# gpujulia

Claude Heiland-Allen

2010–2013

# Contents



## 1 animate.sh

```
#!/bin/bash
SEGMENT=$(ls -1 *.ppm | sort -R | head -n 1)
while [ "z${SEGMENT}" != "z" ]
do
  cat ${SEGMENT}
  SOURCE=$(echo ${SEGMENT} | sed 's|^.*_-_||' | sed 's|\.ppm$||')
  SEGMENT=$(ls -1 ${SOURCE}_-_*.ppm | sort -R | head -n 1)
done |
ppmtoy4m -S 420mpeg2 -F 25:1 |
mplayer -demuxer y4m -
```

## 2 encode_video.sh

```
#!/bin/bash
for PPM in $@
do
  M2V="${PPM%ppm}m2v"
  ppmtoy4m -S 444 -F 25:1 <"${PPM}" |
  y4mscaler -I sar=1/1 -O preset=dvd -O yscale=1/1 |
  mpeg2enc -f 8 -q 3 -b 8000 -B 768 -D 10 -g 15 -G 15 -P -R 2 -o "${M2V}"
done
```

## 3 julia.c

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/stat.h>
```

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

struct shader {
  GLint linkStatus;
  GLhandleARB program;
  GLhandleARB fragment;
  GLhandleARB vertex;
  const GLcharARB *fragmentSource;
  const GLcharARB *vertexSource;
};

#define shader_uniform(self,name) \
  (self)->uniform.name = \
     glGetUniformLocationARB((self)->shader.program, #name)
#define shader_updatei(self,name) \
  glUniform1iARB((self)->uniform.name, (self)->value.name)
#define shader_updatef(self,name) \
  glUniform1fARB((self)->uniform.name, (self)->value.name)
#define shader_updatef2(self,name) \
  glUniform2fARB((self)->uniform.name, (self)->value.name[0], (self)->value.name↙
     ↳ [1])
#define shader_updatef4(self,name) \
  glUniform4fARB((self)->uniform.name, (self)->value.name[0], (self)->value.name↙
     ↳ [1], (self)->value.name[2], (self)->value.name[3])
#define shader_updatem4(self,name) \
  glUniformMatrix4fvARB((self)->uniform.name, 1, 0, &((self)->value.name[0]))

void shader_debug(GLhandleARB obj) {
  int infologLength = 0;
  int maxLength;
  if (glIsShader(obj)) {
    glGetShaderiv(obj, GL_INFO_LOG_LENGTH, &maxLength);
  } else {
    glGetProgramiv(obj, GL_INFO_LOG_LENGTH, &maxLength);
  }
  char *infoLog = malloc(maxLength);
  if (!infoLog) {
    return;
  }
  if (glIsShader(obj)) {
    glGetShaderInfoLog(obj, maxLength, &infologLength, infoLog);
  } else {
    glGetProgramInfoLog(obj, maxLength, &infologLength, infoLog);
  }
  if (infologLength > 0) {
    fprintf(stderr, "%s\n", infoLog);
  }
  free(infoLog);
}

struct shader *shader_init(
  struct shader *shader, const char *vert, const char *frag
) {
  if (! shader) { return 0; }
  shader->linkStatus     = 0;
  shader->vertexSource   = vert;
```

```
  shader->fragmentSource = frag;
  if (shader->vertexSource || shader->fragmentSource) {
    shader->program = glCreateProgramObjectARB();
    if (shader->vertexSource) {
      shader->vertex =
        glCreateShaderObjectARB(GL_VERTEX_SHADER_ARB);
      glShaderSourceARB(shader->vertex,
        1, (const GLcharARB **) &shader->vertexSource, 0
      );
      glCompileShaderARB(shader->vertex);
      shader_debug(shader->vertex);
      glAttachObjectARB(shader->program, shader->vertex);
    }
    if (shader->fragmentSource) {
      shader->fragment =
        glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);
      glShaderSourceARB(shader->fragment,
        1, (const GLcharARB **) &shader->fragmentSource, 0
      );
      glCompileShaderARB(shader->fragment);
      shader_debug(shader->fragment);
      glAttachObjectARB(shader->program, shader->fragment);
    }
    glLinkProgramARB(shader->program);
    shader_debug(shader->program);
    glGetObjectParameterivARB(shader->program,
      GL_OBJECT_LINK_STATUS_ARB, &shader->linkStatus
    );
    if (! shader->linkStatus) { return 0; }
  } else { return 0; }
  return shader;
}

struct {
  struct {
    struct shader shader;
    struct { GLint coords,       c,    r, n, h; } uniform;
    struct { int   coords; float c[2], r, n, h; } value;
  } iter;
  struct {
    struct shader shader;
    struct { GLint pix,       d;    } uniform;
    struct { int   pix; float d[2]; } value;
  } blur;
  GLuint fbo;
  int which;
  GLuint tex[3];
  int texw;
  int texh;
  int ewhich;
  GLuint etex[3];
  int etexw;
  int etexh;
  GLuint blurred;
  GLuint stats[12];
  int winw;
  int winh;
```

```
120     enum { julia_explore, julia_render, julia_walk } mode;
        int i;
        float f;
        int fs;
        unsigned char *buf;
125
        // walk mode
        struct {
          int which;                    // textures
          GLuint tex[3];
130       int texw, texh, over;
          float minx, miny, maxx, maxy; // region parameters
          int n, m, maxn, maxm;         // grid parameters
          float cx, cy, rx, ry, a0, da; // ellipse parameters
          int i, count;                 // animation parameters
135       FILE *outf;                   // saving
        } walk;

      } julia;

140   const char *julia_frag =
      "uniform sampler2D coords;\n"
      "uniform vec2 c;\n"
      "uniform float r;\n"
      "uniform float n;\n"
145   "uniform float h;\n"
      "\n"
      "void main(void) {\n"
      "  vec2 p = gl_TexCoord[0].st;\n"
      "  vec4 q = texture2D(coords, p);\n"
150   "  vec4 o;\n"
      "  if (q.a < 0.5) {\n"
      "    vec2 z = q.rg;\n"
      "    vec2 zz = vec2(z.x * z.x - z.y * z.y, 2.0 * z.x * z.y) + c;\n"
      "    float d = sqrt(dot(zz,zz));\n"
155   "    if (d <= r) {\n"
      "      o = vec4(zz, 0.0, 0.0);\n"
      "    } else {\n"
      "      float v = 0.025 * (h + n - log2(log2(d)/log2(r)));\n"
      "      o = vec4(cos(sqrt(5.0) * v) * 0.5 + 0.5, cos(sqrt(6.0) * v + 1.0) * 0.5 +↙
          ↳  0.5, cos(sqrt(7.0) * v + 2.0) * 0.5 + 0.5, 1.0);"
160   "    }\n"
      "  } else {\n"
      "   o = q;\n"
      "  }\n"
      "  gl_FragData[0] = o;\n"
165   "}\n"
      ;

      const char *blur_frag =
      "uniform sampler2D pix;\n"
170   "uniform vec2 d;\n"
      "\n"
      "void main(void) {\n"
      "  vec2 p = gl_TexCoord[0].st;\n"
      "  vec4 q;\n"
175   "  q  = texture2D(pix, p);\n"
```

```
"  q += texture2D(pix, p + vec2(d.x, 0.0));\n"
"  q += texture2D(pix, p + vec2(d.x, d.y));\n"
"  q += texture2D(pix, p + vec2(0.0, d.y));\n"
"  q *= 0.25;\n"
"  gl_FragData[0] = q;\n"
"}\n"
;

void julia_reshape(int w, int h) {
  julia.winw = w;
  julia.winh = h;
  if (julia.buf) {
    free(julia.buf);
  }
  julia.buf = malloc(w * h * 3);
}

void julia_next(void) {
  julia.mode = julia_explore;
  julia.iter.value.c[0] = (rand() / (double) RAND_MAX - 0.5) * 4.0;
  julia.iter.value.c[1] = (rand() / (double) RAND_MAX - 0.5) * 4.0;
  julia.iter.value.n = 0;
  julia.ewhich = 2;
  julia.f = -1;
  julia.fs = 0;
}

void julia_save(void) {
  julia.mode = julia_render;
  julia.iter.value.n = 0;
  julia.which = 2;
  julia.f = -1;
  julia.fs = 0;
}

void julia_walkfrom(int n, int m) {
  julia.mode = julia_walk;
  julia.iter.value.n = 0;
  julia.walk.which = 2;

  // handle borders
  int bit;
  if      (0 == n              ) { bit = 1; }
  else if (n == julia.walk.maxn) { bit = 0; }
  else if (0 == m              ) { bit = 1; }
  else if (m == julia.walk.maxm) { bit = 0; }
  else                           { bit = (rand() / (double) RAND_MAX) < 0.5; }

  int nn, mm, cx, cy, a0, da;
  switch (100 * (n%4) + 10 * (m%4) + bit) {

  case  10: nn = n-1; mm = m-1; cx = n-1; cy = m; a0 = 0; da =  1; break;
  case  11: nn = n+1; mm = m-1; cx = n+1; cy = m; a0 = 2; da = -1; break;
  case  30: nn = n-1; mm = m+1; cx = n-1; cy = m; a0 = 0; da = -1; break;
  case  31: nn = n+1; mm = m+1; cx = n+1; cy = m; a0 = 2; da =  1; break;

  case 100: nn = n+1; mm = m-1; cx = n; cy = m-1; a0 = 3; da =  1; break;
```

```
    case 101: nn = n+1; mm = m+1; cx = n; cy = m+1; a0 = 1; da = -1; break;
    case 120: nn = n-1; mm = m-1; cx = n; cy = m-1; a0 = 3; da = -1; break;
    case 121: nn = n-1; mm = m+1; cx = n; cy = m+1; a0 = 1; da =  1; break;

    case 210: nn = n-1; mm = m+1; cx = n-1; cy = m; a0 = 0; da = -1; break;
    case 211: nn = n+1; mm = m+1; cx = n+1; cy = m; a0 = 2; da =  1; break;
    case 230: nn = n-1; mm = m-1; cx = n-1; cy = m; a0 = 0; da =  1; break;
    case 231: nn = n+1; mm = m-1; cx = n+1; cy = m; a0 = 2; da = -1; break;

    case 300: nn = n-1; mm = m-1; cx = n; cy = m-1; a0 = 3; da = -1; break;
    case 301: nn = n-1; mm = m+1; cx = n; cy = m+1; a0 = 1; da =  1; break;
    case 320: nn = n+1; mm = m-1; cx = n; cy = m-1; a0 = 3; da =  1; break;
    case 321: nn = n+1; mm = m+1; cx = n; cy = m+1; a0 = 1; da = -1; break;

    default: fprintf(stderr, "julia walk invariant broken\n"); exit(1); break;
    }
    if (julia.walk.outf) {
      fclose(julia.walk.outf);
    }
    char filename[1024];
    snprintf(filename, 1000, "%02d_%02d__%02d_%02d.ppm", n, m, nn, mm);
    struct stat s;
    if (stat(filename, &s) < 0) {
      julia.walk.outf = fopen(filename, "wb");
    } else {
      julia.walk.outf = 0;
    }
    julia.walk.n = nn;
    julia.walk.m = mm;
    julia.walk.cx = julia.walk.minx + (julia.walk.maxx - julia.walk.minx) * ↙
        ↳                    cx  / (float) julia.walk.maxn;
    julia.walk.cy = julia.walk.miny + (julia.walk.maxy - julia.walk.miny) * (julia↙
        ↳ .walk.maxm - cy) / (float) julia.walk.maxm;
    julia.walk.a0 = 3.1415926 / 2.0 * a0;
    julia.walk.da = 3.1415926 / 2.0 * da / (float) julia.walk.count;
    julia.walk.rx = (julia.walk.maxx - julia.walk.minx) / julia.walk.maxn;
    julia.walk.ry = (julia.walk.maxy - julia.walk.miny) / julia.walk.maxm;
    julia.walk.i  = 0;
    if (! julia.walk.outf) {
      julia_walkfrom(julia.walk.n, julia.walk.m);
    }
  }

  void julia_walkstep(void) {
    julia.mode = julia_walk;
    julia.walk.which = 2;
    julia.f = -1;
    julia.fs = 0;
    julia.walk.i += 1;
    if (julia.walk.i == julia.walk.count) {
      julia_walkfrom(julia.walk.n, julia.walk.m);
    }
  }

  void julia_init(int tsize, int etsize) {
    julia.buf = 0;
    glGenFramebuffersEXT(1, &julia.fbo);
```

```
    // rendering
    julia.texw = tsize;
    julia.texh = tsize;
    glGenTextures(3, &julia.tex[0]);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, julia.tex[0]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, tsize, tsize, 0, GL_RGBA, ↙
        ↳ GL_FLOAT, 0);
    glBindTexture(GL_TEXTURE_2D, julia.tex[1]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, tsize, tsize, 0, GL_RGBA, ↙
        ↳ GL_FLOAT, 0);
    glBindTexture(GL_TEXTURE_2D, julia.tex[2]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    float *px = calloc(tsize * tsize * 4, sizeof(float));
    for (int y = 0; y < tsize; ++y) {
    for (int x = 0; x < tsize; ++x) {
      px[4 * (y * tsize + x) + 0] = x * 3.0 / tsize - 1.5;
      px[4 * (y * tsize + x) + 1] = y * 3.0 / tsize - 1.5;
      px[4 * (y * tsize + x) + 2] = 0.0;
      px[4 * (y * tsize + x) + 3] = 0.0;
    }
    }
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, tsize, tsize, 0, GL_RGBA, ↙
        ↳ GL_FLOAT, px);
    free(px);
    // exploration
    julia.etexw = etsize;
    julia.etexh = etsize;
    glGenTextures(3, &julia.etex[0]);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, julia.etex[0]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, etsize, etsize, 0, GL_RGBA, ↙
        ↳ GL_FLOAT, 0);
    glBindTexture(GL_TEXTURE_2D, julia.etex[1]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, etsize, etsize, 0, GL_RGBA, ↙
        ↳ GL_FLOAT, 0);
    glBindTexture(GL_TEXTURE_2D, julia.etex[2]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    float *epx = calloc(etsize * etsize * 4, sizeof(float));
    for (int y = 0; y < etsize; ++y) {
    for (int x = 0; x < etsize; ++x) {
      epx[4 * (y * etsize + x) + 0] = x * 3.0 / etsize - 1.5;
      epx[4 * (y * etsize + x) + 1] = y * 3.0 / etsize - 1.5;
      epx[4 * (y * etsize + x) + 2] = 0.0;
      epx[4 * (y * etsize + x) + 3] = 0.0;
    }
    }
```

```
340     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, etsize, etsize, 0, GL_RGBA, ↙
            ↳ GL_FLOAT, epx);
        free(epx);
        // blurring
        glGenTextures(1, &julia.blurred);
        glEnable(GL_TEXTURE_2D);
345     glBindTexture(GL_TEXTURE_2D, julia.blurred);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, tsize/2, tsize/2, 0, GL_RGBA, ↙
            ↳ GL_FLOAT, 0);
        glBindTexture(GL_TEXTURE_2D, 0);
350     glDisable(GL_TEXTURE_2D);
        glGenTextures(12, &julia.stats[0]);
        glEnable(GL_TEXTURE_2D);
        for (int i = 0; i < 12; ++i) {
          glBindTexture(GL_TEXTURE_2D, julia.stats[i]);
355       glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
          glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
          glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, 1 << i, 1 << i, 0, GL_RGBA, ↙
              ↳ GL_FLOAT, 0);
        }
        glBindTexture(GL_TEXTURE_2D, 0);
360     glDisable(GL_TEXTURE_2D);
        shader_init(&julia.iter.shader, 0, julia_frag);
        shader_uniform(&julia.iter, coords);
        shader_uniform(&julia.iter, c);
        shader_uniform(&julia.iter, r);
365     shader_uniform(&julia.iter, n);
        shader_uniform(&julia.iter, h);
        julia.iter.value.coords = 0;
        julia.iter.value.r = 1024;
        julia.iter.value.h = 0;
370     shader_init(&julia.blur.shader, 0, blur_frag);
        shader_uniform(&julia.blur, pix);
        shader_uniform(&julia.blur, d);
        julia.blur.value.pix = 0;
        julia.blur.value.d[0] = 1.0/tsize;
375     julia.blur.value.d[1] = 1.0/tsize;
        julia.i = 0;

        // walk mode
        julia.walk.count = 75;
380     julia.walk.minx = -0.7 - 1.5;
        julia.walk.miny = -1;
        julia.walk.maxx = -0.7 + 1.5;
        julia.walk.maxy =  1;
        julia.walk.maxn = 12;
385     julia.walk.maxm =  8;
        julia.walk.over = 2;
        julia.walk.texw = 1024 * julia.walk.over;
        julia.walk.texh = 1024 * julia.walk.over;
        float *wpx = calloc(julia.walk.texw * julia.walk.texh * 4, sizeof(float));
390     for (int y = 0; y < julia.walk.texh; ++y) {
        for (int x = 0; x < julia.walk.texw; ++x) {
          wpx[4 * (y * julia.walk.texw + x) + 0] = x * 6.0 / julia.walk.texw - 3.0;
          wpx[4 * (y * julia.walk.texw + x) + 1] = y * 6.0 / julia.walk.texh - 3.0;
```

```
      wpx[4 * (y * julia.walk.texw + x) + 2] = 0.0;
      wpx[4 * (y * julia.walk.texw + x) + 3] = 0.0;
    }
    }
    glGenTextures(3, &julia.walk.tex[0]);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, julia.walk.tex[0]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, julia.walk.texw, julia.walk.↙
        ↪ texh, 0, GL_RGBA, GL_FLOAT, wpx);
    glBindTexture(GL_TEXTURE_2D, julia.walk.tex[1]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, julia.walk.texw, julia.walk.↙
        ↪ texh, 0, GL_RGBA, GL_FLOAT, wpx);
    glBindTexture(GL_TEXTURE_2D, julia.walk.tex[2]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, julia.walk.texw, julia.walk.↙
        ↪ texh, 0, GL_RGBA, GL_FLOAT, wpx);
    free(wpx);
    julia.walk.outf = 0;
    // start
//  julia_walkfrom(0, 1); // julia.walk.maxn / 2 + 1, julia.walk.maxm / 2);
    julia_next();
}

void julia_idle(void) {
  glutPostRedisplay();
}

void julia_keyboard(unsigned char key, int x, int y) {
  switch (key) {
  case 27: exit(0); break;
  default: /* julia_next(); */ break;
  }
}

float julia_escapees(GLuint tex, int log2texsize) {
  glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, julia.fbo);
  glUseProgramObjectARB(julia.blur.shader.program);
  glBindTexture(GL_TEXTURE_2D, tex);
  for (int i = log2texsize - 1; i >= 0; --i) {
    shader_updatei (&julia.blur, pix);
    julia.blur.value.d[0] = 1.0 / (2 << i);
    julia.blur.value.d[1] = 1.0 / (2 << i);
    shader_updatef2(&julia.blur, d);
    glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
        ↪ GL_TEXTURE_2D, julia.stats[i], 0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1, 0, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glViewport(0, 0, 1 << i, 1 << i);
    glBegin(GL_QUADS); { glColor4f(1,1,1,1);
```

```
      glTexCoord2f(0, 0); glVertex2f(0, 0);
      glTexCoord2f(1, 0); glVertex2f(1, 0);
      glTexCoord2f(1, 1); glVertex2f(1, 1);
      glTexCoord2f(0, 1); glVertex2f(0, 1);
    } glEnd();
    glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
        ↳ GL_TEXTURE_2D, 0, 0);
    glBindTexture(GL_TEXTURE_2D, julia.stats[i]);
  }
  glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
  glUseProgramObjectARB(0);
  float f = -1;
  glGetTexImage(GL_TEXTURE_2D, 0, GL_ALPHA, GL_FLOAT, &f);
  glBindTexture(GL_TEXTURE_2D, 0);
  return f;
}

void julia_snapshot(FILE *f) {
  glReadPixels(0, 0, julia.winw, julia.winh, GL_RGB, GL_UNSIGNED_BYTE, julia.buf↙
      ↳ );
  fprintf(f, "P6\n%d %d 255\n", julia.winw, julia.winh);
  for (int y = julia.winh - 1; y >= 0; --y) {
    fwrite(julia.buf + y * julia.winw * 3, julia.winw * 3, 1, f);
  }
}

void julia_display(void) {
  if (julia.mode == julia_walk) {
    // iterate
    julia.iter.value.c[0] = julia.walk.cx + julia.walk.rx * cos(julia.walk.a0 + ↙
        ↳ julia.walk.da * julia.walk.i);
    julia.iter.value.c[1] = julia.walk.cy + julia.walk.ry * sin(julia.walk.a0 + ↙
        ↳ julia.walk.da * julia.walk.i);
    julia.which = 2;
    julia.f = -1;
    julia.fs = 0;
    float tx0 = (julia.walk.texw - julia.walk.over * julia.winw) / 2.0 / julia.↙
        ↳ walk.texw;
    float ty0 = (julia.walk.texh - julia.walk.over * julia.winh) / 2.0 / julia.↙
        ↳ walk.texh;
    float tx1 = (julia.walk.texw + julia.walk.over * julia.winw) / 2.0 / julia.↙
        ↳ walk.texw;
    float ty1 = (julia.walk.texh + julia.walk.over * julia.winh) / 2.0 / julia.↙
        ↳ walk.texh;
    for (int i = 0; i < 4096; ++i) {
      glEnable(GL_TEXTURE_2D);
      glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, julia.fbo);
      glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
          ↳ GL_TEXTURE_2D, julia.walk.tex[(julia.walk.which + 1) & 1], 0);
      glBindTexture(GL_TEXTURE_2D, julia.walk.tex[julia.walk.which]);
      glUseProgramObjectARB(julia.iter.shader.program);
      shader_updatei (&julia.iter, coords);
      shader_updatef2(&julia.iter, c);
      shader_updatef (&julia.iter, r);
      shader_updatef (&julia.iter, h);
      julia.iter.value.n = i;
      shader_updatef (&julia.iter, n);
```

```
495         glMatrixMode(GL_PROJECTION);
            glLoadIdentity();
            gluOrtho2D(0, 1, 0, 1);
            glMatrixMode(GL_MODELVIEW);
            glLoadIdentity();
500         glViewport((julia.walk.texw - julia.walk.over * julia.winw) / 2, (julia.↙
                ↳ walk.texh - julia.walk.over * julia.winh) / 2, julia.walk.over * ↙
                ↳ julia.winw, julia.walk.over * julia.winh);
            glBegin(GL_QUADS); { glColor4f(1,1,1,1);
              glTexCoord2f(tx0, ty0); glVertex2f(0, 0);
              glTexCoord2f(tx1, ty0); glVertex2f(1, 0);
              glTexCoord2f(tx1, ty1); glVertex2f(1, 1);
505           glTexCoord2f(tx0, ty1); glVertex2f(0, 1);
            } glEnd();
            glBindTexture(GL_TEXTURE_2D, 0);
            glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
                ↳ GL_TEXTURE_2D, 0, 0);
            glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
510         julia.walk.which = (julia.walk.which + 1) & 1;
            if ((i & 15) == 15) {
              float f = julia_escapees(julia.walk.tex[julia.walk.which], 9 + julia.↙
                  ↳ walk.over);
              if (! (f > julia.f)) {
                julia.fs = 1;
515           } else {
                julia.fs = 0;
              }
              julia.f = f;
            }
520         if (julia.fs) break;
          }
          glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
          glUseProgramObjectARB(0);
          // display
525       glClear(GL_COLOR_BUFFER_BIT);
          glEnable(GL_BLEND);
          glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
          glBindTexture(GL_TEXTURE_2D, julia.walk.tex[julia.walk.which]);
          glMatrixMode(GL_PROJECTION);
530       glLoadIdentity();
          gluOrtho2D(0, 1, 0, 1);
          glMatrixMode(GL_MODELVIEW);
          glLoadIdentity();
          glViewport(0, 0, julia.winw, julia.winh);
535       if (julia.walk.over > 1) {
            glUseProgramObjectARB(julia.blur.shader.program);
            shader_updatei (&julia.blur, pix);
            julia.blur.value.d[0] = 1.0 / julia.walk.texw;
            julia.blur.value.d[1] = 1.0 / julia.walk.texh;
540         shader_updatef2(&julia.blur, d);
          }
          glBegin(GL_QUADS); { glColor4f(1,1,1,1);
            glTexCoord2f(tx0, ty0); glVertex2f(0, 0);
            glTexCoord2f(tx1, ty0); glVertex2f(1, 0);
545         glTexCoord2f(tx1, ty1); glVertex2f(1, 1);
            glTexCoord2f(tx0, ty1); glVertex2f(0, 1);
          } glEnd();
```

```
    if (julia.walk.over > 1) {
      glUseProgramObjectARB(0);
    }
    glDisable(GL_TEXTURE_2D);
    glDisable(GL_BLEND);
    glutSwapBuffers();
    if (julia.walk.outf) {
      julia_snapshot(julia.walk.outf);
    }
    // next
    julia_walkstep();
  } else if (julia.mode == julia_render) {
    // iteration step
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, julia.tex[julia.which]);
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, julia.fbo);
    glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
        ↳ GL_TEXTURE_2D, julia.tex[(julia.which + 1) & 1], 0);
    glUseProgramObjectARB(julia.iter.shader.program);
    shader_updatei (&julia.iter, coords);
    shader_updatef2(&julia.iter, c);
    shader_updatef (&julia.iter, r);
    shader_updatef (&julia.iter, n);
    shader_updatef (&julia.iter, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1, 0, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glViewport(0, 0, julia.texw, julia.texh);
    glBegin(GL_QUADS); { glColor4f(1,1,1,1);
      glTexCoord2f(0, 0); glVertex2f(0, 0);
      glTexCoord2f(1, 0); glVertex2f(1, 0);
      glTexCoord2f(1, 1); glVertex2f(1, 1);
      glTexCoord2f(0, 1); glVertex2f(0, 1);
    } glEnd();
    julia.iter.value.n += 1;
    glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
        ↳ GL_TEXTURE_2D, 0, 0);
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
    // gather statistics
    int k = julia.iter.value.n;
    if ((k & 15) == 15) {
      float f = julia_escapees(julia.tex[julia.which], 12);
      if (! (f > julia.f)) {
        julia.fs = 1;
      } else {
        julia.fs = 0;
      }
      julia.f = f;
    }
    // blur for display
    julia.which = (julia.which + 1) & 1;
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, julia.fbo);
    glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
        ↳ GL_TEXTURE_2D, julia.blurred, 0);
    glClear(GL_COLOR_BUFFER_BIT);
```

```
    glUseProgramObjectARB(julia.blur.shader.program);
    shader_updatei (&julia.blur, pix);
    julia.blur.value.d[0] = 1.0 / julia.texw;
    julia.blur.value.d[1] = 1.0 / julia.texh;
    shader_updatef2(&julia.blur, d);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glBindTexture(GL_TEXTURE_2D, julia.tex[julia.which]);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1, 0, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glViewport(0, 0, julia.texw/2, julia.texh/2);
    glBegin(GL_QUADS); { glColor4f(1,1,1,1);
      glTexCoord2f(0, 0); glVertex2f(0, 0);
      glTexCoord2f(1, 0); glVertex2f(1, 0);
      glTexCoord2f(1, 1); glVertex2f(1, 1);
      glTexCoord2f(0, 1); glVertex2f(0, 1);
    } glEnd();
    glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, ↙
        ↳ GL_TEXTURE_2D, 0, 0);
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBindTexture(GL_TEXTURE_2D, julia.blurred);
    shader_updatei (&julia.blur, pix);
    julia.blur.value.d[0] = 2.0 / julia.texw;
    julia.blur.value.d[1] = 2.0 / julia.texh;
    shader_updatef2(&julia.blur, d);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1, 0, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glViewport(0, 0, julia.winw, julia.winh);
    float a = julia.winh * 0.5 / julia.winw;
    glBegin(GL_QUADS); { glColor4f(1,1,1,1);
      glTexCoord2f(0, 0.5 - a); glVertex2f(0, 0);
      glTexCoord2f(1, 0.5 - a); glVertex2f(1, 0);
      glTexCoord2f(1, 0.5 + a); glVertex2f(1, 1);
      glTexCoord2f(0, 0.5 + a); glVertex2f(0, 1);
    } glEnd();
    glDisable(GL_TEXTURE_2D);
    glUseProgramObjectARB(0);
    glDisable(GL_BLEND);
    glutSwapBuffers();
    glutReportErrors();
    // do stats stuff
    if (julia.fs) {
      char filename[1024];
      snprintf(filename, 1000, "julia_%+08.16f%+08.16fi.ppm", julia.iter.value.c↙
          ↳ [0], julia.iter.value.c[1]);
      FILE *f = fopen(filename, "wb");
      if (f) {
        julia_snapshot(f);
        fclose(f);
      }
```

```
      julia_next();
    }
  } else if (julia.mode == julia_explore) {
    // iteration step
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, julia.etex[julia.ewhich]);
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, julia.fbo);
    glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT,
        GL_TEXTURE_2D, julia.etex[(julia.ewhich + 1) & 1], 0);
    glUseProgramObjectARB(julia.iter.shader.program);
    shader_updatei (&julia.iter, coords);
    shader_updatef2(&julia.iter, c);
    shader_updatef (&julia.iter, r);
    shader_updatef (&julia.iter, n);
    shader_updatef (&julia.iter, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1, 0, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glViewport(0, 0, julia.etexw, julia.etexh);
    glBegin(GL_QUADS); { glColor4f(1,1,1,1);
      glTexCoord2f(0, 0); glVertex2f(0, 0);
      glTexCoord2f(1, 0); glVertex2f(1, 0);
      glTexCoord2f(1, 1); glVertex2f(1, 1);
      glTexCoord2f(0, 1); glVertex2f(0, 1);
    } glEnd();
    julia.iter.value.n += 1;
    glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT,
        GL_TEXTURE_2D, 0, 0);
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
    julia.ewhich = (julia.ewhich + 1) & 1;
    // gather statistics
    int k = julia.iter.value.n;
    if ((k & 15) == 15) {
      float f = julia_escapees(julia.etex[julia.ewhich], 8);
      if (! (f > julia.f)) {
        julia.fs = 1;
      } else {
        julia.fs = 0;
      }
      julia.f = f;
    }
    // display
    glUseProgramObjectARB(0);
    glClear(GL_COLOR_BUFFER_BIT);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glBindTexture(GL_TEXTURE_2D, julia.etex[julia.ewhich]);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1, 0, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glViewport(0, 0, julia.winw, julia.winh);
    float a = julia.winh * 0.5 / julia.winw;
    glBegin(GL_QUADS); { glColor4f(1,1,1,1);
```

```
      glTexCoord2f(0, 0.5 - a); glVertex2f(0, 0);
      glTexCoord2f(1, 0.5 - a); glVertex2f(1, 0);
      glTexCoord2f(1, 0.5 + a); glVertex2f(1, 1);
      glTexCoord2f(0, 0.5 + a); glVertex2f(0, 1);
    } glEnd();
    glDisable(GL_TEXTURE_2D);
    glDisable(GL_BLEND);
    glutSwapBuffers();
    // do stats stuff
    if (julia.fs) {
      if (julia.iter.value.n > 16 * 6) {
        julia_save();
      } else {
        julia_next();
      }
    }
  }
  glutReportErrors();
}

int main(int argc, char **argv) {
  fprintf(stderr, "julia (GPL) 2010 Claude Heiland-Allen <claude@mathr.co.uk>\n");
  srand(time(NULL));
  glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
  glutInitWindowSize(1024, 1024);
  glutInit(&argc, argv);
  glutCreateWindow("julia");
  glewInit();
  julia_init(4096, 256);
  glutDisplayFunc(julia_display);
  glutReshapeFunc(julia_reshape);
  glutIdleFunc(julia_idle);
  glutKeyboardFunc(julia_keyboard);
  glutMainLoop();
  return 0;
}
```

# 4 MakeDVD.hs

```
import System.Environment(getArgs)
import System.Exit(exitFailure)
import System.IO(hPutStrLn, stderr)
import Control.Monad(when)
import Data.Ord(comparing)
import Data.List(sortBy)
import Data.Map((!))
import qualified Data.Map as M
import qualified Data.Set as S

default(Int)

chunk :: Int -> [a] -> [[a]]
chunk n [] = []
chunk n xs = let (ys, zs) = splitAt n xs in ys : chunk n zs

{-
```

```
  DVD VM usage
  ------------
  g0 = random branching
  g1 = starting video
  fpc : sets g1 at random
  titleset 1 menu entry root : jump table to title g1
  title N : chooses random successor title
-}

dvd :: [String] -> String
dvd mpegs =
  unlines
    [ "<!-- autogenerated file, do not edit -->"
    , "<dvdauthor jumppad=\"no\">"
    , "<vmgm><fpc>{ g1 = random(" ++ show (M.size trans) ++ "); jump titleset 1 menu entry root; }</fpc></vmgm>"
    , titleset . map title . sortBy (comparing snd) . M.toList $ trans
    , "</dvdauthor>"
    ]
  where
    mpeg f t = f ++ "__" ++ t ++ ".mpeg" -- rebuild filename
    froms  = map (take 5         ) mpegs -- FIXME hardcoded filename format
    tos    = map (take 5 . drop 8) mpegs -- FIXME hardcoded filename format
    edges  = zip froms tos
    source = M.fromList (zip mpegs froms)
    sink   = M.fromList (zip mpegs tos)
    nodes  = M.fromList $ zip (S.toList (S.fromList (M.elems source) `S.union` S.fromList (M.elems sink))) [1..]
    trans  = M.fromList $ zip mpegs [1..]
    titleset titles =
      unlines
        [ "<titleset><menus>"
        , menus (length titles)
        , "</menus><titles>"
        , concat titles
        , "</titles></titleset>"
        ]
    title (vob, tit) =
      unlines
        [ "<pgc ><!-- " ++ show tit ++ " -->"
        , "<pre>{"
        , if m > 1 then "  g0 = random(" ++ show m ++ ");" else "  g0 = 1;"
        , "}</pre>"
        , "<vob file=\"" ++ vob ++ "\" />"
        , "<post>{"
        , switch "g0" m [1 .. m] (map snd to) $ \t ->
            "{ jump title " ++ show (trans ! mpeg from t) ++ "; }"
        , "}</post>"
        , "</pgc><!-- " ++ show tit ++ " -->"
        ]
      where
        from = sink ! vob
        m = length to
        to = filter ((from ==) . fst) edges
    menus n =
      unlines
        [ concat submenus
```

```
          , "<pgc entry=\"root\">"
          , "<pre>{"
          , switch "g1" ss [1, 17 ..] [1 .. ss] $ \t -> "jump menu " ++ show t ++ ";"
          , "}</pre><vob file=\"menu.mpeg\" /></pgc>"
          ]
        where
          ss = length submenus
          targets = map unzip . chunk 16 $ zip [1 .. n] [1 .. n]
          submenus = zipWith menu [1..] targets
          menu k (xs, ys) =
            unlines
              [ "<pgc ><!-- " ++ show k ++ " -->"
              , "<pre>{"
              , switch "g1" (length ys) xs ys $ \t -> "jump title " ++ show t ++ ";"
              , "}</pre><vob file=\"menu.mpeg\" />"
              , "</pgc><!-- " ++ show k ++ " -->"
              ]
    switch r k xs ts f
      | k > 1 = unlines
                  [ "if (" ++ r ++ " lt " ++ show (head xs2) ++ ") {"
                  , switch r k2 xs1 ts1 f
                  , "} else {"
                  , switch r (k - k2) xs2 ts2 f
                  , "}"
                  ]
      | k == 1 = f (head ts)
      | otherwise = error "empty switch"
      where
        k2 = k `div` 2
        (xs1, xs2) = splitAt k2 xs
        (ts1, ts2) = splitAt k2 ts

-- usage: ./MakeDVD *__*.mpeg > dvd.xml
main :: IO ()
main = do
  mpegs <- getArgs
  when (length mpegs > 99) $ do -- FIXME possible to use chapters within titles?
    hPutStrLn stderr "only 99 titles allowed, aborting"
    exitFailure
  putStr $ dvd mpegs
```

# 5 Makefile

```
julia: julia.c
	gcc -std=c99 -Wall -pedantic -O2 -o julia julia.c  -lGL -lGLU -lGLEW -lglut
```

# 6 multiplex.sh

```
#!/bin/bash
for M2V in $@
do
  MP2="../silence75.mp2"
  MPEG="${M2V%m2v}mpeg"
  mplex -f 8 -V -o "${MPEG}" "${M2V}" "${MP2}"
```

```
done
```

# 7 README

```
make && mkdir img && cd img && ../julia
the "interesting" images will be saved as ppm files


(old stuff for making grids below...)

make && mkdir img && cd img && ../julia | pnmsplit
rename s/image/image0/ image?
rename s/image/image0/ image??
rename s/image/image0/ image???
rename s/image/image0/ image????
rename s/image// image?????
rename s/\$/.ppm/ ?????
ls -1 | xargs -n 256 pnmcat -lr | pnmsplit
rename s/image/image0/ image?
rename s/image/image0/ image??
rename s/image/row/ image???
rename s/\$/.ppm/ row???
ls -1 | sort -r | xargs -n 256 pnmcat -tb > full.ppm
```

# 8 transitions.sh

```
#!/bin/bash
(
  echo 'digraph G {'
  echo '  node[label="",shape="circle",style="filled"];'
  edges=$(ls -1 | grep ".ppm$" | sed "s|.ppm$||")
  nodes=$(for edge in ${edges} ; do echo ${edge} ; done | sed "s|_-_.*$||" | ↙
      ↳ sort | uniq)
  for node in ${nodes}
  do
    colour=$(echo ${node} | sed "s|_.*$||")
    case ${colour} in
      0)
        echo "node[fillcolor=\"green\"]; node_${node};"
        ;;
      1)
        echo "node[fillcolor=\"white\"]; node_${node};"
        ;;
      2)
        echo "node[fillcolor=\"orange\"]; node_${node};"
        ;;
    esac
  done
  for edge in ${edges}
  do
    source=$(echo ${edge} | sed "s|_-_.*$||")
    sink=$(echo ${edge} | sed "s|^.*_-_||")
    echo "node_${source} -> node_${sink};"
  done
  echo '}'
) > transitions.dot
```

```
30  neato -Tpng < transitions.dot > transitions.png
```