shaders/model.fs

```
#version 330 core

in float    visibility;
in vec3     pass_normal;
in vec2     pass_uv;
flat in int pass_textureID;
in vec3     viewVec;

out vec4 vertexColor;

uniform vec3 sky_color;
uniform int state;
uniform vec3 sunray;

uniform sampler2D textureSampler;

# define STATE_APPLY_PHONG_LIGHTNING (2)
# define STATE_SPECULAR (32)
# define TX_UNIT (1 / 5.0)

float getDampingFactor() {
    if (pass_textureID == 0) {
        return (8.0);
    }
    if (pass_textureID == 4) {
        return (4.0);
    }
    return 16.0;
}

void main(void) {
    //texture
    float uvx = (pass_uv.x * TX_UNIT + pass_textureID * TX_UNIT);
    float uvy = pass_uv.y;
    vec3 txcolor = texture(textureSampler, vec2(uvx, uvy)).rgb;
    vec3 color = txcolor;
    //phong ligthing model
    if ((state & STATE_APPLY_PHONG_LIGHTNING) !=
       STATE_APPLY_PHONG_LIGHTNING) {
        float intensity = max(dot(pass_normal, sunray), 0.2);
        color *= intensity;
        if ((state & STATE_SPECULAR) != STATE_SPECULAR) {
            //specular lighting
            vec3 reflectVec = reflect(-sunray, pass_normal);
            float specAngle = max(dot(reflectVec, viewVec), 0.0);
            float specular = pow(specAngle, getDampingFactor());
            color += specular * vec3(1.0, 1.0, 1.0);
        }
    }
    //apply fog
    vertexColor = vec4(mix(color, sky_color, visibility), 1.0);
}
```

shaders/model.vs

```
#version 330 core

in vec2   pos;
in vec2   uv;
in float  height;
in vec2   normal;
in int    textureID;

out float     visibility;
out vec3      pass_normal;
out vec2      pass_uv;
flat out int  pass_textureID;
out vec3      viewVec;

//view and projection matrix
uniform mat4 mvp_matrix;

//transformation matrix
uniform mat4 transf_matrix;
uniform int state;
uniform int time;

# define TERRAIN_SIZE (16.0)
# define TERRAIN_RENDER_DISTANCE (64)
# define RENDER_DISTANCE (TERRAIN_RENDER_DISTANCE * TERRAIN_SIZE)

# define STATE_APPLY_FOG (1)
# define STATE_APPLY_PHONG_LIGHTNING (2)

void main(void) {

    vec4 world_pos = transf_matrix * vec4(pos.x, height, pos.y, 1.0);
    viewVec = normalize(-world_pos.xyz);
    gl_Position = mvp_matrix * world_pos;

    //fog calculation
    visibility = 0.0f;
    if ((state & STATE_APPLY_FOG) != STATE_APPLY_FOG) {
      //visibility^8
        visibility = length(gl_Position.xz) / float(RENDER_DISTANCE);
      visibility = visibility * visibility;
      visibility = visibility * visibility;
      visibility = visibility * visibility;
      visibility = clamp(visibility, 0, 1);
    }

    pass_normal = normalize(vec3(normal.x, 1.0 / TERRAIN_SIZE, normal.y));
    pass_uv = uv;
    pass_textureID = textureID;
```

```
}
```

srcs/biom.c

```
#include "renderer.h"

static float clamp(float val, float min, float max) {
    if (val > max) {
        return (max);
    }
    return (val < min ? min : val);
}

static int biomMountainGenColor(t_world * world, t_biom * biom, float wx,
   float wy, float wz) {
    float r = wy / world->max_height;
    if (r <= 0.08f) {
        return (TX_WATER);
    } else if (r <= 0.60f) {
        return (TX_GRASS);
    } else if (r <= 0.75f) {
        return (TX_DIRT);
    } else if(r <= 0.90f) {
        return (TX_STONE);
    } else {
        return (TX_SNOW);
    }
}

static float normalizeHeight(t_world * world, float heightFactor) {
    heightFactor += 1;
    heightFactor *= 0.5f;

    float minHeight = 0.08f;
    float maxHeight = 1.0f;
    if (heightFactor < minHeight) {
        heightFactor = minHeight;
    } else if (heightFactor > maxHeight) {
        heightFactor = maxHeight;
    }
    return (world->max_height * heightFactor);
}

/** the height generator function for moutains */
static float biomGenHeight(t_world * world, t_biom * biom, float wx, float
    wz) {

    float heightFactor = 0.0f;

    float frequency = biom->frequency;
    float amplitude = biom->amplitude;
    for (int i = 0 ; i < biom->octaves ; i++) {
```

```
47          heightFactor += pnoise2(world->octaves[i], wx * frequency, wz *
                frequency) * amplitude;
48          frequency *= biom->lacunarity;
49          amplitude *= biom->persistance;
50      }
51      return (normalizeHeight(world, heightFactor));
52  }
53
54  static float biomHeightmapGenHeight(t_world * world, t_biom * biom, float
        wx, float wz) {
55      int px = (int)(wx / TERRAIN_UNIT);
56      int py = (int)(wz / TERRAIN_UNIT);
57      px = clamp(px, 0, world->heightmap->w - 1);
58      py = clamp(py, 0, world->heightmap->h - 1);
59
60      int rgb = heightmapGetHeight(world->heightmap, px, py);
61      float height = rgb / (255.0f * 3.0f);
62      return (clamp(height, 0, height * world->max_height));
63  }
64
65  static int biomHeightmapCanGenerate(t_world * world, t_biom * biom, float
        wx, float wz) {
66      int px = (int)(wx / TERRAIN_UNIT);
67      int py = (int)(wz / TERRAIN_UNIT);
68      return (px >= 0 && py >= 0 && px < world->heightmap->w && py < world->
            heightmap->h);
69  }
70
71  static int biomCanGenerate(t_world * world, t_biom * biom, float wx, float
        wz) {
72      return (1);
73  }
74
75  static void biomRegister(t_world * world,
76                           float (*heightGen)(t_world *, struct s_biom *,
                                float, float),
77                           int (*colorGen)(t_world *, struct s_biom *,
                               float, float, float),
78                           int (*canGen)(t_world *, struct s_biom *,
                               float, float),
79                           float heightGenStep, int octaves,
80                           float amplitude, float persistance,
81                           float frequency, float lacunarity) {
82      t_biom biom;
83      biom.heightGen = heightGen;
84      biom.colorGen = colorGen;
85      biom.canGenerateAt = canGen;
86      biom.heightGenStep = heightGenStep;
87      biom.octaves = octaves;
88      biom.amplitude = amplitude;
89      biom.frequency = frequency;
90      biom.persistance = persistance;
91      biom.lacunarity = lacunarity;
92      array_list_add(world->bioms, &biom);
```

```
}

void biomsInit(t_world * world) {
    if (world->heightmap == NULL) {
        printf("No heightmaps set, generating terrain procedurally\n");
        float step = TERRAIN_UNIT / 16.0f;
        biomRegister(world, biomGenHeight, biomMountainGenColor,
           biomCanGenerate, step, 4, 1.0f, 0.5f, 0.03f, 2.0f);
    } else {
        printf("Heightmap in use\n");
        biomRegister(world, biomHeightmapGenHeight, biomMountainGenColor,
           biomHeightmapCanGenerate, TERRAIN_UNIT, 0, 0, 0, 0, 0);
    }
}

void biomsDelete(t_world * world) {
    array_list_delete(world->bioms);
    free(world->bioms);
}
```

srcs/camera.c

```
#include "renderer.h"

void cameraInit(t_camera * camera) {

    camera->pos.x = TERRAIN_SIZE * 2, camera->pos.y = TERRAIN_SIZE * 2,
       camera->pos.z = TERRAIN_SIZE * 2;
    camera->rot.pitch = 0, camera->rot.yaw = 0, camera->rot.roll = 0;
    camera->fov = DEG_TO_RAD(70.0f);
    camera->near_distance = 0.01f;
    camera->far_distance = TERRAIN_RENDER_DISTANCE *
       TERRAIN_RENDER_DISTANCE * TERRAIN_SIZE;
    camera->movespeed = 2.0f;
}

void cameraDelete(t_camera * camera) {
}

static void cameraUpdateMatrices(t_camera * camera) {

    //matrices
    t_vec3f * viewvec = &(camera->vview);
    t_mat4f * view = &(camera->mview);
    t_mat4f * proj = &(camera->mproj);
    t_mat4f * viewproj = &(camera->mviewproj);

    //view vector
    float pitch = DEG_TO_RAD(camera->rot.pitch);
    float yaw = DEG_TO_RAD(camera->rot.yaw);
    float roll = DEG_TO_RAD(camera->rot.roll);
    float cospitch = cos(pitch);
```

```
29      viewvec->x = cospitch * sin(yaw);
30      viewvec->y = -sin(pitch);
31      viewvec->z = -cospitch * cos(yaw);
32      vec3f_normalize(viewvec, viewvec);
33
34      //view matrix
35      mat4f_identity(view);
36      mat4f_rotateX(view, view, pitch);
37      mat4f_rotateY(view, view, yaw);
38      mat4f_rotateZ(view, view, roll);
39      mat4f_translate(view, view, -camera->pos.x, -camera->pos.y, -camera->
            pos.z);
40
41      //projection matrix
42      float aspect = 1.6f;
43      mat4f_perspective(proj, aspect, camera->fov, camera->near_distance,
            camera->far_distance);
44
45      //combine view and projection
46      mat4f_mult(viewproj, proj, view);
47  }
48
49  void cameraUpdate(t_glh_context * context, t_world * world, t_renderer *
        renderer, t_camera * camera) {
50      //update camera matrices
51      cameraUpdateMatrices(camera);
52      //update camera world index
53      worldGetGridIndex(world, camera->pos.x, camera->pos.z, &(camera->
            terrain_index.x), &(camera->terrain_index.y));
54  }
```

srcs/glh.c

```
1   #include "glh.h"
2
3   int _glh_debug = 0;
4   t_glh_context * _glh_context;
5
6   void errorCallback(int error, const char* description) {
7       fprintf(stderr, "GL error: %s (%d)\n", description, error);
8   }
9
10  /** called to init opengl */
11  int glhInit() {
12      if (!glfwInit()) {
13          return (0);
14      }
15
16      glfwSetErrorCallback(errorCallback);
17      return (1);
18  }
19
```

```
20 /** stop opengl */
21 void glhStop() {
22     if (_glh_context != NULL) {
23         glhDestroyContext(_glh_context);
24     }
25     glfwTerminate();
26 }
27
28 /** create a new glh context */
29 t_glh_context * glhCreateContext(void) {
30     t_glh_context * context = (t_glh_context*)malloc(sizeof(t_glh_context)
        );
31     if (context == NULL) {
32         return (NULL);
33     }
34     context->window = glhWindowCreate();
35     return (context);
36 }
37
38 GLuint glhGenTexture(void) {
39     GLuint txID = 0;
40     glGenTextures(1, &txID);
41     return (txID);
42 }
43
44 void glhDeleteTexture(GLuint txID) {
45     glDeleteTextures(1, &txID);
46 }
47
48 void glhDestroyContext(t_glh_context * context) {
49     if (_glh_context == context) {
50         _glh_context = NULL;
51     }
52
53     glhWindowDestroy(context->window);
54     context->window = NULL;
55     free(context);
56 }
57
58 /** set the opengl context to the given window */
59 void glhMakeContextCurrent(t_glh_context * context) {
60     // set current context
61     glfwMakeContextCurrent(context->window->pointer);
62
63     // singleton update
64     _glh_context = context;
65
66     //initialize glew if needed
67     #ifdef _WIN32
68         GLenum err = glewInit();
69         if (err != GLEW_OK) {
70             fprintf(stderr, "glew err: %s\n", glewGetErrorString(err));
71         }
72     #endif
```

```
73  }
74
75  /** get the last set context */
76  t_glh_context * glhGetContext() {
77      return (_glh_context);
78  }
79
80  t_glh_window * glhGetWindow() {
81      if (glhGetContext() == NULL) {
82          return (NULL);
83      }
84      return (glhGetContext()->window);
85  }
86
87  #ifndef GL_STACK_UNDERFLOW
88  # define GL_STACK_UNDERFLOW (0)
89  #endif
90
91  char * glhGetErrorString(int err) {
92      static char * str[] = { "GL_INVALID_ENUM", "GL_INVALID_VALUE", "
        GL_INVALID_OPERATION", "GL_STACK_OVERFLOW",
93              "GL_STACK_UNDERFLOW", "GL_OUT_OF_MEMORY" };
94      int errs[] = { GL_INVALID_ENUM, GL_INVALID_VALUE, GL_INVALID_OPERATION
        , GL_STACK_OVERFLOW,
95              GL_STACK_UNDERFLOW, GL_OUT_OF_MEMORY, };
96
97      if (err != GL_NO_ERROR) {
98          for (int i = 0; i < 6; i++) {
99              if (errs[i] == err) {
100                 return (str[i]);
101             }
102         }
103     }
104     return (NULL);
105 }
106
107 /** call it to check openGL error after a gl call */
108 void glhCheckError(char * label) {
109     int err = glGetError();
110
111     char * str = glhGetErrorString(err);
112     if (str == NULL) {
113         return;
114     }
115     printf("%s : GLH ERROR CHECK : %s\n", label, str);
116 }
117
118 /** window related functions */
119
120 /** create and return a new gl window */
121 t_glh_window * glhWindowCreate() {
122     static char * DEFAULT_WINDOW_TITLE = "Default Title";
123     static int DEFAULT_WINDOW_WIDTH = 1100;
124     static int DEFAULT_WINDOW_HEIGHT = 1100 / 1.6f;
```

```
125
126     #ifdef __APPLE__
127         glfwWindowHint (GLFW_CONTEXT_VERSION_MAJOR, 3);
128         glfwWindowHint (GLFW_CONTEXT_VERSION_MINOR, 2);
129         glfwWindowHint (GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
130         glfwWindowHint (GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
131     #endif
132
133     void * pointer = glfwCreateWindow(DEFAULT_WINDOW_WIDTH,
            DEFAULT_WINDOW_HEIGHT, DEFAULT_WINDOW_TITLE, NULL, NULL);
134     if (pointer == NULL) {
135         return (NULL);
136     }
137
138     t_glh_window * window = (t_glh_window *)malloc(sizeof(t_glh_window));
139     if (window == NULL) {
140         return (NULL);
141     }
142
143     window->pointer = pointer;
144     window->width = DEFAULT_WINDOW_WIDTH;
145     window->height = DEFAULT_WINDOW_HEIGHT;
146
147     return (window);
148 }
149
150 int glhWindowShouldClose(t_glh_window * window) {
151     return (glfwWindowShouldClose(window->pointer));
152 }
153
154 void glhWindowClose(t_glh_window * window) {
155     glfwSetWindowShouldClose(window->pointer, 1);
156 }
157
158 void glhViewPort(int x, int y, int width, int height) {
159     glViewport(x, y, width, height);
160 }
161
162 /** destroy a window */
163 void glhWindowDestroy(t_glh_window * window) {
164     glfwDestroyWindow(window->pointer);
165 }
166
167 /** set window title */
168 void glhWindowSetTitle(t_glh_window * window, char * title) {
169     glfwSetWindowTitle(window->pointer, title);
170 }
171
172 void glhWindowSetSize(t_glh_window * window, int width, int height) {
173     glfwSetWindowSize(window->pointer, width, height);
174 }
175
176 void glhWindowUpdate(t_glh_window * window) {
177     window->prev_mouseX = window->mouseX;
```

```
178     window->prev_mouseY = window->mouseY;
179     glfwGetCursorPos(window->pointer, &(window->mouseX), &(window->mouseY)
            );
180     glfwGetWindowSize(window->pointer, &(window->width), &(window->height)
            );
181     glfwPollEvents();
182 }
183
184 /** swap buffers */
185 void glhSwapBuffer(t_glh_window * window) {
186     glfwSwapBuffers(window->pointer);
187     window->frames_swapped++;
188 }
189
190 /** clear the buffers */
191 void glhClear(int bufferbits) {
192     glClear(bufferbits);
193 }
194
195 void glhClearColor(float r, float g, float b, float a) {
196     glClearColor(r, g, b, a);
197 }
198
199 /** program functions */
200
201 /** create a new program */
202 t_glh_program * glhProgramNew(void) {
203     t_glh_program * program = (t_glh_program*)malloc(sizeof(t_glh_program)
            );
204     if (program == NULL) {
205         return (NULL);
206     }
207     memset(program, 0, sizeof(t_glh_program));
208     return (program);
209 }
210
211 /** add a shader to the program */
212 int glhProgramAddShader(t_glh_program * program, GLuint shaderID, int
        shaderType) {
213     if (shaderType < 0 || shaderType >= 4) {
214         return (0);
215     }
216     program->shaders[shaderType] = shaderID;
217     return (1);
218 }
219
220 /** link the program */
221 void glhProgramLink(t_glh_program * program, void (*fbindAttributes)(
        t_glh_program *), void (*fLinkUniforms)(t_glh_program *)) {
222
223     //create a new program
224     program->id = glCreateProgram();
225
226     //for each shaders, attach it
```

```
227     int i;
228     for (i = 0 ; i < 4 ; i++) {
229         if (program->shaders[i]) {
230             glAttachShader(program->id, program->shaders[i]);
231         }
232     }
233 
234     //bind attributes to shaders
235     fbindAttributes(program);
236 
237     //link the program
238     glLinkProgram(program->id);
239 
240     {
241         char message[512];
242         int length;
243         glGetProgramInfoLog(program->id, sizeof(message), &length, message
                );
244         if (length > 0) {
245             printf("Linking shader message: %s\n", message);
246         }
247     }
248 
249     //valide program
250     glValidateProgram(program->id);
251 
252     //link uniforms
253     fLinkUniforms(program);
254 }
255 
256 /** delete a program */
257 void glhProgramDelete(t_glh_program * program) {
258     glDeleteProgram(program->id);
259     free(program);
260 }
261 
262 void glhProgramUse(t_glh_program * program) {
263     glUseProgram(program == NULL ? 0 : program->id);
264 }
265 
266 void glhProgramBindAttribute(t_glh_program * program, int attribute, char
       * name) {
267     glBindAttribLocation(program->id, attribute, name);
268 }
269 
270 void glhProgramLoadUniformInt(int location, int value) {
271     glUniform1i(location, value);
272 }
273 
274 void glhProgramLoadUniformFloat(int location, float value) {
275     glUniform1f(location, value);
276 }
277 
278 void glhProgramLoadUniformVec2f(int location, float x, float y) {
```

```
279     glUniform2f(location, x, y);
280 }
281
282 void glhProgramLoadUniformVec3f(int location, float x, float y, float z) {
283     glUniform3f(location, x, y, z);
284 }
285
286 void glhProgramLoadUniformVec4f(int location, float x, float y, float z,
        float w) {
287     glUniform4f(location, x, y, z, w);
288 }
289
290 void glhProgramLoadUniformMatrix4f(int location, float * mat4) {
291     glUniformMatrix4fv(location, 1, GL_FALSE, mat4);
292 }
293
294 int glhProgramGetUniform(t_glh_program * program, char * name) {
295     return (glGetUniformLocation(program->id, name));
296 }
297
298 // shaders
299 void glhShaderDelete(GLuint shaderID) {
300     glDeleteShader(shaderID);
301 }
302
303 GLuint glhShaderLoad(char * filepath, GLenum type) {
304
305     //read file to string
306     int fd = open(filepath, O_RDONLY);
307     if (fd < 0) {
308         return (-1);
309     }
310
311     static int buffsize = 4096;
312     char * source = (char*)malloc(sizeof(char) * buffsize);
313     if (source == NULL) {
314         return (-1);
315     }
316
317     int capacity = buffsize;
318     int length = 0;
319     int r;
320     while ((r = read(fd, source + length, buffsize)) > 0) {
321         length += r;
322         capacity += buffsize;
323         source = (char*)realloc(source, capacity);
324     }
325
326     if (length == capacity) {
327         source = (char*)realloc(source, length + 1);
328     }
329     source[length] = 0;
330
331     close(fd);
```

```
332
333     //create shader
334     GLuint shaderID = glCreateShader(type);
335     //add source
336     glShaderSource(shaderID, 1, (char const **)&source, &length);
337     //free the source file
338     free(source);
339
340     //compile
341     glCompileShader(shaderID);
342
343     //check compilation
344     GLint compiled = 0;
345     glGetShaderiv(shaderID, GL_COMPILE_STATUS, &compiled);
346     if (compiled == GL_FALSE) {
347         GLint length = 0;
348         glGetShaderiv(shaderID, GL_INFO_LOG_LENGTH, &length);
349
350         char buffer[length];
351         glGetShaderInfoLog(shaderID, length, &length, buffer);
352
353         glhShaderDelete(shaderID);
354
355         printf("shader compilation error: %s\n%s\n", filepath, buffer);
356         return (-1);
357     }
358     return (shaderID);
359 }
360
361
362 // vao and vbo bindings
363 GLuint glhVAOGen(void) {
364     GLuint dst;
365     glGenVertexArrays(1, &dst);
366     return (dst);
367 }
368
369 GLuint glhVBOGen(void) {
370     GLuint dst;
371     glGenBuffers(1, &dst);
372     return (dst);
373 }
374
375 void glhVAODelete(GLuint vao) {
376     glDeleteVertexArrays(1, &vao);
377 }
378
379 void glhVBODelete(GLuint vbo) {
380     glDeleteBuffers(1, &vbo);
381 }
382
383 void glhVBOData(GLenum target, GLsizeiptr size, const GLvoid * data,
        GLenum usage) {
384     glBufferData(target, size, data, usage);
```

```
}

void glhVAOBind(GLuint vao) {
    glBindVertexArray(vao);
}

void glhVAOUnbind(void) {
    glhVAOBind(0);
}

void glhVAOSetAttribute(GLuint attributeID, GLint length, GLenum type,
    GLboolean normalized, GLsizei stride, const GLvoid * offset) {
    glVertexAttribPointer(attributeID, length, type, normalized, stride,
        offset);
}

void glhVAOSetAttributeI(GLuint attributeID, GLint length, GLenum type,
    GLsizei stride, const GLvoid * offset) {
    glVertexAttribIPointer(attributeID, length, type, stride, offset);
}

void glhVAOEnableAttribute(GLuint id) {
    glEnableVertexAttribArray(id);
}

void glhVBOBind(GLuint target, GLuint vbo) {
    glBindBuffer(target, vbo);
}

void glhVBOUnbind(GLuint target) {
    glhVBOBind(target, 0);
}

void glhDraw(int dst, int begin, int vertex_count) {
    glDrawArrays(dst, begin, vertex_count);
}

void glhDrawElements(GLenum mode, GLsizei count, GLenum type, const GLvoid
    * indices) {
    glDrawElements(mode, count, type, indices);
}
```

srcs/heightmap.c

```
#include "renderer.h"


int heightmapGetHeight(t_image * image, float x, float y) {
    int px = (int)x;
    int py = image->h - (int)y;
    int idx = (py * image->w + px) * 3;
    if (idx >= image->w * image->h * 3) {
```

```
9           return (0);
10      }
11      unsigned char * rgb = (unsigned char*)(image + 1);
12      unsigned char b = rgb[idx + 0];
13      unsigned char g = rgb[idx + 1];
14      unsigned char r = rgb[idx + 2];
15      return (r + g + b);
16  }
```

srcs/image.c

```
1   #include "renderer.h"
2
3   # define BMP_HEADER_SIZE (54)
4
5   t_image * imageNew(char const * path) {
6
7       int fd = open(path, O_RDONLY);
8       if (fd == -1) {
9           return (NULL);
10      }
11
12      char header[BMP_HEADER_SIZE];
13      read(fd, &header, sizeof(header));
14
15      //magic
16      if (header[0] != 'B' || header[1] != 'M') {
17          close(fd);
18          return (NULL);
19      }
20
21      int offset = *((int*)(header + 0x0A));
22      int w = *((int*)(header + 0x12));
23      int h = *((int*)(header + 0x16));
24      t_image * image = (t_image*)malloc(sizeof(t_image) + 3 * w * h);
25      if (image == NULL) {
26          close(fd);
27          return (NULL);
28      }
29      image->w = w;
30      image->h = h;
31
32      //read useless bytes
33      lseek(fd, offset - BMP_HEADER_SIZE, SEEK_CUR);
34
35      //read raw bytes
36      read(fd, image + 1, w * h * 3);
37      close(fd);
38
39      return (image);
40  }
41
```

```c
void imageDelete(t_image * map) {
    free(map);
}
```

srcs/input.c

```c
#include "renderer.h"

static void inputKey(GLFWwindow * winptr, int key, int scancode, int
    action, int mods) {
    t_world * world = &(getEnv()->world);
    t_renderer * renderer = &(getEnv()->renderer);
    t_camera * camera = &(getEnv()->camera);

        //reset terrain
    if (key == GLFW_KEY_P && action == GLFW_PRESS) {

        int i;
        long long unsigned int seed = time(NULL);
        for (i = 0 ; i < WORLD_OCTAVES ; i++) {
            noiseNextInt(&seed);
            noiseSeed(world->octaves[i], seed);
        }

        HMAP_ITER_START(world->terrains, t_terrain *, terrain) {
            terrainGenerate(world, terrain);
        }
        HMAP_ITER_END(world->terrains, t_terrain *, terrain);
    }


    //triangles
    if (key == GLFW_KEY_F && action == GLFW_PRESS) {
        renderer->state ^= STATE_RENDER_TRIANGLES;
    }

    //fog
    if (key == GLFW_KEY_V && action == GLFW_PRESS) {
        renderer->state ^= STATE_APPLY_FOG;
    }

    //lighting
    if (key == GLFW_KEY_B && action == GLFW_PRESS) {
        renderer->state ^= STATE_APPLY_PHONG_LIGHTNING;
    }

    //lighting
    if (key == GLFW_KEY_M && action == GLFW_PRESS) {
        renderer->state ^= STATE_SPECULAR;
    }

    //culling
```

```
46      if (key == GLFW_KEY_C && action == GLFW_PRESS) {
47          renderer->state ^= STATE_LOCK_CULLING;
48      }
49
50      //culling
51      if (key == GLFW_KEY_X && action == GLFW_PRESS) {
52          renderer->state ^= STATE_CULLING;
53      }
54
55      //sun
56      if (key == GLFW_KEY_Z && action == GLFW_PRESS) {
57          renderer->sunray.x = camera->vview.x;
58          renderer->sunray.y = -camera->vview.y;
59          renderer->sunray.z = camera->vview.z;
60      }
61
62      //close
63      if (key == GLFW_KEY_ESCAPE) {
64          glfwSetWindowShouldClose(winptr, 1);
65      } else if (key == GLFW_KEY_N && action == GLFW_PRESS) {
66          int mod = glfwGetInputMode(winptr, GLFW_CURSOR) ==
                GLFW_CURSOR_DISABLED ? GLFW_CURSOR_NORMAL :
                GLFW_CURSOR_DISABLED;
67          glfwSetInputMode(winptr, GLFW_CURSOR, mod);
68      }
69
70  }
71
72  static void inputCursorPos(GLFWwindow * winptr, double xpos, double ypos)
        {
73  }
74
75  static void inputMouseButton(GLFWwindow * winptr, int button, int action,
        int mods) {
76  }
77
78  static void inputUpdateDebug(t_glh_context * context, t_world * world,
        t_renderer * renderer, t_camera * camera) {
79
80      static int padding = 20;
81      static char * str;
82
83      //debug
84      if (glfwGetKey(context->window->pointer, GLFW_KEY_H) != GLFW_PRESS) {
85          return ;
86      }
87
88      printf("\n");
89
90      printf("************************************************************\n");
91      printf("***************** DEBUG STARTS **************************\n");
92      printf("************************************************************\n");
93
94      long time;
```

```
95      MICROSEC(time);
96      printf("\nCurrent timestamp: %ld\n\n", time);
97
98      printf("\n");
99      printf("-----------------  GLH CONTEXT ------------------------\n");
100     printf("\n");
101
102     printf("%*s: %p\n", padding, "window pointer", context->window->
            pointer);
103     printf("%*s: %d\n", padding, "window width", context->window->width);
104     printf("%*s: %d\n", padding, "window height", context->window->height)
            ;
105     printf("%*s: %.4f\n", padding, "mouse X", context->window->mouseX);
106     printf("%*s: %.4f\n", padding, "mouse Y", context->window->mouseY);
107     printf("%*s: %.4f\n", padding, "prev mouse X", context->window->
            prev_mouseX);
108     printf("%*s: %.4f\n", padding, "prev mouse Y", context->window->
            prev_mouseY);
109     printf("%*s: %d\n", padding, "frames swapped", context->window->
            frames_swapped);
110
111
112     printf("\n");
113     printf("--------------------- WORLD ---------------------------\n");
114     printf("\n");
115     printf("%*s: %lu/%d\n", padding, "loaded terrains", world->terrains->
            size, TERRAIN_KEEP_LOADED_DISTANCE * TERRAIN_KEEP_LOADED_DISTANCE *
             4);
116     printf("\n");
117
118     printf("-------------------- RENDERER -------------------------\n");
119     printf("\n");
120     printf("%*s: %d\n", padding, "terrain program ID", renderer->program->
            id);
121     printf("%*s: %d\n", padding, "triangle drawn on last frame", renderer
            ->vertexCount / 3);
122     printf("\n");
123
124
125     printf("-------------------- CAMERA ---------------------------\n");
126     printf("\n");
127     printf("position: vec3f(x:%.2f ; y:%.2f ; z:%.2f)\n", camera->pos.x,
            camera->pos.y, camera->pos.z);
128     printf("index   : vec2i(x:%d ; y:%d)\n", camera->terrain_index.x,
            camera->terrain_index.y);
129     printf("rotation: vec3f(x:%.2f ; y:%.2f ; z:%.2f)\n", camera->rot.x,
            camera->rot.y, camera->rot.z);
130     printf("viewvec : vec3f(x:%.2f ; y:%.2f ; z:%.2f)\n", camera->vview.x,
             camera->vview.y, camera->vview.z);
131     printf("\n");
132
133     str = mat4f_str(&(camera->mview));
134     printf("view matrix:\n%s\n\n", str);
135     free(str);
```

```
136
137     str = mat4f_str(&(camera->mproj));
138     printf("projection matrix:\n%s\n\n", str);
139     free(str);
140
141     str = mat4f_str(&(camera->mviewproj));
142     printf("viewproj matrix:\n%s\n\n", str);
143     free(str);
144
145     printf("\n");
146
147     printf("*************************************************************\n");
148     printf("*****************  DEBUG ENDS  **************************\n");
149     printf("*************************************************************\n");
150 }
151
152 static void inputUpdateCamera(t_camera * camera) {
153
154     float movespeed = camera->movespeed;
155     static float rotspeed = 0.3f;
156
157     t_glh_window * win = glhGetWindow();
158
159     if (glfwGetInputMode(win->pointer, GLFW_CURSOR) == GLFW_CURSOR_NORMAL)
            {
160         return ;
161     }
162
163
164     //camera speed
165     if (glfwGetKey(win->pointer, GLFW_KEY_KP_ADD) == GLFW_PRESS) {
166         camera->movespeed *= 1.2f;
167     } else if (glfwGetKey(win->pointer, GLFW_KEY_KP_SUBTRACT) ==
           GLFW_PRESS) {
168         camera->movespeed *= 0.833f;
169     }
170
171     //rotation
172     camera->rot.pitch += ((win->mouseY - win->prev_mouseY) * rotspeed);
173     camera->rot.yaw += ((win->mouseX - win->prev_mouseX) * rotspeed);
174
175     //move
176     if (glfwGetKey(win->pointer, GLFW_KEY_W) == GLFW_PRESS) {
177         camera->pos.x += camera->vview.x * movespeed;
178         camera->pos.y += camera->vview.y * movespeed;
179         camera->pos.z += camera->vview.z * movespeed;
180     } else if (glfwGetKey(win->pointer, GLFW_KEY_S) == GLFW_PRESS) {
181         camera->pos.x += -camera->vview.x * movespeed;
182         camera->pos.y += -camera->vview.y * movespeed;
183         camera->pos.z += -camera->vview.z * movespeed;
184     }
185
186     if (glfwGetKey(win->pointer, GLFW_KEY_D) == GLFW_PRESS) {
187         camera->pos.x += -camera->vview.z * movespeed;
```

```
188         camera->pos.z += camera->vview.x * movespeed;
189     }
190     else if (glfwGetKey(win->pointer, GLFW_KEY_A) == GLFW_PRESS) {
191         camera->pos.x += camera->vview.z * movespeed;
192         camera->pos.z += -camera->vview.x * movespeed;
193     }
194 }
195
196 void inputUpdate(t_glh_context * context, t_world * world, t_renderer *
        renderer, t_camera * camera) {
197     inputUpdateCamera(camera);
198     inputUpdateDebug(context, world, renderer, camera);
199 }
200
201 void inputInit(t_glh_context * context) {
202     glfwSetInputMode(context->window->pointer, GLFW_CURSOR,
            GLFW_CURSOR_DISABLED);
203     glfwSetKeyCallback(context->window->pointer, inputKey);
204     glfwSetCursorPosCallback(context->window->pointer, inputCursorPos);
205     glfwSetMouseButtonCallback(context->window->pointer, inputMouseButton)
            ;
206 }
```

srcs/main.c

```
1  #include "renderer.h"
2
3  void printUsage(char * binary, FILE * dst) {
4      fprintf(dst, "usage: ./%s [FLAGS]\n", binary);
5      fprintf(dst, "flags available are:\n");
6      fprintf(dst, "\t-r {SEED} for random/infinite terrain generation\n");
7      fprintf(dst, "\t-f [FILE] for a bmp terrain heightmap loading\n");
8      fprintf(dst, "\t-h [MAX_HEIGHT] to define maximum height of meshes\n")
           ;
9      fprintf(dst, "examples:\n");
10     fprintf(dst, "\t./%s -r 42 -h 256\n", binary);
11     fprintf(dst, "\t./%s -t \"texture.bmp\" -h 12\n", binary);
12 }
13
14 t_env g_env;
15
16 t_env * getEnv(void) {
17     return (&(g_env));
18 }
19
20 static int threadLoop(void * args) {
21     t_env           * env       = getEnv();
22     t_glh_context   * context   = env->context;
23     t_renderer      * renderer  = &(env->renderer);
24     t_world         * world     = &(env->world);
25     t_camera        * camera    = &(env->camera);
26
```

```
27      printf("Thread loop started!\n");
28
29      while (env->is_running) {
30          //update the camera and the world
31          worldUpdate(context, world, renderer, camera);
32
33          //10 ups
34          usleep(50 * 1000);
35
36      }
37
38      printf("Thread loop stopped!\n");
39
40      return (0);
41  }
42
43  int main(int argc, char **argv) {
44
45      //get binary name
46      char * binary = argc == 0 ? "renderer" : argv[0];
47
48      //parse arguments
49      int optind;
50      char mode = 'r';
51      long seed = time(NULL);
52      float maxheight = 1.0f;
53      char * bmpfile = NULL;
54      for (optind = 1; optind < argc; optind++) {
55
56          if (argv[optind][0] != '-' || argv[optind][2] != 0) {
57               printUsage(binary, stderr); return (EXIT_FAILURE);
58          }
59
60          mode = argv[optind][1];
61
62          switch (argv[optind][1]) {
63              case 'r': if (optind + 1 < argc) { seed = atoi(argv[++optind])
                    ; } break;
64              case 'f': if (optind + 1 >= argc) { printUsage(binary, stderr)
                    ; return (EXIT_FAILURE); } else { bmpfile = strdup(argv[++
                    optind]); } break;
65              case 'h': if (optind + 1 >= argc) { printUsage(binary, stderr)
                    ; return (EXIT_FAILURE); } else { maxheight = atof(argv[++
                    optind]); } break;
66              default: printUsage(binary, stderr); return (EXIT_FAILURE);
67          }
68      }
69
70      if (argc <= 1) {
71          printUsage(binary, stdout);
72          printf("\n");
73      }
74      printf("{mode='%c'}, {seed='%ld'}, {maxheight='%f'}, {bmpfile='%s'}\n\
           n", mode, seed, maxheight, bmpfile);
```

```
75
76      printf("Initializing openGL...\n");
77
78      glhInit();
79
80      printf("Creating gl context...\n");
81      t_env * env = getEnv();
82
83      env->context = glhCreateContext();
84
85      t_glh_context * context = env->context;
86
87      if (context == NULL) {
88          fprintf(stderr, "Failed to create gl context.\n");
89          return (EXIT_FAILURE);
90      }
91
92      if (context->window == NULL) {
93          fprintf(stderr, "Failed to create gl window.\n");
94          return (EXIT_FAILURE);
95      }
96
97      printf("Making gl context current...\n");
98      glhMakeContextCurrent(context);
99
100     t_world * world         = &(env->world);
101     t_renderer * renderer   = &(env->renderer);
102     t_camera * camera       = &(env->camera);
103     thrd_t * thrd           = &(env->thrd);
104
105     printf("Initializing camera...\n");
106     cameraInit(camera);
107
108     printf("Initializing renderer...\n");
109     rendererInit(renderer);
110
111     printf("Initializing inputs...\n");
112     inputInit(context);
113
114     printf("Initializing world...\n");
115     worldInit(world, bmpfile, maxheight);
116
117     printf("Creating calculator thread...\n");
118     thrd_create(thrd, threadLoop, &env);
119
120     printf("Rendering started...\n");
121
122     long int total = 0;
123     long int count = 0;
124
125     env->is_running = 1;
126
127     long t1, t2;
128     while (!glhWindowShouldClose(context->window) && env->is_running) {
```

```
129
130         MICROSEC(t1);
131
132         //update the window
133         glhWindowUpdate(context->window);
134
135         //input
136         inputUpdate(context, world, renderer, camera);
137
138         //camera
139         cameraUpdate(context, world, renderer, camera);
140
141         //update the renderer
142         rendererUpdate(context, world, renderer, camera);
143
144         //render
145         rendererRender(context, world, renderer, camera);
146
147         MICROSEC(t2);
148         total += (t2 - t1);
149         count++;
150
151         //swap buffers
152         glhSwapBuffer(context->window);
153     }
154
155     env->is_running = 0;
156
157     //wait for calculator thread to finish
158     printf("Waiting for thread to finish...\n");
159     thrd_join(env->thrd, NULL);
160
161     printf("Loop ended\n");
162
163     printf("Deleting camera...\n");
164     cameraDelete(camera);
165
166     printf("Deleting world...\n");
167     worldDelete(world);
168
169     printf("Deleting renderer...\n");
170     rendererDelete(renderer);
171
172     printf("Destroying gl context..\n");
173     glhDestroyContext(context);
174
175     printf("Stopping openGL...\n");
176     glhStop();
177
178     printf("All done\n");
179
180     printf("Moyenne: %ld\n", total / 1000 / count);
181
182     return (0);
```

```
}
```

srcs/noise.c

```
#include "noise.h"

//SIMPLEX NOISE IMPLEMENTATION


/** the default permutation raw array */
static unsigned char default_permutation[] = {
    151, 160, 137, 91, 90, 15, 8, 99, 37, 240, 21, 10, 23, 131, 13, 201,
    95, 96, 53, 194, 233, 7, 225, 140, 36, 103, 30, 69, 142, 190, 6, 148,
    247, 120, 234, 75, 0, 26, 197, 62, 94, 252, 219, 203, 117, 35, 11, 32,
    57, 177, 33, 88, 237, 149, 56, 87, 174, 20, 125, 136, 171, 168, 8,
        175,
    74, 165, 71, 134, 139, 48, 27, 166, 77, 146, 158, 231, 83, 111, 229,
        122,
    60, 211, 133, 230, 220, 105, 92, 41, 55, 46, 245, 40, 244, 102, 143,
        54,
    65, 25, 63, 161, 1, 216, 80, 73, 209, 76, 132, 187, 208, 89, 18, 169,
    200, 196, 135, 130, 116, 188, 159, 86, 164, 100, 109, 198, 173, 186,
    3, 64, 52, 217, 226, 250, 124, 123, 5, 202, 38, 147, 118, 126, 255,
        82,
    85, 212, 207, 206, 59, 227, 47, 16, 58, 17, 182, 189, 28, 42, 223,
        183,
    170, 213, 119, 248, 152, 2, 44, 154, 163, 70, 221, 153, 101, 155, 167,
    43, 172, 9, 129, 22, 39, 253, 19, 98, 108, 110, 79, 113, 224, 232,
        178,
    185, 112, 104, 218, 246, 97, 228, 251, 34, 242, 193, 238, 210, 144,
        12, 191,
    179, 162, 241, 81, 51, 145, 235, 249, 14, 239, 107, 49, 192, 214, 31,
        181,
    199, 106, 157, 184, 84, 204, 176, 115, 121, 50, 45, 127, 4, 150, 254,
        138,
    236, 205, 93, 222, 114, 67, 29, 24, 72, 243, 141, 128, 195, 78, 66,
        215, 61, 156, 180
};

static int fastfloor(float x) {
    int xi = (int)x;
    return ((x < xi) ? (xi - 1) : xi);
}

t_noise * noiseNew(void) {
    t_noise * noise = (t_noise *)malloc(sizeof(t_noise));
    if (noise == NULL) {
        return (NULL);
    }
    noiseSeed(noise, time(NULL));
    return (noise);
}
```

```
39
40
41  //mersenne twister
42  unsigned int noiseNextInt(long long unsigned int * seed) {
43      *seed = 6364136223846793005ULL * *seed + 1;
44      unsigned int x = *seed >> 32;
45      x ^= x >> 11;
46      x ^= (x << 7) & 0x9D2C5680;
47      x ^= (x << 15) & 0xEFC60000;
48      x ^= x >> 18;
49      return (x);
50  }
51
52  void noiseSeed(t_noise * noise, long long unsigned int seed) {
53      noise->seed = seed;
54
55      //copy default permutation
56      memcpy(noise->p, default_permutation, sizeof(default_permutation));
57
58      int swap_from, swap_to;
59      unsigned char tmp;
60
61      //process permutations
62      int i;
63      for (i = 0 ; i < 512; i++) {
64
65          //generate next pemrutation
66          swap_from   = noiseNextInt(&seed) & 255; //mod 256
67          swap_to     = noiseNextInt(&seed) & 255; //mod 256
68
69          //do the swap
70          tmp                 = noise->p[swap_from];
71          noise->p[swap_from] = noise->p[swap_to];
72          noise->p[swap_to]   = tmp;
73      }
74  }
75
76  void noiseDelete(t_noise * noise) {
77      free(noise);
78  }
79
80  /*
81  static float grad2(int hash, float x, float y) {
82      switch(hash & 0x8) {
83          case 0x0: return  x + y;
84          case 0x1: return -x + y;
85          case 0x2: return  x - y;
86          case 0x3: return -x - y;
87          case 0x4: return  x;
88          case 0x5: return -x;
89          case 0x6: return  y;
90          case 0x7: return -y;
91          default: return 0;
92      }
```

```
}*/

static float grad2(int hash, float x, float y) {
    switch(hash & 0x4) {
        case 0x0: return  x + y;
        case 0x1: return -x + y;
        case 0x2: return  x - y;
        case 0x3: return -x - y;
        default: return 0;
    }
}

float snoise2(t_noise * noise, float x, float y) {
    static float F2 = 0.3660254037f;
    static float G2 = 0.2113248654f;
    static float TWO_G2 = 2.0f * 0.2113248654f;

    float n0, n1, n2;
    float s = (x + y) * F2;
    int i = fastfloor(x + s);
    int j = fastfloor(y + s);
    float t = (i + j) * G2;
    float X0 = i - t;
    float Y0 = j - t;
    float x0 = x - X0;
    float y0 = y - Y0;
    int i1 = (x0 > y0) ? 1 : 0;
    int j1 = (x0 > y0) ? 0 : 1;
    float x1 = x0 - i1 + G2;
    float y1 = y0 - j1 + G2;
    float x2 = x0 - 1.0 + TWO_G2;
    float y2 = y0 - 1.0 + TWO_G2;
    int ii = i & 255;
    int jj = j & 255;
    int gi0 = noise->p[ii + noise->p[jj]];
    int gi1 = noise->p[ii + i1 + noise->p[jj + j1]];
    int gi2 = noise->p[ii + 1 + noise->p[jj + 1]];

    float t0 = 0.5f - x0 * x0 - y0 * y0;
    if (t0 < 0) {
        n0 = 0.0f;
    } else {
      t0 *= t0;
      n0 = t0 * t0 * grad2(gi0, x0, y0);
    }

    float t1 = 0.5f - x1 * x1 - y1 * y1;
    if (t1 < 0) {
        n1 = 0.0f;
    } else {
        t1 *= t1;
        n1 = t1 * t1 * grad2(gi1, x1, y1);
    }

```

```
147     float t2 = 0.5f - x2 * x2 - y2 * y2;
148     if (t2 < 0) {
149         n2 = 0.0f;
150     } else {
151         t2 *= t2;
152         n2 = t2 * t2 * grad2(gi2, x2, y2);
153     }
154 
155     return (70.0 * (n0 + n1 + n2));
156 }
157 
158 static float grad3(int hash, float x, float y, float z) {
159     switch(hash & 0xF) {
160         case 0x0: return  x + y;
161         case 0x1: return -x + y;
162         case 0x2: return  x - y;
163         case 0x3: return -x - y;
164         case 0x4: return  x + z;
165         case 0x5: return -x + z;
166         case 0x6: return  x - z;
167         case 0x7: return -x - z;
168         case 0x8: return  y + z;
169         case 0x9: return -y + z;
170         case 0xA: return  y - z;
171         case 0xB: return -y - z;
172         case 0xC: return  y + x;
173         case 0xD: return -y + z;
174         case 0xE: return  y - x;
175         case 0xF: return -y - z;
176         default: return 0;
177     }
178 }
179 
180 
181 float snoise3(t_noise * noise, float x, float y, float z) {
182     static float F3 = 1.0f / 3.0f;
183     static float G3 = 1.0f / 6.0f;
184 
185     float n0, n1, n2, n3;
186 
187     float s = (x + y + z) * F3;
188     int i = fastfloor(x + s);
189     int j = fastfloor(y + s);
190     int k = fastfloor(z + s);
191 
192     float t = ( i + j + k ) * G3;
193 
194     float X0 = i - t;
195     float Y0 = j - t;
196     float Z0 = k - t;
197 
198     float x0 = x - X0;
199     float y0 = y - Y0;
200     float z0 = z - Z0;
```

```
201
202     int i1, j1, k1;
203     int i2, j2, k2;
204     if (x0 >= y0) {
205         if (y0 >= z0) {
206             i1 = 1;
207             j1 = 0;
208             k1 = 0;
209             i2 = 1;
210             j2 = 1;
211             k2 = 0;
212         } else if (x0 >= z0) {
213             i1 = 1;
214             j1 = 0;
215             k1 = 0;
216             i2 = 1;
217             j2 = 0;
218             k2 = 1;
219         } else {
220             i1 = 0;
221             j1 = 0;
222             k1 = 1;
223             i2 = 1;
224             j2 = 0;
225             k2 = 1;
226         }
227     } else {
228         if (y0 < z0) {
229             i1 = 0;
230             j1 = 0;
231             k1 = 1;
232             i2 = 0;
233             j2 = 1;
234             k2 = 1;
235         } else if (x0 < z0) {
236             i1 = 0;
237             j1 = 1;
238             k1 = 0;
239             i2 = 0;
240             j2 = 1;
241             k2 = 1;
242         } else {
243             i1 = 0;
244             j1 = 1;
245             k1 = 0;
246             i2 = 1;
247             j2 = 1;
248             k2 = 0;
249         }
250     }
251
252     float x1 = x0 - i1 + G3;
253     float y1 = y0 - j1 + G3;
254     float z1 = z0 - k1 + G3;
```

```
255
256     float x2 = x0 - i2 + 2.0f * G3;
257     float y2 = y0 - j2 + 2.0f * G3;
258     float z2 = z0 - k2 + 2.0f * G3;
259
260     float x3 = x0 - 1.0f + 3.0f * G3;
261     float y3 = y0 - 1.0f + 3.0f * G3;
262     float z3 = z0 - 1.0f + 3.0f * G3;
263
264     int ii = i & 255;
265     int jj = j & 255;
266     int kk = k & 255;
267
268     int gi0 = noise->p[ii +  0 + noise->p[jj +  0 + noise->p[kk +  0]]];
269     int gi1 = noise->p[ii + i1 + noise->p[jj + j1 + noise->p[kk + k1]]];
270     int gi2 = noise->p[ii + i2 + noise->p[jj + j2 + noise->p[kk + k2]]];
271     int gi3 = noise->p[ii +  1 + noise->p[jj +  1 + noise->p[kk +  1]]];
272
273     float t0 = 0.6f - x0 * x0 - y0 * y0 - z0 * z0;
274     if (t0 < 0) {
275         n0 = 0.0f;
276     } else {
277         t0 *= t0;
278         n0 = t0 * t0 * grad3(gi0, x0, y0, z0);
279     }
280
281     float t1 = 0.6f - x1 * x1 - y1 * y1 - z1 * z1;
282     if (t1 < 0) {
283         n1 = 0.0f;
284     } else {
285         t1 *= t1;
286         n1 = t1 * t1 * grad3(gi1, x1, y1, z1);
287     }
288
289     float t2 = 0.6f - x2 * x2 - y2 * y2 - z2 * z2;
290     if( t2 < 0 ) {
291         n2 = 0.0f;
292     } else {
293         t2 *= t2;
294         n2 = t2 * t2 * grad3(gi2, x2, y2, z2);
295     }
296
297     float t3 = 0.6f - x3 * x3 - y3 * y3 - z3 * z3;
298     if (t3 < 0) {
299         n3 = 0.0f;
300     } else {
301         t3 *= t3;
302         n3 = t3 * t3 * grad3(gi3, x3, y3, z3);
303     }
304
305     return (32.0f * (n0 + n1 + n2 + n3));
306 }
307
308
```

```
309 //PERLIN NOISE IMPLEMENTATION
310 static int phash(t_noise * noise, int x, int y) {
311     return (noise->p[(noise->p[y % 256] + x) % 256]);
312 }
313
314 static float lin_inter(float x, float y, float s) {
315     return (x + s * (y - x));
316 }
317
318 static float smooth_inter(float x, float y, float s) {
319     return (lin_inter(x, y, s * s * (3-2*s)));
320 }
321
322 float pnoise2(t_noise * noise, float x, float y) {
323     static float SCALE = 1 / 256.0f;
324
325     int x_i = (int)x;
326     int y_i = (int)y;
327     float x_dec = x - x_i;
328     float y_dec = y - y_i;
329     int s = phash(noise, x_i, y_i);
330     int t = phash(noise, x_i + 1, y_i);
331     int u = phash(noise, x_i, y_i + 1);
332     int v = phash(noise, x_i +  1, y_i + 1);
333     float low = smooth_inter(s, t, x_dec);
334     float high = smooth_inter(u, v, x_dec);
335     float n = smooth_inter(low, high, y_dec);
336     return (n * SCALE * 2.0f - 1);
337 }
```

srcs/renderer.c

```
1  #include "renderer.h"
2
3  GLuint u_mvp_matrix;
4  GLuint u_transf_matrix;
5  GLuint u_sky_color;
6  GLuint u_state;
7  GLuint u_time;
8  GLuint u_sunpos;
9
10 static void rendererBindAttributes(t_glh_program * program) {
11     glhProgramBindAttribute(program, 0, "pos");
12     glhProgramBindAttribute(program, 1, "uv");
13     glhProgramBindAttribute(program, 2, "height");
14     glhProgramBindAttribute(program, 3, "normal");
15     glhProgramBindAttribute(program, 4, "textureID");
16 }
17
18 static void rendererLinkUniforms(t_glh_program * program) {
19     u_mvp_matrix = glhProgramGetUniform(program, "mvp_matrix");
20     u_transf_matrix = glhProgramGetUniform(program, "transf_matrix");
```

```
21      u_sky_color = glhProgramGetUniform(program, "sky_color");
22      u_state = glhProgramGetUniform(program, "state");
23      u_time = glhProgramGetUniform(program, "time");
24      u_sunpos = glhProgramGetUniform(program, "sunray");
25  }
26
27  static void rendererGenerateBufferIndices(t_renderer * renderer) {
28
29      long size = sizeof(unsigned short) * (TERRAIN_DETAIL - 1) * (
            TERRAIN_DETAIL - 1) * 6;
30      unsigned short * indices = (unsigned short *) malloc(size);
31
32      int x, z;
33      int i00, i01, i11, i10;
34      int i = 0;
35      for (x = 0 ; x < TERRAIN_DETAIL - 1; x++) {
36          for (z = 0 ; z < TERRAIN_DETAIL - 1; z++) {
37
38              i00 = x * TERRAIN_DETAIL + z;
39              i01 = i00 + 1;
40              i10 = (x + 1) * TERRAIN_DETAIL + z;
41              i11 = i10 + 1;
42              indices[i++] = i00;
43              indices[i++] = i11;
44              indices[i++] = i10;
45              indices[i++] = i00;
46              indices[i++] = i01;
47              indices[i++] = i11;
48          }
49      }
50
51      renderer->terrain_indices = glhVBOGen();
52      glhVBOBind(GL_ELEMENT_ARRAY_BUFFER, renderer->terrain_indices);
53      glhVBOData(GL_ELEMENT_ARRAY_BUFFER, size, indices, GL_STATIC_DRAW);
54      glhVBOUnbind(GL_ELEMENT_ARRAY_BUFFER);
55
56      free(indices);
57  }
58
59  static void rendererGenerateBufferVertices(t_renderer * renderer) {
60
61      long size = sizeof(float) * 4 * TERRAIN_DETAIL * TERRAIN_DETAIL;
62      float * vertices = (float *) malloc(size);
63
64      float unit = 1 / (float)(TERRAIN_DETAIL - 1);
65      int x, z;
66      int i = 0;
67      for (x = 0 ; x < TERRAIN_DETAIL ; x++) {
68          for (z = 0 ; z < TERRAIN_DETAIL ; z++) {
69              vertices[i++] = x * unit;
70              vertices[i++] = z * unit;
71              vertices[i++] = x % 2 == 0 ? 0.0f : 1.0f;
72              vertices[i++] = z % 2 == 0 ? 0.0f : 1.0f;
73          }
```

```
74      }
75
76      renderer->terrain_vertices = glhVBOGen();
77      glhVBOBind(GL_ARRAY_BUFFER, renderer->terrain_vertices);
78      glhVBOData(GL_ARRAY_BUFFER, size, vertices, GL_STATIC_DRAW);
79      glhVBOUnbind(GL_ARRAY_BUFFER);
80
81      free(vertices);
82  }
83
84  static void rendererGenerateBuffers(t_renderer * renderer) {
85      rendererGenerateBufferIndices(renderer);
86      rendererGenerateBufferVertices(renderer);
87  }
88
89  void rendererInit(t_renderer * renderer) {
90
91      //init math lib
92      cmaths_init();
93
94      //create the program
95      renderer->program = glhProgramNew();
96
97      renderer->state = 0;
98
99      //load shaders
100     GLuint fs = glhShaderLoad("./shaders/model.fs", GL_FRAGMENT_SHADER);
101     GLuint vs = glhShaderLoad("./shaders/model.vs", GL_VERTEX_SHADER);
102     glhProgramAddShader(renderer->program, fs, GLH_SHADER_FRAGMENT);
103     glhProgramAddShader(renderer->program, vs, GLH_SHADER_VERTEX);
104
105     //link
106     glhProgramLink(renderer->program, rendererBindAttributes,
            rendererLinkUniforms);
107
108     //generate terrain indices
109     rendererGenerateBuffers(renderer);
110
111     //initialize lists
112     renderer->render_list = array_list_new(256, sizeof(t_terrain *));
113     renderer->delete_list = array_list_new(256, sizeof(t_terrain *));
114
115     //image
116     renderer->texture.txID = glhGenTexture();
117     renderer->texture.image = imageNew("./res/textures.bmp");
118     unsigned char * pixels = (unsigned char*)(renderer->texture.image + 1)
            ;
119     glBindTexture(GL_TEXTURE_2D, renderer->texture.txID);
120     printf("txID: %u w : %d h : %d\n", renderer->texture.txID, renderer->
            texture.image->w, renderer->texture.image->h);
121     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, renderer->texture.image->w,
            renderer->texture.image->h, 0, GL_BGR, GL_UNSIGNED_BYTE, pixels);
122
123     glGenerateMipmap(GL_TEXTURE_2D);
```

```
124     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
            GL_LINEAR_MIPMAP_LINEAR);
125     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_LOD_BIAS, -1.6f);
126
127     //enable depth test
128     glEnable(GL_DEPTH_TEST);
129     glEnable(GL_CULL_FACE);
130     glCullFace(GL_BACK);
131
132     //set default sun ray
133     vec3f_set(&(renderer->sunray), 1.0f, 1.0f, 1.0f);
134     vec3f_normalize(&(renderer->sunray), &(renderer->sunray));
135
136     //line width
137     glLineWidth(2.0f);
138
139     //swap interval for 60 fps max
140     glfwSwapInterval(1);
141
142     glhCheckError("post rendererInit()");
143 }
144
145 static void rendererInitTerrain(t_renderer * renderer, t_terrain * terrain
        ) {
146
147     terrain->initialized = 1;
148
149     //allocate terrain model on GPU
150     terrain->vao = glhVAOGen();
151     terrain->vbo = glhVBOGen();
152
153     //bind vao
154     glhVAOBind(terrain->vao);
155
156     //bind indices
157     glhVBOBind(GL_ELEMENT_ARRAY_BUFFER, renderer->terrain_indices);
158
159     //bind static grid
160     glhVBOBind(GL_ARRAY_BUFFER, renderer->terrain_vertices);
161     glhVAOSetAttribute(0, 2, GL_FLOAT, 0, 4 * sizeof(float), NULL); //
            default vertices pos
162     glhVAOSetAttribute(1, 2, GL_FLOAT, 0, 4 * sizeof(float), (void*)(2 *
            sizeof(float))); //default vertices uv
163     glhVBOUnbind(GL_ARRAY_BUFFER);
164     glhVAOEnableAttribute(0);
165     glhVAOEnableAttribute(1);
166
167     //bind buffer
168     glhVBOBind(GL_ARRAY_BUFFER, terrain->vbo);
169     //set attruibutes
170     glhVAOSetAttribute(2, 1, GL_FLOAT, 0, TERRAIN_VERTEX_SIZE, NULL); //
            height
171     glhVAOSetAttribute(3, 2, GL_FLOAT, 0, TERRAIN_VERTEX_SIZE, (void*)(1 *
             sizeof(float))); //normal
```

```
172     glhVAOSetAttributeI(4, 3, GL_INT, TERRAIN_VERTEX_SIZE, (void*)((2 + 1)
            * sizeof(float))); //texture ID
173     glhVBOUnbind(GL_ARRAY_BUFFER);
174     //enable attributes
175     glhVAOEnableAttribute(2);
176     glhVAOEnableAttribute(3);
177     glhVAOEnableAttribute(4);
178 
179     //unbind vao
180     glhVAOUnbind();
181 }
182 
183 void rendererUpdate(t_glh_context * context, t_world * world, t_renderer *
        renderer, t_camera * camera) {
184     //if rendering list is locked
185     if (renderer->state & STATE_LOCK_CULLING) {
186         return ;
187     }
188 
189     //clear lists
190     array_list_clear(renderer->render_list);
191     array_list_clear(renderer->delete_list);
192 
193     //update lists
194     HMAP_ITER_START(world->terrains, t_terrain *, terrain) {
195 
196         t_vec3f diff;
197         diff.x = terrain->index.x - camera->terrain_index.x;
198         diff.y = 0;
199         diff.z = terrain->index.y - camera->terrain_index.y;
200 
201             //if to far, delete this terrain
202         if (diff.x <= -TERRAIN_KEEP_LOADED_DISTANCE || diff.z <= -
                TERRAIN_KEEP_LOADED_DISTANCE ||
203             diff.x >= TERRAIN_KEEP_LOADED_DISTANCE || diff.z >=
                    TERRAIN_KEEP_LOADED_DISTANCE) {
204             array_list_add(renderer->delete_list, &terrain);
205         } else {
206 
207             float distance = vec3f_length(&diff);
208             if (distance < TERRAIN_RENDER_DISTANCE) {
209                 float normalizer = 1 / distance;
210                 diff.x *= normalizer;
211                 diff.z *= normalizer;
212 
213                 float dot = vec3f_dot_product(&(camera->vview), &diff);
214                 if (distance <= 2 || acos_f(dot) < camera->fov || renderer
                        ->state & STATE_CULLING) {
215                     array_list_add(renderer->render_list, &terrain);
216                 }
217             }
218         }
219     }
220     HMAP_ITER_END(world->terrains, t_terrain *, terrain);
```

```
221
222     //remove terrains
223     ARRAY_LIST_ITER_START(renderer->delete_list, t_terrain **, terrain_ptr
            , i) {
224         t_terrain * terrain = *terrain_ptr;
225         hmap_remove_key(world->terrains, &(terrain->index));
226         terrainDelete(terrain);
227     }
228     ARRAY_LIST_ITER_END(renderer->delete_list, t_terrain **, terrain_ptr,
            i);
229
230     array_list_clear(renderer->delete_list);
231 }
232
233 static int rendererRenderTerrain(t_renderer * renderer, t_terrain *
        terrain) {
234     static int vertexCount = (TERRAIN_DETAIL - 1) * (TERRAIN_DETAIL - 1) *
            6;
235
236     //if it vertices arent up to date
237     if (terrain->vertices != NULL) {
238         //update them
239         glhVBOBind(GL_ARRAY_BUFFER, terrain->vbo);
240         glhVBOData(GL_ARRAY_BUFFER, TERRAIN_DETAIL * TERRAIN_DETAIL *
                TERRAIN_VERTEX_SIZE, terrain->vertices, GL_STATIC_DRAW);
241         //release data
242         free(terrain->vertices);
243         terrain->vertices = NULL;
244         glhVBOUnbind(GL_ARRAY_BUFFER);
245     }
246
247     //load the matrix as a uniform variable
248     glhProgramLoadUniformMatrix4f(u_transf_matrix, (float*)(&(terrain->mat
            )));
249
250     //sun light
251     glhProgramLoadUniformVec3f(u_sunpos, renderer->sunray.x, renderer->
            sunray.y, renderer->sunray.z);
252
253     //bind the model
254     glhVAOBind(terrain->vao);
255
256     //draw it
257     glhDrawElements(GL_TRIANGLES, vertexCount, GL_UNSIGNED_SHORT, NULL);
258
259     return (vertexCount);
260 }
261
262 static void rendererPrepareProgram(t_glh_context * context, t_world *
        world, t_renderer * renderer, t_camera * camera) {
263     //set the texture
264     glActiveTexture(GL_TEXTURE0);
265     glBindTexture(GL_TEXTURE_2D, renderer->texture.txID);
266
```

```
267     //bind the program
268     glhProgramUse(renderer->program);
269 
270     //load uniforms
271     glhProgramLoadUniformMatrix4f(u_mvp_matrix, (float*)&(camera->
            mviewproj));
272 
273     //weather
274     glhProgramLoadUniformVec3f(u_sky_color, 0.46f, 0.70f, 0.99f);
275 
276     //load state
277     glhProgramLoadUniformInt(u_state, renderer->state);
278     glhProgramLoadUniformInt(u_time, world->time);
279 
280     //debug
281     if (renderer->state & STATE_RENDER_TRIANGLES) {
282         glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
283     } else {
284         glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
285     }
286 }
287 
288 void rendererRender(t_glh_context * context, t_world * world, t_renderer *
        renderer, t_camera * camera) {
289 
290     //viewport
291     glhViewPort(0, 0, context->window->width, context->window->height);
292 
293     //clear color buffer
294     glhClearColor(0.46f, 0.70f, 0.99f, 1.0f);
295     glhClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
296 
297     //prepare the renderer (bind program, textures and uniforms)
298     rendererPrepareProgram(context, world, renderer, camera);
299 
300     //total vertex drawn
301     renderer->vertexCount = 0;
302 
303     //new terrains counter
304     int newCount = 0;
305 
306     //for every terrain which has to be rendered
307     ARRAY_LIST_ITER_START(renderer->render_list, t_terrain **, terrain_ptr
            , i) {
308 
309         //get the terrain
310         t_terrain * terrain = *terrain_ptr;
311 
312         //if it is not initialized
313         if (!terrain->initialized && ++newCount <
                MAX_NEW_TERRAINS_PER_FRAME) {
314             //initialize it
315             rendererInitTerrain(renderer, terrain);
316         }
```

```
317
318         if (terrain->initialized) {
319             renderer->vertexCount += rendererRenderTerrain(renderer,
                    terrain);
320         }
321     }
322     ARRAY_LIST_ITER_END(renderer->render_list, t_terrain *, terrain, i);
323
324     glhVAOUnbind();
325     glhProgramUse(NULL);
326 }
327
328
329 void rendererDelete(t_renderer * renderer) {
330     cmaths_deinit();
331     glhProgramDelete(renderer->program);
332     glhVBODelete(renderer->terrain_indices);
333     glhVBODelete(renderer->terrain_vertices);
334     array_list_delete(renderer->render_list);
335     free(renderer->render_list);
336     imageDelete(renderer->texture.image);
337     glhDeleteTexture(renderer->texture.txID);
338 }
```

srcs/terrain.c

```
1  #include "renderer.h"
2
3  static void terrainCalculateNormal(t_world * world, t_biom * biom, float *
       nx, float * nz,
4                                     float wx, float wy, float wz) {
5      float dx = biom->heightGenStep;
6      float dz = biom->heightGenStep;
7      *nx = (biom->heightGen(world, biom, wx + dx, wz) - wy) / dx;
8      *nz = (biom->heightGen(world, biom, wx, wz + dz) - wy) / dz;
9  }
10
11 static void terrainGenerateVertices(t_world * world, float vertices[], int
       gridX, int gridY) {
12     float nx, nz;
13     int i = 0;
14     for (int x = 0 ; x < TERRAIN_DETAIL ; x++) {
15         for (int y = 0 ; y < TERRAIN_DETAIL; y++) {
16
17             float wx = (gridX * (TERRAIN_DETAIL - 1) + x) * TERRAIN_UNIT;
18             float wz = (gridY * (TERRAIN_DETAIL - 1) + y) * TERRAIN_UNIT;
19             t_biom * biom = worldGetBiomAt(world, wx, wz);
20             float wy = biom->heightGen(world, biom, wx, wz);
21             int textureID = biom->colorGen(world, biom, wx, wy, wz);
22
23             terrainCalculateNormal(world, biom, &nx, &nz, wx, wy, wz);
24
```

```
25              *(vertices + i++) = wy;
26              *(vertices + i++) = nx;
27              *(vertices + i++) = nz;
28              *((int*)(vertices + i++)) = textureID;
29          }
30      }
31  }
32
33  void terrainGenerate(t_world * world, t_terrain * terrain) {
34      int gridX = terrain->index.x;
35      int gridY = terrain->index.y;
36      terrain->vertices = (float *) malloc(TERRAIN_DETAIL * TERRAIN_DETAIL *
            TERRAIN_VERTEX_SIZE);
37      terrainGenerateVertices(world, terrain->vertices, gridX, gridY);
38  }
39
40  /** delete the given terrain */
41  void terrainDelete(t_terrain * terrain) {
42      if (terrain->initialized) {
43          terrain->initialized = 0;
44          glhVAODelete(terrain->vao);
45          glhVBODelete(terrain->vbo);
46      }
47
48      if (terrain->vertices != NULL) {
49          free(terrain->vertices);
50          terrain->vertices = NULL;
51      }
52      free(terrain);
53  }
54
55  /** allocate a new terrain on heap + gpu */
56  t_terrain * terrainNew(t_world * world, int gridX, int gridY) {
57
58      //allocate the terrain
59      t_terrain * terrain = (t_terrain*)malloc(sizeof(t_terrain));
60      if (terrain == NULL) {
61          return (NULL);
62      }
63
64      terrain->index.x = gridX;
65      terrain->index.y = gridY;
66      terrain->vertices = NULL;
67      terrain->initialized = 0;
68
69      //generate the transformation matrix for this terrain
70      mat4f_identity(&(terrain->mat));
71      mat4f_translate(&(terrain->mat), &(terrain->mat), terrain->index.x *
            TERRAIN_SIZE * 1.1f, 0, terrain->index.y * TERRAIN_SIZE * 1.1f);
72      mat4f_scale(&(terrain->mat), &(terrain->mat), TERRAIN_SIZE);
73
74      terrainGenerate(world, terrain);
75
76      return (terrain);
```

```
77 }
```

srcs/world.c

```
1  #include "renderer.h"
2
3  static void worldLoadHeightmap(t_world * world, char * file) {
4      if (file == NULL) {
5          world->heightmap = NULL;
6      } else {
7          world->heightmap = imageNew(file);
8      }
9  }
10
11 static int world_vec2i_hash(t_vec2i * vec) {
12     return (vec->x * (TERRAIN_KEEP_LOADED_DISTANCE * 2 + 1) + vec->y);
13 }
14
15 void worldInit(t_world * world, char * bmpfile, float max_height) {
16     glhCheckError("pre worldInit()");
17
18     world->time = 0;
19     world->max_height = max_height;
20     worldLoadHeightmap(world, bmpfile);
21     world->bioms = array_list_new(16, sizeof(t_biom));
22     biomsInit(world);
23
24     //create the terrain hash map
25     world->terrains = hmap_new(TERRAIN_KEEP_LOADED_DISTANCE * 4, (t_hf)
           world_vec2i_hash, (t_cmpf)vec2i_nequals, (t_f)NULL, (t_f)NULL);
26     if (world->terrains == NULL) {
27         fprintf(stderr, "world.c : 1.10 : worldInit() : not enough memory\
               n");
28         return ;
29     }
30
31     //noise creation
32     long long unsigned int seed = time(NULL);
33     int i;
34     for (i = 0 ; i < WORLD_OCTAVES ; i++) {
35         world->octaves[i] = noiseNew();
36         noiseNextInt(&seed);
37         noiseSeed(world->octaves[i], seed);
38     }
39
40 /*
41     terrain = terrainNew(0, 0);
42     terrainGenerate(world, terrain);
43     worldSpawnTerrain(world, terrain);
44 */
45
46     glhCheckError("post worldInit()");
```

```
47  }
48
49  void worldDelete(t_world * world) {
50      if (world->heightmap != NULL) {
51          imageDelete(world->heightmap);
52      }
53
54      HMAP_ITER_START(world->terrains, t_terrain *, terrain) {
55          terrainDelete(terrain);
56      }
57      HMAP_ITER_END(world->terrains, t_terrain *, terrain);
58
59      hmap_delete(world->terrains);
60      free(world->terrains);
61
62      int i;
63      for (i = 0 ; i < WORLD_OCTAVES ; i++) {
64          noiseDelete(world->octaves[i]);
65      }
66
67      biomsDelete(world);
68  }
69
70  t_biom * worldGetBiomAt(t_world * world, float wx, float wz) {
71      float noise = (snoise2(world->octaves[0], wx * 0.002f, wz * 0.002f) +
            1.0f) * 0.5f;
72      int id = (int) (noise * world->bioms->size);
73      return (array_list_get(world->bioms, id));
74  }
75
76  void worldGetGridIndex(t_world * world, float worldX, float worldZ, int *
        gridX, int * gridY) {
77      *gridX = (int)worldX / TERRAIN_SIZE;
78      *gridY = (int)worldZ / TERRAIN_SIZE;
79
80      if (worldX < 0) {
81          *gridX -= 1;
82      }
83
84      if (worldZ < 0) {
85          *gridY -= 1;
86      }
87  }
88
89  t_terrain * worldGetTerrain(t_world * world, int gridX, int gridY) {
90      t_vec2i index;
91      index.x = gridX;
92      index.y = gridY;
93      return (hmap_get(world->terrains, &index));
94  }
95
96  static void worldLoadNewTerrains(t_world * world, t_camera * camera) {
97
```

```
98      int indexx = MAX(0, camera->terrain_index.x - TERRAIN_LOADED_DISTANCE)
        ;
99      int indexy = MAX(0, camera->terrain_index.y - TERRAIN_LOADED_DISTANCE)
        ;
100     int maxx = camera->terrain_index.x + TERRAIN_LOADED_DISTANCE;
101     int maxy = camera->terrain_index.y + TERRAIN_LOADED_DISTANCE;
102     int gridX, gridY;
103     for (gridX = indexx ; gridX  < maxx; gridX++) {
104         for (gridY = indexy ; gridY < maxy; gridY++) {
105
106             if (gridX < 0 || gridY < 0) {
107                 continue ;
108             }
109
110             //if this terrain isnt generated yet
111             if (worldGetTerrain(world, gridX, gridY) == NULL) {
112
113                 //can it be generated?
114                 float wx = gridX * TERRAIN_SIZE;
115                 float wz = gridY * TERRAIN_SIZE;
116                 t_biom * biom = worldGetBiomAt(world, wx, wz);
117                 if (!biom->canGenerateAt(world, biom, wx, wz)) {
118                     continue ;
119                 }
120
121                 //if so, generate it
122                 t_terrain * terrain = terrainNew(world, gridX, gridY);
123
124                 if (terrain != NULL) {
125                     hmap_insert(world->terrains, terrain, &(terrain->index
                        ));
126                 }
127             }
128         }
129     }
130 }
131
132 void worldUpdate(t_glh_context * context, t_world * world, t_renderer *
    renderer, t_camera * camera) {
133     //load new terrains
134     worldLoadNewTerrains(world, camera);
135     world->time++;
136 }
```

includes/glh.h

```
1  #ifndef GLH_H
2  # define GLH_H
3
4  # ifdef __APPLE__
5  #   define GLFW_INCLUDE_GLCOREARB
6  # endif
```

```
7
8  # if defined(_WIN32) |  defined(_WIN64)
9  #   include <GL/glew.h>
10 # endif
11
12 # include "GLFW/glfw3.h"
13 # include <stdio.h>
14 # include <stdlib.h>
15 # include <string.h>
16 # include <fcntl.h>
17 # include <unistd.h>
18
19 # define GLH_WINDOW_EVENT_SCROLL (0)
20 # define GLH_WINDOW_EVENT_MOUSE_CURSOR (1)
21 # define GLH_WINDOW_EVENT_MOUSE_BUTTON (2)
22 # define GLH_WINDOW_EVENT_RESIZE (3)
23 # define GLH_WINDOW_EVENT_FOCUS (4)
24
25 typedef struct  s_glh_window {
26     void            * pointer;
27     int             width;
28     int             height;
29     double          mouseX; //current mouse coordinate X
30     double          mouseY; //current mouse coordinate Y
31     double          prev_mouseX; //previous mouse coordinate X
32     double          prev_mouseY; //previous mouse coordinate Y
33     int             frames_swapped; //number of swap buffer calls (i.e,
          frame put on screen)
34 }               t_glh_window;
35
36 typedef struct  s_glh_context {
37     t_glh_window * window;
38 }               t_glh_context;
39
40 # define GLH_SHADER_VERTEX (0)
41 # define GLH_SHADER_GEOMETRY (1)
42 # define GLH_SHADER_FRAGMENT (2)
43 # define GLH_SHADER_COMPUTE (3)
44 # define GLH_SHADER_MAX_ID (4)
45
46 typedef struct  s_glh_program {
47     GLuint id;
48     GLuint shaders[GLH_SHADER_MAX_ID];
49 }               t_glh_program;
50
51 /** called to init opengl */
52 int glhInit();
53
54 /** stop opengl */
55 void glhStop();
56
57 /** create a new glh context */
58 t_glh_context * glhCreateContext(void);
59
```

```
60  void glhDestroyContext(t_glh_context * context);
61
62  /** set the opengl context to the given window */
63  void glhMakeContextCurrent(t_glh_context * context);
64
65      /** get the last set context */
66  t_glh_context * glhGetContext();
67  t_glh_window * glhGetWindow();
68  char * glhGetErrorString(int err);
69
70  /** call it to check openGL error after a gl call */
71  void glhCheckError(char * label);
72
73  /** window related functions */
74
75  /** create and return a new gl window */
76  t_glh_window * glhWindowCreate();
77  int glhWindowShouldClose(t_glh_window * window);
78  void glhWindowClose(t_glh_window * window);
79  void glhWindowDestroy(t_glh_window * window);
80  void glhWindowSetTitle(t_glh_window * window, char * title);
81  void glhWindowSetSize(t_glh_window * window, int width, int height);
82  void glhWindowUpdate(t_glh_window * window);
83  void glhViewPort(int x, int y, int width, int height);
84
85  /** swap buffers */
86  void glhSwapBuffer(t_glh_window * window);
87
88  /** clear the buffers */
89  void glhClear(int bufferbits);
90  void glhClearColor(float r, float g, float b, float a);
91
92  //textures
93  GLuint glhGenTexture(void);
94  void glhDeleteTexture(GLuint txID);
95
96  /** program functions */
97  t_glh_program * glhProgramNew(void);
98  int             glhProgramAddShader(t_glh_program * program, GLuint
        shaderID, int shaderType);
99  void            glhProgramLink(t_glh_program * program, void (*
        fbindAttributes)(t_glh_program *), void (*fLinkUniforms)(t_glh_program
        *));
100 void            glhProgramDelete(t_glh_program * program);
101 void            glhProgramUse(t_glh_program * program);
102 void            glhProgramBindAttribute(t_glh_program * program, int
        attribute, char * name);
103 void            glhProgramLoadUniformInt(int location, int value);
104 void            glhProgramLoadUniformFloat(int location, float value);
105 void            glhProgramLoadUniformVec2f(int location, float x, float y)
        ;
106 void            glhProgramLoadUniformVec3f(int location, float x, float y,
         float z);
```

```
107 void            glhProgramLoadUniformVec4f(int location, float x, float y,
        float z, float w);
108 void            glhProgramLoadUniformMatrix4f(int location, float * mat4);
109 int             glhProgramGetUniform(t_glh_program * program, char * name)
        ;
110
111 void    glhShaderDelete(GLuint shaderID);
112 GLuint  glhShaderLoad(char * filepath, GLenum type);
113
114 //vao / vbo
115 GLuint glhVAOGen(void);
116 GLuint glhVBOGen(void);
117
118 void glhVAODelete(GLuint vao);
119 void glhVBODelete(GLuint vbo);
120
121 void glhVAOBind(GLuint vao);
122 void glhVAOUnbind(void);
123
124 void glhVBOData(GLenum target, GLsizeiptr size, const GLvoid * data,
        GLenum usage);
125
126 void glhVAOSetAttribute(GLuint attributeID, GLint length, GLenum type,
        GLboolean normalized, GLsizei stride, const GLvoid * offset);
127 void glhVAOSetAttributeI(GLuint attributeID, GLint length, GLenum type,
        GLsizei stride, const GLvoid * offset);
128 void glhVAOEnableAttribute(GLuint id);
129
130 void glhVBOBind(GLuint target, GLuint vbo);
131 void glhVBOUnbind(GLenum target);
132
133 void glhDraw(int dst, int begin, int vertex_count);
134 void glhDrawElements(GLenum mode, GLsizei count, GLenum type, const GLvoid
        * indices);
135
136 #endif
```

includes/noise.h

```
1  #ifndef NOISE_H
2  # define NOISE_H
3
4  # include <stdlib.h>
5  # include <string.h>
6  # include <time.h>
7  # include <stdio.h>
8
9  /** noise */
10 typedef struct  s_noise {
11     long long unsigned int  seed;
12     unsigned char           p[512];
13 }               t_noise;
```

```
14
15 /** noise data structure */
16 t_noise *   noiseNew(void);
17 void        noiseSeed(t_noise * noise, long long unsigned int seed);
18 void        noiseDelete(t_noise * noise);
19
20 /** a simple function which gives a 'pseudo-random' integer from the
       passed one */
21 unsigned int noiseNextInt(long long unsigned int * seed);
22
23 /** simplex noise 2D */
24 float   snoise2(t_noise * noise, float x, float y);
25
26 /** simplex noise 3D */
27 float   snoise3(t_noise * noise, float x, float y, float z);
28
29 /** perlin noise 2D */
30 float   pnoise2(t_noise * noise, float x, float y);
31
32 #endif
```

includes/renderer.h

```
1  #ifndef RENDERER_H
2  # define RENDERER_H
3
4  # include "array_list.h"
5  # include "hmap.h"
6  # include "cmaths.h"
7  # include "vec.h"
8  # include "mat.h"
9  # include "glh.h"
10 # include "tinycthread.h"
11 # include "noise.h"
12 # include <string.h>
13 # include <fcntl.h>
14 # include <unistd.h>
15 # include <string.h>
16 # include <time.h>
17 # include <fcntl.h>
18
19 /** the camera data structures */
20 typedef struct  s_camera {
21     t_vec3f pos; //camera world position
22     t_vec3f rot; //camera rotation toward (x, y, z) axis
23     t_vec3f vview; //view vector (direction where we are currently lookin
          at)
24     float   fov; //field of view
25     float   near_distance; //near plane distance
26     float   far_distance; //far plane distance
27     float   movespeed; // move speed
28     t_mat4f mview; //view matrix
```

```
29      t_mat4f mproj; //projection matrix
30      t_mat4f mviewproj; //projection matrix times view matrix
31      t_vec2i terrain_index; //current world terrain index of the camera
32  }               t_camera;
33
34  //terrain detail (number of vertex per line)
35  # define TERRAIN_DETAIL (16)
36  //terrain width (and height)
37  # define TERRAIN_SIZE (16)
38  # define TERRAIN_UNIT (TERRAIN_SIZE / (float)TERRAIN_DETAIL)
39  // number of terrain to render in term of distance
40  # define TERRAIN_RENDER_DISTANCE (64)
41  //max number of terrains to be newly pushed to GPU per frames
42  # define MAX_NEW_TERRAINS_PER_FRAME (32)
43  // distance where terrain are kept loaded in memory
44  # define TERRAIN_LOADED_DISTANCE (TERRAIN_RENDER_DISTANCE)
45  // distance where terrain are kept loaded in memory
46  # define TERRAIN_KEEP_LOADED_DISTANCE (TERRAIN_LOADED_DISTANCE)
47  # define MAX_NUMBER_OF_TERRAIN_LOADED (TERRAIN_KEEP_LOADED_DISTANCE *
        TERRAIN_KEEP_LOADED_DISTANCE * 2 * 2)
48  // number of floats per vertex
49  # define TERRAIN_VERTEX_SIZE ((1 + 2) * sizeof(float) + 1 * sizeof(int))
50
51  # define STATE_APPLY_FOG (1)
52  # define STATE_APPLY_PHONG_LIGHTNING (2)
53  # define STATE_LOCK_CULLING (4)
54  # define STATE_RENDER_TRIANGLES (8)
55  # define STATE_CULLING (16)
56  # define STATE_SPECULAR (32)
57
58  # define TX_WATER (0)
59  # define TX_GRASS (1)
60  # define TX_DIRT (2)
61  # define TX_STONE (3)
62  # define TX_SNOW (4)
63  # define TX_MAX (5)
64
65  typedef struct  s_image {
66      int w, h;
67  }               t_image;
68
69  typedef struct  s_texture {
70      t_image * image;
71      GLuint  txID;
72  }               t_texture;
73
74  /** a terrain */
75  typedef struct  s_terrain {
76      t_vec2i index;
77      t_mat4f mat;
78      GLuint  vao;
79      GLuint  vbo;
80      float   * vertices;
81      int     initialized;
```

```
82  }               t_terrain;
83
84  # define WORLD_OCTAVES (10)
85
86  /** the world */
87  typedef struct  s_world {
88      t_hmap          * terrains;
89      t_noise         * octaves[WORLD_OCTAVES];
90      t_array_list    * bioms;
91      float           max_height;
92      t_image         * heightmap;
93      int             time;
94  }               t_world;
95
96  typedef struct  s_biom {
97      float   (*heightGen)(t_world *, struct s_biom *, float, float);
98      int     (*colorGen)(t_world *, struct s_biom *, float, float, float);
99      int     (*canGenerateAt)(t_world *, struct s_biom *, float, float);
100     float   heightGenStep;
101     int     octaves;
102     float   amplitude;
103     float   persistance;
104     float   frequency;
105     float   lacunarity;
106 }               t_biom;
107
108 /** the renderer part of the program */
109 typedef struct  s_renderer {
110     t_glh_program   * program; //the rendering GPU program
111     GLuint          terrain_indices; //terrain indices buffer
112     GLuint          terrain_vertices; //terrain vertices buffer (static
            grid)
113     t_array_list    * render_list; //the list of terrain to render
114     t_array_list    * delete_list; //the list of terrain to delete
115     int             state; //the state for rendering
116     t_texture       texture;
117     int             vertexCount; //number of vertices drawn on last frame
118     t_vec3f         sunray; //sun light vector
119 }               t_renderer;
120
121 typedef struct  s_env {
122     t_glh_context   * context;
123     t_world         world;
124     t_renderer      renderer;
125     t_camera        camera; //user camera
126     thrd_t          thrd;
127     int             is_running;
128 }               t_env;
129
130 extern t_env g_env;
131
132 //get env
133 t_env * getEnv(void);
134
```

```
135 //renderer related functions
136 void rendererInit(t_renderer * renderer);
137 void rendererDelete(t_renderer * renderer);
138 void rendererUpdate(t_glh_context * context, t_world * world, t_renderer *
        renderer, t_camera * camera);
139 void rendererRender(t_glh_context * context, t_world * world, t_renderer *
        renderer, t_camera * camera);
140
141 //world related functions
142 void        worldInit(t_world * world, char * bmpfile, float maxheight);
143 void        worldDelete(t_world * world);
144 void        worldUpdate(t_glh_context * context, t_world * world,
       t_renderer * renderer, t_camera * camera);
145 void        worldGetGridIndex(t_world * world, float worldX, float worldZ,
        int * gridX, int * gridY);
146 t_terrain * worldGetTerrain(t_world * world, int gridX, int gridY);
147 t_biom *    worldGetBiomAt(t_world * world, float wx, float wz);
148
149 /** bioms */
150 void    biomsInit(t_world * world);
151 void    biomsDelete(t_world * world);
152
153 //terrains
154 t_terrain * terrainNew(t_world * world, int gridX, int gridY);
155 void        terrainDelete(t_terrain * terrain);
156 int         terrainHash(t_terrain * terrain);
157 int         terrainCmp(t_terrain * left, t_terrain * right);
158 void        terrainLoadHeightMap(t_terrain * terrains, int * n, char const
        * bmpfile);
159 void        terrainGenerate(t_world * world, t_terrain * terrain);
160
161 //camera related functions
162 void cameraInit(t_camera * camera);
163 void cameraDelete(t_camera * camera);
164 void cameraUpdate(t_glh_context * context, t_world * world, t_renderer *
       renderer, t_camera * camera);
165
166 //heightmaps (bmp file)
167 t_image      *  imageNew(char const * path);
168 void            imageDelete(t_image * t_image);
169 int             heightmapGetHeight(t_image * image, float x, float y);
170
171 //inputs
172 void inputInit(t_glh_context * context);
173 void inputUpdate(t_glh_context * context, t_world * world, t_renderer *
       renderer, t_camera * camera);
174
175 #endif
```

maths/cmaths.c

```
1 #include "cmaths.h"
```

```
2
3  float * sin_table;
4
5  float atan_f(float x) {
6      float xabs = ABS(x);
7      return (0.78539816339f * x - x * (xabs - 1) * (0.2447f + 0.0663f *
           xabs));
8  }
9
10 float acos_f(float x) {
11     //lagrange interpolation:
12     return ((-0.69813170079773212f * x * x - 0.87266462599716477f) * x +
           1.5707963267948966f);
13 }
14
15 float asin_f(float x) {
16     return (-acos_f(x) + 1.5707963267948966f);
17 }
18
19 float tan_f(float x) {
20     return (sin_f(x) / cos_f(x));
21 }
22
23 float sin_f(float x) {
24     unsigned int index = (unsigned int)RAD_TO_DEG(ABS(x)) % 360;
25     return (index > 180 ? -sin_table[index - 180] : sin_table[index]);
26 }
27
28 float cos_f(float x) {
29     return (sin_f(x + 1.5707963267948966f));
30 }
31
32 float sqrt_f(float x) {
33     unsigned int i = *(unsigned int*) &x;
34     i  += 127 << 23;
35     i >>= 1;
36     return (*(float*) &i);
37 }
38
39 int cmaths_init(void) {
40
41     sin_table = (float *) malloc(sizeof(float) * 4 * 360);
42     if (sin_table == NULL) {
43         return (0);
44     }
45
46     int i, j;
47     for (i = 0, j = 0 ; i < 180; i++) {
48         sin_table[j++] = (float)sin((double)DEG_TO_RAD((float)i));
49     }
50
51     return (1);
52 }
53
```

```c
int cmaths_deinit(void) {
    free(sin_table);
    sin_table = NULL;
    return (1);
}
```

maths/cmaths.h

```c
/*
*  This file is part of https://github.com/toss-dev/C_maths
*
*  It is under a GNU GENERAL PUBLIC LICENSE
*
*  This library is still in development, so please, if you find any issue, let me know about it on github.com
*  PEREIRA Romain
*/

#ifndef CMATHS_H
# define CMATHS_H

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

# ifndef DEG_TO_RAD
#  define DEG_TO_RAD(X) (X * 0.01745329251f)
# endif

# ifndef RAD_TO_DEG
#  define RAD_TO_DEG(X) (X * 57.2957795131f)
# endif

# ifndef MAX
#  define MAX(X, Y) (X > Y ? X : Y)
# endif

# ifndef MIN
#  define MIN(X, Y) (X < Y ? X : Y)
# endif

# ifndef ABS
#  define ABS(X) (X < 0 ? -X : X)
# endif

int cmaths_init(void);
int cmaths_deinit(void);

float   acos_f(float x);
float   asin_f(float x);
float   atan_f(float x);
```

```
44
45 float    sin_f(float x);
46 float    cos_f(float x);
47 float    tan_f(float x);
48
49 float    sqrt_f(float x);
50
51 #endif
```

maths/mat.h

```
1  /**
2   *  This file is part of https://github.com/toss-dev/C_maths
3   *
4   *  It is under a GNU GENERAL PUBLIC LICENSE
5   *
6   *  This library is still in development, so please, if you find any issue
        , let me know about it on github.com
7   *  PEREIRA Romain
8   */
9
10 #ifndef MAT_H
11 # define MAT_H
12
13 # include "cmaths.h"
14 # include "mat4f.h"
15
16 #endif
```

maths/mat4f.c

```
1  /**
2   *  This file is part of https://github.com/toss-dev/C_maths
3   *
4   *  It is under a GNU GENERAL PUBLIC LICENSE
5   *
6   *  This library is still in development, so please, if you find any issue
        , let me know about it on github.com
7   *  PEREIRA Romain
8   */
9
10 #include "mat4f.h"
11
12 /** create a new matrix */
13 t_mat4f * mat4f_new(void) {
14     return ((t_mat4f*)malloc(sizeof(t_mat4f)));
15 }
16
17 /** delete the given matrix */
18 void mat4f_delete(t_mat4f * mat) {
```

```
19      free(mat);
20  }
21
22  /** copy */
23  t_mat4f * mat4f_copy(t_mat4f * dst, t_mat4f * src) {
24      if (dst == NULL) {
25          if ((dst = mat4f_new()) == NULL) {
26              return (NULL);
27          }
28      }
29      memcpy(dst, src, sizeof(t_mat4f));
30      return (dst);
31  }
32
33  /** set identity */
34  t_mat4f * mat4f_identity(t_mat4f * dst) {
35      if (dst == NULL) {
36          if ((dst = mat4f_new()) == NULL) {
37              return (NULL);
38          }
39      }
40
41      dst->m00 = 1, dst->m01 = 0, dst->m02 = 0, dst->m03 = 0;
42      dst->m10 = 0, dst->m11 = 1, dst->m12 = 0, dst->m13 = 0;
43      dst->m20 = 0, dst->m21 = 0, dst->m22 = 1, dst->m23 = 0;
44      dst->m30 = 0, dst->m31 = 0, dst->m32 = 0, dst->m33 = 1;
45      return (dst);
46  }
47
48  /** set to zero */
49  t_mat4f * mat4f_zero(t_mat4f * dst) {
50      if (dst == NULL) {
51          if ((dst = mat4f_new()) == NULL) {
52              return (NULL);
53          }
54      }
55      memset(dst, 0, sizeof(t_mat4f));
56      return (dst);
57  }
58
59  /** transpose */
60  t_mat4f * mat4f_transpose(t_mat4f * dst, t_mat4f * src) {
61
62      if (dst == NULL) {
63          if ((dst = mat4f_new()) == NULL) {
64              return (NULL);
65          }
66      }
67
68      float m00 = src->m00;
69      float m01 = src->m10;
70      float m02 = src->m20;
71      float m03 = src->m30;
72      float m10 = src->m01;
```

```
73      float m11 = src->m11;
74      float m12 = src->m21;
75      float m13 = src->m31;
76      float m20 = src->m02;
77      float m21 = src->m12;
78      float m22 = src->m22;
79      float m23 = src->m32;
80      float m30 = src->m03;
81      float m31 = src->m13;
82      float m32 = src->m23;
83      float m33 = src->m33;
84
85      dst->m00 = m00, dst->m01 = m01, dst->m02 = m02, dst->m03 = m03;
86      dst->m10 = m10, dst->m11 = m11, dst->m12 = m12, dst->m13 = m13;
87      dst->m20 = m20, dst->m21 = m21, dst->m22 = m22, dst->m23 = m23;
88      dst->m30 = m30, dst->m31 = m31, dst->m32 = m32, dst->m33 = m33;
89
90      return (dst);
91  }
92
93  /** scale */
94  t_mat4f * mat4f_scale(t_mat4f * dst, t_mat4f * src, float scale) {
95
96      if (dst == NULL) {
97          if ((dst = mat4f_new()) == NULL) {
98              return (NULL);
99          }
100     }
101
102     dst->m00 = src->m00 * scale, dst->m01 = src->m01 * scale, dst->m02 =
            src->m02 * scale, dst->m03 = src->m03 * scale;
103     dst->m10 = src->m10 * scale, dst->m11 = src->m11 * scale, dst->m12 =
            src->m12 * scale, dst->m13 = src->m13 * scale;
104     dst->m20 = src->m20 * scale, dst->m21 = src->m21 * scale, dst->m22 =
            src->m22 * scale, dst->m23 = src->m23 * scale;
105
106     //N.B: we do not scale last line
107     //dst->m30 = src->m30 * scale, dst->m31 = src->m31 * scale, dst->m32 =
            src->m32 * scale, dst->m33 = src->m33 * scale;
108
109     return (dst);
110 }
111
112 t_mat4f * mat4f_scale3(t_mat4f * dst, t_mat4f * src, t_vec3f * scale) {
113
114     if (dst == NULL) {
115         if ((dst = mat4f_new()) == NULL) {
116             return (NULL);
117         }
118     }
119
120     dst->m00 = src->m00 * scale->x, dst->m01 = src->m01 * scale->x, dst->
            m02 = src->m02 * scale->x, dst->m03 = src->m03 * scale->x;
```

```
121     dst->m10 = src->m10 * scale->y, dst->m11 = src->m11 * scale->y, dst->
            m12 = src->m12 * scale->y, dst->m13 = src->m13 * scale->y;
122     dst->m20 = src->m20 * scale->z, dst->m21 = src->m21 * scale->z, dst->
            m22 = src->m22 * scale->z, dst->m23 = src->m23 * scale->z;
123
124     //N.B: we do not scale last line
125     //dst->m30 = src->m30 * scale, dst->m31 = src->m31 * scale, dst->m32 =
            src->m32 * scale, dst->m33 = src->m33 * scale;
126
127     return (dst);
128 }
129
130 /** translate */
131 t_mat4f * mat4f_translate(t_mat4f * dst, t_mat4f * src, float tx, float ty
        , float tz) {
132     if (dst == NULL) {
133         if ((dst = mat4f_new()) == NULL) {
134             return (NULL);
135         }
136     }
137
138     dst->m30 += src->m00 * tx + src->m10 * ty + src->m20 * tz;
139     dst->m31 += src->m01 * tx + src->m11 * ty + src->m21 * tz;
140     dst->m32 += src->m02 * tx + src->m12 * ty + src->m22 * tz;
141     dst->m33 += src->m03 * tx + src->m13 * ty + src->m23 * tz;
142     return (dst);
143 }
144
145 t_mat4f * mat4f_translate3(t_mat4f * dst, t_mat4f * src, t_vec3f *
        translate) {
146     return (mat4f_translate(dst, src, translate->x, translate->y,
            translate->z));
147 }
148
149 /** rotate */
150 t_mat4f * mat4f_rotate(t_mat4f * dst, t_mat4f * src, float angle, t_vec3f
        * axis) {
151
152     if (dst == NULL) {
153         if ((dst = mat4f_new()) == NULL) {
154             return (NULL);
155         }
156     }
157
158     float c = (float)cos(angle);
159     float s = (float)sin(angle);
160     float oneminusc = 1.0f - c;
161     float xy = axis->x * axis->y;
162     float yz = axis->y * axis->z;
163     float xz = axis->x * axis->z;
164     float xs = axis->x * s;
165     float ys = axis->y * s;
166     float zs = axis->z * s;
167
```

```
168     float f00 = axis->x * axis->x * oneminusc + c;
169     float f01 = xy * oneminusc + zs;
170     float f02 = xz * oneminusc - ys;
171     // n[3] not used
172     float f10 = xy * oneminusc - zs;
173     float f11 = axis->y * axis->y * oneminusc + c;
174     float f12 = yz * oneminusc + xs;
175     // n[7] not used
176     float f20 = xz * oneminusc + ys;
177     float f21 = yz * oneminusc - xs;
178     float f22 = axis->z * axis->z * oneminusc + c;
179 
180     float t00 = src->m00 * f00 + src->m10 * f01 + src->m20 * f02;
181     float t01 = src->m01 * f00 + src->m11 * f01 + src->m21 * f02;
182     float t02 = src->m02 * f00 + src->m12 * f01 + src->m22 * f02;
183     float t03 = src->m03 * f00 + src->m13 * f01 + src->m23 * f02;
184     float t10 = src->m00 * f10 + src->m10 * f11 + src->m20 * f12;
185     float t11 = src->m01 * f10 + src->m11 * f11 + src->m21 * f12;
186     float t12 = src->m02 * f10 + src->m12 * f11 + src->m22 * f12;
187     float t13 = src->m03 * f10 + src->m13 * f11 + src->m23 * f12;
188     dst->m20 = src->m00 * f20 + src->m10 * f21 + src->m20 * f22;
189     dst->m21 = src->m01 * f20 + src->m11 * f21 + src->m21 * f22;
190     dst->m22 = src->m02 * f20 + src->m12 * f21 + src->m22 * f22;
191     dst->m23 = src->m03 * f20 + src->m13 * f21 + src->m23 * f22;
192     dst->m00 = t00;
193     dst->m01 = t01;
194     dst->m02 = t02;
195     dst->m03 = t03;
196     dst->m10 = t10;
197     dst->m11 = t11;
198     dst->m12 = t12;
199     dst->m13 = t13;
200     return (dst);
201 }
202 
203 t_mat4f * mat4f_rotateX(t_mat4f * dst, t_mat4f * src, float angle) {
204 
205     if (dst == NULL) {
206         if ((dst = mat4f_new()) == NULL) {
207             return (NULL);
208         }
209     }
210 
211     float c = (float)cos(angle);
212     float s = (float)sin(angle);
213     float t00 = src->m00;
214     float t01 = src->m01;
215     float t02 = src->m02;
216     float t03 = src->m03;
217     float t10 = src->m10 * c + src->m20 * s;
218     float t11 = src->m11 * c + src->m21 * s;
219     float t12 = src->m12 * c + src->m22 * s;
220     float t13 = src->m13 * c + src->m23 * s;
221     dst->m20 = src->m10 * -s + src->m20 * c;
```

```
222     dst->m21 = src->m11 * -s + src->m21 * c;
223     dst->m22 = src->m12 * -s + src->m22 * c;
224     dst->m23 = src->m13 * -s + src->m23 * c;
225     dst->m00 = t00;
226     dst->m01 = t01;
227     dst->m02 = t02;
228     dst->m03 = t03;
229     dst->m10 = t10;
230     dst->m11 = t11;
231     dst->m12 = t12;
232     dst->m13 = t13;
233 
234     return (dst);
235 }
236 
237 t_mat4f * mat4f_rotateY(t_mat4f * dst, t_mat4f * src, float angle) {
238 
239     if (dst == NULL) {
240         if ((dst = mat4f_new()) == NULL) {
241             return (NULL);
242         }
243     }
244 
245     float c = (float)cos(angle);
246     float s = (float)sin(angle);
247     float t00 = src->m00 * c + src->m20 * -s;
248     float t01 = src->m01 * c + src->m21 * -s;
249     float t02 = src->m02 * c + src->m22 * -s;
250     float t03 = src->m03 * c + src->m23 * -s;
251     float t10 = src->m10;
252     float t11 = src->m11;
253     float t12 = src->m12;
254     float t13 = src->m13;
255     dst->m20 = src->m00 * s + src->m20 * c;
256     dst->m21 = src->m01 * s + src->m21 * c;
257     dst->m22 = src->m02 * s + src->m22 * c;
258     dst->m23 = src->m03 * s + src->m23 * c;
259     dst->m00 = t00;
260     dst->m01 = t01;
261     dst->m02 = t02;
262     dst->m03 = t03;
263     dst->m10 = t10;
264     dst->m11 = t11;
265     dst->m12 = t12;
266     dst->m13 = t13;
267     return (dst);
268 }
269 
270 t_mat4f * mat4f_rotateZ(t_mat4f * dst, t_mat4f * src, float angle) {
271     if (dst == NULL) {
272         if ((dst = mat4f_new()) == NULL) {
273             return (NULL);
274         }
275     }
```

```
276
277     float c = (float)cos(angle);
278     float s = (float)sin(angle);
279     float t00 = src->m00 * c + src->m10 * s;
280     float t01 = src->m01 * c + src->m11 * s;
281     float t02 = src->m02 * c + src->m12 * s;
282     float t03 = src->m03 * c + src->m13 * s;
283     float t10 = src->m00 * -s + src->m10 * c;
284     float t11 = src->m01 * -s + src->m11 * c;
285     float t12 = src->m02 * -s + src->m12 * c;
286     float t13 = src->m03 * -s + src->m13 * c;
287     dst->m20 = src->m20;
288     dst->m21 = src->m21;
289     dst->m22 = src->m22;
290     dst->m23 = src->m23;
291     dst->m00 = t00;
292     dst->m01 = t01;
293     dst->m02 = t02;
294     dst->m03 = t03;
295     dst->m10 = t10;
296     dst->m11 = t11;
297     dst->m12 = t12;
298     dst->m13 = t13;
299     return (dst);
300 }
301
302 t_mat4f * mat4f_rotateXYZ(t_mat4f * dst, t_mat4f * src, t_vec3f * rot) {
303     if (dst == NULL) {
304         if ((dst = mat4f_new()) == NULL) {
305             return (NULL);
306         }
307     }
308
309     mat4f_rotateX(dst, src, rot->x);
310     mat4f_rotateY(dst, src, rot->y);
311     mat4f_rotateZ(dst, src, rot->z);
312     return (dst);
313 }
314
315 /** transformation matrix */
316 t_mat4f * mat4f_transformation(t_mat4f * dst, t_vec3f * translate, t_vec3f
        * rot, t_vec3f * scale) {
317
318     if ((dst = mat4f_identity(dst)) == NULL) {
319         return (NULL);
320     }
321
322     mat4f_translate3(dst, dst, translate);
323     mat4f_rotateXYZ(dst, dst, rot);
324     mat4f_scale3(dst, dst, scale);
325     return (dst);
326 }
327
328 /** determinant */
```

```
329 float mat4f_determinant(t_mat4f * mat) {
330
331     if (mat == NULL) {
332         return (0);
333     }
334
335     float d = 0;
336     d += mat->m00 * ((mat->m11 * mat->m22 * mat->m33 + mat->m12 * mat->m23
            * mat->m31 + mat->m13 * mat->m21 * mat->m32)
337         - mat->m13 * mat->m22 * mat->m31 - mat->m11 * mat->m23 * mat->m32
               - mat->m12 * mat->m21 * mat->m33);
338     d -= mat->m01 * ((mat->m10 * mat->m22 * mat->m33 + mat->m12 * mat->m23
            * mat->m30 + mat->m13 * mat->m20 * mat->m32)
339         - mat->m13 * mat->m22 * mat->m30 - mat->m10 * mat->m23 * mat->m32
               - mat->m12 * mat->m20 * mat->m33);
340     d += mat->m02 * ((mat->m10 * mat->m21 * mat->m33 + mat->m11 * mat->m23
            * mat->m30 + mat->m13 * mat->m20 * mat->m31)
341         - mat->m13 * mat->m21 * mat->m30 - mat->m10 * mat->m23 * mat->m31
               - mat->m11 * mat->m20 * mat->m33);
342     d -= mat->m03 * ((mat->m10 * mat->m21 * mat->m32 + mat->m11 * mat->m22
            * mat->m30 + mat->m12 * mat->m20 * mat->m31)
343         - mat->m12 * mat->m21 * mat->m30 - mat->m10 * mat->m22 * mat->m31
               - mat->m11 * mat->m20 * mat->m32);
344     return (d);
345 }
346
347 static float _mat4f_determinant3x3(float t00, float t01, float t02, float
       t10, float t11, float t12, float t20, float t21, float t22) {
348     return (t00 * (t11 * t22 - t12 * t21) + t01 * (t12 * t20 - t10 * t22)
           + t02 * (t10 * t21 - t11 * t20));
349 }
350
351 /** invert */
352 t_mat4f * mat4f_invert(t_mat4f * dst, t_mat4f * src) {
353
354     float determinant = mat4f_determinant(src);
355
356     if (determinant != 0) {
357         if (dst == NULL) {
358             if ((dst = mat4f_new()) == NULL) {
359                 return (NULL);
360             }
361         }
362
363         float determinant_inv = 1.0f / determinant;
364
365         float t00 = _mat4f_determinant3x3(src->m11, src->m12, src->m13,
               src->m21, src->m22, src->m23, src->m31, src->m32, src->m33);
366         float t01 = -_mat4f_determinant3x3(src->m10, src->m12, src->m13,
               src->m20, src->m22, src->m23, src->m30, src->m32, src->m33);
367         float t02 = _mat4f_determinant3x3(src->m10, src->m11, src->m13,
               src->m20, src->m21, src->m23, src->m30, src->m31, src->m33);
368         float t03 = -_mat4f_determinant3x3(src->m10, src->m11, src->m12,
               src->m20, src->m21, src->m22, src->m30, src->m31, src->m32);
```

```
369
370         float t10 = -_mat4f_determinant3x3(src->m01, src->m02, src->m03,
                src->m21, src->m22, src->m23, src->m31, src->m32, src->m33);
371         float t11 = _mat4f_determinant3x3(src->m00, src->m02, src->m03,
                src->m20, src->m22, src->m23, src->m30, src->m32, src->m33);
372         float t12 = -_mat4f_determinant3x3(src->m00, src->m01, src->m03,
                src->m20, src->m21, src->m23, src->m30, src->m31, src->m33);
373         float t13 = _mat4f_determinant3x3(src->m00, src->m01, src->m02,
                src->m20, src->m21, src->m22, src->m30, src->m31, src->m32);
374
375         float t20 = _mat4f_determinant3x3(src->m01, src->m02, src->m03,
                src->m11, src->m12, src->m13, src->m31, src->m32, src->m33);
376         float t21 = -_mat4f_determinant3x3(src->m00, src->m02, src->m03,
                src->m10, src->m12, src->m13, src->m30, src->m32, src->m33);
377         float t22 = _mat4f_determinant3x3(src->m00, src->m01, src->m03,
                src->m10, src->m11, src->m13, src->m30, src->m31, src->m33);
378         float t23 = -_mat4f_determinant3x3(src->m00, src->m01, src->m02,
                src->m10, src->m11, src->m12, src->m30, src->m31, src->m32);
379
380         float t30 = -_mat4f_determinant3x3(src->m01, src->m02, src->m03,
                src->m11, src->m12, src->m13, src->m21, src->m22, src->m23);
381         float t31 = _mat4f_determinant3x3(src->m00, src->m02, src->m03,
                src->m10, src->m12, src->m13, src->m20, src->m22, src->m23);
382         float t32 = -_mat4f_determinant3x3(src->m00, src->m01, src->m03,
                src->m10, src->m11, src->m13, src->m20, src->m21, src->m23);
383         float t33 = _mat4f_determinant3x3(src->m00, src->m01, src->m02,
                src->m10, src->m11, src->m12, src->m20, src->m21, src->m22);
384
385         // transpose and divide by the determinant
386         dst->m00 = t00 * determinant_inv;
387         dst->m11 = t11 * determinant_inv;
388         dst->m22 = t22 * determinant_inv;
389         dst->m33 = t33 * determinant_inv;
390         dst->m01 = t10 * determinant_inv;
391         dst->m10 = t01 * determinant_inv;
392         dst->m20 = t02 * determinant_inv;
393         dst->m02 = t20 * determinant_inv;
394         dst->m12 = t21 * determinant_inv;
395         dst->m21 = t12 * determinant_inv;
396         dst->m03 = t30 * determinant_inv;
397         dst->m30 = t03 * determinant_inv;
398         dst->m13 = t31 * determinant_inv;
399         dst->m31 = t13 * determinant_inv;
400         dst->m32 = t23 * determinant_inv;
401         dst->m23 = t32 * determinant_inv;
402         return (dst);
403     }
404
405     return (NULL);
406 }
407
408 /** mult */
409 t_mat4f * mat4f_mult(t_mat4f * dst, t_mat4f * left, t_mat4f * right) {
410
```

```
411     if (dst == NULL) {
412         if ((dst = mat4f_new()) == NULL) {
413             return (NULL);
414         }
415     }
416 
417     float m00 = left->m00 * right->m00 + left->m10 * right->m01 + left->
            m20 * right->m02 + left->m30 * right->m03;
418     float m01 = left->m01 * right->m00 + left->m11 * right->m01 + left->
            m21 * right->m02 + left->m31 * right->m03;
419     float m02 = left->m02 * right->m00 + left->m12 * right->m01 + left->
            m22 * right->m02 + left->m32 * right->m03;
420     float m03 = left->m03 * right->m00 + left->m13 * right->m01 + left->
            m23 * right->m02 + left->m33 * right->m03;
421     float m10 = left->m00 * right->m10 + left->m10 * right->m11 + left->
            m20 * right->m12 + left->m30 * right->m13;
422     float m11 = left->m01 * right->m10 + left->m11 * right->m11 + left->
            m21 * right->m12 + left->m31 * right->m13;
423     float m12 = left->m02 * right->m10 + left->m12 * right->m11 + left->
            m22 * right->m12 + left->m32 * right->m13;
424     float m13 = left->m03 * right->m10 + left->m13 * right->m11 + left->
            m23 * right->m12 + left->m33 * right->m13;
425     float m20 = left->m00 * right->m20 + left->m10 * right->m21 + left->
            m20 * right->m22 + left->m30 * right->m23;
426     float m21 = left->m01 * right->m20 + left->m11 * right->m21 + left->
            m21 * right->m22 + left->m31 * right->m23;
427     float m22 = left->m02 * right->m20 + left->m12 * right->m21 + left->
            m22 * right->m22 + left->m32 * right->m23;
428     float m23 = left->m03 * right->m20 + left->m13 * right->m21 + left->
            m23 * right->m22 + left->m33 * right->m23;
429     float m30 = left->m00 * right->m30 + left->m10 * right->m31 + left->
            m20 * right->m32 + left->m30 * right->m33;
430     float m31 = left->m01 * right->m30 + left->m11 * right->m31 + left->
            m21 * right->m32 + left->m31 * right->m33;
431     float m32 = left->m02 * right->m30 + left->m12 * right->m31 + left->
            m22 * right->m32 + left->m32 * right->m33;
432     float m33 = left->m03 * right->m30 + left->m13 * right->m31 + left->
            m23 * right->m32 + left->m33 * right->m33;
433 
434     dst->m00 = m00;
435     dst->m01 = m01;
436     dst->m02 = m02;
437     dst->m03 = m03;
438     dst->m10 = m10;
439     dst->m11 = m11;
440     dst->m12 = m12;
441     dst->m13 = m13;
442     dst->m20 = m20;
443     dst->m21 = m21;
444     dst->m22 = m22;
445     dst->m23 = m23;
446     dst->m30 = m30;
447     dst->m31 = m31;
448     dst->m32 = m32;
```

```
449     dst->m33 = m33;
450
451     return (dst);
452 }
453
454 /** transform vec4f */
455 t_vec4f * mat4f_transform_vec4f(t_vec4f * dst, t_mat4f * left, t_vec4f *
        right) {
456     if (dst == NULL) {
457         if ((dst = vec4f_new()) == NULL) {
458             return (NULL);
459         }
460     }
461
462     float x = left->m00 * right->x + left->m10 * right->y + left->m20 *
            right->z + left->m30 * right->w;
463     float y = left->m01 * right->x + left->m11 * right->y + left->m21 *
            right->z + left->m31 * right->w;
464     float z = left->m02 * right->x + left->m12 * right->y + left->m22 *
            right->z + left->m32 * right->w;
465     float w = left->m03 * right->x + left->m13 * right->y + left->m23 *
            right->z + left->m33 * right->w;
466
467     dst->x = x;
468     dst->y = y;
469     dst->z = z;
470     dst->w = w;
471
472     return (dst);
473 }
474
475 /** projections matrix bellow: */
476
477 /** orthographic matrix */
478 t_mat4f * mat4f_orthographic(t_mat4f * dst, float left, float right, float
        bot, float top, float near, float far) {
479     if (dst == NULL) {
480         if ((dst = mat4f_new()) == NULL) {
481             return (NULL);
482         }
483     }
484
485     dst->m00 = 2.0f / (right - left);
486     dst->m01 = 0;
487     dst->m02 = 0;
488     dst->m03 = (right + left) / (left - right);
489
490     dst->m10 = 0;
491     dst->m11 = 2.0f / (top - bot);
492     dst->m12 = 0;
493     dst->m13 = (top + bot) / (bot - top);
494
495     dst->m20 = 0;
496     dst->m21 = 0;
```

```
497     dst->m22 = 2 / (near - far);
498     dst->m23 = (far + near) / (near - far);
499 
500     dst->m30 = 0.0f;
501     dst->m31 = 0.0f;
502     dst->m32 = 0.0f;
503     dst->m33 = 1.0f;
504 
505     return (dst);
506 }
507 
508 /** perspective matrix */
509 t_mat4f * mat4f_perspective(t_mat4f * dst, float aspect, float fov, float
        near, float far) {
510     if (dst == NULL) {
511         if ((dst = mat4f_new()) == NULL) {
512             return (NULL);
513         }
514     }
515 
516     float y_scale = (float) (1.0f / tan(fov / 2.0f) * aspect);
517     float x_scale = y_scale / aspect;
518     float frustrum_length = far - near;
519 
520     dst->m00 = x_scale;
521     dst->m01 = 0.0f;
522     dst->m02 = 0.0f;
523     dst->m03 = 0.0f;
524 
525     dst->m10 = 0.0f;
526     dst->m11 = y_scale;
527     dst->m12 = 0.0f;
528     dst->m13 = 0.0f;
529 
530     dst->m20 = 0.0f;
531     dst->m21 = 0.0f;
532     dst->m22 = -((far + near) / frustrum_length);
533     dst->m23 = -1.0f;
534 
535     dst->m30 = 0.0f;
536     dst->m31 = 0.0f;
537     dst->m32 = -((2.0f * near * far) / frustrum_length);
538     dst->m33 = 0.0f;
539 
540     return (dst);
541 }
542 
543 /** to string: return a string allocated with malloc() */
544 char * mat4f_str(t_mat4f * mat) {
545     if (mat == NULL) {
546         return (strdup("mat4f(NULL)"));
547     }
548     char buffer[1024];
```

```
549     sprintf(buffer, "mat4f(%.4f ; %.4f ; %.4f ; %.4f\n       %.4f ; %.4f ;
            %.4f ; %.4f\n       %.4f ; %.4f ; %.4f ; %.4f\n       %.4f ; %.4f ;
            %.4f ; %.4f)",
550          mat->m00, mat->m10, mat->m20, mat->m30, mat->m01, mat->m11, mat->
                m21, mat->m31, mat->m02, mat->m12, mat->m22, mat->m32, mat->m03
                , mat->m13, mat->m23, mat->m33);
551     return (strdup(buffer));
552 }
553
554 /*
555 int main() {
556
557     t_mat4f mat;
558
559     mat4f_identity(&mat);
560     mat4f_scale(&mat, &mat, 1 / 7.0f);
561     char * str = mat4f_str(&mat);
562     puts(str);
563     free(str);
564     return (0);
565 }*/
```

maths/mat4f.h

```
1  /**
2   *  This file is part of https://github.com/toss-dev/C_maths
3   *
4   *  It is under a GNU GENERAL PUBLIC LICENSE
5   *
6   *  This library is still in development, so please, if you find any issue
        , let me know about it on github.com
7   *  PEREIRA Romain
8   */
9
10 #ifndef MAT4F_H
11 # define MAT4F_H
12
13 # include "cmaths.h"
14 # include "vec.h"
15
16 typedef struct  s_mat4f {
17     float m00;
18     float m01;
19     float m02;
20     float m03;
21
22     float m10;
23     float m11;
24     float m12;
25     float m13;
26
27     float m20;
```

```
28      float m21;
29      float m22;
30      float m23;
31
32      float m30;
33      float m31;
34      float m32;
35      float m33;
36  }               t_mat4f;
37
38  /** create a new matrix */
39  t_mat4f * mat4f_new(void);
40
41  /** delete the given matrix */
42  void mat4f_delete(t_mat4f * mat);
43
44  /** copy */
45  t_mat4f * mat4f_copy(t_mat4f * dst, t_mat4f * src);
46
47  /** set identity */
48  t_mat4f * mat4f_identity(t_mat4f * dst);
49
50  /** set to zero */
51  t_mat4f * mat4f_zero(t_mat4f * dst);
52
53  /** transpose */
54  t_mat4f * mat4f_transpose(t_mat4f * dst, t_mat4f * src);
55
56  /** scale */
57  t_mat4f * mat4f_scale(t_mat4f * dst, t_mat4f * mat, float f);
58
59  /** translate */
60  t_mat4f * mat4f_translate(t_mat4f * dst, t_mat4f * src, float tx, float ty
        , float tz);
61  t_mat4f * mat4f_translate3(t_mat4f * dst, t_mat4f * src, t_vec3f *
        translate);
62
63  /** rotate */
64  t_mat4f * mat4f_rotate(t_mat4f * dst, t_mat4f * src, float angle, t_vec3f
        * axis);
65  t_mat4f * mat4f_rotateX(t_mat4f * dst, t_mat4f * src, float angle);
66  t_mat4f * mat4f_rotateY(t_mat4f * dst, t_mat4f * src, float angle);
67  t_mat4f * mat4f_rotateZ(t_mat4f * dst, t_mat4f * src, float angle);
68  t_mat4f * mat4f_rotateXYZ(t_mat4f * dst, t_mat4f * src, t_vec3f * rot);
69
70  /** transformation matrix */
71  t_mat4f * mat4f_transformation(t_mat4f * dst, t_vec3f * translate, t_vec3f
        * rot, t_vec3f * scale);
72
73  /** determinant */
74  float mat4f_determinant(t_mat4f * mat);
75
76  /** invert */
77  t_mat4f * mat4f_invert(t_mat4f * dst, t_mat4f * src);
```

```
78
79 /** mult */
80 t_mat4f * mat4f_mult(t_mat4f * dst, t_mat4f * left, t_mat4f * right);
81
82 /** transform vec4f */
83 t_vec4f * mat4f_transform_vec4f(t_vec4f * dst, t_mat4f * left, t_vec4f *
       right);
84
85 /** projections matrix bellow: */
86
87 /** orthographic matrix */
88 t_mat4f * mat4f_orthographic(t_mat4f * dst, float left, float right, float
        bot, float top, float near, float far);
89
90 /** perspective matrix */
91 t_mat4f * mat4f_perspective(t_mat4f * dst, float aspect, float fov, float
       near, float far);
92
93 /** to string: return a string allocated with malloc() */
94 char * mat4f_str(t_mat4f * mat);
95
96 #endif
```

maths/vec.h

```
1  /**
2   *  This file is part of https://github.com/toss-dev/C_maths
3   *
4   *  It is under a GNU GENERAL PUBLIC LICENSE
5   *
6   *  This library is still in development, so please, if you find any issue
        , let me know about it on github.com
7   *  PEREIRA Romain
8   */
9
10 #ifndef VEC_H
11 # define VEC_H
12
13 # include "vecf.h"
14 # include "veci.h"
15
16 #endif
```

maths/vec2f.c

```
1  /**
2   *  This file is part of https://github.com/toss-dev/C_maths
3   *
4   *  It is under a GNU GENERAL PUBLIC LICENSE
5   *
```

```
 *  This library is still in development, so please, if you find any issue
    , let me know about it on github.com
 *  PEREIRA Romain
 */

#include "vec2f.h"

/** create a new vec2f */
t_vec2f * vec2f_new(void) {
    return ((t_vec2f *)malloc(sizeof(t_vec2f)));
}

/** delete the vec2f */
void vec2f_delete(t_vec2f * vec) {
    free(vec);
}

/** set the vec2f to 0 */
t_vec2f * vec2f_zero(t_vec2f * dst) {
    if (dst == NULL) {
        if ((dst = vec2f_new()) == NULL) {
            return (NULL);
        }
    }
    memset(dst, 0, sizeof(t_vec2f));
    return (dst);
}

/** set the vec2f values */
t_vec2f * vec2f_set(t_vec2f * dst, float x, float y) {
    if (dst == NULL) {
        if ((dst = vec2f_new()) == NULL) {
            return (NULL);
        }
    }
    dst->x = x;
    dst->y = y;
    return (dst);
}

t_vec2f * vec2f_set2(t_vec2f * dst, t_vec2f * vec) {
    if (dst == vec) {
        return (dst);
    }
    return (vec2f_set(dst, vec->x, vec->y));
}

/** add two vec2f */
t_vec2f * vec2f_add(t_vec2f * dst, t_vec2f * left, t_vec2f * right) {
    if (dst == NULL) {
        if ((dst = vec2f_new()) == NULL) {
            return (NULL);
        }
    }
```

```
59      dst->x = left->x + right->x;
60      dst->y = left->y + right->y;
61      return (dst);
62  }
63
64  /** sub two vec2f */
65  t_vec2f * vec2f_sub(t_vec2f * dst, t_vec2f * left, t_vec2f * right) {
66      if (dst == NULL) {
67          if ((dst = vec2f_new()) == NULL) {
68              return (NULL);
69          }
70      }
71      dst->x = left->x - right->x;
72      dst->y = left->y - right->y;
73      return (dst);
74  }
75
76  /** mult the vec2f by the given scalar */
77  t_vec2f * vec2f_mult(t_vec2f * dst, t_vec2f * vec, float scalar) {
78      if (dst == NULL) {
79          if ((dst = vec2f_new()) == NULL) {
80              return (NULL);
81          }
82      }
83      dst->x = vec->x * scalar;
84      dst->y = vec->y * scalar;
85      return (dst);
86  }
87
88  t_vec2f * vec2f_mult2(t_vec2f * dst, t_vec2f * left, t_vec2f * right) {
89      if (dst == NULL) {
90          if ((dst = vec2f_new()) == NULL) {
91              return (NULL);
92          }
93      }
94      dst->x = left->x * right->x;
95      dst->y = left->y * right->y;
96      return (dst);
97  }
98
99  /** scale product */
100 float vec2f_dot_product(t_vec2f * left, t_vec2f * right) {
101     return (left->x * right->x + left->y * right->y);
102 }
103
104 /** length */
105 float vec2f_length_squared(t_vec2f * vec) {
106     return (vec2f_dot_product(vec, vec));
107 }
108
109 float vec2f_length(t_vec2f * vec) {
110     return ((float)sqrt(vec2f_length_squared(vec)));
111 }
112
```

```
113 /** normalize */
114 t_vec2f * vec2f_normalize(t_vec2f * dst, t_vec2f * vec) {
115 
116     if (dst == NULL) {
117         if ((dst = vec2f_new()) == NULL) {
118             return (NULL);
119         }
120     }
121 
122     float norm = 1 / vec2f_length(vec);
123     dst->x = vec->x * norm;
124     dst->y = vec->y * norm;
125     return (dst);
126 }
127 
128 /** negate */
129 t_vec2f * vec2f_negate(t_vec2f * dst, t_vec2f * src) {
130     if (dst == NULL) {
131         if ((dst = vec2f_new()) == NULL) {
132             return (NULL);
133         }
134     }
135     dst->x = -src->x;
136     dst->y = -src->y;
137     return (dst);
138 }
139 
140 /** angle between two vec */
141 float vec2f_angle(t_vec2f * left, t_vec2f * right) {
142     float dls = vec2f_dot_product(left, right) / (vec2f_length(left) *
        vec2f_length(right));
143     if (dls < -1.0f) {
144         dls = -1.0f;
145     } else if (dls > 1.0f) {
146         dls = 1.0f;
147     }
148     return ((float)acos(dls));
149 }
150 
151 /** mix the two vectors */
152 t_vec2f * vec2f_mix(t_vec2f * dst, t_vec2f * left, t_vec2f * right, float
    ratio) {
153 
154     if (dst == NULL) {
155         if ((dst = vec2f_new()) == NULL) {
156             return (NULL);
157         }
158     }
159 
160     dst->x = left->x * ratio + right->x * (1 - ratio);
161     dst->y = left->y * ratio + right->y * (1 - ratio);
162     return (dst);
163 }
164 
```

```c
/** comparison */
int vec2f_equals(t_vec2f * left, t_vec2f * right) {
    return (left == right || (left->x == right->x && left->y == right->y))
        ;
}

int vec2f_nequals(t_vec2f * left, t_vec2f * right) {
    return (!vec2f_equals(left, right));
}

/** hash */
int vec2f_hash(t_vec2f * vec) {
    return ((int)(vec->x * 73856093.0f) ^ (int)(vec->y * 19349663.0f));
}

/** round vec2f */
t_vec2f * vec2f_round(t_vec2f * dst, t_vec2f * vec, int decimals) {
    static float powten[] = {1, 10, 100, 1000, 10000, 100000, 1000000,
        10000000, 100000000, 1000000000};

    if (decimals < 0 || decimals >= 10) {
        return (vec2f_set2(dst, vec));
    }

    if (dst == NULL) {
        if ((dst = vec2f_new()) == NULL) {
            return (NULL);
        }
    }

    dst->x = roundf(powten[decimals] * vec->x) / powten[decimals];
    dst->y = roundf(powten[decimals] * vec->y) / powten[decimals];
    return (dst);
}

/** to string: return a string allocated with malloc() */
char * vec2f_str(t_vec2f * vec) {
    if (vec == NULL) {
        return (strdup("vec2f(NULL)"));
    }
    char buffer[128];
    sprintf(buffer, "vec2f(%f ; %f)", vec->x, vec->y);
    return (strdup(buffer));
}
```

maths/vec2f.h

```c
/**
 *  This file is part of https://github.com/toss-dev/C_maths
 *
 *  It is under a GNU GENERAL PUBLIC LICENSE
 *
```

```
 *  This library is still in development, so please, if you find any issue
    , let me know about it on github.com
 *  PEREIRA Romain
 */

#ifndef VEC2F_H
# define VEC2F_H

# include "cmaths.h"

typedef struct  s_vec2f {
    union {
        float x;
        float uvx;
    };

    union {
        float y;
        float uvy;
    };
}               t_vec2f;

/** create a new vec2f */
t_vec2f * vec2f_new(void);

/** delete the vec2f */
void vec2f_delete(t_vec2f * vec);

/** set the vec2f to 0 */
t_vec2f * vec2f_zero(t_vec2f * dst);

/** set the vec2f values */
t_vec2f * vec2f_set(t_vec2f * dst, float x, float y);
t_vec2f * vec2f_set2(t_vec2f * dst, t_vec2f * vec);

/** add two vec2f */
t_vec2f * vec2f_add(t_vec2f * dst, t_vec2f * left, t_vec2f * right);

/** sub two vec2f */
t_vec2f * vec2f_sub(t_vec2f * dst, t_vec2f * left, t_vec2f * right);

/** mult the vec2f by the given scalar */
t_vec2f * vec2f_mult(t_vec2f * dst, t_vec2f * vec, float scalar);
t_vec2f * vec2f_mult2(t_vec2f * dst, t_vec2f * left, t_vec2f * right);

/** scale product */
float vec2f_dot_product(t_vec2f * left, t_vec2f * right);

/** length */
float vec2f_length_squared(t_vec2f * vec);
float vec2f_length(t_vec2f * vec);

/** normalize */
t_vec2f * vec2f_normalize(t_vec2f * dst, t_vec2f * vec);
```

```c

/** negate */
t_vec2f * vec2f_negate(t_vec2f * dst, t_vec2f * src);

/** angle between two vec */
float vec2f_angle(t_vec2f * left, t_vec2f * right);

/** mix the two vectors */
t_vec2f * vec2f_mix(t_vec2f * dst, t_vec2f * left, t_vec2f * right, float
    ratio);

/** comparison */
int vec2f_equals(t_vec2f * left, t_vec2f * right);
int vec2f_nequals(t_vec2f * left, t_vec2f * right);

/** hash */
int vec2f_hash(t_vec2f * vec);

/** round vec2f */
t_vec2f * vec2f_round(t_vec2f * dst, t_vec2f * vec, int decimals);

/** to string: return a string allocated with malloc() */
char * vec2f_str(t_vec2f * vec);

#endif
```

maths/vec2i.c

```c
/**
 *  This file is part of https://github.com/toss-dev/C_maths
 *
 *  It is under a GNU GENERAL PUBLIC LICENSE
 *
 *  This library is still in development, so please, if you find any issue
    , let me know about it on github.com
 *  PEREIRA Romain
 */

#include "vec2i.h"

/** create a new vec2i */
t_vec2i * vec2i_new(void) {
    return ((t_vec2i *)malloc(sizeof(t_vec2i)));
}

/** delete the vec2i */
void vec2i_delete(t_vec2i * vec) {
    free(vec);
}

/** set the vec2i to 0 */
t_vec2i * vec2i_zero(t_vec2i * dst) {
```

```
24      if (dst == NULL) {
25          if ((dst = vec2i_new()) == NULL) {
26              return (NULL);
27          }
28      }
29      memset(dst, 0, sizeof(t_vec2i));
30      return (dst);
31  }
32
33  /** set the vec2i values */
34  t_vec2i * vec2i_set(t_vec2i * dst, int x, int y) {
35      if (dst == NULL) {
36          if ((dst = vec2i_new()) == NULL) {
37              return (NULL);
38          }
39      }
40      dst->x = x;
41      dst->y = y;
42      return (dst);
43  }
44
45  t_vec2i * vec2i_set2(t_vec2i * dst, t_vec2i * vec) {
46      if (dst == vec) {
47          return (dst);
48      }
49      return (vec2i_set(dst, vec->x, vec->y));
50  }
51
52  /** add two vec2i */
53  t_vec2i * vec2i_add(t_vec2i * dst, t_vec2i * left, t_vec2i * right) {
54      if (dst == NULL) {
55          if ((dst = vec2i_new()) == NULL) {
56              return (NULL);
57          }
58      }
59      dst->x = left->x + right->x;
60      dst->y = left->y + right->y;
61      return (dst);
62  }
63
64  /** sub two vec2i */
65  t_vec2i * vec2i_sub(t_vec2i * dst, t_vec2i * left, t_vec2i * right) {
66      if (dst == NULL) {
67          if ((dst = vec2i_new()) == NULL) {
68              return (NULL);
69          }
70      }
71      dst->x = left->x - right->x;
72      dst->y = left->y - right->y;
73      return (dst);
74  }
75
76  /** mult the vec2i by the given scalar */
77  t_vec2i * vec2i_mult(t_vec2i * dst, t_vec2i * vec, int scalar) {
```

```
78      if (dst == NULL) {
79          if ((dst = vec2i_new()) == NULL) {
80              return (NULL);
81          }
82      }
83      dst->x = vec->x * scalar;
84      dst->y = vec->y * scalar;
85      return (dst);
86  }
87
88  t_vec2i * vec2i_mult2(t_vec2i * dst, t_vec2i * left, t_vec2i * right) {
89      if (dst == NULL) {
90          if ((dst = vec2i_new()) == NULL) {
91              return (NULL);
92          }
93      }
94      dst->x = left->x * right->x;
95      dst->y = left->y * right->y;
96      return (dst);
97  }
98
99  /** scale product */
100 int vec2i_dot_product(t_vec2i * left, t_vec2i * right) {
101     return (left->x * right->x + left->y * right->y);
102 }
103
104 /** length */
105 int vec2i_length_squared(t_vec2i * vec) {
106     return (vec2i_dot_product(vec, vec));
107 }
108
109 int vec2i_length(t_vec2i * vec) {
110     return ((int)sqrt(vec2i_length_squared(vec)));
111 }
112
113 /** normalize */
114 t_vec2i * vec2i_normalize(t_vec2i * dst, t_vec2i * vec) {
115
116     if (dst == NULL) {
117         if ((dst = vec2i_new()) == NULL) {
118             return (NULL);
119         }
120     }
121
122     int norm = 1 / vec2i_length(vec);
123     dst->x = vec->x * norm;
124     dst->y = vec->y * norm;
125     return (dst);
126 }
127
128 /** negate */
129 t_vec2i * vec2i_negate(t_vec2i * dst, t_vec2i * src) {
130     if (dst == NULL) {
131         if ((dst = vec2i_new()) == NULL) {
```

```
132             return (NULL);
133         }
134     }
135     dst->x = -src->x;
136     dst->y = -src->y;
137     return (dst);
138 }
139 
140 /** angle between two vec */
141 int vec2i_angle(t_vec2i * left, t_vec2i * right) {
142     int dls = vec2i_dot_product(left, right) / (vec2i_length(left) *
        vec2i_length(right));
143     if (dls < -1.0f) {
144         dls = -1.0f;
145     } else if (dls > 1.0f) {
146         dls = 1.0f;
147     }
148     return ((int)acos(dls));
149 }
150 
151 /** mix the two vectors */
152 t_vec2i * vec2i_mix(t_vec2i * dst, t_vec2i * left, t_vec2i * right, int
    ratio) {
153 
154     if (dst == NULL) {
155         if ((dst = vec2i_new()) == NULL) {
156             return (NULL);
157         }
158     }
159 
160     dst->x = left->x * ratio + right->x * (1 - ratio);
161     dst->y = left->y * ratio + right->y * (1 - ratio);
162     return (dst);
163 }
164 
165 /** comparison */
166 int vec2i_equals(t_vec2i * left, t_vec2i * right) {
167     return (left == right || (left->x == right->x && left->y == right->y))
        ;
168 }
169 
170 int vec2i_nequals(t_vec2i * left, t_vec2i * right) {
171     return (!vec2i_equals(left, right));
172 }
173 
174 /** hash */
175 int vec2i_hash(t_vec2i * vec) {
176     return ((vec->x * 73856093) ^ (vec->y * 19349663));
177 }
178 
179 /** to string: return a string allocated with malloc() */
180 char * vec2i_str(t_vec2i * vec) {
181     if (vec == NULL) {
182         return (strdup("vec2i(NULL)"));
```

```
183     }
184     char buffer[128];
185     sprintf(buffer, "vec2i(%d ; %d)", vec->x, vec->y);
186     return (strdup(buffer));
187 }
```

maths/vec2i.h

```
1  /**
2   *  This file is part of https://github.com/toss-dev/C_maths
3   *
4   *  It is under a GNU GENERAL PUBLIC LICENSE
5   *
6   *  This library is still in development, so please, if you find any issue
        , let me know about it on github.com
7   *  PEREIRA Romain
8   */
9
10 #ifndef VEC2I_H
11 # define VEC2I_H
12
13 # include "cmaths.h"
14
15 typedef struct  s_vec2i {
16     union {
17         int x;
18         int uvx;
19     };
20
21     union {
22         int y;
23         int uvy;
24     };
25 }               t_vec2i;
26
27 /** create a new vec2i */
28 t_vec2i * vec2i_new(void);
29
30 /** delete the vec2i */
31 void vec2i_delete(t_vec2i * vec);
32
33 /** set the vec2i to 0 */
34 t_vec2i * vec2i_zero(t_vec2i * dst);
35
36 /** set the vec2i values */
37 t_vec2i * vec2i_set(t_vec2i * dst, int x, int y);
38 t_vec2i * vec2i_set2(t_vec2i * dst, t_vec2i * vec);
39
40 /** add two vec2i */
41 t_vec2i * vec2i_add(t_vec2i * dst, t_vec2i * left, t_vec2i * right);
42
43 /** sub two vec2i */
```

```
44 t_vec2i * vec2i_sub(t_vec2i * dst, t_vec2i * left, t_vec2i * right);
45
46 /** mult the vec2i by the given scalar */
47 t_vec2i * vec2i_mult(t_vec2i * dst, t_vec2i * vec, int scalar);
48 t_vec2i * vec2i_mult2(t_vec2i * dst, t_vec2i * left, t_vec2i * right);
49
50 /** scale product */
51 int vec2i_dot_product(t_vec2i * left, t_vec2i * right);
52
53 /** length */
54 int vec2i_length_squared(t_vec2i * vec);
55 int vec2i_length(t_vec2i * vec);
56
57 /** normalize */
58 t_vec2i * vec2i_normalize(t_vec2i * dst, t_vec2i * vec);
59
60 /** negate */
61 t_vec2i * vec2i_negate(t_vec2i * dst, t_vec2i * src);
62
63 /** angle between two vec */
64 int vec2i_angle(t_vec2i * left, t_vec2i * right);
65
66 /** mix the two vectors */
67 t_vec2i * vec2i_mix(t_vec2i * dst, t_vec2i * left, t_vec2i * right, int
      ratio);
68
69 /** comparison */
70 int vec2i_equals(t_vec2i * left, t_vec2i * right);
71 int vec2i_nequals(t_vec2i * left, t_vec2i * right);
72
73 /** hash */
74 int vec2i_hash(t_vec2i * vec);
75
76 /** to string: return a string allocated with malloc() */
77 char * vec2i_str(t_vec2i * vec);
78
79 #endif
```

maths/vec3f.c

```
1  /**
2   *  This file is part of https://github.com/toss-dev/C_maths
3   *
4   *  It is under a GNU GENERAL PUBLIC LICENSE
5   *
6   *  This library is still in development, so please, if you find any issue
        , let me know about it on github.com
7   *  PEREIRA Romain
8   */
9
10 #include "vec3f.h"
11
```

```
12 /** create a new vec3f */
13 t_vec3f * vec3f_new(void) {
14     return ((t_vec3f *)malloc(sizeof(t_vec3f)));
15 }
16
17 /** delete the vec3f */
18 void vec3f_delete(t_vec3f * vec) {
19     free(vec);
20 }
21
22 /** set the vec3f to 0 */
23 t_vec3f * vec3f_zero(t_vec3f * dst) {
24     if (dst == NULL) {
25         if ((dst = vec3f_new()) == NULL) {
26             return (NULL);
27         }
28     }
29     memset(dst, 0, sizeof(t_vec3f));
30     return (dst);
31 }
32
33 /** set the vec3f values */
34 t_vec3f * vec3f_set(t_vec3f * dst, float x, float y, float z) {
35     if (dst == NULL) {
36         if ((dst = vec3f_new()) == NULL) {
37             return (NULL);
38         }
39     }
40     dst->x = x;
41     dst->y = y;
42     dst->z = z;
43     return (dst);
44 }
45
46 t_vec3f * vec3f_set3(t_vec3f * dst, t_vec3f * vec) {
47     if (dst == vec) {
48         return (dst);
49     }
50     return (vec3f_set(dst, vec->x, vec->y, vec->z));
51 }
52
53 /** add two vec3f */
54 t_vec3f * vec3f_add(t_vec3f * dst, t_vec3f * left, t_vec3f * right) {
55     if (dst == NULL) {
56         if ((dst = vec3f_new()) == NULL) {
57             return (NULL);
58         }
59     }
60     dst->x = left->x + right->x;
61     dst->y = left->y + right->y;
62     dst->z = left->z + right->z;
63     return (dst);
64 }
65
```

```
66  /** sub two vec3f */
67  t_vec3f * vec3f_sub(t_vec3f * dst, t_vec3f * left, t_vec3f * right) {
68      if (dst == NULL) {
69          if ((dst = vec3f_new()) == NULL) {
70              return (NULL);
71          }
72      }
73      dst->x = left->x - right->x;
74      dst->y = left->y - right->y;
75      dst->z = left->z - right->z;
76      return (dst);
77  }
78
79  /** mult the vec3f by the given scalar */
80  t_vec3f * vec3f_mult(t_vec3f * dst, t_vec3f * vec, float scalar) {
81      if (dst == NULL) {
82          if ((dst = vec3f_new()) == NULL) {
83              return (NULL);
84          }
85      }
86      dst->x = vec->x * scalar;
87      dst->y = vec->y * scalar;
88      dst->z = vec->z * scalar;
89      return (dst);
90  }
91
92  t_vec3f * vec3f_mult3(t_vec3f * dst, t_vec3f * left, t_vec3f * right) {
93      if (dst == NULL) {
94          if ((dst = vec3f_new()) == NULL) {
95              return (NULL);
96          }
97      }
98      dst->x = left->x * right->x;
99      dst->y = left->y * right->y;
100     dst->z = left->z * right->z;
101     return (dst);
102 }
103
104 /** cross product */
105 t_vec3f * vec3f_cross(t_vec3f * dst, t_vec3f * left, t_vec3f * right) {
106     if (dst == NULL) {
107         if ((dst = vec3f_new()) == NULL) {
108             return (NULL);
109         }
110     }
111
112     dst->x = left->y * right->z - left->z * right->y;
113     dst->y = left->z * right->x - left->x * right->z;
114     dst->z = left->x * right->y - left->y * right->x;
115     return (dst);
116 }
117
118 /** scale product */
119 float vec3f_dot_product(t_vec3f * left, t_vec3f * right) {
```

```
120     return (left->x * right->x + left->y * right->y + left->z * right->z);
121 }
122 
123 /** length */
124 float vec3f_length_squared(t_vec3f * vec) {
125     return (vec3f_dot_product(vec, vec));
126 }
127 
128 float vec3f_length(t_vec3f * vec) {
129     return ((float)sqrt(vec3f_length_squared(vec)));
130 }
131 
132 /** normalize */
133 t_vec3f * vec3f_normalize(t_vec3f * dst, t_vec3f * vec) {
134 
135     if (dst == NULL) {
136         if ((dst = vec3f_new()) == NULL) {
137             return (NULL);
138         }
139     }
140 
141     float norm = 1 / vec3f_length(vec);
142     dst->x = vec->x * norm;
143     dst->y = vec->y * norm;
144     dst->z = vec->z * norm;
145     return (dst);
146 }
147 
148 /** negate */
149 t_vec3f * vec3f_negate(t_vec3f * dst, t_vec3f * src) {
150     if (dst == NULL) {
151         if ((dst = vec3f_new()) == NULL) {
152             return (NULL);
153         }
154     }
155     dst->x = -src->x;
156     dst->y = -src->y;
157     dst->z = -src->z;
158     return (dst);
159 }
160 
161 /** angle between two vec */
162 float vec3f_angle(t_vec3f * left, t_vec3f * right) {
163     float dls = vec3f_dot_product(left, right) / (vec3f_length(left) *
        vec3f_length(right));
164     if (dls < -1.0f) {
165         dls = -1.0f;
166     } else if (dls > 1.0f) {
167         dls = 1.0f;
168     }
169     return ((float)acos(dls));
170 }
171 
172 /** mix the two vectors */
```

```
173 t_vec3f * vec3f_mix(t_vec3f * dst, t_vec3f * left, t_vec3f * right, float
       ratio) {
174
175     if (dst == NULL) {
176         if ((dst = vec3f_new()) == NULL) {
177             return (NULL);
178         }
179     }
180
181     dst->x = left->x * ratio + right->x * (1 - ratio);
182     dst->y = left->y * ratio + right->y * (1 - ratio);
183     dst->z = left->z * ratio + right->z * (1 - ratio);
184     return (dst);
185 }
186
187 /** comparison */
188 int vec3f_equals(t_vec3f * left, t_vec3f * right) {
189     return (left == right || (left->x == right->x && left->y == right->y
           && left->z == right->z));
190 }
191
192 int vec3f_nequals(t_vec3f * left, t_vec3f * right) {
193     return (!vec3f_equals(left, right));
194 }
195
196 /** hash */
197 int vec3f_hash(t_vec3f * vec) {
198     return ((int)(vec->x * 73856093.0f) ^ (int)(vec->y * 19349663.0f) ^ (
           int)(vec->z * 83492791.0f));
199 }
200
201 /** round vec3f */
202 t_vec3f * vec3f_round(t_vec3f * dst, t_vec3f * vec, int decimals) {
203     static float powten[] = {1, 10, 100, 1000, 10000, 100000, 1000000,
           10000000, 100000000, 1000000000};
204
205     if (decimals < 0 || decimals >= 10) {
206         return (vec3f_set3(dst, vec));
207     }
208
209     if (dst == NULL) {
210         if ((dst = vec3f_new()) == NULL) {
211             return (NULL);
212         }
213     }
214
215     dst->x = roundf(powten[decimals] * vec->x) / powten[decimals];
216     dst->y = roundf(powten[decimals] * vec->y) / powten[decimals];
217     dst->z = roundf(powten[decimals] * vec->z) / powten[decimals];
218     return (dst);
219 }
220
221 /** to string: return a string allocated with malloc() */
222 char * vec3f_str(t_vec3f * vec) {
```

```
223     if (vec == NULL) {
224         return (strdup("vec3f(NULL)"));
225     }
226     char buffer[160];
227     sprintf(buffer, "vec3f(%f ; %f ; %f)", vec->x, vec->y, vec->z);
228     return (strdup(buffer));
229 }
```

maths/vec3f.h

```
1  /**
2   *  This file is part of https://github.com/toss-dev/C_maths
3   *
4   *  It is under a GNU GENERAL PUBLIC LICENSE
5   *
6   *  This library is still in development, so please, if you find any issue
        , let me know about it on github.com
7   *  PEREIRA Romain
8   */
9
10 #ifndef VEC3F_H
11 # define VEC3F_H
12
13 # include "cmaths.h"
14
15 typedef struct  s_vec3f {
16     union {
17         float x;
18         float r;
19         float pitch;
20     };
21
22     union {
23         float y;
24         float g;
25         float yaw;
26     };
27
28     union {
29         float z;
30         float b;
31         float roll;
32     };
33 }               t_vec3f;
34
35 /** create a new vec3f */
36 t_vec3f * vec3f_new(void);
37
38 /** delete the vec3f */
39 void vec3f_delete(t_vec3f * vec);
40
41 /** set the vec3f to 0 */
```

```
42 t_vec3f * vec3f_zero(t_vec3f * dst);
43
44 /** set the vec3f values */
45 t_vec3f * vec3f_set(t_vec3f * dst, float x, float y, float z);
46 t_vec3f * vec3f_set3(t_vec3f * dst, t_vec3f * vec);
47
48 /** add two vec3f */
49 t_vec3f * vec3f_add(t_vec3f * dst, t_vec3f * left, t_vec3f * right);
50
51 /** sub two vec3f */
52 t_vec3f * vec3f_sub(t_vec3f * dst, t_vec3f * left, t_vec3f * right);
53
54 /** mult the vec3f by the given scalar */
55 t_vec3f * vec3f_mult(t_vec3f * dst, t_vec3f * vec, float scalar);
56 t_vec3f * vec3f_mult3(t_vec3f * dst, t_vec3f * left, t_vec3f * right);
57
58 /** cross product */
59 t_vec3f * vec3f_cross(t_vec3f * dst, t_vec3f * left, t_vec3f * right);
60
61 /** scale product */
62 float vec3f_dot_product(t_vec3f * left, t_vec3f * right);
63
64 /** length */
65 float vec3f_length_squared(t_vec3f * vec);
66 float vec3f_length(t_vec3f * vec);
67
68 /** normalize */
69 t_vec3f * vec3f_normalize(t_vec3f * dst, t_vec3f * vec);
70
71 /** negate */
72 t_vec3f * vec3f_negate(t_vec3f * dst, t_vec3f * src);
73
74 /** angle between two vec */
75 float vec3f_angle(t_vec3f * left, t_vec3f * right);
76
77 /** mix the two vectors */
78 t_vec3f * vec3f_mix(t_vec3f * dst, t_vec3f * left, t_vec3f * right, float
      ratio);
79
80 /** comparison */
81 int vec3f_equals(t_vec3f * left, t_vec3f * right);
82 int vec3f_nequals(t_vec3f * left, t_vec3f * right);
83
84 /** hash */
85 int vec3f_hash(t_vec3f * vec);
86
87 /** round vec3f */
88 t_vec3f * vec3f_round(t_vec3f * dst, t_vec3f * vec, int decimals);
89
90 /** to string: return a string allocated with malloc() */
91 char * vec3f_str(t_vec3f * vec);
92
93 #endif
```

maths/vec3i.c

```c
/**
 *  This file is part of https://github.com/toss-dev/C_maths
 *
 *  It is under a GNU GENERAL PUBLIC LICENSE
 *
 *  This library is still in development, so please, if you find any issue, let me know about it on github.com
 *  PEREIRA Romain
 */

#include "vec3i.h"

/** create a new vec3i */
t_vec3i * vec3i_new(void) {
    return ((t_vec3i *)malloc(sizeof(t_vec3i)));
}

/** delete the vec3i */
void vec3i_delete(t_vec3i * vec) {
    free(vec);
}

/** set the vec3i to 0 */
t_vec3i * vec3i_zero(t_vec3i * dst) {
    if (dst == NULL) {
        if ((dst = vec3i_new()) == NULL) {
            return (NULL);
        }
    }
    memset(dst, 0, sizeof(t_vec3i));
    return (dst);
}

/** set the vec3i values */
t_vec3i * vec3i_set(t_vec3i * dst, int x, int y, int z) {
    if (dst == NULL) {
        if ((dst = vec3i_new()) == NULL) {
            return (NULL);
        }
    }
    dst->x = x;
    dst->y = y;
    dst->z = z;
    return (dst);
}

t_vec3i * vec3i_set3(t_vec3i * dst, t_vec3i * vec) {
    if (dst == vec) {
        return (dst);
    }
    return (vec3i_set(dst, vec->x, vec->y, vec->z));
}
```

```
52
53 /** add two vec3i */
54 t_vec3i * vec3i_add(t_vec3i * dst, t_vec3i * left, t_vec3i * right) {
55     if (dst == NULL) {
56         if ((dst = vec3i_new()) == NULL) {
57             return (NULL);
58         }
59     }
60     dst->x = left->x + right->x;
61     dst->y = left->y + right->y;
62     dst->z = left->z + right->z;
63     return (dst);
64 }
65
66 /** sub two vec3i */
67 t_vec3i * vec3i_sub(t_vec3i * dst, t_vec3i * left, t_vec3i * right) {
68     if (dst == NULL) {
69         if ((dst = vec3i_new()) == NULL) {
70             return (NULL);
71         }
72     }
73     dst->x = left->x - right->x;
74     dst->y = left->y - right->y;
75     dst->z = left->z - right->z;
76     return (dst);
77 }
78
79 /** mult the vec3i by the given scalar */
80 t_vec3i * vec3i_mult(t_vec3i * dst, t_vec3i * vec, int scalar) {
81     if (dst == NULL) {
82         if ((dst = vec3i_new()) == NULL) {
83             return (NULL);
84         }
85     }
86     dst->x = vec->x * scalar;
87     dst->y = vec->y * scalar;
88     dst->z = vec->z * scalar;
89     return (dst);
90 }
91
92 t_vec3i * vec3i_mult3(t_vec3i * dst, t_vec3i * left, t_vec3i * right) {
93     if (dst == NULL) {
94         if ((dst = vec3i_new()) == NULL) {
95             return (NULL);
96         }
97     }
98     dst->x = left->x * right->x;
99     dst->y = left->y * right->y;
100    dst->z = left->z * right->z;
101    return (dst);
102 }
103
104 /** cross product */
105 t_vec3i * vec3i_cross(t_vec3i * dst, t_vec3i * left, t_vec3i * right) {
```

```
106     if (dst == NULL) {
107         if ((dst = vec3i_new()) == NULL) {
108             return (NULL);
109         }
110     }
111 
112     dst->x = left->y * right->z - left->z * right->y;
113     dst->y = left->z * right->x - left->x * right->z;
114     dst->z = left->x * right->y - left->y * right->x;
115     return (dst);
116 }
117 
118 /** scale product */
119 int vec3i_dot_product(t_vec3i * left, t_vec3i * right) {
120     return (left->x * right->x + left->y * right->y + left->z * right->z);
121 }
122 
123 /** length */
124 int vec3i_length_squared(t_vec3i * vec) {
125     return (vec3i_dot_product(vec, vec));
126 }
127 
128 int vec3i_length(t_vec3i * vec) {
129     return ((int)sqrt(vec3i_length_squared(vec)));
130 }
131 
132 /** normalize */
133 t_vec3i * vec3i_normalize(t_vec3i * dst, t_vec3i * vec) {
134 
135     if (dst == NULL) {
136         if ((dst = vec3i_new()) == NULL) {
137             return (NULL);
138         }
139     }
140 
141     int norm = 1 / vec3i_length(vec);
142     dst->x = vec->x * norm;
143     dst->y = vec->y * norm;
144     dst->z = vec->z * norm;
145     return (dst);
146 }
147 
148 /** negate */
149 t_vec3i * vec3i_negate(t_vec3i * dst, t_vec3i * src) {
150     if (dst == NULL) {
151         if ((dst = vec3i_new()) == NULL) {
152             return (NULL);
153         }
154     }
155     dst->x = -src->x;
156     dst->y = -src->y;
157     dst->z = -src->z;
158     return (dst);
159 }
```

```
160
161 /** angle between two vec */
162 int vec3i_angle(t_vec3i * left, t_vec3i * right) {
163     int dls = vec3i_dot_product(left, right) / (vec3i_length(left) *
        vec3i_length(right));
164     if (dls < -1.0f) {
165         dls = -1.0f;
166     } else if (dls > 1.0f) {
167         dls = 1.0f;
168     }
169     return ((int)acos(dls));
170 }
171
172 /** mix the two vectors */
173 t_vec3i * vec3i_mix(t_vec3i * dst, t_vec3i * left, t_vec3i * right, int
    ratio) {
174
175     if (dst == NULL) {
176         if ((dst = vec3i_new()) == NULL) {
177             return (NULL);
178         }
179     }
180
181     dst->x = left->x * ratio + right->x * (1 - ratio);
182     dst->y = left->y * ratio + right->y * (1 - ratio);
183     dst->z = left->z * ratio + right->z * (1 - ratio);
184     return (dst);
185 }
186
187 /** comparison */
188 int vec3i_equals(t_vec3i * left, t_vec3i * right) {
189     return (left == right || (left->x == right->x && left->y == right->y
        && left->z == right->z));
190 }
191
192 int vec3i_nequals(t_vec3i * left, t_vec3i * right) {
193     return (!vec3i_equals(left, right));
194 }
195
196 /** hash */
197 int vec3i_hash(t_vec3i * vec) {
198     return ((vec->x * 73856093) ^ (vec->y * 19349663) ^ (vec->z *
        83492791));
199 }
200
201 /** to string: return a string allocated with malloc() */
202 char * vec3i_str(t_vec3i * vec) {
203     if (vec == NULL) {
204         return (strdup("vec3i(NULL)"));
205     }
206     char buffer[160];
207     sprintf(buffer, "vec3i(%d ; %d ; %d)", vec->x, vec->y, vec->z);
208     return (strdup(buffer));
209 }
```

maths/vec3i.h

```c
/**
 *  This file is part of https://github.com/toss-dev/C_maths
 *
 *  It is under a GNU GENERAL PUBLIC LICENSE
 *
 *  This library is still in development, so please, if you find any issue, let me know about it on github.com
 *  PEREIRA Romain
 */

#ifndef VEC3I_H
# define VEC3I_H

# include "cmaths.h"

typedef struct  s_vec3i {
    union {
        int x;
        int r;
        int pitch;
    };

    union {
        int y;
        int g;
        int yaw;
    };

    union {
        int z;
        int b;
        int roll;
    };
}               t_vec3i;

/** create a new vec3i */
t_vec3i * vec3i_new(void);

/** delete the vec3i */
void vec3i_delete(t_vec3i * vec);

/** set the vec3i to 0 */
t_vec3i * vec3i_zero(t_vec3i * dst);

/** set the vec3i values */
t_vec3i * vec3i_set(t_vec3i * dst, int x, int y, int z);
t_vec3i * vec3i_set3(t_vec3i * dst, t_vec3i * vec);

/** add two vec3i */
t_vec3i * vec3i_add(t_vec3i * dst, t_vec3i * left, t_vec3i * right);

/** sub two vec3i */
```

```
52 t_vec3i * vec3i_sub(t_vec3i * dst, t_vec3i * left, t_vec3i * right);
53
54 /** mult the vec3i by the given scalar */
55 t_vec3i * vec3i_mult(t_vec3i * dst, t_vec3i * vec, int scalar);
56 t_vec3i * vec3i_mult3(t_vec3i * dst, t_vec3i * left, t_vec3i * right);
57
58 /** cross product */
59 t_vec3i * vec3i_cross(t_vec3i * dst, t_vec3i * left, t_vec3i * right);
60
61 /** scale product */
62 int vec3i_dot_product(t_vec3i * left, t_vec3i * right);
63
64 /** length */
65 int vec3i_length_squared(t_vec3i * vec);
66 int vec3i_length(t_vec3i * vec);
67
68 /** normalize */
69 t_vec3i * vec3i_normalize(t_vec3i * dst, t_vec3i * vec);
70
71 /** negate */
72 t_vec3i * vec3i_negate(t_vec3i * dst, t_vec3i * src);
73
74 /** angle between two vec */
75 int vec3i_angle(t_vec3i * left, t_vec3i * right);
76
77 /** mix the two vectors */
78 t_vec3i * vec3i_mix(t_vec3i * dst, t_vec3i * left, t_vec3i * right, int
      ratio);
79
80 /** comparison */
81 int vec3i_equals(t_vec3i * left, t_vec3i * right);
82 int vec3i_nequals(t_vec3i * left, t_vec3i * right);
83
84 /** hash */
85 int vec3i_hash(t_vec3i * vec);
86
87 /** to string: return a string allocated with malloc() */
88 char * vec3i_str(t_vec3i * vec);
89
90 #endif
```

maths/vec4f.c

```
1 /**
2  *  This file is part of https://github.com/toss-dev/C_maths
3  *
4  *  It is under a GNU GENERAL PUBLIC LICENSE
5  *
6  *  This library is still in development, so please, if you find any issue
       , let me know about it on github.com
7  *  PEREIRA Romain
8  */
```

```c

#include "vec4f.h"

/** create a new vec4f */
t_vec4f * vec4f_new(void) {
    return ((t_vec4f *)malloc(sizeof(t_vec4f)));
}

/** delete the vec4f */
void vec4f_delete(t_vec4f * vec) {
    free(vec);
}

/** set the vec4f to 0 */
t_vec4f * vec4f_zero(t_vec4f * dst) {
    if (dst == NULL) {
        if ((dst = vec4f_new()) == NULL) {
            return (NULL);
        }
    }
    memset(dst, 0, sizeof(t_vec4f));
    return (dst);
}

/** set the vec4f values */
t_vec4f * vec4f_set(t_vec4f * dst, float x, float y, float z, float w) {
    if (dst == NULL) {
        if ((dst = vec4f_new()) == NULL) {
            return (NULL);
        }
    }
    dst->x = x;
    dst->y = y;
    dst->z = z;
    dst->w = w;
    return (dst);
}

t_vec4f * vec4f_set4(t_vec4f * dst, t_vec4f * vec) {
    if (dst == vec) {
        return (dst);
    }
    return (vec4f_set(dst, vec->x, vec->y, vec->z, vec->w));
}

/** add two vec4f */
t_vec4f * vec4f_add(t_vec4f * dst, t_vec4f * left, t_vec4f * right) {
    if (dst == NULL) {
        if ((dst = vec4f_new()) == NULL) {
            return (NULL);
        }
    }
    dst->x = left->x + right->x;
    dst->y = left->y + right->y;
```

```
63      dst->z = left->z + right->z;
64      dst->w = left->w + right->w;
65      return (dst);
66  }
67
68  /** sub two vec4f */
69  t_vec4f * vec4f_sub(t_vec4f * dst, t_vec4f * left, t_vec4f * right) {
70      if (dst == NULL) {
71          if ((dst = vec4f_new()) == NULL) {
72              return (NULL);
73          }
74      }
75      dst->x = left->x - right->x;
76      dst->y = left->y - right->y;
77      dst->z = left->z - right->z;
78      dst->w = left->w - right->w;
79      return (dst);
80  }
81
82  /** mult the vec4f by the given scalar */
83  t_vec4f * vec4f_mult(t_vec4f * dst, t_vec4f * vec, float scalar) {
84      if (dst == NULL) {
85          if ((dst = vec4f_new()) == NULL) {
86              return (NULL);
87          }
88      }
89      dst->x = vec->x * scalar;
90      dst->y = vec->y * scalar;
91      dst->z = vec->z * scalar;
92      dst->w = vec->w * scalar;
93      return (dst);
94  }
95
96  t_vec4f * vec4f_mult3(t_vec4f * dst, t_vec4f * left, t_vec4f * right) {
97      if (dst == NULL) {
98          if ((dst = vec4f_new()) == NULL) {
99              return (NULL);
100         }
101     }
102     dst->x = left->x * right->x;
103     dst->y = left->y * right->y;
104     dst->z = left->z * right->z;
105     dst->w = left->w * right->w;
106     return (dst);
107 }
108
109 /** scale product */
110 float vec4f_dot_product(t_vec4f * left, t_vec4f * right) {
111     return (left->x * right->x + left->y * right->y + left->z * right->z +
            left->w * right->w);
112 }
113
114 /** length */
115 float vec4f_length_squared(t_vec4f * vec) {
```

```
116     return (vec4f_dot_product(vec, vec));
117 }
118 
119 float vec4f_length(t_vec4f * vec) {
120     return ((float)sqrt(vec4f_length_squared(vec)));
121 }
122 
123 /** normalize */
124 t_vec4f * vec4f_normalize(t_vec4f * dst, t_vec4f * vec) {
125 
126     if (dst == NULL) {
127         if ((dst = vec4f_new()) == NULL) {
128             return (NULL);
129         }
130     }
131 
132     float norm = 1 / vec4f_length(vec);
133     dst->x = vec->x * norm;
134     dst->y = vec->y * norm;
135     dst->z = vec->z * norm;
136     dst->w = vec->w * norm;
137     return (dst);
138 }
139 
140 
141 /** negate */
142 t_vec4f * vec4f_negate(t_vec4f * dst, t_vec4f * src) {
143     if (dst == NULL) {
144         if ((dst = vec4f_new()) == NULL) {
145             return (NULL);
146         }
147     }
148     dst->x = -src->x;
149     dst->y = -src->y;
150     dst->z = -src->z;
151     dst->w = -src->w;
152     return (dst);
153 }
154 
155 /** angle between two vec */
156 float vec4f_angle(t_vec4f * left, t_vec4f * right) {
157     float dls = vec4f_dot_product(left, right) / (vec4f_length(left) *
        vec4f_length(right));
158     if (dls < -1.0f) {
159         dls = -1.0f;
160     } else if (dls > 1.0f) {
161         dls = 1.0f;
162     }
163     return ((float)acos(dls));
164 }
165 
166 /** mix the two vectors */
167 t_vec4f * vec4f_mix(t_vec4f * dst, t_vec4f * left, t_vec4f * right, float
    ratio) {
```

```
168
169     if (dst == NULL) {
170         if ((dst = vec4f_new()) == NULL) {
171             return (NULL);
172         }
173     }
174
175     dst->x = left->x * ratio + right->x * (1 - ratio);
176     dst->y = left->y * ratio + right->y * (1 - ratio);
177     dst->z = left->z * ratio + right->z * (1 - ratio);
178     dst->w = left->w * ratio + right->w * (1 - ratio);
179     return (dst);
180 }
181
182 /** comparison */
183 int vec4f_equals(t_vec4f * left, t_vec4f * right) {
184     return (left == right || (left->x == right->x && left->y == right->y
        && left->z == right->z && left->w == right->w));
185 }
186
187 int vec4f_nequals(t_vec4f * left, t_vec4f * right) {
188     return (!vec4f_equals(left, right));
189 }
190
191 /** hash */
192 int vec4f_hash(t_vec4f * vec) {
193     return ((int)(vec->x * 73856093.0f) ^ (int)(vec->y * 19349663.0f) ^ (
        int)(vec->z * 83492791.0f) ^ (int)(vec->w * 3539857.0f));
194 }
195
196 /** round vec4f */
197 t_vec4f * vec4f_round(t_vec4f * dst, t_vec4f * vec, int decimals) {
198     static float powten[] = {1, 10, 100, 1000, 10000, 100000, 1000000,
        10000000, 100000000, 1000000000};
199
200     if (decimals < 0 || decimals >= 10) {
201         return (vec4f_set4(dst, vec));
202     }
203
204     if (dst == NULL) {
205         if ((dst = vec4f_new()) == NULL) {
206             return (NULL);
207         }
208     }
209
210     dst->x = roundf(powten[decimals] * vec->x) / powten[decimals];
211     dst->y = roundf(powten[decimals] * vec->y) / powten[decimals];
212     dst->z = roundf(powten[decimals] * vec->z) / powten[decimals];
213     dst->w = roundf(powten[decimals] * vec->w) / powten[decimals];
214     return (dst);
215 }
216
217 /** to string: return a string allocated with malloc() */
218 char * vec4f_str(t_vec4f * vec) {
```

```
219     if (vec == NULL) {
220         return (strdup("vec4f(NULL)"));
221     }
222     char buffer[256];
223     sprintf(buffer, "vec4f(%f ; %f ; %f ; %f)", vec->x, vec->y, vec->z,
            vec->w);
224     return (strdup(buffer));
225 }
```

maths/vec4f.h

```
1  /**
2   *  This file is part of https://github.com/toss-dev/C_maths
3   *
4   *  It is under a GNU GENERAL PUBLIC LICENSE
5   *
6   *  This library is still in development, so please, if you find any issue
        , let me know about it on github.com
7   *  PEREIRA Romain
8   */
9
10 #ifndef VEC4F_H
11 # define VEC4F_H
12
13 # include "cmaths.h"
14
15 typedef struct  s_vec4f {
16     union {
17         float x;
18         float r;
19     };
20
21     union {
22         float y;
23         float g;
24     };
25
26     union {
27         float z;
28         float b;
29     };
30
31     union {
32         float w;
33         float a;
34     };
35 }               t_vec4f;
36
37 /** create a new vec4f */
38 t_vec4f * vec4f_new(void);
39
40 /** delete the vec4f */
```

```
41 void vec4f_delete(t_vec4f * vec);
42
43 /** set the vec4f to 0 */
44 t_vec4f * vec4f_zero(t_vec4f * dst);
45
46 /** set the vec4f values */
47 t_vec4f * vec4f_set(t_vec4f * dst, float x, float y, float z, float w);
48 t_vec4f * vec4f_set4(t_vec4f * dst, t_vec4f * vec);
49
50 /** add two vec4f */
51 t_vec4f * vec4f_add(t_vec4f * dst, t_vec4f * left, t_vec4f * right);
52
53 /** sub two vec4f */
54 t_vec4f * vec4f_sub(t_vec4f * dst, t_vec4f * left, t_vec4f * right);
55
56 /** mult the vec4f by the given scalar */
57 t_vec4f * vec4f_mult(t_vec4f * dst, t_vec4f * vec, float scalar);
58 t_vec4f * vec4f_mult2(t_vec4f * dst, t_vec4f * left, t_vec4f * right);
59
60 /** scale product */
61 float vec4f_dot_product(t_vec4f * left, t_vec4f * right);
62
63 /** length */
64 float vec4f_length_squared(t_vec4f * vec);
65 float vec4f_length(t_vec4f * vec);
66
67 /** normalize */
68 t_vec4f * vec4f_normalize(t_vec4f * dst, t_vec4f * vec);
69
70 /** negate */
71 t_vec4f * vec4f_negate(t_vec4f * dst, t_vec4f * src);
72
73 /** angle between two vec */
74 float vec4f_angle(t_vec4f * left, t_vec4f * right);
75
76 /** mix the two vectors */
77 t_vec4f * vec4f_mix(t_vec4f * dst, t_vec4f * left, t_vec4f * right, float
      ratio);
78
79 /** comparison */
80 int vec4f_equals(t_vec4f * left, t_vec4f * right);
81 int vec4f_nequals(t_vec4f * left, t_vec4f * right);
82
83 /** hash */
84 int vec4f_hash(t_vec4f * vec);
85
86 /** round vec4f */
87 t_vec4f * vec4f_round(t_vec4f * dst, t_vec4f * vec, int decimals);
88
89 /** to string: return a string allocated with malloc() */
90 char * vec4f_str(t_vec4f * vec);
91
92 #endif
```

maths/vec4i.c

```c
/**
 *  This file is part of https://github.com/toss-dev/C_maths
 *
 *  It is under a GNU GENERAL PUBLIC LICENSE
 *
 *  This library is still in development, so please, if you find any issue, let me know about it on github.com
 *  PEREIRA Romain
 */

#include "vec4i.h"

/** create a new vec4i */
t_vec4i * vec4i_new(void) {
    return ((t_vec4i *)malloc(sizeof(t_vec4i)));
}

/** delete the vec4i */
void vec4i_delete(t_vec4i * vec) {
    free(vec);
}

/** set the vec4i to 0 */
t_vec4i * vec4i_zero(t_vec4i * dst) {
    if (dst == NULL) {
        if ((dst = vec4i_new()) == NULL) {
            return (NULL);
        }
    }
    memset(dst, 0, sizeof(t_vec4i));
    return (dst);
}

/** set the vec4i values */
t_vec4i * vec4i_set(t_vec4i * dst, int x, int y, int z, int w) {
    if (dst == NULL) {
        if ((dst = vec4i_new()) == NULL) {
            return (NULL);
        }
    }
    dst->x = x;
    dst->y = y;
    dst->z = z;
    dst->w = w;
    return (dst);
}

t_vec4i * vec4i_set4(t_vec4i * dst, t_vec4i * vec) {
    if (dst == vec) {
        return (dst);
    }
    return (vec4i_set(dst, vec->x, vec->y, vec->z, vec->w));
```

```
52  }
53
54  /** add two vec4i */
55  t_vec4i * vec4i_add(t_vec4i * dst, t_vec4i * left, t_vec4i * right) {
56      if (dst == NULL) {
57          if ((dst = vec4i_new()) == NULL) {
58              return (NULL);
59          }
60      }
61      dst->x = left->x + right->x;
62      dst->y = left->y + right->y;
63      dst->z = left->z + right->z;
64      dst->w = left->w + right->w;
65      return (dst);
66  }
67
68  /** sub two vec4i */
69  t_vec4i * vec4i_sub(t_vec4i * dst, t_vec4i * left, t_vec4i * right) {
70      if (dst == NULL) {
71          if ((dst = vec4i_new()) == NULL) {
72              return (NULL);
73          }
74      }
75      dst->x = left->x - right->x;
76      dst->y = left->y - right->y;
77      dst->z = left->z - right->z;
78      dst->w = left->w - right->w;
79      return (dst);
80  }
81
82  /** mult the vec4i by the given scalar */
83  t_vec4i * vec4i_mult(t_vec4i * dst, t_vec4i * vec, int scalar) {
84      if (dst == NULL) {
85          if ((dst = vec4i_new()) == NULL) {
86              return (NULL);
87          }
88      }
89      dst->x = vec->x * scalar;
90      dst->y = vec->y * scalar;
91      dst->z = vec->z * scalar;
92      dst->w = vec->w * scalar;
93      return (dst);
94  }
95
96  t_vec4i * vec4i_mult3(t_vec4i * dst, t_vec4i * left, t_vec4i * right) {
97      if (dst == NULL) {
98          if ((dst = vec4i_new()) == NULL) {
99              return (NULL);
100         }
101     }
102     dst->x = left->x * right->x;
103     dst->y = left->y * right->y;
104     dst->z = left->z * right->z;
105     dst->w = left->w * right->w;
```

```
106     return (dst);
107 }
108 
109 /** scale product */
110 int vec4i_dot_product(t_vec4i * left, t_vec4i * right) {
111     return (left->x * right->x + left->y * right->y + left->z * right->z +
            left->w * right->w);
112 }
113 
114 /** length */
115 int vec4i_length_squared(t_vec4i * vec) {
116     return (vec4i_dot_product(vec, vec));
117 }
118 
119 int vec4i_length(t_vec4i * vec) {
120     return ((int)sqrt(vec4i_length_squared(vec)));
121 }
122 
123 /** normalize */
124 t_vec4i * vec4i_normalize(t_vec4i * dst, t_vec4i * vec) {
125 
126     if (dst == NULL) {
127         if ((dst = vec4i_new()) == NULL) {
128             return (NULL);
129         }
130     }
131 
132     int norm = 1 / vec4i_length(vec);
133     dst->x = vec->x * norm;
134     dst->y = vec->y * norm;
135     dst->z = vec->z * norm;
136     dst->w = vec->w * norm;
137     return (dst);
138 }
139 
140 
141 /** negate */
142 t_vec4i * vec4i_negate(t_vec4i * dst, t_vec4i * src) {
143     if (dst == NULL) {
144         if ((dst = vec4i_new()) == NULL) {
145             return (NULL);
146         }
147     }
148     dst->x = -src->x;
149     dst->y = -src->y;
150     dst->z = -src->z;
151     dst->w = -src->w;
152     return (dst);
153 }
154 
155 /** angle between two vec */
156 int vec4i_angle(t_vec4i * left, t_vec4i * right) {
157     int dls = vec4i_dot_product(left, right) / (vec4i_length(left) *
            vec4i_length(right));
```

```
158     if (dls < -1.0f) {
159         dls = -1.0f;
160     } else if (dls > 1.0f) {
161         dls = 1.0f;
162     }
163     return ((int)acos(dls));
164 }
165 
166 /** mix the two vectors */
167 t_vec4i * vec4i_mix(t_vec4i * dst, t_vec4i * left, t_vec4i * right, int
       ratio) {
168 
169     if (dst == NULL) {
170         if ((dst = vec4i_new()) == NULL) {
171             return (NULL);
172         }
173     }
174 
175     dst->x = left->x * ratio + right->x * (1 - ratio);
176     dst->y = left->y * ratio + right->y * (1 - ratio);
177     dst->z = left->z * ratio + right->z * (1 - ratio);
178     dst->w = left->w * ratio + right->w * (1 - ratio);
179     return (dst);
180 }
181 
182 /** comparison */
183 int vec4i_equals(t_vec4i * left, t_vec4i * right) {
184     return (left == right || (left->x == right->x && left->y == right->y
           && left->z == right->z && left->w == right->w));
185 }
186 
187 int vec4i_nequals(t_vec4i * left, t_vec4i * right) {
188     return (!vec4i_equals(left, right));
189 }
190 
191 /** hash */
192 int vec4i_hash(t_vec4i * vec) {
193     return ((vec->x * 73856093) ^ (vec->y * 19349663) ^ (vec->z *
           83492791) ^ (vec->w * 3539857));
194 }
195 
196 /** to string: return a string allocated with malloc() */
197 char * vec4i_str(t_vec4i * vec) {
198     if (vec == NULL) {
199         return (strdup("vec4i(NULL)"));
200     }
201     char buffer[256];
202     sprintf(buffer, "vec4i(%d ; %d ; %d ; %d)", vec->x, vec->y, vec->z,
           vec->w);
203     return (strdup(buffer));
204 }
```

maths/vec4i.h

```c
/**
 *  This file is part of https://github.com/toss-dev/C_maths
 *
 *  It is under a GNU GENERAL PUBLIC LICENSE
 *
 *  This library is still in development, so please, if you find any issue, let me know about it on github.com
 *  PEREIRA Romain
 */

#ifndef VEC4I_H
# define VEC4I_H

# include "cmaths.h"

typedef struct  s_vec4i {
    union {
        int x;
        int r;
    };

    union {
        int y;
        int g;
    };

    union {
        int z;
        int b;
    };

    union {
        int w;
        int a;
    };
}               t_vec4i;

/** create a new vec4i */
t_vec4i * vec4i_new(void);

/** delete the vec4i */
void vec4i_delete(t_vec4i * vec);

/** set the vec4i to 0 */
t_vec4i * vec4i_zero(t_vec4i * dst);

/** set the vec4i values */
t_vec4i * vec4i_set(t_vec4i * dst, int x, int y, int z, int w);
t_vec4i * vec4i_set4(t_vec4i * dst, t_vec4i * vec);

/** add two vec4i */
t_vec4i * vec4i_add(t_vec4i * dst, t_vec4i * left, t_vec4i * right);
```

```
52
53 /** sub two vec4i */
54 t_vec4i * vec4i_sub(t_vec4i * dst, t_vec4i * left, t_vec4i * right);
55
56 /** mult the vec4i by the given scalar */
57 t_vec4i * vec4i_mult(t_vec4i * dst, t_vec4i * vec, int scalar);
58 t_vec4i * vec4i_mult2(t_vec4i * dst, t_vec4i * left, t_vec4i * right);
59
60 /** scale product */
61 int vec4i_dot_product(t_vec4i * left, t_vec4i * right);
62
63 /** length */
64 int vec4i_length_squared(t_vec4i * vec);
65 int vec4i_length(t_vec4i * vec);
66
67 /** normalize */
68 t_vec4i * vec4i_normalize(t_vec4i * dst, t_vec4i * vec);
69
70 /** negate */
71 t_vec4i * vec4i_negate(t_vec4i * dst, t_vec4i * src);
72
73 /** angle between two vec */
74 int vec4i_angle(t_vec4i * left, t_vec4i * right);
75
76 /** mix the two vectors */
77 t_vec4i * vec4i_mix(t_vec4i * dst, t_vec4i * left, t_vec4i * right, int
      ratio);
78
79 /** comparison */
80 int vec4i_equals(t_vec4i * left, t_vec4i * right);
81 int vec4i_nequals(t_vec4i * left, t_vec4i * right);
82
83 /** hash */
84 int vec4i_hash(t_vec4i * vec);
85
86 /** to string: return a string allocated with malloc() */
87 char * vec4i_str(t_vec4i * vec);
88
89 #endif
```

maths/vecf.h

```
1 /**
2  *  This file is part of https://github.com/toss-dev/C_maths
3  *
4  *  It is under a GNU GENERAL PUBLIC LICENSE
5  *
6  *  This library is still in development, so please, if you find any issue
       , let me know about it on github.com
7  *  PEREIRA Romain
8  */
9
```

```
#ifndef VECF_H
# define VECF_H

# include "cmaths.h"
# include "vec2f.h"
# include "vec3f.h"
# include "vec4f.h"

#endif
```

maths/veci.h

```
/**
 *  This file is part of https://github.com/toss-dev/C_maths
 *
 *  It is under a GNU GENERAL PUBLIC LICENSE
 *
 *  This library is still in development, so please, if you find any issue
    , let me know about it on github.com
 *  PEREIRA Romain
 */

#ifndef VECI_H
# define VECI_H

# include "cmaths.h"
# include "vec2i.h"
# include "vec3i.h"
# include "vec4i.h"

#endif
```

datastructures/array_list.c

```
/**
 *  This file is part of https://github.com/toss-dev/C_data_structures
 *
 *  It is under a GNU GENERAL PUBLIC LICENSE
 *
 *  This library is still in development, so please, if you find any issue
    , let me know about it on github.com
 *  PEREIRA Romain
 */

#include "array_list.h"

/**
 * Create a new array list
 * nb : number of elements which the array can hold on first allocation
 * elem_size : size of an elements
```

```
16  *
17  * e.g: t_array_list array = array_list_new(16, sizeof(int));
18  */
19 t_array_list * array_list_new(unsigned long int nb, unsigned int elem_size
       ) {
20     t_array_list * array = (t_array_list *)malloc(sizeof(t_array_list));
21     if (array == NULL) {
22         return (NULL);
23     }
24
25     array->data = calloc(nb, elem_size);
26     array->capacity = nb;
27     array->elem_size = elem_size;
28     array->size = 0;
29     array->default_capacity = nb;
30     return (array);
31 }
32
33 /**
34  * resize array list
35  */
36 static void array_list_resize(t_array_list * array, unsigned size) {
37     array->data = realloc(array->data, size * array->elem_size);
38     array->capacity = size;
39     if (array->size > size) {
40         array->size = size;
41     }
42 }
43
44 static void array_list_expand(t_array_list * array) {
45     unsigned long int size = array->capacity * 2;
46     array_list_resize(array, size);
47 }
48
49 /**
50  *  Add an element at the end of the list
51  */
52 int array_list_add(t_array_list * array, void * data) {
53     if (array->size == array->capacity) {
54         array_list_expand(array);
55     }
56     memcpy(array->data + array->size * array->elem_size, data, array->
           elem_size);
57     array->size++;
58     return (array->size);
59 }
60
61 /**
62  *  Add every elements the end of the list
63  *  this function is faster than calling multiples 'array_list_add()'
64  *  so consider using it :)
65  */
66 void array_list_add_all(t_array_list * array, void * buffer, unsigned long
       int nb) {
```

```
67      unsigned int array_idx = array->size * array->elem_size;
68      while (nb) {
69          unsigned int copy_nb = array->capacity - array->size;
70          if (copy_nb > nb) {
71              copy_nb = nb;
72          }
73          if (copy_nb == 0) {
74              array_list_expand(array);
75              continue ;
76          }
77          unsigned int copy_size = copy_nb * array->elem_size;
78          memcpy(array->data + array_idx, buffer, copy_size);
79          nb -= copy_nb;
80          array->size += copy_nb;
81          buffer += copy_size;
82          array_idx += copy_size;
83      }
84  }
85
86  /**
87   * get item by index
88   */
89  void * array_list_get(t_array_list * array, unsigned int idx) {
90      return (array->data + idx * array->elem_size);
91  }
92
93  /**
94   *  remove the element at given index
95   */
96  void array_list_remove(t_array_list * array, unsigned int idx) {
97      if (array->size == 0 || idx >= array->size) {
98          return ;
99      }
100
101     unsigned int begin = idx * array->elem_size;
102     unsigned int end = (array->size - 1) * array->elem_size;
103     memmove(array->data + begin, array->data + begin + array->elem_size,
            end - begin);
104
105     array->size--;
106 }
107
108 /**
109  *  Clear the list (remove every data, and resize it to the default
        capacity)
110  */
111 void array_list_clear(t_array_list * array) {
112     array->size = 0;
113     array_list_resize(array, array->default_capacity);
114 }
115
116 /**
117  *  Delete DEFINETELY the list from memory
118  */
```

```
119 void array_list_delete(t_array_list * array) {
120     free(array->data);
121 }
122 
123 /**
124  *  Sort the array list using std quicksort algorythm
125  *
126  *  e.g:    t_array_list array = array_list_new(16, sizeof(char) * 2);
127  *          array_list_add(&array, "d");
128  *          array_list_add(&array, "a");
129  *          array_list_add(&array, "f");
130  *          [...]
131  *          array_list_sort(&array, (t_cmp_function)strcmp);
132  */
133 void array_list_sort(t_array_list * array, t_cmp_function cmpf) {
134     qsort(array->data, array->size, array->elem_size, cmpf);
135 }
136 
137 /**
138  *  Get raw data of your array list
139  *  (buffer of every data)
140  *  You should really not use this function
141  */
142 void * array_list_raw(t_array_list * array) {
143     return (array->data);
144 }
145 
146 
147 
148 //TESTS
149 /*
150 int main()
151 {
152     puts("\tARRAY LIST TESTS STARTED");
153     t_array_list * array = array_list_new(16, sizeof(char));
154 
155     unsigned long int i = 0;
156     unsigned long int max = 10000000;
157     unsigned long int t1;
158     unsigned long int t2;
159     unsigned long int t;
160 
161     MICROSEC(t1);
162     while (i < max) {
163         array_list_add(array, "a");
164         ++i;
165     }
166     MICROSEC(t2);
167     t = t2 - t1;
168 
169     printf("\t\t%-30s%lu\n", "elements added : ", max);
170     printf("\t\t%-30s%lu\n", "array number of elements : ", array->size);
171     printf("\t\t%-30s%lu\n", "array capacity now : ", array->capacity);
172     printf("\t\t%-30s%lf s\n", "time taken: ", t / 1000000.0f);
```

```
173
174     {
175         printf("\n\tIterating on array...\n");
176         MICROSEC(t1);
177         ARRAY_LIST_ITER_START(array, char *, item, iterator) {
178             char c = *item;
179             if (c != 'a') {
180                 fprintf(stderr, "ARRAY LIST ITER ERROR!!!!!");
181             }
182             (void)c;
183         }
184         ARRAY_LIST_ITER_END(array, char *, item, iterator);
185         MICROSEC(t2);
186         t = t2 - t1;
187         printf("\t\t%-30s%lf s\n", "time taken: ", t / 1000000.0f);
188     }
189
190     {
191         unsigned long int toremove = max / 1000;
192         printf("\n\tRemoving %lu last elements ...\n", toremove);
193         MICROSEC(t1);
194         while (toremove) {
195             array_list_remove(array, array->size - 1);
196             --toremove;
197         }
198         MICROSEC(t2);
199         t = t2 - t1;
200         printf("\t\t%-30s%lf s\n", "time taken: ", t / 1000000.0f);
201     }
202
203     {
204         unsigned long int toremove = max / 1000;
205         printf("\n\tRemoving %lu first elements ...\n", toremove);
206         MICROSEC(t1);
207         while (toremove) {
208             array_list_remove(array, 0);
209             --toremove;
210         }
211         MICROSEC(t2);
212         t = t2 - t1;
213         printf("\t\t%-30s%lf s\n", "time taken: ", t / 1000000.0f);
214     }
215
216
217
218     {
219         unsigned long int toremove = max / 1000;
220         unsigned long int middle = (max - toremove) / 2;
221         printf("\n\tRemoving %lu middle elements ...\n", toremove);
222         MICROSEC(t1);
223         while (toremove) {
224             array_list_remove(array, middle + toremove);
225             --toremove;
226         }
```

```
227         MICROSEC(t2);
228         t = t2 - t1;
229         printf("\t\t%-30s%lf s\n", "time taken: ", t / 1000000.0f);
230     }
231
232     {
233         printf("\n\tDeleting array...\n");
234         MICROSEC(t1);
235         array_list_delete(array);
236         MICROSEC(t2);
237         t = t2 - t1;
238         printf("\t\t%-30s%lu\n", "array number of elements : ", array->
                size);
239         printf("\t\t%-30s%lu\n", "array capacity now : ", array->capacity)
                ;
240         printf("\t\t%-30s%lf s\n", "time taken: ", t / 1000000.0f);
241     }
242
243     puts("\tARRAY LIST TESTS PASSED");
244     return (1);
245 }
246 */
```

datastructures/array_list.h

```
1  /**
2   *  This file is part of https://github.com/toss-dev/C_data_structures
3   *
4   *  It is under a GNU GENERAL PUBLIC LICENSE
5   *
6   *  This library is still in development, so please, if you find any issue
        , let me know about it on github.com
7   *  PEREIRA Romain
8   */
9
10 #ifndef ARRAY_LIST_H
11 # define ARRAY_LIST_H
12
13 # include "common.h"
14 # include <string.h>
15 # include <stdlib.h>
16 # include <stdio.h>
17
18 typedef struct  s_array_list {
19     char                * data;
20     unsigned long int   capacity;
21     unsigned long int   size;
22     unsigned int        elem_size;
23     unsigned int        default_capacity;
24 }               t_array_list;
25
26 /**
```

```
27  * Create a new array list
28  * nb : number of elements which the array can hold on first allocation
29  * elem_size : size of an elements
30  *
31  * e.g: t_array_list array = array_list_new(16, sizeof(int));
32  */
33 t_array_list * array_list_new(unsigned long int nb, unsigned int elem_size
        );
34
35 /**
36  *  Add an element at the end of the list
37  */
38 int array_list_add(t_array_list * array, void * data);
39
40 /**
41  *  Clear the list (remove every data, and resize it to the default
        capacity)
42  */
43 void array_list_clear(t_array_list * array);
44
45 /**
46  *  Delete DEFINETELY the list from memory
47  */
48 void array_list_delete(t_array_list * array);
49
50 /**
51  *  remove the element at given index
52  */
53 void array_list_remove(t_array_list * array, unsigned int idx);
54
55 /**
56  *  Sort the array list using std quicksort algorythm
57  *
58  *  e.g:    t_array_list array = array_list_new(16, sizeof(char) * 2);
59  *          array_list_push(&array, "d");
60  *          array_list_push(&array, "a");
61  *          array_list_push(&array, "f");
62  *          [...]
63  *          array_list_sort(&array, (t_cmp_function)strcmp);
64  */
65 void array_list_sort(t_array_list * array, t_cmp_function cmpf);
66
67 /**
68  *  Add every elements the end of the list
69  *  this function is faster than calling multiples 'array_list_add()'
70  *  so consider using it :)
71  */
72 void array_list_add_all(t_array_list * array, void * buffer, unsigned long
        int nb);
73
74 /**
75  *  Get raw data of your array list
76  *  (buffer of every data)
77  *  You should really not use this function
```

```
 */
void * array_list_raw(t_array_list * array);

/**
 * get item by index
 */
void * array_list_get(t_array_list * array, unsigned int idx);

/**
 * Iterate on the array list using a macro
 *
 * i.e :    t_array_list array;
 *
 *          [...] //push strings to the list
 *
 *          // print every string which the array list holds
 *          ARRAY_LIST_ITER_START(array, char *, str, i)
 *          {
 *              puts(str);
 *          }
 *          ARRAY_LIST_ITER_END(array, char *, str, i);
 */
# define ARRAY_LIST_ITER_START(L, T, X, I)\
 {\
    unsigned long int I = 0;\
    while (I < (L)->size) {\
        T X = ((T)(L)->data) + I;
# define ARRAY_LIST_ITER_END(L, T, X, I)\
       ++I;\
    }\
}

#endif
```

datastructures/btree.c

```
/**
 *  This file is part of https://github.com/toss-dev/C_data_structures
 *
 *  It is under a GNU GENERAL PUBLIC LICENSE
 *
 *  This library is still in development, so please, if you find any issue
    , let me know about it on github.com
 *  PEREIRA Romain
 */

#include "btree.h"

/**
 *  create a new binary tree
 */
t_btree * btree_new(t_cmp_function cmpf) {
```

```
16      t_btree * btree = (t_btree *)malloc(sizeof(t_btree));
17      if (btree == NULL) {
18          return (NULL);
19      }
20
21      btree->head = NULL;
22      btree->size = 0;
23      btree->cmpf = cmpf;
24      btree->values = array_list_new(16, sizeof(void *));
25      return (btree);
26  }
27
28  /** internal function to create a new node */
29  static t_btree_node *_btree_new_node(void * value) {
30      t_btree_node *node = (t_btree_node*)malloc(sizeof(t_btree_node));
31
32      if (node == NULL) {
33          return (NULL);
34      }
35
36      node->value = value;
37      node->left = NULL;
38      node->right = NULL;
39      return (node);
40  }
41
42  /** internal function : swip the head to the left */
43  static void btree_node_swip_left(t_btree_node ** node) {
44      t_btree_node *head = *node;
45      t_btree_node *right = head->right;
46
47      if (right == NULL) {
48          return ;
49      }
50
51      t_btree_node *tmp = right;
52      while (tmp->left != NULL) {
53          tmp = tmp->left;
54      }
55
56      tmp->left = head;
57      tmp->left->right = NULL;
58
59      *node = right;
60  }
61
62  /** internal function : swip the head to the right */
63  static void btree_node_swip_right(t_btree_node ** node)
64  {
65      t_btree_node *head = *node;
66      t_btree_node *left = head->left;
67
68      if (left == NULL) {
69          return ;
```

```
70      }
71
72      t_btree_node *tmp = left;
73      while (tmp->right != NULL) {
74          tmp = tmp->right;
75      }
76
77      tmp->right = head;
78      tmp->right->left = NULL;
79
80      *node = left;
81  }
82
83  static int _btree_insert(t_cmp_function cmpf, t_btree_node ** parent,
        t_btree_node ** node, void * value) {
84      if (*node == NULL) {
85          *node = _btree_new_node(value);
86          if (*node == NULL) {
87              return (0);
88          }
89          if (parent != NULL) {
90              if ((*parent)->left == NULL) {
91                  btree_node_swip_left(parent);
92              } else if ((*parent)->right == NULL) {
93                  btree_node_swip_right(parent);
94              }
95          }
96          return (1);
97      }
98
99      if (cmpf((*node)->value, value) < 0) {
100         return (_btree_insert(cmpf, node, &((*node)->right), value));
101     }
102     return (_btree_insert(cmpf, node, &((*node)->left), value));
103 }
104
105 /**
106  *  insert a value into the btree
107  */
108 void * btree_insert(t_btree * tree, void * value) {
109     if (_btree_insert(tree->cmpf, NULL, &(tree->head), value)) {
110         array_list_add(tree->values, &(value));
111         tree->size++;
112         return (value);
113     }
114     return (NULL);
115 }
116
117 /** intern function to remove a node and it childs */
118 static void _btree_delete_node(t_btree_node ** node) {
119     if (*node == NULL) {
120         return ;
121     }
122     _btree_delete_node(&((*node)->left));
```

```
123     _btree_delete_node(&((*node)->right));
124     free(*node);
125     *node = NULL;
126 }
127 
128 /**
129  *  delete the btree from the heap
130  */
131 void btree_delete(t_btree * btree) {
132     _btree_delete_node(&(btree->head));
133     array_list_delete(btree->values);
134     btree->size = 0;
135     btree->cmpf = 0;
136 }
137 
138 /** internal function to apply a function infix */
139 static void _btree_apply_infix(t_btree_node * node, t_function f) {
140     if (node->left != NULL) {
141         _btree_apply_infix(node->left, f);
142     }
143     f(node->value);
144     if (node->right != NULL) {
145         _btree_apply_infix(node->right, f);
146     }
147 }
148 
149 /**
150  *  call the function f to every value in the btree
151  *  (from left to head to right) (sort order)
152  */
153 void btree_apply_infix(t_btree * btree, t_function f) {
154     _btree_apply_infix(btree->head, f);
155 }
156 
157 /** internal function to apply a function suffix */
158 static void _btree_apply_suffix(t_btree_node * node, t_function f)
159 {
160     if (node->left != NULL) {
161         _btree_apply_suffix(node->left, f);
162     }
163     if (node->right != NULL) {
164         _btree_apply_suffix(node->right, f);
165     }
166     f(node->value);
167 }
168 
169 /**
170  *  call the function f to every value in the btree
171  *  (from (head to right) to (head to left))
172  */
173 void btree_apply_suffix(t_btree * btree, t_function f) {
174     _btree_apply_suffix(btree->head, f);
175 }
176 
```

```
177 /** internal function to apply a function prefix */
178 static void _btree_apply_prefix(t_btree_node * node, t_function f) {
179     f(node->value);
180     if (node->left != NULL) {
181         _btree_apply_prefix(node->left, f);
182     }
183     if (node->right != NULL) {
184         _btree_apply_prefix(node->right, f);
185     }
186 }
187
188 /**
189  *  call the function f to every value in the btree
190  *  (from (head to left) to (head to right))
191  */
192 void btree_apply_prefix(t_btree * btree, t_function f) {
193     _btree_apply_prefix(btree->head, f);
194 }
195
196 static t_btree_node *_btree_search(t_btree_node * node, void * valueref,
       t_cmp_function cmpf)    {
197     if (node == NULL) {
198         return (NULL);
199     }
200
201     int r = cmpf(node->value, valueref);
202     if (r == 0) {
203         return (node);
204     }
205
206     if (r > 0) {
207         return (_btree_search(node->left, valueref, cmpf));
208     }
209
210     return (_btree_search(node->right, valueref, cmpf));
211 }
212
213 /**
214  *  return  the item which match with the cmpf return value,
215  *          when comparing the node value and the given value reference
216  *          if the cmpf is NULL, the btree one is use
217  *          return NULL if the value isnt found
218  */
219 void * btree_get(t_btree * btree, void * valueref, t_cmp_function cmpf) {
220     t_btree_node * node = _btree_search(btree->head, valueref, cmpf);
221     return (node == NULL ? NULL : node->value);
222 }
223
224 /**
225  *  remove the given node from the btree
226  */
227 void * btree_remove_node(t_btree * tree, t_btree_node *node)
228 {
229     if (node == NULL) {
```

```
230         return (NULL);
231     }
232 
233     if (node->left == NULL) {
234         tree->head = node->right;
235     }
236     else if (node->right == NULL) {
237         tree->head = node->left;
238     } else {
239         t_btree_node *tmp = node->left;
240         while (tmp->right != NULL) {
241             tmp = tmp->right;
242         }
243         tmp->right = node->right;
244         tree->head = tmp;
245     }
246 
247     void * value = node->value;
248     free(node);
249     tree->size--;
250     return (value);
251 }
252 
253 /**
254  *  remove the node if the test with node's value and given value return 0
255  */
256 void * btree_remove_if(t_btree * tree, void * valueref, t_cmp_function
       cmpf) {
257     t_btree_node *node = _btree_search(tree->head, valueref, cmpf);
258     return (btree_remove_node(tree, node));
259 }
260 
261 /**
262  *  remove the node which contains the given value, and return it value
        address
263  */
264 void * btree_remove(t_btree * tree, void * valueref) {
265     return (btree_remove_if(tree, valueref, tree->cmpf));
266 }
267 
268 /*
269 int main() {
270     t_btree * btree = btree_new((t_cmpf)strcmp);
271 
272     btree_insert(btree, strdup("8"));
273     btree_insert(btree, strdup("E"));
274 
275     BTREE_ITER_START(btree, char *, str) {
276         printf("%s\n", str);
277     }
278     BTREE_ITER_END(&btree, char *, str)
279 
280     btree_delete(btree);
281 
```

```
282     return (0);
283 }
284 */
```

datastructures/btree.h

```
1  /**
2   *  This file is part of https://github.com/toss-dev/C_data_structures
3   *
4   *  It is under a GNU GENERAL PUBLIC LICENSE
5   *
6   *  This library is still in development, so please, if you find any issue
        , let me know about it on github.com
7   *  PEREIRA Romain
8   */
9
10 #ifndef BTREE_H
11 # define BTREE_H
12
13 # include "common.h"
14 # include "array_list.h"
15
16 typedef struct  s_btree_node {
17     void                * value;
18     struct s_btree_node * left;
19     struct s_btree_node * right;
20 }               t_btree_node;
21
22 typedef struct  s_btree {
23     t_array_list        * values;
24     t_btree_node        * head;
25     t_cmp_function      cmpf;
26     unsigned long int   size;
27 }               t_btree;
28
29 /**
30  *  create a new binary tree
31  */
32 t_btree * btree_new(t_cmp_function cmpf);
33
34 /**
35  *  delete the btree from the heap
36  */
37 void btree_delete(t_btree * btree);
38
39 /**
40  *  insert a value into the btree
41  */
42 void * btree_insert(t_btree * tree, void * data);
43
44 /**
45  *  return  the item which match with the cmpf return value,
```

```
 *           when comparing the node value and the given value reference
 *           if the cmpf is NULL, the btree one is use
 *           return NULL if the value isnt found
 */
void * btree_get(t_btree * btree, void * dataref, t_cmp_function cmpf);

/**
 *  remove the node if the test with node's value and given value return 0
 */
void * btree_remove_if(t_btree * tree, void * valueref, t_cmp_function
    cmpf);

/**
 *  remove the node which contains the given value, and return it value
    address
 */
void * btree_remove(t_btree * tree, void * valueref);

/**
 *  remove the given node from the btree
 */
void * btree_remove_node(t_btree * tree, t_btree_node * node);

/**
 *  Apply the function to every bin tree data, in the prefix, infix, or
    suffix order
 */
void btree_apply_prefix(t_btree * btree, t_function iterf);
void btree_apply_infix(t_btree * btree, t_function iterf);
void btree_apply_suffix(t_btree * btree, t_function iterf);

/**
 *  Iterate on the binary tree. Item are set in insertion order (not in
    sorted order)
 */

# define BTREE_ITER_START(B, T, V)\
{\
    ARRAY_LIST_ITER_START((B)->values, T, V, __btree_iterator) {
# define BTREE_ITER_END(B, T, V)        }\
    ARRAY_LIST_ITER_END((B)->values, T, V, __btree_iterator)\
}

#endif
```

datastructures/common.h

```
/**
 *  This file is part of https://github.com/toss-dev/C_data_structures
 *
 *  It is under a GNU GENERAL PUBLIC LICENSE
 *
```

```
 *  This library is still in development, so please, if you find any issue
    , let me know about it on github.com
 *  PEREIRA Romain
 */

#ifndef COMMON_H
# define COMMON_H

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

typedef void (*t_function)();
typedef int (*t_cmp_function) (void const * a, void const * b);
typedef unsigned long int (*t_hash_function) (void const * v);


typedef t_function t_f;
typedef t_cmp_function t_cmpf;
typedef t_hash_function t_hf;

# define MICROSEC(V)    {\
    struct timeval tv;\
    gettimeofday(&tv, NULL);\
    V = 1000000 * tv.tv_sec + tv.tv_usec;\
}


#endif
```

datastructures/hmap.c

```
/**
 *  This file is part of https://github.com/toss-dev/C_data_structures
 *
 *  It is under a GNU GENERAL PUBLIC LICENSE
 *
 *  This library is still in development, so please, if you find any issue
    , let me know about it on github.com
 *  PEREIRA Romain
 */

#include "hmap.h"

/**
 *  Create a new hashmap:
 *
 *  capacity : capacity of the hashmap (number of binary tree boxes in
    memory)
 *  hashf    : hash function to use on inserted elements
 *  keycmpf  : comparison function to use when searching a data
```

```
18  */
19 t_hmap * hmap_new(unsigned long int const capacity,
20         t_hash_function hashf, t_cmp_function keycmpf,
21         t_function keyfreef, t_function datafreef) {
22
23     // set the hmap capacity to the closest power of two
24     unsigned long int c = 1;
25     while (c < capacity) {
26         c = c << 1;
27     }
28
29     unsigned long int size = sizeof(t_list) * c;
30     void * values = malloc(size);
31     if (values == NULL) {
32         return (NULL);
33     }
34     memset(values, 0, size);
35
36     t_hmap * hmap = (t_hmap *)malloc(sizeof(t_hmap));
37     if (hmap == NULL) {
38         free(values);
39         return (NULL);
40     }
41
42     hmap->values = values;
43     hmap->capacity = capacity;
44     hmap->size = 0;
45     hmap->hashf = hashf;
46     hmap->keycmpf = keycmpf;
47     hmap->datafreef = datafreef;
48     hmap->keyfreef = keyfreef;
49
50     return (hmap);
51 }
52
53 /**
54  *  Delete the hashmap from the heap
55  *
56  *  hmap    :   hash map
57  *  freef   :   function which will be called on node data and node key on
        node being freed.
58  i.e :  'NULL' if data shouldnt be free, 'free' if the data was allocated
        with a malloc,
59  'myfree' if this is structure which contains multiple allocated fields
60  */
61 void hmap_delete(t_hmap * hmap) {
62     unsigned long int i = 0;
63     while (i < hmap->capacity) {
64         t_list * lst = hmap->values + i;
65         //if the list has been initialized
66         if (lst->head) {
67             LIST_ITER_START(lst, t_hmap_node *, node) {
68                 if (hmap->datafreef) {
69                     hmap->datafreef(node->data);
```

```
70                  }
71
72                  if (hmap->keyfreef) {
73                      hmap->keyfreef(node->key);
74                  }
75              }
76              LIST_ITER_END(lst, t_hmap_node *, node)
77              list_delete(lst);
78          }
79          ++i;
80      }
81  }
82
83  /**
84   *  Insert a value into the hashmap:
85   *
86   *  map  : hmap
87   *  data : value to insert
88   *  key  : key reference for this data
89   *  size : size of the data (i.e, 'sizeof(t_data_structure)', 'strlen(str)
         + 1')
90   *
91   *  return the given data if it was inserted properly, NULL elseway
92   */
93  void const * hmap_insert(t_hmap * hmap, void const * data, void const *
       key)
94  {
95      unsigned long int hash = hmap->hashf(key); //get the hash for this key
96      unsigned long int addr = hash & (hmap->capacity - 1); //get the array
           list from the hash
97
98      t_hmap_node node = {hash, data, key}; //set the node buffer
99
100     t_list * lst = hmap->values + addr; //get the list from it address
101     //if the list hasnt already been initialized
102     if (lst->head == NULL) {
103         list_init(lst); //initialize it
104     }
105     list_add(lst, &node, sizeof(t_hmap_node)); //add the node to the list
106
107     hmap->size++;
108     return (data); //return the data
109 }
110
111 /**
112  *  Get data from the hashmap
113  *
114  *  hmap : hash map
115  *  key  : the node's key to find
116  */
117 void * hmap_get(t_hmap * hmap, void const * key) {
118     unsigned long int hash = hmap->hashf(key); //get the hash for this key
119     unsigned long int addr = hash & (hmap->capacity - 1); //get the lst
           list from the hash
```

```
120
121     t_list * lst = hmap->values + addr; //list of collision for this key
            hash
122
123     if (lst->size == 0) {
124         return (NULL);
125     }
126
127     //so compare the exact key to find the wanted data
128     LIST_ITER_START(lst, t_hmap_node *, node) {
129         if (hmap->keycmpf(key, node->key) == 0) {
130             return ((void *)node->data);
131         }
132     }
133     LIST_ITER_END(lst, t_hmap_node *, node)
134     return (NULL);
135 }
136
137 /**
138  *  Remove the data pointer from the hash map
139  *  return 1 if the element was removed, 0 elseway
140  *  hmap : the hash map
141  *  data : pointer to the data
142  */
143 int hmap_remove_data(t_hmap * hmap, void const * data) {
144     unsigned long int i = 0;
145     while (i < hmap->capacity) {
146         t_list * lst = hmap->values + i;
147         LIST_ITER_START(lst, t_hmap_node *, node) {
148             if (node->data == data) {
149                 //__node is the current LIST_ITER_START node of the linked
                        list
150                 list_remove_node(lst, __node);
151                 hmap->size--;
152
153                 if (hmap->datafreef) {
154                     hmap->datafreef(node->key);
155                 }
156
157                 if (hmap->keyfreef) {
158                     hmap->keyfreef(node->key);
159                 }
160
161                 return (1);
162             }
163         }
164         LIST_ITER_END(array, t_hmap_node *, node)
165         ++i;
166     }
167     return (0);
168 }
169
170 /**
171  *  Remove the data which match with the given key from the hash map
```

```
172  *  return 1 if the element was removed, 0 elseway
173  *
174  *  hmap : the hash map
175  *  key  : pointer to the key
176  */
177 int hmap_remove_key(t_hmap * hmap, void const * key) {
178     unsigned long int hash = hmap->hashf(key); //get the hash for this key
179     unsigned long int addr = hash & (hmap->capacity - 1); //get the array
        list from the hash
180
181     t_list * lst = hmap->values + addr; //lst of collision for this key
        hash
182
183     if (lst->size == 0) {
184         return (0);
185     }
186
187     //so compare the exact key to find the wanted data
188     LIST_ITER_START(lst, t_hmap_node *, node) {
189         if (hmap->keycmpf(key, node->key) == 0) {
190             //__node is the current LIST_ITER_START node of the linked
                list
191             list_remove_node(lst, __node);
192             hmap->size--;
193
194             if (hmap->datafreef) {
195                 hmap->datafreef(node->key);
196             }
197
198             if (hmap->keyfreef) {
199                 hmap->keyfreef(node->key);
200             }
201
202             return (1);
203         }
204     }
205     LIST_ITER_END(array, t_hmap_node *, node)
206     return (0);
207 }
208
209 /**
210  *  default string hash function
211  */
212 unsigned long int strhash(char const * str) {
213     if (str == NULL) {
214         return (0);
215     }
216
217     unsigned long int hash = 5381;
218     int c;
219     while ((c = *str) != '\0') {
220         hash = ((hash << 5) + hash) + c;
221         str++;
222     }
```

```
223     return (hash);
224 }
225 
226 /**
227  *  Default hash for an integer
228  */
229 unsigned long int inthash(int const value) {
230     return (value);
231 }
232 
233 /*
234 int main() {
235     t_hmap hmap = hmap_new(1024, (t_hf)strhash, (t_cmpf)strcmp, free, free
        );
236     hmap_insert(&hmap, strdup("Hello world"), strdup("ima key"));
237     hmap_insert(&hmap, strdup("abc"), strdup("ima key2"));
238     hmap_insert(&hmap, strdup("def"), strdup("ima key3"));
239     hmap_insert(&hmap, strdup("collision1"), strdup("ima key collision"));
240     hmap_insert(&hmap, strdup("collision2"), strdup("ima key collision"));
241 
242     char *value = hmap_get(&hmap, "ima key");
243 
244     printf("{%s}\n", value);
245     printf("other values are:\n");
246 
247     HMAP_ITER_START(&hmap, char *, str) {
248         printf("{%s}\n", str);
249     }
250     HMAP_ITER_END(&hmap, char *, str)
251 
252     hmap_delete(&hmap);
253     return (0);
254 }
255 
256 */
```

datastructures/hmap.h

```
1  /**
2   *  This file is part of https://github.com/toss-dev/C_data_structures
3   *
4   *  It is under a GNU GENERAL PUBLIC LICENSE
5   *
6   *  This library is still in development, so please, if you find any issue
        , let me know about it on github.com
7   *  PEREIRA Romain
8   */
9 
10 #ifndef HMAP_H
11 # define HMAP_H
12 
13 # include "common.h"
```

```c
# include "linked_list.h"

/**
 *  Generic hash map implementation in C89:
 *
 *  ABOUT THE IMPLEMENTATION:
 *      - given pointer address are saved for values. No copy their data are done. (same for keys)
 *      - const where used where on constant data (well...), so you dont mess up the hash map :)
 *      - an array of linked list is used to handle collisions
 *
 *
 *  example for a string hashmap:
 *
 *      t_hmap map = hmap_new(1024, (t_hf)strhash, (t_cmpf)strcmp);
 *      hmap_insert(&map, strdup("hello world"), strdup("ima key"), strlen("Hello world") + 1);
 *      char *helloworld = hmap_get(&map, "ima key"); //now contains "Hello world"
 */

typedef struct  s_hmap_node {
    unsigned long int const hash; //hash of the key
    void const * data; //the data holds
    void const * key; //the key used
}               t_hmap_node;

typedef struct  s_hmap {
    t_list * values; //a buffer of value holders (to handle collision)
    unsigned long int capacity; //number of lists
    unsigned long int size; //number of value set
    t_hash_function hashf; //hash function
    t_cmp_function keycmpf; //key comparison function, where node keys are sent as parameters
    t_function datafreef; //function call when a data object should be freed
    t_function keyfreef; //function called when a key should be freed
}               t_hmap;

/**
 *  Create a new hashmap:
 *
 *  capacity : capacity of the hashmap (number of lists boxes in memory)
 *  hashf    : hash function to use on inserted elements
 *  cmpf     : comparison function to use when searching a data
 */
t_hmap * hmap_new(unsigned long int const capacity, t_hash_function hashf, t_cmp_function keycmpf, t_function keyfreef, t_function datafreef);

/**
 *  Delete the hashmap from the heap
 *
 *  hmap : hash map
```

```
61   *  datafreef : function which will be called on node data before the node
         being freed.
62   *              i.e :   'NULL' if data shouldnt be free, 'free' if the
       data was allocated with a malloc,
63   *                      'myfree' if this is structure which contains
       multiple allocated fields ...
64   *  keyfreef : same for the node key
65   */
66  void hmap_delete(t_hmap * hmap);
67
68  /**
69   *  Insert a value into the hashmap:
70   *
71   *  map  : hmap
72   *  data : value to insert
73   *  key  : key reference for this data
74   *  size : size of the data (i.e, 'sizeof(t_data_structure)', 'strlen(str)
         + 1')
75   *
76   *  return the given data if it was inserted properly, NULL elseway
77   */
78  void const * hmap_insert(t_hmap * hmap, void const * data, void const *
       key);
79
80  /**
81   *  Get data from the hashmap
82   *
83   *  hmap : hash map
84   *  key  : the node's key to find
85   */
86  void * hmap_get(t_hmap * hmap, void const * key);
87
88  /**
89   *  Remove the data pointer from the hash map
90   *  return 1 if the element was removed, 0 elseway
91   *  hmap : the hash map
92   *  data : pointer to the data
93   */
94  int hmap_remove_data(t_hmap * hmap, void const * data);
95
96  /**
97   *  Remove the data which match with the given key from the hash map
98   *  return 1 if the element was removed, 0 elseway
99   *
100  *  hmap : the hash map
101  *  key  : pointer to the key
102  */
103 int hmap_remove_key(t_hmap * hmap, void const * key);
104
105 /**
106  *  Some simple builtin hashes functions, useful for tests.
107  *
108  *  String hash is based on : http://www.cse.yorku.ca/~oz/hash.html
109  */
```

```c
unsigned long int strhash(char const * str);
unsigned long int inthash(int const value);

/**
 *  Macro to iterate fastly though to hash map
 *
 *  i.e:
 *      HMAP_ITER_START(hmap, char *, str) {
 *          puts(str);
 *      }
 *      HMAP_ITER_END(hmap, char *, str)
 */
# define HMAP_ITER_START(H, T, V)\
{\
    unsigned long int i = 0;\
    while (i < (H)->capacity) {\
        t_list * lst = (H)->values + i;\
        if (lst != NULL && lst->head != NULL) {\
            LIST_ITER_START(lst, t_hmap_node *, node) {\
                T V = (T)(node->data);
# define HMAP_ITER_END(H, T, V)\
            }\
            LIST_ITER_END(lst, t_hmap_node *, node)\
        }\
        ++i;\
    }\
}

#endif
```

datastructures/linked_list.c

```c
/**
 *  This file is part of https://github.com/toss-dev/C_data_structures
 *
 *  It is under a GNU GENERAL PUBLIC LICENSE
 *
 *  This library is still in development, so please, if you find any issue
    , let me know about it on github.com
 *  PEREIRA Romain
 */

#include "linked_list.h"

int list_init(t_list * list) {
    list->head = (t_list_node*)malloc(sizeof(t_list_node));
    if (list->head == NULL) {
        return (0);
    }
    list->head->next = list->head;
    list->head->prev = list->head;
    list->size = 0;
```

```
20      return (1);
21  }
22
23  /**
24   * Create a new linked list
25   */
26  t_list * list_new(void) {
27      t_list * list = (t_list *) malloc(sizeof(t_list));
28      if (list == NULL) {
29          return (NULL);
30      }
31
32      if (!list_init(list)) {
33          free(list);
34          return (NULL);
35      }
36
37      return (list);
38  }
39
40  /**
41   *  Add an element at the end of the list
42   */
43  void * list_add(t_list * lst, void const *content, unsigned int
        content_size)
44  {
45      t_list_node *node = (t_list_node*)malloc(sizeof(t_list_node) +
            content_size);
46      if (node == NULL) {
47          return (NULL);
48      }
49      memcpy(node + 1, content, content_size);
50
51      t_list_node *tmp = lst->head->prev;
52
53      lst->head->prev = node;
54      tmp->next = node;
55
56      node->prev = tmp;
57      node->next = lst->head;
58
59      lst->size++;
60
61      return (node + 1);
62  }
63
64  /**
65   * Add an element in head of the list
66   */
67  void * list_addfront(t_list * lst, void const *content, unsigned int
        content_size) {
68      t_list_node * node = (t_list_node *) malloc(sizeof(t_list_node) +
            content_size);
69      if (node == NULL) {
```

```
70          return (NULL);
71      }
72      memcpy(node + 1, content, content_size);
73
74      t_list_node *tmp = lst->head->next;
75
76      lst->head->next = node;
77      tmp->prev = node;
78
79      node->prev = lst->head;
80      node->next = tmp;
81
82      lst->size++;
83
84      return (node + 1);
85  }
86
87  /**
88   *  remove the given node from the list
89   */
90  void list_remove_node(t_list * lst, t_list_node *node) {
91      if (node->prev) {
92          node->prev->next = node->next;
93      }
94      if (node->next) {
95          node->next->prev = node->prev;
96      }
97
98      node->next = NULL;
99      node->prev = NULL;
100     free(node);
101     lst->size--;
102 }
103
104 /**
105  * Remove first / last element of the list. Return 1 if it was removed, 0
        else
106  */
107 int list_remove_first(t_list * lst) {
108     if (lst->size == 0) {
109         return (0);
110     }
111     list_remove_node(lst, lst->head->next);
112     return (1);
113 }
114
115 int list_remove_last(t_list * lst) {
116     if (lst->size == 0) {
117         return (0);
118     }
119     list_remove_node(lst, lst->head->prev);
120     return (1);
121 }
122
```

```
123 /**
124  *  remove list head
125  */
126 void * list_pop(t_list * lst) {
127     if (lst->size == 0) {
128         return (NULL);
129     }
130 
131     void * data = lst->head->next + 1;
132     if (lst->size > 0)
133     {
134         list_remove_first(lst);
135     }
136     return (data);
137 }
138 
139 /** return content at the begining of the list */
140 void * list_head(t_list * lst) {
141     if (lst->size > 0) {
142         return ((void*)lst->head->next + 1);
143     }
144     return (NULL);
145 }
146 
147 
148 /** remove if the comparison return elements are equals (works like strcmp
    ) */
149 int list_remove(t_list * lst, t_cmp_function cmpf, void * cmpd) {
150     t_list_node *node;
151 
152     node = lst->head->next;
153     while (node != lst->head) {
154         if (cmpf(node + 1, cmpd) == 0) {
155             list_remove_node(lst, node);
156             return (1);
157         }
158         node = node->next;
159     }
160     return (0);
161 }
162 
163 /**
164  *  Return the list node data which match with the given comparison
    function
165  *  and reference data. (cmpf should acts like 'strcmp()')
166  */
167 void * list_get(t_list * lst, t_cmp_function cmpf, void * cmpd) {
168     if (lst->size == 0) {
169         return (NULL);
170     }
171 
172     if (cmpf(lst->head + 1, cmpf) == 0) {
173         return (lst->head);
174     }
```

```
175
176     t_list_node *node = lst->head->next;
177     while (node != lst->head) {
178         if (cmpf(node + 1, cmpd) == 0) {
179             return (node + 1);
180         }
181         node = node->next;
182     }
183
184     return (NULL);
185 }
186
187 /**
188  * Remove the node which datas match with the given comparison function
189  * and the given data reference
190  */
191 void list_delete(t_list * lst) {
192     if (lst->size == 0) {
193         goto end;
194     }
195
196     list_clear(lst);
197
198 end:
199     lst->head = NULL;
200     lst->size = 0;
201 }
202
203 /**
204  *  clear the list : remove every nodes
205  */
206 void list_clear(t_list * lst) {
207
208     t_list_node * node = lst->head->next;
209     while (node != lst->head) {
210         t_list_node *next = node->next;
211         free(node);
212         node = next;
213     }
214
215     free(lst->head);
216     list_init(lst);
217 }
218
219 /**
220  * Return a buffer which holds pointers to every elements of the list,
        allocated with 'malloc()'
221  */
222 void * list_buffer(t_list * lst) {
223     void ** buffer = (void**)malloc(sizeof(void*) * (lst->size + 1));
224     if (buffer == NULL) {
225         return (NULL);
226     }
227
```

```
228     t_list_node *node = lst->head->next;
229     unsigned int i = 0;
230 
231     while (node != lst->head) {
232         buffer[i] = (void*)(node + 1);
233         ++i;
234         node = node->next;
235     }
236 
237     buffer[i] = NULL;
238     return ((void*)buffer);
239 }
240 
241 /**
242  *  iterate the function to every node content of the list
243  */
244 void list_iterate(t_list * lst, t_function f)
245 {
246     LIST_ITER_START(lst, void * , content) {
247         f(content);
248     }
249     LIST_ITER_END(lst, void * , content)
250 }
251 
252 
253 /*
254 int main()
255 {
256     puts("\tLINKED LIST TESTS STARTED");
257 
258     t_list * lst = list_new();
259 
260     unsigned long int i = 0;
261     unsigned long int max = 10000000;
262 
263     unsigned long int t1;
264     unsigned long int t2;
265     unsigned long int t;
266 
267     MICROSEC(t1);
268     while (i < max) {
269         list_add(lst, strdup("a"), 2);
270         ++i;
271     }
272     MICROSEC(t2);
273     t = t2 - t1;
274 
275     printf("\t%-30s%lu\n", "elements pushed : ", max);
276     printf("\t%-30s%lu\n", "list number of elements : ", lst->size);
277     printf("\t%-30s%lf s\n", "time taken: ", t / 1000000.0f);
278 
279 
280     list_iterate(lst, free);
281     list_delete(lst);
```

```
282
283     puts("\tLINKED LIST TESTS PASSED");
284
285     return (0);
286 }
287 */
```

datastructures/linked_list.h

```
1  /**
2   *  This file is part of https://github.com/toss-dev/C_data_structures
3   *
4   *  It is under a GNU GENERAL PUBLIC LICENSE
5   *
6   *  This library is still in development, so please, if you find any issue
        , let me know about it on github.com
7   *  PEREIRA Romain
8   */
9
10 #ifndef LINKED_LIST_H
11 # define LINKED_LIST_H
12
13 # include <stdlib.h>
14 # include <string.h>
15 # include <unistd.h>
16 # include "common.h"
17
18 typedef struct  s_list_node {
19     struct s_list_node * next;
20     struct s_list_node * prev;
21 }               t_list_node;
22
23 typedef struct  s_list {
24     t_list_node         * head;
25     unsigned long int   size;
26 }               t_list;
27
28 /** initialize the given list */
29 int list_init(t_list * list);
30
31 /**
32  * Create a new linked list
33  */
34 t_list * list_new(void);
35
36 /**
37  *  Add an element at the end of the list
38  */
39 void * list_add(t_list * lst, void const * content, unsigned int
       content_size);
40
41 /**
```

```
42  * Add an element in head of the list
43  */
44 void * list_addfront(t_list * lst, void const * content, unsigned int
       content_size);
45
46 /**
47  *  Return the list node data which match with the given comparison
       function
48  *  and reference data. (cmpf should acts like 'strcmp()')
49  */
50 void * list_get(t_list * lst, t_cmp_function cmpf, void * cmpd);
51
52 /**
53  * Remove the node which datas match with the given comparison function
54  * and the given data reference
55  */
56 int list_remove(t_list * lst, t_cmp_function cmpf, void * cmpref);
57
58 /**
59  *  remove the given node from the list
60  */
61 void list_remove_node(t_list * lst, t_list_node *node);
62
63 /**
64  * Remove first / last element of the list. Return 1 if it was removed, 0
       else
65  */
66 int list_remove_first(t_list * lst);
67 int list_remove_last(t_list * lst);
68
69 /**
70  * Remove the first element of the list, and return it data
71  */
72 void * list_pop(t_list * lst);
73
74 /**
75  *  Return the first element of the list
76  */
77 void * list_head(t_list * lst);
78
79 /**
80  * Clear the list (remove every node)
81  */
82 void list_clear(t_list * lst);
83
84 /**
85  *  remove the list for the heap
86  */
87 void list_delete(t_list * lst);
88
89 /**
90  *  iterate the function to every node content of the list
91  */
92 void list_iterate(t_list * lst, t_function f);
```

```
93
94
95  /**
96   * Return a buffer which holds pointers to every elements of the list,
        allocated with 'malloc()'
97   */
98  void * list_buffer(t_list * lst);
99
100
101 /** iterate on the list using a macro (optimized) */
102 # define LIST_ITER_START(L, T, V)\
103 {\
104     if (L != NULL && L->head != NULL) {\
105         t_list_node *__node = L->head->next;\
106         while (__node != L->head) {\
107             T V = (T)(__node + 1);
108 # define LIST_ITER_END(L, T, V) \
109             __node = __node->next; \
110         }\
111     }\
112 }
113
114
115
116
117 //ABOVE FUNCTIONS ARENT IMPLEMENTED YET:
118
119 /**
120  *  write the list to a file descriptor
121  */
122 int     list_to_fd(t_list *list, int fd);
123
124 /**
125  *  read and return a list from the given file descriptor
126  */
127 t_list  list_from_fd(int fd);
128
129
130
131 #endif
```