the whole buncing process. Call this routine in a loop to simulate the bouncing process. The loop should end when SDTBouncing_hasFinished() returns true.

Returns

Impact velocity of the bounce

4.9.2.2 void SDTBouncing_free (SDTBouncing ∗ *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.9.2.3 int SDTBouncing_hasFinished (SDTBouncing * *x*)

Checks if the bouncing process is finished, i.e. if the remaining energy is 0.

Returns

1 (true) if the remaining energy is <= 0, 0 (false) otherwise.

4.9.2.4 SDTBouncing * SDTBouncing_new ( )

Object constructor.

Returns

Pointer to the new instance

4.9.2.5 void SDTBouncing_setHeight (SDTBouncing * *x*, double *f*)

Sets the initial height of the falling object.

Parameters

in	*f*	Object height, in m.

4.9.2.6 void SDTBouncing_setIrregularity (SDTBouncing * *x*, double *f*)

Sets the irregularity of the shape of the object.

Parameters

in	*f*	Object shape irregularity (deviation from a spherical shape) [0,1]

4.9.2.7 void SDTBouncing_setRestitution (SDTBouncing * *x*, double *f*)

Sets the coefficient of restitution.

Parameters

in	*f*	Coefficient of restitution of the bouncing process

4.10 Breaking

Typedefs

- typedef struct SDTBreaking SDTBreaking

  Opaque data structure for the breaking object.
- typedef struct SDTBreaking SDTBreaking

  Opaque data structure for the breaking object.

Functions

- SDTBreaking ∗ SDTBreaking_new ()

  Object constructor.
- void SDTBreaking_free (SDTBreaking ∗x)

  Object destructor.
- void SDTBreaking_setStoredEnergy (SDTBreaking ∗x, double f)

  Sets the total energy stored in the object.
- void SDTBreaking_setCrushingEnergy (SDTBreaking ∗x, double f)

  Sets the crushing energy.
- void SDTBreaking_setGranularity (SDTBreaking ∗x, double f)

  Sets the event density of the crumpling process.
- void SDTBreaking_setFragmentation (SDTBreaking ∗x, double f)

  Sets the amount of progressive fragmentation of the object during the process.
- void SDTBreaking_reset (SDTBreaking ∗x)

  Resets the crumpling process, restoring its initial energy and triggering the first micro impact.
- void SDTBreaking_dsp (SDTBreaking ∗x, double ∗outs)

  Single iteration of the whole breaking process. Call this routine in a loop to simulate a breaking process. The loop should end when SDTBreaking_hasFinished() returns true.
- int SDTBreaking_hasFinished (SDTBreaking ∗x)

  Checks if the breaking process is finished, i.e. if the remaining energy is 0.

4.10.1 Detailed Description

Control layer for the impact model, generating breaking sonic textures. Two main outputs are exposed: energy and size. The former should be used to control the impact velocity, the latter should be used to control the size of the resonators.

4.10.2 Function Documentation

4.10.2.1 void SDTBreaking_dsp (SDTBreaking ∗ *x*, double ∗ *outs*)

Single iteration of the whole breaking process. Call this routine in a loop to simulate a breaking process. The loop should end when SDTBreaking_hasFinished() returns true.

Parameters

out	*outs*	Pointer to the output array: impact energy and fragment size

4.10.2.2 void SDTBreaking_free (SDTBreaking ∗ *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.10.2.3 int SDTBreaking_hasFinished (SDTBreaking ∗ *x*)

Checks if the breaking process is finished, i.e. if the remaining energy is 0.

Returns

1 (true) if the remaining energy is <= 0, 0 (false) otherwise.

4.10.2.4 SDTBreaking ∗ SDTBreaking_new ( )

Object constructor.

Returns

Pointer to the new instance

4.10.2.5 void SDTBreaking_reset (SDTBreaking ∗ *x*)

Resets the crumpling process, restoring its initial energy and triggering the first micro impact.

Parameters

out	*outs*	Pointer to the output array: impact energy and fragment size

4.10.2.6 void SDTBreaking_setCrushingEnergy (SDTBreaking ∗ *x*, double *f*)

Sets the crushing energy.

Parameters

in	*f*	Average energy of the micro impacts, compared to the global energy of the process, in N

4.10.2.7 void SDTBreaking_setFragmentation (SDTBreaking ∗ *x*, double *f*)

Sets the amount of progressive fragmentation of the object during the process.

Parameters

in	*f*	Object fragmentation [0, 1]

4.10.2.8 void SDTBreaking_setGranularity (SDTBreaking ∗ *x*, double *f*)

Sets the event density of the crumpling process.

Parameters

in	*f*	Event density [0, 1]

4.10.2.9 void SDTBreaking_setStoredEnergy (SDTBreaking ∗ *x*, double *f*)

Sets the total energy stored in the object.

Parameters

in	*f*	Total stored energy consumed by the micro impacts, in N

4.11 Crumpling

Typedefs

- typedef struct SDTCrumpling SDTCrumpling

  Opaque data structure for the crumpling object.
- typedef struct SDTCrumpling SDTCrumpling

  Opaque data structure for the crumpling object.

Functions

- SDTCrumpling ∗ SDTCrumpling_new ()

  Object constructor.
- void SDTCrumpling_free (SDTCrumpling ∗x)

  Object destructor.
- void SDTCrumpling_setCrushingEnergy (SDTCrumpling ∗x, double f)

  Sets the crushing energy.
- void SDTCrumpling_setGranularity (SDTCrumpling ∗x, double f)

  Sets the event density of the crumpling process.
- void SDTCrumpling_setFragmentation (SDTCrumpling ∗x, double f)

  Sets the amount of fragmentation of the object during the process.
- void SDTCrumpling_dsp (SDTCrumpling ∗x, double ∗outs)

  Single iteration of a crumpling process. Call this routine in a loop to simulate a crumpling process. Unlike in the breaking algorithm, iterations do not cause energy loss and the process can continue indefinitely until explicitly interrupted.

4.11.1 Detailed Description

Control layer for the impact model, generating crumpling sonic textures. Two main outputs are exposed: energy and size. The former should be used to control the impact velocity, the latter should be used to control the size of the resonators.

4.11.2 Function Documentation

4.11.2.1 void SDTCrumpling_dsp (SDTCrumpling ∗ *x*, double ∗ *outs*)

Single iteration of a crumpling process. Call this routine in a loop to simulate a crumpling process. Unlike in the breaking algorithm, iterations do not cause energy loss and the process can continue indefinitely until explicitly interrupted.

Parameters

`out`	*outs*	Pointer to the output array: impact energy and fragment size

4.11.2.2 void SDTCrumpling_free (SDTCrumpling ∗ *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.11.2.3 SDTCrumpling ∗ SDTCrumpling_new ( )

Object constructor.

Returns

Pointer to the new instance

4.11.2.4 void SDTCrumpling_setCrushingEnergy (SDTCrumpling ∗ *x*, double *f*)

Sets the crushing energy.

Parameters

in	*f*	Average energy of the micro impacts, compared to the global energy of the process [0, 1]

4.11.2.5 void SDTCrumpling_setFragmentation (SDTCrumpling ∗ *x*, double *f*)

Sets the amount of fragmentation of the object during the process.

Parameters

in	*f*	Object fragmentation [0, 1]

4.11.2.6 void SDTCrumpling_setGranularity (SDTCrumpling ∗ *x*, double *f*)

Sets the event density of the crumpling process.

Parameters

in	*f*	Event density [0, 1]

4.12 Rolling

Typedefs

- typedef struct SDTRolling SDTRolling

    Opaque data structure for the rolling object.
- typedef struct SDTRolling SDTRolling

    Opaque data structure for the rolling object.

Functions

- SDTRolling * SDTRolling_new ()

    Object constructor.
- void SDTRolling_free (SDTRolling *x)

    Object destructor.
- void SDTRolling_setGrain (SDTRolling *x, double f)

    Sets the grain of the surface. This parameter affects the density of the micro-impacts: Lower values result in a bumpier rolling, higher values result in a smoother rolling.
- void SDTRolling_setDepth (SDTRolling *x, double f)

    Sets the average bump depth. This parameter affects the energy of the micro-impacts.
- void SDTRolling_setMass (SDTRolling *x, double f)

    Sets the rolling mass. The mass parameter of the controlled object should be updated accordingly.
- void SDTRolling_setVelocity (SDTRolling *x, double f)

    Sets the rolling velocity.
- double SDTRolling_dsp (SDTRolling *x, double in)

    Signal processing routine. Call this function at sample rate to compute the force acting on the rolling object.

4.12.1 Detailed Description

Control layer for the impact model, generating rolling sonic textures. The output is a force, which should be applied to an inertial mass hitting a resonator.

4.12.2 Function Documentation

4.12.2.1 double SDTRolling_dsp (SDTRolling * *x*, double *in*)

Signal processing routine. Call this function at sample rate to compute the force acting on the rolling object.

Parameters

`in`	*in*	Surface profile, as an audio signal

Returns

Normal force on the exciter

4.12.2.2 void SDTRolling_free (SDTRolling * *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.12.2.3 SDTRolling * SDTRolling_new ( )

Object constructor.

Returns

Pointer to the new instance

4.12.2.4 void SDTRolling_setDepth (SDTRolling * *x*, double *f*)

Sets the average bump depth. This parameter affects the energy of the micro-impacts.

Parameters

in	*f*	Average depth of the surface bumps

4.12.2.5 void SDTRolling_setGrain (SDTRolling * *x*, double *f*)

Sets the grain of the surface. This parameter affects the density of the micro-impacts: Lower values result in a bumpier rolling, higher values result in a smoother rolling.

Parameters

in	*f*	Surface grain [0, 1]

4.12.2.6 void SDTRolling_setMass (SDTRolling * *x*, double *f*)

Sets the rolling mass. The mass parameter of the controlled object should be updated accordingly.

Parameters

in	*f*	Mass of the rolling object, in Kg

4.12.2.7 void SDTRolling_setVelocity (SDTRolling * *x*, double *f*)

Sets the rolling velocity.

Parameters

in	*f*	Rolling velocity

4.13 Scraping

Typedefs

- typedef struct SDTScraping SDTScraping

  Opaque data structure for the scraping object.
- typedef struct SDTScraping SDTScraping

  Opaque data structure for the scraping object.

Functions

- SDTScraping * SDTScraping_new ()

  Object constructor.
- void SDTScraping_free (SDTScraping *x)

  Object destructor.
- void SDTScraping_setGrain (SDTScraping *x, double f)

  Sets the grain of the surface. This parameter affects the density of the micro-impacts: Lower values result in a rougher scraping, higher values result in a smoother scraping.
- void SDTScraping_setForce (SDTScraping *x, double f)

  Sets the normal force of the scraping probe on the surface. This parameter affects the energy of the micro-impacts.
- void SDTScraping_setVelocity (SDTScraping *x, double f)

  Sets the scraping velocity.
- double SDTScraping_dsp (SDTScraping *x, double in)

  Signal processing routine. Call this function at sample rate to compute the force acting on the scraped surface.

4.13.1 Detailed Description

Control layer for resonators, generating scraping sonic textures. The output is a force, which should be applied directly to a single resonator. Interactors are not needed, although friction with another solid can be used to add a rubbing character to the sound.

4.13.2 Function Documentation

4.13.2.1 double SDTScraping_dsp (SDTScraping * *x*, double *in*)

Signal processing routine. Call this function at sample rate to compute the force acting on the scraped surface.

Parameters

in	*in*	Surface profile, as an audio signal

Returns

Normal force on the resonator

4.13.2.2 void SDTScraping_free (SDTScraping * *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.13.2.3 SDTScraping * SDTScraping_new ( )

Object constructor.

Returns

Pointer to the new instance

4.13.2.4 void SDTScraping_setForce (SDTScraping * *x*, double *f*)

Sets the normal force of the scraping probe on the surface. This parameter affects the energy of the micro-impacts.

Parameters

in	*f*	Normal force of the scraping probe on the resonating surface

4.13.2.5 void SDTScraping_setGrain (SDTScraping * *x*, double *f*)

Sets the grain of the surface. This parameter affects the density of the micro-impacts: Lower values result in a rougher scraping, higher values result in a smoother scraping.

Parameters

in	*f*	Surface grain [0, 1]

4.13.2.6 void SDTScraping_setVelocity (SDTScraping * *x*, double *f*)

Sets the scraping velocity.

Parameters

in	*f*	Probe velocity

4.14 SDTDCMotor.h: Electric motors

Typedefs

- typedef struct SDTDCMotor SDTDCMotor

  Opaque data structure for the electric motor synthesis model.
- typedef struct SDTDCMotor SDTDCMotor

  Opaque data structure for the electric motor synthesis model.

Functions

- SDTDCMotor ∗ SDTDCMotor_new (long maxSize)

  Object constructor.
- void SDTDCMotor_free (SDTDCMotor ∗x)

  Object destructor.
- void SDTDCMotor_setFilters (SDTDCMotor ∗x)

  Sets the filter coefficients. Call this function whenever the sample rate changes.
- void SDTDCMotor_setRpm (SDTDCMotor ∗x, double f)

  Sets the Revolutions Per Minute (RPM) of the engine rotor.
- void SDTDCMotor_setLoad (SDTDCMotor ∗x, double f)

  Sets the mechanical stress on the rotor.
- void SDTDCMotor_setCoils (SDTDCMotor ∗x, long l)

  Sets the number of coils on the rotor.
- void SDTDCMotor_setSize (SDTDCMotor ∗x, double f)

  Sets the size of the chassis. The maximum chassis size depends on the buffer length defined at construction time and on the current sampling rate.
- void SDTDCMotor_setReson (SDTDCMotor ∗x, double f)

  Sets the amount of resonance caused by the chassis.
- void SDTDCMotor_setGearRatio (SDTDCMotor ∗x, double f)

  Sets the gear ratio of the engine.
- void SDTDCMotor_setHarshness (SDTDCMotor ∗x, double f)

  Sets the harshness of the engine sound.
- void SDTDCMotor_setRotorGain (SDTDCMotor ∗x, double f)

  Sets the sound volume coming from the rotor.
- void SDTDCMotor_setGearGain (SDTDCMotor ∗x, double f)

  Sets the sound volume coming from the gears.
- void SDTDCMotor_setBrushGain (SDTDCMotor ∗x, double f)

  Sets the sound volume coming from the commutator ring and brushes.
- void SDTDCMotor_setAirGain (SDTDCMotor ∗x, double f)

  Sets the sound volume of the air turbulence caused by rotation.
- double SDTDCMotor_dsp (SDTDCMotor ∗x)

  Signal processing routine. Call this function at sample rate to synthesize an electric motor sound.

4.14.1 Detailed Description

Physically informed model for the synthesis of electric motor sounds.

Electric motors exploit magnetic induction to convert electric energy into mechanical energy. An ideal electric motor should be perfectly silent. In practice, however, rotors are never perfectly balanced and generate pitched tones depending on their revolutions per minute (RPM). Moreover, contacts between parts cause friction noise. Finally, rotation causes air movement and therefore turbulence noise, sometimes increased by the presence of a cooling fan attached to the rotor.

The pitched tone of the rotor is obtained through additive synthesis, summing a fixed number of harmonic partials. Frequency modulation simulates the unevenness in the rotation caused by attached loads. Resonances modes of the chassis are modeled through a comb filter. Aerodynamic turbulence caused by the spinning parts is synthesized with bandpass-filtered white noise, exactly like in the gas model.

4.14.2 Function Documentation

4.14.2.1 double SDTDCMotor_dsp (SDTDCMotor * *x*)

Signal processing routine. Call this function at sample rate to synthesize an electric motor sound.

Returns

Computed audio sample

4.14.2.2 void SDTDCMotor_free (SDTDCMotor * *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.14.2.3 SDTDCMotor * SDTDCMotor_new (long *maxSize*)

Object constructor.

Parameters

in	*maxSize*	Buffer length of the internal comb filter, in samples

Returns

Pointer to the new instance

4.14.2.4 void SDTDCMotor_setAirGain (SDTDCMotor * *x*, double *f*)

Sets the sound volume of the air turbulence caused by rotation.

Parameters

in	*f*	Air gain [0, 1]

4.14.2.5 void SDTDCMotor_setBrushGain (SDTDCMotor * *x*, double *f*)

Sets the sound volume coming from the commutator ring and brushes.

Parameters

in	*f*	Brush gain [0, 1]

4.14.2.6 void SDTDCMotor_setCoils (SDTDCMotor * *x*, long *l*)

Sets the number of coils on the rotor.

Parameters

in	*l*	Number of coils on the rotor

4.14.2.7 void SDTDCMotor_setGearGain (SDTDCMotor ∗ x, double f)

Sets the sound volume coming from the gears.

Parameters

in	*f*	Gear gain [0, 1]

4.14.2.8 void SDTDCMotor_setGearRatio (SDTDCMotor ∗ x, double f)

Sets the gear ratio of the engine.

Parameters

in	*f*	Gear ratio

4.14.2.9 void SDTDCMotor_setHarshness (SDTDCMotor ∗ x, double f)

Sets the harshness of the engine sound.

Parameters

in	*f*	Harshness [0, 1]

4.14.2.10 void SDTDCMotor_setLoad (SDTDCMotor ∗ x, double f)

Sets the mechanical stress on the rotor.

Parameters

in	*f*	Engine load [0, 1]

4.14.2.11 void SDTDCMotor_setReson (SDTDCMotor ∗ x, double f)

Sets the amount of resonance caused by the chassis.

Parameters

in	*f*	Chassis resonance [0, 1]

4.14.2.12 void SDTDCMotor_setRotorGain (SDTDCMotor ∗ x, double f)

Sets the sound volume coming from the rotor.

Parameters

in	*f*	Rotor gain [0, 1]

4.14.2.13 void SDTDCMotor_setRpm (SDTDCMotor * *x*, double *f*)

Sets the Revolutions Per Minute (RPM) of the engine rotor.

Parameters

in	*f*	Engine RPM

4.14.2.14 void SDTDCMotor_setSize (SDTDCMotor * *x*, double *f*)

Sets the size of the chassis. The maximum chassis size depends on the buffer length defined at construction time and on the current sampling rate.

Parameters

in	*f*	Chassis length, in m

4.15 SDTDemix.h: Transient/tonal/residual components separator

Typedefs

- typedef struct SDTDemix SDTDemix

  Opaque data structure for the percussive/harmonic/residual components separator.
- typedef struct SDTDemix SDTDemix

  Opaque data structure for the percussive/harmonic/residual components separator.

Functions

- SDTDemix ∗ SDTDemix_new (int size, int radius)

  Object constructor.
- void SDTDemix_free (SDTDemix ∗x)

  Object destructor.
- void SDTDemix_setOverlap (SDTDemix ∗x, double f)

  Sets the window overlapping factor.
- void SDTDemix_setNoiseThreshold (SDTDemix ∗x, double f)

  Sets the noise threshold.
- void SDTDemix_setTonalThreshold (SDTDemix ∗x, double f)

  Sets the tornal threshold.
- void SDTDemix_dsp (SDTDemix ∗x, double ∗outs, double in)

  Signal processing routine. Call this function at sample rate to separate an arbitrary signal into its percussive/harmonic/residual components.

4.15.1 Detailed Description

This algorithm looks for vertical and horizontal structures in the spectrogram to separate an arbitrary audio signal into its percussive (transients), harmonic (sustained tones) and residual (noise) components. It is based on a technique called structure tensor, frequently used in image processing for edge and corner detection or to estimate the orientation of an object.

The structure tensor can be thought as a smoothed gradient of the spectrogram, which describes the consistency and direction of changes in the energy content of each bin. The anisotropy (consistency) and direction descriptors extracted from the structure tensor are used to classify the spectrogram bins into three categories: Bins which do not exhibit a particular gradient direction (low anisotropy) become part of the residual, noisy component; Bins which tend to have a vertical orientation (high anisotropy, high direction) are included in the percussive component; Bins with a mostly horizontal orientation (high anisotropy, low direction) fall into the harmonic component.

This percussive/harmonic/residual separation is suitable to separate attacks, decays and noise from a musical signal, or to isolate myoelastic (percussive), phonatory (harmonic) and turbulent (noisy) activities from a vocal signal. In particular, the algorithm can be used as a preprocessing step to improve the results of the myoelastic detector, pitch tracker and spectral moments extractor present in the analysis part of the Sound Design Toolkit.

4.15.2 Function Documentation

4.15.2.1 void SDTDemix_dsp (SDTDemix ∗ *x*, double ∗ *outs*, double *in*)

Signal processing routine. Call this function at sample rate to separate an arbitrary signal into its percussive/harmonic/residual components.

Parameters

out	*outs*	Pointer to a 3-elements output array. outs[0] is the percussive component, outs[1] is the harmonic component and outs[2] is the residual.
in	*in*	Input sample

4.15.2.2 void SDTDemix_free (SDTDemix ∗ *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.15.2.3 SDTDemix ∗ SDTDemix_new (int *size*, int *radius*)

Object constructor.

Parameters

in	*size*	Analysis window length, in samples
in	*radius*	Smoothing kernel radius, in samples

Returns

Pointer to the new instance

4.15.2.4 void SDTDemix_setNoiseThreshold (SDTDemix ∗ *x*, double *f*)

Sets the noise threshold.

Parameters

in	*f*	Amount of signal falling into the residual category

4.15.2.5 void SDTDemix_setOverlap (SDTDemix ∗ *x*, double *f*)

Sets the window overlapping factor.

Parameters

in	*f*	Window overlapping factor

4.15.2.6 void SDTDemix_setTonalThreshold (SDTDemix ∗ *x*, double *f*)

Sets the tornal threshold.

Parameters

in	*f*	Amount of non-residual falling into the tonal category

4.16 SDTEffects.h: Digital audio effects

Modules

- Reverb
- Pitch shift

4.16.1 Detailed Description

Algorithms for audio post-processing, such as reverberation and pitch shifting

4.17 Reverb

Typedefs

- typedef struct SDTReverb SDTReverb

  Opaque data structure for a reverberator object.
- typedef struct SDTReverb SDTReverb

  Opaque data structure for a reverberator object.

Functions

- SDTReverb * SDTReverb_new (long maxDelay)

  Object constructor.
- void SDTReverb_free (SDTReverb *x)

  Object destructor.
- void SDTReverb_setXSize (SDTReverb *x, double f)

  Sets the room width.
- void SDTReverb_setYSize (SDTReverb *x, double f)

  Sets the room height.
- void SDTReverb_setZSize (SDTReverb *x, double f)

  Sets the room depth.
- void SDTReverb_setRandomness (SDTReverb *x, double f)

  Sets how randomly distributed are the resonant modes. This parameter is directly proportional to the irregularity of the shape of the room.
- void SDTReverb_setTime (SDTReverb *x, double f)

  Sets the global, frequency-independent reverberation time.
- void SDTReverb_setTime1k (SDTReverb *x, double f)

  Sets the reverberation time at 1kHz.
- void SDTReverb_update (SDTReverb *x)

  Updates the internal filters. Call this function after every sample rate change.
- double SDTReverb_dsp (SDTReverb *x, double in)

  Signal processing routine. Call this function at sample rate to compute the reverberated signal.

4.17.1 Detailed Description

Artificial reverberator based on Feedback Delay Networks, as found in D. Rocchesso, "Maximally diffusive yet efficient feedback delay networks for artificial reverberation", Signal Processing Letters, IEEE 4.9 (1997): 252-255.

4.17.2 Function Documentation

4.17.2.1 double SDTReverb_dsp (SDTReverb * *x*, double *in*)

Signal processing routine. Call this function at sample rate to compute the reverberated signal.

Parameters

in	*in*	Input sample

Returns

Output sample

4.17.2.2 void SDTReverb_free (SDTReverb * *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.17.2.3 SDTReverb * SDTReverb_new (long *maxDelay*)

Object constructor.

Parameters

in	*maxDelay*	Maximum length of the delay lines, in samples

Returns

Pointer to the new instance

4.17.2.4 void SDTReverb_setRandomness (SDTReverb * *x*, double *f*)

Sets how randomly distributed are the resonant modes. This parameter is directly proportional to the irregularity of the shape of the room.

Parameters

in	*f*	Randomness in the modal distribution [0, 1]

4.17.2.5 void SDTReverb_setTime (SDTReverb * *x*, double *f*)

Sets the global, frequency-independent reverberation time.

Parameters

in	*f*	Reverberation time, in s

4.17.2.6 void SDTReverb_setTime1k (SDTReverb * *x*, double *f*)

Sets the reverberation time at 1kHz.

Parameters

in	*f*	Reverberation time at 1kHz, in s

4.17.2.7 void SDTReverb_setXSize (SDTReverb * *x*, double *f*)

Sets the room width.

Parameters

in	*f*	Room width, in m

4.17.2.8 void SDTReverb_setYSize (SDTReverb * *x*, double *f*)

Sets the room height.

Parameters

in	*f*	Room height, in m

4.17.2.9 void SDTReverb_setZSize (SDTReverb * *x*, double *f*)

Sets the room depth.

Parameters

in	*f*	Room depth, in m

4.18 Pitch shift

Typedefs

- typedef struct SDTPitchShift SDTPitchShift

  Opaque data structure for a pitch shifter object.
- typedef struct SDTPitchShift SDTPitchShift

  Opaque data structure for a pitch shifter object.

Functions

- SDTPitchShift ∗ SDTPitchShift_new (int size, int oversample)

  Object constructor.
- void SDTPitchShift_free (SDTPitchShift ∗x)

  Object destructor.
- void SDTPitchShift_setRatio (SDTPitchShift ∗x, double f)

  Sets the pitch shifting ratio.
- void SDTPitchShift_setOverlap (SDTPitchShift ∗x, double f)

  Sets the analysis window overlapping ratio. Accepted values go from 0.0 to 1.0, with 0.0 meaning no overlap and 1.0 meaning total overlap.
- double SDTPitchShift_dsp (SDTPitchShift ∗x, double in)

  Signal processing routine. Call this function at sample rate to compute the pitch shifted signal.

4.18.1 Detailed Description

Frequency domain pitch shifter, useful to simulate doppler effect or other applications requiring pitch shifting.

4.18.2 Function Documentation

4.18.2.1 double SDTPitchShift_dsp (SDTPitchShift ∗ *x*, double *in*)

Signal processing routine. Call this function at sample rate to compute the pitch shifted signal.

Parameters

in	*in*	Input sample

Returns

Output sample

4.18.2.2 void SDTPitchShift_free (SDTPitchShift ∗ *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.18.2.3 SDTPitchShift ∗ SDTPitchShift_new (int *size*, int *oversample*)

Object constructor.

Parameters

in	*size*	Internal buffer size, in samples
in	*oversample*	FFT oversampling rate

Returns

Pointer to the new instance

4.18.2.4 void SDTPitchShift_setOverlap (SDTPitchShift * *x*, double *f*)

Sets the analysis window overlapping ratio. Accepted values go from 0.0 to 1.0, with 0.0 meaning no overlap and 1.0 meaning total overlap.

Parameters

in	*x*	Pointer to the instance
in	*f*	Overlap ratio [0.0, 1.0]

4.18.2.5 void SDTPitchShift_setRatio (SDTPitchShift * *x*, double *f*)

Sets the pitch shifting ratio.

Parameters

in	*f*	New pitch / original pitch ratio

4.19 SDTFFT.h: Fast Fourier Transform

Typedefs

- typedef struct SDTFFT SDTFFT

  Opaque data structure, representing a FFT object.
- typedef struct SDTFFT SDTFFT

  Opaque data structure, representing a FFT object.

Functions

- SDTFFT ∗ SDTFFT_new (unsigned int n)

  Object constructor.
- void SDTFFT_free (SDTFFT ∗x)

  Object destructor.
- void SDTFFT_fft (SDTFFT ∗x, int inverse, SDTComplex ∗in, SDTComplex ∗out)

  Performs a direct or inverse FFT of a complex-valued signal.
- void SDTFFT_fftr (SDTFFT ∗x, double ∗in, SDTComplex ∗out)

  Performs a direct FFT of a real-valued signal.
- void SDTFFT_ifftr (SDTFFT ∗x, SDTComplex ∗in, double ∗out)

  Performs an inverse FFT of a signal known to be real-valued.

4.19.1 Detailed Description

Data structures and functions to perform frequency analysis on signals by means of the Discrete Fourier Transform and its inverse. This implementation is based on the iterative version of the Cooley-Tukey algorithm, works with double precision floating point arithmetic and provides an optimization for the transformation of real-valued signals.

4.19.2 Function Documentation

4.19.2.1 void SDTFFT_fft (SDTFFT ∗ *x*, int *inverse*, SDTComplex ∗ *in*, SDTComplex ∗ *out*)

Performs a direct or inverse FFT of a complex-valued signal.

Parameters

in	*inverse*	Perform a direct FFT if 0, or an inverse FFT otherwise
in	*in*	Input signal to transform, must be at least of length n
out	*out*	Transformed output, must be at least of length n. When performing an inverse transform, divide every sample by n to obtain the original signal

4.19.2.2 void SDTFFT_fftr (SDTFFT ∗ *x*, double ∗ *in*, SDTComplex ∗ *out*)

Performs a direct FFT of a real-valued signal.

Parameters

in	*in*	Input signal to transform, must be at least of length 2n
out	*out*	Transformed output

4.19.2.3 void SDTFFT_free (SDTFFT ∗ *x*)

Object destructor.

Parameters

in	*Pointer*	to the instance to destroy

4.19.2.4 void SDTFFT_ifftr (SDTFFT * *x*, SDTComplex * *in*, double * *out*)

Performs an inverse FFT of a signal known to be real-valued.

Parameters

in	*in*	Input FFT to invert
out	*out*	Reconstructed signal. Divide every sample by n to obtain the original signal

4.19.2.5 SDTFFT * SDTFFT_new (unsigned int *n*)

Object constructor.

Parameters

in	*n*	FFT window length, must be a power of 2

Returns

Pointer to the newly created instance, or NULL if n is not a power of 2

4.20 SDTFilters.h: Audio filters

Modules

- One pole filter
- Allpass filter
- Envelope follower
- Two poles filter
- Cascade of biquadratic sections
- Moving average
- Delay line
- Comb filter
- Digital waveguide

4.20.1 Detailed Description

Various commonly used LTI systems: filters, delay lines, circular buffers, waveguides and so on. Extensively used in many other SDT modules.

4.21 One pole filter

Typedefs

- typedef struct SDTOnePole SDTOnePole

    Opaque data structure for a one pole filter object.
- typedef struct SDTOnePole SDTOnePole

    Opaque data structure for a one pole filter object.

Functions

- SDTOnePole ∗ SDTOnePole_new ()

    Object constructor.
- void SDTOnePole_free (SDTOnePole ∗x)

    Object destructor.
- void SDTOnePole_setFeedback (SDTOnePole ∗x, double f)

    Manually sets the alpha coefficient.
- void SDTOnePole_lowpass (SDTOnePole ∗x, double f)

    Puts the filter in lowpass mode, at the given cutoff frequency.
- void SDTOnePole_highpass (SDTOnePole ∗x, double f)

    Puts the filter in highpass mode, at the given cutoff frequency.
- double SDTOnePole_dsp (SDTOnePole ∗x, double in)

    Signal processing routine. Call this function at sample rate to compute the filtered signal.

4.21.1 Detailed Description

Simple one pole filter.

4.21.2 Function Documentation

4.21.2.1 double SDTOnePole_dsp (SDTOnePole ∗ *x*, double *in*)

Signal processing routine. Call this function at sample rate to compute the filtered signal.

Parameters

in	*in*	Input sample

Returns

Output sample

4.21.2.2 void SDTOnePole_free (SDTOnePole ∗ *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.21.2.3 void SDTOnePole_highpass (SDTOnePole ∗ *x*, double *f*)

Puts the filter in highpass mode, at the given cutoff frequency.

Parameters

in	*f*	Cutoff frequency, in Hz

4.21.2.4 void SDTOnePole_lowpass (SDTOnePole * *x*, double *f*)

Puts the filter in lowpass mode, at the given cutoff frequency.

Parameters

in	*f*	Cutoff frequency, in Hz

4.21.2.5 SDTOnePole * SDTOnePole_new ( )

Object constructor.

Returns

Pointer to the new instance

4.21.2.6 void SDTOnePole_setFeedback (SDTOnePole * *x*, double *f*)

Manually sets the alpha coefficient.

Parameters

in	*f*	Weight of the input sample

4.22 Allpass filter

Typedefs

- typedef struct SDTAllPass SDTAllPass

    Opaque data structure for an allpass filter object.
- typedef struct SDTAllPass SDTAllPass

    Opaque data structure for an allpass filter object.

Functions

- SDTAllPass ∗ SDTAllPass_new ()

    Object constructor.
- void SDTAllPass_free (SDTAllPass ∗x)

    Object destructor.
- void SDTAllPass_setFeedback (SDTAllPass ∗x, double f)

    Sets the feedback coefficient.
- double SDTAllPass_dsp (SDTAllPass ∗x, double in)

    Signal processing routine. Call this function at sample rate to compute the filtered signal.

4.22.1 Detailed Description

Allpass filter, used to adjust phases in fractional delay lines.

4.22.2 Function Documentation

4.22.2.1 double SDTAllPass_dsp (SDTAllPass ∗ *x*, double *in*)

Signal processing routine. Call this function at sample rate to compute the filtered signal.

Parameters

in	*in*	Input sample

Returns

Output sample

4.22.2.2 void SDTAllPass_free (SDTAllPass ∗ *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.22.2.3 SDTAllPass ∗ SDTAllPass_new ( )

Object constructor.

Returns

Pointer to the new instance

4.22.2.4 void SDTAllPass_setFeedback (SDTAllPass * x, double f)

Sets the feedback coefficient.

Parameters

`in`	*f*	Weight of the input sample

4.23 Envelope follower

Typedefs

- typedef struct SDTEnvelope SDTEnvelope

  Opaque data structure for an envelope tracker object.
- typedef struct SDTEnvelope SDTEnvelope

  Opaque data structure for an envelope tracker object.

Functions

- SDTEnvelope ∗ SDTEnvelope_new ()

  Object constructor.
- void SDTEnvelope_free (SDTEnvelope ∗x)

  Object destructor.
- void SDTEnvelope_setAttack (SDTEnvelope ∗x, double a)

  Sets the attack time.
- void SDTEnvelope_setRelease (SDTEnvelope ∗x, double r)

  Sets the release time.
- double SDTEnvelope_dsp (SDTEnvelope ∗x, double in)

  Signal processing routine. Call this function at sample rate to compute the filtered signal.

4.23.1 Detailed Description

One pole envelope follower, with independent attack and release times.

4.23.2 Function Documentation

4.23.2.1 double SDTEnvelope_dsp (SDTEnvelope ∗ *x*, double *in*)

Signal processing routine. Call this function at sample rate to compute the filtered signal.

Parameters

in	*in*	Input sample

Returns

Output sample

4.23.2.2 void SDTEnvelope_free (SDTEnvelope ∗ *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.23.2.3 SDTEnvelope * SDTEnvelope_new ( )

Object constructor.

Returns

Pointer to the new instance

4.23.2.4 void SDTEnvelope_setAttack (SDTEnvelope * *x*, double *a*)

Sets the attack time.

Parameters

in	*a*	Attack time, in ms

4.23.2.5 void SDTEnvelope_setRelease (SDTEnvelope * *x*, double *r*)

Sets the release time.

Parameters

in	*r*	Release time, in ms

4.24 Two poles filter

Typedefs

- typedef struct SDTTwoPoles SDTTwoPoles

  Opaque data structure for a two poles filter object.
- typedef struct SDTTwoPoles SDTTwoPoles

  Opaque data structure for a two poles filter object.

Functions

- SDTTwoPoles * SDTTwoPoles_new ()

  Object constructor.
- void SDTTwoPoles_free (SDTTwoPoles *x)

  Object destructor.
- void SDTTwoPoles_lowpass (SDTTwoPoles *x, double fc)

  Puts the filter in lowpass mode, at the given cutoff frequency.
- void SDTTwoPoles_highpass (SDTTwoPoles *x, double fc)

  Puts the filter in highpass mode, at the given cutoff frequency.
- void SDTTwoPoles_resonant (SDTTwoPoles *x, double fc, double q)

  Puts the filter in resonant bandpass mode, at the given center frequency and Q.
- double SDTTwoPoles_dsp (SDTTwoPoles *x, double in)

  Signal processing routine. Call this function at sample rate to compute the filtered signal.

4.24.1 Detailed Description

Two poles filter, configurable as lowpass, highpass or resonant bandpass.

4.24.2 Function Documentation

4.24.2.1 double SDTTwoPoles_dsp (SDTTwoPoles * *x*, double *in*)

Signal processing routine. Call this function at sample rate to compute the filtered signal.

Parameters

in	*in*	Input sample

Returns

Output sample

4.24.2.2 void SDTTwoPoles_free (SDTTwoPoles * *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.24.2.3 void SDTTwoPoles_highpass (SDTTwoPoles * *x*, double *fc*)

Puts the filter in highpass mode, at the given cutoff frequency.

Parameters

in	*fc*	Cutoff frequency, in Hz

4.24.2.4 void SDTTwoPoles_lowpass (SDTTwoPoles * *x*, double *fc*)

Puts the filter in lowpass mode, at the given cutoff frequency.

Parameters

in	*fc*	Cutoff frequency, in Hz

4.24.2.5 SDTTwoPoles * SDTTwoPoles_new ( )

Object constructor.

Returns

Pointer to the new instance

4.24.2.6 void SDTTwoPoles_resonant (SDTTwoPoles * *x*, double *fc*, double *q*)

Puts the filter in resonant bandpass mode, at the given center frequency and Q.

Parameters

in	*fc*	Center frequency, in Hz
in	*q*	Q factor, in 1/octave

4.25 Cascade of biquadratic sections

Typedefs

- typedef struct SDTBiquad SDTBiquad

  Opaque data structure for biquad cascade object.
- typedef struct SDTBiquad SDTBiquad

  Opaque data structure for biquad cascade object.

Functions

- SDTBiquad ∗ SDTBiquad_new (int nSections)

  Object constructor.
- void SDTBiquad_free (SDTBiquad ∗x)

  Object destructor.
- void SDTBiquad_butterworthLP (SDTBiquad ∗x, double fc)

  Designs a Butterworth lowpass filter, at the given cutoff frequency.
- void SDTBiquad_butterworthHP (SDTBiquad ∗x, double fc)

  Designs a Butterworth highpass filter, at the given cutoff frequency.
- void **SDTBiquad_butterworthAP** (SDTBiquad ∗x, double fc)
- void SDTBiquad_linkwitzRileyLP (SDTBiquad ∗x, double fc)

  Designs the lowpass part of a Linkwitz-Riley crossover filter, at the given cutoff frequency. WARNING: the filter must have an even number of biquad sections!
- void SDTBiquad_linkwitzRileyHP (SDTBiquad ∗x, double fc)

  Designs the highpass part of a Linkwitz-Riley crossover filter, at the given cutoff frequency. WARNING: the filter must have an even number of biquad sections!
- double SDTBiquad_dsp (SDTBiquad ∗x, double in)

  Signal processing routine. Call this function at sample rate to compute the filtered signal.

4.25.1 Detailed Description

Classic cascade of biquad sections, useful to implement a wide variety of filters.

4.25.2 Function Documentation

4.25.2.1 void SDTBiquad_butterworthHP (SDTBiquad ∗ *x*, double *fc*)

Designs a Butterworth highpass filter, at the given cutoff frequency.

Parameters

in	*fc*	Cutoff frequency, in Hz

4.25.2.2 void SDTBiquad_butterworthLP (SDTBiquad ∗ *x*, double *fc*)

Designs a Butterworth lowpass filter, at the given cutoff frequency.

Parameters

in	*fc*	Cutoff frequency, in Hz

4.25.2.3 double SDTBiquad_dsp (SDTBiquad * *x*, double *in*)

Signal processing routine. Call this function at sample rate to compute the filtered signal.

Parameters

in	*in*	Input sample

Returns

Output sample

4.25.2.4 void SDTBiquad_free (SDTBiquad * *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.25.2.5 void SDTBiquad_linkwitzRileyHP (SDTBiquad * *x*, double *fc*)

Designs the highpass part of a Linkwitz-Riley crossover filter, at the given cutoff frequency. WARNING: the filter must have an even number of biquad sections!

Parameters

in	*fc*	Cutoff frequency, in Hz

4.25.2.6 void SDTBiquad_linkwitzRileyLP (SDTBiquad * *x*, double *fc*)

Designs the lowpass part of a Linkwitz-Riley crossover filter, at the given cutoff frequency. WARNING: the filter must have an even number of biquad sections!

Parameters

in	*fc*	Cutoff frequency, in Hz

4.25.2.7 SDTBiquad * SDTBiquad_new (int *nSections*)

Object constructor.

Parameters

in	*nSections*	Number of sections in the cascade. The order of the resulting filter is twice this value (i.e. nSections = 4 -> order = 8).

Returns

Pointer to the new instance

4.26 Moving average

Typedefs

- typedef struct SDTAverage SDTAverage

  Opaque data structure for a moving average filter object.
- typedef struct SDTAverage SDTAverage

  Opaque data structure for a moving average filter object.

Functions

- SDTAverage ∗ SDTAverage_new (long size)

  Object constructor.
- void SDTAverage_free (SDTAverage ∗x)

  Object destructor.
- void SDTAverage_setWindow (SDTAverage ∗x, unsigned int i)

  Sets the averaging window.
- double SDTAverage_dsp (SDTAverage ∗x, double in)

  Signal processing routine. Call this function at sample rate to compute the filtered signal.

4.26.1 Detailed Description

Moving average filter, producing as output the average of the last input samples.

4.26.2 Function Documentation

4.26.2.1 double SDTAverage_dsp (SDTAverage ∗ *x*, double *in*)

Signal processing routine. Call this function at sample rate to compute the filtered signal.

Parameters

in	*in*	Input sample

Returns

Output sample

4.26.2.2 void SDTAverage_free (SDTAverage ∗ *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.26.2.3 SDTAverage ∗ SDTAverage_new (long *size*)

Object constructor.

Parameters

in	*size*	Moving average buffer size

Returns

Pointer to the new instance

4.26.2.4 void SDTAverage_setWindow (SDTAverage * x, unsigned int i)

Sets the averaging window.

Parameters

in	*size*	Moving average window size [1,bufferSize]

4.27 Delay line

Typedefs

- typedef struct SDTDelay SDTDelay

  Opaque data structure for a delay line object.
- typedef struct SDTDelay SDTDelay

  Opaque data structure for a delay line object.

Functions

- SDTDelay * SDTDelay_new (long maxDelay)

  Object constructor.
- void SDTDelay_free (SDTDelay *x)

  Object destructor.
- void SDTDelay_clear (SDTDelay *x)

  Clears the buffer, therefore silencing the delayed signal.
- void SDTDelay_setDelay (SDTDelay *x, double f)

  Sets the delay time. Fractional values are allowed. The delay time can be continuously changed over time without audible glitches.
- double SDTDelay_dsp (SDTDelay *x, double in)

  Signal processing routine. Call this function at sample rate to output the delayed signal.

4.27.1 Detailed Description

Delay line, supporting fractional and time-varying delay lengths.

4.27.2 Function Documentation

4.27.2.1 double SDTDelay_dsp (SDTDelay * *x*, double *in*)

Signal processing routine. Call this function at sample rate to output the delayed signal.

Parameters

in	*in*	Input sample

Returns

Output sample

4.27.2.2 void SDTDelay_free (SDTDelay * *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.27.2.3 SDTDelay * SDTDelay_new (long *maxDelay*)

Object constructor.

Parameters

in	*maxDelay*	Buffer size, determining the maximum delay length, in samples

Returns

Pointer to the new instance

4.27.2.4 void SDTDelay_setDelay (SDTDelay * *x*, double *f*)

Sets the delay time. Fractional values are allowed. The delay time can be continuously changed over time without audible glitches.

Parameters

in	*f*	Delay time, in samples

4.28 Comb filter

Typedefs

- typedef struct SDTComb SDTComb
  Opaque data structure representing a comb filter object.
- typedef struct SDTComb SDTComb
  Opaque data structure representing a comb filter object.

Functions

- SDTComb ∗ SDTComb_new (long maxXDelay, long maxYDelay)
  Object constructor.
- void SDTComb_free (SDTComb ∗x)
  Object destructor.
- void SDTComb_setXDelay (SDTComb ∗x, double f)
  Sets the delay time for the feed forward section.
- void SDTComb_setYDelay (SDTComb ∗x, double f)
  Sets the delay time for the feedback section.
- void SDTComb_setXYDelay (SDTComb ∗x, double f)
  Sets the delay time for both sections.
- void SDTComb_setXGain (SDTComb ∗x, double f)
  Sets the gain for the feed forward section.
- void SDTComb_setYGain (SDTComb ∗x, double f)
  Sets the gain for the feedback section.
- void SDTComb_setXYGain (SDTComb ∗x, double f)
  Sets the gain for both sections.
- double SDTComb_dsp (SDTComb ∗x, double in)
  Signal processing routine. Call this function at sample rate to output the filtered signal.

4.28.1 Detailed Description

Comb filter, obtained adding to the input signal a rescaled and delayed copy of itself. The filter works both in feed forward (delayed copy added to the output) and feedback (delayed copy added to the input, causing a loop) configurations, with independent gains and delay times.

4.28.2 Function Documentation

4.28.2.1 double SDTComb_dsp (**SDTComb** ∗ *x*, double *in*)

Signal processing routine. Call this function at sample rate to output the filtered signal.

Parameters

in	*in*	Input sample

Returns

Output sample

4.28.2.2 void SDTComb_free (**SDTComb** ∗ *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.28.2.3 SDTComb * SDTComb_new (long *maxXDelay*, long *maxYDelay*)

Object constructor.

Parameters

in	*maxXDelay*	Feed forward buffer size, in samples
in	*maxXDelay*	Feedback buffer size, in samples

Returns

Pointer to the new instance

4.28.2.4 void SDTComb_setXDelay (SDTComb * *x*, double *f*)

Sets the delay time for the feed forward section.

Parameters

in	*f*	Feed forward delay time, in samples

4.28.2.5 void SDTComb_setXGain (SDTComb * *x*, double *f*)

Sets the gain for the feed forward section.

Parameters

in	*f*	Feed forward gain [0,1]

4.28.2.6 void SDTComb_setXYDelay (SDTComb * *x*, double *f*)

Sets the delay time for both sections.

Parameters

in	*f*	Delay time, in samples

4.28.2.7 void SDTComb_setXYGain (SDTComb * *x*, double *f*)

Sets the gain for both sections.

Parameters

in	*f*	Gain [0,1]

4.28.2.8 void SDTComb_setYDelay (SDTComb * *x*, double *f*)

Sets the delay time for the feedback section.

Parameters

in	*f*	Feedback delay time, in samples

4.28.2.9 void SDTComb_setYGain (SDTComb * x, double f)

Sets the gain for the feedback section.

Parameters

in	*f*	Feedback gain [0,1]

4.29 Digital waveguide

Typedefs

- typedef struct SDTWaveguide SDTWaveguide

  Opaque data structure representing a digital waveguide object.
- typedef struct SDTWaveguide SDTWaveguide

  Opaque data structure representing a digital waveguide object.

Functions

- SDTWaveguide ∗ SDTWaveguide_new (int maxDelay)

  Object constructor.
- void SDTWaveguide_free (SDTWaveguide ∗x)

  Object destructor.
- double SDTWaveguide_getFwdOut (SDTWaveguide ∗x)

  Reads the output signal coming from the right side of the waveguide.
- double SDTWaveguide_getRevOut (SDTWaveguide ∗x)

  Reads the output signal coming from the left side of the waveguide.
- void SDTWaveguide_setDelay (SDTWaveguide ∗x, double f)

  Sets the length of the waveguide, in samples.
- void SDTWaveguide_setFwdFeedback (SDTWaveguide ∗x, double f)

  Sets the feedback on the right side. Determines how much energy gets fed back into the system after the wave reaches the right side of the waveguide. Consequently, this value also determines how much attenuated is the output on the same side.
- void SDTWaveguide_setRevFeedback (SDTWaveguide ∗x, double f)

  Sets the feedback on the left side. Determines how much energy gets fed back into the system after the wave reaches the left side of the waveguide. Consequently, this value also determines how much attenuated is the output on the same side.
- void SDTWaveguide_setFwdDamping (SDTWaveguide ∗x, double f)

  Sets the frequency damping on the right side.
- void SDTWaveguide_setRevDamping (SDTWaveguide ∗x, double f)

  Sets the frequency damping on the left side.
- void SDTWaveguide_dsp (SDTWaveguide ∗x, double fwdIn, double revIn)

  Signal processing routine. Call this function at sample rate to compute the output samples. To read them, call the respective functions SDTWaveguide_getFwdOut() and SDTWaveguide_getRevOut().

4.29.1 Detailed Description

Digital waveguide, simulating relection/refraction of waves in a medium such as the air column in a tube or a vibrating string. Composed of two delay lines of the same length, in a mutual feedback configuration.

4.29.2 Function Documentation

4.29.2.1 void SDTWaveguide_dsp (SDTWaveguide ∗ *x*, double *fwdIn*, double *revIn*)

Signal processing routine. Call this function at sample rate to compute the output samples. To read them, call the respective functions SDTWaveguide_getFwdOut() and SDTWaveguide_getRevOut().

Parameters

in	*fwdIn*	Input coming from the left side of the waveguide
in	*fwdIn*	Input coming from the right side of the waveguide

4.29.2.2 void SDTWaveguide_free (SDTWaveguide * *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.29.2.3 double SDTWaveguide_getFwdOut (SDTWaveguide * *x*)

Reads the output signal coming from the right side of the waveguide.

Returns

Output sample

4.29.2.4 double SDTWaveguide_getRevOut (SDTWaveguide * *x*)

Reads the output signal coming from the left side of the waveguide.

Returns

Output sample

4.29.2.5 SDTWaveguide * SDTWaveguide_new (int *maxDelay*)

Object constructor.

Parameters

in	*maxDelay*	Size of the two buffers, in samples

Returns

Pointer to the new instance

4.29.2.6 void SDTWaveguide_setDelay (SDTWaveguide * *x*, double *f*)

Sets the length of the waveguide, in samples.

Parameters

in	*f*	Delay time, in samples

4.29.2.7 void SDTWaveguide_setFwdDamping (SDTWaveguide * *x*, double *f*)

Sets the frequency damping on the right side.

Parameters

in	*f*	High frequency damping [0,1]

4.29.2.8 void SDTWaveguide_setFwdFeedback (SDTWaveguide * x, double f)

Sets the feedback on the right side. Determines how much energy gets fed back into the system after the wave reaches the right side of the waveguide. Consequently, this value also determines how much attenuated is the output on the same side.

Parameters

in	*f*	Feedback gain [0,1]

4.29.2.9 void SDTWaveguide_setRevDamping (SDTWaveguide * x, double f)

Sets the frequency damping on the left side.

Parameters

in	*f*	High frequency damping [0,1]

4.29.2.10 void SDTWaveguide_setRevFeedback (SDTWaveguide * x, double f)

Sets the feedback on the left side. Determines how much energy gets fed back into the system after the wave reaches the left side of the waveguide. Consequently, this value also determines how much attenuated is the output on the same side.

Parameters

in	*f*	Feedback gain [0,1]

4.30 SDTGases.h: Air turbulence and explosions

Modules

- Turbulence against solid objects
- Turbulence through hollow cavities
- Turbulence across thin objects
- Supersonic explosions

4.30.1 Detailed Description

Physical models to simulate wooshes, wind gusts and howls, helicopter rotors and so on. A gas flowing in a more or less constant direction usually doesn't make any sound by itself, its pressure variations being too slow to fall into the audible range. Nevertheless, objects obstructing the air flow are likely to cause turbulence at much higher frequencies, and therefore they do make sounds. Heavily inspired by the work of Andy Farnell in his book "Designing Sound", these models render chaotic turbulences through filtered random noise.

This module also includes the simulation of powerful explosions, as well as objects travelling at supersonic speed such as rifle bullets or cracking whip tails. All these phenomena create shock waves, namely a sudden peak in pressure followed by a negative expansion tail. Although being highly impulsive events, explosions also generate turbulence and other kinds of chaotic scattering which yield complex acoustic textures and have a direct effect on the resulting sound. The SDT explosion model uses a Friedlander waveform to render the impulsive part, and a Feedback Delay Network reverb to simulate scattering.

4.31 Turbulence against solid objects

Typedefs

- typedef struct SDTWindFlow SDTWindFlow

  Opaque data structure for a solid obstacle object.
- typedef struct SDTWindFlow SDTWindFlow

  Opaque data structure for a solid obstacle object.

Functions

- SDTWindFlow * SDTWindFlow_new ()

  Object constructor.
- void SDTWindFlow_free (SDTWindFlow *x)

  Object destructor.
- void SDTWindFlow_setFilters (SDTWindFlow *x)

  Update filter coefficients. Should be always called after setting the sampling rate with SDT_setSampleRate().
- void SDTWindFlow_setWindSpeed (SDTWindFlow *x, double f)

  Sets the wind speed.
- double SDTWindFlow_dsp (SDTWindFlow *x)

  Signal processing routine. Call this function at sample rate to synthesize a wind turbulence sound.

4.31.1 Detailed Description

One of the possible sources of turbulence is the impact on a large solid surface. In this case, turbulence is generated due to the impact of the air molecules on the surface and to their random change of direction caused by the irregularities of the surface itself. The resulting sound is modeled through a bandpass-filtered white noise generator. The center frequency and bandwidth of the filter are empirically set to fixed values, while the resulting output is modulated in amplitude according to the velocity of the air flow.

4.31.2 Function Documentation

4.31.2.1 double SDTWindFlow_dsp (SDTWindFlow * *x*)

Signal processing routine. Call this function at sample rate to synthesize a wind turbulence sound.

Returns

Computed audio sample

4.31.2.2 void SDTWindFlow_free (SDTWindFlow * *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.31.2.3 SDTWindFlow * SDTWindFlow_new ( )

Object constructor.

Returns

Pointer to the new instance

4.31.2.4 void SDTWindFlow_setFilters (SDTWindFlow * *x*)

Update filter coefficients. Should be always called after setting the sampling rate with SDT_setSampleRate().

Parameters

in	*x*	Pointer to a SDTWindFlow instance

4.31.2.5 void SDTWindFlow_setWindSpeed (SDTWindFlow * *x*, double *f*)

Sets the wind speed.

Parameters

in	*x*	Pointer to a SDTWindFlow instance
in	*f*	Wind speed [0,1]

4.32 Turbulence through hollow cavities

Typedefs

- typedef struct SDTWindCavity SDTWindCavity

  Opaque data structure for a hollow cavity object.
- typedef struct SDTWindCavity SDTWindCavity

  Opaque data structure for a hollow cavity object.

Functions

- SDTWindCavity * SDTWindCavity_new (int maxDelay)

  Object constructor.
- void SDTWindCavity_free (SDTWindCavity *x)

  Object destructor.
- void SDTWindCavity_setLength (SDTWindCavity *x, double f)

  Sets the lenght of the cavity.
- void SDTWindCavity_setDiameter (SDTWindCavity *x, double f)

  Sets the diameter of the cavity.
- void SDTWindCavity_setWindSpeed (SDTWindCavity *x, double f)

  Sets the wind speed.
- double SDTWindCavity_dsp (SDTWindCavity *x)

  Signal processing routine. Call this function at sample rate to synthesize wind through a cavity.

4.32.1 Detailed Description

Hollow objects such as pipes, valves, tunnels and doorways force the air moving inside them to oscillate at their resonant frequencies, which depend on the size and shape of the cavity itself. Different modes of resonance can be excited, in a more or less noticeable way, depending on the speed of the air flowing inside the tube. For each mode of resonance there is an optimal speed, which makes the air inside the tube resonate the most. As the speed increases, resonance gets weaker and weaker until it breaks up into the next harmonic. Sound waves trapped in a cylindrical cavity can be effectively simulated using a simple comb filter, namely a delay line with feedback. The different excitation of the various harmonics is modeled by a resonant bandpass filter with a high Q factor, therefore with a narrow band and a high resonance.

4.32.2 Function Documentation

4.32.2.1 double SDTWindCavity_dsp (SDTWindCavity * *x*)

Signal processing routine. Call this function at sample rate to synthesize wind through a cavity.

Returns

Computed audio sample

4.32.2.2 void SDTWindCavity_free (SDTWindCavity * *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.32.2.3 SDTWindCavity ∗ SDTWindCavity_new (int *maxDelay*)

Object constructor.

Parameters

in	*maxDelay*	Size of the comb filter buffer, in samples.

Returns

Pointer to the new instance

4.32.2.4 void SDTWindCavity_setDiameter (SDTWindCavity ∗ *x*, double *f*)

Sets the diameter of the cavity.

Parameters

in	*f*	Diameter of the cavity, in m

4.32.2.5 void SDTWindCavity_setLength (SDTWindCavity ∗ *x*, double *f*)

Sets the lenght of the cavity.

Parameters

in	*f*	Length of the cavity, in m

4.32.2.6 void SDTWindCavity_setWindSpeed (SDTWindCavity ∗ *x*, double *f*)

Sets the wind speed.

Parameters

in	*f*	Wind speed, [0,1]

4.33 Turbulence across thin objects

Typedefs

- typedef struct SDTWindKarman SDTWindKarman

  Opaque data structure for a thin obstacle object.
- typedef struct SDTWindKarman SDTWindKarman

  Opaque data structure for a thin obstacle object.

Functions

- SDTWindKarman * SDTWindKarman_new ()

  Object constructor.
- void SDTWindKarman_free (SDTWindKarman *x)

  Object destructor.
- void SDTWindKarman_setDiameter (SDTWindKarman *x, double f)

  Sets the diameter of the object.
- void SDTWindKarman_setWindSpeed (SDTWindKarman *x, double f)

  Sets the wind speed.
- double SDTWindKarman_dsp (SDTWindKarman *x)

  Signal processing routine. Call this function at sample rate to synthesize wind blowing against a thin object.

4.33.1 Detailed Description

An air flow hitting a thin object, such as a tree branch or a suspended wire, produces a singing or howling sound caused by a phenomenon known as Karman vortex street. This particular kind of turbulence is a repeating pattern of swirling vortices caused by the unsteady separation of flow of a fluid around the object. Karman vortex streets are modeled by white noise, passing through a bandpass filter with narrow bandwidth and high resonance.

4.33.2 Function Documentation

4.33.2.1 double SDTWindKarman_dsp (SDTWindKarman * *x*)

Signal processing routine. Call this function at sample rate to synthesize wind blowing against a thin object.

Returns

Computed audio sample

4.33.2.2 void SDTWindKarman_free (SDTWindKarman * *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.33.2.3 SDTWindKarman * SDTWindKarman_new ( )

Object constructor.

Returns

Pointer to the new instance

4.33.2.4 void SDTWindKarman_setDiameter (SDTWindKarman * x, double f)

Sets the diameter of the object.

Parameters

in	*f*	Diameter of the object, in m. Works best with very small values (<0.1)

4.33.2.5 void SDTWindKarman_setWindSpeed (SDTWindKarman * x, double f)

Sets the wind speed.

Parameters

in	*f*	Wind speed, [0,1]

4.34 Supersonic explosions

Typedefs

- typedef struct SDTExplosion SDTExplosion

  Opaque data structure for an explosion object.
- typedef struct SDTExplosion SDTExplosion

  Opaque data structure for an explosion object.

Functions

- SDTExplosion ∗ SDTExplosion_new (long maxScatter, long maxDelay)

  Object constructor.
- void SDTExplosion_free (SDTExplosion ∗x)

  Object destructor.
- void SDTExplosion_setBlastTime (SDTExplosion ∗x, double f)

  Sets the duration of the initial spike.
- void SDTExplosion_setScatterTime (SDTExplosion ∗x, double f)

  Sets the duration of the scattering.
- void SDTExplosion_setDispersion (SDTExplosion ∗x, double f)

  Sets the balance between initial spike and successive scattering.
- void SDTExplosion_setDistance (SDTExplosion ∗x, double f)

  Sets the distance of the listener from the explosion.
- void SDTExplosion_setWaveSpeed (SDTExplosion ∗x, double f)

  Sets the propagation velocity of the shockwave.
- void SDTExplosion_setWindSpeed (SDTExplosion ∗x, double f)

  Sets the propagation velocity of the blast wind.
- void SDTExplosion_update (SDTExplosion ∗x)

  Updates the internal state of the object. Please call this function after having reset one or more synthesis parameters.
- void SDTExplosion_dsp (SDTExplosion ∗x, double ∗outs)

  Signal processing routine. Call this function at sample rate to synthesize an explosion sound.

4.34.1 Detailed Description

Powerful explosions, as well as objects travelling at supersonic speed such as rifle bullets or cracking whip tails.

4.34.2 Function Documentation

4.34.2.1 void SDTExplosion_dsp (**SDTExplosion** ∗ *x*, double ∗ *outs*)

Signal processing routine. Call this function at sample rate to synthesize an explosion sound.

Returns

Computed audio sample

4.34.2.2 void SDTExplosion_free (**SDTExplosion** ∗ *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.34.2.3 SDTExplosion ∗ SDTExplosion_new (long *maxScatter*, long *maxDelay*)

Object constructor.

Parameters

in	*maxScatter*	Maximum scattering time, in samples)
in	*maxDelay*	Maximum delay between explosion and sound, in samples

Returns

Pointer to the new instance

4.34.2.4 void SDTExplosion_setBlastTime (SDTExplosion ∗ *x*, double *f*)

Sets the duration of the initial spike.

Parameters

in	*f*	Blast time, in s

4.34.2.5 void SDTExplosion_setDispersion (SDTExplosion ∗ *x*, double *f*)

Sets the balance between initial spike and successive scattering.

Parameters

in	*f*	Amount of scattering, [0,1]

4.34.2.6 void SDTExplosion_setDistance (SDTExplosion ∗ *x*, double *f*)

Sets the distance of the listener from the explosion.

Parameters

in	*f*	Distance between explosion and listener, in m

4.34.2.7 void SDTExplosion_setScatterTime (SDTExplosion ∗ *x*, double *f*)

Sets the duration of the scattering.

Parameters

in	*f*	Scattering time, in s

4.34.2.8 void SDTExplosion_setWaveSpeed (SDTExplosion ∗ *x*, double *f*)

Sets the propagation velocity of the shockwave.

Parameters

in	*f*	Propagation velocity of the shockwave, in m/s

4.34.2.9 void SDTExplosion_setWindSpeed (SDTExplosion * x, double f)

Sets the propagation velocity of the blast wind.

Parameters

in	*f*	Propagation velocity of the blast wind, in m/s

4.35 SDTInteractors.h: interactions between solids

Modules

- Interactor interface
- Impact
- Friction

4.35.1 Detailed Description

These models simulate basic mechanical interactions that can occur between two resonators: impacts and friction.

4.36 Interactor interface

Typedefs

- typedef struct SDTInteractor SDTInteractor

    Opaque data structure representing the interactor interface.
- typedef struct SDTInteractor SDTInteractor

    Opaque data structure representing the interactor interface.

Functions

- void SDTInteractor_setFirstResonator (SDTInteractor ∗x, SDTResonator ∗p)

    Sets the pointer to the first interacting resonator.
- void SDTInteractor_setSecondResonator (SDTInteractor ∗x, SDTResonator ∗p)

    Sets the pointer to the second interacting resonator.
- void SDTInteractor_setFirstPoint (SDTInteractor ∗x, long l)

    Sets the contact point index for the first resonator.
- void SDTInteractor_setSecondPoint (SDTInteractor ∗x, long l)

    Sets the contact point index for the second resonator.
- double SDTInteractor_computeForce (SDTInteractor ∗x)

    Computes a force to apply to the contact points, based on the resonators' state at the chosen pickups.
- void SDTInteractor_dsp (SDTInteractor ∗x, double f0, double v0, double s0, double f1, double v1, double s1, double ∗outs)

    Signal processing routine. Convenience method to compute the interaction force, apply it to the resonators and update their state. This method already calls the DSP routines of the two resonators, so be sure not to call them if you use this method.

4.36.1 Detailed Description

This abstract object acts as a generic interface implemented by all interactors. It contains two pointers to the interacting objects, information on the chosen contact points, and an algorithm that, after reading the state of the objects (displacement and velocity) at the specified contact points, accordingly computes a force to apply to those contact points. The generic interactor should never be directly instantiated, instead it should be obtained through the specific SDTImpact and SDTFriction constructors.

4.36.2 Function Documentation

4.36.2.1 void SDTInteractor_dsp (**SDTInteractor** ∗ *x*, double *f0*, double *v0*, double *s0*, double *f1*, double *v1*, double *s1*, double ∗ *outs*)

Signal processing routine. Convenience method to compute the interaction force, apply it to the resonators and update their state. This method already calls the DSP routines of the two resonators, so be sure not to call them if you use this method.

Parameters

in	*f0*	Applied force to the first resonator
in	*v0*	Applied velocity to the first resonator (resets position to 0, or to make contact with second object if present)

in	*s0*	Fragment size of the first resonator
in	*f1*	Applied force to the second resonator
in	*v1*	Applied velocity to the second resonator (resets position to 0, or to make contact with first object if present)
in	*s1*	Fragment size of the second resonator
out	*outs*	Displacement of the resonators at their pickup points

4.36.2.2 void SDTInteractor_setFirstPoint (SDTInteractor * *x*, long *l*)

Sets the contact point index for the first resonator.

Parameters

in	*Number*	of the first resonator pickup chosen for interaction

4.36.2.3 void SDTInteractor_setFirstResonator (SDTInteractor * *x*, SDTResonator * *p*)

Sets the pointer to the first interacting resonator.

Parameters

in	*p*	Pointer to a SDTResonator instance

4.36.2.4 void SDTInteractor_setSecondPoint (SDTInteractor * *x*, long *l*)

Sets the contact point index for the second resonator.

Parameters

in	*Number*	of the second resonator pickup chosen for interaction

4.36.2.5 void SDTInteractor_setSecondResonator (SDTInteractor * *x*, SDTResonator * *p*)

Sets the pointer to the second interacting resonator.

Parameters

in	*p*	Pointer to a SDTResonator instance

4.37 Impact

Typedefs

- typedef struct SDTImpact SDTImpact

  Opaque data structure representing the internal state of an impact interactor.
- typedef struct SDTImpact SDTImpact

  Opaque data structure representing the internal state of an impact interactor.

Functions

- SDTInteractor ∗ SDTImpact_new ()

  Object constructor.
- void SDTImpact_free (SDTInteractor ∗x)

  Object destructor. param[in] Pointer to a SDTInteractor instance, configured for the impact case.
- void SDTImpact_setStiffness (SDTInteractor ∗x, double f)

  Sets the impact stiffness.
- void SDTImpact_setDissipation (SDTInteractor ∗x, double f)

  Sets the dissipation coefficient.
- void SDTImpact_setShape (SDTInteractor ∗x, double f)

  Sets the shape factor.

4.37.1 Detailed Description

Simulates a non-linear impact, computing impact force from the total compression, namely the relative displacement between the two contact points. The algorithm is based on the Hunt-Crossley impact model, with the resulting force being the sum of an elastic component and a dissipative term.

The elastic component is parameterized by the force stiffness (or elasticity) and a non-linear exponent which models the local geometry around the contact area. The linear dissipative component is parameterized by a dissipation (damping) weight.

4.37.2 Function Documentation

4.37.2.1 SDTInteractor ∗ SDTImpact_new ( )

Object constructor.

Returns

Pointer to a SDTInteractor instance, configured for the impact case

4.37.2.2 void SDTImpact_setDissipation (SDTInteractor ∗ *x*, double *f*)

Sets the dissipation coefficient.

Parameters

in	*f*	Dissipation coefficient, positive scalar

4.37.2.3 void SDTImpact_setShape (SDTInteractor ∗ *x*, double *f*)

Sets the shape factor.

Parameters

in	*f*	Shape factor. Must be >1, with 1.5 = spherical shape. Optimal range [1,4]

4.37.2.4 void SDTImpact_setStiffness (SDTInteractor $*$ *x*, double *f*)

Sets the impact stiffness.

Parameters

in	*f*	Impact stiffness ($>>1$)

4.38 Friction

Typedefs

- typedef struct SDTFriction SDTFriction

  Opaque data structure representing the internal state of a friction interactor.
- typedef struct SDTFriction SDTFriction

  Opaque data structure representing the internal state of a friction interactor.

Functions

- SDTInteractor ∗ SDTFriction_new ()

  Object constructor.
- void SDTFriction_free (SDTInteractor ∗x)

  Object destructor. param[in] Pointer to a SDTInteractor instance, configured for the friction case.
- void SDTFriction_setNormalForce (SDTInteractor ∗x, double f)

  Sets the perpendicular force (pressure) applied to the two sliding resonators.
- void SDTFriction_setStribeckVelocity (SDTInteractor ∗x, double f)

  Sets the Stribeck velocity.
- void SDTFriction_setStaticCoefficient (SDTInteractor ∗x, double f)

  Sets the static friction coefficient.
- void SDTFriction_setDynamicCoefficient (SDTInteractor ∗x, double f)

  Sets the dynamic friction coefficient.
- void SDTFriction_setBreakAway (SDTInteractor ∗x, double f)

  Sets the break away coefficient.
- void SDTFriction_setStiffness (SDTInteractor ∗x, double f)

  Sets the contact stiffness.
- void SDTFriction_setDissipation (SDTInteractor ∗x, double f)

  Sets the dissipation coefficient.
- void SDTFriction_setViscosity (SDTInteractor ∗x, double f)

  Sets the contact viscosity.
- void SDTFriction_setNoisiness (SDTInteractor ∗x, double f)

  Sets the surface roughness.

4.38.1 Detailed Description

Elasto-plastic friction model, computing friction force from the relative velocity between the two contact points. The resulting force is the sum of four components: an elastic term, an internal dissipation term, a viscosity term, and finally a random term representing noise related to the surface roughness.

More subtle phenomena, such as pre-sliding behavior (gradual increase of the friction force for very small displacements), are simulated by the "plastic" part of the algorithm and parametrized by several other values, such as static/dynamic friction coefficients, break-away and Stribeck velocity, and so on.

These phenomena are mostly related to the transients and are worth being modeled despite the added complexity of the algorithm because of their importance for a realistic simulation of friction sounds.

4.38.2 Function Documentation

4.38.2.1 SDTInteractor ∗ SDTFriction_new ( )

Object constructor.

Returns

Pointer to a SDTInteractor instance, configured for the friction case

4.38.2.2 void SDTFriction_setBreakAway (SDTInteractor * x, double f)

Sets the break away coefficient.

Parameters

in	*f*	Break away coefficient, positive scalar

4.38.2.3 void SDTFriction_setDissipation (SDTInteractor * x, double f)

Sets the dissipation coefficient.

Parameters

in	*f*	Dissipation coefficient, positive scalar

4.38.2.4 void SDTFriction_setDynamicCoefficient (SDTInteractor * x, double f)

Sets the dynamic friction coefficient.

Parameters

in	*f*	Dynamic friction coefficient [0,1]. Should be less than the static friction coefficient

4.38.2.5 void SDTFriction_setNoisiness (SDTInteractor * x, double f)

Sets the surface roughness.

Parameters

in	*f*	Surface roughness, positive scalar

4.38.2.6 void SDTFriction_setNormalForce (SDTInteractor * x, double f)

Sets the perpendicular force (pressure) applied to the two sliding resonators.

Parameters

in	*f*	Normal force, in N

4.38.2.7 void SDTFriction_setStaticCoefficient (SDTInteractor * x, double f)

Sets the static friction coefficient.

Parameters

in	*f*	Static friction coefficient [0,1]

4.38.2.8 void SDTFriction_setStiffness (SDTInteractor * *x*, double *f*)

Sets the contact stiffness.

Parameters

in	*f*	Contact stiffness, positive scalar

4.38.2.9 void SDTFriction_setStribeckVelocity (SDTInteractor * *x*, double *f*)

Sets the Stribeck velocity.

Parameters

in	*f*	Stribeck velocity, in m/s

4.38.2.10 void SDTFriction_setViscosity (SDTInteractor * *x*, double *f*)

Sets the contact viscosity.

Parameters

in	*f*	Contact viscosity, positive scalar

4.39 SDTLiquids.h: Liquid sounds

Modules

- Bubbles
- Fluid flow

4.39.1 Detailed Description

Models and algorithms to simulate sounds generated by liquids: burbling, splashing, dripping, filling, gushing etc.

4.40 Bubbles

Typedefs

- typedef struct SDTBubble SDTBubble

  Opaque data structure representing a bubble object.
- typedef struct SDTBubble SDTBubble

  Opaque data structure representing a bubble object.

Functions

- SDTBubble ∗ SDTBubble_new ()

  Object constructor.
- void SDTBubble_free (SDTBubble ∗x)

  Object destructor.
- void SDTBubble_setRadius (SDTBubble ∗x, double f)

  Sets the bubble radius.
- void SDTBubble_setDepth (SDTBubble ∗x, double f)

  Sets the bubble depth.
- void SDTBubble_setRiseFactor (SDTBubble ∗x, double f)

  Sets the amount of blooping.
- void SDTBubble_update (SDTBubble ∗x)

  Triggers a new bubble.
- void SDTBubble_normAmp (SDTBubble ∗x)

  Sets bubble amplitude to the maximum instead of computing it from radius and depth.
- double SDTBubble_dsp (SDTBubble ∗x)

  Signal processing routine. Call this function at sample rate to obtain a bubble sound.

4.40.1 Detailed Description

The main responsible for acoustic emission in water and other liquids, rather than the liquid mass on its own, is the gas trapped inside emerging as a population of bubbles. From a physical point of view, a spherical bubble acts as an exponentially decaying sinusoidal oscillator. Frequency, decay time and relative amplitude of each bubble can be derived from its radius and depth.

When the bubble is formed close to the surface and therefore the effective mass around the liquid is reduced, the oscillating frequency rises and a characteristic "blooping" sound is generated. The amount of blooping can be set as an independent parameter in the model.

4.40.2 Function Documentation

4.40.2.1 double SDTBubble_dsp (**SDTBubble** ∗ *x*)

Signal processing routine. Call this function at sample rate to obtain a bubble sound.

Returns

Output sample

4.40.2.2 void SDTBubble_free (**SDTBubble** ∗ *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.40.2.3 SDTBubble * SDTBubble_new ( )

Object constructor.

Returns

Pointer to the new instance

4.40.2.4 void SDTBubble_setDepth (SDTBubble * *x*, double *f*)

Sets the bubble depth.

Parameters

in	*f*	Bubble depth [0, 1]. 0 means very deep, 1 means touching the surface.

4.40.2.5 void SDTBubble_setRadius (SDTBubble * *x*, double *f*)

Sets the bubble radius.

Parameters

in	*f*	Bubble radius, in m [0.00015, 0.150]

4.40.2.6 void SDTBubble_setRiseFactor (SDTBubble * *x*, double *f*)

Sets the amount of blooping.

Parameters

in	*f*	Rise factor, positive scalar. Typical value for bubbles in water = 0.1

4.41 Fluid flow

Typedefs

- typedef struct SDTFluidFlow SDTFluidFlow

  Opaque data structure representing a fluid flow object.
- typedef struct SDTFluidFlow SDTFluidFlow

  Opaque data structure representing a fluid flow object.

Functions

- SDTFluidFlow ∗ SDTFluidFlow_new (int nBubbles)

  Object constructor.
- void SDTFluidFlow_free (SDTFluidFlow ∗x)

  Object destructor.
- void SDTFluidFlow_setMinRadius (SDTFluidFlow ∗x, double f)

  Sets the minimum radius for the bubble population.
- void SDTFluidFlow_setMaxRadius (SDTFluidFlow ∗x, double f)

  Sets the maximum radius for the bubble population.
- void SDTFluidFlow_setExpRadius (SDTFluidFlow ∗x, double f)

  Sets the gamma factor for the radius assignment.
- void SDTFluidFlow_setMinDepth (SDTFluidFlow ∗x, double f)

  Sets the minimum depth value for the bubble population.
- void SDTFluidFlow_setMaxDepth (SDTFluidFlow ∗x, double f)

  Sets the maximum depth value for the bubble population.
- void SDTFluidFlow_setExpDepth (SDTFluidFlow ∗x, double f)

  Sets the gamma factor for the depth assignment.
- void SDTFluidFlow_setRiseFactor (SDTFluidFlow ∗x, double f)

  Sets the amount of blooping for the bubble population.
- void SDTFluidFlow_setRiseCutoff (SDTFluidFlow ∗x, double f)

  Bubbles deeper than this threshold do not rise in frequency.
- void SDTFluidFlow_setAvgRate (SDTFluidFlow ∗x, double f)

  Sets the amount of generated bubbles per second.
- double SDTFluidFlow_dsp (SDTFluidFlow ∗x)

  Signal processing routine. Call this function at sample rate to obtain a liquid sound.

4.41.1 Detailed Description

Rich and complex liquid sound simulations can be generated through a stochastic population of bubbles, modeled by a sinusoidal oscillator bank with each voice modulated in amplitude and frequency according to desired probability distributions. A simple stochastic algorithm controls the behavior of the bubble population: Bubble generation rate follows a Bernoulli process, while radius and depth for each new bubble are chosen at random. To limit the presence of sudden peaks and glitches, voices are updated based on their age: The bubble with the lowest amplitude gets "killed" in favor of the new one.

4.41.2 Function Documentation

4.41.2.1 double SDTFluidFlow_dsp (SDTFluidFlow ∗ *x*)

Signal processing routine. Call this function at sample rate to obtain a liquid sound.

Returns

Output sample

4.41.2.2 void SDTFluidFlow_free (SDTFluidFlow * x)

Object destructor.

Parameters

in	*x*	Poiter to the instance to destroy

4.41.2.3 SDTFluidFlow * SDTFluidFlow_new (int *nBubbles*)

Object constructor.

Parameters

in	*Number*	of voices in the oscillator bank

Returns

Pointer to the new instance

4.41.2.4 void SDTFluidFlow_setAvgRate (SDTFluidFlow * x, double f)

Sets the amount of generated bubbles per second.

Parameters

in	*f*	Average number of bubbles per second

4.41.2.5 void SDTFluidFlow_setExpDepth (SDTFluidFlow * x, double f)

Sets the gamma factor for the depth assignment.

Parameters

in	*f*	Depth gamma factor. O to 1 = shallower bubbles, $>$ 1 = deeper bubbles

4.41.2.6 void SDTFluidFlow_setExpRadius (SDTFluidFlow * x, double f)

Sets the gamma factor for the radius assignment.

Parameters

in	*f*	Radius gamma factor. O to 1 = bigger bubbles, $>$ 1 = smaller bubbles

4.41.2.7 void SDTFluidFlow_setMaxDepth (SDTFluidFlow * x, double f)

Sets the maximum depth value for the bubble population.

Parameters

in	*f*	Maximum depth value of the generated bubbles, [0, 1]

4.41.2.8 void SDTFluidFlow_setMaxRadius (SDTFluidFlow * *x*, double *f*)

Sets the maximum radius for the bubble population.

Parameters

in	*f*	Maximum radius of the generated bubbles, in m [0.00015, 0.150]

4.41.2.9 void SDTFluidFlow_setMinDepth (SDTFluidFlow * *x*, double *f*)

Sets the minimum depth value for the bubble population.

Parameters

in	*f*	Minimum depth value of the generated bubbles, [0, 1]

4.41.2.10 void SDTFluidFlow_setMinRadius (SDTFluidFlow * *x*, double *f*)

Sets the minimum radius for the bubble population.

Parameters

in	*f*	Minimum radius of the generated bubbles, in m [0.00015, 0.150]

4.41.2.11 void SDTFluidFlow_setRiseCutoff (SDTFluidFlow * *x*, double *f*)

Bubbles deeper than this threshold do not rise in frequency.

Parameters

in	*f*	Rise cutoff, [0, 1]

4.41.2.12 void SDTFluidFlow_setRiseFactor (SDTFluidFlow * *x*, double *f*)

Sets the amount of blooping for the bubble population.

Parameters

in	*f*	Rise factor. Typical value for water = 0.1

4.42 SDTMotor.h: Combustion engines

Typedefs

- typedef struct SDTMotor SDTMotor

  Opaque data structure representing a combustion engine object.
- typedef struct SDTMotor SDTMotor

  Opaque data structure representing a combustion engine object.

Functions

- SDTMotor ∗ SDTMotor_new (long maxDelay)

  Object constructor.
- void SDTMotor_free (SDTMotor ∗x)

  Object destructor.
- void SDTMotor_setFilters (SDTMotor ∗x, double damp, double dc)

  Update filter coefficients. Should be always called after setting the sampling rate with SDT_setSampleRate().
- void SDTMotor_setRpm (SDTMotor ∗x, double f)

  Sets the Revolutions Per Minute (RPM) of the engine.
- void SDTMotor_setThrottle (SDTMotor ∗x, double f)

  Sets the throttle load.
- void SDTMotor_setFourStroke (SDTMotor ∗x)

  Simulates the operation cycle of a four-stroke engine.
- void SDTMotor_setTwoStroke (SDTMotor ∗x)

  Simulates the operation cycle of a two-stroke engine.
- void SDTMotor_setNCylinders (SDTMotor ∗x, int i)

  Sets the number of cylinders in the engine block.
- void SDTMotor_setCylinderSize (SDTMotor ∗x, double f)

  Sets the size of each single cylinder. The total volume of the engine is this value multiplied by the number of cylinders.
- void SDTMotor_setCompressionRatio (SDTMotor ∗x, double f)

  Sets the compression ratio of the engine. The compression ratio is computed dividing the cylinder volume at maximum expansion (piston down) by its volume at maximum compression (piston up).
- void SDTMotor_setSparkTime (SDTMotor ∗x, double f)

  Sets the width of the ignition pulse, compared to a full operation cycle.
- void SDTMotor_setAsymmetry (SDTMotor ∗x, double f)

  Sets the amount of irregularity in the operation cycle.
- void SDTMotor_setBackfire (SDTMotor ∗x, double f)

  Sets the amount of backfiring when the engine revs down.
- void SDTMotor_setIntakeSize (SDTMotor ∗x, double f)

  Sets the average length of the intake pipes.
- void SDTMotor_setExtractorSize (SDTMotor ∗x, double f)

  Sets the average length of the extractor pipes.
- void SDTMotor_setExhaustSize (SDTMotor ∗x, double f)

  Sets the length of the main exhaust pipe.
- void SDTMotor_setExpansion (SDTMotor ∗x, double f)

  Sets the amount of expansion of the main exhaust pipe. This is a feature commonly found in two-stroke engines, to avoid the passage of fresh fuel mixture into the exhaust system.
- void SDTMotor_setMufflerSize (SDTMotor ∗x, double f)

  Sets the average length of the muffler chambers.
- void SDTMotor_setMufflerFeedback (SDTMotor ∗x, double f)

  Sets the amount of energy dissipated by the muffler chambers.

- void SDTMotor_setOutletSize (SDTMotor *x, double f)

  Sets the length of the exhaust outlet.
- void SDTMotor_dsp (SDTMotor *x, double *outs)

  Signal processing routine. Call this function at sample rate to synthesize the engine sound. The output is written in an array of three doubles. The first value represents the sound picked up at the intakes, from the front of the vehicle; the second represents the engine vibrations, mostly heard inside the cabin; the third and last output represents the sound coming from the exhaust outlet, towards the rear of the vehicle.

4.42.1 Detailed Description

From a mechanical point of view, an internal combustion engine converts chemical energy into kinetic energy by means of a series of controlled explosions. From an acoustical point of view, the previously described setup is basically a set of resonating pipes, excited by the explosions happening in the combustion chambers. Resonances happening inside intake pipes, cylinders, exhaust collectors, exhaust pipe, exhaust muffler and final outlet are simulated by means of digital waveguides, whose inputs, lengths and feedback gains are controlled by a physical model of the engine operation cycle representing the behavior of the engine block. Four mechanical components are simulated: Piston motion, fuel ignition, intake valves operation and exhaust valves operation. The model provides also a simulation of exhaust backfiring, a phenomenon which occurs especially in sports or muscle cars, where the very rich fuel mixture sometimes doesn't burn completely in the cylinders and self ignites later in the hotter parts of the exhaust system.

4.42.2 Function Documentation

4.42.2.1 void SDTMotor_dsp (SDTMotor * *x*, double * *outs*)

Signal processing routine. Call this function at sample rate to synthesize the engine sound. The output is written in an array of three doubles. The first value represents the sound picked up at the intakes, from the front of the vehicle; the second represents the engine vibrations, mostly heard inside the cabin; the third and last output represents the sound coming from the exhaust outlet, towards the rear of the vehicle.

Parameters

out	*outs*	Pointer to an array of three doubles, destination of the output

4.42.2.2 void SDTMotor_free (SDTMotor * *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.42.2.3 SDTMotor * SDTMotor_new (long *maxDelay*)

Object constructor.

Returns

Pointer to the new instance

4.42.2.4 void SDTMotor_setAsymmetry (SDTMotor * *x*, double *f*)

Sets the amount of irregularity in the operation cycle.

Parameters

in	*f*	Cycle asymmetry [0,1]

4.42.2.5 void SDTMotor_setBackfire (SDTMotor ∗ *x*, double *f*)

Sets the amount of backfiring when the engine revs down.

Parameters

in	*f*	Chance of backfiring [0,1]

4.42.2.6 void SDTMotor_setCompressionRatio (SDTMotor ∗ *x*, double *f*)

Sets the compression ratio of the engine. The compression ratio is computed dividing the cylinder volume at maximum expansion (piston down) by its volume at maximum compression (piston up).

Parameters

in	*f*	Compression ratio

4.42.2.7 void SDTMotor_setCylinderSize (SDTMotor ∗ *x*, double *f*)

Sets the size of each single cylinder. The total volume of the engine is this value multiplied by the number of cylinders.

Parameters

in	*f*	Cylinder volume, in cc

4.42.2.8 void SDTMotor_setExhaustSize (SDTMotor ∗ *x*, double *f*)

Sets the length of the main exhaust pipe.

Parameters

in	*f*	Exhaust size, in m

4.42.2.9 void SDTMotor_setExpansion (SDTMotor ∗ *x*, double *f*)

Sets the amount of expansion of the main exhaust pipe. This is a feature commonly found in two-stroke engines, to avoid the passage of fresh fuel mixture into the exhaust system.

Parameters

in	*f*	Exhaust expansion [0,1]

4.42.2.10 void SDTMotor_setExtractorSize (SDTMotor ∗ *x*, double *f*)

Sets the average length of the extractor pipes.

Parameters

in	*f*	Extractor size, in m

4.42.2.11 void SDTMotor_setFilters (SDTMotor * *x*, double *damp*, double *dc*)

Update filter coefficients. Should be always called after setting the sampling rate with SDT_setSampleRate().

Parameters

in	*x*	Pointer to a SDTMotor instance

4.42.2.12 void SDTMotor_setIntakeSize (SDTMotor * *x*, double *f*)

Sets the average length of the intake pipes.

Parameters

in	*f*	Intake size, in m

4.42.2.13 void SDTMotor_setMufflerFeedback (SDTMotor * *x*, double *f*)

Sets the amount of energy dissipated by the muffler chambers.

Parameters

in	*f*	Muffler feedback [0,1]

4.42.2.14 void SDTMotor_setMufflerSize (SDTMotor * *x*, double *f*)

Sets the average length of the muffler chambers.

Parameters

in	*f*	Muffler size, in m

4.42.2.15 void SDTMotor_setNCylinders (SDTMotor * *x*, int *i*)

Sets the number of cylinders in the engine block.

Parameters

in	*i*	Number of cylinders [1,12]

4.42.2.16 void SDTMotor_setOutletSize (SDTMotor * *x*, double *f*)

Sets the length of the exhaust outlet.

Parameters

in	*f*	Outlet size, in m

4.42.2.17 void SDTMotor_setRpm (SDTMotor * x, double f)

Sets the Revolutions Per Minute (RPM) of the engine.

Parameters

in	*f*	RPM value

4.42.2.18 void SDTMotor_setSparkTime (SDTMotor * *x*, double *f*)

Sets the width of the ignition pulse, compared to a full operation cycle.

Parameters

in	*f*	Ignition time [0,1]

4.42.2.19 void SDTMotor_setThrottle (SDTMotor * *x*, double *f*)

Sets the throttle load.

Parameters

in	*f*	Throttle load [0,1]

4.43 SDTOscillators.h: Oscillators

Typedefs

- typedef struct SDTPinkNoise SDTPinkNoise

  Opaque data structure for a pink noise generator.
- typedef struct SDTPinkNoise SDTPinkNoise

  Opaque data structure for a pink noise generator.

Functions

- SDTPinkNoise ∗ SDTPinkNoise_new (int nOctaves)

  Object constructor.
- void SDTPinkNoise_free (SDTPinkNoise ∗x)

  Object destructor.
- double SDTPinkNoise_dsp (SDTPinkNoise ∗x)

  Signal processing routine. Call this function at sample rate to generate pink noise.
- double SDT_whiteNoise ()

  Signal processing routine. Call this function at sample rate to generate white noise.

4.43.1 Detailed Description

Simple, commonly used sound generators.

4.43.2 Function Documentation

4.43.2.1 void SDTPinkNoise_free (SDTPinkNoise ∗ *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.43.2.2 SDTPinkNoise ∗ SDTPinkNoise_new (int *nOctaves*)

Object constructor.

Parameters

in	*nOctaves*	N. of octave bands for the pink noise generator.

Returns

Pointer to the new instance

4.44 SDTResonators.h: Solid resonators

Typedefs

- typedef struct SDTResonator SDTResonator

    Opaque data structure representing a solid resonator object.

- typedef struct SDTResonator SDTResonator

    Opaque data structure representing a solid resonator object.

Functions

- SDTResonator ∗ SDTResonator_new (unsigned int nModes, unsigned int nPickups)

    Object constructor.

- void SDTResonator_free (SDTResonator ∗x)

    Object destructor.

- double SDTResonator_getPosition (SDTResonator ∗x, unsigned int pickup)

    Gets the displacement of the object at a given pickup point.

- double SDTResonator_getVelocity (SDTResonator ∗x, unsigned int pickup)

    Gets the velocity of the object at a given pickup point.

- int SDTResonator_getNPickups (SDTResonator ∗x)

    Gets the number of pickup points.

- void SDTResonator_setPosition (SDTResonator ∗x, unsigned int pickup, double f)

    Sets a modal displacement at a given pickup point.

- void SDTResonator_setVelocity (SDTResonator ∗x, unsigned int pickup, double f)

    Sets a modal velocity at a given pickup point.

- void SDTResonator_setFrequency (SDTResonator ∗x, unsigned int mode, double f)

    Sets the resonant frequency for a given mode.

- void SDTResonator_setDecay (SDTResonator ∗x, unsigned int mode, double f)

    Sets the decay for a given mode.

- void SDTResonator_setWeight (SDTResonator ∗x, unsigned int mode, double f)

    Sets the weight for a given mode.

- void SDTResonator_setGain (SDTResonator ∗x, unsigned int pickup, unsigned int mode, double f)

    Sets the pickup gain for a given mode and pickup.

- void SDTResonator_setFragmentSize (SDTResonator ∗x, double f)

    Reduces the object into a smaller fragment. This paramenter influences various aspects of the object: Smaller fragments resonate louder and at higher frequencies, but with shorter decay times.

- void SDTResonator_setActiveModes (SDTResonator ∗x, unsigned int i)

    Sets the number of active (actually computed) modes.

- void SDTResonator_applyForce (SDTResonator ∗x, unsigned int pickup, double f)

    Applies a force to the resonator at a given pickup point. The force is distributed across the modes according to their normalized pickup gains (modal gain/sum of all gains). If the function is called multiple times in a single DSP cycle, the applied force gets accumulated.

- double SDTResonator_computeEnergy (SDTResonator ∗x, unsigned int pickup, double f)

    Computes the total energy of the object, after applying all acting forces.

- void SDTResonator_dsp (SDTResonator ∗x)

    Signal processing routine. Call this function at sample rate to update the internal state of the resonator. DO NOT call this function if you plan to use any of the interactor DSP methods instead! See the SDTInteractors.h module documentation for further information.

4.44.1 Detailed Description

Physical model of a solid resonator, represented as a set of parallel mass-spring-damper mechanical oscillators. Each oscillator corresponds to a normal mode of resonance of the object, with the oscillation period, the mass and the damping coefficient of each oscillator corresponding respectively to the resonance frequency, the magnitude and the decay time of each mode. Resonant modes can be mixed and weighted with different gains, to simulate different pickup points on the resonating object. A single mode with a resonant frequency of 0 Hz, infinite decay time and unity pickup gain behaves like an inertial point mass. The model uses the impulse invariant method as discretization scheme.

4.44.2 Function Documentation

4.44.2.1 void SDTResonator_applyForce (SDTResonator * *x*, unsigned int *pickup*, double *f*)

Applies a force to the resonator at a given pickup point. The force is distributed across the modes according to their normalized pickup gains (modal gain/sum of all gains). If the function is called multiple times in a single DSP cycle, the applied force gets accumulated.

Parameters

in		

4.44.2.2 double SDTResonator_computeEnergy (SDTResonator * *x*, unsigned int *pickup*, double *f*)

Computes the total energy of the object, after applying all acting forces.

Parameters

in	*pickup*	Pickup point
in	*f*	External force applied at the pickup point

Returns

Sum of kinetic and potential energy, in J

4.44.2.3 void SDTResonator_free (SDTResonator * *x*)

Object destructor.

Parameters

in	*x*	Pointer to the instance to destroy

4.44.2.4 int SDTResonator_getNPickups (SDTResonator * *x*)

Gets the number of pickup points.

Returns

Number of pickup points

4.44.2.5 double SDTResonator_getPosition (SDTResonator * *x*, unsigned int *pickup*)

Gets the displacement of the object at a given pickup point.

Parameters

in	*pickup*	Pickup point

Returns

Object displacement, in m

4.44.2.6 double SDTResonator_getVelocity (SDTResonator ∗ *x*, unsigned int *pickup*)

Gets the velocity of the object at a given pickup point.

Parameters

in	*pickup*	Pickup point

Returns

Object velocity, in m/s

4.44.2.7 SDTResonator ∗ SDTResonator_new (unsigned int *nModes*, unsigned int *nPickups*)

Object constructor.

Parameters

in	*nModes*	Number of resonant modes
in	*nPickups*	Number of pickup points

Returns

Pointer to the new instance

4.44.2.8 void SDTResonator_setActiveModes (SDTResonator ∗ *x*, unsigned int *i*)

Sets the number of active (actually computed) modes.

Parameters

in		

4.44.2.9 void SDTResonator_setDecay (SDTResonator ∗ *x*, unsigned int *mode*, double *f*)

Sets the decay for a given mode.

Parameters

in	*mode*	Mode number
in		

4.44.2.10 void SDTResonator_setFragmentSize (SDTResonator ∗ *x*, double *f*)

Reduces the object into a smaller fragment. This paramenter influences various aspects of the object: Smaller fragments resonate louder and at higher frequencies, but with shorter decay times.

Parameters

in		

4.44.2.11 void SDTResonator_setFrequency (SDTResonator * *x*, unsigned int *mode*, double *f*)

Sets the resonant frequency for a given mode.

Parameters

in	*mode*	Mode number
in		

4.44.2.12 void SDTResonator_setGain (SDTResonator * *x*, unsigned int *pickup*, unsigned int *mode*, double *f*)

Sets the pickup gain for a given mode and pickup.

Parameters

in	*pickup*	Pickup number
in	*mode*	Mode number
in		

4.44.2.13 void SDTResonator_setPosition (SDTResonator * *x*, unsigned int *pickup*, double *f*)

Sets a modal displacement at a given pickup point.

Parameters

in	*pickup*	Pickup point
in		

4.44.2.14 void SDTResonator_setVelocity (SDTResonator * *x*, unsigned int *pickup*, double *f*)

Sets a modal velocity at a given pickup point.

Parameters

in	*pickup*	Pickup point
in		

4.44.2.15 void SDTResonator_setWeight (SDTResonator * *x*, unsigned int *mode*, double *f*)

Sets the weight for a given mode.

Parameters

in	*mode*	Mode number
in		

4.45 SDTSolids.h: Registering/notifying resonators and interactors

Macros

- #define **SDT_MAX_MODES** 16
- #define **SDT_MAX_PICKUPS** 16
- #define **SDT_MAX_MODES** 16
- #define **SDT_MAX_PICKUPS** 16

Functions

- int SDT_registerResonator (SDTResonator ∗x, char ∗key)

  Registers a resonator into the resonators list with a unique ID. If an interactor with the same ID is present, the resonator is bound to the interactor.
- int SDT_unregisterResonator (char ∗key)

  Unregisters a resonator from the resonator list. If a resonator with the given ID is present, it is unregistered from the list. If also an interactor with the same ID is present, the object is released by the interactor as well.
- int SDT_registerInteractor (SDTInteractor ∗x, char ∗key0, char ∗key1)

  Registers an interactor into the interactors list with two unique IDs, one for each resonator. If resonators with the same IDs are present, they are immediately bound to the interactor.
- int SDT_unregisterInteractor (char ∗key0, char ∗key1)

  Unregisters an interactor from the interactors list. If an interactor with the given IDs is present, it is unregistered from the list.

4.45.1 Detailed Description

Bidirectional observer pattern, implementing a loose coupling between resonator and interactor objects. Particularly useful in patcher languages, where object instantiation is generally asynchronous.

4.45.2 Function Documentation

4.45.2.1 int SDT_registerInteractor (SDTInteractor ∗ *x*, char ∗ *key0*, char ∗ *key1*)

Registers an interactor into the interactors list with two unique IDs, one for each resonator. If resonators with the same IDs are present, they are immediately bound to the interactor.

Parameters

in	*x*	Resonator instance to register
in	*key0*	Unique ID of the first resonator
in	*key1*	Unique ID of the second resonator

4.45.2.2 int SDT_registerResonator (SDTResonator ∗ *x*, char ∗ *key*)

Registers a resonator into the resonators list with a unique ID. If an interactor with the same ID is present, the resonator is bound to the interactor.

Parameters

in	*x*	Resonator instance to register

in	*key*	Unique ID assigned to the resonator instance

4.45.2.3 int SDT_unregisterInteractor (char ∗ *key0*, char ∗ *key1*)

Unregisters an interactor from the interactors list. If an interactor with the given IDs is present, it is unregistered from the list.

Parameters

in	*key0*	Unique ID of the first resonator
in	*key1*	Unique ID of the second resonator

4.45.2.4 int SDT_unregisterResonator (char ∗ *key*)

Unregisters a resonator from the resonator list. If a resonator with the given ID is present, it is unregistered from the list. If also an interactor with the same ID is present, the object is released by the interactor as well.

Parameters

in	*key*	Unique ID of the resonator instance to unregister

4.46 SDTStructs.h: Common data structures

Typedefs

- typedef struct SDTHashmap SDTHashmap

  Opaque data structure for a hashmap object.
- typedef struct SDTHashmap SDTHashmap

  Opaque data structure for a hashmap object.

Functions

- SDTHashmap * SDTHashmap_new (int size)

  Object constructor.
- void SDTHashmap_free (SDTHashmap *x)

  Object destructor.
- void * SDTHashmap_get (SDTHashmap *x, char *key)

  Looks for an entry with the given key in the hashmap.
- int SDTHashmap_put (SDTHashmap *x, char *key, void *value)

  Inserts a key/value pair in the hashmap.
- int SDTHashmap_del (SDTHashmap *x, char *key)

  Deletes a key/value pair from the hashmap.
- void SDTHashmap_clear (SDTHashmap *x)

  Deletes all the entries in the hashmap.

4.46.1 Detailed Description

4.46.2 Function Documentation

4.46.2.1 int SDTHashmap_del (**SDTHashmap** * *x*, char * *key*)

Deletes a key/value pair from the hashmap.

Parameters

in	*key*	Key to look for in the hashmap

Returns

0 if deletion is succesful, 1 otherwise (e.g. key not found)

4.46.2.2 void SDTHashmap_free (**SDTHashmap** * *x*)

Object destructor.

Parameters

in	*x*	pointer to the instance to destroy

4.46.2.3 void * SDTHashmap_get (**SDTHashmap** * *x*, char * *key*)

Looks for an entry with the given key in the hashmap.

Parameters

in	*key*	Key to look for in the hashmap

Returns

Value associated to the key if found, NULL otherwise

4.46.2.4 SDTHashmap ∗ SDTHashmap_new (int *size*)

Object constructor.

Parameters

in	*size*	Number of bins in the hashmap

Returns

Pointer to the new instance

4.46.2.5 int SDTHashmap_put (SDTHashmap ∗ *x*, char ∗ *key*, void ∗ *value*)

Inserts a key/value pair in the hashmap.

Parameters

in	*key*	Key to associate to the value
in	*value*	Value to insert in the hashmap

Returns

0 if insertion is succesful, 1 otherwise (e.g. key already present)

Chapter 5

Data Structure Documentation

5.1 SDTComplex Struct Reference

Data structure containing the real and imaginary part of a complex number.

```
#include <SDTComplex.h>
```

Data Fields

- double **r**
- double **i**

5.1.1 Detailed Description

Data structure containing the real and imaginary part of a complex number.

The documentation for this struct was generated from the following file:

- src/SDT/SDT.framework/Versions/A/Headers/SDTComplex.h

Index