

disco

Claude Heiland-Allen

2019

Contents

1	disco-designer.c	2
2	.gitignore	13
3	index.html	13
4	LICENSE.md	15
5	Makefile	19
6	markov-analysis.c	19
7	markov-synthesis.c	25
8	normalize.c	30
9	README.md	31
10	rhythm-analysis.c	32
11	rhythm-synthesis.c	37
12	timbre-stamp.c	44

1 disco-designer.c

```
#include <SDL2/SDL.h>
#include <SDL2/SDL_audio.h>
#include <SDL2/SDL_opengl.h>

5  #ifdef __EMSCRIPTEN__
#include <emscripten.h>
#else
#include <unistd.h>
#endif
10
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
15  #include <tgmath.h>
#include <time.h>

#define OCTAVES 11
#define OCTAVESIZE (1 << (OCTAVES - 1))
20
#define RHYTHMS 11
#define RHYTHMSIZE (1 << (RHYTHMS - 1))

#define OVERLAP 4
25
#define OHOP (OCTAVESIZE / OVERLAP)
#define RHOP (RHYTHMSIZE / OVERLAP)
```

```
#define CHANNELS 2
30
#define CELLSIZE 64
#define FPS 60

static struct
35 {
    float owindow[OCTAVESIZE]; // octave window
    float rwindow[RHYTHMSIZE]; // rhythm window
    float nbuffer[OCTAVESIZE][CHANNELS]; // source audio block
    float obuffer[OCTAVESIZE][CHANNELS]; // output audio block
40
    float ebuffer[RHYTHMSIZE][OCTAVES]; // source rhythm history
    float rbuffer[RHYTHMSIZE][OCTAVES]; // output rhythm history
    float abuffer[OHOP][CHANNELS]; // source audio
    float R[OCTAVES][RHYTHMS]; // fingerprint
    bool normalize_carrier;
45
    int eindex; // source rhythm history index
    int rindex; // output rhythm history index
    int aindex; // source audio index
    int oindex; // output audio index
    int oavailable; // output audio buffer fill state
50
    int SR; // sample rate
    // temporary data follows
    float haar[2][OCTAVESIZE][CHANNELS];
    double E[OCTAVES];
    float rhaar[2][RHYTHMSIZE];
55
    // video state
    SDL_Window* window;
    SDL_GLContext context;
    int target_octave;
    int target_rhythm;
60
    double target_delta;
} state;

static bool read_fingerprint(const char *filename)
65
{
    FILE *ifile = fopen(filename, "rb");
    if (ifile)
    {
        bool ok = true;
        for (int i = 0; i < OCTAVES; ++i)
70
        {
            for (int j = 0; j < RHYTHMS; ++j)
            {
                double r = 0;
                ok &= (1 == fscanf(ifile, "%lf ", &r));
                state.R[i][j] = r;
            }
        }
        fclose(ifile);
80
        return ok;
    }
    else
    {
        return false;
85 }
```

```
}

static void normalize_fingerprint(void)
{
    double sr = 0;
    for (int i = 0; i < OCTAVES; ++i)
    {
        double r = state.R[i][0];
        sr += r * r;
    }
    sr /= OCTAVES;
    sr = sqrt(sr);
    for (int i = 0; i < OCTAVES; ++i)
    {
        state.R[i][0] /= sr;
    }
#if 0
    sr = 0;
    for (int i = 0; i < OCTAVES; ++i)
    {
        for (int j = 1; j < RHYTHMS; ++j)
        {
            double r = state.R[i][j];
            sr += r * r;
        }
    }
    sr /= OCTAVES * (RHYTHMS - 1);
    sr = sqrt(sr);
#endif
    for (int i = 0; i < OCTAVES; ++i)
    {
        for (int j = 1; j < RHYTHMS; ++j)
        {
            state.R[i][j] /= sr;
        }
    }
}

static void initialize_windows(void)
{
    for (int i = 0; i < OCTAVESIZE; ++i)
    {
        state.owindow[i] = 0.5 - 0.5 * cos(i * 6.283185307179586 / OCTAVESIZE);
    }
    for (int i = 0; i < RHYTHMSIZE; ++i)
    {
        state.rwindow[i] = 0.5 - 0.5 * cos(i * 6.283185307179586 / RHYTHMSIZE);
    }
}

static void audio1(float output[CHANNELS])
{
    while (state.oavailable <= 0)
    {
        // generate noise
        for (int channel = 0; channel < CHANNELS; ++channel)
        {
```

```

state.abuffer[state.aindex][channel] = 2 * (rand() / (double) RAND_MAX - 0.5);
}
state.aindex++;
if (state.aindex == OHOP)
{
    state.aindex = 0;
    for (int i = 0; i < OCTAVESIZE - OHOP; ++i)
    {
        for (int channel = 0; channel < CHANNELS; ++channel)
        {
            state.nbbuffer[i][channel] = state.nbbuffer[i + OHOP][channel];
        }
    }
    for (int i = OCTAVESIZE - OHOP; i < OCTAVESIZE; ++i)
    {
        for (int channel = 0; channel < CHANNELS; ++channel)
        {
            state.nbbuffer[i][channel] = state.abuffer[i - (OCTAVESIZE - OHOP)][channel];
        }
    }
}

// window
int src = 0;
int dst = 1;
for (int i = 0; i < OCTAVESIZE; ++i)
{
    for (int channel = 0; channel < CHANNELS; ++channel)
    {
        state.haar[src][i][channel] = state.nbbuffer[i][channel] * state.owindow[i];
    }
}

// compute Haar wavelet transform
for (int length = OCTAVESIZE >> 1; length > 0; length >>= 1)
{
    for (int i = 0; i < length; ++i)
    {
        for (int channel = 0; channel < CHANNELS; ++channel)
        {
            float a = state.haar[src][2 * i + 0][channel];
            float b = state.haar[src][2 * i + 1][channel];
            float s = (a + b) / 2;
            float d = (a - b) / 2;
            state.haar[dst][i][channel] = s;
            state.haar[dst][length + i][channel] = d;
        }
    }
    for (int i = 0; i < OCTAVESIZE; ++i)
    {
        for (int channel = 0; channel < CHANNELS; ++channel)
        {
            state.haar[src][i][channel] = state.haar[dst][i][channel];
        }
    }
}

```

```

    }

// compute energy per octave
200 memset(state.E, 0, sizeof(state.E));
for (int channel = 0; channel < CHANNELS; ++channel)
{
    state.E[0] += fabsf(state.haar[src][0][channel]); // DC
}
205 for (int octave = 1, length = 1
      ; length <= OCTAVESIZE >> 1
      ; octave += 1, length <<= 1
      )
{
    double rms = 0;
    for (int channel = 0; channel < CHANNELS; ++channel)
    {
        for (int i = 0; i < length; ++i)
        {
            double a = state.haar[src][length + i][channel];
            rms += a * a;
        }
    }
    rms /= length * CHANNELS;
215 rms = sqrt(fmax(rms, 0));
    state.E{octave} = rms;
}

// copy to ebuffer
220 for (int octave = 0; octave < OCTAVES; ++octave)
{
    state.ebuffer[state.eindex][octave] = state.E{octave};
}
state.eindex++;

230 if (state.eindex == RHYTHMSIZE)
{
    for (int octave = 0; octave < OCTAVES; ++octave)
    {
235     // window
        for (int i = 0; i < RHYTHMSIZE; ++i)
        {
            state.rhaar[src][i] = state.ebuffer[i][octave];// * rwindow[i];
        }
}

// compute Haar wavelet transform
240 for (int length = RHYTHMSIZE >> 1; length > 0; length >>= 1)
{
    for (int i = 0; i < length; ++i)
    {
        float a = state.rhaar[src][2 * i + 0];
        float b = state.rhaar[src][2 * i + 1];
        float s = (a + b) / 2;
        float d = (a - b) / 2;
245        state.rhaar[dst][i] = s;
        state.rhaar[dst][length + i] = d;
    }
}

```

```

255         for ( int i = 0; i < RHYTHMSIZE; ++i)
256         {
257             state.rhaar[src][i] = state.rhaar[dst][i];
258         }
259
260         // normalize carrier
261         if (state.normalize_carrier)
262         {
263             state.rhaar[src][0] = 1;
264             for ( int rhythm = 1, length = 1
265                  ; length <= RHYTHMSIZE >> 1
266                  ; rhythm += 1, length <= 1
267                  )
268             {
269                 double rms = 0;
270                 for ( int i = 0; i < length; ++i)
271                 {
272                     double d = state.rhaar[src][length + i];
273                     rms += d * d;
274                 }
275                 rms /= length;
276                 rms = sqrt(rms);
277                 if (rms > 0)
278                 {
279                     for ( int i = 0; i < length; ++i)
280                     {
281                         state.rhaar[src][length + i] /= rms;
282                     }
283                 }
284             }
285         }
286
287         // amplify by control data
288         state.rhaar[src][0] *= state.R{octave}[0];
289         for ( int rhythm = 1, length = 1
290               ; length <= RHYTHMSIZE >> 1
291               ; rhythm += 1, length <= 1
292               )
293         {
294             for ( int i = 0; i < length; ++i)
295             {
296                 state.rhaar[src][length + i] *= state.R{octave}[rhythm];
297             }
298         }
299
300         // compute inverse Haar wavelet transform
301         for ( int length = 1; length <= RHYTHMSIZE >> 1; length <= 1)
302         {
303             for ( int i = 0; i < RHYTHMSIZE; ++i)
304             {
305                 state.rhaar[dst][i] = state.rhaar[src][i];
306             }
307             for ( int i = 0; i < length; ++i)
308             {
309                 float s = state.rhaar[dst][i];
310                 float d = state.rhaar[dst][length + i];

```

```

            float a = s + d;
            float b = s - d;
            state.rhaar[src][2 * i + 0] = a;
            state.rhaar[src][2 * i + 1] = b;
315        }
    }

    // window
    for (int i = 0; i < RHYTHMSIZE; ++i)
320    {
        state.rhaar[src][i] *= state.rwindow[i];
    }

    // overlap-add
325    for (int i = 0; i < RHYTHMSIZE; ++i)
    {
        state.rbuffer[i][octave]
            = i + RHOP < RHYTHMSIZE
            ? state.rbuffer[i + RHOP][octave]
330            : 0;
    }
    for (int i = 0; i < RHYTHMSIZE; ++i)
    {
        state.rbuffer[i][octave] += state.rhaar[src][i];
    }

335} // for octave

    // shunt
340    for (int i = 0; i < RHYTHMSIZE - RHOP; ++i)
    {
        for (int octave = 0; octave < OCTAVES; ++octave)
        {
            state.ebuffer[i][octave] = state.ebuffer[i + RHOP][octave];
        }
    }
    for (int i = RHYTHMSIZE - RHOP; i < RHYTHMSIZE; ++i)
345    {
        for (int octave = 0; octave < OCTAVES; ++octave)
        {
            state.ebuffer[i][octave] = 0;
        }
    }

350    state.eindex -= RHOP;
    state.rindex = 0;
} // if eindex == RHYTHMSIZE

    // copy
360    for (int octave = 0; octave < OCTAVES; ++octave)
    {
        state.E[octave] = state.rbuffer[state.rindex][octave];
    }
    state.rindex++;
365

    // amplify octaves
    for (int channel = 0; channel < CHANNELS; ++channel)

```

```

    {
        state.haar[src][0][channel] *= state.E[0]; // DC
    }
    for ( int octave = 1, length = 1
          ; length <= OCTAVESIZE >> 1
          ; octave += 1, length <= 1
    )
375    {
        for ( int channel = 0; channel < CHANNELS; ++channel)
        {
            for ( int i = 0; i < length; ++i)
            {
                state.haar[src][length + i][channel] *= state.E{octave};
            }
        }
    }

385    // compute inverse Haar wavelet transform
    for ( int length = 1; length <= OCTAVESIZE >> 1; length <= 1)
    {
        for ( int i = 0; i < OCTAVESIZE; ++i)
        {
390            for ( int channel = 0; channel < CHANNELS; ++channel)
            {
                state.haar[dst][i][channel] = state.haar[src][i][channel];
            }
        }
        for ( int i = 0; i < length; ++i)
        {
            for ( int channel = 0; channel < CHANNELS; ++channel)
            {
                float s = state.haar[dst][           i][channel];
                float d = state.haar[dst][length + i][channel];
                float a = s + d;
                float b = s - d;
                state.haar[src][2 * i + 0][channel] = a;
                state.haar[src][2 * i + 1][channel] = b;
            }
        }
    }

410    // window
    for ( int i = 0; i < OCTAVESIZE; ++i)
    {
        for ( int channel = 0; channel < CHANNELS; ++channel)
        {
            state.haar[src][i][channel] *= state.owindow[i];
        }
    }

415    // overlap-add
    for ( int i = 0; i < OCTAVESIZE; ++i)
    {
        for ( int channel = 0; channel < CHANNELS; ++channel)
        {
            state.obuffer[i][channel]
                = i + OHOP < OCTAVESIZE
        }
    }

```

```
425         ? state . obuffer [ i + OHOP ] [ channel ]
        : 0;
    }
}
for ( int i = 0; i < OCTAVESIZE; ++i)
{
    for ( int channel = 0; channel < CHANNELS; ++channel)
    {
        state . obuffer [ i ] [ channel ] += state . haar [ src ] [ i ] [ channel ];
    }
}
435
state . oavailable += OHOP;
state . oindex = 0;
} // if aindex == OHOP
440 } // while oavailable <= 0

for ( int channel = 0; channel < CHANNELS; ++channel)
{
    output [ channel ] = state . obuffer [ state . oindex ] [ channel ];
445
state . oindex++;
state . oavailable--;
}

450 static void audio ( void *userdata , Uint8 *stream , int len )
{
    ( void ) userdata ;
    float *b = ( float * ) stream ;
    int m = len / sizeof ( float ) / CHANNELS;
455
    int k = 0;
    for ( int i = 0; i < m; ++i)
    {
        float out [ CHANNELS ];
        audio1 ( out );
460
        for ( int j = 0; j < CHANNELS; ++j)
        {
            b [ k ++ ] = out [ j ];
        }
    }
465 }

static bool interact ( void )
{
    bool running = true;
470
    SDL_Event event;
    while ( SDL_PollEvent ( &event ) != 0 )
    {
        switch ( event . type )
        {
475
            case SDL_QUIT:
            {
                running = false;
                break;
            }
480
        }
    }
}
```

```

    int x = -1, y = -1;
    Uint32 b = SDL_GetMouseState(&x, &y);
    state.target_octave = x / CELLSIZE + 1;
    state.target_rhythm = RHYTHMS - 1 - y / CELLSIZE;
    state.target_delta = 2 * (0.5 - ((y % CELLSIZE) + 0.5) / CELLSIZE);
485   if ( (b & SDL_BUTTON(SDL_BUTTON_LEFT)) &&
        1 <= state.target_octave && state.target_octave < OCTAVES &&
        1 <= state.target_rhythm && state.target_rhythm < RHYTHMS &&
490   -1 <= state.target_delta && state.target_delta <= 1
      )
    {
      double r = state.R[state.target_octave][state.target_rhythm];
      r += state.target_delta / FPS;
495   r = fmin(fmax(r, 0), 0.1);
      state.R[state.target_octave][state.target_rhythm] = r;
    }
    return running;
}
500
static void video1(void)
{
  glClearColor(1, 1, 1, 1);
  glClear(GL_COLOR_BUFFER_BIT);
505   glBegin(GL_QUADS);
  {
    for (int i = 1; i < OCTAVES; ++i)
    {
      for (int j = 1; j < RHYTHMS; ++j)
      {
        double radius = fmin(fmax(sqrt(state.R[i][j] / 0.1) / 2.0, 1.0 / √
          ↴ CELLSIZE), 0.5);
        const int sx[4] = { -1, -1, 1, 1 };
        const int sy[4] = { -1, 1, 1, -1 };
        for (int k = 0; k < 4; ++k)
510       {
          glColor4f(0, 0, 0, 1);
          glVertex4f
            ( ((i - 0.5 + sx[k] * radius) / (OCTAVES - 1) - 0.5) * 2.0
            , ((j - 0.5 + sy[k] * radius) / (RHYTHMS - 1) - 0.5) * 2.0
515           , 0
           , 1
            );
        }
      }
    }
520   glEnd();
  SDL_GL_SwapWindow(state.window);
}
525
void video(void)
{
  while (interact())
  {
    video1();
535   }
}

```

```
540 void main1( void )
541 {
542     if ( interact() )
543     {
544         video1();
545     }
546 }
547
548 int main( int argc , char **argv )
549 {
550     printf("disco/designer (Free Art License 1.3+) 2019 Claude Heiland-Allen\n");
551     srand( time(0) );
552     memset( &state , 0 , sizeof(state));
553     // load initial state
554     if ( argc > 1 )
555     {
556         if ( ! read_fingerprint(argv[1]) )
557         {
558             printf("error: could not read fingerprint\n");
559             return 1;
560         }
561         if ( 0 )
562             normalize_fingerprint();
563     }
564     state.normalize_carrier = 1;
565     initialize_windows();
566     // initialize SDL2
567     SDL_Init( SDL_INIT_AUDIO | SDL_INIT_VIDEO );
568     SDL_AudioSpec want , have ;
569     want.freq = 44100;
570     want.format = AUDIO_F32;
571     want.channels = CHANNELS;
572     want.samples = 4096;
573     want.callback = audio;
574     SDL_AudioDeviceID dev = SDL_OpenAudioDevice( NULL , 0 , &want , &have , ↴
575         ↴ SDL_AUDIO_ALLOW_ANY_CHANGE );
576     if ( have.freq > 192000 || have.format != AUDIO_F32 || have.channels != ↴
577         ↴ CHANNELS )
578     {
579         printf("want: %d %d %d %d\n" , want.freq , want.format , want.channels , want. ↴
580             ↴ samples );
581         printf("have: %d %d %d %d\n" , have.freq , have.format , have.channels , have. ↴
582             ↴ samples );
583         printf("error: bad audio parameters\n");
584     }
585     else
586     {
587         SDL_GL_SetAttribute( SDL_GL_CONTEXT_MAJOR_VERSION , 2 );
588         SDL_GL_SetAttribute( SDL_GL_CONTEXT_MINOR_VERSION , 1 );
589         state.window = SDL_CreateWindow("disco/designer" , SDL_WINDOWPOS_CENTERED , ↴
590             ↴ SDL_WINDOWPOS_CENTERED , ( OCTAVES - 1 ) * CELLSIZE , ( RHYTHMS - 1 ) * ↴
591                 ↴ CELLSIZE , SDL_WINDOW_OPENGL | SDL_WINDOW_SHOWN );
592         if ( ! state.window )
593         {
594             printf("error: couldn't create SDL2 window\n");
595             return 1;
596         }
597     }
598 }
```

```

590     }
591     state.context = SDL_GL_CreateContext(state.window);
592     if (!state.context)
593     {
594         printf(" error: couldn't create OpenGL context\n");
595         return 1;
596     }
597     state.SR = have.freq;
598     // start audio processing
599     SDL_PauseAudioDevice(dev, 0);
600 #ifdef __EMSCRIPTEN__
601     emscripten_set_main_loop(main1, 0, 1);
602 #else
603     video();
604 #endif
605     }
606     SDL_DestroyWindow(state.window);
607     state.window = NULL;
608     SDL_Quit();
609     return 0;
610 }
```

2 .gitignore

```

timbre-stamp
markov-analysis
markov-synthesis
rhythm-analysis
5 rhythm-synthesis
normalize
disco-designer
designer.html
designer.js
10 designer.js.gz
designer.wasm
designer.wasm.gz
*.dat
*.ogg
15 *.wav
-
```

3 index.html

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
    <title>disco/designer</title>
    <meta name="description" content="disco/designer" />
    <meta name="keywords" content="music noise" />
    <meta name="generator" content="emscripten" />
    <meta name="author" content="mathr" />
    <style>
      canvas
      {
        width: 704px;
```

```
15      height: 832px;
    position: absolute;
    top: 0;
    bottom: 0;
    left: 0;
    right: 0;
    border: 0;
    padding: 0;
    margin: auto;
}
</style>
</head>
<body>
<canvas id="canvas" oncontextmenu="event.preventDefault()" tabindex="-1"></canvas>
<script>
// audio autoplay
const audioContextList = [];
(function() {
    self.AudioContext = new Proxy(self.AudioContext, {
        construct(target, args) {
            const result = new target(...args);
            audioContextList.push(result);
            return result;
        }
    });
})();
function resumeAudio() {
    audioContextList.forEach(ctx => {
        if (ctx.state !== "running") { ctx.resume(); }
    });
}
[ "click", "contextmenu", "auxclick", "dblclick",
  "mousedown", "mouseup", "pointerup", "touchend",
  "keydown", "keyup"
].forEach(name => document.addEventListener(name, resumeAudio));
// emscripten
var Module
    = { preRun: [],
        postRun: [],
        print: function(e) {
            1<arguments.length&&(e=Array.prototype.slice.call(arguments).join(" "));
            console.log(e);
        },
        printErr: function(e) {
            1<arguments.length&&(e=Array.prototype.slice.call(arguments).join(" "));
            console.error(e)
        },
        canvas: function() {
            var e = document.getElementById("canvas");
            e.addEventListener("webglcontextlost", function(e) {
                alert("WebGL context lost. You will need to reload the page.");
                e.preventDefault();
            });
        }
    };

```

```
        return e;
    }()
}
;
</script>
<script async src="designer.js"></script>
</body>
</html>
```

4 LICENSE.md

```
# Free Art License 1.3 (FAL 1.3)
```

```
## Preamble
```

5 The Free Art License grants the right to freely copy, distribute, and transform creative works without infringing the author's rights.

10 The Free Art License recognizes and protects these rights. Their implementation has been reformulated in order to allow everyone to use creations of the human mind in a creative manner, regardless of their types and ways of expression.

15 While the public's access to creations of the human mind usually is restricted by the implementation of copyright law, it is favoured by the Free Art License. This license intends to allow the use of a work's resources; to establish new conditions for creating in order to increase creation opportunities. The Free Art License grants the right to use a work, and acknowledges the right holder's and the user's rights and responsibility.

20 The invention and development of digital technologies, Internet and Free Software have changed creation methods: creations of the human mind can obviously be distributed, exchanged, and transformed. They allow to produce common works to which everyone can contribute to the benefit of all.

25 The main rationale for this Free Art License is to promote and protect these creations of the human mind according to the principles of copyleft: freedom to use, copy, distribute, transform, and prohibition of exclusive appropriation.

```
## Definitions
```

30 *work* either means the initial work, the subsequent works or the common work as defined hereafter:

35 *common work* means a work composed of the initial work and all subsequent contributions to it (originals and copies). The initial author is the one who, by choosing this license, defines the conditions under which contributions are made.

40 *Initial work* means the work created by the initiator of the common work (as defined above), the copies of which can be modified by whoever wants to

45 *Subsequent works* means the contributions made by authors who

participate in the evolution of the common work by exercising the rights to reproduce , distribute , and modify that are granted by the license.

50 *Originals* (sources or resources of the work) means all copies of either the initial work or any subsequent work mentioning a date and used by their author(s) as references for any subsequent updates , interpretations , copies or reproductions .

55 *Copy* means any reproduction of an original as defined by this license .

1. OBJECT

60 The aim of this license is to define the conditions under which one can use this work freely .

2. SCOPE

65 This work is subject to copyright law . Through this license its author specifies the extent to which you can copy , distribute , and modify it .

2.1 FREEDOM TO COPY (OR TO MAKE REPRODUCTIONS)

70 You have the right to copy this work for yourself , your friends or any other person , whatever the technique used .

2.2 FREEDOM TO DISTRIBUTE, TO PERFORM IN PUBLIC

75 You have the right to distribute copies of this work ; whether modified or not , whatever the medium and the place , with or without any charge , provided that you :

- 80 - attach this license without any modification to the copies of this work or indicate precisely where the license can be found ,
 - specify to the recipient the names of the author(s) of the originals , including yours if you have modified the work ,
 - specify to the recipient where to access the originals (either initial or subsequent) .

85 The authors of the originals may , if they wish to , give you the right to distribute the originals under the same conditions as the copies .

2.3 FREEDOM TO MODIFY

90 You have the right to modify copies of the originals (whether initial or subsequent) provided you comply with the following conditions :

- 95 - all conditions in article 2.2 above , if you distribute modified copies ;
 - indicate that the work has been modified and , if it is possible , what kind of modifications have been made ;
 - distribute the subsequent work under the same license or any compatible license .

100 The author(s) of the original work may give you the right to modify it under the same conditions as the copies .

3. RELATED RIGHTS

105 Activities giving rise to author's rights and related rights shall not challenge the rights granted by this license.

For example, this is the reason why performances must be subject to the same license or a compatible license. Similarly, integrating the work in
110 a database, a compilation or an anthology shall not prevent anyone from using the work under the same conditions as those defined in this license.

4. INCORPORATION OF THE WORK

115 Incorporating this work into a larger work that is not subject to the Free Art License shall not challenge the rights granted by this license.

If the work can no longer be accessed apart from the larger work in
120 which it is incorporated, then incorporation shall only be allowed under the condition that the larger work is subject either to the Free Art License or a compatible license.

5. COMPATIBILITY

125 A license is compatible with the Free Art License provided:

- it gives the right to copy, distribute, and modify copies of the work including for commercial purposes and without any other restrictions
130 than those required by the respect of the other compatibility criteria;
- it ensures proper attribution of the work to its authors and access to previous versions of the work when possible;
- it recognizes the Free Art License as compatible (reciprocity);
- it requires that changes made to the work be subject to the same license or to a license which also meets these compatibility criteria.
135

6. YOUR INTELLECTUAL RIGHTS

This license does not aim at denying your author's rights in your
140 contribution or any related right. By choosing to contribute to the development of this common work, you only agree to grant others the same rights with regard to your contribution as those you were granted by this license. Conferring these rights does not mean you have to give up your intellectual rights.

7. YOUR RESPONSIBILITIES

The freedom to use the work as defined by the Free Art License (right to copy, distribute, modify) implies that everyone is responsible for
150 their own actions.

8. DURATION OF THE LICENSE

This license takes effect as of your acceptance of its terms. The act
155 of copying, distributing, or modifying the work constitutes a tacit agreement. This license will remain in effect for as long as the copyright which is attached to the work. If you do not respect the terms of this license, you automatically lose the rights that it confers.
160 If the legal status or legislation to which you are subject makes it impossible for you to respect the terms of this license, you may not

make use of the rights which it confers.

9. VARIOUS VERSIONS OF THE LICENSE

165 This license may undergo periodic modifications to incorporate improvements by its authors (instigators of the Copyleft Attitude movement) by way of new, numbered versions.
You will always have the choice of accepting the terms contained in the version under which the copy of the work was distributed to you, or
170 alternatively , to use the provisions of one of the subsequent versions .

10. SUB-LICENSING

175 Sub-licenses are not authorized by this license. Any person wishing to make use of the rights that it confers will be directly bound to the authors of the common work.

11. LEGAL FRAMEWORK

180 This license is written with respect to both French law and the Berne Convention for the Protection of Literary and Artistic Works.

USER GUIDE

185 #### How to use the Free Art License?

To benefit from the Free Art License , you only need to mention the following elements on your work:

190 [Name of the author , title , date of the work. When applicable , names of authors of the common work and , if possible , where to find the originals].

195 Copyleft: This is a free work , you can copy , distribute , and modify it under the terms of the Free Art License
 [<http://artlibre.org/licence/lal/en/>](http://artlibre.org/licence/lal/en/)

Why to use the Free Art License?

200 1. To give the greatest number of people access to your work.
2. To allow it to be distributed freely .
3. To allow it to evolve by allowing its copy , distribution , and transformation by others .
4. So that you benefit from the resources of a work when it is under the
205 Free Art License: to be able to copy , distribute or transform it freely .
5. But also , because the Free Art License offers a legal framework to disallow any misappropriation . It is forbidden to take hold of your work and bypass the creative process for one's exclusive possession .

210 #### When to use the Free Art License?

Any time you want to benefit and make others benefit from the right to copy , distribute and transform creative works without any exclusive
215 appropriation , you should use the Free Art License . You can for example use it for scientific , artistic or educational projects .

What kinds of works can be subject to the Free Art License?

220 The Free Art License can be applied to digital as well as physical
works.

You can choose to apply the Free Art License on any text , picture ,
sound , gesture , or whatever sort of stuff on which you have sufficient
author 's rights .

225 ### Historical background of this license :

It is the result of observing , using and creating digital technologies ,
free software , the Internet and art . It arose from the Copyleft

230 Attitude meetings which took place in Paris in 2000 . For the first
time , these meetings brought together members of the Free Software
community , artists , and members of the art world . The goal was to adapt
the principles of Copyleft and free software to all sorts of creations .

235 -----

<<http://www.artlibre.org>>
Copyleft Attitude , 2007 .

240 You can make reproductions and distribute this license verbatim
(without any changes) .

Translation : Jonathan Clarke , Benjamin Jean , Griselda Jung , Fanny Mourguet ,
Antoine Pitrou . Thanks to <<http://framalang.org>>

5 Makefile

```
EXES := timbre-stamp rhythm-analysis rhythm-synthesis markov-analysis markov-
       ↴ synthesis normalize

all: $(EXES)

5   clean:
      -rm $(EXES)

%: %.c
      gcc -O3 -march=native $< -o $@ -Wall -Wextra -pedantic -g `pkg-config --cflags --libs gsl` -lsndfile -lm

10  disco-designer: disco-designer.c
      gcc -O3 -march=native $< -o $@ -Wall -Wextra -pedantic -g `sdl2-config --cflags --libs` -lGL -lm

designer.html: disco-designer.c
15    emcc -O3 -Os -o designer.html disco-designer.c --closure 1 -s USE_SDL=2 \
           ↴ -s LEGACY_GEMULATION=1 -s GLFFP_ONLY=1
    gzip -9 -k -f designer.js
    gzip -9 -k -f designer.wasm
```

6 markov-analysis.c

```
// note: uses stack allocated arrays , may need 'ulimit -s unlimited'

#include <math.h>
```

```

#include <stdio.h>
5 #include <stdlib.h>
#include <string.h>

#include <sndfile.h>

10 #include <gsl/gsl_linalg.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>

#define OCTAVES 11
15 #define OCTAVESIZE (1 << (OCTAVES - 1))

#define OVERLAP 16

#define OHOP (OCTAVESIZE / OVERLAP)
20

#define SOMSIZE 16

int main(int argc, char **argv)
{
25   if (argc != 3)
   {
     fprintf
     ( stderr
     , "usage:\n"
     , "%s in.wav out.dat\n"
     , argv[0]
     );
     return 1;
   }
35   SF_INFO info;
   memset(&info, 0, sizeof(info));
   fprintf(stderr, "reading %s\n", argv[1]);
   SNDFILE *ifile = sf_open(argv[1], SFM.READ, &info);
   if (! ifile) abort();
40   const int CHANNELS = info.channels;

   float window[OCTAVESIZE];
   memset(window, 0, sizeof(window));
   for (int i = 0; i < OCTAVESIZE; ++i)
45   {
     window[i] = 0.5 - 0.5 * cos(i * 6.283185307179586 / OCTAVESIZE);
   }

   int Ecount = 0;
50   double Emean[OCTAVES];
   memset(Emean, 0, sizeof(Emean));
   double Ecov[OCTAVES][OCTAVES];
   memset(Ecov, 0, sizeof(Ecov));

55   double imap[SOMSIZE][SOMSIZE][OCTAVES];
   memset(imap, 0, sizeof(imap));

   gsl_rng_env_setup();
   gsl_rng *PRNG = gsl_rng_alloc(gsl_rng_default);
60   int step = 0;

```

```

int lasti = 0, lastj = 0;
double chain[SOMSIZE][SOMSIZE][SOMSIZE][SOMSIZE];
memset(chain, 0, sizeof(chain));
double total[SOMSIZE][SOMSIZE];
memset(total, 0, sizeof(total));

65   for (int pass = 0; pass < 4; ++pass)
{
    fprintf(stderr, "pass %d/4\n", pass + 1);
70   sf_seek(ifile, 0, SEEK_SET);
    float ibuffer[OCTAVESIZE][CHANNELS];
    memset(ibuffer, 0, sizeof(ibuffer));

75   if (pass == 2)
{
    // Cholesky decomposition of covariance matrix
    for (int i = 0; i < OCTAVES; ++i)
    {
        for (int j = 0; j < OCTAVES; ++j)
        {
            Ecov[i][j] /= Ecount;
            fprintf(stderr, "%g ", Ecov[i][j]);
        }
        fprintf(stderr, "\n");
    }
85   gsl_matrix_view cov = gsl_matrix_view_array(&Ecov[0][0], OCTAVES, OCTAVES) ↴
    ↵;
    gsl_linalg_cholesky_decomp1(&cov.matrix);
    // randomize SOM
    for (int i = 0; i < SOMSIZE; ++i)
90   {
        for (int j = 0; j < SOMSIZE; ++j)
        {
            // generate independent Gaussian vectors
            double d[OCTAVES];
            for (int k = 0; k < OCTAVES; ++k)
            {
                d[k] = gsl_ran_gaussian(PRNG, 1);
            }
            gsl_vector_view x = gsl_vector_view_array(d, OCTAVES);
100          // correlate them to the correct covariance matrix
            gsl_blas_dtrmv(CblasLower, CblasNoTrans, CblasNonUnit, &cov.matrix, &x ↴
                ↵.vector);
            for (int k = 0; k < OCTAVES; ++k)
            {
                imap[i][j][k] = Emean[k] / Ecount + d[k];
            }
105          }
        }
    step = 0;
}
110  int reading = OVERLAP;
    while (reading > 0)
    {
115      // read input

```

```

for (int i = 0; i < OCTAVESIZE - OHOP; ++i)
{
    for (int channel = 0; channel < CHANNELS; ++channel)
    {
        ibuffer [i][channel] = ibuffer [i + OHOP][channel];
    }
}
120 for (int i = OCTAVESIZE - OHOP; i < OCTAVESIZE; ++i)
{
    for (int channel = 0; channel < CHANNELS; ++channel)
    {
        ibuffer [i][channel] = 0;
    }
}
125 if (reading == OVERLAP)
{
    int r = sf_readf_float (ifile , &ibuffer [OCTAVESIZE - OHOP][0], OHOP);
    if (r != OHOP)
    {
        reading--;
    }
}
130 else
{
    reading--;
}
135
// window
float haar [2][OCTAVESIZE][CHANNELS];
140 memset(haar, 0, sizeof(haar));
int src = 0;
int dst = 1;
for (int i = 0; i < OCTAVESIZE; ++i)
{
    for (int channel = 0; channel < CHANNELS; ++channel)
    {
        haar [src][i][channel] = ibuffer [i][channel] * window[i];
    }
}
145
150
// compute Haar wavelet transform
for (int length = OCTAVESIZE >> 1; length > 0; length >>= 1)
{
    for (int i = 0; i < length; ++i)
    {
        for (int channel = 0; channel < CHANNELS; ++channel)
        {
            float a = haar[src][2 * i + 0][channel];
            float b = haar[src][2 * i + 1][channel];
            float s = (a + b) / 2;
            float d = (a - b) / 2;
            haar[dst][i][channel] = s;
            haar[dst][length + i][channel] = d;
        }
    }
}
155
160
165
170

```

```

    for (int channel = 0; channel < CHANNELS; ++channel)
    {
        haar[src][i][channel] = haar[dst][i][channel];
    }
}

180 // compute energy per octave
double E[OCTAVES];
memset(E, 0, sizeof(E));
for (int channel = 0; channel < CHANNELS; ++channel)
{
    E[0] += fabsf(haar[src][0][channel]); // DC
}
for (int octave = 1, length = 1
     ; length <= OCTAVESIZE >> 1
     ; octave += 1, length <= 1
     )
{
    double rms = 0;
    for (int channel = 0; channel < CHANNELS; ++channel)
    {
        for (int i = 0; i < length; ++i)
        {
            double a = haar[src][length + i][channel];
            rms += a * a;
        }
    }
    rms /= length * CHANNELS;
    rms = sqrt(fmax(rms, 0));
    E{octave} = rms;
}

205 if (pass == 0)
{
    // compute mean
    for (int octave = 0; octave < OCTAVES; ++octave)
    {
        Emean{octave} += E{octave};
    }
    Ecount += 1;
}
215 else if (pass == 1)
{
    // compute covariance
    for (int octave1 = 0; octave1 < OCTAVES; ++octave1)
    {
        for (int octave2 = 0; octave2 < OCTAVES; ++octave2)
        {
            Ecov{octave1}[octave2]
                += (E{octave1} - Emean{octave1} / Ecount)
                    * (E{octave2} - Emean{octave2} / Ecount);
        }
    }
}
225 else if (pass == 2)
{
}

```

```

230         // compute SOM

    // find best match
    int mini = 0;
    int minj = 0;
235    double mins = 1.0 / 0.0;
    for (int i = 0; i < SOMSIZE; ++i)
    {
        for (int j = 0; j < SOMSIZE; ++j)
        {
            double s = 0;
            for (int k = 0; k < OCTAVES; ++k)
            {
                double d = E[k] - imap[i][j][k];
                s += d * d;
            }
            if (s < mins)
            {
                mins = s;
                mini = i;
                minj = j;
            }
        }
    }

255    // update weights
    for (int i = 0; i < SOMSIZE; ++i)
    {
        for (int j = 0; j < SOMSIZE; ++j)
        {
            double di = (i - mini) * 1.0 / SOMSIZE;
            double dj = (j - minj) * 1.0 / SOMSIZE;
            // FIXME magic numbers to control learning rate
            double kernel = 0.1 * exp(-(di * di + dj * dj) * (step + 1.0) / ↴
                100000);
            for (int k = 0; k < OCTAVES; ++k)
            {
                imap[i][j][k] += kernel * (E[k] - imap[i][j][k]);
            }
        }
        step++;
    }
270    else if (pass == 3)
    {
        // compute Markov chain
275        // find best match
        int mini = 0;
        int minj = 0;
        double mins = 1.0 / 0.0;
        for (int i = 0; i < SOMSIZE; ++i)
        {
            for (int j = 0; j < SOMSIZE; ++j)
            {
                double s = 0;
                for (int k = 0; k < OCTAVES; ++k)

```

```

    {
        double d = E[k] - imap[i][j][k];
        s += d * d;
    }
290    if (s < mins)
    {
        mins = s;
        mini = i;
        minj = j;
295    }
    }
}
chain[lasti][lastj][mini][minj] += 1;
total[lasti][lastj] += 1;
300    lasti = mini;
    lastj = minj;
}
} // while reading
} // for pass
305 sf_close(ifile);

// output
fprintf(stderr, "writing %s\n", argv[2]);
FILE *ofile = fopen(argv[2], "wb");
310    for (int i = 0; i < SOMSIZE; ++i)
        for (int j = 0; j < SOMSIZE; ++j)
            for (int k = 0; k < OCTAVES; ++k)
                fprintf(ofile, "% .18e\n", imap[i][j][k]);
            for (int i = 0; i < SOMSIZE; ++i)
                for (int j = 0; j < SOMSIZE; ++j)
                    for (int ii = 0; ii < SOMSIZE; ++ii)
                        for (int jj = 0; jj < SOMSIZE; ++jj)
                            fprintf(ofile, "% .18e\n", chain[i][j][ii][jj] / total[i][j]);
            fclose(ofile);
320    fprintf(stderr, "total frames %d\n", Ecount);

    return 0;
}

```

7 markov-synthesis.c

```

// note: uses stack allocated arrays, may need 'ulimit -s unlimited'

#include <math.h>
#include <stdio.h>
5 #include <stdlib.h>
#include <string.h>

#include <sndfile.h>

10 // FIXME these defines must match markov-analysis.c

#define OCTAVES 11
#define OCTAVESIZE (1 << (OCTAVES - 1))

15 #define OVERLAP 16

```

```

#define OHOP (OCTAVESIZE / OVERLAP)

#define SOMSIZE 16
20
// FIXME should match analyzed source audio
#define SR 44100

int main(int argc, char **argv)
25
{
    if (argc != 3)
    {
        fprintf
        ( stderr
        , "usage:\n"
        " %s in.dat out.wav\n"
        , argv[0]
        );
        return 1;
35    }

    fprintf(stderr, "reading %s\n", argv[1]);
    FILE *dfile = fopen(argv[1], "rb");
    double imap[SOMSIZE][SOMSIZE][OCTAVES];
40
    int lasti = 0;
    int lastj = 0;
    int mins = 1.0 / 0.0;
    for (int i = 0; i < SOMSIZE; ++i)
        for (int j = 0; j < SOMSIZE; ++j)
45
    {
        double s = 0;
        for (int k = 0; k < OCTAVES; ++k)
        {
            double r = 0;
50
            fscanf(dfile, "%lf ", &r);
            imap[i][j][k] = r;
            s += r * r;
        }
        if (s < mins)
55
        {
            mins = s;
            lasti = i;
            lastj = j;
        }
60    }
    double chain[SOMSIZE][SOMSIZE][SOMSIZE][SOMSIZE];
    for (int i = 0; i < SOMSIZE; ++i)
        for (int j = 0; j < SOMSIZE; ++j)
            for (int ii = 0; ii < SOMSIZE; ++ii)
65
                for (int jj = 0; jj < SOMSIZE; ++jj)
                {
                    double r = 0;
                    fscanf(dfile, "%lf ", &r);
                    chain[i][j][ii][jj] = r;
70    }
    fclose(dfile);

const int CHANNELS = 2;

```

```

75     fprintf(stderr, "writing %s\n", argv[2]);
    SF_INFO outfo =
    {
        0
        , SR
        , CHANNELS
        , SF_FORMAT_WAV | SF_FORMAT_FLOAT
80        , 0
        , 0
    };
    SNDFILE *ofile = sf_open(argv[2], SFM_WRITE, &outfo);
    if (!ofile) abort();
85
    float nbuffer[OCTAVESIZE][CHANNELS];
    memset(nbuffer, 0, sizeof(nbuffer));

    float obuffer[OCTAVESIZE][CHANNELS];
    memset(obuffer, 0, sizeof(obuffer));

    float owindow[OCTAVESIZE];
    memset(owindow, 0, sizeof(owindow));
    for (int i = 0; i < OCTAVESIZE; ++i)
95    {
        owindow[i] = 0.5 - 0.5 * cos(i * 6.283185307179586 / OCTAVESIZE);
    }

    float abuffer[OHOP][CHANNELS];
    memset(abuffer, 0, sizeof(abuffer));

    // FIXME 10mins should be enough for testing?
    for (int duration = 0; duration < 10 * 60 * SR; duration += OHOP)
100    {
        // generate noise
        for (int i = 0; i < OCTAVESIZE - OHOP; ++i)
        {
            for (int channel = 0; channel < CHANNELS; ++channel)
            {
                nbuffer[i][channel] = nbuffer[i + OHOP][channel];
            }
        }
        for (int i = OCTAVESIZE - OHOP; i < OCTAVESIZE; ++i)
        {
            for (int channel = 0; channel < CHANNELS; ++channel)
            {
                nbuffer[i][channel] = rand() / (double) RANDMAX - 0.5;
            }
        }
115
120    // window
    float haar[2][OCTAVESIZE][CHANNELS];
    memset(haar, 0, sizeof(haar));
    int src = 0;
    int dst = 1;
    for (int i = 0; i < OCTAVESIZE; ++i)
    {
        for (int channel = 0; channel < CHANNELS; ++channel)
        {
            haar[src][i][channel] = nbuffer[i][channel] * owindow[i];
130

```

```

        }

    }

// compute Haar wavelet transform
135   for (int length = OCTAVESIZE >> 1; length > 0; length >= 1)
{
    for (int i = 0; i < length; ++i)
    {
        for (int channel = 0; channel < CHANNELS; ++channel)
        {
            float a = haar[src][2 * i + 0][channel];
            float b = haar[src][2 * i + 1][channel];
            float s = (a + b) / 2;
            float d = (a - b) / 2;
145            haar[dst][i][channel] = s;
            haar[dst][length + i][channel] = d;
        }
    }
    for (int i = 0; i < OCTAVESIZE; ++i)
    {
        for (int channel = 0; channel < CHANNELS; ++channel)
        {
            haar[src][i][channel] = haar[dst][i][channel];
        }
155    }
}

// Markov chain
160   double E[OCTAVES];
   int nexti = lasti;
   int nextj = lastj;
   double t = rand() / (double) RANDMAX;
   for (int i = 0; i < SOMSIZE; ++i)
{
165     for (int j = 0; j < SOMSIZE; ++j)
     {
         t -= chain[lasti][lastj][i][j];
         if (t <= 0)
         {
170             nexti = i;
             nextj = j;
             break;
         }
     }
175     if (t <= 0)
         break;
    }
    for (int k = 0; k < OCTAVES; ++k)
    {
180        E[k] = imap[nexti][nextj][k];
    }
   lasti = nexti;
   lastj = nextj;

185   // amplify octaves
   for (int channel = 0; channel < CHANNELS; ++channel)
{

```

```

        haar[src][0][channel] *= E[0]; // DC
    }
190   for ( int octave = 1, length = 1
        ; length <= OCTAVESIZE >> 1
        ; octave += 1, length <=> 1
        )
    {
195     for (int channel = 0; channel < CHANNELS; ++channel)
    {
        for (int i = 0; i < length; ++i)
        {
            haar[src][length + i][channel] *= E{octave};
        }
    }
}

// compute inverse Haar wavelet transform
205   for (int length = 1; length <= OCTAVESIZE >> 1; length <=> 1)
{
    for (int i = 0; i < OCTAVESIZE; ++i)
    {
        for (int channel = 0; channel < CHANNELS; ++channel)
        {
            haar[dst][i][channel] = haar[src][i][channel];
        }
    }
210   for (int i = 0; i < length; ++i)
{
        for (int channel = 0; channel < CHANNELS; ++channel)
        {
            float s = haar[dst][i][channel];
            float d = haar[dst][length + i][channel];
            float a = s + d;
            float b = s - d;
            haar[src][2 * i + 0][channel] = a;
            haar[src][2 * i + 1][channel] = b;
        }
    }
215 }

// window
220   for (int i = 0; i < OCTAVESIZE; ++i)
{
    for (int channel = 0; channel < CHANNELS; ++channel)
    {
        haar[src][i][channel] *= owindow[i];
    }
}
225 }

// overlap-add
230   for (int i = 0; i < OCTAVESIZE; ++i)
{
    for (int channel = 0; channel < CHANNELS; ++channel)
    {
        obuffer[i][channel] = i + OHOP < OCTAVESIZE ? obuffer[i + OHOP][channel] ↴
            ↴ : 0;
    }
}
235 
```

```

245     }
246     for ( int i = 0; i < OCTAVESIZE; ++i)
247     {
248         for ( int channel = 0; channel < CHANNELS; ++channel)
249         {
250             obuffer [ i ] [ channel ] += haar [ src ] [ i ] [ channel ];
251         }
252     }

253     // output
254     sf_writef_float ( ofile , &obuffer [ 0 ] [ 0 ] , OHOP );
255 } // for duration

sf_close ( ofile );
return 0;
}

```

8 normalize.c

```

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
5
#include <sndfile.h>

int main( int argc , char **argv )
{
10    if ( argc != 3 )
    {
        fprintf
        (
            stderr
            , "usage:\n"
            , "%s in.wav out.wav\n"
            , argv [ 0 ]
            );
        return 1;
    }
20    SF_INFO info ;
    memset(&info , 0 , sizeof(info));
    fprintf(stderr , "reading %s\n" , argv [ 1 ]);
    SNDFILE *ifile = sf_open(argv [ 1 ] , SFMREAD , &info );
    if ( ! ifile )
25    {
        return 1;
    }
    float *data = malloc( info . frames * info . channels * sizeof(*data));
    sf_readf_float(ifile , data , info . frames );
30    sf_close(ifile );
    double peak = 0.0;
    for ( size_t i = 0; i < ( size_t ) info . frames * info . channels ; ++i )
    {
        peak = fmax(peak , data [ i ] );
35    }
    fprintf(stderr , "peak %g\n" , peak );
    float gain = 1 / peak;
    for ( size_t i = 0; i < ( size_t ) info . frames * info . channels ; ++i )

```

```

40     {
41         data[ i ] *= gain;
42     }
43     SF_INFO outfo = { 0, info.samplerate, info.channels, SF_FORMAT_WAV | ↴
44         SF_FORMAT_FLOAT, 0, 0 };
45     SNDFILE *ofile = sf_open(argv[2], SFM_WRITE, &outfo);
46     sf_writef_float(ofile, data, info.frames);
47     sf_close(ofile);
48     return 0;
49 }
```

9 README.md

```

# disco

Audio fingerprint discrimination and resynthesis via Haar wavelets.

5 - <https://mathr.co.uk/disco>
- <https://code.mathr.co.uk/disco>
- <https://mathr.co.uk/disco/designer>

## dependencies
10 gcc, make, libsndfile, libgsl

for disco/designer native version: libSDL2, opengl

15 for disco/designer web version: emscripten

## build

make # native versions
20 make disco-designer # native version
make designer.html # web version

## examples

25 Timbre stamp white noise with energy per octave of a control input:

./timbre-stamp input.wav output.wav

Compute energy per octave (audio) per octave (rhythm) fingerprint:
30 ./rhythm-analysis input.wav output.dat

Resynthesize from fingerprint by stamping on white noise:

35 ./rhythm-synthesis input.dat output.wav

Compute energy per octave (audio) Markov chain via self-organizing map:

./markov-analysis input.wav output.dat
40 Resynthesize from Markov chain by stamping on white noise:

./markov-synthesis input.dat output.wav
```

```

45 Normalize an audio file to peak value 1.0:
    ./normalize input.wav output.wav

Run disco/designer interactive demo native version:
50 ./disco-designer

Run disco/designer interactive demo web version:
55 python -m SimpleHTTPServer 8080 & # wasm needs http(s) server
    sensible-browser http://localhost:8080/index.html

## bugs

60 Uses large stack-allocated arrays: if it crashes, try running this first:
    ulimit -s unlimited

## legal
65 Copyright (C) 2019 Claude Heiland-Allen <mailto:claude@mathr.co.uk>

Copyleft: This is a free work, you can copy, distribute, and
modify it under the terms of the Free Art License
70 <http://artlibre.org/licence/lal/en/>

```

10 rhythm-analysis.c

```

// note: uses stack allocated arrays, may need 'ulimit -s unlimited'

#include <math.h>
#include <stdio.h>
5 #include <stdlib.h>
#include <string.h>

#include <sndfile.h>

10 #define OCTAVES 11
#define OCTAVESIZE (1 << (OCTAVES - 1))

#define RHYTHMS 13
#define RHYTHMSIZE (1 << (RHYTHMS - 1))
15 #define OVERLAP 4

#define OHOP (OCTAVESIZE / OVERLAP)
#define RHOP (RHYTHMSIZE / OVERLAP)
20

int main(int argc, char **argv)
{
    if (argc != 3)
    {
25        fprintf
            ( stderr
            , "usage:\n"
            "    %s in.wav out.dat\n"

```

```

30         , argv[0]
31     );
32     return 1;
33 }
34 SF_INFO info;
35 memset(&info, 0, sizeof(info));
36 fprintf(stderr, "reading %s\n", argv[1]);
37 SNDFILE *ifile = sf_open(argv[1], SFM.READ, &info);
38 if (!ifile) abort();
39 const int CHANNELS = info.channels;

40 float ibuffer[OCTAVESIZE][CHANNELS];
41 memset(ibuffer, 0, sizeof(ibuffer));

42 float window[OCTAVESIZE];
43 memset(window, 0, sizeof(window));
44 for (int i = 0; i < OCTAVESIZE; ++i)
45 {
46     window[i] = 0.5 - 0.5 * cos(i * 6.283185307179586 / OCTAVESIZE);
47 }

48 float rwindow[RHYTHMSIZE];
49 memset(rwindow, 0, sizeof(rwindow));
50 for (int i = 0; i < RHYTHMSIZE; ++i)
51 {
52 #if 0
53     // FIXME needs two passes (for DC removal)
54     rwindow[i] = 0.5 - 0.5 * cos(i * 6.283185307179586 / RHYTHMSIZE);
55 #else
56     // FIXME use one pass without windowing for now
57     rwindow[i] = 1;
58 #endif
59 }

60 int eindex = 0;
61 float ebuffer[RHYTHMSIZE][OCTAVES];
62 memset(ebuffer, 0, sizeof(ebuffer));

63 float R[OCTAVES][RHYTHMS];
64 memset(R, 0, sizeof(R));
65 int R_count = 0;

66 int reading = OVERLAP;
67 while (reading > 0)
68 {

69     // read input
70     for (int i = 0; i < OCTAVESIZE - OHOP; ++i)
71     {
72         for (int channel = 0; channel < CHANNELS; ++channel)
73         {
74             ibuffer[i][channel] = ibuffer[i + OHOP][channel];
75         }
76     }
77     for (int i = OCTAVESIZE - OHOP; i < OCTAVESIZE; ++i)
78     {
79         for (int channel = 0; channel < CHANNELS; ++channel)
80 
```

```

    {
        ibuffer [ i ] [ channel ] = 0;
    }
}
90 if ( reading == OVERLAP )
{
    int r = sf_readf_float ( ifile , &ibuffer [ OCTAVESIZE - OHOP ] [ 0 ] , OHOP );
    if ( r != OHOP )
    {
95     reading --;
    }
}
else
{
100    reading --;
}

// window
float haar [ 2 ] [ OCTAVESIZE ] [ CHANNELS ];
105 memset ( haar , 0 , sizeof ( haar ) );
int src = 0;
int dst = 1;
for ( int i = 0; i < OCTAVESIZE; ++i )
{
110    for ( int channel = 0; channel < CHANNELS; ++channel )
    {
        haar [ src ] [ i ] [ channel ] = ibuffer [ i ] [ channel ] * window [ i ];
    }
}
115

// compute Haar wavelet transform
for ( int length = OCTAVESIZE >> 1; length > 0; length >>= 1 )
{
120    for ( int i = 0; i < length; ++i )
    {
        for ( int channel = 0; channel < CHANNELS; ++channel )
        {
            float a = haar [ src ] [ 2 * i + 0 ] [ channel ];
            float b = haar [ src ] [ 2 * i + 1 ] [ channel ];
125            float s = ( a + b ) / 2;
            float d = ( a - b ) / 2;
            haar [ dst ] [ i ] [ channel ] = s;
            haar [ dst ] [ length + i ] [ channel ] = d;
        }
    }
130    for ( int i = 0; i < OCTAVESIZE; ++i )
    {
        for ( int channel = 0; channel < CHANNELS; ++channel )
        {
            haar [ src ] [ i ] [ channel ] = haar [ dst ] [ i ] [ channel ];
        }
    }
}
135

// compute energy per octave
double E [ OCTAVES ];
memset ( E , 0 , sizeof ( E ) );

```

```

    for ( int channel = 0; channel < CHANNELS; ++channel )
    {
        E[0] += fabsf(haar[src][0][channel]); // DC
    }
    for ( int octave = 1, length = 1
          ; length <= OCTAVESIZE >> 1
          ; octave += 1, length <= 1
    )
    {
        double rms = 0;
        for ( int channel = 0; channel < CHANNELS; ++channel )
        {
            for ( int i = 0; i < length; ++i )
            {
                double a = haar[src][length + i][channel];
                rms += a * a;
            }
        }
        rms /= length * CHANNELS;
        rms = sqrt(fmax(rms, 0));
        E[octave] = rms;
    }
    // copy to ebuffer
    for ( int octave = 0; octave < OCTAVES; ++octave )
    {
        ebuffer[eindex][octave] = E[octave];
    }
    eindex++;

    if ( eindex == RHYTHMSIZE )
    {
        for ( int octave = 0; octave < OCTAVES; ++octave )
        {

float rhaar[2][RHYTHMSIZE][CHANNELS];
memset(rhaar, 0, sizeof(rhaar));

// window
for ( int i = 0; i < RHYTHMSIZE; ++i )
{
    rhaar[src][i][0] = ebuffer[i][octave] * rwindow[i];
}

// compute Haar wavelet transform
for ( int length = RHYTHMSIZE >> 1; length > 0; length >>= 1 )
{
    for ( int i = 0; i < length; ++i )
    {
        float a = rhaar[src][2 * i + 0][0];
        float b = rhaar[src][2 * i + 1][0];
        float s = (a + b) / 2;
        float d = (a - b) / 2;
        rhaar[dst][i][0] = s;
        rhaar[dst][length + i][0] = d;
    }
}

```

```

200         for (int i = 0; i < RHYTHMSIZE; ++i)
201             {
202                 rhaar[src][i][0] = rhaar[dst][i][0];
203             }
204     }
205
206     // compute energy per octave
207     R{octave}[0] += fabsf(rhaar[src][0][0]); // DC
208     for (int rhythm = 1, length = 1
209          ; length <= RHYTHMSIZE >> 1
210          ; rhythm += 1, length <<= 1
211          )
212     {
213         double rms = 0;
214         for (int i = 0; i < length; ++i)
215         {
216             double a = rhaar[src][length + i][0];
217             rms += a * a;
218         }
219         rms /= length;
220         rms = sqrt(fmax(rms, 0));
221         R{octave}[rhythm] += rms;
222     }
223
224 } // for octave
R_count++;
eindex -= RHOP;

// shunt
230 for (int i = 0; i < RHYTHMSIZE - RHOP; ++i)
{
    for (int octave = 0; octave < OCTAVES; ++octave)
    {
        ebuffer[i][octave] = ebuffer[i + RHOP][octave];
    }
}
235 for (int i = RHYTHMSIZE - RHOP; i < RHYTHMSIZE; ++i)
{
    for (int octave = 0; octave < OCTAVES; ++octave)
    {
        ebuffer[i][octave] = 0;
    }
}
240
241 } // if (eindex == RHYTHMSIZE)
242 } // while (reading > 0)

sf_close(ifile);

// output fingerprint
250 fprintf(stderr, "writing %s\n", argv[2]);
FILE *ofile = fopen(argv[2], "wb");
for (int i = 0; i < OCTAVES; ++i)
{
    for (int j = 0; j < RHYTHMS; ++j)
    {
        fprintf(ofile, "% .18e\n", R[i][j] / R_count);
255

```

```

    }
    fprintf(ofile , "\n");
}
260 fclose(ofile);
fprintf(stderr , "total frames %d\n" , R_count);

return 0;
}

```

11 rhythm-synthesis.c

```

// note: uses stack allocated arrays, may need `ulimit -s unlimited`

#include <math.h>
#include <stdio.h>
5 #include <stdlib.h>
#include <string.h>

#include <sndfile.h>

10 // FIXME these defines must match rhythm-analysis.c

#define OCTAVES 11
#define OCTAVESIZE (1 << (OCTAVES - 1))

15 #define RHYTHMS 13
#define RHYTHMSIZE (1 << (RHYTHMS - 1))

#define OVERLAP 4

20 #define OHOP (OCTAVESIZE / OVERLAP)
#define RHOP (RHYTHMSIZE / OVERLAP)

// FIXME should match analyzed source audio
#define SR 44100
25

int main(int argc, char **argv)
{
    if (argc != 3)
    {
30        fprintf
            ( stderr
            , "usage:\n"
            "    %s in.dat out.wav\n"
            , argv[0]
            );
    35        return 1;
    }
    const int normalize_control = 0;
    const int normalize_carrier = 1;
40

    float R[OCTAVES][RHYTHMS];
    memset(R, 0, sizeof(R));
    fprintf(stderr , "reading %s\n" , argv[1]);
    FILE *ifile = fopen(argv[1] , "rb");
45    double sr = 0;
    for (int i = 0; i < OCTAVES; ++i)

```

```

{
    for ( int j = 0; j < RHYTHMS; ++j)
    {
        double r = 0;
        fscanf(ifile, "%lf ", &r);
        R[i][j] = r;
        if (j > 0)
            sr += r * r;
    }
}
sr /= OCTAVES * (RHYTHMS - 1);
sr = sqrt(sr);
if (normalize_control)
for ( int i = 0; i < OCTAVES; ++i)
{
    for ( int j = 1; j < RHYTHMS; ++j)
    {
        R[i][j] /= sr;
    }
}
fclose(ifile);

fprintf(stderr, "writing %s\n", argv[2]);
const int CHANNELS = 2;
SF_INFO outfo =
{
    0
    , SR
    , CHANNELS
    , SF_FORMAT_WAV | SF_FORMAT_FLOAT
    , 0
    , 0
    , 0
};
SNDFILE *ofile = sf_open(argv[2], SFM_WRITE, &outfo);
if (!ofile) abort();

float nbuffer[OCTAVESIZE][CHANNELS];
memset(nbuffer, 0, sizeof(nbuffer));

85 float obuffer[OCTAVESIZE][CHANNELS];
memset(obuffer, 0, sizeof(obuffer));

float owindow[OCTAVESIZE];
memset(owindow, 0, sizeof(owindow));
for ( int i = 0; i < OCTAVESIZE; ++i)
{
    owindow[i] = 0.5 - 0.5 * cos(i * 6.283185307179586 / OCTAVESIZE);
}

95 float rwindow[RHYTHMSIZE];
memset(rwindow, 0, sizeof(rwindow));
for ( int i = 0; i < RHYTHMSIZE; ++i)
{
    rwindow[i] = 0.5 - 0.5 * cos(i * 6.283185307179586 / RHYTHMSIZE);
}

100 int eindex = 0;
float ebuffer[RHYTHMSIZE][OCTAVES];

```

```

105     memset( ebuffer , 0, sizeof( ebuffer ) );
110     int rindex = 0;
111     float rbuffer [RHYTHMSIZE][OCTAVES];
112     memset( rbuffer , 0, sizeof( rbuffer ) );
113
114     int aindex = 0;
115     float abuffer [OHOP][CHANNELS];
116     memset( abuffer , 0, sizeof( abuffer ) );
117
118     // FIXME 10mins should be enough for testing?
119     for ( int duration = 0; duration < 10 * 60 * SR; ++duration )
120     {
121         // generate noise
122         for ( int channel = 0; channel < CHANNELS; ++channel )
123         {
124             abuffer [aindex][ channel ] = rand() / (double) RANDMAX - 0.5;
125         }
126         aindex++;
127         if ( aindex == OHOP )
128         {
129             aindex = 0;
130             for ( int i = 0; i < OCTAVESIZE - OHOP; ++i )
131             {
132                 for ( int channel = 0; channel < CHANNELS; ++channel )
133                 {
134                     nbuffer [ i ][ channel ] = nbbuffer [ i + OHOP ][ channel ];
135                 }
136             }
137             for ( int i = OCTAVESIZE - OHOP; i < OCTAVESIZE; ++i )
138             {
139                 for ( int channel = 0; channel < CHANNELS; ++channel )
140                 {
141                     nbuffer [ i ][ channel ] = abuffer [ i - (OCTAVESIZE - OHOP) ][ channel ];
142                 }
143             }
144
145             // window
146             float haar [2][OCTAVESIZE][CHANNELS];
147             memset( haar , 0, sizeof( haar ) );
148             int src = 0;
149             int dst = 1;
150             for ( int i = 0; i < OCTAVESIZE; ++i )
151             {
152                 for ( int channel = 0; channel < CHANNELS; ++channel )
153                 {
154                     haar [src][ i ][ channel ] = nbuffer [ i ][ channel ] * owindow [ i ];
155                 }
156             }
157
158             // compute Haar wavelet transform
159             for ( int length = OCTAVESIZE >> 1; length > 0; length >>= 1 )
160             {
161                 for ( int i = 0; i < length; ++i )
162                 {
163                     for ( int channel = 0; channel < CHANNELS; ++channel )
164                     {
165

```

```

        float a = haar[src][2 * i + 0][channel];
        float b = haar[src][2 * i + 1][channel];
        float s = (a + b) / 2;
        float d = (a - b) / 2;
165      haar[dst][          i ][channel] = s;
        haar[dst][length + i ][channel] = d;
    }
}
for (int i = 0; i < OCTAVESIZE; ++i)
{
    for (int channel = 0; channel < CHANNELS; ++channel)
    {
        haar[src][ i ][channel] = haar[dst][ i ][channel];
    }
175
}
// compute energy per octave
double E[OCTAVES];
180 memset(E, 0, sizeof(E));
for (int channel = 0; channel < CHANNELS; ++channel)
{
    E[0] += fabsf(haar[src][0][channel]); // DC
}
185 for ( int octave = 1, length = 1
        ; length <= OCTAVESIZE >> 1
        ; octave += 1, length <= 1
        )
{
    double rms = 0;
    for (int channel = 0; channel < CHANNELS; ++channel)
    {
        for (int i = 0; i < length; ++i)
        {
            double a = haar[src][length + i ][channel];
            rms += a * a;
        }
    }
    rms /= length * CHANNELS;
    rms = sqrt(fmax(rms, 0));
    E[octave] = rms;
}
200
// copy to ebuffer
205 for (int octave = 0; octave < OCTAVES; ++octave)
{
    ebuffer[eindex][octave] = E[octave];
}
eindex++;
210
if (eindex == RHYTHMSIZE)
{
    for (int octave = 0; octave < OCTAVES; ++octave)
    {
        float rhaar[2][RHYTHMSIZE];
        memset(rhaar, 0, sizeof(rhaar));
215
}
}

```

```

220      // window
221      for (int i = 0; i < RHYTHMSIZE; ++i)
222      {
223          rhaar[src][i] = ebuffer[i][octave] * 1;//rwindow[i];
224      }

225      // compute Haar wavelet transform
226      for (int length = RHYTHMSIZE >> 1; length > 0; length >>= 1)
227      {
228          for (int i = 0; i < length; ++i)
229          {
230              float a = rhaar[src][2 * i + 0];
231              float b = rhaar[src][2 * i + 1];
232              float s = (a + b) / 2;
233              float d = (a - b) / 2;
234              rhaar[dst][i] = s;
235              rhaar[dst][length + i] = d;
236          }
237          for (int i = 0; i < RHYTHMSIZE; ++i)
238          {
239              rhaar[src][i] = rhaar[dst][i];
240          }
241      }

242      // normalize carrier
243      if (normalize_carrier)
244      {
245          rhaar[src][0] = 1;
246          for (int rhythm = 1, length = 1
247               ; length <= RHYTHMSIZE >> 1
248               ; rhythm += 1, length <<= 1
249               )
250          {
251              double rms = 0;
252              for (int i = 0; i < length; ++i)
253              {
254                  double d = rhaar[src][length + i];
255                  rms += d * d;
256              }
257              rms /= length;
258              rms = sqrt(rms);
259              for (int i = 0; i < length; ++i)
260              {
261                  rhaar[src][length + i] /= rms;
262              }
263          }
264      }

265      // amplify by control data
266      rhaar[src][0] *= R{octave}[0];
267      for (int rhythm = 1, length = 1
268           ; length <= RHYTHMSIZE >> 1
269           ; rhythm += 1, length <<= 1
270           )
271      {
272          for (int i = 0; i < length; ++i)
273      }

```

```

275      {
276          rhaar[src][length + i] *= R{octave}[rhythm];
277      }
278
279      // compute inverse Haar wavelet transform
280      for (int length = 1; length <= RHYTHMSIZE >> 1; length <= 1)
281      {
282          for (int i = 0; i < RHYTHMSIZE; ++i)
283          {
284              rhaar[dst][i] = rhaar[src][i];
285          }
286          for (int i = 0; i < length; ++i)
287          {
288              float s = rhaar[dst][i];
289              float d = rhaar[dst][length + i];
290              float a = s + d;
291              float b = s - d;
292              rhaar[src][2 * i + 0] = a;
293              rhaar[src][2 * i + 1] = b;
294          }
295      }
296
297      // window
298      for (int i = 0; i < RHYTHMSIZE; ++i)
299      {
300          rhaar[src][i] *= rwindow[i];
301      }
302
303      // overlap-add
304      for (int i = 0; i < RHYTHMSIZE; ++i)
305      {
306          rbuffer[i][octave]
307              = i + RHOP < RHYTHMSIZE
308                  ? rbuffer[i + RHOP][octave]
309                  : 0;
310      }
311      for (int i = 0; i < RHYTHMSIZE; ++i)
312      {
313          rbuffer[i][octave] += rhaar[src][i];
314      }
315
316  } // for octave
317
318  // shunt
319  for (int i = 0; i < RHYTHMSIZE - RHOP; ++i)
320  {
321      for (int octave = 0; octave < OCTAVES; ++octave)
322      {
323          ebuffer[i][octave] = ebuffer[i + RHOP][octave];
324      }
325  }
326  for (int i = RHYTHMSIZE - RHOP; i < RHYTHMSIZE; ++i)
327  {
328      for (int octave = 0; octave < OCTAVES; ++octave)
329      {
330          ebuffer[i][octave] = 0;

```

```

        }

335     eindex -= RHOP;
     rindex = 0;
} // if eindex == RHYTHMSIZE

340     // copy
for (int octave = 0; octave < OCTAVES; ++octave)
{
    E[octave] = rbuffer[rindex][octave];
}
rindex++;

345     // amplify octaves
for (int channel = 0; channel < CHANNELS; ++channel)
{
    haar[src][0][channel] *= E[0]; // DC
}
for (int octave = 1, length = 1
     ; length <= OCTAVESIZE >> 1
     ; octave += 1, length <= 1
)
{
    for (int channel = 0; channel < CHANNELS; ++channel)
    {
        for (int i = 0; i < length; ++i)
        {
            haar[src][length + i][channel] *= E[octave];
        }
    }
}

355     // compute inverse Haar wavelet transform
for (int length = 1; length <= OCTAVESIZE >> 1; length <= 1)
{
    for (int i = 0; i < OCTAVESIZE; ++i)
    {
        for (int channel = 0; channel < CHANNELS; ++channel)
        {
            haar[dst][i][channel] = haar[src][i][channel];
        }
    }
    for (int i = 0; i < length; ++i)
    {
        for (int channel = 0; channel < CHANNELS; ++channel)
        {
            float s = haar[dst][i][channel];
            float d = haar[dst][length + i][channel];
            float a = s + d;
            float b = s - d;
            haar[src][2 * i + 0][channel] = a;
            haar[src][2 * i + 1][channel] = b;
        }
    }
}

```

```

390         // window
390         for (int i = 0; i < OCTAVESIZE; ++i)
390         {
390             for (int channel = 0; channel < CHANNELS; ++channel)
390             {
390                 haar[src][i][channel] *= owindow[i];
395             }
395         }

400         // overlap-add
400         for (int i = 0; i < OCTAVESIZE; ++i)
400         {
400             for (int channel = 0; channel < CHANNELS; ++channel)
400             {
400                 obuffer[i][channel]
400                     = i + OHOP < OCTAVESIZE
400                         ? obuffer[i + OHOP][channel]
400                         : 0;
405             }
405         }
410         for (int i = 0; i < OCTAVESIZE; ++i)
410         {
410             for (int channel = 0; channel < CHANNELS; ++channel)
410             {
410                 obuffer[i][channel] += haar[src][i][channel];
415             }
415         }

420         // output
420         sf_writef_float(ofile, &obuffer[0][0], OHOP);

420     } // if aindex == OHOP
420 } // for duration
425

```

12 timbre-stamp.c

```

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
5
#include <sndfile.h>

#define OCTAVES 11
#define BLOCKSIZE (1 << (OCTAVES - 1))
10 #define OVERLAP 4
#define HOP (BLOCKSIZE / OVERLAP)

int main(int argc, char **argv)
{
15    if (argc != 3)
    {
        fprintf

```

```

    ( stderr
20     , "usage:\n"
      " %s in.wav out.wav\n"
      , argv[0]
    );
    return 1;
}
SF_INFO info;
memset(&info, 0, sizeof(info));
fprintf(stderr, "reading %s\n", argv[1]);
SNDFILE *ifile = sf_open(argv[1], SFM.READ, &info);
if (! ifile) abort();
30   fprintf(stderr, "writing %s\n", argv[2]);
SF_INFO outfo =
{
  0
  , info.sampleRate
  , info.channels
35  , SF_FORMAT_WAV | SF_FORMAT_FLOAT
  , 0
  , 0
};
SNDFILE *ofile = sf_open(argv[2], SFM.WRITE, &outfo);
40   if (! ofile) abort();
const int CHANNELS = info.channels;

float ibuffer[BLOCKSIZE][CHANNELS];
memset(ibuffer, 0, sizeof(ibuffer));
float obuffer[BLOCKSIZE][CHANNELS];
memset(obuffer, 0, sizeof(obuffer));
float nbuffer[BLOCKSIZE][CHANNELS];
memset(nbuffer, 0, sizeof(nbuffer));
float window[BLOCKSIZE];
50   memset(window, 0, sizeof(window));
for (int i = 0; i < BLOCKSIZE; ++i)
{
  window[i] = 0.5 - 0.5 * cos(i * 6.283185307179586 / BLOCKSIZE);
}
55   int reading = OVERLAP;
while (reading > 0)
{
  // read input
  for (int i = 0; i < BLOCKSIZE - HOP; ++i)
  {
    for (int channel = 0; channel < CHANNELS; ++channel)
    {
55      ibuffer[i][channel] = ibuffer[i + HOP][channel];
      nbuffer[i][channel] = nbuffer[i + HOP][channel];
    }
  }
  for (int i = BLOCKSIZE - HOP; i < BLOCKSIZE; ++i)
70  {
    for (int channel = 0; channel < CHANNELS; ++channel)
    {
      // pre-fill with zero in case of end of input
      ibuffer[i][channel] = 0;
    }
  }
}

```

```

75         nbuffer [ i ] [ channel ] = rand () / ( double ) RANDMAX - 0.5;
    }
}
if ( reading == OVERLAP)
{
80     int r = sf_readf_float ( ifile , &ibuffer [ BLOCKSIZE - HOP ] [ 0 ] , HOP );
    if ( r != HOP )
    {
        reading --;
    }
85 }
else
{
    reading --;
}
90
// window
float haar [ 2 ] [ BLOCKSIZE ] [ CHANNELS ];
memset ( haar , 0 , sizeof ( haar ) );
int src = 0;
int dst = 1;
for ( int i = 0; i < BLOCKSIZE; ++i )
{
    for ( int channel = 0; channel < CHANNELS; ++channel )
    {
100        haar [ src ] [ i ] [ channel ] = ibuffer [ i ] [ channel ] * window [ i ];
    }
}

// compute Haar wavelet transform
for ( int length = BLOCKSIZE >> 1; length > 0; length >= 1 )
{
    for ( int i = 0; i < length; ++i )
    {
        for ( int channel = 0; channel < CHANNELS; ++channel )
110        {
            float a = haar [ src ] [ 2 * i + 0 ] [ channel ];
            float b = haar [ src ] [ 2 * i + 1 ] [ channel ];
            float s = ( a + b ) / 2;
            float d = ( a - b ) / 2;
115            haar [ dst ] [ i ] [ channel ] = s;
            haar [ dst ] [ length + i ] [ channel ] = d;
        }
    }
    for ( int i = 0; i < BLOCKSIZE; ++i )
120    {
        for ( int channel = 0; channel < CHANNELS; ++channel )
        {
            haar [ src ] [ i ] [ channel ] = haar [ dst ] [ i ] [ channel ];
        }
125    }
}

// compute energy per octave
double E [ OCTAVES ];
130 memset ( E , 0 , sizeof ( E ) );
for ( int channel = 0; channel < CHANNELS; ++channel )

```

```

    {
        E[0] += fabsf(haar[src][0][channel]); // DC
    }
135   for ( int octave = 1, length = 1
        ; length <= BLOCKSIZE >> 1
        ; octave += 1, length <=> 1
    )
    {
140       double rms = 0;
        for ( int channel = 0; channel < CHANNELS; ++channel)
        {
            for ( int i = 0; i < length; ++i)
            {
                double a = haar[src][length + i][channel];
                rms += a * a;
            }
        }
        rms /= length * CHANNELS;
150       rms = sqrt(fmax(rms, 0));
        E{octave} = rms;
    }

// window
155   for ( int i = 0; i < BLOCKSIZE; ++i)
    {
        for ( int channel = 0; channel < CHANNELS; ++channel)
        {
            haar[src][i][channel] = nbuffer[i][channel] * window[i];
        }
    }

// compute Haar wavelet transform
160   for ( int length = BLOCKSIZE >> 1; length > 0; length >>= 1)
    {
        for ( int i = 0; i < length; ++i)
        {
            for ( int channel = 0; channel < CHANNELS; ++channel)
            {
165               float a = haar[src][2 * i + 0][channel];
               float b = haar[src][2 * i + 1][channel];
               float s = (a + b) / 2;
               float d = (a - b) / 2;
               haar[dst][i][channel] = s;
               haar[dst][length + i][channel] = d;
            }
        }
170   for ( int i = 0; i < BLOCKSIZE; ++i)
    {
        for ( int channel = 0; channel < CHANNELS; ++channel)
        {
            haar[src][i][channel] = haar[dst][i][channel];
        }
    }
175   }

// amplify octaves
180   for ( int channel = 0; channel < CHANNELS; ++channel)
    {
185   }

```

// amplify octaves

for (int channel = 0; channel < CHANNELS; ++channel)

```

190     {
191         haar[ src ][ 0 ][ channel ] *= E[ 0 ];
192     }
193     for ( int octave = 1, length = 1
194          ; length <= BLOCKSIZE >> 1
195          ; octave += 1, length <=> 1
196      )
197     {
198         for ( int channel = 0; channel < CHANNELS; ++channel )
199         {
200             for ( int i = 0; i < length; ++i )
201             {
202                 haar[ src ][ length + i ][ channel ] *= E[ octave ];
203             }
204         }
205     }
206
207     // compute inverse Haar wavelet transform
208     for ( int length = 1; length <= BLOCKSIZE >> 1; length <=> 1 )
209     {
210         for ( int i = 0; i < BLOCKSIZE; ++i )
211         {
212             for ( int channel = 0; channel < CHANNELS; ++channel )
213             {
214                 haar[ dst ][ i ][ channel ] = haar[ src ][ i ][ channel ];
215             }
216         }
217         for ( int i = 0; i < length; ++i )
218         {
219             for ( int channel = 0; channel < CHANNELS; ++channel )
220             {
221                 float s = haar[ dst ][ i ][ channel ];
222                 float d = haar[ dst ][ length + i ][ channel ];
223                 float a = s + d;
224                 float b = s - d;
225                 haar[ src ][ 2 * i + 0 ][ channel ] = a;
226                 haar[ src ][ 2 * i + 1 ][ channel ] = b;
227             }
228         }
229     }
230
231     // window
232     for ( int i = 0; i < BLOCKSIZE; ++i )
233     {
234         for ( int channel = 0; channel < CHANNELS; ++channel )
235         {
236             haar[ src ][ i ][ channel ] *= window[ i ];
237         }
238     }
239
240     // overlap-add
241     for ( int i = 0; i < BLOCKSIZE; ++i )
242     {
243         for ( int channel = 0; channel < CHANNELS; ++channel )
244         {
245             obuffer[ i ][ channel ]
246             = i + HOP < BLOCKSIZE

```

```
    ? obuffer [ i + HOP] [ channel ]
    : 0;
}
250 for (int i = 0; i < BLOCKSIZE; ++i)
{
    for (int channel = 0; channel < CHANNELS; ++channel)
    {
        obuffer [ i ] [ channel ] += haar [ src ] [ i ] [ channel ];
255    }
}
260 // output
sf_writef_float (ofile , &obuffer [ 0 ] [ 0 ] , HOP);
sf_close (ifile );
sf_close (ofile );
return 0;
265 }
```