# Color Journey Engine: A Perceptually-Uniform Palette Generation Specification

Peter Nicholls

December 18, 2025

**Abstract**

This paper specifies the Color Journey Engine, a deterministic algorithm for generating perceptually-uniform color palettes. Built on the OKLab color space, the engine uses cubic Bézier curves to construct "journeys" through perceptual color space, producing coherent palettes from one to five anchor colors. A novel single-anchor expansion algorithm—to our knowledge not previously formalised—generates harmonious variations from a single color by exploiting lightness-weighted directions in perceptual space. The specification defines perceptual constraints based on just-noticeable difference thresholds, user-facing style controls for temperature, intensity, and smoothness, and multiple loop strategies for extended sequences. A two-layer gamut management approach ensures valid sRGB output while preserving design intent. The engine operates as a pure function with complete determinism, achieving 5.6 million colors per second in the reference implementation. This specification serves as both a formal definition and implementation guide for reproducible palette generation in user interfaces, data visualization, and generative art applications.

**Keywords:** color palette generation, OKLab, perceptual uniformity, Bézier curves, gamut mapping, deterministic algorithms

**Note on Spelling:** This specification uses American English spelling conventions ("color" not "color", "neighbor" not "neighbor", etc.) for consistency with international technical standards, API naming conventions (e.g., CSS Color Level 4, web standards), and the broader computer science and software engineering literature. This choice reflects the engine's intended use in software development contexts where American spelling is the de facto standard for programming interfaces and technical documentation.

# Contents

# Chapter 1

# Introduction and Scope

## 1.1 Motivation

Every domain that works with color—user interface design, data visualisation, generative art, interactive media—requires coherent color palettes. Despite its apparent simplicity, producing such palettes presents subtle challenges. Ad-hoc approaches—linear interpolation in RGB space or arbitrary selections from color wheels—frequently yield palettes with inconsistent perceptual spacing: some adjacent colors appear nearly identical while others jump dramatically.

The fundamental problem is that commonly-used color spaces (sRGB, HSL, HSV) are not *perceptually uniform*. Equal numerical steps do not correspond to equal perceived differences. A mathematically elegant gradient in RGB may appear uneven to the human eye, with certain hue regions changing faster than others.

This specification addresses the need for a *principled, reproducible* approach to palette generation—one grounded in perceptual color science, where mathematical operations correspond meaningfully to visual experience.

### Audience and Purpose

This document serves as an internal technical reference for the engineering team developing and maintaining the C-core Color Journey Engine implementation. It documents design rationale, mathematical foundations, and architectural decisions to enable team alignment—ensuring we can say "see §3.3" and be immediately on the same page.

The paper prioritises the "why" over the "how": implementation details are covered in separate technical documentation, while this specification captures the reasoning behind design choices, the perceptual science foundations, and the constraints that shape the system.

## 1.2 Scope Definition

The Color Journey Engine produces **ordered sequences of discrete color swatches** by constructing perceptually-aware paths through color space. Given one to five anchor colors and a set of style parameters, the engine generates $N$ swatches such that adjacent colors are perceptually distinguishable yet smoothly related.

### What the Engine Does

- Generates deterministic color palettes using perceptual color science
- Outputs discrete swatch arrays (not continuous functions)
- Enforces perceptual constraints ensuring distinguishability
- Provides diagnostic information about palette quality

### What the Engine Does Not Do

The engine deliberately excludes capabilities that belong to callers or separate systems:

- **Accessibility checking**—WCAG contrast verification is caller responsibility
- **Animation control**—temporal playback logic belongs to the application
- **Color naming**—a separate knowledge domain
- **Color management**—ICC profiles and device calibration are external concerns
- **Continuous curve evaluation**—the curve is an internal construction device, not an exposed API

## 1.3 Design Principles

Five core principles guide the engine's design:

**Perceptual Uniformity as Foundation.** All design decisions rest on the perceptual uniformity of OKLab color space Ottosson, 2020. Mathematical distances correspond to perceived differences, enabling meaningful constraints and smooth transitions.

**Discrete Output, Continuous Thinking.** The engine constructs continuous paths through color space internally but outputs discrete swatches. The "journey" is a construction metaphor—users receive a finite list of colors, not an interpolation function.

**Constructive, Not Prescriptive.** Colors emerge *as if* travelling a perceptual path, providing coherence and natural ordering. Users may apply the resulting swatches in any manner; the ordering is a construction device, not a usage constraint.

**Presets Encode Expertise.** Presets capture expert knowledge about which parameter combinations achieve specific aesthetic goals—they are not simplifications for novices. A "Cinematic" preset encodes design wisdom about dramatic color arcs.

**Graceful Degradation.** Rather than failing on unusual inputs, the engine adapts. Grey anchors expand along the lightness axis; extreme colors adjust their expansion direction; pathological inputs produce documented fallback behavior.

## 1.4 Positioning and Novel Contributions

### Prior Art: Three Paradigms of Palette Generation

Palette generation approaches fall into three broad paradigms, each with distinct strengths and limitations.

**Geometric Harmony Rules.** Traditional *color harmony rules*—complementary, triadic, analogous, split-complementary—define geometric relationships on the color wheel (Itten, 1961). These rules have aesthetic merit and deep historical roots in art education, but they operate in perceptually non-uniform spaces (typically HSL or HSV) and provide no guarantees about perceived differences between generated colors. Two colors deemed "complementary" by wheel position may or may not be perceptually distinguishable depending on their lightness and chroma values.

**Interpolation-Based Methods.** *Interpolation-based approaches* generate colors by blending between endpoints. Most creative tools (CSS gradients, design software) use linear interpolation in RGB or HSL space, producing results that can appear perceptually uneven—muddy midpoints, abrupt hue shifts through grey regions, and inconsistent step sizes (Stone, Szafir, and Setlur, 2014). Even when tools support perceptual spaces, they typically output continuous gradients rather than discrete palettes with distinguishability guarantees (Madsen, 2017).

**Data-Driven Methods.** More recent academic work has explored *data-driven approaches* that learn palette patterns from images or existing designs. Liu et al. (Liu

et al., 2013) demonstrate image-driven harmonious color extraction using saliency-hue clustering with harmony optimisation. Such methods excel at capturing aesthetic patterns from existing content but require training data, produce less predictable outputs, and are difficult to parameterise for specific design goals. The Color Journey Engine takes a deliberately different path: deterministic construction with explicit, interpretable parameters.

### The Shift to Perceptual Color Spaces

Modern tools increasingly adopt perceptually uniform color spaces to address the limitations of geometric and RGB-based methods. CIELAB CIE, 1976 was the first widely-used perceptual space but exhibits known uniformity issues, particularly in blue-violet regions Mahy, Van Eycken, and Oosterlinck, 1994. CIEDE2000 Luo, Cui, and Rigg, 2001 improved color difference calculation but remains computationally complex.

OKLab, introduced by Björn Ottosson in 2020 Ottosson, 2020, represents a significant advance: CAM16-level perceptual uniformity Safdar et al., 2017 with computational efficiency suitable for real-time applications. Independent analysis has validated OKLab's perceptual uniformity claims for gradient and color manipulation applications Levien, 2021. OKLab has achieved rapid industry adoption, appearing in CSS Color Level 4 W3C CSS Working Group, 2022, Adobe Photoshop (as the default gradient interpolation space), Unity, and Godot. The Color Journey Engine builds on this established foundation.

### Our Contribution

The Color Journey Engine introduces a *novel combination* of established techniques:

- **Continuous Bézier paths** through OKLab space Farin, 2002, providing flexible curve shapes with well-understood mathematical properties
- **Arc-length parameterisation** Piegl and Tiller, 1997; Kamermans, 2023 for perceptually uniform sampling along the constructed path
- **Perceptual constraints** ($\Delta_{\text{min}}$, $\Delta_{\text{max}}$) ensuring distinguishability without jarring jumps, grounded in just-noticeable difference research
- **Single-anchor mood expansion**—to our knowledge, a novel approach—using lightness-direction heuristics to generate coherent palettes from one color
- **Two-layer gamut management** Morovič, 2008 preventing and correcting out-of-gamut colors while preserving hue, aligned with CSS Color Level 4 practices Verou

and Lilley, 2024

Our contribution lies not in the individual techniques—OKLab, Bézier curves, arc-length sampling, and gamut mapping are established—but in their integration into a coherent system with explicit perceptual constraints and a deterministic, reproducible API. Unlike data-driven methods that require training corpora, the Color Journey Engine produces predictable, parameterisable results. Unlike simple interpolation, it guarantees perceptual distinguishability. Unlike geometric harmony rules, it operates in perceptually uniform space with explicit distance constraints.

The resulting implementation achieves 5.6 million colors per second, enabling real-time palette generation for interactive applications.

## 1.5 Paper Organization

This specification proceeds as follows:

- **§2.1–2.4:** Perceptual color foundations in OKLab
- **§3.1–3.6:** Journey construction—anchors, Bézier curves, and sampling
- **§4.1–4.5:** Perceptual constraint framework
- **§5.1–6.4:** Style controls and modes of operation
- **§7.1–8.4:** Loop strategies and gamut management
- **§9.1–11.4:** Determinism, API design, and caller responsibilities
- **§12.1–12.3:** Summary and future directions

Appendices provide preset reference tables, mathematical notation, implementation examples, and a quick-reference guide.

# Chapter 2

# Perceptual Color Foundations

## 2.1 The Perceptual Uniformity Requirement

Human vision perceives color differences nonlinearly. A linear blend in RGB space does not produce a linear perceptual transition—interpolating between saturated blue and yellow often produces washed-out greys or unexpected hues at the midpoint, because the intermediate colors pass through an unsaturated region of color space.

Similarly, equal numerical changes in HSV or HSL frequently produce uneven perceptual steps. Some hue regions appear to change faster than others; lightness and saturation interact in ways the coordinate system does not capture.

To address this fundamental mismatch, scientists have developed *perceptually uniform* color spaces where Euclidean distances correspond more closely to perceived color differences CIE, 1976; Fairchild, 2013. In such spaces, moving a given distance in any direction produces a change that observers judge to be of similar magnitude.

The engine *must* work in a perceptually uniform space so that equal steps along the journey correspond to equal perceived differences. This is non-negotiable for achieving smooth, natural color transitions.

## 2.2 OKLab Color Space

OKLab, introduced by Björn Ottosson in 2020 Ottosson, 2020, is a modern perceptual color space designed to predict human color perception accurately while remaining computationally efficient. It has gained rapid industry adoption, appearing in CSS Color Level 4 W3C CSS Working Group, 2022 and various creative tools. Independent analysis has validated its perceptual uniformity claims, particularly for gradient interpolation and color manipulation tasks (Levien, 2021).

Compared to alternatives (Mahy, Van Eycken, and Oosterlinck, 1994; Safdar et al., 2017), OKLab offers an excellent balance. The following comparison synthesises Ottosson's original analysis (Ottosson, 2020) with established color science literature (Fairchild, 2013):

| Color Space | Uniformity | Cost | Stability | Selected |
|---|---|---|---|---|
| sRGB | Poor | Low | Good | No |
| HSV/HSL | Poor | Low | Poor | No |
| CIELAB (1976) | Moderate | Moderate | Poor at extremes | No |
| CAM16-UCS | Excellent | High | Good | No |
| **OKLab** | **Excellent** | **Low** | **Excellent** | **Yes** |

*Uniformity* refers to how closely Euclidean distances match perceived differences; *Cost* measures computational complexity (matrix operations vs. iterative solving); *Stability* indicates behavior at gamut boundaries and extreme lightness values. OKLab achieves CAM16-level uniformity (Safdar et al., 2017) with CIELAB-level computational cost—a key factor for real-time applications.

A color in OKLab is represented by three coordinates $(L, a, b)$:

$L$ — **Perceived lightness** ranges from 0 (perceptual black) to 1 (brightest white). Equal differences in $L$ correspond to equal perceived brightness changes.

$a$ — **Green–red opponent axis** with positive values indicating reddish tints and negative values indicating greenish tints. Zero is achromatic along this axis.

$b$ — **Blue–yellow opponent axis** with positive values indicating yellowish tints and negative values indicating bluish tints. Zero is achromatic along this axis.

The $L$, $a$, $b$ axes are designed to be *approximately orthogonal in perception.* Altering $L$ alone should not significantly change perceived hue or chroma; altering $a$ or $b$ primarily affects hue and saturation without changing perceived lightness.

The perceptual distance between two colors is computed as Euclidean distance:

$$\Delta E = \sqrt{(L_2 - L_1)^2 + (a_2 - a_1)^2 + (b_2 - b_1)^2} \tag{2.1}$$

where $\Delta E$ denotes color difference in OKLab perceptual space.[1]

[1] All $\Delta E$ measurements in this specification refer to color difference calculated in OKLab space unless explicitly noted otherwise.

Ottosson designed OKLab such that a $\Delta E$ of approximately 1.0 corresponds to a just-noticeable difference (JND) for an average observer under standard viewing conditions (Ottosson, 2020). This design goal aligns with the broader color science literature on JND thresholds (Luo, Cui, and Rigg, 2001; Fairchild, 2013), though direct perceptual validation studies specific to OKLab remain limited. The 1.0 threshold is a design target rather than an empirically-validated constant across all viewing conditions.

## 2.3 Cartesian vs. Cylindrical Coordinates

For certain operations—particularly hue manipulation—a cylindrical form is more convenient. OKLCh represents the same colors using:

$L$ — **Lightness** identical to OKLab

$C$ — **Chroma** defined as $C = \sqrt{a^2 + b^2}$, representing distance from the neutral axis (colorfulness/saturation)

$h$ — **Hue angle** defined as $h = \text{atan2}(b, a)$, the angle around the $a$–$b$ plane

The conversion is straightforward:

$$C = \sqrt{a^2 + b^2} \tag{2.2}$$

$$h = \text{atan2}(b, a) \tag{2.3}$$

The engine uses both representations internally:

- **Cartesian (OKLab)** for distance calculations and Bézier interpolation, where linear operations are meaningful
- **Cylindrical (OKLCh)** for hue-related manipulations such as warmth bias and complementary color calculations

This separation allows clean reasoning about lightness dynamics (along $L$) versus chromatic dynamics (in the $a$–$b$ plane or around $h$).

## 2.4 Color Space Conversion

The transformation from sRGB to OKLab follows a well-defined pipeline:

1. **Linearise sRGB** — Remove gamma encoding to obtain linear RGB values

2. **RGB to XYZ** — Apply the standard sRGB-to-XYZ matrix
3. **XYZ to LMS** — Apply matrix $M_1$ to approximate cone responses
4. **Nonlinear compression** — Apply cube root: $(l', m', s') = (l^{1/3}, m^{1/3}, s^{1/3})$
5. **LMS to Lab** — Apply matrix $M_2$ to obtain final $(L, a, b)$

The transformation matrices are:

$$M_1 = \begin{pmatrix} 0.8189330101 & 0.3618667424 & -0.1288597137 \\ 0.0329845436 & 0.9293118715 & 0.0361456387 \\ 0.0482003018 & 0.2643662691 & 0.6338517070 \end{pmatrix} \tag{2.4}$$

$$M_2 = \begin{pmatrix} 0.2104542553 & 0.7936177850 & -0.0040720468 \\ 1.9779984951 & -2.4285922050 & 0.4505937099 \\ 0.0259040371 & 0.7827717662 & -0.8086757660 \end{pmatrix} \tag{2.5}$$

Each step models aspects of human vision: $M_1$ approximates L, M, S cone responses; the cube root models nonlinear perceptual compression; $M_2$ extracts lightness as a weighted average and opponent-color signals as cone differences.

### Computational Efficiency

The OKLab conversion is computationally trivial:

- **Per conversion:** approximately 50–100 CPU cycles
- **Operations:** 18 multiplications, 12 additions, 3 cube roots
- **Memory:** no allocations; all register operations
- **Parallelisation:** trivially vectorisable with SIMD

This efficiency enables real-time palette generation. The entire forward/inverse pipeline uses fixed, known constant matrices—no iterative solving, no conditionals, no special cases.

### Implications for the Engine

The choice of OKLab has cascading implications:

1. **Linear interpolation works** — A straight line between two colors in OKLab produces a perceptually smooth gradient

2. **Distance calculations are meaningful** — Constraints like $\Delta_{\min}$ and $\Delta_{\max}$ correspond to actual perceived differences

3. **Dynamics are separable** — The orthogonality of $L$, $a$, $b$ means lightness can be adjusted independently of chromaticity

4. **Hue interpolation is stable** — Unlike HSV, where interpolating across the 0°/360° boundary causes problems, OKLab's Cartesian representation avoids discontinuities

Because OKLab is unbounded—it can represent colors outside any physical display gamut—the engine must include explicit gamut mapping (§8.1).

# Chapter 3

# Journey Construction

## 3.1 The Journey Metaphor

The term "journey" was deliberately chosen over "gradient" or "palette" because it emphasises the *constructive process*—a path through color space—rather than the static result. The journey metaphor captures several key ideas:

**Directionality** There is a start and end (or a cycle)

**Waypoints** Anchors are places the journey must pass through

**Path flexibility** The route between waypoints can vary

**Ordering** Colors are sequenced with a sense of "before" and "after"

Critically, the journey is *not* a continuous function that callers sample arbitrarily. The engine outputs discrete swatches only. The continuous curve is an internal construction mechanism; callers receive $N$ specific colors, not an interpolation function.

This distinction matters: callers should not expect to query "color at $t = 0.347$". The engine can optimize internal representation freely, and behaviors like ping-pong playback become caller responsibility using the discrete output array.

## 3.2 Anchor Colors

**Anchors** are user-specified key colors that define the skeleton of the journey. They are guaranteed to appear in the output and serve as fixed boundary conditions for path construction.

| Property | Specification |
|---|---|
| Minimum anchors | 1 |
| Maximum anchors | 5 |
| Input format | Hex, RGB, or OKLab |
| Internal representation | OKLab coordinates |
| Output guarantee | Each anchor appears exactly in output |

The 5-anchor maximum is a deliberate design constraint informed by cognitive limitations and practical experience. Research on working memory suggests humans can reliably track 4±1 items simultaneously (Cowan, 2001); beyond this, the perceptual "story" of a color journey fragments into disconnected segments. Additionally, cubic Bézier curves between consecutive anchors provide $C^1$ continuity guarantees (Farin, 2002), but as anchor count increases, maintaining smooth visual flow becomes increasingly difficult. This is a *design choice* balancing expressiveness against cognitive and computational complexity, not a technical limitation.

Anchors are processed in the order provided: $A_1 \to A_2 \to \ldots \to A_m$. The journey visits each anchor in sequence; for closed loops, it then returns to $A_1$.

## 3.3 Mood Expansion (Single-Anchor Case)

When only one anchor is provided, the engine cannot construct a traditional transition between colors. Instead, it performs **mood expansion**—creating a set of harmonious colors centred on the single anchor.

*To our knowledge, this single-anchor expansion approach represents a novel contribution.* Traditional palette generators require at least two colors for interpolation; color harmony systems (complementary, triadic, analogous) generate geometrically-related hues but not perceptually-graded variations. Mood expansion bridges this gap by using the anchor's inherent character to determine expansion direction.

**Design Decision: Mood Expansion Direction**

**Choice:** Single-anchor palettes expand along lightness-weighted directions in OKLab, using the anchor's inherent character (hue, chroma, lightness) to determine expansion.

**Rationale:** Expanding in perceptually-coherent directions preserves the anchor's "mood" or character. Grey anchors naturally expand along lightness (the only meaningful axis available); chromatic anchors expand toward complementary or analogous regions based on their position in color space.

**Alternatives Considered:**

- *Naive hue spin* — Rejected: produces arbitrary results disconnected from anchor character
- *Fixed expansion vectors* — Rejected: ignores anchor properties, loses mood coherence
- *Random sampling in OKLab* — Rejected: no perceptual relationship to anchor

**Reference:** §3.2, §3.5

**Example:** A dark navy anchor ($L = 0.25$, $h \approx 250$ř) expands primarily along the lightness axis toward lighter blues and teals, preserving the anchor's cool character. A bright orange anchor ($L = 0.75$, $C = 0.15$) expands toward darker oranges and adjacent warm hues. The expansion direction emerges from the anchor's inherent perceptual properties, not arbitrary geometric rules.

With one anchor $A_1$, the engine:

1. Uses the anchor as the palette's conceptual centre
2. Generates variations along multiple axes:
   - Lighter and darker variants (along $L$ axis)
   - Slight hue rotations (around $h$ in OKLCh)
   - Chroma variations (along $C$ in OKLCh)
3. Forms a closed loop returning to the anchor

Single-anchor journeys are useful for generating monochromatic palettes with depth, creating UI state variations (hover, active, disabled) from a brand color, and animating "breathing" or "pulsing" color effects.

## 3.4 Multi-Anchor Paths

For $m$ anchors ($m \geq 2$), the journey is composed of $m - 1$ segments:

$$\gamma_0 : A_1 \to A_2 \tag{3.1}$$
$$\gamma_1 : A_2 \to A_3 \tag{3.2}$$
$$\vdots \tag{3.3}$$
$$\gamma_{m-2} : A_{m-1} \to A_m \tag{3.4}$$

Each segment is constructed independently but with continuity constraints at junction points.

**$C^0$ Continuity (Position).** The end of segment $\gamma_i$ equals the start of segment $\gamma_{i+1}$. This is automatically satisfied since both meet at anchor $A_{i+1}$.

**$C^1$ Continuity (Tangent).** For smooth transitions, the engine matches the direction of approach and departure at each anchor. This prevents "corners" in the color path.

By default, the engine applies smoothing at anchor junctions to achieve $C^1$ continuity, making the journey feel fluid rather than segmented. If sharp transitions are desired, this can be adjusted via the smoothness parameter (§5.3).

## 3.5 Bézier Curve Construction

Each segment is represented as a **cubic Bézier curve** in OKLab space Farin, 2002, providing flexibility, mathematical elegance, continuity control, and computational efficiency.

A cubic Bézier curve with endpoints $P_0$, $P_3$ and control points $P_1$, $P_2$:

$$\gamma(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3 \tag{3.5}$$

For $0 \leq t \leq 1$: $\gamma(0) = P_0$ (start anchor) and $\gamma(1) = P_3$ (end anchor).

The tangent vectors at endpoints are:

$$\gamma'(0) = 3(P_1 - P_0) \quad \text{(direction leaving start)} \tag{3.6}$$
$$\gamma'(1) = 3(P_3 - P_2) \quad \text{(direction arriving at end)} \tag{3.7}$$

### Control Point Placement

By default, control points are placed on the line between anchors, making the Bézier equivalent to linear interpolation. Style parameters then "nudge" these control points to shape the curve:

- **Intensity** scales control point distance from the anchor line, adding curve drama
- **Temperature** biases the offset direction toward warm or cool hues
- **Smoothness** affects how aggressively tangents are matched at junctions

With default parameters, the engine produces simple linear interpolation—the most predictable baseline.

### Ensuring $C^1$ Continuity

To match tangents at anchor $A_{i+1}$, the control point $Q_1$ of the next segment is placed such that:

$$Q_1 = A_{i+1} + (A_{i+1} - P_2) \tag{3.8}$$

This mirrors $P_2$ across the anchor, ensuring the outgoing direction matches the incoming direction.

## 3.6 Arc-Length Parameterisation

The Bézier parameter $t$ does not correspond to distance along the curve (Piegl and Tiller, 1997; Kamermans, 2023). Equal increments in $t$ produce unequal distances in OKLab space, leading to uneven perceptual steps—swatches cluster in curved regions where the curve bends sharply and spread apart in straighter sections.

This is a well-known property of parametric curves: the parameter $t$ describes progress along the curve's mathematical definition, not progress along its physical length. For perceptually uniform palette generation, we require *arc-length parameterisation*—sampling at equal distances along the actual path through color space.

### The Arc-Length Solution

The engine estimates total arc length and samples at equal arc-length intervals (Levien, 2018):

1. **Estimate total arc length** — Numerically integrate $|\gamma'(t)|$ over $[0, 1]$
2. **Distribute sample points** — Place $N$ samples at equal arc-length intervals
3. **Map back to** $t$ — Find the $t$ value corresponding to each arc-length position

Arc-length parameterisation ensures that output swatches are perceptually equidistant (in terms of path distance in OKLab), not just parametrically equidistant. Combined with OKLab's perceptual uniformity, this produces swatches with consistent perceived differences between adjacent colors.

### Practical Implementation

For efficiency, the engine uses numerical approximation following established techniques (Kamermans, 2023):

- Subdivide curve into many small segments (adaptive subdivision based on curvature)
- Sum Euclidean distances of segments to approximate total arc length
- Use binary search or Newton iteration to find $t$ for target arc lengths
- Cache arc-length tables for repeated sampling of the same curve

The approximation error is bounded by the subdivision granularity. In practice, 100–200 subdivisions per segment provide sufficient accuracy for palette generation while maintaining computational efficiency.

### Anchor Preservation

Anchors always appear exactly in the output sequence—they are not approximated or averaged. If the user provides anchors A, B, C and requests 7 swatches, the output includes A at position 0, C at position 6, and B at approximately the middle, with interpolated colors filling the gaps.

# Chapter 4

# Perceptual Constraints

## 4.1 Just-Noticeable Difference

The **Just Noticeable Difference (JND)** is the smallest color change that an average human observer can detect under standard viewing conditions. This concept originates from Weber's and Fechner's foundational psychophysics work and has been extensively studied in color science (Fairchild, 2013).

In OKLab:

- Theoretical JND threshold: $\Delta E \approx 1.0$ unit (OKLab's design target (Ottosson, 2020))
- Practical threshold: $\Delta E \approx 2.0$ units (accounting for real-world variability)

The practical threshold of 2.0 units is derived from applying a safety margin to the theoretical JND. Color science literature consistently shows that laboratory JND values (measured under controlled illumination, neutral adaptation, and foveal viewing) underestimate the threshold needed for reliable discrimination in real applications (Fairchild, 2013; Luo, Cui, and Rigg, 2001). Factors including ambient lighting variation, observer differences, display calibration, and peripheral viewing all increase the effective threshold. The 2× multiplier is a conservative engineering choice, not an empirically-validated constant.

Notably, the CSS Color Level 4 gamut mapping algorithm uses a similar threshold concept: colors are considered "close enough" to clip when $\Delta E < 0.02$ on OKLab's 0–1 scale (W3C CSS Working Group, 2022), which corresponds to approximately 2 units on the 0–100 scale commonly used in color difference literature. Our threshold choice aligns with this industry practice.

| $\Delta E$ Range | Perceptual Interpretation |
|---|---|
| $< 1.0$ | Imperceptible to most observers |
| $1.0 - 2.0$ | Barely perceptible, subtle |
| $2.0 - 3.0$ | Noticeable, clear difference |
| $3.0 - 5.0$ | Obvious difference, still smooth |
| $> 5.0$ | Pronounced difference, may feel like a "step" |

*Note: This table represents design guidance derived from Fairchild Fairchild, 2013 and practical experience, not a universal standard. Thresholds vary significantly by application context.*

## 4.2 $\Delta_{\text{min}}$ Constraint (Distinguishability)

$$\Delta_{\text{min}} \approx 2.0 \tag{4.1}$$

This is the **minimum allowed perceptual distance** between any two adjacent swatches.

**Why 2.0?** This value is set at the practical JND threshold (§4.1) to ensure that every color step is reliably perceptible across typical viewing conditions. Setting $\Delta_{\text{min}}$ at 1.0 (the theoretical JND) would produce steps that are only *sometimes* distinguishable; the 2.0 threshold provides margin for the variability inherent in real-world color perception (Fairchild, 2013). There is no value in generating colors that viewers cannot reliably distinguish—it wastes palette capacity and creates the impression of redundancy.

### Enforcement

If two anchors are closer than $\Delta_{\text{min}}$:

- **Collapse case:** The engine may treat them as essentially the same anchor, producing no intermediate swatches
- **Skip intermediates:** No interpolation is performed; the journey jumps directly from one to the other

When anchors are extremely close ($\Delta < \Delta_{\text{min}}$), the engine does not attempt to create meaningless intermediate colors.

## 4.3 $\Delta_{\max}$ Constraint (Coherence)

$$\Delta_{\max} \approx 5.0 \tag{4.2}$$

This is the **maximum allowed perceptual distance** between any two adjacent swatches.

**Why 5.0?** This value represents the upper bound of "comfortable" color transitions—steps large enough to be clearly distinct but not so large as to feel discontinuous. The 5.0 threshold emerges from the JND interpretation table (§4.1): differences above ∼5 units cross from "obvious but smooth" into "pronounced step" territory. This is a design heuristic informed by practical experimentation with generated palettes, not a value derived from formal perceptual studies. Alternative implementations may choose different thresholds based on their aesthetic goals.

### Enforcement

If a segment between anchors has total distance $D > \Delta_{\max}$:

1. **Subdivide:** The segment is divided into $n$ sub-segments where $n = \lceil D/\Delta_{\max} \rceil$
2. **Insert intermediates:** $n - 1$ intermediate swatches are inserted
3. **Cap subdivisions:** $n \leq 5$ to prevent excessive palette length

**Why cap at 5?** This limit prevents runaway palette growth when anchors are extremely far apart in color space. The value 5 is chosen to align with the anchor count limit (§3.2)—maintaining cognitive tractability. With at most 5 intermediates between any two anchors, users can reason about the palette structure. If anchors are extremely far apart ($D > 25$ units), each step will exceed $\Delta_{\max}$, which is acceptable: the user has explicitly requested a large color jump by their anchor selection.

## 4.4 Adaptive Sampling

Given anchors $A$ and $B$ with perceptual distance $D$, the adaptive sampling algorithm:

1. If $D < \Delta_{\min}$: append only $B$ to output (skip intermediates)
2. Otherwise: compute $n = \min(5, \lceil D/\Delta_{\max} \rceil)$
3. For $i = 1$ to $n$: interpolate at $t = i/n$ and append to output

Each segment has length $D/n$, guaranteeing:

- Each step $\leq \Delta_{\max}$
- Each step $\geq \Delta_{\min}$ (approximately, given the constraints)

| **Total Distance** $D$ | **Segments** $n$ | **Step Size** $D/n$ |
|---|---|---|
| 3.0 | 1 | 3.0 |
| 7.0 | 2 | 3.5 |
| 12.0 | 3 | 4.0 |
| 20.0 | 4 | 5.0 |
| 30.0 | 5 (capped) | 6.0 |

When $D$ is very large and capped at $n = 5$, step sizes may exceed $\Delta_{\max}$. This is a trade-off for palette manageability.

## 4.5 Constraint Enforcement

Some anchor configurations make constraints impossible to satisfy fully—two anchors very close together cannot yield many distinguishable steps; requesting 100 swatches between nearby colors is unsatisfiable.

The engine prioritises producing output over strict enforcement:

1. **Best effort:** The engine produces the closest valid palette
2. **Diagnostics:** Constraint violations are reported (not thrown as errors)
3. $\Delta_{\min}$ **priority:** Distinguishability takes precedence—colors below JND are never output

Distance constraints are evaluated on the actual path (after dynamics and gamut mapping are applied), not the straight-line distance between anchors. If dynamics increase path length, more intermediate swatches may be needed; if gamut clipping shortens effective distances, fewer may suffice.

### The Perceptual Comfort Zone

The ideal step size falls in the range $2.0 \leq \Delta E \leq 5.0$—steps are clearly distinguishable but not jarring. This is the "sweet spot" where palettes feel both coherent and varied.

# Chapter 5

# User-Facing Style Controls

## 5.1 Temperature Control

Temperature biases the journey's hue path toward warm or cool colors (Hunt, 2004). This control operates in the hue dimension of OKLCh (§2.3), biasing the path selection when multiple hue routes are possible.

| | |
|---|---|
| **Range** | $-1.0$ (cool) to $+1.0$ (warm), default 0.0 |
| **Effect** | Shifts interpolation path through warm or cool hue regions |

When interpolating hue in OKLCh, there are always two possible paths around the color wheel. By default, the engine takes the shortest path. The temperature parameter overrides this:

- **Positive values** ($+0.1$ to $+1.0$): Bias toward warm hues (reds, oranges, yellows)
- **Negative values** ($-0.1$ to $-1.0$): Bias toward cool hues (blues, cyans, greens)
- **Zero**: Take the shortest hue path (default)

The same anchor pair can produce vastly different palettes: purple-to-green with temperature 0 passes through blue (cool, oceanic), while temperature $+1$ passes through red and yellow (warm, sunset-like).

Temperature is implemented via direction blending:

$$\vec{v} = (1 - \alpha|T|) \cdot \hat{v}_{\text{base}} + \alpha|T| \cdot \hat{v}_{\text{warm/cool}} \tag{5.1}$$

where $\hat{v}_{\text{base}}$ is the shortest-path direction, $\hat{v}_{\text{warm/cool}}$ is the target hue direction, and $\alpha \in (0, 1)$ is a blending coefficient (typically $\alpha = 0.8$) that ensures partial preservation of the base direction, preventing jarring snaps to pure warm or cool paths.

## 5.2 Intensity Control

Intensity controls the chromatic drama of the journey—how far control points deviate from straight-line interpolation.

| | |
|---|---|
| **Range** | 0.0 (minimal) to 1.0 (maximum), default 0.5 |
| **Effect** | Scales Bézier control point offset from the anchor line |

- **Low intensity** (0.0–0.3): Subtle, nearly linear paths; control points close to the anchor line
- **Medium intensity** (0.4–0.6): Moderate curves with balanced chromatic excursions
- **High intensity** (0.7–1.0): Dramatic arcs; pronounced curves through color space

Mathematically, intensity scales the control point offset:

$$P_1 = \text{lerp}(P_0, P_3, \tfrac{1}{3}) + \iota \cdot \vec{s}_1 \tag{5.2}$$

$$P_2 = \text{lerp}(P_0, P_3, \tfrac{2}{3}) + \iota \cdot \vec{s}_2 \tag{5.3}$$

where $\iota$ is the intensity parameter and $\vec{s}_1$, $\vec{s}_2$ are style-determined offset vectors (typically into higher-chroma regions).

Intensity affects path geometry, not endpoint colors. The anchors remain unchanged (§3.2); only the route between them becomes more or less dramatic. This is achieved by scaling the Bézier control point offset (§3.5).

## 5.3 Smoothness Control

Smoothness controls the strength of $C^1$ continuity enforcement at anchor junctions.

| | |
|---|---|
| **Range** | 0.0 (sharp) to 1.0 (smooth), default 0.7 |
| **Effect** | Controls transition character at anchor points |

- **High smoothness** (0.8–1.0): Full tangent matching; the journey flows smoothly through anchors
- **Medium smoothness** (0.4–0.7): Partial tangent matching; gentle rounding at junctions
- **Low smoothness** (0.0–0.3): Minimal smoothing; sharper transitions at anchors

High smoothness is appropriate for gradients and animations where flow matters. Low smoothness may be preferred when anchors represent distinct states and clear transitions are desired.

Smoothness also affects perceived velocity consistency—higher values produce more uniform perceptual change rates along the journey.

## 5.4 Secondary Parameters

Beyond the primary triad (temperature, intensity, smoothness), several secondary parameters provide finer control:

**Lightness Bias.** Range $-1.0$ to $+1.0$, default 0. Shifts all colors lighter (positive) or darker (negative). Applied proportionally to avoid clipping:

$$L' = \begin{cases} L + \lambda(1 - L) & \text{if } \lambda > 0 \\ L + \lambda \cdot L & \text{if } \lambda < 0 \end{cases} \tag{5.4}$$

**Chroma.** Range 0.0 to 2.0, default 1.0. Multiplies the chroma (saturation) of all colors. Values $> 1$ increase saturation; values $< 1$ desaturate toward greyscale.

**Contrast.** Range 0.0 to 2.0, default 1.0. Expands or compresses the lightness range. High contrast darkens darks and lightens lights; low contrast pushes toward mid-grey. Implemented via S-curve:

$$f_L(\alpha) = \frac{\alpha^p}{\alpha^p + (1 - \alpha)^p} \tag{5.5}$$

where $p$ is derived from the contrast parameter.

**Vibrancy.** Range 0.0 to 2.0, default 1.0. Selectively boosts chroma of less-saturated colors while leaving already-vibrant colors unchanged (Poynton, 2012). Useful for counteracting desaturation in complementary-color interpolation.

$$C' = C \cdot \left(1 + v \cdot \left(1 - \frac{C}{C_{\max}}\right)\right) \tag{5.6}$$

where $C_{\max}$ is the maximum achievable chroma at the color's current lightness $L$ and hue $h$ within the sRGB gamut boundary (see §8.1 for gamut constraints).

## 5.5 Parameter Interactions

Dynamics parameters are designed to be *largely independent*:

- Lightness affects only $L$
- Chroma affects only $C$
- Temperature affects only hue path
- Contrast affects $L$ distribution, not absolute values

However, some combinations can conflict:

| **Combination** | **Potential Issue** |
|---|---|
| High chroma + low lightness | May exceed gamut |
| High contrast + extreme anchors | May clip at black/white |
| High temperature + cool anchors | Long hue path, more colors needed |

The engine applies all dynamics as specified, then performs gamut mapping (§8.1–§8.4). Conflicts are resolved by clamping to valid values, not by preventing the combination. Diagnostics report when settings caused gamut corrections (§10.4).

With all parameters at default values, the engine produces simple linear interpolation—the most predictable baseline. Presets encode tested combinations that work well together for specific aesthetic goals.

# Chapter 6

# Modes of Operation

## 6.1 Journey Mode

Journey Mode is the default mode of operation, optimising for **smooth, ordered sequences** suitable for timelines, gradients, transitions, and contexts where perceptual continuity matters. It leverages the full journey construction pipeline (§3.1–§3.6) with perceptual constraints (§4.1–§4.5).

| **Aspect** | **Behavior** |
|---|---|
| Adjacency constraint | Enforces $\Delta_{\min} \leq \Delta(C_i, C_{i+1}) \leq \Delta_{\max}$ |
| Path construction | Smooth Bézier curves through anchors |
| Continuity | $C^1$ continuity at anchor junctions |
| Sampling | Arc-length parameterised for perceptual uniformity |

Journey Mode is appropriate for:

- Timeline visualisations (progress indicators, history)
- State transitions (hover, active, disabled)
- Mood-based palettes (warm sunset, cool ocean)
- Animation color cycling
- Any context where "flow" matters

Journey Mode is *not* appropriate for categorical data where each color represents a distinct entity, maximum contrast requirements, or semantic color mapping (red=danger, green=safe).

## 6.2 Categorical Mode

Categorical Mode optimizes for **maximum distinguishability** between swatches, suitable for discrete data categories, legends, and accessibility-critical applications (Fairchild, 2013). Unlike Journey Mode, which optimizes adjacent-color relationships (§4.2), Categorical Mode optimizes all pairwise relationships.

| Aspect | Behavior |
|---|---|
| Distance goal | Maximise perceptual distance between all pairs |
| Path construction | May use hue maximisation, not smooth curves |
| Continuity | Not prioritised |
| Output order | Less meaningful than in Journey Mode |

Instead of constraining adjacent distances, Categorical Mode maximises minimum pairwise distance:

$$\max \min_{i \neq j} \|C_i - C_j\|_{\text{OKLab}} \tag{6.1}$$

subject to staying within displayable gamut and respecting any semantic constraints.

| Aspect | Journey Mode | Categorical Mode |
|---|---|---|
| Curve type | Smooth Bézier | May use angular paths |
| Sampling | Sequential along path | Optimized spacing in color space |
| Anchor role | Waypoints on path | Fixed category colors |
| Output order | Meaningful sequence | Order less important |

Categorical Mode is appropriate for pie charts, bar charts with distinct categories, legend colors, and status indicators (error, warning, success, info).

## 6.3 Perceptual Velocity

Beyond perceptual *distance* (how different two colors are), **perceptual velocity** addresses how *fast* color change feels along a sequence.

Fast hue swings feel "faster" or more noticeable than the same $\Delta E$ spent on subtle lightness shifts. The *type* of change affects perceived speed, not just the magnitude. This observation is consistent with color appearance research showing that hue changes are more salient than equivalent lightness or chroma changes (Fairchild, 2013).

Perceptual velocity $v$ can be modelled as a weighted sum:

$$v = w_L \cdot \frac{dL}{dt} + w_C \cdot \frac{dC}{dt} + w_h \cdot \frac{dh}{dt} \tag{6.2}$$

The following weights are *design heuristics* derived from practical experimentation, not empirically-validated constants:

| **Dimension** | **Relative Weight** | **Perceptual Effect** |
|---|---|---|
| Lightness ($L$) | 1.0 (baseline) | Subtle, gradual feel |
| Chroma ($C$) | $\sim$1.2 | Moderate drama |
| Hue ($h$) | $\sim$1.5–2.0 | High impact, "fast" feel |

*Note: These weights reflect observed behavior in prototype testing. Formal psychophysical validation remains future work.*

Velocity is relevant for temporal applications:

- **Calm presets** prefer $L/C$ changes over hue (low velocity)
- **Energetic presets** use more hue swings (high velocity)
- **Rhythmic effects** alternate high/low velocity regions

### Influencing Velocity

Users do not directly set velocity weights—they are internal heuristics. Instead, perceptual velocity emerges from the interaction of style parameters with anchor placement:

- **High temperature** increases hue travel between anchors, raising perceived velocity
- **High intensity** creates more curved paths with greater chromatic excursion
- **Anchors with similar hues** produce low-velocity journeys dominated by lightness changes
- **Anchors spanning the color wheel** produce high-velocity journeys with rapid hue shifts

Presets encode velocity profiles implicitly: "calm" presets (`smooth`, `muted`) prefer parameters that minimise hue change, while "energetic" presets (`vivid`, `neon`) encourage chromatic drama.

## 6.4 Mode Selection Guidelines

| Use Case | Recommended Mode |
|---|---|
| Output for discrete categories? | Categorical |
| Need smooth animation? | Journey + closed/pingpong loop |
| Need timeline colors? | Journey + open path |
| Need mood palette from single color? | Journey + single anchor |
| Need maximum contrast? | Categorical with high separation |

### Hybrid Approaches

Some applications benefit from combining modes:

1. **Categorical anchors, journey fill:** Use Categorical Mode to select maximally distinct anchors, then Journey Mode for smooth transitions between them
2. **Journey with velocity control:** Use Journey Mode but adjust sampling based on velocity for rhythmic effects
3. **Mode switching:** Different parts of an application use different modes (categorical for legend, journey for timeline)

# Chapter 7

# Loop Strategies

The previous sections established *what* the engine optimizes for—smooth sequential flow in Journey Mode or maximum categorical distinction—and *how* style controls shape the journey's character. This chapter addresses a complementary concern: *what happens at the boundaries* when a palette must extend beyond its natural start-to-end path, whether through cycling, reversal, or progressive evolution.

Five loop strategies provide different behaviors for sequences that need to repeat or continue beyond a single traversal.

## 7.1 Open Strategy

The **open** strategy is the default: the journey proceeds from the first anchor to the last anchor and stops. There is no attempt to connect the end back to the beginning.

$$J(t) : t \in [0, 1] \quad \text{where } J(0) = A_1, \quad J(1) = A_m \tag{7.1}$$

The output is a finite sequence with no wrap consideration. If looped externally by the caller, there will be a visible jump from $A_m$ back to $A_1$.

Open paths are appropriate for one-time state transitions, linear progress indicators, and palettes not intended to cycle.

## 7.2 Closed Strategy

The **closed** strategy forms a complete cycle, returning smoothly from the last anchor back to the first (Farin, 2002):

$$J(0) = J(1) = A_1 \tag{7.2}$$

The engine adds a segment from $A_m$ back to $A_1$, creating a closed path with $C^1$ continuity at the wrap point. The tangent leaving $A_m$ toward $A_1$ matches the tangent arriving at $A_1$ from $A_m$, preventing a "corner" in color space.

In closed loop mode, the output array *omits the duplicate final color*. The last swatch is adjacent to (but not identical to) $A_1$, because $A_1$ is already the first swatch. When looped, the sequence wraps seamlessly.

Closed loops are appropriate for continuous color cycling animations, hue wheels, and looping ambient effects.

## 7.3 Ping-Pong Strategy

The **ping-pong** strategy proceeds from start to end, then reverses back to start, creating back-and-forth oscillation.

For parameter $u \in [0, 2]$:

$$\tilde{t} = \begin{cases} u & 0 \leq u < 1 \\ 2 - u & 1 \leq u < 2 \end{cases} \tag{7.3}$$

No new colors are generated in the reverse pass—the path is exactly retraced. Smooth reversal occurs at the turning point ($A_m$) and smooth loop at the return point ($A_1$).

Ping-pong output includes the forward sequence; callers can traverse the array forward then backward, or the engine can output the full forward-and-back sequence explicitly.

Ping-pong is appropriate for "breathing" or "pulsing" color effects, bidirectional progress indicators, and oscillating state visualisations.

## 7.4 Möbius Strategy

The **Möbius** strategy is a half-twist loop requiring *two complete traversals* to return to the starting color—named after the Möbius strip, a surface with a half-twist where walking one complete loop brings you to the "other side," requiring a second loop to return to your starting position.

In color terms: after one cycle through the journey, you arrive at a color *related to* but distinct from the start; only after the second cycle do you return to the original. This creates animations that feel subtly different on alternating cycles.

One approach inverts the chromatic components at the halfway point:

$$J_{\text{mobius}}(1) = (L, -a, -b) \quad \text{when } J_{\text{mobius}}(0) = (L, a, b) \tag{7.4}$$

This yields the complementary color (180° hue rotation) at the midpoint. The viewer perceives continuous change that takes two "laps" to truly repeat:

- Cycle 1: Journey from Red $\rightarrow \ldots \rightarrow$ Cyan (complement)
- Cycle 2: Journey from Cyan $\rightarrow \ldots \rightarrow$ Red (back to original)

Möbius is an advanced option requiring careful understanding of the twist behavior. It is appropriate for complex animations with implicit alternation, effects that should vary between loops, and artistic/generative applications.

## 7.5 Phased Strategy

The **phased** strategy applies a systematic shift each repetition, so the palette evolves over time rather than repeating exactly.

On cycle $k$:

$$J_k(t) = J_0(t) + k \cdot \text{shift} \tag{7.5}$$

where the shift is a per-cycle offset vector in OKLab space. Typical configurations specify shifts as hue rotations in degrees (e.g., +10ř per cycle, applied in OKLCh) or lightness adjustments in OKLab $L$ units (e.g., +0.05 per cycle). The shift magnitude and dimension are configuration parameters.

Visual effect:

- Cycle 1: Blue $\rightarrow$ Green
- Cycle 2: Blue+10° $\rightarrow$ Green+10° (slightly shifted hues)
- Cycle 3: Blue+20° $\rightarrow$ Green+20°
- ...eventually wraps around

Phased loops are parameterised by shift amount and dimension. This is the most complex loop type, appropriate only when continuous evolution is specifically desired—slowly evolving animations, generative art with progressive change, and long-running visualisations that should not feel static.

## 7.6 Output Semantics

**Design Decision: Unrolled Loop Output**

**Choice:** All loop strategies output a flat array of $N$ colors. Loops are "unrolled" into a sequential list with no nested structure or cycle metadata.
**Rationale:** Flat arrays are universally consumable across languages and use cases. Callers needing cycle information can compute it from strategy and count; most callers don't need it.
**Alternatives Considered:**

- *Segmented output (array of cycles)* — Rejected: adds complexity for minority use case
- *Cycle metadata in output* — Rejected: bloats output, caller can derive if needed
- *Iterator/generator model* — Rejected: not portable across language bindings

**Reference:** §10.3, Appendix D

### Non-Repetition Rule

Loop outputs omit duplicate colors at boundaries. If the loop returns to the starting color, that color is *not* repeated at the end of the array:

- Conceptual path: $A \to B \to C \to D \to E \to A$
- Output array: $[A, B, C, D, E]$ (5 elements)
- When looped: $A, B, C, D, E, A, B, C, D, E, \ldots$ ($A$ follows $E$ naturally)

This prevents doubled colors when looping externally and reduces array length.

### Caller Responsibilities

Simple repetition behaviors are caller responsibility, not engine features. The engine provides smooth loop paths and appropriate swatch sequences; the caller controls "play N times then stop", animation timing, and pause/resume.

# Chapter 8

# Gamut Management

With path construction, perceptual constraints, style controls, and loop strategies established, one critical concern remains: ensuring that every color the engine generates is actually displayable. OKLab's perceptual uniformity comes at a cost—it can represent colors that no physical display can reproduce. This chapter addresses gamut management: the techniques for keeping generated colors within the bounds of real displays.

## 8.1 The Gamut Boundary Problem

OKLab is a perceptually uniform color space, but it encompasses colors that lie outside any real display's capabilities (Morovič, 2008):

- **High-chroma colors** at certain hue angles exceed sRGB
- **Very dark or very light saturated colors** clip
- **Certain hue regions** (cyan-blue, yellow-green) are particularly constrained

When constructing journeys, naive Bézier curves may traverse impossible colors. Hard clipping (clamping RGB values) creates visible discontinuities and hue shifts. Perceptual uniformity breaks down at gamut boundaries.

The fundamental constraint: all journey colors must satisfy

$$\gamma(t) \in G_{\text{sRGB}} \subset \text{OKLab} \quad \forall t \tag{8.1}$$

where $G_{\text{sRGB}}$ represents the sRGB gamut volume mapped into OKLab space.

## 8.2 Design-Time Gamut Awareness

The first layer of gamut management is **prevention**—constructing paths that stay within gamut.

When placing Bézier control points, the engine:

1. **Prefers moderate chroma levels** — Control points default to ∼70–80% of maximum chroma at their hue/lightness
2. **Detects candidate out-of-gamut points** — Before finalising control points, checks sRGB validity
3. **Pulls toward lower chroma** — If out-of-gamut detected, reduces chroma while preserving hue and lightness
4. **Uses gamut envelope models** — Employs OKLCh gamut boundary approximations for efficient checking

This proactive approach reduces the likelihood of out-of-gamut samples, preserving design intent where possible.

## 8.3 Gamut Correction

The second layer handles residual out-of-gamut colors through **soft correction** at sampling time. Our approach aligns with the CSS Color Level 4 gamut mapping algorithm (W3C CSS Working Group, 2022), which has been implemented in reference libraries such as Color.js (Verou and Lilley, 2024).

For a candidate color in OKLCh $(L, C, h)$:

1. Check sRGB validity via conversion
2. If out-of-gamut, perform binary search for maximum valid chroma at that $(L, h)$
3. Return the corrected color with reduced $C$, preserved $L$ and $h$

The binary search converges rapidly (typically 10–15 iterations) because chroma is monotonically related to gamut membership at fixed lightness and hue. The CSS standard specifies terminating the search when $\Delta E < 0.02$ (on OKLab's 0–1 scale), indicating the corrected color is perceptually indistinguishable from the gamut boundary (W3C CSS Working Group, 2022).

**Design Decision: Two-Layer Gamut Handling**

**Choice:** Gamut management uses two layers: (1) design-time prevention via gamut-aware control point placement, and (2) sample-time correction via chroma reduction with hue preservation.

**Rationale:** Post-correction alone can introduce perceptual discontinuities when many samples need large corrections. Prevention minimises corrections needed; correction handles residual cases gracefully. This approach follows established best practices in color management (Morovič, 2008).

**Alternatives Considered:**

- *Hard clipping (RGB clamping)* — Rejected: destroys hue, creates visual artifacts
- *HSV/HSL fallback* — Rejected: breaks perceptual uniformity guarantee
- *Post-processing only* — Rejected: large corrections cause discontinuities
- *Hue rotation to fit gamut* — Rejected: changes color identity unacceptably

**Reference:** §8.2, §8.4

## 8.4 Hue Preservation

When gamut mapping is required, the engine follows a strict priority hierarchy rooted in established gamut mapping practice (Morovič, 2008):

1. **Hue is sacred** — Never shift hue to fit gamut
2. **Lightness is strongly preserved** — Only adjust if absolutely necessary
3. **Chroma absorbs the compromise** — Reduce saturation to fit

This hierarchy is not arbitrary: research on gamut mapping algorithms shows that hue shifts are perceptually more objectionable than chroma reductions (Morovič, 2008; Fairchild, 2013). Humans are particularly sensitive to hue changes in familiar objects (skin tones, sky, foliage), and even small hue shifts can appear "wrong" while equivalent chroma reductions appear merely "less vivid."

This preserves the journey's "character" (its hue story) even when display limitations require desaturation (Hunt, 2004). A "sunset palette" remains warm-hued even if some oranges must be less saturated.

### Interaction with Style Controls

Dynamic controls affect gamut behavior:

| Control | Gamut Implication |
|---|---|
| Chroma $> 1.0$ | Increases gamut pressure; more mapping likely |
| Chroma $< 1.0$ | Reduces gamut pressure; safer |
| Vibrancy high | Pushes mid-journey toward gamut edges |
| Lightness extreme | May limit available chroma range |
| Temperature shift | Changes which hue regions are under pressure |

If dynamics push colors out of gamut, chroma is the first to be moderated. Diagnostics report when dynamic settings caused gamut corrections.

### Diagnostics

The engine reports gamut-related information:

- Number of colors requiring correction
- Maximum chroma reduction applied
- Which anchors were at gamut boundaries

This allows callers to understand why output differs from the theoretical ideal and adjust anchors or dynamics to avoid corrections.

# Chapter 9

# Variation and Determinism

The preceding chapters define the engine's functional behavior: how it constructs paths, applies constraints, responds to style controls, handles loops, and manages gamut boundaries. This chapter addresses a cross-cutting concern that underlies all of these capabilities: the requirement for deterministic, reproducible output—and the controlled variation layer that enables exploration without sacrificing predictability.

## 9.1 Determinism Requirement

The Color Journey Engine *must* be programmatically deterministic. Given the same inputs, it *must* produce the same outputs, up to small numerical differences ($\sim$0.2%) arising from floating-point arithmetic and platform differences. This requirement enables the "config as ID" pattern central to the API philosophy (§10.1).

This is a hard requirement, not a preference:

- **Debugging** — Developers must be able to reproduce bugs
- **Testing** — Automated tests need predictable outputs
- **Sharing** — Configurations can be shared as compact seeds/parameters
- **Version control** — Generated assets can be diffed meaningfully
- **Caching** — Identical inputs can be cached without recomputation

For identical output, the following must match: anchor colors (same order and representation), all configuration parameters, variation mode and seed (if used), and implementation version.

No source of non-determinism may affect palette generation: system time, thread scheduling, external state, true random numbers, hash table iteration order, and uninitialised memory are all prohibited.

## 9.2 Controlled Variation

While determinism is required, users may want to generate alternatives, add organic texture, or explore nearby palettes. The **variation layer** provides this through controlled, seeded perturbations.

| Mode | Effect |
|---|---|
| `off` | No variation; pure interpolation |
| `subtle` | Small perturbations; $\Delta E < 1$ |
| `noticeable` | Moderate perturbations; $\Delta E$ 1–2 |
| `extreme` | Large perturbations; $\Delta E$ 2–4 |

Variation modes are categorical rather than continuous, preventing the paralysis of infinite choice and providing meaningful distinct options.

Variation adds small offsets to interpolated colors in OKLab space:

$$L' = L + \delta_L \tag{9.1}$$

$$a' = a + \delta_a \tag{9.2}$$

$$b' = b + \delta_b \tag{9.3}$$

where perturbations are drawn from a Gaussian distribution with standard deviation scaled by mode.

Anchors are *never* perturbed—variation applies only to interpolated colors. This ensures user-specified key colors remain exact (§3.2). Perturbations taper near anchors (scaling by $\sin(\pi t)$) for smooth transitions.

## 9.3 Seed Handling

The variation layer uses a **pseudo-random number generator (PRNG)** with these properties (Knuth, 1997):

- **Seeded** — Initialised from user-provided seed
- **Deterministic** — Same seed produces same sequence
- **Well-distributed** — Numbers uniformly distributed
- **Specified algorithm** — Implementation uses documented algorithm (e.g., xoshiro256** (Blackman and Vigna, 2021))

| Seed Value | Behavior |
|---|---|
| Explicit integer | Use that seed |
| `null` or omitted | Default seed (0) or variation disabled |

If variation is requested but no seed provided, the engine uses a *default seed (0)*, not a random seed. This preserves determinism. If you want randomness, you must generate and provide the seed.

The order in which random numbers are consumed (per-color L, a, b perturbations) is fixed and documented—changing advancement order would change outputs for existing seeds.

## 9.4 Reproducibility

With determinism, a configuration object serves as a unique identifier for a specific palette. This enables the "config as ID" pattern:

- Store the configuration instead of the full palette
- Transmit compactly between systems
- Regenerate on any conforming implementation

Users can explore alternatives by iterating seeds: `seed=1` produces variant A, `seed=2` produces variant B, and so on. Each seed deterministically produces a unique but related palette.

Full reproducibility is guaranteed within the same engine version. Version changes may affect output (documented as breaking changes in semantic versioning). Recommended practice: store full configuration with generated palettes for future reference.

# Chapter 10

# API Design and Output Structure

## 10.1 API Philosophy

The engine is a **pure function** that transforms configuration into palette (Bloch, 2008):

$$(\text{anchors}, \text{config}) \rightarrow \text{palette}$$

Key characteristics:

- **Stateless** — No memory between calls
- **Deterministic** — Same inputs produce same outputs
- **Focused** — Does one thing well (color journey generation)
- **No side effects** — No global state, no external dependencies

The guiding principle: the engine should do only what the caller cannot do themselves. Complexity belongs in construction, not in the public interface (Gamma et al., 1994).

## 10.2 Input Parameters

| Parameter | Type | Required | Description |
|---|---|---|---|
| `count` | integer | Yes | Number of colors to generate |
| `anchors` | array | Yes | 1–5 colors (hex, RGB, or OKLab) |
| `temperature` | float | No | $-1$ to $+1$, default 0 |
| `intensity` | float | No | 0 to 1, default 0.5 |
| `smoothness` | float | No | 0 to 1, default 0.7 |
| `mode` | enum | No | `journey` or `categorical` |
| `loop` | enum | No | `open`, `closed`, `pingpong`, etc. |
| `seed` | integer | No | Variation seed (omit for no variation) |
| `preset` | string | No | Named preset (expands to parameters) |

When a preset is specified, it expands to a parameter set. Explicit parameters override preset defaults.

## 10.3 Output Structure

The engine returns discrete swatches (not continuous functions) with three components:

**Palette Array.** An ordered list of color swatches, each containing:

- `hex` — sRGB hex code (`#RRGGBB`)
- `ok` — OKLab coordinates (`{L, a, b}`)

Both sRGB (for immediate use) and OKLab (for further computation) are provided.

**Config Echo.** The effective configuration used, enabling reproducibility. Callers can log, store, or transmit the config to regenerate the same palette later.

**Diagnostics.** Quality metrics and constraint status for transparency.

## 10.4 Diagnostic Information

Diagnostics report quality metrics and constraint status:

| Metric | Description |
|---|---|
| `minDeltaE` | Smallest perceptual distance between adjacent swatches |
| `maxDeltaE` | Largest perceptual distance between adjacent swatches |
| `meanDeltaE` | Average distance |
| `constraintViolations` | Count of constraint violations |
| `gamutCorrections` | Count of colors requiring gamut mapping |
| `traversalStrategy` | Algorithm used (e.g., "arc-length") |

Diagnostics are informational, not prescriptive. The engine does not fail on constraint violations—it reports them and produces output anyway.

## 10.5 Scope Boundaries

**Design Decision: Engine Core vs. Caller Responsibilities**

**Choice:** The engine core owns palette generation only. Color naming, accessibility checking (WCAG), color blindness simulation, and ICC color management are explicitly caller responsibilities.

**Rationale:** Keeping the core lean enables production-grade performance (5.6M colors/second) and avoids bloating with features that vary by use case. Language bindings and applications can add these capabilities as needed.

**Alternatives Considered:**

- *Integrated accessibility checking* — Rejected: varies by standard (WCAG 2.1, 3.0), adds dependencies
- *Built-in color naming* — Rejected: naming systems are culturally/contextually specific
- *Color blindness simulation* — Rejected: simulation algorithms evolve independently

**Reference:** §??, Appendix D

Features that can be implemented by callers using the core output are not built into the engine. Examples of excluded features: "play animation at 30fps", "apply palette to image", "export to Photoshop swatch file".

## 10.6 Performance Characteristics

The C-core implementation achieves production-grade performance:

| **Metric** | **Value** |
|---|---|
| Per-color generation | Microsecond range |
| Throughput | 5.6 million colors/second[1] |
| Memory | Minimal; no caching required |

Computational complexity:

- Journey construction: $O(1)$ per anchor (fixed-degree Bézier)
- Arc-length parameterisation: $O(n)$ precomputation, $O(1)$ lookup
- Sampling: $O(n)$ for $n$ colors with constraint checking

This performance enables real-time palette generation in interactive applications, eliminates the need for precomputation or caching, and supports high-throughput batch processing.

# Chapter 11

# Caller Responsibilities

## 11.1 Input Validation

The engine performs basic input validation, but callers should validate inputs before submission:

- **Anchor colors** — Should be valid color specifications. Out-of-gamut anchors are accepted but may require correction during palette generation.
- **Count** — Must be a positive integer. Requests for zero or negative counts return an error.
- **Parameters** — Should respect documented ranges. The engine clamps out-of-range values but callers should validate to avoid surprises.
- **Presets** — Unknown preset names return an error.

## 11.2 Engine Guarantees

The engine provides the following guarantees:

1. **Output count matches request.** The palette contains exactly the requested number of colors.
2. **All outputs are valid sRGB.** Every color in the palette is displayable on standard monitors.
3. **Anchors appear in output.** When anchors are specified, they appear at their designated positions in the palette.
4. **Perceptual constraints are honored.** Adjacent colors differ by at least $\Delta_{\min}$ (where achievable).

5. **Determinism is maintained.** Identical inputs produce identical outputs across calls and platforms.

## 11.3 Caller Responsibilities

The engine generates palettes; callers are responsible for context-specific concerns:

**Accessibility Testing.** The engine does not check WCAG contrast ratios. Callers must verify that text-background pairings meet accessibility requirements for their target compliance level (AA, AAA).

**Color Blindness.** The engine does not simulate or optimize for color vision deficiencies. Callers concerned with color-blind accessibility should test generated palettes with appropriate simulation tools.

**Context Suitability.** Print and screen have different requirements (Poynton, 2012). Callers must verify that generated palettes are suitable for their target medium, including:

- Color gamut (sRGB output may not match CMYK print)
- Viewing conditions (ambient light, screen calibration)
- Substrate (paper type, coating)

**Application Constraints.** Brand colors, corporate guidelines, and domain-specific requirements are caller concerns. The engine provides building blocks; callers assemble them appropriately.

**Reproducibility.** Callers should store the config echo returned by the engine if they need to regenerate palettes later. The engine does not maintain history.

## 11.4 Error Handling

The engine handles errors gracefully:

- **Invalid inputs** — Return an error with diagnostic message. No partial output is produced.

- **Constraint conflicts** — Resolved using the priority hierarchy (§4.5). Adaptations are documented in diagnostics.
- **Impossible requests** — The closest valid alternative is returned. For example, requesting 100 visually distinct colors from a narrow hue range will produce fewer distinct steps.
- **No silent failures** — All constraint violations, gamut corrections, and adaptations appear in the diagnostics output.

# Chapter 12

# Conclusion and Future Directions

## 12.1 Summary

This specification presents the Color Journey Engine, a deterministic system for generating perceptually-uniform color palettes. The engine addresses a persistent challenge in digital color tools: how to create multi-color palettes that feel coherent while respecting the non-linear, device-dependent nature of human color perception.

The solution rests on three pillars. First, the OKLab color space Ottosson, 2020 provides a perceptually uniform foundation where equal numerical distances correspond to equal perceived differences. Second, the journey metaphor—implemented through cubic Bézier curves Farin, 2002 with arc-length parameterisation—ensures smooth, coherent transitions between colors. Third, the constraint system ($\Delta_{\text{min}}$, $\Delta_{\text{max}}$, JND-based adaptive sampling) guarantees that generated colors are both distinguishable and aesthetically pleasing.

The result is a pure function that transforms configuration into palette: stateless, deterministic, and predictable. For quick navigation during team discussions, see Appendix D for a concept map and decision rationale summaries.

## 12.2 Key Contributions

This specification makes several contributions to the practice of algorithmic color palette generation:

1. **Formal Specification.** A complete, reproducible specification for palette generation that eliminates the "magic" often present in color tools. Same inputs yield same outputs, always.

2. **Mood Expansion Algorithm.** A novel approach for generating coherent multi-color journeys from a single anchor, using style parameters to expand in perceptually meaningful directions. To our knowledge, this single-anchor expansion approach has not been formally documented elsewhere.

3. **Perceptual Constraint Framework.** A system for ensuring minimum distinguishability while allowing stylistic variation. The specific threshold values ($\Delta_{\text{min}} \approx 2.0$, $\Delta_{\text{max}} \approx 5.0$) are design heuristics informed by color science literature (Fairchild, 2013) and practical experimentation, not empirically-validated constants.

4. **Two-Layer Gamut Management.** A prevention-then-correction strategy following established gamut mapping principles (Morovič, 2008), maintaining journey character while guaranteeing displayable output.

5. **Clear Responsibility Boundaries.** Explicit delineation between engine core and caller responsibilities, enabling a lean, high-performance core while supporting extension.

### Limitations and Caveats

Several aspects of this specification warrant further investigation:

- The perceptual weights for velocity calculation (§6.3) are design heuristics, not empirically-validated values.
- OKLab's JND correspondence ($\Delta E \approx 1.0$) is a design target (Ottosson, 2020); formal perceptual validation specific to OKLab remains limited.
- The constraint thresholds ($\Delta_{\text{min}}$, $\Delta_{\text{max}}$) and subdivision caps represent engineering judgment; alternative values may be appropriate for different use cases.

## 12.3 Future Directions

Several directions merit future investigation:

**Extended Color Spaces.** While the current specification targets sRGB output, the OKLab foundation supports wider gamuts. Future versions could target Display P3 or Rec. 2020 while maintaining backward compatibility.

**Accessibility Integration.** Though accessibility checking remains a caller responsibility, future work could explore optional WCAG-aware generation modes that constrain palettes to meet contrast requirements.

**Multi-Palette Harmonisation.** Design systems often require multiple coordinated palettes (primary, secondary, semantic). Algorithms for generating harmonised palette families present an interesting extension.

**Symmetry-Constrained Optimisation.** The current engine constructs journeys through direct path specification. Future work could explore optimisation-based approaches that search for "optimal" journeys maximising perceptual distance while satisfying constraints. Recent work on symmetry-constrained search in combinatorial domains (Moosbauer and Poole, 2025) suggests that constraining search to symmetric or well-behaved curve families could dramatically reduce optimisation complexity—a principle that could apply to journey curve optimisation.

**Preset Discovery.** The current preset system encodes expert knowledge manually. Machine learning approaches could discover effective presets from successful palette examples.

**Real-Time Animation.** The engine's performance enables real-time generation. Formalising smooth interpolation between palettes could support animated color transitions in interactive applications.

The Color Journey Engine provides a foundation for these extensions while delivering immediate value: deterministic, perceptually-sound palette generation with a clean, predictable API.

# Appendix A

# Preset Reference

## A.1 Overview

Presets encode tested parameter combinations that produce aesthetically coherent results for common use cases. Each preset represents a named configuration that has been validated through practical experimentation and user feedback.

Presets serve three purposes:

1. **Quick Start** — Provide sensible defaults for common scenarios without requiring parameter knowledge
2. **Learning Aid** — Demonstrate effective parameter combinations as examples
3. **Starting Points** — Serve as baselines for customisation; explicit parameters override preset defaults

When a preset is specified in the API call (§10.2), it expands to a parameter set. Any explicit parameters provided alongside the preset will override the preset's defaults, enabling fine-tuning without starting from scratch.

## A.2 Preset Table

The following table provides the complete preset reference with all parameter values. Parameters not listed use engine defaults (§10.2).

**Table A.1:** Preset Reference

| Preset | Use Case | Temp | Int | Smooth | Mode | Loop |
|---|---|---|---|---|---|---|
| `linear` | Simple interpolation | 0.0 | 0.0 | 0.7 | journey | open |
| `smooth` | Gentle gradients | 0.0 | 0.3 | 0.9 | journey | open |
| `vivid` | High-impact palettes | 0.0 | 0.8 | 0.6 | journey | open |
| `muted` | Subdued, subtle | 0.0 | 0.2 | 0.8 | journey | open |
| `sunset` | Warm gradients | 0.7 | 0.6 | 0.8 | journey | open |
| `sunrise` | Dawn warmth | 0.5 | 0.5 | 0.9 | journey | open |
| `ocean` | Cool aquatic tones | −0.6 | 0.5 | 0.7 | journey | open |
| `forest` | Natural greens | −0.3 | 0.4 | 0.8 | journey | open |
| `autumn` | Fall foliage | 0.6 | 0.7 | 0.6 | journey | open |
| `arctic` | Ice and snow | −0.8 | 0.3 | 0.9 | journey | open |
| `neon` | Electric, vibrant | 0.0 | 1.0 | 0.5 | journey | closed |
| `pastel` | Soft, light | 0.0 | 0.2 | 0.9 | journey | open |
| `earth` | Warm naturals | 0.4 | 0.3 | 0.8 | journey | open |
| `monochrome` | Single-hue depth | 0.0 | 0.1 | 0.95 | journey | open |
| `categorical` | Distinct categories | 0.0 | 0.3 | 0.5 | categorical | open |
| `diverging` | Two-pole data | 0.0 | 0.5 | 0.7 | journey | open |
| `sequential` | Ordered data | 0.0 | 0.4 | 0.8 | journey | open |
| `breathing` | Pulsing animation | 0.0 | 0.4 | 0.95 | journey | pingpong |
| `cycling` | Continuous loop | 0.0 | 0.5 | 0.9 | journey | closed |
| `evolving` | Progressive shift | 0.0 | 0.5 | 0.8 | journey | phased |

**Key.** **Temp** = Temperature (−1 to +1); **Int** = Intensity (0 to 1); **Smooth** = Smoothness (0 to 1); **Mode** = journey or categorical (§6.1, §6.2); **Loop** = open, closed, pingpong, or phased (§7.1–§7.5).

## A.3 Preset Descriptions

### A.3.1 Basic Presets

**`linear`** The most basic preset—pure linear interpolation between anchors with no curve enhancement. Useful as a baseline for comparison or when mathematical predictability is paramount. Zero intensity means control points lie exactly on the anchor line (§3.5).

**`smooth`** Gentle, flowing gradients with high smoothness for seamless $C^1$ continuity at anchor junctions. Appropriate for backgrounds, ambient effects, and contexts where the color transition should feel natural and unobtrusive.

**`vivid`** High-intensity curves that create dramatic chromatic excursions between anchors. The path swings through higher-chroma regions, producing bold, attention-grabbing palettes. Best for creative applications where visual impact is desired.

**`muted`** Subdued palettes with low intensity and high smoothness. Produces sophisticated, understated color combinations suitable for professional interfaces, document themes, and contexts requiring visual restraint.

### A.3.2 Mood-Based Presets

**`sunset`** Warm temperature bias (+0.7) routes the hue path through reds, oranges, and yellows. High smoothness ensures the gradient feels like a natural sky progression. Ideal for conveying warmth, comfort, or evening atmosphere.

**`sunrise`** Similar to sunset but slightly cooler and softer, capturing the gentler warmth of dawn. Higher smoothness (0.9) creates an ethereal quality.

**`ocean`** Cool temperature bias (−0.6) emphasises blues, cyans, and aquatic greens. The path avoids warm hues entirely, creating cohesive underwater or maritime palettes.

**`forest`** Moderate cool bias with natural green emphasis. Lower intensity preserves the organic, muted quality of woodland colors without artificial saturation.

**`autumn`** Warm bias with higher intensity to capture the vibrant oranges, reds, and golds of fall foliage. Slightly lower smoothness allows for more distinct color “stops” resembling individual leaves.

**arctic** Strong cool bias (−0.8) with low intensity and high smoothness for ice, snow, and cold atmospheres. The resulting palettes feel crisp and minimal.

### A.3.3 Stylistic Presets

**neon** Maximum intensity with closed loop for continuous cycling. Creates electric, high-energy palettes suitable for nightclub aesthetics, gaming interfaces, or attention-grabbing animations.

**pastel** Low intensity with very high smoothness produces soft, desaturated palettes. The colors feel gentle and approachable—suitable for children's applications, wellness themes, or light UI backgrounds.

**earth** Warm bias with restrained intensity for natural, organic color combinations. Browns, tans, and warm neutrals dominate, suitable for outdoor, craft, or sustainability themes.

**monochrome** Minimal intensity (0.1) with near-maximum smoothness for single-hue depth exploration. When combined with a single anchor (§3.3), produces sophisticated tonal variations of one color.

### A.3.4 Data Visualisation Presets

**categorical** Uses Categorical Mode (§6.2) to maximise perceptual distance between all color pairs. Essential for charts, legends, and any context where colors represent distinct categories that must be easily distinguished.

**diverging** Journey Mode with balanced parameters, intended for two-anchor use where one anchor represents a negative extreme and the other a positive extreme. The neutral midpoint is automatically generated through interpolation.

**sequential** Optimized for ordered data (low-to-high, early-to-late). Smooth progression with moderate intensity creates clear directionality in the palette.

### A.3.5 Animation Presets

**`breathing`** Ping-pong loop (§7.3) with very high smoothness for pulse effects. The color smoothly advances then retreats, creating a "breathing" rhythm without any discontinuity at reversal points.

**`cycling`** Closed loop (§7.2) for continuous rotation. High smoothness ensures no visible "seam" where the loop wraps from end to beginning.

**`evolving`** Phased loop (§7.5) for long-running animations that should feel alive. Each cycle shifts slightly, preventing the static feeling of exact repetition.

## A.4 Preset Selection Guide

| If you need... | Try preset |
|---|---|
| Simple, predictable gradient | `linear` |
| Warm, inviting atmosphere | `sunset` or `earth` |
| Cool, calm, professional | `ocean` or `arctic` |
| High-energy, attention-grabbing | `vivid` or `neon` |
| Soft, gentle, approachable | `pastel` or `muted` |
| Chart/graph categories | `categorical` |
| Heatmap or sequential data | `sequential` |
| Looping animation | `cycling` or `breathing` |
| Tonal variations of one color | `monochrome` |

# Appendix B

# Mathematical Notation

## B.1 Color Space Notation

This section provides a complete reference for all mathematical notation used throughout the paper. Symbols are grouped by domain for easy lookup.

### B.1.1 OKLab Coordinates

| Symbol | Range | Meaning |
|---|---|---|
| $L$ | $[0, 1]$ | Lightness in OKLab (0 = black, 1 = white) |
| $a$ | $\approx [-0.4, 0.4]$ | Green–red opponent axis (negative = green, positive = red) |
| $b$ | $\approx [-0.4, 0.4]$ | Blue–yellow opponent axis (negative = blue, positive = yellow) |
| $(L, a, b)$ | — | A color point in OKLab Cartesian coordinates |

### B.1.2 OKLCh Coordinates (Cylindrical Form)

| Symbol | Range | Meaning |
|---|---|---|
| $L$ | $[0, 1]$ | Lightness (identical to OKLab) |
| $C$ | $[0, \sim 0.4]$ | Chroma: $C = \sqrt{a^2 + b^2}$ (colorfulness/saturation) |
| $h$ | $[0, 2\pi)$ | Hue angle: $h = \text{atan2}(b, a)$ (radians) |
| $(L, C, h)$ | — | A color point in OKLCh cylindrical coordinates |

### B.1.3 Conversion Matrices

| Symbol | Meaning |
|---|---|
| $M_1$ | XYZ to LMS transformation matrix (Equation 2.4) |
| $M_2$ | LMS to OKLab transformation matrix (Equation 2.5) |

## B.2 Distance and Constraint Notation

### B.2.1 Perceptual Distance

| Symbol | Definition | Meaning |
|---|---|---|
| $\Delta E_{OK}$ | $\sqrt{(L_2 - L_1)^2 + (a_2 - a_1)^2 + (b_2 - b_1)^2}$ | Perceptual distance in OKLab |
| $\Delta E$ | — | Shorthand for $\Delta E_{OK}$ when context is clear |
| JND | $\approx 1.0$ | Just-noticeable difference threshold |
| $D$ | — | Total perceptual distance along a segment |

### B.2.2 Constraint Thresholds

| Symbol | Value | Meaning |
|---|---|---|
| $\Delta_{\min}$ | $\approx 2.0$ | Minimum required distance between adjacent swatches |
| $\Delta_{\max}$ | $\approx 5.0$ | Maximum allowed distance between adjacent swatches |
| $n$ | $\min(5, \lceil D/\Delta_{\max} \rceil)$ | Number of segments for adaptive sampling |

## B.3 Curve and Path Notation

### B.3.1 Bézier Curve Elements

| Symbol | Range | Meaning |
|---|---|---|
| $B(t)$ | — | Bézier curve evaluated at parameter $t$ |
| $\gamma(t)$ | — | Journey path function at parameter $t$ |
| $t$ | $[0, 1]$ | Curve parameter (not arc-length) |
| $s$ | $[0, S]$ | Arc-length parameter (uniform perceptual distance) |
| $S$ | — | Total arc length of the path |
| $P_0$ | — | Start control point (equals start anchor) |
| $P_1$ | — | First interior control point |
| $P_2$ | — | Second interior control point |
| $P_3$ | — | End control point (equals end anchor) |

### B.3.2 Path Construction

| Symbol | Meaning |
|---|---|
| $\gamma_i$ | The $i$-th segment of the journey path |
| $\gamma'(t)$ | Tangent vector (derivative) of the path at $t$ |
| $\lvert\gamma'(t)\rvert$ | Magnitude of tangent (instantaneous speed) |
| $C^0$ | Position continuity at junction points |
| $C^1$ | Tangent (direction) continuity at junction points |

### B.3.3 Bézier Curve Formula

The cubic Bézier curve (Equation 3.5):

$$\gamma(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3$$

Tangent vectors at endpoints:

$$\gamma'(0) = 3(P_1 - P_0)$$
$$\gamma'(1) = 3(P_3 - P_2)$$

## B.4 Anchor Notation

| Symbol | Range | Meaning |
|---|---|---|
| $A_i$ | — | The $i$-th anchor color |
| $A_1$ | — | First anchor (journey start) |
| $A_m$ | — | Last anchor (journey end); $m \leq 5$ |
| $m$ | $[1, 5]$ | Number of anchors provided |

## B.5 Parameter Notation

### B.5.1 Primary Style Parameters

| Symbol | Range | Default | Meaning |
|---|---|---|---|
| $T$ | $[-1, +1]$ | 0 | Temperature (cool to warm hue bias) |
| $\iota$ | $[0, 1]$ | 0.5 | Intensity (control point offset scale) |
| $\sigma$ | $[0, 1]$ | 0.7 | Smoothness ($C^1$ continuity strength) |
| $N$ | $\mathbb{Z}^+$ | — | Requested palette size (count) |

### B.5.2 Secondary Style Parameters

| Symbol | Range | Default | Meaning |
|---|---|---|---|
| $\lambda$ | $[-1, +1]$ | 0 | Lightness bias (darker to lighter) |
| $\chi$ | $[0, 2]$ | 1 | Chroma multiplier (desaturate to saturate) |
| $\kappa$ | $[0, 2]$ | 1 | Contrast (compress to expand $L$ range) |
| $v$ | $[0, 2]$ | 1 | Vibrancy (selective chroma boost) |

### B.5.3 Derived Values

| Expression | Meaning |
|---|---|
| $\vec{s}_1, \vec{s}_2$ | Style-determined offset vectors for control points |
| $\hat{v}_{\text{base}}$ | Unit vector in shortest hue direction |
| $\hat{v}_{\text{warm/cool}}$ | Unit vector toward warm or cool hues |
| $\alpha$ | Blending coefficient for temperature mixing |
| $C_{\max}$ | Maximum chroma at a given $(L, h)$ within gamut |

## B.6 Variation and Determinism Notation

| Symbol | Range | Meaning |
|---|---|---|
| $\delta_L, \delta_a, \delta_b$ | varies | Perturbation offsets in OKLab dimensions |
| seed | $\mathbb{Z}$ | PRNG seed for deterministic variation |
| $k$ | $\mathbb{Z}^+$ | Cycle index for phased loops |

## B.7 Loop Strategy Notation

| Symbol | Range | Meaning |
|---|---|---|
| $J(t)$ | — | Journey function at parameter $t$ |
| $J_k(t)$ | — | Journey on cycle $k$ (phased loops) |
| $u$ | $[0, 2]$ | Extended parameter for ping-pong |
| $\tilde{t}$ | $[0, 1]$ | Remapped parameter after ping-pong transform |

### B.7.1 Loop Parameter Transforms

**Ping-pong transform:**

$$\tilde{t} = \begin{cases} u & 0 \leq u < 1 \\ 2 - u & 1 \leq u < 2 \end{cases}$$

**Möbius twist (chromatic inversion at midpoint):**

$$J_{\text{mobius}}(1) = (L, -a, -b) \quad \text{when } J_{\text{mobius}}(0) = (L, a, b)$$

**Phased shift:**

$$J_k(t) = J_0(t) + k \cdot \text{shift}$$

## B.8 Gamut Notation

| Symbol | Meaning |
|---|---|
| $G_{\text{sRGB}}$ | The sRGB gamut volume mapped into OKLab space |
| $\gamma(t) \in G_{\text{sRGB}}$ | Constraint that all path points are displayable |
| $C'$ | Corrected (reduced) chroma after gamut mapping |
| $L'$ | Adjusted lightness (if gamut mapping required) |

## B.9 Perceptual Velocity Notation

| Symbol | Typical Value | Meaning |
|---|---|---|
| $v$ | — | Perceptual velocity (weighted rate of change) |
| $w_L$ | 1.0 | Weight for lightness change rate |
| $w_C$ | $\sim$1.2 | Weight for chroma change rate |
| $w_h$ | $\sim$1.5–2.0 | Weight for hue change rate |
| $\frac{dL}{dt}, \frac{dC}{dt}, \frac{dh}{dt}$ | — | Rates of change in each dimension |

# Appendix C

# Implementation Examples

## C.1 Basic Usage

This appendix provides practical code examples demonstrating common usage patterns for the Color Journey Engine. All examples use a JavaScript-like pseudocode that maps directly to the API structure defined in §10.2.

### C.1.1 Simple Two-Anchor Gradient

The most basic usage: interpolate between two colors to create a smooth gradient.

```
const palette = generatePalette({
  count: 5,
  anchors: ['#FF6B6B', '#4ECDC4']
});

// Result: 5 colors from coral to teal
// [
//   { hex: '#FF6B6B', ok: { L: 0.70, a: 0.14, b: 0.05 } },
//   { hex: '#E08B7A', ok: { L: 0.68, a: 0.08, b: 0.03 } },
//   { hex: '#8FB8A9', ok: { L: 0.72, a: -0.03, b: 0.02 } },
//   { hex: '#5DC5B8', ok: { L: 0.74, a: -0.08, b: 0.01 } },
//   { hex: '#4ECDC4', ok: { L: 0.76, a: -0.10, b: 0.00 } }
// ]
```

**Listing C.1:** Basic Two-Anchor Gradient

### C.1.2 Multi-Anchor Journey

Three or more anchors create a journey that passes through each waypoint.

```
const palette = generatePalette({
  count: 9,
  anchors: ['#1A1A2E', '#E94560', '#0F3460'],
  smoothness: 0.8
});

// Result: dark blue -> vibrant red -> deep blue
// Anchors appear at positions 0, 4, and 8 (approximately)
// Intermediate colors flow smoothly between them
```

**Listing C.2:** Three-Anchor Journey

## C.2 Single-Anchor Expansion

Single-anchor mode (§3.3) generates harmonious variations from one color, useful for UI state variations and monochromatic palettes.

```
const palette = generatePalette({
  count: 7,
  anchors: ['#6B5B95'],  // Single purple anchor
  preset: 'monochrome'
});

// Engine expands along lightness and subtle hue directions
// Result: tonal variations centred on the anchor
// Includes lighter, darker, and slightly hue-shifted variants
```

**Listing C.3:** Single-Anchor Mood Expansion

### C.2.1 Brand Color State Variants

Generate hover, active, and disabled states from a single brand color.

```
const brandColor = '#2563EB';  // Primary blue

const states = generatePalette({
  count: 5,
  anchors: [brandColor],
  intensity: 0.2,      // Subtle variations
  smoothness: 0.95     // Very smooth transitions
});
```

```
// Use positions for different states:
// states[0] - darkest (pressed/active)
// states[2] - anchor (normal state)
// states[4] - lightest (hover/highlight)
```

**Listing C.4:** UI State Variations

## C.3 Categorical Mode

Categorical Mode (§6.2) maximises perceptual distance between all colors, essential for data visualisation.

```
const chartColors = generatePalette({
  count: 6,
  anchors: ['#2C3E50'],  // Starting point hint
  mode: 'categorical'
});

// Result: 6 maximally distinct colors
// Each pair has maximum perceptual separation
// Suitable for pie charts, bar charts, legends
```

**Listing C.5:** Categorical Palette for Chart

### C.3.1 Categorical with Fixed Anchors

Pin specific semantic colors while filling remaining slots.

```
const statusColors = generatePalette({
  count: 5,
  anchors: [
    '#DC2626',  // Red (error) - fixed
    '#16A34A',  // Green (success) - fixed
    '#2563EB'   // Blue (info) - fixed
  ],
  mode: 'categorical'
});

// Anchors preserved exactly
// Two additional colors generated with maximum distinction
```

**Listing C.6:** Semantic Categorical Palette

## C.4 Temperature Control

Temperature (§5.1) biases the hue path toward warm or cool regions.

```
// Same anchors, different temperatures
const coolPalette = generatePalette({
  count: 7,
  anchors: ['#8B5CF6', '#10B981'],  // Purple to green
  temperature: -0.6                   // Cool: via blues
});

const warmPalette = generatePalette({
  count: 7,
  anchors: ['#8B5CF6', '#10B981'],  // Same anchors
  temperature: +0.6                   // Warm: via reds/yellows
});

// coolPalette passes through cyans and teals
// warmPalette passes through magentas and yellows
// Dramatically different character, same endpoints
```

**Listing C.7:** Warm Temperature Bias

## C.5 Loop Strategies

Loop strategies (§7.1–§7.5) control how the palette wraps or cycles.

### C.5.1 Closed Loop for Animation

```
const cyclingPalette = generatePalette({
  count: 12,
  anchors: ['#E74C3C', '#3498DB', '#2ECC71'],
  loop: 'closed',
  smoothness: 0.9
});

// Last color transitions smoothly to first
```

```
// Output: 12 colors, no duplicate at end
// Loop externally: [...cyclingPalette, cyclingPalette[0], ...]
// Seamless continuous animation
```

**Listing C.8:** Closed Loop for Continuous Cycling

### C.5.2 Ping-Pong for Breathing Effect

```
const breathingPalette = generatePalette({
  count: 16,
  anchors: ['#1E3A5F', '#60A5FA'],  // Dark to light blue
  loop: 'pingpong',
  preset: 'breathing'
});

// Smooth oscillation: dark -> light -> dark
// 16 colors cover full forward-backward cycle
// No discontinuity at reversal point
```

**Listing C.9:** Ping-Pong Breathing Animation

### C.5.3 Phased Loop for Evolution

```
const evolvingPalette = generatePalette({
  count: 24,
  anchors: ['#F59E0B', '#8B5CF6'],
  loop: 'phased'
  // Each cycle shifts hue slightly
});

// Cycle 1: Amber -> Violet
// Cycle 2: Amber+shift -> Violet+shift
// Never exactly repeats, feels alive
```

**Listing C.10:** Phased Loop for Progressive Evolution

## C.6 Full Configuration

Comprehensive example demonstrating all parameters.

```
const result = generatePalette({
  // Required
  count: 8,
  anchors: ['#1A1A2E', '#E94560', '#0F3460'],

  // Primary style controls
  temperature: 0.2,      // Slightly warm bias
  intensity: 0.7,        // Dramatic curves
  smoothness: 0.8,       // Smooth anchor transitions

  // Secondary controls
  lightness: 0.0,        // No lightness shift
  chroma: 1.2,           // Boost saturation 20%
  contrast: 1.1,         // Slightly increase contrast
  vibrancy: 1.0,         // Default vibrancy

  // Behavior
  mode: 'journey',       // Smooth sequential path
  loop: 'open',          // Start to finish, no wrap

  // Reproducibility
  seed: 42               // Deterministic output
});

// Access output components
console.log(result.palette);      // Array of 8 color objects
console.log(result.config);       // Effective configuration used
console.log(result.diagnostics);  // Quality metrics
```

**Listing C.11:** Complete Configuration Example

## C.7 Working with Output

### C.7.1 Accessing Color Formats

```
const result = generatePalette({
  count: 5,
  anchors: ['#FF6B6B', '#4ECDC4']
});
```

```
// Each swatch has both formats
result.palette.forEach((swatch, i) => {
  console.log(`Color ${i}:`);
  console.log(`  Hex: ${swatch.hex}`);           // For CSS/
     display
  console.log(`  OKLab L: ${swatch.ok.L}`);      // For
     computation
  console.log(`  OKLab a: ${swatch.ok.a}`);
  console.log(`  OKLab b: ${swatch.ok.b}`);
});
```

**Listing C.12:** Using Output Color Formats

### C.7.2 Using Diagnostics

```
const result = generatePalette({
  count: 20,
  anchors: ['#FFFFFF', '#000000'],  // White to black
  chroma: 2.0                        // High chroma request
});

const diag = result.diagnostics;
console.log(`Min Delta E: ${diag.minDeltaE}`);
console.log(`Max Delta E: ${diag.maxDeltaE}`);
console.log(`Mean Delta E: ${diag.meanDeltaE}`);
console.log(`Gamut corrections: ${diag.gamutCorrections}`);

if (diag.gamutCorrections > 0) {
  console.log('Some colors were desaturated to fit sRGB gamut');
}

if (diag.constraintViolations > 0) {
  console.log('Some step sizes exceeded ideal range');
}
```

**Listing C.13:** Checking Quality Diagnostics

## C.8 Common Patterns

### C.8.1 Diverging Color Scale

```
// Diverging: negative (red) -> neutral -> positive (blue)
const divergingScale = generatePalette({
  count: 11,
  anchors: ['#DC2626', '#F5F5F5', '#2563EB'],
  preset: 'diverging'
});

// Index 5 is neutral (near white)
// Indices 0-4 are increasingly red
// Indices 6-10 are increasingly blue
```

**Listing C.14:** Diverging Scale for Data

### C.8.2 Sequential Scale with Preset

```
const heatmapScale = generatePalette({
  count: 9,
  anchors: ['#FEF3C7', '#DC2626'],  // Light yellow to red
  preset: 'sequential'
});

// Low values: light warm colors
// High values: intense red
// Ideal for heatmaps, intensity visualisations
```

**Listing C.15:** Sequential Scale Using Preset

### C.8.3 Reproducing a Palette

```
// Generate and store config
const result = generatePalette({
  count: 8,
  anchors: ['#FF6B6B', '#4ECDC4'],
  temperature: 0.3,
  seed: 12345
});

// Store the effective config (compact)
const storedConfig = result.config;
localStorage.setItem('myPalette', JSON.stringify(storedConfig));
```

```
// Later: regenerate identical palette
const savedConfig = JSON.parse(localStorage.getItem('myPalette'))
   ;
const regenerated = generatePalette(savedConfig);

// regenerated.palette is identical to original
```

**Listing C.16:** Storing and Reproducing Configuration

# Appendix D

# Quick Reference

## D.1 Quick Reference for Team Discussions

This appendix provides rapid lookup of key concepts, design decisions, and section references for use during code reviews and design discussions.

### D.1.1 Concept Map

### D.1.2 Decision Rationale Summary

This section summarises the key design decisions documented throughout the paper. Each decision box in the main text explains the choice, rationale, and rejected alternatives.

- **Mood expansion** (§3.3): Single-anchor path uses lightness-weighted directions; naive hue spin rejected to avoid non-perceptual artifacts.
- **Loop output** (§7.6): Unrolled arrays prevent nested structures; segmented outputs rejected for caller simplicity.
- **Gamut handling** (§8.2–§8.3): Two-layer approach (prevention + correction); hard clipping rejected to preserve hue; HSV fallback rejected to maintain perceptual uniformity.
- **API scope** (§10.5): Engine owns core generation; naming, accessibility, and color blindness simulation are caller responsibilities.

| Topic | Section | Key Insight |
|---|---|---|
| Why OKLab? | §2.2 | Perceptual uniformity; CAM16-level accuracy, low cost |
| OKLCh vs OKLab | §2.3 | Cylindrical for hue ops; Cartesian for distance |
| Anchor guarantee | §3.2 | Anchors appear exactly, not "close to" |
| Mood expansion | §3.3 | Single-anchor uses lightness-weighted directions |
| Bézier curves | §3.5 | Cubic curves for flexibility and $C^1$ continuity |
| Arc-length sampling | §3.6 | Uniform perceptual steps, not parametric steps |
| JND threshold | §4.1 | $\sim$1.0 theoretical, $\sim$2.0 practical |
| $\Delta_{\min}$ constraint | §4.2 | JND-based ($\sim$2 $\Delta E$); ensures distinguishability |
| $\Delta_{\max}$ constraint | §4.3 | Coherence cap ($\sim$5 $\Delta E$); no jarring jumps |
| Adaptive sampling | §4.4 | Auto-subdivide long segments, cap at 5 |
| Temperature control | §5.1 | Bias hue path warm (+) or cool (−) |
| Intensity control | §5.2 | Scale control point offset from anchor line |
| Smoothness control | §5.3 | $C^1$ continuity strength at junctions |
| Journey vs Categorical | §6.4 | Sequential flow vs maximum pairwise distinction |
| Perceptual velocity | §6.3 | Hue changes feel "faster" than $L/C$ changes |
| Loop output | §7.6 | Flat array (unrolled), no nested structure |
| Gamut prevention | §8.2 | Control points prefer moderate chroma |
| Gamut correction | §8.3 | Reduce chroma, preserve hue (never shift hue) |
| Determinism | §9.1 | Same inputs = same outputs (pure function) |
| Variation layer | §9.2 | Seeded PRNG; anchors never perturbed |
| API philosophy | §10.1 | Stateless, focused, no side effects |
| Performance | §10.6 | Microsecond-range; 5.6M colors/second |

**Table D.1:** Quick reference mapping topics to sections

### D.1.3 Parameter Quick Reference

### D.1.4 Output Structure Quick Reference

### D.1.5 Common Use Cases

### D.1.6 Perceptual Distance Interpretation

### D.1.7 Troubleshooting Guide

### D.1.8 Constraint Summary

| Parameter | Range | Default | Effect |
|---|---|---|---|
| `count` | $\mathbb{Z}^+$ | (required) | Number of output colors |
| `anchors` | 1–5 colors | (required) | Key colors to pass through |
| `temperature` | $[-1, +1]$ | 0 | Warm/cool hue bias |
| `intensity` | $[0, 1]$ | 0.5 | Curve drama (control point offset) |
| `smoothness` | $[0, 1]$ | 0.7 | $C^1$ continuity at anchors |
| `lightness` | $[-1, +1]$ | 0 | Global lightness shift |
| `chroma` | $[0, 2]$ | 1 | Saturation multiplier |
| `contrast` | $[0, 2]$ | 1 | Lightness range expansion |
| `vibrancy` | $[0, 2]$ | 1 | Selective chroma boost |
| `mode` | enum | `journey` | `journey` or `categorical` |
| `loop` | enum | `open` | `open`, `closed`, `pingpong`, etc. |
| `seed` | integer | null | Variation seed (omit = no variation) |
| `preset` | string | null | Named parameter set |

**Table D.2:** Input parameters quick reference

| Field | Contents |
|---|---|
| `palette[]` | Array of swatch objects |
| `palette[i].hex` | sRGB hex code (`#RRGGBB`) |
| `palette[i].ok` | OKLab coordinates `{L, a, b}` |
| `config` | Effective configuration (for reproducibility) |
| `diagnostics.minDeltaE` | Smallest adjacent distance |
| `diagnostics.maxDeltaE` | Largest adjacent distance |
| `diagnostics.meanDeltaE` | Average distance |
| `diagnostics.gamutCorrections` | Count of gamut-mapped colors |
| `diagnostics.constraintViolations` | Count of constraint issues |

**Table D.3:** Output structure quick reference

| Use Case | Mode | Recommended Preset |
|---|---|---|
| Smooth gradient background | journey | `smooth` |
| Warm sunset atmosphere | journey | `sunset` |
| Cool professional theme | journey | `ocean` or `arctic` |
| Chart legend colors | categorical | `categorical` |
| Heatmap scale | journey | `sequential` |
| Positive/negative diverging | journey | `diverging` |
| UI state variations | journey | `monochrome` |
| Continuous animation loop | journey (closed) | `cycling` |
| Pulsing/breathing effect | journey (pingpong) | `breathing` |
| High-energy vibrant palette | journey | `neon` or `vivid` |
| Soft, approachable colors | journey | `pastel` |

**Table D.4:** Use case to preset mapping

| $\Delta E$ Range | Interpretation |
|---|---|
| $< 1.0$ | Imperceptible to most observers |
| $1.0 - 2.0$ | Barely perceptible, subtle |
| $2.0 - 3.0$ | Noticeable, clear difference |
| $3.0 - 5.0$ | Obvious difference, still smooth |
| $> 5.0$ | Pronounced difference, may feel like a "step" |

**Table D.5:** Perceptual distance interpretation guide (from §4.1)

| Symptom | Likely Cause / Solution |
|---|---|
| Colors look washed out | High chroma caused gamut mapping; reduce `chroma` |
| Unexpected color in middle | Temperature bias routing through different hues; check `temperature` |
| Visible "jump" between colors | Step size exceeds $\Delta_{max}$; add anchors or increase `count` |
| Colors too similar | Anchors too close; use different anchors or Categorical Mode |
| Animation has visible seam | Using `open` loop; switch to `closed` |
| Different output each run | Missing `seed` with variation; provide explicit seed |
| Palette doesn't match saved | Engine version changed; store full config for reproducibility |

**Table D.6:** Common issues and solutions

| Constraint | Value | Purpose |
|---|---|---|
| Maximum anchors | 5 | Cognitive tractability (§3.2) |
| $\Delta_{min}$ | $\sim$2.0 $\Delta E$ | Ensure distinguishability (§4.2) |
| $\Delta_{max}$ | $\sim$5.0 $\Delta E$ | Prevent jarring jumps (§4.3) |
| Max subdivisions | 5 | Prevent runaway palette growth (§4.3) |

**Table D.7:** Engine constraints summary

# References


Blackman, David and Sebastiano Vigna (2021). "Scrambled linear pseudorandom number generators". In: *ACM Transactions on Mathematical Software* 47.4. Originally arXiv:1805.01407, 2018, pp. 1–32. DOI: 10.1145/3460772.

Bloch, Joshua (2008). *Effective Java*. 2nd. Addison-Wesley. ISBN: 978-0-321-35668-0.

CIE (1976). *Recommendations on Uniform Color Spaces, Color-Difference Equations, Psychometric Color Terms*. CIE Publication 15. Commission Internationale de l'Éclairage.

Cowan, Nelson (2001). "The magical number 4 in short-term memory: A reconsideration of mental storage capacity". In: *Behavioral and Brain Sciences* 24.1, pp. 87–114. DOI: 10.1017/S0140525X01003922.

Fairchild, Mark D. (2013). *Color Appearance Models*. 3rd. Wiley. ISBN: 978-1-119-96703-3.

Farin, Gerald (2002). *Curves and Surfaces for CAGD: A Practical Guide*. 5th. Morgan Kaufmann. ISBN: 978-1-55860-737-8.

Gamma, Erich et al. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. ISBN: 978-0-201-63361-0.

Hunt, R. W. G. (2004). *The Reproduction of Colour*. 6th. Wiley. ISBN: 978-0-470-02425-6.

Itten, Johannes (1961). *The Art of Color: The Subjective Experience and Objective Rationale of Color*. Revised edition 1973. Reinhold Publishing.

Kamermans, Mike (2023). *A Primer on Bézier Curves*. Online. Comprehensive treatment of arc-length parameterisation in §14. Accessed: 2025-12-17. URL: https://pomax.github.io/bezierinfo/.

Knuth, Donald E. (1997). *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. 3rd. Addison-Wesley. ISBN: 978-0-201-89684-8.

Levien, Raph (2018). *How long is that Bézier?* Online. Practical algorithms for Bézier arc-length computation. Accessed: 2025-12-17. URL: https://raphlinus.github.io/curves/2018/12/28/bezier-arclength.html.

Levien, Raph (2021). *An interactive review of Oklab*. Online. Independent validation of OKLab perceptual uniformity claims. Accessed: 2025-12-17. URL: https://raphlinus.github.io/color/2021/01/18/oklab-critique.html.

Liu, Yuzhen et al. (2013). "Image-driven harmonious color palette generation". In: *IEEE Transactions on Visualization and Computer Graphics*. Vol. 19. 6. Data-driven approach; contrast with our deterministic path method, pp. 978–989. DOI: 10.1109/TVCG.2012.155.

Luo, Ming Ronnier, Guihua Cui, and Bryan Rigg (2001). "The development of the CIE 2000 colour-difference formula: CIEDE2000". In: *Color Research & Application* 26.5, pp. 340–350. DOI: 10.1002/col.1049.

Madsen, Rune (2017). *Programming Design Systems: Perceptually Uniform Color Spaces*. Online. Accessible introduction to perceptual uniformity for programmers. Accessed: 2025-12-17. URL: https://programmingdesignsystems.com/color/perceptually-uniform-color-spaces/.

Mahy, Marc, Luc Van Eycken, and André Oosterlinck (1994). "Evaluation of uniform color spaces developed after the adoption of CIELAB and CIELUV". In: *Color Research & Application* 19.2, pp. 105–121. DOI: 10.1002/col.5080190207.

Moosbauer, Jakob and Michael J. Poole (2025). "Flip Graphs with Symmetry and New Matrix Multiplication Schemes". In: *arXiv preprint arXiv:2502.04514*. Methodological influence: symmetry-constrained search reduces optimisation complexity.

Morovič, Ján (2008). *Color Gamut Mapping*. Wiley. ISBN: 978-0-470-03032-5.

Ottosson, Björn (2020). *A perceptual color space for image processing*. Online. Accessed: 2025-12-17. URL: https://bottosson.github.io/posts/oklab/.

Piegl, Les and Wayne Tiller (1997). *The NURBS Book*. 2nd. Springer. ISBN: 978-3-540-61545-3.

Poynton, Charles (2012). *Digital Video and HD: Algorithms and Interfaces*. 2nd. Morgan Kaufmann. ISBN: 978-0-12-391926-7.

Safdar, Muhammad et al. (2017). "Perceptually uniform color space for image signals including high dynamic range and wide gamut". In: *Optics Express* 25.13, pp. 15131–15151. DOI: 10.1364/OE.25.015131.

Stone, Maureen C., Danielle Albers Szafir, and Vidya Setlur (2014). "An engineering model for color difference as a function of size". In: *Color and Imaging Conference* 2014.1. Broader context on perceptual colour issues in digital tools, pp. 228–233.

Verou, Lea and Chris Lilley (2024). *Color.js: Gamut Mapping Documentation*. Online. Reference implementation of CSS gamut mapping algorithm. Accessed: 2025-12-17. URL: https://colorjs.io/docs/gamut-mapping.html.

W3C CSS Working Group (2022). *CSS Color Module Level 4*. W3C Candidate Recommendation. Accessed: 2025-12-17. World Wide Web Consortium (W3C). URL: https://www.w3.org/TR/css-color-4/.