Fast Inverse Square Root

Gavan Fantom

MathsJam 2021

Fast Inverse* What?

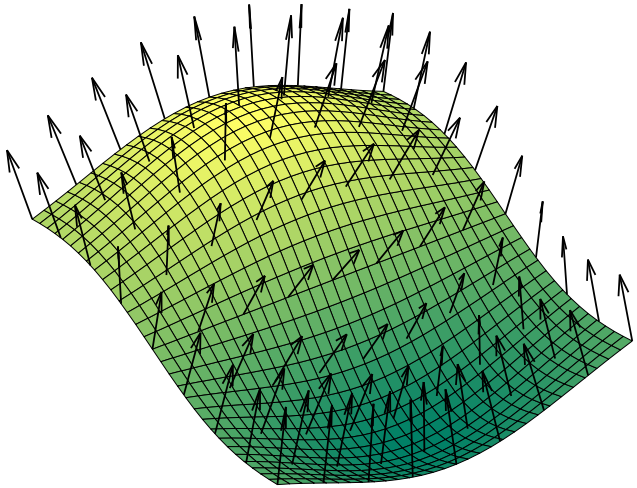
$$f(x) = \frac{1}{\sqrt{x}}$$

... but quickly

* Yes, technically this is the reciprocal

Why Inverse Square Root?

- Real-time Computer Graphics
- Lighting Calculations
- Surface Normals
- Normalise Vectors



$$\hat{\boldsymbol{v}} = \boldsymbol{v} \frac{1}{\sqrt{\|\boldsymbol{v}\|^2}}$$

Why Fast?

- **Computers in the 1990s and early 2000s:**
  - Multiplication: FAST
  - Division: SLOW
  - Square Root: SLOW

- **Games need to perform this operation for every single triangle visible in every single frame – every microsecond counts!**

Quake III Arena

- Released in 1999
- Open sourced in 2005
- One of the most popular PC games of all time
- Contained an unusual way to calculate inverse square roots

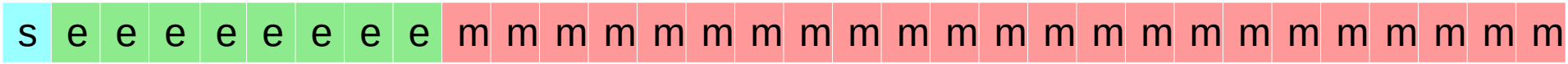
The Code

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                       // evil floating point bit level hacking
    i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//  y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

    return y;
}
```

Two Types of Number

- **Integer**
  - A bit like an integer
  - Signed or unsigned
  - Limited number of digits
- **Floating Point**
  - A bit like a real
  - Signed
  - Exponent
  - Mantissa/significand

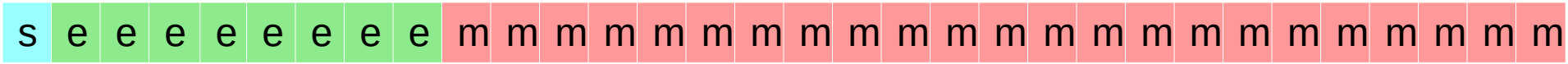
Tricking the Computer



$$n_{float} = (-1)^s(1 + m \cdot 2^{-23}) \cdot 2^{e-127}$$

$$n_{int} = s \cdot 2^{31} + e \cdot 2^{23} + m$$

Tricking the Computer



$$n_{float} = (-1)^s(1 + m \cdot 2^{-23}) \cdot 2^{e-127}$$

$$n_{int} = s \cdot 2^{31} + e \cdot 2^{23} + m$$

$$n_{int} \approx k \cdot log_2(n_{float}) + c$$

Logarithms

$$log_n \frac{1}{\sqrt{x}} = log_n x^{-\frac{1}{2}} = -\frac{1}{2} log_n x$$

Rewriting the algorithm in Maths

$$i_{int} = c_{magic} - \frac{1}{2} n_{int}$$

$$y_0 = i_{float}$$

$$y_1 = y_0(\frac{3}{2} - \frac{1}{2} y_0^2)$$

Rewriting the algorithm in Maths

$$i_{int} = c_{magic} - \frac{1}{2} n_{int} \qquad \approx c_{magic} - \frac{1}{2} k \cdot log_2(n_{float})$$

$$y_0 = i_{float} \qquad \approx 2^{\frac{i_{int}}{k}} + c'_{magic}$$

$$y_1 = y_0(\frac{3}{2} - \frac{1}{2} y_0^2)$$

Rewriting the algorithm in Maths

$$i_{int} = c_{magic} - \frac{1}{2} n_{int}$$

$$y_0 = i_{float}$$

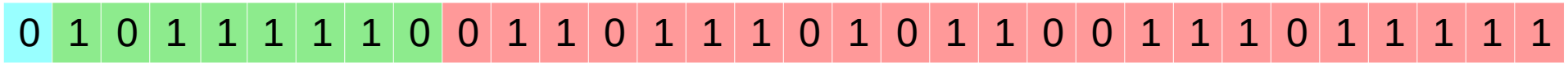
$$y_1 = y_0(\frac{3}{2} - \frac{1}{2} y_0^2)$$

$$\approx c_{magic} - \frac{1}{2} k \cdot log_2(n_{float})$$

$$\approx 2^{\frac{i_{int}}{k}} + c'_{magic} \approx \frac{1}{\sqrt{n_{float}}}$$

So what is the magic constant?

0x5F3759DF



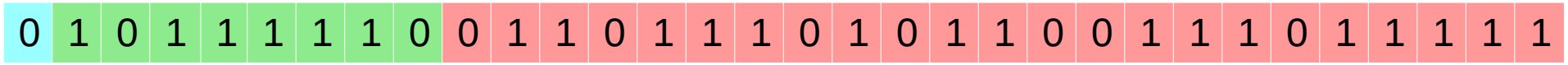
$$c_{magic} = (1 + 3627487 \cdot 2^{-23}) \cdot 2^{63}$$

$$= 1.21621507406235 \cdot 2^{63}$$

$$= 1.12176041049079 \cdot 10^{19}$$

So what is the magic constant?

0x5F3759DF



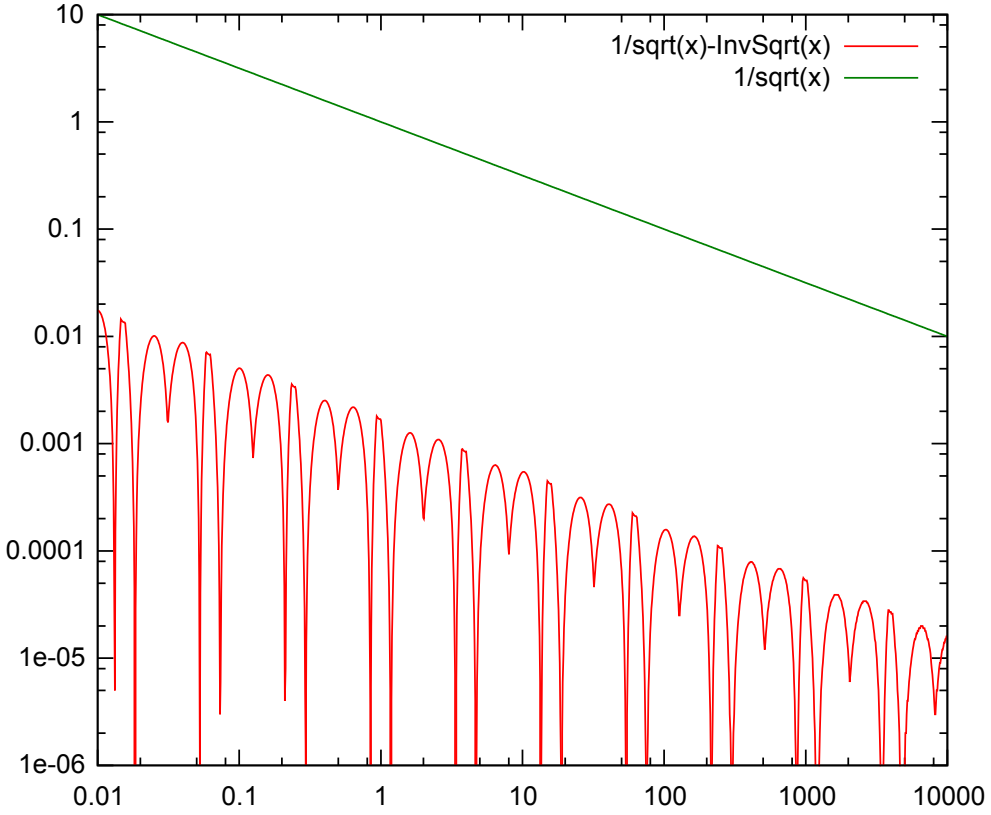
$$\sqrt{2^{127}} = 1.30438178253328 \cdot 10^{19}$$

$$c_{magic} = \sqrt{2^{127}} - correction$$

$$= 1.12176041049079 \cdot 10^{19}$$

Chosen to minimise the error *after* iteration

Error


1/sqrt(x)-InvSqrt(x)
1/sqrt(x)
10
1
0.1
0.01
0.001
0.0001
1e-05
1e-06
0.01
0.1
1
10
100
1000
10000

Fast Inverse Square Root

Gavan Fantom

MathsJam 2021