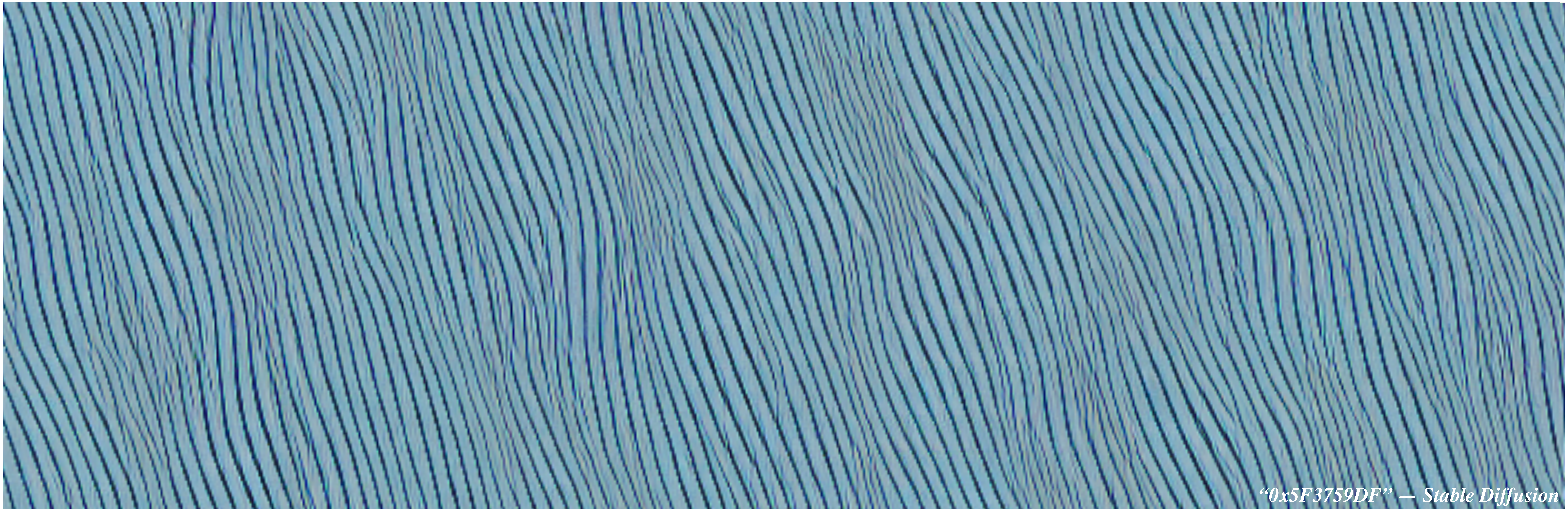


Adam Hyland (adampunk.com)

What's in a number?

Following the Fast Inverse Square Root and its "magic" constant

I will not be explaining the code

Various explanations available at 0x5f37642f.com

Quake 3 Reciprocal Square Root: The Fun Parts

Jerome Coonen
16 April 2022

My favorite is Jerome Coonen's

Roadmap

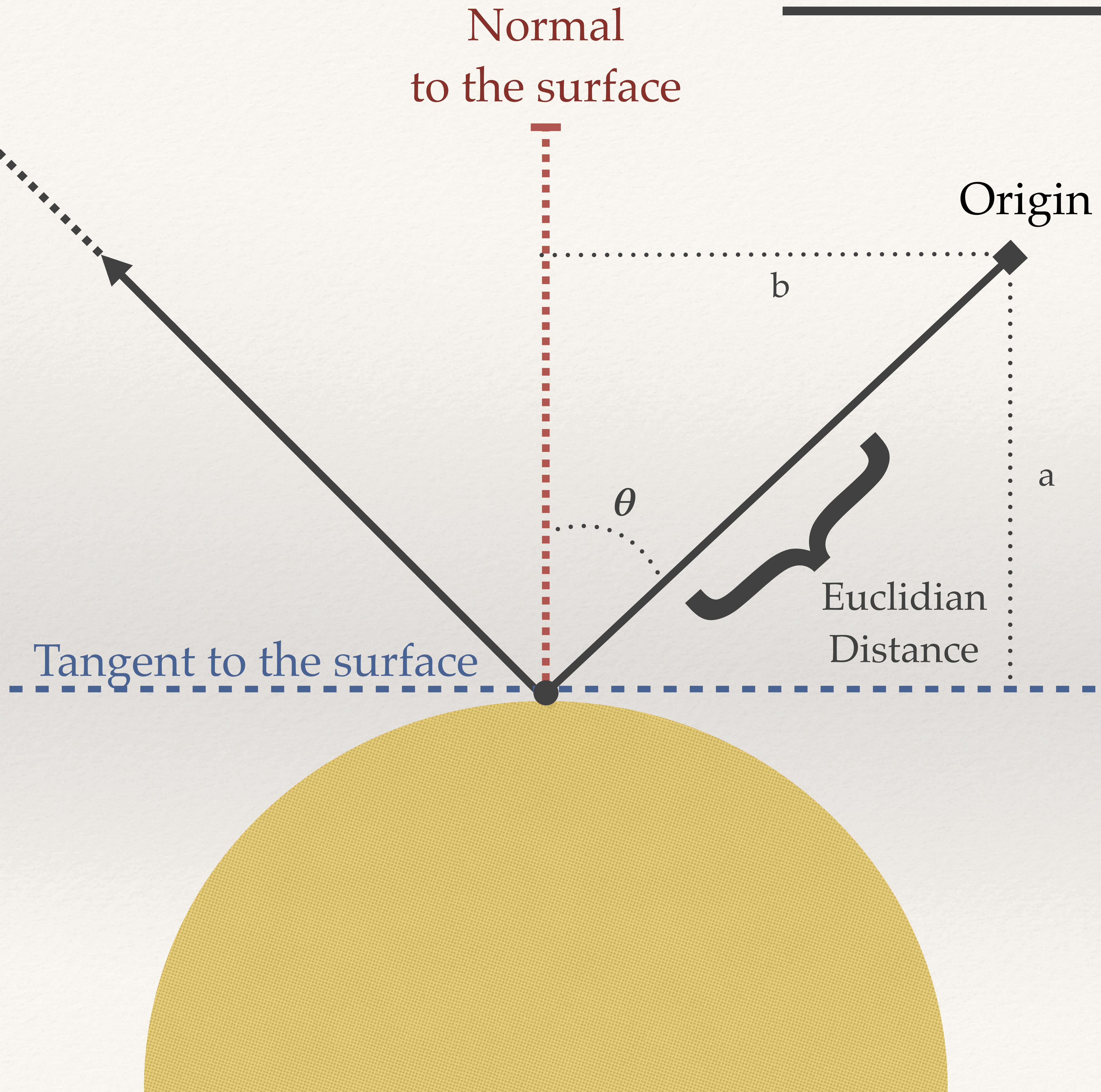
The problem space

Where I come in

Why this is interesting

IEEE-754
Corrupted Stable Diffusion

Reflection off a surface*



Euclidian distance $= \text{sqrt}(a^2 + b^2)$

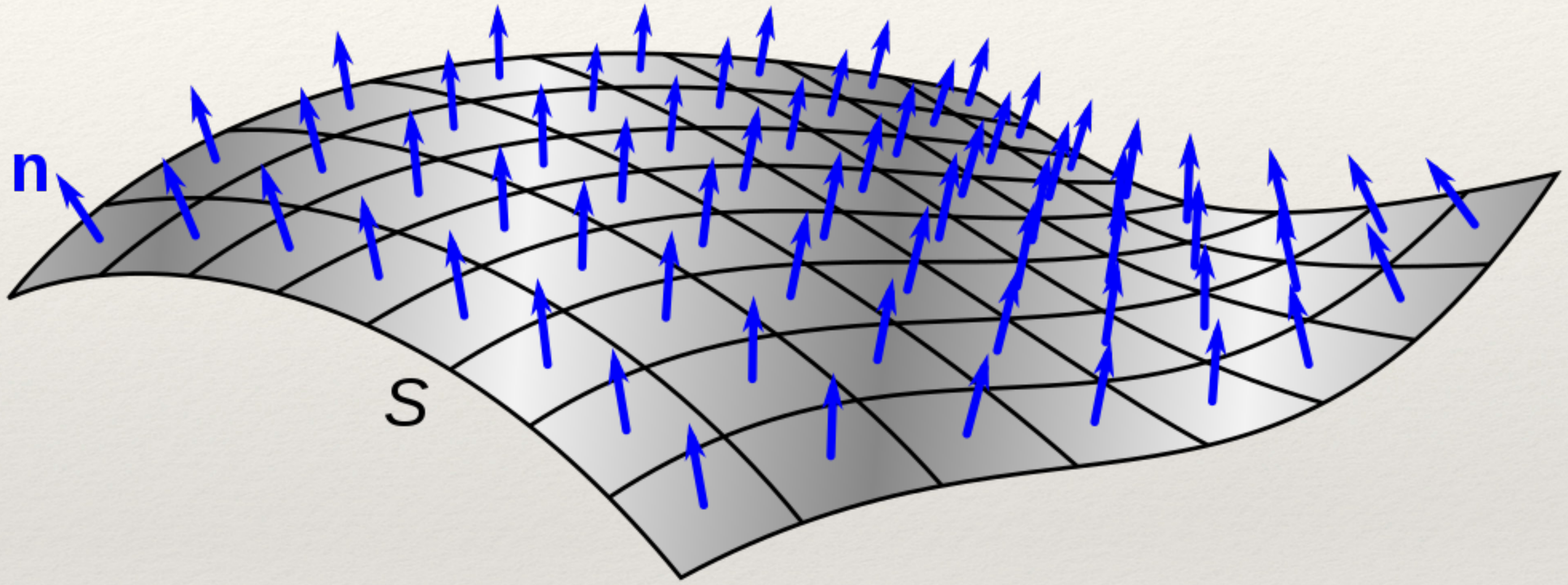
$\sin(\theta) = b / \text{sqrt}(a^2 + b^2)$

* This example is two dimensional where the "normal" to the plane is just (0, 1)

Not Just Lighting

Computation of distance metrics and surface normals is ubiquitous in many arenas:

- Statistics
- Signal processing
- Robotics
- Simulation
- Etc.



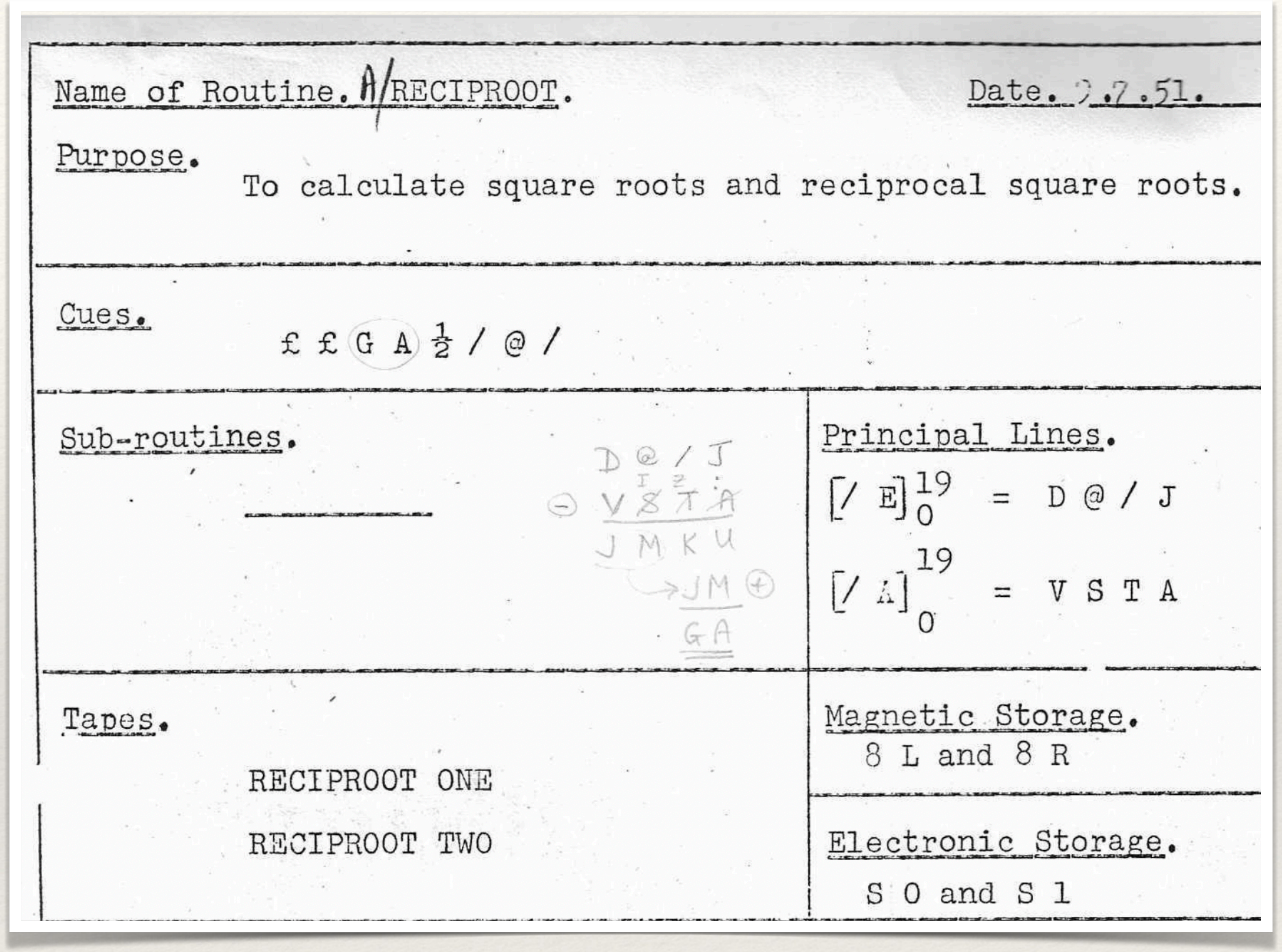
Wikimedia Commons contributors, "File:Normal vectors on a curved surface.svg," Wikimedia Commons, the free media repository, (accessed November 6, 2022).

In a software library in 1951 *

Cecily Popplewell wrote one of the first software libraries on the Manchester Mark I. Ten functions were named in operating manual.

- "half were for input/output and half were mathematical functions." (Campbell-Kelly 1980 p. 145)
- The RECIPROOT routine was one of the five mathematical functions.

A similar routine was written in one of the earliest floating -point schemes, FLOATCODE, for the next version of the Mark I


Name of Routine. A/RECIPROOT. Date. [illegible].7.51.

Purpose. To calculate square roots and reciprocal square roots.

Cues. £ £ G A ½ / @ /

Sub-routines.

Principal Lines.
[/ E]$_{0}^{19}$ = D @ / J
[/ A]$_{0}^{19}$ = V S T A

Tapes.
RECIPROOT ONE
RECIPROOT TWO

Magnetic Storage.
8 L and 8 R

Electronic Storage.
S 0 and S 1

Reciproot routine for the Manchester Mark I, September 1951

* The same year Grace Hopper and her team developed the first compiler

Square Root is difficult to do in hardware

Support for square root limited even deep into the 1990s

Floating-point standard could not require a hardware implementation in 1985*

Table 1. Performance of Recent Microprocessor FPU's for Double-Precision Operands (* = inferred from available information; † = not supported)

Design	Cycle Time [ns]	Latency [cycles]/Throughput [cycles]			
		$a \pm b$	$a \times b$	$a \div b$	$\sqrt{a}$
DEC 21164 Alpha AXP	3.33 ns	4/1	4/1	22–60/22–60*	†
Hal Sparc64	6.49 ns	4/1	4/1	8–9/7–8	†
HP PA7200	7.14 ns	2/1	2/1	15/15	15/15
HP PA8000	5 ns	3/1	3/1	31/31	31/31
IBM RS/6000 POWER2	13.99 ns	2/1	2/1	16–19/15–18*	25/24*
Intel Pentium	6.02 ns	3/1	3/2	39/39	70/70
Intel Pentium Pro	7.52 ns	3/1	5/2	30*/30*	53*/53*
MIPS R4400	4 ns	4/3	8/4	36/35	112/112
MIPS R8000	13.33 ns	4/1	4/1	20/17	23/20
MIPS R10000	3.64 ns	2/1	2/1	18/18	32/32
PowerPC 604	10 ns	3/1	3/1	31/31	†
PowerPC 620	7.5 ns	3/1	3/1	18/18	22/22
Sun SuperSPARC	16.67 ns	3/1	3/1	9/7	12/10
Sun UltraSPARC	4 ns	3/1	3/1	22/22	22/22

Peter Soderquist and Miriam Leeser. 1996. Area and performance tradeoffs in floating-point divide and square-root implementations. ACM Comput. Surv. 28, 3 (Sept. 1996), 518–564. https://doi.org/10.1145/243439.243481

* National Semiconductor's software √ implementation was supplied to help secure their support for the standard

Some implementations are suspect

```
/*
 * linux/kernel/math/sqrt.c
 *
 * (C) 1991 Linus Torvalds
 */

/*
 * simple and stupid temporary real fsqrt() routine
 *
 * There are probably better ways to do this, but this should work ok.
 */
```

Linux system sqrt

```
                Sgl_leftshiftby1(src);
        }
        /*
         * Add comment here.  Explain following algorithm.
         *
         * Trust me, it works.
         *
         */
        Sgl_setzero(result);
```

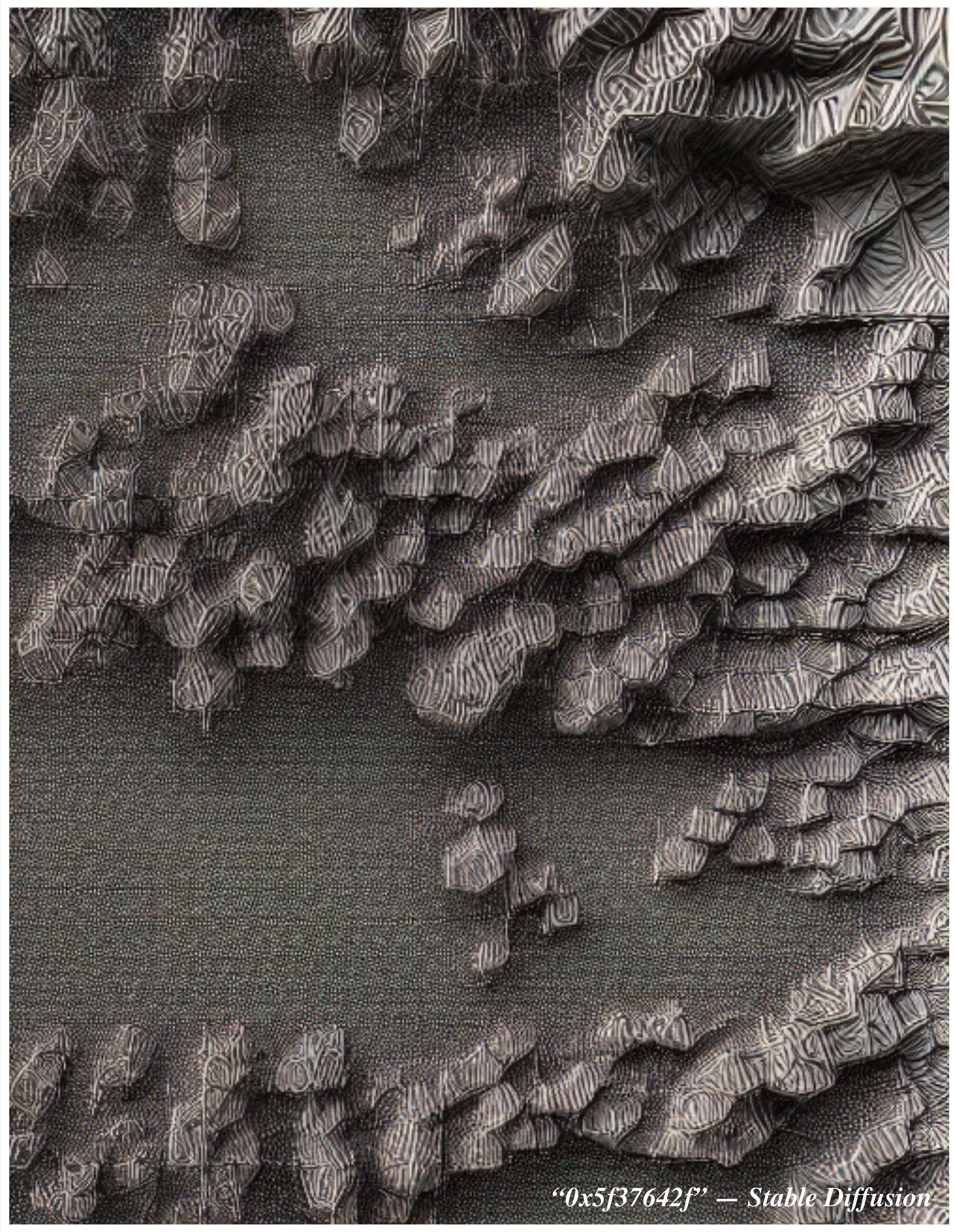
Unix integer square root

A good approximation is *useful*

Square root and 1/sqrt are in the critical path

- Speed
- Timing

Important to a wide variety of use cases



"0x5f37642f" — Stable Diffusion

Useful things are sometimes cargo culted

In times of need

```
        length = 0;
        for (i=0 ; i< 3 ; i++)
                length += v[i]*v[i];
        length = sqrt (length);         // FIXME

        return length;
}
```

Quake II Source code, 1997

Enter the Fast Inverse Square Root

```c
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;
}
```

Quake III source code, 1999

I said I wouldn’t explain but…

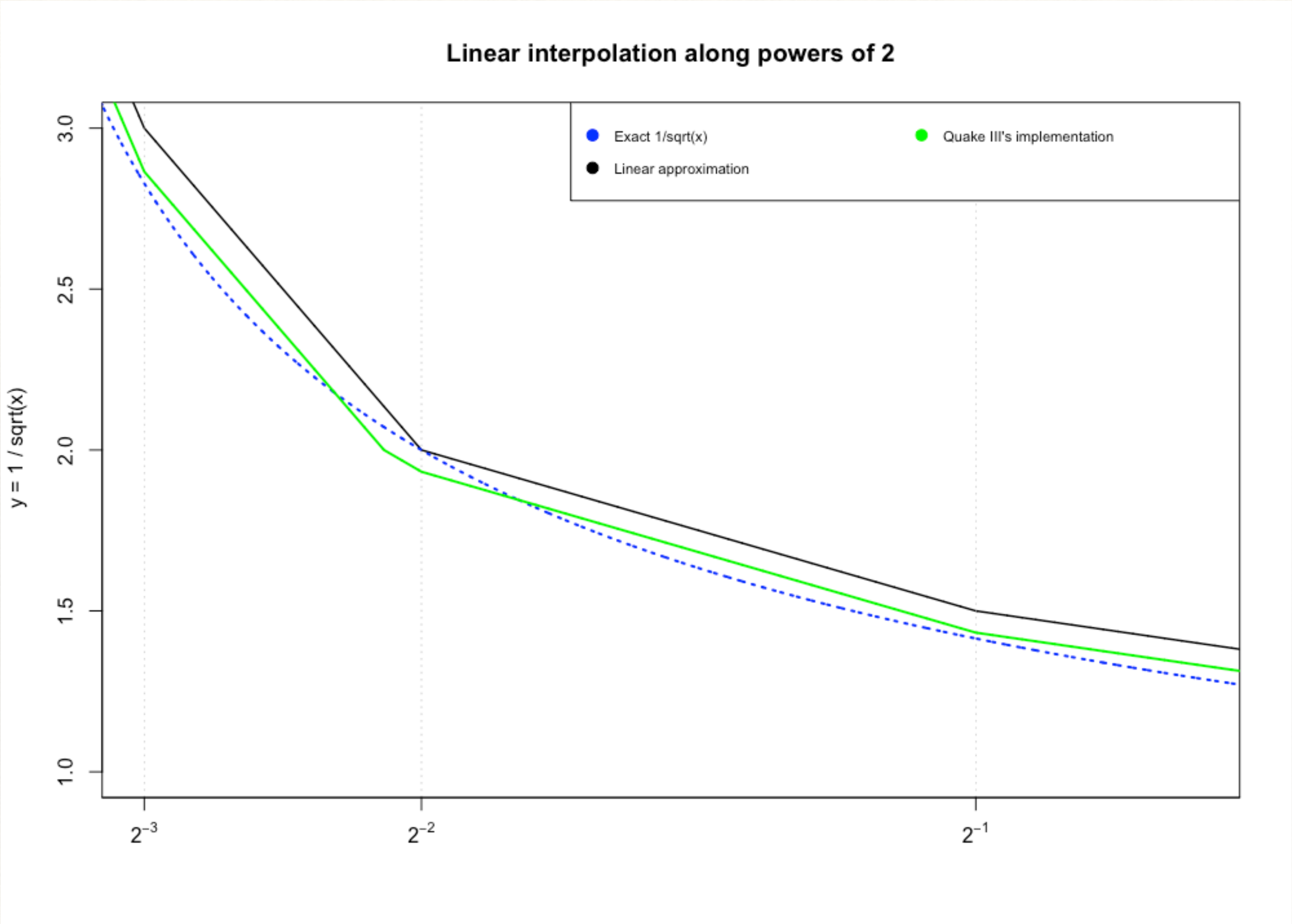
```
i  = * ( long * ) &y;  1
i  = 0x5f3759df - ( i >> 1 );  2
y  = * ( float * ) &i;  1
y  = y * ( threehalfs - ( x2 * y * y ) );  3
```

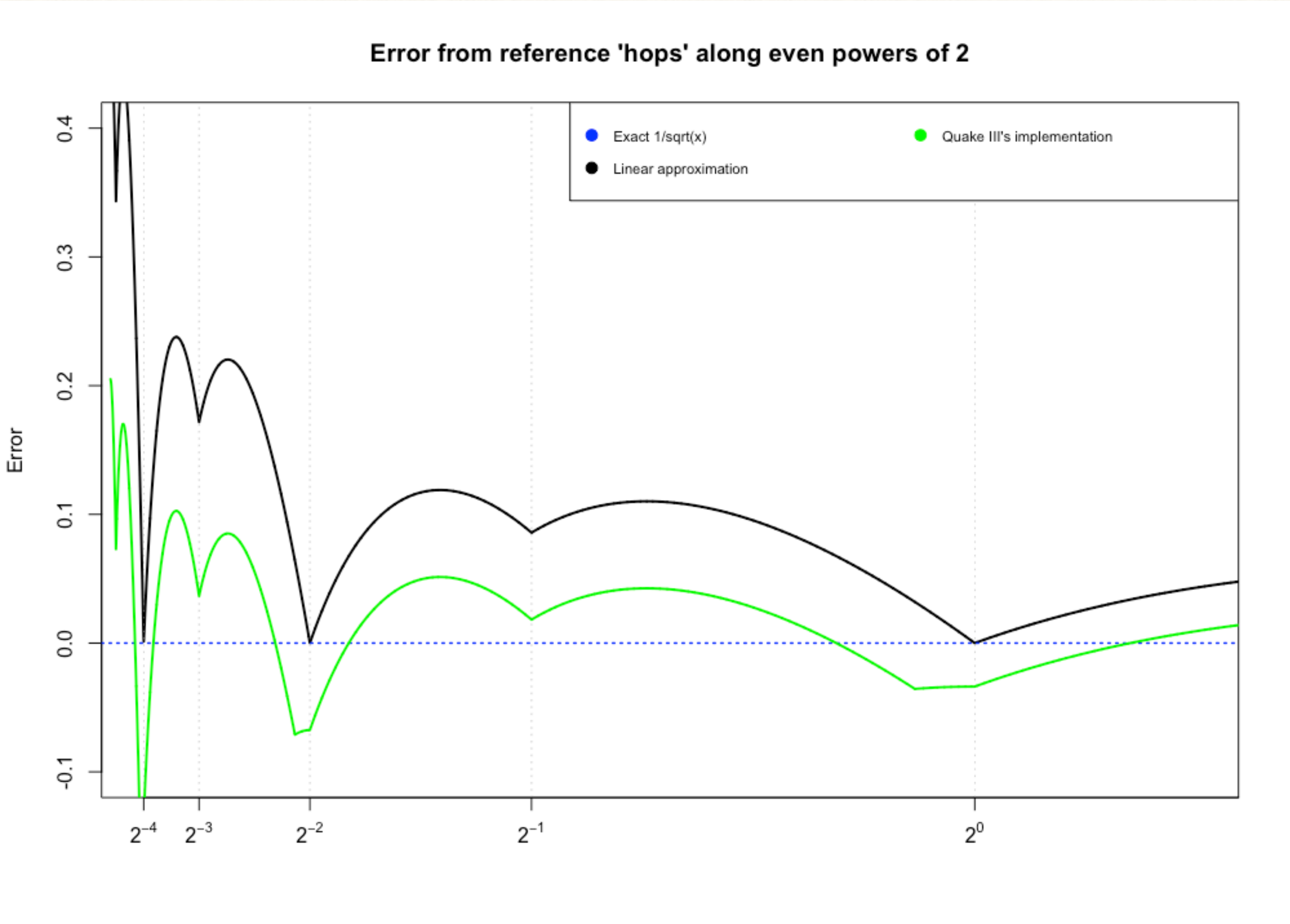
For our purposes, the FISR has three components:

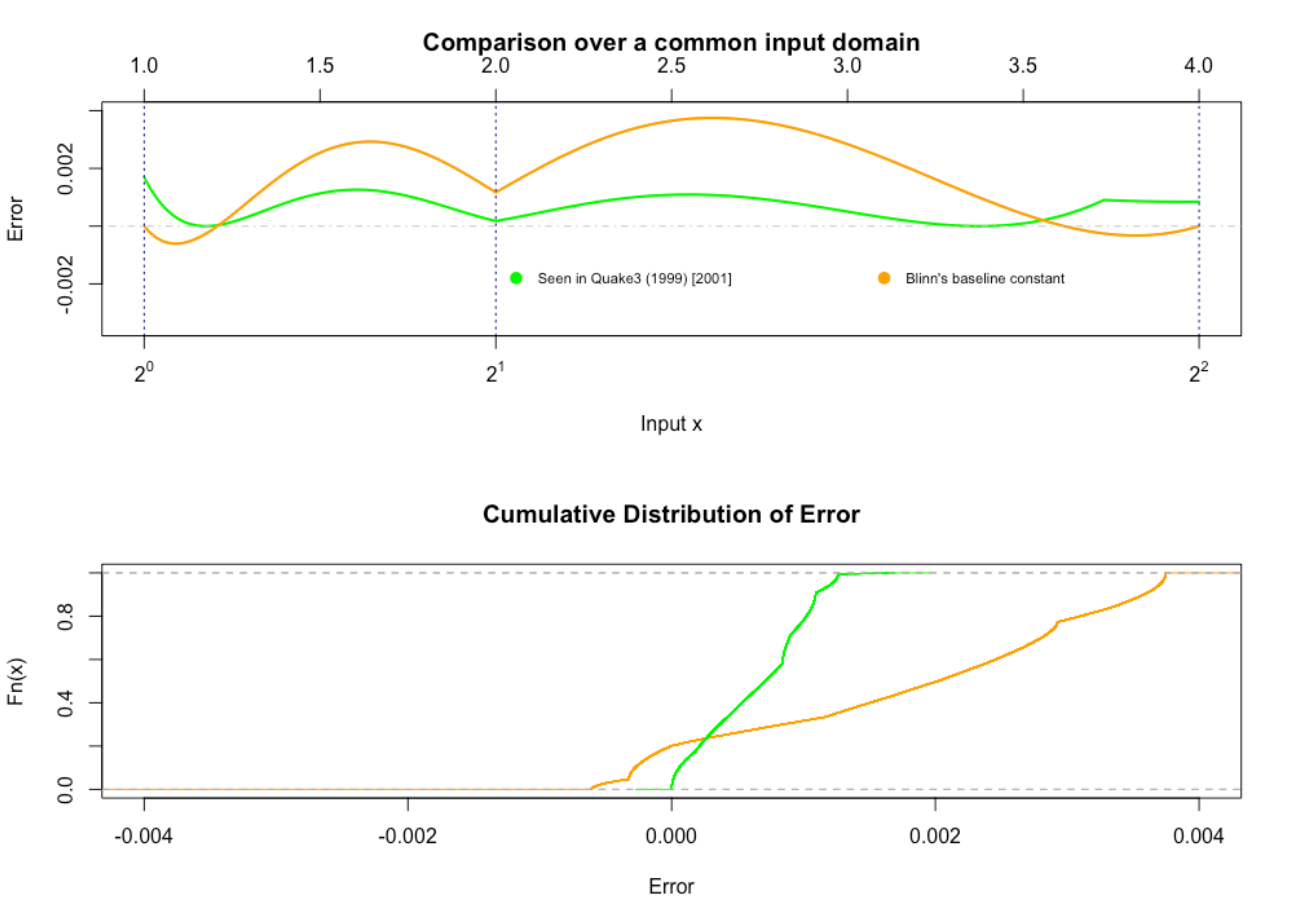
1. A transformation of the input to allow for approximate division of the logarithm of the input by two (and back again)
2. A constant which the above is subtracted from
3. A final step which uses an iterative algorithm to converge on the output

All three together net a close approximation to 1 / sqrt(x)

What does this approximation look like?







Dramatic accuracy improvements

With an input scaled to

$1 < x < 4$

Dramatic improvement over a 'naive' constant

That's all well and good, but…

- (cur | prev) ◯ 17:21, 24 October 2022 Adakiko (talk | contribs) m . . (34,141 bytes) (+6) . . *(Reverted edits by X922073 (talk): unexplained content removal (HG) (3.4.10))* (undo) *(Tag: Rollback)*
- (cur | prev) ◯ 17:20, 24 October 2022 X922073 (talk | contribs) m . . (34,135 bytes) (–6) . . *(remove `fuck`)* (undo) *(Tags: Reverted, Visual edit)*

- (cur | prev) ◯ 14:34, 24 June 2021 David Eppstein (talk | contribs) . . (34,414 bytes) (0) . . *(Undid revision 1030180714 by 94.1.114.3 (talk) Not this shit again. Please do not change the direct quote. See WP:NOTCENSORED.)* (undo | thank) *(Tag: Undo)*
- (cur | prev) ◯ 10:50, 24 June 2021 94.1.114.3 (talk) . . (34,414 bytes) (0) . . *(full word "Fuck" inappropriate in article. changed to f**k)* (undo) *(Tags: Reverted, Visual edit)*

- (cur | prev) ◯ 11:17, 30 August 2018 NickyMcLean (talk | contribs) . . (28,812 bytes) (+58) . . *(Undid revision 857222390 by 37.115.28.79 (talk) This issue has been discussed before and appears in the original source, though I'm not sure about the "evil")* (undo) *(Tag: Undo)*
- (cur | prev) ◯ 10:08, 30 August 2018 37.115.28.79 (talk) . . (28,754 bytes) (–58) . . *(Removed profanity and non-useful text from the comments to the code)* (undo) *(Tag: Visual edit)*

```
// evil floating point bit level hacking
// what the fuck?
```

Forest Garden
Corrupted Stable Diffusion

Was it "from" Quake III?

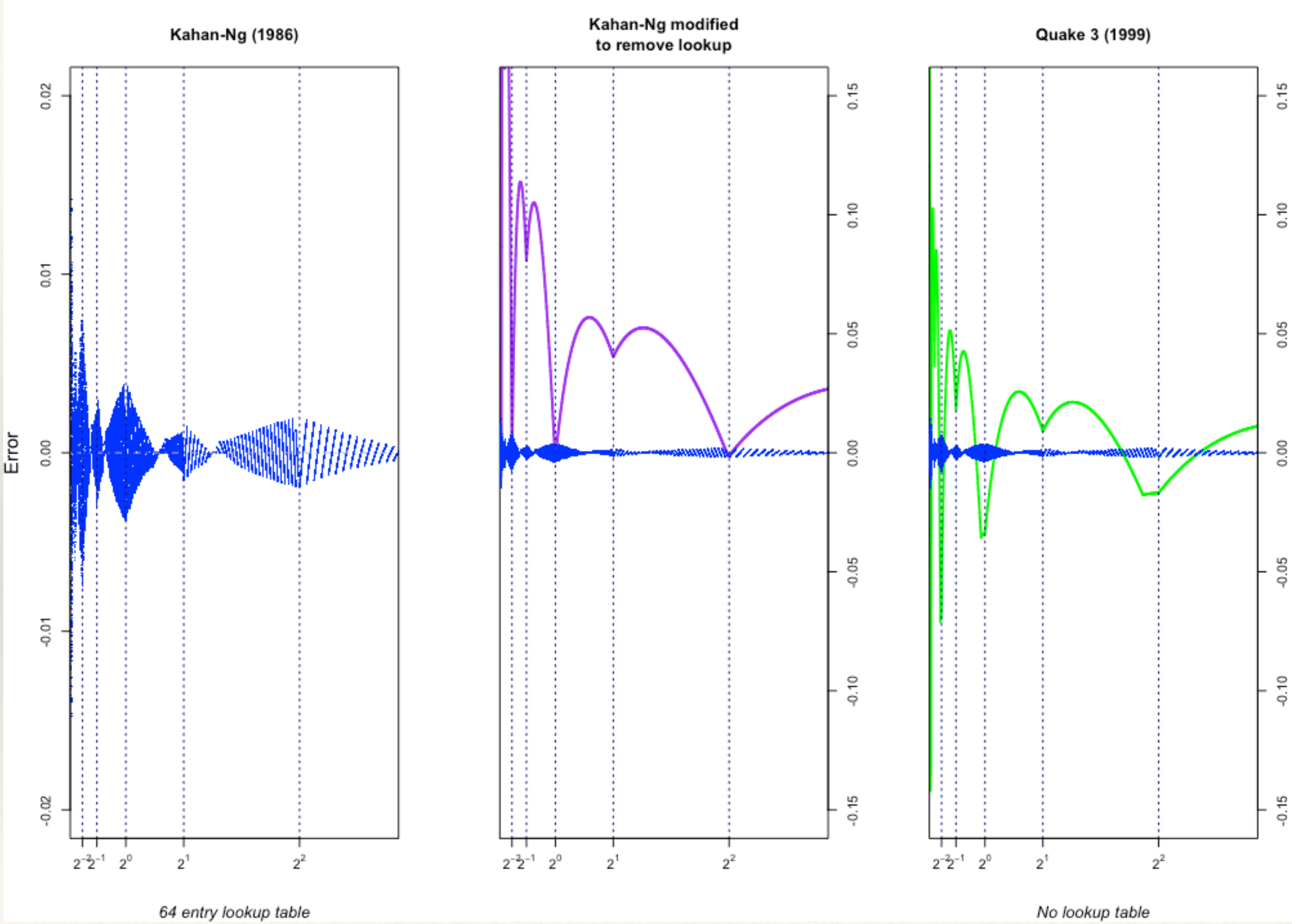
87. Beyond the examples above, Copilot regularly Output's verbatim copies of Licensed Materials. For example, Copilot reproduced verbatim well-known code from the game Quake III, use of which is governed by one of the Suggested Licenses—GPL-2.[17]

Litigation against Github Co-Pilot, October 17, 2022

"Floating-point arithmetic" — Stable Diffusion

Partial view of a citation network

How can we tell they are connected?



Kahan-Ng function breakdown available on github

No. Really. How do we know they are connected?

JH Joao Henriques on 26 Jun 2012

It's amazing how one can squeeze so much out of so limited machine instructions! I particularly liked reading about the math that in the end produced this neat approximation. It reminds me of Carmack's (much hackier) inverse square root trick (http://en.wikipedia.org/wiki/Fast_inverse_square_root).

Reply 0

Cleve Moler STAFF on 27 Jun 2012

Jotaf -- Thanks very much for your comment, and for reminding me about the fast inverse square root hack. I didn't realize that the trick had attained a kind of cult status in the graphics community. The trick uses bit-fiddling integer operations on a floating point number to get a good starting approximation for Newton's iteration. The Wikipedia article that you link to describes the trick in great detail, and also links to an article by Rys Sommefeldt about its origins. Sommefeldt goes back to the late '80s and to me and my colleague Greg Walsh at Ardent Computer. I actually learned about trick from code written by Velvel Kahan and K.C. Ng at Berkeley around 1986. Here is a link to their description, in comments at the end of the fdlibm code for sqrt. http://www.netlib.org/fdlibm/e_sqrt.c . -- Cleve

Reply 0


re-use (with or without citation), reimplementation,
inspiration, passage , …
IBM Share
~1965
Software √
for the Proposed
IEEE Floating-Point
Standard
(1980)
Coonen
(1984)
Mark I
Reciproot
Kahan on IBM 7090
(1962)
Kahan's Magic SQRT
on PDP-10 (Post 1970)
Paper
Implementation
Passage

Embedded a few different ways

1962

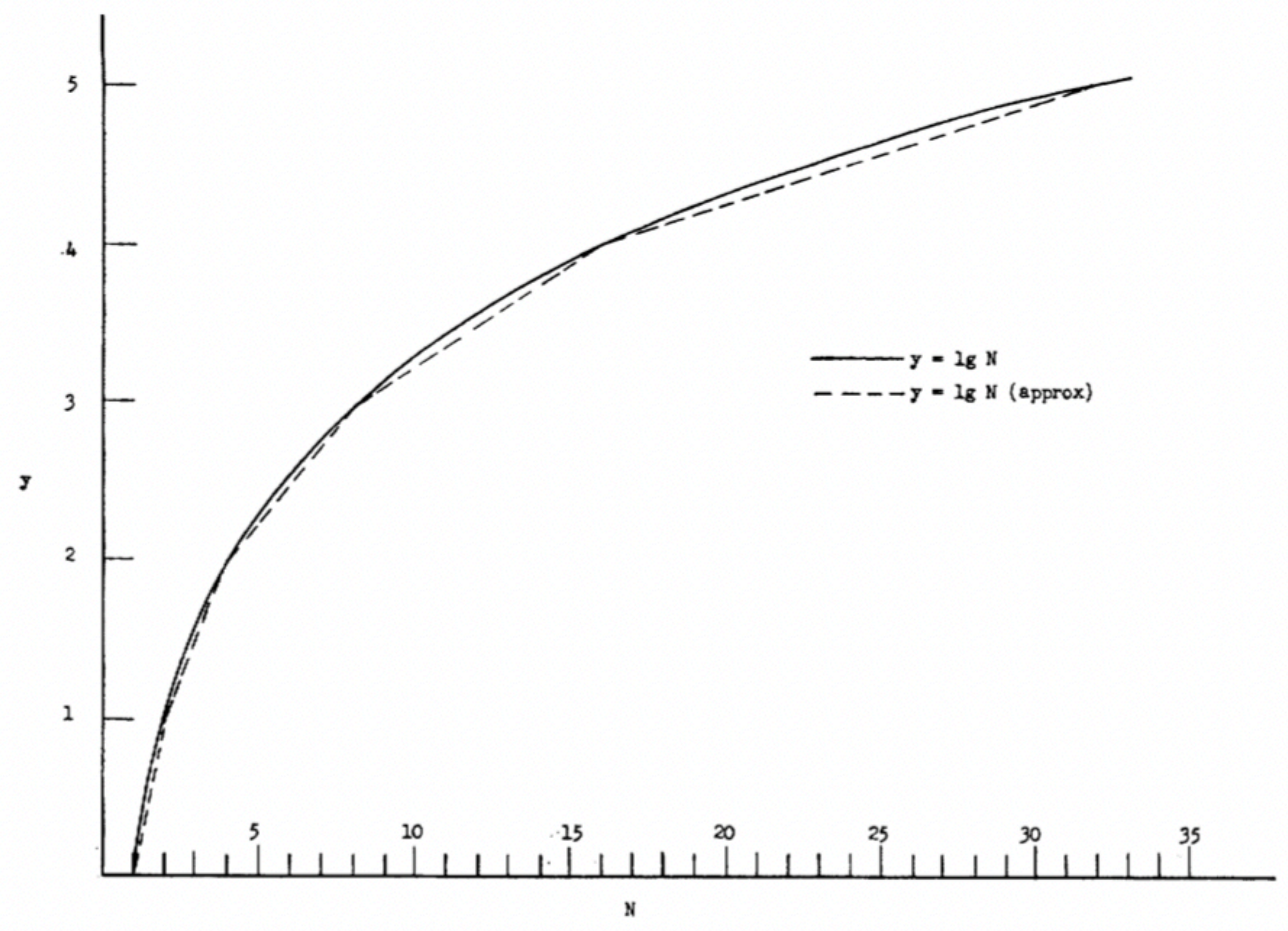
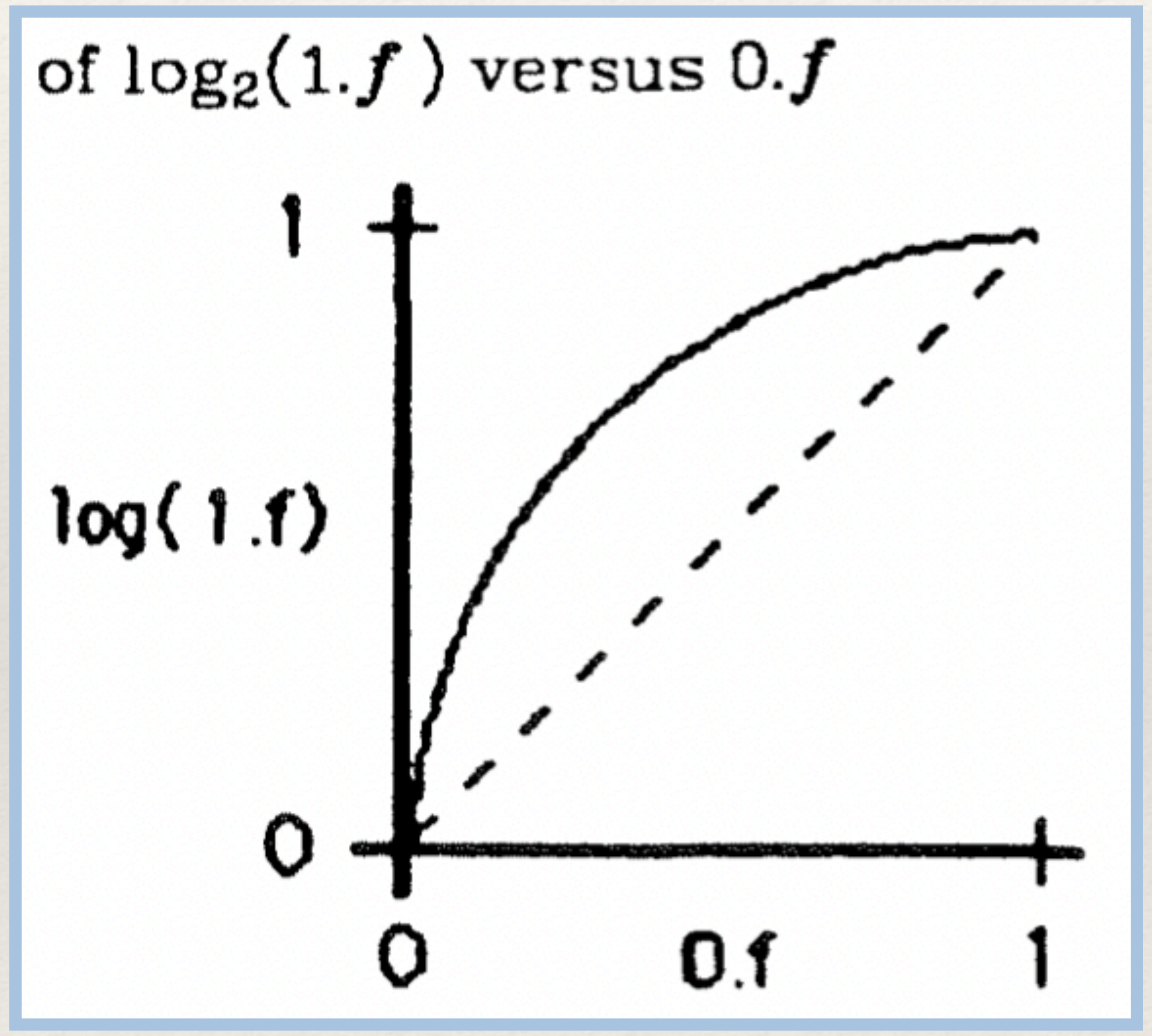


Fig. 2—Logarithmic curve and its straight-line approximation.

Mitchell

J. N. Mitchell, "Computer Multiplication and Division Using Binary Logarithms," in IRE Transactions on Electronic Computers, vol. EC-11, no. 4, pp. 512-517, Aug. 1962, doi: 10.1109/TEC.1962.5219391.

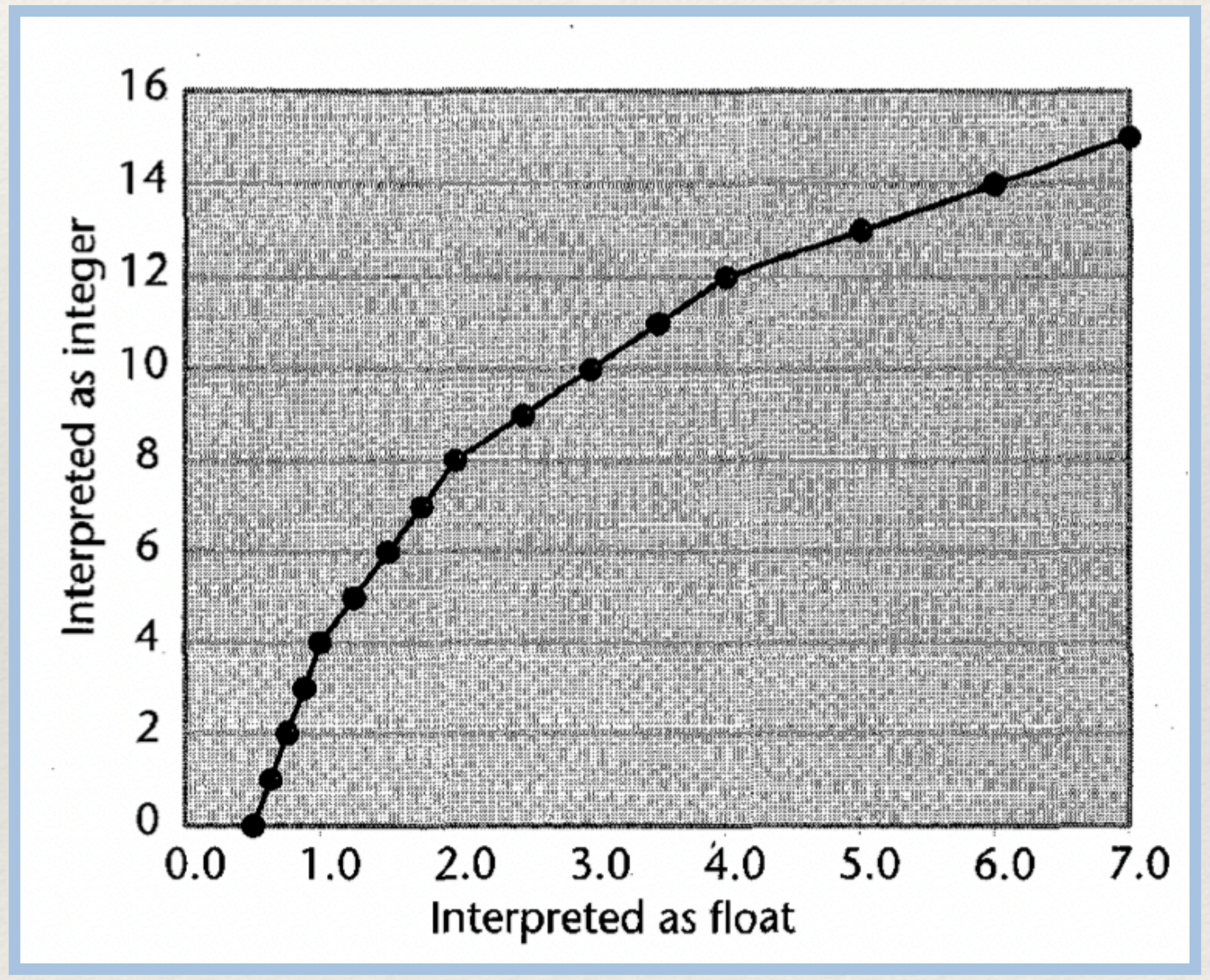
1984



Coonen

J. T. COONEN, "CONTRIBUTIONS TO A PROPOSED STANDARD FOR BINARY FLOATING-POINT ARITHMETIC (COMPUTER ARITHMETIC). PhD Thesis, University of California Berkeley, 1984.

1997

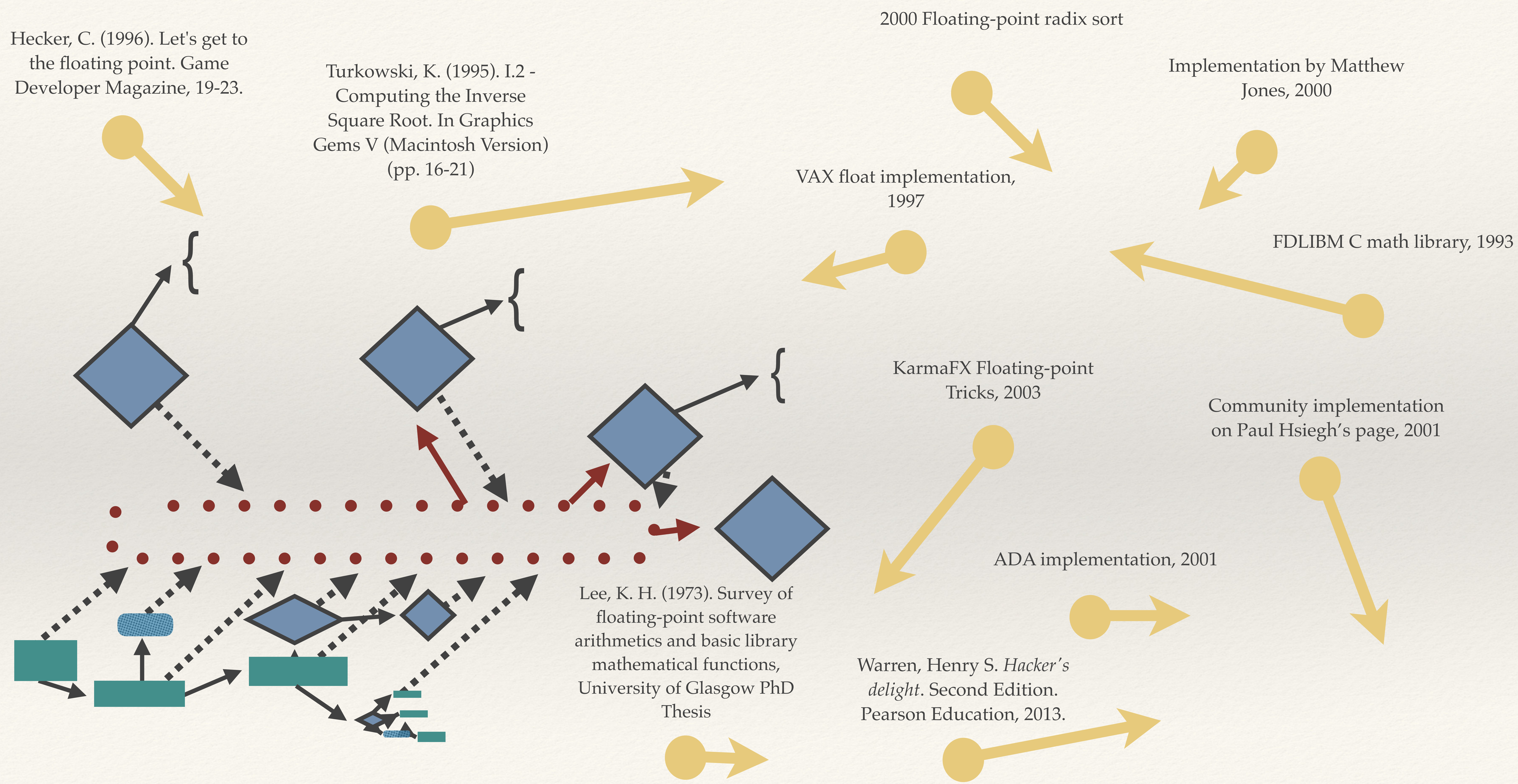


Blinn

J. F. Blinn, "Floating-point tricks," in IEEE Computer Graphics and Applications, vol. 17, no. 4, pp. 80-84, July-Aug. 1997, doi: 10.1109/38.595279.


Fast hardware multipliers
Logarithmic converters
Approximate Arithmetic Circuits
Mitchell (1962)
Fast HDR encoding
BRDF encoding
Low-power IC trig
Blinn (1997)
Neural Nets
Integer quantization
Gradient descent
Schraudolph (1999)
Trick of the trade
Kahan (1999)

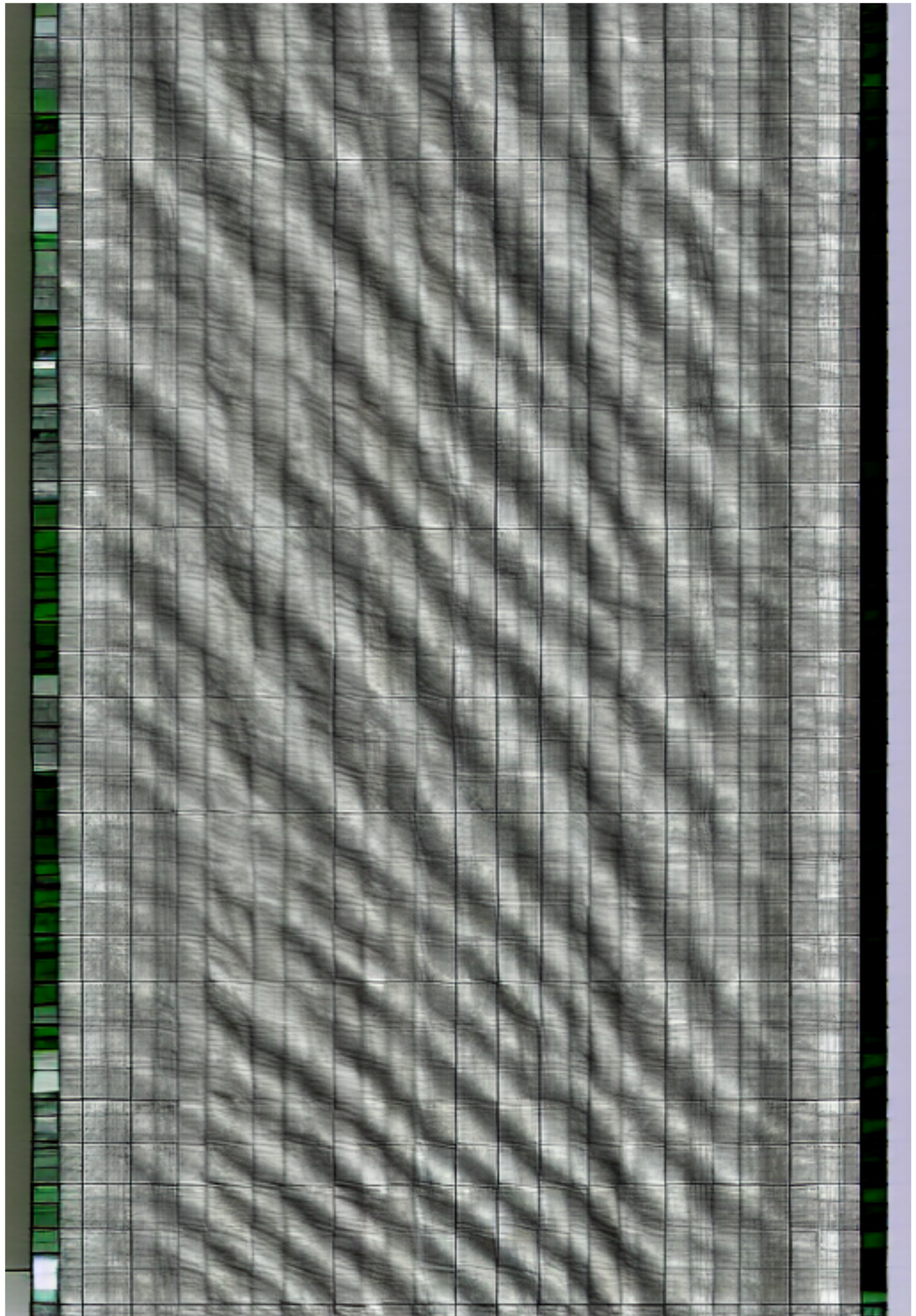
The space gets busy



Just what is being shared?

A half-dozen diagrams ago I mentioned that the FISR has essentially three components:

- Logarithmic transformation (c.f. Mitchell, Coonen)
- Exact constant to minimize error when restoring exponent (Kahan 1999, Moler & Walsh, etc.)
- Newton-Raphson iteration (ubiquitous in numerical computation)



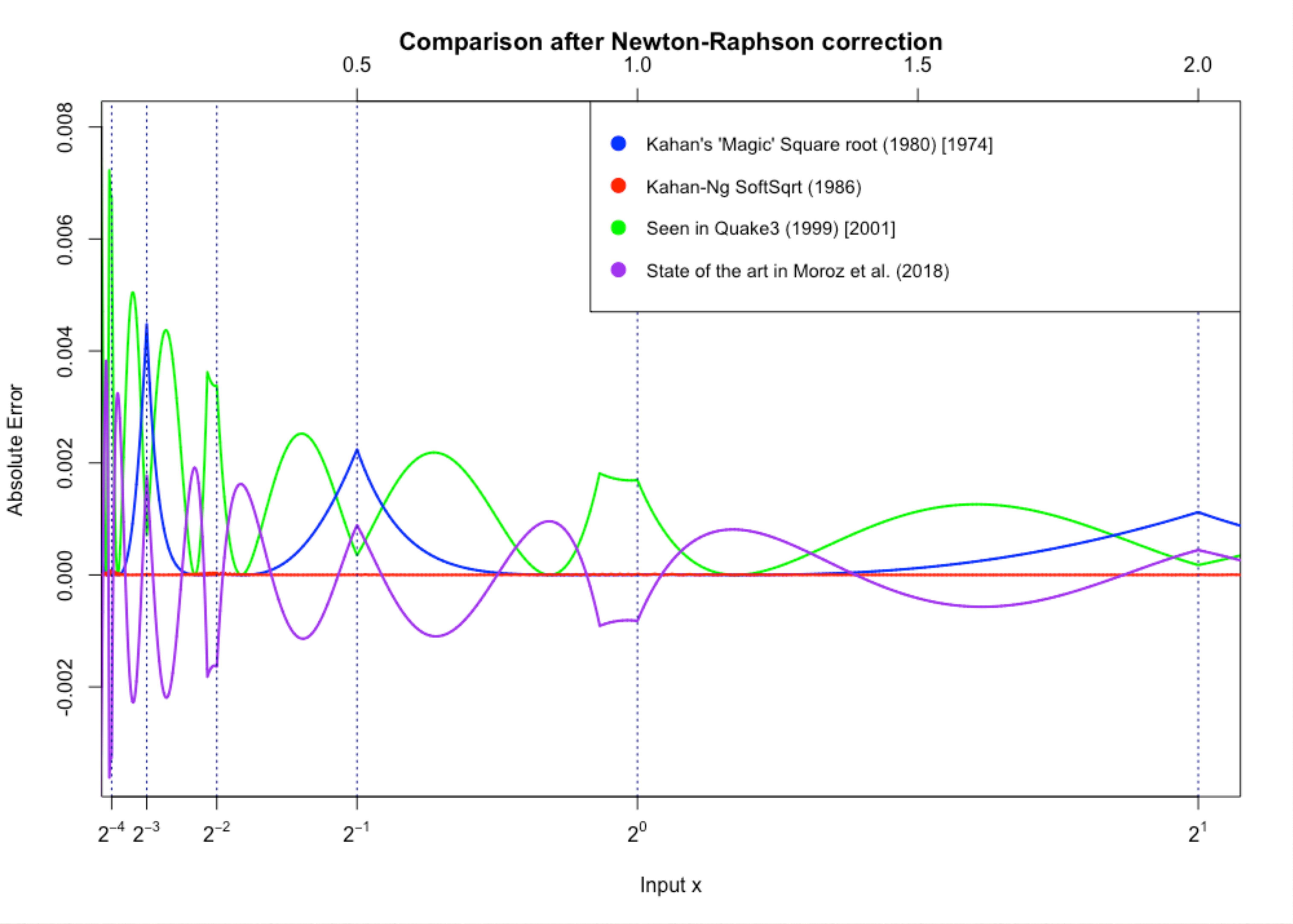
abline(v = 2^seq(-4, 1, by = 1)), Corrupted Stable Diffusion

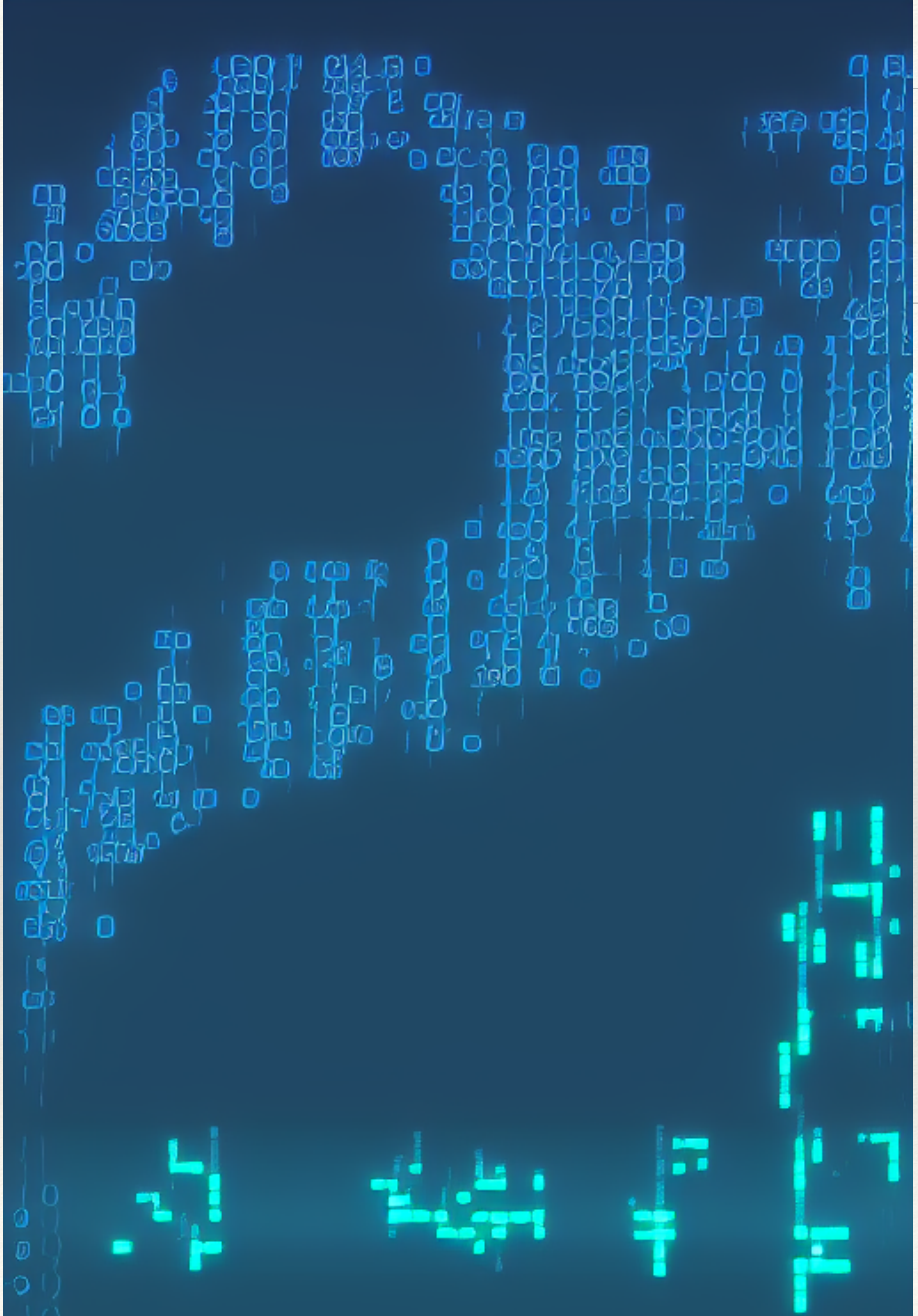
Space gets busy in the 1990s specifically

- This might be an artifact of what is saved or what I can follow, but…
- 1990s sees huge boom in:
  - Consumer microprocessors
  - Programming environments to explore memory tricks
  - Applications where digit accuracy is not necessary (e.g. video games)
- By 2002 there are about a dozen known implementations which do not stem from Quake III, Mitchell's paper, or Blinn's article

How to track

- The constant itself, 0x5F3759DF, is a good start
    - Identifies single-precision version which likely stem from the Moler / Walsh collab.
    - Not shared among related works
- But: won't identify works by influence.





Floating-point computation, Corrupted Stable Diffusion

What now?

- The concept of a logarithm inherent in floating-point representation is simultaneously magic and mundane.
- Patterns of citations for the FISR are very curious.
- Can we find *where* people learn a “trick of the trade”?

Thank you!

Follow 0x5f37642f.com* for more developments

* SEO is dead, long live being impossible to Google