Generated: 2025-11-08 06:57 UTC

Provisional Patent Application Outline

Title: Systems and Methods for Color-Encoded Program Representation with Band-Aware Decoding, Integrity Protection, and Compression

Date: November 7, 2025

Status: Draft outline for counsel review. NOT LEGAL ADVICE.

1. Cross-Reference (Optional)

Reference to earlier internal disclosures (Design Document, Specification, Phase 2 Plan) stored in repo.

2. Field of the Invention

Relates to program representation and execution systems, particularly those encoding executable instructions as color (HSV) pixels within an image artifact enabling spatial decoding, integrated compression, integrity, and UI state rendering.

3. Background

Conventional languages rely on text-based tokens. Visual and esoteric languages (e.g., Piet) traverse color regions but lack integrated hybrid compression tuned for instruction grids, error-correcting hue band semantics, and unified UI/state rendering pipelines. There is a need for a machine-native, image-based representation achieving improved compressibility, robustness to minor color perturbations, spatial provenance, and direct component/state visualization.

4. Summary of the Invention

Discloses systems and methods for:

1. Encoding program instructions as HSV pixels with discrete hue band partitions for opcode classes and saturation/value operand quantization.

2. Executing the encoded program via a spatial virtual machine with row-major or branch-directed traversal.

3. Applying hybrid compression (palette reduction + run-length encoding) over opcode-indexed tiles to generate a compact artifact with deterministic rehydration.

4. Employing integrity and error-correction across hue bands using redundancy and majority voting/parity to tolerate small hue perturbations while preserving opcode class.

5. Rendering application or component state (ColorReact) into executable color grids maintaining one canonical artifact for code, UI, and meta-state (cognition strip).

6. Optionally applying tile-level hashing and a Merkle structure for provenance and tamper detection.

5. Brief Description of Drawings (To Be Produced)

FIG. 1: Hue band partitioning diagram.

FIG. 2: Decoding pipeline from HSV pixel to instruction tuple.

FIG. 3: Spatial VM architecture and control flow modifications.

FIG. 4: Hybrid compression flow (palette extraction followed by RLE over indices).

FIG. 5: ECC redundancy pattern and majority vote decision tree.

FIG. 6: Integrity tile hashing and Merkle tree root generation.

FIG. 7: Component state → color grid rendering pipeline (ColorReact).

FIG. 8: Cognition strip utilization and external consumer interface.

FIG. 9: GPU shader decoding variant (optional embodiment).

6. Detailed Description

6.1 Program Image Structure

Grid G of dimensions W×H, each pixel p = (H,S,V).

Hue bands B = {b0...bn} define opcode classes; example partitions (ADD, LOAD, CONTROL, DATA, I/O, HALT, etc.).

Operand quantization: S and V mapped via linear or piecewise transforms to register IDs, immediates, addresses; specify ranges and resolution.

6.2 Decoding Algorithm

Pseudo-process:

1. Acquire pixel.

2. Determine band by hue interval search.

3. Validate band with ECC parity/redundancy (if enabled).

4. Extract operands from S,V quantization.

5. Form instruction tuple (opcode, operand1, operand2, flags).

6. Dispatch to VM execution unit.

6.3 Virtual Machine Execution Model

Registers: General, data, auxiliary.

Memory model: Linear or segmented address space; optional direct color region addressing.

Control flow: JUMP or BRANCH opcodes alter pixel traversal index; fallback row-major.

Error handling: Invalid hue band triggers exception or safe skip.

6.4 Data Instructions

INTEGER/FLOAT encoded via S,V quantization; sign bit method; float exponent/mantissa partition (detail table).

6.5 Hybrid Compression

Steps:

1. Palette extraction over pixel set constrained by hue band awareness.

2. Remap pixels to palette indices preserving operand resolution.

3. RLE scanning row-wise storing (index, run_length) pairs.

4. Serialize artifact with palette array, run list, metadata: dimensions, integrity hashes, ECC config.

5. Rehydration: decode runs → expand indices → reconstruct HSV palette colors → decode instructions.

6.6 Error-Correcting and Robust Decoding

Redundancy: replicate opcode class across k adjacent pixels; majority vote on hue band.

Parity pixel: additional pixel storing checksum bits for preceding group.

Tolerance: Accept hue perturbation ±δ without band shift if within guard margin; define δ empirically (e.g., ≤2°).

6.7 Integrity and Provenance

Tile hashing: Partition image into T×T tiles; compute SHA-256 per tile.

Merkle tree: Combine tile hashes to root for artifact signature.

Verification: On load, recompute tile hashes; compare root.

6.8 Component Rendering (ColorReact)

Components produce HSV subgrids.

Composition: Layout engine merges subgrids into global artifact; reserves cognition strip row.

Determinism: Same props/state yield identical pixel arrangement; caching and diffing optional.

6.9 Cognition Strip Meta-State

Reserved N pixels; schema: emotion, action intent, memory recall, social cue, goal evaluation.

External consumer reads strip; influences next render cycle or agent decision layer.

6.10 GPU Embodiment (Optional)

Fragment shader: sample pixel, decode hue band, output intermediate instruction buffer.

Compute kernel: parallel band classification + operand extraction; writes to dispatch queue.

6.11 Performance and Robustness Benchmarks (Planned)

Compression ratios vs PNG+zstd, WebP.<

Misdecode probability under synthetic noise.

Decode throughput pix/ms CPU vs GPU variant.

Deterministic rehydration verification (hash equality).

6.12 Alternative Embodiments

Binary-grid variant with channel triplets (reserved for comparative study).

Different color spaces (Lab, YCbCr) with adapted banding.

ECC based on Reed–Solomon across pixel groups.

7. Exemplary Pseudocode (Selected)

Decoding skeleton:

```
for y in range(H):
    for x in range(W):
        H,S,V = read_pixel(x,y)
        band = classify_hue(H)
        if ecc_enabled:
            band = majority_vote(band, neighborhood(x,y))
        operands = quantize_operands(S,V, band)
```

```
instr = (opcode_map[band], operands)

vm.execute(instr)
```

Hybrid compression serialization:

```
palette = extract_palette(pixels)

indices = map_to_indices(pixels, palette)

runs = rle_encode(indices)

artifact = {

'dims': (W,H),

'palette': palette,

'runs': runs,

'integrity': tile_hashes,

'ecc': ecc_config

}
```

8. Advantages Over Prior Art

Unified executable + UI + meta-state artifact.

Tailored compression pipeline exploiting opcode palette constraints.

Hue band ECC preserving semantics under minor perturbations.

Integrity hashing integrated at tile granularity for provenance.

9. Potential Claim Set (Draft Language)

Claim 1 (Method): A method comprising encoding program instructions as hue-banded pixels with saturation and value operand quantization; decoding by classifying hue into an opcode class; applying error-correction across redundant pixel groups; executing via a spatial traversal policy; and generating an execution trace.

Claim 2 (System): A system including a decoder, an error-correction module performing majority voting over hue band redundancies, a hybrid compressor combining palette reduction and run-length encoding over opcode indices, and a virtual machine executing reconstructed instructions.

Claim 3 (Article of Manufacture): A non-transitory computer-readable medium storing a serialized color-program artifact comprising dimensions, palette, run-length encoded indices, integrity tile hashes, and error-correction configuration enabling deterministic rehydration.

Claim 4–N (Dependent): Specific hue partitions, operand mapping formulas, cognition strip schema, tile hash + Merkle integration, GPU parallel decoding, ECC tolerance threshold.

10. Disclosure Completeness Checklist

[ ] Precise hue band table.

[ ] Operand quantization ranges/formulas.

[ ] ECC redundancy pattern diagrams.

[ ] Compression artifact schema specification (JSON/binary).

[ ] Integrity tile size and hashing algorithm.

[ ] Cognition strip pixel ordering and semantics.

[ ] Benchmark methods (timing, noise generation).

[ ] Alternative embodiments (binary-grid, different color spaces).

11. Open Items Before Filing

Generate actual FIG diagrams (vector/SVG).

Finalize hue band intervals and operand scaling constants.

Demonstrate ECC robustness with quantitative misdecode data.

Produce benchmark tables vs baselines.

Decide inclusion of cognition strip in independent claim (may weaken breadth).

12. Disclaimer

This outline is preparatory and requires formal prior-art searching and claim drafting by qualified counsel. No guarantee of patent issuance is made.