**Fast Inverse Square Root Worksheet**

**Name**: Joshua Jardenil

**Favorite Disney Character**: Megamind

There’s not a whole lot of benefit of understanding the bit format of a floating point number, except when peering at numbers in a debugger. However, there still are benefits. This is my favorite example. In 3D graphics, it is very common to *normalize* a three-dimensional vector so that it has a length of 1. For example, if we have a vector u = {3.8, 4.2, 2.9}, the length of u (called |u|) is sqrt(x2+y2+z2), or sqrt(14.44+17.64+8.41) = sqrt(40.46) = 6.36. So to normalize u (called û, or “u-hat”) we divide each component by 6.36 (**multiply by .16**), and get û = {.6, .66, .46}. (You can verify this length is ~1.)

Ultimately, what we’re doing is multiplying u by the inverse square root of the length: u.x = u.x / sqrt(|u|). The problem is that calculating inverse square roots is very expensive unless you have dedicated hardware to solve the problem. Here’s where our bit superpowers come in. With a *single* bitshift and subtraction, we can compute the inverse square root to a decent degree of accuracy.

Here’s the process.

We need **the magic number 0x5F3759DF**: 01011111001101110101100111011111

1) Convert your input float to bits. For example, 40.46 becomes 0 10000100 01000011101011100001010 using the process on the earlier worksheet.

2) Right shift this number once: 00100001000100001110101110000101

3) Calculate the magic number minus the result from #2: 0111110001001100110111001011010

4) Convert this bit pattern back to a float: .16.

Hey, that’s exactly the number we need to multiply by! So with just one right shift and one subtract, you can compute what the inverse square root of 40.46 is!

I recommend not doing this by hand, because, eww, 32-digit arithmetic. Use these tools instead:

[www.calculator.net/binary-calculator.html](http://www.calculator.net/binary-calculator.html)

[www.binaryconvert.com/convert\_float.html](http://www.binaryconvert.com/convert_float.html)

**Problem 1: Normalize u = {2,0,0}** (hint, the answer will be {1,0,0})

1.A: Compute (u.x)2+(u.y)2+(u.z)2 = 4

1.B: Calculate the length of {2,0,0} by taking the sqrt of 1.A = 2

1.C: Compute the inverse square root of 1.A: 0.5

1.D: Multiply each component of u by 1.C to get û: {1, 0, 0}

Now we will do it with the magic number.

1.E: Convert 1.A to a float (in binary): 01000000 10000000 00000000 00000000

1.F: Right shift 1.E by 1 bit: 00100000 01000000 00000000 00000000

1.G: Compute **0x5F3759DF** – 1.F: 01011011001011110101100111011111

1.H: Convert 1.G back to a float: .49356935

1.I: Multiply each component of u to 1.H to get û: {0.98, 0, 0}

Does 1.I match 1.D? Yes

Now do it again for the following vectors:

**Problem 2: Normalize u = {3,4,5}**

|  |  |
| --- | --- |
| 2.A 50 | 2.E 01000010010010000000000000000000 |
| 2.B 5sqrt2 | 2.F 00100001001001000000000000000000 |
| 2.C 1/5sqrt2 | 2.G 00111110000100110101100111011111 |
| 2.D {3/5sqrt2, 4/5sqrt2, 1/sqrt2} | 2.H 0.149 |
| Does 2.D match 2.I? Yes | 2.I {0.447, 0.596, 1.79} |

**Problem 3: Normalize u = {-10,-10,10}**

|  |  |
| --- | --- |
| 3.A 300 | 3.E 01000011100101100000000000000000 |
| 3.B 10sqrt3 | 3.F 00100001110010110000000000000000 |
| 3.C 1/10sqrt3 | 3.G 00111101011011000101100111011111 |
| 3.D <-1/sqrt3, -1/sqrt3, -1/sqrt3> | 3.H 0.577 |
| Does 3.D match 3.I? Yes | 3. {-5.77, -5.77, -5.77} |

**Problem 4: Normalize u = {3.7,13.7,1.3}**

|  |  |
| --- | --- |
| 4.A 179 | 4.E 01000011001100110000000000000000 |
| 4.B 13.379 | 4.F 00100001100110011000000000000000 |
| 4.C 0.075 | 4.G 0111101100111011101100111011111 |
| 4.D {0.225, 0.975, 0.075} | 4.H 0.077 |
| Does 4.D match 4.I? No | 4.I {0.231, 1.001, 0.077} |

**Problem 5: Normalize u = {42,-6.1,22.2}**

|  |  |
| --- | --- |
| 5.A 2284 | 5.E 01000101000011101100000000000000 |
| 5.B 47.791 | 5.F 00100010100001110110000000000000 |
| 5.C 0.021 | 5.G 0.0215 |
| 5.D {0.882, -0.126, 0.462} | 5.H {0.903, -0.129, 0.473} |
| Does 5.D match 5.I? Yes / No | 5.I |