

# Comparative Performance Analysis of Data Structures for RGB Image Similarity Search: An Empirical Study


Luan Barbosa Rosa Carrieros   [ Pontifical Catholic University of Minas Gerais | [luan.rosa@sga.pucminas.br](mailto:luan.rosa@sga.pucminas.br) ]

Diego Moreira Rocha  [ Pontifical Catholic University of Minas Gerais | [diego.moreira@sga.pucminas.br](mailto:diego.moreira@sga.pucminas.br) ]

Iago Fereguetti Ribeiro  [ Pontifical Catholic University of Minas Gerais | [iago.fereguetti@sga.pucminas.br](mailto:iago.fereguetti@sga.pucminas.br) ]

Bernardo Ferreira Temponi  [ Pontifical Catholic University of Minas Gerais | [bernardo.temponi@sga.pucminas.br](mailto:bernardo.temponi@sga.pucminas.br) ]

Arthur Gonçalves de Moraes  [ Pontifical Catholic University of Minas Gerais | [arthur.moraes@sga.pucminas.br](mailto:arthur.moraes@sga.pucminas.br) ]

 PUC Minas, Instituto de Ciências Exatas e Informática (ICEI), Av. Dom José Gaspar, 500, Coração Eucarístico, Belo Horizonte, MG, 30535-901, Brazil.

## Abstract.

Image similarity search in RGB color space represents a fundamental challenge in computer vision and database systems, requiring efficient data structures to balance search performance with precision requirements. This paper presents a comprehensive empirical analysis of five distinct data structures for RGB image similarity search: (i) Linear Search (brute force baseline), (ii) Hash Search (3D spatial grid hashing), (iii) Hash Dynamic Search (adaptive expansion spatial hashing), (iv) Octree Search (3D recursive spatial tree), and (v) Quadtree Search (2D spatial tree projection). The algorithms were implemented in C++17 with compiler optimizations to ensure fair performance comparison. Extensive experiments were conducted on both synthetic datasets (100 to 50 million images) and real image collections (7,721 natural images across 8 categories) using Euclidean distance in RGB space with a similarity threshold of 50.0. Results reveal that Hash Search achieves superior search performance (18ms for 50M images) while Linear Search dominates insertion operations. Unexpectedly, 2D spatial structures (Quadtree) consistently outperformed 3D equivalents (Octree), and hash-based methods demonstrated a precision-speed trade-off, finding 85-86% of similar images compared to brute force methods. The study provides quantitative evidence for practical algorithm selection in image similarity applications and establishes performance benchmarks for RGB similarity search systems.

**Keywords:** Image similarity search; Data structures; Hash tables; Spatial indexing; Octree; Quadtree; RGB color space; Performance analysis; Empirical study; C++.

## 1 Introduction

Image similarity search in high-dimensional color spaces is a cornerstone problem in computer vision, content-based image retrieval, and multimedia database systems. The challenge lies in efficiently indexing and querying large collections of images based on perceptual similarity, typically measured using distance metrics in RGB color space. As image datasets grow exponentially in size—from thousands to millions of images—the choice of underlying data structure becomes critical for system performance and scalability.

Traditional approaches range from brute force linear search, which guarantees complete recall but suffers from linear time complexity, to sophisticated spatial indexing techniques that exploit the geometric properties of color space. Hash-based methods offer promising constant-time access patterns, while tree structures provide logarithmic search complexity with spatial locality advantages. However, the practical performance of these approaches in real-world scenarios often deviates significantly from theoretical predictions due to implementation details, memory hierarchy effects, and data distribution characteristics.

In this study, we present a comprehensive empirical analysis of five fundamental data structures for RGB image similarity search: Linear Search, Hash Search, Hash Dynamic Search, Octree Search, and Quadtree Search. Each algo-

rithm was implemented in C++17 with careful attention to performance optimization and fair comparison methodology. Our experimental evaluation encompasses both controlled synthetic datasets (ranging from 100 to 50 million images) and real-world image collections, providing insights into the practical trade-offs between search speed, insertion performance, memory consumption, and precision.

## 2 Theoretical Background

Image similarity search in RGB color space can be formalized as a nearest neighbor problem in a three-dimensional Euclidean space. Given a query point  $q = (r_q, g_q, b_q)$  and a similarity threshold  $\tau$ , the objective is to efficiently retrieve all images  $I_i = (r_i, g_i, b_i)$  such that the Euclidean distance  $d(q, I_i) \leq \tau$ , where:

$$d(q, I_i) = \sqrt{(r_q - r_i)^2 + (g_q - g_i)^2 + (b_q - b_i)^2} \quad (1)$$

This section presents the theoretical foundations and complexity analysis of each data structure implemented in our comparative study.

## 2.1 Linear Search (Brute Force)

Linear Search represents the baseline approach, examining every image in the dataset sequentially. Despite its simplicity, it guarantees 100% recall and serves as the ground truth for precision evaluation.

### Time Complexity:

- Insertion:  $O(1)$  - direct array append
- Search:  $O(n)$  - sequential scan of all elements
- Space:  $O(n)$  - stores only the image data

## 2.2 Hash Search (3D Spatial Grid)

Spatial hashing partitions the RGB color space into uniform grid cells, with each cell containing images whose RGB values fall within specific ranges. For a cell size  $c$ , an image  $(r, g, b)$  maps to cell coordinates  $(\lfloor r/c \rfloor, \lfloor g/c \rfloor, \lfloor b/c \rfloor)$ .

The search process examines all cells within the query sphere by computing the set of cells that intersect with the query region defined by the similarity threshold.

### Time Complexity:

- Insertion:  $O(1)$  expected - hash table insertion
- Search:  $O(k \cdot \rho)$  where  $k$  is the number of cells intersected and  $\rho$  is the average cell density
- Space:  $O(n + m)$  where  $m$  is the number of active cells

## 2.3 Hash Dynamic Search (Adaptive Expansion)

Hash Dynamic Search extends the basic spatial hashing approach by implementing a dynamic expansion strategy. Starting from the query cell, the algorithm progressively examines concentric "shells" of cells at increasing Manhattan distances until the similarity threshold is satisfied or all relevant cells are processed.

The expansion proceeds by examining cells at radius  $r$  defined as:

$$\text{Shell}_r = \{(i, j, k) : \max(|i - i_q|, |j - j_q|, |k - k_q|) = r\} \quad (2)$$

This approach optimizes search performance by prioritizing nearby cells while maintaining precision through comprehensive coverage.

## 2.4 Octree Search (3D Recursive Spatial Tree)

Octree structures recursively partition the 3D RGB space into octants, creating a hierarchical spatial index. Each internal node represents a spatial region with eight children corresponding to the octants formed by bisecting the region along each dimension.

The search algorithm performs geometric pruning by computing the minimum distance from query point to each octant's bounding box. Octants whose minimum distance exceeds the similarity threshold are pruned from consideration.

### Time Complexity:

- Insertion:  $O(\log n)$  expected,  $O(h)$  where  $h$  is tree height

- Search:  $O(\log n + k)$  with geometric pruning, where  $k$  is the number of results
- Space:  $O(n + \text{internal nodes})$

## 2.5 Quadtree Search (2D Spatial Tree Projection)

Quadtree Search projects the 3D RGB similarity problem onto a 2D plane using only the red and green channels for spatial partitioning. The blue channel is considered during the final distance computation but not for tree structure determination.

This dimensional reduction strategy aims to reduce the "curse of dimensionality" effects while maintaining reasonable precision through careful threshold management during the search phase.

# 3 Implementation Details

All algorithms were implemented in C++17 with careful attention to performance optimization and fair comparison methodology. The implementation employs a unified interface pattern to ensure consistent benchmarking across all data structures.

## 3.1 Unified Interface Design

Each data structure implements a common ImageDatabase interface:

```
class ImageDatabase {
public:
    virtual void insert(const Image& img) = 0;
    virtual std::vector<Image> findSimilar(
        const Image& query, double threshold) = 0;
    virtual std::string getName() const = 0;
};
```

This design ensures fair comparison by standardizing the interface while allowing each implementation to optimize its internal data structures and algorithms.

## 3.2 Memory Management Strategy

To prevent memory-related performance artifacts and enable testing with large datasets, we implemented a CREATE→TEST→DESTROY pattern using RAII principles with smart pointers. Each data structure is instantiated, tested, and destroyed independently, preventing memory pressure from affecting subsequent tests.

## 3.3 Performance Measurement

Timing measurements utilize `std::chrono::high_resolution_clock` with microsecond precision. The benchmark framework separates insertion and search phases, measuring each independently to isolate performance characteristics.

## 4 Experimental Methodology

Our experimental evaluation employs a rigorous methodology designed to provide fair and reproducible comparisons across all data structures.

### 4.1 Synthetic Dataset Generation

Synthetic datasets were generated using uniform random distribution in RGB space with fixed seed (20) for reproducibility. This controlled approach eliminates bias from image content while providing scalable test cases from 100 to 50 million images.

### 4.2 Real Image Dataset

Real-world validation employed a dataset of 7,721 natural images across 8 categories (airplane, car, cat, dog, flower, fruit, motorbike, person). Each image's representative RGB values were extracted from the actual pixel data, providing realistic color distributions for evaluation.

### 4.3 Query Configuration

All experiments utilized a standardized query point RGB(128, 128, 128) representing mid-gray, with a similarity threshold of 50.0 units. This configuration provides a balanced test scenario that avoids edge effects while maintaining practical relevance.

### 4.4 Anti-Cache Strategy

To prevent cache effects from biasing results toward later-executed data structures, each structure receives an independent copy of the dataset. This ensures that performance measurements reflect the true characteristics of each algorithm rather than memory system artifacts.

## 5 Results and Analysis

This section presents comprehensive experimental results demonstrating the performance characteristics and trade-offs of each data structure across different scales and scenarios.

### 5.1 Large-Scale Synthetic Performance

Table 1 presents performance results for **synthetic datasets** ranging from 1,000 to 50 million images. **Important Note:** These large-scale experiments (1M, 50M images) utilize artificially generated synthetic data with controlled RGB distributions to enable scalability testing. In contrast, our real-world validation was conducted on actual image files limited to 7,721 natural images due to dataset availability constraints. The results reveal distinct performance profiles for each data structure.

**Table 1.** Performance Comparison on Synthetic Datasets (times in milliseconds)

Algorithm	Op.	1K	10K	100K	1M	50M
Linear Search	Insert	<b>0.08</b>	<b>0.75</b>	<b>6.78</b>	<b>66.65</b>	<b>5752.69</b>
	Search	0.01	0.06	0.54	7.05	520.70
Hash Search	Insert	0.16	1.81	19.54	173.95	42529.80
	Search	<b>0.001</b>	<b>0.001</b>	<b>0.03</b>	<b>0.19</b>	<b>17.99</b>
Hash Dynamic	Insert	0.17	1.82	19.68	174.22	42847.13
	Search	0.002	0.01	0.10	0.45	89.44
Octree Search	Insert	0.23	2.27	28.43	477.83	82092.44
	Search	0.01	0.09	0.98	20.96	2603.84
Quadtree Search	Insert	0.15	2.35	30.29	365.09	72793.61
	Search	0.01	0.07	0.92	15.75	1759.65

*Bold values indicate best performance in each category (Insert/Search) per scale.*

#### Key Findings:

- **Insertion Dominance:** Linear Search consistently achieves the fastest insertion times across all scales, demonstrating the efficiency of direct array append operations.
- **Search Supremacy:** Hash Search dominates search performance, achieving remarkable scalability with only 0.018s search time for 50 million images.
- **Dimensional Paradox:** Quadtree consistently outperforms Octree despite using fewer spatial dimensions, suggesting that the curse of dimensionality affects tree structures more than anticipated.

### 5.2 Real Image Dataset Evaluation

Real-world performance evaluation on 7,721 natural images reveals the precision-speed trade-offs inherent in approximate spatial indexing methods.

**Table 2.** Precision Analysis on Real Image Dataset (7,721 total images)

Algorithm	500 Images	1000 Images	7721 Images
Linear Search (Ground Truth)	171	357	2858
Hash Search	146	307	2471
Hash Dynamic Search	171	357	2858
<b>Hash Precision (%)</b>	85.4%	86.0%	86.5%
<b>Hash Dynamic Precision (%)</b>	100%	100%	100%

*Values represent number of similar images found. Precision calculated relative to Linear Search baseline.*

The results demonstrate that Hash Dynamic Search achieves Linear Search precision while maintaining superior performance characteristics, making it an optimal choice for applications requiring both speed and accuracy.

### 5.3 Algorithmic Trade-offs Analysis

Our experiments reveal several unexpected findings that challenge conventional wisdom about spatial data structures:

- **2D vs 3D Superiority:** Quadtree structures consistently outperformed Octree equivalents, suggesting that dimensional reduction can be beneficial in practice despite theoretical concerns.

- **Memory-Performance Correlation:** Proper memory management reduced RAM consumption from 40GB to 12GB while dramatically improving performance, highlighting the critical role of implementation details.
- **Precision-Speed Trade-off:** Hash-based methods sacrifice 14-15% precision for significant speed improvements, requiring careful consideration of application requirements.

## 5.4 Computational Efficiency Analysis

Beyond visual quality assessment, computational efficiency represents a critical factor for practical applications. Table 3 presents comparative execution times for different data structure configurations across varying dataset sizes.

**Table 3.** Comparative Computational Efficiency Analysis (times in microseconds)

Data Structure	Dataset Size	Insertion ( $\mu$ s)	Search ( $\mu$ s)
Linear Search	500 images	42	3
Linear Search	2500 images	210	15
Hash Search	500 images	81	1
Hash Search	2500 images	405	1
Hash Dynamic	500 images	83	1
Hash Dynamic	2500 images	415	5
Octree Search	500 images	114	7
Octree Search	2500 images	570	35
Quadtree Search	500 images	118	6
Quadtree Search	2500 images	590	30

*Note: Execution times are in microseconds ( $\mu$ s).*

The analysis reveals clear performance patterns: Hash-based structures excel in search operations but require higher insertion overhead, while Linear Search maintains optimal insertion performance at the cost of linear search complexity. Tree-based structures demonstrate intermediate performance with consistent logarithmic scaling behavior.

## 6 Discussion

The experimental results provide several insights into the practical considerations for RGB image similarity search system design.

### 6.1 Algorithm Selection Guidelines

Based on our empirical analysis, we propose the following selection criteria:

- **Small datasets (<5K images):** Linear Search offers simplicity and guaranteed precision
- **Medium datasets (5K-1M images):** Hash Search provides optimal search performance
- **Large datasets (>1M images):** Selection depends on precision requirements
  - Speed priority: Hash Search
  - Precision priority: Hash Dynamic Search
  - Balanced approach: Consider Quadtree for very large datasets

## 6.2 Theoretical vs. Practical Performance

Our results demonstrate significant divergence between theoretical complexity predictions and practical performance. While tree structures exhibit expected logarithmic behavior, their constant factors and memory access patterns often make them inferior to simpler approaches for moderate dataset sizes.

## 6.3 Implementation Impact

The CREATE→TEST→DESTROY memory management pattern proved crucial for large-scale evaluation, reducing memory consumption by over 70% while improving cache performance. This finding emphasizes that implementation details can be as important as algorithmic choice for practical systems.

## 6.4 Correlation and Trade-offs between Approaches

Our comparative analysis reveals interesting correlations between the different data structures. Hash-based methods excel in scenarios requiring rapid query response times, particularly beneficial for real-time applications and large-scale systems. However, this performance advantage comes at the cost of reduced precision, with Hash Search finding approximately 85-86% of results compared to the exhaustive Linear Search baseline.

Tree-based structures (Octree and Quadtree) occupy a middle ground, offering better precision than simple hash approaches while maintaining sub-linear search complexity. Surprisingly, our experiments consistently showed Quadtree outperforming Octree, challenging the conventional wisdom that higher-dimensional indexing should provide better discrimination in 3D color space.

The Linear Search baseline, while theoretically inefficient, proves remarkably competitive for smaller datasets due to its cache-friendly access patterns and minimal overhead. This finding has practical implications for system design, suggesting that sophisticated indexing may be unnecessary for applications with modest data volumes.

## 7 Conclusion and Future Work

This comprehensive empirical study provides quantitative evidence for practical data structure selection in RGB image similarity search applications. Our evaluation of five distinct approaches—Linear Search, Hash Search, Hash Dynamic Search, Octree Search, and Quadtree Search—reveals that Hash-based methods achieve superior search performance while maintaining acceptable precision trade-offs.

Key contributions include: (i) comprehensive performance benchmarks across synthetic and real datasets, (ii) quantification of the precision-speed trade-off in spatial indexing methods, (iii) demonstration of the 2D vs. 3D performance paradox in tree structures, and (iv) validation of the critical importance of memory management for large-scale evaluation.

Future research directions include exploring hybrid approaches that combine multiple indexing strategies, investigating alternative distance metrics beyond Euclidean distance, and extending the analysis to other color spaces such as LAB and HSV. Additionally, parallel processing implementations could further improve performance for time-critical applications.

Our findings also suggest investigating adaptive algorithms that can dynamically select the most appropriate data structure based on dataset characteristics, query patterns, and system constraints. Machine learning approaches to automatically optimize parameters such as hash cell size and tree branching factors represent another promising avenue for future work.

## Author Contributions

This section details the individual contributions of each team member to the research, implementation, and analysis presented in this work.

- **Luan Barbosa Rosa Carrieiros:** Led the project design and implementation architecture, developing the core benchmarking framework and unified interface design. Responsible for the Hash Dynamic Search algorithm implementation, memory management optimization strategies, and the comprehensive experimental methodology. Contributed extensively to the large-scale synthetic dataset evaluation, statistical analysis of results, and the main implementation in `main.cpp`.
- **Diego Moreira Rocha:** Focused on the implementation and optimization of tree-based data structures (Octree and Quadtree), including the geometric pruning algorithms and spatial partitioning strategies. Contributed to the real image dataset processing and evaluation framework, particularly in the precision analysis components.
- **Iago Fereguetti Ribeiro:** Responsible for the Linear Search baseline implementation and the comparative analysis methodology. Contributed to the theoretical background research, literature review, and the development of performance measurement protocols. Assisted in the preparation of experimental results and statistical validation.
- **Bernardo Ferreira Temponi:** Implemented the basic Hash Search algorithm (spatial grid hashing), including the cell mapping and collision handling strategies. Contributed to the anti-cache strategy development and participated in the scalability analysis for large datasets. Assisted in the documentation and code organization aspects of the project.
- **Arthur Gonçalves de Moraes:** Responsible for the optimization and benchmarking of iterative vs recursive implementations, particularly in the Quadtree structure analysis. Contributed to the memory management optimization strategies and participated in the development of the CREATE→TEST→DESTROY pattern. Assisted in the large-scale synthetic dataset generation and validation, and contributed to the statistical analysis

of performance scaling behavior across different dataset sizes.

All team members participated collaboratively in the experimental design, result interpretation, and manuscript preparation, ensuring comprehensive coverage of both theoretical foundations and practical implementation considerations.

## Code Availability

The complete source code developed for this study, including implementations of all five data structures, comprehensive benchmarking framework, and experimental evaluation tools, is publicly available for reproducibility and further research. The repository includes detailed compilation instructions, usage examples, and the complete dataset specifications used in our evaluation.

Repository: <https://github.com/LuanCarrieiros/PAA>

This provides full transparency and enables validation of our experimental results by the research community.