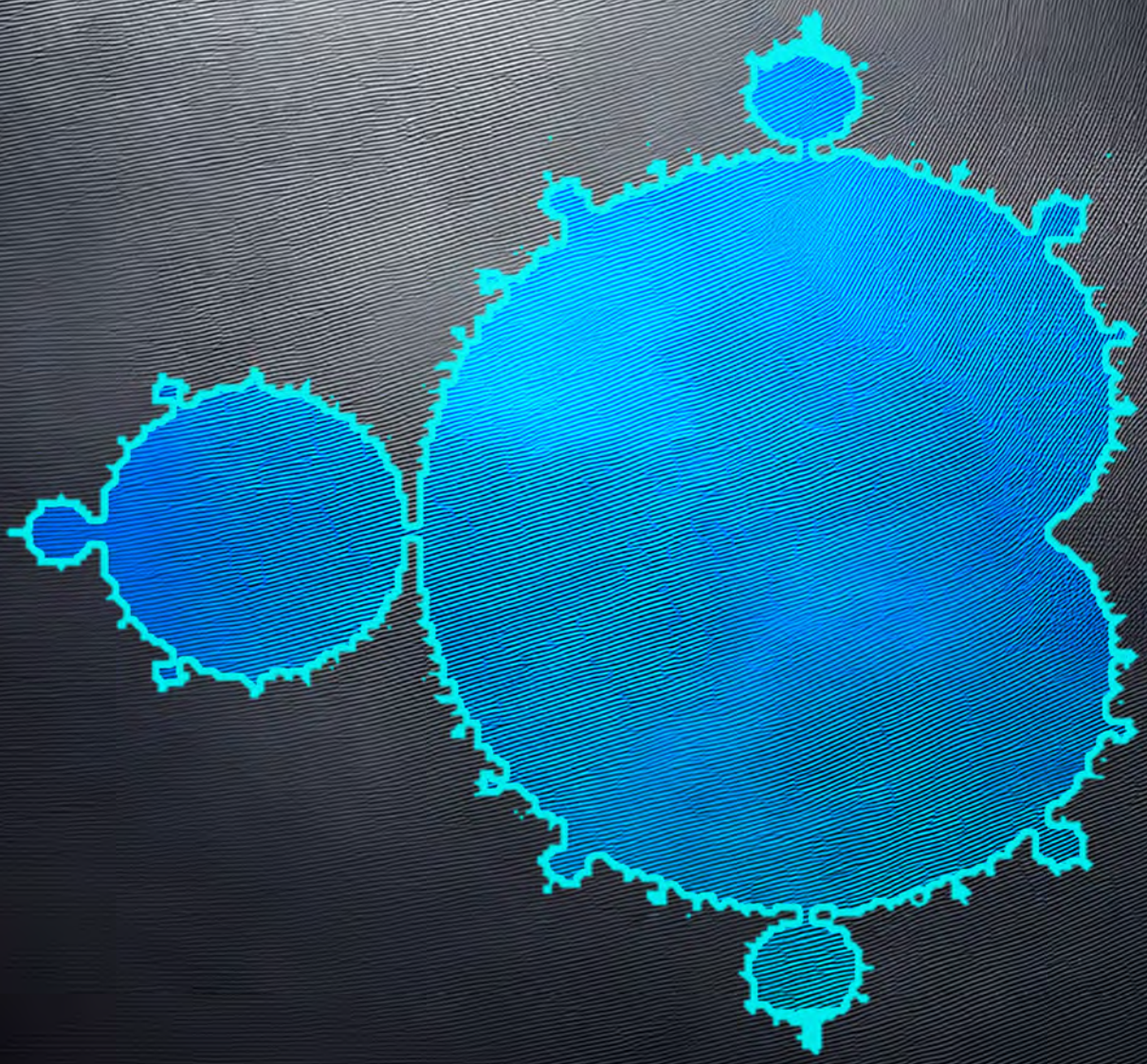


Mandelbrot Metal

Mandelbrot Metal

Executive Summary

Mandelbrot Metal is my high-performance fractal renderer and explorer for iPhone and iPad. It delivers smooth, real-time exploration of the Mandelbrot set with adaptive precision (FP32 → FP64 → FP128 double-double), continuous coloring, super-sampling anti-aliasing, a flexible palette system with gamma-aware interpolation, contrast control, high-resolution export, and sharing ecosystem. The renderer prioritizes interactivity during navigation and automatically raises quality when you pause, producing studio-grade images without manual tweaking.

Overview

I built Mandelbrot Metal to make deep fractal exploration fast, precise, and beautiful. It runs a Metal® compute kernel that evaluates the escape-time algorithm in parallel, maps continuous iteration values to color via GPU lookup tables (LUTs), and presents the result with minimal CPU–GPU overhead. My goals were simple: instantaneous feel while moving, and uncompromised fidelity when still.

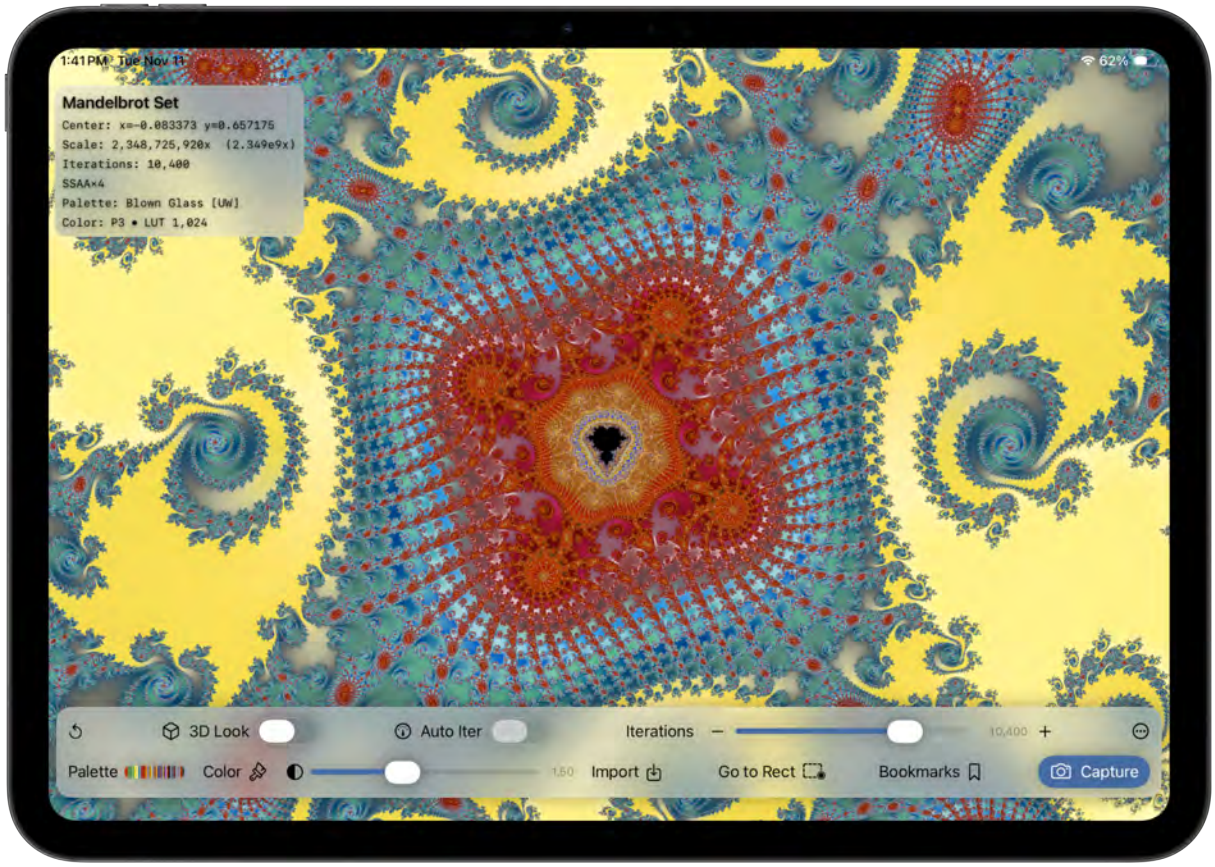


Figure 1 – Mandelbrot Metal on the iPad Pro 11” (M4)

Key Features

- Real-time GPU rendering of the Mandelbrot set
- Extreme deep zooms up to 1×10^{16} x (1e16x)
- 140+ curated scientific and artistic color palettes
- Palette Editor with gradient stops and live preview
- Gradient Import from any image with gamma-aware sampling
- Lookup table (LUT) palettes (1×N textures) with configurable resolution (256/1024)
- 40+ Ultra-Wide (768-step) palettes
- Instant palette swapping (coloring decoupled from iteration work)
- Nonlinear Iteration Slider for fine and wide-range control
- Auto Iteration scaling with zoom depth
- Adaptive Precision (FP32 → FP64 → double-double)
- 3D Look mode
- Deterministic capture and bookmarking
- Bookmark and Palette sharing (import/export in JSON format)

Figure 2 – Mandelbrot Metal on the iPhone 17 Pro Max

The Mandelbrot Set: An Overview

Fractals

A fractal is a mathematical set or geometric structure that exhibits self-similarity across scales and contains detail at arbitrarily fine resolutions. Formally, fractals are often characterized by having a Hausdorff (or fractal) dimension that exceeds their topological dimension, reflecting their intrinsic complexity. See my The Coastline Paradox: When Math Meets the Infinite and What Are Fractals.

Fractals are typically generated by iterative or recursive processes, where the repeated application of simple rules produces structures of great intricacy. The Mandelbrot set is a canonical example: its boundary is infinitely complex, and magnification reveals patterns that echo the structure of the whole, a hallmark of self-similarity.

There are other canonical fractals including the Julia Set and the Burning Ship. I will be adding these to the app in the future.

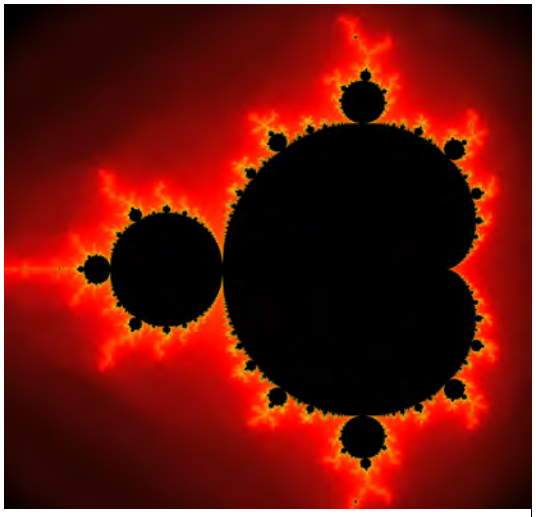


Figure 3 – The complete Mandelbrot Set as captured in Mandelbrot Metal, exhibiting its classic cardioid and bulb shapes.

Introduction to the Mandelbrot Set

At the heart of the complex plane lies the enigmatic fractal known as the Mandelbrot set, a mathematical marvel distinguished by its intricate boundaries and infinite depth. When a specific operation is repeatedly applied to complex numbers, those outside the set diverge toward infinity, while those within remain, tracing subtle and mesmerizing patterns.

The boundary itself is a stage for endlessly detailed wanderings, revealing astonishing variety and beauty.

Origins and Historical Context

Named for Benoit B. Mandelbrot, a research fellow at the IBM Thomas J. Watson Research Center in Yorktown Heights, New York, the set emerged from his pioneering work in geometric forms during the mid-1970s. Mandelbrot's explorations gave rise to fractal geometry, a field dedicated to studying shapes with fractional dimensions.

Exploring the Set

The boundary of the Mandelbrot set exemplifies a fractal, presenting a unique subject for mathematical exploration. Through specialized software, computers become virtual microscopes, revealing ever more detailed structures at increasing magnification. From afar, the set appears as a squat, wart-covered figure eight on its side, with a shadowy black interior surrounded by vibrant halos of electric blue, deep yellow, red, green, and more. This visual complexity invites endless curiosity and study.

Complex Numbers and the Complex Plane

A complex number is of the form: $a + bi$, where:

- a is the real part
- b is the imaginary coefficient
- i is the imaginary unit ($\sqrt{-1}$)
- both a and b are real numbers

Complex numbers can be plotted on a set of coordinates in which the horizontal axis the real part and the vertical axis is the imaginary part.

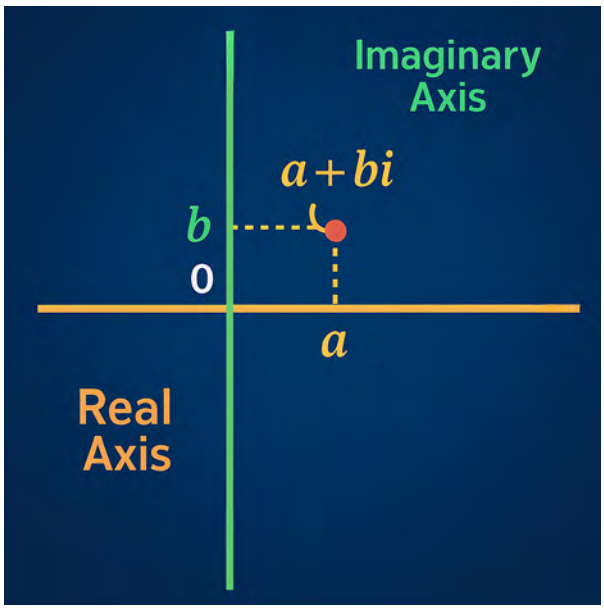


Figure 4 – The complex plane with its real and imaginary axes, showing the coordinates of the complex number

Complex Number Addition

Two complex numbers $a + bi$ and $c + di$ add as follows:

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$

Complex Number Multiplication

Two complex numbers $a + bi$ and $c + di$ multiply as follows:

$$(a + bi)(c + di) = (ac - bd) + (ad + bc)i$$

Theory of Operation

Mapping the Complex Number Plan to Display Screen Coordinates

The Mandelbrot set is defined as the set of all complex numbers c for which the recurrence:

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

remains bounded as n approaches infinity.

If $|z_n|$ (here the absolute value means the complex modulus) never exceeds 2 within a practical iteration budget, I treat c as in-set; otherwise, it escapes. The boundary is fractal—infinitely detailed and self-similar in structure. This is the region of interest.

Why the bailout is 2: If $|z| > 2$ at any step, $|z_{n+1}| = |z^2 + c| \geq |z|^2 - |c|$, which grows without bound.

If $|z_n|$ never exceeds a bailout radius (I use 2) within a maximum iteration budget, I treat c as inside the Mandelbrot set; otherwise, it's outside. The boundary exhibits infinite, nontrivial structure at every scale.

Visually, the Mandelbrot set is a map of stability in the complex plane—black areas remain bounded, while colored areas diverge, revealing intricate self-similar patterns.

I map each display pixel (x, y) to a complex coordinate via a linear transform. Each pixel (x, y) on the screen corresponds to a complex number c in the plane. As you zoom and pan, the app recalculates the mapping so that:

$$\mathrm{Im}(c) = y_{\min} + \frac{y}{H}(y_{\max} - y_{\min}), \quad \mathrm{Re}(c) = x_{\min} + \frac{x}{W}(x_{\max} - x_{\min})$$

Zooming in or out changes these bounds—not the underlying math.

Escape-Time Core Algorithm

For each pixel's *c*:

1. Set z = 0
2. Repeat $z \leftarrow z^2 + c$ until up to a max iteration count
3. If $|z| > 2$, the point has escaped — color is based on iteration count at escape
4. If never escaped, color it black (inside set)

This is known as the escape-time algorithm.

Coloring the Mandelbrot Set

Interior (the set itself)

Points *c* that do not escape—meaning $|z_n|$ never exceeds the escape radius (typically 2) within the maximum allowed iterations—are considered part of the Mandelbrot set. These are colored black in Mandelbrot Metal.

Exterior (escape points)

Points c for which $|z_n|$ exceeds the escape radius are classified as outside the set. For these points, we record the iteration count at which escape occurs. This value is then converted using a smooth coloring function (such as the continuous escape-time formula *μ)* and finally mapped to a color via a 1×N lookup table (LUT).

This means:

- Slow escapes (low iteration counts) map to one part of the palette.
- Fast escapes (high iteration counts, near the boundary) map to another.
- Smooth interpolation between these values avoids visible bands.

Result:

- The black regions form the infinitely detailed Mandelbrot set itself, while the rich gradients and colors are applied to the escaping points, revealing the filaments, spirals, and self-similar structures around the boundary.

Auto Iteration and Manual Control

Rendering fidelity in the Mandelbrot set depends critically on the maximum iteration count. Shallow zooms require only modest iteration budgets to resolve detail, but deeper zooms demand exponentially more iterations to prevent premature escape detection and banding artifacts.

Auto Iteration

To balance speed and accuracy, I use an auto iteration function that increases the iteration cap as you zoom in. The Auto Iteration controller scales the maximum iteration budget with zoom depth using a polynomial in log scale:

$$n_{\max} \approx n_0 + a \log_{10}(\rho) + b\,[\log_{10}(\rho)]^2 + c$$

where:

- $n_{\max}$ is the iteration limit for the current zoom
- ρ is the zoom factor (relative to baseline scale)
- n_0, *a, b, c* are tuned coefficients chosen empirically for smooth progression

This scaling ensures that as zoom depth increases, iteration budgets rise automatically, keeping the image smooth and free of noise without user intervention.

Manual Slider (Nonlinear Response)

In addition to auto iteration, the app provides a manual iteration slider for fine-tuning. The slider is nonlinear, meaning its position is mapped to iteration count via an exponential or logarithmic curve rather than linearly.

This design has two benefits:

1. Precision at low values — users can make small adjustments where low iteration counts are sensitive to change.

2. Reach at high values — the slider still allows access to very high iteration counts (thousands+) without requiring impractically long slider travel.

In practice, this nonlinear control makes manual tuning feel natural, letting you glide smoothly between quick, coarse previews and high-quality, detail-preserving renders.

Further, advanced users can change the cap on maxIterations from the default 15,000 to 5,000 or 50,000.

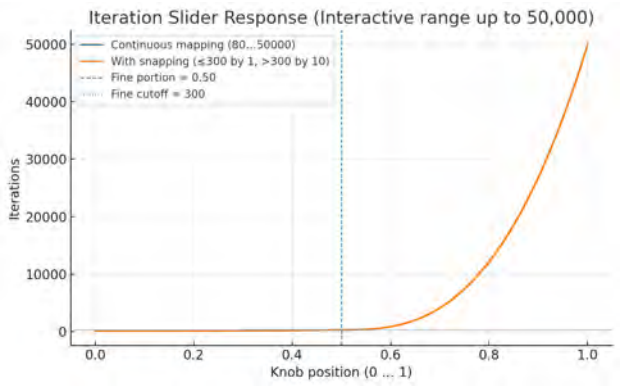


Figure 5 – Iteration slider manual response curve is non-linear to provide precise control at low iterations.

Coloring Smoothing Algorithm

To avoid aliasing, I compute a smooth iteration value, μ, where:

$$\mu = n + 1 - \frac{\ln(\ln|z|)}{\ln 2}$$

I then map μ into a palette—built-in or lookup table (LUT)—with linear interpolation between gradient stops. Contrast is applied in a perceptual space for predictable mid-tone control.

This produces gradients instead of harsh bands. Palettes are applied by mapping μ to a color gradient. Colors are averaged in linear space, then re-encoded for display.

When Exact LUT is enabled, Mandelbrot Metal provides optional step marker overlays. These make visible the discrete palette sampling (intentional banding) so users can clearly see the LUT step boundaries. This is useful for educational purposes, palette debugging, and artistic effects.

Color Palettes

Mandelbrot Metal features a large palette library. Some of the capabilities and features of the palette system include:

- **Built-in Library** – 140+ professionally crafted palettes spanning scientific, artistic, and creative styles.
- **Favorites** – Mark your favorite palettes and they will appear at the top of the library with a ★.
- **Palette Editor** – Create fully custom palettes by defining and adjusting gradient stops, with live preview and precise color stop positioning. Custom palettes may be individually deleted.
- **LUT-Based Implementation** – Palettes are stored as 1×N lookup textures in either sRGB or Display P3 color space. Linear interpolation between stops is performed in a gamma-aware manner for perceptually accurate color blending.

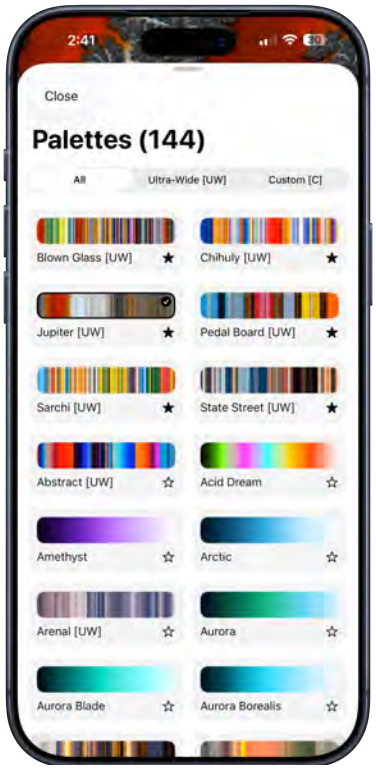


Figure 6 – The palette library in Mandelbrot Metal

- **Configurable Resolution** – LUT width can be switched between 256 and 1024 samples, allowing a trade-off between gradient smoothness and memory usage.
- **Ultra-Wide (768-step) palettes** – Reveal fine detail or ultra-smooth ramps. Currently, there are 40+ in the library.
- **Real-Time Swapping** – Palette changes apply instantly without triggering a re-iteration pass; coloring is decoupled from the iteration workload.
- **GPU-Optimized** – All palettes are designed for efficient GPU sampling, with minimal overhead in both deep zoom and high-iteration scenarios.
- **Import Gradient** – Sample images in your photo library as a source for custom gradients to fine-tune and then save as to the palette library for reuse.
- **Per-Palette Metadata** – Each palette can store metadata such as name, category, description, and favorite status, making it easy to organize large collections.
- **Palette Sharing and Inter-Device Portability** – Custom and imported palettes can now be exported or imported as JSON LUT files for use on other devices. Each palette's metadata is serialized (name, category, color-space, favorite status), ensuring perfect reconstruction when shared.

The palette library includes a range from scientific to artistic schemes, all GPU-optimized. You can create your own palettes by editing gradient stops.

GPU Pipeline (Metal)

Mandelbrot Metal's real-time renderer is built around a tightly optimized Metal compute pipeline. On supported devices, **all interactive exploration runs on the GPU**, with the CPU primarily orchestrating state, precision, and deep-zoom fallbacks.

Frame Input & Viewport State

Each frame starts by capturing the current viewing state:

- **Complex viewport**
  - Center point $c_0 = x_0 + y_0 i$
  - Scale / magnification (pixels → complex plane)
  - Aspect ratio and pan offsets
- **Render configuration**
  - Max iterations (adaptive, based on zoom level & device)
  - Anti-aliasing mode (SSAA / 1×)
  - 3D Look parameters (height scale, light direction, ambient/diffuse balance)
  - Dithering and color-space flags (sRGB vs Display-P3, HDR readiness)
- **Palette / LUT state**
  - Active palette ID
  - Packed LUT texture (256 or 768 steps) mapped into a unified 1024-tap pipeline

This state is written into **per-frame uniform buffers** and pushed to the GPU before each dispatch.

Command Buffer & Compute Pipeline Setup

For each frame:

1. The CPU creates a **MTLCommandBuffer** and a **compute command encoder**.
2. It binds:
    - The output texture (RGBA float or half-float, depending on quality settings)
    - The uniform buffer (viewport, iteration settings, 3D Look controls)
    - The palette/LUT texture and any auxiliary buffers (heightmap, normal hints, etc. if used)
3. It selects the appropriate **compute pipeline state**:
    - Standard Mandelbrot GPU kernel for interactive zoom/pan
    - Optional variants for higher iteration caps or SSAA

The renderer uses **tile-based dispatching** sized to match the device's threadgroup sweet spot (balancing occupancy and cache behavior).

Compute Pass 1 — Orbit Iteration & Escape Time

The first stage computes the Mandelbrot orbit and escape metrics per pixel:

- Each GPU thread corresponds to **one pixel sample** (or one sample in a multi-sample SSAA scheme).
- The kernel:
    1. Maps the pixel coordinate (x, y) to a complex c via the current viewport transform.
    2. Iterates:

$$\mathbf{z_{n+1} = z_n^2 + c}$$

    ...up to the configured max iterations or until $|z_n|^2 > 4$.

    3. Tracks:
        - Escape iteration count
        - Smoothed distance / log-escape value for continuous coloring
        - Optional derivative or height hint for 3D Look shading
- The results are written into:
    - Either an intermediate buffer (escape metrics)
    - Or directly into the output texture as a **normalized "color index"** (e.g., a float mapping into the LUT).

This pass is heavily optimized to **minimize divergence** and keep the hot loop in registers, leveraging Metal's SIMD-friendly math.

Compute Pass 2 — Palette Lookup & 3D Look Shading

The second stage converts iteration/escape data into the final color:

1. **Palette lookup**
    - Uses the normalized escape value to index into a **1D LUT texture**.
    - Supports both:
        - Standard palettes (effectively 256 steps)
        - Ultra-Wide palettes (up to 768 steps)
    - All palettes are mapped into a **1024-tap unified LUT space**, so the kernel always performs the same simple lookup and interpolation logic.
2. **3D Look shading**
    - Interprets the escape data (or derivative/height hints) as a **height field**.
    - Computes an approximate **surface normal** per pixel.
    - Applies a simple but effective lighting model:
        - Ambient + diffuse + optional specular from a virtual light at a configurable azimuth/elevation.
    - Modulates the LUT color with the lighting response to create **sculpted contours and depth**.
3. **Dithering & color space**
    - Applies subtle dithering to reduce banding, especially in smooth gradients and Ultra-Wide palettes.
    - Converts the final color into the correct output space:
        - Standard dynamic range (sRGB)
        - Or **wide color (Display-P3)** on capable devices.

The result is written directly back into the output texture that will be presented for the frame.

Anti-Aliasing & Super-Sampling (SSAA)

When SSAA is enabled:

- The GPU renders at a **higher internal resolution** (e.g., 2× in each dimension).
- Multiple sub-pixel samples are computed per visible pixel.
- A final **resolve step averages** these high-res samples into the display-resolution texture.

This reduces aliasing on fine filaments and intricate boundary regions, at the cost of more GPU work. The app dynamically exposes or constrains SSAA options based on device capabilities.

Command Buffer Commit & Presentation

Once the compute passes complete:

1. The command encoder is ended and the command buffer is **committed**.

2. The rendered texture is:
    - Bound into SwiftUI / UIKit drawing layers
    - Or blitted into an on-screen drawable (e.g., via CAMetalLayer or a bridge object)
3. The UI overlays (HUD, toolbars, orbit HUD, etc.) are composited on the CPU/GPU graphics stack above the fractal layer.

From the user's perspective, this yields a **smooth 60 FPS-class exploration** at shallow and moderate zooms on modern devices.

Transition to Deep Mode (CPU Path)

When the magnification surpasses the GPU precision envelope, Mandelbrot Metal detects that FP32/FP16 math is no longer sufficient:

- The app gracefully **hands off to Deep Mode**:
    - GPU path is paused for that region.
    - CPU-based high-precision tiling (Double / Double-Double) takes over.
- The GPU remains active for:
    - UI compositing
    - HUD / overlays
    - Transitional effects while Deep Mode tiles progressively fill in.

This hybrid design ensures the GPU is used **exactly where it shines most** (real-time exploration), while the CPU path covers extreme zooms that demand higher numerical precision.

Super-Sampling Anti-Aliasing (SSAA)

General SSAA (unweighted box filter)

For a pixel centered at (x, y) and N sub-pixel samples at offsets $(\delta x, \delta y)$:

$$\mathbf{C}(x, y) = \frac{1}{N} \sum_{k=1}^{N} \mathbf{L}(x + \delta x_k,\ y + \delta y_k)$$

L returns linear-space RGB—or intensity—values.

Weighted form (any filter):

$$\mathbf{C}(x, y) = \sum_{k=1}^{N} w_k\, \mathbf{L}(x + \delta x_k,\ y + \delta y_k), \quad \sum_{k=1}^{N} w_k = 1,\ w_k \geq 0$$

Sample positions for SSAA×m (m × m grid)

Let $\mu, \nu \in (0, \ldots, m-1)$ and $M = m^2$. A canonical grid about the pixel center:

$$\delta x_{uv} = \frac{u + \frac{1}{2} - \frac{m}{2}}{m}, \quad \delta y_{uv} = \frac{v + \frac{1}{2} - \frac{m}{2}}{m}, \quad u, v \in \{0, \ldots, m-1\}, \; N = m^2$$

Color-management note (average in linear space)

If $\mathbf{\Gamma}$ represents the display's transfer function (e.g., sRGB or Display P3), I perform the averaging in linear color space, then re-encode the result using Γ:

$$\mathbf{c}_{\text{display}} = \Gamma\left(\frac{1}{N}\sum_{k=1}^{N} \Gamma^{-1}(\mathbf{c}_k)\right)$$

Mandelbrot-specific instantiation

Map each sub-sample to the complex plane C_k, run escape-time, compute smooth iteration μ_k, palette in linear space, then average:

$$\mu_k = n_k + 1 - \frac{\ln(\ln|z_k|)}{\ln 2}$$

$$\mathbf{c}_{\text{display}} = \Gamma\left(\frac{1}{N}\sum_{k=1}^{N} \mathbf{c}_k\right)$$

If not converting to linear, I will replace C_k with:

$$\mathbf{c}_k = \text{Palette}_{\text{lin}}(\mu_k)$$

At deep zooms, the boundary's high spatial frequency can alias. When HQ idle is active—and only when interaction pauses—I switch to SSAA:

- For SSAA×2, ×3, and ×4, I sample a rotated grid of 4/9/16 sub-pixels per pixel, respectively.
- Each sub-sample runs a full escape test; colors are averaged in registers and written once.
- Sample patterns are chosen to suppress moiré; the palette lookup is done per sub-sample to keep gradients smooth.

Auto Iterations & Manual Slider

Detail demands grow with magnification. I compute an iteration target from the current scale relative to a baseline:

$$n_{\max} \approx n_{\text{base}} + a\log_{10}(\rho) + b\,[\log_{10}(\rho)]^2 + c$$

where ρ is the scale ratio and *a, b,* and *c* are tuned constants. This keeps fine filaments crisp without unnecessary work at wide views.

Manual control is provided by a nonlinear slider mapping slider position s ∈ [0,1] to iteration count:

$$n(s) = \lfloor n_{\min} (\frac{n_{\max}}{n_{\min}})^{s^\gamma} \rfloor, \quad s \in [0,1], \ \gamma > 0$$

Adaptive Precision & Double-Double Precision

The renderer dynamically promotes numeric precision based on zoom depth and pixel scale. FP32 is used for shallow zooms, FP64 for deeper views, and double-double arithmetic (~106-bit mantissa) for extreme zooms up to 10^{16} and export-quality stills. Double-double is implemented as:

1. FP32 (single) for wide views and moderate zooms—the fastest.
2. FP64 (double) once pixel size in the complex plane falls below a threshold (e.g., $<10^{-8}$), preventing coordinate collapse.
3. Double-Double (DD) for extreme zooms: I represent a value *x* as *x = x_hi + x_lo*, where each term is a 64-bit double. With Dekker-Veltkamp-style error-free transforms, I implement add/multiply with explicit error correction:

    - Two-Sum (error-free addition of doubles)
    - Two-Prod-FMA (error-captured multiplication using Fused Multiply-Add)
    - Complex multiply uses DD operations on both real and imaginary parts, preserving ~106 bits (~32 decimal digits) of precision.

When I switch: I monitor the world-space delta between adjacent pixel centers and compare it against the effective mantissa of the current mode (including accumulated rounding in the escape loop). If precision margin drops below a safety factor, I promote the kernel to the next mode. Promotion occurs during HQ idle, so interaction stays snappy. Demotion occurs when zooming back out.

Perturbation Rendering

At extreme zoom depths, Mandelbrot Metal employs *perturbation rendering* to maintain both speed and accuracy. Instead of recalculating every pixel independently at full precision, the renderer computes a single **reference orbit** at very high precision (double-double). Each pixel then evaluates only the *delta* (perturbation) from that reference.

This reduces per-pixel workload and avoids catastrophic loss of significance while zooming beyond 1e15 and deeper. The framework falls back to standard double-double arithmetic when perturbation accuracy margins are insufficient.

Key benefits:

- Enables **deeper zooms** at interactive frame rates.

- Greatly reduces **GPU workload** by avoiding redundant high-precision orbit computations.
- Works seamlessly with existing **auto-iteration scaling** and **palette smoothing**.

Why This Matters

The Mandelbrot set is the best example I know of emergent complexity from simple rules. With Mandelbrot Metal, you don't just view pictures—you explore a living mathematical landscape at full fidelity, guided by a renderer that adapts its precision and quality to what you're doing.

Gradient Import

Gradient Import lets you turn any image into a palette. The app samples a row, column, or averaged stripe, converts the colors to linear space, smooths them, and stores them as a 1×N LUT. You can then edit stops, adjust contrast, and save the palette with metadata. This enables artistic exploration using palettes derived from nature, artwork, or photographs.

- LUT = Look-Up Table. In graphics, this usually means a precomputed array of colors that can be indexed quickly.
- 1×N = a 1-pixel-tall, N-pixel-wide texture.
    - The "1" means it's just a single row (one dimension of variation).
    - The "N" means it has *N* samples (256 or 1024 pixels wide)
    -

So essentially, it's a strip of colors laid out horizontally.

- Each entry corresponds to a palette "stop."
- When rendering, the shader takes the normalized smooth iteration value (say between 0 and 1), multiplies by N, and samples from this texture.
- Because it's a GPU texture, the hardware automatically interpolates between samples (linear filtering), so colors blend smoothly.
- User may toggle Exact LUT on/off. With Exact LUT disabled, the colors do not blend smoothly and are allowed to band.

Imported gradients are persistent across app restarts and can be renamed and stored in the palette library alongside built-in palettes. Each imported gradient is editable, can be saved under a custom name, and restored later from library storage.

Image Capture

Mandelbrot Metal's capture system is designed to preserve the fidelity of your fractal exploration with deterministic, high-resolution output.

How Capture Works

- **Exact Canvas Render** – The capture path renders the fractal at the device's full canvas resolution to avoid tiling seams or scaling artifacts.
- **Resolution Options** – Users can export directly at canvas size or upscale to 4K, 6K, 8K, or custom dimensions, with aspect-fit to maintain framing.

- **PNG Export** – Images are saved as lossless PNGs with proper color profile embedding (sRGB or Display-P3), ensuring accuracy across displays and print workflows.
- **Gesture Commitment** – Any in-progress pan or zoom gesture is finalized before capture, so the exported image matches exactly what was seen on screen.

Determinism & Reproducibility

- **Bookmarks** – Each capture can be tied to a bookmark that stores center coordinates, scale, iteration settings, palette, 3D state, and contrast values. This guarantees the same image can be reproduced later. See Sharing and Collaboration below.
- **GPU Pipeline Stability** – Capture results are deterministic per precision mode; identical settings will yield identical exports across sessions.

So, Mandelbrot Metal is not just a *viewer* but a production-grade fractal renderer, capable of generating reproducible, high-fidelity outputs suitable for wallpapers, teaching, or even large-format prints.

Sharing and Collaboration

Version 1.1 introduced **Bookmark and Palette Sharing**, a foundation for community-driven exploration.

Overview

Mandelbrot Metal now lets you exchange your discoveries — both visual and mathematical — with other users. You can export any saved bookmark or palette as a portable JSON file and import shared files from others to instantly recreate their views and color schemes. See the Appendix for the JSON schemas.

Bookmarks

Bookmarks are stored presets in JSON format. Each bookmark contains:

- A unique name given by the user
- Center coordinates (Real and Imaginary components)
- Magnification (scale)
- Precision mode (FP32 / FP64 / Double-Double)
- Max iteration limit (or Auto Iteration flag)
- Palette and contrast settings
- 3D Look parameters

Bookmarks are exported as a small human-readable JSON document. Importing restores the exact view and parameters so any user can reproduce the identical render on their device. Bookmarks also include version tags to ensure forward compatibility as the app evolves.

Palettes

Custom palettes can be shared the same way. Each exported palette file encodes its gradient stops, color space, and LUT metadata (Display-P3 or sRGB). Imported palettes appear instantly in your library with their original names and categories. You can favorite, rename, or delete them just like built-ins.

Import and Export Process

- **Export:** In *Manage Bookmarks* or *Manage Palettes*, swipe-right on an item to reveal the Export button or tap "Export All" to create a bundle of everything.
- **Import:** Tap the Import button at the top of the sheet to add one or many shared items.

All imports are validated and de-duplicated automatically.

Community Portal (Upcoming)

Later in 2025, the **Mandelbrot Metal Community Portal** will extend sharing beyond local files. Users will be able to browse, upload, and rate shared bookmarks and palettes directly from the app or mandelbrot-metal.com. Each submission will include metadata (preview thumbnail, zoom depth, iteration count, palette name) for easy search and curation.

Why It Matters

Fractal exploration thrives on collaboration. By standardizing how coordinates and palettes are exchanged, Mandelbrot Metal turns every user into a contributor — helping build a shared atlas of infinite complexity.

3D Look Rendering Mode

The 3D Look feature introduces surface lighting to the Mandelbrot Metal rendering pipeline. Unlike traditional flat color mapping, this mode derives a normalized "height field" from the smooth iteration count (ν-smoothing). Surface gradients are estimated per pixel, either via GPU hardware derivatives (ddx/ddy) or CPU finite differences when in deep mode. Normals are constructed from these gradients and lit with a configurable Blinn–Phong model:

- **Lambertian diffuse** with ambient bias (0.25 + 0.75·N·L)
- **Specular highlights** with adjustable shininess and specular strength
- **User-defined lighting parameters** (height scale, light direction, overall strength)

These parameters are synchronized across GPU and CPU paths to ensure consistency even at extreme zooms. The effect is blended with the base palette color, creating a controllable illusion of depth without altering the underlying fractal geometry.

The result is a shaded fractal rendering that preserves color fidelity while revealing fine topological structure. Because the 3D Look operates on the iteration data already computed for each pixel, performance overhead is modest, and the mode is available interactively in both GPU and CPU rendering contexts.

Coordinate Jump

While Mandelbrot Metal excels at intuitive touch navigation, some users—particularly researchers and enthusiasts—want direct access to precise regions of the fractal without relying on manual zoom and pan. To support this, Mandelbrot Metal introduces Coordinate Jump, a feature that allows you to specify an exact rectangular domain in the complex plane and immediately reposition the viewport there.

How it works

Users enter four bounds: minX, minY, maxX, maxY. These values define a rectangular region in the complex plane. The app then:

1. **Centers** the viewport on the rectangle.
2. **Adjusts** the width or height as needed to match the device's display aspect ratio, ensuring that the rendered image fills the screen without stretching or adding borders.
3. **Calculates** the appropriate pixels-per-unit (scale) so the selected domain is mapped with pixel-accurate fidelity to the canvas.

The result is a mathematically faithful view of the requested coordinates, scaled and framed exactly to the device. Shared coordinate sets can be exported as bookmarks for collaborative study.

Interesting Coordinates

Region / Landmark	minX	minY	maxX	maxY	Notes
Full Set	-2.0	-1.5	1.0	1.5	Frames the entire Mandelbrot set.
Main Cardioid	-0.9	-0.3	-0.6	0.3	Shows the heart-shaped center and largest bulb.
Seahorse Valley	-0.775	0.085	-0.725	0.115	Iconic spiral "seahorse tails."
Elephant Valley	0.225	-0.05	0.275	0.05	Distinct "elephant trunk" spirals.
Baby Mandelbrot	-1.30	-0.05	-1.20	0.05	Mini replica ("satellite set").
Spiral Valley	-0.7435	0.1310	-0.7425	0.1320	Fine spiral filigree.
Triple Spiral Valley	-0.088	0.654	-0.084	0.658	Nested spirals — detailed but not too deep.
Satellite Chain	-1.78	0.0	-1.72	0.08	Chain of mini-Mandelbrots linked together.

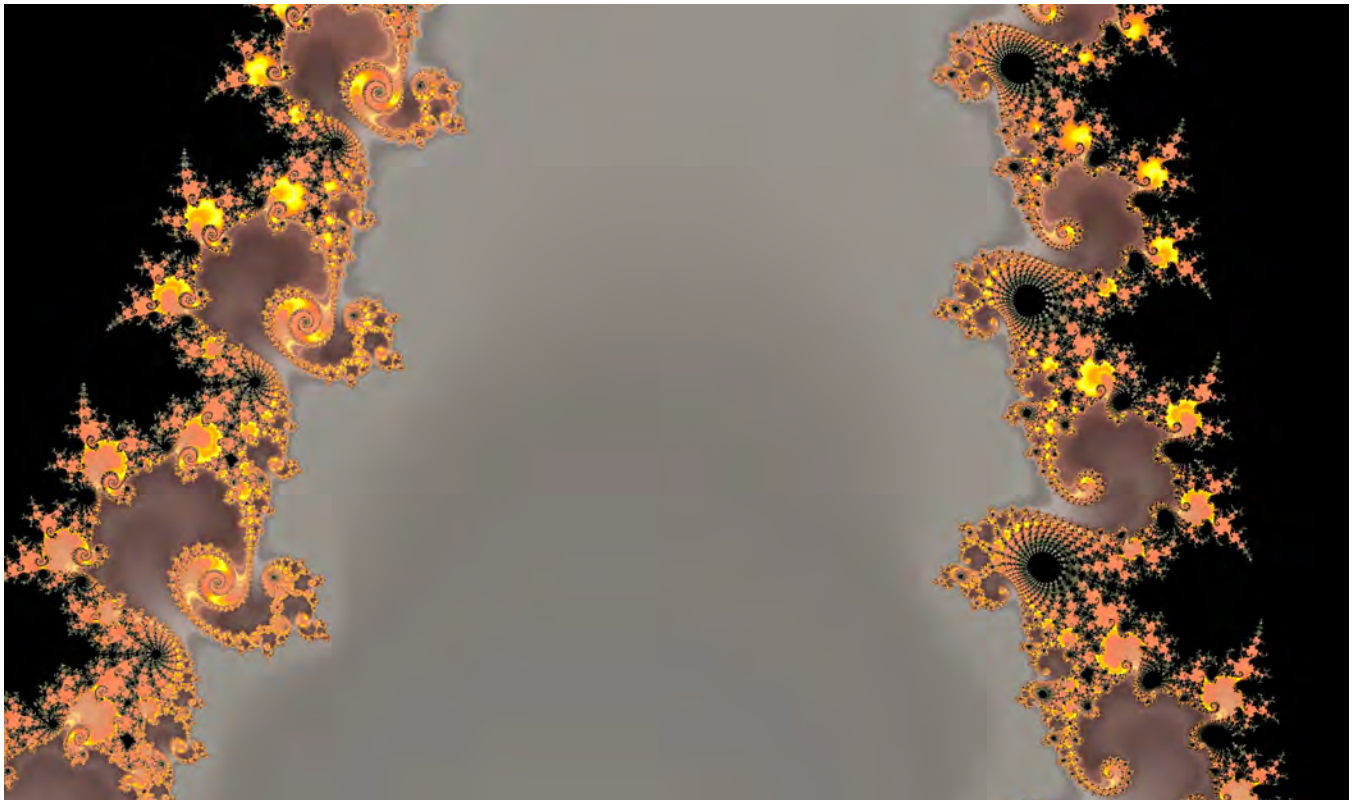


Figure 7 – The Seahorse Valley rendered via Coordinate Jump

Why it matters

- **Reproducibility**: Users can record and share coordinate sets, ensuring that colleagues or students can reproduce the exact same view.
- **Access to known features**: Iconic regions such as **Seahorse Valley** (≈ −0.75 + 0.10i), **Elephant Valley** (≈ 0.25 + 0i), and miniature "baby" Mandelbrots (≈ −1.25 + 0i) can be reached directly.
- **Curated exploration**: Mandelbrot Metal will publish curated coordinate lists on the project website, allowing one-tap access to notable locations.

Some Examples of Captured Images

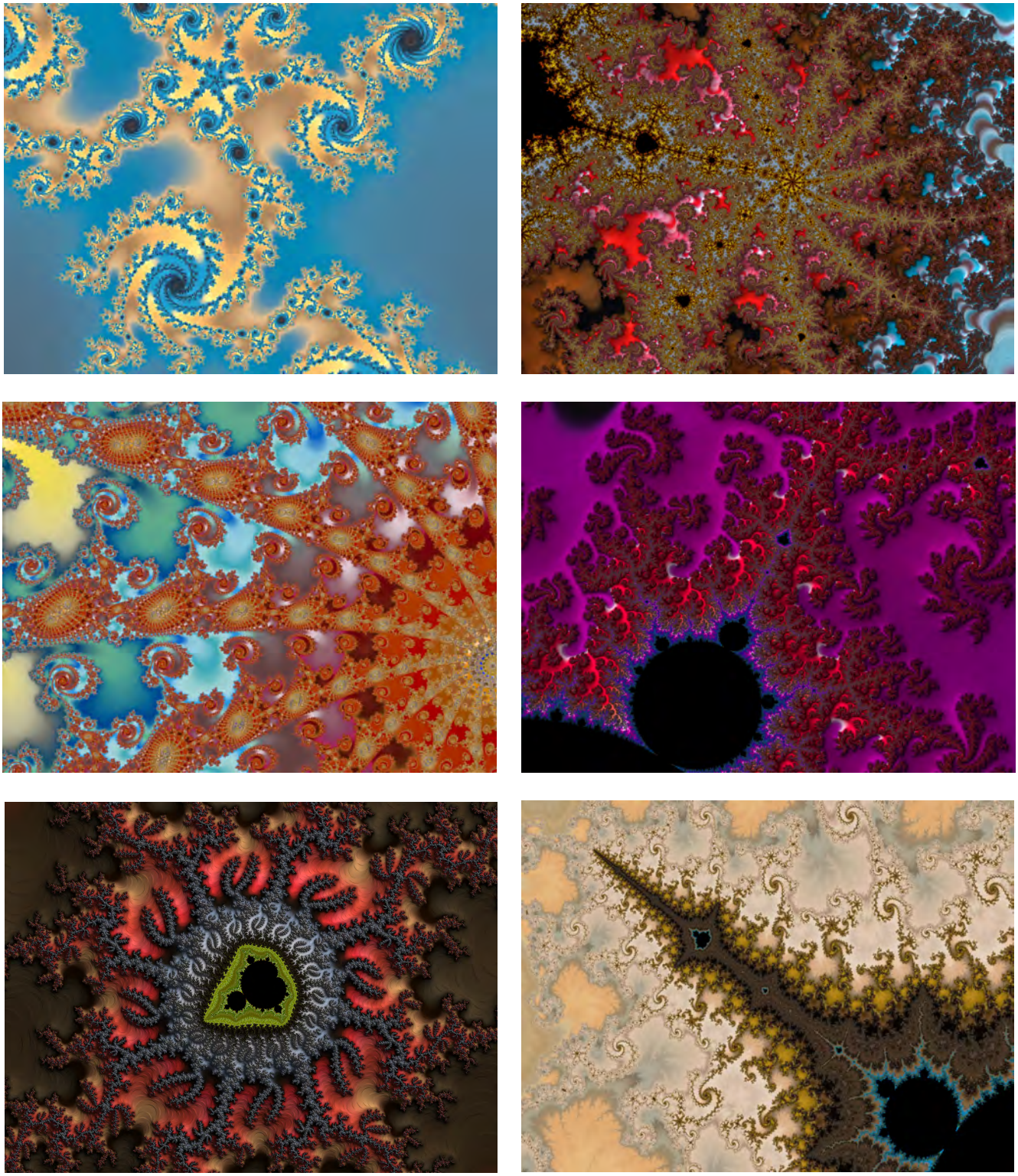


Figure 8 – Sample rendered Mandelbrot set images in Mandelbrot Metal, showing detail and smooth coloring at deep zoom levels. Each point represents a complex number c used in the Mandelbrot formula. Zoomed regions reveal repeating patterns. For more artwork samples, visit the Gallery.

Stability and Reproducibility

- Arithmetic pathways are deterministic per device and precision mode; palette LUTs are versioned; bookmarks store center/scale/iterations/contrast/palette name, so scenes are reproducible.
- The HUD exposes center, scale, iteration budget (or auto), SSAA status, and palette name to make captures auditable.

Benchmarks

On recent model iOS and iPadOS devices, the compute kernel sustains interactive rates at 1K–3K iterations with SSAA off during interaction, promoting to SSAA×2, SSAA×3, SSAA×4, and higher iterations when idle.

Introduced in version 1.0.4, Deep Rendering Mode is fully parallelized across all available CPU cores on Apple Silicon, delivering up to 15× faster performance. Deep Mode now renders tiles progressively from center out in real time, rather than waiting for full completion, dramatically reducing perceived latency during extreme-precision exploration.

Double-double precision remains reserved for the deepest zooms and is enabled automatically only when needed to preserve responsiveness.

Recent Benchmarks (using version 1.0.4)

Device	Precision	Iterations	SSAA	Resolution	Frame rate (interactive)
iPhone 16 Plus	FP64	800	Off	2796 × 1290	~75–90 fps
iPad Pro 13” M4	FP64	8,000	Off	2752 × 2064	~60–70 fps
iPad Pro 13” M4	Double-Double	15,000	×2	2752 × 2064	~45–60 fps

Figure 9 — Deep Rendering Mode automatically engages at magnifications above ≈ 2×10^7, switching from the GPU path to a high-precision CPU path optimized for parallel execution across all performance and efficiency cores. Each tile renders independently, updating the display progressively from center out, as results complete. This architecture maintains accuracy for extreme-precision (FP64 and Double-Double) arithmetic while achieving real-time, tile-by-tile feedback during the deepest zoom levels.

Roadmap/Future Work

The roadmap on my website shows the features and improvements in progress and planned. It is publicly available at mandelbrot-metal.com/roadmap.

About the Developer

Mandelbrot Metal is an independent software studio dedicated to exploring the intersection of art, math, and performance. Founded by developer Michael Stebel, the studio’s mission is to push the limits of Apple’s hardware and graphics technologies to create apps that are as powerful as they are beautiful.

Our flagship app, **Mandelbrot Metal**, brings the infinite complexity of the Mandelbrot set—and other fractals—to life on iPhone and iPad. With real-time rendering, ultra-deep zoom capability, 3D effects, over 140 vibrant curated color palettes, and bookmark & palette sharing, Mandelbrot Metal transforms abstract

mathematics into an immersive visual experience — putting the beauty of infinity at your fingertips. For more information and app support, visit mandelbrot-metal.com.

Mandelbrot Metal is a registered DBA (Doing Business As) of Michael Stebel Consulting, LLC, a Florida limited liability company.

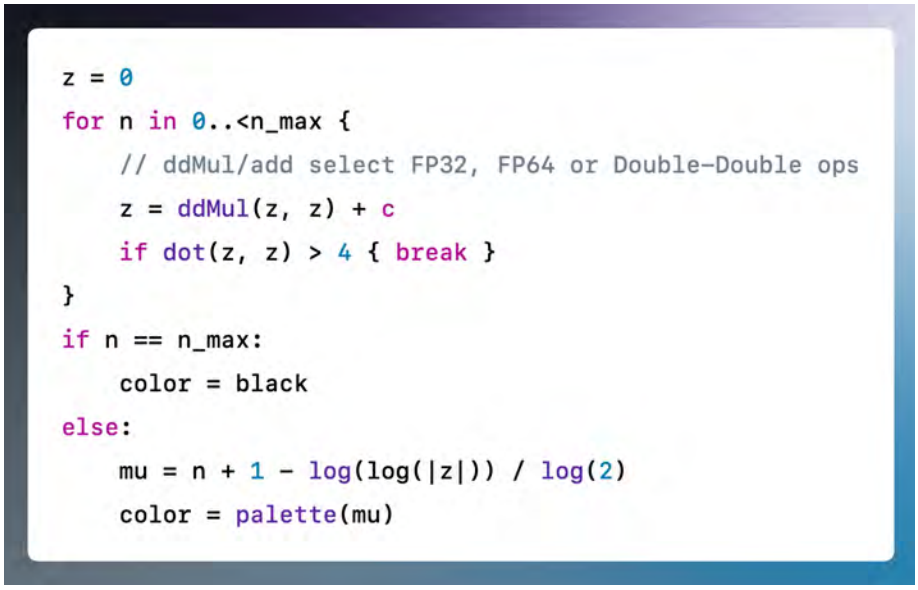
Trademarks

Apple, the Apple logo, iPhone, iPad, Mac, macOS, iOS, Xcode, Swift, and Metal are trademarks of Apple Inc., registered in the U.S. and other countries and regions. All other product names, logos, and brands are property of their respective owners.

Appendix

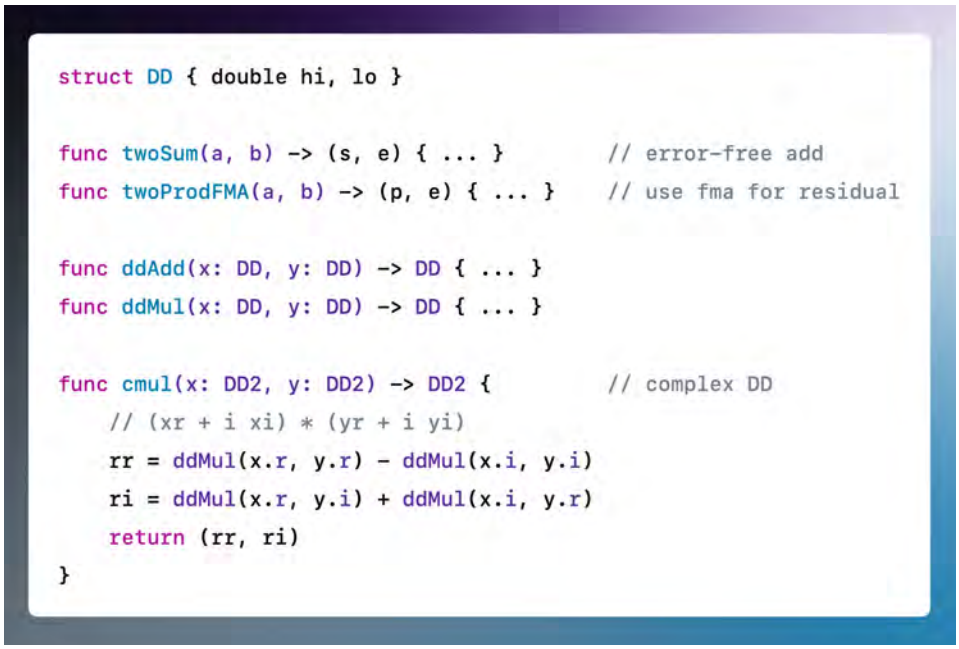
Pseudocode for Concept Clarity

Escape + Smooth Coloring (per sample)



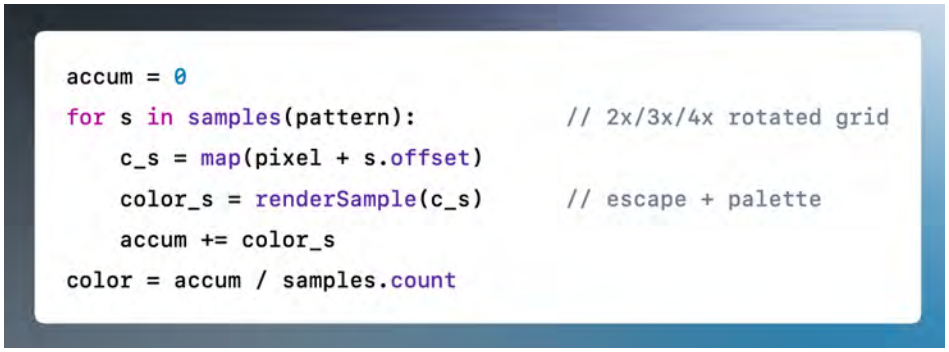
```
z = 0
for n in 0..<n_max {
    // ddMul/add select FP32, FP64 or Double-Double ops
    z = ddMul(z, z) + c
    if dot(z, z) > 4 { break }
}
if n == n_max:
    color = black
else:
    mu = n + 1 - log(log(|z|)) / log(2)
    color = palette(mu)
```

Double-Double building blocks (sketch)



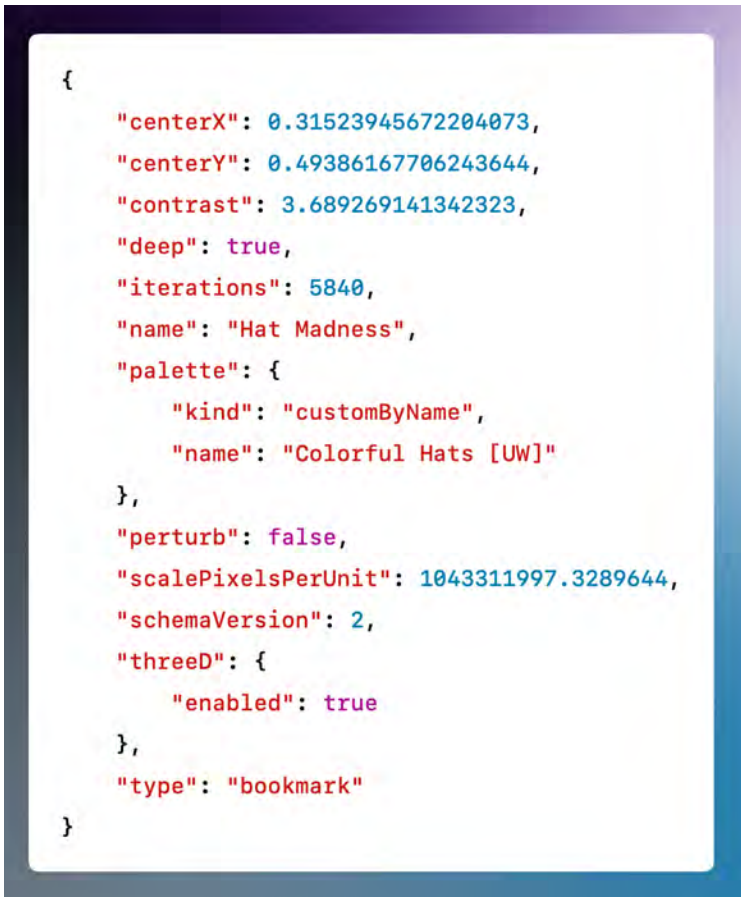
```
struct DD { double hi, lo }

func twoSum(a, b) -> (s, e) { ... }        // error-free add
func twoProdFMA(a, b) -> (p, e) { ... }    // use fma for residual

func ddAdd(x: DD, y: DD) -> DD { ... }
func ddMul(x: DD, y: DD) -> DD { ... }

func cmul(x: DD2, y: DD2) -> DD2 {         // complex DD
    // (xr + i xi) * (yr + i yi)
    rr = ddMul(x.r, y.r) - ddMul(x.i, y.i)
    ri = ddMul(x.r, y.i) + ddMul(x.i, y.r)
    return (rr, ri)
}
```

SSAA



```
accum = 0
for s in samples(pattern):            // 2x/3x/4x rotated grid
    c_s = map(pixel + s.offset)
    color_s = renderSample(c_s)       // escape + palette
    accum += color_s
color = accum / samples.count
```

Bookmark JSON Schema



```
{
    "centerX": 0.31523945672204073,
    "centerY": 0.49386167706243644,
    "contrast": 3.689269141342323,
    "deep": true,
    "iterations": 5840,
    "name": "Hat Madness",
    "palette": {
        "kind": "customByName",
        "name": "Colorful Hats [UW]"
    },
    "perturb": false,
    "scalePixelsPerUnit": 1043311997.3289644,
    "schemaVersion": 2,
    "threeD": {
        "enabled": true
    },
    "type": "bookmark"
}
```