# S5214299

# Theo Butler

# Game Engine Technical Report

## Project Overview

This project was to develop a 3D Game Engine using industry standards such as: Cmake, Git, Doxygen. To achieve this, research is required to establish the best method to create the game engine using an Entity Component System (ECS) or a more innovative method like Data Oriented Design (DOD). Firstly, I will analyse the viability of each method and decide which to use, then design the engine based on this, implement it, and create a simple 3D game using my engine.

### Initial Specification

### Firstly, the engine needs to be separate from the game files, otherwise the engine would just be a game. The engine needs to be 3D as most games are now 3D and I should follow the industry standard as closely as possible. I also need to be able to load multiple assets into the game such as: models; textures; shaders and audio to create a game level. The data should be abstracted out to a “level File” as this allows for quick changes and no hard coded components of my game, which then allows for shorter compile times and faster level creation, whilst also allowing the ability to create a multi-level game. Finally, the renderer of this game engine should be separate from the game engine code, which should allow for a clearer and separated appearance where the renderer simply becomes a library to be included and this allows for different renderers to be used.

### Research

Entity Component System (ECS)

ECS is defined as follows: “entity” is defined as “an entity is a distinct object representing an actor in a simulated space”; “component” is defined “a component is a singular behaviour ascribed to an entity and “system “ is defined “a system will iterate many components to perform low-level functions”. (Rungta, 2022)

|  |  |
| --- | --- |
| **Advantages of ECS** | **Disadvantages of ECS** |
| * Shorter and less complicated code (Rungta, 2022). * No programmer required for designers to modify game logic (T-Machine, 2007). * Circumvents the “impossible” problem of hard-coding all entity relationships at start of project (T-Machine, 2007). * Allows for easy implementation of game-design ideas that cross-cut traditional OOP objects (T-Machine, 2007). | * Difficult to apply correctly, easy to misuse. Good components require more thinking about design (Rungta, 2022). * ECS requires writing many small systems which alliterate on potentially huge numbers of entities, which develop the risk of writing very inefficient code (Rungta, 2022). * Each system iterates through the complete list…the iteration cost of all systems together can be very high (Acervo Lima, 2015). |

Data Oriented Design (DOD)

DOD’s main goal is defined as “achieving high performance on modern hardware platforms. Specifically, that means making good use of memory accesses, multiple cores, and removing any unnecessary code” (Llopis, 2011)

|  |  |
| --- | --- |
| **Advantages of DOD** | **Disadvantages of DOD** |
| * Parallelization, allows easier implementation to “split it among multiple threads with minimal synchronization between them” (Llopis, 2009). * Cache utilization, DOD results in “efficient use of the instruction cache because the same code is executed over and over” (Llopis, 2009). * Testing, “when dealing directly with data, it couldn’t be easier to write unit tests” (Llopis, 2009). | * Very different, “different from what most programmers are used to or learned… it requires turning our mental model of the program ninety degrees” (Llopis, 2009). * Challenging, “challenging to interface with existing code, written in a more OOP or procedural way” (Llopis, 2009). * Difficulty, “Good programming is hard, and bad programming is really really easy” (Acton, Mike, 2014). |

Collisions

In writing the code I used the sphere collision response formula excluding the mass component:

s1.vel = vec{v1x}{(m1-m2)/(m1+m2)} + vec{v2x}{(2\*m2)/(m1+m2)} + vec{v1y}

s2.vel = vec{v1x}{(2\*m1)/(m1+m2)} + vec{v2x}{(m2-m1)/(m1+m2)} + vec{v2y}

(Kent, 2019)

When using these formulae, I decided that in my engine I would forgo the inclusion of the mass of objects to speed up the development process. This however would be an easy variable that I could include simply by changing the formulae currently in my sphere collider to a closer formula to that above.

In conclusion, I decided to create my game engine through an Entity Component System as more resources exist online which I can draw knowledge from, and I also understand this system well. However, I will also try to bear in mind some principles of DOD when creating my engine.

### Engine Design

Diagram

Description automatically generated

Figure 1

Figure 1 is a UML diagram showing the top level of design for my ECS. I have abstracted the Renderer and the core of the engine into separate areas. Renderer is empty as it is just a system that the main ECS runs to generate the area and entities.

Figure 2 shows the full design of my game engine in UML format and clearly shows the components inheriting from component and the resource types inheriting from resource.

Diagram

Description automatically generated

Figure 2 V

A picture containing diagram

Description automatically generated Figure 3 shows the structure of the resources. When a type of resource is initialized through the game code, it generates a resource structure to house the specific type of resource. This resource is then sent to Resources, which is a structure made on initialization of core that holds all resources ready for use on an object. This allows materials to be obtained by entities whilst still being abstracted out of the entities.

Figure 3

Diagram

Description automatically generated

Figure 4

Figure 4 shows the Component system; every component type that has been created in the engine can be given to any entity in the initialization of said entity. For example, in each of the Model, Shader and Sound Source components, the materials they point to are initialized first in the initialization loop, and then these components get placed on the entity such that it knows its own resources and can locate them easily. Transform is given to every entity on initialization, in Core when “add entity” is called, immediately after the engine adds a Transform. I took this idea from Unity as it is possible to create a completely empty entity, but it will always have Transform, as every single entity type needs a Transform. Transform holds the location position and rotation of objects and also some vital functions, e.g. inside Transform’s “on Tick” function is all of the movement mechanics according to physics. Sphere Collider is a very useful component which allows objects to bounce off one another, and while this could be included in Convex collider, it would have taken a lot more processing time if the system makes Sphere Colliders from planes. Sphere Collider contains the test for the collisions which is run on every update of the game (every core::start loop) and if that test is successful, it calculates the maths based on several factors such as if the other object is movable or not, and other such information. Finally, this contains the Convex Collider which iterates through the planes of the object to test if another collider has entered them. If that is true, then it will take the incident point and incident direction and use similar maths to the Sphere Collider to redirect the momentum of the other collider.

### Conclusion and future work

Looking at the original project specification, this engine fulfils all the criteria: the game is abstracted out to a level file; it is a 3D engine; it is fully disconnected from the renderer and it follows industry standards. In conclusion, this engine is a success but there is still more work possible to add to the engine, for example: more components doing different things, more input methods, including mass in the physics calculations or a 3D GUI system for building games.

# Bibliography

Acervo Lima, 2015. *Entity-Component-System.* [Online]   
Available at: https://wiki.acervolima.com/entity-component-system/  
[Accessed 19 January 2022].

Acton, Mike, 2014. *CppCon 2014: Mike Acton "Data-Oriented Design and C++".* [Online]   
Available at: https://www.youtube.com/watch?v=rX0ItVEVjHc  
[Accessed 19 January 2022].

Kent, 2019. *SIMPLE SPHERE-SPHERE COLLISION DETECTION AND COLLISION RESPONSE.* [Online]   
Available at: https://studiofreya.com/3d-math-and-physics/simple-sphere-sphere-collision-detection-and-collision-response/  
[Accessed 10 february 2022].

Llopis, N., 2009. *Data-Oriented Design (Or Why You Might Be Shooting Yourself in The Foot With OOP).* [Online]   
Available at: https://gamesfromwithin.com/data-oriented-design  
[Accessed 19 January 2022].

Llopis, N., 2011. *Data-Oriented Design - Now And In The Future.* [Online]   
Available at: https://www.gamedeveloper.com/disciplines/sponsored-feature-data-oriented-design---now-and-in-the-future  
[Accessed 19 January 2022].

Rungta, K., 2022. *Entity Component System.* [Online]   
Available at: https://www.guru99.com/entity-component-system.html  
[Accessed 19 January 2022].

T-Machine, 2007. *Entity Systems are the future of MMOG development – Part 1.* [Online]   
Available at: http://t-machine.org/index.php/2007/09/03/entity-systems-are-the-future-of-mmog-development-part-1/  
[Accessed 19 January 2022].