

Re: Precision issue with Float and Double (C++)

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.mfc/2008-08/msg00912.html>

- *From:* Joseph M. Newcomer <newcomer@xxxxxxxxxxxxx>
 - *Date:* Thu, 28 Aug 2008 19:41:10 -0400
-

See below...

On Thu, 28 Aug 2008 19:36:12 +0100, "David Webber" <dave@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote:

"Joseph M. Newcomer" <newcomer@xxxxxxxxxxxxx> wrote in message news:8ildb4lur15s40mvnaid8eo0k597rbb8t3@xxxxxxxxxxxx

- Rule 1: There is no such thing as "decimal precision" in floating point
- Rule 2: If you are having problems with "decimal precision", consult rule 1
- Rule 3: If are convinced double should work like decimal arithmetic, consult Rule 1.
- Rule 4: If you still believe double should work like decimal arithmetic, choose another profession.

:~)

Rule 5: interpret Joe's rules carefully :~)

Admittedly, they are somewhat oversimplified. But the naive programmers who think floating point is decimal arithmetic tend to not give that attitude up easily, and I get bored arguing with them. So I simplify the discussion somewhat. I've forgotten a lot of my numerical programming (I don't do it for a living, and there was a period of over a decade where I never wrote a single floating-point computation) but one thing I absolutely remember was that cumulative LSB error can be fatal, and when it is, it means your algorithm is fundamentally wrong. There is no way to avoid knowing about LSB errors in numeric programming if you are doing serious numeric programming, and those who haven't seen it before need to stop, take a deep breathe, and then learn it. They somehow think there is a "quick fix", and if there's one thing that doesn't exist, it's a quick fix.

If I were ever confronted with this kind of problem, I'd grab my copy of Numerical Recipes (which is first edition, alas; they're now up to the third edition) and start reading carefully.

Re: Precision issue with Float and Double (C++)

..
If the error is "significant enough to affect my calculation" then your calculation is not designed correctly. One of the first things we teach students in computing is that you must *not ever* create a computation which is sensitive to the binary representation issues of floating point, and in fact, numerical analysts have known about this problem since the 1940s (John von Neumann, the inventor of the modern stored-program computer, identified all of these issues AND how to deal with them, before I was born, which was 1947, and they were known to a lesser degree in the 1920s with mechanical calculators (decimal arithmetic has EXACTLY the same problems), ...

Indeed. It is the subject of "Numerical Maths" which, as you observe, rather predates that of "Windows programming"

My first programming exercise in the course I took in numeric programming in the summer of 1967 was an algorithm that was guaranteed to converge if the starting point was greater than $1/2$ the interval between two points...

Damn! You beat me! Mine was in October 1968. In the year when students across the channel were revolting and tearing down their universities, I was learning to write programs to invert 20x20 matrices (my first computer exercise) using fancy numerical methods in FORTRAN. I reckon I was missing out on something there.

I was in graduate school at CMU. While some of the students had problems with personal hygiene (living in the labs while they were trying to make deadlines, for example), most merely got unpleasant but none were truly revolting.

Anyway: first instruction: call everything REAL*8 (=double). Last instruction: multiply the matrix you first thought of by the one you got, and compare with the unit matrix to see what accuracy was achieved! (Fussy buggers, these numerical maths lecturers!)

Re: Precision issue with Float and Double (C++)

I have a friend who is a physics professor, and who demanded one year that students not use calculators for exams. He got tired of getting answers to eight decimal places that were off by six orders of magnitude, and he was willing to accept integer answers but graded heavily against order-of-magnitude errors.

von Neumann gave us the 36-bit floating point unit, because he said that a 9-bit exponent and 27-bit mantissa provided both sufficient range and sufficient precision for all real problems. That's why machines for so many years used 36-bit words.

..
Note that if you do not handle these properly, your Jupiter probe will miss the planet by several million miles.

Well if you mix up feet and metres, then you can get a even bigger error crash into he surface of Mars – chacun a son gout, as far as errors are concerned :-)

In Larry Niven's "Known Space" series, he uses something called the "Alderson Drive", named in honor of a friend of his who was a JPL programmer. Dan Alderson was The Man for programming your space mission, and especially the embedded computers on the space probe itself. If he programmed your mission, it got to where it was going. And the problem there (which I think von Neumann didn't fully understand) was that cumulative LSB errors really, really, REALLY matter. Alderson (who died from complications of diabetes, I think back in the 1990s) was someone who intimately understood the details of several different floating point representations and knew how to maintain accuracy over, literally, billions and billions of iterations (even a slow computer executes a lot of cycles in seven years of flight).

Joe Traub, former head of the CMU Computer Science Department (and was there for most of my CMU career, which spanned Perlis, Traub, and Habermann), told me that it is not uncommon to have cumulative errors during the middle of a computation which are orders of magnitude greater than the actual answer, but with clever programming, the algorithms would balance out the errors and thus eliminate them in the final answer (this was considered one of the "breakthrough" techniques invented in the early 1970s).

We were involved, in the late 1970s, with people who were using our big (for its era) multiprocessor (16x300KIPS, filled a room, had massive amounts of memory (a couple megabytes, as I recall), and only required a few kilowatts of power to run. They were doing what were called "mesh" computations, and one of the classic problems, as it was described to me, was that the cumulative LSB errors would result in the mesh being so distorted that two lines would cross. When that happened, the computation diverged and was unrecoverable. So every so often, an "equalizing" algorithm was run that "renormalized" the mesh so the computations could resume. This was time-consuming and they didn't want to do it very often, so one technique they were experimenting with on

Re: Precision issue with Float and Double (C++)

their big mainframes was to visually display the mesh, and have a human being watch it; when the mesh points got too close, the human watcher would hit a key that invoked the renormalization.

Numerical errors are a Fact Of Life. During the Manhattan Project, rooms full of (almost always) women (often the wives of the physicists) running Marchant mechanical calculators were doing computations that could diverge, and whole new algorithms were designed to allow massively parallel computation to be done while maintaining numerical accuracy (the precise results required more digits of precision than the calculators possessed). Also, cross-checking and the use of redundant computations were developed. So we had pipelined computations, parallel computations, and redundant computations, all thought of as "modern" concepts in computing, were first implemented by von Neumann and others at Los Alamos in the early-to-middle 1940s.

joe

Dave

Joseph M. Newcomer [MVP]

email: newcomer@xxxxxxxxxxxxx

Web: <http://www.flounder.com>

MVP Tips: http://www.flounder.com/mvp_tips.htm

.