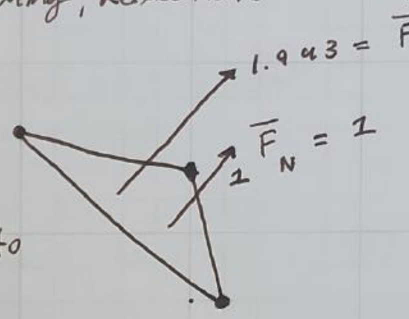
UNIVERSITY OF WYOMING
COLLEGE OF ENGINEERING AND APPLIED SCIENCE

Name: Tyson Limato

Date: 03/07/2024 Class: COSC 3020 Assignment: Wild Card project

- Modern cases
  - ↳ float y = 1/sqrt(x)
    - ↳ include <math.h>
- Traditionally ⟶ fast inverse square Root Quake 3
  - ↳ motivation: Physics, lighting, Reflections all require normalized vectors



① length vector = $\sqrt{x^2+y^2+z^2}$

② scale everything down by the length of vector to normalize said vector

↳ $\frac{x}{\sqrt{x^2+y^2+z^2}}$, $\frac{y}{\sqrt{x^2+y^2+z^2}}$, $\frac{z}{\sqrt{x^2+y^2+z^2}}$

↳ or The dimension multiplied by $\frac{1}{\text{length}}$

③ $x^2+y^2+z^2 \Rightarrow \underbrace{x \cdot x} + y \cdot y + z \cdot z$

Fast multiplication & addition
Slow sqrt function & slow division

↳ especially when we have 1000's of surfaces making up polygons all of which require vectors to be normalized.

↳ fast inverse square root serves as an approximation assuming a "good enough" result for usage in a videogame.

↳ Touts a error of (At most 1%, 3X speed up)

UNIVERSITY OF WYOMING
COLLEGE OF ENGINEERING AND APPLIED SCIENCE

Name: 1

Date: Class: Assignment:

- flow of the function:
  - input: float number
  - internal values: long i [32 bit number], float x2, y [32 bit numbers (decimal)], const float threehalfs [constant 32 bit Decimal]
    - x_2 is then resolved as: $\frac{1}{2}$ input
    - y is resolved as: input

→ The rest is then broken into 3 steps:
① Evil Bit Hack ② What The Fuck ③ Newton Iteration

- Bit Representation {computer organization, Dig. sys. Design, C}
  - in C longs: 00000000 00000000 00000000 00000000
  - in C float: → IEEE 754: 0 00000000 . 00000000000000000000000

    0 ← Sign Bit; 00000000 ← 8 bits for exponent in scientific Notation [128 −127]; . → in binary the only non-zero value is 1; 00000000000000000000000 ← Mantisa
  - ex: given a 23 bit M & a 8 bit exponent
    - $2^{23} \cdot E + M$
    - Derived by: $\underbrace{\left(1+\frac{M}{2^{23}}\right)\cdot 2^{E-127} + x \cdot 2^{0-127}}_{\pm 1.\text{mantisa} \cdot 2^{\text{exponent}}}$
  - if we take the $\log_2$ of $\left(1+\frac{M}{2^{23}}\right)\cdot 2^{E-127}$
    - $\log_2\left(1+\frac{M}{2^{23}}\right) + E - 127$ which simplifies via the approximation $\log_2(1+x) \approx x+\mu$ for small values of x, where μ is our correction coefficient. with $+0.0430 = \mu$ giving the smallest average error
    - $\frac{M}{2^{23}} + \mu + E - 127 \Rightarrow \frac{M}{2^{23}} + \frac{2^{23}E}{2^{23}} + \mu - 127$

      $\Rightarrow \frac{1}{2^{23}}\left(M + 2^{23}\cdot E\right) + \mu - 127$ ∴ Bit Representation of a # is its own logarithm!

① Evil Bit Hack

↳ Input is stored in float y.
Floats by default don't allow us to use Bit manipulation ~~IEEE 754~~
(note: << doubles a # & >> $\frac{1}{2}$ it, odd #'s round)

↳ we want to keep our decimal # & the bits that make it up but in a long without the conversion messing with our bits → just copy bits 1-1 from a float y to a long

↳ Thus we convert the memory address!

$i = *(\text{long } *) \,\&y;$

Assign to i · read bits at address · Convert float address to a long address · address of float

② what the fuck

↳ $y = 13.5435$

$\log(y) \approx i = 01001000 \;\; 00 \cdots\cdots$ bits

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

$= \boxed{-(i >> 1)}$

↳ where does 0x5f3759df come from?

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

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

↳ substitute logarithm with Bit representation

$$\frac{1}{2^{23}}\left(M_\Gamma + 2^{23}\cdot E_\Gamma\right) + \mu - 127 = -\frac{1}{2}\left(\frac{1}{2^{23}}\left(M_y + 2^{23}\cdot E_y\right) + \mu - 127\right)$$

↳ solve for M_Γ & $E_\Gamma \Longrightarrow (M_\Gamma + 2^{23}E_\Gamma) = \frac{3}{2}2^{23}(127-\mu) - \frac{1}{2}(M_y + 2^{23}E_y)$

↳ where $0x5f3759df = \frac{3}{2}2^{23}(127-\mu)$, $\mu = 0.0430$

UNIVERSITY OF WYOMING

COLLEGE OF ENGINEERING AND APPLIED SCIENCE

Name: 3

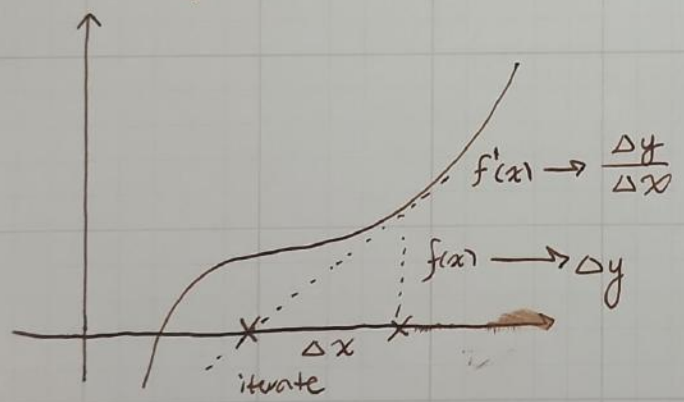
Date: Class: Assignment:

- This results in the complete long estimate: $i = 0x5f3759df - (i >> 1)$;
  - ↳ convert Back to float: $y = *(\text{float }*)\ \&i$;
    Same logic as our long conversion.

③ Newton Iteration

↳ after step ② we have a decent approximation But we have some error terms. Newtons method gives us $f(x) = 0$. This is done by taking an approximation & then returning a better approximation.

↳ The Quake III Developers found that a single iteration gives an error within 1%.



$$\left.\begin{array}{l} \Delta x = \dfrac{\Delta y}{\frac{\Delta y}{\Delta x}} = \dfrac{f(x)}{f'(x)} \\ x - \dfrac{f(x)}{f'(x)} \end{array}\right.$$

↳ New Approximation!

$y = y * (\text{threehalves} - (x2 \cdot y \cdot y))$;

↳ $0 = \frac{1}{y^2} - x \implies y = \frac{1}{\sqrt{x}}$

↳ $y \cdot \left(\frac{3}{2}x - (x \cdot y^2)\right) \longrightarrow x_n = x - \frac{f(x)}{f'(x)}$

To-Do:

- Code simple function & measure Runtime ☐
- Write out Time complexity of All the steps & Both the fast inverse sqR & the simplified ☐
- Compare time to the math.h implementation of 1/sqr(x) ☐
- Do the manual Bit math to show difference in iteration count ☐