

```
1  /*
2  The SDL2 verison of the collision performance testing code for Games  ↗
   Programming
3
4  My compiler settings to including SDL2 lib and set optimisation level
5  g++ -std=c++11 spike26_sdl2.cpp -o "spike26_sdl2" -L/usr/local/lib -lSDL2  ↗
   -I/usr/local/include/SDL2 -D_THREAD_SAFE -O3
6
7  Try different optimization (letter "O" then number) levels
8  -O0 level zero
9  -O3 level 3
10
11 Look for TODO's and comments for ideas of what things to change.
12
13 Note: You REALLY need to
14 - increase the test time,
15 - turn the rendering OFF, and
16 - turn optimisation OFF
17 to see biggest difference! (Then explain why!)
18
19 There are lots of ugly code features.
20 Enjoy! - Oh i did :)
21
22 Clinton Woodward <cwoodward@swin.edu.au>
23 Updated 2017-10-26 */
24
25 #include <iostream>
26 #include <ctime>
27 #include "SDL.h"
28
29 const int SCREEN_WIDTH = 800;
30 const int SCREEN_HEIGHT = 600;
31 const int SCREEN_BPP = 32;
32
33 const int BOX_WIDTH = 50;
34 const int BOX_HEIGHT = 50;
35 const int BOX_SPEED = 10;
36 const int BOX_COUNT = 100;
37
38 const int TEST_TIME = 15 * 1000; // ie, 3*1000 = 3 seconds
39
40 enum BoxState { CONTACT_NO, CONTACT_YES };
41
42 struct CrashBox {
43     int x, y; // pos
44     int dx, dy; // vel
45     int w, h; // size
46     BoxState state;
47 };
```

```
48
49 // Global variables. (Apparently evil.)
50 CrashBox boxes[BOX_COUNT];
51
52 void (*crash_test_all_ptr)(); // global function pointer!
53
54
55 //----- ↗
56
57 void init_boxes()
58 {
59     // seed value - Set explicitly if you want repeatable results!!
60     srand((unsigned)time(0));
61     for (int i = 0; i < BOX_COUNT; i++) {
62         // position
63         boxes[i].x = rand() % SCREEN_WIDTH;
64         boxes[i].y = rand() % SCREEN_HEIGHT;
65         // size
66         boxes[i].w = (rand() % BOX_WIDTH) + 1;
67         boxes[i].h = (rand() % BOX_HEIGHT) + 1;
68         // velocity (both positive and negative "delta" values)
69         boxes[i].dx = (rand() % (BOX_SPEED * 2)) - (BOX_SPEED / 2);
70         boxes[i].dy = (rand() % (BOX_SPEED * 2)) - (BOX_SPEED / 2);
71     }
72 }
73
74
75 void render_box(CrashBox& box, SDL_Renderer* renderer, SDL_Color& color) ↗
76 {
77     SDL_Rect r = { box.x, box.y, box.w, box.h };
78     SDL_SetRenderDrawColor(renderer, color.r, color.g, color.b, color.a);
79     SDL_RenderFillRect(renderer, &r);
80 }
81
82 void render_boxes(SDL_Renderer* renderer) {
83     SDL_Color white = { 255,255,255,255 };
84     SDL_Color red = { 255,0,0,255 };
85
86     for (int i = 0; i < BOX_COUNT; i++) {
87         if (boxes[i].state == CONTACT_NO) {
88             render_box(boxes[i], renderer, white);
89         }
90         else {
91             render_box(boxes[i], renderer, red);
92         }
93     }
94 }
```

```
95 //-----  
-----  
96  
97 bool crash_test_A(int i, int j) // via index  
98 { //yeap - lazyfoo style!  
99  
100     //The sides of the rectangles  
101     int leftA, leftB;  
102     int rightA, rightB;  
103     int topA, topB;  
104     int bottomA, bottomB;  
105  
106     CrashBox A, B;  
107     A = boxes[i];  
108     B = boxes[j];  
109  
110     //Calculate the sides of rect A  
111     leftA = A.x;  
112     rightA = A.x + A.w;  
113     topA = A.y;  
114     bottomA = A.y + A.h;  
115  
116     //Calculate the sides of rect B  
117     leftB = B.x;  
118     rightB = B.x + B.w;  
119     topB = B.y;  
120     bottomB = B.y + B.h;  
121  
122     //If any of the sides from A are outside of B  
123     if (bottomA <= topB) return false;  
124     if (topA >= bottomB) return false;  
125     if (rightA <= leftB) return false;  
126     if (leftA >= rightB) return false;  
127  
128     //If none of the sides from A are outside B  
129     return true;  
130 }  
131  
132 bool crash_test_B(CrashBox A, CrashBox B) {  
133     int leftA, leftB;  
134     int rightA, rightB;  
135     int topA, topB;  
136     int bottomA, bottomB;  
137  
138     leftA = A.x;  
139     rightA = A.x + A.w;  
140     topA = A.y;  
141     bottomA = A.y + A.h;  
142
```

```
143     leftB = B.x;
144     rightB = B.x + B.w;
145     topB = B.y;
146     bottomB = B.y + B.h;
147
148     if (bottomA <= topB) return false;
149     if (topA >= bottomB) return false;
150     if (rightA <= leftB) return false;
151     if (leftA >= rightB) return false;
152     return true;
153 }
154
155
156 bool crash_test_C(CrashBox& A, CrashBox& B) { // via struct (ref!)
157     int leftA, leftB;
158     int rightA, rightB;
159     int topA, topB;
160     int bottomA, bottomB;
161
162     leftA = A.x;
163     rightA = A.x + A.w;
164     topA = A.y;
165     bottomA = A.y + A.h;
166
167     leftB = B.x;
168     rightB = B.x + B.w;
169     topB = B.y;
170     bottomB = B.y + B.h;
171
172     if (bottomA <= topB) return false;
173     if (topA >= bottomB) return false;
174     if (rightA <= leftB) return false;
175     if (leftA >= rightB) return false;
176
177     return true;
178 }
179
180 bool crash_test_D(CrashBox& A, CrashBox& B) {
181     if ((A.y + A.h) <= B.y) return false;
182     if (A.y >= (B.y + B.h)) return false;
183     if ((A.x + A.w) <= B.x) return false;
184     if (A.x >= (B.x + B.w)) return false;
185     return true;
186 }
187
188 bool crash_test_E(CrashBox& A, CrashBox& B) {
189     return ((A.y + A.h) <= B.y || A.y >= (B.y + B.h)
190         || (A.x + A.w) <= B.x || A.x >= (B.x + B.w));
191 }
```

```

192
193 bool crash_test_F(CrashBox& A, CrashBox& B) {
194     if ((A.y + A.h) <= B.y) return false;
195     else if (A.y >= (B.y + B.h)) return false;
196     else if ((A.x + A.w) <= B.x) return false;
197     else if (A.x >= (B.x + B.w)) return false;
198     else return true;
199 }
200
201 //----- ↗
    -----
202
203 /* Extra check for minimal gains == worst performing func thanks to cache ↗
    miss
204 void crash_test_all_A1() {
205     // check i against j
206     for (int i = 0; i < BOX_COUNT; i++) {
207         for (int j = 0; j < BOX_COUNT; j++) {
208             if (crash_test_A(i, j)) {
209                 if (i != j) { // <-- difference between A1 and A2.
210                     boxes[i].state = CONTACT_YES;
211                     boxes[j].state = CONTACT_YES;
212                 }
213             }
214         }
215     }
216 }
217
218 /* Also garbage
219 void crash_test_all_A2() {
220     for (int i = 0; i < BOX_COUNT; i++) {
221         for (int j = i + 1; j < BOX_COUNT; j++) {
222             if (crash_test_A(i, j)) {
223                 boxes[i].state = CONTACT_YES;
224                 boxes[j].state = CONTACT_YES; }
225         }
226     }
227 }
228
229 // Yeh we pass the struct but internal copies are made causing this to be ↗
    hecka slow too
230 void crash_test_all_B() {
231     for (int i = 0; i < BOX_COUNT; i++) {
232         for (int j = i + 1; j < BOX_COUNT; j++) {
233             if (crash_test_B(boxes[i], boxes[j])) {
234                 boxes[i].state = CONTACT_YES;
235                 boxes[j].state = CONTACT_YES;
236             }
237         }
    }

```

```
238     }
239 }
240
241 // Pass via struct reference, very cool and works nice but all the internal
242 // copies and assignments cause a performance hit still
243 void crash_test_all_C() {
244     for (int i = 0; i < BOX_COUNT; i++) {
245         for (int j = i + 1; j < BOX_COUNT; j++) {
246             if (crash_test_C(boxes[i], boxes[j])) {
247                 boxes[i].state = CONTACT_YES;
248                 boxes[j].state = CONTACT_YES;
249             }
250         }
251     }
252 }
253
254
255 // WOOOOOOO WE DID IT BOIS! this one is pre speed.
256 void crash_test_all_D() {
257     for (int i = 0; i < BOX_COUNT; i++) {
258         for (int j = i + 1; j < BOX_COUNT; j++) {
259             if (crash_test_D(boxes[i], boxes[j])) {
260                 boxes[i].state = CONTACT_YES;
261                 boxes[j].state = CONTACT_YES;
262             }
263         }
264     }
265 }
266
267 void crash_test_all_E() {
268     for (int i = 0; i < BOX_COUNT; i++) {
269         for (int j = i + 1; j < BOX_COUNT; j++) {
270             if (crash_test_E(boxes[i], boxes[j])) {
271                 boxes[i].state = CONTACT_YES;
272                 boxes[j].state = CONTACT_YES;
273             }
274         }
275     }
276 }
277
278 void crash_test_all_F() {
279     for (int i = 0; i < BOX_COUNT; i++) {
280         for (int j = i + 1; j < BOX_COUNT; j++) {
281             if (crash_test_F(boxes[i], boxes[j])) {
282                 boxes[i].state = CONTACT_YES;
283                 boxes[j].state = CONTACT_YES;
284             }
285         }
286     }
287 }
```

```

286     }
287 }
288
289 void crash_test_all_G() {
290     auto cached_boxes = boxes;
291     int count = BOX_COUNT;
292
293     for (int i = 0; i < count; i++) {
294         for (int j = i + 1; j < count; j++) {
295             if (crash_test_F(cached_boxes[i], cached_boxes[j])) {
296                 cached_boxes[i].state = CONTACT_YES;
297                 cached_boxes[j].state = CONTACT_YES;
298             }
299         }
300     }
301 }
302
303 //----- ↗
304
305 void update_boxes()
306 {
307     // First move all boxes
308     for (int i = 0; i < BOX_COUNT; i++) {
309         // update position using current velocity
310         boxes[i].x = boxes[i].x + boxes[i].dx;
311         boxes[i].y = boxes[i].y + boxes[i].dy;
312         // check for wrap-around condition
313         if (boxes[i].x >= SCREEN_WIDTH) boxes[i].x -= SCREEN_WIDTH;
314         if (boxes[i].x < 0) boxes[i].x += SCREEN_WIDTH;
315         if (boxes[i].y >= SCREEN_HEIGHT) boxes[i].y -= SCREEN_HEIGHT;
316         if (boxes[i].y < 0) boxes[i].y += SCREEN_HEIGHT;
317     }
318
319     // 1. mark all boxes as not collided
320     for (int i = 0; i < BOX_COUNT; i++)
321         boxes[i].state = CONTACT_NO;
322
323     // 2. call whatever function has been set to test all i against j ↗
324     boxes
325     crash_test_all_ptr();
326 }
327
328 //----- ↗
329
330 int run_test(const char* title, void (*function_ptr)()) {
331     // get SDL to setup all subsystems

```

```
332     if (SDL_Init(SDL_INIT_VIDEO) == -1) {
333         std::cout << " Failed to initialize SDL : " << SDL_GetError() <<
334             std::endl;
335         return -1;
336     }
337     // window with title and size we want
338     SDL_Window* window = SDL_CreateWindow(
339         title,
340         SDL_WINDOWPOS_UNDEFINED,
341         SDL_WINDOWPOS_UNDEFINED,
342         SCREEN_WIDTH,
343         SCREEN_HEIGHT,
344         SDL_WINDOW_HIDDEN
345     );
346
347     if (window == nullptr) {
348         std::cout << "Failed to create window : " << SDL_GetError();
349         return -1;
350     }
351
352     // renderer for the window ...
353     SDL_Renderer* renderer = SDL_CreateRenderer(
354         window,
355         -1,
356         0//SDL_RENDERER_ACCELERATED
357     );
358
359     if (renderer == nullptr) {
360         std::cout << "Failed to create renderer : " << SDL_GetError();
361         return -1;
362     }
363
364     // some pretty test output to the console
365     printf("-- New Test: %s\n", title);
366     crash_test_all_ptr = function_ptr;
367
368     // sanity check that the crash test function pointer has been set
369     if (crash_test_all_ptr == nullptr) {
370         printf("EH? Set the crash_test_all_ptr first!\n");
371         return 1;
372     }
373
374     // initialise each crashbox
375     init_boxes();
376
377     // initialise test count/time values
378     Uint32 loop_count = 0;
379     Uint32 tick_start = SDL_GetTicks(); // start time == now!
```



```
380     Uint32 tick_target = tick_start + TEST_TIME; // when to stop
381
382     // CLASSIC GAME LOOP
383     bool running = true;
384     SDL_Event event;
385     while (running) {
386         // UPDATE
387         // 1. check for quit event
388         while (SDL_PollEvent(&event)) {
389             if (event.type == SDL_QUIT) running = false;
390         }
391         // 2. count...
392         loop_count++;
393         // 3. check for test time finished
394         if (SDL_GetTicks() >= tick_target) running = false;
395         // 4. move all the crash boxes and check for collisions
396         update_boxes();
397
398         // RENDER
399         if (false) {
400             // 1. clear the background
401             SDL_SetRenderDrawColor(renderer, 100, 100, 100, 255);
402             SDL_RenderClear(renderer);
403             // 2. render all boxes
404             render_boxes(renderer);
405             // 3. show it
406             SDL_RenderPresent(renderer);
407         }
408         //SDL_Delay(50); // uncomment to see at slow speed.
409     }
410     // note the end time
411     Uint32 tick_end = SDL_GetTicks();
412
413     // CLEAN UP TIME (and close the SDL window)
414     SDL_DestroyRenderer(renderer);
415     SDL_DestroyWindow(window);
416     SDL_Quit();
417
418     // SHOW STATS
419     printf("Loops: %d\n", loop_count);
420     printf("Time: %d (ms)\n", (tick_end - tick_start));
421     printf("Loops/Second: %f\n", (float(loop_count) / (tick_end -
422         tick_start) * 1000.0));
423
424     return 0;
425 }
426 int main(int argc, char* args[])
427 {
```

```
428     //run_test("Test A1", crash_test_all_A1);
429     //run_test("Test A2", crash_test_all_A2);
430     //run_test("Test B", crash_test_all_B);
431     //run_test("Test C", crash_test_all_C);
432     run_test("Test D", crash_test_all_D);
433     run_test("Test E", crash_test_all_E);
434     run_test("Test F", crash_test_all_F);
435     run_test("Test G", crash_test_all_G);
436
437     return 0;
438 }
439
440
```