Purple Grain

Purple Grain - A granular synthesizer for PureData

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

File c_granular_synth.c

Incorporate pointers to previous grains
Define maximum grain scheduling as grain density
Smoothen output buffer values when grains overlap
Incorporate more windowing functions apart from Gauss
Pitch detection of samples

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>
```

Data Fields

- t_word ∗ soundfile

  pointer towards the soundfile

- int soundfile_length

  lenght of the soundfile in samples

- 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

- int spray_input

  randomizes the start position of each grain

- float gauss_q_factor

  used to manipulate grain envelope slope

- float pitch_factor

  scaled by pitch/key value given by MIDI input

- t_int playback_position

  which sample of the grain goes to the output next

- t_int current_start_pos

  position in the soundfle, determined by slider position

- t_int sprayed_start_pos

  start position is affected by spray_true_offset

- t_int playback_cycle_end

  determines when to reset playback_pos to current_start_pos

- t_int spray_true_offset

  actual starting position offset (initally set to 0) calculated on the run

- 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 36 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

pure data struct of the *envelope* object

```
#include <envelope.h>
```

Data Fields

- t_object x_obj

  object used for method input/output handling
- int attack

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

  decay time in the range of 0 - 4000ms, adjustable through slider
- float peak

  maximum value reached within one adsr cycle
- float sustain

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

  release time in the range of 0 - 10000ms, adjustable through slider
- int attack_samples

  attack time in samples
- int decay_samples

  decay time in samples
- int release_samples

  release time in samples
- enum adsr_stage adsr

  current ADSR stage

3.3.1 Detailed Description

pure data struct of the *envelope* object

pure data struct of the *envelope* object, defines all necessary variables for enevelope generation

Definition at line 41 of file envelope.h.

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

- envelope.h

3.4 grain Struct Reference

pure data struct of the *grain* object

```
#include <grain.h>
```

Data Fields

- struct grain ∗ next_grain

  next grain according to the current one, passed back and forth between instances of granular_synth and every instantiated grain

- struct grain ∗ previous_grain

  previous grain according to the current one, passed back and forth between instances of granular_synth and every instantiated grain

- t_int grain_size_samples

  size of the grain in samples

- t_int grain_index

  index of the current grain

- t_int internal_step_count

  count of steps

- t_float start

  starting point

- t_float end

  ending point

- t_float time_stretch_factor

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

- t_float current_sample_pos

  position of the current sample

- t_float next_sample_pos

  position of the next sample according to the current one

- bool grain_active

  current state of the grain, inactive or active

3.4.1 Detailed Description

pure data struct of the *grain* object

pure data struct of the *grain* object, defines all necessary variables for grain management

Definition at line 32 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

Data Fields

- t_float **f**
- t_object **x_obj**
- t_float **sr**
- c_granular_synth ∗ **synth**
- t_int **start_pos**
- t_int **midi_pitch**
- t_int **midi_velo**
- t_int **attack**
- t_int **decay**
- t_int **release**
- t_int **spray_input**
- t_float **sustain**
- t_float **time_stretch_factor**
- t_float **gauss_q_factor**
- t_word ∗ **soundfile**
- t_symbol ∗ **soundfile_arrayname**
- int **grain_size**
- int **soundfile_length**
- float **pitch_factor**
- float **soundfile_length_ms**
- 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_start_pos

  inlet for start position slider
- t_inlet ∗ in_grain_size

  inlet for grain size slider
- t_inlet ∗ in_time_stretch_factor

  inlet for time stretch factor slider
- t_inlet ∗ in_gauss_q_factor

  inlet for gauss q factor slider
- t_inlet ∗ in_spray

  inlet for spray slider
- 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_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.
- 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
- 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 43 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

frees *granular_synth* object

Parameters

x	input pointer of *c_granular_synth_free* object

Definition at line 364 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

Author

Kretschmar, Nikita

resets playback position

Parameters

x	input pointer of *c_granular_synth_reset_playback_position* object

Definition at line 337 of file c_granular_synth.c.

3.5.2.3 pd_granular_synth_tilde_dsp() `void pd_granular_synth_tilde_dsp ( t_pd_granular_synth_tilde * x, t_signal ** sp ) [related]`

adds *pd_granular_synth_tilde* to the signal processing chain

adds *pd_granular_synth_tilde* to the signal processing chain, activate in pd window by checking the mark at 'DSP' option

Definition at line 217 of file pd_granular_synth∼.c.

3.5.2.4 pd_granular_synth_tilde_free() `void pd_granular_synth_tilde_free ( t_pd_granular_synth_tilde * x ) [related]`

frees inlets

frees inlets of *pd_granular_synth_tilde*

Parameters

x	**input pointer of *pd_granular_synth_tilde* object**

Definition at line 159 of file pd_granular_synth∼.c.

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

Creates a new pd_granular_synth_tilde object.

< 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 pitch factor, before adjustment through slider**

< **default value for MIDI input velocity, equals noteoff event**

< **default value for MIDI input pitch/key, equals note C3**

< **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**

< **default value for spray randomizer, before adjustment through slider**

Note

The main inlet is created automatically

Definition at line 86 of file pd_granular_synth∼.c.

3.5.2.6 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

< returns pointer to dataspace for the next dsp-object

< returns argument equal to argument of the perform-routine plus the number of pointer variables +1

Definition at line 133 of file pd_granular_synth~.c.

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

setup of pd_granular_synth_tilde

setup of pd_granular_synth_tilde, with alternative constructor for using the name 'purple grain' in puredata

Definition at line 388 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

pure data struct of the *window* object

```
#include <envelope.h>
```

Data Fields

- t_object x_obj
  object used for method input/output handling
- t_int q_factor
  q factor of the gauss distribution
- t_sample * window_samples_table
  array containing the window samples

3.6.1 Detailed Description

pure data struct of the *window* object

pure data struct of the *window* object, defines all necessary variables for windowing

Definition at line 61 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,
  t_int start_pos, float time_stretch_factor, int attack, int decay, float sustain, int release, float
  gauss_q_factor, int spray_input, float pitch_factor, int midi_pitch)
{
    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->midi_pitch = midi_pitch;
    x->pitch_factor =  time_stretch_factor * (float)midi_pitch/48.0;
    x->reverse_playback = (x->pitch_factor < 0);
    x->output_buffer = 0.0;
    x->current_start_pos = start_pos;
    x->sprayed_start_pos = start_pos;
    x->current_grain_index = 0;
    x->current_gauss_stage_index = 0;
    x->spray_input = spray_input;
    x->spray_true_offset = 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);

    c_granular_synth_set_num_grains(x);
    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;

        if(x->spray_input != 0 && x->spray_true_offset == 0 && x->midi_velo != 0)
        {
            x->spray_true_offset = spray_dependant_playback_nudge(x->spray_input);
            if(x->spray_true_offset != 0)
            {
                c_granular_synth_reset_playback_position(x);
                c_granular_synth_adjust_current_grain_index(x);
                c_granular_synth_populate_grain_table(x);
            }
        }
        else
        {
            x->playback_position++;
            if(x->playback_position >= x->soundfile_length)
            {
                x->playback_position = 0;
            }
            else if(x->playback_position < 0)
            {
                x->playback_position = x->soundfile_length - 1 + x->playback_position;
            }
            else if(x->playback_position >= x->playback_cycle_end)
            {
                x->playback_position = x->current_start_pos;
            }
        }

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

        gauss_val = gauss(x);
        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;
            }
            else
            {
                if(x->adsr_env->adsr != RELEASE)
                {
                    x->current_adsr_stage_index = 0;
                    x->adsr_env->adsr = RELEASE;
                }
                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->pitch_factor) / x->grain_size_samples);
}
void c_granular_synth_adjust_current_grain_index(c_granular_synth *x)
{
    if(x->num_grains > 0)
    {
        int index = ceil((x->sprayed_start_pos * fabs(x->pitch_factor)) / x->grain_size_samples);
        x->current_grain_index = (index == 0) ? 0 : index % x->num_grains;
    }
}
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;

    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->sprayed_start_pos + x->grain_size_samples + start_offset),
                                        j, x->pitch_factor);
            if(j < x->current_grain_index) grains_table[j+1].next_grain = &grains_table[j];

            start_offset += x->pitch_factor * x->grain_size_samples;
        }
        grains_table[0].next_grain = &grains_table[x->num_grains - 1];
    }
    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->sprayed_start_pos + start_offset),
                                        j, x->pitch_factor);
            if(j > 0) grains_table[j-1].next_grain = &grains_table[j];

            start_offset += x->pitch_factor * x->grain_size_samples;
        }
        grains_table[x->num_grains - 1].next_grain = &grains_table[0];
    }

    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, t_int grain_size_ms, t_int start_pos,
 float time_stretch_factor, t_int midi_velo, t_int midi_pitch, t_int attack, t_int decay, float
 sustain, t_int release, float gauss_q_factor, t_int spray_input)
{

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

    if(x->midi_pitch != midi_pitch)
    {
        x->midi_pitch = (int)midi_pitch;
        if(x->midi_velo != 0) x->pitch_factor = time_stretch_factor * x->midi_pitch / 48.0;
    }

    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 = (int)grain_size_ms;
            int grain_size_samples = get_samples_from_ms((int)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;
            x->pitch_factor = time_stretch_factor * x->midi_pitch / 48.0;

        }
        c_granular_synth_set_num_grains(x);
        c_granular_synth_adjust_current_grain_index(x);
        c_granular_synth_populate_grain_table(x);
    }

    if(x->spray_input != spray_input)
    {
        x->spray_input = (int)spray_input;
    }

    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 = (int)attack;
        }
        if(x->adsr_env->decay != decay)
        {
            x->adsr_env->decay = (int)decay;
        }
        if(x->adsr_env->sustain != sustain)
        {
            x->adsr_env->sustain = sustain;
        }
        if(x->adsr_env->release != release)
        {
            x->adsr_env->release = (int)release;
        }
        x->adsr_env = envelope_new(x->adsr_env->attack,
                                   x->adsr_env->decay,
                                   x->adsr_env->sustain,
                                   x->adsr_env->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->sprayed_start_pos = x->current_start_pos + x->spray_true_offset;
    while(x->sprayed_start_pos < 0)
    {
        x->sprayed_start_pos += (x->soundfile_length - 1);
    }
    while(x->sprayed_start_pos >= x->soundfile_length)
    {
        x->sprayed_start_pos -= x->soundfile_length;
    }
    x->playback_position = x->sprayed_start_pos;


    x->playback_cycle_end = x->playback_position + x->grain_size_samples;
    while(x->playback_cycle_end >= x->soundfile_length)
    {
        x->playback_cycle_end -= x->soundfile_length;
    }
}

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, t_int start_pos, float time_stretch_factor, int attack, int decay, float sustain, int release, float gauss_q_factor, int spray_input, float pitch_factor, int midi_pitch)

  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)

  main synthesizer process
- void c_granular_synth_set_num_grains (c_granular_synth ∗x)

  sets number of grains
- void c_granular_synth_adjust_current_grain_index (c_granular_synth ∗x)

  adjusts current grain index
- void c_granular_synth_populate_grain_table (c_granular_synth ∗x)

  generates a grain table
- 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, t_int grain_size_ms, t_int start_pos, float time_stretch_factor, t_int midi_velo, t_int midi_pitch, t_int attack, t_int decay, float sustain, t_int release, float gauss_q_factor, t_int spray_input)

  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

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

Version

1.0

Date

2021-07-25

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

Author

Strobl, Micha

Wennemann,Tim

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 180 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,
            t_int start_pos,
            float time_stretch_factor,
            int attack,
            int decay,
            float sustain,
            int release,
            float gauss_q_factor,
            int spray_input,
            float pitch_factor,
            int midi_pitch )
```

initial setup of soundfile and adjustment silder related variables

initial setup of soundfile and adjustment silder related variables

Parameters

soundfile	contains the soundfile which can be read in via inlet
soundfile_length	length of the soundfile in samples

Parameters

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
gauss_q_factor	used to manipulate grain envelope slope in the range of 0.01 - 1, adjustable through slider
spray_input	randomizes the start position of each grain, actual starting position offset (initally set to 0) calculated on the run
pitch_factor	scaled by pitch/key value given by MIDI input
midi_pitch	MIDI input pitch/key value, usable through virtual or external MIDI device

Returns

c_granular_synth∗

Definition at line 42 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

Author

Philipp, Adrian

Strobl, Micha

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 195 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 )
```

main synthesizer process

Author

Kretschmar, Nikita

Philipp, Adrian

Strobl, Micha

Wennemann,Tim

refreshs plaback positions, starts grain scheduleing, sets gauss value, generates ADSR value according to current state

Parameters

x	input pointer of *c_granular_synth_process* object
in	input pointer of *c_granular_synth_process* object
out	output pointer of *c_granular_synth_process* object
vector_size	size of the input vector

Note

adsr must be in release state

Definition at line 95 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,
        t_int grain_size_ms,
        t_int start_pos,
        float time_stretch_factor,
        t_int midi_velo,
        t_int midi_pitch,
        t_int attack,
        t_int decay,
        float sustain,
        t_int release,
        float gauss_q_factor,
        t_int spray_input )
```

checks on current input states

Author

Philipp, Adrian

Wennemann,Tim

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, also used for noteon detection
in	*midi_pitch*	MIDI input pitch/key value, usable through virtual or external MIDI device
in	*grain_size_ms*	size of a grain in milliseconds, adjustable through slider
in	*start_pos*	position within the soundfile, adjustable through slider
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
in	*spray_input*	randomizes the start position of each grain, adjustable through slider

Definition at line 255 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

Author

Kretschmar, Nikita

Philipp, Adrian

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 169 of file c_granular_synth.c.

4.3.2.7 calculate_adsr_value()

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

calculates ADSR value

Author

Kretschmar, Nikita

Philipp, Adrian

Strobl, Micha

Wennemann,Tim

calculates single momentary ADSR value according to current state

Parameters

x	input pointer of *c_granular_synth* object

Returns

ADSR value of type float

Definition at line 33 of file envelope.c.

4.3.2.8 gauss()

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

calculates gauss value

calculates gauss value according to *grain* index

Parameters

x	reference to the actual synthesizer

Returns

gauss value of type float

Definition at line 124 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

Author

Strobl, Micha

recursive scheduling of successive grain playback with time and/or start position shifts

Parameters

g	grain
synth	pointer to c_granular_synth object that schedules the grain

Definition at line 72 of file grain.c.

4.4 c_granular_synth.h

```
00001
00013 #ifndef c_granular_synth_h
00014 #define c_granular_synth_h
00015
00016 #include <stdio.h>
00017 #include <stdlib.h>
00018 #include <stdbool.h>
00019 #include "math.h"
00020 #include "grain.h"
00021 #include "envelope.h"
00022 #include "m_pd.h"
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 #define NUMELEMENTS(x)   (sizeof(x) / sizeof((x)[0]))
00029
00036 typedef struct c_granular_synth
00037 {
00038     t_word      *soundfile;
00039     int         soundfile_length,
```

```c
                current_grain_index,
                current_adsr_stage_index,
                current_gauss_stage_index,
                grain_size_ms,
                grain_size_samples,
                num_grains,
                midi_pitch,
                midi_velo,
                spray_input;
    float       gauss_q_factor,
                pitch_factor;
    t_int       playback_position,
                current_start_pos,
                sprayed_start_pos,
                playback_cycle_end,
                spray_true_offset;
    bool        reverse_playback;
    float       *soundfile_table;
    t_float     output_buffer,
                time_stretch_factor,
                sr;
    grain       *grains_table;
    envelope    *adsr_env;
} 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,
 t_int start_pos, float time_stretch_factor, int attack, int decay, float sustain, int release, float
 gauss_q_factor, int spray_input, float pitch_factor, int midi_pitch);
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, t_int grain_size_ms, t_int start_pos,
 float time_stretch_factor, t_int midi_velo, t_int midi_pitch, t_int attack, t_int decay, float
 sustain, t_int release, float gauss_q_factor, t_int spray_input);
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
- void envelope_free (envelope ∗x)

  frees envelope

4.5.1 Detailed Description

handles envelope generation

Author

Kretschmar, Nikita

Philipp, Adrian

Strobl, Micha

Wennemann,Tim
Audiocommunication Group, Technische Universität Berlin

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

Version

1.1

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

Author

Kretschmar, Nikita

Philipp, Adrian

Strobl, Micha

Wennemann,Tim

calculates single momentary ADSR value according to current state

Parameters

x	input pointer of *c_granular_synth* object

Returns

ADSR value of type float

Definition at line 33 of file envelope.c.

4.5.2.2 envelope_free()

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

frees envelope

frees envelope

Parameters

x	input pointer of *envelope* object

Definition at line 143 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

generates new ADSR envelope according to its four components

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*

Definition at line 99 of file envelope.c.

4.5.2.4 gauss() float gauss (c_granular_synth * *x*)

calculates gauss value

calculates gauss value according to *grain* index

Parameters

x	reference to the actual synthesizer

Returns

gauss value of type float

Definition at line 124 of file envelope.c.

4.6 envelope.c

```c

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

float calculate_adsr_value(c_granular_synth *x)
{
    float adsr_val = 0;
    float attack_val = 0;
    switch(x->adsr_env->adsr)
    {
        case ATTACK:
            attack_val = (1.0/x->adsr_env->attack_samples);
            adsr_val = x->current_adsr_stage_index++ * attack_val;
            x->adsr_env->peak = adsr_val;
            if(x->current_adsr_stage_index >= x->adsr_env->attack_samples)
            {
                x->current_adsr_stage_index = 0;
                x->adsr_env->adsr = DECAY;
            }
            break;
        case DECAY:
            adsr_val = 1.0 + ((x->adsr_env->sustain-1.0)/x->adsr_env->decay_samples*x->current_adsr_stage_index++);
            x->adsr_env->peak = adsr_val;
            if(x->current_adsr_stage_index >= x->adsr_env->decay_samples)
            {
                x->current_adsr_stage_index = 0;
                x->adsr_env->adsr = SUSTAIN;
            }
            break;
        case SUSTAIN:
            adsr_val = x->adsr_env->sustain;
            if(x->adsr_env->peak != x->adsr_env->sustain) x->adsr_env->peak = x->adsr_env->sustain;
            break;
        case RELEASE:
            if(x->midi_velo > 0)
            {
                x->adsr_env->adsr = ATTACK;
                x->current_adsr_stage_index = 0;
                break;
            }
            adsr_val = x->adsr_env->peak -
 ((x->adsr_env->peak/x->adsr_env->release_samples)*x->current_adsr_stage_index++);
            if(x->current_adsr_stage_index >= x->adsr_env->release_samples)
            {
                x->current_adsr_stage_index = 0;
                x->adsr_env->adsr = SILENT;
            }
            break;
        case SILENT:
            if(x->midi_velo>0)
            {
                x->adsr_env->adsr = ATTACK;
                x->current_adsr_stage_index = 0;
                break;
            }
            adsr_val = 0;
            x->adsr_env->peak = 0;
            break;
    }
    return adsr_val;
}

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

{
    envelope *x = (envelope *) malloc(sizeof(envelope));
    t_float SAMPLERATE = sys_getsr();

    x->adsr = SILENT;
    x->attack = attack;
    x->decay = decay;
    x->sustain = sustain;
    x->peak = 0.0;
    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)
{
    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

  pure data struct of the envelope object
- struct window

  pure data struct of the window object

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

Version

1.0

Date

2021-09-27

Definition in file envelope.h.

4.7.2 Function Documentation

4.7.2.1 envelope_free()

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

frees envelope

frees envelope, necessary reset for further instances of envelope generation

Parameters

x	input pointer of *envelope_free* object

frees envelope

Parameters

x	input pointer of *envelope* object

Definition at line 143 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

generates new ADSR envelope according to its four components

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∗

Definition at line 99 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

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

typedef struct envelope
{
    t_object x_obj;
    int     attack;
    int     decay;
    float   peak,
            sustain;
    int     release;
    int     attack_samples,
            decay_samples,
            release_samples;
    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);

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:

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

Kretschmar, Nikita

Philipp, Adrian

Strobl, Micha

Wennemann,Tim
Audiocommunication Group, Technische Universität Berlin

handles grain creation and basic scheduling according to input parameters set by the synthesizer

Version

1.1

Date

2021-09-27

Copyright

Copyright (c) 2021

Definition in file grain.c.

4.9.2 Function Documentation

4.9.2.1 grain_free()

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

frees grain

frees grain

Parameters

x	input pointer of grain object

Definition at line 142 of file grain.c.

4.9.2.2 grain_internal_scheduling()

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

scheduling of grain playback

Author

Strobl, Micha

recursive scheduling of successive grain playback with time and/or start position shifts

Parameters

g	grain
synth	pointer to c_granular_synth object that schedules the grain

Definition at line 72 of file grain.c.

4.9.2.3 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 with *grain_index* according to set *grain_size_samples*, *start_pos*, *time_stretch_factor* based on *soundfile_size*

Parameters

grain_size_samples	size of a grain as amount of contained samples
soundfile_size	size of the soundfile in samples

Parameters

start_pos	starting position within the soundfile, adjustable through slider
grain_index	corresponding index of a grain
time_stretch_factor	resizes sample length within a grain, adjustable through slider

Returns

grain

Definition at line 31 of file grain.c.

4.10 grain.c

```
00001
00016 #include "grain.h"
00017 #include "c_granular_synth.h"
00018 #include "envelope.h"
00019 #include "purple_utils.h"
00020
00031 grain grain_new(int grain_size_samples, int soundfile_size, float start_pos, int grain_index, float
       time_stretch_factor)
00032 {
00033     grain x;
00034     x.grain_active = false;
00035     x.grain_size_samples = grain_size_samples;
00036     x.grain_index = grain_index;
00037     x.internal_step_count = 0;
00038     x.time_stretch_factor = time_stretch_factor;
00039     bool reverse_playback = x.time_stretch_factor < 0.0;
00040
00041     x.start = start_pos;
00042     if(x.start < 0) x.start += (soundfile_size - 1);
00043     x.end = x.start + ((x.grain_size_samples - 1) * x.time_stretch_factor);
00044
00045     if(x.end < 0) x.end += soundfile_size - 1;
00046     if(x.end > soundfile_size - 1) x.end -= (soundfile_size - 1);
00047
00048     x.current_sample_pos = x.start;
00049     x.next_sample_pos = x.current_sample_pos + x.time_stretch_factor;
00050
00051     if(reverse_playback)
00052     {
00053         if(x.next_sample_pos < 0) x.next_sample_pos += (soundfile_size - 1);
00054         if(x.next_sample_pos < x.end && x.start > x.end) x.next_sample_pos = x.end;
00055
00056     }
00057     else
00058     {
00059         if(x.next_sample_pos > (soundfile_size - 1)) x.next_sample_pos -= (soundfile_size - 1);
00060         if(x.next_sample_pos >= x.end && x.start < x.end) x.next_sample_pos = x.end;
00061     }
00062
00063     return x;
00064 }
00072 void grain_internal_scheduling(grain* g, c_granular_synth* synth)
00073 {
00074     if(synth->reverse_playback)
00075     {
00076         g->grain_active = g->grain_index == synth->current_grain_index ||
00077         (((synth->soundfile_length - 1 - synth->playback_position) <= g->start) &&
00078           ((synth->soundfile_length - 1 - synth->playback_position) >= g->end)));
00079     }
00080     else
00081     {
00082         g->grain_active = g->grain_index == synth->current_grain_index ||
00083         ((g->start <= synth->playback_position) &&
00084          (g->end >= synth->playback_position));
00085     }
00086
00087     if(g->grain_active)
00088     {
        float   left_sample,
                right_sample,
                frac,
                integral,
                weighted;

        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 += synth->pitch_factor;

        if(g->next_sample_pos > (synth->soundfile_length - 1))
        {
            g->next_sample_pos -= (synth->soundfile_length - 1);
        }

        if(g->next_sample_pos < 0.0)
        {
            g->next_sample_pos += (synth->soundfile_length - 1);
        }
        g->internal_step_count++;

        if(g->internal_step_count >= g->grain_size_samples)
        {
            g->current_sample_pos = g->start;
            g->next_sample_pos = g->current_sample_pos + synth->pitch_factor;
            g->internal_step_count = 0;
            synth->spray_true_offset = 0;
            c_granular_synth_reset_playback_position(synth);
        }

        if(g->next_grain)
        {
            grain_internal_scheduling(g->next_grain, synth);
        }

    }
    else {
        g->current_sample_pos = g->start;
        g->next_sample_pos = g->current_sample_pos + synth->pitch_factor;
        g->internal_step_count = 0;
        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

    pure data struct of the grain object

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

Version

1.0

Date

2021-09-27

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, necessary reset for further instances of grain genration

Parameters

x	input pointer of *grain_free* object

frees grain

Parameters

x	input pointer of grain object

Definition at line 142 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 with *grain_index* according to set *grain_size_samples*, *start_pos*, *time_stretch_factor* based on *soundfile_size*

Note

include order forced this method to be included in c_granular_synth.h

generates new grain with *grain_index* according to set *grain_size_samples*, *start_pos*, *time_stretch_factor* based on *soundfile_size*

Parameters

grain_size_samples	size of a grain as amount of contained samples
soundfile_size	size of the soundfile in samples
start_pos	starting position within the soundfile, adjustable through slider
grain_index	corresponding index of a grain
time_stretch_factor	resizes sample length within a grain, adjustable through slider

Returns

grain

Definition at line 31 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

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

} grain;

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


void grain_free(grain *x);

#ifdef __cplusplus
}
#endif

#endif
```

4.13 purple_utils.c File Reference

useful utilities for value conversion and manipulation

```
#include <stdio.h>
#include <stdlib.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
- float get_ms_from_samples (int num_samples, float sr)
  calculates sample time in ms
- float get_interpolated_sample_value (float sample_left, float sample_right, float frac)
  calculates interpolated sample value
- void switch_float_values (float ∗a, float ∗b)
  swaps to values
- int spray_dependant_playback_nudge (int spray_input)
  randomizes spray input value

4.13.1 Detailed Description

useful utilities for value conversion and manipulation

Author

Kretschmar, Nikita

Philipp, Adrian

Strobl, Micha

Wennemann,Tim
Audiocommunication Group, Technische Universität Berlin

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

Version

1.1

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 64 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

calculates sample time from *num_samples* according to defined *sr*

Parameters

num_samples	number of samples
sr	defined samplerate

Returns

float sample time

Definition at line 46 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

calculates number of samples from *ms* according to defined *sr*

Parameters

ms	sample time in ms
sr	defined sample rate

Returns

int number of samples

Definition at line 29 of file purple_utils.c.

4.13.2.4 spray_dependant_playback_nudge()

```
int spray_dependant_playback_nudge (
            int spray_input )
```

randomizes spray input value

randomizes spray input value for randomized start position of each grain

Parameters

spray_input	spray input value

Returns

int randomized value

Definition at line 89 of file purple_utils.c.

4.13.2.5 switch_float_values()

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

swaps to values

swaps to values *a* with *b* using a temporary third pointer

Parameters

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

Definition at line 76 of file purple_utils.c.

4.14 purple_utils.c

```
00001
00017 #include <stdio.h>
00018 #include <stdlib.h>
00019 #include <math.h>
00020 #include "m_pd.h"
00021 #include "purple_utils.h"
00029 int get_samples_from_ms(int ms, float sr)
00030 {
00031     if(sr)
00032     {
00033         return ceil((sr / 1000) * ms);
```

```
    }
    else{
        return 0;
    }
}
float get_ms_from_samples(int num_samples, float sr)
{
    if(sr)
    {
        return (num_samples * 1000) / sr;
    }
    else{
        return 0;
    }
}
float get_interpolated_sample_value(float sample_left, float sample_right, float frac)
{
    float weighted_a = sample_left * (1 - frac);
    float weighted_b = sample_right * frac;
    return (weighted_a + weighted_b);
}
void switch_float_values(float *a, float *b)
{
    float *temp_ptr = a;
    a = b;
    b = temp_ptr;
    return;
}
int spray_dependant_playback_nudge(int spray_input)
{
    if(spray_input == 0) return 0;
    int off = rand() % (2 * spray_input);
    return off - spray_input;
}
```

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
- float get_ms_from_samples (int num_samples, float sr)

  calculates sample time in ms
- float get_interpolated_sample_value (float sample_left, float sample_right, float frac)

  calculates interpolated sample value
- void switch_float_values (float ∗a, float ∗b)

  swaps to values
- int spray_dependant_playback_nudge (int spray_input)

  randomizes spray input value

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

1.1

Date

2021-09-27

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 64 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

calculates sample time from *num_samples* according to defined *sr*

Parameters

num_samples	number of samples
sr	defined samplerate

Returns

float sample time

Definition at line 46 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

calculates number of samples from *ms* according to defined *sr*

Parameters

ms	sample time in ms
sr	defined sample rate

Returns

int number of samples

Definition at line 29 of file purple_utils.c.

4.15.2.4 spray_dependant_playback_nudge() `int spray_dependant_playback_nudge ( int spray_input )`

randomizes spray input value

randomizes spray input value for randomized start position of each grain

Parameters

spray_input	spray input value

Returns

int randomized value

Definition at line 89 of file purple_utils.c.

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

swaps to values

swaps to values *a* with *b* using a temporary third pointer

Parameters

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

Definition at line 76 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);
int spray_dependant_playback_nudge(int spray_input);

#endif
```