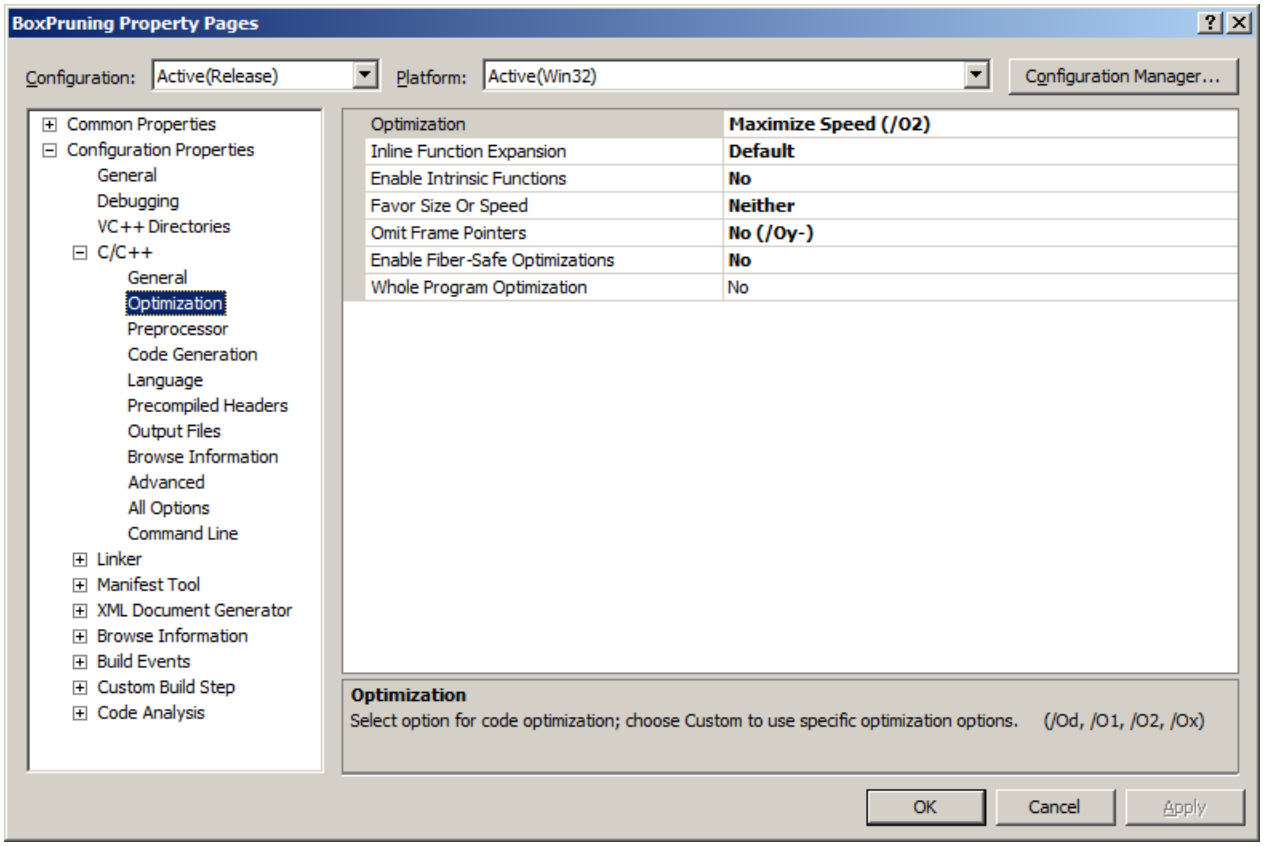
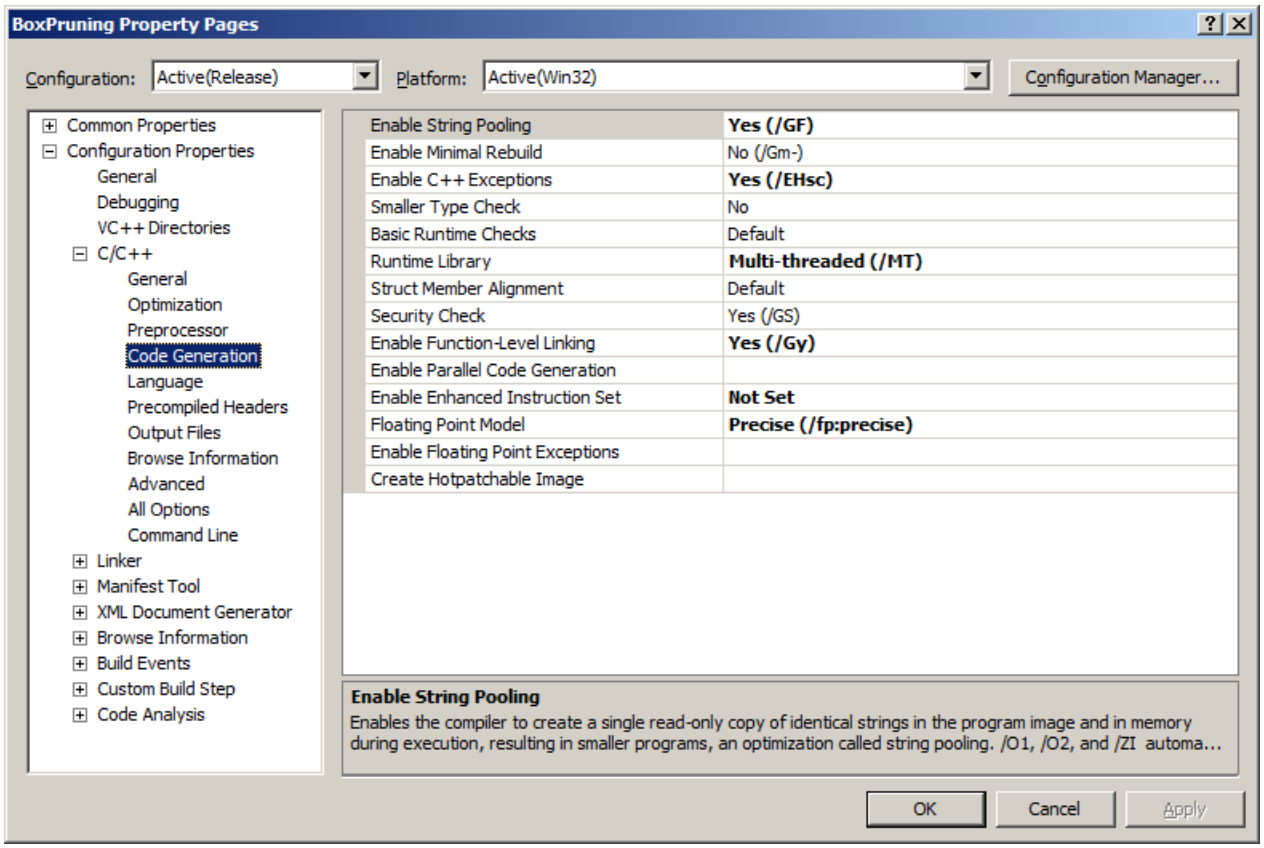
"Box pruning revisited" – an optimization project by Pierre Terdiman – 2017

Part 2 - compiler options

After converting the project, the default compiler options for the Release configuration are as follows:





These are basically the only options affecting performance, if we ignore a few additional ones in the linker. After some years programming, you get a feeling for which ones of these options are important, and which ones are not.

So… what do you think, optimization experts?
Which one of these will make the most difference here?

For me, looking at that list, the suspicious ones that should be changed immediately are the following:

Inline Function Expansion: Default Enable C++ Exceptions: Yes (/EHsc) Enable Enhanced Instruction Set: Not Set Floating Point Model: Precise (/fp:precise)

These guys are the usual suspects, as far as I'm concerned.

But let's test this.

We first change the “Inline Function Expansion” option from “*Default*” to “*Only __inline (/Ob1)*”

Results:

```
Complete test (brute force): found 11811 intersections in 837222 K-cycles.
Complete test (box pruning): found 11811 intersections in 96230 K-cycles.
```

In general, inlining has a strong impact on performance, and there would be a need for an entirely separate blog post about the art of inlining. But in this specific case, it does not make any difference: the "*Default*" setting works well enough. The disassembly is the same 202 instructions as before so it did not change anything.

Note that it does not mean nothing gets inlined. "*Default*" does not mean "*no inlining*". If you *really* disable inlining with the “Disabled /Ob0” option, you get the following results:

```
Complete test (brute force): found 11811 intersections in 1155114 K-cycles.
Complete test (box pruning): found 11811 intersections in 174538 K-cycles.
```

That is quite a difference. So clearly, inlining is one of the important things to get right, and in this specific case the default option is already right.

Good.

Now, we switch inlining back to "*Default*" and move on to the next option.

This time we disable exceptions (“Enable C++ Exceptions: No”).

Results:

```
Complete test (brute force): found 11811 intersections in 817837 K-cycles.
Complete test (box pruning): found 11811 intersections in 95789 K-cycles.
```

It appears that disabling exceptions has a noticeable effect on the brute-force implementation, but no clear effect on the box-pruning function.

The disassembly shows the same 202 instructions as before, so the small performance gain is not actually here, it's just noise in the results. It is important to check the disassembly to confirm that the "optimization" actually changed something. Otherwise you can often see what you want to see in the results...

Well, exceptions usually add a tiny overhead for each function, and it can accumulate and add up to “a lot” (relatively speaking). But we don’t have a lot of functions here, and as we just saw before with

inlining, the few functions we have are properly inlined. So it makes sense that disabling exceptions in this case does not change much.

Worth checking though. Don't assume. Check and check and check.

Next, we turn exceptions back on (to measure the effect of each option individually) and then we move on to this:

Enable Enhanced Instruction Set: Not Set

This one is interesting. The default is "Not Set" but if you look at the disassembly so far, it is clearly using some SSE registers (for example xmm0). So it turns out that by default, the compiler uses SSE instructions these days. Which explains why switching to /arch:SSE2 gives these results:

```
Complete test (brute force): found 11811 intersections in 829134 K-cycles.
Complete test (box pruning): found 11811 intersections in 96448 K-cycles.
```

Again it appears that the option has a small effect on the brute-force code (if this isn't just noise), and no effect on the box-pruning code. And indeed, the disassembly shows the same 202 instructions as before.

Now, as an experiment, we can switch back to regular x87 code with the */arch:IA32* compiler flag ("Enable Enhanced Instruction Set : No Enhanced Instructions"). This gives the following results:

```
Complete test (brute force): found 11811 intersections in 921893 K-cycles.
Complete test (box pruning): found 11811 intersections in 110605 K-cycles.
```

Ok, so that makes more sense: using SSE2 does in fact have a clear impact on performance, it's just that the default "Not Set" option was actually already using it. That's a bit confusing, but at least the results now make sense.

While we are looking at these results, please note how enabling the SSE2 compiler flag only provides modest gains, from ~110000 to ~97000: that's about a 10% speedup only. This is again why we didn't enable that compile flag back in *PhysX 2.x*. Contrary to what naive users claimed online, using the flag does not magically make your code 4X faster.

Finally, we focus our attention on the Floating Point Model, and switch it to */fp:fast*. You might guess the results by now:

```
Complete test (brute force): found 11811 intersections in 825688 K-cycles.
Complete test (box pruning): found 11811 intersections in 96115 K-cycles.
```

And yes, the disassembly is exactly the same as before. Frustrating.

At that point I got tired of what looked like a pointless experiment. I just quickly setup the remaining options with my usual settings, compiled, and...

...the code got measurably faster:

```
Complete test (brute force): found 11811 intersections in 817855 K-cycles.
Complete test (box pruning): found 11811 intersections in 93208 K-cycles.
```

Like, a 100% reproducible good 3000 K-Cycles faster.

And the speedup was not imaginary, it was indeed reflected in the disassembly: 192 instructions.

What….

After a short investigation I discovered that the only compiler option that made any difference, the one responsible for the speedup was "*Omit Frame Pointers*".

….the hell?

So, experts, did you predict that? :)

I did not.

This was doubly surprising to me.

The first surprise is that /O2 (which was enabled by default in Release, right from the start) is supposed to include /Oy, i.e. the "Omit Frame Pointers" optimization.

But this link explains the problem:

> "The /Ox (Full Optimization) and /O1, /O2 (Minimize Size, Maximize Speed) options imply /Oy. Specifying /Oy– after the /Ox, /O1, or /O2 option disables /Oy, whether it is explicit or implied."

So, the issue is that the combo box in the compiler options exposes "No (/Oy-)", which is an explicit "disable" command rather than a "use whatever is the default". So as the MSDN says, it overrides and undoes the /O2 directive. Oops.

The second surprise for me was that it had such a measurable impact on performance: about 5%. Not bad for something that wasn't even on my radar. To be fair this issue doesn't exist with 64-bit builds (as

far as I know), so maybe that's why I never saw that one before. According to the MSDN /Oy frees up one more register, EBP, for storing frequently used variables and sub-expressions. Somehow I didn't realize that before, and it certainly explains why things get faster.

Ok!

Compiler options: checked!

Other than that, I removed some obsolete files from the project, but made no code changes. This version is now going to be our base version against which the next optimizations will be measured.

For reference, this was my best run on the office PC:

```
Complete test (brute force): found 11811 intersections in 819814 K-cycles.
102256 K-cycles.
93092 K-cycles.
92885 K-cycles.
99537 K-cycles.
93146 K-cycles.
93325 K-cycles.
95795 K-cycles.
97573 K-cycles.
97606 K-cycles.
98829 K-cycles.
97040 K-cycles.
95197 K-cycles.
98149 K-cycles.
93226 K-cycles.
93008 K-cycles.
95254 K-cycles.
Complete test (box pruning): found 11811 intersections in 92885 K-cycles.
```

And on the home PC:

```
Complete test (brute force): found 11811 intersections in 781996 K-cycles.
102578 K-cycles.
98972 K-cycles.
98898 K-cycles.
99183 K-cycles.
98920 K-cycles.
98823 K-cycles.
98948 K-cycles.
99047 K-cycles.
99132 K-cycles.
98822 K-cycles.
```

98975 K-cycles.
100701 K-cycles.
98892 K-cycles.
99025 K-cycles.
99294 K-cycles.
98981 K-cycles.
Complete test (box pruning): found 11811 intersections in 98822 K-cycles.

The progress will be captured in this table:

	Home PC	Office PC
Version1	101662	96203
Version2 - base	98822	92885

What we learnt:

The default performance-related compiler options in Release mode are pretty good these days, but you can still do better if you go there and tweak them.

Next time, we will start looking at the code and investigate what we can modify. That is, we will start the proper optimizations.