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

Part 7 – integer comparisons

I spent a lot of time using *PIX* on the *Xbox360*. This is a profiler tool, which incidentally was just released on PC (I did not try it yet, I don't know how it compares to the Xbox version).

On the Xbox it typically identified and reported 3 main problems: L2 cache misses, FCMPs, and LHS.

Well, granted: it reported a lot more than this, but these guys were the typical issues one had to pay attention to.

We just took care of L2 cache misses, or data cache misses, in the previous posts.

LHS (for *Load-Hit-Store*) were a nuisance specific to the Xbox360 that we will ignore for now. They do exist on PC, but their performance penalty is not as severe as on the previous-gen Xbox.

So that leaves the FCMPs, i.e. the float comparisons. Weird thing, the FCMPs. They used to be very costly on old Pentiums IIRC. And then I think for a while they were ok. And then with the Xbox360 suddenly they were bad again. These things come and go.

This gave birth to a whole bunch of optimizations where FCMPs got replaced with regular integer comparisons, which were faster. It made a huge difference on the 360, and I kept doing this kind of stuff as a habit. But to be fair, this may not be needed anymore on modern PCs. And that's what we are going to investigate today.

We have to do it now, before starting the SIMD work, because it might also affect the SIMD code. There is floating point SIMD and integer SIMD, and with SSE2 the instruction sets are quite different. It is unclear if there would be a benefit using one or the other at this point, so we will investigate, and maybe try both.

But SIMD will bring some complications to the table, so it is easier to start slowly and first play with integer comparisons while the code is still scalar.

I already wrote about this issue in the already mentioned SAP document. Go read Appendix A this time, page 22. I'll wait.

Back? Good.

As mentioned in the document, the changes are quite straightforward. We replace our AABB with its integer counterpart:

```
struct IntegerAABB
{
    udword  mMinX;
    udword  mMinY;
    udword  mMinZ;
    udword  mMaxX;
    udword  mMaxY;
    udword  mMaxZ;
};
```

Then we encode its min & max values using the function from the SAP document:

```
static __forceinline udword encodeFloat(udword ir)
{
    if(ir & 0x80000000) //negative?
        return ~ir;//reverse sequence of negative numbers
    else
        return ir | 0x80000000; // flip sign
}
```

And then we just replace some "`float`" with "`udword`" in the main loop, float comparisons silently become integer comparisons, and that's pretty much it really. Once you understand the encoding function, the modifications are trivial. Well, as long as we're not using SIMD at least.

The results are as follows:

Home PC:

> Complete test (brute force): found 11811 intersections in 782125 K-cycles.
> 41727 K-cycles.
> 41126 K-cycles.
> 40956 K-cycles.
> 41431 K-cycles.
> 41042 K-cycles.
> 41169 K-cycles.
> 40928 K-cycles.
> 40916 K-cycles.
> 40912 K-cycles.
> 40906 K-cycles.
> 41116 K-cycles.
> 40927 K-cycles.
> 40911 K-cycles.
> 40969 K-cycles.
> 41143 K-cycles.
> 41251 K-cycles.
> Complete test (box pruning): found 11811 intersections in 40906 K-cycles.

Office PC:

Complete test (brute force): found 11811 intersections in 823453 K-cycles.
44674 K-cycles.
45127 K-cycles.
45929 K-cycles.
44652 K-cycles.
44192 K-cycles.
44502 K-cycles.
45403 K-cycles.
44062 K-cycles.
44859 K-cycles.
44060 K-cycles.
44679 K-cycles.
43987 K-cycles.
44537 K-cycles.
44417 K-cycles.
44842 K-cycles.
44395 K-cycles.
Complete test (box pruning): found 11811 intersections in 43987 K-cycles.

The gains are summarized here:

Home PC	Timings (K-Cycles)	Delta (K-Cycles)	Speedup	Overall X factor
(Version1)	(101662)			
Version2 - base	98822	0	0%	1.0
Version3	93138	~5600	~5%	~1.06
Version4	81834	~11000	~12%	~1.20
Version5	78140	~3600	~4%	~1.26
Version6a	60579	~17000	~22%	~1.63
Version6b	41605	~18000	~31%	~2.37
Version7	40906	-	-	-

Office PC	Timings (K-Cycles)	Delta (K-Cycles)	Speedup	Overall X factor
(Version1)	(96203)			
Version2 - base	92885	0	0%	1.0
Version3	88352	~4500	~5%	~1.05
Version4	77156	~11000	~12%	~1.20
Version5	73778	~3300	~4%	~1.25
Version6a	58451	~15000	~20%	~1.58
Version6b	45634	~12000	~21%	~2.03
Version7	43987	-	-	-

Sooooo..... That one is not very conclusive. It appears to be a tiny bit faster, but this is far from the speedups we used to get on the Xbox.

I would say that the jury is still out on this one, but either way, the speedups are too small to justify the trouble at this point. So we will ignore integer comparisons for now and perhaps revisit this later with SIMD.

The results are curious though, because this stuff used to matter a lot. What happened?

Again, you can design a small experiment to reveal the truth. The only difference between version 6b and version 7 is the switch from floating-point to integer-comparisons.

So what we can do is compile both *without* the SSE2 flag, i.e. with the explicit */arch:IA32* flag to ensure that x87 code is used.

Here are the results on the home PC:

Version 6b with /SSE2 flag	41605 K-cycles
Version 6b with x87 code	52163 K-cycles
Version 7 with /SSE2 flag	40906 K-cycles
Version 7 with x87 code	40897 K-cycles

Ah-ah! So that was it, look at this!

With the x87 code, switching from float to integer comparisons has a huge impact.

With the SSE2 code, switching from float to integer comparisons has almost no impact.

Using integer comparisons with the x87 code gave the same speedup as switching to SSE2 code with float comparisons.

So that's what happened: using integer comparisons gave big speedups in the past with x87 code (and even more so on the Xbox 360), but compiling with the /SSE2 flag gives the same speedups out-of-the-box, making this optimization obsolete.

And that's our lesson of the day: don't assume anything. Question everything. Constantly revisit your previous conclusions.

We will now ignore version 7 entirely: the minor gains are not worth the trouble.

What we learnt:

Replacing float comparisons with integer comparisons in scalar code might still provide some minor gains, but not as much as in the past. Using the SSE2 flag now gives essentially the same speedups out-of-the-box.

One great optimization one day on one platform might become worthless (or even detrimental) another day, on another platform.

You can learn something even from a “failed” optimization. Things that did not work are also worth reporting and talking about.

Ok! With this question out of the way, we're ready to tackle the SIMD version.

See you next time.