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

Global OVERLAP_DENSITY
check if necessary, set dynamically by user input

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_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**
- float gauss_q_factor

  used to manipulate grain envelope slope
- float **pitch_factor**
- t_int playback_position

  which sample of the grain goes to the output next
- t_int current_start_pos

  position in the soundfile, determined by slider position
- t_int **sprayed_start_pos**
- t_int playback_cycle_end

  determines when to reset playback_pos to current_start_pos
- t_int **spray_true_offset**
- 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

pure data struct of the *envelope* object

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

Data Fields

- t_object x_obj
  object used for method input/output handling
- 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_float **peak**
- t_float sustain
  sustain time in the range of 0 - 1, adjustable through slider
- t_int release
  release time in the range of 0 - 10000ms, adjustable through slider
- t_int attack_samples
  attack time in samples
- t_int decay_samples
  decay time in samples
- t_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 39 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>`

Collaboration diagram for grain:

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 30 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
  release time in the range of 0 - 10000ms, adjustable through slider
- t_int spray_input
  randomizes the start position of each grain in the range of , adjustable through slider

- t_float sustain

  sustain time in the range of 0 - 1, 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 slope

- 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 pitch_factor

  in the range of

- float soundfile_length_ms

  lenght of the soundfile in milliseconds

- 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_inlet * in_spray

  inlet for spray 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 22 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 351 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 324 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 212 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 138 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 65 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 112 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

Warning

"sample multiply defined" error in class_sethelpsymbol(pd_granular_synth_tilde_class, gensym("pd_↩ granular_synth~"));

Definition at line 384 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 59 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

```
00001
00016 #include "c_granular_synth.h"
00017 #include "envelope.h"
00018 #include "grain.h"
00019 #include "purple_utils.h"
00020
00039 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, int spray_input, float pitch_factor, int midi_pitch)
00040 {
00041     c_granular_synth *x = (c_granular_synth *)malloc(sizeof(c_granular_synth));
00042     x->soundfile_length = soundfile_length;
00043     x->sr = sys_getsr();
00044     x->grain_size_ms = grain_size_ms;
00045     x->grain_size_samples = get_samples_from_ms(x->grain_size_ms, x->sr);
00046     x->soundfile_table = (float *) malloc(x->soundfile_length * sizeof(float));
00047     x->time_stretch_factor = time_stretch_factor;
00048     x->midi_pitch = midi_pitch;
00049     x->pitch_factor =  time_stretch_factor * (float)midi_pitch/48.0;
00050     x->reverse_playback = (x->pitch_factor < 0);
00051     x->output_buffer = 0.0;
00052     x->current_start_pos = start_pos;
00053     x->sprayed_start_pos = start_pos;
00054     x->current_grain_index = 0;
00055     x->current_gauss_stage_index = 0;
00056     x->spray_input = spray_input;
00057     x->spray_true_offset = 0;
00058     c_granular_synth_adjust_current_grain_index(x);
00059
00060     c_granular_synth_reset_playback_position(x);
00061
00062     x->current_adsr_stage_index = 0;
00063     x->adsr_env = envelope_new(attack, decay, sustain, release);
00064
00065     // Retrigger when user sets different grain size
00066     c_granular_synth_set_num_grains(x);
00067     post("C main file - new method - number of grains = %d", x->num_grains);
00068     c_granular_synth_adjust_current_grain_index(x);
00069
00070     for(int i = 0; i<soundfile_length;i++)
00071     {
00072         x->soundfile_table[i] = soundfile[i].w_float;
00073     }
00074
00075     x->grains_table = NULL;
00076     c_granular_synth_populate_grain_table(x);
00077
00078     return x;
00079 }
00080
00089 void c_granular_synth_process(c_granular_synth *x, float *in, float *out, int vector_size)
00090 {
00091     int i = vector_size;
00092     float gauss_val, adsr_val;
00093
00094      while(i--)
00095     {
00096         x->output_buffer = 0;
00097
00098         if(x->spray_input != 0 && x->spray_true_offset == 0 && x->midi_velo != 0)
00099         {
00100             x->spray_true_offset = spray_dependant_playback_nudge(x->spray_input);
00101             if(x->spray_true_offset != 0)
```

```
00102             {
00103                 c_granular_synth_reset_playback_position(x);
00104                 c_granular_synth_adjust_current_grain_index(x);
00105                 c_granular_synth_populate_grain_table(x);
00106             }
00107         }
00108         else
00109         {
00110             x->playback_position++;
00111             if(x->playback_position >= x->soundfile_length)
00112             {
00113                 x->playback_position = 0;
00114             }
00115             else if(x->playback_position < 0)
00116             {
00117                 x->playback_position = x->soundfile_length - 1 + x->playback_position;
00118             }
00119             else if(x->playback_position >= x->playback_cycle_end)
00120             {
00121                 x->playback_position = x->current_start_pos;
00122             }
00123         }
00124 
00125         grain_internal_scheduling(&x->grains_table[x->current_grain_index], x);
00126 
00127         gauss_val = gauss(x);
00128         x->output_buffer *= gauss_val;
00129 
00130 
00131         if(x->midi_velo > 0)
00132         {
00133             adsr_val = calculate_adsr_value(x);
00134         }
00135         else
00136         {
00137             if(x->adsr_env->adsr == SILENT)
00138             {
00139                 adsr_val = 0;
00140             }
00141             // Must be in Release State
00142             else
00143             {
00144                 if(x->adsr_env->adsr != RELEASE)
00145                 {
00146                     x->current_adsr_stage_index = 0;
00147                 }
00148                 x->adsr_env->adsr = RELEASE;
00149 
00150                 //x->current_adsr_stage_index = 0;
00151                 adsr_val = calculate_adsr_value(x);
00152             }
00153         }
00154         x->output_buffer *= adsr_val;
00155         *out++ = x->output_buffer;
00156     }
00157 
00158 }
00159 
00165 void c_granular_synth_set_num_grains(c_granular_synth *x)
00166 {
00167     x->num_grains = (int)ceilf(fabsf(x->soundfile_length * x->pitch_factor) / x->grain_size_samples);
00168 }
00174 void c_granular_synth_adjust_current_grain_index(c_granular_synth *x)
00175 {
00176     if(x->num_grains > 0)
00177     {
00178         int index = ceil((x->sprayed_start_pos * fabs(x->pitch_factor)) / x->grain_size_samples);
00179         x->current_grain_index = (index == 0) ? 0 : index % x->num_grains;
00180     }
00181 }
00188 void c_granular_synth_populate_grain_table(c_granular_synth *x)
00189 {
00190     grain *grains_table;
00191     grains_table = (grain *) calloc(x->num_grains, sizeof(grain));
00192     int j;
00193     float start_offset = 0;
00194 
00195     if(x->reverse_playback)
00196     {
00197         for(j = x->current_grain_index; j >= 0; j--)
00198         {
00199 
00200             grains_table[j] = grain_new(x->grain_size_samples,
00201                                         x->soundfile_length,
00202                                         (x->sprayed_start_pos + x->grain_size_samples + start_offset),
00203                                         j, x->pitch_factor);
00204             if(j < x->current_grain_index) grains_table[j+1].next_grain = &grains_table[j];
```

```
00205
00206             start_offset += x->pitch_factor * x->grain_size_samples;
00207         }
00208         grains_table[0].next_grain = &grains_table[x->num_grains - 1];
00209     }
00210     // Playback in forward direction
00211     else
00212     {
00213         for(j = x->current_grain_index; j<x->num_grains; j++)
00214         {
00215             grains_table[j] = grain_new(x->grain_size_samples,
00216                                         x->soundfile_length,
00217                                         (x->sprayed_start_pos + start_offset),
00218                                         j, x->pitch_factor);
00219             if(j > 0) grains_table[j-1].next_grain = &grains_table[j];
00220
00221             start_offset += x->pitch_factor * x->grain_size_samples;
00222         }
00223         grains_table[x->num_grains - 1].next_grain = &grains_table[0];
00224     }
00225
00226     c_granular_synth_reset_playback_position(x);
00227
00228     if(x->grains_table) free(x->grains_table);
00229     x->grains_table = grains_table;
00230 }
00246 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)
00247 {
00248
00249     if(x->midi_velo != midi_velo)
00250     {
00251         x->midi_velo = (int)midi_velo;
00252     }
00253
00254     if(x->midi_pitch != midi_pitch)
00255     {
00256         x->midi_pitch = (int)midi_pitch;
00257         if(x->midi_velo != 0) x->pitch_factor = time_stretch_factor * x->midi_pitch / 48.0;
00258     }
00259
00260     if(x->grain_size_ms != grain_size_ms ||
00261        x->current_start_pos != start_pos ||
00262        x->time_stretch_factor != time_stretch_factor ||
00263        !x->grains_table)
00264     {
00265         if(x->grain_size_ms != grain_size_ms)
00266         {
00267             x->grain_size_ms = (int)grain_size_ms;
00268             int grain_size_samples = get_samples_from_ms((int)grain_size_ms, x->sr);
00269             x->grain_size_samples = grain_size_samples;
00270         }
00271         if(x->current_start_pos != start_pos)
00272         {
00273             x->current_start_pos = (int)start_pos;
00274         }
00275
00276         if(x->time_stretch_factor != time_stretch_factor)
00277         {
00278             x->time_stretch_factor = time_stretch_factor;
00279             x->pitch_factor = time_stretch_factor * x->midi_pitch / 48.0;
00280
00281         }
00282         c_granular_synth_set_num_grains(x);
00283         c_granular_synth_adjust_current_grain_index(x);
00284         c_granular_synth_populate_grain_table(x);
00285     }
00286
00287     if(x->spray_input != spray_input)
00288     {
00289         x->spray_input = (int)spray_input;
00290     }
00291
00292     if (x->adsr_env->attack != attack || x->adsr_env->decay != decay || x->adsr_env->sustain !=
      sustain || x->adsr_env->release != release)
00293     {
00294         if(x->adsr_env->attack != attack)
00295         {
00296             x->adsr_env->attack = (int)attack;
00297         }
00298         if(x->adsr_env->decay != decay)
00299         {
00300             x->adsr_env->decay = (int)decay;
00301         }
00302         if(x->adsr_env->sustain != sustain)
00303         {
```

```
00304             x->adsr_env->sustain = sustain;
00305         }
00306         if(x->adsr_env->release != release)
00307         {
00308             x->adsr_env->release = (int)release;
00309         }
00310         x->adsr_env = envelope_new(x->adsr_env->attack, decay, sustain, release);
00311     }
00312
00313     if(x->gauss_q_factor != gauss_q_factor)
00314     {
00315         x->gauss_q_factor = gauss_q_factor;
00316     }
00317 }
00324 void c_granular_synth_reset_playback_position(c_granular_synth *x)
00325 {
00326     x->sprayed_start_pos = x->current_start_pos + x->spray_true_offset;
00327     while(x->sprayed_start_pos < 0)
00328     {
00329         x->sprayed_start_pos += (x->soundfile_length - 1);
00330     }
00331     while(x->sprayed_start_pos >= x->soundfile_length)
00332     {
00333         x->sprayed_start_pos -= x->soundfile_length;
00334     }
00335     x->playback_position = x->sprayed_start_pos;
00336
00337
00338     x->playback_cycle_end = x->playback_position + x->grain_size_samples;
00339     while(x->playback_cycle_end >= x->soundfile_length)
00340     {
00341         x->playback_cycle_end -= x->soundfile_length;
00342     }
00343 }
00344
00351 void c_granular_synth_free(c_granular_synth *x)
00352 {
00353     if(x)
00354     {
00355         free(x->soundfile_table);
00356         free(x->grains_table);
00357         envelope_free(x->adsr_env);
00358         free(x);
00359     }
00360 }
```

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, 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)

  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, 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

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 174 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,
            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
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

Parameters

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 , 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	**multiplicator of MIDI input pitch/key value, adjustable through slider**
midi_pitch	**MIDI input pitch/key value, usable through virtual or external MIDI device**

Returns

c_granular_synth∗

Definition at line 39 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 188 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 89 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

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

Definition at line 246 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 165 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 120 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 88 of file grain.c.

4.4 c_granular_synth.h

```
00001
00011 #ifndef c_granular_synth_h
00012 #define c_granular_synth_h
00013
00014 #include <stdio.h>
00015 #include <stdlib.h>
00016 #include <stdbool.h>
00017 #include "math.h"
00018 #include "grain.h"
00019 #include "envelope.h"
00020 #include "m_pd.h"
00021
00022 #ifdef __cplusplus
00023 extern "C" {
00024 #endif
00025
00026 #define NUMELEMENTS(x)  (sizeof(x) / sizeof((x)[0]))
00027
00034 typedef struct c_granular_synth
00035 {
00036     t_word      *soundfile;
00037     int         soundfile_length,
00038                 current_grain_index,
00039                 current_adsr_stage_index,
00040                 current_gauss_stage_index,
00041                 grain_size_ms,
00042                 grain_size_samples,
00043                 num_grains,
00044                 midi_pitch,
00045                 midi_velo,
00046                 spray_input;
00047     float       gauss_g_factor,
00048                 pitch_factor;
00049     t_int       playback_position,
00050                 current_start_pos,
00051                 sprayed_start_pos,          // start_pos affected by spray offset
00052                 playback_cycle_end,
00053                 spray_true_offset;
00054     bool        reverse_playback;
00055     float       *soundfile_table;
00056     t_float     output_buffer,
00057                 time_stretch_factor,
00058                 sr;
00059     grain       *grains_table;
```

```
00060     envelope    *adsr_env;
00061     //float* windowing_table;  // smoothing window function applied to grain output
00062 } c_granular_synth;
00063
00064 void c_granular_synth_free(c_granular_synth *x);
00065 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, int spray_input, float pitch_factor, int midi_pitch);
00066 void c_granular_synth_generate_window_function(c_granular_synth *x);
00067 void c_granular_synth_process(c_granular_synth *x, float *in, float *out, int vector_size);
00068 void c_granular_synth_set_num_grains(c_granular_synth *x);
00069 void c_granular_synth_adjust_current_grain_index(c_granular_synth *x);
00070 void c_granular_synth_populate_grain_table(c_granular_synth *x);
00071 void grain_internal_scheduling(grain* g, c_granular_synth* synth);
00072 void c_granular_synth_reset_playback_position(c_granular_synth *x);
00073 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);
00074 extern t_float SAMPLERATE;
00075 float calculate_adsr_value(c_granular_synth *x);
00076 float gauss (c_granular_synth *x);
00077
00078 #ifdef __cplusplus
00079 }
00080 #endif
00081
00082 #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

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

frees envelope

Parameters

x	input pointer of *envelope_free* object

Definition at line 140 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∗

Definition at line 95 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 120 of file envelope.c.

4.6 envelope.c

```
00001
00017 #include "envelope.h"
00018 #include "grain.h"
00019 #include "purple_utils.h"
00020 #include "m_pd.h"
00021 #include "c_granular_synth.h"
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             x->adsr_env->peak = adsr_val;
00039             if(x->current_adsr_stage_index >= x->adsr_env->attack_samples)
00040             {
00041                 x->current_adsr_stage_index = 0;
00042                 x->adsr_env->adsr = DECAY;
00043             }
00044             break;
00045         case DECAY:
00046             adsr_val = 1.0 +
       ((x->adsr_env->sustain-1.0)/x->adsr_env->decay_samples*x->current_adsr_stage_index++);
00047             x->adsr_env->peak = adsr_val;
00048             if(x->current_adsr_stage_index >= x->adsr_env->decay_samples)
00049             {
00050                 x->current_adsr_stage_index = 0;
00051                 x->adsr_env->adsr = SUSTAIN;
00052             }
00053             break;
00054         case SUSTAIN:
00055             adsr_val = x->adsr_env->sustain;
00056             if(x->adsr_env->peak != x->adsr_env->sustain) x->adsr_env->peak = 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->peak -
       ((x->adsr_env->peak/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             x->adsr_env->peak = 0;
00081             break;
00082     }
00083     return adsr_val;
00084 }
```

```
00085
00095 envelope *envelope_new(int attack, int decay, float sustain, int release)
00096
00097 {
00098     envelope *x = (envelope *) malloc(sizeof(envelope));
00099     t_float SAMPLERATE = sys_getsr();
00100
00101     x->adsr = SILENT;
00102     x->attack = attack;
00103     x->decay = decay;
00104     x->sustain = sustain;
00105     x->peak = 0.0;
00106     x->release = release;
00107
00108     x->attack_samples = get_samples_from_ms(attack, SAMPLERATE);
00109     x->decay_samples = get_samples_from_ms(decay, SAMPLERATE);
00110     x->release_samples = get_samples_from_ms(release, SAMPLERATE);
00111     return x;
00112 }
00113
00120 float gauss(c_granular_synth *x)
00121 {
00122     //t_int grain_size = x.grain_size_samples;
00123     if (x->grain_size_samples == 0)
00124         return 0;
00125     if (x->current_gauss_stage_index >= x->grain_size_samples)
00126     {
00127         x->current_gauss_stage_index = 0;
00128     }
00129     float numerator = pow(x->current_gauss_stage_index++ -(x->grain_size_samples/2), 2);
00130     float denominatior = x->gauss_q_factor * pow(x->grain_size_samples, 2);
00131     float gauss_value = expf(-numerator/denominatior);
00132     return gauss_value;
00133 }
00134
00140 void envelope_free(envelope *x)
00141 {
00142     free(x);
00143 }
```

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

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_free* object

Definition at line 140 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*

Definition at line 95 of file envelope.c.

4.8 envelope.h

```
00001
00011 #ifndef envelope_h
00012 #define envelope_h
00013
00014 #include "m_pd.h"
00015 #include "grain.h"
00016 #include <stdio.h>
00017 #include <stdlib.h>
00018 #include <math.h>
00019
00020
00021 #ifdef __cplusplus
00022 extern "C" {
00023 #endif
00024
00025 enum adsr_stage {
00026     ATTACK,
00027     DECAY,
00028     SUSTAIN,
00029     RELEASE,
00030     SILENT
00031 };
00032
00039 typedef struct envelope
00040 {
00041     t_object x_obj;
```

```
00042     t_int attack;
00043     t_int decay;
00044     t_float peak,
00045          sustain;
00046     t_int release;
00047     t_int attack_samples,
00048           decay_samples,
00049           release_samples;
00050     enum adsr_stage adsr;
00051 } envelope;
00052
00053 int getsamples_from_ms(int ms, float sr);
00059 typedef struct window
00060 {
00061     t_object x_obj;
00062     t_int q_factor;
00063     t_sample *window_samples_table;
00064 }window;
00065
00066 envelope *envelope_new(int attack, int decay, float sustain, int release);
00067
00073 void envelope_free(envelope *x);
00074
00075 #ifdef __cplusplus
00076 }
00077 #endif
00078
00079 #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

Kretschmar, Nikita
Philipp, Adrian
Strobl, Micha
Wennemann,Tim
Audiocommunication Group, Technische Universität Berlin

handles grain creation according to set input parameters

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, set dynamically by user input

Definition at line 25 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 183 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 88 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 with *grain_index* according to set *grain_size_samples*, *start_pos*, *time_stretch_factor* based on *soundfile_size*

Parameters

grain_size_samples	size of samples contained in a grain
soundfile_size	size of the soundfile which can be read in via inlet
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

PurpleGrain

Returns

grain

Definition at line 37 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
00025 #define OVERLAP_DENSITY = 8
00026
00037 grain grain_new(int grain_size_samples, int soundfile_size, float start_pos, int grain_index, float
      time_stretch_factor)
00038 {
00039     grain x;
00040     //grain *next_grain = NULL;
00041     //grain *previous_grain = NULL;
00042     x.grain_active = false;
00043     x.grain_size_samples = grain_size_samples;
00044     x.grain_index = grain_index;
00045     x.internal_step_count = 0;
00046     x.time_stretch_factor = time_stretch_factor;
00047     bool reverse_playback = x.time_stretch_factor < 0.0;
00048
00049
00050     //x.start = fabsf(x.grain_size_samples * grain_index * x.time_stretch_factor);
00051     x.start = start_pos;
00052     if(x.start < 0) x.start += (soundfile_size - 1);
00053     x.end = x.start + ((x.grain_size_samples - 1) * x.time_stretch_factor);
00054
00055     if(x.end < 0) x.end += soundfile_size - 1;
00056     if(x.end > soundfile_size - 1) x.end -= (soundfile_size - 1);
00057
00058     /*
00059     if(time_stretch_factor < 0.0)
00060     {
00061         switch_float_values(&x.start, &x.end);
00062     }
00063      */
00064
00065     x.current_sample_pos = x.start;
00066     x.next_sample_pos = x.current_sample_pos + x.time_stretch_factor;
00067
00068     if(reverse_playback)
00069     {
00070         if(x.next_sample_pos < 0) x.next_sample_pos += (soundfile_size - 1);
00071         if(x.next_sample_pos < x.end && x.start > x.end) x.next_sample_pos = x.end;
00072
00073     }
00074     else
00075     {
00076         if(x.next_sample_pos > (soundfile_size - 1)) x.next_sample_pos -= (soundfile_size - 1);
00077         if(x.next_sample_pos >= x.end && x.start < x.end) x.next_sample_pos = x.end;
00078     }
00079
00080     return x;
00081 }
00088 void grain_internal_scheduling(grain* g, c_granular_synth* synth)
00089 {
00090     if(synth->reverse_playback)
00091     {
00092         // ???
00093         //g->grain_active = ((int)g->start == synth->current_start_pos) ||
00094         g->grain_active = g->grain_index == synth->current_grain_index ||
00095         ((((synth->soundfile_length - 1 - synth->playback_position) <= g->start) &&
00096           ((synth->soundfile_length - 1 - synth->playback_position) >= g->end)));
00097         /*
00098         (((g->start * synth->time_stretch_factor * -1) >= synth->playback_position) &&
00099         ((g->end * synth->time_stretch_factor * -1) <= (synth->playback_position *
      synth->time_stretch_factor * -1)));
00100          */
00101     }
00102     else
00103     {
00104         //g->grain_active = ((int)g->start == synth->current_start_pos) ||
00105         g->grain_active = g->grain_index == synth->current_grain_index ||
00106         ((g->start <= synth->playback_position) &&
```

```
00107           (g->end >= synth->playback_position));
00108     }
00109
00110     if(g->grain_active)
00111     {
00112         float   left_sample,
00113                 right_sample,
00114                 frac,
00115                 integral,
00116                 weighted;
00117
00118
00119         // For negative time_stretch_factor values read samples in backwards direction
00120         left_sample = synth->soundfile_table[(int)floorf(g->current_sample_pos)];
00121         right_sample = synth->soundfile_table[(int)ceilf(g->current_sample_pos)];
00122         frac = modff(g->current_sample_pos, &integral);
00123         weighted = get_interpolated_sample_value(left_sample, right_sample,frac);
00124         synth->output_buffer += weighted;
00125         g->current_sample_pos = g->next_sample_pos;
00126         g->next_sample_pos += synth->pitch_factor;
00127         // does the next index exceed the soundfile length? (Forward Playback)
00128         if(g->next_sample_pos > (synth->soundfile_length - 1))
00129         {
00130             g->next_sample_pos -= (synth->soundfile_length - 1);
00131         }
00132         // Or does it go negatively past 0 (Reverse Playback)
00133         if(g->next_sample_pos < 0.0)
00134         {
00135             g->next_sample_pos += (synth->soundfile_length - 1);
00136         }
00137         g->internal_step_count++;
00138
00139         /*
00140         if((!synth->reverse_playback && g->current_sample_pos >= g->end)
00141            || (synth->reverse_playback && g->current_sample_pos <= g->end)
00142            || g->next_sample_pos > synth->soundfile_length - 1
00143            || g->next_sample_pos < 0.0)
00144          */
00145         if(g->internal_step_count >= g->grain_size_samples)
00146         {
00147             //g->grain_active = false;
00148             // Grain wieder auf seinen Startpunkt setzen, wie bei Initialisierung in new-methode
00149             g->current_sample_pos = g->start;
00150             g->next_sample_pos = g->current_sample_pos + synth->pitch_factor;
00151             g->internal_step_count = 0;
00152             synth->spray_true_offset = 0;
00153             c_granular_synth_reset_playback_position(synth);
00154             //synth->playback_position = synth->current_start_pos;
00155         }
00156
00157         // checken ob nächstes grain aktiv ist
00158         if(g->next_grain)
00159         {
00160             grain_internal_scheduling(g->next_grain, synth);
00161         }
00162
00163     }
00164     else {
00165         // Grain nicht oder nicht mehr aktiv
00166         // seine current pos auf seinen start zurücksetzen
00167         g->current_sample_pos = g->start;
00168         g->next_sample_pos = g->current_sample_pos + synth->pitch_factor;
00169         g->internal_step_count = 0;
00170         /*
00171         g->current_sample_pos = g->grain_size_samples * g->grain_index * g->time_stretch_factor;
00172         g->next_sample_pos = g->current_sample_pos + g->time_stretch_factor;
00173          */
00174         return;
00175
00176     }
00177 }
00183 void grain_free(grain *x)
00184 {
00185     free(x);
00186 }
```

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

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_fre object

Definition at line 183 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 samples contained in a grain
soundfile_size	size of the soundfile which can be read in via inlet
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 37 of file grain.c.

4.12 grain.h

```
00001
00011 #ifndef grain_h
00012 #define grain_h
00013
00014 #include "m_pd.h"
00015
00016 #include <stdio.h>
00017 #include <stdlib.h>
00018 #include <math.h>
00019 #include <stdbool.h>
00020
00021 #ifdef __cplusplus
00022 extern "C" {
00023 #endif
00024
00030 typedef struct grain
00031 {
00032     struct grain        *next_grain,
00033                         *previous_grain;
00034     t_int               grain_size_samples,
00035                         grain_index,
00036                         internal_step_count;
00037     t_float             start,
00038                         end,
00039                         time_stretch_factor,
00040                         current_sample_pos,
00041                         next_sample_pos;
00042     bool                grain_active;
00043
00044 } grain;
00045
00051 grain grain_new(int grain_size_samples, int soundfile_size, float start_pos, int grain_index, float
       time_stretch_factor);
00052
00053
00059 void grain_free(grain *x);
00060
00061 #ifdef __cplusplus
00062 }
00063 #endif
00064
00065 #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 <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)

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

Kretschmar, Nikita
Philipp, Adrian
Strobl, Micha
Wennemann,Tim

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

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 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

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 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 *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 75 of file purple_utils.c.

4.14 purple_utils.c

```
00001
00016 #include <stdio.h>
00017 #include <stdlib.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 }
00082
00083 int spray_dependant_playback_nudge(int spray_input)
00084 {
00085     if(spray_input == 0) return 0;
00086     int off = rand() % (2 * spray_input);
00087     return off - spray_input;
00088 }
```

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)

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

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 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

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 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 *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 75 of file purple_utils.c.

4.16 purple_utils.h

```
00001
00015 #ifndef purple_utils_h
00016 #define purple_utils_h
00017
00018 int get_samples_from_ms(int ms, float sr);
00019 float get_ms_from_samples(int num_samples, float sr);
00020 float get_interpolated_sample_value(float sample_left, float sample_right, float frac);
00021 void switch_float_values(float *a, float *b);
00022 int spray_dependant_playback_nudge(int spray_input);
00023
00024 #endif
```