

A Detailed Study of the Numerical Accuracy of GPU-Implemented Math Functions

Dan Fay[1], Ali Sazegari[1], and Dan Connors[2]

[1] Apple Computer, Inc.

[2] University of Colorado

Modern programmable GPUs have demonstrated their ability to significantly accelerate important classes of non-graphics applications; however, GPUs' substandard support for floating-point arithmetic can severely limit their usefulness for general-purpose computing. Current GPUs do not support double-precision computation and their single-precision support glosses over important aspects of the IEEE-754 floating-point standard[1], such as correctly rounded results and proper closure of the number system. Additionally, numerical consistency needs to exist between different GPU vendors, different GPU software platforms (shader language compiler, driver and operating system), and vendors' GPU families.

Previous studies of GPUs' numerical accuracy[2][3][4] quantified only the "overall" accuracy of different arithmetic and math functions on the GPU by providing an average error and/or an error bounds for each operation. Since many algorithms' correctness depends on the precise, consistent results provided by the IEEE-754 floating-point standard[5], it is also essential to quantify the GPUs' correctness for important edge cases[6]. These edge cases deliberately expose numeric errors likely to occur in IEEE-754 implementations, such as inputs that involve denormalized numbers, +/- 0, infinities, and Not a Number (NaN).

To investigate the issues of edge-case correctness and robustness, we tested the accuracy of the basic arithmetic operators (add, subtract, multiply and divide) as well as other important math functions (sine, cosine, tangent, exponential, etc.). These tests were run on a variety of different GPU platforms from both ATi and nVIDIA. For the math functions, we tested the GPUs' results produced using the math functions built in to the OpenGL Shading Language (GLSL)[7] along with the results produced with a GPU port of the high-performance Cephes Math Library[8]. Finally, we compared these results against reference values produced by *libm* as well as against *vForce*, Apple's high-performance vectorized math library[9].

Our results show that there are serious errors with the GPUs' results at certain edge cases, in addition to the incorrect handling of denormalized numbers. One example of this is the incorrect handling of -0. This causes problems with division; for example, +1/-0 should equal -infinity, not +infinity. Another example is the square root, where the `sqrt()` function in GLSL completely ignores the sign of the operand, returning a positive normal number instead of a NaN. Finally, we have observed inconsistencies between GPUs from different vendors. An example of this is with 0/0: the nVIDIA GeForce FX 7300 correctly produces a NaN result, while the ATi x1600 hardware produces an incorrect result of +0.

References:

[1] "IEEE 754: Standard for Binary Floating-Point Arithmetic." Available at <http://grouper.ieee.org/groups/754/> .

[2] Karl E. Hillesland and Anselmo Lastra, "GPU Floating-Point Paranoia." In Proc. GP², August 2004.

[3] "GPUBench Test: Precision." Available at http://graphics.stanford.edu/projects/gpubench/test_precision.html .

[4] Guillaume Da Graca and David Defour, "Implementation of float-float operators on graphics hardware." In Proc. 7th conference on Real Numbers and Computers, July 2006.

[5] David Goldberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic." Available at http://docs.sun.com/source/806-3568/ncg_goldberg.html .

[6] Coonen, Jerome T.: Contributions to a Proposed Standard for Binary Floating-Point Arithmetic. PhD dissertation, Univ. of California, Berkeley, 1984.

[7] John Kessenich, "The OpenGL Shading Language. Available at <http://www.opengl.org/registry/specs/ARB/GLSLangSpec.Full.1.20.6.pdf> .

[8] Steven Moshier, "Cephes Mathematical Library." Available at <http://www.moshier.net/#Cephes> .

[9] "Vector Libraries." Available at http://developer.apple.com/hardware/drivers/ve/vector_libraries.html .