GSS & AGSS Design Specification

Overview

GSS (Gaussian Sprite Sheets): A CSS-like DSL for defining high-performance Gaussian field rendering with mathematical elegance.

AGSS (Agentic GSS): Extension of GSS with AI-powered parameter optimization and automated tuning capabilities.

Architecture

```
┌──────────────────────────────────────────────────────────────────┐
│                        GSS/AGSS System                           │
├──────────────────────────────────────────────────────────────────┤
│  Input Layer                                                     │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐            │
│  │ GSS Parser   │  │ CSS Vars     │  │ @agent blocks│            │
│  │ (LPEG)       │  │ Manifest     │  │ Parser       │            │
│  └──────┬───────┘  └──────┬───────┘  └──────┬───────┘            │
│         │                 │                 │                    │
│         └─────────────────┼─────────────────┘                    │
│                           ▼                                      │
│  ┌────────────────────────────────────────────────────┐          │
│  │          Kernel Graph IR                           │          │
│  │  • Gaussian nodes  • Mix nodes    • Sum nodes      │          │
│  │  • Ramp LUTs       • Iso nodes    • Composite nodes│          │
│  └─────────────────────┬──────────────────────────────┘          │
│                        ▼                                         │
│  ┌────────────────────────────────────────────────────┐          │
│  │          Lowering Pipeline                         │          │
│  │  • Semantic analysis   • Type checking             │          │
│  │  • Optimization        • Code generation           │          │
│  └─────────────────────┬──────────────────────────────┘          │
│                        ▼                                         │
│  ┌────────────────────────────────────────────────────┐          │
│  │          Runtime Kernels                           │          │
│  │  • Gaussian tile renderer  • Ramp LUT lookup       │          │
│  │  • Marching squares iso    • Blend modes           │          │
│  │  • Tile cache              • WASM backend          │          │
│  └─────────────────────┬──────────────────────────────┘          │
│                        ▼                                         │
│  ┌────────────────────────────────────────────────────┐          │
│  │          AGSS Agent Runtime                        │          │
│  │  • Parameter sweep strategies (grid/random/bayes)  │          │
│  │  • Performance measurement (FPS/latency/SSIM)      │          │
│  │  • Tape-deck controls (Play/Pause/Stop/Record)     │          │
│  │  • Structured logging & checkpointing              │          │
│  └────────────────────────────────────────────────────┘          │
│                                                                  │
│  Output Layer                                                    │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐            │
│  │ Canvas RGBA  │  │ Metrics CSV  │  │ Agent Logs   │            │
│  │ Buffer       │  │ Reports      │  │ (MD/JSON)    │            │
│  └──────────────┘  └──────────────┘  └──────────────┘            │
└──────────────────────────────────────────────────────────────────┘
```

GSS Grammar (EBNF)

```
Stylesheet   ::= { Block }
Block        ::= "gss" Identifier? "{" { Stmt } "}"
Stmt         ::= FieldStmt | RampStmt | IsoStmt | BlendStmt
               | BindStmt | AnimateStmt | LayerStmt | SizeStmt

FieldStmt    ::= "field:" FieldExpr ";"
FieldExpr    ::= "gaussian" "(" ArgList ")"
               | "mix" "(" FieldExpr "," FieldExpr "," Weight ")"
               | "sum" "(" FieldExpr { "," FieldExpr } ")"

RampStmt     ::= "ramp:" (Ident | "custom(" ColorStops ")") ";"
IsoStmt      ::= "iso:" NumberOrPercent ["," NumberOrPercent] ";"
BlendStmt    ::= "blend:" BlendMode ";"
BindStmt     ::= "bind:" VarList ";"
AnimateStmt  ::= "animate:" Ident "(" ArgList ")" ";"
LayerStmt    ::= "layer" Identifier "{" { Stmt } "}"
SizeStmt     ::= "size:" Length "x" Length ";"

BlendMode    ::= "normal" | "multiply" | "screen" | "softlight" | "overlay"
ArgList      ::= Expr { "," Expr }
VarList      ::= Ident { "," Ident }
ColorStops   ::= ColorStop { "," ColorStop }
ColorStop    ::= Color "@" Percent
```

AGSS Grammar Extension

```
AgentBlock   ::= "@agent" Identifier "{" OptimizeBlock "}"
OptimizeBlock ::= "optimize" "{" { AgentStmt } "}"

AgentStmt    ::= TargetStmt | VaryStmt | BudgetStmt
               | StrategyStmt | RecordStmt

TargetStmt   ::= "target:" Metric ":" Value ";"
VaryStmt     ::= "vary:" ParamRange { "," ParamRange } ";"
ParamRange   ::= Ident "∈" "[" Number "," Number "]" ["step" Number]
BudgetStmt   ::= "budget:" ("trials:" Number | "time:" Duration) ";"
StrategyStmt ::= "strategy:" Strategy ";"
RecordStmt   ::= "record:" FieldList ";"

Strategy     ::= "grid" | "random" | "bayes" | "anneal"
Metric       ::= "fps" | "latency" | "ssim" | "quality"
```

Kernel Graph IR

Node Types

```
-- Gaussian field node
GaussianNode = {
    type = "gaussian",
    muX = expr,      -- center X (CSS var or literal)
    muY = expr,      -- center Y
    sigma = expr,    -- standard deviation
    id = unique_id
}

-- Mix node (weighted blend)
MixNode = {
    type = "mix",
    input1 = node_ref,
    input2 = node_ref,
    weight = expr,   -- 0.0 to 1.0
    id = unique_id
}

-- Sum node (additive blend)
SumNode = {
    type = "sum",
    inputs = {node_ref, ...},
    normalize = bool,
    id = unique_id
}

-- Ramp LUT node
RampNode = {
    type = "ramp",
    input = node_ref,
    palette = "viridis" | "plasma" | "custom",
    lut = {r, g, b, a}[256],  -- precomputed
    id = unique_id
}

-- Iso contour node
IsoNode = {
    type = "iso",
    input = node_ref,
    threshold = expr,
    width = expr,
    id = unique_id
}

-- Composite node (final output)
CompositeNode = {
    type = "composite",
    layers = {node_ref, ...},
    blend_mode = "normal" | "multiply" | ...,
    id = unique_id
}
```

Runtime Kernels

Gaussian Tile Renderer

```
function gaussian_tile(buf, ox, oy, w, h, muX, muY, sigma, rampLUT)
    local inv2s2 = 1.0 / (2.0 * sigma * sigma)

    for y = 0, h-1 do
        for x = 0, w-1 do
            local dx = (x + ox) - muX
            local dy = (y + oy) - muY
            local r2 = dx*dx + dy*dy
            local g = math.exp(-r2 * inv2s2)

            -- Clamp and lookup color
            g = math.max(0.0, math.min(1.0, g))
            local idx = math.floor(g * 255)
            local color = rampLUT[idx]

            -- Write RGBA
            local offset = (y * w + x) * 4
            buf[offset + 0] = color.r
            buf[offset + 1] = color.g
            buf[offset + 2] = color.b
            buf[offset + 3] = color.a
        end
    end
end
```

Marching Squares Iso Contours

```
function marching_squares(field, w, h, threshold)
    local contours = {}

    for y = 0, h-2 do
        for x = 0, w-2 do
            -- Sample 2x2 cell
            local v00 = field[y*w + x]
            local v10 = field[y*w + x+1]
            local v01 = field[(y+1)*w + x]
            local v11 = field[(y+1)*w + x+1]

            -- Build case mask
            local mask = 0
            if v00 >= threshold then mask = mask + 1 end
            if v10 >= threshold then mask = mask + 2 end
            if v11 >= threshold then mask = mask + 4 end
            if v01 >= threshold then mask = mask + 8 end

            -- Lookup edge configuration and interpolate
            local edges = MARCHING_SQUARES_TABLE[mask]
            for _, edge in ipairs(edges) do
                table.insert(contours, interpolate_edge(edge, x, y, v00, v10, v01,
v11, threshold))
            end
        end
    end

    return contours
end
```

Performance Targets

Milestone A (GSS)

- **640×480 single blob**: ≥60 FPS
- **1280×720 two blobs + iso**: ≥30 FPS
- **First paint**: ≤300 ms
- **Parameter update**: ≤60 ms

Optimization Strategies

1. **Tile-based rendering**: 128×128 tiles, batch 4-8 per yield
2. **Tile caching**: Key by (muX, muY, sigma, viewport)
3. **Downsampled iso**: Factor 2-4 for marching squares
4. **LUT precomputation**: 256-entry color ramps
5. **FP64 math, RGBA8 output**: Precision vs performance
6. **WASM backend**: LuaJIT → WASM via WASI

AGSS Agent Runtime

Search Strategies

```lua
-- Grid search
function grid_search(params, ranges, step_sizes)
    for sigma = ranges.sigma.min, ranges.sigma.max, step_sizes.sigma do
        for muX = ranges.muX.min, ranges.muX.max, step_sizes.muX do
            yield {sigma=sigma, muX=muX}
        end
    end
end

-- Random search
function random_search(params, ranges, num_trials)
    for i = 1, num_trials do
        local sample = {}
        for k, range in pairs(ranges) do
            sample[k] = range.min + math.random() * (range.max - range.min)
        end
        yield sample
    end
end

-- Bayesian optimization (simplified)
function bayesian_search(params, ranges, num_trials)
    local gp = GaussianProcess.new()

    for i = 1, num_trials do
        local sample = gp:suggest(ranges)
        local reward = yield sample
        gp:observe(sample, reward)
    end
end
```

Measurement & Logging

```lua
function measure_performance(render_fn, params)
    local start = os.clock()
    local frames = 0

    -- Render for 1 second
    while os.clock() - start < 1.0 do
        render_fn(params)
        frames = frames + 1
    end

    local elapsed = os.clock() - start
    local fps = frames / elapsed
    local latency = (elapsed / frames) * 1000  -- ms

    return {
        fps = fps,
        latency = latency,
        frames = frames,
        elapsed = elapsed
    }
end

function log_trial(trial_num, params, metrics)
    local entry = {
        trial = trial_num,
        timestamp = os.time(),
        params = params,
        metrics = metrics,
        checksum = compute_checksum(params)
    }

    -- Append to CSV
    append_csv("experiments.csv", entry)

    -- Append to Markdown log
    append_markdown("experiments.md", entry)
end
```

Acceptance Criteria

A1: Engine Boundary + JS Fallback

- ✅ ≥60 FPS with 500 splats
- ✅ Tile-based rendering with top-K selection
- ✅ Deterministic seed (0x1badc0de)

A2: Benchmark Harness

- ✅ CSV metrics: fps_mean, p95, CPU%, memory MB
- ✅ SSIM, ΔE color difference

A3: Baseline Comparisons

- ✅ CSS/Canvas/SVG baseline renderers
- ✅ SSIM thresholds verified

A4: GSS Parse/Compile

- ✅ Classes update canvas in ≤1 frame
- ✅ CSS var manifest generation

A5: Agent Loop

- ✅ Reward improvement ≥10 iterations
- ✅ Safety stop on SSIM drop

A6: WASM Path

- ✅ Same tests pass
- ✅ Hot-swap between engines

File Structure

```
gss/
├── grammar/
│   ├── gss.lpeg          # LPEG grammar definition
│   └── agss.lpeg         # AGSS extension grammar
├── parser/
│   ├── parser.lua        # Main parser
│   ├── ast.lua           # AST node definitions
│   └── semantic.lua      # Semantic analysis
├── ir/
│   ├── graph.lua         # Kernel graph IR
│   ├── nodes.lua         # Node type definitions
│   └── lowering.lua      # Lowering pipeline
├── runtime/
│   ├── gaussian.lua      # Gaussian kernel
│   ├── ramp.lua          # Ramp LUT
│   ├── iso.lua           # Marching squares
│   ├── blend.lua         # Blend modes
│   ├── cache.lua         # Tile cache
│   └── wasm.lua          # WASM backend
├── agss/
│   ├── agent.lua         # Agent runtime
│   ├── strategies.lua    # Search strategies
│   ├── metrics.lua       # Performance measurement
│   └── logging.lua       # Structured logging
├── tests/
│   ├── test_parser.lua
│   ├── test_kernels.lua
│   ├── test_agss.lua
│   └── test_integration.lua
└── benchmarks/
    ├── bench_gss.lua
    └── bench_agss.lua
```

Implementation Phases

Phase 1: Grammar & Parser (Days 1-2)

- Define GSS grammar in LPEG
- Implement parser with AST generation
- Add semantic analysis

- Unit tests for parser

Phase 2: Kernel Graph IR (Days 2-3)

- Define IR node types
- Implement graph construction
- Add topological sorting
- Lowering pipeline to LUASCRIPT

Phase 3: Runtime Kernels (Days 3-5)

- Gaussian tile renderer
- Ramp LUT system
- Marching squares iso
- Blend modes
- Tile caching

Phase 4: Performance Optimization (Days 5-6)

- Benchmark harness
- Profile and optimize
- WASM backend
- Meet performance targets

Phase 5: AGSS Implementation (Days 6-7)

- Agent grammar extension
- Search strategies
- Measurement & logging
- Tape-deck UI controls

Phase 6: Integration & Testing (Days 7-8)

- Full integration tests
- End-to-end validation
- Documentation
- Demo gallery

Mathematical Elegance

All Unicode operators must work identically to ASCII equivalents:

```
-- Gaussian formula (Unicode)
g = e^(-(r²)/(2σ²))

-- Gaussian formula (ASCII)
g = math.exp(-(r*r)/(2*sigma*sigma))

-- Difference must be ≤1 ULP
assert(math.abs(unicode_result - ascii_result) <= 1e-15)
```

Leadership Principles

Steve Jobs: “Ship it now. CSS-like elegance. Minimal grammar. Wow factor.”

Donald Knuth: “Provably correct. Formula-to-pixels equivalence. Formal rigor.”

Status: Design Complete ✅
Next: Implementation Phase 1 - Grammar & Parser