Fast Inverse Square Root

David Eberly
Magic Software, Inc.
http://www.magic-software.com

Created: January 26, 2002

The following code is an edited version of code posted to comp.graphics.algorithms on January 9, 2002. The subject that started the thread was *Fast 2D distance approximation* of which the posted code was one of the follow–ups. The posted code is purported to be from Quake3 and provides an approximation to the inverse square root of a number, $1/\sqrt{x}$.

```
float InvSqrt (float x)
{
    float xhalf = 0.5f*x;
    int i  = *(int*)&x;
    i = 0x5f3759df - (i >> 1);  // This line hides a LOT of math!
    x = *(float*)&i;
    x = x*(1.5f - xhalf*x*x);  // repeat this statement for a better approximation
    return x;
}
```

So what does this code really do and what is that magic number 0x5f3759df?

The idea is to specify an x and compute y so that $y = 1/\sqrt{x}$. Define $F(y) = 1/y^2 - x$. The y you want is the positive root of $F(y) = 0$. You can solve this with Newton's method. Choose an initial guess y_0. The iteration scheme is

$$y_{n+1} = y_n - F(y_n)/F'(y_n), \quad n \geq 0$$

where $F'(y) = -2/y^3$ is the derivative of $F(y)$. The equation reduces to

$$y_{n+1} = \frac{y_n(3 - xy_n^2)}{2}.$$

In the limit as you increase n, the y_n values converge to the true value of $1/\sqrt{x}$. If y_0 is a good initial guess, then 1 or 2 iterations should give you a decent approximation. The source code had the second iteration commented out, so I suspect one iteration was good enough for Quake3's purposes.

Now the problem is selecting a good initial guess. This is where the line of code involving x, manipulated as an integer via variable i, is clever. The IEEE 32–bit float has a mantissa M filling bit positions 0 through 22, an 8–bit biased exponent E filling bits 23 through 30, and a sign bit in position 31. The function expects nonnegative input, so the sign bit is 0 for the input x. The bias is 127. The true exponent is $e = E - 127$. The corresponding number in readable form is $x = 1.M * 2^e$. You want y_0 to be a good approximation to $1/\sqrt{x} = (1/\sqrt{1.M}) * 2^{-e/2}$.

The biased exponent for $-e/2$ is $-e/2 + 127$. In terms of integer arithmetic, this is `0xbe - (E >> 1)` where E is the biased exponent for x. Now look at the magic number `0x5f3759df` that shows up in the code. The sign bit is 0. The next 8 bits form the hex number `0xbe`. No coincidence! The statement

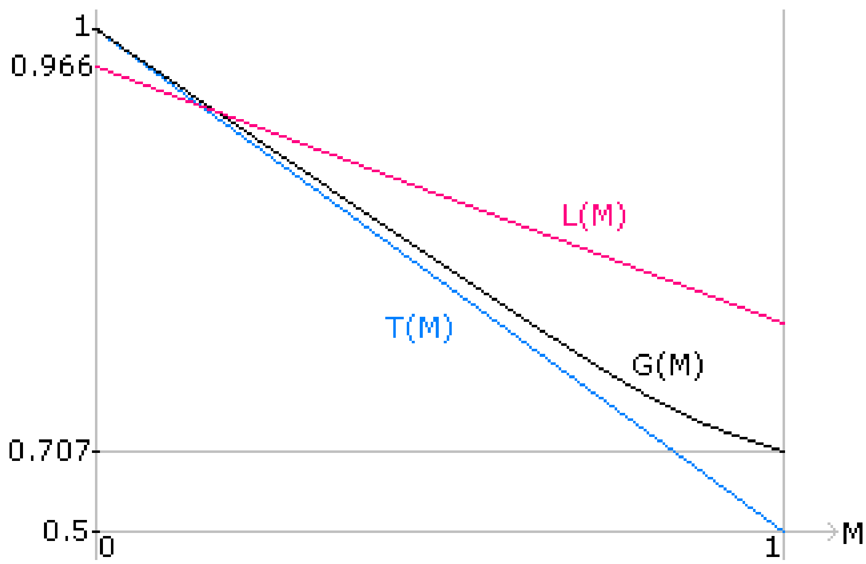
```
i  = 0x5f3759df - (i >> 1);
```

implicitly computes the biased exponent $-e/2 + 127$.

Now you need an approximation for $1/\sqrt{1.M} = 1/\sqrt{1+M}$ where $0 \le M < 1$. Define $G(M) = 1/\sqrt{1+M}$. You can approximate this by a linear function $T(M) = 1 - (M/2)$ using a Taylor series expansion at $M = 0$. The approximation is good for M near zero, but not good at $M = 1$. In fact, as M increases the difference between $G(M)$ and $T(M)$ increases (G is always larger). To try to balance the differences, you want a better fitting line, one that cuts through the graph of $G(M)$. The one corresponding to the posted code is

$$L(M) = 0.966215 - M/4$$

The picture below shows the graph of $G(M)$, the linear function $T(M)$, and the linear function $L(M)$.



For $0 \le M < 1$, $L(M)$ produces numbers in $[0.715215, 0.966215]$ which are not normalized. Instead write

$$L(M) = (1.93243 - M/2)/2.$$

The values $1.93243 - M/2$ are 1+something, so are normalized. The actual floating point representation used for 1.93243 is `0x3ff759df`. When you subtract 1 from the exponent to account for the division by 2 in $L(M)$, you get `0x5f3759df`, the magic number in the code. Therefore, the statement

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

also implicitly computes the mantissa for the initial guess y_0.

I do not know why $0.966215 - M/4$ was chosen in the first place. I thought it might be based on requiring the slope to be $-1/4$ and choosing a constant C using a least squares integral approach that minimizes the integral of squared errors between $C - M/4$ and $1/\sqrt{1+M}$ where the integration is on $0 \leq M \leq 1$, but a quick check showed this is not the case.