Efficiently Computing the Inverse Square Root Using Integer Operations

Ben Self

May 31, 2012

Contents

1 Introduction

In the field of computer science, clarity of code is usually valued over efficiency. However, sometimes this rule is broken, such as in situations where a small piece of code must be run millions of times per second. One famous example of this can be found in the rendering-related source code of the game *Quake III Arena*. In computer graphics, vector normalization is a heavily used operation when computing lighting and shading. For example, a renderer will commonly need to do a shading computation at least once for each pixel on the screen. In the modest case where a game is running at 30 frames per second on a screen with a resolution of 800×600 pixels and only a single vector must be normalized for each shading computation, 14,400,000 vector normalizations will need to be performed per second! To normalize a vector $\mathbf{x}$, one must multiply each component by $\frac{1}{|\mathbf{x}|} = \frac{1}{\sqrt{x_1^2+\cdots+x_n^2}}$. Thus it is important that an inverse square root can be computed efficiently. In *Quake III Arena*, this task is performed by the following function [2] (complete with the original comments):

```
float Q_rsqrt( float number )
{
    const float threehalfs = 1.5F;
    float x2 = number * 0.5F;
    float y  = number;
    long 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 ) );
    return y;
}
```

Unfortunately, while the code performs well and is quite accurate, it is not at all clear what is actually going on, and the comments don't provide much insight. What is the meaning of the constant `0x5f3759df`, and why are integer operations being performed on floating point numbers? This paper will provide a detailed explanation of the operations being performed and the accuracy of the resulting value.

2 Background

2.1 Floating Point Representation

Floating point numbers (defined by `float` in the code) consist of a sign bit, an 8-bit exponent, and a 23-bit mantissa.

s	E	M
bit 31	bits 30...23	bits 22...0

s denotes the sign bit, where 0 corresponds to positive and 1 corresponds to negative. The exponent E is an integer biased by 127, meaning that the interpreted value is 127 less than the actual value stored. This is to allow for both negative and positive values. The mantissa M represents a real number in the range $[0, 1)$; the leading 1 is implied and thus not explicitly included. The value represented by a floating point number is thus given by

$$(-1)^s(1+M)2^{E-127}.$$

2.2 Integer Representation and Operations

Integers (defined by `long` in the code) are represented by a single 32-bit field. Integers use the two's complement representation to allow for both negative and positive values, meaning that bit 31 denotes the sign of the integer; however, for the purposes of this paper, we will only be dealing with positive integers (as the square root is only real for positive values).

The shift right operator on line 7, `>>`, shifts all bits to the right by the specified amount, duplicating bit 31 and truncating bits which are shifted past bit 0. For a positive integer x, $x \texttt{>>} 1$ results in $\lfloor x/2 \rfloor$, as it is equivalent to moving the "binary point" (the equivalent of the decimal point in base 2) to the left by 1 bit and truncating the fraction. Lines 6 and 8 are casting operations which are used to convert between different representations of a value. Line 6 interprets the raw bits of a floating point number as an integer, allowing for integer operations to be performed, and line 8 does the reverse.

3 The Algorithm

Given a number $x > 0$, the algorithm uses Newton's method to approximate $\frac{1}{\sqrt{x}}$. Newton's method is an iterative root-finding algorithm which requires an initial guess y_0. Line 7 computes the initial guess y_0 and line 9 performs a single iteration of Newton's method.

We will begin by assuming that we have a reasonable initial guess and prove the error bounds of Newton's method. Then we will address the details of making the initial guess.

3.1 Newton's Method

Our goal is to find a y which satisfies the equation $\frac{1}{\sqrt{x}} = y$. If we square, invert, then subtract x from each side, we see that this is equivalent to solving $\frac{1}{y^2} - x = 0$, with the restriction that y must be positive. If we let $f(y) = \frac{1}{y^2} - x$, the problem then becomes finding the positive root of $f(y)$. We can approximate the roots of $f(y)$ using Newton's method, an iterative root-finding algorithm requiring an initial guess y_0 of a root, and where

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)}.$$

Geometrically, y_{n+1} can be interpreted as the zero of the tangent line to $f(y)$ at y_n, as shown in Figure 1.

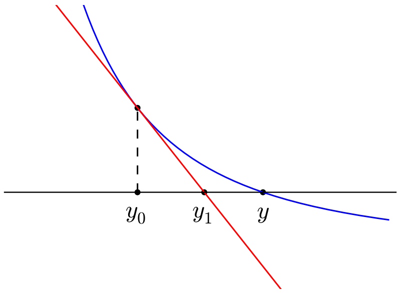

Figure 1: Geometric interpretation of Newton's method.

For a more detailed derivation and convergence analysis of Newton's method, see [3]. For our purposes, we will only examine our particular choice for $f(y)$.

Substituting our function $f(y)$ into the formula for y_{n+1}, we see that

$$y_{n+1} = y_n - \frac{1/y_n^2 - x}{-2/y_n^3} = y_n \left(\frac{3}{2} - \frac{1}{2} x y_n^2 \right),$$

which corresponds to the line

```
y  = y * ( threehalfs - ( x2 * y * y ) );
```

meaning that after the initial guess y_0 is determined, a single iteration of Newton's method is performed to obtain the final result.

Suppose the initial guess y_0 has relative error of at most $\epsilon_0 > 0$, so that $\left| \frac{y-y_0}{y} \right| \le \epsilon_0$. We can rewrite this as

$$y(1-\epsilon_0) \le y_0 \le y(1+\epsilon_0).$$

Therefore, $y_0 = y(1+\delta)$ for some $\delta \in [-\epsilon_0, \epsilon_0]$. By performing a single iteration of Newton's method we obtain

$$\begin{aligned} y_1 &= y(1+\delta) \left(\frac{3}{2} - \frac{1}{2} x \left(y(1+\delta) \right)^2 \right) \\ &= \frac{1}{\sqrt{x}} (1+\delta) \left(\frac{3}{2} - \frac{1}{2} x \left(\frac{1}{\sqrt{x}} (1+\delta) \right)^2 \right) \\ &= \frac{1}{\sqrt{x}} \left(1 - \frac{3}{2}\delta^2 - \frac{1}{2}\delta^3 \right) \\ &= y \left(1 - \frac{3}{2}\delta^2 - \frac{1}{2}\delta^3 \right). \end{aligned}$$

Now let $g(\delta) = y_1$. We can find the maximum relative error after applying Newton's method by minimizing and maximizing $g(\delta)$ for $\delta \in [-\epsilon_0, \epsilon_0]$. The roots of $g'(\delta) = -3y\delta(1+\delta/2)$ occur at $\delta = 0$ and $\delta = -2$, meaning that these are the critical points of $g(\delta)$. However, for small enough ϵ_0 (specifically, for $\epsilon_0 < 2$), we must only consider the point $\delta = 0$ as well as the endpoints $\delta = \pm\epsilon_0$ (we will later see that ϵ_0 is well below 2). Evaluating at these points, we see that $g(0) = y$, $g(-\epsilon_0) = y\left(1 - \frac{3}{2}\epsilon_0^2 + \frac{1}{2}\epsilon_0^3\right)$, and $g(\epsilon_0) = y\left(1 - \frac{3}{2}\epsilon_0^2 - \frac{1}{2}\epsilon_0^3\right)$. Thus, the minimum occurs at $\delta = \epsilon_0$ and the maximum occurs at $\delta = 0$, so

$$y\left(1 - \frac{3}{2}\epsilon_0^2 - \frac{1}{2}\epsilon_0^3\right) < y_1 < y.$$

Therefore, we conclude that if our initial guess y_0 has an initial relative error of at most ϵ_0, by performing an iteration of Newton's method our new relative error becomes at most $\frac{3}{2}\epsilon_0^2 + \frac{1}{2}\epsilon_0^3$, or

$$\left|\frac{y - y_1}{y}\right| \leq \epsilon_1 = \frac{3}{2}\epsilon_0^2 + \frac{1}{2}\epsilon_0^3.$$

3.2 Computing the Initial Guess

We now turn to the algorithm which determines the initial guess y_0. This algorithm relies heavily on the same bit strings being interpreted as both integers and floats. If b is a bit string, then we will denote the sign bit as b_s, the bits contained in the exponent field as b_E, and the bits contained in the mantissa field as b_M. Furthermore, we will write b when we wish to interpret b as an integer or to perform integer operations, and $\mathbf{b}$ when we wish to interpret it as a float or to perform floating point operations. Similarly, this applies to the mantissa b_M: b_M is the integer interpretation and $\mathbf{b_M}$ is interpreted as a real in the range $[0, 1)$.

The code responsible for making the initial guess y_0 is the single following line:

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

For easier analysis, we rewrite these operations in the following way. Suppose we are trying to find the inverse square root of $\mathbf{x}$. Let c be the "magic number" (`0x5f3759df` in the original code), and let $\mathbf{y_0}$ be our initial guess. Then our code becomes:

```
t = x >> 1;
y_0 = c - t;
```

Although c is defined as an integer, we can interpret its bits as a floating point number, and so our notation also applies to c.

3.2.1 Idea Behind the Initial Guess

If we first look at the simpler case where we ignore some of the artifacts that occur when performing integer operations on floating point numbers, we can get an idea of the reasoning behind the the algorithm for computing $\mathbf{y_0}$. Since $\mathbf{x}$ is positive, x_s is 0. Thus the desired value we are trying to approximate with $\mathbf{y_0}$ is

$$\frac{1}{\sqrt{\mathbf{x}}} = \frac{1}{\sqrt{(1+\mathbf{x_M})2^{x_E-127}}} = \frac{1}{\sqrt{1+\mathbf{x_M}}}2^{-(x_E-127)/2}.$$

Recall that performing a shift right operation by 1 divides an integer by 2. Ignoring the issue of a bit being shifted from x_E into x_M, we consider the exponent and mantissa fields to be divided by 2 separately so that $t_E = x_E/2$ and $\mathbf{t_M} = \mathbf{x_M}/2$. If we now treat the subtraction operation $c - t$ separately for the exponent and mantissa fields (meaning that we ignore the possibility that the mantissa "borrows" a bit from the exponent), $y_{0E} = c_E - x_E/2$ and $\mathbf{y_{0M}} = \mathbf{c_M} - \mathbf{x_M}/2$. Thus, $\mathbf{y_0} = (\mathbf{c_M} - \mathbf{x_M}/2)2^{c_E - x_E/2 - 127}$. The value of the exponent field in $c =$ `0x5f3759df` is `0xbe` $= 190$. Substituting this value for c_E, we see that $\mathbf{y_0}$ becomes $\frac{\sqrt{2}}{2}(\mathbf{c_M} - \mathbf{x_M}/2)2^{-(x_E-127)/2}$. By selecting c_E to be 190, the exponent $2^{-(x_E-127)/2}$ is the same in $\mathbf{y_0}$ as it is in $\frac{1}{\sqrt{\mathbf{x}}}$. Therefore, $\mathbf{c_M}$ should be picked appropriately so that $\frac{\sqrt{2}}{2}(\mathbf{c_M} - \mathbf{x_M}/2)$ is a linear approximation of $\frac{1}{\sqrt{1+\mathbf{x_M}}}$.

In practice it is not as straightforward, as the integer shift right and subtraction operations on the exponent and mantissa are not performed separately, meaning that bits can be shifted or borrowed from one field to the other. However, the high level idea of the approximation remains the same.

3.2.2 Detailed Analysis

We proceed by analyzing the following two cases separately:

1. If x_E is even (before bias), then bit 0 of x_E is 0, and the shift right operation divides both the exponent and mantissa by 2 (as the sign bit is 0, the leading bit of the resulting exponent remains 0).

2. If x_E is odd, then bit 0 of x_E is 1, meaning that when performing the shift right operation, a leading 1 is shifted into the mantissa. The exponent and mantissa are divided by 2, but the bit from the exponent bleeding over then adds the value of 1/2 to the mantissa.

First we consider the case when x_E is even. By performing `t = x >> 1`, $t_E = x_E/2$ and $t_M = \lfloor x_M/2 \rfloor$. For simplicity however, we will consider $\mathbf{t_M}$ to be $\mathbf{x_M}/2$, as the resulting error from dropping bit 0 is only 2^{-24}, which is insignificant compared to other errors. First, we notice that c_s must be 0. This is because $x_s = t_s$ is 0, meaning that if c_s were 1, $c - t$ would produce a sign bit of 1, indicating a negative value, which we cannot have. We now consider $c_E - x_E/2$, which has two requirements:

1. $c_E - x_E/2$ must be non-negative. If it were negative, the sign bit would be borrowed from, and because the sign bit is 0, integer wraparound would occur and cause the sign bit to become 1, meaning the resulting float would be negative, which we cannot have.

2. $c_E - x_E/2$ must not be zero. If $c_M - x_M/2$ is negative, a bit will be borrowed from $c_E - x_E/2$, and if $c_E - x_E/2$ is 0, the resulting exponent will become negative, breaking our first condition.

Thus we require that $c_E - x_E/2 \geq 1$. Recall that x_E is an 8-bit unsigned integer (before bias), meaning that it is in the range [0..255]. The values 0 and 255 are reserved for special cases (0, denormalization, ∞, and NaN), so a valid x_E falls into [1..254]. Recalling that we are only dealing with an even x_E, we are further restricted to even integers in [2..254]. Thus, $x_E/2 \in [1..127]$, meaning that c_E must be at least 128.

We now consider the result of subtracting the mantissas. If $\mathbf{c_M} \geq \mathbf{x_M}/2$, then no borrowing from the exponent field occurs, and we immediately obtain the result

$$\mathbf{y_0} = (1 + \mathbf{c_M} - \mathbf{x_M}/2)2^{c_E - x_E/2 - 127}.$$

However, if $\mathbf{c_M} < \mathbf{x_M}/2$, then subtracting the mantissas will cause a bit to be borrowed from the exponent field (as shown above, at least one bit is guaranteed to be available for this purpose as $c_E - x_E/2 \geq 1$), reducing the resulting exponent by 1 and thus dividing the result by 2. Because $\mathbf{c_M} - \mathbf{c_E}/2 < 0$ but the bits in the mantissa still represent a value in the range $[0, 1)$, this will cause the resulting mantissa's value to "wrap around", effectively adding 1 to what would have been a negative result. The resulting mantissa is therefore $1 + \mathbf{c_M} - \mathbf{x_M}/2$ and we obtain the result

$$\mathbf{y_0} = (2 + \mathbf{c_M} - \mathbf{x_M}/2)2^{c_E - x_E/2 - 128}.$$

Rewriting the exponents to be the same in each case, we summarize so far with the following:

$$\mathbf{y_0} = \begin{cases} (2 + 2\mathbf{c_M} - \mathbf{x_M})2^{c_E - x_E/2 - 128}, & \text{if } \mathbf{c_M} \geq \mathbf{x_M}/2 \\ (2 + \mathbf{c_M} - \mathbf{x_M}/2)2^{c_E - x_E/2 - 128}, & \text{if } \mathbf{c_M} < \mathbf{x_M}/2 \end{cases}.$$

Now we consider the case when x_E is odd. This time, by performing `t = x >> 1`, bit 0 of the exponent (which is a 1, as the exponent is odd) is shifted into bit 22 of the mantissa, adding the value of 1/2. Therefore, again ignoring the negligible error of 2^{-24} by treating $\lfloor x_M/2 \rfloor$ as $x_M/2$, we see that $t_E = x_E/2 - 1/2 = (x_E - 1)/2$ and $\mathbf{t_M} = \mathbf{x_M}/2 + 1/2 = (\mathbf{x_M} + 1)/2$. Once again, we see that c_s must be 0 in order to get a non-negative initial guess. Similarly to the case where x_E is even, we again require that $c_E - (x_E - 1)/2 \geq 1$ so that the sign bit does not become 1 and so that a bit is available for the mantissa to borrow from during subtraction if necessary. Once again, as the values 0 and 255 are reserved for x_E and x_E is odd, its range is [1..253], meaning that $(x_E - 1)/2 \in [0..127]$, so again, c_E must be at least 128.

We now look at the result of subtracting the mantissas. If $\mathbf{c_M} \geq (\mathbf{x_M} + 1)/2$, then once again no borrowing from the exponent field occurs and we see that

$$\mathbf{y_0} = (1 + \mathbf{c_M} - (\mathbf{x_M} + 1)/2)2^{c_E - (x_E - 1)/2 - 127}.$$

If $\mathbf{c_M} < (\mathbf{x_M}+1)/2$, we must account for the bit borrowed from the exponent field. As before, the exponent is reduced by 1, dividing the result by 2, and the value of the mantissa wraps around to fall in the range $[0, 1)$, equivalent to adding 1. Therefore in this case,

$$\mathbf{y_0} = (2 + \mathbf{c_M} - (\mathbf{x_M}+1)/2)2^{c_E-(x_E-1)/2-128}.$$

We again summarize our results up to this point, rewriting the exponents to be more consistent:

$$\mathbf{y_0} = \begin{cases} (2+2\mathbf{c_M}-\mathbf{x_M})2^{c_E-x_E/2-128}, & \text{if } x_E \text{ is even, } \mathbf{c_M} \geq \mathbf{x_M}/2 \\ (2+\mathbf{c_M}-\mathbf{x_M}/2)2^{c_E-x_E/2-128}, & \text{if } x_E \text{ is even, } \mathbf{c_M} < \mathbf{x_M}/2 \\ \frac{\sqrt{2}}{2}(2+4\mathbf{c_M}-2\mathbf{x_M})2^{c_E-x_E/2-128}, & \text{if } x_E \text{ is odd, } \mathbf{c_M} \geq (\mathbf{x_M}+1)/2 \\ \frac{\sqrt{2}}{2}(3+2\mathbf{c_M}-\mathbf{x_M})2^{c_E-x_E/2-128}, & \text{if } x_E \text{ is odd, } \mathbf{c_M} < (\mathbf{x_M}+1)/2 \end{cases}.$$

3.2.3 Error of the Initial Guess

We now want to find the relative error of our initial guess $\mathbf{y_0}$. Let $\mathbf{y}$ be the actual value of $\frac{1}{\sqrt{\mathbf{x}}}$. Recall that

$$\mathbf{y} = \frac{1}{\sqrt{\mathbf{x}}} = \frac{1}{\sqrt{(1+\mathbf{x_M})2^{x_E-127}}} = \frac{2^{-(x_E-127)/2}}{\sqrt{1+\mathbf{x_M}}}.$$

Then the relative error of $\mathbf{y_0}$ to $\mathbf{y}$ is $\epsilon_0 = \left|\frac{\mathbf{y}-\mathbf{y_0}}{\mathbf{y}}\right|$. Define ϵ to be $\frac{\mathbf{y}-\mathbf{y_0}}{\mathbf{y}}$. As we have not yet determined the optimal value of c, ϵ depends on c_E, $\mathbf{c_M}$, x_E, and $\mathbf{x_M}$. We again proceed by considering the cases where x_E is even and odd separately.

First suppose x_E is even. Let

$$f_e(\mathbf{x_M}, \mathbf{c_M}) = \begin{cases} 2\mathbf{c_M} - \mathbf{x_M}, & \text{if } \mathbf{c_M} \geq \mathbf{x_M}/2 \\ \mathbf{c_M} - \mathbf{x_M}/2, & \text{if } \mathbf{c_M} < \mathbf{x_M}/2 \end{cases}.$$

Then we can write $\mathbf{y_0}$ as

$$\mathbf{y_0} = (2 + f_e(\mathbf{x_M}, \mathbf{c_M}))2^{c_E-x_E/2-128}.$$

In this case we see that

$$\begin{aligned} \epsilon &= \frac{\mathbf{y}-\mathbf{y_0}}{\mathbf{y}} = 1 - \frac{\mathbf{y_0}}{\mathbf{y}} \\ &= 1 - \frac{(2+f_e(\mathbf{x_M},\mathbf{c_M}))2^{c_E-x_E/2-128}}{\left(\frac{2^{-(x_E-127)/2}}{\sqrt{1+\mathbf{x_M}}}\right)} \\ &= 1 - \sqrt{2}\sqrt{1+\mathbf{x_M}}(2+f_e(\mathbf{x_M},\mathbf{c_M}))2^{c_E-192}. \end{aligned}$$

Now suppose x_E is odd. Let

$$f_o(\mathbf{x_M}, \mathbf{c_M}) = \begin{cases} 4\mathbf{c_M} - 2\mathbf{x_M}, & \text{if } \mathbf{c_M} \geq (\mathbf{x_M}+1)/2 \\ 1 + 2\mathbf{c_M} - \mathbf{x_M}, & \text{if } \mathbf{c_M} < (\mathbf{x_M}+1)/2 \end{cases}.$$

We can again write $\mathbf{y_0}$ as

$$\mathbf{y_0} = \frac{\sqrt{2}}{2}(2 + f_o(\mathbf{x_M}, \mathbf{c_M}))2^{c_E - x_E/2 - 128}.$$

As before, we simplify to see that

$$\epsilon = 1 - \frac{\sqrt{2}}{2}\sqrt{1 + \mathbf{x_M}}(2 + f_o(\mathbf{x_M}, \mathbf{c_M}))2^{c_E - 192}.$$

Combining our results to cover all cases,

$$\epsilon = \begin{cases} 1 - \sqrt{2}\sqrt{1 + \mathbf{x_M}}(2 + f_e(\mathbf{x_M}, \mathbf{c_M}))2^{c_E - 192}, & \text{if } x_E \text{ is even} \\ 1 - \sqrt{1 + \mathbf{x_M}}(2 + f_o(\mathbf{x_M}, \mathbf{c_M}))2^{c_E - 192}, & \text{if } x_E \text{ is odd} \end{cases}.$$

3.2.4 Finding the Optimal Magic Number

We now wish to find a value for c which will minimize the relative error $\left|\frac{\mathbf{y} - \mathbf{y_0}}{\mathbf{y}}\right|$. We first decide on c_E by examining the exponent of $\mathbf{y}$. Once again, recall that

$$\mathbf{y} = \frac{1}{\sqrt{\mathbf{x}}} = \frac{1}{\sqrt{1 + \mathbf{x_M}}}2^{-(x_E - 127)/2}.$$

This result almost looks to be in the form $(\mathbf{1} + \mathbf{y_M})2^{y_E - 127}$. However, the mantissa (including the implied leading 1) must lie in the range $[1, 2)$, and since $\mathbf{x_M}$ lies $[0, 1)$ (as it does not include the implied leading 1), $\frac{1}{\sqrt{1+\mathbf{x_M}}}$ lies between $\sqrt{2}/2$ and 1, and so cannot be a valid mantissa. Furthermore, the exponent must be an integer, and since we don't know whether x_E is even or odd, the division by 2 may cause the exponent to be non-integral. As usual, we will have to look at the even and odd cases separately.

First suppose x_E is even. If we bring a $\sqrt{2}$ out of the exponent, we can rewrite $\mathbf{y}$ as

$$\sqrt{\frac{2}{1 + \mathbf{x_M}}}2^{-(x_E - 126)/2}.$$

Since $\sqrt{\frac{2}{1+\mathbf{x_M}}}$ is now in the proper range for a mantissa and $-(x_E - 126)/2$ is clearly an integer, the expression above is in the proper form and we have found the mantissa and exponent of $\mathbf{y}$. In particular, accounting for bias, $y_E - 127 = -(x_E - 126)/2$, meaning that $y_E = 190 - x_E/2$.

Now suppose x_E is odd. This time we bring a 2 out of the exponent, rewriting $\mathbf{y}$ as

$$\frac{2}{\sqrt{1 + \mathbf{x_M}}}2^{-(x_E - 125)/2}.$$

$\frac{2}{\sqrt{1+\mathbf{x_M}}}$ is again now in the proper range for a mantissa and $-(x_E - 125)/2$ is clearly an integer, so the expression above is in the proper form. Therefore, $y_E - 127 = -(x_E - 125)/2$, meaning that $y_E = 189 - \lfloor x_E \rfloor/2$.

Using right shift notation, we can rewrite these results. If x_E is even, then $y_E = 190 - (x_E >> 1)$, and if x_E is odd, then $y_E = 189 - (x_E >> 1)$. The expressions for y_E are nearly identical to the line of code to compute $\mathbf{y_0}$ except that the code works with all of y_0, c, and x, rather than just the exponent fields. However, as we want y_{0E} to be close to y_E, we have found appropriate values for c_E: 189 or 190. We cannot use both of them, as we must use the same constant for both even and odd cases, so we decide to let $c_E = 190$ as in the original code and continue. This meets our previous requirement that c_E be at least 128.

Having picked c_E, we can now simplify our expression for ϵ:

$$\epsilon = \begin{cases} 1 - \frac{\sqrt{2}}{4}\sqrt{1+\mathbf{x_M}}(2 + f_e(\mathbf{x_M}, \mathbf{c_M})), & \text{if } x_E \text{ is even} \\ 1 - \frac{1}{4}\sqrt{1+\mathbf{x_M}}(2 + f_o(\mathbf{x_M}, \mathbf{c_M})), & \text{if } x_E \text{ is odd} \end{cases}.$$

Our task is now to find a value for $\mathbf{c_M} \in [0, 1)$ such that $\max_{\mathbf{x_M} \in [0,1)}\{|\epsilon|\}$ is minimized. To do this, we first fix $\mathbf{c_M}$ and determine the value (or values) of $\mathbf{x_M}$ which maximize $|\epsilon|$ for that fixed $\mathbf{c_M}$. As usual, we examine the cases for x_E even and x_E odd separately.

First suppose x_E is even. Recall the definition of $f_e(\mathbf{x_M}, \mathbf{c_M})$,

$$f_e(\mathbf{x_M}, \mathbf{c_M}) = \begin{cases} 2\mathbf{c_M} - \mathbf{x_M}, & \text{if } \mathbf{c_M} \geq \mathbf{x_M}/2 \\ \mathbf{c_M} - \mathbf{x_M}/2, & \text{if } \mathbf{c_M} < \mathbf{x_M}/2 \end{cases}.$$

$f_e(\mathbf{x_M}, \mathbf{c_M})$ is clearly continuous and differentiable for $\mathbf{c_M} > \mathbf{x_M}/2$ and for $\mathbf{c_M} < \mathbf{x_M}/2$. If $\mathbf{c_M} = \mathbf{x_M}/2$, we see that $2\mathbf{c_M} - \mathbf{x_M} = \mathbf{c_M} - \mathbf{x_M}/2$, so $f_e(\mathbf{x_M}, \mathbf{c_M})$ is continuous there as well. Thus, $f_e(\mathbf{x_M}, \mathbf{c_M})$ is continuous and piecewise differentiable on $[0, 1]$. The same is true of ϵ, as it is a composition of continuous differentiable functions and $f_e(\mathbf{x_M}, \mathbf{c_M})$. Therefore, its maximums and minimums will occur at critical points or endpoints of its pieces.

First we consider the endpoints. Define $g_1(\mathbf{c_M})$, $g_2(\mathbf{c_M})$, and $g_3(\mathbf{c_M})$ to be ϵ when $\mathbf{x_M}$ is 0, 1, and $2\mathbf{c_M}$, respectively. Then

$$g_1(\mathbf{c_M}) = 1 - \frac{\sqrt{2}}{2}(1 + \mathbf{c_M}),$$

$$g_2(\mathbf{c_M}) = \begin{cases} \frac{1}{2} - \mathbf{c_M}, & \text{if } \mathbf{c_M} \geq 1/2 \\ \frac{1}{4} - \mathbf{c_M}/2, & \text{if } \mathbf{c_M} < 1/2 \end{cases},$$

$$g_3(\mathbf{c_M}) = 1 - \frac{\sqrt{2}}{2}\sqrt{1 + 2\mathbf{c_M}}.$$

Now we consider the critical points. We see that

$$\frac{\partial \epsilon}{\partial \mathbf{x_M}} = \begin{cases} -\frac{\sqrt{2}}{4}\left(\frac{1+\mathbf{c_M}-\mathbf{x_M}/2}{\sqrt{1+\mathbf{x_M}}} - \sqrt{1+\mathbf{x_M}}\right), & \text{if } \mathbf{c_M} > \mathbf{x_M}/2 \\ -\frac{1}{8}\left(\frac{2+\mathbf{c_M}-\mathbf{x_M}/2}{\sqrt{1+\mathbf{x_M}}} - \sqrt{1+\mathbf{x_M}}\right), & \text{if } \mathbf{c_M} < \mathbf{x_M}/2 \end{cases}.$$

The first case is 0 when $\mathbf{x_M} = \frac{2}{3}\mathbf{c_M}$, which also satisfies the condition that $\mathbf{c_M} > \mathbf{x_M}/2$. The second case is 0 when $\mathbf{x_M} = \frac{2}{3}(1 + \mathbf{c_M})$, and the condition

$\mathbf{c_M} < \mathbf{x_M}/2$ can only be satisfied when $\mathbf{c_M} < \frac{1}{2}$. Thus, there is a critical point at $\mathbf{x_M} = \frac{2}{3}\mathbf{c_M}$ and another at $\mathbf{x_M} = \frac{2}{3}(1 + \mathbf{c_M})$ if $\mathbf{c_M} < \frac{1}{2}$, so we define $g_4(\mathbf{c_M})$ and $g_5(\mathbf{c_M})$, corresponding respectively to these critical points, as

$$g_4(\mathbf{c_M}) = 1 - \frac{\sqrt{2}}{2}\left(1 + \frac{2}{3}\mathbf{c_M}\right)^{\frac{3}{2}},$$

$$g_5(\mathbf{c_M}) = \begin{cases} 0 & \text{if } \mathbf{c_M} \geq \frac{1}{2} \\ 1 - \frac{5\sqrt{30}}{36}\left(1 + \frac{2}{5}\mathbf{c_M}\right)^{\frac{3}{2}} & \text{if } \mathbf{c_M} < \frac{1}{2} \end{cases}.$$

Now suppose instead x_E is odd. The definition of $f_o(\mathbf{x_M}, \mathbf{c_M})$ is

$$f_o(\mathbf{x_M}, \mathbf{c_M}) = \begin{cases} 4\mathbf{c_M} - 2\mathbf{x_M}, & \text{if } \mathbf{c_M} \geq (\mathbf{x_M} + 1)/2 \\ 1 + 2\mathbf{c_M} - \mathbf{x_M}, & \text{if } \mathbf{c_M} < (\mathbf{x_M} + 1)/2 \end{cases}.$$

$f_o(\mathbf{x_M}, \mathbf{c_M})$ is continuous and differentiable for $\mathbf{c_M} > (\mathbf{x_M} + 1)/2$ and $\mathbf{c_M} < (\mathbf{x_M} + 1)/2$ and if $\mathbf{c_M} = (\mathbf{x_M} + 1)/2$, $4\mathbf{c_M} - 2\mathbf{x_M} = 1 + 2\mathbf{c_M} - \mathbf{x_M}$. Therefore, $f_o(\mathbf{x_M}, \mathbf{c_M})$ is continuous and piecewise differentiable on $[0, 1]$, as is ϵ, so the maximums and minimums of ϵ will occur at critical points or endpoints of its pieces.

We first consider the endpoints where $\mathbf{x_M}$ is 0, 1, and $2\mathbf{c_M} - 1$, and define $g_6(\mathbf{c_M})$, $g_7(\mathbf{c_M})$, and $g_8(\mathbf{c_M})$ to be ϵ corresponding to these values of $\mathbf{x_M}$, respectively. Then

$$g_6(\mathbf{c_M}) = \begin{cases} \frac{1}{2} - \mathbf{c_M}, & \text{if } \mathbf{c_M} \geq \frac{1}{2} \\ \frac{1}{4} - \mathbf{c_M}/2, & \text{if } \mathbf{c_M} < \frac{1}{2} \end{cases},$$

$$g_7(\mathbf{c_M}) = 1 - \frac{\sqrt{2}}{2}(1 + \mathbf{c_M}).$$

The point $\mathbf{x_M} = 2\mathbf{c_M} - 1$ can only occur if $\mathbf{c_M} \geq \frac{1}{2}$, so we define $g_8(\mathbf{c_M})$ as

$$g_8(\mathbf{c_M}) = \begin{cases} 1 - \sqrt{2\mathbf{c_M}}, & \text{if } \mathbf{c_M} \geq \frac{1}{2} \\ 0, & \text{if } \mathbf{c_M} < \frac{1}{2} \end{cases}.$$

Now we again consider the critical points by finding the zeros of

$$\frac{\partial \epsilon}{\partial \mathbf{x_M}} = \begin{cases} -\frac{1}{4}\left(\frac{1 + 2\mathbf{c_M} - \mathbf{x_M}}{\sqrt{1 + \mathbf{x_M}}} - 2\sqrt{1 + \mathbf{x_M}}\right), & \text{if } \mathbf{c_M} > (\mathbf{x_M} + 1)/2 \\ -\frac{1}{4}\left(\frac{3 + 2\mathbf{c_M} - \mathbf{x_M}}{2\sqrt{1 + \mathbf{x_M}}} - \sqrt{1 + \mathbf{x_M}}\right), & \text{if } \mathbf{c_M} < (\mathbf{x_M} + 1)/2 \end{cases}.$$

The first case is 0 when $\mathbf{x_M} = (2\mathbf{c_M} - 1)/3$, which satisfies the condition $\mathbf{c_M} > (\mathbf{x_M} + 1)/2$ when $\mathbf{c_M} > \frac{1}{2}$. The second case is 0 when $\mathbf{x_M} = (2\mathbf{c_M} + 1)/3$, which always satisfies the condition $\mathbf{c_M} < (\mathbf{x_M} + 1)/2$ (when $\mathbf{c_M} < 1$). We define $g_9(\mathbf{c_M})$ and $g_{10}(\mathbf{c_M})$ corresponding respectively to these critical points as

$$g_9(\mathbf{c_M}) = \begin{cases} 1 - \left(\frac{2}{3}(1 + \mathbf{c_M})\right)^{\frac{3}{2}}, & \text{if } \mathbf{c_M} > \frac{1}{2} \\ 0, & \text{if } \mathbf{c_M} \leq \frac{1}{2} \end{cases},$$

$$g_{10}(\mathbf{c_M}) = 1 - \frac{1}{2}\left(\frac{2}{3}(2+\mathbf{c_M})\right)^{\frac{3}{2}}.$$

We now have a $g_j(\mathbf{c_M})$ covering each possible case where ϵ could be minimized or maximized over all $\mathbf{x_M} \in [0,1)$. Let $h(\mathbf{c_M}) = \max_{1\leq j\leq 10}\{|g_j(\mathbf{c_M})|\}$. Then for fixed $\mathbf{c_M}$,

$$\max_{\mathbf{x_M}\in[0,1)}\{|\epsilon|\} = h(\mathbf{c_M}).$$

Our last step is to determine the value for $\mathbf{c_M} \in [0,1)$ such that $h(\mathbf{c_M})$ is minimized. Using a numerical minimizer for this task, we find the optimal value to be

$$\mathbf{c_M} \approx 0.4327448899640689,$$

giving a maximum error of

$$\epsilon_0 = |\epsilon| \approx 0.03421281.$$

This choice for $\mathbf{c_M}$ corresponds to the value `0x37642f` for the right half of the magic number. This differs from the choice of $\mathbf{c_M}$ for the original magic number, which was `0x3759df`. However, the original choice of $\mathbf{c_M}$ corresponds to a mantissa field of 0.432430148124695, which is not far off from our result.

3.3 Results

We have determined that the maximum relative error of the initial guess y_0 is approximately $\epsilon_0 = 0.03421281$ and that by applying one iteration of Newton's method, the maximum relative error becomes $\epsilon_1 = \frac{3}{2}\epsilon_0^2 + \frac{1}{2}\epsilon_0^3$. Combining these results, we obtain the maximum relative error of our final result,

$$\epsilon_1 \approx 0.0017758.$$

Figure 2 shows a plot of the initial guess compared to the actual value and Figure 3 shows the final result after an iteration of Newton's method.

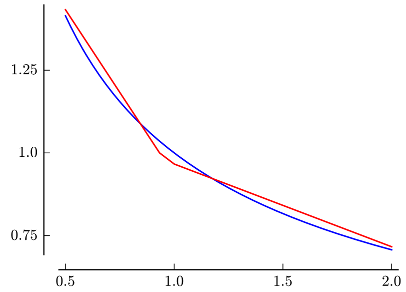

Figure 2: The initial guess (red) compared to $\frac{1}{\sqrt{x}}$ (blue).

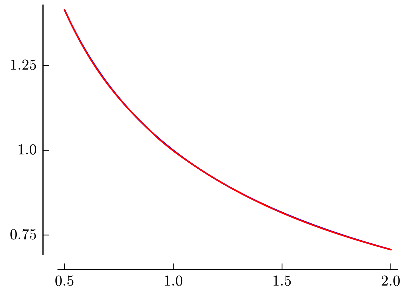

Figure 3: The final result (red) compared to $\frac{1}{\sqrt{x}}$ (blue).

It is worth noting that the original magic number used differs slightly from the value we derived. It is possible that the original value was determined using a criterion other than the minimum of the maximum error. Eberly [1] optimized a variety of different criteria to obtain several other choices for c_M, though none of them were exactly the same as the original. It also may be the case that the optimization to determine the original value for c_M was performed after the iteration of Newton's method, as the true value requiring optimization is the final result, not the initial guess. Although in theory this should produce the same value for c_M, it is easy to overlook the details of floating point math in hardware which may introduce small amounts of error in practice. Notably, by testing all possible floating point values using each choice for c_M, Lomont [4] found that while the initial guess using the value for c_M derived here performed better than the original choice, the new value actually produced a slightly higher maximum error after an iteration of Newton's method. This indicates that Newton's method is in fact a source of error being overlooked. Further analysis would need to be performed to determine why this is the case.

4 Conclusion

Although the fast inverse square root function is difficult to decode at first glance, after a detailed analysis we have a thorough understanding of the motivation behind the method and its inner workings. After analyzing the specific case of computing $x^{-1/2}$ for 32-bit floating point numbers, several questions naturally arise. Is it possible to extend the algorithm to work using 64-bit double precision floats? The answer, fortunately, is *yes,* as the algorithm is not dependent on the length of the bit strings being manipulated. Furthermore, our optimal value for $\mathbf{c_M}$ remains the same as long as we have computed enough digits. Code for a 64-bit version is provided by McEniry in [5]. Another question that arises is whether it is possible to derive similar algorithms which approximate x to powers other than $-1/2$. [5] briefly discusses this in the conclusion, providing some equations as a starting point for $\frac{1}{\sqrt[n]{x}}$, $\sqrt[n]{x}$, and x^a. Notably, we can easily approximate $\sqrt{x}$, since $\sqrt{x} = x\frac{1}{\sqrt{x}}$.

References

[1] David Eberly, *Fast Inverse Square Root (Revisited)*, 2010.

[2] id software, *quake3-1.32b/code/game/q_math.c*, Quake III Arena, 1999.

[3] Lee W. Johnson and R. Dean Riess, *Newton's Method*, Numerical Analysis, 1982, pp. 160–161.

[4] Chris Lomont, *Fast Inverse Square Root*, 2003.

[5] Charles McEniry, *The Mathematics Behind the Fast Inverse Square Root Function Code*, 2007.