Fast inverse square root algorithm

A little introduction to the magical algorithm

By Ege Günes, 02.06.2025

Today's agenda

- Computer Games and Graphics
- A simple introduction to importance of vectors in computer graphics
- THE Magical Algorithm
- Analysis of the algorithm

Most of us like playing computer games! Especially the ones with good graphics!

GTA 5 with enhanced resolution

Forza Horizon 5 4K

Quake III Arena (1999)

Why do we use vectors for computer graphics?

- A great data structure to visualise real world
- Very small file sizes -> practically 3 integers from the starting point 0
- Thus ideal for implementing directions -> Ray-tracing
- There is a vast amount of operations we can apply to them

most importantly normalisation!

Normalisation

Normalisation is the process to convert a vector into a unit vector. The original direction is preserved, while having the magnitude(length) of 1.

This is meaningful as the direction of the vector is important rather than the length of it.

The normalisation of a vector is the division of the vector by the square root of the sum if each element squared.

By dividing each component to its length we can get the normalised components.

Mathematically: v * (1/square_root(x^2 + y^2)) => we use 1 / sqrt(some_num)

Excursion: Let's normalise a vector

I am using a (3,4) vector for example. The length of it would be square_root(3^2 + 4^2) = 5.

3/5 would be the x component of our unix vector and 4/5 would be the y respectively

As it can be seen, the direction did not change. For graphics and 3D objects three dimensional vectors are used and the normalisation is implemented in the same way.

Mathematical Background

Mathematically it is easy and fast for computers to calculate huge integers, such as square of a number as addition and multiplication are common operators and implemented to be used very often, thus they are fast. In contrast to the division operation!

$$x^2 + y^2 + z^2 = x*x + y*y + z*z$$

However it is generally not so easy to calculate the square root of something. Especially when there are lots of vectors and surfaces to be normalised.

So it is possible to invent an optimised algorithm for such a problem. Even an approximation of the result makes the computer to save time.

The fast inverse square root algorithm by Quake 3 is such an approximation with 1% error chance and 3 times faster than the commonly used algorithms at its time.

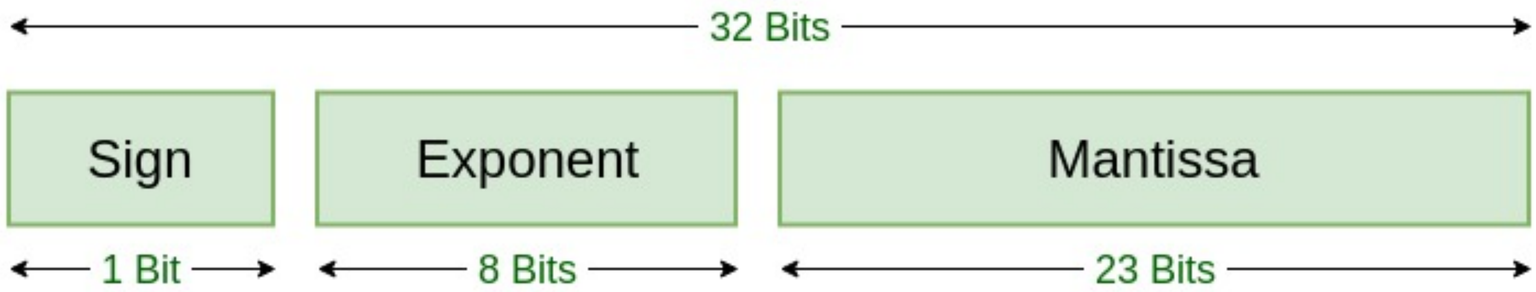
The Algorithm

```
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;
}
```

With the original comments on the code.

Before analysing we should remember some key key concepts

IEEE 754 Standard for Floating point



Single Precision
IEEE 754 Floating-Point Standard

Bit Representation without sign: (2^23 * E) + M

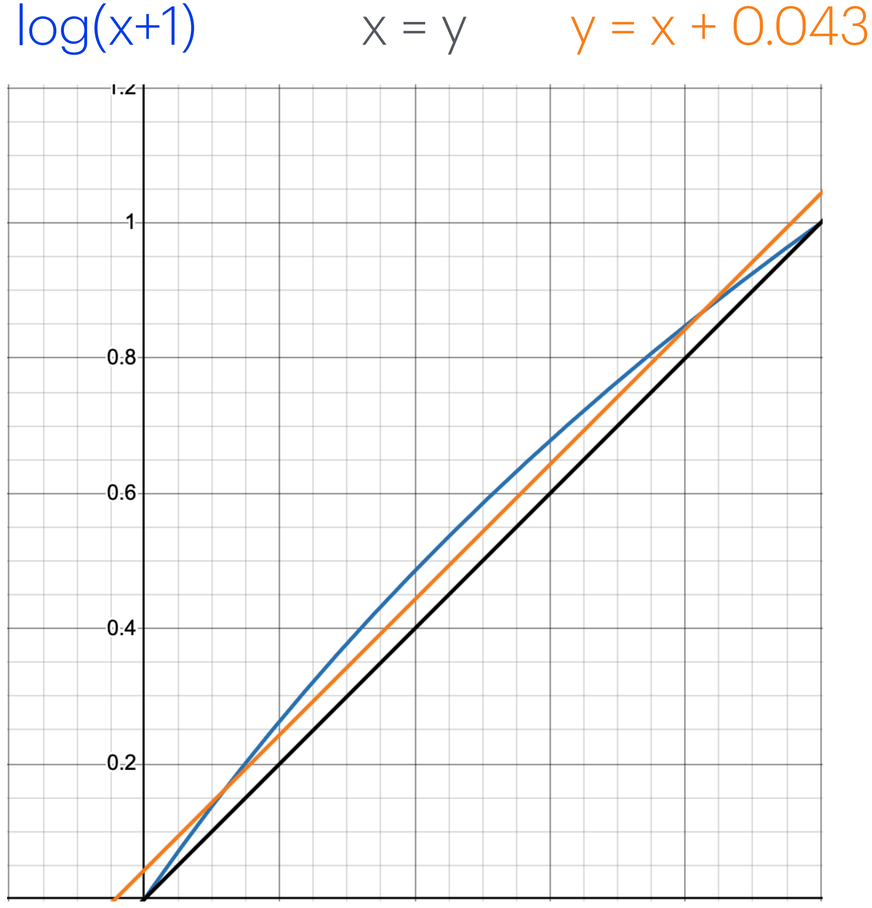
The value of the expression: (-1)^S * ((1+(M/2^23)) * 2^(E-127))

Taking the logarithm of the value

$$\log_2\left(\left(1+\frac{\mathrm{M}}{2^{23}}\right)*2^{\mathrm{E}-127}\right)$$

$$\frac{\mathrm{M}}{2^{23}}+\mu+\mathrm{E}-127$$

$$\frac{1}{2^{23}}\left(\mathrm{M}+2^{23}*\mathrm{E}\right)+\mu-127$$



The bits of a number are in some sense its own logarithm! ...kinda

The evil bit hack

```
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;
}
```

The evil bit hack

0 11011100 01010111010101010010101 : float

```
i  = * ( long * ) &y;
```

01101110 00101011 10101010 10010101 : long

And later on the long will be changed and predicted as long in a reversed way

0 100111011 10101110101010010101 : float

What the f*ck

```
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;
}
```

It is easier to play with logarithms then taking the square root

$$\log\left(\frac{1}{\sqrt{y}}\right) = \log\left(y^{-\frac{1}{2}}\right) = -\frac{1}{2}\log(y)$$

The magical number is a precomputed value, assuring the approximation gives out better results

```
i  = 0x5f3759df - ( i >> 1 );
```

$$\Gamma = \frac{1}{\sqrt{y}}$$

$$\log(\Gamma) = \log\left(\frac{1}{\sqrt{y}}\right)$$

$$\log(\Gamma) = -\frac{1}{2}\log(y)$$

$$\frac{1}{2^{23}}(\mathrm{M}_\Gamma + 2^{23} * \mathrm{E}_\Gamma) + \mu - 127 = -\frac{1}{2}\left(\frac{1}{2^{23}}(\mathrm{M}_y + 2^{23} * \mathrm{E}_y) + \mu - 127\right)$$

$$(\mathrm{M}_\Gamma + 2^{23} * \mathrm{E}_\Gamma) = \frac{3}{2}2^{23}(127 - \mu) - \frac{1}{2}(\mathrm{M}_y + 2^{23} * \mathrm{E}_y)$$

Newton iteration

The newtons update step: ynew = y - f(y)/f'(y)

$$y_{new} = y - \frac{f(y)}{f'(y)}$$

f(y) = 1/y^2 - x

f'(y) = -2/y^3

$$f'(y) = -2/y^3$$

Applying the rule: new = y - (1/y^2 - x)/(-2/y^3) --> y + y/2 * (1 - x*y^2)

$$y_{new} = y - \frac{\frac{1}{y^2} - x}{-2/y^3} = y + \frac{y}{2}(1 - xy^2)$$

Simplification:

$$y_{new} = y\left(\frac{3}{2} - \frac{x}{2}y^2\right)$$

```
y  = y * (threehalfs - (x2 * y * y)); // 1st Newton iteration
```

Resources

- https://app.diagrams.net/
- https://en.wikipedia.org/wiki/Fast_inverse_square_root
- https://www.youtube.com/watch?v=p8u_k2LIZyo
- https://www.youtube.com/watch?v=NCuf2tjUsAY
- https://www.youtube.com/watch?v=uCv5VRf8op0
- https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/
- https://www.youtube.com/watch?v=iAXzw0I3Ncg
- https://store.steampowered.com/app/2200/Quake_III_Arena/?l=german&cc=au
- https://chat.deepseek.com/a/chat/s/281d3e6c-dbe9-499d-ad23-08de4f3a6a8b