cca

Claude Heiland-Allen

2016–2017

Contents

1 blurx_frag.glsl

```
#version 330 core

uniform sampler2DArray source;
uniform sampler1D kernel;

uniform int layer;
uniform float blur;

in vec2 coord;

layout(location = 0) out float blurred;

void main()
{
  ivec2 s = textureSize(source, 0).xy;
  ivec2 p = ivec2(floor(vec2(s) * coord));
  float r = pow(2.0, blur);
  int k = int(ceil(r));
  vec2 b = vec2(0.0);
  for (int i = -k; i <= k; ++i)
  {
    float x = 0.5 + 0.5 * float(i)/r;
    float k = texture(kernel, x).x;
```

```
    b += k * vec2(texelFetch(source, ivec3((p.x + i + s.x) % s.x, p.y, layer), ↙
        ↳ 0).x, 1.0);
  }
  blurred = b.x / b.y;
}
```

2 blury_frag.glsl

```
#version 330 core

uniform sampler2D source;
uniform sampler1D kernel;

uniform float blur;

in vec2 coord;

layout(location = 0) out float blurred;

void main()
{
  ivec2 s = textureSize(source, 0).xy;
  ivec2 p = ivec2(floor(vec2(s) * coord));
  float r = pow(2.0, blur);
  int k = int(ceil(r));
  vec2 b = vec2(0.0);
  for (int i = -k; i <= k; ++i)
  {
    float y = 0.5 + 0.5 * float(i)/r;
    float k = texture(kernel, y).x;
    b += k * vec2(texelFetch(source, ivec2(p.x, (p.y + i + s.y) % s.y), 0).x, ↙
        ↳ 1.0);
  }
  blurred = b.x / b.y;
}
```

3 cca.c

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

#include <GL/glew.h>
#include <GLFW/glfw3.h>

#undef WANT_SOUND
// #define WANT_SOUND

#ifdef WANT_SOUND
#include <jack/jack.h>
#include <sndfile.h>
#endif

const float pi = 3.141592653589793;
```

```
20  const int width = 1920;
    const int height = 1080;
    const int colour_width = 1920;
    const int colour_height = 1080;
    int mutation = 0;
25  int overdrive = 1;
    int record = 0;
    int highquality = 0;
    int rt = 1;
    #define FPS 25
30
    #ifdef WANT_SOUND

    #define SR 48000

35  typedef struct { double y; } HIP;
    static inline double hip(HIP *s, double x, double hz) {
      double c = fmin(fmax(1 - 2 * pi * hz / SR, 0), 1);
      double n = 0.5 * (1 + c);
      double y = x + c * s->y;
40    double o = n * (y - s->y);
      s->y = y;
      return o;
    }

45  float audio[4][1920];
    HIP filter[2];
    jack_client_t *client;
    jack_port_t *port[2];
    SNDFILE *sndfile;
50  int ix;

    void audiocb(float *l, float *r, int nframes) {
      float g = 1;
      for (int i = 0; i < nframes; ++i) {
55      float h = i / (float) nframes;
        float h1 = 1 - h;
        l[i] = tanhf(g * hip(&filter[0], h1 * audio[0][ix] + h  * audio[2][ix], 5));
        r[i] = tanhf(g * hip(&filter[1], h1 * audio[1][ix] + h  * audio[3][ix], 5));
        ix = (ix + 1) % 1920;
60    }
      memcpy(&audio[0][0], &audio[2][0], 1920 * sizeof(float));
      memcpy(&audio[1][0], &audio[3][0], 1920 * sizeof(float));
    }

65  static volatile int incb = 0;
    int processcb(jack_nframes_t nframes, void *arg) {
      (void) arg;
      jack_default_audio_sample_t *out[2];
      out[0] = jack_port_get_buffer(port[0], nframes);
70    out[1] = jack_port_get_buffer(port[1], nframes);
      incb = 1;
      audiocb(out[0], out[1], nframes);
      incb = 0;
      return 0;
75  }
```

```
int startaudio() {
  if (rt) {
    if (! (client = jack_client_open("cca", JackNoStartServer, 0))) {
      fprintf(stderr, "jack server not running?\n");
      exit(1);
    }
    jack_set_process_callback(client, processcb, 0);
    port[0] = jack_port_register(client, "output_1", JACK_DEFAULT_AUDIO_TYPE, ↙
        ↳ JackPortIsOutput, 0);
    port[1] = jack_port_register(client, "output_2", JACK_DEFAULT_AUDIO_TYPE, ↙
        ↳ JackPortIsOutput, 0);
    if (jack_activate(client)) {
      fprintf (stderr, "cannot activate JACK client");
      exit(1);
    }
    if (jack_connect(client, "cca:output_1", "system:playback_1")) {
      fprintf(stderr, "cannot connect output port\n");
    }
    if (jack_connect(client, "cca:output_2", "system:playback_2")) {
      fprintf(stderr, "cannot connect output port\n");
    }
  } else {
    SF_INFO info = { 0, 48000, 2, SF_FORMAT_WAV | SF_FORMAT_FLOAT, 0, 0 };
    sndfile = sf_open("cca.wav", SFM_WRITE, &info);
  }
  return 1;
}

#endif

void debug_program(GLuint program) {
  GLint status = 0;
  glGetProgramiv(program, GL_LINK_STATUS, &status);
  GLint length = 0;
  glGetProgramiv(program, GL_INFO_LOG_LENGTH, &length);
  char *info = 0;
  if (length) {
    info = malloc(length + 1);
    info[0] = 0;
    glGetProgramInfoLog(program, length, 0, info);
    info[length] = 0;
  }
  if ((info && info[0]) || ! status) {
    fprintf(stderr, "program link info:\n%s", info ? info : "(no info log)");
  }
  if (info) {
    free(info);
  }
}

void debug_shader(GLuint shader, GLenum type, const char *source) {
  GLint status = 0;
  glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
  GLint length = 0;
  glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &length);
  char *info = 0;
  if (length) {
```

```
    info = malloc(length + 1);
    info[0] = 0;
    glGetShaderInfoLog(shader, length, 0, info);
    info[length] = 0;
  }
  if ((info && info[0]) || ! status) {
    const char *type_str = "unknown";
    switch (type) {
      case GL_VERTEX_SHADER: type_str = "vertex"; break;
      case GL_FRAGMENT_SHADER: type_str = "fragment"; break;
      case GL_COMPUTE_SHADER: type_str = "compute"; break;
    }
    fprintf(stderr, "%s shader compile info:\n%s\nshader source:\n%s", type_str,↙
        ↪ info ? info : "(no info log)", source ? source : "(no source)");
  }
  if (info) {
    free(info);
  }
}

GLuint vertex_fragment_shader(const char *vert, const char *frag) {
  GLuint program = glCreateProgram();
  {
    GLuint shader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(shader, 1, &vert, 0);
    glCompileShader(shader);
    debug_shader(shader, GL_VERTEX_SHADER, vert);
    glAttachShader(program, shader);
    glDeleteShader(shader);
  }
  {
    GLuint shader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(shader, 1, &frag, 0);
    glCompileShader(shader);
    debug_shader(shader, GL_FRAGMENT_SHADER, frag);
    glAttachShader(program, shader);
    glDeleteShader(shader);
  }
  glLinkProgram(program);
  debug_program(program);
  return program;
}

char *read_file(const char *name) {
  FILE *f = fopen(name, "r");
  fseek(f, 0, SEEK_END);
  long len = ftell(f);
  fseek(f, 0, SEEK_SET);
  char *str = malloc(len + 1);
  fread(str, len, 1, f);
  str[len] = 0;
  fclose(f);
  return str;
}

struct genome {
  float blur[4];
```

```
  float speed[4];
  float decay[4];
  float coupling[4][4];
  float colour[4][4];
  float offset[4];
};

struct genome current, mutated;

void mix(struct genome *a, const struct genome *b, double p)
{
  float *x = &a->blur[0];
  const float *y = &b->blur[0];
  for (int i = 0; i < 48; ++i)
  {
    x[i] = (rand() / (double) RAND_MAX) < p ? y[i] : x[i];
  }
}

void randomize(struct genome *s)
{
#define R (rand() / (double) RAND_MAX)
  s->blur[0] = 1 + 7 * R;
  s->blur[1] = 1 + 7 * R;
  s->blur[2] = 1 + 7 * R;
  s->blur[3] = 1 + 7 * R;
  s->speed[0] = 12 * R;
  s->speed[1] = 12 * R;
  s->speed[2] = 12 * R;
  s->speed[3] = 12 * R;
  s->decay[0] = 1 - pow(2, -12 * R);
  s->decay[1] = 1 - pow(2, -12 * R);
  s->decay[2] = 1 - pow(2, -12 * R);
  s->decay[3] = 1 - pow(2, -12 * R);
  for (int i = 0; i < 4; ++i)
    for (int j = 0; j < 4; ++j)
      s->coupling[i][j] = (i == j ? 2 : 1) * (2 * R - 1);
  for (int i = 0; i < 4; ++i)
    for (int j = 0; j < 4; ++j)
      s->colour[i][j] = 2 * R - 1;
  for (int i = 0; i < 4; ++i)
    s->offset[i] = R;
#undef R
}

void reseed()
{
  int count = width * height * 4;
  float *noise = malloc(count * sizeof(float));
  for (int j = 0; j < 4; ++j)
  {
    for (int i = 0; i < count; ++i)
      noise[i] = (j < 2) * rand() / (double) RAND_MAX;
    glActiveTexture(GL_TEXTURE0 + j);
    glTexImage3D(GL_TEXTURE_2D_ARRAY, 0, GL_R32F, width, height, 4, 0, GL_RED, ⬐
        ↳ GL_FLOAT, noise);
    if (j < 2)
```

```
        glGenerateMipmap(GL_TEXTURE_2D_ARRAY);
    }
    free(noise);
}

void print_parameters(const struct genome *s)
{
    fprintf(stderr, "%.8f", s->blur[0]);
    for (int i = 1; i < 4; ++i)
        fprintf(stderr, " %.8f", s->blur[i]);
    fprintf(stderr, " /");
    for (int i = 0; i < 4; ++i)
        fprintf(stderr, " %.8f", s->speed[i]);
    fprintf(stderr, " /");
    for (int i = 0; i < 4; ++i)
        fprintf(stderr, " %.8f", s->decay[i]);
    for (int i = 0; i < 4; ++i)
    {
        fprintf(stderr, " /");
        for (int j = 0; j < 4; ++j)
            fprintf(stderr, " %.8f", s->coupling[i][j]);
    }
    for (int i = 0; i < 4; ++i)
    {
        fprintf(stderr, " /");
        for (int j = 0; j < 4; ++j)
            fprintf(stderr, " %.8f", s->colour[i][j]);
    }
    fprintf(stderr, " /");
    for (int i = 0; i < 4; ++i)
        fprintf(stderr, " %.8f", s->offset[i]);
    fprintf(stderr, "\n");
}

int parse_parameters(struct genome *s)
{
    if (48 != scanf(
        "%f %f %f %f / %f %f %f %f / %f %f %f %f / "
        "%f %f %f %f / %f %f %f %f / %f %f %f %f / %f %f %f %f / "
        "%f %f %f %f / %f %f %f %f / %f %f %f %f / %f %f %f %f / "
        "%f %f %f %f",
        &s->blur[0], &s->blur[1], &s->blur[2], &s->blur[3],
        &s->speed[0], &s->speed[1], &s->speed[2], &s->speed[3],
        &s->decay[0], &s->decay[1], &s->decay[2], &s->decay[3],
        &s->coupling[0][0], &s->coupling[0][1], &s->coupling[0][2], &s->coupling
            ↪ [0][3],
        &s->coupling[1][0], &s->coupling[1][1], &s->coupling[1][2], &s->coupling
            ↪ [1][3],
        &s->coupling[2][0], &s->coupling[2][1], &s->coupling[2][2], &s->coupling
            ↪ [2][3],
        &s->coupling[3][0], &s->coupling[3][1], &s->coupling[3][2], &s->coupling
            ↪ [3][3],
        &s->colour[0][0], &s->colour[0][1], &s->colour[0][2], &s->colour[0][3],
        &s->colour[1][0], &s->colour[1][1], &s->colour[1][2], &s->colour[1][3],
        &s->colour[2][0], &s->colour[2][1], &s->colour[2][2], &s->colour[2][3],
        &s->colour[3][0], &s->colour[3][1], &s->colour[3][2], &s->colour[3][3],
        &s->offset[0], &s->offset[1], &s->offset[2], &s->offset[3]
```

```
  ))
  {
    fprintf(stderr, "cca: error: parse failed\n");
    return 0;
  }
  return 1;
}

void keycb(GLFWwindow *window, int key, int scancode, int action, int mods) {
  (void) scancode;
  (void) mods;
  if (action == GLFW_PRESS) {
    switch (key) {
      case GLFW_KEY_Q:
      case GLFW_KEY_ESCAPE:
        glfwSetWindowShouldClose(window, GL_TRUE);
        break;
      case GLFW_KEY_SPACE:
        randomize(&current);
        break;
      case GLFW_KEY_ENTER:
        reseed();
        break;
      case GLFW_KEY_O:
        overdrive = 256 / overdrive;
        fprintf(stderr, "overdrive %s\n", overdrive == 1 ? "disabled" : "enabled
            ");
        break;
      case GLFW_KEY_M:
        mutation = 1 - mutation;
        fprintf(stderr, "mutation %s\n", mutation == 0 ? "disabled" : "enabled")
            ;
        break;
      case GLFW_KEY_H:
        highquality = 1 - highquality;
        fprintf(stderr, "high quality %s\n", highquality == 0 ? "disabled" : "
            enabled");
        break;
      case GLFW_KEY_P:
        print_parameters(&current);
        break;
      case GLFW_KEY_L:
        parse_parameters(&current);
        break;
      case GLFW_KEY_R:
        record += 250;
        break;
      case GLFW_KEY_S:
        record += 1;
        break;
    }
  }
}

int main()
{
  srand(time(0));
```

```
  glfwInit();
  glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_API);
  glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
  glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
  glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
  glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
  glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
//  glfwWindowHint(GLFW_DECORATED, GL_FALSE);
  GLFWwindow *window = glfwCreateWindow(width, height, "Coupled Cellular Automata", 0, 0);
  glfwMakeContextCurrent(window);
  glewExperimental = GL_TRUE;
  glewInit();
  glGetError(); // discard common error from glew
  glfwSetKeyCallback(window, keycb);

  GLuint tex[5];
  glGenTextures(5, &tex[0]);
  int count = width * height * 4;
  float *image = malloc(count * sizeof(float));
  for (int j = 0; j < 5; ++j)
  {
    for (int i = 0; i < count; ++i) image[i] = rand() / (double) RAND_MAX;
    glActiveTexture(GL_TEXTURE0 + j);
    glBindTexture(GL_TEXTURE_2D_ARRAY, tex[j]);
    glTexImage3D(GL_TEXTURE_2D_ARRAY, 0, GL_R32F, width, height, 4, 0, GL_RED, GL_FLOAT, image);
    glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_MIN_FILTER, j < 2 ? GL_LINEAR_MIPMAP_LINEAR : GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_MAG_FILTER, j < 2 ? GL_LINEAR : GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_WRAP_T, GL_REPEAT);
  }

  // temporary texture for blurring
  GLuint btex;
  glGenTextures(1, &btex);
  glActiveTexture(GL_TEXTURE0 + 5);
  glBindTexture(GL_TEXTURE_2D, btex);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_R32F, width, height, 0, GL_RED, GL_FLOAT, image);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

  // Gaussian kernel
  float kernel[256];
  for (int i = 0; i < 256; ++i)
  {
    float x = 3.0f * ((i + 0.5f) / 256.0f - 0.5f);
    kernel[i] = expf(-x * x);
  }
  GLuint ktex;
  glGenTextures(1, &ktex);
```

```
    glActiveTexture(GL_TEXTURE0 + 6);
    glBindTexture(GL_TEXTURE_1D, ktex);
405 glTexImage1D(GL_TEXTURE_1D, 0, GL_R32F, 256, 0, GL_RED, GL_FLOAT, &kernel[0]);
    glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR)↙
        ↳ ;
    glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glGenerateMipmap(GL_TEXTURE_1D);
410
    GLenum status;

    // framebuffer for blurring
    GLuint fbob[5];
415 glGenFramebuffers(5, &fbob[0]);
    GLenum buffersb = GL_COLOR_ATTACHMENT0;
    glBindFramebuffer(GL_FRAMEBUFFER, fbob[4]);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ↙
        ↳ btex, 0);
    glDrawBuffers(1, &buffersb);
420 status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
    if (status != GL_FRAMEBUFFER_COMPLETE) fprintf(stderr, "FBO ERR %d\n", status)↙
        ↳ ;
    for (int i = 0; i < 4; ++i)
    {
      glBindFramebuffer(GL_FRAMEBUFFER, fbob[i]);
425   glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, tex[4], 0, i↙
          ↳ );
      glDrawBuffers(1, &buffersb);
      status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
      if (status != GL_FRAMEBUFFER_COMPLETE) fprintf(stderr, "FBO ERR %d\n", ↙
          ↳ status);
    }
430 // framebuffers for ping pong rendering
    GLuint fbo8[2];
    glGenFramebuffers(2, &fbo8[0]);
    GLenum buffers8[8] =
      { GL_COLOR_ATTACHMENT0
435   , GL_COLOR_ATTACHMENT1
      , GL_COLOR_ATTACHMENT2
      , GL_COLOR_ATTACHMENT3
      , GL_COLOR_ATTACHMENT4
      , GL_COLOR_ATTACHMENT5
440   , GL_COLOR_ATTACHMENT6
      , GL_COLOR_ATTACHMENT7
      };
    // ping
    glBindFramebuffer(GL_FRAMEBUFFER, fbo8[0]);
445 glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, tex[1], 0, 0);
    glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT1, tex[1], 0, 1);
    glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT2, tex[1], 0, 2);
    glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT3, tex[1], 0, 3);
    glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT4, tex[3], 0, 0);
450 glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT5, tex[3], 0, 1);
    glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT6, tex[3], 0, 2);
    glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT7, tex[3], 0, 3);
    glDrawBuffers(8, buffers8);
    status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
```

```
455     if (status != GL_FRAMEBUFFER_COMPLETE) fprintf(stderr, "FBO ERR %d\n", status)↙
            ↳ ;
        // pong
        glBindFramebuffer(GL_FRAMEBUFFER, fbo8[1]);
        glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, tex[0], 0, 0);
        glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT1, tex[0], 0, 1);
460     glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT2, tex[0], 0, 2);
        glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT3, tex[0], 0, 3);
        glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT4, tex[2], 0, 0);
        glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT5, tex[2], 0, 1);
        glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT6, tex[2], 0, 2);
465     glFramebufferTextureLayer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT7, tex[2], 0, 3);
        glDrawBuffers(8, buffers8);
        status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
        if (status != GL_FRAMEBUFFER_COMPLETE) fprintf(stderr, "FBO ERR %d\n", status)↙
            ↳ ;

470     // texture and framebuffer for colouring
        GLuint ctex;
        glGenTextures(1, &ctex);
        int ccount = colour_width * colour_height * 4;
        unsigned char *cimage = malloc(ccount);
475     for (int i = 0; i < ccount; ++i)
          cimage[i] = 255;
        glActiveTexture(GL_TEXTURE0 + 7);
        glBindTexture(GL_TEXTURE_2D, ctex);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, colour_width, colour_height, 0, ↙
            ↳ GL_RGBA, GL_UNSIGNED_BYTE, cimage);
480     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, colour_width != width ||↙
            ↳  colour_height != height ? GL_LINEAR_MIPMAP_LINEAR : GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, colour_width != width ||↙
            ↳  colour_height != height ? GL_LINEAR : GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        if (colour_width != width || colour_height != height)
485       glGenerateMipmap(GL_TEXTURE_2D);
        GLuint fboc;
        glGenFramebuffers(1, &fboc);
        GLenum buffersc = GL_COLOR_ATTACHMENT0;
        glBindFramebuffer(GL_FRAMEBUFFER, fboc);
490     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ↙
            ↳ ctex, 0);
        glDrawBuffers(1, &buffersc);
        status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
        if (status != GL_FRAMEBUFFER_COMPLETE) fprintf(stderr, "FBO ERR %d\n", status)↙
            ↳ ;

495     char *vert = read_file("cca_vert.glsl");
        char *frag = read_file("cca_frag.glsl");
        GLuint p_update = vertex_fragment_shader(vert, frag);
        free(frag);
        GLint u_blur     = glGetUniformLocation(p_update, "blur");
500     GLint u_speed    = glGetUniformLocation(p_update, "speed");
        GLint u_decay    = glGetUniformLocation(p_update, "decay");
        GLint u_coupling = glGetUniformLocation(p_update, "coupling");
        GLint u_state    = glGetUniformLocation(p_update, "state");
        GLint u_history  = glGetUniformLocation(p_update, "history");
```

```
505     GLint u_blurred  = glGetUniformLocation(p_update, "blurred");
        GLint u_highquality = glGetUniformLocation(p_update, "highquality");
        glUseProgram(p_update);
        glUniform1i(u_blurred, 4);
        frag = read_file("colour_frag.glsl");
510     GLuint p_colour = vertex_fragment_shader(vert, frag);
        free(frag);
        GLint u_cstate  = glGetUniformLocation(p_colour, "state");
        GLint u_ccolour = glGetUniformLocation(p_colour, "colour");
        GLint u_coffset = glGetUniformLocation(p_colour, "offset");
515     frag = read_file("blurx_frag.glsl");
        GLuint p_blurx = vertex_fragment_shader(vert, frag);
        free(frag);
        GLuint u_kernelx = glGetUniformLocation(p_blurx, "kernel");
        GLuint u_sourcex = glGetUniformLocation(p_blurx, "source");
520     GLuint u_layerx  = glGetUniformLocation(p_blurx, "layer");
        GLuint u_blurx   = glGetUniformLocation(p_blurx, "blur");
        glUseProgram(p_blurx);
        glUniform1i(u_kernelx, 6);
        frag = read_file("blury_frag.glsl");
525     GLuint p_blury = vertex_fragment_shader(vert, frag);
        free(frag);
        GLuint u_kernely = glGetUniformLocation(p_blury, "kernel");
        GLuint u_sourcey = glGetUniformLocation(p_blury, "source");
        GLuint u_blury   = glGetUniformLocation(p_blury, "blur");
530     glUseProgram(p_blury);
        glUniform1i(u_kernely, 6);
        glUniform1i(u_sourcey, 5);
        frag = read_file("display_frag.glsl");
        GLuint p_display = vertex_fragment_shader(vert, frag);
535     free(frag);
        GLint u_dtex = glGetUniformLocation(p_display, "tex");
        glUseProgram(p_display);
        glUniform1i(u_dtex, 7);
        free(vert);
540
        GLuint vao;
        glGenVertexArrays(1, &vao);
        glBindVertexArray(vao);
        glViewport(0, 0, width, height);
545
    #ifdef WANT_SOUND
        startaudio();
    #endif

550     int which = 0;
        randomize(&current);
        reseed();
        int frame = 0;
        while (! glfwWindowShouldClose(window))
555     {
          glfwPollEvents();

          glUseProgram(p_update);
          glUniform4fv(u_blur,  1, &current.blur [0]);
560       glUniform4fv(u_speed, 1, &current.speed[0]);
          glUniform4fv(u_decay, 1, &current.decay[0]);
```

```
    glUniformMatrix4fv(u_coupling, 1, GL_FALSE, &current.coupling[0][0]);
    for (int o = 0; o < overdrive; ++o)
    {
      if (mutation)
      {
        randomize(&mutated);
        mix(&current, &mutated, 0.001);
      }
      if (highquality)
        for (int i = 0; i < 4; ++i)
        {
          glUseProgram(p_blurx);
          glUniform1i(u_sourcex, which);
          glUniform1i(u_layerx, i);
          glUniform1f(u_blurx, current.blur[i]);
          glBindFramebuffer(GL_FRAMEBUFFER, fbob[4]);
          glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
          glUseProgram(p_blury);
          glUniform1f(u_blury, current.blur[i]);
          glBindFramebuffer(GL_FRAMEBUFFER, fbob[i]);
          glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
        }
      glUseProgram(p_update);
      glUniform1i(u_highquality, highquality);
      glUniform1i(u_state, which);
      glUniform1i(u_history, which + 2);
      glBindFramebuffer(GL_FRAMEBUFFER, fbo8[which]);
      glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
      which = 1 - which;
      glActiveTexture(GL_TEXTURE0 + which);
      glGenerateMipmap(GL_TEXTURE_2D_ARRAY);
    }

#ifdef WANT_SOUND
    if (rt)
      while (incb)
        ;
    glReadPixels(0, 270, 1920, 1, GL_RED, GL_FLOAT, &audio[2][0]);
    glReadPixels(0, 810, 1920, 1, GL_RED, GL_FLOAT, &audio[3][0]);
    if (! rt)
    {
      float audioo[2][SR/FPS];
      audiocb(&audioo[0][0], &audioo[1][0], SR/FPS);
      float frames[SR/FPS][2];
      for (int i = 0; i < SR/FPS; ++i)
        for (int c = 0; c < 2; ++c)
          frames[i][c] = audioo[c][i];
      sf_writef_float(sndfile, &frames[0][0], SR/FPS);
    }
#endif

    glUseProgram(p_colour);
    glUniform1i(u_cstate, which);
    glUniformMatrix4fv(u_ccolour, 1, GL_FALSE, &current.colour[0][0]);
    glUniform4fv(u_coffset, 1, &current.offset[0]);
    glBindFramebuffer(GL_FRAMEBUFFER, fboc);
    glViewport(0, 0, colour_width, colour_height);
```

```
      glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
620   glActiveTexture(GL_TEXTURE0 + 7);
      if (colour_width != width || colour_height != height)
        glGenerateMipmap(GL_TEXTURE_2D);
      if (record > 0)
      {
625     glReadPixels(0, 0, colour_width, colour_height, GL_RGB, GL_UNSIGNED_BYTE, ↙
            ↳ cimage);
        fprintf(stdout, "P6\n%d %d\n255\n", colour_width, colour_height);
        fwrite(cimage, colour_width * colour_height * 3, 1, stdout);
        fflush(stdout);
        record--;
630   }
      glUseProgram(p_display);
      glBindFramebuffer(GL_FRAMEBUFFER, 0);
      glViewport(0, 0, width, height);
      glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
635   glfwSwapBuffers(window);
      int e = glGetError();
      if (e)
        fprintf(stderr, "%d\n", e);
      frame++;
640   if (! rt && frame >= FPS * 300) {
#ifdef WANT_SOUND
      sf_close(sndfile);
#endif
      break;
645   }
    }
    fprintf(stderr, "%d frames rendered\n", frame);
    glfwTerminate();
    return 0;
650 }
```

4 cca_frag.glsl

```
#version 330 core

uniform sampler2DArray state;
uniform sampler2DArray history;
5 uniform sampler2DArray blurred;

uniform bool highquality;
uniform vec4 blur;
uniform mat4 coupling;
10 uniform vec4 speed;
uniform vec4 decay;

in vec2 coord;

15 layout(location = 0) out float state_out0;
layout(location = 1) out float state_out1;
layout(location = 2) out float state_out2;
layout(location = 3) out float state_out3;
layout(location = 4) out float history_out0;
20 layout(location = 5) out float history_out1;
layout(location = 6) out float history_out2;
```

```
layout(location = 7) out float history_out3;

void main()
{
  vec4 s1, s100, s, h;
  for (int k = 0; k < 4; ++k)
  {
    vec3 p = vec3(coord, float(k));
    s1[k] = texture(state, p, 1.0).x;
    s100[k] = texture(state, p, 100.0).x;
    if (highquality)
      s[k] = texture(blurred, p).x;
    else
      s[k] = texture(state, p, blur[k]).x;
    h[k] = texture(history, p).x;
  }
  s = coupling * (s - s100) + h;
  s = speed * s;
  s = mix(s1, vec4(0.5) + 0.5 * cos(s), 0.125);
  h = mix(s, h, decay);
  state_out0 = s[0];
  state_out1 = s[1];
  state_out2 = s[2];
  state_out3 = s[3];
  history_out0 = h[0];
  history_out1 = h[1];
  history_out2 = h[2];
  history_out3 = h[3];
}
```

5 cca_vert.glsl

```
#version 330 core

out vec2 coord;

void main()
{
  float x = float(gl_VertexID & 1);
  float y = float(gl_VertexID & 2) / 2.0;
  vec2 p = vec2(x, y);
  gl_Position = vec4(2.0 * p - vec2(1.0), 0.0, 1.0);
  coord = p;
}
```

6 colloid.cca

```
5.25743628 6.56365585 2.14442182 7.81166124 / 2.39061189 6.82049799 9.31450653 ↙
    ↘ 10.75788498 / 0.98226684 0.98571151 0.46472713 0.99911839 / 1.12231863 ↙
    ↘ 0.57909209 0.63941175 -0.53162605 / -0.95472813 1.46051788 0.99605125 ↙
    ↘ 0.12737699 / 0.73561740 0.79917657 -0.89208925 0.15141620 / -0.73087114 ↙
    ↘ -0.16872367 -0.56519902 0.65794528 / 0.48887905 0.54333013 0.05433431 ↙
    ↘ -0.19676182 / 0.18424408 0.59043974 -0.24384657 0.58267939 / -0.27281055 ↙
    ↘ -0.69142890 0.37566018 0.69675756 / 0.33007154 0.52593577 0.38802323 ↙
    ↘ -0.10876913 / 0.55251396 0.51371747 0.67980242 0.57514989
```

7 colour_frag.glsl

```
#version 330 core

uniform sampler2DArray state;

uniform mat4 colour;
uniform vec4 offset;

in vec2 coord;

out vec4 colour_out;

float cubic_interp(float t, vec4 a)
{
  const mat4 m = mat4(0.0,2.0,0.0,0.0, -1.0,0.0,1.0,0.0, 2.0,-5.0,4.0,-1.0, ↙
      ↪ -1.0,3.0,-3.0,1.0);
  vec4 f = vec4(1.0, t, t*t, t*t*t);
  return dot(0.5 * f * transpose(m), a);
}

float cubic_row(sampler2DArray tex, ivec2 p, int layer, int lod, float t)
{
  ivec2 s = textureSize(tex, lod).xy;
  float a = texelFetch(tex, ivec3((p + ivec2(s.x - 1, 0)) % s, layer), lod).x;
  float b = texelFetch(tex, ivec3( p                       , layer), lod).x;
  float c = texelFetch(tex, ivec3((p + ivec2(1      , 0)) % s, layer), lod).x;
  float d = texelFetch(tex, ivec3((p + ivec2(2      , 0)) % s, layer), lod).x;
  return cubic_interp(t, vec4(a, b, c, d));
}

float cubic(sampler2DArray tex, vec2 p, int layer, int lod)
{
  ivec2 s = textureSize(tex, lod).xy;
  vec2 q = mod(vec2(s) * (p + vec2(1.0)) - vec2(0.5), s);
  ivec2 r = ivec2(floor(q));
  vec2 t = q - vec2(r);
  float a = cubic_row(tex, (r + ivec2(0, s.y - 1)) % s, layer, lod, t.x);
  float b = cubic_row(tex,  r                        , layer, lod, t.x);
  float c = cubic_row(tex, (r + ivec2(0, 1      )) % s, layer, lod, t.x);
  float d = cubic_row(tex, (r + ivec2(0, 2      )) % s, layer, lod, t.x);
  return cubic_interp(t.y, vec4(a, b, c, d));
}

float cubic_mipmap_linear(sampler2DArray tex, vec2 p, int layer, float lod)
{
  int lod0 = int(floor(lod));
  int lod1 = lod0 + 1;
  float t = lod - float(lod0);
  return mix(cubic(tex, p, layer, lod0), cubic(tex, p, layer, lod1), t);
}

void main()
{
  vec2 dx = dFdx(coord);
  vec2 dy = dFdy(coord);
  vec4 s = vec4
```

```
55      ( texture(state, vec3(coord, 0.0), 100.0).x
        , texture(state, vec3(coord, 1.0), 100.0).x
        , texture(state, vec3(coord, 2.0), 100.0).x
        , texture(state, vec3(coord, 3.0), 100.0).x
        );
60    vec4 s00 = vec4
        ( texture(state, vec3(coord, 0.0)).x
        , texture(state, vec3(coord, 1.0)).x
        , texture(state, vec3(coord, 2.0)).x
        , texture(state, vec3(coord, 3.0)).x
65      );
      vec4 s01 = vec4
        ( texture(state, vec3(coord + dy, 0.0)).x
        , texture(state, vec3(coord + dy, 1.0)).x
        , texture(state, vec3(coord + dy, 2.0)).x
70      , texture(state, vec3(coord + dy, 3.0)).x
        );
      vec4 s10 = vec4
        ( texture(state, vec3(coord + dx, 0.0)).x
        , texture(state, vec3(coord + dx, 1.0)).x
75      , texture(state, vec3(coord + dx, 2.0)).x
        , texture(state, vec3(coord + dx, 3.0)).x
        );
      vec4 s11 = vec4
        ( texture(state, vec3(coord + dx + dy, 0.0)).x
80      , texture(state, vec3(coord + dx + dy, 1.0)).x
        , texture(state, vec3(coord + dx + dy, 2.0)).x
        , texture(state, vec3(coord + dx + dy, 3.0)).x
        );
      vec4 c00 = colour * (s00 - s) + offset;
85    vec4 c01 = colour * (s01 - s) + offset;
      vec4 c10 = colour * (s10 - s) + offset;
      vec4 c11 = colour * (s11 - s) + offset;
      vec4 c = 0.25 * (c00 + c01 + c10 + c11);
      float e = clamp(8.0 * length(vec2(length(c00 - c11), length(c10 - c01))), 0.0,↙
          ↳ 1.0);
90    colour_out = mix(vec4(1.0), clamp(c, 0.0, 1.0), e);
}
```

8 display_frag.glsl

```
#version 330 core

uniform sampler2D tex;

 5  in vec2 coord;

out vec4 colour_out;

void main()
10 {
  colour_out = texture(tex, coord);
}
```

9 .gitignore

```
cca
```

```
*.pgm
*.ppm
*.png
*.jpg
*.mkv
*.ogv
```

10 Makefile

```
cca: cca.c
	gcc -std=c99 -Wall -Wextra -pedantic -g -O3 -march=native -o cca cca.c -↙
	    ↳ lGL -lglfw -lGLEW `pkg-config --cflags --libs jack` -lsndfile -lm
```

11 mottled.cca

```
5.76533604 6.60849762 4.47446585 2.16664052 / 5.71545076 0.64990157 11.13152790 ↙
    ↳ 5.61171484 / 0.88078868 0.95492649 0.99646878 0.29966256 / -1.40496886 ↙
    ↳ 0.26280659 0.72964638 0.86794668 / -0.44996864 -1.72562432 -0.20833899 ↙
    ↳ -0.94577324 / 0.35387334 0.76373875 1.23302424 0.80471152 / 0.54986048 ↙
    ↳ 0.91892153 -0.80502993 -1.85384190 / 0.56668401 -0.18570143 -0.88585067 ↙
    ↳ 0.00801803 / -0.53357702 0.23276579 0.54967815 0.41899815 / 0.34108272 ↙
    ↳ 0.40493283 -0.64571607 0.85248381 / -0.84980458 0.71188581 0.93813014 ↙
    ↳ -0.55228901 / 0.48734620 0.83388823 0.15782885 0.76236188
```

12 rainclouds.cca

```
4.70329809 0.83233130 4.32183743 0.03508234 / 3.74213481 0.52699101 9.45791817 ↙
    ↳ 0.77844852 / 0.96449685 0.84980780 0.68310463 0.98278064 / 1.06336260 ↙
    ↳ 0.37009406 -0.88236272 0.89353037 / 0.79083997 1.51485407 -0.87305307 ↙
    ↳ 0.58772850 / -0.49640280 -0.87207800 -0.93314052 -0.99195361 / 0.74384838 ↙
    ↳ -0.22634144 0.57418317 -1.91916525 / -0.66435844 -0.56681770 0.45682716 ↙
    ↳ 0.50367934 / -0.34548169 -0.57508725 -0.99805325 -0.45109075 / 0.22884475 ↙
    ↳ 0.23325066 -0.41228232 -0.44862863 / 0.68910366 0.99839032 -0.11522026 ↙
    ↳ 0.22078492 / 0.25581619 0.33282045 0.55715764 0.71113843
```

13 README.md

```
Coupled Cellular Automata
=========================

Usage
-----

Needs GLEW and GLFW.  Type `make` to build, `./cca` to run.


Keyboard Controls
-----------------

ESC, Q    -- quit
SPACE     -- randomize parameters
ENTER     -- randomize state
P         -- print parameters to stderr
L         -- load parameters from stdin
M         -- toggle mutation
```

```
O        -- toggle overdrive (faster calculations)
H        -- toggle high quality (slower but prettier)
R        -- PPM recording (to stdout, so redirect it!)
S        -- PPM screenshot (to stdout, so redirect it!)
```

14 throttle.sh

```
#!/bin/bash
kill -s SIGSTOP "${@}"
running=0
stop_threshold=85
cont_threshold=75
while true
do
  temperature="$(nvidia-smi -q -d TEMPERATURE | grep 'GPU Current Temp' | sed 's/^.*: \(.*\) C$/\1/')"
  if (( running ))
  then
    if (( temperature > stop_threshold ))
    then
      echo "STOP ${temperature} > ${stop_threshold}"
      kill -s SIGSTOP "${@}"
      running=0
    fi
  else
    if (( temperature < cont_threshold ))
    then
      echo "CONT ${temperature} < ${cont_threshold}"
      kill -s SIGCONT "${@}"
      running=1
    fi
  fi
  sleep 1
done |
ts
```

15 wasp.cca

```
2.29705381 4.44075346 6.14918470 4.03819036 / 7.84241009 7.79764080 6.04927540 /
    10.14037418 / 0.19861820 0.90111142 0.99377704 0.91746503 / 1.45148647 /
    -0.52063566 -0.75107902 0.95574188 / -0.34188682 1.23749959 -0.92044818 /
    -0.53210002 / 0.13777913 -0.67436540 -1.63563883 -0.81862247 / 0.96365893 /
    0.60934752 0.90462303 1.48264015 / 0.60098952 0.25476402 0.17344452 /
    -0.82474703 / -0.63504755 -0.28925931 0.18480055 0.67202073 / -0.98965251 /
    -0.80698693 0.36208305 -0.93641275 / -0.25064489 -0.41655365 -0.33660427 /
    -0.52490163 / 0.53140533 0.45615834 0.21542011 0.86046195
```

16 whirlpool.cca

```
0.06671737 0.40420341 6.95843267 2.77287650 / 11.44176006 7.56961203 11.63460350 /
     7.49047756 / 0.91454554 0.69334853 0.95919001 0.72107166 / -1.35305583 /
    0.79673553 0.86282814 -0.22450675 / -0.90425944 0.26366398 -0.38018605 /
    0.68250418 / 0.38063395 -0.96966517 -1.92765665 -0.01288136 / -0.91002351 /
    0.64786130 0.53134024 0.76839894 / -0.20389771 -0.69208330 -0.51310241 /
    -0.18721837 / -0.59103245 -0.77349424 0.50600076 -0.68407238 / 0.48810777 /
    0.44510138 0.56434053 -0.92044210 / 0.72932273 -0.66650385 -0.61343652 /
    -0.94720519 / 0.06511583 0.12469579 0.41414404 0.11298612
```