Fractal Overdrive: An Aesthetic Evaluation of Numeric Error

Stephen Longfield*
Charles Eckman
stephen.longfield@gmail.com
charles@cceckman.com

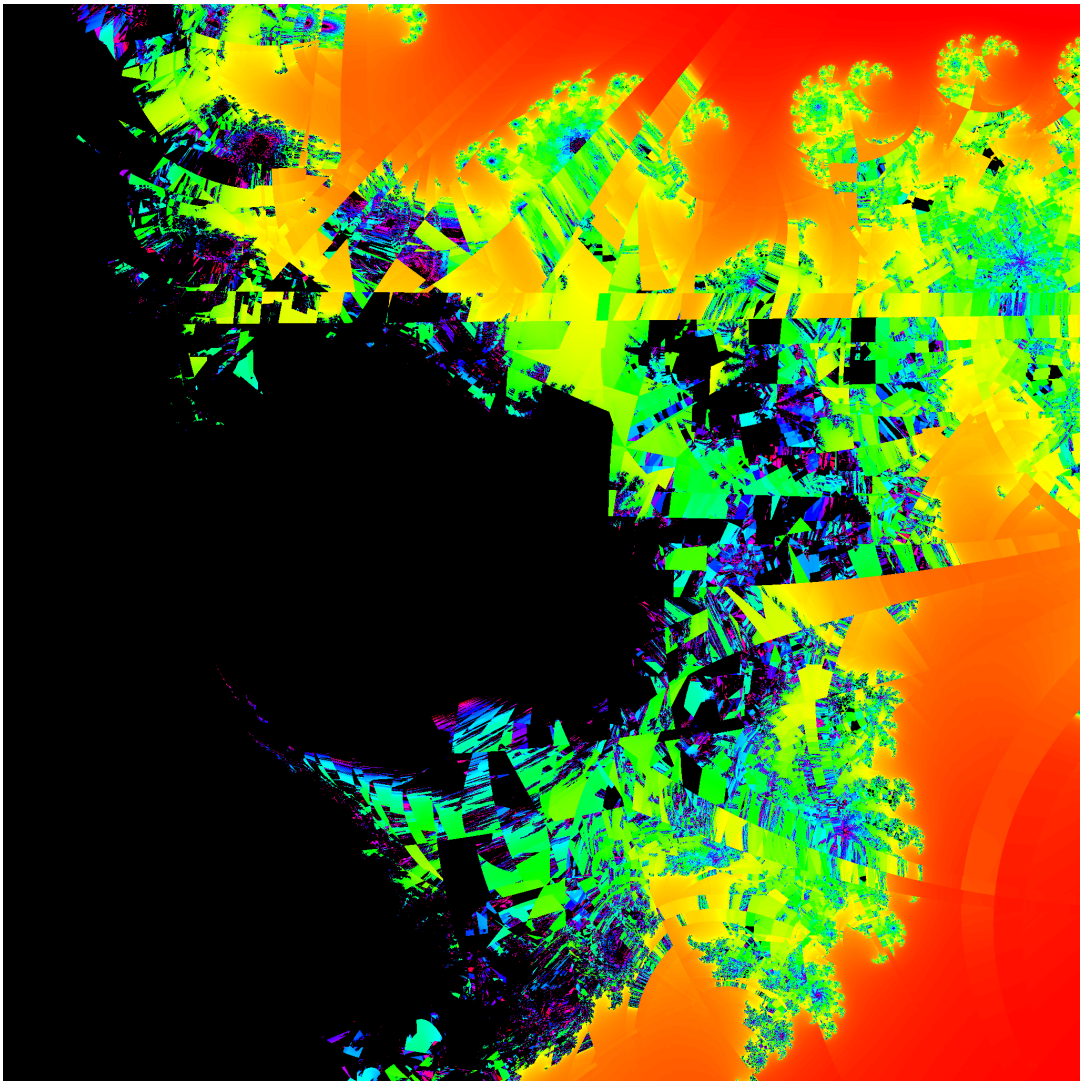


Figure 1: We meant to do that.

ABSTRACT

As erstwhile rock stars, the authors have intentionally applied distortion to certain 1-D signals for aesthetic effect. Sometimes this distortion comes from saturating the signal (overdrive); other times, it comes from a more subtle transformation, such as vacuum tube or transistor amplification. Some of the authors' peers use "fractal audio processing" [1] for distortive effects.

In this paper, we explore another dimension, and investigate how different ways of representing small numbers distort the world differently. To get a vibe check on numerical imprecision, we rendered two fractals using various numeric formats.

We got some neat pictures. Wanna see?

CCS CONCEPTS

• **Computing methodologies** → **Representation of exact numbers**; *Rasterization*; • **Human-centered computing** → **Empirical**

*This was Stephen's idea.

studies in interaction design; **Empirical studies in visualization**.

KEYWORDS

Numbers, Numeric Representation, Pretty Pictures

1 REAL MATH[1]

1.333984375 Feed me fractals

In this paper, we investigate two fractals: the Mandelbrot set and a Newton fractal. Both map a pixel coordinate (x, y) into the complex plane as a $c = x + yi$.

We evaluate the Mandelbrot set in the usual way:

$$z_0 = 0$$
$$z_{n+1} = {z_n}^2 + c$$

up to an iteration limit (n). Pixels that escape ($|z| \geq 2$) are given a hue according to [7]. Pixels that do not escape are colored black.

We compute the Newton fractal on the polynomial $p(z) = z^3 - 1$:

$$z_0 = c$$
$$z_{n+1} = \frac{p(z_n)}{p'(z_n)}$$

and plot whether z reaches zero within a specified iteration limit. Points that reach zero (within some threshold) are grouped based on which zero of the function they are close to. Pixels are colored by assigning each group a hue, and given a color value according to how many iterations it took to converge to zero.

1.666015625 Exact computation via `BigRational`

Since we're rendering for a computer screen, [2] we can (and do!) use exact inputs.

The raster grid (pixel grid) has integer locations, with $xres \times yres$ pixels. We map each (integer) pixel location (px, py) to a (rational) vector within the render window, centered at $(0, 0)$

$$\hat{x} = \frac{px}{xres} - \frac{1}{2}$$
$$\hat{y} = \frac{1}{2} - \frac{py}{yres}$$

We render a portion of the complex plane centered at $(\frac{x_c}{scale}, \frac{y_c}{scale})$, with equal width and height $\frac{size}{scale}$. We constrain these to all be rational numbers, which allows us to compute exact (rational) pixel coordinates:

$$x = \hat{x} \times \frac{size}{scale} + \frac{x_c}{scale}$$
$$y = \hat{y} \times \frac{size}{scale} + \frac{y_c}{scale}$$

We perform these computations using an arbitrary-precision rational number type, `num::BigRational` [10]. [3]

In principle, we could also carry out the fractal computations exactly using `BigRational`. The fractal formulae above require complex arithmetic, which is simple, plus some comparison operators ("greater than four" for Mandelbrot, "zeroish" for Newton).

When we tried to render the fractals using `BigRational`, though, it reached a hard timeout (against the author's patience). Moreover, rationality has little place in an aesthetic evaluation such as this, so we stick with finite-precision types.

2 UNREAL NUMBERS

Instead, we convert `BigRational` values to various numeric formats, approximating as closely as the format allows. Some of the conversions might even be correct (e.g. those we didn't write).

The formats we investigate are:

- `f32` and `f64`: IEEE 754 single- and double-precision floating-point formats[2]
- `MaskedFloat<N,M>`, an IEEE 754 float with some exponent and/or mantissa bits removed
- `IxFy`, fixed-point numeric formats [9, 12]
- `P32`, `P16`, `P8`: 32/16/8-bit posits™, an alternate float-like format[4]

2.125 MaskedFloat

MaskedFloat isn't yet a popular floating point type, but one we made up for this paper to try to create more interesting errors than we were seeing with just IEEE `f64`. It involves taking an `f64`, and masking off some of the bits, for our own amusement.

A normal `f64` consists of one bit of **s**ign, 11 bits of e**x**ponent, and then 52 **f**ractional bits, representing the value:

$$(-1)^s \times (1.f_{51}f_{50}f_{48}...f_0) \times 2^{x-1023}$$

If we want to experiment with what the world could be like if these were different (smaller) sizes, we can force some of those bits to one or zero. For the fractional bits, this is easy, as they are an unsigned value–just setting the least-significant bits to zero gets rid of it.

For the exponential bits, this is more complicated. If we want to constrain the exponent to be effectively 4 bits (i.e., range from -7 to 8), we have to constrain the exponent value to be between 1016 and 1031). Thankfully, we can do this with a bit of bit-twiddling. To make that easy to see, first, let's see those values in binary:

$$1016 = 0b01111111000$$
$$1031 = 0b10000000111$$

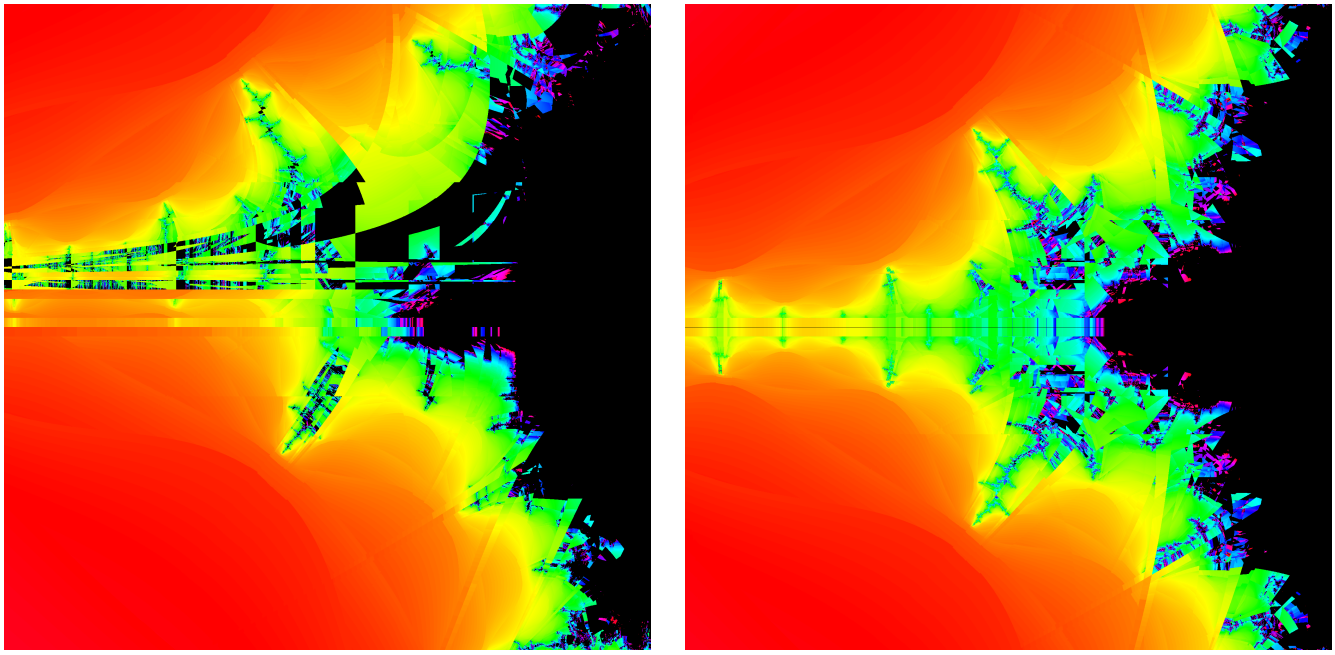
So then, all we need to do is check if the most-significant fractional bit is set, then if it and any of bits [9:3] are set (i.e., it is greater than 1031), clamp down to 1031, and if the msb is not set, and any of bits [9:3] are also not set (i.e., less than 1016), clamp up to 1016.

This naive masking does have the side effect of re-introducing some of the problems near zero that IEEE had carefully removed when they added subnormals, as well as adding some more, unique

[1] Rational, not real. Rationality void where prohibited.

[2] Either our web interface, or the PDF version of this paper. If you're getting the print proceedings for SIGBOVIK, tough luck.

[3] See also "BigMood: A novel format for arbitrary-precision emotional processing", to be published in SIGBOVIK 2025.



(a)
bug

(b)
no
bug

Figure 2: An asymmetric error in MaskedFloat

problems. Since our goal in this paper is "shenanigans", that's great news for us.

The exponent has a special value, all-zeros, that is used along with an all-zero fraction to represent, unsurprisingly, zero. A naive exponent mask would turn that value instead into the smallest possible exponent (e.g., in the previous example, 2^{-1}), which isn't quite accurate; though that inaccuracy can be interesting, zero is a useful number for fractal computation, so we preserve this special-case behavior.

2.25 Fixed-point

We use the `fixed` Rust library [9] to perform fixed-point arithmetic.[4] The library offers a family of fixed-point formats specified as `IxFy`, with x signed integer bits and y fractional bits.

In this paper, we show a subset of the fixed-point formats we thought were coolest.

2.375 Posits™and quires

For posits,™we use the `softposit` Rust library[11] and we implement conversion from `BigRational` via the associated quire types. The quire formats are wider fixed-precision formats associated with posits™. For instance, the quire associated with the 32-bit posit™is 512 bits wide, with a precision of 2^{-240} – roughly equivalent to an `I241F240` fixed-point.

[4]We have especially high confidence in this library's numeric implementations. If any library has had all the bugs ironed out, it's `fixed`.

2.5 Sources of distortion

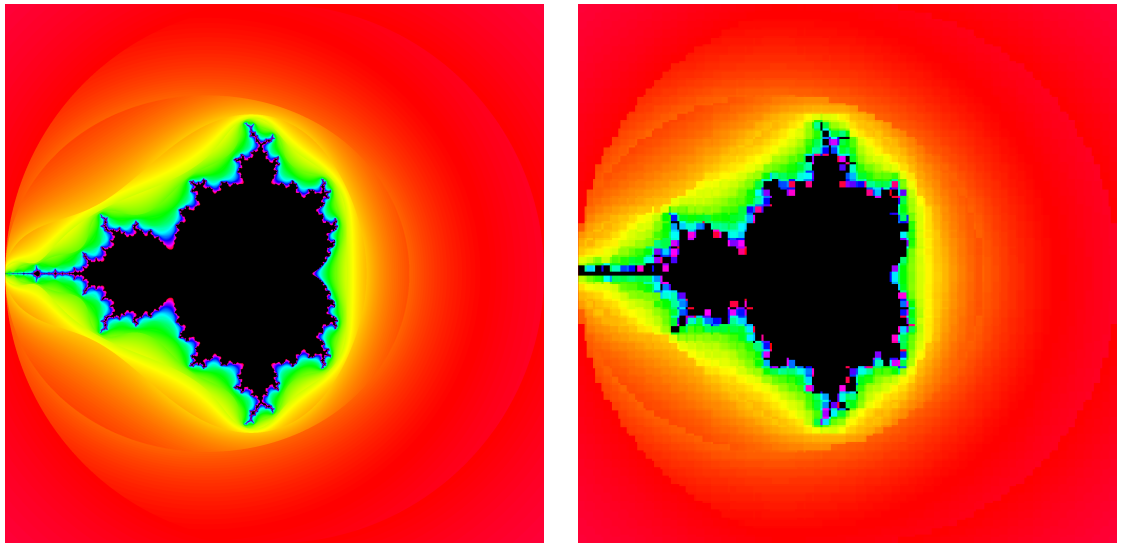
When rendering fractals, distortion, like inspiration, can come from anywhere.

2.5.19921875 Bugs. When implementing the MaskedFloat type, the authors originally did not properly account for the special case around zero, and instead represented it as the smallest possible positive number. This resulted in strange, but pretty neat looking, asymmetrical error, depicted in Figure 2.

2.5.400390625 Scale. The Mandelbrot and Newton fractals both exhibit self-similarity, repeating themselves at various scales. However, not all of the formats sampled can operate at multiple scales: all of them have some scaling limits, but some are more limited than others.

This is most obvious when dealing with fixed-point formats. `I11F5`, for instance, can only represent 128 distinct values in the range $|z| < 2$ - where the Mandelbrot set lives. That's fewer than the pixels used to render these images – hence, pixelation, as shown in Figure 3.

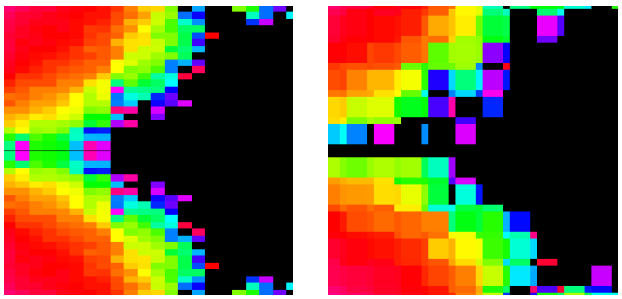
The nature and pattern of the pixelation depends on the format used, as shown in Figure 4. In a region of size $\frac{2}{3}$ centered on $\frac{-4}{3} + 0i$, the `P8` and `I11F5` formats show different aesthetic qualities. `P8` (depicted) and floating-point stretch into rectangles, as is apparent when far from $x = y$; while fixed-point gives the appearance of overlapping tiles.



(a)
f32

(b)
I11F5

Figure 3: f32 and I11F5, -2 to 2



(a)
P8

(b)
I11F5

Figure 4: Two pixelations near $\frac{-4}{3}$

2.5.599609375 *Precision error.* It's possible to preserve dynamic range (see section 2.75) but lose precision, as in `MaskedFloat<6,3>`: it has 6 bits of exponent (scale), but only 4 bits at any given scale.

Figure 5 shows this in the Newton fractal. The area close to the origin maintains high resolution, but it becomes steadily more pixelated further out. In some sense, this error mirrors the human eye's capabilities: a foveal region in the center of vision/complex plane.

2.5.798828125 *Exponent error.* This is the key characteristic of MaskedFloat when configured for high precision (many mantissa bits) but a small dynamic range. The places where the exponent saturates, we introduce distortion. This distortion appears as arcs of discontinuity in the render. See Figure 6 for a comparison of the f64 rendering of a zoom in of the Mandelbrot fractal and MaskedFloat (configured with 4 exponent bits and 51 mantissa bits.)

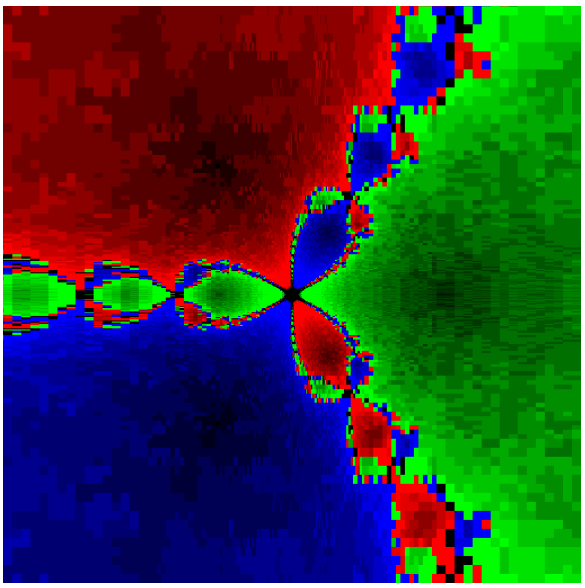
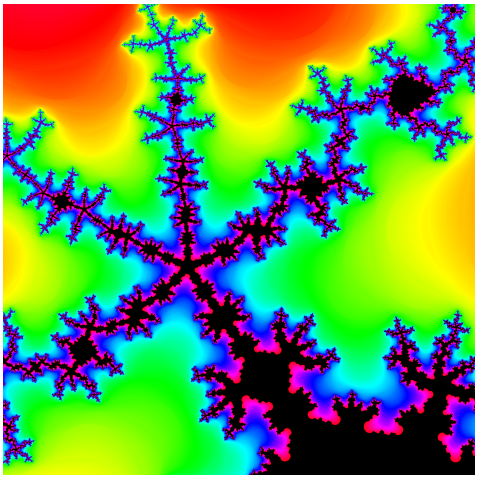


Figure 5: MaskedFloat<6,3>

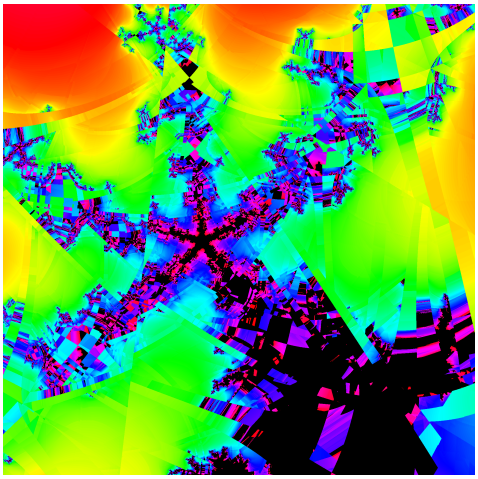
2.75 A note on dynamic range

For the fractals of interest, coloring a point involves iterating on a value. Since in many of the numerical formats, larger numbers means large possible errors, the range of values this calculation reaches matters.

This affects how much distortion we see in the Mandelbrot fractal compared to Newton: in the Mandelbrot set, the iterated values tend to stay near their starting point, or become larger than our escape threshold 4, but in the Newton fractal, intermediate values

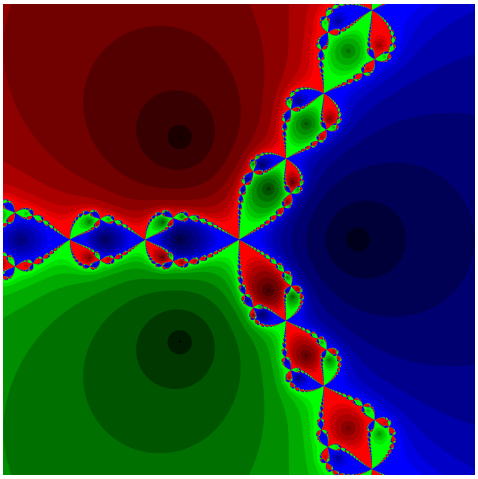


(a)
f64

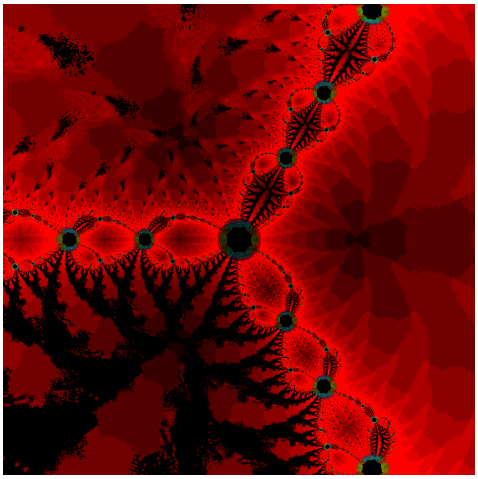


(b)
Masked-
Float

Figure 6: Exponent errors in MaskedFloat<3,50>



(a)
f64



(b)
I22F10

Figure 7: Fixed point distortion of Newton's fractal

can become very large when the slope is near zero, and the final numbers are very small, as $f(z)$ approaches zero.

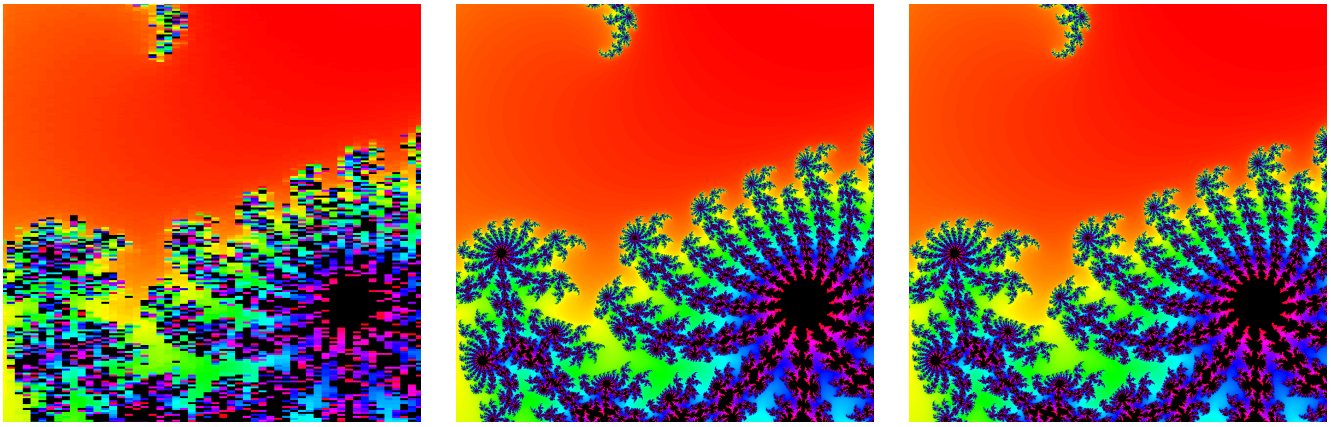
4 WHICH PICTURES ARE PRETTIEST?

You may also be interested in one of the threads near the source code [6], where we have noted regions of interest.

4.142578125 If it's fixed, it's broken

Figures 3 and 4 don't do a lot to recommend fixed-point to us. Pixelation can be nice if you're a child of the 80s or something, but it's not really weird enough that we should use a whole new numeric format. Just use ImageMagick. [5] [8]

The Newton fractal is a different story, as already shown in Figure 5. Since the Newton fractal may result in very small or very large intermediate values, fixed-point numbers are uniquely poorly suited for accurate computation. In Figure 7, we can see a nice fractal visualization of approaches to zeros become a fractal hellscape of non-convergence and incorrect answers, by switching from `f64` to `I22F10`. Despite no longer containing useful information, the distorted Newton's fractal looks much cooler.

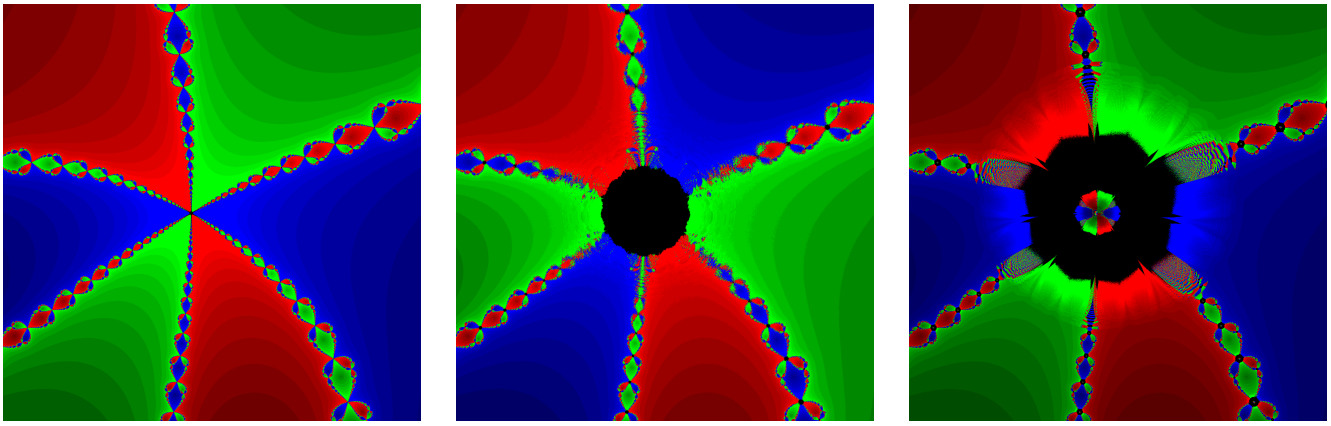


(a)
f32

(b)
f64

(c)
P32

Figure 8: f32, f64, and P32 over a whorl



(a)
P32

(b)
P16

(c)
Masked-
Float<4,50>

Figure 9: P32, P16, and MaskedFloat<4,50> on the Newton fractal

4.28515625 Pixels and Posits™

Posits™maintain greater precision than `f32`, with the same number of bits. Figure 8 shows `f32`, `f64`, `P32` over the same area.

Clearly posits™are the best option for everything. Right? Come. Follow the word of Gustafson. [4] Enter the circle between zero and infinity. join us Join Us JOIN US

Sorry, got a little chanty there–didn't mean to scare you. Can we interest you in an informative pamphlet? [3]

4.427734375 Newtonian surfaces

Let's face it, there's a reason that Mandelbrot is popular: there's lots of different shapes and colors. But despite being less popular, Newton fractals have some interesting artifacts too! Seriously! Promise!

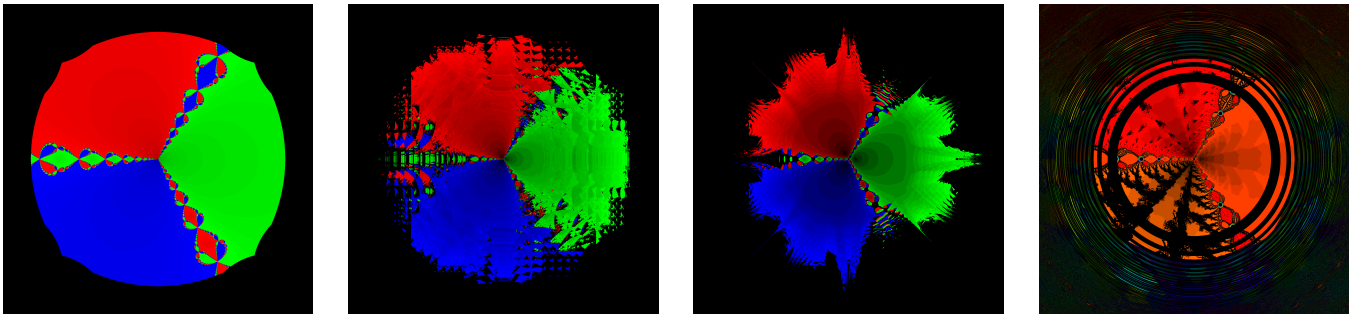
Like Figure 9: floating-point and `P32` formats all run up against the iteration limit at the center area. However, evaluation at higher iterations "closes" the hole (not shown).

`P16`, though, runs up against its limits, rippling out noise and forming a hole in the center, which persists at higher iterations. MaskedFloat goes even further and produces a distorted mirror within the empty space.

It's a little creepy, honestly. Have you seen ??? It's like that. Right? Right.

4.572265625 Newtonian event horizons

As you zoom farther out in the Newton fractal, we reach the point where the formats can no longer store the values of the starting position or its intermediate values (or can only store with reduced accuracy). The authors refer to this as the outer event horizon. Figure 10, shows the shape of this event horizon for various types.



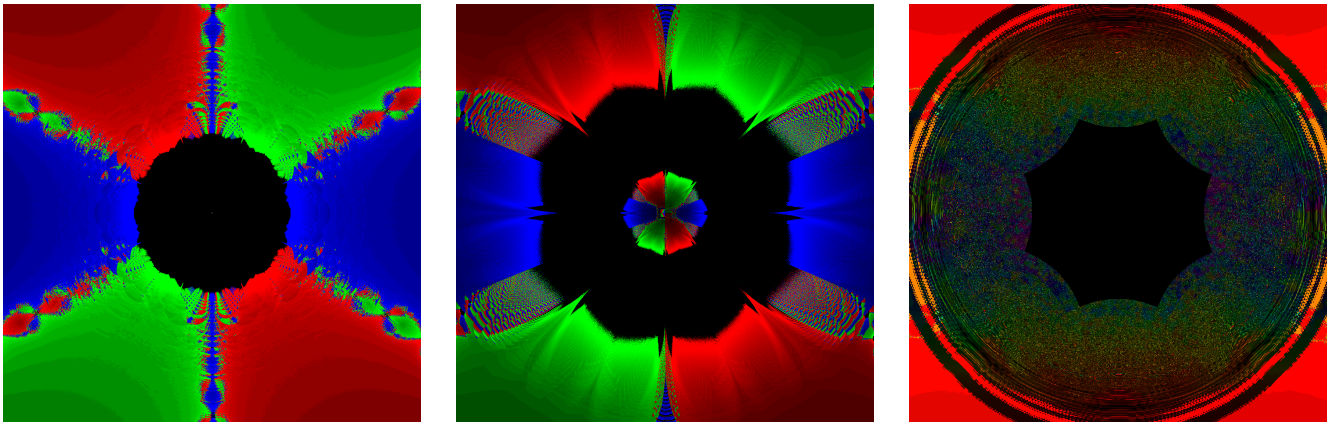
(a)
F32

(b)
P16

(c)
MaskedFloat<4,50>

(d)
I22F10

Figure 10: Newton fractal – outer event horizon (mixed scales)



(a)
P16

(b)
MaskedFloat<4,50>

(c)
I20F12

Figure 11: Newton fractal – inner event horizon

Note that these are not at the same zoom level–`f32` can represent much, much larger numbers than the others.

If instead of zooming out towards infinity, we zoom in towards zero, we reach the inner event horizon–where the formats cannot represent how small the numbers have become. Figure 11 shows the shapes and edges of this inner event horizon. Interestingly, the inner event horizon appears at approximately the same scale for `P16`, `MaskedFloat<4,50>`, and `I20F12`; Figure 11 depicts a consistent scale.

Slightly above the inner event horizon, there's also an interesting distortion region, where the intermediate values get twisted and warped at the edge of their range. See Figure 12 for some examples, from the wobbly `P16`, to the funhouse mirror of MaskedFloat, and lastly a demonic sunrise over `I20F12`.

4.712890625 Hyperbolic startbursts

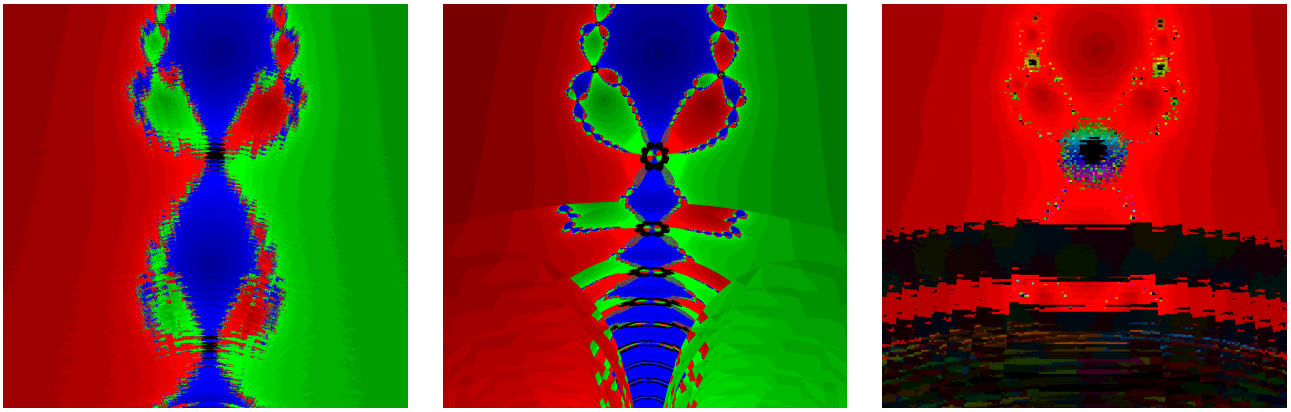
One additional pattern of distortion that MaskedFloat formats demonstrate is the "hyperbolic starburst",[5] so named for its appearance during an earlier (buggier) implementation of MaskedFloat. This pattern seems to appear in Mandelbrot near denser regions of the set e.g. repetitions of the Mandelbrot "beetle" shape.

For instance, Figure 13 shows two MaskedFloat configurations in the same range: centered on a beetle, in the lighting off of the north bulb. Note that some portions of `MaskedFloat<3,50>` mirror the fine structure of the fractal (lighting bolts), while others appear to be curvilinear.

4.857421875 Errors abound?

Figure 14 shows something even odder: an apparent structural difference between `f64` and a MaskedFloat format. Whether this points to an offset error in the format, or another issue, we leave for future investigators.

[5] It's only a starburst if you imagine its shape greatly exaggerated.

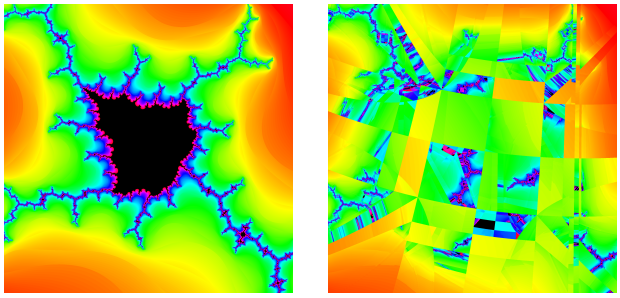


(a)
P16

(b)
Masked-
Float<4,50>

(c)
I20F12

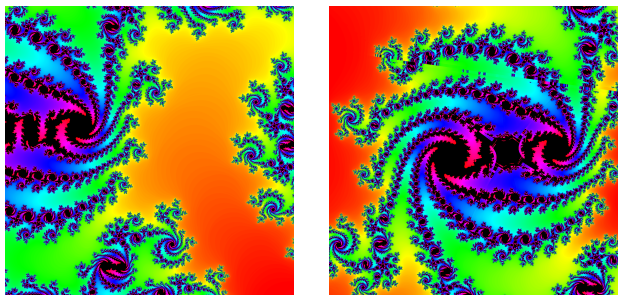
Figure 12: Near the inner Newton event horizon



(a)
Masked-
Float<4,50>

(b)
Masked-
Float<3,50>

Figure 13: A Mandelbrot-like region of Mandelbrot



(a)
f64

(b)
Masked-
Float<4,50>

Figure 14: Computational errors in MaskedFloat?

7 ASSESSMENT AND FUTURE WORK

We unequivocally recommend the MaskedFloat family of numeric formats for making weird-looking fractal art, and whatever format(s) your hardware supports for everything else.

The authors only explored the Newton fractal on the Wikipedia example polynomial $p(z) = z^3 - 1$. It's possible other Newton fractals would lead to more cool distortion. As the core operation in Newton's fractal is very similar to the core operation in machine learning's gradient descent calculation, there's probably an ML-adjacent paper one could shovel out about this, if you want big-corp funding.

ACKNOWLEDGMENTS

Thanks to M+T for leaving Stephen enough sleep to work on this. Thanks to Q for supporting Charles while working on this.

Well, we say "work"...

REFERENCES

[1] [n. d.]. Artists – Fractal Audio Systems. Retrieved March 23, 2024 from https://www.fractalaudio.com/artists/

[2] 2008. IEEE Standard for Floating-Point Arithmetic. Standard IEEE Std 754-2008. IEEE Computer Society, New York, NY, USA. https://web.archive.org/web/20160806053349/http://www.csee.umbc.edu/~tsimo1/CMSC455/IEEE-754-2008.pdf

[3] Dietrich Geisler and Edwin Peguero. 2019. The Cult of Posits. https://www.cs.cornell.edu/courses/cs6120/2019fa/blog/posits/

[4] Posit Working Group. 2022. Standard for Posit™ Arithmetic. Technical Report. Retrieved March 22, 2024 from https://posithub.org/docs/posit_standard-2.pdf

[5] ImageMagick Studio LLC. [n. d.]. https://imagemagick.org/

[6] Longfield and Eckman. 2024. Share your discoveries! https://github.com/cceckman/fractal-farlands/issues/11 Github issue thread.

[7] Robert P. Munafo. 2023. Continuous Dwell. Retrieved March 20, 2024 from https://mrob.com/pub/muency/continuousdwell.html personal web site.

[8] Randall Munroe. 2020. Dependency. https://xkcd.com/2347/

[9] The `fixed` crate authors. [n. d.]. Crate `fixed`. Retrieved March 22, 2024 from https://docs.rs/fixed/1.26.0/fixed/index.html

[10] The `num` crate authors. [n. d.]. Type Definition num::BigRational. Retrieved March 22, 2024 from https://docs.rs/num/latest/num/type.BigRational.html

[11] The `softposit` crate authors. [n. d.]. Crate `softposit`. Retrieved March 22, 2024 from https://docs.rs/softposit/0.4.0/softposit/

[12] William G. Wong. 2017. What's the Difference Between Fixed-Point, Floating-Point, and Numerical Formats? https://www.electronicdesign.com/technologies/embedded/article/21805517/whats-the-difference-between-fixed-point-floating-point-and-numerical-formats

APPENDIX

A ARTIFACTS

Just go to Github: https://github.com/cceckman/fractal-farlands