Purple Grain

Generated by Doxygen 1.8.17

1 Real Time Audio Programming in C

1.0.1 Granular Synth

Nikita Kretschmar - 459160 Adrian Philipp - 459173 Michael Strobl - 367103 Tim Wennemann - 462830

2 Todo List

Global OVERLAP_DENSITY

check if necessary

3 Data Structure Documentation

3.1 c_granular_synth Struct Reference

pure data struct of the *c_granular_synth* object

```
#include <c_granular_synth.h>
```

Collaboration diagram for c_granular_synth:

Data Fields

- t_word ∗ soundfile

  pointer towards the soundfile
- int soundfile_length

  lenght of the soundfile in samples
- int current_start_pos

  position in the soundfle, determined by slider position
- int playback_cycle_end

  determines when to reset playback_pos to current_start_pos
- int current_grain_index

  index of the current grain
- int current_adsr_stage_index

  index of the current ADSR stage
- int current_gauss_stage_index

  index of the current gauss stage
- int grain_size_ms

  size of a grain in milliseconds, adjustable through slider
- int grain_size_samples

  size of a grain in samples
- int num_grains

  number of grains
- int midi_pitch

  pitch/key value given by MIDI input
- int midi_velo

velocity value given by MIDI input

- float gauss_q_factor

  used to manipulate grain envelope height
- t_int playback_position

  which sample of the grain goes to the output next
- bool reverse_playback

  used fo switch playback to reverse, depends on time_stretch_factor value negativity
- float ∗ soundfile_table

  array containing the original soundfile
- t_float output_buffer

  used to sum up the current samples of all active grains
- t_float time_stretch_factor

  resizes sample length within a grain, adjustable through slider
- t_float sr

  defined samplerate
- grain ∗ grains_table

  array containing the grains
- envelope ∗ adsr_env

  ADSR envelope.

3.1.1 Detailed Description

pure data struct of the *c_granular_synth* object

pure data struct of the *c_granular_synth* object, defines all necessary variables for synth operation

Definition at line 34 of file c_granular_synth.h.

The documentation for this struct was generated from the following file:

- c_granular_synth.h

3.2 c_granular_synth_tilde_ Struct Reference

pure data struct of the *c_granular_synth_tilde* object

3.2.1 Detailed Description

pure data struct of the *c_granular_synth_tilde* object

pure data struct of the *c_granular_synth_tilde* object, sets all necessary in- and outlets and defines corresponding variables for synth operation

The documentation for this struct was generated from the following file:

- pd_granular_synth∼.c

3.3 envelope Struct Reference

Data Fields

- t_object **x_obj**
- t_int **attack**
- t_int **decay**
- t_float **sustain**
- t_int **release**
- t_int **duration**
- t_int **attack_samples**
- t_int **decay_samples**
- t_int **release_samples**
- t_sample ∗ **envelope_samples_table**
- enum adsr_stage **adsr**

3.3.1 Detailed Description

Definition at line 39 of file envelope.h.

The documentation for this struct was generated from the following file:

- envelope.h

3.4 grain Struct Reference

Collaboration diagram for grain:

Data Fields

- struct grain ∗ **next_grain**
- struct grain ∗ **previous_grain**
- t_int **grain_size_samples**
- t_int **grain_index**
- t_int **internal_step_count**
- t_float **start**
- t_float **end**
- t_float **time_stretch_factor**
- t_float **current_sample_pos**
- t_float **next_sample_pos**
- bool **grain_active**

3.4.1 Detailed Description

Definition at line 27 of file grain.h.

The documentation for this struct was generated from the following file:

- grain.h

3.5 pd_granular_synth_tilde Struct Reference

Collaboration diagram for pd_granular_synth_tilde:

Data Fields

- t_object x_obj

  object used for method input/output handling
- t_float f

  of type float, used for various input handling
- t_float sr

  defined samplerate
- c_granular_synth * synth

  pure data garnular synth object
- t_int grain_size

  size of a grain in milliseconds, adjustable through slider
- t_int start_pos

  position within the soundfile, adjustable through slider
- t_int midi_pitch

  pitch/key value given by MIDI input
- t_int midi_velo

  velocity value given by MIDI input
- t_int attack

  attack time in the range of 0 - 4000ms, adjustable through slider
- t_int decay

  decay time in the range of 0 - 4000ms, adjustable through slider
- t_int release

  sustain time in the range of 0 - 1, adjustable through slider
- t_float sustain

  release time in the range of 0 - 10000ms, adjustable through slider
- t_float time_stretch_factor

  resizes sample length within a grain, for negative values read samples in backwards direction, adjustable through slider
- t_float gauss_q_factor

  used to manipulate grain envelope height
- t_word * soundfile

  Pointer to the soundfile Array
- t_symbol * soundfile_arrayname

  String used in pd to identify array that holds the soundfile
- int soundfile_length

lenght of the soundfile in samples
- float soundfile_length_ms

  lenght of the soundfile in milliseconds
- t_word ∗ envelopeTable

  array containing the envelope
- t_inlet ∗ in_grain_size

  inlet for grain size slider
- t_inlet ∗ in_start_pos

  inlet for start position slider
- t_inlet ∗ in_time_stretch_factor

  inlet for time stretch factor slider
- t_inlet ∗ in_midi_pitch

  inlet for MIDI input pitch/key value
- t_inlet ∗ in_midi_velo

  inlet for MIDI input velocity value
- t_inlet ∗ in_attack

  inlet attack slider
- t_inlet ∗ in_decay

  inlet for decay slider
- t_inlet ∗ in_sustain

  inlet for sustain slider
- t_inlet ∗ in_release

  inlet for release slider
- t_inlet ∗ in_gauss_q_factor

  inlet for gauss q factor slider
- t_outlet ∗ out

  main outlet

Related Functions

(Note that these are not member functions.)

- void c_granular_synth_reset_playback_position (c_granular_synth ∗x)

  resets playback position
- void c_granular_synth_free (c_granular_synth ∗x)

  frees granular_synth object
- void ∗ pd_granular_synth_tilde_new (t_symbol ∗soundfile_arrayname)

  Creates a new pd_granular_synth_tilde object.
  For more information please refer to the `Pure Data Docs`
- t_int ∗ pd_granular_synth_tilde_perform (t_int ∗w)

performs pd_granular_synth_tilde

- void pd_granular_synth_tilde_free (t_pd_granular_synth_tilde ∗x)

  frees inlets of pd_granular_synth_tilde
- void pd_granular_synth_tilde_dsp (t_pd_granular_synth_tilde ∗x, t_signal ∗∗sp)

  adds pd_granular_synth_tilde to the signal processing chain
- void pd_granular_synth_tilde_setup (void)

  Setup of pd_granular_synth_tilde

3.5.1 Detailed Description

Definition at line 30 of file pd_granular_synth∼.c.

3.5.2 Friends And Related Function Documentation

3.5.2.1 c_granular_synth_free() `void c_granular_synth_free ( c_granular_synth ∗ x ) [related]`

frees granular_synth object

Parameters

x	input pointer of *c_granular_synth_free* object

Definition at line 314 of file c_granular_synth.c.

3.5.2.2 c_granular_synth_reset_playback_position() `void c_granular_synth_reset_playback_position ( c_granular_synth ∗ x ) [related]`

resets playback position

Parameters

x	input pointer of *c_granular_synth_reset_playback_position* object

Definition at line 302 of file c_granular_synth.c.

3.5.2.3 pd_granular_synth_tilde_free() `void pd_granular_synth_tilde_free ( t_pd_granular_synth_tilde ∗ x ) [related]`

frees inlets of *pd_granular_synth_tilde*

Parameters

x	input pointer of *pd_granular_synth_tilde* object

Definition at line 139 of file pd_granular_synth∼.c.

3.5.2.4 pd_granular_synth_tilde_new()

```
void * pd_granular_synth_tilde_new (
        t_symbol * soundfile_arrayname )  [related]
```

Creates a new pd_granular_synth_tilde object.
For more information please refer to the `Pure Data Docs`

< default value for soundfile length in samples

< default value for soundfile length in ms

< default value for grain size, before adjustment through slider

< default value for starting position, before adjustment through slider

< default value for time stretch factor, before adjustment through slider

< default value for MIDI input velocity, equals noteoff event

< default value for attack time, before adjustment through slider

< default value for decay time, before adjustment through slider

< default value for sustain time, before adjustment through slider

< default value for release time, before adjustment through slider

< default value for gauss q factor, before adjustment through slider

Note

The main inlet is created automatically

Definition at line 71 of file pd_granular_synth∼.c.

3.5.2.5 pd_granular_synth_tilde_perform()

```
t_int * pd_granular_synth_tilde_perform (
        t_int * w )  [related]
```

performs *pd_granular_synth_tilde*

Parameters

w	main input for performing pd_granular_synth_tilde

< passes all (slider) changes to synth

< return a pointer to the dataspace for the next dsp-object

< return argument equals the argument of the perform-routine plus the number of pointer variables +1

Definition at line 114 of file pd_granular_synth∼.c.

3.5.2.6 pd_granular_synth_tilde_setup() `void pd_granular_synth_tilde_setup ( void ) [related]`

Setup of pd_granular_synth_tilde

Warning

> "sample multiply defined" error class_sethelpsymbol(pd_granular_synth_tilde_class, gensym("pd_granular_synth∼"));

Definition at line 381 of file pd_granular_synth∼.c.

The documentation for this struct was generated from the following files:

- pd_granular_synth∼.c
- c_granular_synth.c

3.6 window Struct Reference

Data Fields

- t_object **x_obj**
- t_int **q_factor**
- t_sample ∗ **window_samples_table**

3.6.1 Detailed Description

Definition at line 55 of file envelope.h.

The documentation for this struct was generated from the following file:

- envelope.h

4 File Documentation

4.1 c_granular_synth.c File Reference

main file of the synthesizer's implementation

```
#include "c_granular_synth.h"
#include "envelope.h"
#include "grain.h"
#include "purple_utils.h"
```

Include dependency graph for c_granular_synth.c:

4.2 c_granular_synth.c

```

#include "c_granular_synth.h"
#include "envelope.h"
#include "grain.h"
#include "purple_utils.h"

c_granular_synth *c_granular_synth_new(t_word *soundfile, int soundfile_length, int grain_size_ms, int start_pos, float time_stretch_factor, int attack, int decay, float sustain, int release, float gauss_q_factor)
{
    c_granular_synth *x = (c_granular_synth *)malloc(sizeof(c_granular_synth));
    x->soundfile_length = soundfile_length;
    x->sr = sys_getsr();
    x->grain_size_ms = grain_size_ms;
    x->grain_size_samples = get_samples_from_ms(x->grain_size_ms, x->sr);
    x->soundfile_table = (float *) malloc(x->soundfile_length * sizeof(float));
    x->time_stretch_factor = time_stretch_factor;
    x->reverse_playback = (x->time_stretch_factor < 0);
    x->output_buffer = 0.0;
    x->current_start_pos = start_pos;
    x->current_grain_index = 0;
    x->current_gauss_stage_index = 0;
    c_granular_synth_adjust_current_grain_index(x);

    c_granular_synth_reset_playback_position(x);

    x->current_adsr_stage_index = 0;
    x->adsr_env = envelope_new(attack, decay, sustain, release);

    // Retrigger when user sets different grain size
    c_granular_synth_set_num_grains(x);
    post("C main file - new method - number of grains = %d", x->num_grains);
    c_granular_synth_adjust_current_grain_index(x);

    for(int i = 0; i<soundfile_length;i++)
    {
        x->soundfile_table[i] = soundfile[i].w_float;
    }

    x->grains_table = NULL;
    c_granular_synth_populate_grain_table(x);

    return x;
}

void c_granular_synth_process(c_granular_synth *x, float *in, float *out, int vector_size)
{
    int i = vector_size;
    float gauss_val, adsr_val;

    while(i--)
    {
        x->output_buffer = 0;
        // oder kann man playback jetzt einfach immer +1 hochgehen?
        x->playback_position++;
        if(x->playback_position >= x->soundfile_length)
        {
            x->playback_position = 0;
        }
        if(x->playback_position >= x->playback_cycle_end)
        {
            x->playback_position = x->current_start_pos;
        }
        //ab hier dann schauen welches grain aktiv ist
        //x->playback_position = x->grains_table[x->current_grain_index].current_sample_pos;

        grain_internal_scheduling(&x->grains_table[x->current_grain_index], x);

        //gauss_val = gauss(x->gauss_q_factor, x->grains_table[x->current_grain_index],x->grains_table[x->current_grain_index].end - x->playback_position);
        gauss_val = gauss(x);
        //gauss_val = gauss(x->gauss_q_factor, x->grain_size_samples,x->current_grain_index);
        x->output_buffer *= gauss_val;


        if(x->midi_velo > 0)
        {
            adsr_val = calculate_adsr_value(x);
        }
        else
        {
            if(x->adsr_env->adsr == SILENT)
            {
                adsr_val = 0;
            }
            // Must be in Release State
            else
            {
                if(x->adsr_env->adsr != RELEASE) x->current_adsr_stage_index = 0;
                x->adsr_env->adsr = RELEASE;
                //x->current_adsr_stage_index = 0;
                adsr_val = calculate_adsr_value(x);
            }
        }
        x->output_buffer *= adsr_val;



        *out++ = x->output_buffer;
    }

}

void c_granular_synth_set_num_grains(c_granular_synth *x)
{
    x->num_grains = (int)ceilf(fabsf(x->soundfile_length * x->time_stretch_factor) / x->grain_size_samples);
}
void c_granular_synth_adjust_current_grain_index(c_granular_synth *x)
{
    //int index = x->current_start_pos / x->grain_size_samples;
    int index = ceil((x->current_start_pos * fabs(x->time_stretch_factor)) / x->grain_size_samples);
    x->current_grain_index = index;
}
void c_granular_synth_populate_grain_table(c_granular_synth *x)
{
    grain *grains_table;
    grains_table = (grain *) calloc(x->num_grains, sizeof(grain));
    int j;
    float start_offset = 0;
    // Grain Table schreiben ab "current_grain_index"
    // Bis jetzt schreibt for schlaife nur bis ans Ende der Num Grains
    // Muss als Ring Buffer auch die ersten Grains befüllen!!



    if(x->reverse_playback)
    {
        for(j = x->current_grain_index; j >= 0; j--)
        {
            grains_table[j] = grain_new(x->grain_size_samples,
                                        x->soundfile_length,
                                        (x->current_start_pos + x->grain_size_samples), // ???
                                        j, x->time_stretch_factor);
            if(j < x->current_grain_index) grains_table[j+1].next_grain = &grains_table[j];
            /*
            if(grains_table[j].start >= x->playback_position &&  grains_table[j].end <= x->playback_position)
            {
                grains_table[j].grain_active = true;
            }
             */
            start_offset += x->time_stretch_factor * x->grain_size_samples;
        }
        grains_table[0].next_grain = &grains_table[x->num_grains - 1];
    }
    // Playback inf forward direction
    else
    {
        for(j = x->current_grain_index; j<x->num_grains; j++)
        {
            grains_table[j] = grain_new(x->grain_size_samples,
                                        x->soundfile_length,
                                        x->current_start_pos + (start_offset), // ???
                                        j, x->time_stretch_factor);
            if(j > 0) grains_table[j-1].next_grain = &grains_table[j];
            /*
            if(grains_table[j].start <= x->playback_position &&  grains_table[j].end >= x->playback_position)
            {
                grains_table[j].grain_active = true;
            }
             */
            start_offset += x->time_stretch_factor * x->grain_size_samples;
        }
        grains_table[x->num_grains - 1].next_grain = &grains_table[0];
    }

    // Das stand vorher in der process methode
    //x->playback_position = x->current_start_pos;
    c_granular_synth_reset_playback_position(x);

    if(x->grains_table) free(x->grains_table);
    x->grains_table = grains_table;
}
void c_granular_synth_properties_update(c_granular_synth *x, int grain_size_ms, int start_pos, float time_stretch_factor, int midi_velo, int midi_pitch, int attack, int decay, float sustain, int release, float gauss_q_factor)
{
    if(x->grain_size_ms != grain_size_ms || x->current_start_pos != start_pos || x->time_stretch_factor != time_stretch_factor || !x->grains_table)
    {
        if(x->grain_size_ms != grain_size_ms)
        {
            x->grain_size_ms = grain_size_ms;
            int grain_size_samples = get_samples_from_ms(grain_size_ms, x->sr);
            x->grain_size_samples = grain_size_samples;
        }
        if(x->current_start_pos != start_pos)
        {
            x->current_start_pos = start_pos;
        }
        if(x->time_stretch_factor != time_stretch_factor)
        {
            x->time_stretch_factor = time_stretch_factor;
        }
        c_granular_synth_set_num_grains(x);
        c_granular_synth_adjust_current_grain_index(x);
        c_granular_synth_populate_grain_table(x);
    }

    if(x->midi_pitch != midi_pitch)
    {
        x->midi_pitch = midi_pitch;
    }

    if(x->midi_velo != midi_velo)
    {
        x->midi_velo = midi_velo;
    }

    if (x->adsr_env->attack != attack || x->adsr_env->decay != decay || x->adsr_env->sustain != sustain || x->adsr_env->release != release)
    {
        if(x->adsr_env->attack != attack)
        {
            x->adsr_env->attack = attack;
        }
        if(x->adsr_env->decay != decay)
        {
            x->adsr_env->decay = decay;
        }
        if(x->adsr_env->sustain != sustain)
        {
            x->adsr_env->sustain = sustain;
        }
        if(x->adsr_env->release != release)
        {
            x->adsr_env->release = release;
        }
        x->adsr_env = envelope_new(attack, decay, sustain, release);
    }

    if(x->gauss_q_factor != gauss_q_factor)
    {
        x->gauss_q_factor = gauss_q_factor;
    }
}
void c_granular_synth_reset_playback_position(c_granular_synth *x)
{
    x->playback_position = x->current_start_pos;
    x->playback_cycle_end = x->current_start_pos + x->grain_size_samples;
}

void c_granular_synth_free(c_granular_synth *x)
{
    if(x)
    {
        free(x->soundfile_table);
        free(x->grains_table);
        envelope_free(x->adsr_env);
        free(x);
    }
}
```

4.3 c_granular_synth.h File Reference

header file of *granular_synth.c* file

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "math.h"
#include "grain.h"
#include "envelope.h"
#include "m_pd.h"
```

Include dependency graph for c_granular_synth.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct c_granular_synth

  pure data struct of the c_granular_synth object

Macros

- #define **NUMELEMENTS**(x) (sizeof(x) / sizeof((x)[0]))

Typedefs

- typedef struct c_granular_synth **c_granular_synth**

Functions

- void **c_granular_synth_free** (c_granular_synth ∗x)
- c_granular_synth ∗ c_granular_synth_new (t_word ∗soundfile, int soundfile_length, int grain_size_ms, int start_pos, float time_stretch_factor, int attack, int decay, float sustain, int release, float gauss_q_factor)

  initial setup of soundfile and adjustment silder related variables
- void **c_granular_synth_generate_window_function** (c_granular_synth ∗x)
- void c_granular_synth_process (c_granular_synth ∗x, float ∗in, float ∗out, int vector_size)

  refresh plaback positions, opens grain scheduleing, writes gaus value, writes into output
- void c_granular_synth_set_num_grains (c_granular_synth ∗x)

sets number of grains sets number of grains according to soundfile_length and grain_size_samples
- void c_granular_synth_adjust_current_grain_index (c_granular_synth ∗x)

  adjusts current grain index adjusts current grain index according to currents_start_pos and grain_size_samples
- void c_granular_synth_populate_grain_table (c_granular_synth ∗x)

  generates a grain table generates a grain table according to current_grain_index for negative time_stretch_factor values samples are read in backwards direction
- void grain_internal_scheduling (grain ∗g, c_granular_synth ∗synth)

  scheduling of grain playback
- void **c_granular_synth_reset_playback_position** (c_granular_synth ∗x)
- void c_granular_synth_properties_update (c_granular_synth ∗x, int grain_size_ms, int start_pos, float time↩_stretch_factor, int midi_pitch, int midi_velo, int attack, int decay, float sustain, int release, float gauss_q_↩factor)

  checks on current input states
- float calculate_adsr_value (c_granular_synth ∗x)

  calculates ADSR value
- float gauss (c_granular_synth ∗x)

  calculates gauss value calculates gauss value according to grainindex

Variables

- t_float **SAMPLERATE**

4.3.1 Detailed Description

header file of *granular_synth.c* file

Author

Kretschmar, Nikita

Philipp, Adrian

Strobl, Micha

Wennemann,Tim Audiocommunication Group, Technische Universität Berlin

Definition in file c_granular_synth.h.

4.3.2 Function Documentation

4.3.2.1 c_granular_synth_adjust_current_grain_index()

```
void c_granular_synth_adjust_current_↩
grain_index (
            c_granular_synth ∗ x )
```

adjusts current grain index adjusts current grain index according to *currents_start_pos* and *grain_size_samples*

Parameters

x	input pointer of *c_granular_synth_adjust_current_grain_index* object

Definition at line 151 of file c_granular_synth.c.

4.3.2.2 c_granular_synth_new()

```
c_granular_synth* c_granular_synth_new (
        t_word * soundfile,
        int soundfile_length,
        int grain_size_ms,
        int start_pos,
        float time_stretch_factor,
        int attack,
        int decay,
        float sustain,
        int release,
        float gauss_q_factor )
```

initial setup of soundfile and adjustment silder related variables

Parameters

soundfile	contains the soundfile which can be read in via inlet
soundfile_length	lenght of the soundfile in samples
grain_size_ms	size of a grain in milliseconds, adjustable through slider
start_pos	position within the soundfile, adjustable through slider
time_stretch_factor	resizes sample length within a grain, for negative values read samples in backwards direction, adjustable through slider
attack	attack time in the range of 0 - 4000ms, adjustable through slider
decay	decay time in the range of 0 - 4000ms, adjustable through slider
sustain	sustain time in the range of 0 - 1, adjustable through slider
release	release time in the range of 0 - 10000ms, adjustable through slider

Returns

c_granular_synth∗

Definition at line 35 of file c_granular_synth.c.

4.3.2.3 c_granular_synth_populate_grain_table()

```
void c_granular_synth_populate_grain_table (
        c_granular_synth * x )
```

generates a grain table generates a grain table according to *current_grain_index* for negative *time_stretch_factor* values samples are read in backwards direction

Parameters

x	input pointer of *c_granular_synth_populate_grain_table* object

Definition at line 163 of file c_granular_synth.c.

4.3.2.4 c_granular_synth_process()

```
void c_granular_synth_process (
        c_granular_synth * x,
        float * in,
        float * out,
        int vector_size )
```

refresh plaback positions, opens grain scheduleing, writes gaus value, writes into output

Parameters

x	input pointer of c_granular_synth_process object
in	input
out	output
vector_size	vectoral size of

Definition at line 80 of file c_granular_synth.c.

4.3.2.5 c_granular_synth_properties_update()

```
void c_granular_synth_properties_update (
        c_granular_synth * x,
        int grain_size_ms,
        int start_pos,
        float time_stretch_factor,
        int midi_velo,
        int midi_pitch,
        int attack,
        int decay,
        float sustain,
        int release,
        float gauss_q_factor )
```

checks on current input states

checks slider positions, MIDI input and ADSR state to update correspondent values

Parameters

in	***x***	**input pointer of c_granular_synth_properties_update object**
in	***midi_velo***	**MIDI input velocity value, usable through virtual or external MIDI device**
in	***midi_pitch***	**MIDI input pitch/key value, usable through virtual or external MIDI device, also used for noteon detection**
in	***grain_size_ms***	**size of a grain in milliseconds, adjustable through slider**
in	***start_pos***	**position within the soundfile, adjustable through slider**

Parameters

in	*time_stretch_factor*	**resizes sample length within a grain, adjustable through slider**
in	*attack*	**attack time in the range of 0 - 4000ms, adjustable through slider**
in	*decay*	**decay time in the range of 0 - 4000ms, adjustable through slider**
in	*sustain*	**sustain time in the range of 0 - 1, adjustable through slider**
in	*release*	**release time in the range of 0 - 10000ms, adjustable through slider**
in	*gauss_q_factor*	**envelope manipulation value in the range of 0.01 - 1, adjustable through slider**

Definition at line 237 of file c_granular_synth.c.

4.3.2.6 c_granular_synth_set_num_grains()

```
void c_granular_synth_set_num_grains (
        c_granular_synth * x )
```

sets number of grains sets number of grains according to *soundfile_length* and *grain_size_samples*

Parameters

x	input pointer of *c_granular_synth_set_num_grains* object

Definition at line 142 of file c_granular_synth.c.

4.3.2.7 calculate_adsr_value()

```
float calculate_adsr_value (
        c_granular_synth * x )
```

calculates ADSR value

calculates single atm ADSR value according to current state

Parameters

x	**input pointer of *calculate_adsr_value* object**

Returns

ADSR value of type float

Definition at line 29 of file envelope.c.

4.3.2.8 gauss()

```
float gauss (
        c_granular_synth * x )
```

calculates gauss value calculates gauss value according to *grainindex*

Parameters

x	reference to the actual synthesizer

Returns

gauss value of type float

Definition at line 122 of file envelope.c.

4.3.2.9 grain_internal_scheduling()

```
void grain_internal_scheduling (
            grain * g,
            c_granular_synth * synth )
```

scheduling of grain playback

sheduling of grain playback

Parameters

g	grain
synth	synthesized output of c_granular_synth object

<

<

<

<

<

Definition at line 89 of file grain.c.

4.4 c_granular_synth.h

```

#ifndef c_granular_synth_h
#define c_granular_synth_h

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "math.h"
#include "grain.h"
#include "envelope.h"
#include "m_pd.h"

#ifdef __cplusplus
extern "C" {
#endif

#define NUMELEMENTS(x)  (sizeof(x) / sizeof((x)[0]))

typedef struct c_granular_synth
{
    t_word      *soundfile;
    int         soundfile_length,
                current_start_pos,
                playback_cycle_end,
                current_grain_index,
                current_adsr_stage_index,
                current_gauss_stage_index,
                grain_size_ms,
                grain_size_samples,
                num_grains,
                midi_pitch,
                midi_velo;
    float       gauss_q_factor;
    t_int       playback_position;
    bool        reverse_playback;
    float       *soundfile_table;
    t_float     output_buffer,
                time_stretch_factor,
                sr;
    grain       *grains_table;
    envelope    *adsr_env;
    //float* windowing_table;  // smoothing window function applied to grain output
} c_granular_synth;

void c_granular_synth_free(c_granular_synth *x);
c_granular_synth *c_granular_synth_new(t_word *soundfile, int soundfile_length, int grain_size_ms, int start_pos, float time_stretch_factor, int attack, int decay, float sustain, int release, float gauss_q_factor);
void c_granular_synth_generate_window_function(c_granular_synth *x);
void c_granular_synth_process(c_granular_synth *x, float *in, float *out, int vector_size);
void c_granular_synth_set_num_grains(c_granular_synth *x);
void c_granular_synth_adjust_current_grain_index(c_granular_synth *x);
void c_granular_synth_populate_grain_table(c_granular_synth *x);
void grain_internal_scheduling(grain* g, c_granular_synth* synth);
void c_granular_synth_reset_playback_position(c_granular_synth *x);
void c_granular_synth_properties_update(c_granular_synth *x, int grain_size_ms, int start_pos, float time_stretch_factor, int midi_pitch, int midi_velo, int attack, int decay, float sustain, int release, float gauss_q_factor);
extern t_float SAMPLERATE;
float calculate_adsr_value(c_granular_synth *x);
float gauss (c_granular_synth *x);

#ifdef __cplusplus
}
#endif

#endif
```

4.5 envelope.c File Reference

handles envelope generation

```
#include "envelope.h"
#include "grain.h"
#include "purple_utils.h"
#include "m_pd.h"
#include "c_granular_synth.h"
```

Include dependency graph for envelope.c:

Functions

- float calculate_adsr_value (c_granular_synth ∗x)

  calculates ADSR value
- envelope ∗ envelope_new (int attack, int decay, float sustain, int release)

  generates new ADSR envelope
- float gauss (c_granular_synth ∗x)

  calculates gauss value calculates gauss value according to grainindex
- void envelope_free (envelope ∗x)

  frees envelope

4.5.1 Detailed Description

handles envelope generation

Author

Nikita Kretschmar

Adrian Philipp

Micha Strobl

Tim Wennemann

generates ADSR envelope according to adjustable attack, decay, sustain and release parameters

Version

0.1

Date

2021-09-27

Copyright

Copyright (c) 2021

Definition in file envelope.c.

4.5.2 Function Documentation

4.5.2.1 calculate_adsr_value()

```
float calculate_adsr_value (
        c_granular_synth * x )
```

calculates ADSR value

calculates single atm ADSR value according to current state

Parameters

x	**input pointer of *calculate_adsr_value* object**

Returns

ADSR value of type float

Definition at line 29 of file envelope.c.

4.5.2.2 envelope_free()

```
void envelope_free (
        envelope * x )
```

frees envelope

Parameters

x	input pointer of *envelope_free* object

Definition at line 142 of file envelope.c.

4.5.2.3 envelope_new()

```
envelope* envelope_new (
        int attack,
        int decay,
        float sustain,
        int release )
```

generates new ADSR envelope

Parameters

attack	attack time in the range of 0 - 4000ms, adjustable through slider
decay	**decay time in the range of 0 - 4000ms, adjustable through slider**
sustain	**sustain time in the range of 0 - 1, adjustable through slider**
release	**release time in the range of 0 - 10000ms, adjustable through slider**

Returns

envelope∗

Note

muss bei Note on wieder raus -> start mit silent

Definition at line 94 of file envelope.c.

4.5.2.4 gauss()

```
float gauss (
        c_granular_synth * x )
```

calculates gauss value calculates gauss value according to *grainindex*

Parameters

x	reference to the actual synthesizer

Returns

gauss value of type float

Definition at line 122 of file envelope.c.

4.6 envelope.c

```
00001
00016 #include "envelope.h"
00017 #include "grain.h"
00018 #include "purple_utils.h"
00019 #include "m_pd.h"
00020 #include "c_granular_synth.h"
00021
00022
00029 float calculate_adsr_value(c_granular_synth *x)
00030 {
00031     float adsr_val = 0;
00032     float attack_val = 0;
00033     switch(x->adsr_env->adsr)
00034     {
00035         case ATTACK:
00036             attack_val = (1.0/x->adsr_env->attack_samples);
00037             adsr_val = x->current_adsr_stage_index++ * attack_val;
00038             if(x->current_adsr_stage_index >= x->adsr_env->attack_samples)
00039             {
00040                 x->current_adsr_stage_index = 0;
00041                 x->adsr_env->adsr = DECAY;
00042             }
00043             break;
00044         case DECAY:
00045             //decay_val = (x->adsr_env->sustain-1.0)/x->adsr_env->decay_samples;
00046             adsr_val = 1.0 +
       ((x->adsr_env->sustain-1.0)/x->adsr_env->decay_samples*x->current_adsr_stage_index++);
00047             //adsr_val = 1.0 +
       ((x->adsr_env->sustain-1.0)*(x->current_adsr_stage_index++/x->adsr_env->decay_samples));
00048
00049             if(x->current_adsr_stage_index >= x->adsr_env->decay_samples)
00050             {
00051                 x->current_adsr_stage_index = 0;
00052                 x->adsr_env->adsr = SUSTAIN;
00053             }
00054             break;
00055         case SUSTAIN:
00056             adsr_val = x->adsr_env->sustain;
00057             break;
00058         case RELEASE:
00059             if(x->midi_velo > 0)
00060             {
00061                 x->adsr_env->adsr = ATTACK;
00062                 x->current_adsr_stage_index = 0;
00063                 break;
00064             }
00065             adsr_val = x->adsr_env->sustain -
       ((x->adsr_env->sustain/x->adsr_env->release_samples)*x->current_adsr_stage_index++);
00066             if(x->current_adsr_stage_index >= x->adsr_env->release_samples)
00067             {
00068                 x->current_adsr_stage_index = 0;
00069                 x->adsr_env->adsr = SILENT;
00070             }
00071             break;
00072         case SILENT:
00073             if(x->midi_velo>0)
00074             {
00075                 x->adsr_env->adsr = ATTACK;
00076                 x->current_adsr_stage_index = 0;
00077                 break;
00078             }
00079             adsr_val = 0;
00080             break;
00081     }
00082     return adsr_val;
00083 }
00084
00094 envelope *envelope_new(int attack, int decay, float sustain, int release)
00095
00096 {
00097     envelope *x = (envelope *) vas_mem_alloc(sizeof(envelope));
00098     t_float SAMPLERATE = sys_getsr();
00099
00103     x->adsr = SILENT;
```

```

    x->attack = attack;
    x->decay = decay;
    x->sustain = sustain;
    x->release = release;

    x->attack_samples = get_samples_from_ms(attack, SAMPLERATE);
    x->decay_samples = get_samples_from_ms(decay, SAMPLERATE);
    x->release_samples = get_samples_from_ms(release, SAMPLERATE);
    return x;
}

float gauss(c_granular_synth *x)
{
    //t_int grain_size = x.grain_size_samples;
    if (x->grain_size_samples == 0)
        return 0;
    if (x->current_gauss_stage_index >= x->grain_size_samples)
    {
        x->current_gauss_stage_index = 0;
    }
    float numerator = pow(x->current_gauss_stage_index++ -(x->grain_size_samples/2), 2);
    float denominatior = x->gauss_q_factor * pow(x->grain_size_samples, 2);
    float gauss_value = expf(-numerator/denominatior);
    return gauss_value;
}

void envelope_free(envelope *x)
{
    free(x);
}
```

4.7 envelope.h File Reference

header file of *envelope.c* file

```
#include "m_pd.h"
#include "grain.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

Include dependency graph for envelope.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct envelope
- struct window

Typedefs

- typedef struct envelope **envelope**
- typedef struct window **window**

Enumerations

- enum **adsr_stage** {
  ATTACK, **DECAY**, **SUSTAIN**, **RELEASE**,
  SILENT }

Functions

- int **getsamples_from_ms** (int ms, float sr)
- envelope * envelope_new (int attack, int decay, float sustain, int release)

  generates new ADSR envelope
- void envelope_free (envelope *x)

  frees envelope

4.7.1 Detailed Description

header file of *envelope.c* file

Author

Kretschmar, Nikita

Philipp, Adrian

Strobl, Micha

Wennemann,Tim Audiocommunication Group, Technische Universität Berlin

Definition in file envelope.h.

4.7.2 Function Documentation

4.7.2.1 envelope_free()

```
void envelope_free (
        envelope * x )
```

frees envelope

Parameters

x	input pointer of *envelope_free* object

Definition at line 142 of file envelope.c.

4.7.2.2 envelope_new()

```
envelope* envelope_new (
        int attack,
        int decay,
        float sustain,
        int release )
```

generates new ADSR envelope

Parameters

attack	attack time in the range of 0 - 4000ms, adjustable through slider
decay	**decay time in the range of 0 - 4000ms, adjustable through slider**
sustain	**sustain time in the range of 0 - 1, adjustable through slider**
release	**release time in the range of 0 - 10000ms, adjustable through slider**

Returns

envelope∗

Note

muss bei Note on wieder raus -> start mit silent

Definition at line 94 of file envelope.c.

4.8 envelope.h

```

#ifndef envelope_h
#define envelope_h

#include "m_pd.h"
#include "grain.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


#ifdef __cplusplus
extern "C" {
#endif

/*
    ADSR Angaben bestimmt in s oder ms?
    Konvertierung in Samples notwendig?
    Check Funktion dass Enveloe Länge nicht länger alsLänge des Soundfiles ist?
 */

enum adsr_stage {
    ATTACK,
    DECAY,
    SUSTAIN,
    RELEASE,
    SILENT
};

typedef struct envelope
{
    t_object x_obj;
    t_int attack;
    t_int decay;
    t_float sustain;
    t_int release;
    t_int duration;
    t_int attack_samples,
            decay_samples,
            release_samples;
    t_sample *envelope_samples_table;
    enum adsr_stage adsr;
} envelope;

int getsamples_from_ms(int ms, float sr);
typedef struct window
{
    t_object x_obj;
    t_int q_factor;
    t_sample *window_samples_table;
}window;

envelope *envelope_new(int attack, int decay, float sustain, int release);

//float gauss(float q_factor, int grain_size, int sample);

void envelope_free(envelope *x);

#ifdef __cplusplus
}
#endif

#endif
```

4.9 grain.c File Reference

handles grain creation

```
#include "grain.h"
#include "c_granular_synth.h"
#include "envelope.h"
#include "purple_utils.h"
```

Include dependency graph for grain.c:

Macros

- #define OVERLAP_DENSITY = 8

  set maximum amount of simoultaneously playing grains

Functions

- grain grain_new (int grain_size_samples, int soundfile_size, float start_pos, int grain_index, float time_stretch_factor)

  generates new grain
- void grain_internal_scheduling (grain ∗g, c_granular_synth ∗synth)

  scheduling of grain playback
- void grain_free (grain ∗x)

  frees grain

4.9.1 Detailed Description

handles grain creation

Author

Nikita Kretschmar

Adrian Philipp

Micha Strobl

Tim Wennemann Audiocommunication Group, Technische Universität Berlin

Version

0.1

Date

2021-09-27

Copyright

Copyright (c) 2021

Definition in file grain.c.

4.9.2 Macro Definition Documentation

4.9.2.1 OVERLAP_DENSITY `#define OVERLAP_DENSITY = 8`

set maximum amount of simoultaneously playing grains

Todo check if necessary

To-Do: Set dynamically by user input

Definition at line 35 of file grain.c.

4.9.3 Function Documentation

4.9.3.1 grain_free()

```
void grain_free (
            grain * x )
```

frees grain

frees grain

Parameters

x	input pointer of grain_fre object

Definition at line 187 of file grain.c.

4.9.3.2 grain_internal_scheduling()

```
void grain_internal_scheduling (
            grain * g,
            c_granular_synth * synth )
```

scheduling of grain playback

sheduling of grain playback

Parameters

g	grain
synth	synthesized output of c_granular_synth object

<

<

<

<

<

Definition at line 89 of file grain.c.

4.9.3.3 grain_new()

```
grain grain_new (
        int grain_size_samples,
        int soundfile_size,
        float start_pos,
        int grain_index,
        float time_stretch_factor )
```

generates new grain

generates new grain depending on *grain_size_samples*, *soundfile_size* and *grain_index*

Parameters

grain_size_samples	size of samples contained in a grain
soundfile_size	size of the soundfile which can be read in via inlet
grain_index	corresponding index of a grain
time_stretch_factor	resizes sample length within a grain, adjustable through slider

Returns

grain

Definition at line 46 of file grain.c.

4.10 grain.c

```
00001 // duration in ms and/or samples
00002 // dur_in_ms * (samplerate/1000) = dur_in_samples
00003
00004 // fade in/out -> hanning fenster in main file?
00005
00006 // start point [in samples] relative to the sound file -> PASS IN original playback point
00007 // endpoint = startpoint + duration
00008 // overlap
00009
00010 // length of the entire sound file [in samples]
00025 #include "grain.h"
00026 #include "c_granular_synth.h"
00027 #include "envelope.h"
00028 #include "purple_utils.h"
00029
00030 //static t_class *grain_class;
00035 #define OVERLAP_DENSITY = 8
00036
00037
00046 grain grain_new(int grain_size_samples, int soundfile_size, float start_pos, int grain_index, float
       time_stretch_factor)
00047 {
00048     grain x;
```

```
    grain *next_grain = NULL;
    grain *previous_grain = NULL;
    x.grain_active = false;
    x.grain_size_samples = grain_size_samples;
    x.grain_index = grain_index;
    x.internal_step_count = 0;
    x.time_stretch_factor = time_stretch_factor;


    //x.start = fabsf(x.grain_size_samples * grain_index * x.time_stretch_factor);
    x.start = start_pos;
    x.end = x.start + ((x.grain_size_samples - 1) * x.time_stretch_factor);

    if(x.end < 0) x.end = soundfile_size - 1 - x.end;
    if(x.end > soundfile_size - 1) x.end = soundfile_size - 1;

    /*
    if(time_stretch_factor < 0.0)
    {
        switch_float_values(&x.start, &x.end);
    }
     */

    x.current_sample_pos = x.start;
    x.next_sample_pos = x.current_sample_pos + x.time_stretch_factor;
    if(x.next_sample_pos < 0) x.next_sample_pos = soundfile_size - 1 - x.next_sample_pos;
    if(x.next_sample_pos >= x.end) x.next_sample_pos = x.end;

    // If the endpoint exceeds the soundfile length in positive or negative direction
    // clamp the grain length to a point the size of the file
    // maybe just use fabsf(x.end) < soundfile_size

    return x;
}
void grain_internal_scheduling(grain* g, c_granular_synth* synth)
{
    if(synth->time_stretch_factor <= -1.0)
    {
        //
    }
    if(synth->reverse_playback)
    {
        // ???
        //g->grain_active = ((int)g->start == synth->current_start_pos) ||
        g->grain_active = g->grain_index == synth->current_grain_index ||
        (((synth->soundfile_length - 1 - synth->playback_position) <= g->start) &&
          ((synth->soundfile_length - 1 - synth->playback_position) >= g->end)));
        /*
        (((g->start * synth->time_stretch_factor * -1) >= synth->playback_position) &&
        ((g->end * synth->time_stretch_factor * -1) <= (synth->playback_position *
 synth->time_stretch_factor * -1)));
         */
    }
    else
    {
        //g->grain_active = ((int)g->start == synth->current_start_pos) ||
        g->grain_active = g->grain_index == synth->current_grain_index ||
        ((g->start <= synth->playback_position) &&
         (g->end >= synth->playback_position));
    }

    if(g->grain_active)
    {
        float   left_sample,
                right_sample,
                frac,
                integral,
                weighted;


        // For negative time_stretch_factor values read samples in backwards direction
        left_sample = synth->soundfile_table[(int)floorf(g->current_sample_pos)];
        right_sample = synth->soundfile_table[(int)ceilf(g->current_sample_pos)];
        frac = modff(g->current_sample_pos, &integral);
        weighted = get_interpolated_sample_value(left_sample, right_sample,frac);
        synth->output_buffer += weighted;
        g->current_sample_pos = g->next_sample_pos;
        g->next_sample_pos += g->time_stretch_factor;
        // does the next index exceed the soundfile length? (Forward Playback)
        if(g->next_sample_pos > synth->soundfile_length)
        {
            float diff = g->next_sample_pos - synth->soundfile_length;
            g->next_sample_pos = diff;
        }
        // Or does it go negatively past 0 (Reverse Playback)
        if(g->next_sample_pos < 0.0)
        {
            g->next_sample_pos += synth->soundfile_length - 1;
        }
        g->internal_step_count++;

        /*
        if((!synth->reverse_playback && g->current_sample_pos >= g->end)
           || (synth->reverse_playback && g->current_sample_pos <= g->end)
           || g->next_sample_pos > synth->soundfile_length - 1
           || g->next_sample_pos < 0.0)
         */
        if(g->internal_step_count >= g->grain_size_samples)
        {
            //g->grain_active = false;
            // Grain wieder auf seinen Startpunkt setzen, wie bei Initialisierung in new-methode
            g->current_sample_pos = g->start;
            g->next_sample_pos = g->current_sample_pos + g->time_stretch_factor;
            g->internal_step_count = 0;
            //synth->playback_position = synth->current_start_pos;
        }

        // checken ob nächstes grain aktiv ist
        if(g->next_grain)
        {
            grain_internal_scheduling(g->next_grain, synth);
        }

    }
    else {
        // Grain nicht oder nicht mehr aktiv
        // seine current pos auf seinen start zurücksetzen
        g->current_sample_pos = g->start;
        g->next_sample_pos = g->current_sample_pos + g->time_stretch_factor;
        g->internal_step_count = 0;
        /*
        g->current_sample_pos = g->grain_size_samples * g->grain_index * g->time_stretch_factor;
        g->next_sample_pos = g->current_sample_pos + g->time_stretch_factor;
         */
        return;

    }
}
void grain_free(grain *x)
{
    free(x);
}
```

4.11 grain.h File Reference

header file to *grain.c* file

```
#include "m_pd.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
```

Include dependency graph for grain.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct grain

Typedefs

- typedef struct grain **grain**

Functions

- grain grain_new (int grain_size_samples, int soundfile_size, float start_pos, int grain_index, float time_↩ stretch_factor)

  generates new grain
- void grain_free (grain ∗x)

  frees grain

4.11.1 Detailed Description

header file to *grain.c* file

Author

Kretschmar, Nikita

Philipp, Adrian

Strobl, Micha

Wennemann,Tim Audiocommunication Group, Technische Universität Berlin

Definition in file grain.h.

4.11.2 Function Documentation

4.11.2.1 grain_free()

```
void grain_free (
        grain ∗ x )
```

frees grain

frees grain

Parameters

x	input pointer of grain_fre object

Definition at line 187 of file grain.c.

4.11.2.2 grain_new()

```
grain grain_new (
        int grain_size_samples,
        int soundfile_size,
        float start_pos,
        int grain_index,
        float time_stretch_factor )
```

generates new grain

generates new grain depending on *grain_size_samples*, *soundfile_size* and *grain_index*

Parameters

grain_size_samples	size of samples contained in a grain
soundfile_size	size of the soundfile which can be read in via inlet
grain_index	corresponding index of a grain
time_stretch_factor	resizes sample length within a grain, adjustable through slider

Returns

grain

Definition at line 46 of file grain.c.

4.12 grain.h

```

#ifndef grain_h
#define grain_h

#include "m_pd.h"

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>

#ifdef __cplusplus
extern "C" {
#endif

//import SAMPLERATE from granular_synth.h

typedef struct grain
{
    struct grain        *next_grain,
                        *previous_grain;
    t_int               grain_size_samples,   // Grain size in samples
                        grain_index,
                        internal_step_count;
    t_float             start,
                        end,
                        time_stretch_factor,
                        current_sample_pos,
                        next_sample_pos;
    bool                grain_active;

    // statt start nehme source_read_position
    // dann laufe über so viele Schritte wie grain_size_samples groß ist
    // Schrittweite modulieren, hochzählen und nach außen zurückgeben


    //grain *next_grain;          // next and previous pointers have to be passed back and forth
    //grain *previous_grain;      // between instance of granular_synth and every instantiated grain
} grain;


grain grain_new(int grain_size_samples, int soundfile_size, float start_pos, int grain_index, float
 time_stretch_factor);

// Include order forced this method to be included in c_granular_synth.h
//void grain_internal_scheduling(grain* g, c_granular_synth* synth);

void grain_free(grain *x);

#ifdef __cplusplus
}
#endif

#endif
```

4.13 purple_utils.c File Reference

useful utilities for value conversion and manipulation
useful utilities for value conversion and manipulation
outsourced into own .c file for better code readability

```
#include <stdio.h>
#include <math.h>
#include "m_pd.h"
#include "purple_utils.h"
```

Include dependency graph for purple_utils.c:

Functions

- int get_samples_from_ms (int ms, float sr)

  calculates number of samples from ms and sr

- float get_ms_from_samples (int num_samples, float sr)

  calculates sample time in ms from num_samples and sr

- float get_interpolated_sample_value (float sample_left, float sample_right, float frac)

  calculates interpolated sample value
  calculates interpolated sample value between sample_left and sample_right

- void switch_float_values (float ∗a, float ∗b)

  swaps to values swaps to values of float type

4.13.1 Detailed Description

useful utilities for value conversion and manipulation
useful utilities for value conversion and manipulation
outsourced into own .c file for better code readability

Author

Nikita Kretschmar

Adrian Philipp

Micha Strobl

Tim Wennemann

Version

0.1

Date

2021-09-27

Copyright

Copyright (c) 2021

Definition in file purple_utils.c.

4.13.2 Function Documentation

4.13.2.1 get_interpolated_sample_value()

```
float get_interpolated_sample_value (
            float sample_left,
            float sample_right,
            float frac )
```

calculates interpolated sample value
calculates interpolated sample value between *sample_left* and *sample_right*

Parameters

sample_left	value at the beginning of sample
sample_right	value at the end of sample
frac	position after decimal point

Returns

float interpolated sample value

Definition at line 63 of file purple_utils.c.

4.13.2.2 get_ms_from_samples()

```
float get_ms_from_samples (
            int num_samples,
            float sr )
```

calculates sample time in ms from *num_samples* and *sr*

Parameters

num_samples	number of samples
sr	defined samplerate

Returns

float sample time

Definition at line 45 of file purple_utils.c.

4.13.2.3 get_samples_from_ms()

```
int get_samples_from_ms (
        int ms,
        float sr )
```

calculates number of samples from *ms* and *sr*

Parameters

ms	sample time in ms
sr	defined sample rate

Returns

int number of samples

Definition at line 28 of file purple_utils.c.

4.13.2.4 switch_float_values() `void switch_float_values ( float * a, float * b )`

swaps to values swaps to values of float type

Parameters

a	first value to swapped with second
b	second value to be swappend with first

Definition at line 75 of file purple_utils.c.

4.14 purple_utils.c

```
00001
00017 #include <stdio.h>
00018 #include <math.h>
00019 #include "m_pd.h"
00020 #include "purple_utils.h"
00028 int get_samples_from_ms(int ms, float sr)
00029 {
00030     if(sr)
00031     {
00032         return ceil((sr / 1000) * ms);
00033     }
00034     else{
00035         return 0;
00036     }
00037 }
00045 float get_ms_from_samples(int num_samples, float sr)
00046 {
00047     if(sr)
00048     {
00049         return (num_samples * 1000) / sr;
00050     }
00051     else{
00052         return 0;
00053     }
00054 }
00063 float get_interpolated_sample_value(float sample_left, float sample_right, float frac)
00064 {
00065     float weighted_a = sample_left * (1 - frac);
00066     float weighted_b = sample_right * frac;
00067     return (weighted_a + weighted_b);
00068 }
00075 void switch_float_values(float *a, float *b)
00076 {
00077     float *temp_ptr = a;
00078     a = b;
00079     b = temp_ptr;
00080     return;
00081 }
```

4.15 purple_utils.h File Reference

header file to *purple_utils.c* file

This graph shows which files directly or indirectly include this file:

Functions

- int get_samples_from_ms (int ms, float sr)

  calculates number of samples from ms and sr
- float get_ms_from_samples (int num_samples, float sr)

  calculates sample time in ms from num_samples and sr
- float get_interpolated_sample_value (float sample_left, float sample_right, float frac)

  calculates interpolated sample value
  calculates interpolated sample value between sample_left and sample_right
- void switch_float_values (float ∗a, float ∗b)

  swaps to values swaps to values of float type

4.15.1 Detailed Description

header file to *purple_utils.c* file

Author

Kretschmar, Nikita

Philipp, Adrian

Strobl, Micha

Wennemann,Tim Audiocommunication Group, Technische Universität Berlin

Version

0.1

Date

2021-09-28

Copyright

Copyright (c) 2021

Definition in file purple_utils.h.

4.15.2 Function Documentation

4.15.2.1 get_interpolated_sample_value()

```
float get_interpolated_sample_value (
        float sample_left,
        float sample_right,
        float frac )
```

calculates interpolated sample value
calculates interpolated sample value between *sample_left* and *sample_right*

Parameters

sample_left	value at the beginning of sample
sample_right	value at the end of sample
frac	position after decimal point

Returns

float interpolated sample value

Definition at line 63 of file purple_utils.c.

4.15.2.2 get_ms_from_samples()

```
float get_ms_from_samples (
            int num_samples,
            float sr )
```

calculates sample time in ms from *num_samples* and *sr*

Parameters

num_samples	number of samples
sr	defined samplerate

Returns

float sample time

Definition at line 45 of file purple_utils.c.

4.15.2.3 get_samples_from_ms()

```
int get_samples_from_ms (
            int ms,
            float sr )
```

calculates number of samples from *ms* and *sr*

Parameters

ms	sample time in ms
sr	defined sample rate

Returns

int number of samples

Definition at line 28 of file purple_utils.c.

4.15.2.4 switch_float_values()

```
void switch_float_values (
        float * a,
        float * b )
```

swaps to values swaps to values of float type

Parameters

a	first value to swapped with second
b	second value to be swappend with first

Definition at line 75 of file purple_utils.c.

4.16 purple_utils.h

```

#ifndef purple_utils_h
#define purple_utils_h

int get_samples_from_ms(int ms, float sr);
float get_ms_from_samples(int num_samples, float sr);

float get_interpolated_sample_value(float sample_left, float sample_right, float frac);
void switch_float_values(float *a, float *b);

#endif /* purple_utils_h */
```