Profiling a Prototype Game Engine Based on an Entity Component System in C++

Dennis Nilsson & Anton Björkman
Malmö University
2019-05-21

Abstract—**This paper introduces a performance comparison between an object-oriented artifact and a data-oriented artifact in the context of using an entity component system. It will also provide a discussion about the modifiability and readability attributes of data-oriented and object-oriented design.**

A game engine must handle a variety of game objects which may share common attributes such as a transformation, collision and input management. Instead of solving this via multiple inheritance, entity component system uses composition to decouple traits into data (components) and systems operating on them. Game objects (entities) then serve as containers for all components necessary for a particular purpose, such as a player character of a piece of platform. Composition is not necessarily data oriented however, as evident by recent development in e.g. the Unity engine [10]. This thesis will implement a small prototype for a data-oriented entity component system game engine and compare it to an object-oriented one observing the performance.

Keywords: Data-oriented design, Object-oriented design, Entity Component System

I. Introduction

Games are becoming larger for each generation inevitably resulting in more computations for the hardware to perform. One reason for the continuing enlargement of games is the aspiration of achieving higher realism than previous generations and or adding more content. Unity is currently component-based but is introducing a new component which is an entity component system. This is because component-based systems did not age well because it did not align data in memory possibly resulting in low cache coherence [10]. Programming using an object-oriented design without considering the data allocation can result in inefficient allocation of data in memory due to factors such as encapsulation of data, inheritance, polymorphism. Memory pools could be used to allocate objects dynamically while maintaining data locality. Encapsulation of data refers to bundling of data and methods operating on that data in the same class. Logic and data bundled together in a class could lead to low cache coherence. A low cache-coherence can possibly require more fetches from memory from the central processing unit (CPU) affecting performance negatively. A data-driven design such as an entity component system has the property of aligning data locally which could lead to improved cache-coherence.

We would like to thank our supervisor Carl Johan Gribel and examinator Olle Lindeberg for supporting us while writing this thesis at Malmö University.

Today's games require efficient computation of data to maintain stable frames per second (fps). Unity currently uses a component-based entity system. This simplifies game development since its intuitive to add components to objects. A consequence of this is, in order to be able to create or destroy objects in Unity, a global list must be modified containing the names of each object. This will require a mutex lock. Another consequence is an extension of the previous consequence. Every game object has a C# wrapper pointing to the C++ object. There is no control over where in memory it is allocated, meaning it could be anywhere [10]. This could potentially increase the amount of cache misses. A cache miss happens when data required for the CPU's next operation does not exist in the cache. The CPU then has to fetch new data from memory in order to continue its work. Data-oriented design approach such as a entity component system focuses on aligning data locally. This can possibly reduce the amount of cache misses. The purpose of this study is to explore the gains and consequences of a data-oriented artifact compared an object-oriented artifact. This will be done by conducting benchmarks on two artifacts, a data-oriented and an object-oriented. The benchmarks will record the time it takes to render frames in milliseconds. By analyzing the results of the benchmarks, this thesis aims to contribute by confirming the gains and consequences in performance when using a data-oriented or an object-oriented artifact. The relevance of this thesis is providing important insight for developers when performance is a prioritized quality attribute. Future work extending our thesis could be examining the number of CPU cache misses per benchmark. Furthermore benchmarks in 3D environments would be of great importance for contemporary developers within the game industry.

The structure of this paper is as follows: This section, the introduction. Section 2 takes up background and related work. Section 3 explains the method used conducting the experiment. Section 4 presents results and data. Section 5 & 6 includes discussion and conclusion. Abbreviations will be avoided to minimize confusion.

II. Background and related work

The complexity of games striving for a realistic experience graphics-wise are increasing for each generation of consoles. This derives partly from the increasing amount of polygons in striving for realistic graphics [9]. It also derives from the fact that the more data a game scene contains, the more computation power is required to maintain for example 60 frames per second, as opposed to a game scene containing less data. Stable frames per second relies on computations being finished within the time limit of the game loop [12]. As an example, to be able to maintain ∼60 fps, ∼16 milliseconds (ms) is the game loops time limit. Not being able to achieve stable frames per second is considered as game-breaking in the game industry.

Even though computation power has been increasing since the arrival of computers, there are complications such as the difference in clock speed between the CPU and memory [12]. Factors such as increased power consumption, heat and thermal losses is contributing in the challenge of developing faster CPU's [7]. While CPU's has mostly increased in clock speed, DRAM's been mostly increasing in size. This is a bottleneck in performance because the clock speeds in CPUs exceed the clock speeds in DRAM memory [2]. Possibly with the outcome that the CPU waits a couple of hundred cycles until the DRAM is able to provide the necessary data for processing [12]. Since cache storage exist in the CPU's for speeding up memory access, developers should take advantage of that. By utilizing as much of the hardware's capacity as possible. As Bob Nystrom mentions, "Modern CPUs have caches to speed up memory access. These can access memory adjacent to recently accessed memory much quicker. Take advantage of that to improve performance by increasing data locality - keeping data in contiguous memory in the order that you process it" [12].

A data-oriented design tends to promote data locality. It is important to align data so that it can be processed after each other in the cache line. By following a data-oriented design developers could increase the cache coherence. An object-oriented design unfortunately tend to spread data randomly across the memory due to storing objects on the heap, unless a memory pool is implemented and used. A positive effect of an object-oriented approach is the readability. For instance a human would be described in one class, as one object with all the properties a human holds.

This thesis will address the gains and consequences of data-oriented and object-oriented design in a game-like environment when using an entity component system. The main contribution of the benchmarks is displaying the performance of, in render time in milliseconds per frame, a data-oriented artifact and an object-oriented artifact using an entity component system. RQ: What are the gains and consequences of data-oriented and object-oriented artifacts simulating a game-like environment using an entity component system? Hypothesis: The data-oriented artifact renders frames faster than the object-oriented artifact in milliseconds.

A. *Object oriented*

Object-oriented design brings several attributes such as readability, inheritance, polymorphism, encapsulation of data and reusability. The readability attribute simplifies the communication within a project. Object-oriented design strives after describing things as humans perceive it. It makes it easier for team members to familiarize with a project. A possible negative consequence of object-oriented design is the potential dynamic allocation of objects in run time. This can be counteracted by allocating memory for objects before run time. If there is not already allocated memory for an object, it must be instantiated leading to the allocation of memory on the heap of that object. Dynamically allocated objects not making use of a memory pool might lead to low cache coherence because the data is not sequential which in turn could lower performance [6]. Figure 1 shows an illustration of how unaligned data may look like in memory.

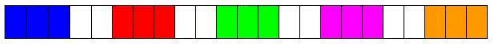


Figure 1: Visualization of unaligned data in memory.

B. *Data oriented*

Data-oriented design focuses on data and cache coherency with the aim of having fewer cache misses. The design pattern is especially important for systems running in real-time such as game engines [6]. Data is not fast or slow but the hardware operating on it is. Meaning where it resides in memory matters. It is comparable to a book where the content is the data and the reading speed is the processor. If a book has the same size and two readers, the one finishing first will be the one reading faster. But imagine if the pages were scrambled in one of them. The one finishing first would probably be the one with the coherent text, almost no matter the reading speed of the readers. Data-oriented design purpose is aligning data in memory so when code is running the things to process is next to each other in memory. Like reading a book with coherent text. Figure 2 shows an illustration of how aligned data may look like in memory.

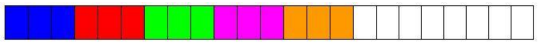


Figure 2: Visualization of aligned data in memory.

C. *Entity Component System*

Entity Component System is an architectural pattern that in a sense is an inverted object-oriented design. Instead of the entities using its own implementations of logic, the entities are acted upon by components which in turn are controlled through systems [4]. Entity component systems are novel and there is not much research within the area. Unity is integrating entity component system as a component in their game engine. This is due to the ability of retaining the user-friendliness of the component-based system currently used meanwhile also gaining performance and parallelism [10]. An illustration of an entity component system is shown in Figure 3.

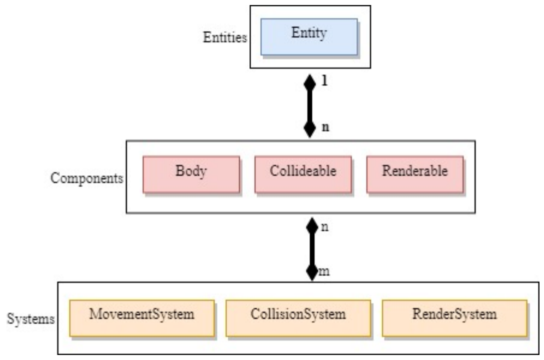


Figure 3: Visualization of an Entity Component System.

D. Related work

In Olof Wallentins paper [19], he analyzes a theoretical implementation of a component-based entity system. In order to do this, he made a comparison between several component-based entity systems. The comparison was made with an in-house game engine using different C++ component-based entity systems at the company he was working for. In his discussion he mentions that a data-oriented approach makes it harder for developers to understand and or lookup the associations within the system.

In Kim Svenssons and Tord Eliassons paper [8], they compare the functional and object-oriented paradigms in terms of memory usage and execution time. To compare these, they use Javascript and four algorithms binary search tree, shellsort, tower of hanoi and Dijkstra's algorithm. They state in their discussion that functional programming and object-oriented programming has different purposes but overall the functional programming performed worse than object-oriented programming. Their methodology is similar to ours as they conduct experiments and measure the different results in milliseconds. Their experiments involve different input data ranging from 1000 to 10 000 values.

In Walid Faryabis paper [6], he compares a data-oriented design to an object-oriented design by implementing his own entity component system. To compare these, he made performance tests in Unity with C# which consisted of simulating a sine wave and game objects with and without animations. Each test consisted of a set amount entities and measured the frames per second. The data collected was then calculated to a mean. His results shows that data-oriented design provides better average frames per seconds while using his entity component system. Our thesis will not provide results in form of frames per second. The results presented our this thesis will be time taken to render a frame in milliseconds. The chosen metric for this thesis is motivated by the difference between 90-100 fps not being equivalent to the difference between 20-30 fps. The difference is easily understood if one divides $1 \div 90$ and $1 \div 20$. Our thesis will be using EntityX [17] as the entity component system and implement a simple game-like artifact to conduct the experiments. Walid's comparisons are done with 3D models and animations using OpenGL. This thesis will make use of SFML to render 2D graphics [14].

III. Method

A. Methodological background

The methodology used in this thesis is research design and the chosen strategy is conducting experiments [13]. The independent variable in the experiments are the number of entities. The dependant variable is the time it takes, in milliseconds, per frame to render. The data generation method is observation. Each benchmark will be observed from which the data will be extracted. The data is analyzed quantitatively.

B. Research setting

All experiments were conducted using the same computer under the operating system Windows 10 Home in C++ using the integrated development environment Visual Studio Community 2017 [11]. The computer's build-spec are Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz, 16.0GB DDR4 RAM, 128GB SSD, Intel(R) HD Graphics 530.

1) Entity Component System - EntityX: EntityX is a type-safe C++ entity component system that includes its own benchmarks and is still maintained by the author [17]. Furthermore a benchmark suite has been created using up to 2 million entities in the benchmarks [1]. Projects has been made using EntityX [3], [16], [18], [5] to create 2D and 3D games. Therefore EntityX is a suitable framework for this thesis.

2) Simple and Fast Multimedia Library - SFML: EntityX was chosen as a framework for benchmarking. Since the benchmarks are measuring time per rendered frame an application for rendering also had to be chosen. An example using SFML was included in EntityX. This example was decided to be used as a foundation for the benchmarks. The reasons that motivated the use of SFML is the following. SFML is a cross-platform and multi-language framework consisting of five modules: system, window, graphics, audio and network. SFML has an active community which continues to implement new features and maintaining old features. SFML has official bindings for the C, .Net and other languages [14]. Several games has been made using SFML or some parts of SFML, some of these can be found on Steam [15].

3) Integrated SFML with EntityX on Windows 10 Home: EntityX included an example.cc using SFML for rendering. In this example there is a random function that did not function properly. The random function worked properly when rewritten as:

```
float r (int a, float b = 0)
{
      return ((float) std::rand())/RAND_MAX * a + b;
}
```

The random function original:

```
float r(int a, float b = 0)
{
        return static_cast<float>(std::rand() %
        (a * 1000) + b * 1000) / 1000.0;
}
```

C. Research approach

The chosen research approach is experiment. Hypothesis: The data-oriented artifact renders frames faster than the object-oriented artifact in milliseconds. The experiment is conducted in the following way. The process of the experiment is observing benchmarks while measuring and recording the metric render time per frame in milliseconds. 20 benchmarks will be conducted, each with different amounts of entities, explained in section III-C1. The benchmark results are compared. The prediction is that data-oriented artifact will have higher performance. The artifact with the lowest time per rendered frame is the one with the highest performance. The experiment is intended to show the gains and or consequences of our artifacts.

1) The benchmarks: The benchmarks are implemented equivalently in regards of code. For the benchmarks, two artifacts were used. These artifacts implements the following two different paradigms, data-oriented and object-oriented design. The differences between the data-oriented artifact and the object-oriented artifact is shown in Figure 4. The data-oriented artifact uses three components Body, Collideable and Renderable. The object-oriented artifact only uses one component Circle consisting of Body, Collideable and Renderable. To understand the difference between the artifacts in memory allocation Figure 5 shows the memory layout for each one. In the data-oriented artifact the components are separated and placed one after another. The object oriented artifact is similar but there is only one component, Circle, that is aligned after each other. Each benchmark was executed for the same amount of time with different amount of entities. The amount of entities tested in the benchmarks range from 500 to 10000 entities in increments of 500 entities. During the benchmarks each frame's render time was logged. How many milliseconds it takes to render one frame has been calculated to an average using the values retrieved from the benchmarks.

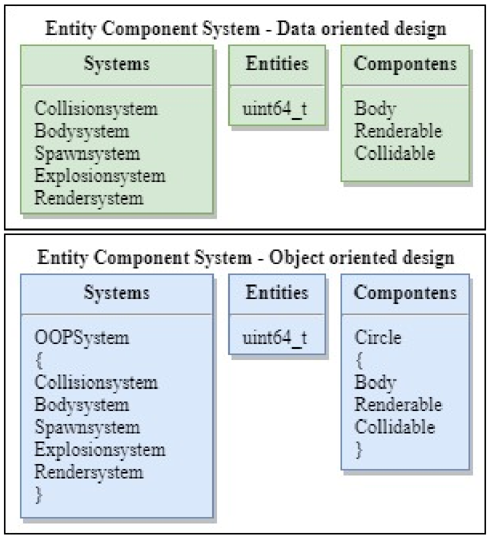


Figure 4: Describes the data-oriented and object-oriented artifact in terms of the Entities, Systems and Components.

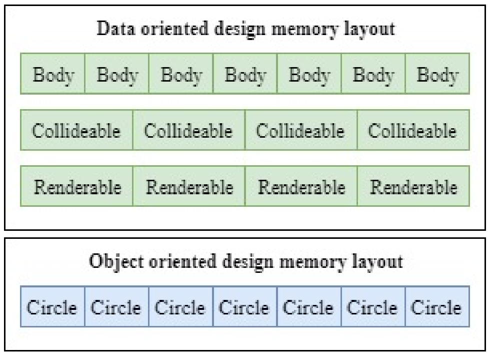


Figure 5: Describes the data-oriented and object-oriented memory layout.

2) Systems: Each system in our artifacts has a specific task. There are five systems CollisionSystem, BodySystem, SpawnSystem, ExplosionSystem and RenderSystem. The CollisionSystem handles collisions between the entities. The BodySystem handles the movement of an entity i.e. their position and direction. The SpawnSystem spawns entities into the environment. Each frame the SpawnSystem spawns as many entities as the total count of the initial value, which is set at the start of the application. The ExplosionSystem handles the removal of a colliding entities each frame i.e. making that entity inactive using a flag. The RenderSystem renders the entities.

3) Components: As shown in Figure 6: Body, Renderable, Collideable and Circle components used in our artifacts. The Body component has a size of 28 bytes. It consists of two vectors and three floats. The Collideable component consist of one float, having a size of 4 bytes. The Renderable component consist of a smart pointer to a shape, the size of this component is 8 bytes. The Circle component is built on the other components i.e. Body, Collideable and Renderable having a size of 40 bytes.

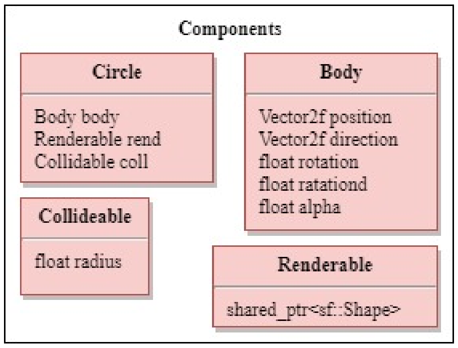


Figure 6: Describes the components used by our artifacts.

D. Data collection

Each benchmark records the rendered time per frame in milliseconds. The recorded time is calculated and stored as floats in a vector. After the benchmark finishes the floats are written to a text file.

1) Implementation of render time measurement: Before the game loop the current time is taken by using the clock from the C++ standard library. After the game loop the elapsed time is stored and the time taken between is calculated. The measured time is stored in a vector using the push back function in the C++ standard vector library.

2) Implementation of writing to file: A C++ standard library vector of floats is declared globally. At the end of the benchmark i.e after 10 000 milliseconds the data is streamed to a text file.

E. A quantitative data analysis

The benchmarks in this thesis are measuring how much time it takes for a frame to be rendered in milliseconds. An average value is calculated and presented through diagrams. The diagrams show a comparison of both the artifacts in render time per frame, render time per entity and performance gained of the data-oriented artifact in milliseconds. The amount of entities we decided to test is based upon reasonable amounts of objects in a game scene.

F. Limitations

The benchmarks will only measure the time taken per rendered frame in milliseconds. This metric is stored as a float. The range in object's sizes is between 4 bytes and 40 bytes. All benchmarks are executed in the same environment for the same amount of time. The artifacts used in the benchmarks are equal in implementation other than one being data-oriented and the other object-oriented. An important aspect regarding the benchmarks is that we are assuming that cpu prefetches are made sequentially. Since prefetches are cpu dependant we cannot guarantee the actual process of prefetchning made by the cpu in the benchmarks without taking our cpu in account. This is the reason for choosing to only measure the time per frame rendered in milliseconds and conducting all experiments using the same computer. Benchmarks range from from 500 entities to 10 000 entities, by the increments of 500 entities each benchmark.

IV. DATA

A. Artifact

Pseudo code for the artifact below, describing every frame in the game loop. Figure 7 shows an artifact with all the systems implemented running 1000 entities.

```
Implementation per frame in pseudo code of both
artifacts
        int c = 0;
        Count entities , store in variable c

        for i := 0 to amountOfEntites − c
            Create an EntityX entity
            Add to EntityX memory pool
            Assign components to entities

        for i := 0 to entities
            Handle movement

        for i := 0 to entities
            Handle bouncing on screen bounds

        for i := 0 to entities
            Check collisions
                if(collision)
                    EntityX inactivates entity

        for i := 0 to entities
            Render entities with SFML
```

B. Benchmarks

Table 1, 2 and 3 presents the results provided by running the benchmarks using our research approach. Table 4 presents detailed information regarding the benchmarks.

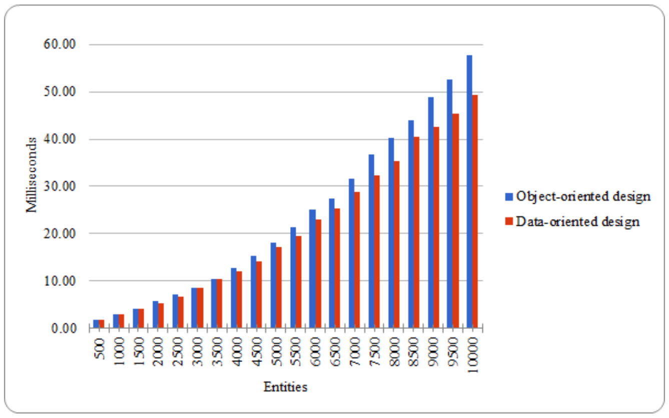


Table 1: Benchmarks ranging from 500 entities to 10 000 entities. Average render timer per frame.

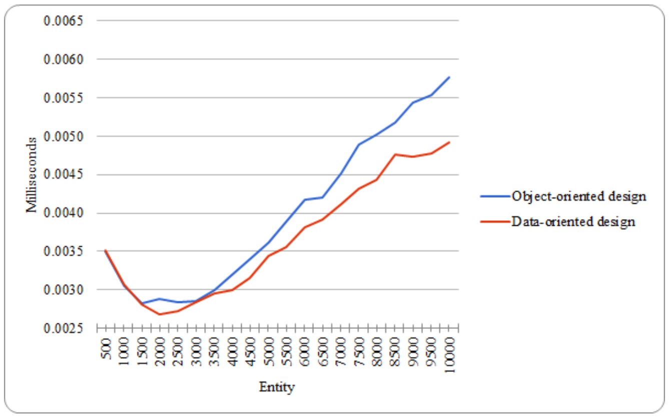


Table 2: Average render time per entity per frame.

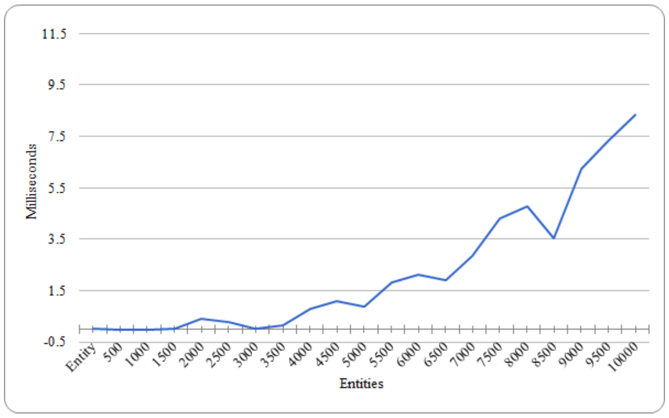
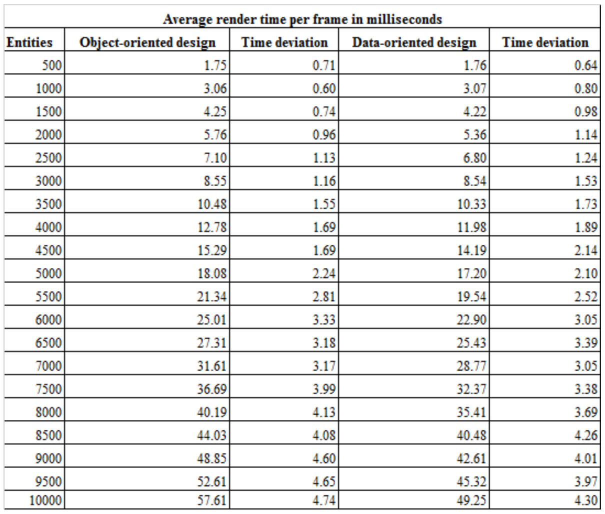


Table 3: Performance gained per frame in milliseconds using the data-oriented artifact



Average render time per frame in milliseconds				
Entities	Object-oriented design	Time deviation	Data-oriented design	Time deviation
500	1.75	0.71	1.76	0.64
1000	3.06	0.60	3.07	0.80
1500	4.25	0.74	4.22	0.98
2000	5.76	0.96	5.36	1.14
2500	7.10	1.13	6.80	1.24
3000	8.55	1.16	8.54	1.53
3500	10.48	1.55	10.33	1.73
4000	12.78	1.69	11.98	1.89
4500	15.29	1.69	14.19	2.14
5000	18.08	2.24	17.20	2.10
5500	21.34	2.81	19.54	2.52
6000	25.01	3.33	22.90	3.05
6500	27.31	3.18	25.43	3.39
7000	31.61	3.17	28.77	3.05
7500	36.69	3.99	32.37	3.38
8000	40.19	4.13	35.41	3.69
8500	44.03	4.08	40.48	4.26
9000	48.85	4.60	42.61	4.01
9500	52.61	4.65	45.32	3.97
10000	57.61	4.74	49.25	4.30

Table 4: Detailed information about Table 1.

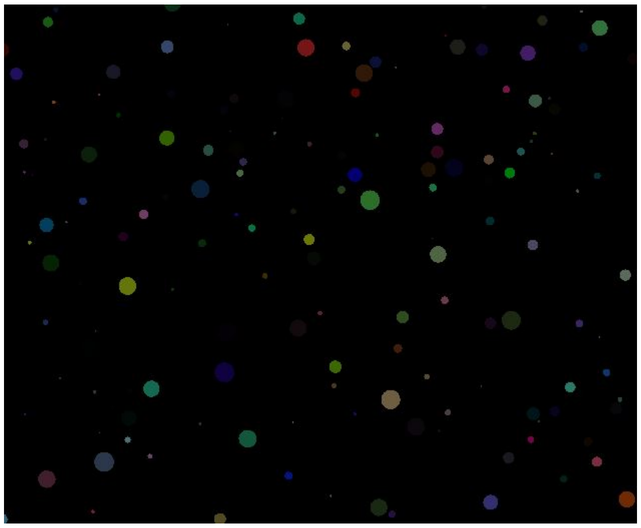


Figure 7: Screenshot of our artifact running 1000 entities.

V. Discussion

This section is about the performance, modifiability and readability attributes of our data-oriented and object-oriented artifacts.

A. *Prelude*

Data is data which is neither fast or slow. What determines the reading and writing speed of data is the hardware connecting it and or storing it. Furthermore there is a difference in clock speed between the CPU and DRAM. The bottleneck introduced is that the CPU might have to wait hundreds of cycles for a single byte if the data does not reside in its own cache. Data locality could improve performance because it should minimize the amount of cache misses occurring in run time i.e. improving the cache coherence. Before writing this thesis, we suspected that data-oriented artifact would be faster than the object-oriented artifact. What we didn't know was if our data-oriented artifact was faster than our object-oriented artifact and if so how much faster.

A game running in ∼60 fps requires all computations to be finished within ∼16 ms. Suppose that a function takes 1ms and all functions in the systems are equally fast. This would mean that the budget is 16 functions. A function could be handling the movement of game objects among other tasks. These functions would have to be efficient to keep within the time limit. Failing to make them efficient will affect the fps. As shown in Table 4, it takes about ∼15ms for the game loop to finish running with 4500 entities with the object-oriented artifact. That is about ∼1ms slower per rendered frame compared to the data-oriented implementation. This time could be spent better on other resources, such as controlling a population with artificial intelligence. The relation seen in the benchmarks did not correspond fully with our hypothesis. The data-oriented artifact renders frames faster than the object-oriented artifact when the amount of entities is over 3500.

B. *Performance*

By looking at the results of the benchmarks, the data-oriented artifact running over 3500 entities resulted in lower render times per frame than the object-oriented artifact. The object-oriented artifact running up to 3500 entities resulted in similar or lower render times per frame than the data-oriented artifact. A possible explanation for the performance gain of the object-oriented artifact when running an amount of entities under 3500 could be that relevant data resides in the cache. If this is the case, it means that the CPU does not have to fetch data from memory. But when the entities exceed 3500 in amount, it would seem that the required data for the next instruction does not longer fit in the CPU cache. Which leads to a cache miss, meaning the CPU can't process its next instruction without fetching new data from memory. But in the case of the data-oriented artifact the data is possibly more efficiently aligned resulting in fewer cache misses when running a large amount of entities, in this case over 3500 entities. For clarification, the components, such as the body component, is stored in an array which in turn aligns the data efficiently. For example if data about the body component is required to perform an operation, the only array that needs to be traversed is the array containing body components. In the object-oriented artifact the components has been stripped down and put together into a single class to describe the object Circle. Therefore if in need of data regarding the body of Circle or any other property residing in Circle, the traversal of the array containing Circles is required to find the specific data and then use and or modify that data for an operation.

C. *Modifiability*

One of the potential downsides of object-oriented design is the lack of modifiability. A class created to describe a human containing both data and logic can make modifications hard. The data could for example be the weight, length, eye color, shoe size of the human. The logic could for example be controlling the movement of the human of which we will now continue to refer to as an entity. Changes within this class might affect more than wanted because of coupling. A possible downside using an entity component system is that there is no longer a class to describe a human. Instead a combination of components act upon an entity. This enables developers to remove or add components to any entity easily. This is one of the benefits of data-oriented design considering games are intermittently modified. Using an entity component system could provide easy modification during game development. An example could be a character representing a hero in a game. Suppose that, in code, the character is represented by the composition of components acting upon the entity i.e. the character in the game. After a few updates in the game, the developers decide to remake the character into a villain with new weapons, abilities and armor. This would require modification of code. New weapons, abilities, and armor can be added as new components. Using this approach the developers could remake a character with ease just by removing the old components and assigning new ones.

D. Readability

A possible gain of object-oriented design is the readability. Programming is a demanding task and good communication is an advantage for efficient teamwork. Depicting the objects as if they resided in the real world eases the communication since humans have a similar perception of the world. The possible loss of data-oriented design is the readability. Getting an overview of the project becomes a much more tedious task since its not easy knowing which components are implemented or which entities the components will act upon.

E. Summary

To summarize, the experiments shows that there is a relation to data locality and performance. By conducting benchmarks on data-oriented and object-oriented artifacts using entity component system the research question could be answered. The contribution is showing that when running benchmarks with an amount of entities over 3500 our data-oriented artifact provided better performance than our object-oriented artifact when using EntityX.

VI. Conclusion

This paper set out to explore the gains and consequences of a data-oriented artifact compared to an object-oriented artifact. The benchmarks provide the results that the data-oriented artifact is faster than the object-oriented running amounts of entities over 3500. Addressing this is an important contribution to contemporary game developers. This thesis has mostly focused on performance and the performance gained by data-oriented design artifacts implies that the findings are likely to be of importance to game developers. Furthermore it is important to developers having an interest of increasing performance or learning more about data locality.

In terms of future research we particularly suggest performing benchmarks while recording the number of CPU cache misses in an object-oriented designed artifact compared to a data-oriented. This is important to further present a possible performance gain by applying data-oriented design when developing performance-reliant applications such as games. The recording of the amount of CPU cache misses can be done within Visual Studio, an integrated development environment. Furthermore performing benchmarks in 3D environments, preferably in AAA video games [20], to be able to show the gain in performance of data-oriented design in large games. A research question for future work could be: How can developers increase the readability in data-oriented design?

In the following two sections, this thesis conclusions will be presented.

A. Gains

1) Data-oriented artifact: Data-oriented artifact shown to be faster in benchmarks running over 3500 entities, presented in Table 1, 2, 3, 4. This implies that its worth considering a data-oriented approach when developing games that are in need of performance. The modifiability of the code is also improved due to modularity. The parts that make up the game objects are independent of each other. Adding or removing components from an entity does not affect other parts of the application.

2) Object-oriented artifact: The object-oriented artifact shown to be very similar in performance when running less than 3500 entities, presented in Table 1, 2, 3, 4. The readability promotes better communication. It makes it easier to familiarize with the code in the project. The gain of implementation is the straightforwardness of it. Implementing a circle containing properties is more intuitive than using a body, shape and collideable to represent a circle.

B. Consequences

1) Data-oriented artifact: One of the consequences of data-oriented design in our artifact is the lack of readability. Since the objects only contain data, the developer can not follow the data flow of the object as in object-oriented design.

2) Object-oriented artifact: The performance is shown to be slower when running more than 3500 entities, presented in Table 1, 2, 3, 4. Because of high coupling modifiability is affected. If modification were to be done on Circle in the experiment artifact, all connected systems would have to be modified as well.

References

[1] Alex Beimler. Ecs benchmarks, 2014. Available at: https://github.com/abeimler/ecs_benchmark/. [Accessed: 2019-04-11].

[2] Carlos Carvalho. The gap between processor and memory speeds. Semantic Scholar, 2002. Universidade do Minho, Braga, Portugal.

[3] Giovanni Giuseppe Costa. Battlecity2014, 2014. Available at: https://github.com/ggc87/BattleCity2014/. [Accessed: 2019-04-16].

[4] Marc Erich Latoschik Dennis Wiebush. Decoupling the entity-component-system pattern using semantic traits for reusable realtime interactive systems. *In 8th on Software Engineering and Architectures for Realtime Interactive Systems*, 2015.

[5] Giovani Milanez Espindola. Spacetd, 2015. Available at: https://github.com/giovani-milanez/SpaceTD/. [Accessed: 2019-04-16].

[6] Walid Faryabi. Data-oriented design approach for processor intensive games. Master's thesis, Norwegian University of Science and Technology, https://brage.bibsys.no/xmlui/, 2018. Available at: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2575669/18676_FULLTEXT.pdf?sequence=1. [Accessed: 2019-04-16].

[7] Alexander Fox. Why cpu clock speed isn't increasing, 2018. Available at: https://www.maketecheasier.com/why-cpu-clock-speed-isnt-increasing/. [Accessed: 2019-04-16].

[8] Tord Eliasson Kim Svensson Sand. A comparison of functional and object-oriented programming paradigms in javascript. DIVA, 6 2017. Blekinge Tekniska Högskola. Karlskrona, Sweden.

[9] Guy W. Lecky-Thompson. Video game design revealed. Cengage Learning, https://www.cengage.co.uk/, 2008. Page 117.

[10] Lucas Meijer. On dots: Entity component system, 2019. Available at: https://blogs.unity3d.com/2019/03/08/on-dots-entity-component-system/. [Accessed: 2019-04-16].

[11] Microsoft. Visual studio ide, 2019. Available at: https://visualstudio.microsoft.com/. [Accessed: 2019-04-10].

[12] Robert Nystrom. Game programming patterns, 2009. Available at: http://gameprogrammingpatterns.com/. [Accessed: 2019-04-16].

[13] Jinsoo Park Sundha Ram Peter Adams, Salvatore T. March. Design science in information systems research. *MIS Quarterly*, (1):75–105, March 2004.

[14] Simple and Fast Multimedia Library. Sfml. Available at: https://www.sfml-dev.org/. [Accessed: 2019-04-16].

[15] Simple and Fast Multimedia Library. Sfml projects, 2015. Available at: https://sfmlprojects.org/. [Accessed: 2019-04-16].

[16] Trans-Neptunian Studios. Triangulum, 2014. Available at: https://github.com/TransNeptunianStudios/Triangulum/. [Accessed: 2019-04-16].

[17] Alec Thomas. Entityx, 2014. Available at: Available at: https://github.com/alecthomas/entityx/ [Accessed: 2019-04-11].

[18] Will Usher. Asteroids, 2017. Available at: https://github.com/Twinklebear/asteroids/. [Accessed: 2019-04-16].

[19] Olof Wallentin. Component-based entity systems modular object construction and high performance gameplay. DIVA, 2014. Uppsala Universitet. Uppsala, Sweden.

[20] Wikipedia. Aaa (video game industry), 2019. Available at: https://en.wikipedia.org/wiki/AAA_(video_game_industry). [Accessed: 2019-04-19].

VII. Response Letter

———— Overall evaluation ———— SCORE: 1 (Good paper, minor modifications strongly suggested) —– TEXT: Good start on the thesis. This is a technically challenging project and still you've set up a working prototype – very impressive! It's great that there are some initial measurements as well. Grade: 1. It's hard to grade an unfinished thesis – revisions are obviously required, but it's still a good paper so far.

Below is a (pretty long) list of notes and suggestions based on the thesis in its current form. It's pretty dry and should not be interpreted as a teardown by any means – you have something really good going and this is all intended to help you out. Just wanted to state that even though it is kind of obvious.

Fix introduction - Introduction is changed and modified after the feedback, generalisation has been removed or referenced. Are there any more generalisations that are not good to keep?

Elaborate/argue more on the choice of methodology. For instance, why not use the 7-step design science process by Hevner/Peffer? [ ] Alan R. Hevner. Salvatore T. March. Jinsoo Park. Sudha Ram. "Design Science in IS Research", MIS Quarterly, vol. 28(1), pp75–105. March 2004. [ ] Peffers et al. "A Design Science Research Methodology for Information Systems Research", Journal of Management Information Systems, vol. 24 Issue 3, Winter 2007-8, pp. 45-78.
By looking at other theses methodology we chose to design our methodology similar to theirs.

Section 2: * Avoid informal and loaded words such as "huge" and "a lot". Either state an amount, or keep it neutral by saying e.g. simply that there is "a difference".
Ok, deleted or modified

* These two sentences are not quite compatible. Have CPU clock speeds halted or increased?: "Furthermore CPU clock speeds has halted due to factors such as increased power consumption, heat and thermal losses [7]. As mentioned the CPUs has mostly been increasing in clock speed while the DRAMs has mostly been increasing in size."
ok, changed to "Even though computation power has been increasing since the arrival of computers, there are complications such as the difference in clock speed between the CPU and memory [12]. Factors such as increased power consumption, heat and thermal losses is contributing to the challenge of developing faster CPU's [7]. While CPUs mostly has been increasing in clock speed, DRAMs been mostly increasing in size."

* It's hardware's and not hardwares', and other similar examples.
Ok, changed * "As mentioned" can usually be avoided by restructuring the text. Ok, deleted/restructured
2.4, first sentence. Don't spell out the publication name. Ok, changed
"In Olof Wallentins paper[19], he analyzes a theoretical implementation of a Component-based Entity System." "In Walid Faryabis paper [6], he compares a data-oriented design to an object-oriented design by implementing his own entity component system."

Are there any methodological similarities between your work and Svensson Eliasson?
Added "Their methodology is similar to ours as they conduct experiments and measure the different results in milliseconds. Their experiments involve different input data ranging from 1000 to 10 000 values."

Faryabi is a good reference, but you could elaborate more on how your experiment differs (other measuring ms vs. fps). Granted that you have not yet outlined it at this point in the paper, but you can still make a quick mention of similarities and/or differences.
Ok, change/<added to Faryabi "Our thesis will be using EntityX [17] as entity component system and implement a simple game-like artifacts to conduct the experiments. Walids comprising are done with 3D models and animations using OpenGL and this theises will use SFML[14] to render 2D graphics."

Consider acknowledging more clearly that ECS in general and data-local ECS in particular is a novel and not well researched area. Why not mention that Unity is integrating data-locality in their architecture very actively right now – this adds a lot of relevancy and rigor to your work.
Ok, changed to "Entity component systems are novel and there is not much research within the area. Unity is integrating entity component system as a component in their game engine. This is due to the ability of retaining the user-friendliness of the component-based system currently used meanwhile also gaining performance and parallelism [10] " We also added this to the introduction "Unity currently uses a component based entity system. This simplifies game development since its intuitive to add components to objects. A consequence of this is, in order to be able to create or destroy objects in Unity, a global list must be modified containing the names of each object. This will require a mutex lock. Another consequence is an extension of the previous consequence. Every game object has a C# wrapper pointing to the C++ object. There is no control over where in memory it is allocated, meaning it could be anywhere [10]. "

Your RQ; "RQ: What are the gains and consequences of data-oriented entity component system design compared to an object oriented-design?
Ok, changed to What are the gains and consequences of data-oriented and object-oriented artifacts simulating a game-like environment using an entity component system?

is a bit broad and you should specify the setting being measured: e.g. a multibody, "game-like" simulation implemented using ECS, intended to mimic the setting of a game.

Hypothesis: A data-oriented design equals faster rendering times." The phrase "equals faster rendering times" implies mathematical equality which is probably not even possible to prove. Consider a more modest hypothesis, and tie in with the setting.
Ok, changed to The data-oriented artifact renders frames faster than the object-oriented artifact in milliseconds.

Section 3: What is happening on the screen? What does it look like? You explain the components being part of the benchmark/prototype/artifact, but not what the Systems are doing. This is important since the system logics tells us about the time complexity of the simulation, which in turn tells us something about how the rendering time is expected to grow with the number of entities.

Ok, added screenshot at the result section Ok, added Implementation per frame in pseudo code of both artifacts int c = 0; Count entities, store in variable c

for i := 0 to amountOfEntites - c Create an EntityX entity Add to EntityX memory pool Assign components to entities

for i := 0 to entities Handle movement

for i := 0 to entities Handle bouncing on screen bounds

for i := 0 to entities Check collisions if(collision) EntityX inactivates entity

for i := 0 to entities Render entities with SFML

Ok, added new Section Systems in Research approach "Each system in our artifacts has a specific task. There are five systems CollisionSystem, BodySystem, SpawnSystem, ExplosionSystem and RenderSystem. The CollisionSystem handles collisions between the entities. The BodySystem handles the movement of an entity i.e. their position and direction. The SpawnSystem spawns entities into the environment. Each frame the SpawnSystem spawns as many entities as the total count of the initial value, which is set at the start of the application. The ExplosionSystem handles the removal of a colliding entities each frame i.e. making that entity inactive using a flag. The RenderSystem renders the entities."

Avoid C++ syntax such as "shared ptr" and std::vector in the text body. Use more common terms.

Ok, from The Renderable component consist of a shared ptr to a sf::Shape, the size of this component is 8 bytes." To "The Renderable component consist of a smart pointer to a shape, the size of this component is 8 bytes."

Ok, changed to "A C++ standard library vector of floats is declared globally."

2.1: The allocation vs. locality discussion here is a bit confusing to me…

Ok, changed to "Object-oriented design brings several attributes such as readability, inheritance, polymorphism, encapsulation of data and reusability. The readability attribute simplifies the communication within a project. Object-oriented design strives after describing things as humans perceive it. It makes it easier for team members to familiarize with a project. A possible negative consequence of object-oriented design is the potential dynamic allocation of objects in run time. This can be counteracted by allocating memory for objects before run time. If there is not already allocated memory for an object, it must be instantiated leading to the allocation of memory on the heap of that object. Dynamically allocated objects not making use of a memory pool might lead to low cache coherence because the data is not sequential which in turn could lower performance [6]."

2.1, 2.2: Should not begin with "A design pattern that …" Ok, changed

3.2.1 Consider "thesis" instead of "essay". Ok, changed to "Therefore EntityX is a suitable framework that fulfills the purpose for this thesis."

You could mention that entityx is implemented using pretty advanced C++ optimization techniques such as CRTP (at least I think it is!), and variadic templates to handle component families.

"EntityX is a C++ entity component system that includes its own benchmarks and is still maintained by the author [17]. Furthermore a benchmark suite has been created that has tested using up to 2 million entities [1]. Projects have been made using EntityX [3], [16], [18], [5] to create 2D and 3D games. Therefore EntityX is a suitable framework that fulfills the purpose for this thesis."

Did not change

Figures: In the final version, try to use the same font face in figures as is used in the rest of the thesis. Also, please use vector rather than raster format so that figures are antialiased when rendered.

Ok, Fixed

Chapter 4: * Table 4: One or two decimal digits is enough. Ok, fixed * The three graphs can be merged into one, i.e. put all (six) bars in the same graph.

fixed * There's a pretty big jump from 1k to 10k entities. Consider adding more series from 0.5k to 10k in steps of 0.5k (500, 1000, 1500, …). In this case, you could plot the data as a curve instead of bars. This way we can see if the rendering time increases in a linear fashion, or something else (probably not linear). If you don't want to make this many runs manually you could automate it using a batch script which runs the program over and over with different inputs.

Fixed, now running 20 benchmarks per artifact.

* Another metric to analyze is time/entity, i.e. divide each frame time with the of entities. Looking at the data, time/entity goes up for DOD with of entities while is goes down and then up for OOP – any idea why? Does DOD/OOP scale differently with of entities.

Ok, fixed * I would like some kind of difference-metric as well, showing how much faster DOD is vs. OOP. Either document the difference in milliseconds, or use a normalized factor where OOP is 1 and DOD is (e.g.) 0.8, per measurement. How does the difference change with the number of entities?

Ok, fixed, shown in Table 3.

Chapter 5 currently is a bit unstructured. There's some background theory on memory bottlenecks and games which would fit better earlier in the thesis (some may be recapitulated here for clarity of course, but no exhaustive explanations), and so on. A typical "Discussion" section *explains* the results, why they look like they do, possible unwanted factors (e.g. bottlenecks not related to memory fetching), and, importantly, it ties back to earlier work.

Ok, added "By looking at the results of the benchmarks, data-oriented artifact running over 3500 entities resulted in lower render times per frame. The object-oriented artifact running up to 3500 entities resulted in similar or lower render times per frame. A possible explanation for the performance gain of the object-oriented artifact when running an amount of entities under 3500 could be that relevant data resides in the cache. If this is the case, it could mean that the CPU does not have to fetch data from memory. But when the entities exceed 3500 in amount, it would seem that the required data for the next instruction does not longer fit in the CPU cache. Which leads to a cache miss, meaning the CPU can't process its next instruction without fetching new data from memory. But in the case of the data-oriented artifact the data is more efficiently aligned resulting in fewer cache misses when running a large amount of entities, in this case over 3500 entities. For clarification, the components, such as the body component, is stored in an array which in turn aligns the data efficiently. For example if data about the body component is required to perform an operation, the only array that needs to be traversed is the array containing body components. In the object-oriented artifact the components has been stripped down and put together into a single class to describe the object Circle. Therefore if in need of data regarding the body of Circle or any other property residing in Circle, the traversal of the array containing Circles is required to find the specific data and then use and or modify that data for an operation."

"What was unknown was how much faster a data-oriented designed system was compared to an object-oriented one."
Ok, changed to "What was unknown was if our data-oriented artifact was faster than our object-oriented artifact and if so how much faster."
Original "Before writing this thesis, said things were already known. What was unknown was how much faster a data-oriented designed system was compared to an object-oriented one. Concerning the object-oriented design, dynamic allocation of data during runtime would eventuate into spreading data across the memory. The dynamic allocation of data will be made by following the standards of object-oriented design. For example, if Class A contains Class B, even if Class A resides in the cache line, the data from Class B does not. It contains the a pointer to the address where Class B resides in memory, which will require the CPU to fetch information from that address."
Changed to Before writing this thesis, we suspected that data-oriented artifact would be faster than the object-oriented artifact. What was unknown was if our data-oriented artifact was faster than our object-oriented artifact and if so how much faster.

" The contribution is showing that data-oriented design provides better performance"
Ok, changed to "The contribution is showing that when running benchmarks with an amount of entities over 3500 our data-oriented artifact provided better performance than our object-oriented artifact when using EntityX."

Be a little careful with general statements like these and only draw conclusions based on your particular setting/test case. Keep interjecting that the results/conclusions apply to "our artifact" or something like that (can't really call it a "game" as such…).
Ok, we have changed to so we interject that the results apply to our artifact

———————— REVIEW 2 ——————— SUBMISSION: 2 TITLE: Profiling a prototype game engine based on an Entity Component System in C++ AUTHORS: Dennis Nilsson and Anton Björkman

———— Overall evaluation ———— SCORE: 1 (Good paper, minor modifications strongly suggested) ––– TEXT:

Väsentlig kritik:
- I figure 4 så finns beskrivningen av data-oriented dubblerad och objekt-oriented inte alls vilket gör att jag inte alls vet hur systemen ser ut.
- Fixat bilden
- Vad görs i varje frame? Utan information om detta så är det omöjligt att avgöra om resultatet har någon som helst signifikans för ett riktigt spel!
- Fixat med en sektion Systems som förklarar de olika system och vad de gör i varje frame. Finns även en förklaring i sektionen för Artefakter.
- Vad gjordes när komponenterna i systemet allokerades? Hur är minneslayouten? Kort sagt: Jag vet inte hur data är utlagt i minnet och jag vet inte vad som görs med det. Svårt att förstå innebörden av resultaten!
En bild och text som förklarar memory layouten är tillagd i sektionen 3.3.1 Sen så saknas standardavvikelse etc. för tidmätningen och den brukar vara stor i Windowssystem så jag vill se den. Har skillnaden i tid någon signifikans alls?
Tabell 4 är nu uppdaterad med standardavvikelse.
Det borde finnas en analys avdelning där det förklaras vad prestanda skillnaden beror på. I andra paragrafen i sektionen 5 har vi gjort en analys på detta.

Lite detaljkritik: Kommentarer: Första stycket (och även litegrann i fortsättningen): Många svepande generaliseringar som jag inte håller med om, ska de stå kvar så krävs referenser. T.ex. så är objekt idealiska ur cahce synpunkt om det normala accessmönstret är är att arbeta inom ett objekt. Om ndet normala är att man t.ex. går igenom alla objekt och accessar ett attribut så är objekt ofta sämre.
Ok, vi har tagit bort generaliseringar och lagt till referenser, är det några felaktiga generaliseringar kvar?
Det finns heller ingen självklarhet i att objektorientering leder till att saker är utspridda i heapen, det är vanligt i C++ världen att man ser till att en sorts objekt hamnar tätt samlade på heapen genom memory pools. Polymorfism leder dock till problem med detta.
Vi har ändrat det, det står inte kvar. Se sektion 2.1 t.ex

Vad menas med data-oriented design? Jag har träffat på det begreppet i olika sammanhang, vilken betydelse lägger ni i begreppet? Data oriented design är separation av logik och data och att lägga data kontinuerligt i minnet. https://www.dice.se/wp-content/uploads/2014/12/Introduction_to_Data-Oriented_Design.pdf Sen undrar jag om ni verkligen jmfr data – objekt i största allmänhet ? Är det inte snarare för bara för Entity component system ni jmfr?
Inte i största allmänhet, vi jämför data-oriented implementation med en objekt-orienterad implementation när vi använder ett ECS med andra ord EntityX. Så det vi hade skrivit tidigare blev syftningsfel. Det är ändrat nu.
3.2.3 orginal funktionen beräknar ett random värde inom [b, a+b) med tre decimaler, er funktion beräknar ett random värde inom [0,a). Ok, bra att veta, tack så mycket! Vi ändrade det.

Changes made after receiving feedback on this response letter

Ni säger att ni valt design research – refererar ni till den? T.ex. Hevner eller Peffers?
Ja, referens tillagd Se över språket en extra gång – förstår engelska är knepigt i formella texter. Korrekturläst och tillputsad efter bästa förmåga
Sektion 5 är lång och behöver brytas upp Sektion 5 är nu uppdelad i subrubriker bestående av prelude, performance, modifiability, readability och summary Använd gärna en riktig latex formatmall. Carl-Magnus krävde någon av följande förra året: Ändrat till IEEE journal
Caching/Prefetching är i slutändan lite mer komplext är vad vi antar här (d.v.s. att fetches alltid sker "framåt" i sekvensiell data). Se t.ex. denna tråd (särskilt andra posten): https://news.ycombinator.com/item?id=8542405 Ni skulle kunna nämna, t.ex. under Limitations, att prefetching i slutändan är processorberoende och att vi/ni inte kan veta exakt vad som händer utan att ta den aktuella processorn i fråga i beaktande och/eller göra detaljerad profiling (vilket ni nämner under future work) – det enda vi/ni vet är den slutgiltiga tider per frame (vilket är helt okej), och att denna skillnad rimligen beror på bättre cache-beteende.
Ja, instämmer, lagt till i limitations.
“An important aspect regarding the benchmarks is that we are assuming that cpu prefetches are made sequentially. Since prefetches are cpu dependant we cannot guarantee the actual process of prefetchning made by the cpu in the benchmarks without taking our cpu in account. This is the reason for choosing to only measure the time per frame rendered in milliseconds and conducting all experiments using the same computer.”