Floating point

Goldsmiths Computing

January 13, 2019

Motivation

Represent a wider range of numbers than with fixed point

- fixed point: constant absolute precision
- floating point: constant relative precision

Convenient approximation to Real numbers

- but only an approximation
- ... some unexpected behaviours too

Definition

Floating point behaviour is defined by an engineering standard:

- IEEE 754 (1985, revised 2008)

Implemented by most hardware platforms:

- floating-point units in CPUs
    - (software support for FPU features varies)
- graphics cards (CUDA, OpenGL)
    - and other coprocessors

Operations

Operations on floats:

- **add** return the sum of two floating point numbers
- **sub** return the difference of two floating point numbers
- **mul** return the product of two floating point numbers
- **div** return the quotient of two floating point numbers
- **sqrt** return the square root of one floating point number

Operations on floating point units:

- **rounding mode** should rounding go towards $+\infty$, 0, $-\infty$ or even?
- **trapping** should the FPU generate an exception for conditions such as overflow or divide by zero?

General idea

Represent a number n as:

$$n = \text{sign} \times \text{significand} \times 2^{\text{exponent}}$$

sign	1 or -1
significand	number in $[1, 2)$
exponent	$\lfloor \log_2 n \rfloor$

Represent this in a fixed-size field using:

sign bit	0 (positive) or 1 (negative)
mantissa	*fractional* part of significand
exponent	exponent + bias

$$n = (-1)^s \times (1 + m) \times 2^{e-B}$$

Single-precision

- 32-bit quantity:
  - 1 sign bit
  - 8 exponent bits
    - bias is 127, range is ± 2^{-126} to 2^{127}
  - 23 mantissa bits
    - plus “hidden bit” gives 24 binary (~7 decimal) digits of precision

Representation

Example

$$0.5 = 1 \times (1 + 0) \times 2^{-1}$$

sign	0
mantissa	0
exponent	126 (`0x7e`)
overall	`0x3f000000`

Zero?

No representation for zero in this scheme

$$\lfloor \log_2(x) \rfloor = -\infty$$

Special representation of zero:

exponent field 0

mantissa 0

sign 0 or 1

Double-precision

- 64-bit quantity:
  - 1 sign bit
  - 11 exponent bits
    - bias is 1023, range is ± 2^{-1022} to 2^{1023}
  - 52 mantissa bits
    - plus “hidden bit” gives 53 binary (~16 decimal) digits of precision

Representation

Example

$$0.75 = 1 \times (1 + 0.5) \times 2^{-1}$$

sign	0
mantissa	shift(1, 51)
exponent	1022 (`0x3fe`)
overall	`0x3fe8000000000000`

Epsilon

Floating point has a larger *range* than *precision*

- calculations with floating points will usually not give an exactly representable answer
    - (even if the input numbers were exact)

Epsilon

ε is the smallest float which you can add to 1.0 and get an answer that isn't 1.0:

single-precision $2^{-24} + 2^{-47}$

double-precision $2^{-53} + 2^{-105}$

For all $0 < x < \varepsilon$

$$1 + x \longrightarrow 1$$

Inverse square root

Game programming history

- need to take $f(x) = \frac{1}{\sqrt{x}}$ often and quickly

Use $\log_2((1 + m) \times 2^{e-B}) \approx e - B + m$:

```
int i = *(int *) &f;
i = 0x5f3759df - (i >> 1);
f = *(float *) &i;
```

This was good in 1999 (*Quake III Arena*)

- nowadays we have hardware to do this
- SSE `rsqrtss`

Work

1. Reading
   - David Goldberg, *What every computer scientist should know about floating point arithmetic*, Computing (1991)