Improving The Entity Component System

Paul O'Callaghan

Institute of Technology Carlow

Introduction

This research looked at the answering the question; Can we improve the Entity Component System by dropping the need for Entities?

The reason for looking at this research stems from the fact that the games industry is growing at a rapid pace requiring more features to stand out in an ever-growing market.

The need for more features comes with the need for faster look up and iteration times of classes while having a way to easily manage them in code. One way in helping manage the code base is the design pattern known as **Entity Component System**.

This helps decouple code and remove dependencies and interactions between code. In an ideal situation a change to one component will have little to no impact on every other component created.

Entities are used as component containers and this has an overhead for lookups during an iteration, as the types need to be compared and then checked whether they have said component before performing the calculation.

In this research we wanted to see how would the design pattern fare without Entities. In effect becoming a Component System. Design on a single header library was performed to create the Dropped-Entity Component System or DECS for short

Terms

Entity Component System (ECS) – Standard pattern used in the games industry. Entities are id containers holding components which are data classes(or a struct), with Systems performing the calculations on the components and entities that are active. The library used for tests was an open-sourced version created by Sam Bloomberg. https://github.com/redxdev/ECS

Dropped Entity Component System (DECS) – An amended design of ECS. It removes the need for entities as containers also known as an integer based ECS model.

1. DECS Tests – memory usage during use of DECS implementation.

2. ECS Tests – memory usage during use of standard ECS implementation.

Methods

For this research we needed quantitative data. We were measuring the speeds of insertion, removal, insertion from a pool, deletion and iteration.

The libraries used were written in C++.

IDE used was Visual Studio 2019 Professional with C++ optimisations set to Maximum Optimisation setting **O2**, favour speed and **Ot**, to favour fast code.

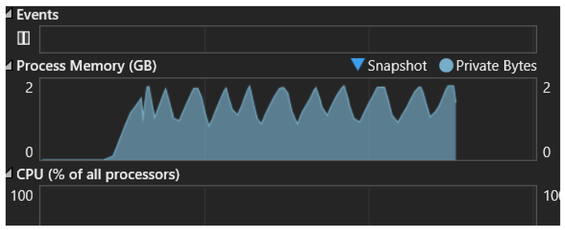
The machine this was performed on was an Asus Tuf Gaming Laptop with a AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx 2.10 GHz processor with 8GB installed RAM.

Data was gathered by setting up one hundred thousand components to be inserted, removed, inserted from a pool, deleted, copy construction and calling the update method inside a loop set to run 100 times. The first thing done inside the loop was to delete every component before taking time.

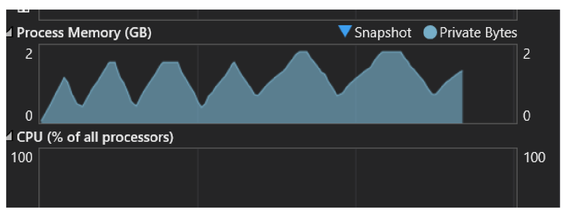
Reason for performing tests in such a manner was to simulate a gameplay where components would be added and removed from entities multiple times. The use of a large quantity of components allowed us to get more accurate timed data to compare with other libraries that exist.

The timer was set up to return the slowest time, fastest time and average time of each action.

1. DECS Tests



2. ECS Tests



Results

Data was collected after the methods performed one hundred times storing data for the fastest, slowest and average time of all complete executions.

Some data was skewed because the used ECS library didn't have a built-in pooling system so the add from pool data is not correct. Though the result does show a benefit of using pooled objects over construction of new classes.

As EnTT used sparse set logic that outperformed the first implementation of DECS it was then also decided that DECS would use a sparse set.

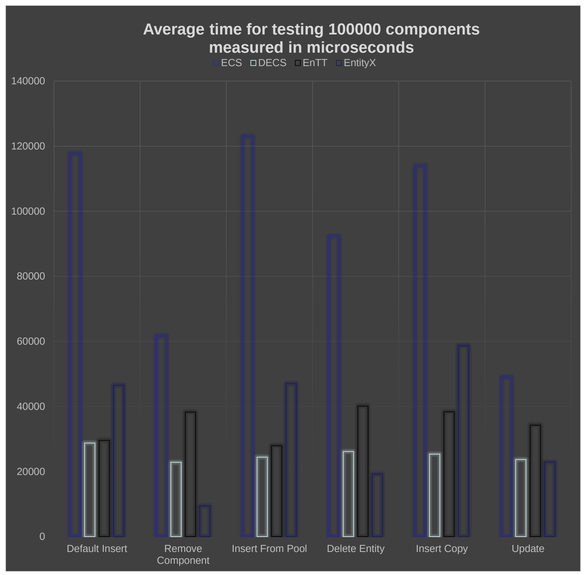
As the standard ECS library didn't support pooling for remove the delete component test was reworked into the delete entity test. The results still showed that using an integer-based model was still faster.

For 3 out of the 6 tests DECS library turned out to be better than the other libraries currently out there.

EntityX performed better in the remove and delet compoennts and the update speed was almost a tie with EntityX outperforming DECS by microseconds.

For removal of a component, DECS was 3 times faster than standard ECS. For each of the three inserts of a component DECS was 9 - 10 times faster, and the update loop DECS was a little more than 2 times faster.

DECS vs The World



Conclusions

Can we improve the design pattern know as Entity Component System by dropping the Entity?

Yes, we can. By removing the entity from the design pattern and replacing it with an integer-based model there are benefits in all aspects.

The biggest improvements were seen in all aspects outperforming the standard model by as much as 10 times.

It is also worth noting that by using a different method for pooling our components we outpaced all other ECS libraries.

The implications of using an integer-based Entity Component System as the standard design can boost performance for all games that rely on removing and insertion of components regularly. This means we can do more with games as we have the benefits of increased performance.

In future it would be beneficial to look at other algorithms for removal and deletion of components while maintaining the order in the dense list and maintaining the update speeds of each component.

To close out, an integer-based model is the way forward for improving the Entity Component System and should be standard. Using a pool further improves insertion times of components.

In future, different containers for components in systems should be looked at to improve performance further such as sparse arrays as with EnTT.

DECS Library

1. **https://github.com/PaulJame5/Dropped-Entity-Component-System**