Hybrid ECS Scene-Graph Architecture optimized for Structural Expressiveness

Bălăban Iosua-Patrik

I. Abstract

Modern game engines must process hundreds of thousands of entities every frame while maintaining flexible hierarchical scene organization. Classical Entity Component Systems (ECS) offer excellent cache locality and performance but lack natural hierarchical structure, while traditional scene graphs provide hierarchy at the cost of severe cache inefficiency due to pointer-based traversal. This article investigates the performance trade-offs between a basic ECS, a traditional scene graph, and a hybrid ECS scene-graph architecture. The proposed hybrid model preserves hierarchical organization while storing entities in contiguous memory chunks to reduce cache misses. Experimental results on a synthetic dataset of 131,072 entities demonstrate that the hybrid approach achieves performance very close to a pure ECS and significantly outperforms a traditional scene graph. These findings suggest that the hybrid architecture offers a practical compromise between performance and structural flexibility in real-time systems.

II. Classification

ACM Computing Classification System (CCS):

- Software and its engineering, Software organization and properties, Software system structures, Software architectures
- Computing methodologies, Parallel computing methodologies, Parallel algorithms

AMS Subject Classification:

- 68M20 Performance evaluation; benchmarking
- 68N19 Other programming techniques
- 68M14 Distributed systems
- 68M99 None of the above, but in this section

III. Introduction

3.1 Background and Motivation

Real-time applications such as game engines and simulations must update and traverse large numbers of entities under strict performance constraints. Memory access patterns and cache behavior have become dominant factors in performance on modern CPUs. As a result, data-oriented design techniques, particularly Entity Component Systems (ECS), have gained widespread adoption.

Despite their performance advantages, classical ECS architectures struggle to represent hierarchical spatial relationships, which are naturally modeled using scene graphs. Scene graphs,

however, rely heavily on pointer-based structures, leading to fragmented memory layouts and frequent cache misses.

3.2 Problem Statement

There is a fundamental tension between: Performance, achieved through contiguous memory layouts and predictable access patterns (ECS), and Structural expressiveness, achieved through hierarchical scene graphs. Existing approaches often sacrifice one of these aspects. This article addresses the problem of combining hierarchical scene representation with cache-efficient entity processing.

3.3 Related Work and Open Problems

Previous work on ECS frameworks such as EnTT, Flecs, Unity DOTS, and Bevy ECS demonstrates the effectiveness of contiguous storage and Structure-of-Arrays (SoA) layouts. Conversely, mainstream engines like Unity, Unreal Engine, and Godot rely on hierarchical scene graphs, which suffer from poor cache locality.

The unresolved challenge is designing a system that preserves hierarchy without reintroducing the pointer-chasing overhead that ECS was designed to eliminate.

3.4 Research Questions

This work aims to answer the following questions:

- How large is the performance penalty of a traditional scene graph compared to ECS?
- Can a hybrid ECS–scene-graph model approach ECS-level performance?
- What are the dominant costs in each architectural model?

3.5 Contributions

The main contributions of this paper are:

- A hybrid ECS scene-graph architecture that clusters entities in contiguous memory per node.
- A mathematical cost model explaining the observed performance behavior.
- An experimental evaluation demonstrating near-ECS performance with hierarchical support.

IV. Original Proposed Approach

4.1 Overview of the Hybrid Architecture

The proposed architecture combines:

- A hierarchical scene graph composed of nodes connected by parent–child relationships.
- Local ECS-style contiguous arrays of entities stored within each node.

This design ensures that most computation occurs over cache-friendly data, while pointer traversal is limited to node transitions rather than per-entity access.

4.2 Data Models Compared

The study evaluates three models:

- Basic ECS: Entities stored in contiguous arrays with linear iteration.

- Scene Graph: Pointer-based tree with individual nodes scattered in memory.
- Hybrid ECS Scene Graph: Pointer-based hierarchy with contiguous entity chunks per node.

4.3 Mathematical Performance Models

4.3.1 Basic ECS Model

In a classical ECS, iteration is sequential:

$$\mathbf{T_{ECS}(N) = N \cdot C_{seq}}$$

Where:

N = number of entities

C_{seq} = amortized cost of sequential access (includes: L1 hits + prefetching + CPU pipeline efficiency)

Due to prefetching and low cache miss rates, C_{seq} is nearly constant.

4.3.2 Scene Graph Model

Scene graph traversal involves unpredictable pointer dereferencing:

$$\mathbf{T_{graph}(N) = \sum_{i=1}^{N} (C_{ptr}(i) + C_{miss}(i))}$$

Where:

$C_{ptr}(i)$ = cost of following a pointer (loading the address from memory)

$C_{miss}(i)$ = cost of cache misses caused by pointer chasing

Cache misses dominate execution time, making pointer chasing the primary bottleneck.

4.3.3 Hybrid Model

The hybrid model separates costs into tree traversal and chunk processing:

$$\mathbf{T_{hybrid}(N,k) = TreeCost(k) + ChunkCost(N)}$$

Where:

1. N = total number of entities
2. k = number of graph nodes
3. TreeCost(k): overhead from the graph itself (pointer jumps)
4. ChunkCost(N): work done inside ECS-style arrays (fast loops)

Since $k \leq N$, the dominant cost remains sequential iteration, yielding near-ECS performance.

4.4 Original Aspects

Unlike prior approaches, this architecture limits pointer chasing to structural navigation while ensuring that per-entity processing remains cache-efficient. This clear separation of slow and fast paths is the key original insight of the work.

V. Modeling the Experimental Component

5.1 Methodology

All three architectures were implemented in the Zig programming language to maintain explicit control over memory layout and eliminate runtime overhead. Identical entities and workloads were used across all experiments.

5.2 Experimental Setup

- Number of entities: 131,072 (2^{17})
- Operation: Full traversal and execution of an identical work function per entity
- Metric: Execution time in microseconds

5.3 Experimental Results

Architecture	Execution Time (µs)
Basic ECS	1,389
Scene Graph	23,215
Hybrid	1,487

OS: Arch Linux x86_64
Host: ASUS TUF Gaming A15 FA507RC_FA507RC 1.0
Kernel: 6.17.9-arch1-1.1-g14
CPU: AMD Ryzen 7 6800H (16) @ 4.78 Ghz.
Memory: 15225MiB

5.4 Interpretation

The scene graph is approximately 20 times slower than ECS due to cache misses and pointer chasing. The hybrid model performs within a small margin of ECS, confirming that most work occurs on contiguous memory blocks.

VI. Results and Conclusions

6.1 Answering the Research Questions

- RQ1: Scene graphs incur a significant performance penalty due to memory access patterns.
- RQ2: The hybrid model achieves performance nearly identical to ECS.
- RQ3: Pointer chasing dominates scene graph cost, while sequential iteration dominates ECS and hybrid systems.

6.2 Conclusions

The proposed hybrid ECS scene-graph architecture successfully reconciles hierarchical structure with high-performance entity processing. By clustering entities in contiguous memory within each node, the system avoids the traditional pitfalls of scene graphs while preserving their expressive power.

6.3 Validity and Limitations

The results align with known CPU cache behavior. However, experiments were conducted on synthetic data; real-world workloads may introduce additional complexity.

6.4 Future Work

Future research directions include:

- Scaling experiments to millions of entities,
- Evaluating multi-threaded and SIMD execution,
- Integrating the hybrid model into a full rendering pipeline.

VII. Bibliography

1. Papagiannakis, G., et al. (2023). *A computational entity-component-system in a scene-graph (ECSS)*. (arXiv:2302.07691)
2. Choparinov, E. D. (2024). A Graph-Based Approach To Concurrent ECS Design. Thesis.
3. Härkönen, T. (2019). Advantages and Implementation of ECS. (Bachelor of Science Thesis)
4. Hatledal, L., et al. (2020). Vico: An Entity-Component-System Based Co-simulation Framework
5. Akenine-Möller, T., et al. (2019). Real-Time Rendering (4th ed.).
6. Gregory, J. (2021). Game Engine Architecture (3rd ed.).
7. Mike Acton, Data-Oriented Design and C++, GDC 2014.