mandelbrot-perturbator

Claude Heiland-Allen

2015–2019

Contents

1 BUGS.md

```
bugs
====


* too much series skipping at low zoom levels
* blank image at very deep zooms (series approximation overflow?)
* excessive memory usage
  * double precision: ( 8 * 2 * 3 + 4 + 4) * 2 + 4 * 4 = 128 bytes per pixel
  * higher precision: (16 * 2 * 3 + 4 + 4) * 2 + 4 * 4 = 224 bytes per pixel
```

2 c/bin/index.html

```
<!DOCTYPE html>
<html lang="en-us">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>mandelbrot-perturbator</title>
    <style>
```

```
      body { text-align: center; }
      canvas.emscripten { border: 0px none; }
    </style>
    <script>
      var hashToObject = function(){
        var
        i = 0,
        retObj = {},
        pair = null,
        sPageURL = window.location.hash.substring(2),
        qArr = sPageURL.split('&');

        for (; i < qArr.length; i++){
          pair = qArr[i].split('=');
          retObj[pair[0]] = decodeURIComponent(pair[1]);
        };

        return retObj;
      };

      var go = function(real, imag, radius, maxiters)
      {
        window.location.hash = "#!real=" + real + "&imag=" + imag + "&radius=" +↙
            ↳  radius + "&maxiters=" + maxiters;
      }

      var submitForm = function()
      {
        var real = encodeURIComponent(document.getElementById("real").value);
        var imag = encodeURIComponent(document.getElementById("imag").value);
        var radius = encodeURIComponent(document.getElementById("radius").value)↙
            ↳ ;
        var maxiters = encodeURIComponent(document.getElementById("maxiters").↙
            ↳ value);
        go(real, imag, radius, maxiters);
        return false;
      }

      window.onhashchange = function()
      {
        var hash = hashToObject();
        for(var key in hash){
            document.getElementById(key).value = decodeURIComponent(hash[key]);
        }
        Module.ccall("set", "number", ["string", "string", "string"], [hash["↙
            ↳ real"], hash["imag"], hash["radius"]]);
      }
    </script>
  </head>
  <body>
    <canvas class="emscripten" id="canvas" oncontextmenu="event.preventDefault()↙
        ↳ "></canvas>
    <form>
      real <textarea id="real" name="real">-0.75</textarea><br />
      imag <textarea id="imag" name="imag">0.0</textarea><br />
      radius <input id="radius" name="radius" value="2.0" /><br />
      maxiters <input id="maxiters" name="maxiters" value="1000000" /><br />
```

```
60        <input type="button" value="Go" onclick="submitForm();" />
      </form>
      <script type='text/javascript'>

        var hash = hashToObject();
65      for(var key in hash){
          if (key.length > 0)
            document.getElementById(key).value = decodeURIComponent(hash[key]);
        }

70      var Module = {
          preRun: [ function() { ENV = hashToObject(); } ],
          postRun: [],
          print: function() {
              console.log(Array.prototype.slice.call(arguments).join(" "));
75        },
          printErr: function() {
              console.warn(Array.prototype.slice.call(arguments).join(" "));
          },
          canvas: document.getElementById('canvas')
80      }

      </script>
      <script async type="text/javascript" src="perturbator-glfw3.js"></script>
    </body>
85  </html>
```

3 c/bin/Makefile

```
# perturbator -- efficient deep zooming for Mandelbrot sets
# Copyright (C) 2015-2019 Claude Heiland-Allen
# License GPL3+ http://www.gnu.org/licenses/gpl.html

5   prefix ?= $(HOME)/opt
CC ?= gcc

PKGCONFIG := PKG_CONFIG_PATH="$(prefix)/lib/pkgconfig" pkg-config
VERSION := "\"`git describe --tags --dirty --broken`\""
10  COMPILE := $(CC) -std=c99 -Wall -Wextra -pedantic -fPIC -O3 -ffast-math -pipe -↙
      ↳ MMD -g -DVERSION=$(VERSION) `$(PKGCONFIG) --cflags mandelbrot-perturbator ↙
      ↳ mandelbrot-numerics mandelbrot-symbolics gtk+-3.0`
LIBS    := `$(PKGCONFIG) --libs mandelbrot-perturbator mandelbrot-numerics ↙
      ↳ mandelbrot-symbolics gtk+-3.0` -lX11 -lGL -lGLEW -lglfw -lmpc -lmpfr -lgmp↙
      ↳  -lm -lstdc++
SOURCES := $(wildcard *.c)
OBJECTS := $(patsubst %.c,%.o,$(SOURCES))
DEPENDS := $(patsubst %.o,%.d,$(OBJECTS))
15  EXES    := $(patsubst %.o,%,$(OBJECTS))

all: $(EXES)

clean:
20        @echo "CLEAN" ; rm -f $(OBJECTS) $(DEPENDS) $(EXES)

install: $(EXES)
        install -d "$(prefix)/bin"
        install -m 755 -t "$(prefix)/bin" $(EXES)
```

```
25
%: %.o
	@echo "EXE      $@" ; gcc -o $@ $< $(LIBS) -ffast-math || ( echo "ERROR ↙
	    ↳   gcc -o $@ $< $(LIBS) -ffast-math" && false )

%.o: %.c
30	@echo "O        $@" ; $(COMPILE) -o $@ -c $< || ( echo "ERROR   $(COMPILE↙
	    ↳ ) -o $@ $<" && false )

m-perturbator-glfw3.js: m-perturbator-glfw3.c ../lib/libmandelbrot-perturbator.a
	emcc -s USE_GLFW=3 -s USE_PTHREADS=1 -s TOTAL_MEMORY=536870912 -O3 -o $@↙
	    ↳  $< -I../lib -L../lib -I$(prefix)/include -L$(prefix)/lib ../lib/↙
	    ↳ libmandelbrot-perturbator.a ${prefix}/lib/libmandelbrot-numerics.a↙
	    ↳  ${prefix}/lib/libmpc.a ${prefix}/lib/libmpfr.a ${prefix}/lib/↙
	    ↳ libgmp.a -lglfw

35 .SUFFIXES:
.PHONY: all clean install
.SECONDARY: $(OBJECTS)

-include $(DEPENDS)
```

4 c/bin/m-perturbator-automorph.c

```
// mandelbrot-perturbator -- efficient deep zooming for Mandelbrot sets
// Copyright (C) 2015-2019 Claude Heiland-Allen
// License GPL3+ http://www.gnu.org/licenses/gpl.html

 5 // http://mathr.co.uk/blog/2016-03-05_julia_morphing_symmetry.html

/*
adapt $prefix to where you installed mandelbrot-{numerics,symbolics}

10 $ gcc -std=c99 -Wall -Wextra -pedantic \
  -I$prefix/include/ -I../lib \
  -L$prefix/lib/ -L../lib \
  -o m-automorph m-automorph.c \
  -lmpc -lmpfr -lgmp -lm -lperturbator \
15   -lmandelbrot-numerics -lmandelbrot-symbolics \
  -lGLEW -lGL -lglfw -pthread -fopenmp -lstdc++ \
  -O3 -march=native
*/

20 #define _POSIX_C_SOURCE 199309L

#include <math.h>
#include <assert.h>
#include <stdbool.h>
25 #include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
30
#include <mpc.h>
#include <mpfr.h>
```

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

#include <mandelbrot-numerics.h>
#include <mandelbrot-symbolics.h>
#include <mandelbrot-perturbator.h>

/*
> writeFile "exangles.txt" $ unlines [ ", { " ++ show (plain (binary lo)) ++ "\n↙
    ↪    , " ++ show (plain (binary hi)) ++ " }" | (lo, hi) <- mapMaybe ↙
    ↪ addressAngles [ Angled 1 (InternalAngle (1 % 2)) (Angled 2 (InternalAngle ↙
    ↪ (1 % 2)) (Angled 3 (InternalAngle (1 % 2)) (Angled 4 (InternalAngle (1 % ↙
    ↪ 2)) (Angled 5 (InternalAngle (5 % 11)) (Angled 54 (InternalAngle (1 % 2)) ↙
    ↪ (Angled 64 (InternalAngle (1 % 2)) (Angled 69 (a%3) (Angled 143 (b%3) (↙
    ↪ Angled 291 (c%3) (Angled 587 (d%3) (Unangled 1179)))))))))) | [a,b,c,d] ↙
    ↪ <- replicateM 4 [1,2] ] ]
*/
const char *exangles[16][2] =
{ { ↙
    ↪ ".(011111000001111100000111110000011111000001111011111000011111000010000011111000001111100
    ↪ "
  , ↙
      ↪ ".(011111000001111100000111110000011111000001111011111000011111000010000011111000001111
      ↪ " }
, { ↙
    ↪ ".(011111000001111100000111110000011111000001111011111000011111000010000011111000001111100
    ↪ "
  , ↙
      ↪ ".(011111000001111100000111110000011111000001111011111000011111000010000011111000001111
      ↪ " }
, { ↙
    ↪ ".(011111000001111100000111110000011111000001111011111000011111000010000011111000001111100
    ↪ "
  , ↙
      ↪ ".(011111000001111100000111110000011111000001111011111000011111000010000011111000001111
      ↪ " }
, { ↙
    ↪ ".(011111000001111100000111110000011111000001111011111000011111000010000011111000001111100
    ↪ "
  , ↙
      ↪ ".(011111000001111100000111110000011111000001111011111000011111000010000011111000001111
      ↪ " }
, { ↙
    ↪ ".(011111000001111100000111110000011111000001111011111000011111000010000011111000001111100
    ↪ "
  , ↙
      ↪ ".(011111000001111100000111110000011111000001111011111000011111000010000011111000001111
      ↪ " }
, { ↙
    ↪ ".(011111000001111100000111110000011111000001111011111000011111000010000011111000001111100
    ↪ "
  , ↙
      ↪ ".(011111000001111100000111110000011111000001111011111000011111000010000011111000001111
      ↪ " }
, { ↙
    ↪ ".(011111000001111100000111110000011111000001111011111000011111000010000011111000001111100
    ↪ "
```

```
    , ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000001111100001000001111100000011111
      ↳ " }
  , { ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000001111100001000001111100000011111000
      ↳ "
60    , ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000001111100001000001111100000011111
      ↳ " }
  , { ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111110
      ↳ "
    , ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111111
      ↳ " }
  , { ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111110
      ↳ "
    , ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111111
      ↳ " }
65  , { ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111110
      ↳ "
    , ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111111
      ↳ " }
  , { ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111110
      ↳ "
    , ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111111
      ↳ " }
  , { ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111110
      ↳ "
70    , ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111111
      ↳ " }
  , { ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111110
      ↳ "
    , ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111111
      ↳ " }
  , { ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111110
      ↳ "
    , ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111111
      ↳ " }
75  , { ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111110
      ↳ "
    , ↙
      ↳ ".(011111000001111100000111110000011111000000111101111000010000011110111011111000000111111
      ↳ " }
```

```
};

static inline int max(int a, int b) {
  return a > b ? a : b;
}

static const char *simple_vert =
  "#version 130\n"
  "out vec2 texCoord;\n"
  "const vec2 t[4] = vec2[4](vec2(0.0, 0.0), vec2(1.0, 0.0), vec2(0.0, 1.0), vec2(1.0, 1.0));\n"
  "void main() {\n"
  "  gl_Position = vec4(t[gl_VertexID] * 2.0 - 1.0, 0.0, 1.0);\n"
  "  texCoord = t[gl_VertexID] * vec2(1.0, -1.0) + vec2(0.0, 1.0);\n"
  "}\n"
  ;

static const char *simple_frag =
  "#version 130\n"
  "uniform isampler2D tex_x;\n"
  "uniform  sampler2D tex_y;\n"
  "uniform  sampler2D tex_z;\n"
  "uniform  sampler2D tex_w;\n"
  "uniform bool show_lines;\n"
  "uniform float weight;\n"
  "in vec2 texCoord;\n"
  "out vec4 colour;\n"
  "int px(vec2 tc) {\n"
  "  int n = texture(tex_x, tc).x;\n"
  "  int k = 0;\n"
  "  k += (n % 2) >= 1 ? 1 : 2;\n"
  "  k += (texture(tex_z, tc).x) >= 0 ? 4 : 8;\n"
  "  k += 16 * n;\n"
  "  return k;\n"
  "}\n"
  "void main() {\n"
  "  float e = 1.0;\n"
  "  vec2 dx = dFdx(texCoord);\n"
  "  vec2 dy = dFdy(texCoord);\n"
  "  if (show_lines) {\n"
  "    e = px(texCoord) == px(texCoord + dx + dy) && px(texCoord + dx) == px(texCoord + dy) ? 1.0 : 0.0;\n"
  "  }\n"
  "  int me = texture(tex_x, texCoord).x;\n"
  "  float mef = texture(tex_y, texCoord).x;\n"
  "  float s = tanh(clamp(texture(tex_w, texCoord).x / weight, 0.0, 8.0));\n"
  "  colour = vec4(me <= 0 && mef <= 0.0 ? vec3(1.0, 0.7, 0.0) : vec3(mix(0.0, mix(0.9, 1.0, e), s)), 1.0);\n"
  "}\n"
  ;

struct state_s {
  GLFWwindow *window;

  bool show_lines;
  GLuint show_lines_u;
  double log2weight;
```

```
  GLuint weight_u;

  int width;
  int height;

  int precision;
  mpfr_t centerx;
  mpfr_t centery;
  mpfr_t radius;

};
typedef struct state_s state_t;

static void refresh_callback(void *user_pointer) {
  state_t *state = user_pointer;
  assert(state);
  glUniform1i(state->show_lines_u, state->show_lines);
  glUniform1f(state->weight_u, exp2f(state->log2weight));
  glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
  glfwSwapBuffers(state->window);
}

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, NULL, buffer);
    buffer[length] = 0;
    if (buffer[0]) {
      fprintf(stderr, "%s: OpenGL shader program info log\n", name);
      fprintf(stderr, "%s\n", buffer);
    }
    free(buffer);
  } else {
    fprintf(stderr, "%s: OpenGL shader program creation failed\n", name);
  }
}

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_GEOMETRY_SHADER: tname = "geometry"; break;
    case GL_FRAGMENT_SHADER: tname = "fragment"; 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, NULL, 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);
  } else {
    fprintf(stderr, "%s: OpenGL %s shader creation failed\n", name, tname);
  }
}

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

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;
}

extern int main(int argc, char **argv) {
  state_t state;
  memset(&state, 0, sizeof(state));

  int workers = 4;
  int width = 1920;
  int height = 1080;
  int maxiters = 1 << 24;
  int chunk = 1 << 10;
  double escape_radius = 25;
  double glitch_threshold = 1e-6;
  double weight = 0;
  int sharpness = 8;

  m_block a, b;
  m_block_init(&a);
  m_block_init(&b);
  m_block_from_string(&a, "01111");
  m_block_from_string(&b, "10000");
  m_binangle as[2], bs[2];
  m_binangle_init(&as[0]);
```

```
    m_binangle_init(&bs[0]);
    m_binangle_init(&as[1]);
    m_binangle_init(&bs[1]);

    mpq_t q;
    mpq_init(q);

    mpc_t c[2], delta;
    mpc_init2(c[0], 53);
    mpc_init2(c[1], 53);
    mpc_init2(delta, 53);

    mpfr_init2(state.radius, 53);
    mpfr_init2(state.centerx, 53);
    mpfr_init2(state.centery, 53);

    uint8_t *ppm = malloc(width * height * 3);
    assert(ppm);

    glfwInit();
    glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_API);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 2);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 1);
    glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
    GLFWwindow *window = glfwCreateWindow(width, height, "automorph", 0, 0);
    glfwMakeContextCurrent(window);
    glewExperimental = GL_TRUE;
    glewInit();
    glGetError(); // discard common error from glew

    GLuint tex[4];
    glGenTextures(4, tex);
    for (int t = 0; t < 4; ++t)
    {
      glActiveTexture(GL_TEXTURE0 + t);
      glBindTexture(GL_TEXTURE_2D, tex[t]);
      glTexImage2D(GL_TEXTURE_2D, 0, t == 0 ? GL_R32I : GL_R32F, width, height, 0,↙
          ↳ GL_RED, t == 0 ? GL_UNSIGNED_INT : GL_FLOAT, 0);
      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_MIRRORED_REPEAT);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);
    }

    GLuint vao;
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);

    GLuint program = compile_program("colourize", simple_vert, simple_frag);
    glUseProgram(program);
    glUniform1i(glGetUniformLocation(program, "tex_x"), 0);
    glUniform1i(glGetUniformLocation(program, "tex_y"), 1);
    glUniform1i(glGetUniformLocation(program, "tex_z"), 2);
    glUniform1i(glGetUniformLocation(program, "tex_w"), 3);
    state.show_lines_u = glGetUniformLocation(program, "show_lines");
    state.weight_u = glGetUniformLocation(program, "weight");
```

```
  state.window = window;
  state.width = width;
  state.height = height;
  state.log2weight = weight;

  struct perturbator *context = perturbator_new(workers, width, height, maxiters
      , chunk, escape_radius, glitch_threshold);

  glfwSetWindowUserPointer(window, &state);

  for (int count = 0; count < 16; ++count) {
    glfwPollEvents();
    if (glfwWindowShouldClose(window)) {
      break;
    }
    m_binangle_from_string(&as[0], exangles[count][0]);
    m_binangle_from_string(&bs[0], exangles[count][1]);
    m_block_append(&as[1].per, &bs[0].per, &as[0].per);
    m_block_append(&as[1].per, &as[1].per, &b);
    m_block_append(&bs[1].per, &bs[0].per, &bs[0].per);
    m_block_append(&bs[1].per, &bs[1].per, &a);

    m_binangle_to_rational(q, &as[0]);
    m_r_exray_in *ray = m_r_exray_in_new(q, sharpness);
    for (int s = 0; s < 2 * as[0].per.length * sharpness; ++s) {
      m_r_exray_in_step(ray, 16);
    }
    m_r_exray_in_get(ray, c[0]);
    m_r_nucleus(c[0], c[0], as[0].per.length, 64, 0);
    m_r_exray_in_delete(ray);

    m_binangle_to_rational(q, &as[1]);
    ray = m_r_exray_in_new(q, sharpness);
    for (int s = 0; s < 2 * as[1].per.length * sharpness; ++s) {
      m_r_exray_in_step(ray, 16);
    }
    m_r_exray_in_get(ray, c[1]);
    m_r_nucleus(c[1], c[1], as[1].per.length, 64, 0);
    m_r_exray_in_delete(ray);

    mpc_sub(delta, c[0], c[1], MPC_RNDNN);
    mpc_abs(state.radius, delta, MPFR_RNDN);
    mpfr_mul_d(state.radius, state.radius, 12, MPFR_RNDN);
    mpfr_set_prec(state.centerx, mpc_get_prec(c[0]));
    mpfr_set_prec(state.centery, mpc_get_prec(c[0]));
    mpfr_set(state.centerx, mpc_realref(c[0]), MPFR_RNDN);
    mpfr_set(state.centery, mpc_imagref(c[0]), MPFR_RNDN);
    perturbator_start(context, state.centerx, state.centery, state.radius);
    perturbator_stop(context, false);
    glActiveTexture(GL_TEXTURE0); glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, width,
        height, GL_RED, GL_INT, perturbator_get_dwell_n(context));
    glActiveTexture(GL_TEXTURE1); glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, width,
        height, GL_RED, GL_FLOAT, perturbator_get_dwell_f(context));
    glActiveTexture(GL_TEXTURE2); glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, width,
        height, GL_RED, GL_FLOAT, perturbator_get_dwell_a(context));
    glActiveTexture(GL_TEXTURE3); glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, width,
        height, GL_RED, GL_FLOAT, perturbator_get_distance(context));
```

```
      refresh_callback(&state);
      glReadPixels(0, 0, width, height, GL_RGB, GL_UNSIGNED_BYTE, ppm);
      printf("P6\n%d %d\n255\n", width, height);
      for (int y = height - 1; y >= 0; --y) {
355     fwrite(ppm + y * width * 3, width * 3, 1, stdout);
      }
      fflush(stdout);
    }

360   m_binangle_clear(&as[0]);
    m_binangle_clear(&as[1]);
    m_binangle_clear(&bs[0]);
    m_binangle_clear(&bs[1]);
    m_block_clear(&a);
365   m_block_clear(&b);
    mpq_clear(q);
    mpc_clear(delta);
    mpc_clear(c[0]);
    mpc_clear(c[1]);
370
    perturbator_stop(context, true);
    free(ppm);
    glfwDestroyWindow(window);
    glfwTerminate();
375   return 0;
  (void) argc;
  (void) argv;
  }
```

5 c/bin/m-perturbator-colourize.c

```
  // mandelbrot-perturbator -- efficient deep zooming for Mandelbrot sets
  // Copyright (C) 2017 Claude Heiland-Allen
  // License GPL3+ http://www.gnu.org/licenses/gpl.html

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

  int main(int argc, char **argv)
10 {
    size_t width = 0;
    size_t height = 0;
    if (argc > 1)
      width = atoi(argv[1]);
15  if (argc > 2)
      height = atoi(argv[2]);
    size_t pixels = width * height;
    size_t ibytes = 4 * sizeof(float) * pixels;
    size_t obytes = 3 * pixels;
20  float *i = malloc(ibytes);
    unsigned char *o = malloc(obytes);
    fread(i, ibytes, 1, stdin);
    double n = 0;
    double count = 0;
25  double mi = 1.0/0.0;
    double ma = -1.0/0.0;
```

```
  #pragma omp parallel for reduction(+:count) reduction(+:n) reduction(min:mi) ↙
      ↳ reduction(max:ma)
  for (size_t p = 0; p < pixels; ++p)
  {
    double ni = i[4 * p + 0];
    double nf = i[4 * p + 1];
    if (ni <= 0.0 && nf <= 0.0)
    {
      o[3 * p + 0] = 255;
      o[3 * p + 1] = 179;
      o[3 * p + 2] = 0;
    }
    else
    {
      float de = i[4 * p + 3];
      int g = 255 * tanhf(de);
      o[3 * p + 0] = g;
      o[3 * p + 1] = g;
      o[3 * p + 2] = g;
      double nif = ni + nf;
      mi = mi > nif ? nif : mi;
      ma = ma < nif ? nif : ma;
      n += nif;
      count += 1;
    }
  }
  fprintf(stderr, "%.16g %.16g %.16g\n", mi, n / count, ma);
  printf("P6\n%ld %ld\n255\n", width, height);
  fwrite(o, obytes, 1, stdout);
  free(i);
  free(o);
  return 0;
}
```

6 c/bin/m-perturbator-glfw3.c

```
// perturbator -- efficient deep zooming for Mandelbrot sets
// Copyright (C) 2015-2019 Claude Heiland-Allen
// License GPL3+ http://www.gnu.org/licenses/gpl.html

#define _POSIX_C_SOURCE 199309L

#include <math.h>
#include <assert.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include <mpfr.h>
#include <mpc.h>

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

```
#include <mandelbrot-numerics.h>
#include <mandelbrot-perturbator.h>

#ifdef __EMSCRIPTEN__
#include <emscripten.h>
#include <emscripten/html5.h>
#include <emscripten/threading.h>
#endif


static inline int max(int a, int b) {
  return a > b ? a : b;
}

static int envi(const char *name, int def) {
  const char *e = getenv(name);
  if (e) {
    return atoi(e);
  } else {
    return def;
  }
}

static double envd(const char *name, double def) {
  const char *e = getenv(name);
  if (e) {
    return atof(e);
  } else {
    return def;
  }
}

static int envr(mpfr_t out, const char *name, const char *def) {
  const char *e = getenv(name);
  if (e) {
    return mpfr_set_str(out, e,   10, MPFR_RNDN);
  } else {
    return mpfr_set_str(out, def, 10, MPFR_RNDN);
  }
}

#ifdef __EMSCRIPTEN__

static const char *blit_vert =
  "uniform vec4 bounds;\n"
  "attribute highp float vertexID;\n"
  "varying highp vec2 texCoord;\n"
  "void main() {\n"
  "  if (vertexID == 0.0) { gl_Position = vec4(-1.0, -1.0, 0.0, 1.0); texCoord =↙
      ↳ bounds.xy; } else\n"
  "  if (vertexID == 1.0) { gl_Position = vec4( 1.0, -1.0, 0.0, 1.0); texCoord =↙
      ↳ bounds.zy; } else\n"
  "  if (vertexID == 2.0) { gl_Position = vec4(-1.0,  1.0, 0.0, 1.0); texCoord =↙
      ↳ bounds.xw; } else\n"
  "                       { gl_Position = vec4( 1.0,  1.0, 0.0, 1.0); texCoord =↙
      ↳ bounds.zw; }\n"
  "}\n"
```

```
75    ;

      static const char *blit_frag =
        "uniform sampler2D tex;\n"
        "varying highp vec2 texCoord;\n"
80      "void main() {\n"
        "  gl_FragColor = texture2D(tex, texCoord);\n"
        "}\n"
        ;

85    static const char *simple_vert =
        "attribute highp float vertexID;\n"
        "varying highp vec2 texCoord;\n"
        "void main() {\n"
        "  if (vertexID == 0.0) { gl_Position = vec4(-1.0, -1.0, 0.0, 1.0); texCoord =↙
            ↳ vec2(0.0, 1.0); } else\n"
90      "  if (vertexID == 1.0) { gl_Position = vec4( 1.0, -1.0, 0.0, 1.0); texCoord =↙
            ↳ vec2(1.0, 1.0); } else\n"
        "  if (vertexID == 2.0) { gl_Position = vec4(-1.0,  1.0, 0.0, 1.0); texCoord =↙
            ↳ vec2(0.0, 0.0); } else\n"
        "                       { gl_Position = vec4( 1.0,  1.0, 0.0, 1.0); texCoord =↙
            ↳ vec2(1.0, 0.0); }\n"
        "}\n"
        ;
95
      static const char *simple_frag =
        "#extension GL_OES_standard_derivatives : enable\n"
        "uniform sampler2D tex;\n"
        "uniform bool show_lines;\n"
100     "uniform highp float weight;\n"
        "varying highp vec2 texCoord;\n"
        "int px(vec2 tc) {\n"
        "  highp vec4 t = texture2D(tex, tc);\n"
        "  int k = 0;\n"
105     "  k += (t.x / 2.0) - floor(t.x / 2.0) >= 0.5 ? 1 : 2;\n"
        "  k += t.z >= 0.0 ? 4 : 8;\n"
        "  k += 16 * int(floor(t.x));\n"
        "  return k;\n"
        "}\n"
110     "highp float tanh_(highp float y) {\n"
        "  highp float x = clamp(y, -3.0, 3.0);\n"
        "  return x * (27.0 + x * x) / (27.0 + 9.0 * x * x);\n"
        "}\n"
        "void main() {\n"
115     "  highp float e = 1.0;\n"
        "  highp vec2 dx = dFdx(texCoord);\n"
        "  highp vec2 dy = dFdy(texCoord);\n"
        "  if (show_lines) {\n"
        "    e = px(texCoord) == px(texCoord + dx + dy) && px(texCoord + dx) == px(↙
            ↳ texCoord + dy) ? 1.0 : 0.0;\n"
120     "  }\n"
        "  highp vec2 me = texture2D(tex, texCoord).xy;\n"
        "  highp float s = tanh_(clamp(texture2D(tex, texCoord).w / weight, 0.0, 8.0))↙
            ↳ ;\n"
        "  highp float unescaped = dot(me, me);\n"
        "  bool glitched = me.x >= 0.0 && me.y < 0.0;\n"
125     "  gl_FragColor = vec4(glitched ? vec3(1.0,0.0,0.0) : unescaped <= 0.0 ? vec3↙
```

```
        ↪ (1.0, 0.7, 0.0) : vec3(mix(0.0, mix(0.9, 1.0, e), s)), unescaped == 0.0 ↙
        ↪ ? 0.0 : 1.0);\n"
    "}\n"
    ;

#else

static const char *blit_vert =
  "#version 130\n"
  "uniform vec4 bounds;\n"
  "in float vertexID;\n"
  "out vec2 texCoord;\n"
  "void main() {\n"
  "  if (vertexID == 0.0) { gl_Position = vec4(-1.0, -1.0, 0.0, 1.0); texCoord =↙
      ↪ bounds.xy; } else\n"
  "  if (vertexID == 1.0) { gl_Position = vec4( 1.0, -1.0, 0.0, 1.0); texCoord =↙
      ↪ bounds.zy; } else\n"
  "  if (vertexID == 2.0) { gl_Position = vec4(-1.0,  1.0, 0.0, 1.0); texCoord =↙
      ↪ bounds.xw; } else\n"
  "                       { gl_Position = vec4( 1.0,  1.0, 0.0, 1.0); texCoord =↙
      ↪ bounds.zw; }\n"
  "}\n"
  ;

static const char *blit_frag =
  "#version 130\n"
  "uniform sampler2D tex;\n"
  "in vec2 texCoord;\n"
  "out vec4 fragColor;\n"
  "void main() {\n"
  "  fragColor = texture(tex, texCoord);\n"
  "}\n"
  ;

static const char *simple_vert =
  "#version 130\n"
  "in float vertexID;\n"
  "out vec2 texCoord;\n"
  "void main() {\n"
  "  if (vertexID == 0.0) { gl_Position = vec4(-1.0, -1.0, 0.0, 1.0); texCoord =↙
      ↪ vec2(0.0, 1.0); } else\n"
  "  if (vertexID == 1.0) { gl_Position = vec4( 1.0, -1.0, 0.0, 1.0); texCoord =↙
      ↪ vec2(1.0, 1.0); } else\n"
  "  if (vertexID == 2.0) { gl_Position = vec4(-1.0,  1.0, 0.0, 1.0); texCoord =↙
      ↪ vec2(0.0, 0.0); } else\n"
  "                       { gl_Position = vec4( 1.0,  1.0, 0.0, 1.0); texCoord =↙
      ↪ vec2(1.0, 0.0); }\n"
  "}\n"
  ;

static const char *simple_frag =
  "#version 130\n"
  "uniform isampler2D tex_x;\n"
  "uniform  sampler2D tex_y;\n"
  "uniform  sampler2D tex_z;\n"
  "uniform  sampler2D tex_w;\n"
  "uniform bool show_lines;\n"
```

```
    "uniform highp float weight;\n"
    "in vec2 texCoord;\n"
    "out vec4 fragColor;\n"
    "int px(vec2 tc) {\n"
    "  int n = texture(tex_x, tc).x;\n"
    "  int k = 0;\n"
    "  k += (n % 2) >= 1 ? 1 : 2;\n"
    "  k += (texture(tex_z, tc).x) >= 0 ? 4 : 8;\n"
    "  k += 16 * n;\n"
    "  return k;\n"
    "}\n"
    "void main() {\n"
    "  float e = 1.0;\n"
    "  vec2 dx = dFdx(texCoord);\n"
    "  vec2 dy = dFdy(texCoord);\n"
    "  if (show_lines) {\n"
    "    e = px(texCoord) == px(texCoord + dx + dy) && px(texCoord + dx) == px(texCoord + dy) ? 1.0 : 0.0;\n"
    "  }\n"
    "  int me = texture(tex_x, texCoord).x;\n"
    "  float mef = texture(tex_y, texCoord).x;\n"
    "  float s = tanh(clamp(texture(tex_w, texCoord).x / weight, 0.0, 8.0));\n"
    "  bool glitched = me >= 0 && mef < 0.0;\n"
    "  float g = -mef;\n"
    "  fragColor = vec4(glitched ? tanh(clamp(vec3(g,g*0.1,g*0.01), vec3(0.0), vec3(4.0))) : unescaped <= 0.0 ? vec3(1.0, 0.7, 0.0) : vec3(mix(0.0, mix(0.9, 1.0, e), s)), unescaped == 0.0 ? 0.0 : 1.0);\n"
    "}\n"
    ;

#endif

struct state_s {
  GLFWwindow *window;
  struct perturbator *context;

  bool should_restart;
  bool should_redraw;
  bool should_save_now;
  bool should_save_when_done;
  bool should_view_morph;

  GLuint fbo;

  GLuint blit_vao;
  GLuint blit_p;
  GLint bounds_u;

  GLuint colourize_vao;
  GLuint colourize_p;
  bool show_lines;
  GLint show_lines_u;
  double log2weight;
  GLint weight_u;

  int width;
  int height;
```

```
  int precision;
  mpfr_t centerx;
  mpfr_t centery;
  mpfr_t radius;
  int maxiters;

  double srcx0;
  double srcy0;
  double srcx1;
  double srcy1;

  uint8_t *ppm;
};
typedef struct state_s state_t;
state_t state_d;

static void refresh_callback(void *user_pointer);

#ifdef __EMSCRIPTEN__
void go(state_t *state);
void refresh_callback(void *user_pointer);
#endif

static void button_handler(GLFWwindow *window, int button, int action, int mods) {
  state_t *state = glfwGetWindowUserPointer(window);
  if (action == GLFW_PRESS) {
    double srcx0 = 0, srcy0 = 0, srcx1 = state->width, srcy1 = state->height;
    double x = 0, y = 0;
    glfwGetCursorPos(window, &x, &y);
    mpfr_t cx, cy, r;
    mpfr_init2(cx, state->precision);
    mpfr_init2(cy, state->precision);
    mpfr_init2(r, 53);
    mpfr_set(cx, state->centerx, MPFR_RNDN);
    mpfr_set(cy, state->centery, MPFR_RNDN);
    mpfr_set(r, state->radius, MPFR_RNDN);
    double w = state->width;
    double h = state->height;
    double yy = h - y;
    double dx = 2 * ((x + 0.5) / w - 0.5) * (w / h);
    double dy = 2 * (0.5 - (y + 0.5) / h);
    mpfr_t ddx, ddy;
    mpfr_init2(ddx, 53);
    mpfr_init2(ddy, 53);
    mpfr_mul_d(ddx, r, dx, MPFR_RNDN);
    mpfr_mul_d(ddy, r, dy, MPFR_RNDN);
    switch (button) {
      case GLFW_MOUSE_BUTTON_LEFT:
        mpfr_mul_2si(ddx, ddx, -1, MPFR_RNDN);
        mpfr_mul_2si(ddy, ddy, -1, MPFR_RNDN);
        mpfr_add(cx, cx, ddx, MPFR_RNDN);
        mpfr_add(cy, cy, ddy, MPFR_RNDN);
        mpfr_mul_2si(r, r, -1, MPFR_RNDN);
        state->precision += 1;
        mpfr_set_prec(state->centerx, state->precision);
```

```
            mpfr_set_prec(state->centery, state->precision);
            mpfr_set(state->centerx, cx, MPFR_RNDN);
285         mpfr_set(state->centery, cy, MPFR_RNDN);
            mpfr_set(state->radius, r, MPFR_RNDN);
            state->should_restart = true;
            srcx0 = 0.5 * (srcx0 - x) + x;
            srcx1 = 0.5 * (srcx1 - x) + x;
290         srcy0 = 0.5 * (srcy0 - yy) + yy;
            srcy1 = 0.5 * (srcy1 - yy) + yy;
            break;
          case GLFW_MOUSE_BUTTON_RIGHT:
            mpfr_sub(cx, cx, ddx, MPFR_RNDN);
295         mpfr_sub(cy, cy, ddy, MPFR_RNDN);
            mpfr_mul_2si(r, r, 1, MPFR_RNDN);
            state->precision -= 1;
            mpfr_set_prec(state->centerx, state->precision);
            mpfr_set_prec(state->centery, state->precision);
300         mpfr_set(state->centerx, cx, MPFR_RNDN);
            mpfr_set(state->centery, cy, MPFR_RNDN);
            mpfr_set(state->radius, r, MPFR_RNDN);
            state->should_restart = true;
            srcx0 = 2.0 * (srcx0 - x) + x;
305         srcx1 = 2.0 * (srcx1 - x) + x;
            srcy0 = 2.0 * (srcy0 - yy) + yy;
            srcy1 = 2.0 * (srcy1 - yy) + yy;
            break;
          case GLFW_MOUSE_BUTTON_MIDDLE:
310         mpfr_add(cx, cx, ddx, MPFR_RNDN);
            mpfr_add(cy, cy, ddy, MPFR_RNDN);
            mpfr_set(state->centerx, cx, MPFR_RNDN);
            mpfr_set(state->centery, cy, MPFR_RNDN);
            state->should_restart = true;
315         srcx0 -= state->width / 2 - x;
            srcx1 -= state->width / 2 - x;
            srcy0 -= state->height / 2 - yy;
            srcy1 -= state->height / 2 - yy;
            break;
320       }
        mpfr_clear(cx);
        mpfr_clear(cy);
        mpfr_clear(r);
        mpfr_clear(ddx);
325     mpfr_clear(ddy);
    #ifdef __EMSCRIPTEN__
        glBindFramebuffer(GL_FRAMEBUFFER, state->fbo);
        glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
        glBindFramebuffer(GL_FRAMEBUFFER, 0);
330 #else
        glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
        glBindFramebuffer(GL_DRAW_FRAMEBUFFER, state->fbo);
        glBlitFramebuffer(0, 0, state->width, state->height, 0, 0, state->width, ↙
            ↳ state->height, GL_COLOR_BUFFER_BIT, GL_NEAREST);
        glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
335 #endif

        state->srcx0 = srcx0;
        state->srcy0 = srcy0;
```

```
      state->srcx1 = srcx1;
      state->srcy1 = srcy1;
    }
#ifdef __EMSCRIPTEN__
    if (state->should_restart)
    {
      state->should_restart = false;
      go(state);
    }
#endif
(void) mods;
}

static void key_handler(GLFWwindow *window, int key, int scancode, int action, ↙
    ↳ int mods) {
  state_t *state = glfwGetWindowUserPointer(window);
  if (action == GLFW_PRESS) {
    switch (key) {
      case GLFW_KEY_Q:
      case GLFW_KEY_ESCAPE:
        glfwSetWindowShouldClose(window, GL_TRUE);
        break;
      case GLFW_KEY_J:
        state->should_view_morph = true;
        break;
      case GLFW_KEY_L:
        state->show_lines = ! state->show_lines;
        state->should_redraw = true;
        break;
      case GLFW_KEY_C:
        glBindFramebuffer(GL_FRAMEBUFFER, state->fbo);
        glClear(GL_COLOR_BUFFER_BIT);
        glBindFramebuffer(GL_FRAMEBUFFER, 0);
        glClear(GL_COLOR_BUFFER_BIT);
        state->should_redraw = true;
        break;
      case GLFW_KEY_S:
        if (mods & GLFW_MOD_SHIFT) {
          state->should_save_now = true;
        } else {
          state->should_save_when_done = true;
        }
        break;
      case GLFW_KEY_9:
        state->log2weight -= 0.25;
        state->should_redraw = true;
        break;
      case GLFW_KEY_0:
        state->log2weight += 0.25;
        state->should_redraw = true;
        break;
    }
  }
#ifdef __EMSCRIPTEN__
  if (state->should_restart)
  {
    state->should_restart = false;
```

```
395       go(state);
        }
        if (state->should_redraw)
        {
          state->should_redraw = false;
400       refresh_callback(state);
        }
      #endif
      (void) scancode;
      }
405
      static void refresh_callback(void *user_pointer) {
        state_t *state = user_pointer;
        assert(state);
        glBindVertexArray(state->blit_vao);
410     glUseProgram(state->blit_p);
        glUniform4f(state->bounds_u, state->srcx0 / state->width, state->srcy0 / state↙
            ↳ ->height, state->srcx1 / state->width, state->srcy1 / state->height);
        glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
        glBindVertexArray(state->colourize_vao);
        glUseProgram(state->colourize_p);
415     glActiveTexture(GL_TEXTURE0); glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, state->↙
            ↳ width, state->height, GL_RED, GL_INT, perturbator_get_dwell_n(state->↙
            ↳ context));
        glActiveTexture(GL_TEXTURE1); glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, state->↙
            ↳ width, state->height, GL_RED, GL_FLOAT, perturbator_get_dwell_f(state->↙
            ↳ context));
        glActiveTexture(GL_TEXTURE2); glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, state->↙
            ↳ width, state->height, GL_RED, GL_FLOAT, perturbator_get_dwell_a(state->↙
            ↳ context));
        glActiveTexture(GL_TEXTURE3); glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, state->↙
            ↳ width, state->height, GL_RED, GL_FLOAT, perturbator_get_distance(state->↙
            ↳ context));
        glUniform1i(state->show_lines_u, state->show_lines);
420     glUniform1f(state->weight_u, exp2f(state->log2weight));
        glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
        glfwSwapBuffers(state->window);
      }

425   static void handle_view_morph(struct perturbator *context, state_t *state) {
        if (state->should_view_morph) {
          state->should_view_morph = false;
          mpc_t nucleus, size;
          mpc_init2(nucleus, 53);
430       mpc_init2(size, 53);
          int period = perturbator_get_primary_reference(context, mpc_realref(nucleus)↙
              ↳ , mpc_imagref(nucleus));
          m_r_size(size, nucleus, period);
          mpfr_t r, r2;
          mpfr_init2(r, 53);
435       mpfr_init2(r2, 53);
          mpc_abs(r, size, MPFR_RNDN);
          mpfr_sqrt(r2, r, MPFR_RNDN);
          mpfr_mul(r, r, r2, MPFR_RNDN);
          mpfr_sqrt(r, r, MPFR_RNDN);
440       mpfr_mul_2si(r, r, 3, MPFR_RNDN);
      /*
```

```
      mpfr_set(r, state->radius, MPFR_RNDN);
      int e = mpfr_get_exp(r);
      mpfr_mul_2si(r, r, e / 2, MPFR_RNDN);
*/
      int p = max(53, 53 - 2 * mpfr_get_exp(r));
      state->precision = p;
      mpfr_set_prec(state->centerx, p);
      mpfr_set_prec(state->centery, p);
      mpfr_set(state->centerx, mpc_realref(nucleus), MPFR_RNDN);
      mpfr_set(state->centery, mpc_imagref(nucleus), MPFR_RNDN);
      mpfr_set(state->radius, r, MPFR_RNDN);
      state->should_restart = true;
      mpc_clear(nucleus);
      mpc_clear(size);
      mpfr_clear(r2);
      mpfr_clear(r);
    }
  }

  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, NULL, buffer);
      buffer[length] = 0;
      if (buffer[0]) {
        fprintf(stderr, "%s: OpenGL shader program info log\n", name);
        fprintf(stderr, "%s\n", buffer);
      }
      free(buffer);
    } else {
      fprintf(stderr, "%s: OpenGL shader program creation failed\n", name);
    }
  }

  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_GEOMETRY_SHADER: tname = "geometry"; break;
      case GL_FRAGMENT_SHADER: tname = "fragment"; 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, NULL, 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);
  } else {
    fprintf(stderr, "%s: OpenGL %s shader creation failed\n", name, tname);
  }
}

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

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); }
  glBindAttribLocation(program, 0, "vertexID");
  glLinkProgram(program);
  debug_program(program, name);
  return program;
}

#ifdef __EMSCRIPTEN__

void go(state_t *state)
{
  size_t bytes = state->precision * 3 + 100; // FIXME
  char *script = malloc(bytes);
  mpfr_snprintf(script, bytes, "go('%Re', '%Re', '%Re', '%d');", state->centerx, state->centery, state->radius, state->maxiters);
  emscripten_run_script(script);
  free(script);
}

void main1(void)
{
  state_t *state = &state_d;
  glfwPollEvents();
  refresh_callback(state);
}

EMSCRIPTEN_KEEPALIVE int set(const char *cx, const char *cy, const char *r)
{
  static bool first = true;
  state_t *state = &state_d;
```

```
    if (! first)
    {
      perturbator_stop(state->context, true);
      first = false;
    }
    mpfr_set_str(state->radius, r, 10, MPFR_RNDN);
    state->precision = max(53, 53 - mpfr_get_exp(state->radius));
    mpfr_set_prec(state->centerx, state->precision);
    mpfr_set_prec(state->centery, state->precision);
    mpfr_set_str(state->centerx, cx, 10, MPFR_RNDN);
    mpfr_set_str(state->centery, cy, 10, MPFR_RNDN);
    perturbator_start(state->context, state->centerx, state->centery, state->↙
        ↳ radius);
    return 0;
}

#endif


extern int main(int argc, char **argv) {
  state_t *state = &state_d;
  memset(state, 0, sizeof(*state));

#ifdef __EMSCRIPTEN__
  int workers = envi("threads", emscripten_num_logical_cores());
#else
  int workers = envi("threads", 4);
#endif
  int width = envi("width", 640);
  int height = envi("height", 360);
  int detect_glitches = envi("detect_glitches", 1);
  int approx_skip = envi("approx_skip", 0);
  int maxiters = envi("maxiters", 1 << 20);
  int chunk = envi("chunk", 1 << 8);
  double escape_radius = envd("escaperadius", 25);
  double glitch_threshold = envd("glitchthreshold", 1e-6);
  int precision = envi("precision", 53);
  int zoom_start = envi("zoom_start", 0);
  int zoom_count = envi("zoom_count", 0);
  double weight = envd("weight", 0);

  mpfr_init2(state->radius, 53);
  envr(state->radius, "radius", "2.0");

  int e = max(53, 53 - mpfr_get_exp(state->radius));
  if (e > precision) {
    fprintf(stderr, "WARNING: increasing precision to %d\n", e);
    precision = e;
  }

  mpfr_init2(state->centerx, precision);
  mpfr_init2(state->centery, precision);
  envr(state->centerx, "real", "-0.75");
  envr(state->centery, "imag", "0.0");

  glfwInit();
  glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_API);
```

```
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 2);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 1);
    glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
    state->window = glfwCreateWindow(width, height, "perturbator", 0, 0);
    glfwMakeContextCurrent(state->window);
    glewExperimental = GL_TRUE;
    glewInit();
    glGetError(); // discard common error from glew

#ifdef __EMSCRIPTEN__
    {
      EMSCRIPTEN_WEBGL_CONTEXT_HANDLE gl = emscripten_webgl_get_current_context();
      EM_BOOL have_gl_extension_OES_texture_float = ↙
          ↳ emscripten_webgl_enable_extension(gl, "OES_texture_float");
      EM_BOOL have_gl_extension_OES_standard_derivatives = ↙
          ↳ emscripten_webgl_enable_extension(gl, "OES_standard_derivatives");
      assert(have_gl_extension_OES_texture_float);
      assert(have_gl_extension_OES_standard_derivatives);
    }
#endif

    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable(GL_BLEND);
    glClearColor(1.0, 0.7, 0.0, 1.0);
    glDisable(GL_DEPTH_TEST);

    GLuint ftex;
    glGenTextures(1, &ftex);
    glActiveTexture(GL_TEXTURE4);
    glBindTexture(GL_TEXTURE_2D, ftex);
//  printf("%dx%d GL_RGBA\n", width, height);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, ↙
        ↳ GL_UNSIGNED_BYTE, 0);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glGenFramebuffers(1, &state->fbo);
    glBindFramebuffer(GL_FRAMEBUFFER, state->fbo);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ↙
        ↳ ftex, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBindFramebuffer(GL_FRAMEBUFFER, 0);
    glClear(GL_COLOR_BUFFER_BIT);

    GLuint tex[4];
    glGenTextures(4, tex);
    for (int t = 0; t < 4; ++t)
    {
      glActiveTexture(GL_TEXTURE0 + t);
      glBindTexture(GL_TEXTURE_2D, tex[t]);
#ifdef __EMSCRIPTEN__
      glTexImage2D(GL_TEXTURE_2D, 0, t == 0 ? GL_R32I : GL_R32F, width, height, 0,↙
          ↳ GL_RED, t == 0 ? GL_INT : GL_FLOAT, 0); // FIXME
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
#else
```

```
        glTexImage2D(GL_TEXTURE_2D, 0, t == 0 ? GL_R32I : GL_R32F, width, height, 0,↙
            ↳ GL_RED, t == 0 ? GL_INT : GL_FLOAT, 0);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);
    #endif
665     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
      }

      GLuint vbo;
670   GLubyte vbo_data[4] = { 0, 1, 2, 3 };
      glGenBuffers(1, &vbo);
      glBindBuffer(GL_ARRAY_BUFFER, vbo);
      glBufferData(GL_ARRAY_BUFFER, sizeof(vbo_data), vbo_data, GL_STATIC_DRAW);

675   glGenVertexArrays(1, &state->blit_vao);
      glBindVertexArray(state->blit_vao);
      state->blit_p = compile_program("blit", blit_vert, blit_frag);
      glUseProgram(state->blit_p);
      glUniform1i(glGetUniformLocation(state->blit_p, "tex"), 1);
680   state->bounds_u = glGetUniformLocation(state->blit_p, "bounds");
      GLint attr = glGetAttribLocation(state->blit_p, "vertexID");
      glVertexAttribPointer(attr, 1, GL_UNSIGNED_BYTE, GL_FALSE, 0, 0);
      glEnableVertexAttribArray(attr);
      glBindVertexArray(0);
685
      glGenVertexArrays(1, &state->colourize_vao);
      glBindVertexArray(state->colourize_vao);
      state->colourize_p = compile_program("colourize", simple_vert, simple_frag);
      glUseProgram(state->colourize_p);
690   state->show_lines_u = glGetUniformLocation(state->colourize_p, "show_lines");
      state->weight_u = glGetUniformLocation(state->colourize_p, "weight");
      attr = glGetAttribLocation(state->colourize_p, "vertexID");
      glVertexAttribPointer(attr, 1, GL_UNSIGNED_BYTE, GL_FALSE, 0, 0);
      glEnableVertexAttribArray(attr);
695
      state->ppm = malloc(width * height * 3);
      assert(state->ppm);

      state->should_restart = false;
700   state->should_redraw = false;
      state->should_save_now = false;
      state->should_save_when_done = false;
      state->width = width;
      state->height = height;
705   state->precision = precision;
      state->log2weight = weight;
      state->maxiters = maxiters;
      state->srcx0 = 0;
      state->srcy0 = 0;
710   state->srcx1 = state->width;
      state->srcy1 = state->height;

      state->context = perturbator_new(workers, width, height, maxiters, chunk,↙
          ↳ escape_radius, glitch_threshold);
      perturbator_set_detect_glitches(state->context, detect_glitches);
715   perturbator_set_approx_skip(state->context, approx_skip);
```

```
  glfwSetWindowUserPointer(state->window, state);

#ifdef __EMSCRIPTEN__

  glfwSetMouseButtonCallback(state->window, button_handler);
  glfwSetKeyCallback(state->window, key_handler);
  perturbator_start(state->context, state->centerx, state->centery, state->radius);
  emscripten_set_main_loop(main1, 0, 1);

#else

  if (zoom_count) {
    mpfr_set_d(state->radius, 256, MPFR_RNDN);
    mpfr_div_2exp(state->radius, state->radius, zoom_start, MPFR_RNDN);
    for (int z = zoom_start; z < zoom_count; ++z) {
      glfwPollEvents();
      if (glfwWindowShouldClose(state->window)) {
        break;
      }
      fprintf(stderr, "%8d FRAME\n", z);
      perturbator_start(state->context, state->centerx, state->centery, state->radius);
      perturbator_stop(state->context, false);
      refresh_callback(state);
      glReadPixels(0, 0, width, height, GL_RGB, GL_UNSIGNED_BYTE, state->ppm);
      printf("P6\n%d %d\n255\n", width, height);
      for (int y = height - 1; y >= 0; --y) {
        fwrite(state->ppm + y * width * 3, width * 3, 1, stdout);
      }
      fflush(stdout);
      mpfr_div_2exp(state->radius, state->radius, 1, MPFR_RNDN);
    }

  } else {
    glfwSetMouseButtonCallback(state->window, button_handler);
    glfwSetKeyCallback(state->window, key_handler);

    bool first = true;
    while (! glfwWindowShouldClose(state->window)) {
      int e = glGetError();
      if (e)
        fprintf(stderr, "E: %d\n", e);

      state->should_restart = false;
//    fprintf(stderr, "start\n");
      if (! first) {
        perturbator_stop(state->context, true);
      }
      first = false;
      perturbator_start(state->context, state->centerx, state->centery, state->radius);

//    fprintf(stderr, "wait_timeout\n");
      while (perturbator_active(state->context)) {
        struct timespec delta = { 0, 33333333 };
```

```
770           nanosleep(&delta, 0);
      //        fprintf(stderr, "refresh\n");
              refresh_callback(state);

              glfwPollEvents();
775           if (glfwWindowShouldClose(state->window)) {
                break;
              }

              if (state->should_save_now) {
780             state->should_save_now = false;
                glReadPixels(0, 0, state->width, state->height, GL_RGB, ↙
                    ↳ GL_UNSIGNED_BYTE, state->ppm);
                printf("P6\n%d %d\n255\n", state->width, state->height);
                fwrite(state->ppm, state->width * state->height * 3, 1, stdout);
                fflush(stdout);
785           }

              handle_view_morph(state->context, state);

              if (state->should_restart) {
790             state->should_restart = false;
      //          fprintf(stderr, "start\n");
                perturbator_stop(state->context, true);
                perturbator_start(state->context, state->centerx, state->centery, ↙
                    ↳ state->radius);
              }
795   //        fprintf(stderr, "wait_timeout\n");
            }

            if (glfwWindowShouldClose(state->window)) {
              break;
800         }

            refresh_callback(state);

            while (! state->should_restart) {
805
              if (state->should_save_now || state->should_save_when_done) {
                state->should_save_now = false;
                state->should_save_when_done = false;
                glReadPixels(0, 0, state->width, state->height, GL_RGB, ↙
                    ↳ GL_UNSIGNED_BYTE, state->ppm);
810             printf("P6\n%d %d\n255\n", state->width, state->height);
                fwrite(state->ppm, state->width * state->height * 3, 1, stdout);
                fflush(stdout);
              }

815           glfwWaitEvents();
              if (glfwWindowShouldClose(state->window)) {
                break;
              }

820   //        if (state->should_redraw) {
      //          state->should_redraw = false;
      //          fprintf(stderr, "refresh\n");
              refresh_callback(state);
```

```
    //            glfwSwapBuffers(state->window);
825   //        }

            handle_view_morph(state->context, state);

          }
830
        }

      //  fprintf(stderr, "delete\n");
        perturbator_stop(state->context, true);
835   }


    //  image_delete(context);
      free(state->ppm);
840   glfwDestroyWindow(state->window);
      glfwTerminate();

    #endif

845   return 0;
    (void) argc;
    (void) argv;
    }
```

7 c/bin/m-perturbator-gtk.c

```
    // mandelbrot-perturbator -- efficient deep zooming for Mandelbrot sets
    // Copyright (C) 2019 Claude Heiland-Allen
    // License GPL3+ http://www.gnu.org/licenses/gpl.html

 5  #define _POSIX_C_SOURCE 200809L
    #include <complex.h>
    #include <math.h>
    #include <stdio.h>
    #include <stdint.h>
10  #include <string.h>
    #include <mpfr.h>
    #include <gtk/gtk.h>
    #include <gdk/gdkx.h>
    #include <cairo-pdf.h>
15
    #include <mandelbrot-numerics.h>
    #include <mandelbrot-symbolics.h>
    #include <mandelbrot-perturbator.h>

20  #define pi 3.141592653589793
    #define twopi 6.283185307179586

    static int serialize(const char *filename);
    static int deserialize(const char *filename);
25
    struct view {
      mpfr_t cx, cy, radius, pixel_spacing;
    };
```

```
30  static struct view *view_new() {
      struct view *v = malloc(sizeof(struct view));
      mpfr_init2(v->cx, 53);
      mpfr_init2(v->cy, 53);
      mpfr_init2(v->radius, 53);
35    mpfr_init2(v->pixel_spacing, 53);
      return v;
    }

    static void view_delete(struct view *v) {
40    mpfr_clear(v->cx);
      mpfr_clear(v->cy);
      mpfr_clear(v->radius);
      mpfr_clear(v->pixel_spacing);
      free(v);
45  }

    static struct view *view_copy(const struct view *v) {
      struct view *v2 = view_new();
      mpfr_set_prec(v2->cx, mpfr_get_prec(v->cx));
50    mpfr_set_prec(v2->cy, mpfr_get_prec(v->cy));
      mpfr_set(v2->cx, v->cx, GMP_RNDN);
      mpfr_set(v2->cy, v->cy, GMP_RNDN);
      mpfr_set(v2->radius, v->radius, GMP_RNDN);
      return v2;
55  }

    struct point {
      struct point *next;
      struct point *pred;
60    mpc_t xy;
    };

    static struct point *point_new(struct point *next) {
      struct point *l = calloc(1, sizeof(*l));
65    l->next = next;
      if (next)
      {
        l->pred = next->pred;
        next->pred = l;
70    }
      mpc_init2(l->xy, 53);
      return l;
    }

75  static void points_delete(struct point *l) {
      struct point *next = l;
      while ((l = next)) {
        next = l->next;
        mpc_clear(l->xy);
80      l->next = 0;
        l->pred = 0;
        free(l);
      }
    }
85
    static struct point *points_copy(const struct point *last, struct point **tail)
```

```
{
  struct point *result = 0;
  int first = 1;
  while (last)
  {
    result = point_new(result);
    if (first)
    {
      if (tail)
      {
        *tail = result;
      }
      first = 0;
    }
    mpfr_set_prec(mpc_realref(result->xy), mpfr_get_prec(mpc_realref(last->xy)));
    mpfr_set_prec(mpc_imagref(result->xy), mpfr_get_prec(mpc_imagref(last->xy)));
    mpc_set(result->xy, last->xy, MPC_RNDNN);
    last = last->pred;
  }
  return result;
}

enum annotation_t {
  annotation_ray_out,
  annotation_ray_in,
  annotation_nucleus,
  annotation_misiurewicz,
  annotation_text,
  annotation_wake,
  annotation_atom,
  annotation_domain
};

struct annotation_ray_out {
  struct point *line_start;
  struct point *line_end;
  int line_type;
  double stroke_r;
  double stroke_g;
  double stroke_b;
};

struct annotation_ray_in {
  struct point *line_start;
  struct point *line_end;
  int line_type;
  double stroke_r;
  double stroke_g;
  double stroke_b;
  struct m_binangle angle;
  struct m_r_exray_in *ray;
  int depth;
/*
  int have_nucleus;
  int have_landing;
```

```
  mpfr_t nucleusx, nucleusy, landingx, landingy, eps_2;
*/
};

struct annotation_text {
  mpc_t xy;
};

struct annotation_nucleus {
  mpc_t xy;
  int period;
  mpfr_t size;
  mpfr_t domain_size;
};

struct annotation_misiurewicz {
  mpc_t xy;
  int period;
  int preperiod;
  mpfr_t domain_size;
};

struct annotation_wake {
  struct annotation *next_wake;
  struct annotation *pred_wake;
  char *ray_lo;
  char *ray_hi;
  int ray_depth;
  struct point *line_start;
  struct point *line_end;
  int fill_type;
  double fill_r;
  double fill_g;
  double fill_b;
  mpq_t width;
};

struct annotation_atom {
  mpc_t nucleus;
  int period;
  m_shape shape;
  struct point *line_start;
  struct point *line_end;
  int fill_type;
  double fill_r;
  double fill_g;
  double fill_b;
};

struct annotation_domain {
  mpc_t nucleus;
  int period;
  int loperiod;
  struct point *line_start;
  struct point *line_end;
  int line_type;
  double line_r;
```

```
  double line_g;
  double line_b;
};

struct annotation {
  struct annotation *next;
  int id;
  char *label;
  int selected;
  enum annotation_t tag;
  union {
    struct annotation_ray_out ray_out;
    struct annotation_ray_in ray_in;
    struct annotation_text text;
    struct annotation_nucleus nucleus;
    struct annotation_misiurewicz misiurewicz;
    struct annotation_wake wake;
    struct annotation_atom atom;
    struct annotation_domain domain;
  } u;
};

static bool annotation_set_colour(struct annotation *a, double r, double g, ↙
    ↳ double b)
{
  switch (a->tag)
  {
    case annotation_ray_out:
      a->u.ray_out.stroke_r = r;
      a->u.ray_out.stroke_g = g;
      a->u.ray_out.stroke_b = b;
      return true;
    case annotation_ray_in:
      a->u.ray_in.stroke_r = r;
      a->u.ray_in.stroke_g = g;
      a->u.ray_in.stroke_b = b;
      return true;
    case annotation_wake:
      a->u.wake.fill_r = r;
      a->u.wake.fill_g = g;
      a->u.wake.fill_b = b;
      return true;
    case annotation_atom:
      a->u.atom.fill_r = r;
      a->u.atom.fill_g = g;
      a->u.atom.fill_b = b;
      return true;
    case annotation_domain:
      a->u.domain.line_r = r;
      a->u.domain.line_g = g;
      a->u.domain.line_b = b;
      return true;
    case annotation_text:
    case annotation_nucleus:
    case annotation_misiurewicz:
      return false;
  }
```

```
255     return false;
    }

    static bool annotation_set_line_type(struct annotation *a, int t)
    {
260   switch (a->tag)
      {
        case annotation_ray_out:
          a->u.ray_out.line_type = t;
          return true;
265     case annotation_ray_in:
          a->u.ray_in.line_type = t;
          return true;
        case annotation_domain:
          a->u.domain.line_type = t;
270       return true;
        case annotation_atom:
        case annotation_wake:
        case annotation_text:
        case annotation_nucleus:
275     case annotation_misiurewicz:
          return false;
      }
      return false;
    }
280
    static bool annotation_set_fill_type(struct annotation *a, int t)
    {
      switch (a->tag)
      {
285     case annotation_wake:
          a->u.wake.fill_type = t;
          return true;
        case annotation_atom:
          a->u.atom.fill_type = t;
290       return true;
        case annotation_ray_out:
        case annotation_ray_in:
        case annotation_text:
        case annotation_nucleus:
295     case annotation_misiurewicz:
        case annotation_domain:
          return false;
      }
      return false;
300 }

    enum colour_theme_t
    {
      colour_theme_monochrome = 0,
305   colour_theme_low_colour = 1,
      colour_theme_full_colour = 2
    };

    static struct
310 {
      // fractal image
```

```
    double dpi_print;
    double dpi_screen;
    cairo_surface_t *surface;
315 cairo_surface_t *mask;
    cairo_surface_t *selection_surface;
    GtkWidget *da, *window;
    int width_print, height_print;
    int width_screen, height_screen;
320 // zoom tool
    gulong zoom_motion_notify_handler, zoom_button_press_handler, ↙
        ↳ zoom_button_release_handler;
    // info tool
    gulong info_button_press_handler;
    // ray out tool
325 gulong ray_out_button_press_handler;
    // nucleus tool
    gulong nucleus_motion_notify_handler, nucleus_button_press_handler, ↙
        ↳ nucleus_button_release_handler;
    // misurewicz tool
    gulong misiurewicz_motion_notify_handler, misiurewicz_button_press_handler, ↙
        ↳ misiurewicz_button_release_handler;
330 // selection tool
    gulong selection_button_press_handler;
    // bond tool
    gulong bond_button_press_handler;
    // transient box drawing
335 int box;
    double box_aspect, box_x1, box_y1, box_x2, box_y2;
    // transient ball drawing
    int ball;
    double ball_x1, ball_y1, ball_x2, ball_y2;
340 // mandelbrot view
    struct view *view;
    double escape_radius;
    mpfr_t escape_radius2;
    int maximum_iterations;
345 int chunk;
    int threads;
    double glitch_threshold;
    struct perturbator *image;
    // annotations
350 GtkTreeStore *annostore;
    GtkWidget *annotree;
    struct annotation *anno;
    // doubly-linked list of wakes sorted by width, narrowest first
    // real nodes have next_wake != 0 && pred_wake != 0
355 // sentinel nodes at end have a null in one
    struct annotation wakes_head;
    struct annotation wakes_tail;
    GtkWidget *filaments_period, *filaments_preperiod, *filaments_depth;
    // async computations
360 // FIXME REMOVE {{
    double progress;
    int cancelled;
    GtkWidget *dialog, *bar;
    // FIXME REMOVE }}
365 GAsyncQueue *annotation_queue; // annotations to GUI go via here
```

```
  GThreadPool *task_workers; // annotation tasks go to here
  GtkWidget *task_box; // annotation tasks in progress displayed here
  GtkWidget *log;
  // theme
  GtkToolItem *dark_widget;
  GtkToolItem *colour_widget;
  int dark;
  int colour;
  // dashed lines
  int line_type;
  GtkListStore *dashstore[2];
  GtkWidget *dashcombo;
  // pattern fills
  int fill_type;
  cairo_pattern_t *fillpattern[2][26];
  GtkListStore *fillstore[2];
  GtkWidget *fillcombo;
  // colours
  GtkWidget *colourbutton;
  double colour_r;
  double colour_g;
  double colour_b;
} G;

static void set_dark_theme(bool use_dark_theme);
static void set_colour_theme(enum colour_theme_t use_colour_theme);

static void screen_to_param(double x, double y, mpfr_t cx, mpfr_t cy) {
  double sx = (x - G.width_screen / 2.0) / (G.height_screen / 2.0);
  double sy = (G.height_screen / 2.0 - y) / (G.height_screen / 2.0);
  mpfr_set_prec(cx, mpfr_get_prec(G.view->cx));
  mpfr_set_prec(cy, mpfr_get_prec(G.view->cy));
  mpfr_set_d(cx, sx, GMP_RNDN);
  mpfr_set_d(cy, sy, GMP_RNDN);
  mpfr_mul(cx, cx, G.view->radius, GMP_RNDN);
  mpfr_mul(cy, cy, G.view->radius, GMP_RNDN);
  mpfr_add(cx, cx, G.view->cx, GMP_RNDN);
  mpfr_add(cy, cy, G.view->cy, GMP_RNDN);
}

static void param_to_screen(double *x, double *y, const mpfr_t cx, const mpfr_t ↙
    ↳ cy) {
  mpfr_t tx, ty;
  mpfr_init2(tx, mpfr_get_prec(G.view->cx));
  mpfr_init2(ty, mpfr_get_prec(G.view->cy));
  mpfr_sub(tx, cx, G.view->cx, GMP_RNDN);
  mpfr_sub(ty, cy, G.view->cy, GMP_RNDN);
  mpfr_div(tx, tx, G.view->radius, GMP_RNDN);
  mpfr_div(ty, ty, G.view->radius, GMP_RNDN);
  double sx = mpfr_get_d(tx, GMP_RNDN);
  double sy = mpfr_get_d(ty, GMP_RNDN);
  *x = sx * G.height_screen / 2.0 + G.width_screen / 2.0;
  *y = G.height_screen / 2.0 - sy * G.height_screen / 2.0;
  mpfr_clear(tx);
  mpfr_clear(ty);
}
```

```
static void param_to_screen_clamped(double *x, double *y, const mpfr_t cx, const↙
    ↪ mpfr_t cy) {
  mpfr_t tx, ty, bound;
  mpfr_init2(tx, mpfr_get_prec(G.view->cx));
  mpfr_init2(ty, mpfr_get_prec(G.view->cy));
  mpfr_init2(bound, 53);
  // FIXME should depend on image aspect ratio ?
  // FIXME should clip to circle ?
  mpfr_set_si(bound, 10, MPFR_RNDN);
  mpfr_sub(tx, cx, G.view->cx, GMP_RNDN);
  mpfr_sub(ty, cy, G.view->cy, GMP_RNDN);
  mpfr_div(tx, tx, G.view->radius, GMP_RNDN);
  mpfr_div(ty, ty, G.view->radius, GMP_RNDN);
  mpfr_min(tx, tx, bound, MPFR_RNDN);
  mpfr_min(ty, ty, bound, MPFR_RNDN);
  mpfr_neg(bound, bound, MPFR_RNDN);
  mpfr_max(tx, tx, bound, MPFR_RNDN);
  mpfr_max(ty, ty, bound, MPFR_RNDN);
  double sx = mpfr_get_d(tx, GMP_RNDN);
  double sy = mpfr_get_d(ty, GMP_RNDN);
  *x = sx * G.height_screen / 2.0 + G.width_screen / 2.0;
  *y = G.height_screen / 2.0 - sy * G.height_screen / 2.0;
  mpfr_clear(tx);
  mpfr_clear(ty);
  mpfr_clear(bound);
}

static void log_append(const char *text) {
  GtkTextBuffer *buf = gtk_text_view_get_buffer(GTK_TEXT_VIEW(G.log));
  GtkTextIter end;
  gtk_text_buffer_get_iter_at_offset(buf, &end, -1);
  gtk_text_buffer_insert(buf, &end, text, -1);
  gtk_text_buffer_get_iter_at_offset(buf, &end, -1);
  GtkTextMark *eof = gtk_text_buffer_get_mark(buf, "EOF");
  if (eof) {
    gtk_text_buffer_move_mark(buf, eof, &end);
  } else {
    eof = gtk_text_buffer_create_mark(buf, "EOF", &end, FALSE);
  }
  gtk_text_view_scroll_to_mark(GTK_TEXT_VIEW(G.log), eof, 0.0, FALSE, 0.0, 0.0);
}

static gboolean log_append_thread(void *text) {
  log_append(text);
  free(text);
  return G_SOURCE_REMOVE;
}

static gboolean render_thread_progress(void *userdata) {
  (void) userdata;
  if (! G.cancelled)
  {
    if (G.progress == 0)
    {
      gtk_progress_bar_pulse(GTK_PROGRESS_BAR(G.bar));
    }
    else
```

```
    {
      gtk_progress_bar_set_fraction (GTK_PROGRESS_BAR (G.bar), G.progress);
    }
  }
  return G_SOURCE_REMOVE;
}

static int progresscb(void *userdata, double progress) {
  (void) userdata;
  G.progress = progress;
  gdk_threads_add_idle (render_thread_progress, 0);
  return ! G.cancelled;
}

static void dorender(struct view *v, struct perturbator *image, int maxiters) {
  char *buf = calloc(1, 65536); // FIXME fixed size buffer
  mpfr_snprintf(buf, 65536, "RENDER\nreal: %Re\nimag: %Re\nradius: %Re\n", v->cx, v->cy, v->radius);
  gdk_threads_add_idle(log_append_thread, buf);
  perturbator_set_maximum_iterations(image, maxiters);
  perturbator_start(image, v->cx, v->cy, v->radius);
  struct timespec timeout;
  do
  {
    progresscb(0, perturbator_get_progress(image));
    clock_gettime(CLOCK_REALTIME, &timeout);
    timeout.tv_nsec += 0.25 * 1000 * 1000 * 1000;
    while (timeout.tv_nsec >= 1000 * 1000 * 1000)
    {
      timeout.tv_nsec -= 1000 * 1000 * 1000;
      timeout.tv_sec += 1;
    }
  } while ((! G.cancelled) && perturbator_wait(image, &timeout));
  perturbator_stop(image, G.cancelled);
}

static int channel(float c) {
  return fminf(fmaxf(roundf(255 * c), 0), 255);
}

static void hsv2rgb(float h, float s, float v, float *red, float *grn, float *blu) {
  float i, f, p, q, t, r, g, b;
  if (s == 0) { r = g = b = v; } else {
    h = 6 * (h - floorf(h));
    int ii = i = floorf(h);
    f = h - i;
    p = v * (1 - s);
    q = v * (1 - (s * f));
    t = v * (1 - (s * (1 - f)));
    switch(ii) {
      case 0: r = v; g = t; b = p; break;
      case 1: r = q; g = v; b = p; break;
      case 2: r = p; g = v; b = t; break;
      case 3: r = p; g = q; b = v; break;
```

```
      case 4: r = t; g = p; b = v; break;
      default:r = v; g = p; b = q; break;
    }
  }
  *red = r;
  *grn = g;
  *blu = b;
}

static void colour(int *r, int *g, int *b, int final_n, float final_z_abs, float final_z_arg, float de, int final_p, int t) {
  (void) final_n;
  float radius = 0.0f, angle = 0.0f;
  if (t > 0) {
    radius = 1 - final_z_abs;
    angle = final_z_arg;
  } else if (t < 0) {
    radius = -log2f(1.0f - final_z_abs);
    angle = fmodf(powf(2.0, floorf(radius) + 3.0f) * final_z_arg, 1.0f);
    radius = 1.0 - fmodf(fmodf(radius, 1.0f) + 1.0, 1.0);
  }
  float k = powf(0.5f, 0.5f - radius);
  float grid_weight = 0.05f;
/*
  float grid = fminf
    ( fminf( smoothstep(grid_weight - dradius, grid_weight + dradius, radius)
           , 1.0f - smoothstep(1.0f - grid_weight - dradius, 1.0f - grid_weight + dradius, radius))
    , fminf( smoothstep(grid_weight * k - dangle, grid_weight * k + dangle, angle)
           , 1.0f - smoothstep(1.0 - grid_weight * k - dangle, 1.0 - grid_weight * k + dangle, angle))
    );
*/
  int grid =
    grid_weight     < radius && radius < 1.0f - grid_weight &&
    grid_weight * k < angle  && angle  < 1.0f - grid_weight * k;
  float hue = (final_p - 1.0f) / 24.618033988749895f;
  float sat = 0.0f;//final_p > 0.0f ? 0.5f : 0.0f;
  float val = fmin(0.5 + 0.5 * tanh(fmin(fmax(fabs(de), 0.0), 4.0)), 0.8 + 0.2 * grid);
  if (t == 0) {
    hue = 0.0f;
    sat = 0.0f;
    val = 1.0f;
  }
  float red, grn, blu;
  hsv2rgb(hue, sat, G.dark ? 1 - val : val, &red, &grn, &blu);
  *r = channel(red);
  *g = channel(grn);
  *b = channel(blu);
}

static void docolour(struct perturbator *image) {
  if (!G.surface) {
    return;
  }
```

```
585   const int32_t *dwell_n  = perturbator_get_dwell_n (image);
      const float   *dwell_f  = perturbator_get_dwell_f (image);
      const float   *dwell_a  = perturbator_get_dwell_a (image);
      const float   *distance = perturbator_get_distance(image);
      cairo_surface_flush(G.surface);
590   uint32_t *data = (uint32_t *) cairo_image_surface_get_data(G.surface);
      int channels = 4;
      int stride = cairo_image_surface_get_stride(G.surface) / channels;
      #pragma omp parallel for
      for (int j = 0; j < G.height_print; ++j) {
595     for (int i = 0; i < G.width_print; ++i) {
          int ki = j * G.width_print + i;
          double final_n_int = dwell_n[ki];
          double final_z_abs = dwell_f[ki];
          double final_z_arg = fmod(dwell_a[ki]+ 1, 1);
600       double de          = distance[ki];
          int final_n = floor(final_n_int + final_z_abs);
          int final_p = de < 0 ? -final_n_int : 0;
          int t = de < 0 ? -1 : de > 0 ? 1 : 0;
          int r, g, b;
605       colour(&r, &g, &b, final_n, final_z_abs, final_z_arg, de, final_p, t);
          int px = (r << 16) | (g << 8) | b;
          int ko = j * stride + i;
          data[ko] = (0xFF << 24) | px;
        }
610   }
      cairo_surface_mark_dirty(G.surface);
    }

    struct render_data {
615   struct view *view;
      struct perturbator *image;
      int maximum_iterations;
    };

620 static gboolean render_thread_cancelled(gpointer user_data) {
      struct render_data *d = user_data;
      G.dialog = 0;
      G.bar = 0;
      log_append("\nCANCELLED");
625   free(d);
      return G_SOURCE_REMOVE;
    }

    static gboolean render_thread_done(gpointer user_data) {
630   struct render_data *d = user_data;
      gtk_widget_destroy(G.dialog);
      G.dialog = 0;
      G.bar = 0;
      if (G.view) {
635     view_delete(G.view);
        G.view = 0;
      }
      G.view = d->view;
      free(d);
640   gtk_widget_queue_draw(G.da);
      return G_SOURCE_REMOVE;
```

```
}

static gpointer render_thread(gpointer user_data) {
  struct render_data *d = user_data;
  if (! G.cancelled) {
    dorender(d->view, d->image, d->maximum_iterations);
  }
  int cancelled = G.cancelled;
  if (! cancelled) {
    docolour(d->image);
  }
  if (cancelled) {
    view_delete(d->view);
    d->view = 0;
    gdk_threads_add_idle(render_thread_cancelled, d);
  } else {
    gdk_threads_add_idle(render_thread_done, d);
  }
  return 0;
}

static void start_render(struct view *v) {

  GtkDialogFlags flags = GTK_DIALOG_MODAL | GTK_DIALOG_DESTROY_WITH_PARENT;
  G.dialog = gtk_dialog_new_with_buttons("Progress", GTK_WINDOW(G.window), flags
      ↳ , NULL, NULL);
  GtkWidget *box = gtk_dialog_get_content_area(GTK_DIALOG(G.dialog));
  G.bar = gtk_progress_bar_new();
  gtk_container_add(GTK_CONTAINER(box), G.bar);

  struct render_data *d = malloc(sizeof(struct render_data));
  d->view = v;
  d->image = G.image;
  d->maximum_iterations = G.maximum_iterations;
  G.cancelled = 0;
  G.progress = 0;

  GThread *thread = g_thread_new("render", render_thread, d);

  gtk_widget_show_all(G.dialog);
  gtk_dialog_run(GTK_DIALOG(G.dialog));

  G.cancelled = 1;
  g_thread_join(thread);
  if (G.dialog) {
    gtk_widget_destroy(G.dialog);
  }
}

static struct annotation *get_selected_annotation(void)
{
  GtkTreeSelection *select = gtk_tree_view_get_selection(GTK_TREE_VIEW(G.
      ↳ annotree));
  GtkTreeIter iter;
  GtkTreeModel *model;
  gpointer thing;
  if (gtk_tree_selection_get_selected(select, &model, &iter))
```

```
  {
    gtk_tree_model_get(model, &iter, 1, &thing, -1);
    if (thing)
    {
      struct annotation *anno = (struct annotation *) thing;
      return anno;
    }
  }
  return 0;
}

static gboolean add_annotation(void *userdata) {
  static int nextid = 0;
  struct annotation *ls = userdata;
  ls->id = ++nextid;
  ls->next = G.anno;
  G.anno = ls;
  // maintain list of wakes sorted by width
  if (ls->tag == annotation_wake)
  {
    struct annotation *ws = G.wakes_head.u.wake.next_wake;
    while (ws->u.wake.next_wake)
    {
      if (mpq_cmp(ls->u.wake.width, ws->u.wake.width) <= 0)
        break;
      ws = ws->u.wake.next_wake;
    }
    // insert before ws
    ls->u.wake.next_wake = ws;
    ls->u.wake.pred_wake = ws->u.wake.pred_wake;
    ws->u.wake.pred_wake->u.wake.next_wake = ls;
    ws->u.wake.pred_wake = ls;
  }
  GtkTreeIter iter;
  gtk_tree_store_append(G.annostore, &iter, NULL);
  gtk_tree_store_set(G.annostore, &iter, 0, ls->label, 1, ls, -1);
  return G_SOURCE_REMOVE;
}

static void unlink_annotation(struct annotation *a)
{
  struct annotation *prev = 0;
  for (struct annotation *ls = G.anno; ls; ls = ls->next) {
    if (ls->next == a) {
      prev = ls;
      break;
    }
  }
  if (prev) {
    prev->next = prev->next->next;
  } else {
    if (G.anno == a) {
      G.anno = G.anno->next;
    }
  }
  a->next = 0;
  if (a->tag == annotation_wake)
```

```
  {
    a->u.wake.next_wake->u.wake.pred_wake = a->u.wake.pred_wake;
    a->u.wake.pred_wake->u.wake.next_wake = a->u.wake.next_wake;
    a->u.wake.next_wake = 0;
    a->u.wake.pred_wake = 0;
  }
}

static void free_annotation(struct annotation *a) {
  free(a->label);
  switch (a->tag) {
    case annotation_ray_out:
      points_delete(a->u.ray_out.line_start);
      a->u.ray_out.line_start = 0;
      a->u.ray_out.line_end = 0;
      break;
    case annotation_ray_in:
      points_delete(a->u.ray_in.line_start);
      a->u.ray_in.line_start = 0;
      a->u.ray_in.line_end = 0;
      m_r_exray_in_delete(a->u.ray_in.ray);
      m_binangle_clear(&a->u.ray_in.angle);
/*
      mpfr_clear(a->u.ray_in.nucleusx);
      mpfr_clear(a->u.ray_in.nucleusy);
      mpfr_clear(a->u.ray_in.landingx);
      mpfr_clear(a->u.ray_in.landingy);
      mpfr_clear(a->u.ray_in.eps_2);
*/
      break;
    case annotation_text:
      mpfr_clear(mpc_realref(a->u.text.xy));
      mpfr_clear(mpc_imagref(a->u.text.xy));
      break;
    case annotation_nucleus:
      mpc_clear(a->u.nucleus.xy);
      mpfr_clear(a->u.nucleus.size);
      mpfr_clear(a->u.nucleus.domain_size);
      break;
    case annotation_misiurewicz:
      mpc_clear(a->u.misiurewicz.xy);
      mpfr_clear(a->u.misiurewicz.domain_size);
      break;
    case annotation_wake:
      points_delete(a->u.wake.line_start);
      mpq_clear(a->u.wake.width);
      free(a->u.wake.ray_lo);
      free(a->u.wake.ray_hi);
      break;
    case annotation_atom:
      points_delete(a->u.atom.line_start);
      mpc_clear(a->u.atom.nucleus);
      break;
    case annotation_domain:
      points_delete(a->u.domain.line_start);
      mpc_clear(a->u.domain.nucleus);
      break;
```

```
  }
  free(a);
}

static void delete_annotation(struct annotation *a)
{
  unlink_annotation(a);
  free_annotation(a);
}

static double image_peek_dwell(struct perturbator *image, int i, int j)
{
  int w = perturbator_get_width(image);
  int h = perturbator_get_height(image);
  if (! (0 <= i && i < w && 0 <= j && j < h))
  {
    return 0.0 / 0.0; // unknown
  }
  const int32_t *dwell_n = perturbator_get_dwell_n(image);
  const float   *dwell_f = perturbator_get_dwell_f(image);
  int k = j * w + i;
  double dwelln = dwell_n[k];
  double dwellf = dwell_f[k];
  if (dwelln <= 0 || dwellf <= 0) { // interior
    return 1.0 / 0.0;
  }
  return dwelln + dwellf;
}

struct task;
static struct task *task_new_ray_out(GAsyncQueue *output, const mpc_t c, int dwell, int line_type, double stroke_r, double stroke_g, double stroke_b);
static struct task *task_new_ray_in(GAsyncQueue *output, const m_binangle *angle, int depth, int line_type, double stroke_r, double stroke_g, double stroke_b);
static struct task *task_new_ray_extend(GAsyncQueue *output, struct annotation *anno, int depth);
static struct task *task_new_nucleus(GAsyncQueue *output, const mpc_t c, const mpfr_t r, int dwell);
static struct task *task_new_bond(GAsyncQueue *output, const mpc_t c, int period);
static struct task *task_new_misiurewicz(GAsyncQueue *output, const mpc_t c, const mpfr_t r, int maxpreperiod, int maxperiod);
static struct task *task_new_muunit(GAsyncQueue *output, const mpc_t nucleus, int period);
static struct task *task_new_muunit_nucleus(GAsyncQueue *output, const mpc_t nucleus, int nucleus_period, int ray_period, const mpq_t angle);
static struct task *task_new_filaments(GAsyncQueue *output, const mpc_t inner_nucleus, int inner_period, int outer_period, int preperiod, int depth, int line_type, double stroke_r, double stroke_g, double stroke_b);
static struct task *task_new_wake(GAsyncQueue *output, struct annotation *lo, struct annotation *hi, int fill_type, double fill_r, double fill_g, double fill_b);
static struct task *task_new_wake2(GAsyncQueue *output, char *lo, char *hi, int depth, int fill_type, double fill_r, double fill_g, double fill_b);
static struct task *task_new_atom(GAsyncQueue *output, const mpc_t nucleus, int period, int fill_type, double fill_r, double fill_g, double fill_b);
```

```
static struct task *task_new_domain(GAsyncQueue *output, const mpc_t nucleus, ↙
    ↳ int period, int loperiod, int fill_type, double fill_r, double fill_g, ↙
    ↳ double fill_b);
static gboolean task_enqueue_cb(gpointer userdata);

static void start_ray_out(double x, double y)
{
  double dwell = image_peek_dwell(G.image, x, y);
  if (isinf(dwell) || isnan(dwell))
  {
    return;
  }
  mpc_t c;
  mpc_init2(c, 53);
  screen_to_param(x, y, mpc_realref(c), mpc_imagref(c));
  struct task *t = task_new_ray_out(G.annotation_queue, c, ceil(dwell), G.↙
      ↳ line_type, G.colour_r, G.colour_g, G.colour_b);
  mpc_clear(c);
  task_enqueue_cb(t);
}

static void start_ray_in(struct m_binangle *angle, int depth) {
  struct task *t = task_new_ray_in(G.annotation_queue, angle, depth, G.line_type↙
      ↳ , G.colour_r, G.colour_g, G.colour_b);
  task_enqueue_cb(t);
}

static void start_ray_extend(struct annotation *anno, int depth) {
  unlink_annotation(anno);
  struct task *t = task_new_ray_extend(G.annotation_queue, anno, depth);
  task_enqueue_cb(t);
}

static void start_nucleus(const mpfr_t x, const mpfr_t y, const mpfr_t r, int ↙
    ↳ maxperiod)
{
  mpc_t c;
  mpc_init2(c, mpfr_get_prec(x));
  mpc_set_fr_fr(c, x, y, MPC_RNDNN);
  struct task *t = task_new_nucleus(G.annotation_queue, c, r, maxperiod);
  mpc_clear(c);
  task_enqueue_cb(t);
}

static void start_bond(const mpc_t c, int period)
{
  struct task *t = task_new_bond(G.annotation_queue, c, period);
  task_enqueue_cb(t);
}

static void start_wake(struct annotation *anno, int direction)
{
  // find neighbouring angle in direction
  mpq_t p, q, r;
  mpq_init(p);
  mpq_init(q);
  mpq_init(r);
```

```
905     m_binangle_to_rational(p, &anno->u.ray_in.angle);
        if (direction > 0)
        {
          mpq_set_si(q, 1, 1);
        }
910     else
        {
          mpq_set_si(q, 0, 1);
        }
        struct annotation *other = 0;
915     struct annotation *a = G.anno;
        mpc_t d;
        mpc_init2(d, 53);
        while (a)
        {
920       if (a->tag == annotation_ray_in &&
            a->u.ray_in.angle.pre.length == anno->u.ray_in.angle.pre.length &&
            a->u.ray_in.angle.per.length == anno->u.ray_in.angle.per.length)
          {
            m_binangle_to_rational(r, &a->u.ray_in.angle);
925         // if (p < r < q || q < r < p) && endpoint is nearby, set q = r
            int pr = mpq_cmp(p, r);
            int rq = mpq_cmp(r, q);
            mpc_sub(d, a->u.ray_in.line_start->xy, anno->u.ray_in.line_start->xy, ↙
                ↳ MPC_RNDNN);
            mpc_abs(mpc_realref(d), d, MPFR_RNDN);
930         mpfr_div(mpc_realref(d), mpc_realref(d), G.view->radius, MPFR_RNDN);
            double distance = mpfr_get_d(mpc_realref(d), MPFR_RNDN);
            if (((pr < 0 && rq < 0) || (0 < pr && 0 < rq)) && distance < 0.1)
            {
              other = a;
935           mpq_set(q, r);
            }
          }
          a = a->next;
        }
940     mpc_clear(d);
        mpq_clear(p);
        mpq_clear(q);
        mpq_clear(r);
        if (other)
945     {
          GtkTreeModel *model = GTK_TREE_MODEL(G.annostore);
          GtkTreeIter iter;
          gboolean ok = gtk_tree_model_get_iter_first(model, &iter);
          while (ok)
950       {
            gpointer thing;
            gtk_tree_model_get(model, &iter, 1, &thing, -1);
            if (thing == anno || thing == other)
            {
955           ok = gtk_tree_store_remove(G.annostore, &iter);
            }
            else
            {
              ok = gtk_tree_model_iter_next(model, &iter);
960         }
```

```
    }
    unlink_annotation(anno);
    unlink_annotation(other);
    struct task *t = task_new_wake(G.annotation_queue, direction > 0 ? anno : ↙
        ↳ other, direction > 0 ? other : anno, G.fill_type, G.colour_r, G.↙
        ↳ colour_g, G.colour_b);
    task_enqueue_cb(t);
  }
}

static void start_wake2(int depth, char *lo, char *hi)
{
  struct task *t = task_new_wake2(G.annotation_queue, lo, hi, depth, G.fill_type↙
      ↳ , G.colour_r, G.colour_g, G.colour_b);
  task_enqueue_cb(t);
}

static void start_atom(const mpc_t c, int period)
{
  struct task *t = task_new_atom(G.annotation_queue, c, period, G.fill_type, G.↙
      ↳ colour_r, G.colour_g, G.colour_b);
  task_enqueue_cb(t);
}

static void start_domain(const mpc_t c, int period, int loperiod)
{
  struct task *t = task_new_domain(G.annotation_queue, c, period, loperiod, G.↙
      ↳ fill_type, G.colour_r, G.colour_g, G.colour_b);
  task_enqueue_cb(t);
}

static void box_rectangle(double *x, double *y, double *w, double *h) {
  double dx = fabs(G.box_x2 - G.box_x1);
  double dy = fabs(G.box_y2 - G.box_y1);
  if (dx <= G.box_aspect * dy) {
    *h = dy;
    *w = *h * G.box_aspect;
  } else {
    *w = dx;
    *h = *w / G.box_aspect;
  }
  *x = G.box_x1 - *w;
  *y = G.box_y1 - *h;
  *w *= 2;
  *h *= 2;
}

static void draw_box(cairo_t *cr) {
  double x, y, w, h;
  box_rectangle(&x, &y, &w, &h);
  cairo_rectangle(cr, x, y, w, h);
  if (G.dark)
  {
    cairo_set_source_rgba(cr, 1.0, 0.75, 0.75, 0.25);
  }
  else
  {
```

```
      cairo_set_source_rgba(cr, 0.5, 0.25, 0.25, 0.25);
    }
    cairo_fill_preserve(cr);
    if (G.dark)
    {
      cairo_set_source_rgba(cr, 1.0, 0.75, 0.75, 1);
    }
    else
    {
      cairo_set_source_rgba(cr, 0.5, 0.25, 0.25, 1);
    }
    cairo_stroke(cr);
}

static void ball_circle(double *x, double *y, double *r) {
  double dx = G.ball_x2 - G.ball_x1;
  double dy = G.ball_y2 - G.ball_y1;
  *x = G.ball_x1;
  *y = G.ball_y1;
  *r = hypot(dx, dy);
}

static void draw_ball(cairo_t *cr) {
  double x, y, r;
  ball_circle(&x, &y, &r);
  cairo_arc(cr, x, y, r, 0, 6.283185307179586);
  if (G.dark)
  {
    cairo_set_source_rgba(cr, 1.0, 0.75, 0.75, 0.25);
  }
  else
  {
    cairo_set_source_rgba(cr, 0.5, 0.25, 0.25, 0.25);
  }
  cairo_fill_preserve(cr);
  if (G.dark)
  {
    cairo_set_source_rgba(cr, 1.0, 0.75, 0.75, 1);
  }
  else
  {
    cairo_set_source_rgba(cr, 0.5, 0.25, 0.25, 1);
  }
  cairo_stroke(cr);
}

static void draw_text(cairo_t *ctx, cairo_t *ctxmask, cairo_t *ctxsel, double x,↙
    ↪  double y, const char *label)
{
  cairo_surface_flush(G.mask);
  uint32_t *pixels = (uint32_t *) cairo_image_surface_get_data(G.mask);
  int channels = 4;
  int stride = cairo_image_surface_get_stride(G.mask) / channels;
  bool dodraw = true;
  if (0 <= x && x < G.width_screen && 0 <= y && y < G.height_screen) {
    int i = x;
    int j = y;
```

```
      int k = stride * j + i;
1070  dodraw = (pixels[k] != 0xffffffff);
    }
    if (dodraw)
    {
      if (-G.width_screen <= x && x < 2 * G.width_screen && -G.height_screen <= y ↙
          ↳ && y < 2 * G.height_screen) {
1075    cairo_select_font_face(ctx, "LMSans10", CAIRO_FONT_SLANT_NORMAL, ↙
            ↳ CAIRO_FONT_WEIGHT_BOLD);
        cairo_set_font_size(ctx, 16.0 * G.dpi_screen / 100);
        cairo_text_extents_t e;
        cairo_text_extents(ctx, label, &e);
        cairo_move_to(ctx, x - e.width / 2.0, y + e.height / 2.0);
1080    cairo_text_path(ctx, label);
        cairo_fill(ctx);
        // mask out rectangle centered on same point
        cairo_move_to(ctxmask, x - e.width * 2, y + e.height * 2);
        cairo_line_to(ctxmask, x + e.width * 2, y + e.height * 2);
1085    cairo_line_to(ctxmask, x + e.width * 2, y - e.height * 2);
        cairo_line_to(ctxmask, x - e.width * 2, y - e.height * 2);
        cairo_line_to(ctxmask, x - e.width * 2, y + e.height * 2);
        cairo_fill(ctxmask);
        cairo_move_to(ctxsel, x - e.width * 2, y + e.height * 2);
1090    cairo_line_to(ctxsel, x + e.width * 2, y + e.height * 2);
        cairo_line_to(ctxsel, x + e.width * 2, y - e.height * 2);
        cairo_line_to(ctxsel, x - e.width * 2, y - e.height * 2);
        cairo_line_to(ctxsel, x - e.width * 2, y + e.height * 2);
        cairo_fill(ctxsel);
1095  }
    }
  }

  static cairo_pattern_t *generate_fill_pattern(cairo_surface_t *parent, int p, ↙
      ↳ double r, double g, double b)
1100 {
    cairo_surface_t *surface = 0;
    if (parent)
    {
      surface = cairo_surface_create_similar(parent, CAIRO_CONTENT_COLOR_ALPHA, ↙
          ↳ 16, 16);
1105  }
    else
    {
      surface = cairo_image_surface_create(CAIRO_FORMAT_ARGB32, 16, 16);
    }
1110  cairo_t *cr = cairo_create(surface);
    cairo_set_line_cap(cr, CAIRO_LINE_CAP_ROUND);
    cairo_set_line_join(cr, CAIRO_LINE_JOIN_ROUND);
    cairo_set_line_width(cr, 1);
    cairo_set_source_rgba(cr, r, g, b, 0.5);
1115  switch (((p % 26) + 26) % 26)
    {
      // short and long dashes in 4 directions
      case  0: { cairo_move_to(cr,  5,  8); cairo_line_to(cr, 11,  8); break; }
      case  1: { cairo_move_to(cr,  3,  8); cairo_line_to(cr, 13,  8); break; }
1120  case  2: { cairo_move_to(cr,  8,  5); cairo_line_to(cr,  8, 11); break; }
      case  3: { cairo_move_to(cr,  8,  4); cairo_line_to(cr,  8, 13); break; }
```

```
        case  4: { cairo_move_to(cr,  5,  5); cairo_line_to(cr, 11, 11); break; }
        case  5: { cairo_move_to(cr,  3,  4); cairo_line_to(cr, 13, 13); break; }
        case  6: { cairo_move_to(cr,  5, 11); cairo_line_to(cr, 11,  5); break; }
        case  7: { cairo_move_to(cr,  3, 13); cairo_line_to(cr, 13,  5); break; }
        // small and large crosses in 2 directions
        case  8: { cairo_move_to(cr,  5,  8); cairo_line_to(cr, 11,  8); ↙
            ↳ cairo_move_to(cr,  8,  5); cairo_line_to(cr,  8, 11); break; }
        case  9: { cairo_move_to(cr,  3,  8); cairo_line_to(cr, 13,  8); ↙
            ↳ cairo_move_to(cr,  8,  3); cairo_line_to(cr,  8, 13); break; }
        case 10: { cairo_move_to(cr,  5,  5); cairo_line_to(cr, 11, 11); ↙
            ↳ cairo_move_to(cr,  5, 11); cairo_line_to(cr, 11,  5); break; }
        case 11: { cairo_move_to(cr,  3,  3); cairo_line_to(cr, 13, 13); ↙
            ↳ cairo_move_to(cr,  3, 13); cairo_line_to(cr, 13,  3); break; }
        case 12: { cairo_move_to(cr,  3,  8); cairo_line_to(cr, 13,  8); ↙
            ↳ cairo_move_to(cr,  8,  3); cairo_line_to(cr,  8, 13);
                  cairo_move_to(cr,  3,  3); cairo_line_to(cr, 13, 13); ↙
                      ↳ cairo_move_to(cr,  3, 13); cairo_line_to(cr, 13,  3); break↙
                      ↳ ; }
        case 13: { cairo_move_to(cr,  5,  8); cairo_line_to(cr, 11,  8); ↙
            ↳ cairo_move_to(cr,  8,  5); cairo_line_to(cr,  8, 11);
                  cairo_move_to(cr,  5,  5); cairo_line_to(cr, 11, 11); ↙
                      ↳ cairo_move_to(cr,  5, 11); cairo_line_to(cr, 11,  5); break↙
                      ↳ ; }
        // small and large triangles in 4 directions
        case 14: { cairo_move_to(cr,  5,  5); cairo_line_to(cr, 11,  5); ↙
            ↳ cairo_line_to(cr,  8, 11); cairo_close_path(cr); break; }
        case 15: { cairo_move_to(cr,  3,  3); cairo_line_to(cr, 13,  3); ↙
            ↳ cairo_line_to(cr,  8, 13); cairo_close_path(cr); break; }
        case 16: { cairo_move_to(cr,  5, 11); cairo_line_to(cr, 11, 11); ↙
            ↳ cairo_line_to(cr,  8,  5); cairo_close_path(cr); break; }
        case 17: { cairo_move_to(cr,  3, 13); cairo_line_to(cr, 13, 13); ↙
            ↳ cairo_line_to(cr,  8,  3); cairo_close_path(cr); break; }
        case 18: { cairo_move_to(cr,  5,  5); cairo_line_to(cr,  5, 11); ↙
            ↳ cairo_line_to(cr, 11,  8); cairo_close_path(cr); break; }
        case 19: { cairo_move_to(cr,  3,  3); cairo_line_to(cr,  3, 13); ↙
            ↳ cairo_line_to(cr, 13,  8); cairo_close_path(cr); break; }
        case 20: { cairo_move_to(cr, 11,  5); cairo_line_to(cr, 11, 11); ↙
            ↳ cairo_line_to(cr,  5,  8); cairo_close_path(cr); break; }
        case 21: { cairo_move_to(cr, 13,  3); cairo_line_to(cr, 13, 13); ↙
            ↳ cairo_line_to(cr,  3,  8); cairo_close_path(cr); break; }
        // small and large squares in 2 directions
        case 22: { cairo_move_to(cr,  5,  5); cairo_line_to(cr, 11,  5); ↙
            ↳ cairo_line_to(cr, 11, 11); cairo_line_to(cr,  5, 11); cairo_close_path↙
            ↳ (cr); break; }
        case 23: { cairo_move_to(cr,  3,  3); cairo_line_to(cr, 13,  3); ↙
            ↳ cairo_line_to(cr, 13, 13); cairo_line_to(cr,  3, 13); cairo_close_path↙
            ↳ (cr); break; }
        case 24: { cairo_move_to(cr,  5,  8); cairo_line_to(cr,  8,  5); ↙
            ↳ cairo_line_to(cr, 11,  8); cairo_line_to(cr,  8, 11); cairo_close_path↙
            ↳ (cr); break; }
        case 25: { cairo_move_to(cr,  3,  8); cairo_line_to(cr,  8,  3); ↙
            ↳ cairo_line_to(cr, 13,  8); cairo_line_to(cr,  8, 13); cairo_close_path↙
            ↳ (cr); break; }
    }
    cairo_stroke(cr);
    cairo_destroy(cr);
    cairo_pattern_t *pattern = cairo_pattern_create_for_surface(surface);
```

```
  cairo_pattern_set_extend(pattern, CAIRO_EXTEND_REPEAT);
  cairo_surface_destroy(surface);
  return pattern;
}

static void initialize_fill_patterns(void)
{
  for (int dark = 0; dark < 2; ++dark)
  {
    for (int p = 0; p < 26; ++p)
    {
      G.fillpattern[dark][p] = generate_fill_pattern(0, p, dark, dark, dark);
    }
  }
}

struct dash { double pattern[8]; int count; };
static const struct dash dashes[16] =
#define gap 4
  { { { 0 }, 0 }
  , { { 2, gap }, 2 }
  , { { 4, gap }, 2 }
  , { { 8, gap }, 2 }
  , { { 2, gap, 4, gap }, 4 }
  , { { 2, gap, 8, gap }, 4 }
  , { { 4, gap, 8, gap }, 4 }
  , { { 2, gap, 2, gap, 4, gap }, 6 }
  , { { 2, gap, 2, gap, 8, gap }, 6 }
  , { { 2, gap, 4, gap, 4, gap }, 6 }
  , { { 2, gap, 8, gap, 8, gap }, 6 }
  , { { 4, gap, 8, gap, 8, gap }, 6 }
  , { { 2, gap, 2, gap, 2, gap, 4, gap }, 8 }
  , { { 2, gap, 2, gap, 2, gap, 8, gap }, 8 }
  , { { 2, gap, 2, gap, 4, gap, 4, gap }, 8 }
  , { { 4, gap, 2, gap, 8, gap, 2, gap }, 8 }
  };
#undef gap

static void draw_annotation(cairo_t *ctx, cairo_t *ctxmask, cairo_t *ctxsel, ↙
    ↳ struct annotation *ls) {
  double alpha = 1;
  if (ls->tag == annotation_wake)
  {
    alpha = 0.25;
  }
  if (ls->selected) {
    if (G.dark)
    {
      cairo_set_source_rgba(ctx, 1.0, 0.75, 0.75, alpha);
    }
    else
    {
      cairo_set_source_rgba(ctx, 0.5, 0.25, 0.25, alpha);
    }
    cairo_set_line_width(ctx, 3);
  } else {
    if (G.dark)
```

```
    {
1210  cairo_set_source_rgba(ctx, 1, 1, 1, alpha);
    }
    else
    {
      cairo_set_source_rgba(ctx, 0, 0, 0, alpha);
1215  }
    cairo_set_line_width(ctx, 1);
  }
  cairo_set_source_rgba(ctxsel, ((ls->id >> 16) & 0xFF) / 255.0, ((ls->id >> 8) ↙
      ↳ & 0xFF) / 255.0, ((ls->id >> 0) & 0xFF) / 255.0, 1);
  struct point *l = 0;
1220  int lt = 0;
  int ft = -1;
  double fcr = -1;
  double fcg = -1;
  double fcb = -1;
1225  double scr = -1;
  double scg = -1;
  double scb = -1;
  int circle_at_line_end = 0;
  switch (ls->tag) {
1230    case annotation_ray_in:
      l = ls->u.ray_in.line_start;
      lt = ls->u.ray_in.line_type;
      scr = ls->u.ray_in.stroke_r;
      scg = ls->u.ray_in.stroke_g;
1235      scb = ls->u.ray_in.stroke_b;
      break;
    case annotation_ray_out:
      l = ls->u.ray_out.line_start;
      lt = ls->u.ray_out.line_type;
1240      scr = ls->u.ray_out.stroke_r;
      scg = ls->u.ray_out.stroke_g;
      scb = ls->u.ray_out.stroke_b;
      break;
    case annotation_text: {
1245      double x, y;
      param_to_screen(&x, &y, mpc_realref(ls->u.text.xy), mpc_imagref(ls->u.text↙
          ↳ .xy));
      draw_text(ctx, ctxmask, ctxsel, x, y, ls->label);
      break;
    }
1250    case annotation_nucleus: {
      double x, y;
      param_to_screen(&x, &y, mpc_realref(ls->u.nucleus.xy), mpc_imagref(ls->u.↙
          ↳ nucleus.xy));
      draw_text(ctx, ctxmask, ctxsel, x, y, ls->label);
      break;
1255  }
    case annotation_misiurewicz: {
      double x, y;
      param_to_screen(&x, &y, mpc_realref(ls->u.misiurewicz.xy), mpc_imagref(ls↙
          ↳ ->u.misiurewicz.xy));
      draw_text(ctx, ctxmask, ctxsel, x, y, ls->label);
1260      break;
    }
```

```
      case annotation_wake:
        l = ls->u.wake.line_start;
        ft = ls->u.wake.fill_type;
        scr = fcr = ls->u.wake.fill_r;
        scg = fcg = ls->u.wake.fill_g;
        scb = fcb = ls->u.wake.fill_b;
        break;
      case annotation_atom:
        l = ls->u.atom.line_start;
        ft = ls->u.atom.fill_type;
        scr = fcr = ls->u.atom.fill_r;
        scg = fcg = ls->u.atom.fill_g;
        scb = fcb = ls->u.atom.fill_b;
        break;
      case annotation_domain:
        l = ls->u.domain.line_start;
        lt = ls->u.domain.line_type;
        scr = ls->u.domain.line_r;
        scg = ls->u.domain.line_g;
        scb = ls->u.domain.line_b;
        circle_at_line_end = 1;
        break;
    }
    if (l) {
      cairo_set_line_width(ctxsel, 8.0);
      int first = 1;
      double x, y, oldx = 0, oldy = 0;
      int oldinbounds = 0;
      int count = 0;
      struct point *l0 = l;
      for (; l; l = l->next) {
        count++;
        param_to_screen_clamped(&x, &y, mpc_realref(l->xy), mpc_imagref(l->xy));
        int inbounds = 0 <= x && x < G.width_screen && 0 <= y && y < G.↙
            ↳ height_screen;
        if (oldinbounds) {
          cairo_line_to(ctx, x, y);
          cairo_line_to(ctxsel, x, y);
        } else {
          if (inbounds) {
            if (first) {
              cairo_move_to(ctx, x, y);
              cairo_move_to(ctxsel, x, y);
            } else {
              cairo_line_to(ctx, oldx, oldy);
              cairo_line_to(ctx, x, y);
              cairo_line_to(ctxsel, oldx, oldy);
              cairo_line_to(ctxsel, x, y);
            }
          } else {
            // segment out of bounds
            cairo_line_to(ctx, x, y);
            cairo_line_to(ctxsel, x, y);
          }
        }
        oldinbounds = inbounds;
        oldx = x;
```

```
      oldy = y;
      first = 0;
    }
    if (ls->selected)
    {
      cairo_set_dash(ctx, dashes[0].pattern, 0, 0);
    }
    else
    {
      cairo_set_dash(ctx, dashes[lt & 0xF].pattern, dashes[lt & 0xF].count, 0);
    }
    if (ft >= 0)
    {
      cairo_pattern_t *old = cairo_get_source(ctx);
      cairo_pattern_reference(old);
      if (G.colour == colour_theme_full_colour)
      {
        cairo_set_source_rgba(ctx, fcr, fcg, fcb, 0.25);
        cairo_fill_preserve(ctx);
      }
      if (G.colour == colour_theme_monochrome)
      {
        fcr = fcg = fcb = G.dark;
      }
      cairo_pattern_t *fp = generate_fill_pattern(cairo_get_target(ctx), ft, fcr
          , fcg, fcb);
      cairo_set_source(ctx, fp);
      cairo_fill_preserve(ctx);
      cairo_set_source(ctx, old);
      cairo_pattern_destroy(fp);
      cairo_fill_preserve(ctxsel);
    }
    if (G.colour && scr >= 0)
    {
      cairo_set_source_rgba(ctx, scr, scg, scb, 1);
    }
    cairo_stroke(ctx);
    cairo_stroke(ctxsel);
    if (circle_at_line_end)
    {
      int i = 0;
      for (l = l0; l; l = l->next)
      {
        if (i % (count / 4) == 0)
        {
          param_to_screen_clamped(&x, &y, mpc_realref(l->xy), mpc_imagref(l->xy)
              );
          cairo_arc(ctx, x, y, 8.0 * G.dpi_screen / 100, 0, twopi);
          if (i == 0)
            cairo_fill(ctx);
          else
            cairo_stroke(ctx);
          cairo_arc(ctxsel, x, y, 8.0 * G.dpi_screen / 100, 0, twopi);
          if (i == 0)
            cairo_fill(ctx);
          else
            cairo_stroke(ctxsel);
```

```
            }
            ++i;
          }
        }
      }
    }

    static gboolean configure_event_cb(GtkWidget *widget, GdkEventConfigure *event, ↙
        ↳ gpointer data) {
      (void) event;
      (void) data;
      if (G.surface) {
        return TRUE;
      }
      G.surface = cairo_image_surface_create(CAIRO_FORMAT_ARGB32, G.width_print, G.↙
          ↳ height_print);
      G.mask = cairo_image_surface_create(CAIRO_FORMAT_ARGB32, G.width_screen, G.↙
          ↳ height_screen);
      G.selection_surface = cairo_image_surface_create(CAIRO_FORMAT_ARGB32, G.↙
          ↳ width_screen, G.height_screen);
      struct view *v = view_copy(G.view);
      start_render(v);
      gtk_widget_queue_draw(widget);
      return TRUE;
    }

    struct draw_annotation_foreach_data
    {
      int selected;
      cairo_t *cr;
      cairo_t *crmask;
      cairo_t *crsel;
    };

    static gboolean draw_annotation_foreach_func(GtkTreeModel *model, GtkTreePath *↙
        ↳ path, GtkTreeIter *iter, gpointer userdata)
    {
      (void) path;
      struct draw_annotation_foreach_data *d = userdata;
      gpointer thing;
      gtk_tree_model_get(model, iter, 1, &thing, -1);
      struct annotation *ls = thing;
      if (ls && ls->selected == d->selected && ls->tag != annotation_atom && ls->tag↙
          ↳ != annotation_wake) {
        draw_annotation(d->cr, d->crmask, d->crsel, ls);
      }
      return FALSE;
    }

    static void clip_subtract_loop(cairo_t *cr, struct point *l)
    {
      cairo_move_to(cr, -10, -10);
      cairo_line_to(cr, -10, G.height_screen + 10);
      cairo_line_to(cr, G.width_screen + 10, G.height_screen + 10);
      cairo_line_to(cr, G.width_screen + 10, -10);
      cairo_line_to(cr, -10, -10);
      cairo_close_path(cr);
```

```
  int first = 1;
  for (; l; l = l->next)
  {
    double x = -10, y = -10;
    param_to_screen_clamped(&x, &y, mpc_realref(l->xy), mpc_imagref(l->xy));
    if (first) {
      cairo_move_to(cr, x, y);
      first = 0;
    }
    else
    {
      cairo_line_to(cr, x, y);
    }
  }
  cairo_close_path(cr);
  cairo_clip(cr);
}

static gboolean draw_cb(GtkWidget *widget, cairo_t *cr, gpointer data) {
  (void) widget;
  (void) data;
  cairo_save(cr);
  cairo_scale(cr, G.width_screen / (double) G.width_print, G.height_screen / (↙
      ↳ double) G.height_print);
  cairo_set_source_surface(cr, G.surface, 0, 0);
  cairo_paint(cr);
  cairo_restore(cr);
  if (G.box) {
    draw_box(cr);
  }
  if (G.ball) {
    draw_ball(cr);
  }
  cairo_t *crmask = cairo_create(G.mask);
  cairo_set_source_rgba(crmask, 0, 0, 0, 1);
  cairo_paint(crmask);
  cairo_set_source_rgba(crmask, 1, 1, 1, 1);
  cairo_t *crsel = cairo_create(G.selection_surface);
  cairo_set_source_rgba(crsel, 0, 0, 0, 1);
  cairo_paint(crsel);
  cairo_set_antialias(crsel, CAIRO_ANTIALIAS_NONE);
  for (int selected = 1; selected >= 0; --selected)
  {
    struct draw_annotation_foreach_data userdata = { selected, cr, crmask, crsel↙
        ↳ };
    gtk_tree_model_foreach(GTK_TREE_MODEL(G.annostore), ↙
        ↳ draw_annotation_foreach_func, &userdata);
  }

  cairo_set_fill_rule(cr, CAIRO_FILL_RULE_EVEN_ODD);
  cairo_set_fill_rule(crsel, CAIRO_FILL_RULE_EVEN_ODD);
  // draw atoms
  struct annotation *as = G.anno;
  while (as)
  {
    if (as->tag == annotation_atom)
    {
```

```
      draw_annotation(cr, crmask, crsel, as);
      clip_subtract_loop(cr, as->u.atom.line_start);
      clip_subtract_loop(crsel, as->u.atom.line_start);
    }
    as = as->next;
  }
  // draw wakes from narrowest to widest
  struct annotation *ws = G.wakes_head.u.wake.next_wake;
  while (ws->u.wake.next_wake)
  {
    draw_annotation(cr, crmask, crsel, ws);
    clip_subtract_loop(cr, ws->u.wake.line_start);
    clip_subtract_loop(crsel, ws->u.wake.line_start);
    ws = ws->u.wake.next_wake;
  }
  cairo_destroy(crmask);
  cairo_destroy(crsel);
  return FALSE;
}

static void save_png(const char *filename) {
  cairo_surface_t *surface = cairo_image_surface_create(CAIRO_FORMAT_ARGB32, G.↙
      ↳ width_print, G.height_print);
  cairo_t *cr = cairo_create(surface);
  cairo_scale(cr, G.width_print / (double) G.width_screen, G.height_print / (↙
      ↳ double) G.height_screen);
  draw_cb(0, cr, 0);
  cairo_destroy(cr);
  cairo_surface_write_to_png(surface, filename);
  cairo_surface_destroy(surface);
}

static void save_pdf(const char *filename) {
  double w = 210 / 25.4 * 72;
  double h = 297 / 25.4 * 72;
  // double gw = (210 - 40) / 25.4 * 72;
  double gh = (297 - 40) / 25.4 * 72;
  double s = gh / G.width_screen;
  cairo_surface_t *pdf = cairo_pdf_surface_create(filename, w, h);
  cairo_t *cr = cairo_create(pdf);
  cairo_translate(cr, w / 2, h / 2);
  cairo_rotate(cr, -pi / 2);
  cairo_scale(cr, s, s);
  cairo_translate(cr, -G.width_screen / 2, -G.height_screen / 2);
  cairo_rectangle(cr, 0, 0, G.width_screen, G.height_screen);
  cairo_clip(cr);
  draw_cb(0, cr, 0);
  cairo_show_page(cr);
  cairo_destroy(cr);
  cairo_surface_destroy(pdf);
}

static void close_window(void) {
  if (G.surface) {
    cairo_surface_destroy(G.surface);
  }
  gtk_main_quit();
```

```
}

static void save_clicked_cb(GtkToolButton *save, gpointer userdata) {
  (void) save;
  (void) userdata;
  GtkWidget *dialog = gtk_file_chooser_dialog_new("Save File", GTK_WINDOW(G.window), GTK_FILE_CHOOSER_ACTION_SAVE, "_Cancel", GTK_RESPONSE_CANCEL, "_Save", GTK_RESPONSE_ACCEPT, NULL);
  gtk_file_chooser_set_do_overwrite_confirmation(GTK_FILE_CHOOSER(dialog), TRUE);
  gint res = gtk_dialog_run(GTK_DIALOG(dialog));
  if (res == GTK_RESPONSE_ACCEPT) {
    char *filename = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog));
    serialize(filename); // FIXME TODO handle errors
    g_free(filename);
  }
  gtk_widget_destroy(dialog);
}

static void save_image_clicked_cb(GtkToolButton *save_image, gpointer userdata) {
  (void) save_image;
  (void) userdata;
  GtkWidget *dialog = gtk_file_chooser_dialog_new("Save PNG", GTK_WINDOW(G.window), GTK_FILE_CHOOSER_ACTION_SAVE, "_Cancel", GTK_RESPONSE_CANCEL, "_Save", GTK_RESPONSE_ACCEPT, NULL);
  gtk_file_chooser_set_do_overwrite_confirmation(GTK_FILE_CHOOSER(dialog), TRUE);
  gint res = gtk_dialog_run(GTK_DIALOG(dialog));
  if (res == GTK_RESPONSE_ACCEPT) {
    char *filename = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog));
    save_png(filename); // FIXME TODO handle errors
    g_free(filename);
  }
  gtk_widget_destroy(dialog);
}

static void save_pdf_clicked_cb(GtkToolButton *save_image, gpointer userdata) {
  (void) save_image;
  (void) userdata;
  GtkWidget *dialog = gtk_file_chooser_dialog_new("Save PDF", GTK_WINDOW(G.window), GTK_FILE_CHOOSER_ACTION_SAVE, "_Cancel", GTK_RESPONSE_CANCEL, "_Save", GTK_RESPONSE_ACCEPT, NULL);
  gtk_file_chooser_set_do_overwrite_confirmation(GTK_FILE_CHOOSER(dialog), TRUE);
  gint res = gtk_dialog_run(GTK_DIALOG(dialog));
  if (res == GTK_RESPONSE_ACCEPT) {
    char *filename = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog));
    save_pdf(filename); // FIXME TODO handle errors
    g_free(filename);
  }
  gtk_widget_destroy(dialog);
}

static void load_clicked_cb(GtkToolButton *load, gpointer userdata) {
  (void) load;
  (void) userdata;
```

```
1580    GtkWidget *dialog = gtk_file_chooser_dialog_new("Load File", GTK_WINDOW(G.↙
            ↳ window), GTK_FILE_CHOOSER_ACTION_OPEN, "_Cancel", GTK_RESPONSE_CANCEL, "↙
            ↳ _Open", GTK_RESPONSE_ACCEPT, NULL);
        gint res = gtk_dialog_run(GTK_DIALOG(dialog));
        if (res == GTK_RESPONSE_ACCEPT) {
          char *filename = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog));
          deserialize(filename); // FIXME TODO handle errors
1585      g_free(filename);
        }
        gtk_widget_destroy(dialog);
      }

1590  #if 0
      static void resize_clicked_cb(GtkToolButton *resize, gpointer userdata) {
        (void) resize;
        (void) userdata;
        GtkDialogFlags flags = GTK_DIALOG_MODAL | GTK_DIALOG_DESTROY_WITH_PARENT;
1595    GtkWidget *dialog = gtk_dialog_new_with_buttons("Size", GTK_WINDOW(G.window), ↙
            ↳ flags, "_Cancel", GTK_RESPONSE_CANCEL, "_Resize", GTK_RESPONSE_ACCEPT, ↙
            ↳ NULL);
        GtkWidget *box = gtk_dialog_get_content_area (GTK_DIALOG(dialog));
        GtkWidget *entryw = gtk_entry_new();
        GtkWidget *entryh = gtk_entry_new();
        char *labelw = malloc(100);
1600    snprintf(labelw, 99, "%d", G.width_screen);
        char *labelh = malloc(100);
        snprintf(labelh, 99, "%d", G.height_screen);
        gtk_entry_set_text(GTK_ENTRY(entryw), labelw);
        gtk_entry_set_text(GTK_ENTRY(entryh), labelh);
1605    gtk_container_add(GTK_CONTAINER(box), entryw);
        gtk_container_add(GTK_CONTAINER(box), entryh);
        gtk_widget_show_all(dialog);
        gint res = gtk_dialog_run(GTK_DIALOG(dialog));
        if (res == GTK_RESPONSE_ACCEPT) {
1610      const char *textw = gtk_entry_get_text(GTK_ENTRY(entryw));
          int w = atoi(textw);
          const char *texth = gtk_entry_get_text(GTK_ENTRY(entryh));
          int h = atoi(texth);
          if (w > 0 && h > 0) {
1615        resize_image(w, h);
          }
        }
        free(labelw);
        free(labelh);
1620    gtk_widget_destroy(dialog);
      }
      #endif

      static void home_clicked_cb(GtkToolButton *home, gpointer userdata) {
1625    (void) home;
        (void) userdata;
        struct view *v = view_new();
        mpfr_set_d(v->cx, -0.75, GMP_RNDN);
        mpfr_set_d(v->cy, 0.0, GMP_RNDN);
1630    mpfr_set_d(v->radius, 1.5, GMP_RNDN);
        start_render(v);
      }
```

```
static void zoom_out_clicked_cb(GtkToolButton *zoom_out, gpointer userdata) {
  (void) zoom_out;
  (void) userdata;
  struct view *v = view_copy(G.view);
  mpfr_mul_d(v->radius, v->radius, 10.0, GMP_RNDN);
  start_render(v);
}

static void wakel_clicked_cb(GtkToolButton *wakel, gpointer userdata) {
  (void) wakel;
  (void) userdata;
  struct annotation *anno = get_selected_annotation();
  if (anno && anno->tag == annotation_ray_in)
  {
    start_wake(anno, 1);
  }
}

static void waker_clicked_cb(GtkToolButton *waker, gpointer userdata) {
  (void) waker;
  (void) userdata;
  struct annotation *anno = get_selected_annotation();
  if (anno && anno->tag == annotation_ray_in)
  {
    start_wake(anno, -1);
  }
}


static void more_iters_clicked_cb(GtkToolButton *more_iters, gpointer userdata) ↙
    ↳ {
  (void) more_iters;
  (void) userdata;
  if (G.maximum_iterations < (1 << 30))
  {
    G.maximum_iterations *= 2;
    struct view *v = view_copy(G.view);
    start_render(v);
  }
}

static void fewer_iters_clicked_cb(GtkToolButton *fewer_iters, gpointer userdata↙
    ↳ ) {
  (void) fewer_iters;
  (void) userdata;
  if (G.maximum_iterations > (1 << 8))
  {
    G.maximum_iterations /= 2;
    struct view *v = view_copy(G.view);
    start_render(v);
  }
}

static gboolean zoom_button_press_event_cb (GtkWidget *widget, GdkEventButton *↙
    ↳ event, gpointer data) {
  (void) data;
```

```
  if (event->button == GDK_BUTTON_PRIMARY) {
    G.box = TRUE;
    G.box_x1 = event->x;
    G.box_y1 = event->y;
    G.box_x2 = event->x;
    G.box_y2 = event->y;
    gtk_widget_queue_draw(widget);
  } else {
    G.box = FALSE;
    gtk_widget_queue_draw(widget);
  }
  return TRUE;
}

static gboolean zoom_button_release_event_cb(GtkWidget *widget, GdkEventButton *event, gpointer data) {
  (void) data;
  if (event->button == GDK_BUTTON_PRIMARY) {
    if (G.box) {
      if (G.view) {
        double x, y, w, h;
        box_rectangle(&x, &y, &w, &h);
        if (w > 0 && h > 0) {
          struct view *v = view_new();
          x = (x + w/2.0 - G.width_screen / 2.0) / (G.height_screen / 2.0);
          y = (G.height_screen / 2.0 - (y + h/2.0)) / (G.height_screen / 2.0);
          double r = (h/2.0) / (G.height_screen / 2.0);
          mpfr_t clickx, clicky, logradius;
          mpfr_init2(clickx, 53);
          mpfr_init2(clicky, 53);
          mpfr_init2(logradius, 53);
          mpfr_set_d(clickx, x, GMP_RNDN);
          mpfr_set_d(clicky, y, GMP_RNDN);
          mpfr_mul(clickx, clickx, G.view->radius, GMP_RNDN);
          mpfr_mul(clicky, clicky, G.view->radius, GMP_RNDN);
          mpfr_mul_d(v->radius, G.view->radius, r, GMP_RNDN);
          mpfr_log2(logradius, v->radius, GMP_RNDN);
          int prec = fmax(53, 16 - mpfr_get_d(logradius, GMP_RNDN));
          mpfr_prec_round(v->cx, prec, GMP_RNDN);
          mpfr_prec_round(v->cy, prec, GMP_RNDN);
          mpfr_add(v->cx, G.view->cx, clickx, GMP_RNDN);
          mpfr_add(v->cy, G.view->cy, clicky, GMP_RNDN);
          mpfr_clear(clickx);
          mpfr_clear(clicky);
          mpfr_clear(logradius);
          start_render(v);
        }
      }
      G.box = FALSE;
      gtk_widget_queue_draw (widget);
    }
  }
  return TRUE;
}

static gboolean zoom_motion_notify_event_cb(GtkWidget *widget, GdkEventMotion *event, gpointer data) {
```

```
  (void) data;
  if (event->state & GDK_BUTTON1_MASK) {
    G.box_x2 = event->x;
    G.box_y2 = event->y;
    gtk_widget_queue_draw(widget);
  }
  return TRUE;
}

static void zoom_toggled_cb(GtkWidget *zoom, gpointer userdata) {
  (void) userdata;
  gboolean active = gtk_check_menu_item_get_active(GTK_CHECK_MENU_ITEM(zoom));
  if (active) {
    G.box_aspect = G.width_screen / (double) G.height_screen;
    G.zoom_motion_notify_handler = g_signal_connect(G.da, "motion-notify-event",↙
        ↳ G_CALLBACK(zoom_motion_notify_event_cb), NULL);
    G.zoom_button_press_handler = g_signal_connect(G.da, "button-press-event", ↙
        ↳ G_CALLBACK(zoom_button_press_event_cb), NULL);
    G.zoom_button_release_handler = g_signal_connect(G.da, "button-release-event↙
        ↳ ", G_CALLBACK(zoom_button_release_event_cb), NULL);
  } else {
    if (G.zoom_motion_notify_handler) {
      g_signal_handler_disconnect(G.da, G.zoom_motion_notify_handler);
    }
    if (G.zoom_button_press_handler) {
      g_signal_handler_disconnect(G.da, G.zoom_button_press_handler);
    }
    if (G.zoom_button_release_handler) {
      g_signal_handler_disconnect(G.da, G.zoom_button_release_handler);
    }
    G.zoom_motion_notify_handler = 0;
    G.zoom_button_press_handler = 0;
    G.zoom_button_release_handler = 0;
  }
}

static void info_update(double x, double y) {
  int i = x * G.width_print / G.width_screen, j = y * G.height_print / G.↙
      ↳ height_screen;
  if (G.image && 0 <= i && i < G.width_print && 0 <= j && j < G.height_print) {
    mpfr_t cx, cy;
    mpfr_init2(cx, 53);
    mpfr_init2(cy, 53);
    screen_to_param(x, y, cx, cy);
    int k = j * G.width_print + i;
    char *buf = calloc(1, 65536);
    mpfr_snprintf(buf, 65536,
      "\nINFO\n"
      "pixel: %d %d\n"
      "real: %Re\n"
      "imag: %Re\n"
      "dwell_n: %d\n"
      "dwell_f: %g\n"
      "angle_f: %g\n"
      "distance: %g\n"
      "domain_n: %d\n"
      "domain_x: %g\n"
```

```
1795          "domain_y: %g\n"
              , i, j, cx, cy
              , perturbator_get_dwell_n(G.image)[k]
              , perturbator_get_dwell_f(G.image)[k]
              , perturbator_get_dwell_a(G.image)[k]
1800          , perturbator_get_distance(G.image)[k]
              , perturbator_get_domain_n(G.image)[k]
              , perturbator_get_domain_x(G.image)[k]
              , perturbator_get_domain_y(G.image)[k]
              );
1805        log_append(buf);
            free(buf);
            mpfr_clear(cx);
            mpfr_clear(cy);
          }
1810    }

        #if 0
        static double compute_minimum_exterior_dwell(void)
        {
1815      const int32_t *dwell_n = perturbator_get_dwell_n(G.image);
          const float   *dwell_f = perturbator_get_dwell_f(G.image);
          double dwell = 1.0 / 0.0;
          for (int i = 0; i < G.width_print; ++i)
          {
1820        for (int j = 0; j < G.height_print; j += G.height_print - 1)
            {
              int k = j * G.width_print + i;
              double n = dwell_n[k];
              double f = dwell_f[k];
1825          if (n > 0)
              {
                dwell = fmin(dwell, n + f);
              }
            }
1830      }
          for (int i = 0; i < G.width_print; i += G.width_print - 1)
          {
            for (int j = 0; j < G.height_print; ++j)
            {
1835          int k = j * G.width_print + i;
              double n = dwell_n[k];
              double f = dwell_f[k];
              if (n > 0)
              {
1840            dwell = fmin(dwell, n + f);
              }
            }
          }
          return dwell;
1845    }
        #endif

        static gboolean info_button_press_event_cb(GtkWidget *widget, GdkEventButton *↙
            ↳ event, gpointer data) {
          (void) widget;
1850      (void) data;
```

```
  if (event->button == GDK_BUTTON_PRIMARY) {
    info_update(event->x, event->y);
  }
  return TRUE;
}

static void info_toggled_cb(GtkWidget *info, gpointer userdata) {
  (void) userdata;
  gboolean active = gtk_check_menu_item_get_active(GTK_CHECK_MENU_ITEM(info));
  if (active) {
    G.info_button_press_handler = g_signal_connect(G.da, "button-press-event", ↙
        ↪ G_CALLBACK(info_button_press_event_cb), NULL);
  } else {
    if (G.info_button_press_handler) {
      g_signal_handler_disconnect(G.da, G.info_button_press_handler);
    }
    G.info_button_press_handler = 0;
  }
}

static void selection_update(double x, double y) {
  int i = x, j = y;
  cairo_surface_flush(G.selection_surface);
  uint32_t *pixels = (uint32_t *) cairo_image_surface_get_data(G.↙
      ↪ selection_surface);
  int channels = 4;
  int stride = cairo_image_surface_get_stride(G.selection_surface) / channels;
  if (0 <= i && i < G.width_screen && 0 <= j && j < G.height_screen)
  {
    int id = pixels[j * stride + i] & 0x00ffffff;
    struct annotation *ls = G.anno;
    while (ls)
    {
      ls = ls->next;
    }
    GtkTreeSelection *ts = gtk_tree_view_get_selection(GTK_TREE_VIEW(G.annotree)↙
        ↪ );
    gtk_tree_selection_unselect_all(ts);
    GtkTreeModel *model = GTK_TREE_MODEL(G.annostore);
    GtkTreeIter iter;
    if (gtk_tree_model_get_iter_first(model, &iter))
    {
      do
      {
        gpointer thing = 0;
        gtk_tree_model_get(model, &iter, 1, &thing, -1);
        if (thing)
        {
          struct annotation *ls = thing;
          ls->selected = (ls->id == id);
          if (ls->selected)
          {
            gtk_tree_selection_select_iter(ts, &iter);
          }
        }
      } while(gtk_tree_model_iter_next(model, &iter));
    }
```

```
1905    }
      }


      static gboolean selection_button_press_event_cb(GtkWidget *widget, ↙
          ↳ GdkEventButton *event, gpointer data) {
1910    (void) widget;
        (void) data;
        if (event->button == GDK_BUTTON_PRIMARY) {
          selection_update(event->x, event->y);
        }
1915    return TRUE;
      }

      static void selection_toggled_cb(GtkWidget *sel, gpointer userdata) {
        (void) userdata;
1920    gboolean active = gtk_check_menu_item_get_active(GTK_CHECK_MENU_ITEM(sel));
        if (active) {
          G.selection_button_press_handler = g_signal_connect(G.da, "button-press-↙
              ↳ event", G_CALLBACK(selection_button_press_event_cb), NULL);
        } else {
          if (G.selection_button_press_handler) {
1925        g_signal_handler_disconnect(G.da, G.selection_button_press_handler);
          }
          G.selection_button_press_handler = 0;
        }
      }
1930
      static gboolean ray_out_button_press_event_cb(GtkWidget *widget, GdkEventButton ↙
          ↳ *event, gpointer data) {
        (void) data;
        if (event->button == GDK_BUTTON_PRIMARY) {
          start_ray_out(event->x, event->y);
1935      gtk_widget_queue_draw(widget);
        }
        return TRUE;
      }

1940  static void ray_out_toggled_cb(GtkWidget *ray_out, gpointer userdata) {
        (void) userdata;
        gboolean active = gtk_check_menu_item_get_active(GTK_CHECK_MENU_ITEM(ray_out))↙
            ↳ ;
        if (active) {
          G.ray_out_button_press_handler = g_signal_connect(G.da, "button-press-event↙
              ↳ ", G_CALLBACK(ray_out_button_press_event_cb), NULL);
1945    } else {
          if (G.ray_out_button_press_handler) {
            g_signal_handler_disconnect(G.da, G.ray_out_button_press_handler);
          }
          G.ray_out_button_press_handler = 0;
1950    }
      }

      static void ray_in_clicked_cb(GtkToolButton *toolbutton, gpointer user_data) {
        (void) toolbutton;
1955    (void) user_data;
        const char *label = "";
```

```
  struct annotation *anno = get_selected_annotation();
  if (anno)
  {
1960    label = anno->label;
  }
  GtkDialogFlags flags = GTK_DIALOG_MODAL | GTK_DIALOG_DESTROY_WITH_PARENT;
  GtkWidget *dialog = gtk_dialog_new_with_buttons("Ray In", GTK_WINDOW(G.window)↙
      ↳ , flags, "_Cancel", GTK_RESPONSE_CANCEL, "_Ray In", GTK_RESPONSE_ACCEPT,↙
      ↳ NULL);
  GtkWidget *box = gtk_dialog_get_content_area (GTK_DIALOG(dialog));
1965  GtkWidget *aentry = gtk_entry_new();
  GtkWidget *ppentry = gtk_entry_new();
  GtkWidget *pentry = gtk_entry_new();
  GtkWidget *dentry = gtk_entry_new();
  gtk_entry_set_placeholder_text(GTK_ENTRY(aentry), "Angle");
1970  gtk_entry_set_placeholder_text(GTK_ENTRY(ppentry), "Pre-period");
  gtk_entry_set_placeholder_text(GTK_ENTRY(pentry), "Period");
  gtk_entry_set_placeholder_text(GTK_ENTRY(dentry), "Depth");
  gtk_entry_set_text(GTK_ENTRY(aentry), label);
  gtk_entry_set_text(GTK_ENTRY(dentry), "200");
1975  gtk_container_add(GTK_CONTAINER(box), aentry);
  gtk_container_add(GTK_CONTAINER(box), ppentry);
  gtk_container_add(GTK_CONTAINER(box), pentry);
  gtk_container_add(GTK_CONTAINER(box), dentry);
  gtk_widget_show_all(dialog);
1980  gint res = gtk_dialog_run(GTK_DIALOG(dialog));
  if (res == GTK_RESPONSE_ACCEPT) {
    const char *atext = gtk_entry_get_text(GTK_ENTRY(aentry));
    const char *pptext = gtk_entry_get_text(GTK_ENTRY(ppentry));
    const char *ptext = gtk_entry_get_text(GTK_ENTRY(pentry));
1985    const char *dtext = gtk_entry_get_text(GTK_ENTRY(dentry));
    int pp = atoi(pptext);
    int p = atoi(ptext);
    struct m_binangle *angle = calloc(1, sizeof(*angle));
    m_binangle_init(angle);
1990    const char *result;
    if (p > 0 && strlen(atext) >= (size_t) (pp + p))
    {
      char *atext2 = calloc(1, 1 + pp + 1 + p + 1 + 1);
      int k = 0;
1995      atext2[k++] = '.';
      for (int j = 0; j < pp; ++j)
      {
        atext2[k++] = atext[j];
      }
2000      atext2[k++] = '(';
      for (int j = pp; j < pp + p; ++j)
      {
        atext2[k++] = atext[j];
      }
2005      atext2[k++] = ')';
      atext2[k++] = 0;
      result = m_binangle_from_string(angle, atext2);
      free(atext2);
    }
2010    else
    {
```

```
      result = m_binangle_from_string(angle, atext);
    }
    int depth = atoi(dtext);
    if (angle && result && *result == 0 && depth > 0) {
      start_ray_in(angle, depth);
    }
  }
  gtk_widget_destroy(dialog);
}

static void extend_clicked_cb(GtkToolButton *toolbutton, gpointer user_data) {
  (void) toolbutton;
  (void) user_data;
  GtkTreeSelection *select = gtk_tree_view_get_selection(GTK_TREE_VIEW(G.↙
      ↳ annotree));
  GtkTreeIter iter;
  GtkTreeModel *model;
  gpointer thing;
  struct annotation *anno = 0;
  if (gtk_tree_selection_get_selected(select, &model, &iter))
  {
    gtk_tree_model_get(model, &iter, 1, &thing, -1);
    if (thing)
    {
      anno = (struct annotation *) thing;
    }
  }
  if (anno && anno->tag == annotation_ray_in) {
    GtkDialogFlags flags = GTK_DIALOG_MODAL | GTK_DIALOG_DESTROY_WITH_PARENT;
    GtkWidget *dialog = gtk_dialog_new_with_buttons("Extend", GTK_WINDOW(G.↙
        ↳ window), flags, "_Cancel", GTK_RESPONSE_CANCEL, "_Extend", ↙
        ↳ GTK_RESPONSE_ACCEPT, NULL);
    GtkWidget *box = gtk_dialog_get_content_area (GTK_DIALOG(dialog));
    GtkWidget *dentry = gtk_entry_new();
    gtk_entry_set_text(GTK_ENTRY(dentry), "1000");
    gtk_container_add(GTK_CONTAINER(box), dentry);
    gtk_widget_show_all(dialog);
    gint res = gtk_dialog_run(GTK_DIALOG(dialog));
    if (res == GTK_RESPONSE_ACCEPT) {
      const char *dtext = gtk_entry_get_text(GTK_ENTRY(dentry));
      int depth = atoi(dtext);
      if (depth > 0) {
        gtk_tree_store_remove(G.annostore, &iter);
        start_ray_extend(anno, depth);
      }
    }
    gtk_widget_destroy(dialog);
  }
}

static void zoom_to_clicked_cb(GtkToolButton *toolbutton, gpointer user_data) {
  (void) toolbutton;
  (void) user_data;
  struct annotation *anno = get_selected_annotation();
  if (anno && anno->tag == annotation_nucleus) {
    GtkDialogFlags flags = GTK_DIALOG_MODAL | GTK_DIALOG_DESTROY_WITH_PARENT;
    GtkWidget *dialog = gtk_dialog_new_with_buttons("Zoom To", GTK_WINDOW(G.↙
```

```
        ↪ window), flags, "_Cancel", GTK_RESPONSE_CANCEL, "_Zoom To", ↙
        ↪ GTK_RESPONSE_ACCEPT, NULL);
    GtkWidget *box = gtk_dialog_get_content_area (GTK_DIALOG(dialog));
    GtkWidget *dentry = gtk_entry_new();
    gtk_entry_set_text(GTK_ENTRY(dentry), "0.75");
    gtk_container_add(GTK_CONTAINER(box), dentry);
    gtk_widget_show_all(dialog);
    gint res = gtk_dialog_run(GTK_DIALOG(dialog));
    if (res == GTK_RESPONSE_ACCEPT) {
      const char *dtext = gtk_entry_get_text(GTK_ENTRY(dentry));
      mpfr_t factor;
      mpfr_init2(factor, 53);
      mpfr_set_str(factor, dtext, 0, MPFR_RNDN);
      gtk_widget_destroy(dialog);
      struct view *v = view_new();
      mpfr_pow(v->radius, anno->u.nucleus.size, factor, MPFR_RNDN);
      mpfr_clear(factor);
      mpfr_set_prec(v->cx, 53 - mpfr_get_exp(v->radius));
      mpfr_set_prec(v->cy, 53 - mpfr_get_exp(v->radius));
      mpfr_set(v->cx, mpc_realref(anno->u.nucleus.xy), MPFR_RNDN);
      mpfr_set(v->cy, mpc_imagref(anno->u.nucleus.xy), MPFR_RNDN);
      start_render(v);
    } else {
      gtk_widget_destroy(dialog);
    }
  }
}

static void muunit_clicked_cb(GtkToolButton *toolbutton, gpointer user_data) {
  (void) toolbutton;
  (void) user_data;
  struct annotation *anno = get_selected_annotation();
  if (anno && anno->tag == annotation_nucleus) {
    task_enqueue_cb(task_new_muunit(G.annotation_queue, anno->u.nucleus.xy, anno↙
        ↪ ->u.nucleus.period));
  }
}

static void atom_clicked_cb(GtkToolButton *toolbutton, gpointer user_data) {
  (void) toolbutton;
  (void) user_data;
  struct annotation *anno = get_selected_annotation();
  if (anno && anno->tag == annotation_nucleus) {
    start_atom(anno->u.nucleus.xy, anno->u.nucleus.period);
  }
}

static void domain_clicked_cb(GtkToolButton *toolbutton, gpointer user_data) {
  (void) toolbutton;
  (void) user_data;
  struct annotation *anno = get_selected_annotation();
  if (anno && anno->tag == annotation_nucleus) {
    int loperiod = atoi(gtk_entry_get_text(GTK_ENTRY(G.filaments_period)));
    start_domain(anno->u.nucleus.xy, anno->u.nucleus.period, loperiod);
  }
}
```

```
2120  static void filaments_clicked_cb(GtkToolButton *toolbutton, gpointer user_data)
      {
        (void) toolbutton;
        (void) user_data;
        struct annotation *anno = get_selected_annotation();
2125    if (anno && anno->tag == annotation_nucleus) {
          int preperiod = atoi(gtk_entry_get_text(GTK_ENTRY(G.filaments_preperiod)));
          int outer_period = atoi(gtk_entry_get_text(GTK_ENTRY(G.filaments_period)));
          int depth = atoi(gtk_entry_get_text(GTK_ENTRY(G.filaments_depth)));
          task_enqueue_cb(task_new_filaments(G.annotation_queue, anno->u.nucleus.xy, ↙
              ↳ anno->u.nucleus.period, outer_period, preperiod, depth, G.line_type, G↙
              ↳ .colour_r, G.colour_g, G.colour_b));
2130    }
      }

      static gboolean nucleus_button_press_event_cb(GtkWidget *widget, GdkEventButton ↙
          ↳ *event, gpointer data) {
        (void) data;
2135    if (event->button == GDK_BUTTON_PRIMARY) {
          G.ball = TRUE;
          G.ball_x1 = event->x;
          G.ball_y1 = event->y;
          G.ball_x2 = event->x;
2140      G.ball_y2 = event->y;
          gtk_widget_queue_draw(widget);
        } else {
          G.ball = FALSE;
          gtk_widget_queue_draw(widget);
2145    }
        return TRUE;
      }

      static gboolean nucleus_button_release_event_cb(GtkWidget *widget, ↙
          ↳ GdkEventButton *event, gpointer data) {
2150    (void) data;
        if (event->button == GDK_BUTTON_PRIMARY) {
          if (G.ball) {
            if (G.view) {
              double x, y, r;
2155          ball_circle(&x, &y, &r);
              if (r > 0) {
                int i = x * G.width_print / G.width_screen, j = y * G.height_print / G↙
                    ↳ .height_screen;
                int dwell = G.maximum_iterations;
                if (0 <= i && i < G.width_print && 0 <= j && j < G.height_print)
2160            {
                  int k = j * G.width_print + i;
                  double dwelln = perturbator_get_dwell_n(G.image)[k];
                  double dwellf = perturbator_get_dwell_f(G.image)[k];
                  if (! (dwelln <= 0 || dwellf <= 0))
2165              {
                    dwell = floor(dwelln + dwellf);
                  }
                }
                x = (x - G.width_screen / 2.0) / (G.height_screen / 2.0);
2170            y = (G.height_screen / 2.0 - y) / (G.height_screen / 2.0);
                r = r / (G.height_screen / 2.0);
```

```
            mpfr_t clickx, clicky, clickr, logradius;
            mpfr_init2(clickx, 53);
            mpfr_init2(clicky, 53);
            mpfr_init2(clickr, 53);
            mpfr_init2(logradius, 53);
            mpfr_set_d(clickx, x, GMP_RNDN);
            mpfr_set_d(clicky, y, GMP_RNDN);
            mpfr_mul(clickx, clickx, G.view->radius, GMP_RNDN);
            mpfr_mul(clicky, clicky, G.view->radius, GMP_RNDN);
            mpfr_mul_d(clickr, G.view->radius, r, GMP_RNDN);
            mpfr_log2(logradius, clickr, GMP_RNDN);
            int prec = fmax(53, 16 - mpfr_get_d(logradius, GMP_RNDN));
            mpfr_prec_round(clickx, prec, GMP_RNDN);
            mpfr_prec_round(clicky, prec, GMP_RNDN);
            mpfr_add(clickx, G.view->cx, clickx, GMP_RNDN);
            mpfr_add(clicky, G.view->cy, clicky, GMP_RNDN);
            mpfr_clear(logradius);
            start_nucleus(clickx, clicky, clickr, dwell);
            mpfr_clear(clickx);
            mpfr_clear(clicky);
            mpfr_clear(clickr);
          }
        }
        G.ball = FALSE;
        gtk_widget_queue_draw(widget);
      }
    }
    return TRUE;
}

static gboolean nucleus_motion_notify_event_cb(GtkWidget *widget, GdkEventMotion↙
    ↳ *event, gpointer data) {
    (void) data;
    if (event->state & GDK_BUTTON1_MASK) {
      G.ball_x2 = event->x;
      G.ball_y2 = event->y;
      gtk_widget_queue_draw(widget);
    }
    return TRUE;
}

static void nucleus_toggled_cb(GtkWidget *nucleus, gpointer userdata) {
    (void) userdata;
    gboolean active = gtk_check_menu_item_get_active(GTK_CHECK_MENU_ITEM(nucleus))↙
        ↳ ;
    if (active) {
      G.nucleus_motion_notify_handler = g_signal_connect(G.da, "motion-notify-↙
          ↳ event", G_CALLBACK(nucleus_motion_notify_event_cb), NULL);
      G.nucleus_button_press_handler = g_signal_connect(G.da, "button-press-event↙
          ↳ ", G_CALLBACK(nucleus_button_press_event_cb), NULL);
      G.nucleus_button_release_handler = g_signal_connect(G.da, "button-release-↙
          ↳ event", G_CALLBACK(nucleus_button_release_event_cb), NULL);
    } else {
      if (G.nucleus_motion_notify_handler) {
        g_signal_handler_disconnect(G.da, G.nucleus_motion_notify_handler);
      }
      if (G.nucleus_button_press_handler) {
```

```
      g_signal_handler_disconnect(G.da, G.nucleus_button_press_handler);
    }
    if (G.nucleus_button_release_handler) {
      g_signal_handler_disconnect(G.da, G.nucleus_button_release_handler);
    }
    G.nucleus_motion_notify_handler = 0;
    G.nucleus_button_press_handler = 0;
    G.nucleus_button_release_handler = 0;
  }
}

static void start_misiurewicz(mpfr_t x, mpfr_t y, mpfr_t r, int maxpreperiod, ↙
    ↳ int maxperiod)
{
  mpc_t c;
  mpc_init2(c, mpfr_get_prec(x));
  mpc_set_fr_fr(c, x, y, MPC_RNDNN);
  struct task *t = task_new_misiurewicz(G.annotation_queue, c, r, maxpreperiod, ↙
      ↳ maxperiod);
  mpc_clear(c);
  task_enqueue_cb(t);
}

static gboolean misiurewicz_button_press_event_cb(GtkWidget *widget, ↙
    ↳ GdkEventButton *event, gpointer data) {
  (void) data;
  if (event->button == GDK_BUTTON_PRIMARY) {
    G.box = TRUE;
    G.box_x1 = event->x;
    G.box_y1 = event->y;
    G.box_x2 = event->x;
    G.box_y2 = event->y;
    gtk_widget_queue_draw(widget);
  } else {
    G.box = FALSE;
    gtk_widget_queue_draw(widget);
  }
  return TRUE;
}

static gboolean misiurewicz_button_release_event_cb(GtkWidget *widget, ↙
    ↳ GdkEventButton *event, gpointer data) {
  (void) data;
  if (event->button == GDK_BUTTON_PRIMARY) {
    if (G.box) {
      if (G.view) {
        double x, y, w, h;
        box_rectangle(&x, &y, &w, &h);
        if (w > 0 && h > 0) {
          int i = (x + w/2.0) * G.width_print / G.width_screen, j = (y + h/2.0) ↙
              ↳ * G.height_print / G.height_screen;
          int dwell = G.maximum_iterations;
          if (0 <= i && i < G.width_print && 0 <= j && j < G.height_print)
          {
            int k = j * G.width_print + i;
            double dwelln = perturbator_get_dwell_n(G.image)[k];
            double dwellf = perturbator_get_dwell_f(G.image)[k];
```

```
            if (! (dwelln <= 0 || dwellf <= 0))
            {
              dwell = floor(dwelln + dwellf);
            }
          }
          x = (x + w/2.0 - G.width_screen / 2.0) / (G.height_screen / 2.0);
          y = (G.height_screen / 2.0 - (y + h/2.0)) / (G.height_screen / 2.0);
          double r = (h/2.0) / (G.height_screen / 2.0);
          mpfr_t clickx, clicky, clickr, logradius;
          mpfr_init2(clickx, 53);
          mpfr_init2(clicky, 53);
          mpfr_init2(clickr, 53);
          mpfr_init2(logradius, 53);
          mpfr_set_d(clickx, x, GMP_RNDN);
          mpfr_set_d(clicky, y, GMP_RNDN);
          mpfr_mul(clickx, clickx, G.view->radius, GMP_RNDN);
          mpfr_mul(clicky, clicky, G.view->radius, GMP_RNDN);
          mpfr_mul_d(clickr, G.view->radius, r, GMP_RNDN);
          mpfr_log2(logradius, clickr, GMP_RNDN);
          int prec = fmax(53, 16 - mpfr_get_d(logradius, GMP_RNDN));
          mpfr_prec_round(clickx, prec, GMP_RNDN);
          mpfr_prec_round(clicky, prec, GMP_RNDN);
          mpfr_add(clickx, G.view->cx, clickx, GMP_RNDN);
          mpfr_add(clicky, G.view->cy, clicky, GMP_RNDN);
          mpfr_clear(logradius);
          const char *txt = gtk_entry_get_text(GTK_ENTRY(G.filaments_period));
          int maxperiod = atoi(txt);
          start_misiurewicz(clickx, clicky, clickr, dwell, maxperiod > 0 ? ↙
              ↳ maxperiod : dwell);
          mpfr_clear(clickx);
          mpfr_clear(clicky);
          mpfr_clear(clickr);
        }
      }
      G.box = FALSE;
      gtk_widget_queue_draw(widget);
    }
  }
  return TRUE;
}

static gboolean misiurewicz_motion_notify_event_cb(GtkWidget *widget, ↙
    ↳ GdkEventMotion *event, gpointer data) {
  (void) data;
  if (event->state & GDK_BUTTON1_MASK) {
    G.box_x2 = event->x;
    G.box_y2 = event->y;
    gtk_widget_queue_draw(widget);
  }
  return TRUE;
}

static void misiurewicz_toggled_cb(GtkWidget *misiurewicz, gpointer userdata) {
  (void) userdata;
  gboolean active = gtk_check_menu_item_get_active(GTK_CHECK_MENU_ITEM(↙
      ↳ misiurewicz));
  if (active) {
```

```
2330      G.box_aspect = 1.0;
          G.misiurewicz_motion_notify_handler = g_signal_connect(G.da, "motion-notify-↙
              ↳ event", G_CALLBACK(misiurewicz_motion_notify_event_cb), NULL);
          G.misiurewicz_button_press_handler = g_signal_connect(G.da, "button-press-↙
              ↳ event", G_CALLBACK(misiurewicz_button_press_event_cb), NULL);
          G.misiurewicz_button_release_handler = g_signal_connect(G.da, "button-↙
              ↳ release-event", G_CALLBACK(misiurewicz_button_release_event_cb), NULL)↙
              ↳ ;
        } else {
2335      if (G.misiurewicz_motion_notify_handler) {
            g_signal_handler_disconnect(G.da, G.misiurewicz_motion_notify_handler);
          }
          if (G.misiurewicz_button_press_handler) {
            g_signal_handler_disconnect(G.da, G.misiurewicz_button_press_handler);
2340      }
          if (G.misiurewicz_button_release_handler) {
            g_signal_handler_disconnect(G.da, G.misiurewicz_button_release_handler);
          }
          G.misiurewicz_motion_notify_handler = 0;
2345      G.misiurewicz_button_press_handler = 0;
          G.misiurewicz_button_release_handler = 0;
        }
      }

2350  // bond

      static gboolean bond_button_press_event_cb(GtkWidget *widget, GdkEventButton *↙
          ↳ event, gpointer data) {
        (void) widget;
        (void) data;
2355    if (event->button == GDK_BUTTON_PRIMARY) {
          int i = event->x * G.width_print / G.width_screen;
          int j = event->y * G.height_print / G.height_screen;
          if (0 <= i && i < G.width_print && 0 <= j && j < G.height_print) {
            int32_t dwell = perturbator_get_dwell_n(G.image)[j * G.width_print + i];
2360        if (dwell < 0) {
              int period = -dwell;
              mpc_t c;
              mpc_init2(c, 53);
              screen_to_param(event->x, event->y, mpc_realref(c), mpc_imagref(c));
2365          start_bond(c, period);
              mpc_clear(c);
            }
          }
        }
2370    return TRUE;
      }

      static void bond_toggled_cb(GtkWidget *bond, gpointer userdata) {
        (void) userdata;
2375    gboolean active = gtk_check_menu_item_get_active(GTK_CHECK_MENU_ITEM(bond));
        if (active) {
          G.bond_button_press_handler = g_signal_connect(G.da, "button-press-event", ↙
              ↳ G_CALLBACK(bond_button_press_event_cb), NULL);
        } else {
          if (G.bond_button_press_handler) {
2380        g_signal_handler_disconnect(G.da, G.bond_button_press_handler);
```

```
    }
    G.bond_button_press_handler = 0;
  }
}

static void update_fill_type(int t)
{
  G.fill_type = t;
  gtk_combo_box_set_active(GTK_COMBO_BOX(G.fillcombo), t);
}

static void update_line_type(int t)
{
  G.line_type = t;
  gtk_combo_box_set_active(GTK_COMBO_BOX(G.dashcombo), t);
}

static void update_colour(double r, double g, double b)
{
  G.colour_r = r;
  G.colour_g = g;
  G.colour_b = b;
  GdkRGBA c = { r, g, b, 1 };
  gtk_color_chooser_set_rgba(GTK_COLOR_CHOOSER(G.colourbutton), &c);
}

static void annotree_selection_changed_cb(GtkTreeSelection *selection, gpointer
    ↘ data) {
  (void) data;
  GtkTreeIter iter;
  GtkTreeModel *model;
  gpointer thing;
  if (gtk_tree_selection_get_selected(selection, &model, &iter)) {
    gtk_tree_model_get(model, &iter, 1, &thing, -1);
    struct annotation *ls;
    for (ls = G.anno; ls; ls = ls->next) {
      ls->selected = 0;
    }
    ls = (struct annotation *) thing;
    if (ls) {
      ls->selected = 1;
      switch(ls->tag)
      {
        case annotation_ray_out:
          update_line_type(ls->u.ray_out.line_type);
          update_colour(ls->u.ray_out.stroke_r, ls->u.ray_out.stroke_g, ls->u.↙
              ↘ ray_out.stroke_b);
          break;
        case annotation_ray_in:
          update_line_type(ls->u.ray_in.line_type);
          update_colour(ls->u.ray_in.stroke_r, ls->u.ray_in.stroke_g, ls->u.↙
              ↘ ray_in.stroke_b);
          break;
        case annotation_wake:
          update_fill_type(ls->u.wake.fill_type);
          update_colour(ls->u.wake.fill_r, ls->u.wake.fill_g, ls->u.wake.fill_b)↙
              ↘ ;
```

```
            break;
2435        case annotation_atom:
            update_fill_type(ls->u.atom.fill_type);
            update_colour(ls->u.atom.fill_r, ls->u.atom.fill_g, ls->u.atom.fill_b)↙
                ↳ ;
            break;
          case annotation_domain:
2440        update_line_type(ls->u.domain.line_type);
            update_colour(ls->u.domain.line_r, ls->u.domain.line_g, ls->u.domain.↙
                ↳ line_b);
            break;
          case annotation_text:
          case annotation_nucleus:
2445      case annotation_misiurewicz:
            break;
        }
      }
    }
2450  gtk_widget_queue_draw(G.da);
}

static void delete_clicked_cb(GtkToolButton *toolbutton, gpointer user_data) {
  (void) toolbutton;
2455  (void) user_data;
  GtkTreeSelection *select = gtk_tree_view_get_selection(GTK_TREE_VIEW(G.↙
      ↳ annotree));
  GtkTreeIter iter;
  GtkTreeModel *model;
  gpointer thing;
2460  if (gtk_tree_selection_get_selected(select, &model, &iter)) {
    gtk_tree_model_get(model, &iter, 1, &thing, -1);
    if (thing) {
      gtk_tree_store_remove(GTK_TREE_STORE(model), &iter);
      delete_annotation((struct annotation *) thing);
2465  }
  }
}

static int serialize(const char *filename) {
2470  FILE *out = fopen(filename, "wb");
  if (out) {
    int last_line_type = -1;
    int last_fill_type = -1;
    double last_r = -1;
2475  double last_g = -1;
    double last_b = -1;
    fprintf(out, "m-perturbator-gtk %s\n", VERSION);
    fprintf(out, "size %d %d\n", G.width_print, G.height_print);
    mpfr_fprintf
2480  ( out
    , "view %Pd %Re %Re %Re\n"
    , mpfr_get_prec(G.view->cx)
    , G.view->cx
    , G.view->cy
2485  , G.view->radius
    );
    fprintf(out, "dark %d\n", G.dark);
```

```
    fprintf(out, "colour %d\n", G.colour);
    for (struct annotation *anno = G.anno; anno; anno = anno->next) {
      switch (anno->tag) {
        case annotation_ray_out: {
          if (last_r != anno->u.ray_out.stroke_r ||
              last_g != anno->u.ray_out.stroke_g ||
              last_b != anno->u.ray_out.stroke_b)
          {
            last_r = anno->u.ray_out.stroke_r;
            last_g = anno->u.ray_out.stroke_g;
            last_b = anno->u.ray_out.stroke_b;
            fprintf(out, "rgb %f %f %f\n", last_r, last_g, last_b);
          }
          if (last_line_type != anno->u.ray_out.line_type)
          {
            last_line_type = anno->u.ray_out.line_type;
            fprintf(out, "line_type %d\n", last_line_type);
          }
          struct point *p = anno->u.ray_out.line_end;
          mpfr_fprintf
            ( out
            , "ray_out %Pd %Re %Re\n"
            , mpfr_get_prec(mpc_realref(p->xy))
            , mpc_realref(p->xy)
            , mpc_imagref(p->xy)
            );
          break;
        }
        case annotation_ray_in: {
          if (last_r != anno->u.ray_in.stroke_r ||
              last_g != anno->u.ray_in.stroke_g ||
              last_b != anno->u.ray_in.stroke_b)
          {
            last_r = anno->u.ray_in.stroke_r;
            last_g = anno->u.ray_in.stroke_g;
            last_b = anno->u.ray_in.stroke_b;
            fprintf(out, "rgb %f %f %f\n", last_r, last_g, last_b);
          }
          if (last_line_type != anno->u.ray_in.line_type)
          {
            last_line_type = anno->u.ray_in.line_type;
            fprintf(out, "line_type %d\n", last_line_type);
          }
          char *s = m_binangle_to_new_string(&anno->u.ray_in.angle);
          fprintf(out, "ray_in %d %s\n", anno->u.ray_in.depth, s);
          free(s);
          break;
        }
        case annotation_text: {
          mpfr_fprintf
            ( out
            , "text %Pd %Re %Re %s\n"
            , mpfr_get_prec(mpc_realref(anno->u.text.xy))
            , mpc_realref(anno->u.text.xy)
            , mpc_imagref(anno->u.text.xy)
            , anno->label
            );
```

```
2545            break;
              }
              case annotation_nucleus: {
                mpfr_fprintf
                  ( out
2550              , "nucleus %Pd %Re %Re %d %Re %Re %s\n"
                  , mpfr_get_prec(mpc_realref(anno->u.nucleus.xy))
                  , mpc_realref(anno->u.nucleus.xy)
                  , mpc_imagref(anno->u.nucleus.xy)
                  , anno->u.nucleus.period
2555              , anno->u.nucleus.domain_size
                  , anno->u.nucleus.size
                  , anno->label
                  );
                break;
2560            }
              case annotation_misiurewicz: {
                mpfr_fprintf
                  ( out
                  , "misiurewicz %Pd %Re %Re %d %d %Re %s\n"
2565              , mpfr_get_prec(mpc_realref(anno->u.misiurewicz.xy))
                  , mpc_realref(anno->u.misiurewicz.xy)
                  , mpc_imagref(anno->u.misiurewicz.xy)
                  , anno->u.misiurewicz.preperiod
                  , anno->u.misiurewicz.period
2570              , anno->u.misiurewicz.domain_size
                  , anno->label
                  );
                break;
              }
2575          case annotation_wake: {
                if (last_r != anno->u.wake.fill_r ||
                    last_g != anno->u.wake.fill_g ||
                    last_b != anno->u.wake.fill_b)
                {
2580              last_r = anno->u.wake.fill_r;
                  last_g = anno->u.wake.fill_g;
                  last_b = anno->u.wake.fill_b;
                  fprintf(out, "rgb %f %f %f\n", last_r, last_g, last_b);
                }
2585            if (last_fill_type != anno->u.wake.fill_type)
                {
                  last_fill_type = anno->u.wake.fill_type;
                  fprintf(out, "fill_type %d\n", last_fill_type);
                }
2590            fprintf(out, "wake %d %s %s\n", anno->u.wake.ray_depth, anno->u.wake.↙
                    ↳ ray_lo, anno->u.wake.ray_hi);
                break;
              }
              case annotation_atom: {
                if (last_r != anno->u.atom.fill_r ||
2595                last_g != anno->u.atom.fill_g ||
                    last_b != anno->u.atom.fill_b)
                {
                  last_r = anno->u.atom.fill_r;
                  last_g = anno->u.atom.fill_g;
2600              last_b = anno->u.atom.fill_b;
```

```
            fprintf(out, "rgb %f %f %f\n", last_r, last_g, last_b);
          }
          if (last_fill_type != anno->u.atom.fill_type)
          {
2605        last_fill_type = anno->u.atom.fill_type;
            fprintf(out, "fill_type %d\n", last_fill_type);
          }
          mpfr_fprintf
            ( out
2610        , "atom %Pd %Re %Re %d\n"
            , mpfr_get_prec(mpc_realref(anno->u.atom.nucleus))
            , mpc_realref(anno->u.atom.nucleus)
            , mpc_imagref(anno->u.atom.nucleus)
            , anno->u.atom.period
2615        );
          break;
        }
        case annotation_domain: {
          if (last_r != anno->u.domain.line_r ||
2620          last_g != anno->u.domain.line_g ||
              last_b != anno->u.domain.line_b)
          {
            last_r = anno->u.domain.line_r;
            last_g = anno->u.domain.line_g;
2625        last_b = anno->u.domain.line_b;
            fprintf(out, "rgb %f %f %f\n", last_r, last_g, last_b);
          }
          if (last_line_type != anno->u.domain.line_type)
          {
2630        last_line_type = anno->u.domain.line_type;
            fprintf(out, "line_type %d\n", last_line_type);
          }
          mpfr_fprintf
            ( out
2635        , "domain %Pd %Re %Re %d %d\n"
            , mpfr_get_prec(mpc_realref(anno->u.domain.nucleus))
            , mpc_realref(anno->u.domain.nucleus)
            , mpc_imagref(anno->u.domain.nucleus)
            , anno->u.domain.period
2640        , anno->u.domain.loperiod
            );
          break;
        }
      }
2645  }
    fclose(out);
    return 0;
  } else {
    return 1;
2650 }
}

static int deserialize(const char *filename) {
  FILE *in = fopen(filename, "rb");
2655 if (in) {
    while (G.anno) {
      GtkTreeIter iter;
```

```
      gpointer thing;
      gtk_tree_model_get_iter_first(GTK_TREE_MODEL(G.annostore), &iter);
2660  if (gtk_tree_model_iter_next(GTK_TREE_MODEL(G.annostore), &iter)) {
        gtk_tree_model_get(GTK_TREE_MODEL(G.annostore), &iter, 1, &thing, -1);
        if (thing) {
          delete_annotation((struct annotation *) thing);
          gtk_tree_store_remove(G.annostore, &iter);
2665    }
      }
    }
    char *line = 0;
    size_t linelen = 0;
2670 ssize_t readlen = 0;
    while (-1 != (readlen = getline(&line, &linelen, in))) {
      if (readlen && line[readlen-1] == '\n') {
        line[readlen - 1] = 0;
      }
2675  if (0 == strncmp(line, "m-perturbator-gtk ", 18)) {
        if (0 < strcmp(line + 18, VERSION)) {
          printf("version mismatch: file %s, program %s\n", line + 18, VERSION);
        }
      } else if (0 == strncmp(line, "size ", 5)) {
2680    int w = 0, h = 0;
        sscanf(line + 5, "%d %d\n", &w, &h);
        // resize_image(w, h); // FIXME TODO make this work...
      } else if (0 == strncmp(line, "view ", 5)) {
        int p = 53;
2685    int set_result;
        char *xs = malloc(readlen);
        char *ys = malloc(readlen);
        char *rs = malloc(readlen);
        sscanf(line + 5, "%d %s %s %s", &p, xs, ys, rs);
2690    struct view *v = view_new();
        mpfr_set_prec(v->cx, p);
        mpfr_set_prec(v->cy, p);
        set_result = mpfr_set_str(v->cx, xs, 0, GMP_RNDN);
        set_result = set_result + mpfr_set_str(v->cy, ys, 0, GMP_RNDN);
2695    set_result = set_result + mpfr_set_str(v->radius, rs, 0, GMP_RNDN);
        free(xs);
        free(ys);
        free(rs);
        if (set_result < 0) { printf("view line not valid, check chars\n"); ↙
            ↳ return 1; }
2700    start_render(v);
      } else if (0 == strncmp(line, "dark ", 5)) {
        sscanf(line + 5, "%d", &G.dark);
        set_dark_theme(G.dark);
      } else if (0 == strncmp(line, "colour ", 7)) {
2705    sscanf(line + 7, "%d", &G.colour);
        set_colour_theme(G.colour);
      } else if (0 == strncmp(line, "rgb ", 4)) {
        sscanf(line + 4, "%lf %lf %lf", &G.colour_r, &G.colour_g, &G.colour_b);
        update_colour(G.colour_r, G.colour_g, G.colour_b);
2710  } else if (0 == strncmp(line, "line_type ", 10)) {
        sscanf(line + 10, "%d", &G.line_type);
        update_line_type(G.line_type);
      } else if (0 == strncmp(line, "fill_type ", 10)) {
```

```
      sscanf(line + 10, "%d", &G.fill_type);
2715  update_fill_type(G.fill_type);
    } else if (0 == strncmp(line, "ray_out ", 8)) {
      int p = 53;
      int set_result;
      char *xs = malloc(readlen);
2720  char *ys = malloc(readlen);
      sscanf(line + 8, "%d %s %s", &p, xs, ys);
      mpfr_t cx, cy;
      mpfr_init2(cx, p);
      mpfr_init2(cy, p);
2725  set_result = mpfr_set_str(cx, xs, 0, GMP_RNDN);
      set_result = set_result + mpfr_set_str(cy, ys, 0, GMP_RNDN);
      free(xs);
      free(ys);
      double x = 0, y = 0;
2730  param_to_screen(&x, &y, cx, cy);
      mpfr_clear(cx);
      mpfr_clear(cy);
      if (set_result < 0) { printf("ray_out line not valid, check chars\n"); ↙
          ↳ return 1; }
      start_ray_out(x, y);
2735  } else if (0 == strncmp(line, "ray_in ", 7)) {
      int depth = 1000;
      char *as = malloc(readlen);
      sscanf(line + 7, "%d %s", &depth, as);
      struct m_binangle *angle = malloc(sizeof(*angle));
2740  m_binangle_init(angle);
      m_binangle_from_string(angle, as);
      m_binangle_canonicalize(angle);
      start_ray_in(angle, depth);
      free(as);
2745  } else if (0 == strncmp(line, "text ", 5)) {
      int p = 53;
      int set_result;
      char *xs = malloc(readlen);
      char *ys = malloc(readlen);
2750  char *ss = malloc(readlen);
      sscanf(line + 5, "%d %s %s %s", &p, xs, ys, ss);
      struct annotation *anno = calloc(1, sizeof(struct annotation));
      anno->tag = annotation_text;
      anno->label = ss;
2755  mpfr_init2(mpc_realref(anno->u.text.xy), p);
      mpfr_init2(mpc_imagref(anno->u.text.xy), p);
      set_result = mpfr_set_str(mpc_realref(anno->u.text.xy), xs, 0, GMP_RNDN)↙
          ↳ ;
      set_result = set_result + mpfr_set_str(mpc_imagref(anno->u.text.xy), ys,↙
          ↳  0, GMP_RNDN);
      free(xs);
2760  free(ys);
      if (set_result < 0) {
        free(ss);
        mpfr_clear(mpc_realref(anno->u.text.xy));
        mpfr_clear(mpc_imagref(anno->u.text.xy));
2765    printf("text line not valid, check chars \n");
        return 1;
      }
```

```
      add_annotation(anno);
    } else if (0 == strncmp(line, "nucleus ", 8)) {
      int p = 53;
      int set_result;
      char *xs = malloc(readlen);
      char *ys = malloc(readlen);
      int period = 0;
      char *dzs = malloc(readlen);
      char *szs = malloc(readlen);
      char *ss = malloc(readlen);
      sscanf(line + 8, "%d %s %s %d %s %s %s", &p, xs, ys, &period, dzs, szs, ↙
          ↳ ss);
      struct annotation *anno = calloc(1, sizeof(struct annotation));
      anno->tag = annotation_nucleus;
      anno->label = ss;
      anno->u.nucleus.period = period;
      mpfr_init2(mpc_realref(anno->u.nucleus.xy), p);
      mpfr_init2(mpc_imagref(anno->u.nucleus.xy), p);
      mpfr_init2(anno->u.nucleus.domain_size, 53);
      mpfr_init2(anno->u.nucleus.size, 53);
      set_result  = mpfr_set_str(mpc_realref(anno->u.nucleus.xy), xs, 0, ↙
          ↳ GMP_RNDN);
      set_result += mpfr_set_str(mpc_imagref(anno->u.nucleus.xy), ys, 0, ↙
          ↳ GMP_RNDN);
      set_result += mpfr_set_str(anno->u.nucleus.domain_size, dzs, 0, GMP_RNDN↙
          ↳ );
      set_result += mpfr_set_str(anno->u.nucleus.size, szs, 0, GMP_RNDN);
      free(xs);
      free(ys);
      free(dzs);
      free(szs);
      if (set_result < 0 || period <= 0) {
        free(ss);
        mpc_clear(anno->u.nucleus.xy);
        mpfr_clear(anno->u.nucleus.domain_size);
        mpfr_clear(anno->u.nucleus.size);
        free(anno);
        printf("nucleus line not valid, check chars \n");
        return 1;
      }
      add_annotation(anno);
    } else if (0 == strncmp(line, "misiurewicz ", 12)) {
      int p = 53;
      int set_result;
      char *xs = malloc(readlen);
      char *ys = malloc(readlen);
      int preperiod = -1;
      int period = 0;
      char *dzs = malloc(readlen);
      char *ss = malloc(readlen);
      sscanf(line + 12, "%d %s %s %d %d %s %s", &p, xs, ys, &preperiod, &↙
          ↳ period, dzs, ss);
      struct annotation *anno = calloc(1, sizeof(struct annotation));
      anno->tag = annotation_misiurewicz;
      anno->label = ss;
      anno->u.misiurewicz.preperiod = preperiod;
      anno->u.misiurewicz.period = period;
```

```
2820            mpfr_init2(mpc_realref(anno->u.misiurewicz.xy), p);
                mpfr_init2(mpc_imagref(anno->u.misiurewicz.xy), p);
                mpfr_init2(anno->u.misiurewicz.domain_size, 53);
                set_result  = mpfr_set_str(mpc_realref(anno->u.misiurewicz.xy), xs, 0, ↙
                    ↳ GMP_RNDN);
                set_result += mpfr_set_str(mpc_imagref(anno->u.misiurewicz.xy), ys, 0, ↙
                    ↳ GMP_RNDN);
2825            set_result += mpfr_set_str(anno->u.misiurewicz.domain_size, dzs, 0, ↙
                    ↳ GMP_RNDN);
                free(xs);
                free(ys);
                free(dzs);
                if (set_result < 0 || preperiod <= 0 || period <= 0) {
2830              free(ss);
                  mpc_clear(anno->u.misiurewicz.xy);
                  mpfr_clear(anno->u.misiurewicz.domain_size);
                  free(anno);
                  printf("misiurewicz line not valid, check chars \n");
2835              return 1;
                }
                add_annotation(anno);
              } else if (0 == strncmp(line, "wake ", 5)) {
                int depth = 0;
2840            char *lo = malloc(readlen);
                char *hi = malloc(readlen);
                sscanf(line + 5, "%d %s %s", &depth, lo, hi);
                start_wake2(depth, lo, hi);
                free(lo);
2845            free(hi);
              } else if (0 == strncmp(line, "atom ", 5)) {
                int p = 53;
                int set_result;
                char *xs = malloc(readlen);
2850            char *ys = malloc(readlen);
                int period = 0;
                sscanf(line + 5, "%d %s %s %d", &p, xs, ys, &period);
                mpc_t nucleus;
                mpc_init2(nucleus, p);
2855            set_result  = mpfr_set_str(mpc_realref(nucleus), xs, 0, MPFR_RNDN);
                set_result += mpfr_set_str(mpc_imagref(nucleus), ys, 0, MPFR_RNDN);
                free(xs);
                free(ys);
                start_atom(nucleus, period);
2860            mpc_clear(nucleus);
              } else if (0 == strncmp(line, "domain ", 7)) {
                int p = 53;
                int set_result;
                char *xs = malloc(readlen);
2865            char *ys = malloc(readlen);
                int period = 0;
                int loperiod = 0;
                if (5 != sscanf(line + 7, "%d %s %s %d %d", &p, xs, ys, &period, &↙
                    ↳ loperiod))
                {
2870              loperiod = 0;
                  sscanf(line + 7, "%d %s %s %d", &p, xs, ys, &period);
                }
```

```
          mpc_t nucleus;
          mpc_init2(nucleus, p);
          set_result  = mpfr_set_str(mpc_realref(nucleus), xs, 0, MPFR_RNDN);
          set_result += mpfr_set_str(mpc_imagref(nucleus), ys, 0, MPFR_RNDN);
          free(xs);
          free(ys);
          start_domain(nucleus, period, loperiod);
          mpc_clear(nucleus);
        }
      }
      free(line);
      fclose(in);
      return 0;
    } else {
      return 1;
    }
}

static gint sortable_annotation_compare(GtkTreeModel *model, GtkTreeIter *a, ↙
    ↳ GtkTreeIter *b, gpointer user_data)
{
  (void) user_data;
  gpointer thinga = 0, thingb = 0;
  gtk_tree_model_get(model, a, 1, &thinga, -1);
  gtk_tree_model_get(model, b, 1, &thingb, -1);
  struct annotation *la = (struct annotation *) thinga;
  struct annotation *lb = (struct annotation *) thingb;
  if (la && lb) {
    if (la->tag < lb->tag) return -1;
    if (la->tag > lb->tag) return 1;
    switch (la->tag)
    {
      case annotation_text:
        return strcmp(la->label, lb->label);
      case annotation_ray_out:
        return strcmp(la->label, lb->label);
      case annotation_ray_in:
      {
        mpq_t a, b;
        mpq_init(a);
        mpq_init(b);
        m_binangle_to_rational(a, &la->u.ray_in.angle);
        m_binangle_to_rational(b, &lb->u.ray_in.angle);
        int r = mpq_cmp(a, b);
        mpq_clear(a);
        mpq_clear(b);
        return (r > 0) - (0 > r);
      }
      case annotation_nucleus:
      {
        int dp = la->u.nucleus.period - lb->u.nucleus.period;
        return (dp > 0) - (0 > dp);
      }
      case annotation_misiurewicz:
      {
        int pp = la->u.misiurewicz.preperiod - lb->u.misiurewicz.preperiod;
        int p = la->u.misiurewicz.period - lb->u.misiurewicz.period;
```

```
          pp = (pp > 0) - (0 > pp);
          p = (p > 0) - (0 > p);
          return pp ? pp : p;
        }
        case annotation_wake:
          return strcmp(la->label, lb->label);
        case annotation_atom:
          return strcmp(la->label, lb->label);
        case annotation_domain:
        {
          int pp = la->u.domain.loperiod - lb->u.domain.loperiod;
          int p = la->u.domain.period - lb->u.domain.period;
          pp = (pp > 0) - (0 > pp);
          p = (p > 0) - (0 > p);
          return pp ? pp : p;
        }
      }
    }
    if (la) return -1;
    if (lb) return 1;
    return 0;
}

static void task_func(gpointer data, gpointer userdata);
static gpointer annotation_thread(gpointer userdata);

static void dashing_selection_changed_cb(GtkWidget *widget, gpointer userdata)
{
  (void) userdata;
  G.line_type = gtk_combo_box_get_active(GTK_COMBO_BOX(widget));
  struct annotation *anno = get_selected_annotation();
  if (anno)
  {
    if (annotation_set_line_type(anno, G.line_type))
    {
      gtk_widget_queue_draw(G.da);
    }
  }
}

static void fill_selection_changed_cb(GtkWidget *widget, gpointer userdata)
{
  (void) userdata;
  G.fill_type = gtk_combo_box_get_active(GTK_COMBO_BOX(widget));
  struct annotation *anno = get_selected_annotation();
  if (anno)
  {
    if (annotation_set_fill_type(anno, G.fill_type))
    {
      gtk_widget_queue_draw(G.da);
    }
  }
}

static void set_dark_theme(bool use_dark_theme)
{
  G.dark = use_dark_theme;
```

```
  int active = gtk_combo_box_get_active(GTK_COMBO_BOX(G.dashcombo));
  gtk_combo_box_set_model(GTK_COMBO_BOX(G.dashcombo), GTK_TREE_MODEL(G.dashstore
      ↳ [G.dark]));
  gtk_combo_box_set_active(GTK_COMBO_BOX(G.dashcombo), active);
  active = gtk_combo_box_get_active(GTK_COMBO_BOX(G.fillcombo));
  gtk_combo_box_set_model(GTK_COMBO_BOX(G.fillcombo), GTK_TREE_MODEL(G.fillstore
      ↳ [G.dark]));
  gtk_combo_box_set_active(GTK_COMBO_BOX(G.fillcombo), active);
#ifdef GDK_WINDOWING_X11
  GdkWindow *window = gtk_widget_get_parent_window(G.dashcombo);
  if (! window)
  {
    fprintf(stderr, "bad window for widget %p\n", (void *) G.dashcombo);
    return;
  }
  GdkDisplay *display = gdk_display_get_default();
  if (GDK_IS_X11_DISPLAY(display))
  {
    Display* xdisplay = GDK_DISPLAY_XDISPLAY(display);
    XChangeProperty
      ( xdisplay
      , GDK_WINDOW_XID(window)
      , XInternAtom(xdisplay, "_GTK_THEME_VARIANT", False)
      , XInternAtom(xdisplay, "UTF8_STRING", False)
      , 8
      , PropModeReplace
      , (unsigned char *) (use_dark_theme ? "dark" : "light")
      , use_dark_theme ? 4 : 5
      );
  }
#endif
  docolour(G.image);
}

static void set_colour_theme(enum colour_theme_t use_colour_theme)
{
  G.colour = use_colour_theme;
}

static void dark_toggled_cb(GtkWidget *widget, gpointer userdata)
{
  (void) userdata;
  gboolean active = gtk_check_menu_item_get_active(GTK_CHECK_MENU_ITEM(widget));
  if (active)
  {
    set_dark_theme(1);
    gtk_widget_queue_draw(G.da);
  }
}

static void light_toggled_cb(GtkWidget *widget, gpointer userdata)
{
  (void) userdata;
  gboolean active = gtk_check_menu_item_get_active(GTK_CHECK_MENU_ITEM(widget));
  if (active)
  {
    set_dark_theme(0);
```

```
    gtk_widget_queue_draw(G.da);
  }
}

static void monochrome_toggled_cb(GtkWidget *widget, gpointer userdata)
{
  (void) userdata;
  gboolean active = gtk_check_menu_item_get_active(GTK_CHECK_MENU_ITEM(widget));
  if (active)
  {
    set_colour_theme(colour_theme_monochrome);
    gtk_widget_queue_draw(G.da);
  }
}

static void low_colour_toggled_cb(GtkWidget *widget, gpointer userdata)
{
  (void) userdata;
  gboolean active = gtk_check_menu_item_get_active(GTK_CHECK_MENU_ITEM(widget));
  if (active)
  {
    set_colour_theme(colour_theme_low_colour);
    gtk_widget_queue_draw(G.da);
  }
}

static void full_colour_toggled_cb(GtkWidget *widget, gpointer userdata)
{
  (void) userdata;
  gboolean active = gtk_check_menu_item_get_active(GTK_CHECK_MENU_ITEM(widget));
  if (active)
  {
    set_colour_theme(colour_theme_full_colour);
    gtk_widget_queue_draw(G.da);
  }
}

static void colour_activated_cb(GtkWidget *widget, gpointer userdata)
{
  (void) userdata;
  GdkRGBA rgba = {0,0,0,1};
  gtk_color_chooser_get_rgba(GTK_COLOR_CHOOSER(widget), &rgba);
  G.colour_r = rgba.red;
  G.colour_g = rgba.green;
  G.colour_b = rgba.blue;
  struct annotation *anno = get_selected_annotation();
  if (anno)
  {
    if (annotation_set_colour(anno, G.colour_r, G.colour_g, G.colour_b))
    {
      gtk_widget_queue_draw(G.da);
    }
  }
}

extern int main(int argc, char **argv) {
  m_symbolics_init();
```

```
  memset(&G, 0, sizeof(G));
  gtk_disable_setlocale(); // FIXME TODO find a better way for safe (de)↙
      ↳ serialisation
3100  gtk_init(&argc, &argv);
  GdkRectangle workarea = {0};
  gdk_monitor_get_workarea(
    gdk_display_get_primary_monitor(
      gdk_display_get_default()
3105  ), &workarea);
  G.dpi_screen = workarea.width * 2 / 3 * 25.4 / (297 - 40);
  G.dpi_print = 0;
  for (int i = 1; i < argc - 1; ++i)
  {
3110  if (! strcmp(argv[i], "--dpi"))
    {
      G.dpi_print = atof(argv[i + 1]);
    }
  }
3115  if (! (G.dpi_print > 0))
  {
    G.dpi_print = G.dpi_screen;
  }
  G.width_screen  = (297 - 40) / 25.4 * G.dpi_screen;
3120  G.height_screen = (210 - 40) / 25.4 * G.dpi_screen;
  G.width_print   = (297 - 40) / 25.4 * G.dpi_print;
  G.height_print  = (210 - 40) / 25.4 * G.dpi_print;
  G.threads = g_get_num_processors();
  G.dark = false;
3125
  G.wakes_head.u.wake.next_wake = &G.wakes_tail;
  G.wakes_tail.u.wake.pred_wake = &G.wakes_head;
  G.maximum_iterations = 1 << 16;
  G.chunk = 256;
3130  G.escape_radius = 512;
  G.glitch_threshold = 1e-3;
  mpfr_init2(G.escape_radius2, 53);
  mpfr_set_d(G.escape_radius2, G.escape_radius * G.escape_radius, GMP_RNDN);
  G.view = view_new();
3135  mpfr_set_d(G.view->cx, -0.75, GMP_RNDN);
  mpfr_set_d(G.view->cy, 0.0, GMP_RNDN);
  mpfr_set_d(G.view->radius, 1.5, GMP_RNDN);
  G.image = perturbator_new(G.threads, G.width_print, G.height_print, G.↙
      ↳ maximum_iterations, G.chunk, G.escape_radius, G.glitch_threshold);

3140  G.window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  char title[1024];
  snprintf(title, 1024, "Mandelbrot Perturbator GTK (version %s)", VERSION);
  gtk_window_set_title(GTK_WINDOW(G.window), title);
  gtk_window_set_resizable(GTK_WINDOW(G.window), TRUE);
3145  g_signal_connect(G.window, "destroy", G_CALLBACK(close_window), NULL);

  GtkWidget *h0 = gtk_box_new(GTK_ORIENTATION_HORIZONTAL, 0);
  GtkWidget *v1 = gtk_box_new(GTK_ORIENTATION_VERTICAL, 0);
  GtkWidget *h2 = gtk_box_new(GTK_ORIENTATION_HORIZONTAL, 0);
3150  GtkWidget *v3 = gtk_box_new(GTK_ORIENTATION_VERTICAL, 0);
  GtkWidget *v4 = gtk_box_new(GTK_ORIENTATION_VERTICAL, 0);
```

```
    G.da = gtk_drawing_area_new();
    gtk_widget_set_size_request(G.da, G.width_screen, G.height_screen);
    g_signal_connect(G.da, "draw", G_CALLBACK(draw_cb), NULL);
    g_signal_connect(G.da,"configure-event", G_CALLBACK(configure_event_cb), NULL)↙
        ↳ ;
    gtk_widget_set_events(G.da, gtk_widget_get_events(G.da) | ↙
        ↳ GDK_BUTTON_PRESS_MASK | GDK_BUTTON_RELEASE_MASK | ↙
        ↳ GDK_POINTER_MOTION_MASK);

    GtkWidget *menu = gtk_menu_bar_new();

    GtkWidget *file = gtk_menu_item_new_with_label("File");
    GtkWidget *filem = gtk_menu_new();

    GtkWidget *open = gtk_menu_item_new_with_label("Open");
    g_signal_connect(open, "activate", G_CALLBACK(load_clicked_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(filem), open);

    GtkWidget *save = gtk_menu_item_new_with_label("Save");
    g_signal_connect(save, "activate", G_CALLBACK(save_clicked_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(filem), save);

    GtkWidget *save_image = gtk_menu_item_new_with_label("Save PNG");
    g_signal_connect(save_image, "activate", G_CALLBACK(save_image_clicked_cb), ↙
        ↳ NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(filem), save_image);

    GtkWidget *save_pdf = gtk_menu_item_new_with_label("Save PDF");
    g_signal_connect(save_pdf, "activate", G_CALLBACK(save_pdf_clicked_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(filem), save_pdf);

    GSList *dark_group = NULL;

    GtkWidget *dark = gtk_radio_menu_item_new_with_label(dark_group, "Dark");
    dark_group = gtk_radio_menu_item_get_group(GTK_RADIO_MENU_ITEM(dark));
    gtk_check_menu_item_set_active(GTK_CHECK_MENU_ITEM(dark), G.dark);
    g_signal_connect(dark, "toggled", G_CALLBACK(dark_toggled_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(filem), dark);

    GtkWidget *light = gtk_radio_menu_item_new_with_label(dark_group, "Light");
    dark_group = gtk_radio_menu_item_get_group(GTK_RADIO_MENU_ITEM(light));
    gtk_check_menu_item_set_active(GTK_CHECK_MENU_ITEM(light), ! G.dark);
    g_signal_connect(light, "toggled", G_CALLBACK(light_toggled_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(filem), light);

    GSList *colour_group = NULL;

    GtkWidget *monochrome = gtk_radio_menu_item_new_with_label(colour_group, "↙
        ↳ Monochrome");
    colour_group = gtk_radio_menu_item_get_group(GTK_RADIO_MENU_ITEM(monochrome));
    gtk_check_menu_item_set_active(GTK_CHECK_MENU_ITEM(monochrome), G.colour == ↙
        ↳ colour_theme_monochrome);
    g_signal_connect(monochrome, "toggled", G_CALLBACK(monochrome_toggled_cb), ↙
        ↳ NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(filem), monochrome);

    GtkWidget *low_colour = gtk_radio_menu_item_new_with_label(colour_group, "Low ↙
```

```
        ↳ Colour");
    colour_group = gtk_radio_menu_item_get_group(GTK_RADIO_MENU_ITEM(low_colour));
    gtk_check_menu_item_set_active(GTK_CHECK_MENU_ITEM(low_colour), G.colour == ↙
        ↳ colour_theme_low_colour);
3205 g_signal_connect(low_colour, "toggled", G_CALLBACK(low_colour_toggled_cb), ↙
        ↳ NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(filem), low_colour);

    GtkWidget *full_colour = gtk_radio_menu_item_new_with_label(colour_group, "↙
        ↳ Full Colour");
    colour_group = gtk_radio_menu_item_get_group(GTK_RADIO_MENU_ITEM(full_colour))↙
        ↳ ;
3210 gtk_check_menu_item_set_active(GTK_CHECK_MENU_ITEM(full_colour), G.colour == ↙
        ↳ colour_theme_full_colour);
    g_signal_connect(full_colour, "toggled", G_CALLBACK(full_colour_toggled_cb), ↙
        ↳ NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(filem), full_colour);

    gtk_menu_item_set_submenu(GTK_MENU_ITEM(file), filem);
3215 gtk_menu_shell_append(GTK_MENU_SHELL(menu), file);

    GSList *group = NULL;

    GtkWidget *explore = gtk_menu_item_new_with_label("Explore");
3220 GtkWidget *explorem = gtk_menu_new();

    GtkWidget *home = gtk_menu_item_new_with_label("Home");
    g_signal_connect(home, "activate", G_CALLBACK(home_clicked_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(explorem), home);
3225
    GtkWidget *zoom_out = gtk_menu_item_new_with_label("Zoom Out");
    g_signal_connect(zoom_out, "activate", G_CALLBACK(zoom_out_clicked_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(explorem), zoom_out);

3230 GtkWidget *zoom = gtk_radio_menu_item_new_with_label(group, "Zoom");
    group = gtk_radio_menu_item_get_group(GTK_RADIO_MENU_ITEM(zoom));
    gtk_check_menu_item_set_active(GTK_CHECK_MENU_ITEM(zoom), TRUE);
    g_signal_connect(zoom, "toggled", G_CALLBACK(zoom_toggled_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(explorem), zoom);
3235
    GtkWidget *zoom_to = gtk_menu_item_new_with_label("Zoom To");
    g_signal_connect(zoom_to, "activate", G_CALLBACK(zoom_to_clicked_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(explorem), zoom_to);

3240 GtkWidget *info = gtk_radio_menu_item_new_with_label(group, "Info");
    group = gtk_radio_menu_item_get_group(GTK_RADIO_MENU_ITEM(info));
    g_signal_connect(info, "toggled", G_CALLBACK(info_toggled_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(explorem), info);

3245 GtkWidget *select_ = gtk_radio_menu_item_new_with_label(group, "Select");
    group = gtk_radio_menu_item_get_group(GTK_RADIO_MENU_ITEM(select_));
    g_signal_connect(select_, "toggled", G_CALLBACK(selection_toggled_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(explorem), select_);

3250 GtkWidget *more_iters = gtk_menu_item_new_with_label("Increase Iterations");
    g_signal_connect(more_iters, "activate", G_CALLBACK(more_iters_clicked_cb), ↙
        ↳ NULL);
```

```
    gtk_menu_shell_append(GTK_MENU_SHELL(explorem), more_iters);

    GtkWidget *fewer_iters = gtk_menu_item_new_with_label("Decrease Iterations");
3255 g_signal_connect(fewer_iters, "activate", G_CALLBACK(fewer_iters_clicked_cb), ↙
        ↳ NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(explorem), fewer_iters);

    gtk_menu_item_set_submenu(GTK_MENU_ITEM(explore), explorem);
    gtk_menu_shell_append(GTK_MENU_SHELL(menu), explore);
3260

    GtkWidget *feature = gtk_menu_item_new_with_label("Feature");
    GtkWidget *featurem = gtk_menu_new();

3265 GtkWidget *nucleus = gtk_radio_menu_item_new_with_label(group, "Nucleus");
    group = gtk_radio_menu_item_get_group(GTK_RADIO_MENU_ITEM(nucleus));
    g_signal_connect(nucleus, "toggled", G_CALLBACK(nucleus_toggled_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(featurem), nucleus);

3270 GtkWidget *bond = gtk_radio_menu_item_new_with_label(group, "Bond");
    group = gtk_radio_menu_item_get_group(GTK_RADIO_MENU_ITEM(bond));
    g_signal_connect(bond, "toggled", G_CALLBACK(bond_toggled_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(featurem), bond);

3275 GtkWidget *misiurewicz = gtk_radio_menu_item_new_with_label(group, "↙
        ↳ Misiurewicz");
    group = gtk_radio_menu_item_get_group(GTK_RADIO_MENU_ITEM(misiurewicz));
    g_signal_connect(misiurewicz, "toggled", G_CALLBACK(misiurewicz_toggled_cb), ↙
        ↳ NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(featurem), misiurewicz);

3280 GtkWidget *ray_out = gtk_radio_menu_item_new_with_label(group, "Ray Out");
    group = gtk_radio_menu_item_get_group(GTK_RADIO_MENU_ITEM(ray_out));
    g_signal_connect(ray_out, "toggled", G_CALLBACK(ray_out_toggled_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(featurem), ray_out);

3285 GtkWidget *ray_in = gtk_menu_item_new_with_label("Ray In");
    g_signal_connect(ray_in, "activate", G_CALLBACK(ray_in_clicked_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(featurem), ray_in);

    GtkWidget *extend = gtk_menu_item_new_with_label("Extend Ray");
3290 g_signal_connect(extend, "activate", G_CALLBACK(extend_clicked_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(featurem), extend);

    GtkWidget *wakel = gtk_menu_item_new_with_label("Wake <");
    g_signal_connect(wakel, "activate", G_CALLBACK(wakel_clicked_cb), NULL);
3295 gtk_menu_shell_append(GTK_MENU_SHELL(featurem), wakel);

    GtkWidget *waker = gtk_menu_item_new_with_label("Wake >");
    g_signal_connect(waker, "activate", G_CALLBACK(waker_clicked_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(featurem), waker);
3300
    GtkWidget *atom = gtk_menu_item_new_with_label("Atom");
    g_signal_connect(atom, "activate", G_CALLBACK(atom_clicked_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(featurem), atom);

3305 GtkWidget *domain = gtk_menu_item_new_with_label("Domain");
```

```
    g_signal_connect(domain, "activate", G_CALLBACK(domain_clicked_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(featurem), domain);

    GtkWidget *mu_unit = gtk_menu_item_new_with_label("Mu-Unit");
    g_signal_connect(mu_unit, "activate", G_CALLBACK(muunit_clicked_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(featurem), mu_unit);

    GtkWidget *filaments = gtk_menu_item_new_with_label("Filaments");
    g_signal_connect(filaments, "activate", G_CALLBACK(filaments_clicked_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(featurem), filaments);

    GtkWidget *delete_ = gtk_menu_item_new_with_label("Delete");
    g_signal_connect(delete_, "activate", G_CALLBACK(delete_clicked_cb), NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(featurem), delete_);

    gtk_menu_item_set_submenu(GTK_MENU_ITEM(feature), featurem);
    gtk_menu_shell_append(GTK_MENU_SHELL(menu), feature);

    GtkWidget *tbar = gtk_toolbar_new();

   G.filaments_period = gtk_entry_new();
    gtk_entry_set_placeholder_text(GTK_ENTRY(G.filaments_period), "Period");
    gtk_entry_set_width_chars(GTK_ENTRY(G.filaments_period), 6);
    GtkToolItem *filaments_period = gtk_tool_item_new();
    gtk_tool_item_set_homogeneous(filaments_period, 0);
    gtk_container_add(GTK_CONTAINER(filaments_period), G.filaments_period);
    gtk_toolbar_insert(GTK_TOOLBAR(tbar), filaments_period, -1);
   G.filaments_preperiod = gtk_entry_new();
    gtk_entry_set_placeholder_text(GTK_ENTRY(G.filaments_preperiod), "n-Fold");
    gtk_entry_set_width_chars(GTK_ENTRY(G.filaments_preperiod), 6);
    GtkToolItem *filaments_preperiod = gtk_tool_item_new();
    gtk_tool_item_set_homogeneous(filaments_preperiod, 0);
    gtk_container_add(GTK_CONTAINER(filaments_preperiod), G.filaments_preperiod);
    gtk_toolbar_insert(GTK_TOOLBAR(tbar), filaments_preperiod, -1);
    G.filaments_depth = gtk_entry_new();
    gtk_entry_set_placeholder_text(GTK_ENTRY(G.filaments_depth), "Depth");
    gtk_entry_set_width_chars(GTK_ENTRY(G.filaments_depth), 6);
    GtkToolItem *filaments_depth = gtk_tool_item_new();
    gtk_tool_item_set_homogeneous(filaments_depth, 0);
    gtk_container_add(GTK_CONTAINER(filaments_depth), G.filaments_depth);
    gtk_toolbar_insert(GTK_TOOLBAR(tbar), filaments_depth, -1);

    for (int dark = 0; dark <= 1; ++dark)
    {
      G.dashstore[dark] = gtk_list_store_new(1, GDK_TYPE_PIXBUF);
      for (int lt = 0; lt < 16; ++lt)
      {
        cairo_surface_t *surface = cairo_image_surface_create(CAIRO_FORMAT_ARGB32, 64, 16);
        cairo_t *cr = cairo_create(surface);
        cairo_set_source_rgba(cr, 0, 0, 0, 0);
        cairo_paint(cr);
        if (dark)
        {
          cairo_set_source_rgba(cr, 1, 1, 1, 1);
        }
```

```
        else
        {
          cairo_set_source_rgba(cr, 0, 0, 0, 1);
        }
3365    cairo_set_line_width(cr, 1);
        cairo_set_line_cap(cr, CAIRO_LINE_CAP_ROUND);
        cairo_set_line_join(cr, CAIRO_LINE_JOIN_ROUND);
        cairo_set_dash(cr, dashes[lt].pattern, dashes[lt].count, 0);
        cairo_move_to(cr, 8, 8);
3370    cairo_line_to(cr, 56, 8);
        cairo_stroke(cr);
        GtkTreeIter iter;
        gtk_list_store_append(G.dashstore[dark], &iter);
        gtk_list_store_set(G.dashstore[dark], &iter, 0, ↙
            ↳ gdk_pixbuf_get_from_surface(surface, 0, 0, 64, 16), -1);
3375    cairo_destroy(cr);
        cairo_surface_destroy(surface);
      }
    }
    G.dashcombo = gtk_combo_box_new_with_model(GTK_TREE_MODEL(G.dashstore[G.dark])↙
        ↳ );
3380 gtk_combo_box_set_active(GTK_COMBO_BOX(G.dashcombo), 0);
    GtkCellRenderer *renderer2 = gtk_cell_renderer_pixbuf_new();
    gtk_cell_layout_pack_start(GTK_CELL_LAYOUT(G.dashcombo), renderer2, FALSE);
    gtk_cell_layout_set_attributes(GTK_CELL_LAYOUT(G.dashcombo), renderer2, "↙
        ↳ pixbuf", 0, NULL);
    g_signal_connect(G.dashcombo, "changed", G_CALLBACK(↙
        ↳ dashing_selection_changed_cb), NULL);
3385 GtkToolItem *dashing = gtk_tool_item_new();
    gtk_tool_item_set_homogeneous(dashing, 0);
    gtk_container_add(GTK_CONTAINER(dashing), G.dashcombo);
    gtk_toolbar_insert(GTK_TOOLBAR(tbar), dashing, -1);

3390 initialize_fill_patterns();
    for (int dark = 0; dark <= 1; ++dark)
    {
      G.fillstore[dark] = gtk_list_store_new(1, GDK_TYPE_PIXBUF);
      for (int ft = 0; ft < 26; ++ft)
3395  {
        cairo_surface_t *surface = cairo_image_surface_create(CAIRO_FORMAT_ARGB32,↙
            ↳  64, 16);
        cairo_t *cr = cairo_create(surface);
        cairo_set_source_rgba(cr, 0, 0, 0, 0);
        cairo_paint(cr);
3400    cairo_set_source(cr, G.fillpattern[dark][ft]);
        cairo_paint(cr);
        GtkTreeIter iter;
        gtk_list_store_append(G.fillstore[dark], &iter);
        gtk_list_store_set(G.fillstore[dark], &iter, 0, ↙
            ↳ gdk_pixbuf_get_from_surface(surface, 0, 0, 64, 16), -1);
3405    cairo_destroy(cr);
        cairo_surface_destroy(surface);
      }
    }
    G.fillcombo = gtk_combo_box_new_with_model(GTK_TREE_MODEL(G.fillstore[G.dark])↙
        ↳ );
3410 gtk_combo_box_set_active(GTK_COMBO_BOX(G.fillcombo), 0);
```

```
    gtk_cell_layout_pack_start(GTK_CELL_LAYOUT(G.fillcombo), renderer2, FALSE);
    gtk_cell_layout_set_attributes(GTK_CELL_LAYOUT(G.fillcombo), renderer2, "↙
        ↳ pixbuf", 0, NULL);
    g_signal_connect(G.fillcombo, "changed", G_CALLBACK(fill_selection_changed_cb)↙
        ↳ , NULL);
    GtkToolItem *filling = gtk_tool_item_new();
3415 gtk_tool_item_set_homogeneous(filling, 0);
    gtk_container_add(GTK_CONTAINER(filling), G.fillcombo);
    gtk_toolbar_insert(GTK_TOOLBAR(tbar), filling, -1);

    G.colourbutton = gtk_color_button_new();
3420 g_signal_connect(G.colourbutton, "color-set", G_CALLBACK(colour_activated_cb),↙
        ↳ NULL);
    GtkToolItem *colortool = gtk_tool_item_new();
    gtk_tool_item_set_homogeneous(colortool, 0);
    gtk_container_add(GTK_CONTAINER(colortool), G.colourbutton);
    gtk_toolbar_insert(GTK_TOOLBAR(tbar), colortool, -1);
3425
    gtk_container_add(GTK_CONTAINER(h0), menu);
    gtk_container_add(GTK_CONTAINER(h0), tbar);

    G.log = gtk_text_view_new();
3430 gtk_text_view_set_editable(GTK_TEXT_VIEW(G.log), FALSE);
    gtk_text_view_set_cursor_visible(GTK_TEXT_VIEW(G.log), FALSE);
    gtk_text_view_set_wrap_mode(GTK_TEXT_VIEW(G.log), GTK_WRAP_CHAR);
    GtkWidget *scroll = gtk_scrolled_window_new(NULL, NULL);
    gtk_widget_set_size_request(scroll, G.width_screen, -1);
3435 gtk_widget_set_vexpand(G.log, TRUE);
    gtk_widget_set_hexpand(G.log, FALSE);
    gtk_widget_set_vexpand(scroll, TRUE);
    gtk_widget_set_hexpand(scroll, FALSE);
    gtk_container_add(GTK_CONTAINER(scroll), G.log);
3440
    G.annostore = gtk_tree_store_new(2, G_TYPE_STRING, G_TYPE_POINTER);
    G.annotree = gtk_tree_view_new_with_model(GTK_TREE_MODEL(G.annostore));
    g_object_unref(G_OBJECT(G.annostore));
    GtkTreeIter iter;
3445 gtk_tree_store_append(G.annostore, &iter, NULL);
    gtk_tree_store_set(G.annostore, &iter, 0, "(none)", 1, NULL, -1);
    GtkCellRenderer *renderer = gtk_cell_renderer_text_new();
    GtkTreeViewColumn *column = gtk_tree_view_column_new_with_attributes("↙
        ↳ Annotations", renderer, "text", 0, NULL);
    gtk_tree_view_column_set_sort_column_id(column, 0);
3450 gtk_tree_sortable_set_sort_func(GTK_TREE_SORTABLE(G.annostore), 0, ↙
        ↳ sortable_annotation_compare, 0, 0);
    gtk_tree_view_append_column(GTK_TREE_VIEW(G.annotree), column);
    GtkTreeSelection *select = gtk_tree_view_get_selection(GTK_TREE_VIEW(G.↙
        ↳ annotree));
    gtk_tree_selection_set_mode(select, GTK_SELECTION_SINGLE);
    g_signal_connect(select, "changed", G_CALLBACK(annotree_selection_changed_cb),↙
        ↳ NULL);
3455 GtkWidget *scroll2 = gtk_scrolled_window_new(NULL, NULL);
    gtk_widget_set_hexpand(G.annotree, TRUE);
    gtk_widget_set_vexpand(G.annotree, TRUE);
    gtk_widget_set_hexpand(scroll2, TRUE);
    gtk_widget_set_vexpand(scroll2, TRUE);
3460 gtk_container_add(GTK_CONTAINER(scroll2), G.annotree);
```

```
  G.annotation_queue = g_async_queue_new();
  g_thread_new("annotate", annotation_thread, G.annotation_queue);
  G.task_workers = g_thread_pool_new(task_func, NULL, g_get_num_processors(), ↙
      ↳ FALSE, NULL);
3465  G.task_box = gtk_box_new(GTK_ORIENTATION_VERTICAL, 0);
  GtkWidget *scroll3 = gtk_scrolled_window_new(NULL, NULL);
  gtk_widget_set_size_request(scroll3, -1, -1);
  gtk_widget_set_vexpand(G.task_box, TRUE);
  gtk_widget_set_hexpand(G.task_box, TRUE);
3470  gtk_widget_set_vexpand(scroll3, TRUE);
  gtk_widget_set_hexpand(scroll3, TRUE);
  gtk_container_add(GTK_CONTAINER(scroll3), G.task_box);

  gtk_container_add(GTK_CONTAINER(G.window), v1);
3475  gtk_container_add(GTK_CONTAINER(v1), h0);
  gtk_container_add(GTK_CONTAINER(v1), h2);
  gtk_container_add(GTK_CONTAINER(h2), v3);
  gtk_container_add(GTK_CONTAINER(v3), G.da);
  gtk_container_add(GTK_CONTAINER(v3), scroll);
3480  gtk_container_add(GTK_CONTAINER(h2), v4);
  gtk_container_add(GTK_CONTAINER(v4), scroll2);
  gtk_container_add(GTK_CONTAINER(v4), scroll3);

  gtk_window_maximize(GTK_WINDOW(G.window));
3485  gtk_widget_show_all(G.window);
  zoom_toggled_cb(zoom, 0);

  gtk_main();
  m_symbolics_exit();
3490  return 0;
}

enum task_t
{
3495  task_ray_out,
  task_ray_in,
  task_ray_extend,
  task_nucleus,
  task_bond,
3500  task_misiurewicz,
  task_muunit,
  task_muunit_nucleus,
  task_filaments,
  task_wake,
3505  task_wake2, // traces its own rays
  task_atom,
  task_domain
};

3510 struct task_ray_out
{
  mpc_t c;
  int dwell;
  int line_type;
3515  double stroke_r;
  double stroke_g;
```

```
  double stroke_b;
};

struct task_ray_in
{
  m_binangle angle;
  int depth;
  int line_type;
  double stroke_r;
  double stroke_g;
  double stroke_b;
};

struct task_ray_extend
{
  struct annotation *anno;
  int depth;
};

struct task_nucleus
{
  mpc_t c;
  mpfr_t r;
  int maxperiod;
};

struct task_bond
{
  mpc_t c;
  int period;
};

struct task_misiurewicz
{
  mpc_t c;
  mpfr_t r;
  int maxpreperiod;
  int maxperiod;
};

struct task_muunit
{
  mpc_t nucleus;
  int period;
};

struct task_muunit_nucleus
{
  mpc_t nucleus;
  int nucleus_period;
  int ray_period;
  mpq_t angle;
};

struct task_filaments
{
  mpc_t inner_nucleus;
```

```
  int inner_period;
  int outer_period;
  int preperiod;
  int depth;
  int line_type;
  double stroke_r;
  double stroke_g;
  double stroke_b;
};

struct task_wake
{
  struct annotation *lo;
  struct annotation *hi;
  int fill_type;
  double fill_r;
  double fill_g;
  double fill_b;
};

struct task_wake2
{
  char *lo;
  char *hi;
  int depth;
  int fill_type;
  double fill_r;
  double fill_g;
  double fill_b;
};

struct task_atom
{
  mpc_t nucleus;
  int period;
  int fill_type;
  double fill_r;
  double fill_g;
  double fill_b;
};

struct task_domain
{
  mpc_t nucleus;
  int loperiod;
  int period;
  int line_type;
  double line_r;
  double line_g;
  double line_b;
};

struct task
{
  enum task_t tag;
  char *label;
  GAsyncQueue *output;
```

```
  GtkWidget *hbox;
  GtkWidget *progress_bar;
  double progress;
  bool cancelled;
  union
  {
    struct task_ray_out ray_out;
    struct task_ray_in ray_in;
    struct task_ray_extend ray_extend;
    struct task_nucleus nucleus;
    struct task_bond bond;
    struct task_misiurewicz misiurewicz;
    struct task_muunit muunit;
    struct task_muunit_nucleus muunit_nucleus;
    struct task_filaments filaments;
    struct task_wake wake;
    struct task_wake2 wake2;
    struct task_atom atom;
    struct task_domain domain;
  } u;
};

static void task_cancel_cb(GtkWidget *cancel, gpointer userdata)
{
  struct task *t = userdata;
  t->cancelled = true;
  gtk_widget_set_sensitive(cancel, FALSE);
}

static gboolean task_enqueue_cb(gpointer userdata)
{
  struct task *t = userdata;
  t->progress_bar = gtk_progress_bar_new();
  GtkWidget *label = gtk_label_new(t->label);
  GtkWidget *cancel = gtk_button_new_with_label("Cancel");
  g_signal_connect(cancel, "clicked", G_CALLBACK(task_cancel_cb), t);
  t->hbox = gtk_box_new(GTK_ORIENTATION_HORIZONTAL, 0);
  gtk_container_add(GTK_CONTAINER(t->hbox), cancel);
  gtk_container_add(GTK_CONTAINER(t->hbox), t->progress_bar);
  gtk_container_add(GTK_CONTAINER(t->hbox), label);
  gtk_container_add(GTK_CONTAINER(G.task_box), t->hbox);
  gtk_widget_show_all(G.task_box);
  g_thread_pool_push(G.task_workers, t, NULL);
  return G_SOURCE_REMOVE;
}

static void task_enqueue(struct task *t)
{
  gdk_threads_add_idle(task_enqueue_cb, t);
}

static gboolean task_set_progress_cb(gpointer userdata)
{
  struct task *t = userdata;
  if (t->progress_bar)
  {
    if (t->progress == 0.0)
```

```
      {
        gtk_progress_bar_pulse(GTK_PROGRESS_BAR(t->progress_bar));
3690    }
      else
      {
        gtk_progress_bar_set_fraction(GTK_PROGRESS_BAR(t->progress_bar), t->↙
            ↳ progress);
      }
3695  }
  return G_SOURCE_REMOVE;
}

static void task_set_progress(struct task *t)
3700 {
  gdk_threads_add_idle(task_set_progress_cb, t);
}

static gboolean task_destroy_cb(gpointer userdata)
3705 {
  struct task *t = userdata;
  t->output = 0;
  if (t->hbox)
  {
3710    gtk_widget_destroy(t->hbox);
    t->hbox = 0;
  }
  t->progress_bar = 0;
  if (t->label)
3715  {
    free(t->label);
    t->label = 0;
  }
  switch (t->tag)
3720  {
    case task_ray_out:
    {
      mpc_clear(t->u.ray_out.c);
      break;
3725    }
    case task_ray_in:
    {
      m_binangle_clear(&t->u.ray_in.angle);
      break;
3730    }
    case task_ray_extend:
    {
      // nop, always succeeds
      break;
3735    }
    case task_nucleus:
    {
      mpc_clear(t->u.nucleus.c);
      mpfr_clear(t->u.nucleus.r);
3740    break;
    }
    case task_bond:
    {
```

```
        mpc_clear(t->u.bond.c);
3745    break;
      }
      case task_misiurewicz:
      {
        mpc_clear(t->u.misiurewicz.c);
3750    mpfr_clear(t->u.misiurewicz.r);
        break;
      }
      case task_muunit:
      {
3755    mpc_clear(t->u.muunit.nucleus);
        break;
      }
      case task_muunit_nucleus:
      {
3760    mpc_clear(t->u.muunit_nucleus.nucleus);
        mpq_clear(t->u.muunit_nucleus.angle);
        break;
      }
      case task_filaments:
3765  {
        mpc_clear(t->u.filaments.inner_nucleus);
        break;
      }
      case task_wake:
3770  {
        // nop, always succeeds
        break;
      }
      case task_wake2:
3775  {
        free(t->u.wake2.lo);
        free(t->u.wake2.hi);
        break;
      }
3780  case task_atom:
      {
        mpc_clear(t->u.atom.nucleus);
        break;
      }
3785  case task_domain:
      {
        mpc_clear(t->u.domain.nucleus);
        break;
      }
3790  }
    free(t);
    return G_SOURCE_REMOVE;
  }

3795 static void task_done(struct task *t)
  {
    gdk_threads_add_idle(task_destroy_cb, t);
  }

3800 static struct task *task_new_ray_out(GAsyncQueue *output, const mpc_t c, int ↙
```

```
    ↪ dwell, int line_type, double stroke_r, double stroke_g, double stroke_b)
{
  struct task *t = calloc(1, sizeof(*t));
  t->output = output;
  const char *format = "Ray Out from %+Re + %Re i dwell %d";
3805  int bytes = mpfr_snprintf(NULL, 0, format, mpc_realref(c), mpc_imagref(c), ↙
    ↪ dwell);
  if (bytes < 0)
  {
    bytes = 0; // error?
  }
3810  t->label = malloc(bytes + 1);
  mpfr_snprintf(t->label, bytes + 1, format, mpc_realref(c), mpc_imagref(c), ↙
    ↪ dwell);
  if (bytes == 0)
  {
    t->label[bytes] = 0; // paranoia
3815  }
  t->tag = task_ray_out;
  mpc_init2(t->u.ray_out.c, mpfr_get_prec(mpc_realref(c)));
  mpc_set(t->u.ray_out.c, c, MPC_RNDNN);
  t->u.ray_out.dwell = dwell;
3820  t->u.ray_out.line_type = line_type;
  t->u.ray_out.stroke_r = stroke_r;
  t->u.ray_out.stroke_g = stroke_g;
  t->u.ray_out.stroke_b = stroke_b;
  return t;
3825 }

static struct task *task_new_ray_in(GAsyncQueue *output, const m_binangle *angle↙
    ↪ , int depth, int line_type, double stroke_r, double stroke_g, double ↙
    ↪ stroke_b)
{
  struct task *t = calloc(1, sizeof(*t));
3830  t->output = output;
  t->label = m_binangle_to_new_string(angle);
  t->tag = task_ray_in;
  m_binangle_init(&t->u.ray_in.angle);
  m_binangle_set(&t->u.ray_in.angle, angle);
3835  t->u.ray_in.depth = depth;
  t->u.ray_in.line_type = line_type;
  t->u.ray_in.stroke_r = stroke_r;
  t->u.ray_in.stroke_g = stroke_g;
  t->u.ray_in.stroke_b = stroke_b;
3840  return t;
}

static struct task *task_new_ray_extend(GAsyncQueue *output, struct annotation *↙
    ↪ anno, int depth)
{
3845  struct task *t = calloc(1, sizeof(*t));
  t->output = output;
  t->label = strdup(anno->label);
  t->tag = task_ray_extend;
  t->u.ray_extend.anno = anno;
3850  t->u.ray_extend.depth = depth;
  return t;
```

```
}

static struct task *task_new_nucleus(GAsyncQueue *output, const mpc_t c, const ↙
    ↳ mpfr_t r, int dwell)
{
  struct task *t = calloc(1, sizeof(*t));
  t->output = output;
  const char *format = "nucleus near %+Re + %+Re i @ %Re";
  int bytes = mpfr_snprintf(0, 0, format, mpc_realref(c), mpc_imagref(c), r);
  if (bytes < 0)
  {
    bytes = 0;
  }
  t->label = malloc(bytes + 1);
  mpfr_snprintf(t->label, bytes + 1, format, mpc_realref(c), mpc_imagref(c), r);
  t->tag = task_nucleus;
  mpc_init2(t->u.nucleus.c, mpfr_get_prec(mpc_realref(c)));
  mpc_set(t->u.nucleus.c, c, MPC_RNDNN);
  mpfr_init2(t->u.nucleus.r, 53);
  mpfr_set(t->u.nucleus.r, r, MPFR_RNDN);
  t->u.nucleus.maxperiod = dwell;
  return t;
}

static struct task *task_new_bond(GAsyncQueue *output, const mpc_t c, int period↙
    ↳ )
{
  struct task *t = calloc(1, sizeof(*t));
  t->output = output;
  const char *format = "Bond near %+Re + %+Re i P %d";
  int bytes = mpfr_snprintf(0, 0, format, mpc_realref(c), mpc_imagref(c), period↙
      ↳ );
  if (bytes < 0)
  {
    bytes = 0;
  }
  t->label = malloc(bytes + 1);
  mpfr_snprintf(t->label, bytes + 1, format, mpc_realref(c), mpc_imagref(c), ↙
      ↳ period);
  t->tag = task_bond;
  mpc_init2(t->u.bond.c, mpfr_get_prec(mpc_realref(c)));
  mpc_set(t->u.bond.c, c, MPC_RNDNN);
  t->u.bond.period = period;
  return t;
}

static struct task *task_new_misiurewicz(GAsyncQueue *output, const mpc_t c, ↙
    ↳ const mpfr_t r, int maxpreperiod, int maxperiod)
{
  struct task *t = calloc(1, sizeof(*t));
  t->output = output;
  const char *format = "Misiurewicz near %+Re + %+Re i @ %Re";
  int bytes = mpfr_snprintf(0, 0, format, mpc_realref(c), mpc_imagref(c), r);
  if (bytes < 0)
  {
    bytes = 0;
  }
```

```
  t->label = malloc(bytes + 1);
3905  mpfr_snprintf(t->label, bytes + 1, format, mpc_realref(c), mpc_imagref(c), r);
  t->tag = task_misiurewicz;
  mpc_init2(t->u.misiurewicz.c, mpfr_get_prec(mpc_realref(c)));
  mpc_set(t->u.misiurewicz.c, c, MPC_RNDNN);
  mpfr_init2(t->u.misiurewicz.r, 53);
3910  mpfr_set(t->u.misiurewicz.r, r, MPFR_RNDN);
  t->u.misiurewicz.maxpreperiod = maxpreperiod;
  t->u.misiurewicz.maxperiod = maxperiod;
  return t;
}
3915
static struct task *task_new_muunit(GAsyncQueue *output, const mpc_t c, int ↙
    ↳ period)
{
  struct task *t = calloc(1, sizeof(*t));
  t->output = output;
3920  const char *format = "Mu-unit near %+Re + %+Re i P %d";
  int bytes = mpfr_snprintf(0, 0, format, mpc_realref(c), mpc_imagref(c), period↙
      ↳ );
  if (bytes < 0)
  {
    bytes = 0;
3925  }
  t->label = malloc(bytes + 1);
  mpfr_snprintf(t->label, bytes + 1, format, mpc_realref(c), mpc_imagref(c), ↙
      ↳ period);
  t->tag = task_muunit;
  mpc_init2(t->u.muunit.nucleus, mpfr_get_prec(mpc_realref(c)));
3930  mpc_set(t->u.muunit.nucleus, c, MPC_RNDNN);
  t->u.muunit.period = period;
  return t;
}

3935 static struct task *task_new_muunit_nucleus(GAsyncQueue *output, const mpc_t ↙
    ↳ nucleus, int nucleus_period, int ray_period, const mpq_t angle)
{
  struct task *t = calloc(1, sizeof(*t));
  t->output = output;
  const char *format = "Mu-unit nucleus near %+Re + %+Re i P %d = %d * %d";
3940  int bytes = mpfr_snprintf(0, 0, format, mpc_realref(nucleus), mpc_imagref(↙
      ↳ nucleus), nucleus_period * ray_period, nucleus_period, ray_period);
  if (bytes < 0)
  {
    bytes = 0;
  }
3945  t->label = malloc(bytes + 1);
  mpfr_snprintf(t->label, bytes + 1, format, mpc_realref(nucleus), mpc_imagref(↙
      ↳ nucleus), nucleus_period * ray_period, nucleus_period, ray_period);
  t->tag = task_muunit_nucleus;
  mpc_init2(t->u.muunit_nucleus.nucleus, mpfr_get_prec(mpc_realref(nucleus)));
  mpc_set(t->u.muunit_nucleus.nucleus, nucleus, MPC_RNDNN);
3950  t->u.muunit_nucleus.nucleus_period = nucleus_period;
  t->u.muunit_nucleus.ray_period = ray_period;
  mpq_init(t->u.muunit_nucleus.angle);
  mpq_set(t->u.muunit_nucleus.angle, angle);
  return t;
```

```
3955  }

      static struct task *task_new_filaments(GAsyncQueue *output, const mpc_t ↙
          ↳ inner_nucleus, int inner_period, int outer_period, int preperiod, int ↙
          ↳ depth, int line_type, double stroke_r, double stroke_g, double stroke_b)
      {
        struct task *t = calloc(1, sizeof(*t));
3960    t->output = output;
        const char *format = "Filaments near %+Re + %+Re i P %d / %d / %d";
        int bytes = mpfr_snprintf(0, 0, format, mpc_realref(inner_nucleus), ↙
            ↳ mpc_imagref(inner_nucleus), inner_period, outer_period, preperiod);
        if (bytes < 0)
        {
3965      bytes = 0;
        }
        t->label = malloc(bytes + 1);
        mpfr_snprintf(t->label, bytes + 1, format, mpc_realref(inner_nucleus), ↙
            ↳ mpc_imagref(inner_nucleus), inner_period, outer_period, preperiod);
        t->tag = task_filaments;
3970    mpc_init2(t->u.filaments.inner_nucleus, mpfr_get_prec(mpc_realref(↙
            ↳ inner_nucleus)));
        mpc_set(t->u.filaments.inner_nucleus, inner_nucleus, MPC_RNDNN);
        t->u.filaments.inner_period = inner_period;
        t->u.filaments.outer_period = outer_period;
        t->u.filaments.preperiod = preperiod;
3975    t->u.filaments.depth = depth;
        t->u.filaments.line_type = line_type;
        t->u.filaments.stroke_r = stroke_r;
        t->u.filaments.stroke_g = stroke_g;
        t->u.filaments.stroke_b = stroke_b;
3980    return t;
      }

      static struct task *task_new_wake(GAsyncQueue *output, struct annotation *lo, ↙
          ↳ struct annotation *hi, int fill_type, double fill_r, double fill_g, double↙
          ↳  fill_b)
      {
3985    struct task *t = calloc(1, sizeof(*t));
        t->output = output;
        t->label = strdup("Wake");
        t->tag = task_wake;
        t->u.wake.fill_type = fill_type;
3990    t->u.wake.fill_r = fill_r;
        t->u.wake.fill_g = fill_g;
        t->u.wake.fill_b = fill_b;
        t->u.wake.lo = lo;
        t->u.wake.hi = hi;
3995    return t;
      }

      static struct task *task_new_wake2(GAsyncQueue *output, char *lo, char *hi, int ↙
          ↳ depth, int fill_type, double fill_r, double fill_g, double fill_b)
      {
4000    struct task *t = calloc(1, sizeof(*t));
        t->output = output;
        t->label = strdup("Wake");
        t->tag = task_wake2;
```

```
  t->u.wake2.fill_type = fill_type;
  t->u.wake2.fill_r = fill_r;
  t->u.wake2.fill_g = fill_g;
  t->u.wake2.fill_b = fill_b;
  t->u.wake2.lo = strdup(lo);
  t->u.wake2.hi = strdup(hi);
  t->u.wake2.depth = depth;
  return t;
}

static struct task *task_new_atom(GAsyncQueue *output, const mpc_t c, int period
    , int fill_type, double fill_r, double fill_g, double fill_b)
{
  struct task *t = calloc(1, sizeof(*t));
  t->output = output;
  const char *format = "Atom near %+Re + %+Re i P %d";
  int bytes = mpfr_snprintf(0, 0, format, mpc_realref(c), mpc_imagref(c), period
      );
  if (bytes < 0)
  {
    bytes = 0;
  }
  t->label = malloc(bytes + 1);
  mpfr_snprintf(t->label, bytes + 1, format, mpc_realref(c), mpc_imagref(c),
      period);
  t->tag = task_atom;
  mpc_init2(t->u.atom.nucleus, mpfr_get_prec(mpc_realref(c)));
  mpc_set(t->u.atom.nucleus, c, MPC_RNDNN);
  t->u.atom.period = period;
  t->u.atom.fill_type = fill_type;
  t->u.atom.fill_r = fill_r;
  t->u.atom.fill_g = fill_g;
  t->u.atom.fill_b = fill_b;
  return t;
}

static struct task *task_new_domain(GAsyncQueue *output, const mpc_t c, int
    period, int loperiod, int line_type, double line_r, double line_g, double
    line_b)
{
  struct task *t = calloc(1, sizeof(*t));
  t->output = output;
  const char *format = "Domain %dp%d near %+Re + %+Re i ";
  int bytes = mpfr_snprintf(0, 0, format, loperiod, period, mpc_realref(c),
      mpc_imagref(c));
  if (bytes < 0)
  {
    bytes = 0;
  }
  t->label = malloc(bytes + 1);
  mpfr_snprintf(t->label, bytes + 1, format, loperiod, period, mpc_realref(c),
      mpc_imagref(c));
  t->tag = task_domain;
  mpc_init2(t->u.domain.nucleus, mpfr_get_prec(mpc_realref(c)));
  mpc_set(t->u.domain.nucleus, c, MPC_RNDNN);
  t->u.domain.period = period;
  t->u.domain.loperiod = loperiod;
```

```
  t->u.domain.line_type = line_type;
4055  t->u.domain.line_r = line_r;
  t->u.domain.line_g = line_g;
  t->u.domain.line_b = line_b;
  return t;
}
4060
static gboolean annotation_cb(gpointer userdata)
{
  struct annotation *a = userdata;
  add_annotation(a);
4065  gtk_widget_queue_draw(G.da);
  return G_SOURCE_REMOVE;
}

static gpointer annotation_thread(gpointer userdata)
4070 {
  GAsyncQueue *q = userdata;
  struct annotation *a;
  while ((a = g_async_queue_pop(q)))
  {
4075    gdk_threads_add_idle(annotation_cb, a);
  }
  return 0;
}

4080 static void task_func(gpointer data, gpointer userdata)
{
  (void) userdata;
  struct task *t = data;
  switch (t->tag)
4085  {

    case task_ray_out:
    {
      struct annotation *a = calloc(1, sizeof(*a));
4090    a->tag = annotation_ray_out;
      a->u.ray_out.line_type = t->u.ray_out.line_type;
      a->u.ray_out.stroke_r = t->u.ray_out.stroke_r;
      a->u.ray_out.stroke_g = t->u.ray_out.stroke_g;
      a->u.ray_out.stroke_b = t->u.ray_out.stroke_b;
4095    char *accumulated_bits = malloc(t->u.ray_out.dwell + 10);
      int i = 0;
      int sharpness = 16;
      m_r_exray_out *ray = m_r_exray_out_new(t->u.ray_out.c, sharpness, t->u.↙
          ↳ ray_out.dwell * 2, 65536);
      double total = 0.5 * sharpness * t->u.ray_out.dwell * (1 + t->u.ray_out.↙
          ↳ dwell);
4100    double progress = 0;
      double increment = t->u.ray_out.dwell;
      m_newton success = m_failed;
      int first = 1;
      while (! t->cancelled && (m_failed != (success = m_r_exray_out_step(ray)))↙
          ↳ )
4105    {
        progress = progress + increment;
        t->progress = progress / total;
```

```
            task_set_progress(t);
            a->u.ray_out.line_start = point_new(a->u.ray_out.line_start);
4110        if (first)
            {
              first = 0;
              a->u.ray_out.line_end = a->u.ray_out.line_start;
            }
4115        m_r_exray_out_get(ray, a->u.ray_out.line_start->xy);
            if (m_r_exray_out_have_bit(ray))
            {
              increment = increment - 1;
              bool bit = m_r_exray_out_get_bit(ray);
4120          accumulated_bits[i++] = bit ? '1' : '0';
              if (i >= t->u.ray_out.dwell + 10 - 1)
              {
                // should never happen?
                success = m_failed;
4125            break;
              }
            }
            if (success == m_converged)
            {
4130          break;
            }
          }
          if (! t->cancelled && success == m_converged)
          {
4135        accumulated_bits[i] = 0;
            int nbits = i;
            char *reversed_bits = malloc(nbits + 1);
            for (int i = 0; i < nbits; ++i)
            {
4140          reversed_bits[i] = accumulated_bits[nbits-1 - i];
            }
            reversed_bits[nbits] = 0;
            a->label = reversed_bits;
            g_async_queue_push(t->output, a);
4145      }
          else
          {
            free_annotation(a);
          }
4150      free(accumulated_bits);
          m_r_exray_out_delete(ray);
          break;
        } // task_ray_out

4155    case task_ray_in:
        {
          struct annotation *a = calloc(1, sizeof(*a));
          a->u.ray_in.line_type = t->u.ray_in.line_type;
          a->u.ray_in.stroke_r = t->u.ray_in.stroke_r;
4160      a->u.ray_in.stroke_g = t->u.ray_in.stroke_g;
          a->u.ray_in.stroke_b = t->u.ray_in.stroke_b;
          a->label = m_binangle_to_new_string(&t->u.ray_in.angle);
          a->tag = annotation_ray_in;
          m_binangle_init(&a->u.ray_in.angle);
```

```
4165        m_binangle_set(&a->u.ray_in.angle, &t->u.ray_in.angle);
            int depth = t->u.ray_in.depth;
            int sharpness = 16;
            mpq_t q;
            mpq_init(q);
4170        m_binangle_to_rational(q, &t->u.ray_in.angle);
            mpq_canonicalize(q);
            m_r_exray_in *ray = m_r_exray_in_new(q, sharpness);
            mpq_clear(q);
            int i = 0;
4175        double total = 0.5 * sharpness * depth * (1 + depth);
            double progress = 0;
            int increment = 1;
            m_newton success = m_failed;
            int first = 1;
4180        while (! t->cancelled && (m_failed != (success = m_r_exray_in_step(ray, ↙
                ↳ 64))))
            {
              progress = progress + increment;
              t->progress = progress / total;
              task_set_progress(t);
4185          a->u.ray_in.line_start = point_new(a->u.ray_in.line_start);
              m_r_exray_in_get(ray, a->u.ray_in.line_start->xy);
              if (first)
              {
                first = 0;
4190            a->u.ray_in.line_end = a->u.ray_in.line_start;
              }
              if (++i >= sharpness)
              {
                i = 0;
4195            increment = increment + 1;
                if (increment >= depth)
                {
                  success = m_converged;
                  break;
4200            }
              }
              if (success == m_converged)
              {
                break;
4205          }
            }
            a->u.ray_in.ray = ray;
            a->u.ray_in.depth = increment;
            if (! t->cancelled && success == m_converged)
4210        {
              g_async_queue_push(t->output, a);
            }
            else
            {
4215          free_annotation(a);
            }
            break;
          } // task_ray_in

4220      case task_ray_extend:
```

```
    {
      struct annotation *a = t->u.ray_extend.anno;
      m_r_exray_in *ray = a->u.ray_in.ray;
      int sharpness = 16; // FIXME must match ray_in above
4225  int start_depth = a->u.ray_in.depth;
      int end_depth = t->u.ray_extend.depth;
      int i = 0;
      double total = 0.5 * sharpness * (end_depth - start_depth) * (end_depth + ↙
          ↳ start_depth);
      double progress = 0;
4230  int increment = start_depth;
      m_newton success = m_failed;
      while (! t->cancelled && (m_failed != (success = m_r_exray_in_step(ray, ↙
          ↳ 64))))
      {
        progress = progress + increment;
4235    t->progress = progress / total;
        task_set_progress(t);
        a->u.ray_in.line_start = point_new(a->u.ray_in.line_start);
        m_r_exray_in_get(ray, a->u.ray_in.line_start->xy);
        if (++i >= sharpness)
4240    {
          i = 0;
          increment = increment + 1;
          if (increment >= end_depth)
          {
4245        success = m_converged;
            break;
          }
        }
        if (success == m_converged)
4250    {
          break;
        }
      }
      a->u.ray_in.depth = increment;
4255  g_async_queue_push(t->output, a);
      break;
    } // task_ray_extend

    case task_nucleus:
4260  {
      struct annotation *a = calloc(1, sizeof(*a));
      a->tag = annotation_nucleus;
      int prec = 53 - 2 * mpfr_get_exp(t->u.nucleus.r);
      mpc_init2(a->u.nucleus.xy, prec);
4265  mpfr_init2(a->u.nucleus.size, 53);
      mpfr_init2(a->u.nucleus.domain_size, 53);
      int period = m_r_ball_period_do(t->u.nucleus.c, t->u.nucleus.r, t->u.↙
          ↳ nucleus.maxperiod);
      task_set_progress(t);
      bool done = false;
4270  if (period && ! t->cancelled)
      {
        a->label = malloc(100);
        snprintf(a->label, 99, "%d", period);
        a->u.nucleus.period = period;
```

```
4275            if (prec > 4 * period + 53) prec = 4 * period + 53;
                if (prec < 53) prec = 53;
                mpfr_prec_round(mpc_realref(a->u.nucleus.xy), prec, MPFR_RNDN);
                mpfr_prec_round(mpc_imagref(a->u.nucleus.xy), prec, MPFR_RNDN);
                mpc_set(a->u.nucleus.xy, t->u.nucleus.c, MPC_RNDNN);
4280            m_newton ok = m_r_nucleus(a->u.nucleus.xy, a->u.nucleus.xy, period, 64, ↙
                    ↳ 0);
                task_set_progress(t);
                if (ok != m_failed && ! t->cancelled)
                {
                  mpc_t delta;
4285              mpc_init2(delta, 53);
                  mpc_sub(delta, t->u.nucleus.c, a->u.nucleus.xy, MPC_RNDNN);
                  mpfr_t distance;
                  mpfr_init2(distance, 53);
                  mpc_abs(distance, delta, MPFR_RNDN);
4290              bool nearby = mpfr_less_p(distance, t->u.nucleus.r);
                  mpc_clear(delta);
                  mpfr_clear(distance);
                  task_set_progress(t);
                  if (nearby && ! t->cancelled)
                  {
4295                mpc_t size;
                    mpc_init2(size, 53);
                    m_r_size(size, a->u.nucleus.xy, period);
                    mpc_abs(a->u.nucleus.size, size, MPFR_RNDN);
4300                mpc_clear(size);
                    task_set_progress(t);
                    if (! t->cancelled)
                    {
                      prec = 53 - mpfr_get_exp(a->u.nucleus.size);
4305                  if (prec > 4 * period + 53) prec = 4 * period + 53;
                      if (prec < 53) prec = 53;
                      mpfr_prec_round(mpc_realref(a->u.nucleus.xy), prec, MPFR_RNDN);
                      mpfr_prec_round(mpc_imagref(a->u.nucleus.xy), prec, MPFR_RNDN);
                      m_r_nucleus(a->u.nucleus.xy, a->u.nucleus.xy, period, 2, 0);
4310                  task_set_progress(t);
                      if (! t->cancelled)
                      {
                        m_r_domain_size(a->u.nucleus.domain_size, a->u.nucleus.xy, ↙
                            ↳ period);
                        task_set_progress(t);
4315                    if (! t->cancelled)
                        {
                          g_async_queue_push(t->output, a);
                          done = true;
                        }
4320                  }
                    }
                  }
                }
              }
4325          if (! done)
              {
                free_annotation(a);
              }
              break;
```

```
4330      } // task_nucleus

          case task_bond:
          {
            bool done = false;
4335        struct annotation *a = 0;
            mpfr_prec_t prec = mpfr_get_prec(mpc_realref(t->u.bond.c));
            mpc_t c, root, parent;
            mpc_init2(c, prec);
            mpc_init2(root, prec);
4340        mpc_init2(parent, prec);
            mpc_set(c, t->u.bond.c, MPC_RNDNN);
            m_r_nucleus(c, c, t->u.bond.period, 64, 0);
            task_set_progress(t);
            mpq_t angle;
4345        mpq_init(angle);
            if (! t->cancelled)
            {
              int p = m_r_parent(angle, root, parent, c, t->u.bond.period, 16, 0);
              task_set_progress(t);
4350          if (p && ! t->cancelled) {
                a = calloc(1, sizeof(*a));
                a->label = malloc(100);
                mpfr_snprintf(a->label, 99, "%Qd", angle);
                a->tag = annotation_text;
4355            mpc_init2(a->u.text.xy, mpfr_get_prec(mpc_realref(root)));
                mpc_set(a->u.text.xy, root, MPC_RNDNN);
                g_async_queue_push(t->output, a);
                done = true;
              }
4360        }
            mpq_clear(angle);
            mpc_clear(root);
            mpc_clear(parent);
            mpc_clear(c);
4365        if (! done)
            {
              if (a)
              {
                free_annotation(a);
4370          }
            }
            break;
          } // task_bond

4375      case task_misiurewicz:
          {
            bool done = false;
            struct annotation *a = 0;
            int preperiod = -1, period = 0;
4380        if (m_r_box_misiurewicz_do(&preperiod, &period, t->u.misiurewicz.c, t->u.↙
                ↳ misiurewicz.r, t->u.misiurewicz.maxpreperiod, t->u.misiurewicz.↙
                ↳ maxperiod))
            {
              task_set_progress(t);
              if (! t->cancelled)
              {
```

```
4385            a = calloc(1, sizeof(*a));
                a->label = malloc(100);
                snprintf(a->label, 99, "%dp%d", preperiod, period);
                a->tag = annotation_misiurewicz;
                a->u.misiurewicz.preperiod = preperiod;
4390            a->u.misiurewicz.period = period;
                mpc_init2(a->u.misiurewicz.xy, mpfr_get_prec(mpc_realref(t->u.↙
                    ↳ misiurewicz.c)));
                mpfr_init2(a->u.misiurewicz.domain_size, 53);
                m_r_misiurewicz(a->u.misiurewicz.xy, t->u.misiurewicz.c, preperiod, ↙
                    ↳ period, 64);
                task_set_progress(t);
4395            if (! t->cancelled)
                {
                  m_r_misiurewicz_naive(a->u.misiurewicz.xy, a->u.misiurewicz.xy, ↙
                      ↳ preperiod, period, 64);
                  task_set_progress(t);
                  if (! t->cancelled)
                  {
4400                mpfr_set_si(a->u.misiurewicz.domain_size, 0, MPFR_RNDN); // FIXME ↙
                        ↳ implement this
                    g_async_queue_push(t->output, a);
                    done = true;
                  }
4405            }
              }
            }
            if (! done)
            {
4410          if (a)
              {
                free_annotation(a);
              }
            }
4415        break;
          } // task_misiurewicz

          case task_muunit:
          {
4420        mpq_t q;
            mpq_init(q);
            for (int period = 2; period <= 6; ++period)
            {
              int den = (1 << period) - 1;
4425          for (int num = 1; num < den; num += 2)
              {
                mpq_set_ui(q, num, den);
                mpq_canonicalize(q);
                task_enqueue(task_new_muunit_nucleus(t->output, t->u.muunit.nucleus, t↙
                    ↳ ->u.muunit.period, period, q));
4430          }
            }
            mpq_clear(q);
            break;
          } // task muunit
4435
          case task_muunit_nucleus:
```

```
    {
      bool done = false;
      struct annotation *a = 0;
4440  mpc_t c, z, cc;
      mpc_init2(c, 53);
      mpc_init2(z, 53);
      mpc_init2(cc, 53);
      m_r_exray_in *ray = m_r_exray_in_new(t->u.muunit_nucleus.angle, 16);
4445  for (int i = 0; i < 16 * 4 * t->u.muunit_nucleus.ray_period; ++i)
      {
        m_r_exray_in_step(ray, 64);
      }
      m_r_exray_in_get(ray, c);
4450  m_r_exray_in_delete(ray);
      if (m_failed != m_r_nucleus(c, c, t->u.muunit_nucleus.ray_period, 64, 0) ↙
          ↳ && ! t->cancelled)
      {
        task_set_progress(t);
        mpc_set_dc(z, 0, MPC_RNDNN);
4455    for (int r = 0; r <= 64; ++r)
        {
          mpfr_mul_d(mpc_realref(cc), mpc_realref(c), r / 64.0, MPFR_RNDN);
          mpfr_mul_d(mpc_imagref(cc), mpc_imagref(c), r / 64.0, MPFR_RNDN);
          if (m_failed == m_r_attractor(z, z, cc, 1, 64)) break;
4460    }
        mpfr_mul_d(mpc_realref(cc), mpc_realref(z), 2, MPFR_RNDN);
        mpfr_mul_d(mpc_imagref(cc), mpc_imagref(z), 2, MPFR_RNDN);
        mpfr_prec_t prec = mpfr_get_prec(mpc_realref(t->u.muunit_nucleus.nucleus↙
            ↳ ));
        mpfr_set_prec(mpc_realref(z), prec);
4465    mpfr_set_prec(mpc_imagref(z), prec);
        mpfr_set_prec(mpc_realref(c), prec);
        mpfr_set_prec(mpc_imagref(c), prec);
        mpc_set_dc(z, 0, MPC_RNDNN);
        if (m_failed != m_r_interior(z, c, z, t->u.muunit_nucleus.nucleus, cc, t↙
            ↳ ->u.muunit_nucleus.nucleus_period, 64) && ! t->cancelled)
4470    {
          task_set_progress(t);
          int period = t->u.muunit_nucleus.nucleus_period * t->u.muunit_nucleus.↙
              ↳ ray_period;
          if (m_failed != m_r_nucleus(c, c, period, 64, 0) && ! t->cancelled)
          {
4475        task_set_progress(t);
            a = calloc(1, sizeof(*a));
            a->label = malloc(100);
            snprintf(a->label, 99, "%d", period);
            a->tag = annotation_nucleus;
4480        if (prec > 4 * period + 53) prec = 4 * period + 53;
            if (prec < 53) prec = 53;
            mpc_init2(a->u.nucleus.xy, prec);
            mpfr_init2(a->u.nucleus.size, 53);
            mpfr_init2(a->u.nucleus.domain_size, 53);
4485        a->u.nucleus.period = period;
            if (m_failed != m_r_nucleus(a->u.nucleus.xy, c, period, 64, 0) && ! ↙
                ↳ t->cancelled)
            {
              task_set_progress(t);
```

```
                mpc_t size;
4490            mpc_init2(size, 53);
                m_r_size(size, a->u.nucleus.xy, period);
                mpc_abs(a->u.nucleus.size, size, MPFR_RNDN);
                mpc_clear(size);
                task_set_progress(t);
4495            if (! t->cancelled)
                {
                  prec = 53 - mpfr_get_exp(a->u.nucleus.size);
                  if (prec > 4 * period + 53) prec = 4 * period + 53;
                  if (prec < 53) prec = 53;
4500              mpfr_prec_round(mpc_realref(a->u.nucleus.xy), prec, MPFR_RNDN);
                  mpfr_prec_round(mpc_imagref(a->u.nucleus.xy), prec, MPFR_RNDN);
                  m_r_nucleus(a->u.nucleus.xy, a->u.nucleus.xy, period, 2, 0);
                  task_set_progress(t);
                  if (! t->cancelled)
4505              {
                    m_r_domain_size(a->u.nucleus.domain_size, a->u.nucleus.xy, ↙
                        ↳ period);
                    task_set_progress(t);
                    if (! t->cancelled)
                    {
4510                  g_async_queue_push(t->output, a);
                      done = true;
                    }
                  }
                }
4515          }
            }
          }
          if (! done)
4520      {
            if (a)
            {
              free_annotation(a);
            }
4525      }
          break;
        } // task_muunit_nucleus

        case task_filaments:
4530    {
          if (t->u.filaments.outer_period <= 0) t->u.filaments.outer_period = 1;
          if (t->u.filaments.preperiod <= 0) t->u.filaments.preperiod = 0;
          if (t->u.filaments.depth <= 0) t->u.filaments.depth = 1;
          char *lo = 0;
4535      char *hi = 0;
          m_r_external_angles(&lo, &hi, t->u.filaments.inner_nucleus, t->u.filaments↙
              ↳ .inner_period, 0.75, 63);
          if (lo && hi)
          {
            m_binangle inner, INNER, outer, OUTER, a, b, i, o;
4540        m_binangle_init(&inner);
            m_binangle_init(&INNER);
            m_binangle_init(&outer);
            m_binangle_init(&OUTER);
```

```
        m_binangle_init(&a);
4545    m_binangle_init(&b);
        m_binangle_init(&i);
        m_binangle_init(&o);
        m_binangle_from_string(&inner, lo);
        m_binangle_from_string(&INNER, hi);
4550    free(lo);
        free(hi);
        lo = 0;
        hi = 0;
        // FIXME verify correctness
4555    mpc_t outer_nucleus;
        mpc_init2(outer_nucleus, mpfr_get_prec(mpc_realref(t->u.filaments.↙
            ↳ inner_nucleus)));
        m_r_nucleus(outer_nucleus, t->u.filaments.inner_nucleus, t->u.filaments.↙
            ↳ outer_period, 64, 0);
        m_r_external_angles(&lo, &hi, outer_nucleus, t->u.filaments.outer_period↙
            ↳ , 0.75, 63);
        mpc_clear(outer_nucleus);
4560    if (lo && hi)
        {
          m_binangle_from_string(&outer, lo);
          m_binangle_from_string(&OUTER, hi);
          int depth = 2 * (4 * t->u.filaments.outer_period + t->u.filaments.↙
              ↳ inner_period * (t->u.filaments.preperiod + 1));
4565      int outer_preperiod_max = t->u.filaments.depth - t->u.filaments.↙
              ↳ preperiod;
          if (outer_preperiod_max < 1) outer_preperiod_max = 1;
          for (int outer_preperiod = 0; outer_preperiod <= outer_preperiod_max; ↙
              ↳ ++outer_preperiod)
          {
            for (int outer_pre = outer_preperiod ? 1 : 0; outer_pre < 1 << ↙
                ↳ outer_preperiod; outer_pre += 2)
4570        {
              mpz_set_ui(a.pre.bits, outer_pre);
              a.pre.length = outer_preperiod;
              mpz_set_ui(a.per.bits, 0);
              a.per.length = 1;
4575          for (int inner_pre = 0; inner_pre < 1 << t->u.filaments.preperiod;↙
                  ↳ ++inner_pre)
              {
                mpz_set_ui(b.pre.bits, inner_pre);
                b.pre.length = t->u.filaments.preperiod;
                mpz_set_ui(b.per.bits, 0);
4580            b.per.length = 1;
                m_binangle_tune(&i, &b, &inner.per, &INNER.per);
                m_binangle_tune(&o, &a, &outer.per, &OUTER.per);
                m_block_append(&o.pre, &i.pre, &o.pre);
                task_enqueue(task_new_ray_in(t->output, &o, depth, t->u.↙
                    ↳ filaments.line_type, t->u.filaments.stroke_r, t->u.↙
                    ↳ filaments.stroke_g, t->u.filaments.stroke_b));
4585            if (m_binangle_other_representation(&a) && outer.per.length > 1)
                {
                  m_binangle_tune(&i, &b, &inner.per, &INNER.per);
                  m_binangle_tune(&o, &a, &outer.per, &OUTER.per);
                  m_block_append(&o.pre, &i.pre, &o.pre);
4590              task_enqueue(task_new_ray_in(t->output, &o, depth, t->u.↙
```

```
                        ↪ filaments.line_type, t->u.filaments.stroke_r, t->u.↙
                        ↪ filaments.stroke_g, t->u.filaments.stroke_b));
                    }
                  }
                }
              }
4595          }
              m_binangle_clear(&inner);
              m_binangle_clear(&INNER);
              m_binangle_clear(&outer);
              m_binangle_clear(&OUTER);
4600          m_binangle_clear(&a);
              m_binangle_clear(&b);
              m_binangle_clear(&i);
              m_binangle_clear(&o);
            }
4605        if (lo) free(lo);
            if (hi) free(hi);
            break;
          } // task_filaments

4610      case task_wake:
          {
            task_set_progress(t);
            // copy lines
            struct annotation *a = calloc(1, sizeof(*a));
4615        a->tag = annotation_wake;
            int bytes = strlen("[") + strlen(t->u.wake.lo->label) + strlen("|") + ↙
                ↪ strlen(t->u.wake.hi->label) + strlen("]") + 1;
            a->label = calloc(1, bytes + 1);
            snprintf(a->label, bytes, "[%s|%s]", t->u.wake.lo->label, t->u.wake.hi->↙
                ↪ label);
            a->u.wake.ray_lo = strdup(t->u.wake.lo->label);
4620        a->u.wake.ray_hi = strdup(t->u.wake.hi->label);
            int dlo = t->u.wake.lo->u.ray_in.depth;
            int dhi = t->u.wake.hi->u.ray_in.depth;
            a->u.wake.ray_depth = dlo > dhi ? dlo : dhi;
            a->u.wake.fill_type = t->u.wake.fill_type;
4625        a->u.wake.fill_r = t->u.wake.fill_r;
            a->u.wake.fill_g = t->u.wake.fill_g;
            a->u.wake.fill_b = t->u.wake.fill_b;
            task_set_progress(t);
            a->u.wake.line_start = points_copy(t->u.wake.lo->u.ray_in.line_end, &a->u.↙
                ↪ wake.line_end);
4630        task_set_progress(t);
            struct point *p = 0, *q = 0;
            p = points_copy(t->u.wake.hi->u.ray_in.line_end, &q);
            // reverse and prepend
            task_set_progress(t);
4635        while (p)
            {
              q = p->next;
              p->pred = 0;
              p->next = a->u.wake.line_start;
4640          a->u.wake.line_start = p;
              if (p->next) p->next->pred = p;
              p = q;
```

```
      }
      task_set_progress(t);
      mpq_t slo, shi;
      mpq_init(slo);
      mpq_init(shi);
      m_binangle_to_rational(slo, &t->u.wake.lo->u.ray_in.angle);
      m_binangle_to_rational(shi, &t->u.wake.hi->u.ray_in.angle);
      mpq_init(a->u.wake.width);
      mpq_sub(a->u.wake.width, shi, slo);
      mpq_clear(shi);
      mpq_clear(slo);
      g_async_queue_push(t->output, t->u.wake.lo);
      g_async_queue_push(t->output, t->u.wake.hi);
      g_async_queue_push(t->output, a);
      break;
    } // task_wake

    case task_wake2:
    {
      struct annotation *a = calloc(1, sizeof(*a));
      a->tag = annotation_wake;
      int bytes = strlen("[") + strlen(t->u.wake2.lo) + strlen("|") + strlen(t->u.wake2.hi) + strlen("]") + 1;
      a->label = calloc(1, bytes + 1);
      snprintf(a->label, bytes, "[%s|%s]", t->u.wake2.lo, t->u.wake2.hi);
      a->u.wake.ray_lo = strdup(t->u.wake2.lo);
      a->u.wake.ray_hi = strdup(t->u.wake2.hi);
      a->u.wake.ray_depth = t->u.wake2.depth;
      a->u.wake.fill_type = t->u.wake2.fill_type;
      a->u.wake.fill_r = t->u.wake2.fill_r;
      a->u.wake.fill_g = t->u.wake2.fill_g;
      a->u.wake.fill_b = t->u.wake2.fill_b;
      task_set_progress(t);
      // trace lo ray
      m_binangle b;
      m_binangle_init(&b);
      m_binangle_from_string(&b, t->u.wake2.lo);
      int depth = t->u.wake2.depth;
      int sharpness = 16;
      mpq_t q1, q2;
      mpq_init(q1);
      mpq_init(q2);
      m_binangle_to_rational(q1, &b);
      m_r_exray_in *ray = m_r_exray_in_new(q1, sharpness);
      int i = 0;
      double total = 0.5 * sharpness * depth * (1 + depth) * 2;
      double progress = 0;
      int increment = 1;
      m_newton success1 = m_failed;
      m_newton success2 = m_failed;
      int first = 1;
      while (! t->cancelled && (m_failed != (success1 = m_r_exray_in_step(ray, 64))))
      {
        progress = progress + increment;
        t->progress = progress / total;
        task_set_progress(t);
```

```
        // prepend point
        a->u.wake.line_start = point_new(a->u.wake.line_start);
4700    m_r_exray_in_get(ray, a->u.wake.line_start->xy);
        if (first)
        {
          first = 0;
          a->u.wake.line_end = a->u.wake.line_start;
4705    }
        if (++i >= sharpness)
        {
          i = 0;
          increment = increment + 1;
4710      if (increment >= depth)
          {
            success1 = m_converged;
            break;
          }
4715    }
        if (success1 == m_converged)
        {
          break;
        }
4720  }
      m_r_exray_in_delete(ray);
      // trace hi ray
      m_binangle_from_string(&b, t->u.wake2.hi);
      m_binangle_to_rational(q2, &b);
4725  ray = m_r_exray_in_new(q2, sharpness);
      i = 0;
      increment = 1;
      while (! t->cancelled && (m_failed != (success2 = m_r_exray_in_step(ray, ↙
          ↳ 64))))
      {
4730    progress = progress + increment;
        t->progress = progress / total;
        task_set_progress(t);
        // append point
        struct point *end = point_new(0);
4735    m_r_exray_in_get(ray, end->xy);
        a->u.wake.line_end->next = end;
        end->pred = a->u.wake.line_end;
        a->u.wake.line_end = end;
        if (++i >= sharpness)
4740    {
          i = 0;
          increment = increment + 1;
          if (increment >= depth)
          {
4745        success2 = m_converged;
            break;
          }
        }
        if (success2 == m_converged)
4750    {
          break;
        }
      }
```

```
        m_r_exray_in_delete(ray);
4755    mpq_init(a->u.wake.width);
        mpq_sub(a->u.wake.width, q2, q1);
        mpq_clear(q1);
        mpq_clear(q2);
        if (! t->cancelled && success1 == m_converged && success2 == m_converged)
4760    {
          g_async_queue_push(t->output, a);
        }
        else
        {
4765      free_annotation(a);
        }
        break;
      } // task_wake2

4770  case task_atom:
      {
        struct annotation *a = calloc(1, sizeof(*a));
        a->tag = annotation_atom;
        mpfr_prec_t prec = mpfr_get_prec(mpc_realref(t->u.atom.nucleus));
4775    a->label = malloc(100);
        snprintf(a->label, 99, "Atom p%d", t->u.atom.period);
        a->u.atom.fill_type = t->u.atom.fill_type;
        a->u.atom.fill_r = t->u.atom.fill_r;
        a->u.atom.fill_g = t->u.atom.fill_g;
4780    a->u.atom.fill_b = t->u.atom.fill_b;
        mpc_init2(a->u.atom.nucleus, prec);
        mpc_set(a->u.atom.nucleus, t->u.atom.nucleus, MPC_RNDNN);
        a->u.atom.period = t->u.atom.period;
        a->u.atom.shape = m_r_shape(a->u.atom.nucleus, a->u.atom.period);
4785    mpc_t z, c, interior;
        mpc_init2(interior, 53);
        mpc_init2(z, prec);
        mpc_init2(c, prec);
        mpc_set_dc(z, 0, MPC_RNDNN);
4790    mpc_set(c, a->u.atom.nucleus, MPC_RNDNN);
        for (int i = 0; i < 64; ++i)
        {
          if (t->cancelled) break;
          t->progress = (i + 0.5) / 64;
4795      task_set_progress(t);
          double s = (i + (a->u.atom.shape == m_circle ? 0.5 : 0)) * twopi / 64;
          mpc_set_dc(interior, cexp(s * I), MPC_RNDNN);
          m_r_interior(z, c, z, c, interior, a->u.atom.period, 64);
          a->u.atom.line_start = point_new(a->u.atom.line_start);
4800      if (i == 0)
          {
            a->u.atom.line_end = a->u.atom.line_start;
          }
          mpc_set_prec(a->u.atom.line_start->xy, prec);
4805      mpc_set(a->u.atom.line_start->xy, c, MPC_RNDNN);
        }
        mpc_clear(c);
        mpc_clear(z);
        mpc_clear(interior);
4810    if (t->cancelled)
```

```
      {
        free_annotation(a);
      }
      else
      {
        g_async_queue_push(t->output, a);
      }
      break;
    } // task_atom

    case task_domain:
    {
      struct annotation *a = calloc(1, sizeof(*a));
      a->tag = annotation_domain;
      mpfr_prec_t prec = mpfr_get_prec(mpc_realref(t->u.domain.nucleus));
      a->label = malloc(100);
      a->u.domain.line_type = t->u.domain.line_type;
      a->u.domain.line_r = t->u.domain.line_r;
      a->u.domain.line_g = t->u.domain.line_g;
      a->u.domain.line_b = t->u.domain.line_b;
      mpc_init2(a->u.domain.nucleus, prec);
      mpc_set(a->u.domain.nucleus, t->u.domain.nucleus, MPC_RNDNN);
      a->u.domain.period = t->u.domain.period;
      mpc_t c, domain;
      mpc_init2(domain, 53);
      mpc_init2(c, prec);
      mpc_set_dc(c, 0, MPC_RNDNN);
      mpfr_set_d(mpc_realref(domain), 1.0/0.0, MPFR_RNDN);
      a->u.domain.loperiod = t->u.domain.loperiod;
      if (a->u.domain.loperiod <= 0)
      {
        for (int i = 1; i < a->u.domain.period; ++i)
        {
          mpc_sqr(c, c, MPC_RNDNN);
          mpc_add(c, c, a->u.domain.nucleus, MPC_RNDNN);
          mpc_norm(mpc_imagref(domain), c, MPFR_RNDN);
          if (mpfr_less_p(mpc_imagref(domain), mpc_realref(domain)))
          {
            a->u.domain.loperiod = i;
            mpfr_set(mpc_realref(domain), mpc_imagref(domain), MPFR_RNDN);
          }
        }
      }
      snprintf(a->label, 99, "Domain %dp%d", a->u.domain.loperiod, a->u.domain.↙
          ↳ period);
      int sharpness = 64;
      for (int i = 0; i < sharpness; ++i)
      {
        if (t->cancelled) break;
        t->progress = (i + 0.5) / sharpness;
        task_set_progress(t);
        mpc_set(c, a->u.domain.nucleus, MPC_RNDNN);
        double _Complex s = cexp(((i + 0.5) / sharpness) * twopi * I);
        for (int j = 1; j <= sharpness; ++j)
        {
          mpc_set_dc(domain, j * s / sharpness, MPC_RNDNN);
          m_r_domain_coord(c, c, domain, a->u.domain.loperiod, a->u.domain.↙
```

```
                ↳ period, 16);
            if (j == sharpness)
            {
              a->u.domain.line_start = point_new(a->u.domain.line_start);
              if (i == 0)
              {
                a->u.domain.line_end = a->u.domain.line_start;
              }
              mpc_set_prec(a->u.domain.line_start->xy, prec);
              mpc_set(a->u.domain.line_start->xy, c, MPC_RNDNN);
            }
          }
        }
        mpc_clear(c);
        mpc_clear(domain);
        if (t->cancelled)
        {
          free_annotation(a);
        }
        else
        {
          g_async_queue_push(t->output, a);
        }
        break;
      } // task_atom

    }
    task_done(t);
  }
```

8 c/bin/m-perturbator-offline.c

```
// mandelbrot-perturbator -- efficient deep zooming for Mandelbrot sets
// Copyright (C) 2015,2016,2017 Claude Heiland-Allen
// License GPL3+ http://www.gnu.org/licenses/gpl.html

#define _POSIX_C_SOURCE 199309L

#include <math.h>
#include <assert.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include <mpfr.h>
#include <mpc.h>

#include <mandelbrot-perturbator.h>

static inline int max(int a, int b) {
  return a > b ? a : b;
}

static int envi(const char *name, int def) {
```

```
  const char *e = getenv(name);
  if (e) {
    return atoi(e);
  } else {
    return def;
  }
}

static double envd(const char *name, double def) {
  const char *e = getenv(name);
  if (e) {
    return atof(e);
  } else {
    return def;
  }
}

static int envr(mpfr_t out, const char *name, const char *def) {
  const char *e = getenv(name);
  if (e) {
    return mpfr_set_str(out, e,   10, MPFR_RNDN);
  } else {
    return mpfr_set_str(out, def, 10, MPFR_RNDN);
  }
}

extern int main(int argc, char **argv) {
  int workers = envi("threads", 4);
  int width = envi("width", 1280);
  int height = envi("height", 720);
  int detect_glitches = envi("detect_glitches", 1);
  int approx_skip = envi("approx_skip", 0);
  int maxiters = envi("maxiters", 1 << 18);
  int chunk = envi("chunk", 1 << 8);
  double escape_radius = envd("escaperadius", 25);
  double glitch_threshold = envd("glitchthreshold", 1e-6);
  int precision = envi("precision", 53);
  mpfr_t radius, centerx, centery;
  mpfr_init2(radius, 53);
  envr(radius, "radius", "2.0");
  int e = max(53, 53 - mpfr_get_exp(radius));
  if (e > precision) {
    fprintf(stderr, "WARNING: increasing precision to %d\n", e);
    precision = e;
  }
  mpfr_init2(centerx, precision);
  mpfr_init2(centery, precision);
  envr(centerx, "real", "-0.75");
  envr(centery, "imag", "0.0");
  struct perturbator *context = perturbator_new(workers, width, height, maxiters
    ↳ , chunk, escape_radius, glitch_threshold);
  perturbator_set_detect_glitches(context, detect_glitches);
  perturbator_set_approx_skip(context, approx_skip);
  perturbator_start(context, centerx, centery, radius);
  perturbator_stop(context, false);
  fwrite(perturbator_get_dwell_n(context), width * height * sizeof(int32_t), 1,
    ↳ stdout);
```

```
  fwrite(perturbator_get_dwell_f(context), width * height * sizeof(float), 1, ↙
      ↳ stdout);
  fwrite(perturbator_get_dwell_a(context), width * height * sizeof(float), 1, ↙
      ↳ stdout);
  fwrite(perturbator_get_distance(context), width * height * sizeof(float), 1, ↙
      ↳ stdout);
  fwrite(perturbator_get_domain_n(context), width * height * sizeof(int32_t), 1,↙
      ↳  stdout);
  fwrite(perturbator_get_domain_x(context), width * height * sizeof(float), 1, ↙
      ↳ stdout);
  fwrite(perturbator_get_domain_y(context), width * height * sizeof(float), 1, ↙
      ↳ stdout);
  fflush(stdout);
  return 0;
(void) argc;
(void) argv;
}
```

9 c/include/mandelbrot-perturbator.h

```
// mandelbrot-perturbator -- efficient deep zooming for Mandelbrot sets
// Copyright (C) 2015-2019 Claude Heiland-Allen
// License GPL3+ http://www.gnu.org/licenses/gpl.html

#ifndef MANDELBROT_PERTURBATOR_H
#define MANDELBROT_PERTURBATOR_H 1

#include <mpfr.h>

#ifdef __cplusplus
extern "C" {
#endif

struct perturbator;
struct perturbator *perturbator_new(int workers, int width, int height, int ↙
    ↳ maxiters, int maxchunk, double escape_radius, double glitch_threshold);
// void perturbator_delete(struct perturbator *context);
void perturbator_set_maximum_iterations(struct perturbator *context, int ↙
    ↳ maxiters);
void perturbator_set_detect_glitches(struct perturbator *context, int ↙
    ↳ detect_glitches);
void perturbator_set_approx_skip(struct perturbator *context, int approx_skip);

// rendering control
void perturbator_start(struct perturbator *context, const mpfr_t x, const mpfr_t↙
    ↳  y, const mpfr_t r);
void perturbator_stop(struct perturbator *context, int force);
bool perturbator_wait(struct perturbator *context, const struct timespec *↙
    ↳ waituntil);
int perturbator_active(struct perturbator *context);
double perturbator_get_progress(struct perturbator *context);

// get output data
int perturbator_get_width(struct perturbator *context);
int perturbator_get_height(struct perturbator *context);
// pointers are valid until perturbator is deleted, contents are updated during ↙
    ↳ rendering
```

```
const int32_t *perturbator_get_dwell_n (struct perturbator *context);
const float   *perturbator_get_dwell_f (struct perturbator *context);
const float   *perturbator_get_dwell_a (struct perturbator *context);
const float   *perturbator_get_distance(struct perturbator *context);
const int32_t *perturbator_get_domain_n(struct perturbator *context);
const float   *perturbator_get_domain_x(struct perturbator *context);
const float   *perturbator_get_domain_y(struct perturbator *context);

int perturbator_get_primary_reference(struct perturbator *context, mpfr_t x, mpfr_t y);

#ifdef __cplusplus
}
#endif

#endif
```

10 c/lib/complex.hpp

```
// mandelbrot-perturbator -- efficient deep zooming for Mandelbrot sets
// Copyright (C) 2015,2016,2017 Claude Heiland-Allen
// License GPL3+ http://www.gnu.org/licenses/gpl.html

#ifndef MANDELBROT_PERTURBATOR_COMPLEX_H
#define MANDELBROT_PERTURBATOR_COMPLEX_H 1

#include <cmath>

template <typename R>
struct complex
{
  R re, im;
  inline complex(const complex<R> &c) : re(c.re), im(c.im) { };
  inline complex(const R &r, const R &i) : re(r), im(i) { };
  inline complex(const R &r) : re(r), im(0) { };
  inline complex(const int &r) : re(r), im(0) { };
  inline complex() : re(0), im(0) { };
  inline complex<R> &operator+=(const complex<R> &y)
  {
    return *this = *this + y;
  };
  inline complex<R> &operator*=(const complex<R> &y)
  {
    return *this = *this * y;
  };
};

template<typename R>
inline const R real(const complex<R> &x) { return x.re; }

template<typename R>
inline const R imag(const complex<R> &x) { return x.im; }

template<typename R>
inline const R norm(const complex<R> &x) { return x.re * x.re + x.im * x.im; }

template<typename R>
```

```
inline const R arg(const complex<R> &x) { return std::atan2(x.im, x.re); }

template<typename R>
inline complex<R> operator-(const complex<R> &x)
{
  return complex<R>(-x.re, -x.im);
}

template<typename R>
inline complex<R> operator-(const int &y, const complex<R> &x)
{
  return complex<R>(y - x.re, -x.im);
}

template<typename R>
inline complex<R> operator-(const complex<R> &x, const complex<R> &y)
{
  return complex<R>(x.re - y.re, x.im - y.im);
}

template<typename R>
inline complex<R> operator-(const complex<R> &x, const int &y)
{
  return complex<R>(x.re - y, x.im);
}

template<typename R>
inline complex<R> operator+(const complex<R> &x, const int &y)
{
  return complex<R>(x.re + y, x.im);
}

template<typename R>
inline complex<R> operator+(const complex<R> &x, const R &y)
{
  return complex<R>(x.re + y, x.im);
}

template<typename R>
inline complex<R> operator+(const complex<R> &x, const complex<R> &y)
{
  return complex<R>(x.re + y.re, x.im + y.im);
}

template<typename R>
inline complex<R> operator*(const int &x, const complex<R> &y)
{
  return complex<R>(x * y.re, x * y.im);
}

template<typename R>
inline complex<R> operator*(const R &x, const complex<R> &y)
{
  return complex<R>(x * y.re, x * y.im);
}

template<typename R>
```

```
inline complex<R> operator*(const complex<R> &y, const R &x)
{
  return complex<R>(x * y.re, x * y.im);
}

template<typename R>
inline complex<R> operator/(const complex<R> &y, const R &x)
{
  return complex<R>(y.re / x, y.im / x);
}

template<typename R>
inline complex<R> operator*(const complex<R> &x, const complex<R> &y)
{
  return complex<R>(x.re * y.re - x.im * y.im, x.re * y.im + x.im * y.re);
}

template<typename R>
inline complex<R> operator/(const complex<R> &x, const complex<R> &y)
{
  return complex<R>(x.re * y.re + x.im * y.im, -x.re * y.im + x.im * y.re) / ↙
      ↳ norm(y);
}

#endif
```

11 c/lib/edouble.cc

```
// mandelbrot-perturbator -- efficient deep zooming for Mandelbrot sets
// Copyright (C) 2015,2016,2017 Claude Heiland-Allen
// License GPL3+ http://www.gnu.org/licenses/gpl.html

#ifndef MANDELBROT_EDOUBLE_CC
#define MANDELBROT_EDOUBLE_CC 1

#include <cmath>
#include <algorithm>
#include <iostream>

#include <mpfr.h>

#include "complex.hpp"

// ldexp is ~2x slower than scaling via table of powers of two
//#define EDOUBLE_USE_LDEXP
// union hacking is 10% faster than scaling via table
#define EDOUBLE_USE_UNION
union idouble { double d; int64_t i; };

// not checking for 0 or inf or nan or overflow or underflow is ~35% faster...
// but it's unsafe so disable it for forseeable future
//#define EDOUBLE_UNSAFE

// annotating branch prediction makes addition ~10% faster
#define unlikely(x) __builtin_expect(x,false)

inline double sign(double x) {
```

```
30    return (x > 0) - (x < 0);
    }

    class edouble {
    private:
35    static const int64_t maxexponent = (1LL << 60) - 2;
      static const int64_t minexponent = 2 - (1LL << 60);
      static const int64_t supexponent = maxexponent - 2000;
      static const int64_t infexponent = minexponent + 2080;
      double x;
40    int64_t e;
    public:
      static const double scaling[129];
      inline edouble() : x(0), e(0) { };
      inline edouble(const edouble &that) : x(that.x), e(that.e) { };
45    inline edouble(const double &x0, const int64_t &e0) {
    #ifndef EDOUBLE_UNSAFE
        if (unlikely(! std::isnormal(x0))) {
          x = (std::isnan(x0) || std::isinf(x0)) ? x0 : 0.0; // squash subnormals
          e = 0;
50      } else {
    #endif
    #ifdef EDOUBLE_USE_UNION
          idouble f;
          f.d = x0;
55        int64_t e1(((f.i & 0x7FF0000000000000LL)>>52) - 1022LL);
          f.i = (f.i & 0x800FFFFFFFFFFFFFLL) | 0x3FE0000000000000LL;
          double x1(f.d);
    #else
          int tmp(0);
60        double x1(std::frexp(x0, &tmp));
          int64_t e1(tmp);
    #endif
          e1 += e0;
    #ifndef EDOUBLE_UNSAFE
65        if (unlikely(e0 > supexponent || e1 > maxexponent)) {
            x = sign(x0) / 0.0; // infinity with sign
            e = 0;
          } else if (unlikely(e0 < infexponent || e1 < minexponent)) {
            x = sign(x0) * 0.0; // 0 with sign
70          e = 0;
          } else {
    #endif
            x = x1;
            e = e1;
75  #ifndef EDOUBLE_UNSAFE
          }
        }
    #endif
      };
80    inline edouble(const long double &x0, const int64_t &e0) {
        if (unlikely(! std::isnormal(x0))) {
          x = (std::isnan(x0) || std::isinf(x0)) ? x0 : 0.0; // squash subnormals
          e = 0;
        } else {
85        int tmp(0);
          long double x1(frexp(x0, &tmp));
```

```
      int64_t e1(tmp);
      e1 += e0;
      if (unlikely(e0 > supexponent || e1 > maxexponent)) {
        x = sign(x0) / 0.0;
        e = 0;
      } else if (unlikely(e0 < infexponent || e1 < minexponent)) {
        x = sign(x0) * 0.0;
        e = 0;
      } else {
        x = x1;
        e = e1;
      }
    }
  };
  inline edouble(const double &x) : edouble(x, 0) { };
  inline edouble(const long double &x) : edouble(x, 0) { };
  inline explicit edouble(const mpfr_t &a) {
    long e(0);
    double x(mpfr_get_d_2exp(&e, a, MPFR_RNDN));
    *this = edouble(x, e);
  }
  inline edouble(const int &x) : edouble(double(x)) { };
  inline ~edouble() { };
  inline int64_t exponent() const { return e; };
  inline void to_mpfr(mpfr_t &m) {
    mpfr_set_d(m, x, MPFR_RNDN);
    mpfr_mul_2si(m, m, e, MPFR_RNDN);
  };
  inline long double to_ld() {
    int tmp(e);
    if (unlikely(e > int64_t(tmp))) {
      return sign(x) / 0.0;
    }
    if (unlikely(e < int64_t(tmp))) {
      return sign(x) * 0.0;
    }
    return std::ldexp((long double) x, tmp);
  }
  friend inline bool operator==(const edouble &a, const edouble &b);
  friend inline bool operator!=(const edouble &a, const edouble &b);
  friend inline int compare(const edouble &a, const edouble &b);
  friend inline int sign(const edouble &a);
  friend inline edouble abs(const edouble &a);
  friend inline edouble sqrt(const edouble &a);
  friend inline edouble operator+(const edouble &a, const edouble &b);
  friend inline edouble operator-(const edouble &a);
  friend inline edouble operator-(const edouble &a, const edouble &b);
  friend inline edouble operator*(const edouble &a, const edouble &b);
  friend inline edouble recip(const edouble &a);
  friend inline edouble operator/(const edouble &a, const edouble &b);
  friend inline bool isnan(const edouble &a);
  friend inline bool isinf(const edouble &a);
  friend inline edouble ldexp(const edouble &a, int64_t e);
  friend inline std::ostream& operator<<(std::ostream& o, const edouble& a);
  friend inline std::istream& operator>>(std::istream& i, edouble& a);
  inline edouble &operator+=(const edouble &a) {
    *this = *this + a;
```

```
    return *this;
  };
  inline edouble &operator-=(const edouble &a) {
    *this = *this - a;
    return *this;
  };
  inline edouble &operator*=(const edouble &a) {
    *this = *this * a;
    return *this;
  };
  inline edouble &operator/=(const edouble &a) {
    *this = *this / a;
    return *this;
  };
};
// (0:) . take 128 . map (min 1) . iterate (* 2) $ 2 ^^ negate 128
const double edouble::scaling[129] = {0.0,5.877471754111438e
    ↘ -39,1.1754943508222875e-38,2.350988701644575e-38,4.70197740328915e
    ↘ -38,9.4039548065783e-38,1.88079096131566e-37,3.76158192263132e
    ↘ -37,7.52316384526264e-37,1.504632769052528e-36,3.009265538105056e
    ↘ -36,6.018531076210112e-36,1.2037062152420224e-35,2.407412430484045e
    ↘ -35,4.81482486096809e-35,9.62964972193618e-35,1.925929944387236e
    ↘ -34,3.851859888774472e-34,7.703719777548943e-34,1.5407439555097887e
    ↘ -33,3.0814879110195774e-33,6.162975822039155e-33,1.232595164407831e
    ↘ -32,2.465190328815662e-32,4.930380657631324e-32,9.860761315262648e
    ↘ -32,1.9721522630525295e-31,3.944304526105059e-31,7.888609052210118e
    ↘ -31,1.5777218104420236e-30,3.1554436208840472e-30,6.310887241768095e
    ↘ -30,1.262177448353619e-29,2.524354896707238e-29,5.048709793414476e
    ↘ -29,1.0097419586828951e-28,2.0194839173657902e-28,4.0389678347315804e
    ↘ -28,8.077935669463161e-28,1.6155871338926322e-27,3.2311742677852644e
    ↘ -27,6.462348535570529e-27,1.2924697071141057e-26,2.5849394142282115e
    ↘ -26,5.169878828456423e-26,1.0339757656912846e-25,2.0679515313825692e
    ↘ -25,4.1359030627651384e-25,8.271806125530277e-25,1.6543612251060553e
    ↘ -24,3.308722450212111e-24,6.617444900424222e-24,1.3234889800848443e
    ↘ -23,2.6469779601696886e-23,5.293955920339377e-23,1.0587911840678754e
    ↘ -22,2.117582368135751e-22,4.235164736271502e-22,8.470329472543003e
    ↘ -22,1.6940658945086007e-21,3.3881317890172014e-21,6.776263578034403e
    ↘ -21,1.3552527156068805e-20,2.710505431213761e-20,5.421010862427522e
    ↘ -20,1.0842021724855044e-19,2.168404344971009e-19,4.336808689942018e
    ↘ -19,8.673617379884035e-19,1.734723475976807e-18,3.469446951953614e
    ↘ -18,6.938893903907228e-18,1.3877787807814457e-17,2.7755575615628914e
    ↘ -17,5.551115123125783e-17,1.1102230246251565e-16,2.220446049250313e
    ↘ -16,4.440892098500626e-16,8.881784197001252e-16,1.7763568394002505e
    ↘ -15,3.552713678800501e-15,7.105427357601002e-15,1.4210854715202004e
    ↘ -14,2.842170943040401e-14,5.684341886080802e-14,1.1368683772161603e
    ↘ -13,2.2737367544323206e-13,4.547473508864641e-13,9.094947017729282e
    ↘ -13,1.8189894035458565e-12,3.637978807091713e-12,7.275957614183426e
    ↘ -12,1.4551915228366852e-11,2.9103830456733704e-11,5.820766091346741e
    ↘ -11,1.1641532182693481e-10,2.3283064365386963e-10,4.656612873077393e
    ↘ -10,9.313225746154785e-10,1.862645149230957e-9,3.725290298461914e
    ↘ -9,7.450580596923828e-9,1.4901161193847656e-8,2.9802322387695313e
    ↘ -8,5.960464477539063e-8,1.1920928955078125e-7,2.384185791015625e
    ↘ -7,4.76837158203125e-7,9.5367431640625e-7,1.9073486328125e
    ↘ -6,3.814697265625e-6,7.62939453125e-6,1.52587890625e-5,3.0517578125e
    ↘ -5,6.103515625e-5,1.220703125e-4,2.44140625e-4,4.8828125e-4,9.765625e
    ↘ -4,1.953125e-3,3.90625e-3,7.8125e-3,1.5625e-2,3.125e-2,6.25e
    ↘ -2,0.125,0.25,0.5,1.0};
```

```
inline bool operator==(const edouble &a, const edouble &b) {
  return a.e == b.e && a.x == b.x;
}

inline bool operator!=(const edouble &a, const edouble &b) {
  return a.e != b.e || a.x != b.x;
}

inline int compare(const double &a, const double &b) {
  return sign(a - b);
}

inline int compare(const edouble &a, const edouble &b) {
#ifndef EDOUBLE_UNSAFE
  if (unlikely(a.x == 0.0)) {
    return -sign(b.x);
  }
  if (unlikely(b.x == 0.0)) {
    return sign(a.x);
  }
#endif
  int64_t e(std::max(a.e, b.e));
  int64_t da(a.e - e);
  int64_t db(b.e - e);
#ifndef EDOUBLE_UNSAFE
  int ia(da);
  int ib(db);
  if (unlikely(int64_t(ia) != da)) {
    // a -> 0
    return -sign(b.x);
  }
  if (unlikely(int64_t(ib) != db)) {
    // b -> 0
    return sign(a.x);
  }
#endif
#ifdef EDOUBLE_USE_LDEXP
  return compare(std::ldexp(a.x, ia), std::ldexp(b.x, ib));
#else
#ifdef EDOUBLE_USE_UNION
  double ax = a.x;
  if (0 > da)
  {
    if (da > -1000)
    {
      idouble f;
      f.d = ax;
      f.i = (f.i & 0x800FFFFFFFFFFFFFLL) | ((1022LL + da) << 52);
      ax = f.d;
    }
    else
      ax = 0.0;
  }
  double bx = b.x;
  if (0 > db)
  {
```

```
    if (db > -1000)
    {
      idouble f;
      f.d = bx;
      f.i = (f.i & 0x800FFFFFFFFFFFFLL) | ((1022LL + db) << 52);
      bx = f.d;
    }
    else
      bx = 0.0;
  }
  return compare(ax, bx);
#else
  double sa = edouble::scaling[std::max(da + 128, 0L)];
  double sb = edouble::scaling[std::max(db + 128, 0L)];
  return compare(a.x * sa, b.x * sb);
#endif
#endif
}

inline bool operator<(const edouble &a, const edouble &b) {
  return compare(a, b) < 0;
}

inline bool operator<=(const edouble &a, const edouble &b) {
  return compare(a, b) <= 0;
}

inline bool operator>(const edouble &a, const edouble &b) {
  return compare(a, b) > 0;
}

inline bool operator>=(const edouble &a, const edouble &b) {
  return compare(a, b) >= 0;
}

inline int sign(const edouble &a) {
  return sign(a.x);
}

inline edouble abs(const edouble &a) {
  return { std::abs(a.x), a.e };
}

inline edouble sqrt(const edouble &a) {
  return { std::sqrt((a.e & 1) ? 2.0 * a.x : a.x), (a.e & 1) ? (a.e - 1) / 2 : a↙
      ↳ .e / 2 };
}

inline edouble operator+(const edouble &a, const edouble &b) {
#ifndef EDOUBLE_UNSAFE
  if (unlikely(a.x == 0.0)) {
    return b;
  }
  if (unlikely(b.x == 0.0)) {
    return a;
  }
#endif
```

```
  int64_t e(std::max(a.e, b.e));
  int64_t da(a.e - e);
  int64_t db(b.e - e);
#ifndef EDOUBLE_UNSAFE
  int ia(da);
  int ib(db);
  if (unlikely(int64_t(ia) != da)) {
    // a -> 0
    return b;
  }
  if (unlikely(int64_t(ib) != db)) {
    // b -> 0
    return a;
  }
#endif
#ifdef EDOUBLE_USE_LDEXP
  return edouble(std::ldexp(a.x, ia) + std::ldexp(b.x, ib), e);
#else
#ifdef EDOUBLE_USE_UNION
  double ax = a.x;
  if (0 > da)
  {
    if (da > -1000)
    {
      idouble f;
      f.d = ax;
      f.i = (f.i & 0x800FFFFFFFFFFFFFLL) | ((1022LL + da) << 52);
      ax = f.d;
    }
    else
      ax = 0.0;
  }
  double bx = b.x;
  if (0 > db)
  {
    if (db > -1000)
    {
      idouble f;
      f.d = bx;
      f.i = (f.i & 0x800FFFFFFFFFFFFFLL) | ((1022LL + db) << 52);
      bx = f.d;
    }
    else
      bx = 0.0;
  }
  return edouble(ax + bx, e);
#else
  double sa = edouble::scaling[std::max(da + 128, 0L)];
  double sb = edouble::scaling[std::max(db + 128, 0L)];
  return edouble(a.x * sa + b.x * sb, e);
#endif
#endif
}

inline edouble operator-(const edouble &a) {
  return { -a.x, a.e };
}
```

```
inline edouble operator-(const edouble &a, const edouble &b) {
  return a + (-b);
}

inline edouble operator*(const edouble &a, const edouble &b) {
  return edouble(a.x * b.x, a.e + b.e);
}

inline double recip(const double &a) {
  return 1.0 / a;
}

inline edouble recip(const edouble &a) {
  return edouble(recip(a.x), -a.e);
}

inline edouble operator/(const edouble &a, const edouble &b) {
  return edouble(a.x / b.x, a.e - b.e);
}

inline bool isnan(const edouble &a) {
  using std::isnan;
  return isnan(a.x);
}

inline bool isinf(const edouble &a) {
  using std::isinf;
  return isinf(a.x);
}

inline edouble ldexp(const edouble &a, int64_t e) {
  return edouble(a.x, a.e + e);
}

inline void to_mpfr(float from, mpfr_t &to) {
  mpfr_set_flt(to, from, MPFR_RNDN);
}

inline void to_mpfr(double from, mpfr_t &to) {
  mpfr_set_d(to, from, MPFR_RNDN);
}

inline void to_mpfr(long double from, mpfr_t &to) {
  mpfr_set_ld(to, from, MPFR_RNDN);
}

inline void to_mpfr(edouble from, mpfr_t &to) {
  from.to_mpfr(to);
}

inline long double to_ld(float x) {
  return x;
}

inline long double to_ld(double x) {
  return x;
```

```
}

inline long double to_ld(long double x) {
  return x;
}

inline long double to_ld(edouble x) {
  return x.to_ld();
}

template <typename S, typename T>
inline T to_R(S x, const T &dummy) {
  (void) dummy;
  return T(to_ld(x));
}

inline edouble to_R(edouble x, const edouble &dummy) {
  (void) dummy;
  return x;
}

template <typename S, typename T>
inline complex<T> to_C(complex<S> x, const T &dummy) {
  return complex<T>(to_R(real(x), dummy), to_R(imag(x), dummy));
}

inline complex<edouble> to_C(complex<edouble> x, const edouble &dummy) {
  (void) dummy;
  return x;
}

inline std::ostream& operator<<(std::ostream& o, const edouble& a) {
  return o << a.x << " " << a.e;
}

inline std::istream& operator>>(std::istream& i, edouble& a) {
  double x;
  int64_t e;
  i >> x >> e;
  a = edouble(x, e);
  return i;
}

int64_t exponent(float z)
{
  int e;
  frexp(z, &e);
  return e;
}

int64_t exponent(double z)
{
  int e;
  frexp(z, &e);
  return e;
}
```

```
445 int64_t exponent(long double z)
    {
      int e;
      frexpl(z, &e);
      return e;
450 }

    int64_t exponent(edouble z)
    {
      return z.exponent();
455 }

    int64_t exponent(const mpfr_t z)
    {
      if (mpfr_regular_p(z))
460     return mpfr_get_exp(z);
      return 0;
    }

    bool isfinite(edouble z)
465 {
      return !(isinf(z) || isnan(z));
    }

    #ifdef EDOUBLE_STANDALONE
470
    #include <cstdlib>
    #include <cstdio>

    #include "floatexp.h"
475
    edouble erandom()
    {
      int r = std::rand();
      return edouble((1.0 + (r / (double) RAND_MAX)) * ((r & 1) - 0.5), (r & 511) - ↙
          ↳ 256);
480 }

    floatexp frandom()
    {
      int r = std::rand();
485   return floatexp((1.0 + (r / (double) RAND_MAX)) * ((r & 1) - 0.5), (r & 511) -↙
          ↳ 256);
    }

    int main() {
    #if 0
490   mpfr_t m;
      mpfr_init2(m, 53);
      edouble x(2);
      do {
        std::cout << x << std::endl;
495     x.to_mpfr(m);
        mpfr_printf("%Re\n", m);
        x *= x;
      } while (! isinf(x));
      mpfr_clear(m);
```

```
#endif
#define N (1<<14)
  srand(0xDe4dF00d);
  edouble r[N];
  for (int i = 0; i < N; ++i)
    r[i] = erandom();
  printf("+ edouble\n"); fflush(stdout);
  edouble esum(0);
  for (int i = 0; i < N; ++i)
    for (int j = 0; j < N; ++j)
      esum += r[i] + r[j];
  std::cerr << esum << std::endl;
  printf("* edouble\n"); fflush(stdout);
  edouble eprod(1);
  for (int i = 0; i < N; ++i)
    for (int j = 0; j < N; ++j)
      eprod *= r[i] * r[j];
  std::cerr << eprod << std::endl;

  srand(0xDe4dF00d);
  floatexp s[N];
  for (int i = 0; i < N; ++i)
    s[i] = frandom();
  printf("+ floatexp\n"); fflush(stdout);
  floatexp fsum(0.0);
  std::cerr << fsum.val << " " << fsum.exp << std::endl;
  for (int i = 0; i < N; ++i)
    for (int j = 0; j < N; ++j)
      fsum += s[i] + s[j];
  std::cerr << fsum.val << " " << fsum.exp << std::endl;
  printf("* floatexp\n"); fflush(stdout);
  floatexp fprod(1);
  for (int i = 0; i < N; ++i)
    for (int j = 0; j < N; ++j)
      fprod *= s[i] * s[j];
  std::cerr << fprod.val << " " << fprod.exp << std::endl;
  printf("DONE\n"); fflush(stdout);
  return 0;
}

#endif

#endif
```

12 c/lib/floatexp.h

```
#define _GCC_
#define BOOL bool
#define FALSE false
#define TRUE true
#define __int64 int64_t
#define __inline inline
#include <string.h>
#if 0
#include <windows.h>
#endif
#include <math.h>
```

```
#ifndef _GCC_
#include "CFixedFloat.h"
#endif
#ifndef __FLOATEXP_H__
#define __FLOATEXP_H__
#define MAX_PREC 1020

#ifdef _GCC_
#define _ALIGN_(val,exp) exp += ((*((__int64*)&val) & 0x7FF0000000000000LL)>>52) - 1023; *((__int64*)&val) = (*((__int64*)&val) & 0x800FFFFFFFFFFFFFLL) | 0x3FF0000000000000LL;
#else
#define _ALIGN_(val,exp) exp += ((*((__int64*)&val) & 0x7FF0000000000000)>>52) - 1023; *((__int64*)&val) = (*((__int64*)&val) & 0x800FFFFFFFFFFFFF) | 0x3FF0000000000000;
#endif
class floatexp
{
public:
        double val;
        __int64 exp;
        floatexp &abs()
        {
                if(val<0)
                        val=-val;
                return *this;
        }
        __inline void initFromDouble(double a)
        {
                val=a;
                exp=0;
                _ALIGN_(val,exp)
        }
        __inline void align()
        {
#ifdef _GCC_
                exp += ((*((__int64*)&val) & 0x7FF0000000000000LL)>>52) - 1023;
#else
                exp += ((*((__int64*)&val) & 0x7FF0000000000000)>>52) - 1023;
#endif
//              __int64 tmpval = (*((__int64*)&val) & 0x800FFFFFFFFFFFFF) | 0x3FF0000000000000;
//              memcpy(&val,&tmpval,sizeof(double));
#ifdef _GCC_
                *((__int64*)&val) = (*((__int64*)&val) & 0x800FFFFFFFFFFFFFLL) | 0x3FF0000000000000LL;
#else
                *((__int64*)&val) = (*((__int64*)&val) & 0x800FFFFFFFFFFFFF) | 0x3FF0000000000000;
#endif
        }
        __inline double setExp(double newval,__int64 newexp)
        {
//              __int64 tmpval = (*((__int64*)&newval) & 0x800FFFFFFFFFFFFF) | ((newexp+1023)<<52);
//              memcpy(&newval,&tmpval,sizeof(double));
//              return newval;
```

```
#ifdef _GCC_
                *((__int64*)&newval) = (*((__int64*)&newval) & 0x800FFFFFFFFFFFFFLL) | ((newexp+1023)<<52);
#else
                *((__int64*)&newval) = (*((__int64*)&newval) & 0x800FFFFFFFFFFFFF) | ((newexp+1023)<<52);
#endif
                return newval;
        }
        floatexp()
        {
                val = 0;
                exp = 0;
        }
        floatexp(double a)
        {
                initFromDouble(a);
        }
        floatexp(double a, __int64 e)
        {
                initFromDouble(a);
                exp += e;
        }
        __inline floatexp &operator =(const floatexp &a)
        {
                val=a.val;
                exp=a.exp;
                return *this;
        }
        __inline floatexp &operator =(int a)
        {
                initFromDouble((double)a);
                return *this;
        }
        __inline floatexp &operator =(double a)
        {
                initFromDouble(a);
                return *this;
        }
        __inline floatexp operator *(const floatexp &a)
        {
                floatexp r;
                r.val = a.val*val;
                r.exp = a.exp+exp;
                _ALIGN_(r.val,r.exp)
                return r;
        }
        __inline floatexp operator /(const floatexp &a)
        {
                floatexp r;
                r.val = val/a.val;
                r.exp = exp - a.exp;
                _ALIGN_(r.val,r.exp)
                return r;
        }
        __inline floatexp operator +(const floatexp &a)
        {
```

```
                floatexp r;
                __int64 diff;
                if(exp>a.exp){
                        diff = exp-a.exp;
                        r.exp = exp;
                        if(diff>MAX_PREC)
                                r.val=val;
                        else{
                                double aval = setExp(a.val,-diff);
                                r.val = val+aval;
                        }
                }
                else{
                        diff = a.exp-exp;
                        r.exp = a.exp;
                        if(diff>MAX_PREC)
                                r.val=a.val;
                        else{
                                double aval = setExp(val,-diff);
                                r.val = a.val+aval;
                        }
                }
                _ALIGN_(r.val,r.exp)
                return r;
        }
        __inline floatexp operator -()
        {
                floatexp r=*this;
                r.val=-r.val;
                return r;
        }
        __inline floatexp &operator +=(const floatexp &a)
        {
#ifdef _GCC_
                floatexp r = *this+a;
                *this = r;
#else
                *this = *this+a;
#endif
                return *this;
        }
        __inline floatexp &operator *=(const floatexp &a)
        {
#ifdef _GCC_
                floatexp r = *this*a;
                *this = r;
#else
                *this = *this*a;
#endif
                return *this;
        }
        __inline floatexp operator -(const floatexp &a)
        {
                floatexp r;
                __int64 diff;
                if(exp>a.exp){
                        diff = exp-a.exp;
```

```
                        r.exp = exp;
                        if(diff>MAX_PREC)
                                r.val = val;
                        else{
                                double aval = setExp(a.val,-diff);
                                r.val = val-aval;
                        }
                }
                else{
                        diff = a.exp-exp;
                        r.exp = a.exp;
                        if(diff>MAX_PREC)
                                r.val=-a.val;
                        else{
                                double aval = setExp(val,-diff);
                                r.val = aval-a.val;
                        }
                }
                _ALIGN_(r.val,r.exp)
                return r;
        }
        __inline floatexp &operator -=(const floatexp &a)
        {
#ifdef _GCC_
                floatexp r = *this-a;
                *this = r;
#else
                *this = *this-a;
#endif
                return *this;
        }
        __inline BOOL operator >(const floatexp &a)
        {
                if(val>0){
                        if(a.val<0)
                                return TRUE;
                        if(exp>a.exp)
                                return TRUE;
                        else if(exp<a.exp)
                                return FALSE;
                        return val>a.val;
                }
                else{
                        if(a.val>0)
                                return FALSE;
                        if(exp>a.exp)
                                return FALSE;
                        else if(exp<a.exp)
                                return TRUE;
                        return val>a.val;
                }
        }
        __inline BOOL operator <(const floatexp &a)
        {
                if(val>0){
                        if(a.val<0)
                                return FALSE;
```

```
230                     if(exp>a.exp)
                                return FALSE;
                        else if(exp<a.exp)
                                return TRUE;
                        return val<a.val;
235             }
                else{
                        if(a.val>0)
                                return TRUE;
                        if(exp>a.exp)
240                             return TRUE;
                        else if(exp<a.exp)
                                return FALSE;
                        return val<a.val;
                }
245     }
        __inline BOOL operator <=(const int a)
        {
                floatexp aa(a);
                return (*this<a || *this==a);
250     }
        __inline BOOL operator ==(const floatexp &a)
        {
                if(exp!=a.exp)
                        return FALSE;
255             return val==a.val;
        }
        bool iszero()
        {
                return (val==0 && exp==0);
260     }
        double todouble()
        {
                if(exp<-MAX_PREC || exp>MAX_PREC)
                        return 0;
265             return setExp(val,exp);
        }
        double todouble(int nScaling)
        {
                if(!nScaling)
270                     return todouble();
                floatexp ret = *this;
                while(nScaling>9){
                        ret.val*=1e10;
                        _ALIGN_(ret.val,ret.exp)
275                     nScaling-=10;
                }
                while(nScaling>2){
                        ret.val*=1e3;
                        _ALIGN_(ret.val,ret.exp)
280                     nScaling-=3;
                }
                while(nScaling){
                        ret.val*=1e1;
                        _ALIGN_(ret.val,ret.exp)
285                     nScaling--;
                }
```

```
                if(ret.exp<-MAX_PREC || ret.exp>MAX_PREC)
                        return 0;
                return setExp(ret.val,ret.exp);
        }
        __inline floatexp &operator /=(double a)
        {
                val/=a;
                _ALIGN_(val,exp)
                return *this;
        }
        __inline floatexp &operator *=(double a)
        {
                val*=a;
                _ALIGN_(val,exp)
                return *this;
        }
#ifndef _GCC_
        __inline floatexp &operator =(const CFixedFloat &a)
        {
                exp=0;
                val=0;

                floatexp startexp=1;
                floatexp partmax = FIXEDFLOAT_PARTMAX;
                int nStart = 0;
                for(nStart=0;nStart<a.m_nValues;nStart++){
                        if(a.m_pValues[nStart])
                                break;
                        startexp=startexp/partmax;
                }
                if(nStart>a.m_nValues)
                        nStart=a.m_nValues;
                int n=nStart+24/FIXEDFLOAT_DIGITS;
                if(n>a.m_nValues-1)
                        n=a.m_nValues-1;
                for(;n>=nStart && n>=0;n--)
                        val = val/FIXEDFLOAT_PARTMAX + a.m_pValues[n];
                _ALIGN_(val,exp)
                *this = *this * startexp;

                if(a.m_bSign)
                        val = -val;
                return *this;
        }
#else
        __inline floatexp setLongDouble(long double a)
        {
                __int64 val[2]={0};
                memcpy(val,(void*)&a,sizeof(val));
                exp = (val[1]&0x7FFF)-16383;
                val[1] = (val[1]&0x8000) + 16383;
                if(val[0] && val[1]){
                        memcpy((void*)&a,val,sizeof(val));
                        this->val = (double)a;
                }
                else{
                        this->val=1;
```

```
                        exp=-16383;
345                 }
                    return *this;
            }
            __inline long double toLongDouble()
            {
350                 long double ret = val;
                    __int64 *v = (__int64*)&ret;
                    v[1] = (v[1]&0x8000) | (exp + 16383);
                    return ret;
            }
355 #endif
    };
    #endif //__FLOATEXP_H__
```

13 c/lib/generator.hs

```
    -- perturbator -- efficient deep zooming for Mandelbrot sets
    -- Copyright (C) 2015,2016 Claude Heiland-Allen
    -- License GPL3+ http://www.gnu.org/licenses/gpl.html

  5 {-# LANGUAGE FlexibleInstances #-}
    --module Main (main) where

    import Control.Monad (forM_, unless)
    import Control.Monad.Trans.RWS (put, get, tell, execRWS, RWS)
 10 import Data.Either (lefts, rights)
    import qualified Data.Foldable as F
    import Data.Function (on)
    import Data.List (group, groupBy, sort, sortBy, intercalate, partition)
    import Data.Map.Strict (Map)
 15 import qualified Data.Map.Strict as M
    import Data.Monoid (Monoid, (<>))
    import Data.Set (Set)
    import qualified Data.Set as S
    import System.IO (hFlush, stdout)
 20
    --import Debug.Trace (traceShow)
    --debug x = traceShow x x

    -- not in ghc-7.6
 25 isRight :: Either a b -> Bool
    isRight (Right _) = True
    isRight _ = False

    -- new in ghc-7.10
 30 sortOn :: Ord o => (a -> o) -> [a] -> [a]
    sortOn f = sortBy (compare `on` f)

    groupOn :: Eq e => (a -> e) -> [a] -> [[a]]
    groupOn f = groupBy ((==) `on` f)
 35
    class Diff e where
      diff :: e -> e

    class Simplify e where
 40   simplify :: e -> e
```

```
class Collect e where
  collect :: e -> e

normalize :: (Simplify e, Collect e) => e -> e
normalize = simplify . collect . simplify

data E = Sum [E] | Product [E] | N Integer | A Integer | Z | C | DeltaZ | DiffA ↙
    ↳ Integer | DiffZ | DiffDeltaZ | DeltaC{- must be last -}
  deriving (Read, Show, Eq, Ord)

instance Num E where
  fromInteger = N
  e + f = Sum [e, f]
  e * f = Product [e, f]
  negate e = Product [N (-1), e]

instance Diff E where
  diff (Sum es) = Sum (map diff es)
  diff (Product []) = 0
  diff (Product (e:es)) = diff e * Product es + e * diff (Product es)
  diff (N _) = 0
  diff (A n) = DiffA n
  diff Z = DiffZ
  diff C = 1
  diff DeltaZ = DiffDeltaZ
  diff DeltaC = 0
  diff e = error $ "diff: " ++ show e

instance Simplify E where
  simplify (Sum es) = simplifySum (sortBy sumCmp (map simplify es))
    where
      simplifySum ss = case ss of
        [] -> 0
        [e] -> e
        N 0 : gs -> simplifySum $ gs
        N i : N j : gs -> simplifySum $ (N (i + j) : gs)
        Sum fs : gs -> simplify $ Sum (fs ++ gs)
        gs -> Sum gs
      sumCmp (Sum a) (Sum b) = compare a b
      sumCmp (Sum _) _ = LT
      sumCmp _ (Sum _) = GT
      sumCmp (N a) (N b) = compare a b
      sumCmp (N _) _ = LT
      sumCmp _ (N _) = GT
      sumCmp a b = compare a b
  simplify (Product es) = simplifyProduct (sortBy prodCmp (map simplify es))
    where
      simplifyProduct ss = case ss of
        [] -> 1
        [e] -> e
        N 0 : _ -> 0
        N 1 : gs -> simplifyProduct $ gs
        N i : N j : gs -> simplifyProduct $ (N (i * j) : gs)
        Sum fs : gs -> simplify $ Sum [ Product (f : gs) | f <- fs ]
        Product fs : gs -> simplify $ Product (fs ++ gs)
        gs -> Product gs
```

```
        prodCmp (Product a) (Product b) = compare a b
        prodCmp (Product _) _ = LT
        prodCmp _ (Product _) = GT
        prodCmp (Sum a) (Sum b) = compare a b
        prodCmp (Sum _) _ = LT
        prodCmp _ (Sum _) = GT
        prodCmp (N a) (N b) = compare a b
        prodCmp (N _) _ = LT
        prodCmp _ (N _) = GT
        prodCmp a b = compare a b
    simplify e = e

instance Collect E where
    collect (Sum es) = Sum (concatMap accum . groupOn snd . sortOn snd . map ↙
        ↳ single $ es)
      where
        single (Product (N n : fs)) = (n, fs)
        single (Product fs) = (1, fs)
        single e = (1, [e])
        accum ps = case sum $ map fst ps of
          0 -> []
          1 -> [Product (      snd (head ps))]
          n -> [Product (N n : snd (head ps))]
    collect e = e

perturb :: E -> E
perturb Z = Z + DeltaZ
perturb C = C + DeltaC
perturb (Sum es) = Sum (map perturb es)
perturb (Product es) = Product (map perturb es)
perturb e = e

perturbed :: E -> E
perturbed e = perturb e - e

series :: Integer -> E
series n = sum [ A i * DeltaC ^ i | i <- [1 .. n] ]

sub :: E -> E -> E
sub e (Sum es) = Sum (map (sub e) es)
sub e (Product es) = Product (map (sub e) es)
sub e DeltaZ = e
sub _ f = f

collate :: E -> [(Integer, E)]
collate (Sum es) = map accum . groupOn fst . sortOn fst . map single $ es
  where
    single (Product fs) = (toInteger (length gs), Product (reverse hs))
      where
        (gs, hs) = span (DeltaC ==) . reverse $ fs
    single e = single (Product [e])
    accum ps = (fst (head ps), sum (map snd ps))
collate e = collate (Sum [e])


data CE
  = CSum [CE] | CProduct [CE] | CN Integer
```

```
    | ARe Integer | AIm Integer | ZRe | ZIm | CRe | CIm
    | DeltaZRe | DeltaZIm | DeltaCRe | DeltaCIm
    | DiffARe Integer | DiffAIm Integer
    | DiffZRe | DiffZIm | DiffDeltaZRe | DiffDeltaZIm
    | I
    deriving (Read, Show, Eq, Ord)

instance Num CE where
  fromInteger = CN
  e + f = CSum [e, f]
  e * f = CProduct [e, f]
  negate e = CProduct [CN (-1), e]

instance Simplify CE where
  simplify (CSum es) = simplifySum (sortBy sumCmp (map simplify es))
    where
      simplifySum ss = case ss of
        [] -> 0
        [e] -> e
        CN 0 : gs -> simplifySum $ gs
        CN i : CN j : gs -> simplifySum $ (CN (i + j) : gs)
        CSum fs : gs -> simplify $ CSum (fs ++ gs)
        gs -> CSum gs
      sumCmp (CSum a) (CSum b) = compare a b
      sumCmp (CSum _) _ = LT
      sumCmp _ (CSum _) = GT
      sumCmp (CN a) (CN b) = compare a b
      sumCmp (CN _) _ = LT
      sumCmp _ (CN _) = GT
      sumCmp a b = compare a b
  simplify (CProduct es) = simplifyProduct (sortBy prodCmp (map simplify es))
    where
      simplifyProduct ss = case ss of
        [] -> 1
        [e] -> e
        I : I : gs -> simplifyProduct $ (CN (-1) : gs)
        I : gs -> case simplifyProduct gs of
          CProduct hs -> CProduct (I : hs)
          h -> CProduct [I, h]
        CN 0 : _ -> 0
        CN 1 : gs -> simplifyProduct $ gs
        CN i : CN j : gs -> simplifyProduct $ (CN (i * j) : gs)
        CSum fs : gs -> simplify $ CSum [ CProduct (f : gs) | f <- fs ]
        CProduct fs : gs -> simplify $ CProduct (fs ++ gs)
        gs -> CProduct gs
      prodCmp (CProduct a) (CProduct b) = compare a b
      prodCmp (CProduct _) _ = LT
      prodCmp _ (CProduct _) = GT
      prodCmp (CSum a) (CSum b) = compare a b
      prodCmp (CSum _) _ = LT
      prodCmp _ (CSum _) = GT
      prodCmp I I = EQ
      prodCmp I _ = LT
      prodCmp _ I = GT
      prodCmp (CN a) (CN b) = compare a b
      prodCmp (CN _) _ = LT
      prodCmp _ (CN _) = GT
```

```
      prodCmp a b = compare a b
  simplify e = e

instance Collect CE where
  collect (CSum es) = CSum (concatMap accum . groupOn snd . sortOn snd . map ↙
    ↳ single $ es)
    where
      single (CProduct (CN n : fs)) = (n, fs)
      single (CProduct fs) = (1, fs)
      single e = (1, [e])
      accum ps = case sum $ map fst ps of
        0 -> []
        1 -> [CProduct (       snd (head ps))]
        n -> [CProduct (CN n : snd (head ps))]
  collect e = e

complex :: E -> CE
complex Z  = ZRe  + I * ZIm
complex C  = CRe  + I * CIm
complex DeltaZ = DeltaZRe + I * DeltaZIm
complex DeltaC = DeltaCRe + I * DeltaCIm
complex (A n) = ARe n + I * AIm n
complex (N n) = CN n
complex (Sum es) = CSum (map complex es)
complex (Product es) = CProduct (map complex es)
complex DiffZ = DiffZRe + I * DiffZIm
complex DiffDeltaZ = DiffDeltaZRe + I * DiffDeltaZIm
complex (DiffA n) = DiffARe n + I * DiffAIm n

newtype Pair t = Pair{ unPair :: (t, t) }
  deriving (Read, Show, Eq, Ord)

instance Functor Pair where
  fmap f (Pair (a, b)) = Pair (f a, f b)

decomplex :: CE -> Pair CE
decomplex (CSum es)= Pair (CSum res, CSum ims)
  where (res, ims) = unzip (map (unPair . decomplex) es)
decomplex (CProduct es)
  | even n    = Pair (CProduct (i : fs), 0)
  | otherwise = Pair (0, CProduct (i : fs))
  where
    fs = filter (I /=) es
    n = length (filter (I ==) es)
    i = case n `mod` 4 of
      0 ->  1
      1 ->  1
      2 -> -1
      3 -> -1
decomplex I = Pair (0, 1)
decomplex e = Pair (e, 0)

cnormalize :: E -> Pair CE
cnormalize = fmap normalize . decomplex . normalize . complex . normalize


data OSum = OSum [OProduct]
```

```
  deriving (Read, Show, Eq, Ord)
data OProduct = OProduct Integer [OPower]
  deriving (Read, Show, Eq)
instance Ord OProduct where
  compare (OProduct x xs) (OProduct y ys) = compare (abs x) (abs y) <> compare (signum x) (signum y) <> compare xs ys
data OPower = OPower OVar Integer
  deriving (Read, Show, Eq, Ord)
data OVar = OVar Integer
  deriving (Read, Show, Eq, Ord)

osum :: CE -> OSum
osum (CSum cs) = OSum (sort . map oproduct $ cs)
osum c = OSum [oproduct c]

oproduct :: CE -> OProduct
oproduct (CProduct (CN n : cs)) = OProduct n (sort . map opower . group $ cs)
oproduct (CProduct cs) = OProduct 1 (sort . map opower . group $ cs)
oproduct (CN n) = OProduct n []
oproduct c = OProduct 1 [opower [c]]

opower :: [CE] -> OPower
opower cs@(c:_) = OPower (ovar c) (toInteger $ length cs)

ovar :: CE -> OVar
ovar CRe = OVar 0
ovar CIm = OVar 1
ovar ZRe = OVar 2
ovar ZIm = OVar 3
ovar DiffZRe = OVar 4
ovar DiffZIm = OVar 5
ovar (ARe n) = OVar $ 4 * n + 2
ovar (AIm n) = OVar $ 4 * n + 3
ovar (DiffARe n) = OVar $ 4 * n + 4
ovar (DiffAIm n) = OVar $ 4 * n + 5
ovar e = error $ "ovar: " ++ show e

data PSum = PSum [PMul1]
  deriving (Read, Show, Eq, Ord)
data PMul1 = PMul1 Integer [PMul2]
  deriving (Read, Show, Eq, Ord)
data PMul2 = PMul2 Bool Integer [PPower]
  deriving (Read, Show, Eq, Ord)
type PPower = OPower
type PVar = OVar

pmul1s :: PMul1 -> Integer
pmul1s (PMul1 i _) = i

pmul1p :: PMul1 -> [PMul2]
pmul1p (PMul1 _ p) = p

psum :: OSum -> PSum
psum (OSum ps) = PSum . map (\pms@(PMul1 i _ : _) -> PMul1 i (concatMap pmul1p pms)) . groupOn pmul1s . sort . map (pmul1 0) $ ps

pmul1 :: Integer -> OProduct -> PMul1
```

```
pmul1 n (OProduct i os)
  | odd i = PMul1 (abs i) [PMul2 (i < 0) n os]
  | otherwise = pmul1 (n + 1) (OProduct (i `div` 2) os)

-- outermost first
data P
  = P'Add1 P P
  | P'Mul1 P Integer
  | P'Add2 P P
  | P'Neg P
  | P'Mul2 P Integer
  | P'Mul P P
  | P'Sqr P
  | P'Var Integer
  | P'Constant Integer
  deriving (Read, Show, Eq, Ord)

class Compile p where
  compile :: p -> P

instance Compile OVar where
  compile (OVar v) = P'Var v

instance Compile OPower where
  compile (OPower v 1) = compile v
  compile (OPower v n)
    | even n = P'Sqr (compile (OPower v (n `div` 2)))
    | otherwise = P'Mul (compile (OPower v (n `div` 2))) (compile (OPower v ((n ↙
      ↳ `div` 2) + 1)))

instance Compile PMul2 where
  compile (PMul2 s i []) = P'Constant ((if s then negate else id) (2^i))
  compile (PMul2 s i xs) = (if s then P'Neg else id) . (if i == 0 then id else ↙
    ↳ (`P'Mul2` i)) . compile $ xs

instance Compile [OPower] where
  compile [x] = compile x
  compile [x,y] = P'Mul (compile x) (compile y)
  compile xs = case splitAt (length xs `div` 2) xs of
    (ys, zs) -> P'Mul (compile ys) (compile zs)

instance Compile PMul1 where
  compile p@(PMul1 _ []) = error $ "compile: " ++ show p
  compile (PMul1 1 xs) = compile xs
  compile (PMul1 n xs) = P'Mul1 (compile xs) n

instance Compile [PMul2] where
  compile [x] = compile x
  compile [x,y] = P'Add2 (compile x) (compile y)
  compile xs = case splitAt (length xs `div` 2) xs of
    (ys, zs) -> P'Add2 (compile ys) (compile zs)

instance Compile PSum where
  compile p@(PSum []) = error $ "compile:  " ++ show p
  compile (PSum xs) = compile xs

instance Compile [PMul1] where
```

```
  compile [x] = compile x
  compile [x,y] = P'Add1 (compile x) (compile y)
  compile xs = case splitAt (length xs `div` 2) xs of
    (ys, zs) -> P'Add1 (compile ys) (compile zs)

parallel :: [P] -> RWS () [Phase] (Map P Int) ()
parallel expressions = case groupOn pType . S.toList . S.fromList $ expressions ↙
  ↳ of
  [] -> return ()
  (ops:opss) -> do
    results <- mapM temporary ops
    let subs = map subexpressions ops
    arguments <- mapM (mapM temporary) subs
    unless (null (concat arguments)) $
      tell [Phase (phaseOp . pType . head $ ops) (zip results arguments)]
    parallel (concat (subs ++ opss))

data Phase = Phase Op [(Either Integer Int, [Either Integer Int])]
  deriving (Read, Show, Eq, Ord)

data T = T'Add1 | T'Mul1 | T'Add2 | T'Neg | T'Mul2 | T'Mul | T'Sqr | T'Var | T'↙
  ↳ Constant
  deriving (Read, Show, Eq, Ord, Enum, Bounded)

pType :: P -> T
pType P'Add1{} = T'Add1
pType P'Mul1{} = T'Mul1
pType P'Add2{} = T'Add2
pType P'Neg{} = T'Neg
pType P'Mul2{} = T'Mul2
pType P'Mul{} = T'Mul
pType P'Sqr{} = T'Sqr
pType P'Var{} = T'Var
pType P'Constant{} = T'Constant

subexpressions :: P -> [P]
subexpressions (P'Add1 a b) = [a, b]
subexpressions (P'Mul1 a b) = [a, P'Constant b]
subexpressions (P'Add2 a b) = [a, b]
subexpressions (P'Neg a) = [a]
subexpressions (P'Mul2 a b) = [a, P'Constant b]
subexpressions (P'Mul a b) = [a, b]
subexpressions (P'Sqr a) = [a]
subexpressions _ = []

temporary :: Monoid w => P -> RWS r w (Map P Int) (Either Integer Int)
temporary (P'Constant n) = return (Left n)
temporary p = do
  m <- get
  case M.lookup p m of
    Nothing -> do
      let n = M.size m
      put (M.insert p n m)
      return (Right n)
    Just n -> return (Right n)

data Op = OpAdd | OpAddI | OpMulI | OpNeg | OpMul2 | OpMul | OpSqr | OpVar | ↙
```

```
    ↳ OpSet
  deriving (Read, Show, Eq, Ord, Enum, Bounded)

phaseOp :: T -> Op
phaseOp T'Add1 = OpAdd
phaseOp T'Mul1 = OpMulI
phaseOp T'Add2 = OpAdd
phaseOp T'Neg  = OpNeg
phaseOp T'Mul2 = OpMul2
phaseOp T'Mul  = OpMul
phaseOp T'Sqr  = OpSqr

referenceCounts :: [Phase] -> Map Int Int
referenceCounts ps = M.fromListWith (+) [ (a, 1) | Phase _ ras <- ps, (_, args) ↙
    ↳ <- ras, Right a <- args ]

duplicates :: Map Int Int -> Set Int
duplicates = M.keysSet . M.filter (1 <)

inplace :: [Phase] -> [Phase]
inplace phases = renames phases . fst $ execRWS (tell () >> unifies phases) () S↙
    ↳ .empty

compact :: [Phase] -> [Phase]
compact phases = renamem phases (M.fromList (zip (S.toList (M.keysSet (↙
    ↳ referenceCounts phases))) [0..]))

renames :: [Phase] -> Set (Set Int) -> [Phase]
renames phases renamings = map f phases
  where
    rens = S.toList renamings
    f (Phase op ras) = Phase op (map g ras)
    g (res, args) = (h res, map h args)
    h (Right r) = case filter (S.member r) rens of
      [] -> Right r
      [s] -> Right (S.findMin s)
      ss -> error $ "renames: " ++ show ss
    h l = l

renamem :: [Phase] -> Map Int Int -> [Phase]
renamem phases renamingm = map f phases
  where
    f (Phase op ras) = Phase op (map g ras)
    g (res, args) = (h res, map h args)
    h (Right r) = Right (renamingm M.! r)
    h l = l

unify :: Monoid w => Set Int -> RWS r w (Set (Set Int)) ()
unify xs = do
  s <- get
  let (disjoint, intersecting) = S.partition (S.null . S.intersection xs) s
  put $ S.insert (S.unions (xs : S.toList intersecting)) disjoint

unifies :: Monoid w => [Phase] -> RWS r w (Set (Set Int)) ()
unifies phases = forM_ phases $ \(Phase op ras) -> case op of
    OpNeg  -> mapM_ go ras
    OpMul2 -> mapM_ go ras
```

```
      _ -> return ()
485   where
        dups = duplicates . referenceCounts $ phases
        go (res, args)
          | any (`S.member` dups) ios = return ()
          | otherwise = unify (S.fromList ios)
490       where
            ios = rights (res : args)

    splitAdds :: [Phase] -> [Phase]
    splitAdds = concatMap f
495   where
        f (Phase OpAdd ras) = case partition (all isRight . snd) ras of
          ([], ai) -> [Phase OpAddI ai]
          (a, []) -> [Phase OpAdd a]
          (a, ai) -> [Phase OpAdd a, Phase OpAddI ai]
500     f p = [p]


    codegen :: Integer -> String -> [(Int, Phase)] -> [(FilePath, String)]
    codegen order fname ps =
505
      [ (stem ++ ".c",
      "#include <complex>\n\
      \#include <algorithm>\n\
      \#include <limits.h>\n\
510   \#include <math.h>\n\
      \#include <stdbool.h>\n\
      \#include <stdint.h>\n\
      \#include <stdio.h>\n\
      \#include <stdlib.h>\n\
515   \#include <mpfr.h>\n\
      \\n\
      \#include " ++ show (stem ++ ".h") ++ "\n\
      \#include " ++ show "../edouble.cc" ++ "\n\
      \\n\
520   \static const struct {\n" ++ unlines (map struct ps) ++ "  " ++ int ++ " v[" ↙
          ↪ ++ show order ++ "][2];\n  " ++ int ++ " dv[" ++ show order ++"][2];\n} ↙
          ↪ " ++ stem ++ "_series_spec =\n\
      \{\n" ++ intercalate "\n,\n" (map values ps ++ [valids (last ps), dvalids (↙
          ↪ last ps)]) ++ "};\n\
      \\n\
      \struct " ++ stem ++ "_series *" ++ stem ++ "_series_new(const mpfr_t cx, ↙
          ↪ const mpfr_t cy) {\n\
      \  struct " ++ stem ++ "_series *s = (struct " ++ stem ++ "_series *) malloc(↙
          ↪ sizeof(*s));\n\
525   \  if (! s) { return 0; }\n\
      \  mpfr_prec_t p = std::max(mpfr_get_prec(cx), mpfr_get_prec(cy));\n\
      \  for (int i = 0; i < " ++ show count ++ "; ++i) {\n\
      \    mpfr_init2(s->v[i], p);\n\
      \    mpfr_set_si(s->v[i], 0, MPFR_RNDN);\n\
530   \  };\n\
      \  mpfr_set(s->v[0], cx, MPFR_RNDN);\n\
      \  mpfr_set(s->v[1], cy, MPFR_RNDN);\n\
      \  mpfr_set(s->v[2], cx, MPFR_RNDN);\n\
      \  mpfr_set(s->v[3], cy, MPFR_RNDN);\n\
535   \  mpfr_set_si(s->v[4], 1, MPFR_RNDN);\n\
```

```
    \  mpfr_set_si(s->v[6], 1, MPFR_RNDN);\n\
    \  s->n = 1;\n\
    \  return s;\n\
    \}\n\
    \\n\
    \void " ++ stem ++ "_series_delete(struct " ++ stem ++ "_series *s) {\n\
    \  for (int i = 0; i < " ++ show count ++ "; ++i) {\n\
    \    mpfr_clear(s->v[i]);\n\
    \  }\n\
    \  free(s);\n\
    \}\n\
    \\n\
    \int " ++ stem ++ "_series_get_n(const struct " ++ stem ++ "_series *s) {\n\
    \  return s->n;\n\
    \}\n\
    \\n\
    \bool " ++ stem ++ "_series_step(struct " ++ stem ++ "_series *s, mpfr_exp_t exponent, mpfr_exp_t threshold) {\n\n" ++
    unlines (map genphase (init ps)) ++
    "  bool valid = true, dvalid = true;\n\
    \  mpfr_exp_t e0;\n\
    \  mpfr_exp_t e1;\n\
    \  mpfr_exp_t de0;\n\
    \  mpfr_exp_t de1;\n\
    \  for (int i = 0; i < " ++ show order ++ "; ++i) {\n\
    \    e1 = LONG_MIN;\n\
    \    de1 = LONG_MIN;\n\
    \    for (int j = 0; j < 2; ++j) {\n\
    \      if (! mpfr_zero_p(s->v[" ++ stem ++ "_series_spec.v[i][j]])) {\n\
    \        e1 = std::max(e1, mpfr_get_exp(s->v[" ++ stem ++ "_series_spec.v[i][j]]));\n\
    \      }\n\
    \      if (! mpfr_zero_p(s->v[" ++ stem ++ "_series_spec.dv[i][j]])) {\n\
    \        de1 = std::max(de1, mpfr_get_exp(s->v[" ++ stem ++ "_series_spec.dv[i][j]]));\n\
    \      }\n\
    \    }\n\
    \    if (i > 0) {\n\
    \      valid = e0 - exponent >= e1 + threshold;\n\
    \      dvalid = de0 - exponent >= de1 + threshold;\n\
    \      e0 = std::max(e0 - exponent, e1);\n\
    \      de0 = std::max(de0 - exponent, de1);\n\
    \    } else {\n\
    \      e0 = e1;\n\
    \      de0 = de1;\n\
    \    }\n\
    \  }\n\
    \  if ((! valid) || (! dvalid)) { return false; }\n\
    \\n\
    \  s->n += 1;\n\n" ++
    unlines [genphase (last ps)] ++
    "  return true;\n\
    \}\n\
    \\n\
    \struct " ++ fname ++ "_reference *" ++ stem ++ "_reference_new(const struct " ++ stem ++ "_series *s) {\n\
    \  return " ++ fname ++ "_reference_new(s->v[0], s->v[1], s->v[2], s->v[3], s
```

```
      ↪ ->n);\n\
  \}\n\
  \\n")

  , (stem ++ "_native.c",
  "template <typename R>\n\
  \struct " ++ stem ++ "_approx {\n\
  \  std::complex<R> v[" ++ show order ++ "];\n\
  \  std::complex<R> dv[" ++ show (order + 1) ++ "];\n\
  \  int exponent;\n\
  \};\n\
  \\n\
  \template <typename R>\n\
  \struct " ++ stem ++ "_approx<R> *"  ++ stem ++ "_approx_new(const struct " ++↙
      ↪  stem ++ "_series *s, int exponent, const R &dummy) {\n\
  \  (void) dummy;\n\
  \  struct " ++ stem ++ "_approx<R> *a = (struct " ++ stem ++ "_approx<R> *) ↙
      ↪ malloc(sizeof(*a));\n\
  \  mpfr_t t;\n\
  \  mpfr_init2(t, mpfr_get_prec(s->v[0]));\n\
  \  {\n\
  \    R dre = mpfr_get(s->v[4], MPFR_RNDN, R(0));\n\
  \    R dim = mpfr_get(s->v[5], MPFR_RNDN, R(0));\n\
  \    a->dv[0] = std::complex<R>(dre, dim);\n\
  \  }\n\
  \  for (int i = 0; i < " ++ show order ++ "; ++i) {\n\
  \    mpfr_set(t, s->v[4 * (i + 1) + 2], MPFR_RNDN);\n\
  \    mpfr_mul_2si(t, t, (i + 1) * exponent, MPFR_RNDN);\n\
  \    R re = mpfr_get(t, MPFR_RNDN, R(0));\n\
  \    mpfr_set(t, s->v[4 * (i + 1) + 3], MPFR_RNDN);\n\
  \    mpfr_mul_2si(t, t, (i + 1) * exponent, MPFR_RNDN);\n\
  \    R im = mpfr_get(t, MPFR_RNDN, R(0));\n\
  \    a->v[i] = std::complex<R>(re, im);\n\
  \    mpfr_set(t, s->v[4 * (i + 1) + 4], MPFR_RNDN);\n\
  \    mpfr_mul_2si(t, t, (i + 1) * exponent, MPFR_RNDN);\n\
  \    R dre = mpfr_get(t, MPFR_RNDN, R(0));\n\
  \    mpfr_set(t, s->v[4 * (i + 1) + 5], MPFR_RNDN);\n\
  \    mpfr_mul_2si(t, t, (i + 1) * exponent, MPFR_RNDN);\n\
  \    R dim = mpfr_get(t, MPFR_RNDN, R(0));\n\
  \    a->dv[i + 1] = std::complex<R>(dre, dim);\n\
  \  }\n\
  \  mpfr_clear(t);\n\
  \  a->exponent = -exponent;\n\
  \  return a;\n\
  \}\n\
  \\n\
  \template <typename R>\n\
  \void " ++ stem ++ "_approx_do(const struct " ++ stem ++ "_approx<R> *a, std::↙
      ↪ complex<R> dc, std::complex<R> *dz_out, std::complex<R> *ddz_out) {\n\
  \  std::complex<R> z(ldexp(std::real(dc), a->exponent), ldexp(std::imag(dc), a↙
      ↪ ->exponent));\n\
  \  std::complex<R> zi(z);\n\
  \  std::complex<R> s(0);\n\
  \  std::complex<R> ds(a->dv[0]);\n\
  \  for (int i = 0; i < " ++ show order ++ "; ++i) {\n\
  \    s += a->v[i] * zi;\n\
  \    ds += a->dv[i + 1] * zi;\n\
```

```
    \    zi *= z;\n\
    \  }\n\
    \  *dz_out = s;\n\
    \  *ddz_out = ds;\n\
    \}\n\
    \\n\
    \template<typename R>\n\
    \int " ++ stem ++ "_approx_get_exponent(const struct " ++ stem ++ "_approx<R> ↙
        ↳ *a) {\n\
    \  return a->exponent;\n\
    \}\n\
    \template <typename R>\n\
    \const std::complex<R> *" ++ stem ++ "_approx_get_coefficients(const struct " ↙
        ↳ ++ stem ++ "_approx<R> *a) {\n\
    \  return &a->v[0];\n\
    \}\n\
    \template <typename R>\n\
    \const std::complex<R> *" ++ stem ++ "_approx_get_dcoefficients(const struct "↙
        ↳  ++ stem ++ "_approx<R> *a) {\n\
    \  return &a->dv[0];\n\
    \}\n\
    \")

    , (stem ++ ".h",
    "#ifndef " ++ stem ++ "_h\n\
    \#define " ++ stem ++ "_h 1\n\
    \\n\
    \#include <complex>\n\
    \#include <mpfr.h>\n\
    \\n\
    \#include " ++ show (fname ++ "_ref.h") ++ "\n\
    \\n\
    \struct " ++ stem ++ "_series {\n\
    \  mpfr_t v[" ++ show count ++ "];\n\
    \  int n;\n\
    \};\n\
    \\n\
    \struct " ++ stem ++ "_series *" ++ stem ++ "_series_new(const mpfr_t cx, ↙
        ↳ const mpfr_t cy);\n\
    \void " ++ stem ++ "_series_delete(struct " ++ stem ++ "_series *s);\n\
    \int " ++ stem ++ "_series_get_n(const struct " ++ stem ++ "_series *s);\n\
    \bool " ++ stem ++ "_series_step(struct " ++ stem ++ "_series *s, mpfr_exp_t ↙
        ↳ exponent, mpfr_exp_t threshold);\n\
    \struct " ++ fname ++ "_reference *" ++ stem ++ "_reference_new(const struct "↙
        ↳  ++ stem ++ "_series *s);\n\
    \template <typename R>\n\
    \struct " ++ stem ++ "_approx;\n\
    \template <typename R>\n\
    \struct " ++ stem ++ "_approx<R> *"  ++ stem ++ "_approx_new(const struct " ++↙
        ↳  stem ++ "_series *s, int exponent, const R &dummy);\n\
    \template <typename R>\n\
    \void " ++ stem ++ "_approx_delete(struct " ++ stem ++ "_approx<R> *s);\n\
    \template <typename R>\n\
    \int " ++ stem ++ "_approx_get_exponent(const struct " ++ stem ++ "_approx<R> ↙
        ↳ *a);\n\
    \template <typename R>\n\
    \const std::complex<R> *" ++ stem ++ "_approx_get_coefficients(const struct " ↙
```

```
      ↪ ++ stem ++ "_approx<R> *a);\n\
690   \template <typename R>\n\
      \const std::complex<R> *" ++ stem ++ "_approx_get_dcoefficients(const struct "↙
      ↪  ++ stem ++ "_approx<R> *a);\n\
      \template <typename R>\n\
      \void " ++ stem ++ "_approx_do(const struct " ++ stem ++ "_approx<R> *a, std::↙
      ↪ complex<R> dc, std::complex<R> *dz_out, std::complex<R> *ddz_out);\n\
      \\n\
695   \#include \"" ++ stem ++ "_native.c\"\n\
      \\n\
      \#endif\n")]

      where
700     int
          | count <= 2^( 8 :: Int) = "uint8_t"
          | count <= 2^(16 :: Int) = "uint16_t"
          | count <= 2^(32 :: Int) = "uint32_t"
          | otherwise      = "uint64_t"
705     stem = fname ++ "_" ++ show order
        struct (i, Phase _ ras@((_, args):_)) = "  " ++ int ++ " p" ++ show i ++ "["↙
        ↪  ++ show (length ras) ++ "][" ++ show (length args + 1) ++ "];"
        values (_, Phase _ ras) = "{\n" ++ intercalate ",\n" (map value ras) ++ "\n↙
        ↪ }"
        value (Right res, args) = "{ " ++ show res ++ " , " ++ intercalate " , " (↙
        ↪ map show (rights args) ++ map show (lefts args)) ++ " }"
        count = 1 + maximum [ i | (_, Phase _ ras) <- ps, (res, args) <-ras, Right i↙
        ↪  <- res : args ]
710     valids (_, Phase OpSet ras)
          = "{\n" ++ intercalate ",\n"
              [ "{ " ++ show re ++ " , " ++ show im ++ " }"
              | o <- [1 .. fromIntegral order]
              , (Right sre, [ Right re ]) <- ras
715           , sre == 4 * o + 2
              , (Right sim, [ Right im ]) <- ras
              , sim == 4 * o + 3
              ] ++ "\n}\n"
        dvalids (_, Phase OpSet ras)
720       = "{\n" ++ intercalate ",\n"
              [ "{ " ++ show re ++ " , " ++ show im ++ " }"
              | o <- [1 .. fromIntegral order]
              , (Right sre, [ Right re ]) <- ras
              , sre == 4 * o + 4
725           , (Right sim, [ Right im ]) <- ras
              , sim == 4 * o + 5
              ] ++ "\n}\n"
        genphase (p, Phase op ras) =
          "  #pragma omp parallel for\n\
730       \  for (int i = 0; i < " ++ show (length ras) ++ "; ++i) {\n" ++
          ( case op of
              OpAdd  -> o3  "mpfr_add"
              OpAddI -> o3i "mpfr_add_si"
              OpMulI -> o3i "mpfr_mul_si"
735           OpNeg  -> o2  "mpfr_neg"
              OpMul2 -> o3i "mpfr_mul_2si"
              OpMul  -> o3  "mpfr_mul"
              OpSqr  -> o2  "mpfr_sqr"
              OpSet  -> o2  "mpfr_set"
```

```
740             _ -> error $ "genphase: " ++ show op ) ++
        "  }\n"
        where
          o3  s = "  " ++ s ++ "\n\
                  \    ( s->v[" ++ stem ++ "_series_spec.p" ++ show p ++ "[i][0]]\↙
                    ↪ n\
745               \    , s->v[" ++ stem ++ "_series_spec.p" ++ show p ++ "[i][1]]\↙
                    ↪ n\
                  \    , s->v[" ++ stem ++ "_series_spec.p" ++ show p ++ "[i][2]]\↙
                    ↪ n\
                  \    , MPFR_RNDN\n\
                  \    );\n"
          o3i s = "  " ++ s ++ "\n\
750               \    ( s->v[" ++ stem ++ "_series_spec.p" ++ show p ++ "[i][0]]\↙
                    ↪ n\
                  \    , s->v[" ++ stem ++ "_series_spec.p" ++ show p ++ "[i][1]]\↙
                    ↪ n\
                  \    ,      " ++ stem ++ "_series_spec.p" ++ show p ++ "[i][2]\n↙
                    ↪ \
                  \    , MPFR_RNDN\n\
                  \    );\n"
755       o2  s = "  " ++ s ++ "\n\
                  \    ( s->v[" ++ stem ++ "_series_spec.p" ++ show p ++ "[i][0]]\↙
                    ↪ n\
                  \    , s->v[" ++ stem ++ "_series_spec.p" ++ show p ++ "[i][1]]\↙
                    ↪ n\
                  \    , MPFR_RNDN\n\
                  \    );\n"
760
main' :: String -> [Integer] -> E -> [(FilePath, String)]
main' stem orders f = main''' stem f ++ concat [main'' stem order f | order <- ↙
    ↪ orders ] ++ codegenWrap stem orders ++ codegenMake stem orders

main''' :: String -> E -> [(FilePath, String)]
765 main''' stem f = codegenRef stem ref
  where
    ref = zip [0..] . splitAdds . compact . inplace . reverse $ finalize : ↙
        ↪ phases
      where
        finalize = Phase OpSet $ zip (map (Right . (variables M.!)) vs) (map ↙
            ↪ ((:[]) . Right . (variables M.!)) es)
770     initial = fst $ execRWS initialize () M.empty
        initialize = tell () >> mapM_ temporary ([var CRe, var CIm] ++ vs ++ es)
        (variables, phases) = execRWS (parallel es) () initial
        (vs, es) = unzip [ (var v, expr e) | (v, e) <- ves ]
        var = compile . ovar
775     expr = compile . psum . osum
        ves =
          [ (ZRe, fre)
          , (ZIm, fim)
          , (DiffZRe, gre)
780       , (DiffZIm, gim)
          ]
        Pair (fre, fim) = cnormalize $ f
        Pair (gre, gim) = cnormalize . normalize . diff $ f

785 codegenMake :: String -> [Integer] -> [(FilePath, String)]
```

```
codegenMake stem orders =
  [ ( "Makefile",
  "OBJECTS = \\\n" ++
  unlines [ "\t" ++ stem ++ "_" ++ show order ++ ".o \\" | order <- orders ] ++
  "\t" ++ stem ++ "_ref.o \\\n\t" ++ stem ++ ".o\n\
  \\n\
  \all: $(OBJECTS)\n\
  \.SUFFIXES:\n\
  \.PHONY: all\n\
  \%.o: %.c ../edouble.cc\n\
  \\tg++ -std=c++11 -pedantic -Wall -Wextra -fopenmp -O3 -march=native -c $<\n\
  \\n")
  ]

codegenWrap :: String -> [Integer] -> [(FilePath, String)]
codegenWrap stem orders =
  [ ( stem ++ ".h",
  "#ifndef " ++ stem ++ "_h\n\
  \#define " ++ stem ++ "_h 1\n\
  \\n\
  \#include <complex>\n\
  \\n\
  \#include " ++ show (stem ++ "_ref.h") ++ "\n\
  \\n" ++
  unlines [ "#include " ++ show (stem ++ "_" ++ show order ++ ".h") | order <- orders ] ++
  "\n\
  \struct " ++ stem ++ "_series {\n\
  \  int order;\n\
  \  void *series;\n\
  \};\n\
  \\n\
  \struct " ++ stem ++ "_series *" ++ stem ++ "_series_new(const int order, const mpfr_t cx, const mpfr_t cy);\n\
  \void " ++ stem ++ "_series_delete(struct " ++ stem ++ "_series *s);\n\
  \bool " ++ stem ++ "_series_step(struct " ++ stem ++ "_series *s, const mpfr_exp_t exponent, const mpfr_exp_t threshold);\n\
  \int " ++ stem ++ "_series_get_n(const struct " ++ stem ++ "_series *s);\n\
  \struct " ++ stem ++ "_reference *" ++ stem ++ "_series_reference_new(const struct " ++ stem ++ "_series *s);\n\
  \\n\
  \#include \"" ++ stem ++ "_approx_native.c\"\n\
  \\n\
  \#endif\n\
  \\n")

  , ( stem ++ "_approx_native.h",
  "template <typename R>\n\
  \struct " ++ stem ++ "_approx;\n\
  \template <typename R>\n\
  \struct " ++ stem ++ "_approx<R> *" ++ stem ++ "_series_approx_new(const struct " ++ stem ++ "_series *s, const int exponent, const R &dummy);\n\
  \template <typename R>\n\
  \void " ++ stem ++ "_approx_delete(struct " ++ stem ++ "_approx<R> *a);\n\
  \template <typename R>\n\
  \int " ++ stem ++ "_approx_get_order(const struct " ++ stem ++ "_approx<R> *a);\n\
```

```
    \template <typename R>\n\
    \int " ++ stem ++ "_approx_get_exponent(const struct " ++ stem ++ "_approx<R> *a);\n\
    \template <typename R>\n\
    \const std::complex<R> *" ++ stem ++ "_approx_get_coefficients(const struct " ++ stem ++ "_approx<R> *a);\n\
    \template <typename R>\n\
    \const std::complex<R> *" ++ stem ++ "_approx_get_dcoefficients(const struct " ++ stem ++ "_approx<R> *a);\n\
    \template <typename R>\n\
    \void " ++ stem ++ "_approx_do(const struct " ++ stem ++ "_approx<R> *a, const std::complex<R> dc, std::complex<R> *dz_out, std::complex<R> *ddz_out);\n\
    \\n")

    , ( stem ++ ".c",
    "#ifndef " ++ stem ++ "_c\n\
    \#define " ++ stem ++ "_c 1\n\
    \\n\
    \#include <stdlib.h>\n\
    \\n\
    \#include \"" ++ stem ++ ".h\"\n\
    \\n\
    \struct " ++ stem ++ "_series *" ++ stem ++ "_series_new(const int order, const mpfr_t cx, const mpfr_t cy) {\n\
    \  struct " ++ stem ++ "_series *s = (struct " ++ stem ++ "_series *) calloc(1, sizeof(*s));\n\
    \  if (! s) { return 0; }\n\
    \  s->order = order;\n\
    \  switch (order) {\n" ++
    unlines [ "    case " ++ show order ++ ": s->series = " ++ stem ++ "_" ++ show order ++ "_series_new(cx, cy); break;" | order <- orders ] ++
    "  }\n\
    \  if (! s->series) {\n\
    \    free(s);\n\
    \    return 0;\n\
    \  }\n\
    \  return s;\n\
    \}\n\
    \\n\
    \void " ++ stem ++ "_series_delete(struct " ++ stem ++ "_series *s) {\n\
    \  if (! s) {\n\
    \    return;\n\
    \  }\n\
    \  switch (s->order) {\n" ++
    unlines [ "    case " ++ show order ++ ": " ++ stem ++ "_" ++ show order ++ "_series_delete((struct " ++ stem ++ "_" ++ show order ++ "_series *) s->series); break;" | order <- orders ] ++
    "  }\n\
    \  free(s);\n\
    \}\n\
    \\n\
    \bool " ++ stem ++ "_series_step(struct " ++ stem ++ "_series *s, const mpfr_exp_t exponent, const mpfr_exp_t threshold) {\n\
    \  if (! s) {\n\
    \    return false;\n\
    \  }\n\
```

```
  \  switch (s->order) {\n" ++
  unlines [ "    case " ++ show order ++ ": return " ++ stem ++ "_" ++ show ↙
      ↪ order ++ "_series_step((struct " ++ stem ++ "_" ++ show order ++ "↙
      ↪ _series *) s->series, exponent, threshold);" | order <- orders ] ++
  "  }\n\
  \  return false;\n\
  \}\n\
  \\n\
  \int " ++ stem ++ "_series_get_n(const struct " ++ stem ++ "_series *s) {\n\
  \  if (! s) {\n\
  \    return 0;\n\
  \  }\n\
  \  switch (s->order) {\n" ++
  unlines [ "    case " ++ show order ++ ": return " ++ stem ++ "_" ++ show ↙
      ↪ order ++ "_series_get_n((const struct " ++ stem ++ "_" ++ show order ++ ↙
      ↪ "_series *) s->series);" | order <- orders ] ++
  "  }\n\
  \  return 0;\n\
  \}\n\
  \\n\
  \struct " ++ stem ++ "_reference *" ++ stem ++ "_series_reference_new(const ↙
      ↪ struct " ++ stem ++ "_series *s) {\n\
  \  if (! s) {\n\
  \    return 0;\n\
  \  }\n\
  \  switch (s->order) {\n" ++
  unlines [ "    case " ++ show order ++ ": return " ++ stem ++ "_" ++ show ↙
      ↪ order ++ "_reference_new((const struct " ++ stem ++ "_" ++ show order ++↙
      ↪  "_series *) s->series);" | order <- orders ] ++
  " }\n\
  \  return 0;\n\
  \}\n\
  \\n\
  \#endif\n\
  \\n")

  , ( stem ++ "_approx_native.c",
  "template <typename R>\n\
  \struct " ++ stem ++ "_approx {\n\
  \  int order;\n\
  \  void *approx;\n\
  \};\n\
  \\n\
  \template <typename R>\n\
  \struct " ++ stem ++ "_approx<R> *" ++ stem ++ "_series_approx_new(const ↙
      ↪ struct " ++ stem ++ "_series *s, const int exponent, const R &dummy) {\n↙
      ↪ \
  \  (void) dummy;\n\
  \  if (! s) {\n\
  \    return 0;\n\
  \  }\n\
  \  struct " ++ stem ++ "_approx<R> *a = (struct " ++ stem ++ "_approx<R> *) ↙
      ↪ calloc(1, sizeof(*a));\n\
  \  a->order = s->order;\n\
  \  switch (s->order) {\n" ++
  unlines [ "    case " ++ show order ++ ": a->approx = " ++ stem ++ "_" ++ show↙
      ↪  order ++ "_approx_new((const struct " ++ stem ++ "_" ++ show order ++ "↙
```

```
      ↪ _series *) s->series, exponent, dummy); break;" | order <- orders ] ++
    "  }\n\
    \  if (! a->approx) {\n\
    \    free(a);\n\
    \    return 0;\n\
    \  }\n\
    \  return a;\n\
    \}\n\
    \\n\
    \template <typename R>\n\
    \void " ++ stem ++ "_approx_delete(struct " ++ stem ++ "_approx<R> *a) {\n\
    \  if (! a) {\n\
    \    return;\n\
    \  }\n\
    \  free(a->approx);  // FIXME check this\n\
    \  free(a);\n\
    \}\n\
    \\n\
    \template <typename R>\n\
    \int " ++ stem ++ "_approx_get_order(const struct " ++ stem ++ "_approx<R> *a)↙
      ↪ {\n\
    \  if (! a) {\n\
    \    return 0;\n\
    \  }\n\
    \  return a->order;\n\
    \}\n\
    \\n\
    \template <typename R>\n\
    \int " ++ stem ++ "_approx_get_exponent(const struct " ++ stem ++ "_approx<R> ↙
      ↪ *a) {\n\
    \  if (! a) {\n\
    \    return 0;\n\
    \  }\n\
    \  switch (a->order) {\n" ++
    unlines [ "    case " ++ show order ++ ": return " ++ stem ++ "_" ++ show ↙
      ↪ order ++ "_approx_get_exponent((const struct " ++ stem ++ "_" ++ show ↙
      ↪ order ++ "_approx<R> *) a->approx);" | order <- orders ] ++
    "  }\n\
    \  return 0;\n\
    \}\n\
    \\n\
    \template <typename R>\n\
    \const std::complex<R> *" ++ stem ++ "_approx_get_coefficients(const struct " ↙
      ↪ ++ stem ++ "_approx<R> *a) {\n\
    \  if (! a) {\n\
    \    return 0;\n\
    \  }\n\
    \  switch (a->order) {" ++
    unlines [ "    case " ++ show order ++ ": return " ++ stem ++ "_" ++ show ↙
      ↪ order ++ "_approx_get_coefficients((const struct " ++ stem ++ "_" ++ ↙
      ↪ show order ++ "_approx<R> *) a->approx);" | order <- orders ] ++
    "  }\n\
    \  return 0;\n\
    \}\n\
    \\n\
    \template <typename R>\n\
    \const std::complex<R> *" ++ stem ++ "_approx_get_dcoefficients(const struct "↙
```

```
      ↪ ++ stem ++ "_approx<R> *a) {\n\
  \  if (! a) {\n\
  \    return 0;\n\
  \  }\n\
  \  switch (a->order) {" ++
  unlines [ "    case " ++ show order ++ ": return " ++ stem ++ "_" ++ show ↙
      ↪ order ++ "_approx_get_dcoefficients((const struct " ++ stem ++ "_" ++ ↙
      ↪ show order ++ "_approx<R> *) a->approx);" | order <- orders ] ++
  "  }\n\
  \  return 0;\n\
  \}\n\
  \\n\
  \template <typename R>\n\
  \void " ++ stem ++ "_approx_do(const struct " ++ stem ++ "_approx<R> *a, const↙
      ↪  std::complex<R> dc, std::complex<R> *dz_out, std::complex<R> *ddz_out) ↙
      ↪ {\n\
  \  if (! a) {\n\
  \    *dz_out = 0;\n\
  \    *ddz_out = 0;\n\
  \    return;\n\
  \  }\n\
  \  switch (a->order) {\n" ++
  unlines [ "    case " ++ show order ++ ": " ++ stem ++ "_" ++ show order ++ "↙
      ↪ _approx_do((const struct " ++ stem ++ "_" ++ show order ++ "_approx<R> ↙
      ↪ *) a->approx, dc, dz_out, ddz_out); return;" | order <- orders ] ++
  "  }\n\
  \  *dz_out = 0;\n\
  \  *ddz_out = 0;\n\
  \  return;\n\
  \}\n\
  \\n")
  ]

codegenRef :: String -> [(Int, Phase)] -> [(FilePath, String)]
codegenRef stem ps =
  [(stem ++ "_ref.c",
  "#include <complex>\n\
  \#include <algorithm>\n\
  \#include <limits.h>\n\
  \#include <math.h>\n\
  \#include <stdbool.h>\n\
  \#include <stdint.h>\n\
  \#include <stdlib.h>\n\
  \#include <mpfr.h>\n\
  \\n\
  \#include " ++ show (stem ++ "_ref.h") ++ "\n\
  \\n\
  \static const struct {\n" ++ unlines (map struct ps) ++ "} " ++ stem ++ "↙
      ↪ _reference_spec =\n\
  \{\n" ++ intercalate "\n,\n" (map values ps) ++ "};\n\
  \\n\
  \struct " ++ stem ++ "_reference {\n\
  \  mpfr_t v[" ++ show count ++ "];\n\
  \  int n;\n\
  \};\n\
  \\n\
  \struct " ++ stem ++ "_reference *" ++ stem ++ "_reference_new(const mpfr_t cx↙
```

```
    ↪ , const mpfr_t cy, const mpfr_t zx, const mpfr_t zy, int n) {\n\
\   struct " ++ stem ++ "_reference *r = (struct " ++ stem ++ "_reference *) ↙
    ↪ malloc(sizeof(*r));\n\
\   if (! r) { return 0; }\n\
\   mpfr_prec_t p = std::max(std::max(mpfr_get_prec(cx), mpfr_get_prec(cy)), ↙
    ↪ std::max(mpfr_get_prec(zx), mpfr_get_prec(zy)));\n\
\   for (int i = 0; i < " ++ show count ++ "; ++i) {\n\
\     mpfr_init2(r->v[i], p);\n\
\     mpfr_set_si(r->v[i], 0, MPFR_RNDN);\n\
\   };\n\
\   mpfr_set(r->v[0], cx, MPFR_RNDN);\n\
\   mpfr_set(r->v[1], cy, MPFR_RNDN);\n\
\   mpfr_set(r->v[2], zx, MPFR_RNDN);\n\
\   mpfr_set(r->v[3], zy, MPFR_RNDN);\n\
\   r->n = n;\n\
\   return r;\n\
\}\n\
\\n\
\void " ++ stem ++ "_reference_delete(struct " ++ stem ++ "_reference *r) {\n\
\   for (int i = 0; i < " ++ show count ++ "; ++i) {\n\
\     mpfr_clear(r->v[i]);\n\
\   }\n\
\   free(r);\n\
\}\n\
\\n\
\void " ++ stem ++ "_reference_step(struct " ++ stem ++ "_reference *r) {\n\n"↙
    ↪ ++
unlines (map genphase ps) ++
"   r->n += 1;\n\
\}\n\
\\n\
\int " ++ stem ++ "_reference_get_n(const struct " ++ stem ++ "_reference *r) ↙
    ↪ {\n\
\   return r->n;\n\
\}\n\
\\n\
\std::complex<float> " ++ stem ++ "_reference_get_zf(const struct " ++ stem ++↙
    ↪  "_reference *r) {\n\
\   return std::complex<float>(mpfr_get_flt(r->v[2], MPFR_RNDN), mpfr_get_flt(r↙
    ↪ ->v[3], MPFR_RNDN));\n\
\}\n\
\\n\
\std::complex<double> " ++ stem ++ "_reference_get_z(const struct " ++ stem ++↙
    ↪  "_reference *r) {\n\
\   return std::complex<double>(mpfr_get_d(r->v[2], MPFR_RNDN), mpfr_get_d(r->v↙
    ↪ [3], MPFR_RNDN));\n\
\}\n\
\\n\
\std::complex<long double> " ++ stem ++ "_reference_get_zl(const struct " ++ ↙
    ↪ stem ++ "_reference *r) {\n\
\   return std::complex<long double>(mpfr_get_ld(r->v[2], MPFR_RNDN), ↙
    ↪ mpfr_get_ld(r->v[3], MPFR_RNDN));\n\
\}\n\
\\n\
\void " ++ stem ++ "_reference_get_zr(const struct " ++ stem ++ "_reference *↙
    ↪ ref, mpfr_t zx, mpfr_t zy) {\n\
\   mpfr_set_prec(zx, mpfr_get_prec(ref->v[2]));\n\
```

```
    \  mpfr_set_prec(zy, mpfr_get_prec(ref->v[3]));\n\
    \  mpfr_set(zx, ref->v[2], MPFR_RNDN);\n\
    \  mpfr_set(zy, ref->v[3], MPFR_RNDN);\n\
    \}\n\
    \\n")

    , (stem ++ "_ref.h",
    "#ifndef " ++ stem ++ "_ref_h\n\
    \#define " ++ stem ++ "_ref_h 1\n\
    \\n\
    \#include <complex>\n\
    \#include <mpfr.h>\n\
    \\n\
    \struct " ++ stem ++ "_reference;\n\
    \struct " ++ stem ++ "_reference *" ++ stem ++ "_reference_new(const mpfr_t cx, const mpfr_t cy, const mpfr_t zx, const mpfr_t zy, int n);\n\
    \void " ++ stem ++ "_reference_delete(struct " ++ stem ++ "_reference *r);\n\
    \void " ++ stem ++ "_reference_step(struct " ++ stem ++ "_reference *r);\n\
    \int " ++ stem ++ "_reference_get_n(const struct " ++ stem ++ "_reference *r);\n\
    \std::complex<float> " ++ stem ++ "_reference_get_zf(const struct " ++ stem ++ "_reference *r);\n\
    \std::complex<double> " ++ stem ++ "_reference_get_z(const struct " ++ stem ++ "_reference *r);\n\
    \std::complex<long double> " ++ stem ++ "_reference_get_zl(const struct " ++ stem ++ "_reference *r);\n\
    \void " ++ stem ++ "_reference_get_zr(const struct " ++ stem ++ "_reference *ref, mpfr_t zx, mpfr_t zy);\n\
    \\n\
    \#endif\n")]

    where
      int
        | count <= 2^( 8 :: Int) = "uint8_t"
        | count <= 2^(16 :: Int) = "uint16_t"
        | count <= 2^(32 :: Int) = "uint32_t"
        | otherwise      = "uint64_t"
      struct (i, Phase _ ras@((_, args):_)) = "  " ++ int ++ " p" ++ show i ++ "[" ++ show (length ras) ++ "][" ++ show (length args + 1) ++ "];"
      values (_, Phase _ ras) = "{\n" ++ intercalate ",\n" (map value ras) ++ "\n}"
      value (Right res, args) = "{ " ++ show res ++ " , " ++ intercalate " , " (map show (rights args) ++ map show (lefts args)) ++ " }"
      count = 1 + maximum [ i | (_, Phase _ ras) <- ps, (res, args) <-ras, Right i <- res : args ]

      genphase (p, Phase op ras) =
        "  #pragma omp parallel for\n\
        \  for (int i = 0; i < " ++ show (length ras) ++ "; ++i) {\n" ++
        ( case op of
            OpAdd  -> o3  "mpfr_add"
            OpAddI -> o3i "mpfr_add_si"
            OpMulI -> o3i "mpfr_mul_si"
            OpNeg  -> o2  "mpfr_neg"
            OpMul2 -> o3i "mpfr_mul_2si"
            OpMul  -> o3  "mpfr_mul"
            OpSqr  -> o2  "mpfr_sqr"
```

```
            OpSet  -> o2  "mpfr_set"
            _ -> error $ "ref.genphase: " ++ show op ) ++
        "  }\n"
        where
          o3  s = "  " ++ s ++ "\n\
                  \    ( r->v[" ++ stem ++ "_reference_spec.p" ++ show p ++ "[i↙
                    ↪ ][0]]\n\
                  \    , r->v[" ++ stem ++ "_reference_spec.p" ++ show p ++ "[i↙
                    ↪ ][1]]\n\
                  \    , r->v[" ++ stem ++ "_reference_spec.p" ++ show p ++ "[i↙
                    ↪ ][2]]\n\
                  \    , MPFR_RNDN\n\
                  \    );\n"
          o3i s = "  " ++ s ++ "\n\
                  \    ( r->v[" ++ stem ++ "_reference_spec.p" ++ show p ++ "[i↙
                    ↪ ][0]]\n\
                  \    , r->v[" ++ stem ++ "_reference_spec.p" ++ show p ++ "[i↙
                    ↪ ][1]]\n\
                  \    ,      " ++ stem ++ "_reference_spec.p" ++ show p ++ "[i↙
                    ↪ ][2]\n\
                  \    , MPFR_RNDN\n\
                  \    );\n"
          o2  s = "  " ++ s ++ "\n\
                  \    ( r->v[" ++ stem ++ "_reference_spec.p" ++ show p ++ "[i↙
                    ↪ ][0]]\n\
                  \    , r->v[" ++ stem ++ "_reference_spec.p" ++ show p ++ "[i↙
                    ↪ ][1]]\n\
                  \    , MPFR_RNDN\n\
                  \    );\n"

main'' :: String -> Integer -> E -> [(FilePath, String)]
main'' stem n f = codegen n stem . zip [0..] . splitAdds . compact . inplace . ↙
    ↪ reverse $ finalize : phases
  where
    finalize = Phase OpSet $ zip (map (Right . (variables M.!)) vs) (map ((:[]) ↙
        ↪ . Right . (variables M.!)) es)
    initial = fst $ execRWS initialize () M.empty
    initialize = tell () >> mapM_ temporary ([var CRe, var CIm] ++ vs ++ es)
    (variables, phases) = execRWS (parallel es) () initial
    (vs, es) = unzip [ (var v, expr e) | (v, e) <- ves ]
    var = compile . ovar
    expr = compile . psum . osum
    ves =
      [ (ZRe, fre)
      , (ZIm, fim)
      , (DiffZRe, gre)
      , (DiffZIm, gim)
      ] ++ concat
      [ [ (ARe i, are)
        , (AIm i, aim)
        , (DiffARe j, bre)
        , (DiffAIm j, bim)
        ]
      | ((i, Pair (are, aim)), (j, Pair (bre, bim))) <- ies `zip` jes
      ]
    Pair (fre, fim) = cnormalize $ f
    Pair (gre, gim) = cnormalize . normalize . diff $ f
```

```
    ies
      = take (fromInteger n)
      . map (fmap cnormalize)
      . collate
      . normalize
      . sub (series n)
      . normalize
      . perturbed
      $ f
    jes
      = take (fromInteger n)
      . map (fmap cnormalize)
      . collate
      . normalize
      . diff
      . normalize
      . sub (series n)
      . normalize
      . perturbed
      $ f

main :: IO ()
main = mapM_ (\(f, s) -> writeFile f s >> print f >> hFlush stdout) . main' "z2c↙
    ↪ " [4,6,8,12,16,24,32,48,64] $ Z^(2::Int) + C
```

14 c/lib/Makefile

```
# mandelbrot-perturbator -- efficient deep zooming for Mandelbrot sets
# Copyright (C) 2015,2016,2017 Claude Heiland-Allen
# License GPL3+ http://www.gnu.org/licenses/gpl.html

prefix ?= $(HOME)/opt
CC ?= g++

PROJECT := mandelbrot-perturbator
PKGCONFIG := PKG_CONFIG_PATH="$(prefix)/lib/pkgconfig" pkg-config
COMPILE := $(CC) -xc++ -std=c++11 -Wall -Wextra -pedantic -fPIC -Ofast -march=↙
    ↪ native -fopenmp -pipe -g -MMD -I../include -c
LINK    := $(CC) -Ofast -fopenmp -shared -g
AR      := ar
LIBRARY := lib$(PROJECT)
SOURCES := perturbator.c
OBJECTS := $(patsubst %.c,%.o,$(SOURCES))
DEPENDS := $(patsubst %.o,%.d,$(OBJECTS))

all: $(LIBRARY).a $(LIBRARY).so pkgconfig/$(PROJECT).pc

clean:
	@echo "CLEAN" ; rm -f $(OBJECTS) $(DEPENDS) $(LIBRARY).a $(LIBRARY).so ↙
	    ↪ pkgconfig/$(PROJECT).pc

install: $(LIBRARY).a $(LIBRARY).so ../include/$(PROJECT).h pkgconfig/$(PROJECT)↙
    ↪ .pc
	install -d "$(prefix)/include" "$(prefix)/lib" "$(prefix)/lib/pkgconfig"
	install -m 644 -t "$(prefix)/include" ../include/$(PROJECT).h
	install -m 644 -t "$(prefix)/lib" $(LIBRARY).a $(LIBRARY).so
	install -m 644 -t "$(prefix)/lib/pkgconfig" pkgconfig/$(PROJECT).pc
```

```
$(LIBRARY).a: $(OBJECTS)
	@echo "A       $@" ; $(AR) -rs $@ $^ || ( echo "ERROR   $(AR) -rs $@ $^" && false )

$(LIBRARY).so: $(OBJECTS)
	@echo "SO      $@" ; $(LINK) -o $@ $^ -lmpc -lmpfr -lgmp -lm || ( echo "ERROR   $(LINK) -o $@ $^ -lmpc -lmpfr -lgmp -lm" && false )

%.o: %.c
	@echo "O       $@" ; $(COMPILE) -o $@ $< || ( echo "ERROR   $(COMPILE) -o $@ $<" && false )

pkgconfig/$(PROJECT).pc: pkgconfig/$(PROJECT).pc.in
	@echo "PC      $@" ; ( echo "prefix=$(prefix)" ; cat pkgconfig/$(PROJECT).pc.in ) > pkgconfig/$(PROJECT).pc || ( echo 'ERROR    ( echo "prefix=$(prefix)" ; cat pkgconfig/$(PROJECT).pc.in ) > pkgconfig/$(PROJECT).pc' && false )

.SUFFIXES:
.PHONY: all clean install

-include $(DEPENDS)
```

15 c/lib/perturbator.c

```
// mandelbrot-perturbator -- efficient deep zooming for Mandelbrot sets
// Copyright (C) 2015-2019 Claude Heiland-Allen
// License GPL3+ http://www.gnu.org/licenses/gpl.html

#include <cassert>
#include <cmath>
#include <cstdbool>
#include <cstdint>
#include <cstdio>
#include <cstdlib>
#include <cstring>

#include <atomic>

#include <pthread.h>

#include <mpc.h>

#include "mandelbrot-perturbator.h"

#include "complex.hpp"
#include "edouble.cc"

static const double twopi = 6.283185307179586;

float mpfr_get(const mpfr_t &op, mpfr_rnd_t rnd, float dummy) {
  (void) dummy;
  return mpfr_get_flt(op, rnd);
}

double mpfr_get(const mpfr_t &op, mpfr_rnd_t rnd, double dummy) {
```

```
  (void) dummy;
  return mpfr_get_d(op, rnd);
}

long double mpfr_get(const mpfr_t &op, mpfr_rnd_t rnd, long double dummy) {
  (void) dummy;
  return mpfr_get_ld(op, rnd);
}

edouble mpfr_get(const mpfr_t &op, mpfr_rnd_t rnd, edouble dummy) {
  (void) dummy;
  (void) rnd;
  return edouble(op);
}

static inline bool isfinite(double x)
{
  return !(std::isnan(x) || std::isinf(x));
}

static inline bool isfinite(long double x)
{
  return !(std::isnan(x) || std::isinf(x));
}

template< typename R >
static inline bool isfinite(const complex< R > &z)
{
  return isfinite(real(z)) && isfinite(imag(z));
}

enum float_type
  { ft_float
  , ft_double
  , ft_long_double
  , ft_edouble
  };

enum render_method
  { rm_plain
  , rm_perturb
  };

#include "z2c.c"

extern "C" {

enum m_shape { m_cardioid, m_circle };
typedef enum m_shape m_shape;

enum m_newton { m_failed, m_stepped, m_converged };
typedef enum m_newton m_newton;

extern int m_r_box_period_do(const mpc_t center, const mpfr_t radius, int ↙
    ↳ maxperiod);
extern m_newton m_r_nucleus(mpc_t c_out, const mpc_t c_guess, int period, int ↙
    ↳ maxsteps);
```

```
extern m_shape m_r_shape(const mpc_t nucleus, int period);
extern int m_r_domain_size(mpfr_t size, const mpc_t nucleus, int period);

}

// don't use float, derivative overflows too easily...
#define EXP_THRESHOLD_PLAIN_FLOAT (-21)
#define EXP_THRESHOLD_PLAIN_DOUBLE (-50)
#define EXP_THRESHOLD_PLAIN_LONG_DOUBLE (sizeof(long double) == sizeof(double) ? EXP_THRESHOLD_PLAIN_DOUBLE : -60)
#define EXP_THRESHOLD_PERTURB_FLOAT -120
#define EXP_THRESHOLD_PERTURB_DOUBLE (-1016)
#define EXP_THRESHOLD_PERTURB_LONG_DOUBLE (sizeof(long double) == sizeof(double) ? EXP_THRESHOLD_PERTURB_DOUBLE : -16376)
#define EXP_THRESHOLD_PERTURB_EDOUBLE (-(1LL<<62LL))

struct series_node {
  struct series_node *next;
  int64_t exponent;
  int iters;
  mpc_t z;
  struct z2c_approx *approx;
};


template <typename R>
struct reference;

struct perturbator {
  int workers;
  int width;
  int height;
  int detect_glitches;
  int correct_glitches;
  int approx_skip;
  int maxiters;
  double escape_radius;
  double glitch_threshold;
  int precision;
  int maxchunk;
  mpc_t center;
  mpfr_t radius;
  double escape_radius_2;
  double log_escape_radius_2;
  int newton_steps_root;
  int newton_steps_child;
  int order;
  int64_t threshold;
  int logging;

  // cache
  mpc_t last_reference;
  int last_period;
  struct z2c_series *series;
  struct series_node *nodes;

  // struct pixel<R>
```

```
  void *pixel_data;
  size_t pixel_data_size; // bytes

  uint32_t *index_data;
  size_t pixel_count; // width * height

  pthread_mutex_t mutex;
  pthread_cond_t cond, cond_done;
  pthread_t *threads;

  int volatile active_workers;
  bool volatile running;

  int volatile queue_id;
  int volatile start_id;

  std::atomic<int> scanline;

  std::atomic<int> done_pixels;

  enum float_type ft;
  enum render_method rm;
  void *volatile refs;

  // [j * width + i]
  int32_t *output_dwell_n;
  float   *output_dwell_f;
  float   *output_dwell_a;
  float   *output_distance;
  int32_t *output_domain_n;
  float   *output_domain_x;
  float   *output_domain_y;
};

extern const int32_t *perturbator_get_dwell_n (struct perturbator *img) { return img->output_dwell_n; }
extern const float   *perturbator_get_dwell_f (struct perturbator *img) { return img->output_dwell_f; }
extern const float   *perturbator_get_dwell_a (struct perturbator *img) { return img->output_dwell_a; }
extern const float   *perturbator_get_distance(struct perturbator *img) { return img->output_distance; }
extern const int32_t *perturbator_get_domain_n(struct perturbator *img) { return img->output_domain_n; }
extern const float   *perturbator_get_domain_x(struct perturbator *img) { return img->output_domain_x; }
extern const float   *perturbator_get_domain_y(struct perturbator *img) { return img->output_domain_y; }

static bool image_running(struct perturbator *img) {
  return img->running;
}

extern int perturbator_active(struct perturbator *img) {
  pthread_mutex_lock(&img->mutex);
  bool active = img->active_workers;
  pthread_mutex_unlock(&img->mutex);
```

```
  return active;
}

#define LOG_VIEW  1
#define LOG_QUEUE 2
#define LOG_CACHE 4

#define image_log(img, aspect, ...) do{ \
  if ((img)->logging & (aspect)) { mpfr_fprintf(stderr, __VA_ARGS__); } \
  }while(0)


struct series_node *image_cached_approx(struct perturbator *img, bool reused, ↙
    ↳ const mpc_t c, int64_t exponent, mpc_t z, int *iter);


int mpfr_add(mpfr_t &rop, const mpfr_t &op1, float op2, mpfr_rnd_t rnd) {
  return mpfr_add_d(rop, op1, op2, rnd);
}

int mpfr_add(mpfr_t &rop, const mpfr_t &op1, double op2, mpfr_rnd_t rnd) {
  return mpfr_add_d(rop, op1, op2, rnd);
}

int mpfr_add(mpfr_t &rop, const mpfr_t &op1, long double op2, mpfr_rnd_t rnd) {
  mpfr_t tmp;
  mpfr_init2(tmp, 64);
  to_mpfr(op2, tmp);
  int r = mpfr_add(rop, op1, tmp, rnd);
  mpfr_clear(tmp);
  return r;
}

int mpfr_add(mpfr_t &rop, const mpfr_t &op1, edouble op2, mpfr_rnd_t rnd) {
  mpfr_t tmp;
  mpfr_init2(tmp, 53);
  to_mpfr(op2, tmp);
  int r = mpfr_add(rop, op1, tmp, rnd);
  mpfr_clear(tmp);
  return r;
}

template <typename R>
void to_mpc(const complex<R> &from, mpc_t &to) {
  to_mpfr(real(from), mpc_realref(to));
  to_mpfr(imag(from), mpc_imagref(to));
}

template <typename T>
int mpc_add(mpc_t &rop, const mpc_t &op1, const complex<T> &op2, mpc_rnd_t rnd) ↙
    ↳ {
  (void) rnd;
          mpfr_add(mpc_realref(rop), mpc_realref(op1), real(op2), MPFR_RNDN);
  return  mpfr_add(mpc_imagref(rop), mpc_imagref(op1), imag(op2), MPFR_RNDN);
}

template <typename T>
```

```
int mpc_div(mpc_t &rop, const mpc_t &op1, const complex<T> &op2, mpc_rnd_t rnd) {
  mpc_t tmp;
  mpc_init2(tmp, 64);
  to_mpc(op2, tmp);
  int r = mpc_div(rop, op1, tmp, rnd);
  mpc_clear(tmp);
  return r;
}

template <typename T>
complex<T> mpc_get(const mpc_t &op, mpc_rnd_t rnd, T dummy) {
  (void) rnd;
  return complex<T>(mpfr_get(mpc_realref(op), MPFR_RNDN, dummy), mpfr_get(mpc_imagref(op), MPFR_RNDN, dummy));
}

template <typename R>
struct pixel {
  complex<R> c;
  complex<R> z;
  complex<R> dz;
  complex<R> mz;
  uint32_t iters;
};

template <typename R>
static int cmp_pixel_by_iters_asc_r(const void *a, const void *b, void *p) {
  const struct pixel<R> *q = (const struct pixel<R> *) p;
  const uint32_t *x = (const uint32_t *) a;
  const uint32_t *y = (const uint32_t *) b;
  if (q[*x].iters < q[*y].iters) { return -1; }
  if (q[*x].iters > q[*y].iters) { return  1; }
/*
  R x2 = norm(x->z);
  R y2 = norm(y->z);
  if (x2 < y2) { return -1; }
  if (x2 > y2) { return  1; }
*/
  return 0;
}

template <typename R>
struct reference {
  struct reference<R> *next;

  int has_parent;
  mpc_t parent_c;
  mpc_t parent_z;

  int queue_id;
  int start_id;

  int period;
  int iters;
  mpc_t c;
  mpc_t z;
```

```
  complex<R> z_d_old;
  complex<R> z_d;
  R z_d2eM6;

/*
  [...][src.....................][...]
       |            |            |
       |            V            |
  [...][active->][...][<-glitch][...]
*/
  struct pixel<R> *pixel_data; // owned by context, ordered by pixel index
  uint32_t *index_data[2]; // owned by context, this ref owns the src region ↙
      ↳ below
  int index_src; // 0 or 1
  int index_src_offset; // index into index_data[index_src];
  int index_src_minix;
  int index_src_count;
};

template <typename R>
static void reference_release(struct reference<R> *ref) {
  if (ref) {
    ref->pixel_data = 0;
    ref->index_data[0] = 0;
    ref->index_data[1] = 0;
    if (ref->has_parent) {
      mpc_clear(ref->parent_c);
      mpc_clear(ref->parent_z);
      ref->has_parent = 0;
    }
    mpc_clear(ref->c);
    mpc_clear(ref->z);
    free(ref);
  }
}


template <typename R>
int image_dequeue_plain(struct perturbator *img)
{
  pthread_mutex_lock(&img->mutex);
  img->active_workers -= 1;
  int j = img->scanline++;
  if (img->running && j < img->height) {
    img->active_workers += 1;
  }
  else
  {
    j = -1;
    if (! img->active_workers)
    {
      img->running = false;
    }
  }
  if (! img->running)
  {
    pthread_cond_broadcast(&img->cond_done);
```

```
  }
  pthread_mutex_unlock(&img->mutex);
  return j;
}

template <typename R>
static struct reference<R> *image_dequeue(struct perturbator *img, struct reference<R> *ref) {
  pthread_mutex_lock(&img->mutex);
//  assert(img->ft == FT);
  if (ref) {
    image_log(img, LOG_QUEUE, "%8d DONE   %8d\n", ref->start_id, ref->queue_id);
  }
  // release the old reference
  reference_release(ref);
  // if no workers are working and the queue is empty, it would be empty forever
  img->active_workers -= 1;
  while (img->running && ! img->refs) {
    if (! img->active_workers) {
      img->running = false;
      pthread_cond_broadcast(&img->cond);
    } else {
      pthread_cond_wait(&img->cond, &img->mutex);
    }
  }

  if (img->running && img->refs) {
    // still work to do
    ref = (struct reference<R> *) img->refs;
    img->refs = ref->next;
    ref->start_id = (img->start_id += 1);
    img->active_workers += 1;
    image_log(img, LOG_QUEUE, "%8d START  %8d%8d%8d%16d\n", ref->start_id, ref->queue_id, ref->iters, ref->period, ref->index_src_count);
  } else {
    ref = 0;
  }
  if (! img->running)
  {
    pthread_cond_broadcast(&img->cond_done);
  }
  pthread_mutex_unlock(&img->mutex);
  return ref;
}

template <typename R>
static void image_enqueue(struct perturbator *img, struct reference<R> *ref) {
  pthread_mutex_lock(&img->mutex);
//  assert(img->ft == FT);
  ref->queue_id = (img->queue_id += 1);
  image_log(img, LOG_QUEUE, "         QUEUE  %8d%8d%8d%16d\n", ref->queue_id, ref->iters, ref->period, ref->index_src_count);
  // initialize
  if (ref->has_parent) {
    mpc_init2(ref->z, img->precision);
    mpc_set_ui_ui(ref->z, 0, 0, MPC_RNDNN);
    mpc_init2(ref->c, img->precision);
```

```
      mpc_set_ui_ui(ref->c, 0, 0, MPC_RNDNN);
    }
    // link into reference queue (sorted by count descending)
    struct reference<R> *before = 0;
    struct reference<R> *after = (struct reference<R> *) img->refs;
    while (after && after->index_src_count > ref->index_src_count) {
      before = after;
      after = after->next;
    }
    ref->next = after;
    if (before) {
      before->next = ref;
    } else {
      img->refs = ref;
    }
    // wake waiting workers
    pthread_cond_broadcast(&img->cond);
    pthread_mutex_unlock(&img->mutex);
}

template <typename R>
static void *image_worker(void *arg);

template <typename R>
static void *image_worker_plain(void *arg);

template <typename R>
void perturbator_start_internal_plain(struct perturbator *img) {
  img->threads = (pthread_t *) calloc(1, img->workers * sizeof(*img->threads));
  img->running = true;
  img->scanline = 0;
  pthread_mutex_lock(&img->mutex);
  img->active_workers = img->workers;
  for (int i = 0; i < img->workers; ++i) {
    pthread_create(&img->threads[i], 0, image_worker_plain<R>, img);
  }
  pthread_mutex_unlock(&img->mutex);
}

template <typename R>
void perturbator_start_internal(struct perturbator *img) {

  // initialize index buffer
  uint32_t count = img->width * img->height;
  if (count != img->pixel_count)
  {
    if (img->index_data)
    {
      free(img->index_data);
    }
    img->index_data = (uint32_t *) calloc(1, 2 * count * sizeof(uint32_t));
    img->pixel_count = count;
  }
  #pragma omp parallel for
  for (uint32_t i = 0; i < count; ++i)
  {
    img->index_data[i] = i;
```

```
  }

  // initialize iteration buffer
  size_t bytes = count * sizeof(struct pixel<R>);
  if (bytes != img->pixel_data_size)
  {
    if (img->pixel_data)
    {
      free(img->pixel_data);
    }
    img->pixel_data = calloc(1, bytes);
    img->pixel_data_size = bytes;
  }
  struct pixel<R> *p = (struct pixel<R> *) img->pixel_data;
  R vdiameter = R(2.0) * mpfr_get(img->radius, MPFR_RNDN, R(0));
  R hdiameter = img->width * vdiameter / img->height;
  #pragma omp parallel for
  for (int j = 0; j < img->height; ++j) {
    R y = R((j + 0.5) / img->height - 0.5) * vdiameter;
    for (int i = 0; i < img->width; ++i) {
      R x = R((i + 0.5) / img->width - 0.5) * hdiameter;
      complex<R> c(x, -y);
      int index = j * img->width + i;
      p[index].c = c;
      p[index].z = 0;
      p[index].dz = 0;
      p[index].mz = complex<R>(R(1.0 / 0.0), R(0.0));
      p[index].iters = 0;
    }
  }

  // create reference
//  assert(img->ft == FT);
  struct reference<R> *ref = (struct reference<R> *) calloc(1, sizeof(*ref));
  mpc_init2(ref->c, img->precision);
  mpc_init2(ref->z, img->precision);
  mpc_set(ref->c, img->center, MPC_RNDNN);
  mpc_set_ui_ui(ref->z, 0, 0, MPC_RNDNN);
  ref->iters = 0;
  ref->period = 0;
  ref->pixel_data = (struct pixel<R> *) img->pixel_data;
  ref->index_data[0] = img->index_data;
  ref->index_data[1] = img->index_data + img->pixel_count;
  ref->index_src = 0;
  ref->index_src_offset = 0;
  ref->index_src_minix = 0;
  ref->index_src_count = img->pixel_count;

  img->threads = (pthread_t *) calloc(1, img->workers * sizeof(*img->threads));
  img->running = true;
  image_enqueue(img, ref);
  pthread_mutex_lock(&img->mutex);
  img->active_workers = img->workers;
  for (int i = 0; i < img->workers; ++i) {
    pthread_create(&img->threads[i], 0, image_worker<R>, img);
  }
  pthread_mutex_unlock(&img->mutex);
```

```
}

template <typename R>
void release_refs(struct perturbator *img) {
//  assert(img->ft == FT);
  for (struct reference<R> *ref = (struct reference<R> *) img->refs; ref; ) {
    struct reference<R> *next = ref->next;
    reference_release(ref);
    ref = next;
  }
  img->refs = 0;
}

template<typename R>
void rebase_to_new_reference(int offset, int count, const uint32_t *ix, pixel<R>*px, complex<R> dc) {
  for (int k = 0; k < count; ++k) {
    px[ix[offset + k]].c += dc;
  }
}

float ldexp(float x, int64_t e) { return std::ldexp(x, e); } // FIXME e range
double ldexp(double x, int64_t e) { return std::ldexp(x, e); } // FIXME e range
long double ldexp(long double x, int64_t e) { return std::ldexp(x, e); } // FIXME e range

template<typename R>
void initialize_from_series_approximation(int offset, int count, const uint32_t *ix, pixel<R> *px, complex<R> dc, const z2c_approx_t<edouble> *approx, int64_t e) {
  int o = approx->order;
  complex<R> *a = (complex<R> *) malloc((2 * o + 1) * sizeof(*a));
  for (int i = 0; i < o; ++i) {
    edouble re = ldexp(real(approx->a[i]), e * (i + 1));
    edouble im = ldexp(imag(approx->a[i]), e * (i + 1));
    a[i] = complex<R>(to_R(re, R(0)), to_R(im, R(0)));
  }
  for (int i = 0; i < o; ++i) {
    edouble re = (i + 1) * ldexp(real(approx->a[i]), e * i);
    edouble im = (i + 1) * ldexp(imag(approx->a[i]), e * i);
    a[i+o] = complex<R>(to_R(re, R(0)), to_R(im, R(0)));
  }
  for (int k = 0; k < count; ++k) {
    uint32_t k2 = ix[offset + k];
    px[k2].c += dc;
    complex<R> z(0), dz(0), c(ldexp(real(px[k2].c), -e), ldexp(imag(px[k2].c), -e));
    complex<R> cn(1);
    for (int i = 0; i < o; ++i) {
      dz += a[i+o] * cn;
      cn *= c;
      z += a[i] * cn;
    }
    px[k2].z = z;
    px[k2].dz = dz;
  }
  free(a);
```

```
}

template <typename R>
static void *image_worker(void *threadarg) {
  struct perturbator *img = (struct perturbator *) threadarg;
  image_log(img, LOG_QUEUE, "          ENTER\n");

  pthread_mutex_lock(&img->mutex);

  int detect_glitches = img->detect_glitches;
  int correct_glitches = img->correct_glitches;
  int maxiters = img->maxiters;
  R escape_radius_2 = img->escape_radius_2;
  R log_escape_radius_2 = img->log_escape_radius_2;
  int newton_steps_root = img->newton_steps_root;
//  int newton_steps_child = img->newton_steps_child;
  int maxchunk = img->maxchunk;
  R glitch_threshold = img->glitch_threshold;
  int precision = img->precision;
  mpc_t center;
  mpc_init2(center, mpc_get_prec(img->center));
  mpc_set(center, img->center, MPC_RNDNN);
  mpfr_t radius;
  mpfr_init2(radius, 53);
  mpfr_set(radius, img->radius, MPFR_RNDN);
  R pixel_spacing = mpfr_get(radius, MPFR_RNDN, R(0)) * R(2.0 / img->height);
  mpc_t last_reference;
  mpc_init2(last_reference, mpc_get_prec(img->last_reference));
  mpc_set(last_reference, img->last_reference, MPC_RNDNN);
  int last_period = img->last_period;

  pthread_mutex_unlock(&img->mutex);

  mpc_t *z_hi = (mpc_t *) calloc(1, (1 + maxchunk) * sizeof(*z_hi));
  for (int k = 0; k <= maxchunk; ++k) {
    mpc_init2(z_hi[k], precision);
  }
  complex<R> *z_d = (complex<R> *) calloc(1, (1 + maxchunk) * sizeof(*z_d));
  R *z_size = (R *) calloc(1, (1 + maxchunk) * sizeof(*z_size));

  struct reference<R> *ref = 0;
  while ( (ref = image_dequeue(img, ref)) ) {

    // find reference atom
    if (ref->has_parent) {

      // using .z .dz to do one Newton step instead of using nucleus...
      // nucleus = c - z / dz
      uint32_t k = ref->index_data[ref->index_src][ref->index_src_minix];
      mpc_add(ref->c, ref->parent_c, ref->pixel_data[k].c, MPC_RNDNN);
      mpc_add(ref->z, ref->parent_z, ref->pixel_data[k].z, MPC_RNDNN);
      mpc_div(ref->z, ref->z,        ref->pixel_data[k].dz, MPC_RNDNN);
      if (mpfr_number_p(mpc_realref(ref->z)) && mpfr_number_p(mpc_imagref(ref->z↙
          ↳ ))) {
        mpc_sub(ref->c, ref->c, ref->z, MPC_RNDNN);
      }
      // compute rebase offset
```

```
        mpc_sub(ref->z, ref->parent_c, ref->c, MPC_RNDNN);
635     complex<R> dc = -mpc_get(ref->z, MPC_RNDNN, R(0));
        mpc_set_ui_ui(ref->z, 0, 0, MPC_RNDNN);
        // rebase pixels to new reference
        rebase_to_new_reference(ref->index_src_offset, ref->index_src_count, ref->↙
            ↳ index_data[ref->index_src], ref->pixel_data, dc);

640   } else {

        int64_t exponent = mpfr_get_exp(radius);
        // find an appropriate initial reference
        bool ok = false;
645     bool reused = false;
        int period = 0;
        mpc_t nucleus;
        mpc_init2(nucleus, precision);
        mpc_set_ui_ui(nucleus, 0, 0, MPC_RNDNN);
650     if (! ok) {
          // try reuse
          mpc_t delta;
          mpc_init2(delta, 53);
          mpfr_t delta2, radius2;
655       mpfr_init2(delta2, 53);
          mpfr_init2(radius2, 53);
          mpfr_sqr(radius2, radius, MPFR_RNDN);
          mpfr_mul_d(radius2, radius2, 65536, MPFR_RNDN);
          mpc_sub(delta, last_reference, center, MPC_RNDNN);
660       mpc_norm(delta2, delta, MPFR_RNDN);
          if (mpfr_less_p(delta2, radius2)) {
            mpc_set(nucleus, last_reference, MPC_RNDNN);
            period = last_period;
            if (! mpfr_zero_p(delta2)) {
665           exponent = std::max(exponent, int64_t(mpfr_get_exp(delta2) / 2));
            }
            ok = true;
            reused = true;
            image_log(img, LOG_CACHE, "          REUSE\n");
670       }
          mpc_clear(delta);
          mpfr_clear(delta2);
          mpfr_clear(radius2);
        }
675     if (! ok) {
          // try box period
          period = m_r_box_period_do(center, radius, maxiters);
          if (period) {
            m_r_nucleus(nucleus, center, period, newton_steps_root);
680         if (m_cardioid == m_r_shape(nucleus, period)) {
              ok = true;
              image_log(img, LOG_CACHE, "          BOXED                    %8d\n", ↙
                  ↳ period);
            }
          }
685     }
        if (! ok) {
          // try modified partials
          image_log(img, LOG_CACHE, "          SEARCH\n");
```

```
        mpc_t z, dc0;
690     mpc_init2(z, precision);
        mpc_set_ui_ui(z, 0, 0, MPC_RNDNN);
        mpc_init2(dc0, 53);
        mpfr_t z2, mz2, dc2, radius2;
        mpfr_init2(z2, 53);
695     mpfr_init2(mz2, 53);
        mpfr_set_d(mz2, 65536, MPFR_RNDN);
        mpfr_init2(dc2, 53);
        mpfr_init2(radius2, 53);
        mpfr_sqr(radius2, radius, MPFR_RNDN);
700     mpfr_mul_d(radius2, radius2, 65536, MPFR_RNDN);

        for (period = 1; period < maxiters; ++period) {
          if (! image_running(img)) {
            break;
705       }
          mpc_sqr(z, z, MPC_RNDNN);
          mpc_add(z, z, center, MPC_RNDNN);
          mpc_norm(z2, z, MPFR_RNDN);
          if (mpfr_get_d(z2, MPFR_RNDN) > escape_radius_2) {
710         break;
          }
          mpfr_mul_2si(z2, z2, -period, MPFR_RNDN);
          if (mpfr_less_p(z2, mz2)) {
            mpfr_set(mz2, z2, MPFR_RNDN);
715         m_r_nucleus(ref->z, ref->c, period, newton_steps_root);
            if (m_cardioid == m_r_shape(ref->z, period)) {
              mpc_sub(dc0, ref->c, ref->z, MPC_RNDNN);
              mpc_norm(dc2, dc0, MPFR_RNDN);
              if (mpfr_less_p(dc2, radius2)) {
720             mpc_set(nucleus, ref->z, MPC_RNDNN);
                ok = true;
                image_log(img, LOG_CACHE, "          FOUND                    %8d\n↙
                    ↳ ", period);
                break;
              }
725         }
          }
        }
        mpc_clear(z);
        mpc_clear(dc0);
730     mpfr_clear(z2);
        mpfr_clear(mz2);
        mpfr_clear(dc2);
        mpfr_clear(radius2);
      }
735   if (! ok && image_running(img)) {
        // reuse fallback
        mpc_t delta;
        mpc_init2(delta, 53);
        mpfr_t delta2, radius2;
740     mpfr_init2(delta2, 53);
        mpfr_init2(radius2, 53);
        mpfr_sqr(radius2, radius, MPFR_RNDN);
        mpc_sub(delta, last_reference, center, MPC_RNDNN);
        mpc_norm(delta2, delta, MPFR_RNDN);
```

```
745           mpfr_add(delta2, delta2, radius2, MPFR_RNDN);
              mpc_set(nucleus, last_reference, MPC_RNDNN);
              period = last_period;
              exponent = std::max(exponent, int64_t(mpfr_get_exp(delta2) / 2));
              ok = true;
750           reused = true;
              mpc_clear(delta);
              mpfr_clear(delta2);
              mpfr_clear(radius2);
              image_log(img, LOG_CACHE, "          REUSE FALLBACK\n");
755         }

            if (ok) {
              pthread_mutex_lock(&img->mutex);
              mpc_set_prec(img->last_reference, img->precision);
760           mpc_set(img->last_reference, nucleus, MPC_RNDNN);
              img->last_period = period;
              pthread_mutex_unlock(&img->mutex);
            }
            mpc_set(ref->c, nucleus, MPC_RNDNN);
765         mpc_sub(nucleus, ref->c, center, MPC_RNDNN);
            complex<R> dc = -mpc_get(nucleus, MPC_RNDNN, R(0));
            mpc_clear(nucleus);
            mpc_set_ui_ui(ref->z, 0, 0, MPC_RNDNN);
            ref->z_d_old = 0;
770         ref->z_d = 0;
            struct series_node *snode = image_cached_approx(img, reused, ref->c, ↙
                ↳ exponent, ref->z, &ref->iters);
            ref->z_d = mpc_get(ref->z, MPC_RNDNN, R(0));

            image_log(img, LOG_CACHE, "          SERIES          %8d\n", ref->iters);
775
            // rebase pixels to new reference with approximation
            initialize_from_series_approximation(ref->index_src_offset, ref->↙
                ↳ index_src_count, ref->index_data[ref->index_src], ref->pixel_data, ↙
                ↳ dc, snode->approx->u.e, ::exponent(pixel_spacing));
            image_log(img, LOG_CACHE, "          APPROX          %8d\n", ref->iters);

780       } // if parent

          // prepare for output pixels
          int chunk = ref->index_src_count > maxchunk ? maxchunk : ref->↙
              ↳ index_src_count;
          chunk = chunk > 0 ? chunk : 1; // FIXME paranoid
785 //    int start_id = ref->start_id;

          for (int iters = ref->iters; iters < maxiters; iters += chunk) {
            if (! image_running(img)) {
              break;
790         }

            // compute ref chunk
            mpc_set(z_hi[0], ref->z, MPC_RNDNN);
            z_d[0] = ref->z_d;
795         for (int k = 1; k <= chunk; ++k) {
              mpc_sqr(ref->z, ref->z, MPC_RNDNN);
              mpc_add(ref->z, ref->z, ref->c, MPC_RNDNN);
```

```
        mpc_set(z_hi[k], ref->z, MPC_RNDNN);
        z_d[k] = mpc_get(ref->z, MPC_RNDNN, R(0));
    }
    ref->z_d = z_d[chunk];
    for (int k = 0; k <= chunk; ++k) {
        z_size[k] = norm(z_d[k]) * glitch_threshold;
    }
    complex<R> ref_c = mpc_get(ref->c, MPC_RNDNN, R(0));

    // advance pixels
    int src = ref->index_src;
    int dst = 1 - src;
    int input_count = ref->index_src_count;
    int active_start = ref->index_src_offset;
    int glitch_end   = ref->index_src_offset + ref->index_src_count;
    std::atomic<int> active_count(0);
    std::atomic<int> glitch_count(0);

    #pragma omp parallel for if (input_count >= 1024)
    for (int k = 0; k < input_count; ++k) {
        if (image_running(img)) {
            int index = ref->index_src_offset + k;
            index = ref->index_data[src][index];

            struct pixel<R> *in = &ref->pixel_data[index];
            complex<R> mz = in->mz;
            complex<R> dz = in->dz;
            complex<R> z = in->z;
            complex<R> c = in->c;
            complex<R> rz = z_d[0] + z;
            R mzx = mz.re;
            R mzy = mz.im;
            R mz2 = mzx * mzx + mzy * mzy;
            R dzx = dz.re;
            R dzy = dz.im;
            R zx = z.re;
            R zy = z.im;
            R cx = c.re;
            R cy = c.im;
            R cmag = /*ref_c.re * ref_c.re + ref_c.im * ref_c.im + */cx * cx + cy ↙
                ↳ * cy;
            R rzx = rz.re;
            R rzy = rz.im;
            R zdx = z_d[0].re;
            R zdy = z_d[0].im;
            R four = R(4.0);
            R two = R(2.0);
            R one = R(1.0);

            bool active = true;
            for (int i = 1; i <= chunk; ++i) {

                // dz = R(2.0) * rz * dz + R(1.0);
                R dzxn = two * (rzx * dzx - rzy * dzy) + one;
                R dzyn = two * (rzx * dzy + rzy * dzx);
                R zdrx = zdx + rzx;
                R zdry = zdy + rzy;
```

```
                // z = (z_d[i-1] + rz) * z + c;
                R zxn = zdrx * zx - zdry * zy + cx;
                R zyn = zdrx * zy + zdry * zx + cy;
                R zdxn = z_d[i].re;
                R zdyn = z_d[i].im;
#if 0
                R condition_number_squared
                  = (four * (zx * zx + zy * zy + rzx * rzx + rzy * rzy /* + zdx * ↙
                      ↳ zdx + zdy * zdy*/) + one/*two*/)
                  / ( (zxn * zxn + zyn * zyn /* + zdxn * zdxn + zdyn * zdyn*/)
                    / (/*zdx * zdx + zdy * zdy + */ zx * zx + zy * zy + cmag)
                    );
                R condition_number_squared // (|2z_n(Z_n+z_n)|/|z_{n+1}|)^2
                  = four * (zx * zx + zy * zy) * (rzx * rzx + rzy * rzy) / (zxn * ↙
                      ↳ zxn + zyn * zyn);
#endif

                zdx = zdxn;
                zdy = zdyn;
                // rz = z_d[i] + z;
                rzx = zdx + zxn;
                rzy = zdy + zyn;
                // rz2 = norm(rz);
                R rz2 = rzx * rzx + rzy * rzy;

                dzx = dzxn;
                dzy = dzyn;
                zx = zxn;
                zy = zyn;

                if (rz2 <= mz2)
                {
                  complex<R> mz(mzx, mzy);
                  complex<R> rz(rzx, rzy);
                  complex<R> a(rz / mz);
                  img->output_domain_n[index] = iters + i;
                  img->output_domain_x[index] = to_ld(a.re);
                  img->output_domain_y[index] = to_ld(a.im);
                  mz2 = rz2;
                  mzx = rzx;
                  mzy = rzy;
                }

                bool is_glitched = rz2 < z_size[i]; // Pauldelbrot
                //bool is_glitched = condition_number_squared > (1 << 10);

                if (detect_glitches && is_glitched) {
                  // glitched
                  if (correct_glitches)
                  {
                    int my_glitch_count = ++glitch_count;
                    ref->index_data[dst][glitch_end - my_glitch_count] = index;
                    struct pixel<R> *out = in;
                    out->c = c;
                    out->z = complex<R>(rzx, rzy);
                    out->dz = complex<R>(dzx, dzy);
                    out->mz = complex<R>(mzx, mzy);
```

```
                out->iters = iters + i;
910             }
                else
                {
                  img->output_dwell_n [index] = iters + i;
                  img->output_dwell_f [index] = log2l(to_ld(rz2 / z_size[i])); // ↙
                      ↳ negative
915               img->output_dwell_a [index] = 0;
                  img->output_distance[index] = 0;
                  img->done_pixels++;
                }
                active = false;
920             break;
              } else if (rz2 > escape_radius_2) {
                // escaped
                img->output_dwell_n [index] = iters + i;
                img->output_dwell_f [index] = 1 - log2(log(to_ld(rz2)) / to_ld(↙
                    ↳ log_escape_radius_2)); // smooth iters
925             img->output_dwell_a [index] = arg(complex<long double>(to_ld(rzx),↙
                    ↳  to_ld(rzy))) / twopi;
                complex<R> d(dzx * pixel_spacing, dzy * pixel_spacing);
                img->output_distance[index] = sqrt(to_ld(rz2)) * log(to_ld(rz2)) /↙
                    ↳  to_ld(sqrt(norm(d))); // de
                img->done_pixels++;
                active = false;
930             break;
              }

            } // for i
            if (active) {
935           // still iterating
              int my_active_count = active_count++;
              ref->index_data[dst][active_start + my_active_count] = index;
              struct pixel<R> *out = in;
              out->c = c;
940           out->z = complex<R>(zx, zy);
              out->dz = complex<R>(dzx, dzy);
              out->mz = complex<R>(mzx, mzy);
              out->iters = iters + chunk;
            }
945
          } // if running
        } // for k

        if (! image_running(img)) { break; }
950
        if (glitch_count) {
          qsort_r(&ref->index_data[dst][glitch_end - glitch_count], glitch_count, ↙
              ↳ sizeof(uint32_t), cmp_pixel_by_iters_asc_r<R>, ref->pixel_data);
          int end = 0;
          for (int start = 0; start < glitch_count; start = end) {
955
            // compute minimum of span
            int kk = ref->index_data[dst][glitch_end - glitch_count + start];
            uint32_t start_iters = ref->pixel_data[kk].iters;
            R min_z2 = 1.0 / 0.0;
960         uint32_t min_ix = start;
```

```
            for (end = start; end < glitch_count && ref->pixel_data[ref->↙
                ↳ index_data[dst][glitch_end - glitch_count + end]].iters == ↙
                ↳ start_iters; ++end) {
              R z2 = norm(ref->pixel_data[ref->index_data[dst][glitch_end - ↙
                  ↳ glitch_count + end]].z);
              if (z2 < min_z2) {
                min_z2 = z2;
                min_ix = end;
              }
            }

            // enqueue a new reference for glitch
            int count = end - start;
            struct reference<R> *new_ref = (struct reference<R> *) calloc(1, ↙
                ↳ sizeof(*new_ref));
            // copy parent
            new_ref->has_parent = 1;
            mpc_init2(new_ref->parent_c, img->precision);
            mpc_set(new_ref->parent_c, ref->c, MPC_RNDNN);
            mpc_init2(new_ref->parent_z, img->precision);
            mpc_set(new_ref->parent_z, z_hi[start_iters - iters], MPC_RNDNN);
            new_ref->period = start_iters;
            new_ref->iters = start_iters;
            new_ref->pixel_data = ref->pixel_data;
            new_ref->index_data[0] = ref->index_data[0];
            new_ref->index_data[1] = ref->index_data[1];
            new_ref->index_src = dst;
            new_ref->index_src_offset = glitch_end - glitch_count + start;
            new_ref->index_src_count = count;
            new_ref->index_src_minix = glitch_end - glitch_count + min_ix;
            image_enqueue(img, new_ref);

          } // for start
        } // if glitch_count
        // swap active
        ref->index_src_count = active_count;
        ref->index_src = dst;
        if (! active_count) {
          break;
        }
      } // for iters

    } // while ref

    for (int k = 0; k <= maxchunk; ++k) {
      mpc_clear(z_hi[k]);
    }
    free(z_hi);
    free(z_d);
    free(z_size);

    image_log(img, LOG_QUEUE, "          END\n");
    return 0;
}


template <typename R>
```

```
static m_newton attractor_step(complex<R> &z, complex<R> z_guess, complex<R> c, ↙
    ↳ int period, R epsilon2) {
  complex<R> zz = z_guess;
  complex<R> dzz = 1;
  for (int i = 0; i < period; ++i) {
    dzz = 2 * zz * dzz;
    zz = zz * zz + c;
  }
  if (norm(zz - z_guess) <= epsilon2) {
    z = z_guess;
    return m_converged;
  }
  complex<R> z_new = z_guess - (zz - z_guess) / (dzz - 1);
  complex<R>  d = z_new - z_guess;
  if (norm(d) <= epsilon2) {
    z = z_new;
    return m_converged;
  }
  if (isfinite(d)) {
    z = z_new;
    return m_stepped;
  } else {
    z = z_guess;
    return m_failed;
  }
}

template <typename R>
static m_newton attractor(complex<R> &z_out, complex<R> z_guess, complex<R> c, ↙
    ↳ int period, int maxsteps, R epsilon2) {
  m_newton result = m_failed;
  complex<R> z = z_guess;
  for (int i = 0; i < maxsteps; ++i) {
    if (m_stepped != (result = attractor_step(z, z, c, period, epsilon2))) {
      break;
    }
  }
  z_out = z;
  return result;
}

template <typename R>
static bool interior_de(R &de_out, complex<R> &dz_out, complex<R> z, complex<R> ↙
    ↳ c, int p, int steps, R epsilon2) {
  complex<R> z00 = 0;
  if (m_failed != attractor(z00, z, c, p, steps, epsilon2)) {
    complex<R> z0 = z00;
    complex<R> dz0 = 1;
    for (int j = 0; j < p; ++j) {
      dz0 = 2 * z0 * dz0;
      z0 = z0 * z0 + c;
    }
    if (norm(dz0) <= 1) {
      complex<R> z1 = z00;
      complex<R> dz1 = 1;
      complex<R> dzdz1 = 0;
      complex<R> dc1 = 0;
```

```
        complex<R> dcdz1 = 0;
        for (int j = 0; j < p; ++j) {
          dcdz1 = 2 * (z1 * dcdz1 + dz1 * dc1);
          dc1 = 2 * z1 * dc1 + 1;
          dzdz1 = 2 * (dz1 * dz1 + z1 * dzdz1);
          dz1 = 2 * z1 * dz1;
          z1 = z1 * z1 + c;
        }
        de_out = (1 - norm(dz1)) / sqrt(norm(dcdz1 + dzdz1 * dc1 / (1 - dz1)));
        dz_out = dz1;
        return true;
      }
    }
    return false;
  }

  template <typename R>
  struct partial
  {
    complex<R> z;
    int p;
  };

  template <typename R>
  static void *image_worker_plain(void *threadarg) {
    struct perturbator *img = (struct perturbator *) threadarg;
    image_log(img, LOG_QUEUE, "          ENTER\n");

    pthread_mutex_lock(&img->mutex);

    int width = img->width;
    int height = img->height;
    int maxiters = img->maxiters;
    int maxpartials = maxiters;
    R escape_radius_2 = img->escape_radius_2;
    R log_escape_radius_2 = img->log_escape_radius_2;
    complex<R> center = mpc_get(img->center, MPC_RNDNN, R(0));
    R radius = mpfr_get(img->radius, MPFR_RNDN, R(0));
    R pixel_spacing = radius * R(2.0 / height);
    R epsilon2 = std::nextafter(R(256), R(1.0/0.0)) - R(256);
    epsilon2 *= epsilon2;

    pthread_mutex_unlock(&img->mutex);

    struct partial<R> *partials = (struct partial<R> *)malloc(sizeof(*partials) * ↙
        ↳ maxpartials);

    int j;
    while ( 0 <= (j = image_dequeue_plain<R>(img)) )
    {
      if (! image_running(img)) break;

      R y = ((j + R(0.5)) / height - R(0.5)) * R(2.0) * radius;

      int bias = 0;
      for (int i = 0; i < width; ++i)
      {
```

```
        if (! image_running(img)) break;
1125
        int index = j * width + i;

        R x = ((i + R(0.5)) / width - R(0.5)) * R(2.0) * radius * R(width) / R(↙
            ↳ height);

1130    int npartials = 0;
        complex<R> dc(x, -y);
        complex<R> c(center + dc);
        complex<R> z(0, 0);
        complex<R> dz(0, 0);
1135    complex<R> mz(1.0/0.0, 0);
        R mi(R(1.0)/R(0.0));
        bool done = false;
        for (int k = 1; k < maxiters; ++k)
        {
1140      dz = R(2.0) * z * dz + R(1.0);
          z = z * z + c;
          R z2 = norm(z);
          if (z2 <= mi)
          {
1145        complex<R> a = z / mz;
            img->output_domain_n[index] = k;
            img->output_domain_x[index] = a.re;
            img->output_domain_y[index] = a.im;
            mi = z2;
1150        mz = z;
            if (bias >= 0)
            {
              // store partial
              if (partials && npartials < maxpartials)
1155          {
                partials[npartials].z = z;
                partials[npartials].p = k;
                npartials++;
              }
1160        }
            else
            {
              // do interior check
              complex<R> dz = 0;
1165          R de = -1;
              if (interior_de(de, dz, z, c, k, 64, epsilon2))
              {
                // output interior
                img->output_dwell_n [index] = -k; // period
1170            img->output_dwell_f [index] = sqrt(norm(dz)); // radius
                img->output_dwell_a [index] = arg(dz) / twopi; // angle
                img->output_distance[index] = -de / pixel_spacing; // de
                img->done_pixels++;
                done = true;
1175            bias = -1;
                break;
              }
            }
          }
```

```
1180          if (z2 > escape_radius_2)
              {
                // output exterior
                img->output_dwell_n [index] = k;
                img->output_dwell_f [index] = 1 - log2(log(z2) / log_escape_radius_2);↙
                    ↳ // smooth iters
1185            img->output_dwell_a [index] = arg(z) / twopi;
                img->output_distance[index] = sqrt(z2) * log(z2) / sqrt(norm(dz * ↙
                    ↳ pixel_spacing)); // de
                img->done_pixels++;
                done = true;
                bias = 1;
1190            break;
              }
            }

            if (! image_running(img)) break;
1195
            if (bias >= 0 && ! done)
            {
              // check stored partials
              for (int p = 0; p < npartials; ++p)
1200          {
                // do interior check
                z = partials[p].z;
                int k = partials[p].p;
                complex<R> dz = 0;
1205            R de = -1;
                if (interior_de(de, dz, z, c, k, 64, epsilon2))
                {
                  // output interior
                  img->output_dwell_n [index] = -k; // period
1210              img->output_dwell_f [index] = sqrt(norm(dz)); // radius
                  img->output_dwell_a [index] = arg(dz) / twopi; // angle
                  img->output_distance[index] = -de / pixel_spacing; // de
                  img->done_pixels++;
                  bias = -1;
1215              break;
                }
              }
            }

1220      } // for i
        } // while ref

        free(partials);
        image_log(img, LOG_QUEUE, "          END\n");
1225    return 0;
      }


      extern struct perturbator *perturbator_new(int workers, int width, int height, ↙
          ↳ int maxiters, int maxchunk, double escape_radius, double glitch_threshold)↙
          ↳ {
1230    struct perturbator *img = (struct perturbator *) calloc(1, sizeof(*img));

        img->workers = workers;
```

```
    img->width = width;
    img->height = height;
    img->detect_glitches = 1;
    img->correct_glitches = 1;
    img->maxiters = maxiters;
    img->maxchunk = maxchunk;
    img->escape_radius = escape_radius;
    img->glitch_threshold = glitch_threshold;
    img->precision = 53;

    mpc_init2(img->center, 53);
    mpfr_init2(img->radius, 53);

    img->escape_radius_2 = escape_radius * escape_radius;
    img->log_escape_radius_2 = log(img->escape_radius_2);

    img->newton_steps_root = 64;
    img->newton_steps_child = 8;

    img->order = 64;
    img->threshold = 64;
    img->logging = LOG_VIEW | LOG_CACHE | LOG_QUEUE;

    mpc_init2(img->last_reference, 53);
    mpc_set_ui_ui(img->last_reference, 0, 0, MPC_RNDNN);
    img->last_period = 1;

    img->output_dwell_n  = (int32_t *) calloc(1, width * height * sizeof(*img->↙
        ↳ output_dwell_n));
    img->output_dwell_f  = (float   *) calloc(1, width * height * sizeof(*img->↙
        ↳ output_dwell_f));
    img->output_dwell_a  = (float   *) calloc(1, width * height * sizeof(*img->↙
        ↳ output_dwell_a));
    img->output_distance = (float   *) calloc(1, width * height * sizeof(*img->↙
        ↳ output_distance));
    img->output_domain_n = (int32_t *) calloc(1, width * height * sizeof(*img->↙
        ↳ output_domain_n));
    img->output_domain_x = (float   *) calloc(1, width * height * sizeof(*img->↙
        ↳ output_domain_x));
    img->output_domain_y = (float   *) calloc(1, width * height * sizeof(*img->↙
        ↳ output_domain_y));

    pthread_mutex_init(&img->mutex, 0);
    pthread_cond_init(&img->cond, 0);
    pthread_cond_init(&img->cond_done, 0);

    image_log(img, LOG_QUEUE | LOG_CACHE, "sequence action   queued   iters  ↙
        ↳ period    active count\n");
    image_log(img, LOG_QUEUE | LOG_CACHE, "-------- ------ -------- ------- ↙
        ↳ ------- ---------------\n");
    return img;
}


extern void perturbator_start(struct perturbator *img, const mpfr_t centerx, ↙
    ↳ const mpfr_t centery, const mpfr_t radius) {
```

```
1280    img->precision = std::max(53, int(53 - 2 * mpfr_get_exp(radius)));

        image_log(img, LOG_VIEW, "real=%Re\nimag=%Re\nradius=%Re\nprecision=%d\n", ↙
            ↳ centerx, centery, radius, img->precision);

        mpc_set_prec(img->center, img->precision);
1285    mpc_set_fr_fr(img->center, centerx, centery, MPC_RNDNN);
        mpfr_set(img->radius, radius, MPFR_RNDN);

        memset(img->output_dwell_n,  0, img->width * img->height * sizeof(*img->↙
            ↳ output_dwell_n));
        memset(img->output_dwell_f,  0, img->width * img->height * sizeof(*img->↙
            ↳ output_dwell_f));
1290    memset(img->output_dwell_a,  0, img->width * img->height * sizeof(*img->↙
            ↳ output_dwell_a));
        memset(img->output_distance, 0, img->width * img->height * sizeof(*img->↙
            ↳ output_distance));
        memset(img->output_domain_n, 0, img->width * img->height * sizeof(*img->↙
            ↳ output_domain_n));
        memset(img->output_domain_x, 0, img->width * img->height * sizeof(*img->↙
            ↳ output_domain_x));
        memset(img->output_domain_y, 0, img->width * img->height * sizeof(*img->↙
            ↳ output_domain_y));
1295    img->done_pixels = 0;

        mpfr_t pixel_spacing;
        mpfr_init2(pixel_spacing, 53);
        mpfr_set(pixel_spacing, img->radius, MPFR_RNDN);
1300    mpfr_mul_d(pixel_spacing, pixel_spacing, 2.0 / img->height, MPFR_RNDN);
        int64_t e = mpfr_get_exp(pixel_spacing);
        mpfr_clear(pixel_spacing);

        img->rm =
1305      e >= EXP_THRESHOLD_PLAIN_LONG_DOUBLE ? rm_plain : rm_perturb;
        switch (img->rm)
        {
          case rm_plain:
            img->ft =
1310          e >= EXP_THRESHOLD_PLAIN_DOUBLE ? ft_double :
              ft_long_double;
            switch (img->ft)
            {
              case ft_double:
1315            image_log(img, LOG_VIEW, "          TYPE     f53e11          double ↙
                    ↳ plain\n");
                perturbator_start_internal_plain<double>(img);
                return;
              case ft_long_double:
                image_log(img, LOG_VIEW, "          TYPE     f64e15     long double ↙
                    ↳ plain\n");
1320            perturbator_start_internal_plain<long double>(img);
                return;
            }
            break;
          case rm_perturb:
1325        img->ft =
              e >= EXP_THRESHOLD_PERTURB_DOUBLE ? ft_double :
```

```
        e >= EXP_THRESHOLD_PERTURB_LONG_DOUBLE ? ft_long_double :
        ft_edouble;
      switch (img->ft)
      {
        case ft_double:
          image_log(img, LOG_VIEW, "          TYPE     f53e11          double ↙
              ↳ perturb\n");
          perturbator_start_internal<double>(img);
          return;
        case ft_long_double:
          image_log(img, LOG_VIEW, "          TYPE     f64e15     long double ↙
              ↳ perturb\n");
          perturbator_start_internal<long double>(img);
          return;
        case ft_edouble:
          image_log(img, LOG_VIEW, "          TYPE     f53e64         edouble ↙
              ↳ perturb\n");
          perturbator_start_internal<edouble>(img);
          return;
      }
      break;
  }
  assert(! "valid float type");
}


extern void perturbator_stop(struct perturbator *img, int force) {
  if (force) {
    img->running = false;
  }
  pthread_mutex_lock(&img->mutex);
  pthread_cond_broadcast(&img->cond);
  pthread_mutex_unlock(&img->mutex);
  for (int i = 0; i < img->workers; ++i) {
    pthread_join(img->threads[i], 0);
  }
  switch (img->rm)
  {
    case rm_plain:
      break;
    case rm_perturb:
      switch (img->ft) {
        case ft_float:       release_refs<float>(img); return;
        case ft_double:      release_refs<double>(img); return;
        case ft_long_double: release_refs<long double>(img); return;
        case ft_edouble:     release_refs<edouble>(img); return;
      }
      assert(! "valid float type");
      break;
  }
}

extern bool perturbator_wait(struct perturbator *img, const struct timespec *ts)
{
  int timed_out = 0;
  pthread_mutex_lock(&img->mutex);
```

```
  while (timed_out == 0 && img->running)
  {
    timed_out = pthread_cond_timedwait(&img->cond_done, &img->mutex, ts);
  }
  bool active = img->running;
  pthread_mutex_unlock(&img->mutex);
  return active;
}

void image_delete_cache(struct perturbator *img) {
  if (img->series) {
    z2c_series_delete(img->series);
    img->series = 0;
  }
  while (img->nodes) {
    struct series_node *next = img->nodes->next;
    mpc_clear(img->nodes->z);
    z2c_approx_delete(img->nodes->approx);
    img->nodes = next;
  }
}

struct series_node *image_cached_approx(struct perturbator *img, bool reused, const mpc_t c, int64_t exponent, mpc_t z, int *iter) {
  pthread_mutex_lock(&img->mutex);
  if (! reused) {
    image_delete_cache(img);
  }
  if (! img->series) {
    img->series = z2c_series_new(img->order, mpc_realref(c), mpc_imagref(c), ft_edouble);
  }
  int64_t exponent0 = 16;
  if (img->nodes)  {
    exponent0 = img->nodes->exponent;
  }
  // cache all the things
  for (int64_t e = exponent0; e >= exponent; --e) {
    while (z2c_series_step(img->series, e + 2, img->threshold, img->approx_skip)) {
//      if (! image_running(img)) {
//        return 0;
//      }
      // nop
    }
    int iters = z2c_series_get_n(img->series);
    if ((! img->nodes) || (img->nodes && iters >= img->nodes->iters)) {
      if (img->nodes && iters == img->nodes->iters) {
        img->nodes->exponent = e;
      } else {
        struct series_node *node = (struct series_node *) calloc(1, sizeof(*node));
        node->next = img->nodes;
        node->exponent = e;
        node->iters = iters;
        mpc_init2(node->z, 53); // updated by get_z
```

```
        struct z2c_reference *reference = z2c_series_reference_new(img->series);
1435    z2c_reference_get_zr(reference, mpc_realref(node->z), mpc_imagref(node->z));
        z2c_reference_delete(reference);
        node->approx = z2c_approx_new(img->series);
        img->nodes = node;
      }
1440  }
  }
  // lookup in the cache
  struct series_node *node = img->nodes, *prev = img->nodes;
  while (node && node->exponent < exponent) {
1445  prev = node;
  node = node->next;
  }
  if (node) {
  mpc_set(z, node->z, MPC_RNDNN);
1450  *iter = node->iters;
  pthread_mutex_unlock(&img->mutex);
  return node;
  } else {
  pthread_mutex_unlock(&img->mutex);
1455  return prev;
  }
}

int perturbator_get_primary_reference(struct perturbator *img, mpfr_t x, mpfr_t y) {
1460  pthread_mutex_lock(&img->mutex);
  int period = img->last_period;
  int prec = mpc_get_prec(img->last_reference);
  mpfr_set_prec(x, prec);
  mpfr_set_prec(y, prec);
1465  mpfr_set(x, mpc_realref(img->last_reference), MPFR_RNDN);
  mpfr_set(y, mpc_imagref(img->last_reference), MPFR_RNDN);
  pthread_mutex_unlock(&img->mutex);
  return period;
}
1470

void perturbator_set_detect_glitches(struct perturbator *img, int detect_glitches) {
  img->detect_glitches = detect_glitches;
}

1475 void perturbator_set_approx_skip(struct perturbator *img, int approx_skip) {
  img->approx_skip = approx_skip;
}

void perturbator_set_maximum_iterations(struct perturbator *img, int maxiters) {
1480  img->maxiters = maxiters;
}

double perturbator_get_progress(struct perturbator *img)
{
1485  return img->done_pixels / (double) (img->width * img->height);
}
```

```
int perturbator_get_width(struct perturbator *img)
{
  return img->width;
}

int perturbator_get_height(struct perturbator *img)
{
  return img->height;
}
```

16 c/lib/pkgconfig/mandelbrot-perturbator.pc.in

```
exec_prefix=${prefix}
libdir=${exec_prefix}/lib
includedir=${prefix}/include

Name: mandelbrot-perturbator
Description: Efficient deep zooming for the Mandelbrot set
Version: 0.1.0.0
URL: https://code.mathr.co.uk/mandelbrot-perturbator
Requires:
Libs: -fopenmp -L${libdir} -lmandelbrot-perturbator
Libs.private: -lmpc -lmpfr -lgmp -lm
Cflags: -fopenmp -I${includedir}
```

17 c/lib/z2c.c

```
// mandelbrot-perturbator -- efficient deep zooming for Mandelbrot sets
// Copyright (C) 2015,2016,2017 Claude Heiland-Allen
// License GPL3+ http://www.gnu.org/licenses/gpl.html

#include <algorithm>
#include <climits>
#include <cmath>
#include <cstdbool>
#include <cstdint>
#include <cstdio>
#include <cstdlib>
#include <mpfr.h>

#include "edouble.cc"
#include "complex.hpp"

template< typename R >
struct z2c_series_t {
  int order;
  mpfr_t v[11];
  complex<R> *a[2];
  int n;
};

struct z2c_series {
  float_type tag;
  union {
    z2c_series_t<double> *d;
    z2c_series_t<long double> *l;
```

```
    z2c_series_t<edouble> *e;
  } u;
};

template< typename R >
struct z2c_approx_t {
  int order;
  complex<R> *a;
};

struct z2c_approx {
  float_type tag;
  union {
    z2c_approx_t<double> *d;
    z2c_approx_t<long double> *l;
    z2c_approx_t<edouble> *e;
  } u;
};

struct z2c_reference {
  mpfr_t v[6];
  int n;
};

struct z2c_reference *z2c_reference_new(const mpfr_t cx, const mpfr_t cy, const mpfr_t zx, const mpfr_t zy, int n) {
  struct z2c_reference *r = (struct z2c_reference *) malloc(sizeof(*r));
  if (! r) { return 0; }
  mpfr_prec_t p = std::max(std::max(mpfr_get_prec(cx), mpfr_get_prec(cy)), std::max(mpfr_get_prec(zx), mpfr_get_prec(zy)));
  for (int i = 0; i < 6; ++i) {
    mpfr_init2(r->v[i], p);
    mpfr_set_si(r->v[i], 0, MPFR_RNDN);
  };
  mpfr_set(r->v[0], cx, MPFR_RNDN);
  mpfr_set(r->v[1], cy, MPFR_RNDN);
  mpfr_set(r->v[2], zx, MPFR_RNDN);
  mpfr_set(r->v[3], zy, MPFR_RNDN);
  r->n = n;
  return r;
}

void z2c_reference_delete(struct z2c_reference *r) {
  for (int i = 0; i < 6; ++i) {
    mpfr_clear(r->v[i]);
  }
  free(r);
}

void z2c_reference_step(struct z2c_reference *r) {
  mpfr_sqr(r->v[4], r->v[2], MPFR_RNDN);
  mpfr_sqr(r->v[5], r->v[3], MPFR_RNDN);
  mpfr_sub(r->v[4], r->v[4], r->v[5], MPFR_RNDN);
  mpfr_mul(r->v[5], r->v[2], r->v[3], MPFR_RNDN);
  mpfr_mul_2si(r->v[5], r->v[5], 1, MPFR_RNDN);
  mpfr_add(r->v[2], r->v[4], r->v[0], MPFR_RNDN);
  mpfr_add(r->v[3], r->v[5], r->v[1], MPFR_RNDN);
```

```
  r->n += 1;
}

int z2c_reference_get_n(const struct z2c_reference *r) {
  return r->n;
}

void z2c_reference_get_zr(const struct z2c_reference *ref, mpfr_t zx, mpfr_t zy)
    ↪ {
  mpfr_set_prec(zx, mpfr_get_prec(ref->v[2]));
  mpfr_set_prec(zy, mpfr_get_prec(ref->v[3]));
  mpfr_set(zx, ref->v[2], MPFR_RNDN);
  mpfr_set(zy, ref->v[3], MPFR_RNDN);
}

template< typename R >
struct z2c_approx_t<R> *z2c_approx_t_new(const struct z2c_series_t<R> *s) {
  struct z2c_approx_t<R> *a = (struct z2c_approx_t<R> *) malloc(sizeof(*a));
  a->order = s->order;
  a->a = (complex<R> *) calloc(1, s->order * sizeof(complex<R>));
  for (int i = 0; i < s->order; ++i) {
    a->a[i] = s->a[0][i];
  }
  return a;
}

struct z2c_approx *z2c_approx_new(const struct z2c_series *s) {
  struct z2c_approx *a = (struct z2c_approx *) malloc(sizeof(*a));
  switch (s->tag) {
    case ft_double:      a->u.d = z2c_approx_t_new(s->u.d); break;
    case ft_long_double: a->u.l = z2c_approx_t_new(s->u.l); break;
    case ft_edouble:     a->u.e = z2c_approx_t_new(s->u.e); break;
    default: assert(! "float type valid");
  }
  a->tag = s->tag;
  return a;
}

template< typename R >
void z2c_approx_t_delete(struct z2c_approx_t<R> *a) {
  free(a->a);
  free(a);
}

void z2c_approx_delete(struct z2c_approx *a) {
  switch (a->tag) {
    case ft_double:      z2c_approx_t_delete(a->u.d); break;
    case ft_long_double: z2c_approx_t_delete(a->u.l); break;
    case ft_edouble:     z2c_approx_t_delete(a->u.e); break;
    default: assert(! "float type valid");
  }
  free(a);
}

template< typename R >
void z2c_approx_t_do(const struct z2c_approx_t<R> *a, complex<R> dc, complex<R> ↙
    ↪ *dz_out, complex<R> *ddz_out) {
```

```
140   complex<R> z(dc);
      complex<R> zi(1);
      complex<R> s(0);
      complex<R> ds(0);
      for (int i = 0; i < a->order; ++i) {
145     ds += (i + 1) * a->a[i] * zi;
        zi *= z;
        s += a->a[i] * zi;
      }
      *dz_out = s;
150   *ddz_out = ds;
    }

    void z2c_approx_do(const struct z2c_approx *a, complex<edouble> dc, complex<↙
        ↳ edouble> *dz_out, complex<edouble> *ddz_out) {
      switch (a->tag) {
155     case ft_double:
        {
          complex<double> dc_ = to_C(dc, double(0));
          complex<double> dz_out_, ddz_out_;
          z2c_approx_t_do(a->u.d, dc_, &dz_out_, &ddz_out_);
160       *dz_out = to_C(dz_out_, edouble(0));
          *ddz_out = to_C(ddz_out_, edouble(0));
          break;
        }
        case ft_long_double:
165     {
          complex<long double> dc_ = to_C(dc, (long double)(0));
          complex<long double> dz_out_, ddz_out_;
          z2c_approx_t_do(a->u.l, dc_, &dz_out_, &ddz_out_);
          *dz_out = to_C(dz_out_, edouble(0));
170       *ddz_out = to_C(ddz_out_, edouble(0));
          break;
        }
        case ft_edouble:
        {
175       z2c_approx_t_do(a->u.e, dc, dz_out, ddz_out);
          break;
        }
        default:
        {
180       assert(! "float type valid");
        }
      }
    }

185 template< typename R >
    struct z2c_series_t<R> *z2c_series_t_new(int order, const mpfr_t cx, const ↙
        ↳ mpfr_t cy, const R dummy) {
      (void) dummy;
      struct z2c_series_t<R> *s = (struct z2c_series_t<R> *) malloc(sizeof(*s));
      if (! s) { return 0; }
190   s->order = order;
      s->a[0] = (complex<R> *) calloc(1, order * sizeof(complex<R>));
      s->a[1] = (complex<R> *) calloc(1, order * sizeof(complex<R>));
      mpfr_prec_t p = std::max(mpfr_get_prec(cx), mpfr_get_prec(cy));
      for (int i = 0; i < 11; ++i) {
```

```
195       mpfr_init2(s->v[i], p);
          mpfr_set_si(s->v[i], 0, MPFR_RNDN);
        };
        mpfr_set(s->v[0], cx, MPFR_RNDN);
        mpfr_set(s->v[1], cy, MPFR_RNDN);
200     mpfr_set(s->v[2], cx, MPFR_RNDN);
        mpfr_set(s->v[3], cy, MPFR_RNDN);
        mpfr_set_si(s->v[4], 1, MPFR_RNDN);
        s->a[0][0] = complex<R>(1);
        s->n = 1;
205     return s;
      }

      struct z2c_series *z2c_series_new(int order, const mpfr_t cx, const mpfr_t cy, ↙
          ↳ float_type type) {
        struct z2c_series *s = (struct z2c_series *) malloc(sizeof(*s));
210     switch (type) {
          case ft_double:      s->u.d = z2c_series_t_new(order, cx, cy, double(0)); ↙
              ↳ break;
          case ft_long_double: s->u.l = z2c_series_t_new(order, cx, cy, (long double)↙
              ↳ (0)); break;
          case ft_edouble:     s->u.e = z2c_series_t_new(order, cx, cy, edouble(0)); ↙
              ↳ break;
          default: assert(! "float type valid");
215     }
        s->tag = type;
        return s;
      }

220   template< typename R >
      void z2c_series_t_delete(struct z2c_series_t<R> *s) {
        for (int i = 0; i < 11; ++i) {
          mpfr_clear(s->v[i]);
        }
225     free(s->a[0]);
        free(s->a[1]);
        free(s);
      }

230   void z2c_series_delete(struct z2c_series *s) {
        switch (s->tag) {
          case ft_double:      z2c_series_t_delete(s->u.d); break;
          case ft_long_double: z2c_series_t_delete(s->u.l); break;
          case ft_edouble:     z2c_series_t_delete(s->u.e); break;
235       default: assert(! "float type valid");
        }
        free(s);
      }

240   template< typename R >
      int z2c_series_t_get_n(const struct z2c_series_t<R> *s) {
        return s->n;
      }

245   int z2c_series_get_n(struct z2c_series *s) {
        switch (s->tag) {
          case ft_double:      return z2c_series_t_get_n(s->u.d);
```

```
    case ft_long_double: return z2c_series_t_get_n(s->u.l);
    case ft_edouble:     return z2c_series_t_get_n(s->u.e);
    default: assert(! "float type valid");
  }
  return 0;
}

template< typename R >
bool z2c_series_t_step(struct z2c_series_t<R> *s, mpfr_exp_t exponent1, mpfr_exp_t threshold, int approx_skip) {
  // 6,7 <- z^2 + c
  mpfr_sqr(s->v[6], s->v[2], MPFR_RNDN);
  mpfr_sqr(s->v[7], s->v[3], MPFR_RNDN);
  mpfr_sub(s->v[6], s->v[6], s->v[7], MPFR_RNDN);
  mpfr_mul(s->v[7], s->v[2], s->v[3], MPFR_RNDN);
  mpfr_mul_2si(s->v[7], s->v[7], 1, MPFR_RNDN);
  mpfr_add(s->v[6], s->v[6], s->v[0], MPFR_RNDN);
  mpfr_add(s->v[7], s->v[7], s->v[1], MPFR_RNDN);
  // step coefficients
  complex<R> z(mpfr_get(s->v[2], MPFR_RNDN, R(0)), mpfr_get(s->v[3], MPFR_RNDN, R(0)));
  R two(2);
  #pragma omp parallel for
  for (int i = 0; i < s->order/2; ++i) {
    // load balancing
    int ns[2] = { i + 1, s->order - i };
    for (int j = 0; j < 2; ++j) {
      int n = ns[j];
      // special case
      if (n == 1) {
        s->a[1][n-1] = two * z * s->a[0][n-1] + R(1);
      } else {
        // a(n) <- 2 z a(n) + sum a(m) a(n-m)
        complex<R> sum(0);
        for (int m = 1; m < (n+1)/2; ++m) {
          sum += s->a[0][m - 1] * s->a[0][n-m - 1];
        }
        sum *= two;
        if (0 == (n & 1)) {
          sum += s->a[0][n/2 - 1] * s->a[0][n/2 - 1];
        }
        s->a[1][n-1] = two * z * s->a[0][n-1] + sum;
      }
    }
  }
  // check valid
  bool valid = true;
  int64_t e0 = LONG_MIN;
  int64_t e1 = LONG_MIN;
  for (int i = 0; i < s->order; ++i) {
    e1 = LONG_MIN;
    if (real(s->a[1][i]) != 0) { e1 = std::max(e1, exponent(real(s->a[1][i]))); }
    if (imag(s->a[1][i]) != 0) { e1 = std::max(e1, exponent(imag(s->a[1][i]))); }
    if (i > 0) {
      valid = e0 - exponent1 >= e1 + threshold;
```

```
      e0 = std::max(e0 - exponent1, e1);
    } else {
      e0 = e1;
    }
  }
  if (approx_skip) {
    valid = s->n < approx_skip;
  }
  if (! valid) { return false; }
  // step
  s->n += 1;
  for (int i = 0; i < 2; ++i) {
    mpfr_set(s->v[i+2], s->v[i+6], MPFR_RNDN);
  }
  for (int i = 0; i < s->order; ++i) {
    s->a[0][i] = s->a[1][i];
  }
  // ok
  return true;
}

bool z2c_series_step(struct z2c_series *s, mpfr_exp_t exponent1, mpfr_exp_t ↙
    ↳ threshold, int approx_skip) {
  switch (s->tag) {
    case ft_double:      return z2c_series_t_step(s->u.d, exponent1, threshold, ↙
        ↳ approx_skip);
    case ft_long_double: return z2c_series_t_step(s->u.l, exponent1, threshold, ↙
        ↳ approx_skip);
    case ft_edouble:     return z2c_series_t_step(s->u.e, exponent1, threshold, ↙
        ↳ approx_skip);
    default: assert(! "float type valid");
  }
  return 0;
}

template< typename R >
struct z2c_reference *z2c_series_t_reference_new(const struct z2c_series_t<R> *s↙
    ↳ ) {
  return z2c_reference_new(s->v[0], s->v[1], s->v[2], s->v[3], s->n);
}

struct z2c_reference *z2c_series_reference_new(const struct z2c_series *s) {
  switch (s->tag) {
    case ft_double:      return z2c_series_t_reference_new(s->u.d);
    case ft_long_double: return z2c_series_t_reference_new(s->u.l);
    case ft_edouble:     return z2c_series_t_reference_new(s->u.e);
    default: assert(! "float type valid");
  }
  return 0;
}
```

18 .gitignore

```
c/bin/m-perturbator-automorph
c/bin/m-perturbator-colourize
c/bin/m-perturbator-glfw3
c/bin/m-perturbator-gtk
```

```
c/bin/m-perturbator-offline
c/lib/edouble
*.a
*.d
*.f32
*.js
*.js.mem
*.o
*.p
*.pc
*.pdf
*.png
*.ppm
*.so
*.xcf
-
```

19 README-gtk.md

```
# m-perturbator-gtk

Graphical user interface for Mandelbrot set exploration and annotation.

<https://mathr.co.uk/mandelbrot/perturbator/gtk/>

## Overview

There is a menu bar at the top, with File, Explore, Feature menus, followed
by Period, n-Fold, Depth entry boxes.  Then there are widgets that control the
appearance of annotations: line dash combo box, fill pattern combo box and
colour selection button.

Below the menu bar on the top left is the fractal display.

Below the fractal display is the log panel, which displays information.  The
default size of the fractal display is optimized for 16:9 display aspect.
The aspect ratio is designed for printing on landscape A4 paper with a 2cm
border on all sides.

The top right has the annotation list, which can be sorted by clicking on its
header.  The appearance widgets control the current annotation and future
annotations, the widgets are updated on selecting an annotation.  Selecting
an annotation highlights it in the fractal display.

Bottom right is the task queue, which displays progress of computations and
allows them to be cancelled.

The program version is displayed in the title bar and is also saved in
parameter files (if you need to copy/paste for bug reports).

To quit the program use the window close button.

## File menu

### Open

Load parameters from a file.  Existing annotations are deleted first.
```

```
### Save

Save parameters to a file. The format is a line-based text file. Annotations
are saved as specifications so (for example) rays will be recalculated on load.

### Save Image

Save image as displayed to a PNG file. Parameters are not yet saved to image
metadata, so save the parameters too if you want to be able to reload.

### Dark / Light

Switch overall theme between dark on light and light on dark. Colours should
be value-inverted when changing theme (so light and dark are swapped fully) but
this is not yet implemented. Recommend using greys instead of black or white
until this is finished.

### Monochrome / Low Colour / Full Colour

Switch colour theme between three levels. The lower colour levels may be
useful for economical low-ink printing.

## Explore menu

### Home

Reset view to the initial top level view.

### Zoom Out

Zoom out by a factor of 10.

### Zoom

Activate zoom tool. When zoom tool is active, dragging with the left mouse
button on the fractal marks a rectangle which will be zoomed to when the left
button is released. Press the right mouse button before releasing the left
mouse button to cancel the action.

### Zoom To

When a nucleus annotation is selected, activating this menu item opens a
dialog to enter a zoom fraction, where the default 0.75 is near an embedded
Julia set and 1.0 is at the mu-atom.

### Info

Activate info tool. When info tool is active, clicking with the left mouse
button on the fractal prints information about the point in the log pane at
the bottom of the window.

#### Pixel

Pixel coordinates, origin top left.

#### Real / Imag
```

```
Fractal coordinates.

#### Dwell N

Integer part of the iteration count, or the period of the containing component
when negative. Note: deep zooms don't yet have interior checking, so Dwell N
and the other values will be 0 instead of a finite negative number.

#### Dwell F

Fractional part of the iteration count, or the radius within the containing
component when Dwell N is negative.

#### Angle F

Argument of the final iterate, or the internal angle within the containing
component when Dwell N is negative.

#### Distance

Distance estimate in pixels. Positive for exterior distance estimate, negative
for interior distance estimate. The boundary of the Mandelbrot set is within
a factor of two of this distance, approximately.

### Select

Activate select tool. When select tool is active, clicking on an annotation
in the fractal display will select it in the annotation list.

### Increase / Decrease Iterations

Adjust the maximum iteration count used to calculate the fractal. Deeper zooms
typically need more iterations.

## Feature menu

### Nucleus

Activate nucleus tool. When nucleus tool is active, dragging with the left
mouse button in the fractal display marks a circle. On releasing the button,
the lowest period atom in the circle is found and annotated. If no atom
could be found then nothing happens. Press the right button before releasing
the left button to cancel the action.

### Bond

Activate bond tool. When bond tool is active, clicking on a child bulb (a
circle-like component) will annotate its root with the internal angle from its
parent.

### Misiurewicz

Activate Misiurewicz point tool. When Misiurewicz point tool is active,
dragging with the left button in the fractal display marks a square. On
releasing the button, the lowest (pre)period Misiurewicz point in the square
is found and annotated. The Period entry box in the menu bar can be used to
```

limit the search. If no Misiurewicz point is found nothing happens. Press the right button before releasing the left button to cancel the action.

Ray Out

Activate ray out tool. When ray out tool is active, clicking in the fractal display traces an external ray outwards towards infinity. The ray is listed in the annotation list as a string of binary digits.

Ray In

Opens a dialog where an angle can be entered in binary form, for example `.(01)` for a periodic ray or `.001100(001)` for a pre-periodic ray. Alternatively a string of binary digits can be entered with the pre-period and period entered in decimal in the boxes below. This latter form is useful when combined with the ray out tool, as any annotation selected when the ray in menu is activated will be copied to the angle entry box, so only the pre(periods) need to be entered. The depth of the ray can be adjusted if desired.

Extend Ray

When a ray in annotation is selected, open a dialog to give a new depth. If the new depth is higher than the current depth, the ray will be extended.

Wake < / >

When a ray in annotation is selected, and another ray in annotation with the same pre(period) lands at the same point, annotate the wake between them. The second ray is the next ray in the anti-clockwise direction from the selected ray for Wake <, clockwise for Wake >.

Atom

When a nucleus annotation is selected, annotate the corresponding atom by tracing its boundary and filling the resulting polygon.

Domain

When a nucleus annotation is selected, annotate the corresponding atom domain by tracing its boundary and stroking the resulting polygon. Atom domains are typically much bigger than atoms for cardioid-like atoms, around four times the size for circle-like atoms.

Mu-Unit

When a nucleus annotation is selected, annotate the nuclei in its tuned mu-unit up to a certain period multiplier. It works by tracing rays and finding nuclei relative to the top level cardioid, then using atom coordinates (based on the derivative of the multiplier) to translate to the desired nucleus. The process is not perfect, so some nuclei may be marked incorrectly.

Filaments

When a nucleus annotation is selected, annotate its filaments (when the Period entry box in the menu bar has its period) or the filaments of its embedded Julia set (when the Period entry box in the menu bar has the period of its

```
210 influencing island). The n-Fold entry box in the menu bar selects how deep
    towards the central island the rays should go (it increases by 1 for each
    factor of 2 of rotational symmetry). The Depth entry box in the menu bar sets
    how many ray pairs to draw (it increases by 1 for each doubling of ray count).

215 ### Delete

    Delete the currently selected annotation. There is no undo.

    ## Legal
220
    Copyright (c) 2015-2019 Claude Heiland-Allen <mailto:claude@mathr.co.uk>

    Free Software under GNU AGPL3+. When (re)distributing binaries or providing
    access to them as a service, you must also provide the corresponding source
225 code (including any modifications).


    --
    <https://mathr.co.uk>
```

20 README.md

```
    mandelbrot-perturbator
    ======================

    efficient deep zooming for Mandelbrot sets
 5  Copyright (C) 2015,2016,2017 Claude Heiland-Allen
    License GPL3+ http://www.gnu.org/licenses/gpl.html


    quickstart
10  ----------

    This will install to ~/opt/ by default:

        git clone https://code.mathr.co.uk/mandelbrot-numerics.git
15      git clone https://code.mathr.co.uk/mandelbrot-symbolics.git
        git clone https://code.mathr.co.uk/mandelbrot-perturbator.git
        make -C mandelbrot-numerics/c/lib install
        make -C mandelbrot-numerics/c/bin install     # optional
        make -C mandelbrot-symbolics/c/lib install
20      make -C mandelbrot-symbolics/c/bin install    # optional
        make -C mandelbrot-perturbator/c/lib install
        make -C mandelbrot-perturbator/c/bin install
        m-perturbator-glfw3

25
    rendering methods
    -----------------

    different rendering methods used for different depths:
30
        kind    number maxdepth
        plain   f32    1e-7
        plain   f64    1e-15
        plain   f80    1e-??
```

```
    perturb f32     1e-38
    perturb f64     1e-308
    perturb f80     1e-4932
    perturb f??i16  1e-9864
    perturb f??i32  1e-646456993
    perturb f??i64  1e-2776511644261678080

implemented so far:

    perturb f64/f80/f64i64 "z^2 + c" with order 24 series approximation


future api
----------

    struct buffer
    buffer      *create(int width, int height)
    void         destroy(buffer *)
    const float *get_data(const buffer *)
    int          get_width(const buffer *)
    int          get_height(const buffer *)

    struct view
    view        *create()
    void         destroy(view *)
    void         set_x(view *, const mpfr_t)
    void         set_y(view *, const mpfr_t)
    void         set_r(view *, const mpfr_t)
    void         get_x(const view *, mpfr_t)
    void         get_y(const view *, mpfr_t)
    void         get_r(const view *, mpfr_t)

    struct options
    options     *create()
    void         destroy(options *)
    void         set_threads(options *, int)
    void         set_maxiters(options *, int)
    void         set_maxperiod(options *, int)
    void         set_escape_radius(options *, double)
    void         set_glitch_radius(options *, double)

    struct context
    context     *create(enum formula, int order)
    void         destroy(context *)
    void         start(context *, buffer *, const view *, const options *)
    bool         done(context *)
    void         wait(context *)
    void         stop(context *)
```

21 TODO.md

```
todo
====

phase 1
* switch to new reference if all pixels are glitched
```

```
phase 2
* don't use shape test
* generate parallel and interruptible nucleus code
* generate parallel and interruptible box period code
* generate parallel and interruptible atom domain size estimate code
* remove dependency on mandelbrot-numerics

phase 3
* generate per-pixel perturbation code
* generate more formulas

phase 4
* implement plain, benchmark vs perturbed
* gtk3 example program

phase 5
* GPU acceleration
```