ECS Back and Forth

Michele Caini

skypjack

June 15, 2019

Italian C++ Community

Released under CC BY-SA 4.0
Entity–Component–System (ECS) is an architectural pattern.

It favors composition over inheritance and sacrifices encapsulation.
Entity-Component-System (ECS) offers better code organization and higher performance.

but

It is not the Holy Grail of game development.
A Big Array to rule them all

Entity identifiers are **indexes**, bitsets are component masks.

More **holes**, more jumps, more wasted memory, less performance.
Holes, holes everywhere

Iterate **bitmasks**, use entities to get components when needed.

Components are only *apparently* tightly packed, in fact they are not.
Pros and Cons

The big array is good enough for small games:

- Straightforward to implement and to maintain.
- Best performance on construction/destruction of components.
- Pretty good performance when arrays of components are dense.
- Too much memory is wasted in real world cases.
- Holes defeat the purpose of keeping instances tightly packed.
- We don’t know what entities own what components.

It can be refined to match the requirements of medium (?) sized games.
Archetypes (the easy version)

- Entities are moved between archetypes.
- More combinations means higher fragmentation.
- Multithreading friendly (with block-based archetypes).
Fragmentation: yay or nay?

- Components are only tightly packed per archetype.
- **Fragmentation cannot be any way worse than this.**
- Cache or search archetypes matched with queries.
Pros and Cons

Well suited when performance matters:

- Really good performance both on single and multiple components.
- Multithreading is straightforward to achieve in some cases.
- Best performance on bulk creation of entities and components.
- Assigning and removing components is intrinsically slow.
- Fragmentation can affect performance to an extent.
- Some operations are not supported out-of-the-box (eg sorting).

It can be refined to increase even further benefits and performance.
They call me Packed Array

Lookup, insertion, deletion, ... complexity is $O(1)$.

Iteration is $O(N)$ and the dense array is tightly packed.
They call me Packed Array

Lookup, insertion, deletion, ... **complexity** is $O(1)$.

**Iteration** is $O(N)$ and the dense array is tightly packed.
They call me Packed Array

Lookup, insertion, deletion, ... complexity is \( O(1) \).

\[
\begin{array}{cccc}
2 & 0 & 3 & 1 \\
\end{array}
\]

---

Iteration is \( O(N) \) and the dense array is tightly packed.
They call me Packed Array

Lookup, insertion, deletion, ... **complexity** is $O(1)$.

**Iteration** is $O(N)$ and the dense array is tightly packed.
A real world example: EnTT

A customized **sparse set** is used for the pools of components.

Multiple access patterns supported, from **perfect SoA** to fully random.
Pros and Cons

Well suited when performance matters:

- Grouping functionalities can reach outstanding performance.
- Best performance when it comes to iterating single components.
- Multithreading friendly, not necessarily built-in.
- Users **must** know what are their data to get the best.
- Users **must** know what are their critical paths and what are not.
- Indirection can affect performance to an extent in some cases.

It can be refined to reduce or even eliminate indirection in most cases.
Are they in the same ballpark?

- Know you game/software.
  - The big array plays in a different (lower) league.
  - Archetypes vs Sparse sets: 1M of elements, differences of $0.1N$ ms.

- Almost static vs dynamic entities.
  - Archetypes for low level systems (eg rendering).
  - Sparse sets for high level systems (eg gameplay).
  - Both are just fine for going full-ECS.

- Performance on construction/destruction matters.
  - Archetypes: many batch creations, few assignments/deletions.
  - Sparse sets: components to the rescue (eg messaging system).

- Interested in how things are laid out?
  - Archetypes offer many small groups for known patterns.
  - Sparse sets offer always a $(T\ast, \text{size})$ couple.
EnTT is a C++ framework mainly known for its ECS model.

Some things you can spot here and there if you pay attention:

- Type erasure: pools for components, signals, and so on.
- SFINAE (Substitution Failure Is Not An Error): any file of your choice.
- CRTP (Curiously Recurring Template Pattern): emitter class.
- Tag dispatching: process and scheduler classes.
- Type traits: named types to make EnTT work across boundaries.
- Small object optimization: meta_any class.

And much, much more...
Questions?

Italian C++ Conference 2019

++it Community

June 15, Milan
References

Links

- ECS back and forth series
  - Introduction
  - Where are my entities?
  - Sparse sets and grouping functionalities
  - Why you don't need to store deleted entities
  - To be continued...
- EnTT - Gaming meets modern C++
- EntityX - A fast, type-safe C++ Entity-Component system
- deco - Prototype data-oriented ECS
- Unity DOTS - Data-Oriented Technology Stack