Splatter Solver: A Multi-Stage Greedy Optimization Algorithm for Probabilistic Image Reconstruction

ALGORITHMIC SOLUTION FOR THE CHRISTMAS CONTEST 2025

Contents

Author's address: Algorithmic Solution for the Christmas Contest 2025.

1 INTRODUCTION

1.1 Problem Statement

Given a target image $\mathcal{T}$ of dimensions $H \times W$ with pixels $T(x, y) \in [0, 255]^3$ (RGB color space), and an initially white canvas C where every pixel is $(255, 255, 255)$, the task is to produce a sequence of at most $N = 10{,}000$ paintball shots that minimizes the reconstruction error.

Each shot s_i is parameterized by:

- Position: $(x_i, y_i) \in \{0, \ldots, H-1\} \times \{0, \ldots, W-1\}$
- Radius parameter: $D_i \in \{1, \ldots, 100\}$
- Color: $C_i = (r_i, g_i, b_i) \in [0, 255]^3$

The key challenge is that painting is *probabilistic*: each pixel within the shot's influence region is painted independently with a probability that depends on its distance from the shot center. Furthermore, later shots *overwrite* earlier ones—there is no color blending.

1.2 Quality Metric

The reconstruction quality is measured by the **Sum of Squared Errors (SSE)**:

$$\mathrm{SSE}(C, \mathcal{T}) = \sum_{(x,y)} \|C(x, y) - T(x, y)\|^2 \tag{1}$$

where $C(x, y)$ is the final canvas color at pixel (x, y), and $\|\cdot\|^2$ denotes the squared Euclidean norm in RGB space:

$$\|(r_1, g_1, b_1) - (r_2, g_2, b_2)\|^2 = (r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2 \tag{2}$$

1.3 Solution Overview

Our algorithm consists of six sequential stages:

(1) **Coarse Greedy Stage**: Captures major image features using large radii ($D \in \{80, 40, 20\}$) on a sparse grid.
(2) **Fine Greedy Stage**: Fills in details using small radii ($D \in \{1, \ldots, 8\}$) at every pixel.
(3) **Color Refitting**: Re-optimizes shot colors accounting for overwrite probabilities via a reverse sweep.
(4) **Order-Preserving Pruning**: Reduces the shot count from ∼11,000 to exactly 10,000 while preserving temporal order.
(5) $D = 1$ **Postponement**: Moves single-pixel shots to the end and refills with highest-error pixels.
(6) **Local Geometry Tuning**: Fine-tunes shot positions and radii via local search.

2 THE PROBABILISTIC PAINTING MODEL

2.1 Paint Probability Function

When a shot is fired at position (c_x, c_y) with radius parameter D and color C, each pixel (x, y) at Euclidean distance $d = \sqrt{(x - c_x)^2 + (y - c_y)^2}$ from the center is painted with probability:

$$p(d, D) = \begin{cases} 1 & \text{if } d \leq 0.6D \\ 1 - t^2, \quad t = \dfrac{d - 0.6D}{0.4D} & \text{if } 0.6D < d < D \\ 0 & \text{if } d \geq D \end{cases} \tag{3}$$

This can be written more compactly using the clamp function:

$$p(d, D) = 1 - t^2, \quad \text{where } t = \text{clamp}\left(\frac{d - 0.6D}{0.4D}, 0, 1\right) \tag{4}$$

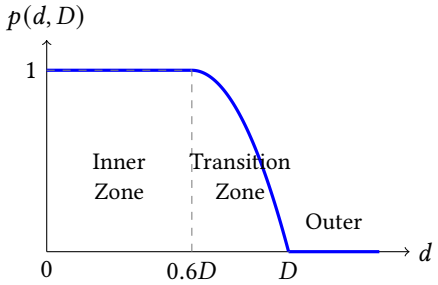

Fig. 1. Paint probability as a function of distance from shot center. The "solid core" (inner zone) always receives paint, while the "spray zone" (transition zone) has quadratically decreasing probability.

2.2 Physical Intuition

The probability model captures the physics of a paintball impact:

- **Inner Zone** ($d \leq 0.6D$): The solid core of the paintball impact, where paint concentration is high enough to guarantee coverage.
- **Transition Zone** ($0.6D < d < D$): The spray region where paint splatters with decreasing density, modeled by the quadratic falloff $1 - t^2$.
- **Outer Zone** ($d \geq D$): Beyond the splash radius; no paint reaches here.

2.3 The Overwrite Model

Crucially, later shots *overwrite* earlier ones without blending. If pixel (x, y) is painted by shot s_i with color C_i, and later shot s_j (where $j > i$) also paints this pixel with color C_j, then the final color is C_j, not a blend of C_i and C_j.

This creates a "last writer wins" semantics that is essential to our optimization strategy.

3 EXPECTED SSE FRAMEWORK

3.1 The Key Insight: Optimizing Expected Error

Since the actual painting outcome is stochastic, directly optimizing the final SSE is intractable—it would require Monte Carlo simulation. Instead, we optimize the **expected** SSE, which admits closed-form updates and enables deterministic greedy optimization.

Definition 3.1 (Expected SSE)**.** Let $E(x, y)$ denote the expected squared error at pixel (x, y) after applying some sequence of shots. This is the expected value of $\|C(x, y) - T(x, y)\|^2$ over all random outcomes of the painting process.

3.2 The Markov Update Property

Theorem 3.2 (Expected SSE Update). *Let $E(x, y)$ be the expected SSE at pixel (x, y) before applying a shot at (c_x, c_y) with radius D and color C. Let $p = p(d, D)$ where $d = \sqrt{(x - c_x)^2 + (y - c_y)^2}$. Then the new expected SSE is:*

$$E'(x, y) = (1 - p) \cdot E(x, y) + p \cdot \|C - T(x, y)\|^2 \tag{5}$$

Proof. There are two possible outcomes for pixel (x, y):

- With probability $(1 - p)$: The pixel is not painted by this shot. Its expected squared error remains $E(x, y)$ (by the law of total expectation, this accounts for all possible earlier painting outcomes).
- With probability p: The pixel is painted with color C. The squared error becomes exactly $\|C - T(x, y)\|^2$, regardless of what color was there before (due to overwriting).

By the law of total expectation:

$$E'(x, y) = (1 - p) \cdot E(x, y) + p \cdot \|C - T(x, y)\|^2 \tag{6}$$

□

Remark 3.3. The update formula (5) can be rewritten as:

$$E'(x, y) = E(x, y) + p \cdot \left(\|C - T(x, y)\|^2 - E(x, y)\right) \tag{7}$$

This shows that the expected SSE *increases* if $\|C - T(x, y)\|^2 > E(x, y)$ (the shot color is worse than current expectation) and *decreases* otherwise.

3.3 Total Expected Improvement

Definition 3.4 (Expected Improvement)**.** The expected improvement from a shot is the decrease in total expected SSE:

$$\Delta = \sum_{(x,y)} \left(E(x, y) - E'(x, y)\right) = \sum_{(x,y)} p(x, y) \cdot \left(E(x, y) - \|C - T(x, y)\|^2\right) \tag{8}$$

where $p(x, y)$ is the paint probability at pixel (x, y).

The greedy strategy is to select, at each step, the shot (c_x, c_y, D, C) that maximizes Δ.

4 OPTIMAL COLOR SELECTION

4.1 The Color Optimization Problem

Given a fixed shot position (c_x, c_y) and radius D, what is the optimal color C^* that maximizes expected improvement?

Let $\mathcal{P} = \{(x, y) : p(x, y) > 0\}$ be the set of pixels with positive paint probability. We seek to maximize:

$$\Delta(C) = \sum_{(x,y)\in\mathcal{P}} p(x, y) \cdot \left(E(x, y) - \|C - T(x, y)\|^2\right) \tag{9}$$

4.2 Derivation of Optimal Color

Theorem 4.1 (Optimal Shot Color). *The optimal color $C^* = (r^*, g^*, b^*)$ that maximizes expected improvement is the probability-weighted average of target colors:*

$$C^* = \frac{\sum_{(x,y)} p(x, y) \cdot T(x, y)}{\sum_{(x,y)} p(x, y)} \tag{10}$$

where the sums are over all pixels with $p(x, y) > 0$.

Proof. We can decompose $\Delta(C)$ as:

$$\Delta(C) = \underbrace{\sum_{(x,y)} p(x, y) \cdot E(x, y)}_{A} - \underbrace{\sum_{(x,y)} p(x, y) \cdot \|C - T(x, y)\|^2}_{B(C)} \tag{11}$$

Since A is independent of C, maximizing $\Delta(C)$ is equivalent to minimizing $B(C)$. Expanding $B(C)$ using $\|C - T\|^2 = \|C\|^2 - 2\langle C, T\rangle + \|T\|^2$:

$$B(C) = \sum_{(x,y)} p(x, y) \cdot \|T(x, y)\|^2 - 2 \sum_{(x,y)} p(x, y) \cdot \langle C, T(x, y)\rangle + \sum_{(x,y)} p(x, y) \cdot \|C\|^2 \tag{12}$$

Let $W = \sum p(x, y)$ be the total weight, $\bar{T} = \frac{1}{W} \sum p(x, y) \cdot T(x, y)$ be the weighted mean, and $S = \sum p(x, y) \cdot \|T(x, y)\|^2$. Then:

$$B(C) = S - 2W\langle C, \bar{T}\rangle + W\|C\|^2 = W\left(\left\|C - \bar{T}\right\|^2 - \left\|\bar{T}\right\|^2\right) + S \tag{13}$$

This is minimized when $C = \bar{T}$, proving the theorem. □

4.3 Computing the Optimal Improvement

With the optimal color $C^* = \bar{T}$, the minimum value of $B(C)$ is:

$$B(C^*) = S - W\left\|\bar{T}\right\|^2 = \sum_{(x,y)} p(x, y) \cdot \|T(x, y)\|^2 - \frac{\left(\sum_{(x,y)} p(x, y) \cdot T(x, y)\right)^2}{\sum_{(x,y)} p(x, y)} \tag{14}$$

The maximum improvement is:

$$\Delta^* = A - B(C^*) = \sum_{(x,y)} p(x,y) \cdot E(x,y) - B(C^*) \tag{15}$$

4.4 Efficient Computation via Sufficient Statistics

For each candidate shot, we accumulate:

$$W = \sum_{(x,y)} p(x,y) \qquad \text{(total weight)} \tag{16}$$

$$\mathrm{S}_E = \sum_{(x,y)} p(x,y) \cdot E(x,y) \qquad \text{(weighted expected SSE)} \tag{17}$$

$$\mathrm{S}_R = \sum_{(x,y)} p(x,y) \cdot T_R(x,y) \qquad \text{(weighted red targets)} \tag{18}$$

$$\mathrm{S}_G = \sum_{(x,y)} p(x,y) \cdot T_G(x,y) \qquad \text{(weighted green targets)} \tag{19}$$

$$\mathrm{S}_B = \sum_{(x,y)} p(x,y) \cdot T_B(x,y) \qquad \text{(weighted blue targets)} \tag{20}$$

$$\mathrm{S}_N = \sum_{(x,y)} p(x,y) \cdot \|T(x,y)\|^2 \qquad \text{(weighted target norms)} \tag{21}$$

The optimal color is $(r^*, g^*, b^*) = (\mathrm{S}_R/W, \mathrm{S}_G/W, \mathrm{S}_B/W)$, rounded to integers in $[0, 255]$.

5 STAGE 1: COARSE GREEDY PLACEMENT

5.1 Motivation

Large-radius shots are efficient for covering broad regions with relatively uniform color. The coarse stage captures major image features—large color regions, dominant hues, and general structure—using shots with $D \in \{80, 40, 20\}$.

5.2 Grid-Based Candidate Selection

Instead of evaluating every pixel as a potential shot center (computationally prohibitive for large D), we use a sparse grid with spacing $G = 18$ pixels. This creates approximately $(H/G) \times (W/G)$ candidate positions.

At each grid cell (i, j), we evaluate shots with $D \in \{80, 40, 20\}$ and select the best (D, C) configuration.

5.3 Lazy Priority Queue with Stamping

Evaluating all candidates after each shot is expensive. We use a *lazy priority queue* with version stamping:

(1) Initialize: Evaluate all cells, insert into max-heap by improvement.
(2) Extract: Pop the top entry $(improvement, cellId, version)$.

(3) Validate: If the version doesn't match the cell's current version, the entry is stale; re-evaluate and re-insert.
(4) Apply: If valid and improvement > 0, apply the shot and mark affected cells as dirty.
(5) Repeat: Continue until budget exhausted or no positive improvement remains.

Definition 5.1 (Dirty Region)**.** When a shot is applied at (c_x, c_y) with radius D, all grid cells whose centers are within distance $D + D_{\max}$ of the shot center may have changed evaluations and are marked dirty.

The stamping mechanism uses a global stamp counter. When cells are marked dirty, their stamp is updated. A heap entry is valid only if its version matches the cell's current stamp.

5.4 Algorithm

Algorithm 1 Coarse Greedy Stage

```
Initialize expected SSE canvas from white
Build grid of candidate centers with spacing G
Evaluate all cells for best (D, C), insert into heap
while shots made < MAX_COARSE and heap not empty do
    (improvement, cellId, version) ← heap.pop()
    if version ≠ cellStamp[cellId] then
        continue                                   ▷ Stale entry
    end if
    if cellSeen[cellId] ≠ globalStamp for this cell then
        Re-evaluate cell, update version, re-insert
        continue
    end if
    if improvement ≤ 0 then
        break
    end if
    Apply shot, update expected SSE canvas
    Mark affected cells as dirty (increment global stamp)
    Re-evaluate current cell and re-insert
end while
```

6 STAGE 2: FINE GREEDY REFINEMENT

6.1 Motivation

After coarse placement, the canvas has good overall structure but lacks fine detail. The fine greedy stage uses small radii $D \in \{1, \ldots, 8\}$ at every pixel position to refine details.

6.2 Multi-Offset Optimization

A naive approach evaluates each $(D, position)$ pair separately. We optimize by precomputing a *multi-offset* structure that, for each offset $(\Delta x, \Delta y)$ from a potential center, stores all $(D, probability)$ pairs where that offset contributes.

Definition 6.1 (Multi-Offset Entry)**.** For offset $(\Delta x, \Delta y)$ with distance $d = \sqrt{\Delta x^2 + \Delta y^2}$, the multi-offset entry contains:

$$\{(D, p(d, D)) : D \in \{1, \ldots, D_{\max}\} \text{ and } p(d, D) > 0\} \tag{22}$$

When evaluating a center (c_x, c_y), we iterate over all multi-offset entries and accumulate sufficient statistics for all D values simultaneously in a single pass over affected pixels.

6.3 Block-Based Dirty Tracking

For fine shots with small D, maintaining per-pixel dirty flags is expensive. We partition the image into 4×4 blocks and track dirty blocks instead of individual pixels.

When a shot is applied, all blocks within distance $D + D_{\max}$ of the shot center are marked dirty. A pixel's cached evaluation is valid only if its containing block hasn't been dirtied since the last evaluation.

6.4 Algorithm Structure

The fine greedy stage follows the same lazy heap pattern as the coarse stage, but:

- Candidates are individual pixels, not grid cells
- Multiple D values are evaluated simultaneously per candidate
- Dirty tracking uses blocks instead of direct region enumeration

7 STAGE 3: REVERSE-SWEEP COLOR REFITTING

7.1 The Problem with Greedy Colors

During greedy placement, each shot's color is optimized assuming it will be the final painter of its affected pixels. However, later shots may overwrite some of these pixels, making the original color suboptimal.

7.2 Accounting for Overwrite Probability

Definition 7.1 (No-Later-Paint Probability)**.** For shot i and pixel (x, y), define $q_i(x, y)$ as the probability that no shot with index $j > i$ paints pixel (x, y):

$$q_i(x, y) = \prod_{j>i} (1 - p_j(x, y)) \tag{23}$$

where $p_j(x, y)$ is the probability that shot j paints pixel (x, y).

Definition 7.2 (Effective Weight)**.** The effective weight of shot i at pixel (x, y) is:

$$w_i(x, y) = p_i(x, y) \cdot q_i(x, y) \tag{24}$$

This is the probability that shot i both paints the pixel *and* is the final painter (not overwritten by later shots).

7.3 Optimal Color with Overwrite Awareness

Theorem 7.3 (Refit Optimal Color). *Given a fixed sequence of shot positions and radii, the optimal color for shot i that minimizes expected SSE is:*

$$C_i^* = \frac{\sum_{(x,y)} w_i(x,y) \cdot T(x,y)}{\sum_{(x,y)} w_i(x,y)} \tag{25}$$

Proof. The expected SSE contribution from shot i is:

$$\sum_{(x,y)} w_i(x,y) \cdot \|C_i - T(x,y)\|^2 \tag{26}$$

This is a weighted least-squares problem with the same form as Theorem 4.1, yielding the weighted mean as the optimal solution. □

7.4 Efficient Reverse Sweep

The key observation is that $q_i(x,y)$ can be computed incrementally by processing shots from last to first:

Algorithm 2 Reverse-Sweep Color Refitting

```
Initialize q(x, y) ← 1 for all pixels                ▷ No shots after the last one
for i = n − 1 down to 0 do
    Compute w_i(x, y) = p_i(x, y) · q(x, y) for all affected pixels
    Compute optimal color: C_i^* = (Σ w_i(x,y) · T(x,y)) / (Σ w_i(x,y))
    Update shot i's color to C_i^*
    Update: q(x, y) ← q(x, y) · (1 − p_i(x, y)) for all affected pixels
end for
```

This achieves $O(\text{total affected pixels})$ time complexity for the entire refit.

8 STAGE 4: ORDER-PRESERVING PRUNING

8.1 The Pruning Problem

We generate approximately 11,000 shots but can only output 10,000. The goal is to select the $K = 10{,}000$ most beneficial shots while *preserving their original temporal order*.

Order preservation is crucial because:

(1) The overwrite model means shot order affects final pixel colors
(2) Reordering would invalidate all our expected SSE calculations

8.2 Benefit Computation

Definition 8.1 (Shot Benefit)**.** The benefit of keeping shot i is:

$$\text{benefit}_i = \sum_{(x,y)} w_i(x,y) \cdot E_{\text{prefix}}(x,y) - \sum_{(x,y)} w_i(x,y) \cdot \|C_i - T(x,y)\|^2 \tag{27}$$

where:

- $w_i(x, y) = p_i(x, y) \cdot q_{i+1}(x, y)$ is the effective weight (probability shot i is final painter)
- $E_{\text{prefix}}(x, y)$ is the expected SSE at pixel (x, y) from all *kept* shots before i
- $q_{i+1}(x, y)$ is computed considering only *kept* shots after i

8.3 Iterative Selection

The challenge is that benefits depend on which other shots are kept, creating a circular dependency. We use iterative refinement:

Algorithm 3 Order-Preserving Pruning

```
Initialize keepMask ← all true
for iteration = 1 to MAX_ITER do
    Reverse pass: Compute w_i(x, y) using current keepMask
    Forward pass: Compute E_prefix(x, y) and benefits
    Sort shots by benefit (descending)
    keepMask ← top K shots
end for
return keepMask
```

8.4 Implementation Details

For efficiency, we precompute and cache:

- Per-shot lists of $(pixelIndex, weight)$ pairs
- Per-shot $\sum w_i(x, y) \cdot \|C_i - T(x, y)\|^2$ (weighted loss)

During the forward pass, we only need to look up $E_{\text{prefix}}(x, y)$ for precomputed pixel indices.

9 STAGE 5: $D = 1$ SHOT POSTPONEMENT

9.1 The Special Nature of $D = 1$ Shots

Shots with $D = 1$ have a unique property: they paint *only* the center pixel with probability 1 (since $d = 0 \leq 0.6 \cdot 1$ for the center).

Lemma 9.1. *A $D = 1$ shot at position (c_x, c_y) with color C:*

(1) Paints pixel (c_x, c_y) with probability 1
(2) Paints no other pixel (all other pixels have $d \geq 1 = D$)

This makes $D = 1$ shots *deterministic* and *completely local*.

9.2 Optimality of Postponement

Theorem 9.2 ($D = 1$ Postponement Optimality). *For any sequence containing $D = 1$ shots, moving all $D = 1$ shots to the end (while preserving their relative order) and setting each to the exact target color yields equal or better expected SSE.*

Proof sketch. Since $D = 1$ shots paint exactly one pixel with probability 1:

(1) A $D = 1$ shot at position (x, y) placed at the end sets $E(x, y) = 0$ if using target color
(2) Its effect cannot be overwritten (it's last)
(3) Moving it later cannot hurt—it either has the same effect or potentially improves by avoiding being overwritten

□

9.3 Refilling Strategy

After moving existing $D = 1$ shots to the end, we may have budget remaining. The optimal strategy is:

(1) Compute expected SSE after all non-$D = 1$ shots
(2) Select the k pixels with highest remaining expected SSE
(3) Add $D = 1$ shots at these positions with exact target colors

Algorithm 4 $D = 1$ Postponement and Refill

```
Separate shots into non-D = 1 and D = 1 lists
Apply all non-D = 1 shots to compute residual expected SSE
budget ← 10,000 − |non-D = 1 shots|
Select top budget pixels by expected SSE
for each selected pixel (x, y) do
    Add shot: (x, y, D = 1, T(x, y))
end for
```

10 STAGE 6: LOCAL GEOMETRY TUNING

10.1 Motivation

The greedy stages fix shot positions to grid points or pixel centers. Local tuning allows shots to "drift" to nearby better positions.

10.2 The Local Search Space

For each shot in the suffix (last 10,000 shots), we search a $3 \times 3 \times 3$ neighborhood in (x, y, D):

$$\mathcal{N}(s) = \{(x + \Delta x, y + \Delta y, D + \Delta D) : \Delta x, \Delta y, \Delta D \in \{-1, 0, +1\}\} \tag{28}$$

10.3 Evaluation with Context

For each candidate configuration, we need:

(1) **Prefix state:** Expected SSE at each pixel before this shot (from shots $0, \ldots, i-1$)
(2) **Suffix weights:** Probability of being overwritten by later shots (from shots $i + 1, \ldots, n - 1$)

We use local patches to cache the prefix state efficiently:

Definition 10.1 (Local Patch)**.** For shot i at position (c_x, c_y) with radius D, the local patch stores prefix expected SSE values for all pixels within distance $D + 2$ of the center (to cover all candidates in the $3 \times 3 \times 3$ neighborhood).

10.4 Algorithm

Algorithm 5 Local Geometry Tuning (Single Pass)

Forward pass: For each shot i to tune, snapshot prefix SSE onto local patch
Initialize $q(x, y) \leftarrow 1$ for all pixels
for $i = n - 1$ down to $suffixStart$ **do**
  if shot i should be tuned **then**
    Search: For each $(\Delta x, \Delta y, \Delta D) \in \{-1, 0, +1\}^3$:
      Evaluate candidate using patch and current q
      Track best improvement with deterministic tie-breaking
    Update shot i to best configuration
  end if
  Update: $q(x, y) \leftarrow q(x, y) \cdot (1 - p_i(x, y))$ using new configuration
end for

10.5 Deterministic Tie-Breaking

When multiple configurations have equal improvement (within floating-point tolerance $\epsilon = 10^{-12}$), we use a deterministic tie-breaking order:

(1) Prefer smaller $|\Delta D|$ (stability in radius)
(2) Prefer smaller $|\Delta x| + |\Delta y|$ (stability in position)
(3) Prefer smaller $|\Delta x|$
(4) Prefer smaller $|\Delta y|$
(5) Lexicographic order on $(\Delta D, \Delta x, \Delta y)$

This ensures reproducibility across runs.

10.6 Iterative Refinement

Local tuning is performed iteratively (up to 30 iterations within time budget):

(1) Tune all suffix shots
(2) Refit colors via reverse sweep
(3) Repeat until time limit

Each iteration allows shots to propagate improvements from neighbors.

11 IMPLEMENTATION OPTIMIZATIONS

11.1 Precomputed Offset Tables

For each radius $D \in \{1, \ldots, 100\}$ and each horizontal offset $\Delta x \in \{-D, \ldots, D\}$, we precompute the list of $(\Delta y, probability)$ pairs where probability is non-zero.

This structure enables:

- $O(\text{affected pixels})$ iteration per shot (vs. $O(D^2)$ full grid scan)

- Cache-friendly memory access patterns (row-by-row iteration)

11.2 Fast Path / Slow Path Split

For shots entirely within the image bounds (center at least D pixels from all edges), we use an optimized inner loop without bounds checking. Near-boundary shots use a slower path with explicit checks.

11.3 Floating-Point Precision

- Expected SSE values use `double` precision
- Total expected SSE uses `long double` for accuracy over many shots
- Probabilities use `float` to reduce memory bandwidth

11.4 Time Budget Management

The algorithm monitors elapsed time and terminates local tuning iterations when approaching the 10-second limit (with 0.5-second safety margin).

12 COMPLEXITY ANALYSIS

12.1 Space Complexity

- Target image: $O(HW)$
- Expected SSE canvas: $O(HW)$
- Offset tables: $O\left(\sum_{D=1}^{100} D^2\right) = O(D_{\max}^3)$
- Shot storage: $O(N)$ where $N \leq 11{,}000$

Total: $O(HW + D_{\max}^3)$

12.2 Time Complexity

Let $A_D = \pi D^2$ denote the number of pixels affected by a radius-D shot.

- **Coarse stage:** $O(n_{\text{coarse}} \cdot A_{D_{\text{coarse}}} \cdot \log(\text{grid cells}))$
- **Fine stage:** $O(n_{\text{fine}} \cdot A_{D_{\text{fine}}} \cdot \log(HW))$
- **Color refit:** $O\left(\sum_i A_{D_i}\right)$ per sweep
- **Pruning:** $O\left(iterations \cdot \sum_i A_{D_i}\right)$
- $D = 1$ **postponement:** $O(HW)$ for top-k selection
- **Local tuning:** $O\left(iterations \cdot n_{\text{suffix}} \cdot 27 \cdot A_{D_{\max}}\right)$

12.3 Practical Performance

On typical contest inputs ($H, W \approx 500$), the algorithm completes all stages well within the 10-second budget, spending roughly:

- 10% on coarse greedy
- 40% on fine greedy
- 10% on pruning and refitting
- 40% on local tuning iterations

13 CONCLUSION

We have presented a comprehensive solution to the probabilistic image reconstruction problem. The key contributions are:

(1) **Expected SSE framework:** Transforming a stochastic optimization problem into a tractable deterministic one via expected value analysis.
(2) **Closed-form optimal colors:** Deriving that the optimal shot color is always the probability-weighted mean of target colors.
(3) **Multi-scale greedy:** Combining coarse (large D) and fine (small D) greedy stages for efficient coverage.
(4) **Overwrite-aware refitting:** Accounting for later shots' overwrite probabilities when optimizing colors.
(5) $D = 1$ **postponement:** Exploiting the deterministic nature of single-pixel shots for optimal placement.
(6) **Local geometry tuning:** Fine-tuning shot parameters via systematic local search.

The algorithm achieves high-quality reconstructions by carefully balancing exploration (greedy shot selection) with exploitation (refitting and tuning), all within the strict time budget of competitive programming.

A DERIVATION DETAILS

A.1 Weighted Least Squares for Color Optimization

The objective function for color optimization is:

$$B(C) = \sum_{(x,y)} p(x,y) \cdot \|C - T(x,y)\|^2 \tag{29}$$

Expanding the norm:

$$B(C) = \sum p \cdot (\|C\|^2 - 2\langle C, T\rangle + \|T\|^2) \tag{30}$$

$$= W\,\|C\|^2 - 2\langle C, \sum p \cdot T\rangle + \sum p \cdot \|T\|^2 \tag{31}$$

Taking the gradient with respect to C:

$$\nabla_C B = 2W \cdot C - 2\sum p \cdot T = 0 \tag{32}$$

Solving:

$$C^* = \frac{\sum p \cdot T}{W} \tag{33}$$

A.2 Minimum Value of $B(C)$

Substituting $C^* = \bar{T} = \frac{\sum p \cdot T}{W}$:

$$B(C^*) = W\left\|\bar{T}\right\|^2 - 2\langle \bar{T}, W\bar{T}\rangle + \sum p \cdot \|T\|^2 \tag{34}$$

$$= W\left\|\bar{T}\right\|^2 - 2W\left\|\bar{T}\right\|^2 + \sum p \cdot \|T\|^2 \tag{35}$$

$$= \sum p \cdot \|T\|^2 - W\left\|\bar{T}\right\|^2 \tag{36}$$

This is the variance formula: $B(C^*) = \sum p \cdot \left\| T - \bar{T} \right\|^2$.