fractal-channel-hopping

Claude Heiland-Allen

2011–2019

Contents

1 audio/fch.pd 2
2 audio/voice˜.pd 4
3 COPYING 5
4 .gitignore 17
5 NEWS 17
6 photo/fractal-photo-mosaic.c 18
7 photo/Makefile 25
8 photo/prepare.sh 25
9 photo/README.md 25
10 README 26
11 src/channel.c 26
12 src/channel.h 28
13 src/config.h 29
14 src/fch.frag 30
15 src/list.c 31
16 src/list.h 33
17 src/main.c 34
18 src/Makefile 45
19 src/pfifo.c 46
20 src/pfifo.h 47
21 src/record.c 48
22 src/record.h 50
23 src/s2c.sh 51
24 src/yuv2rgb.frag 51
25 start.sh 52

1 audio/fch.pd

```
#N canvas 3 63 450 524 10;
#X obj 21 20 table \$0-notes 24;
#X obj 21 42 loadbang;
#X obj 21 86 s \$0-notes;
#X obj 21 108 loadbang;
#X msg 21 151 1;
#X obj 75 197 + 1;
#X obj 75 219 mod 24;
#X obj 21 195 f;
#X obj 21 217 t f f;
#X obj 21 239 mod 4;
#X obj 21 261 pack f f;
#X obj 72 241 tabread \$0-notes;
#X obj 21 283 route 0 1 2 3;
```

```
#X obj 18 464 dac~;
#X obj 277 125 loadbang;
#X msg 277 147 \; pd dsp 1;
#X obj 21 313 voice~;
#X obj 31 333 voice~;
#X obj 39 355 voice~;
#X obj 51 373 voice~;
#X obj 18 396 hip~ 10;
#X obj 18 420 lop~ 10000;
#X obj 131 283 route 0 1 2 3;
#X obj 131 313 voice~;
#X obj 141 333 voice~;
#X obj 149 355 voice~;
#X obj 161 373 voice~;
#X obj 128 396 hip~ 10;
#X obj 128 418 lop~ 10000;
#X msg 21 64 0 0 9 5 2 10 7 3 0 8 5 1 10 6 3 11 8 4 1 9 6 2 11 7 4
;
#X obj 21 173 metro 2500;
#X obj 208 478 writesf~ 2;
#X msg 262 260 open -bytes 4 fch.wav \, start;
#X obj 21 130 delay 1000;
#X obj 280 289 delay 1.8e+06;
#X msg 280 345 stop;
#X msg 280 394 \; pd quit;
#X obj 293 370 delay 1000;
#X obj 262 211 delay 60000;
#X floatatom 154 213 5 0 0 0 - - -, f 5;
#X connect 1 0 29 0;
#X connect 3 0 33 0;
#X connect 4 0 30 0;
#X connect 5 0 6 0;
#X connect 6 0 7 1;
#X connect 7 0 5 0;
#X connect 7 0 8 0;
#X connect 7 0 39 0;
#X connect 8 0 9 0;
#X connect 8 1 11 0;
#X connect 9 0 10 0;
#X connect 10 0 12 0;
#X connect 10 0 22 0;
#X connect 11 0 10 1;
#X connect 12 0 16 0;
#X connect 12 1 17 0;
#X connect 12 2 18 0;
#X connect 12 3 19 0;
#X connect 14 0 15 0;
#X connect 16 0 20 0;
#X connect 17 0 20 0;
#X connect 18 0 20 0;
#X connect 19 0 20 0;
#X connect 20 0 21 0;
#X connect 21 0 13 0;
#X connect 21 0 31 0;
#X connect 22 0 23 0;
#X connect 22 1 24 0;
#X connect 22 2 25 0;
```

```
#X connect 22 3 26 0;
#X connect 23 0 27 0;
#X connect 24 0 27 0;
#X connect 25 0 27 0;
#X connect 26 0 27 0;
#X connect 27 0 28 0;
#X connect 28 0 13 1;
#X connect 28 0 31 1;
#X connect 29 0 2 0;
#X connect 30 0 7 0;
#X connect 32 0 31 0;
#X connect 33 0 4 0;
#X connect 33 0 38 0;
#X connect 34 0 35 0;
#X connect 34 0 37 0;
#X connect 35 0 31 0;
#X connect 37 0 36 0;
#X connect 38 0 32 0;
#X connect 38 0 34 0;
```

2 audio/voice˜.pd

```
#N canvas 3 58 450 300 10;
#X obj 174 19 inlet;
#X obj 174 41 mtof;
#X obj 30 33 noise˜;
#X obj 83 64 vcf˜ 50;
#X obj 19 286 outlet˜;
#X obj 83 84 vcf˜ 50;
#X obj 83 104 vcf˜ 50;
#X obj 83 124 vcf˜ 50;
#X obj 83 144 vcf˜ 50;
#X obj 83 164 vcf˜ 50;
#X obj 83 184 vcf˜ 50;
#X obj 83 205 vcf˜ 50;
#X obj 83 225 vcf˜ 50;
#X obj 83 245 vcf˜ 50;
#X obj 83 265 vcf˜ 50;
#X obj 174 63 * 2;
#X obj 174 85 * 2;
#X obj 174 107 * 2;
#X obj 174 129 * 2;
#X obj 174 151 * 2;
#X obj 174 173 * 2;
#X obj 174 195 * 2;
#X obj 174 217 * 2;
#X obj 174 239 * 2;
#X obj 174 261 * 2;
#X connect 0 0 1 0;
#X connect 1 0 3 1;
#X connect 1 0 15 0;
#X connect 2 0 3 0;
#X connect 2 0 5 0;
#X connect 2 0 6 0;
#X connect 2 0 7 0;
#X connect 2 0 8 0;
#X connect 2 0 9 0;
```

```
#X connect 2 0 10 0;
#X connect 2 0 11 0;
#X connect 2 0 12 0;
#X connect 2 0 13 0;
#X connect 2 0 14 0;
#X connect 3 0 4 0;
#X connect 5 0 4 0;
#X connect 6 0 4 0;
#X connect 7 0 4 0;
#X connect 8 0 4 0;
#X connect 9 0 4 0;
#X connect 10 0 4 0;
#X connect 11 0 4 0;
#X connect 12 0 4 0;
#X connect 13 0 4 0;
#X connect 14 0 4 0;
#X connect 15 0 5 1;
#X connect 15 0 16 0;
#X connect 16 0 6 1;
#X connect 16 0 17 0;
#X connect 17 0 7 1;
#X connect 17 0 18 0;
#X connect 18 0 8 1;
#X connect 18 0 19 0;
#X connect 19 0 9 1;
#X connect 19 0 20 0;
#X connect 20 0 10 1;
#X connect 20 0 21 0;
#X connect 21 0 11 1;
#X connect 21 0 22 0;
#X connect 22 0 12 1;
#X connect 22 0 23 0;
#X connect 23 0 13 1;
#X connect 23 0 24 0;
#X connect 24 0 14 1;
```

3 COPYING

```
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

  Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so.  This is fundamentally incompatible with the aim of
protecting users' freedom to change the software.  The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable.  Therefore, we
have designed this version of the GPL to prohibit the practice for those
products.  If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions
of the GPL, as needed to protect the freedom of users.

  Finally, every program is threatened constantly by software patents.
States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish to
avoid the special danger that patents applied to a free program could
make it effectively proprietary.  To prevent this, the GPL assures that
patents cannot be used to render the program non-free.

  The precise terms and conditions for copying, distribution and
modification follow.

                       TERMS AND CONDITIONS

  0. Definitions.

  "This License" refers to version 3 of the GNU General Public License.
```

```
actually uses , or expects or is expected to use , the product. A product
is a consumer product regardless of whether the product has substantial
commercial , industrial or non-consumer uses , unless such uses represent
the only significant mode of use of the product.

  ”Installation Information” for a User Product means any methods ,
procedures , authorization keys , or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source. The information must
suffice to ensure that the continued functioning of the modified object
code is in no case prevented or interfered with solely because
modification has been made.

  If you convey an object code work under this section in , or with , or
specifically for use in , a User Product , and the conveying occurs as
part of a transaction in which the right of possession and use of the
User Product is transferred to the recipient in perpetuity or for a
fixed term (regardless of how the transaction is characterized) , the
Corresponding Source conveyed under this section must be accompanied
by the Installation Information. But this requirement does not apply
if neither you nor any third party retains the ability to install
modified object code on the User Product (for example , the work has
been installed in ROM).

  The requirement to provide Installation Information does not include a
requirement to continue to provide support service , warranty , or updates
for a work that has been modified or installed by the recipient , or for
the User Product in which it has been modified or installed. Access to a
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
protocols for communication across the network.

  Corresponding Source conveyed , and Installation Information provided ,
in accord with this section must be in a format that is publicly
documented (and with an implementation available to the public in
source code form) , and must require no special password or key for
unpacking , reading or copying.

  7. Additional Terms.

  ”Additional permissions” are terms that supplement the terms of this
License by making exceptions from one or more of its conditions.
Additional permissions that are applicable to the entire Program shall
be treated as though they were included in this License , to the extent
that they are valid under applicable law. If additional permissions
apply only to part of the Program , that part may be used separately
under those permissions , but the entire Program remains governed by
this License without regard to the additional permissions.

  When you convey a copy of a covered work , you may at your option
remove any additional permissions from that copy , or from any part of
it. (Additional permissions may be written to require their own
removal in certain cases when you modify the work.) You may place
additional permissions on material , added by you to a covered work ,
for which you have or can give appropriate copyright permission.

  Notwithstanding any other provision of this License , for material you
```

```
    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

  If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.
```

4 .gitignore

```
src/fractal-channel-hopping
src/*.frag.c
photo/fractal-photo-mosaic
get_iplayer
rtmpdump
tmp
*.ogv
*.m2v
*.wav
*.mp2
*.mkv
*.data
*.mp4
*.mpeg
*.webm
```

5 NEWS

```
v6      play videos from folders
v5      fix for get-iplayer mode name changes
v4      mplayer simplified stuff
v3      graphics speed boost etc
v2      preliminary sound mixing
v1      first publicised version
```

6 photo/fractal-photo-mosaic.c

```
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <GL/glew.h>
#include <GLFW/glfw3.h>

#define COUNT (103+1)
#define WIDTH 1920
#define HEIGHT 1080

static unsigned char raw[COUNT][HEIGHT][WIDTH][3];
static unsigned char raw16[COUNT][16][16][3];

static double lab16[COUNT][16][16][3];
static double lab2[COUNT][2][2][3];

static float graph[COUNT][8][8];

static int visited[COUNT];

static unsigned char out[HEIGHT][WIDTH][3];

static const char *vert =
"#version 400 core\n"
"uniform vec2 delta;\n"
"uniform float zoom;\n"
"layout(location = 0) in vec2 pos;\n"
"layout(location = 1) in vec2 tc;\n"
"out vec2 c;\n"
"void main() {\n"
"  gl_Position = vec4(zoom * (pos - delta) + delta, 0.0, 1.0);\n"
"  c = vec2(tc.x, 1.0 - tc.y);\n"
"}\n"
;

static const char *frag =
"#version 400 core\n"
"uniform sampler2DArray tex;\n"
"uniform sampler2DArray map;\n"
"uniform int ix;\n"
"uniform vec3 blend;\n"
"in vec2 c;\n"
"layout(location = 0) out vec3 colour;\n"
"void main() {\n"
"  float l = textureQueryLod(tex, c).y;\n"
"  vec3 p0 = vec3(c, float(ix));\n"
"  vec3 p1 = vec3(8.0 * p0.xy, texture(map, p0).x);\n"
"  p1.xy -= floor(p1.xy);\n"
"  vec3 p2 = vec3(8.0 * p1.xy, texture(map, p1).x);\n"
"  p2.xy -= floor(p2.xy);\n"
"  vec3 sum = vec3(0.0);\n"
"  sum += textureLod(tex, p0, l    ).rgb * blend.x;\n"
"  sum += textureLod(tex, p1, l + 3).rgb * blend.y;\n"
"  sum += textureLod(tex, p2, l + 6).rgb * blend.z;\n"
"  colour = sum;\n"
"}\n"
;

static void debug_program(GLuint program, const char *name) {
  if (program) {
    GLint linked = GL_FALSE;
    glGetProgramiv(program, GL_LINK_STATUS, &linked);
    if (linked != GL_TRUE) {
      fprintf(stderr, "%s: OpenGL shader program link failed\n", name);
    }
    GLint length = 0;
    glGetProgramiv(program, GL_INFO_LOG_LENGTH, &length);
    char *buffer = (char *) malloc(length + 1);
    glGetProgramInfoLog(program, length, 0, buffer);
    buffer[length] = 0;
    if (buffer[0]) {
      fprintf(stderr, "%s: OpenGL shader program info log\n", name);
      fprintf(stderr, "%s\n", buffer);
    }
    free(buffer);
    assert(linked == GL_TRUE);
  } else {
    fprintf(stderr, "%s: OpenGL shader program creation failed\n", name);
  }
}

static void debug_shader(GLuint shader, GLenum type, const char *name) {
  const char *tname = 0;
  switch (type) {
    case GL_VERTEX_SHADER:   tname = "vertex";   break;
    case GL_FRAGMENT_SHADER: tname = "fragment"; break;
    case GL_COMPUTE_SHADER:  tname = "compute";  break;
    default:                 tname = "unknown";  break;
  }
  if (shader) {
    GLint compiled = GL_FALSE;
    glGetShaderiv(shader, GL_COMPILE_STATUS, &compiled);
    if (compiled != GL_TRUE) {
      fprintf(stderr, "%s: OpenGL %s shader compile failed\n", name, tname);
    }
    GLint length = 0;
    glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &length);
    char *buffer = (char *) malloc(length + 1);
    glGetShaderInfoLog(shader, length, 0, buffer);
    buffer[length] = 0;
    if (buffer[0]) {
      fprintf(stderr, "%s: OpenGL %s shader info log\n", name, tname);
      fprintf(stderr, "%s\n", buffer);
    }
    free(buffer);
    assert(compiled == GL_TRUE);
  } else {
    fprintf(stderr, "%s: OpenGL %s shader creation failed\n", name, tname);
  }
}
```

```
static void compile_shader(GLint program, GLenum type, const char *name, const ↙
    ↳ GLchar *source) {
  GLuint shader = glCreateShader(type);
  glShaderSource(shader, 1, &source, 0);
  glCompileShader(shader);
  debug_shader(shader, type, name);
  glAttachShader(program, shader);
  glDeleteShader(shader);
}

static GLint compile_program(const char *name, const GLchar *vert, const GLchar ↙
    ↳ *frag) {
  GLint program = glCreateProgram();
  if (vert) { compile_shader(program, GL_VERTEX_SHADER  , name, vert); }
  if (frag) { compile_shader(program, GL_FRAGMENT_SHADER, name, frag); }
  glLinkProgram(program);
  debug_program(program, name);
  return program;
}
static double xyz2lab_f(double t)
{
  static const double e = 0.008856;
  static const double k = 903.3;
  if (t > e)
    return cbrt(t);
  else
    return (k * t + 16) / 116;
}
static void xyz2lab(double x, double y, double z, double *l, double *a, double *↙
    ↳ b)
{
  static const double xn = 0.95047;
  static const double yn = 1.00000;
  static const double zn = 1.08883;
  x /= xn;
  y /= yn;
  z /= zn;
  x = xyz2lab_f(x);
  y = xyz2lab_f(y);
  z = xyz2lab_f(z);
  *l = 116 * y - 16;
  *a = 500 * (x - y);
  *b = 200 * (y - z);
}

static double srgb2xyz_f(double c)
{
  if (c < 0.04045)
    return c / 12.92;
  else
    return pow((c + 0.055) / 1.055, 2.4);
}
static void srgb2xyz(double r, double g, double b, double *x, double *y, double ↙
    ↳ *z)
{
  static const double m[3][3] =
    { { 0.4124, 0.3576, 0.1805 }
    , { 0.2126, 0.7152, 0.0722 }
    , { 0.0193, 0.1192, 0.9505 }
    };
  r = srgb2xyz_f(r);
  g = srgb2xyz_f(g);
  b = srgb2xyz_f(b);
  *x = m[0][0] * r + m[0][1] * g + m[0][2] * b;
  *y = m[1][0] * r + m[1][1] * g + m[1][2] * b;
  *z = m[2][0] * r + m[2][1] * g + m[2][2] * b;
}


static void srgb2lab(double r, double g, double b, double *l, double *a, double ↙
    ↳ *bb)
{
  double x, y, z;
  srgb2xyz(r, g, b, &x, &y, &z);
  xyz2lab(x, y, z, l, a, bb);
}


extern int main()
{
  srand(0x1cedcafe);

  FILE *fraw = fopen("image.data", "rb");
  fread(&raw[0][0][0][0], (COUNT-1) * WIDTH * HEIGHT * 3, 1, fraw);
  fclose(fraw);

  fraw = fopen("thumbs.data", "rb");
  fread(&raw16[0][0][0][0], (COUNT-1) * 16 * 16 * 3, 1, fraw);
  fclose(fraw);

  glfwInit();
  glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
  glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 0);
  glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
  glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
  glfwWindowHint(GLFW_OPENGL_DEBUG_CONTEXT, GL_TRUE);
  glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
  GLFWwindow *window = glfwCreateWindow(WIDTH, HEIGHT, "fractalize", 0, 0);
  glfwMakeContextCurrent(window);
  if (! window) {
    fprintf(stderr, "glfw\n");
    return 1;
  }
  glewInit();

  GLuint program = compile_program("shader", vert, frag);
  glUseProgram(program);
  GLint utex = glGetUniformLocation(program, "tex");
  GLint umap = glGetUniformLocation(program, "map");
  GLint uix = glGetUniformLocation(program, "ix");
  GLint uzoom = glGetUniformLocation(program, "zoom");
  GLint udelta = glGetUniformLocation(program, "delta");
  GLint ublend = glGetUniformLocation(program, "blend");
  glUniform1i(utex, 0);
  glUniform1i(umap, 1);

  GLuint vao;
  glGenVertexArrays(1, &vao);
  glBindVertexArray(vao);
  GLuint vbo;
  glGenBuffers(1, &vbo);
  glBindBuffer(GL_ARRAY_BUFFER, vbo);
  GLfloat vbo_data[] =
    { -1, -1, 0, 0
    , -1,  1, 0, 1
    ,  1, -1, 1, 0
    ,  1,  1, 1, 1
    };
  glBufferData(GL_ARRAY_BUFFER, 16 * sizeof(GLfloat), vbo_data, GL_STATIC_DRAW);
  glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 4 * sizeof(GLfloat), 0);
  glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 4 * sizeof(GLfloat), ((char *)↙
      ↳ 0) + 2 * sizeof(GLfloat));
  glEnableVertexAttribArray(0);
  glEnableVertexAttribArray(1);

  GLuint tex[2];
  glGenTextures(2, &tex[0]);

  glActiveTexture(GL_TEXTURE0 + 0);
  glBindTexture(GL_TEXTURE_2D_ARRAY, tex[0]);
  glTexImage3D(GL_TEXTURE_2D_ARRAY, 0, GL_RGB, WIDTH, HEIGHT, COUNT, 0, GL_RGB, ↙
      ↳ GL_UNSIGNED_BYTE, &raw[0][0][0][0]);
  glGenerateMipmap(GL_TEXTURE_2D_ARRAY);
  glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_MIN_FILTER, ↙
      ↳ GL_LINEAR_MIPMAP_LINEAR);
  glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
  glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

  for (int k = 0; k < COUNT - 1; ++k)
  for (int j = 0; j < 16; ++j)
  for (int i = 0; i < 16; ++i)
    srgb2lab(raw16[k][j][i][0]/255.0, raw16[k][j][i][1]/255.0, raw16[k][j][i↙
        ↳ ][2]/255.0, &lab16[k][j][i][0], &lab16[k][j][i][1], &lab16[k][j][i↙
        ↳ ][2]);

  for (int k = 0; k < COUNT - 1; ++k)
  for (int j = 0; j < 2; ++j)
  for (int i = 0; i < 2; ++i)
  {
    double l = 0;
    double a = 0;
    double b = 0;
    for (int jj = 0; jj < 8; ++jj)
    for (int ii = 0; ii < 8; ++ii)
    {
      l += lab16[k][8 * j + jj][8 * i + ii][0];
      a += lab16[k][8 * j + jj][8 * i + ii][1];
      b += lab16[k][8 * j + jj][8 * i + ii][2];
    }
    l /= 8 * 8;
```

```
      a /= 8 * 8;
      b /= 8 * 8;
      lab2[k][j][i][0] = l;
      lab2[k][j][i][1] = a;
      lab2[k][j][i][2] = b;
    }

    for (int k = 0; k < COUNT - 1; ++k)
    for (int j = 0; j < 8; ++j)
    for (int i = 0; i < 8; ++i)
    {
      double min_metric = 1.0 / 0.0;
      int min_index = -1;
      for (int kk = 0; kk < COUNT - 1; ++kk)
      {
        if (kk == k) continue;
        double s = 0;
        for (int jj = 0; jj < 2; ++jj)
        for (int ii = 0; ii < 2; ++ii)
        for (int c = 0; c < 3; ++c)
        {
          double x = lab16[k][2*j + jj][2*i + ii][c];
          double y = lab2[kk][jj][ii][c];
          double d = x - y;
          s += d * d;
        }
        if (s < min_metric)
        {
          min_metric = s;
          min_index = kk;
        }
      }
      graph[k][j][i] = min_index;
    }
    for (int j = 0; j < 8; ++j)
    for (int i = 0; i < 8; ++i)
      graph[COUNT - 1][j][i] = 0;

    glActiveTexture(GL_TEXTURE0 + 1);
    glBindTexture(GL_TEXTURE_2D_ARRAY, tex[1]);
    glTexImage3D(GL_TEXTURE_2D_ARRAY, 0, GL_R32F, 8, 8, COUNT, 0, GL_RED, GL_FLOAT↙
        ↳ , &graph[0][0][0]);
    glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

    int ix = COUNT-1;
    int zoomi = 4;
    int zoomj = 4;
    int speed = 150;
    double bdx = 0, bdy = 0;
    for (int frame = 1; frame <= 15 * 60 * 60; ++frame)
    {
      if ((frame % speed) == 0)
      {
        ix = graph[ix][zoomj][zoomi];
        int mi = 0x7fffffff;
        for (int j = 0; j < 8; ++j)
        for (int i = 0; i < 8; ++i)
        {
          int k = graph[ix][j][i];
          int m = visited[k];
          mi = m < mi ? m : mi;
        }
        int n = 0;
        for (int j = 0; j < 8; ++j)
        for (int i = 0; i < 8; ++i)
        {
          int k = graph[ix][j][i];
          int m = visited[k];
          n += mi == m;
        }
        int coin = rand() % n;
        n = 0;
        for (int j = 0; j < 8; ++j)
        for (int i = 0; i < 8; ++i)
        {
          int k = graph[ix][j][i];
          int m = visited[k];
          if (mi == m)
          {
            if (coin == n)
            {
              zoomj = j;
              zoomi = i;
              visited[k] += 1;
            }
            n += 1;
          }
        }
      }
      // zoom blending
      double k = ((frame % speed) + 0.5) / speed;
      double zoom = pow(8, k); // hardcoded power - grid size...
      double blend2 = 1 - cos(2 * 3.141592653 * (k + 0) / 3);
      double blend1 = 1 - cos(2 * 3.141592653 * (k + 1) / 3);
      double blend0 = 1 - cos(2 * 3.141592653 * (k + 2) / 3);
      double blendt = blend0 + blend1 + blend2;
      blend0 /= blendt;
      blend1 /= blendt;
      blend2 /= blendt;
      double dx = zoomi * 8 / (8 - 1.0);
      double dy = 8 - zoomj * 8 / (8 - 1.0);
      dx /= 4;
      dy /= 4;
      dx -= 1;
      dy -= 1;
      bdx *= 0.95;
      bdy *= 0.95;
      bdx += 0.05 * dx;
      bdy += 0.05 * dy;
      glUniform1i(uix, ix);
      glUniform1f(uzoom, zoom);
      glUniform2f(udelta, bdx, bdy);
      glUniform3f(ublend, blend0, blend1, blend2);
      glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
      glfwSwapBuffers(window);
      glReadPixels(0, 0, WIDTH, HEIGHT, GL_RGB, GL_UNSIGNED_BYTE, &out[0][0][0]);
      printf("P6\n%d %d\n255\n", WIDTH, HEIGHT);
      for (int j = HEIGHT - 1; j >= 0; --j)
      {
        fwrite(&out[j][0][0], WIDTH * 3, 1, stdout);
      }
      fflush(stdout);
    }
    return 0;
  }
```

7 photo/Makefile

```
fractal-photo-mosaic: fractal-photo-mosaic.c
        gcc -std=c99 -Wall -Wextra -pedantic -O3 -o fractal-photo-mosaic fractal-photo-mosaic.c -lglfw -lGL -lGLEW -lm
```

8 photo/prepare.sh

```
#!/bin/sh
for i in *.jpeg *.png
do
  convert "${i}" -geometry 1920x1080^ -gravity center                                  -extent 1920x1080 -blur 0x32 "/tmp/background.png"
  convert "${i}" -geometry 1920x1080  -gravity center -alpha opaque -background transparent -extent 1920x1080              "/tmp/foreground.png"
  composite -compose Over -gravity center "/tmp/foreground.png" "/tmp/background.png" "${i}.ppm"
done
for i in *.ppm
do
  cat "${i}" | tail -c $((1920 * 1080 * 3))
done > image.data
for i in *.ppm
do
  convert "${i}" -geometry '16x16!' "/tmp/thumb.ppm"
  cat "/tmp/thumb.ppm" | tail -c $((16 * 16 * 3))
done > thumbs.data
```

9 photo/README.md

```
fractal-photo-mosaic
====================

render a zooming fractal video from a collection of images

usage
-----

    mkdir images
    cd images
```

```
wget *.jpeg *.png # needs exactly 103 images total, TODO FIXME hardcoding
../prepare.sh
cd ..
ln -s images/image.data
ln -s images/thumbs.data
make
./fractal-photo-mosaic |
ffmpeg -i soundtrack.wav -f image2pipe -codec ppm -framerate 60 -i - \
  -pix_fmt yuv420p -profile:v high -level:v 4.1 -b:v 20M -b:a 192k \
  fractal-photo-mosaic.mkv
```

10 README

QUICK START

```
# get a computer with graphics drivers supporting OpenGL/GLSL 130
# on Debian-based systems (you might need multimedia repositories)
sudo apt install \
  build-essential \
  libglew1.5-dev \
  freeglut3-dev \
  libjack-jackd2-dev \
  jackd \
  mjpegtools \
  mplayer \
  ffmpeg
# get the source, if you're reading this you probably already have
git clone http://code.mathr.co.uk/fractal-channel-hopping.git
cd fractal-channel-hopping/
git tag -ln99
git checkout v6   # or a different tag
# prepare videos
mkdir v
cd v
ln -s /path/to/some/videos/ "Channel 01"
ln -s /path/to/more/videos/ "Channel 02"
# etc, up to 12 channels
# start JACK server
../start.sh
# watch tv
# 'Shift-R' to start/stop recording
# 'F11' fullscreen
# 'Shift-Q' or 'ESC' to quit
```

NOTES

```
be patient during startup (takes a few seconds)
sometimes it doesn't quit cleanly, "killall mplayer" perhaps
tested with NVIDIA proprietary drivers and AMD open source drivers
recording is in PAL DVD format (_.mpeg)
```

11 src/channel.c

```
/*
  fractal-channel-hopping -- infinite fractal television zoom
  Copyright (C) 2011,2015,2019 Claude Heiland-Allen
```

```c
  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#define _DEFAULT_SOURCE

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "config.h"
#include "channel.h"

void *channel_main(void *arg) {
  struct channel *channel = arg;
  if (channel->width == 256 && channel->height == 256) {
    int bytes = 256 * 256 + 128 * 128 * 2;
    const char *fmt = "mplayer -quiet -really-quiet -loop 0 -shuffle -aspect 1/1 -vf scale=256:256 -ao 'jack:name=%s:port=fch.*%s_[1-2]' -vo 'yuv4mpeg:file=tmp/%s.fifo' -fixed-vo '%s/'*.mp4 >'tmp/%s.mplayer.log' 2>&1 & cat 'tmp/%s.fifo'";
    int cmdlen = strlen(fmt) + 6 * strlen(channel->name) + 64;
    char *cmd = malloc(cmdlen);
    snprintf(cmd, cmdlen - 2, fmt, channel->name, channel->name, channel->name, channel->name, channel->name, channel->name);
    cmd[cmdlen-1] = 0;
    FILE *video;
    if ((video = popen(cmd, "r"))) {
      const char *vhdr25 = "YUV4MPEG2 W256 H256 F25:1 Ip A1:1\n";
      const char *vhdr50 = "YUV4MPEG2 W256 H256 F50:1 Ip A1:1\n";
      const char *fhdr = "FRAME\n";
      char hdr[64];
      if (1 != fread(hdr, strlen(vhdr25), 1, video)) { channel->aborted = 1; goto cleanup; }
      hdr[strlen(vhdr25)] = 0;
      if (strcmp(vhdr25, hdr) && strcmp(vhdr50, hdr)) { channel->aborted = 2; goto cleanup; }
      while (! channel->quit) {
        if (1 != fread(hdr, strlen(fhdr), 1, video)) { channel->aborted = 3; break; }
        hdr[strlen(fhdr)] = 0;
        if (strcmp(fhdr, hdr)) { channel->aborted = 4; break; }
        if (1 != fread(channel->image, bytes, 1, video)) { channel->aborted = 5; break; }
      }
      cleanup:
      pclose(video);
    }
    free(cmd);
  }
  if (channel->aborted) {
    fprintf(stderr, "channel '%s' aborted '%d'\n", channel->name, channel->↙
       ↳ aborted);
  }
  pthread_exit(0);
  return 0;
}

struct channel *channel_start(const char *name) {
  struct channel *channel = calloc(1, sizeof(struct channel));
  // clear YUV to black
  memset(&channel->image[0], 0, VIDEO_WIDTH * VIDEO_HEIGHT);
  memset(&channel->image[VIDEO_WIDTH * VIDEO_HEIGHT], 128, 2 * VIDEO_WIDTH/2 * ↙
      ↳ VIDEO_HEIGHT/2);
  channel->name   = strdup(name);
  channel->width  = VIDEO_WIDTH;
  channel->height = VIDEO_HEIGHT;
  if (pthread_create(&channel->thread, 0, channel_main, channel)) {
    free(channel->name);
    free(channel);
    return 0;
  }
  return channel;
}

int channel_stop(struct channel *channel) {
  channel->quit = 1;
  pthread_join(channel->thread, 0);
  int r = channel->aborted;
  free(channel->name);
  free(channel);
  return r;
}
```

12 src/channel.h

```
/*
  fractal-channel-hopping -- infinite fractal television zoom
  Copyright (C) 2011 Claude Heiland-Allen

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef CHANNEL_H
#define CHANNEL_H 1

#include <pthread.h>

#include "config.h"

struct channel {
  pthread_t thread;
  char *name;
  int quit;
  int aborted;
  int width;
  int height;
  unsigned char image[VIDEO_WIDTH * VIDEO_HEIGHT + 2 * VIDEO_WIDTH/2 * ↙
      ↳ VIDEO_HEIGHT/2];
};

struct channel *channel_start(const char *name);
int channel_stop(struct channel *channel);

#endif
```

13 src/config.h

```
/*
  fractal-channel-hopping -- infinite fractal television zoom
  Copyright (C) 2011 Claude Heiland-Allen

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef CONFIG_H
#define CONFIG_H 1

#define VIDEO_WIDTH  256
#define VIDEO_HEIGHT 256
#define GRID_WIDTH    16
#define GRID_HEIGHT   16

//#define TEXTURE_SIZE 256
#define TEXTURE_X      1.0f
#define TEXTURE_Y      0.5625f
```

```

#define CHANNEL_COUNT_MAX 16

#define OUTPUT_WIDTH  1024
#define OUTPUT_HEIGHT 576

#endif
```

14 src/fch.frag

```
/*
  fractal-channel-hopping -- infinite fractal television zoom
  Copyright (C) 2011,2019 Claude Heiland-Allen

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#version 130

uniform sampler2D images; // 4x4 tile sheet
uniform vec3 matches[12]; // hardcoded maximum channels
uniform int focus;
uniform float blend0;
uniform float blend1;
uniform float blend2;

vec3 get(int n, vec2 p, float bias, bool wrap) {
  int j = int(floor(float(n) / 4.0));
  int i = n - 4 * j;
  vec2 p1 = p;
  if (wrap) {
    float k = pow(2.0, 8.0 - bias);
    p1 = floor(p1 * k) / k;
  }
  p1 *= 254.0 / 256.0;
  p1 += 1.0 / 256.0;
  vec2 q = (vec2(float(i), float(j)) + p1) / 4.0;
  return textureLod(images, q, bias).rgb;
}

int match(int def, vec3 c) {
  float d = 1024.0;
  int m = def;
  for (int i = 0; i < 12; ++i) {
    if (length(matches[i]) > 0.0) {
      float d2 = length(c - matches[i]);
      if (d2 < d) {
        d = d2;
        m = i;
      }
    }
  }
  return m;
}

void main(void) {
  int  f0 = focus;
  vec2 p0 = gl_TexCoord[0].xy;
  vec3 c0 = get(f0, p0, 0.0, false);
  vec3 ca = get(f0, p0, 4.0, true);

  int  f1 = match(f0, ca);
  vec2 p1 = 16.0 * p0; p1 -= floor(p1);
  vec3 c1 = get(f1, p1, 0.0, false);
  vec3 cb = get(f1, p1, 4.0, true);

  int  f2 = match(f1, cb);
  vec2 p2 = 16.0 * p1; p2 -= floor(p2);
  vec3 c2 = get(f2, p2, 0.0, false);

  float channel[12];
  channel[0] = 0.0;
  channel[1] = 0.0;
  channel[2] = 0.0;
  channel[3] = 0.0;
  channel[4] = 0.0;
  channel[5] = 0.0;
  channel[6] = 0.0;
  channel[7] = 0.0;
  channel[8] = 0.0;
  channel[9] = 0.0;
  channel[10] = 0.0;
  channel[11] = 0.0;
  channel[f0] += blend0;
  channel[f1] += blend1;
  channel[f2] += blend2;
  gl_FragData[0] = vec4(blend0 * c0 + blend1 * c1 + blend2 * c2, 1.0);
  gl_FragData[1] = vec4(channel[0], channel[1], channel[2], channel[3]);
  gl_FragData[2] = vec4(channel[4], channel[5], channel[6], channel[7]);
  gl_FragData[3] = vec4(channel[8], channel[9], channel[10], channel[11]);
}
```

15 src/list.c

```
/*
  fractal-channel-hopping -- infinite fractal television zoom
  Copyright (C) 2011 Claude Heiland-Allen

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.
  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <assert.h>
#include "list.h"

void list_init(struct list *l) {
  assert(l);
  l->head = &(l->headNode);
  l->tail = &(l->tailNode);
  l->head->pred = 0;
  l->head->next = l->tail;
  l->tail->pred = l->head;
  l->tail->next = 0;
}

int list_ishead(struct node *n) {
  assert(n);
  return !n->pred;
}

int list_istail(struct node *n) {
  assert(n);
  return !n->next;
}

int list_isempty(struct list *l) {
  assert(l && l->head && l->tail);
  return l->head->next == l->tail;
}

int list_length(struct list *l) {
  struct node *n = l->head->next;
  int i = 0;
  while (n != l->tail) {
    n = n->next;
    i++;
  }
  return i;
}

void list_remove(struct node *n) {
  assert(n && n->pred && n->next);
  n->pred->next = n->next;
  n->next->pred = n->pred;
  n->next = 0;
  n->pred = 0;
}

void list_insertbefore(struct node *n, struct node *beforethis) {
  assert(n && beforethis && beforethis->pred);
  n->next = beforethis;
  n->pred = beforethis->pred;
  beforethis->pred->next = n;
  beforethis->pred = n;
}

void list_insertafter(struct node *n, struct node *afterthis) {
  assert(n && afterthis && afterthis->next);
  n->next = afterthis->next;
  n->pred = afterthis;
  afterthis->next->pred = n;
  afterthis->next = n;
}

void list_inserttail(struct list *l, struct node *n) {
  assert(l);
  list_insertbefore(n, l->tail);
}

struct node *list_removehead(struct list *l) {
  assert(l && ! list_isempty(l));
  struct node *n = l->head->next;
  list_remove(n);
  return n;
}
```

16 src/list.h

```
/*
  fractal-channel-hopping -- infinite fractal television zoom
  Copyright (C) 2011 Claude Heiland-Allen

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef LIST_H
#define LIST_H 1

struct node {
  struct node *next;
  struct node *pred;
};

struct list {
  struct node *head;
  struct node *tail;
  // private
  struct node headNode;
  struct node tailNode;
};

void list_init(struct list *l);
int list_ishead(struct node *n);
int list_istail(struct node *n);
int list_isempty(struct list *l);
int list_length(struct list *l);
void list_remove(struct node *n);
void list_insertbefore(struct node *n, struct node *beforethis);
void list_insertafter(struct node *n, struct node *afterthis);
void list_inserttail(struct list *l, struct node *n);
struct node *list_removehead(struct list *l);

#endif
```

17 src/main.c

```
/*
  fractal-channel-hopping -- infinite fractal television zoom
  Copyright (C) 2011,2019 Claude Heiland-Allen

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#define _DEFAULT_SOURCE
#define _POSIX_C_SOURCE 200809

#include <assert.h>
#include <math.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

#include <GL/glew.h>
#include <GL/glut.h>

#include <jack/jack.h>

#include "channel.h"
#include "record.h"
```

```c
#include "fch.frag.c"
#include "yuv2rgb.frag.c"

int max(int x, int y) {
  return x > y ? x : y;
}

unsigned int roundtwo(unsigned int x) {
  assert(x <= 1u << 31u); // termination condition
  unsigned int y = 1;
  while (y < x) y <<= 1;
  return y;
}

unsigned int logtwo(unsigned int x) {
  assert(x <= 1u << 31u); // termination condition
  unsigned int y = 1, z = 0;
  while (y < x) { y <<= 1; z += 1; };
  return z;
}

int focus = 0;
int speed = 75;
int frames = 0;
int tframes = 0;

int count;
struct channel **channels;

struct record *recorder = 0;

int winw, winh;
int tsize;
GLuint texiny, texinu, texinv, timages, toutput, tmatch[3]; // hardcoded: (12) channels / 4
GLuint fbo;

// fractalization shader
GLhandleARB prog;
GLhandleARB frag;
GLint uimages, umatches, ufocus, ublend0, ublend1, ublend2;
GLfloat vmatches[4 * 4][3];

// yuv2rgb shader
GLhandleARB prog_yuv2rgb;
GLhandleARB frag_yuv2rgb;
GLint yuv2rgb_y, yuv2rgb_u, yuv2rgb_v;

double zoom;
int zoomi;
int zoomj;
double blend0;
double blend1;
double blend2;

jack_client_t *jclient;
jack_port_t *jporto[2];
jack_port_t *jport[CHANNEL_COUNT_MAX][2];
float jlevel[2][CHANNEL_COUNT_MAX];
int jwhich = 0;

void errorcb(const char *desc) {
  fprintf(stderr, "JACK error: %s\n", desc);
}

void shutdowncb(void *arg) {
  exit(1);
}

int processcb(jack_nframes_t nframes, void *arg) {
  jack_default_audio_sample_t *in[CHANNEL_COUNT_MAX][2], *out[2];
  for (int c = 0; c < count; ++c) {
    for (int k = 0; k < 2; ++k) {
      in[c][k] = (jack_default_audio_sample_t *) jack_port_get_buffer(jport[c][k], nframes);
    }
  }
  out[0] = (jack_default_audio_sample_t *) jack_port_get_buffer(jporto[0], nframes);
  out[1] = (jack_default_audio_sample_t *) jack_port_get_buffer(jporto[1], nframes);
  for (jack_nframes_t i = 0; i < nframes; ++i) {
    for (int k = 0; k < 2; ++k) {
      jack_default_audio_sample_t o = 0;
      for (int c = 0; c < count; ++c) {
        o += jlevel[jwhich][c] * in[c][k][i];
      }
      out[k][i] = o;
    }
  }
  return 0;
}

void realloctexture(GLuint t, GLenum fmt, float r, float g, float b) {
  // resize texture
  glBindTexture(GL_TEXTURE_2D, t);
  glTexImage2D(GL_TEXTURE_2D, 0, fmt, tsize, tsize, 0, fmt, GL_UNSIGNED_BYTE, 0);
  glBindTexture(GL_TEXTURE_2D, 0);
  // clear texture
  glViewport(0, 0, tsize, tsize);
  glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
  glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, t, 0);
  GLenum dbs[] = { GL_COLOR_ATTACHMENT0_EXT };
  glDrawBuffers(1, dbs);
  glClearColor(r, g, b, 0);
  glClear(GL_COLOR_BUFFER_BIT);
  glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, 0, 0);
  glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
}

void reshapecb(int w, int h) {
  if (recorder && (winw != w || winh != h)) {
    record_stop(recorder);
    recorder = 0;
  }
  winw = w;
  winh = h;
  int oldtsize = tsize;
  tsize = roundtwo(max(w, h));
  if (oldtsize != tsize) {
    realloctexture(toutput,   GL_RGBA, 0, 0, 0);
    realloctexture(tmatch[0], GL_RGBA, 0, 0, 0);
    realloctexture(tmatch[1], GL_RGBA, 0, 0, 0);
    realloctexture(tmatch[2], GL_RGBA, 0, 0, 0);
  }
}

void displaycb(void) {
  // upload channels of video to tile sheet
  {
    // upload YUV planes
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texiny);
    for (int c = 0; c < count; ++c) {
      glTexSubImage2D(GL_TEXTURE_2D, 0, (c % 4) * VIDEO_WIDTH, (c / 4) * ↙
          ↳ VIDEO_HEIGHT, VIDEO_WIDTH, VIDEO_HEIGHT, GL_LUMINANCE, ↙
          ↳ GL_UNSIGNED_BYTE, &channels[c]->image[0]);
    }
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, texinu);
    for (int c = 0; c < count; ++c) {
      glTexSubImage2D(GL_TEXTURE_2D, 0, (c % 4) * VIDEO_WIDTH/2, (c / 4) * ↙
          ↳ VIDEO_HEIGHT/2, VIDEO_WIDTH/2, VIDEO_HEIGHT/2, GL_LUMINANCE, ↙
          ↳ GL_UNSIGNED_BYTE, &channels[c]->image[VIDEO_WIDTH * VIDEO_HEIGHT]);
    }
    glActiveTexture(GL_TEXTURE2);
    glBindTexture(GL_TEXTURE_2D, texinv);
    for (int c = 0; c < count; ++c) {
      glTexSubImage2D(GL_TEXTURE_2D, 0, (c % 4) * VIDEO_WIDTH/2, (c / 4) * ↙
          ↳ VIDEO_HEIGHT/2, VIDEO_WIDTH/2, VIDEO_HEIGHT/2, GL_LUMINANCE, ↙
          ↳ GL_UNSIGNED_BYTE, &channels[c]->image[VIDEO_WIDTH * VIDEO_HEIGHT + ↙
          ↳ VIDEO_WIDTH/2 * VIDEO_HEIGHT/2]);
    }
    // convert to RGB
    glViewport(0, 0, 1024, 1024);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, 1, 0, 1, -1, 1);
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
    glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
        ↳ GL_TEXTURE_2D, timages, 0);
    GLenum dbs[] = { GL_COLOR_ATTACHMENT0_EXT };
    glDrawBuffers(1, dbs);
    glUseProgramObjectARB(prog_yuv2rgb);
    glUniform1i(yuv2rgb_y, 0);
    glUniform1i(yuv2rgb_u, 1);
    glUniform1i(yuv2rgb_v, 2);
    glBegin(GL_QUADS); {
      glTexCoord2f(0, 0); glVertex2f(0, 0);
      glTexCoord2f(1, 0); glVertex2f(1, 0);
      glTexCoord2f(1, 1); glVertex2f(1, 1);
      glTexCoord2f(0, 1); glVertex2f(0, 1);
    } glEnd();
    glUseProgramObjectARB(0);
    glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
        ↳ GL_TEXTURE_2D, 0, 0);
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
    glBindTexture(GL_TEXTURE_2D, 0);
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, 0);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, timages);
    glGenerateMipmap(GL_TEXTURE_2D);
    glGetTexImage(GL_TEXTURE_2D, 8 /* hardcoded log2(4*256) - 2 */, GL_RGB, ↙
        ↳ GL_FLOAT, vmatches);
    glBindTexture(GL_TEXTURE_2D, 0);
  }

  // set up zooming view
  glViewport(0, 0, winw, winh);
  {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, GRID_WIDTH, 0, GRID_HEIGHT, -1, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    double dx =               zoomi * GRID_WIDTH  / (GRID_WIDTH  - 1.0);
    double dy = GRID_HEIGHT - zoomj * GRID_HEIGHT / (GRID_HEIGHT - 1.0);
    glTranslatef( dx,  dy, 0);
    glScalef(zoom, zoom, zoom);
    glTranslatef(-dx, -dy, 0);
  }

  // mega shader action
  glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
  glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
      ↳ GL_TEXTURE_2D, toutput,   0);
  glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT1_EXT, ↙
      ↳ GL_TEXTURE_2D, tmatch[0], 0);
  glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT2_EXT, ↙
      ↳ GL_TEXTURE_2D, tmatch[1], 0);
  glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT3_EXT, ↙
      ↳ GL_TEXTURE_2D, tmatch[2], 0);
  GLenum dbs[] = { GL_COLOR_ATTACHMENT0_EXT, GL_COLOR_ATTACHMENT1_EXT, ↙
      ↳ GL_COLOR_ATTACHMENT2_EXT, GL_COLOR_ATTACHMENT3_EXT };
  glDrawBuffers(4, dbs);
  glUseProgramObjectARB(prog);
  glBindTexture(GL_TEXTURE_2D, timages);
  glUniform1i(uimages, 0);
  glUniform3fv(umatches, 12, &vmatches[0][0]);
  glUniform1i(ufocus, focus);
  glUniform1f(ublend0, blend0);
  glUniform1f(ublend1, blend1);
  glUniform1f(ublend2, blend2);
  glBegin(GL_QUADS); {
    glTexCoord2f(0, 1); glVertex2f(0         , 0          );
    glTexCoord2f(1, 1); glVertex2f(GRID_WIDTH, 0          );
    glTexCoord2f(1, 0); glVertex2f(GRID_WIDTH, GRID_HEIGHT);
    glTexCoord2f(0, 0); glVertex2f(0         , GRID_HEIGHT);
  } glEnd();
  glBindTexture(GL_TEXTURE_2D, 0);
  glUseProgramObjectARB(0);
  glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, 0, 0);
  glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT1_EXT, GL_TEXTURE_2D, 0, 0);
  glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT2_EXT, GL_TEXTURE_2D, 0, 0);
  glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT3_EXT, GL_TEXTURE_2D, 0, 0);
  glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);

  // show output image
  glLoadIdentity();
  glBindTexture(GL_TEXTURE_2D, toutput);
  glBegin(GL_QUADS); {
    float tx = winw * 1.0f / tsize;
    float ty = winh * 1.0f / tsize;
    glTexCoord2f(0,  0);  glVertex2f(0         , 0          );
    glTexCoord2f(tx, 0);  glVertex2f(GRID_WIDTH, 0          );
    glTexCoord2f(tx, ty); glVertex2f(GRID_WIDTH, GRID_HEIGHT);
    glTexCoord2f(0,  ty); glVertex2f(0         , GRID_HEIGHT);
  } glEnd();
  glBindTexture(GL_TEXTURE_2D, 0);

  // grab auxiliary images to audio mixer
  glBindTexture(GL_TEXTURE_2D, tmatch[0]);
  glGenerateMipmap(GL_TEXTURE_2D);
  glGetTexImage(GL_TEXTURE_2D, logtwo(tsize), GL_RGBA, GL_FLOAT, &jlevel[1 - jwhich][0]);
  glBindTexture(GL_TEXTURE_2D, tmatch[1]);
  glGenerateMipmap(GL_TEXTURE_2D);
  glGetTexImage(GL_TEXTURE_2D, logtwo(tsize), GL_RGBA, GL_FLOAT, &jlevel[1 - jwhich][4]);
  glBindTexture(GL_TEXTURE_2D, tmatch[2]);
  glGenerateMipmap(GL_TEXTURE_2D);
  glGetTexImage(GL_TEXTURE_2D, logtwo(tsize), GL_RGBA, GL_FLOAT, &jlevel[1 - jwhich][8]);
  glBindTexture(GL_TEXTURE_2D, 0);
  double s = 0;
  for (int c = 0; c < count; ++c) {
    s += jlevel[1 - jwhich][c];
  }
  if (! s) {
    s = 1;
  }
  for (int c = 0; c < count; ++c) {
    jlevel[1 - jwhich][c] /= s;
  }
  jwhich = 1 - jwhich;

  // PPM recording on stdout
  if (recorder) {
    record_frame(recorder);
  }

  glutSwapBuffers();

  // bug free code?
  glutReportErrors();

  // we done a frame
  tframes++;
  frames++;
}

float smatch[4][16][4][16][3];
float matches[4][4][3];

struct timespec clock0;
struct timespec clock1;

void timercb(int v) {
  glutTimerFunc(1, timercb, v);
  clock_gettime(CLOCK_REALTIME, &clock1);
  double dt = (clock1.tv_sec - clock0.tv_sec) + 1.0e-9 * (clock1.tv_nsec - ↙
     ↳ clock0.tv_nsec);
  if (dt < 0.04 * tframes) {
    return;
  }
  if (frames == speed) {
    // refocus
    glBindTexture(GL_TEXTURE_2D, timages);
    glGenerateMipmap(GL_TEXTURE_2D);
    glGetTexImage(GL_TEXTURE_2D, 4, GL_RGB, GL_FLOAT, &smatch[0][0][0][0][0]);
    glGetTexImage(GL_TEXTURE_2D, 8, GL_RGB, GL_FLOAT, &matches[0][0][0]);
    glBindTexture(GL_TEXTURE_2D, 0);
    float ml = 65536.0;
    int   mi = focus;
    for (int c = 0; c < count; ++c) {
      float s = 0;
      for (int k = 0; k < 3; ++k) {
        float ds = smatch[focus / 4][zoomj][focus % 4][zoomi][k] - vmatches[c][k↙
           ↳ ];
        s += ds * ds;
      }
      if (s < ml) {
        ml = s;
        mi = c;
      }
    }
    focus = mi;
    ml = 65536.0;
    for (int c = 0; c < count; ++c) {
      float s = 0;
      for (int k = 0; k < 3; ++k) {
        float ds = matches[focus / 4][focus % 4][k] - vmatches[c][k];
        s += ds * ds;
      }
```

```c
      if (s < ml) {
        ml = s;
        mi = c;
      }
    }
    focus = mi;
    // self-centering random walk
    if (rand() % 8) {
      int coin = rand() % (GRID_WIDTH - 1);
      zoomi += coin < zoomi ? -1 : 1;
    }
    if (rand() % 8) {
      int coin = rand() % (GRID_HEIGHT - 1);
      zoomj += coin < zoomj ? -1 : 1;
    }
    frames = 0;
  }

  // zoom blending
  double k = frames * 1.0 / speed;
  zoom = pow(16, k); // hardcoded power - grid size...
  blend2 = 1 - cos(2 * 3.141592653 * (k + 0) / 3);
  blend1 = 1 - cos(2 * 3.141592653 * (k + 1) / 3);
  blend0 = 1 - cos(2 * 3.141592653 * (k + 2) / 3);
  double blendt = blend0 + blend1 + blend2;
  blend0 /= blendt;
  blend1 /= blendt;
  blend2 /= blendt;

  glutPostRedisplay();
}

int fullscreen = 0;

void keyboardcb(unsigned char key, int x, int y) {
  switch (key) {
  case 'R':
    if (recorder) {
      record_stop(recorder);
      recorder = 0;
    } else {
      recorder = record_start(winw, winh, jclient, jporto);
    }
    break;
  case 'Q':
  case 27:
    exit(0);
    break;
  }
}

void keyspecialcb(int key, int x, int y) {
  switch (key) {
  case GLUT_KEY_F11:
    fullscreen = !fullscreen;
    if (fullscreen) {
      glutFullScreen();
```

```
        glutSetCursor(GLUT_CURSOR_NONE);
      } else {
        glutReshapeWindow(OUTPUT_WIDTH, OUTPUT_HEIGHT);
        glutSetCursor(GLUT_CURSOR_INHERIT);
      }
      break;
    }
}

void exitcb(void) {
  if (recorder) {
    record_stop(recorder);
    recorder = 0;
  }
  jack_client_close(jclient);
  killpg(getpgrp(), SIGKILL); // kill all our processes
  for (int c = 0; c < count; ++c) {
    channel_stop(channels[c]);
  }
  free(channels);
}

int main(int argc, char **argv) {

  // initialisation
  if (argc <= 1) { return 1; }
  count = argc - 1;
  channels = calloc(count, sizeof(struct channel *));

  srand(time(0));
  zoomj = rand() % GRID_HEIGHT;
  zoomi = rand() % GRID_WIDTH;
  glutInitWindowSize(OUTPUT_WIDTH, OUTPUT_HEIGHT);
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
  glutCreateWindow("fractal-channel-hopping");
  glewInit();

  // set up jack first

  jack_set_error_function(errorcb);
  if (!(jclient = jack_client_open("fch", 0, 0))) {
    fprintf (stderr, "jack server not running?\n");
    return 1;
  }
  jack_set_process_callback(jclient, processcb, 0);
  jack_on_shutdown(jclient, shutdowncb, 0);
  for (int c = 0; c < count; ++c) {
    for (int k = 0; k < 2; ++k) {
      char namebuf[64];
      snprintf(namebuf, 62, "%s_%d", argv[c + 1], k + 1);
      namebuf[63] = 0;
      jport[c][k] = jack_port_register(jclient, namebuf, JACK_DEFAULT_AUDIO_TYPE↙
          ↳ , JackPortIsInput, 0);
    }
  }
  for (int k = 0; k < 2; ++k) {
    char namebuf[64];
    snprintf(namebuf, 62, "output_%d", k + 1);
    namebuf[63] = 0;
    jporto[k] = jack_port_register(jclient, namebuf, JACK_DEFAULT_AUDIO_TYPE, ↙
        ↳ JackPortIsOutput, 0);
  }
  if (jack_activate(jclient)) {
    fprintf (stderr, "cannot activate JACK client");
    return 1;
  }
  jack_connect(jclient, "fch:output_1", "system:playback_1");
  jack_connect(jclient, "fch:output_2", "system:playback_2");

  // then start streaming channels which connect to us via jack

  for (int c = 0; c < count; ++c) {
    channels[c] = channel_start(argv[c + 1]);
  }

  // and prepare the display

  glHint(GL_GENERATE_MIPMAP_HINT, GL_FASTEST);
  glEnable(GL_TEXTURE_2D);
  glGenFramebuffersEXT(1, &fbo);

  // YUV planar video
  tsize = 1024;
  glGenTextures(1, &texiny);
  realloctexture(texiny, GL_LUMINANCE, 0, 0, 0);
  glBindTexture(GL_TEXTURE_2D, texiny);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glBindTexture(GL_TEXTURE_2D, 0);
  tsize /= 2;
  glGenTextures(1, &texinu);
  realloctexture(texinu, GL_LUMINANCE, 0.5, 0.5, 0.5);
  glBindTexture(GL_TEXTURE_2D, texinu);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glBindTexture(GL_TEXTURE_2D, 0);
  glGenTextures(1, &texinv);
  realloctexture(texinv, GL_LUMINANCE, 0.5, 0.5, 0.5);
  glBindTexture(GL_TEXTURE_2D, texinv);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glBindTexture(GL_TEXTURE_2D, 0);

  // RGB interleaved video
  tsize = 1024;
  glGenTextures(1, &timages);
  realloctexture(timages, GL_RGBA, 0, 0, 0);
  glBindTexture(GL_TEXTURE_2D, timages);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR)↙
      ↳ ;
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glBindTexture(GL_TEXTURE_2D, 0);
  // RGB output
  tsize = roundtwo(max(OUTPUT_WIDTH, OUTPUT_HEIGHT));
  glGenTextures(1, &toutput);
  realloctexture(toutput, GL_RGBA, 0, 0, 0);
  glBindTexture(GL_TEXTURE_2D, toutput);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glBindTexture(GL_TEXTURE_2D, 0);
  glGenTextures(3, &tmatch[0]);
  realloctexture(tmatch[0], GL_RGBA, 0, 0, 0);
  realloctexture(tmatch[1], GL_RGBA, 0, 0, 0);
  realloctexture(tmatch[2], GL_RGBA, 0, 0, 0);

  // fractalization shader
  GLint success;
  prog = glCreateProgramObjectARB();
  frag = glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);
  glShaderSourceARB(frag, 1, &fch_frag, 0);
  glCompileShaderARB(frag);
  glAttachObjectARB(prog, frag);
  glLinkProgramARB(prog);
  glGetObjectParameterivARB(prog, GL_OBJECT_LINK_STATUS_ARB, &success);
  if (! success) {
    GLhandleARB obj = prog;
    int infologLength = 0;
    int maxLength;
    if (glIsShader(obj)) {
      glGetShaderiv(obj, GL_INFO_LOG_LENGTH, &maxLength);
    } else {
      glGetProgramiv(obj, GL_INFO_LOG_LENGTH, &maxLength);
    }
    char *infoLog = malloc(maxLength);
    if (!infoLog) {
      exit(1);
    }
    if (glIsShader(obj)) {
      glGetShaderInfoLog(obj, maxLength, &infologLength, infoLog);
    } else {
      glGetProgramInfoLog(obj, maxLength, &infologLength, infoLog);
    }
    if (infologLength > 0) {
      fprintf(stderr, "%s\n", infoLog);
    }
    free(infoLog);
    exit(1);
  }
  uimages  = glGetUniformLocationARB(prog, "images");
  umatches = glGetUniformLocationARB(prog, "matches");
  ufocus   = glGetUniformLocationARB(prog, "focus");
  ublend0  = glGetUniformLocationARB(prog, "blend0");
  ublend1  = glGetUniformLocationARB(prog, "blend1");
  ublend2  = glGetUniformLocationARB(prog, "blend2");

  // YUV2RGB shader
  prog_yuv2rgb = glCreateProgramObjectARB();
  frag_yuv2rgb = glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);
  glShaderSourceARB(frag_yuv2rgb, 1, &yuv2rgb_frag, 0);
  glCompileShaderARB(frag_yuv2rgb);
  glAttachObjectARB(prog_yuv2rgb, frag_yuv2rgb);
  glLinkProgramARB(prog_yuv2rgb);
  glGetObjectParameterivARB(prog_yuv2rgb, GL_OBJECT_LINK_STATUS_ARB, &success);
  if (! success) {
    GLhandleARB obj = prog_yuv2rgb;
    int infologLength = 0;
    int maxLength;
    if (glIsShader(obj)) {
      glGetShaderiv(obj, GL_INFO_LOG_LENGTH, &maxLength);
    } else {
      glGetProgramiv(obj, GL_INFO_LOG_LENGTH, &maxLength);
    }
    char *infoLog = malloc(maxLength);
    if (!infoLog) {
      exit(1);
    }
    if (glIsShader(obj)) {
      glGetShaderInfoLog(obj, maxLength, &infologLength, infoLog);
    } else {
      glGetProgramInfoLog(obj, maxLength, &infologLength, infoLog);
    }
    if (infologLength > 0) {
      fprintf(stderr, "%s\n", infoLog);
    }
    free(infoLog);
    exit(1);
  }
  yuv2rgb_y = glGetUniformLocationARB(prog_yuv2rgb, "y");
  yuv2rgb_u = glGetUniformLocationARB(prog_yuv2rgb, "u");
  yuv2rgb_v = glGetUniformLocationARB(prog_yuv2rgb, "v");

  // callbacks
  glutKeyboardFunc(keyboardcb);
  glutSpecialFunc(keyspecialcb);
  glutReshapeFunc(reshapecb);
  glutDisplayFunc(displaycb);
  glutTimerFunc(1, timercb, 1);
  atexit(exitcb);

  // main loop
  clock_gettime(CLOCK_REALTIME, &clock0);
  glutMainLoop();
  return 0;
}
```

18 src/Makefile

```
# fractal-channel-hopping -- infinite fractal television zoom
# Copyright (C) 2011 Claude Heiland-Allen
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.

CC=gcc
CFLAGS=-std=c99 -Wall -pedantic -Wextra -Wno-unused-parameter -O3 -march=native ↙
    ↳ -pthread -lGLEW -lGL -lglut -ljack -lm

BINARY  = fractal-channel-hopping
SOURCES = channel.c record.c pfifo.c list.c main.c
HEADERS = channel.h record.h pfifo.h list.h config.h
GENHEAD = fch.frag.c yuv2rgb.frag.c

all: $(BINARY)

clean:
	-rm $(BINARY) $(GENHEAD)

.SUFFIXES:
.PHONY: all clean

$(BINARY): $(SOURCES) $(HEADERS) $(GENHEAD)
	$(CC) $(CFLAGS) -o $(BINARY) $(SOURCES)

%.frag.c: %.frag s2c.sh
	./s2c.sh $*_frag < $< > $@
```

19 src/pfifo.c

```
/*
  fractal-channel-hopping -- infinite fractal television zoom
  Copyright (C) 2011 Claude Heiland-Allen

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <assert.h>
#include <string.h>
#include "pfifo.h"

void *pfifo_consumerthread(void *);

struct pfifo *pfifo_create(pfifo_consumer *consumer, void *consumerdata) {
  struct pfifo *p = malloc(sizeof(struct pfifo));
  if (!p) return 0;
  list_init(&(p->list));
  p->running = 1;
  p->consumer = consumer;
  p->consumerdata = consumerdata;
  p->mutex = malloc(sizeof(pthread_mutex_t));
  pthread_mutex_init(p->mutex, 0);
  p->nonempty = malloc(sizeof(pthread_cond_t));
  pthread_cond_init(p->nonempty, 0);
  pthread_create(&(p->thread), 0, pfifo_consumerthread, p);
  return p;
}

void pfifo_destroy(struct pfifo *p) {
  p->running = 0;
  pthread_join(p->thread, 0);
  // FIXME proper cleanup...
}

void pfifo_enqueue(struct pfifo *p, size_t length, const void *data) {
  struct pfifo_node *n = malloc(sizeof(struct pfifo_node));
  assert(n);
  n->length = length;
  n->data = malloc(length);
  assert(n->data);
  memcpy(n->data, data, length);
  pthread_mutex_lock(p->mutex);
  list_inserttail(&(p->list), &(n->node));
  pthread_mutex_unlock(p->mutex);
  pthread_cond_signal(p->nonempty);
}

void *pfifo_consumerthread(void *fifo) {
  struct pfifo *p = fifo;
  pthread_mutex_lock(p->mutex);
  while (p->running) {
    while (list_isempty(&(p->list))) pthread_cond_wait(p->nonempty, p->mutex);
    struct pfifo_node *n = (struct pfifo_node *) list_removehead(&(p->list));
    p->consumer(p->consumerdata, n->length, n->data);
    free(n->data);
    free(n);
  }
  pthread_exit(0);
  return 0;
}
```

20 src/pfifo.h

```
/*
  fractal-channel-hopping -- infinite fractal television zoom
  Copyright (C) 2011 Claude Heiland-Allen

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.
  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef PFIFO_H
#define PFIFO_H 1

#include <stdlib.h>
#include <pthread.h>
#include "list.h"

typedef void (pfifo_consumer)(void *, size_t, void *);

struct pfifo_node {
  struct node node;
  size_t length;
  void *data;
};

struct pfifo {
  pfifo_consumer *consumer;
  void *consumerdata;
  struct list list;
  pthread_t thread;
  pthread_mutex_t *mutex;
  pthread_cond_t *nonempty;
  int running;
};

struct pfifo *pfifo_create(pfifo_consumer *consumer, void *consumerdata);
void pfifo_destroy(struct pfifo *fifo);
void pfifo_enqueue(struct pfifo *fifo, size_t length, const void *data);

#endif
```

21 src/record.c

```
/*
  fractal-channel-hopping -- infinite fractal television zoom
  Copyright (C) 2011,2019 Claude Heiland-Allen

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.
  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#define _DEFAULT_SOURCE

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

#include <GL/glew.h>
#include <jack/jack.h>

#include "pfifo.h"
#include "record.h"

void record_writer(void *arg, size_t l, void *data) {
  struct record *record = arg;
  unsigned char *buffer = data;
  fwrite(record->header, strlen(record->header), 1, record->ppm);
  for (int y = record->height - 1; y >= 0; --y) {
    fwrite(buffer + record->width * y * 3, record->width * 3, 1, record->ppm);
  }
  if (! jack_port_connected_to(record->jport[0], "record:input_1")) {
    jack_connect(record->jclient, jack_port_name(record->jport[0]), "record:↙
      ↳ input_1");
  }
  if (! jack_port_connected_to(record->jport[1], "record:input_2")) {
    jack_connect(record->jclient, jack_port_name(record->jport[1]), "record:↙
      ↳ input_2");
      }
}

struct record *record_start(int w, int h, jack_client_t *jclient, jack_port_t **↙
    ↳ jports) {
  struct record *record = malloc(sizeof(struct record));
  record->width = w;
  record->height = h;
  record->bytes = w * h * 3;
  record->buffer = malloc(record->bytes);
  snprintf(record->header, 62, "P6\n%d %d 255\n", w, h);
  record->header[63] = 0;
  const char *vfmt = "ppmtoy4m -v0 -S444 -F25:1 2>'tmp/record.ppm.log' | ffmpeg ↙
      ↳ -loglevel 0 -f yuv4mpegpipe -i pipe:- -ac 2 -f jack -i record -target ↙
      ↳ pal-dvd -shortest 'fractal-channel-hopping_%Y-%m-%d_%H-%M-%S_%z.mpeg' >'↙
      ↳ tmp/record.ffmpeg.log' 2>&1";
  int vlen = strlen(vfmt) + 64;
  char *vcmd = malloc(vlen);
  time_t t = time(NULL);
  struct tm tm;
  localtime_r(&t, &tm);
  if (0 == strftime(vcmd, vlen, vfmt, &tm)) {
    return 0;
  }
  record->jclient = jclient;
  record->jport[0] = jports[0];
  record->jport[1] = jports[1];
  record->ppm = popen(vcmd, "w");
  record->pfifo = pfifo_create(record_writer, record);
  return record;
}

void record_frame(struct record *record) {
  glReadPixels(0, 0, record->width, record->height, GL_RGB, GL_UNSIGNED_BYTE, ↙
      ↳ record->buffer);
  pfifo_enqueue(record->pfifo, record->bytes, record->buffer);
}

void record_stop(struct record *record) {
  pfifo_destroy(record->pfifo);
  pclose(record->ppm);
  free(record->buffer);
  free(record);
}
```

22 src/record.h

```
/*
  fractal-channel-hopping -- infinite fractal television zoom
  Copyright (C) 2011 Claude Heiland-Allen

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef RECORD_H
#define RECORD_H 1

#include <stdio.h>

#include <jack/jack.h>

#include "pfifo.h"

struct record {
  int width;
  int height;
  int bytes;
  unsigned char *buffer;
  char header[64];
  FILE *ppm;
  FILE *wav;
  struct pfifo *pfifo;
  jack_client_t *jclient;
  jack_port_t *jport[2];
};

struct record *record_start(int w, int h, jack_client_t *jclient, jack_port_t **jports);
void record_frame(struct record *record);
void record_stop(struct record *record);

#endif
```

23 src/s2c.sh

```
#!/bin/bash
# fractal-channel-hopping -- infinite fractal television zoom
# Copyright (C) 2011 Claude Heiland-Allen
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.
echo "/* machine-generated file, do not edit */"
echo "static const char *$1 ="
sed 's|^|"|' |
sed 's|$|\\n"|'
echo ";"
```

24 src/yuv2rgb.frag

```
/*
  fractal-channel-hopping -- infinite fractal television zoom
  Copyright (C) 2011 Claude Heiland-Allen

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#version 130
```

```

uniform sampler2D y;
uniform sampler2D u;
uniform sampler2D v;

void main(void) {
  const mat4 m = mat4(
    1.1643828125,  0.0,           1.59602734375, -0.87078515625,
    1.1643828125, -0.39176171875, -0.81296875,     0.52959375,
    1.1643828125,  2.017234375,    0.0,           -1.081390625,
    0.0,           0.0,            0.0,            1.0
  );
  vec2 p = gl_TexCoord[0].xy;
  vec3 yuv = vec3(texture(y, p).r, texture(u, p).r, texture(v, p).r);
  gl_FragData[0] = vec4(yuv, 1.0) * m;
}
```

25 start.sh

```
#!/bin/bash
# fractal-channel-hopping -- infinite fractal television zoom
# Copyright (C) 2011,2019 Claude Heiland-Allen
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.
#
S="$(dirname "$(readlink -f "$0")")"
FCH="${S}/src/fractal-channel-hopping"
CHANNELS=(Channel*)
make -C "${S}/src" &&
mkdir -p tmp &&
for c in "${CHANNELS[@]}"
do
  rm -f "tmp/${c}.fifo"
  mkfifo "tmp/${c}.fifo"
done &&
"${FCH}" "${CHANNELS[@]}"
```