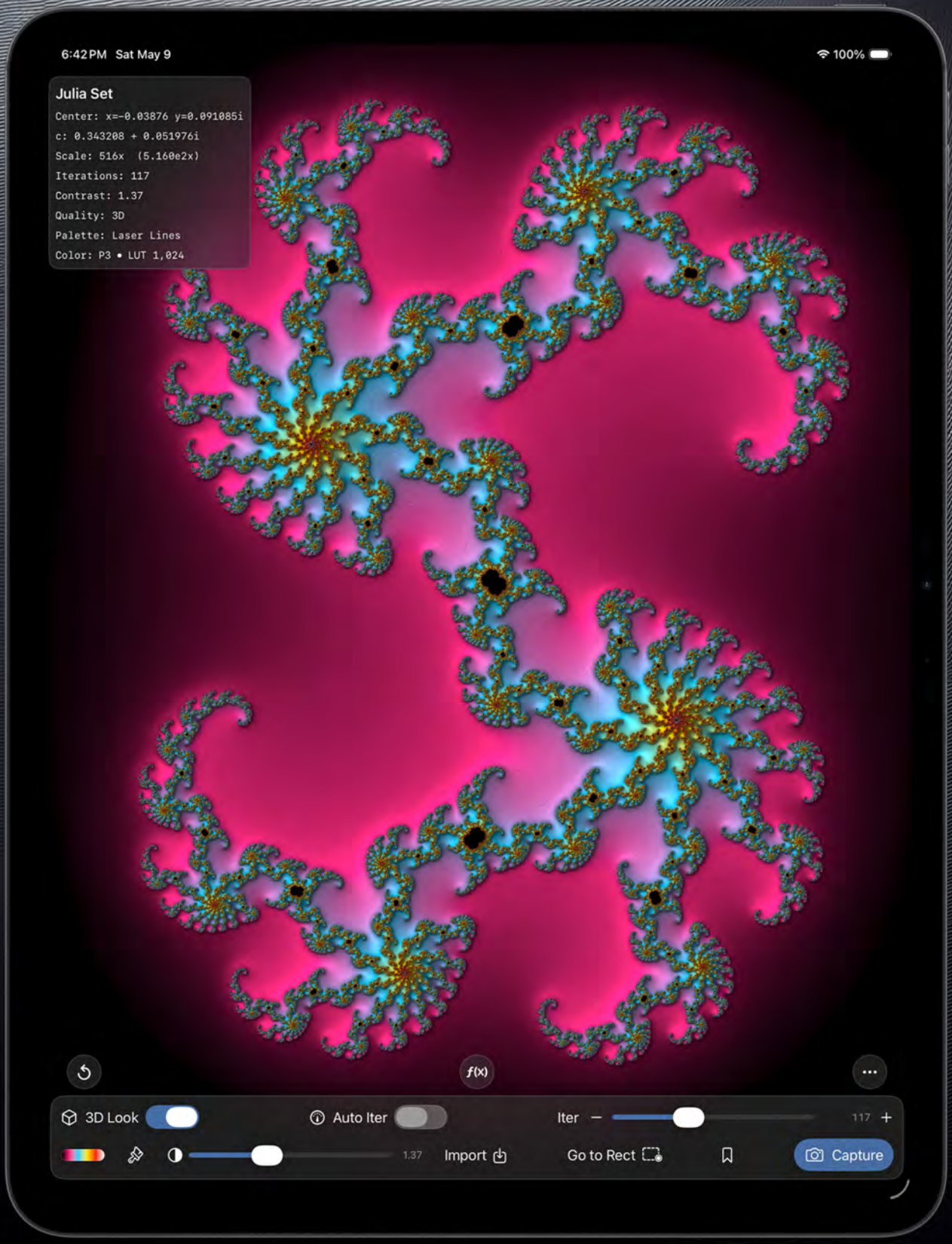


Mandelbrot Metal

Technical White Paper

Mandelbrot Metal

Technical White Paper — Version 2.0: Mandelbrot and Julia Sets

Executive Summary

Mandelbrot Metal is a high-performance fractal renderer and explorer for iPhone and iPad. Version 2.0 expands the app beyond a Mandelbrot-only workflow, adding Julia Set exploration and a direct tap-to-Julia path: single-tap any point in a Mandelbrot render to generate the matching Julia Set from that exact complex value. The release preserves the same Metal-accelerated rendering architecture, adaptive numeric precision (float -> DS -> DD), continuous coloring, super-sampling anti-aliasing, and flexible palette system with gamma-aware interpolation.

The renderer prioritizes interactivity during navigation and automatically increases precision, iteration depth, and image quality when interaction pauses. This design produces deterministic, studio-grade images without requiring manual tuning. Additional v2.0 features include a reversible Julia side trip, HUD display of the active Julia c constant, high-resolution export, Julia-aware bookmarks, and sharing metadata that preserves both fractal mode and Julia source values.

Mandelbrot Metal combines mathematical rigor with production-grade rendering to transform fractal exploration into a responsive, precise, and visually uncompromising experience.

Overview

Mandelbrot Metal was created to make deep fractal exploration fast, precise, and beautiful. It runs a Metal compute kernel that evaluates Mandelbrot and Julia escape-time dynamics in parallel, maps continuous iteration values to color via GPU lookup tables (LUTs), and presents results with minimal CPU–GPU overhead.

The design goals are straightforward: instantaneous responsiveness during navigation and uncompromising fidelity when still. While interacting, the renderer prioritizes speed; when interaction stops, it automatically raises precision, iteration count, and quality. This ensures high-quality output without exposing users to unnecessary technical complexity.

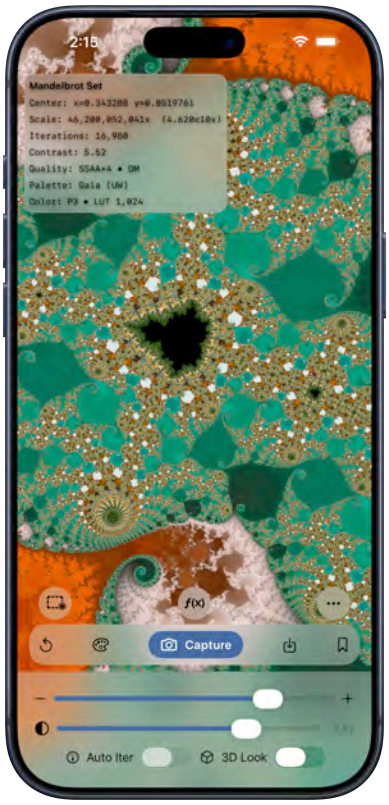


Figure 1 – Mandelbrot Metal on the iPhone 17 Pro Max

The result is a fractal explorer that behaves like a real-time instrument while moving and a production renderer when idle, now with both explicit mode selection and one-tap generation of Julia Sets from Mandelbrot coordinates.

How to Read this Paper

Readers interested in fractal mathematics may start with The Mandelbrot Set: An Overview, Julia Sets in Version 2.0, and Tap-to-Julia Exploration. Those focused on rendering technology should jump to GPU Pipeline (Metal), Adaptive Precision, and Deep Rendering. Artists and creators may prefer Color Palettes, 3D Look Rendering Mode, Image Capture, and Sharing and Collaboration.

Key Features of the App

- Real-time GPU rendering of the Mandelbrot Set and Julia Set
- Toolbar mode selector with Mandelbrot Set as the default and Julia Set as the second fractal family
- Single-tap any Mandelbrot point to generate the matching Julia Set from that exact complex value
- Single-tap in Julia mode to return to the exact Mandelbrot view, zoom, and iteration state that created it
- HUD displays the active Julia c constant in Julia mode
- Extreme deep zooms up to $10^{16}\times$
- 155+ curated scientific and artistic color palettes
- 55+ Ultra-Wide (768-step) 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 1024 resolution
- Instant palette swapping (coloring decoupled from iteration work)
- Nonlinear Iteration Slider for fine and wide-range control

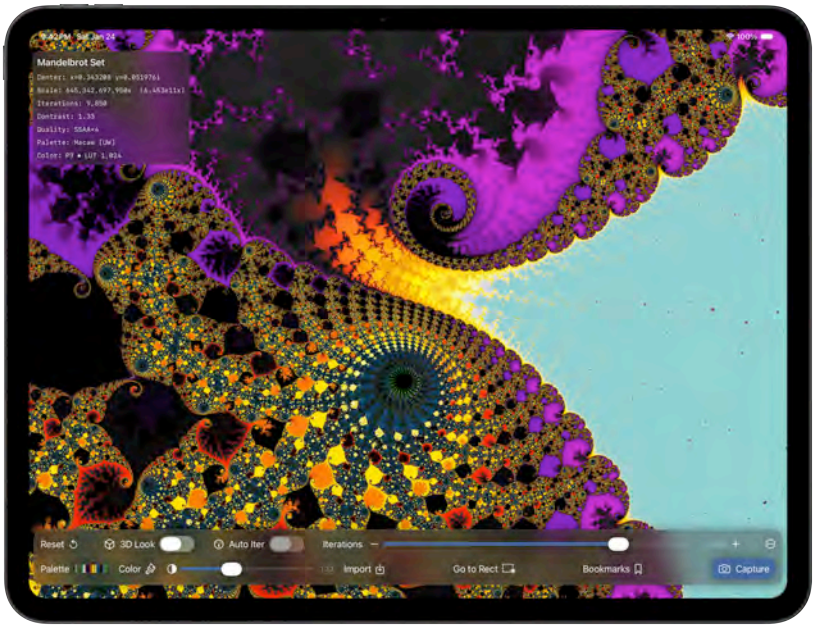


Figure 2 – Mandelbrot Metal running on the iPad Pro 13-inch (M5)

- Auto Iteration scaling with zoom depth
- Adaptive Precision (float → DS → DD)
- 3D Look mode
- HUD (heads-up display)
- Deterministic capture and bookmarking with sortable management views, including saved fractal mode
- Bookmark and Palette sharing (import/export in JSON format)
- Julia bookmarks and shared bookmark files preserve the saved c value
- Refined iPad control layout with a cleaner utility row above the floating toolbar
- CLI program converts photos to palettes for advanced users

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.

Julia Sets in Version 2.0

Julia sets use the same recurrence as the Mandelbrot set, $z \leftarrow z^2 + c$, but invert the role of the pixel coordinate. Instead of testing each coordinate as c with z starting at zero, Julia mode treats each pixel as the starting z and uses a fixed constant c for the whole image.

In the current renderer, that constant is stored in the Metal uniforms as juliaC and defaults to (-0.8, 0.156). In v2.0, juliaC can also be derived directly from user exploration: tapping a Mandelbrot point converts the tapped screen location into its complex coordinate and stores that coordinate as the Julia constant. The app exposes Julia as a fractal-family mode alongside Mandelbrot, so users can move between two related mathematical views while retaining the same palette, SSAA, contrast, 3D Look, capture, and bookmark workflows.

Mandelbrot-specific optimizations such as cardioid/bulb interior tests and perturbation remain scoped to Mandelbrot mode. Julia rendering uses the shared GPU path at normal zoom levels and can hand off to the CPU deep-render path at higher magnifications, using the saved Julia constant while retaining the same color and image-quality pipeline.

Tap-to-Julia Exploration

Version 2.0 makes the mathematical relationship between the Mandelbrot set and Julia sets interactive. When the user single-taps a point in Mandelbrot mode, the app first commits any in-progress pan or zoom gesture, records a return snapshot of the current Mandelbrot center, scale, maximum iterations, and Auto Iterations state, then converts the tapped screen location into a complex coordinate.

That coordinate becomes the Julia constant c. The renderer switches to Julia mode, recenters the Julia view around its default origin, pushes the updated viewport into the Metal/CPU render path, saves the state, and redraws. The Julia Set shown is therefore not just a preset; it is generated from the exact point the user selected in the Mandelbrot render.

Single-tapping again while viewing that Julia Set restores the saved Mandelbrot snapshot, including center, zoom scale, iteration value, and Auto Iterations state. This makes Julia exploration a reversible side trip from a Mandelbrot location rather than a destructive mode switch.

Beyond Mandelbrot and Julia, other canonical fractals, such as the Burning Ship, remain candidates for future app versions.

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.

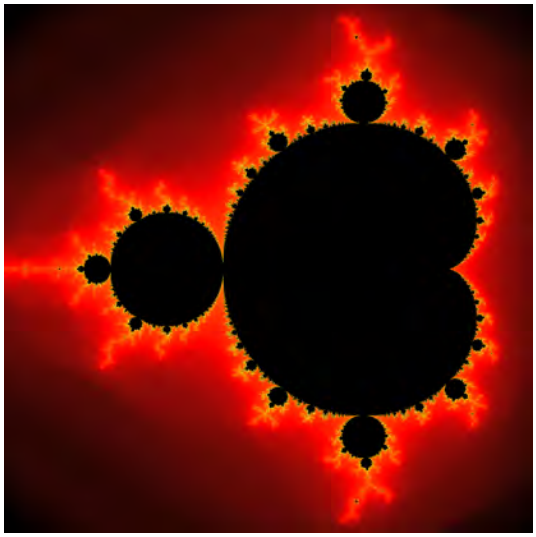


Figure 3 – Mandelbrot set as rendered by Mandelbrot Metal

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 the study of 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, where $i = \sqrt{-1}$
- both *a* and *b* are real numbers

A complex number has two parts: a **real** number (horizontal direction) and an **imaginary** number (vertical direction). You can think of it as a point on a map: one value for left/right, and one value for up/down.

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

Adding complex numbers is just moving around this map. Multiplying them rotates and scales the point, and that rotation is part of why the Mandelbrot set forms such intricate patterns when repeated.

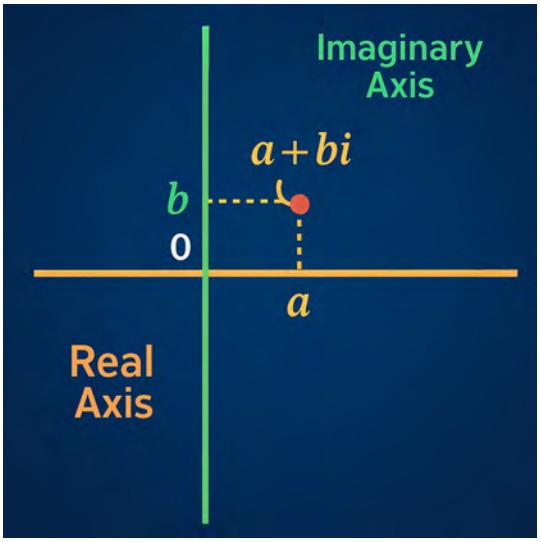


Figure 4 – The complex plane with its real and imaginary axes, showing the coordinates of the complex number

Why it matters for fractal art

The Mandelbrot set is built by repeatedly multiplying a complex number by itself and adding the original point. The two-dimensional nature of complex numbers is what gives the set its infinite depth and structure.

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 Plane to Display Screen Coordinates

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

$$Z_{n+1} = Z_n^{\,2} + C$$

$$z_0 = 0$$

…remains bounded as n approaches infinity.

If $|z_n|^2$ (where $|z_n|$ denotes the complex modulus of z_n —its distance from the origin) never exceeds 4 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 is the bailout 4? If $|z_n|^2 > 4$, then $|z_{n+1}| = |z^2 + c| \geq |z|^2 - |c|$, which grows without bound; therefore 4 is a safe escape radius. We use the square of the modulus to avoid a costly square root calculation.

If |zn|2 never exceeds a bailout radius within a maximum iteration budget, I treat c as inside the Mandelbrot set; otherwise, it is 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, the renderer evaluates the same recurrence with mode-specific initialization:

1. Mandelbrot mode: set z = 0 and use the mapped complex coordinate as c.
2. Julia mode: use the mapped complex coordinate as the starting z and use a fixed Julia constant as c.
3. Repeat $z \leftarrow z^2 + c$ up to the maximum iteration count or until $|z_n| > 4$.
4. Escaped points are colored from their smooth escape value; non-escaped points are rendered as interior for that iteration budget.

This is the escape-time algorithm shared by both fractal families.

In the tap-to-Julia workflow, the selected Mandelbrot coordinate supplies c for the Julia recurrence. The pixel coordinate then supplies z0 for the Julia render, making each tapped Mandelbrot location a seed for a full related Julia plane.

Coloring the Mandelbrot Set

Interior (the set itself)

Any points *c* that do not escape—meaning $|z_n|^2$ never exceeds the escape radius 4 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 coefficients tuned independently per device class to balance responsiveness and detail.

This scaling ensures that as zoom depth increases, the iteration budget rises 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.

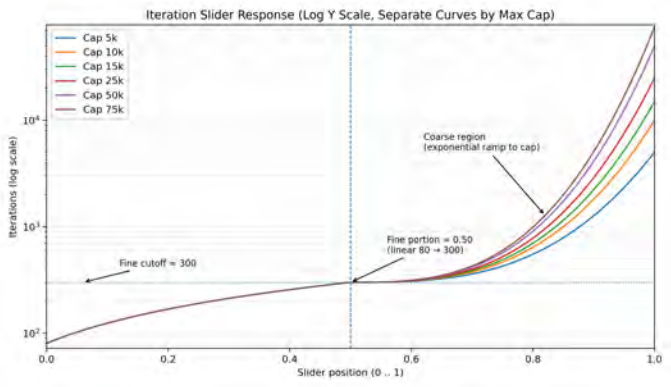


Figure 5 – The Iteration slider manual response curve is non-linear to provide precise control at low iterations.

Furthermore, advanced users can change the maximum iterations cap to 5,000 / 10,000 / 15,000 / 25,000 / 50,000 (the default) / 75,000. Lower caps provide smoother slider performance, while raising the cap enhances extreme detail (at the cost of slower renders). The cap setting persists across launches.

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 perceptual space to achieve 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.

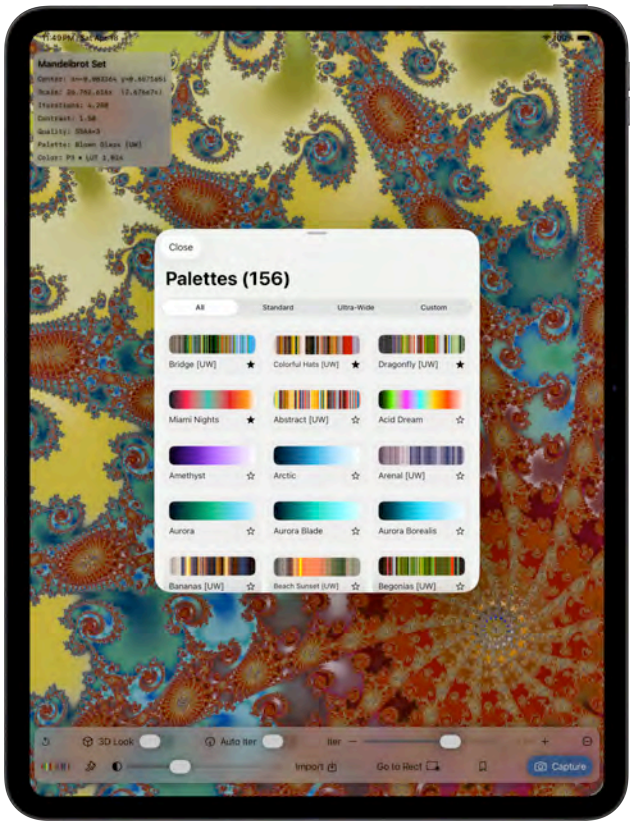


Figure 6 – The palette library in Mandelbrot Metal

Color Palettes

Mandelbrot Metal features an extensive palette library. Some of the capabilities and features of the palette system include:

- **Built-in Library** – 155+ 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 ★.
- **Ultra-Wide (768-step) palettes** – High-density 768-step ramps preserve fine detail and produce ultra-smooth gradients, especially in large prints and deep zooms. The library currently includes more than 55 Ultra-Wide palettes.
- **LUT-Based Implementation** – Palettes are represented as 1×1024 lookup textures in either sRGB or Display-P3 color space. Source palettes are defined as a small set of key color stops (or as pre-sampled ramps) and resampled into the render LUT with gamma-aware interpolation for perceptually smooth blending. IOW, gradients are interpolated in linear light, then re-encoded to sRGB or Display-P3.
- **Unified Render Resolution** – The GPU always samples a fixed 1024-tap LUT. Standard palettes are expanded into 512-step ramps and then up-sampled to 1024 taps for rendering, while Ultra-Wide palettes start from 768-step ramps and are likewise mapped into the same 1024-tap space.
- **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.

- **Free photo2palette CLI Program** – A command-line interface program designed for advanced users and programmers to create palettes in JSON format that can be imported and shared. Provides more options and control than the in-app Import Gradient feature.
- **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 using the import gradient function. Advanced users can use the more versatile CLI program, photo2palette.swift, available to all users on GitHub.

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 Mandelbrot 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
  - Source palette (512 or 768 steps) → resampled automatically into a 1×1024 LUT for the GPU
  - The shader never samples a 512- or 768-step table directly; all sampling is against the resampled 1024-tap LUT

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

In v2.0 the uniform state also includes fractalMode (0 for Mandelbrot Set, 1 for Julia Set) and juliaC, the fixed Julia constant used when Julia mode is active.

The same state is persisted through app state, bookmarks, and shared bookmark JSON. The current sharing schema records fractalMode along with juliaCX and juliaCY, so imported Julia bookmarks recreate the same fixed c constant used by the original render.

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 and fractal mode:
    - A shared Metal fractal kernel for Mandelbrot and Julia rendering during 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 orbit and escape metrics per pixel for the active fractal family:

- 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 the complex plane, then interprets that coordinate according to the active mode.

$$Z_{n+1} = Z_n^{\,2} + C$$

    2. Iterates:

    ...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 (512 steps)
     - Ultra-Wide palettes (up to 768 steps)
   - All palettes are mapped to a **1024-tap unified LUT**, so the kernel always performs the same simple lookup and interpolation.
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 Mandelbrot magnification surpasses the interactive GPU precision envelope, Mandelbrot Metal transitions from the GPU-first path to a high-precision CPU deep-render path:

- 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 updates
    - 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. In v2.0, Julia mode shares the established visual pipeline and can use the CPU deep-render path at higher magnifications; Mandelbrot perturbation remains Mandelbrot-specific.

Learn more about my render pipeline architecture in my blog post.

Super-Sampling Anti-Aliasing (SSAA)

General SSAA (unweighted box filter)

SSAA is automatically activated only when interaction stops, preventing framerate drops during navigation.

For a pixel centered at (x, y) and N subpixel 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 $\mu, \nu \in (0, \ldots, m\text{-}1)$ 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 **Γ** 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**:

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

Fractal-specific instantiation

Map each subsample 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 Mandelbrot 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 subsample runs a full escape test. Colors are averaged in registers and written once.
- Sample patterns are chosen to suppress moiré patterns. The palette lookup is done per subsample 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. The current implementation uses float for shallow views, DS (double-single style arithmetic) as precision margins tighten, and DD (double-double style arithmetic, about a 106-bit mantissa) for deeper Mandelbrot zoom stability and export-quality stills.

1. Float mode for wide views and moderate zooms, providing the fastest path.
2. DS mode is used once pixel spacing in the complex plane approaches float precision limits; DD mode is used when additional precision margin is required.
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 the 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 Mandelbrot zoom depths, Mandelbrot Metal can use perturbation rendering to maintain both speed and accuracy. Instead of recalculating every pixel independently at full precision, the renderer computes a high-precision reference orbit and evaluates per-pixel deltas (perturbations) from that orbit.

This reduces per-pixel workload and helps avoid loss of significance at deep magnification. When perturbation accuracy margins are insufficient, the renderer falls back to direct high-precision iteration

(DS/DD), for example, when the reference orbit diverges too quickly, or local gradients exceed stability thresholds.

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 remains the best example I know of emergent complexity from simple rules, and Julia sets reveal how those same rules change when the constant is fixed and the starting point varies. With Mandelbrot Metal, you do not just view pictures-you explore living mathematical landscapes at full fidelity, guided by a renderer that adapts its precision and quality to what you are 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 (e.g., or 768 steps)

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.

Photo2Palette CLI Program

Photo2palette.swift is a free command-line tool that samples colors from an image and converts them into a palette for Mandelbrot Metal, either as:

- Swift code (`registerCustom(...)`) – not intended for end-users…
  or
- A JSON file that imports directly via Mandelbrot Metal's Manage Palettes → Import function.

Photo2Palette supports horizontal or vertical sampling, configurable step counts (from 32 to 768-step Ultra-Wide palettes), and optional color adjustments.

The code repository, with documentation, is available on GitHub.

Image Capture

Mandelbrot Metal's capture system is designed to preserve the fidelity of your fractal exploration with deterministic, high-resolution output. Capture records the active fractal mode, and for Julia renders it also preserves the active c value through the saved scene state. The capture path uses the highest-precision numeric mode required for that frame.

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.
- **Precision Capture** – Capture always occurs using the highest precision numeric mode needed for that frame.
- **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 fractal mode, Julia c value when applicable, 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

Bookmark and Palette Sharing is a foundation for community-driven exploration.

Overview

Mandelbrot Metal lets users exchange their discoveries — both visual and mathematical — with other users. They 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 store all the parameters required to render exactly what's on the screen. They are exportable in JSON format. In v2.0, Manage Bookmarks supports sorting by Newest, Name, Palette, Mode, Iterations, and

Scale. Rows can show saved mode metadata, making mixed Mandelbrot and Julia collections easier to scan. Each bookmark contains:

- A unique name given by the user
- Active fractal mode (Mandelbrot Set or Julia Set)
- Julia constant c (stored as juliaCX and juliaCY for Julia renders)
- Center coordinates (real and imaginary components)
- Magnification (scale)
- Precision mode (float / DS / DD, plus deep-mode state)
- Max iteration limit (or Auto Iteration flag)
- Palette and contrast settings
- 3D Look parameters

Bookmarks are exported as small human-readable JSON documents. The sharing schema now carries fractalMode plus juliaCX and juliaCY fields so Julia bookmarks and shared bookmark files preserve the source c value while remaining backward-compatible with older bookmark data. Importing restores the saved mode, view, Julia constant, and render parameters so any user can reproduce the identical render on their device.

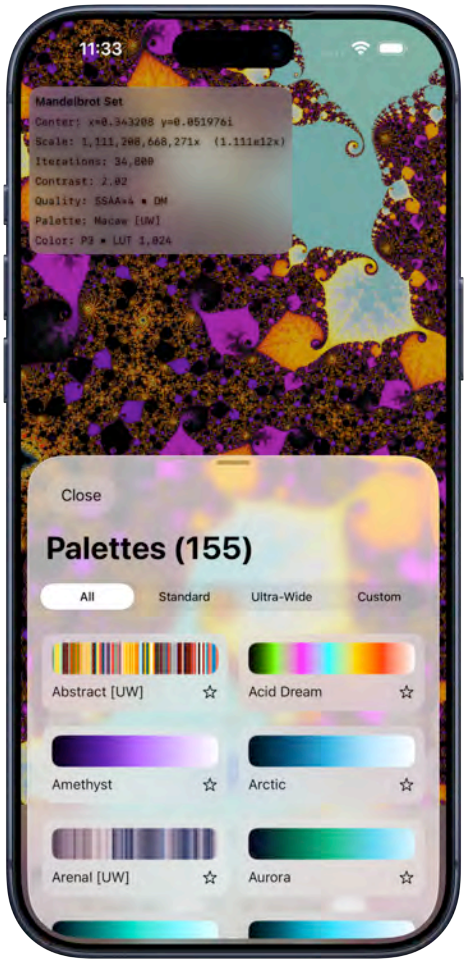


Figure 7 — Bookmark management in Mandelbrot Metal

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 the user's library with their original names and categories. Users can favorite, rename, or delete. The built-in palettes cannot be edited or removed.

Import and Export Process

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

All imports are automatically validated and deduplicated.

Community Portal (Upcoming)

On the roadmap, I plan to expand the **Mandelbrot Metal Community Portal** sharing beyond file downloads. 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 (nu-smoothing). Surface gradients are estimated per pixel using the active fractal's escape behavior, with Julia-aware finite differences on the GPU path and Mandelbrot deep-mode handling in the CPU path.

- **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 the render paths to ensure consistency even at extreme Mandelbrot zoom levels. 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 iteration data already computed for each pixel, performance overhead is modest. The effect is available across the Mandelbrot render paths and in Julia mode through the shared render pipeline.

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 lets you specify an exact rectangular domain in the complex plane and immediately reposition the viewport to it.

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 to the canvas with pixel-accurate fidelity.

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.

Some 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.

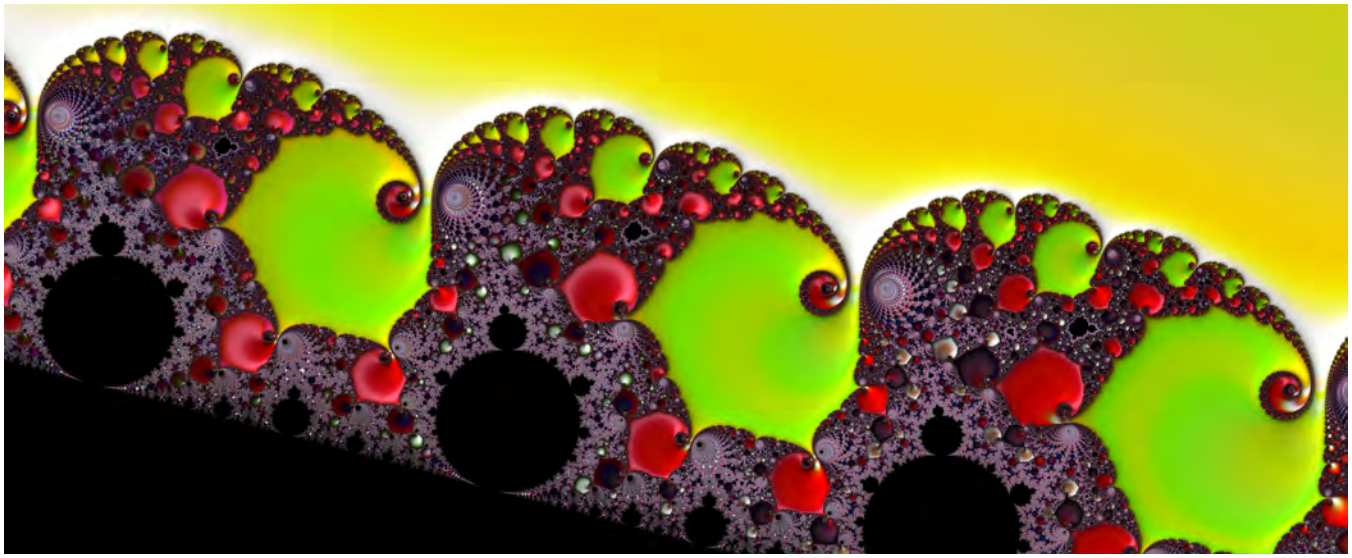


Figure 8 – A portion of the Elephant Valley rendered via Coordinate Jump

Why it matters

- **Reproducibility**: Users can record and share coordinate sets, enabling colleagues or students to 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

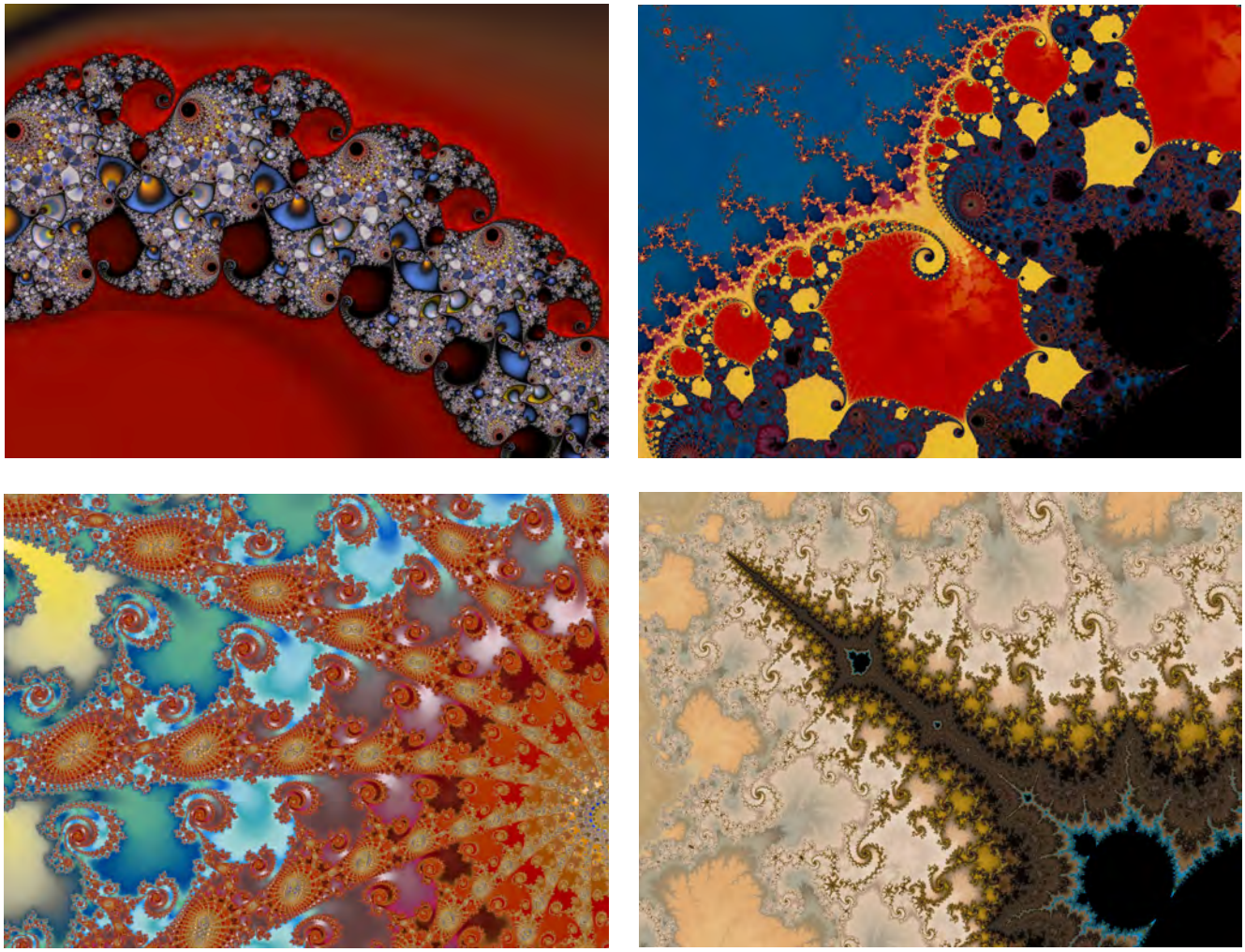


Figure 9 – Sample rendered Mandelbrot set images in Mandelbrot Metal, showing detail and smooth coloring at deep zoom levels. Each point represents a complex number c used in the Mandelbrot formula. Zoomed regions reveal repeating patterns.
For more artwork samples, visit the **Gallery**.

Stability and Reproducibility

- Arithmetic pathways are deterministic across devices. Palette LUTs are versioned. Bookmarks store:
    - Active fractal mode
    - Scale
    - Iteration budget (i.e., max iterations)
    - Contrast
    - Palette name, so scenes are reproducible.
    - 3D Look status
    - Perturbation status

    - Deep mode status
- The HUD displays:
    - Center coordinates
    - Scale
    - Iteration budget (or auto)
    - Contrast
    - Quality (SSAA, 3D, Deep Mode) statuses
    - Palette name
    - Color mode settings
    - Active Julia c value when Julia mode is selected

Benchmarks

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

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

DD is engaged automatically whenever float/DS precision margins are insufficient.

Recent Benchmarks

Device	Precision	Iterations	SSAA	Resolution	Frame rate (interactive)
iPhone 16 Plus	GPU float/DS adaptive	800	Off	2796 × 1290	~75–90 fps
iPad Pro 13” M4	GPU DS/DD adaptive	8,000	Off	2752 × 2064	~60–70 fps
iPad Pro 13” M4	GPU/CPU DD deep path	15,000	×2	2752 × 2064	~45–60 fps

Figure 10 - Deep Rendering Mode engages when Mandelbrot magnification crosses the configured deep-render threshold, 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 the center out, as results complete. This architecture maintains accuracy for extreme-precision Mandelbrot arithmetic while providing real-time, tile-by-tile feedback during the deepest zoom levels.

Version 2.0 Release Notes

The 2.0 release focuses on making Julia exploration feel native to Mandelbrot Metal rather than bolted on as a separate renderer.

- Single-tap a Mandelbrot point to create the matching Julia Set from that coordinate.
- Single-tap in Julia mode to return to the exact Mandelbrot view that created it.

- Bookmarks, app state, and shared bookmark JSON preserve both fractal mode and Julia c values.
- The HUD title follows the active fractal family and shows c in Julia mode.
- Shader hot-path cleanup reduces redundant color and bounds work without reducing samples, precision, iterations, or palette fidelity.
- Progressive CPU deep-render redraws are coalesced so tile refinement remains visible while redundant display updates are reduced.
- The ultra-wide palette catalog has been expanded and palette metadata refined for cleaner browsing.

Roadmap/Future Work

The roadmap on my website shows the features and improvements in progress and planned. With Julia sets and tap-to-Julia exploration now available in v2.0, likely future directions include a direct Julia constant editor, additional fractal families, richer community sharing workflows, and continued deep-zoom performance tuning.

About the Developer

Mandelbrot Metal is an independent software studio focused on creating breakthrough experiences at the intersection of art, science, mathematics, and high-performance computing. Founded by developer Michael Stebel, the studio pushes the limits of hardware and graphics technologies to build apps that are not only technically advanced but also visually extraordinary.

Our flagship app, Mandelbrot Metal, brings the infinite complexity of the Mandelbrot set—and other fractals—to life on iPhone and iPad. In the sixty days following its launch, the app climbed the charts to #3 in the App Store's Paid Graphics & Design category, placing it among the top ten paid creative apps on iPhone and iPad. Even more meaningful: it has remained in the top 25 of this category since launch, consistently charting alongside some of the most established creative tools in the App Store.

The app delivers real-time rendering, ultra-deep Mandelbrot zoom, one-tap Julia Set exploration, 3D effects, over 155 vibrant curated color palettes, and bookmarking and palette sharing. Mandelbrot Metal transforms abstract mathematics into an immersive visual experience - putting the beauty of infinity at your fingertips.

For more information, visit the website.

Notices

DBA

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

Trademarks

Appendix

Pseudocode for Concept Clarity

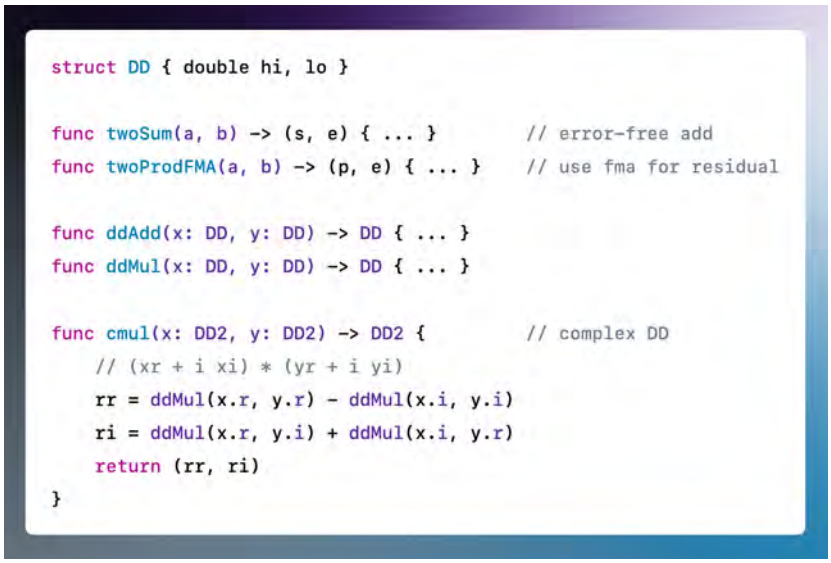
Escape + Smooth Coloring (per sample)

```
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)
```

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
```

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)
}
```

Bookmark JSON Schema

```
{
  "centerX" : 0.0022952677647057316,
  "centerY" : 0.019206639693567663,
  "contrast" : 2.8627814362769834,
  "deep" : true,
  "fractalMode" : 1,
  "iterations" : 560,
  "juliaCX" : -0.8,
  "juliaCY" : 0.156,
  "name" : "Julia 2",
  "palette" : {
    "kind" : "customByName",
    "name" : "Acid Dream"
  },
  "perturb" : false,
  "scalePixelsPerUnit" : 435.67901548436834,
  "schemaVersion" : 4,
  "threeD" : {
    "enabled" : true
  },
  "type" : "bookmark"
}
```

Palette JSON Schema

```
{
  "colorSpace" : "display-p3",
  "name" : "Descanso [C]",
  "schemaVersion" : 1,
  "stops" : [
    {
      "b" : 0.228509783744812,
      "g" : 0.14416256546974182,
      "r" : 0.11988352984189987,
      "t" : 0
    },
    {
      "b" : 0.24407154321670532,
      "g" : 0.15973743796348572,
      "r" : 0.13245081901550293,
      "t" : 0.0019569471624266144
    },
    ...
  ],
  "type" : "palette"
}
```