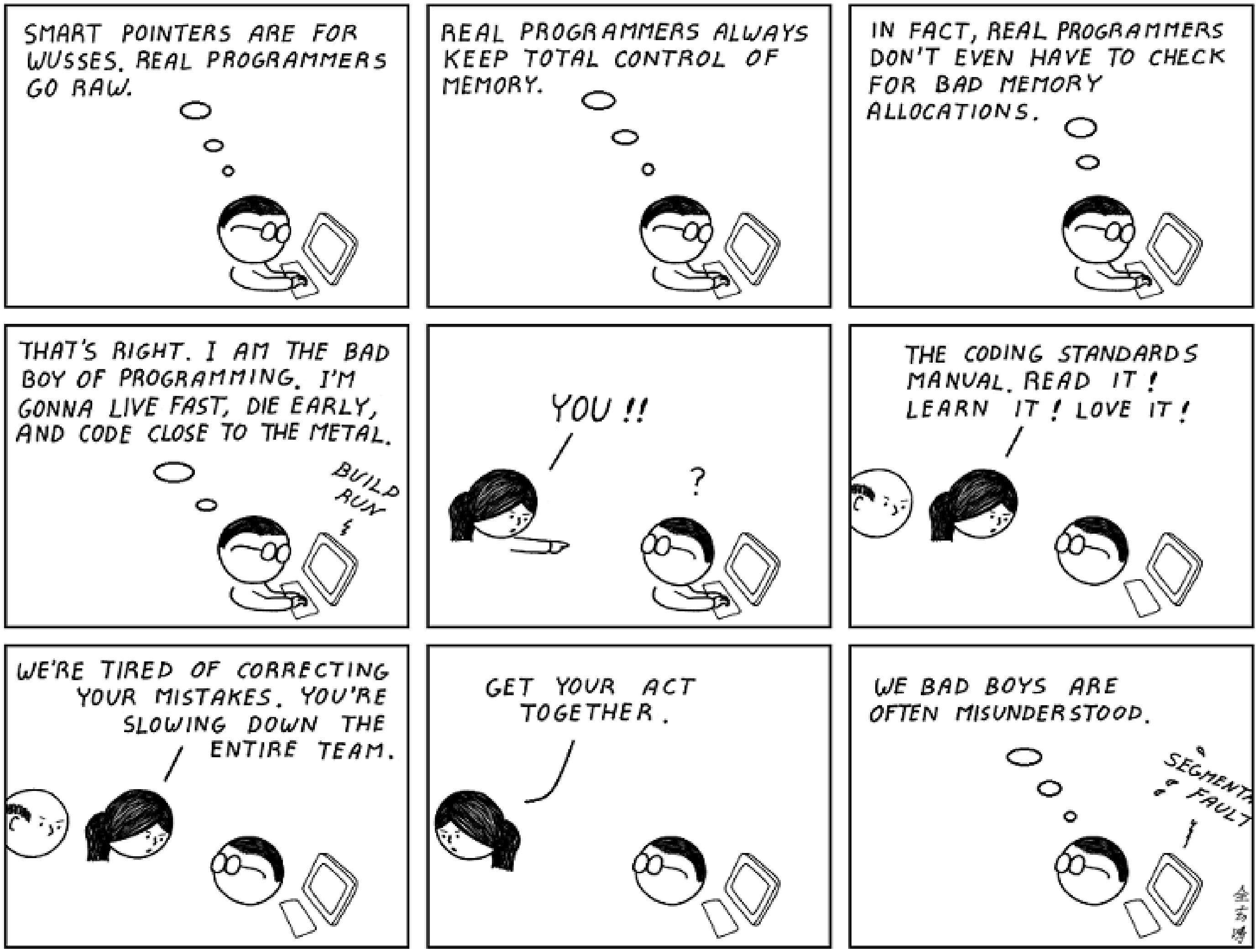
DEPARTMENT OF COMPUTER GRAPHICS AND INTERACTION

Crash Course in Memory Management (in C/C++)

Jakub Hendrich, Daniel Meister


SMART POINTERS ARE FOR WUSSES. REAL PROGRAMMERS GO RAW.
REAL PROGRAMMERS ALWAYS KEEP TOTAL CONTROL OF MEMORY.
IN FACT, REAL PROGRAMMERS DON'T EVEN HAVE TO CHECK FOR BAD MEMORY ALLOCATIONS.
THAT'S RIGHT. I AM THE BAD BOY OF PROGRAMMING. I'M GONNA LIVE FAST, DIE EARLY, AND CODE CLOSE TO THE METAL.
BUILD RUN
YOU !!
?
THE CODING STANDARDS MANUAL. READ IT! LEARN IT! LOVE IT!
WE'RE TIRED OF CORRECTING YOUR MISTAKES. YOU'RE SLOWING DOWN THE ENTIRE TEAM.
GET YOUR ACT TOGETHER.
WE BAD BOYS ARE OFTEN MISUNDERSTOOD.
SEGMENTATION FAULT

http://abstrusegoose.com/483

Memory Storage Types in C/C++

- global / static local – in the data segment
  - lives forever (throughout the process lifetime)
- automatic – on the stack
  - lexical scope (within a function / method / block)
- dynamic – on the heap
  - ? → programmer defined

What about allocation / disposal, initialization / finalization?

Java vs C/C++ differences

- Java: (almost) everything is an object on the heap
  - variables are references
  - automatic garbage collection
- C/C++: free to choose the most suitable storage
  - variables are objects themselves, or pointers, or references
  - automatic variables garbage collection: RAII
  - no automatic garbage collection of dynamic variables

Issues found

- Dynamic memory abuse
  - up to 740 MB or 16 million allocations
    - crippling the performance severely!
  - ~59 MB and tens of allocations should be enough
    - color buffers in 8 contexts → 8 * (800 * 600 * 3 * sizeof(float)) bytes
    - depth buffers → 8 * (800 * 600 * sizeof(float)) bytes

    + context & matrix stacks & vertex buffer
  - local variables: should be automatic, not dynamic
  - class instances should normally contain the data within themselves, not via another level(s) of reference:
    - class CMatrix { float matrix[4][4]; } vs class CMatrix { float ** matrix; }

Issues found

- Uninitialized memory usage
  - automatic & dynamic variables of built-in types have no implicit initialization
    - mostly the color buffer (float*) not prepared for reading by the testapp
- No/bad destruction/disposal
  - causes memory leaks or corruption
    - calling the destructor explicitly instead of delete / delete[ ]
    - new vs new[ ], delete vs delete[ ]
- Taking address of temporary
  - suspicious and error-prone idiom
    - temporary objects die immediately after full-expression evaluation

Diagnostics

- Compiler output

```
OurVector4 applyTransform (OurVector4 v) { … }

void sglEllipse(float cx, float cy, float cz, float a, float b) { …
  OurVector4* center = new OurVector4(cx,cy,cz,1);
  center = &applyTransform(*center);
  drawPixel(*center);
  delete center;
… }
```

```
sgl/sgl.cpp: In function ‘void sglEllipse(float, float, float, float, float)’:
sgl/sgl.cpp:514:35: warning: taking address of temporary [-fpermissive]
  center = &applyTransform(*center);
                                   ^
```

http://en.cppreference.com/w/cpp/language/lifetime

Diagnostics

- Memory loggers/debuggers
  - top
  - eFence
  - Valgrind suite (Memcheck et al.)

```
==686== Memcheck, a memory error detector
==686== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==686== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==686== Command: ./testapp
```

Valgrind output

```
==575== Memcheck, a memory error detector
==575== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==575== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==575== Command: ./testapp
==575==
==575==
==575== HEAP SUMMARY:
==575==     in use at exit: 0 bytes in 0 blocks
==575==   total heap usage: 25 allocs, 25 frees, 46,204,900 bytes allocated
==575==
==575== All heap blocks were freed -- no leaks are possible
==575==
==575== For counts of detected and suppressed errors, rerun with: -v
==575== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Valgrind output

```
==32083== HEAP SUMMARY:
==32083==     in use at exit: 0 bytes in 0 blocks
==32083==   total heap usage: 759,144 allocs, 759,144 frees, 110,074,668 bytes allocated
==32083==
==32083== All heap blocks were freed -- no leaks are possible


==32331== Conditional jump or move depends on uninitialised value(s)
==32331==    at 0x804E985: float const& std::min<float>(float const&, float const&)
      (stl_algobase.h:199)
==32331==    by 0x804963D: WriteTGA(char const*) (testapp.cpp:157)
==32331==    by 0x804C6F1: main (testapp.cpp:1186)
```

Valgrind output

```
==32182== 960,000 (192,000 direct, 768,000 indirect) bytes in 12,000 blocks are definitely
    lost in loss record 890 of 890
==32182==    at 0x4007D83: operator new[](unsigned int) (in
    /usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
==32182==    by 0x80542EE: CMatrix::operator=(CMatrix const&) (sgl.cpp:160)
==32182==    by 0x805243B: sglBegin(sglEElementType) (sgl.cpp:695)
==32182==    by 0x804B6B2: DrawTestScene1A() (testapp.cpp:750)
==32182==    by 0x804C6D7: main (testapp.cpp:1184)
==32182==
==32182== LEAK SUMMARY:
==32182==    definitely lost: 6,080,080 bytes in 380,005 blocks
==32182==    indirectly lost: 24,320,272 bytes in 1,520,017 blocks
==32182==      possibly lost: 48 bytes in 3 blocks
==32182==    still reachable: 0 bytes in 0 blocks
==32182==         suppressed: 0 bytes in 0 blocks
```

Valgrind output

```
==32182== Mismatched free() / delete / delete []
==32182==    at 0x40086BD: operator delete(void*) (in
       /usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
==32182==    by 0x8054475: CMatrix::~CMatrix() (sgl.cpp:179)
==32182==    by 0x8051561: sglInit() (sgl.cpp:414)
==32182==    by 0x804C55C: Init() (testapp.cpp:1011)
==32182==    by 0x804C5ED: main (testapp.cpp:1139)
==32182==  Address 0x403c1a8 is 0 bytes inside a block of size 16 alloc'd
==32182==    at 0x4007D83: operator new[](unsigned int) (in
       /usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
==32182==    by 0x80540AD: CMatrix::CMatrix(float const*) (sgl.cpp:118)
==32182==    by 0x805153E: sglInit() (sgl.cpp:414)
==32182==    by 0x804C55C: Init() (testapp.cpp:1011)
==32182==    by 0x804C5ED: main (testapp.cpp:1139)
```

Valgrind output

```
==32182== More than 100 errors detected.  Subsequent errors
==32182== will still be recorded, but in less detail than before.
==32182==
==32182== More than 10000000 total errors detected.  I'm not reporting any more.
==32182== Final error counts will be inaccurate.  Go fix your program!
==32182== Rerun with --error-limit=no to disable this cutoff.  Note
==32182== that errors may occur in your program without prior warning from
==32182== Valgrind, because errors are no longer being displayed.
```

Thank you for your attention!

Jakub Hendrich

14.10.2024