A Proposal for a Computational Framework Architecture and Design for Massive Virtual World Generation and Simulation

Zintis Erics(✉) and Arnis Cirulis

Faculty of Engineering, Vidzeme University of Applied Sciences, Valmiera, Latvia
{zintis.erics,arnis.cirulis}@va.lv

Abstract. In recent decades, computer games have become a pervasive form of entertainment leading to dramatic growth in complexity and scale in terms of content and employed technologies. However, several flaws in design methodology complicate the development process of massive and highly interactive virtual worlds. This paper proposes a novel architecture for a computational framework to cope with the growing complexity and scale of computer games. The research done should provide both a conceptual basis for a data driven development methodology and a practical implementation of systems designed for integration of various computation technologies. This paper lays out the proposed architecture, discusses concepts and design considerations involved and notes future research required.

Keywords: Computer simulations · Computer games · Virtual worlds · Level of detail · Algorithmic efficiency

1 Introduction

In the last few decades computer games have gained in popularity as a widely available form of entertainment, even more so in the last few years [3]. This rise can be mostly attributed to computer games being able to offer various challenges and immersion into virtual worlds. These capabilities are further enhanced by the continued rise in computational power and availability of consumer grade hardware furthering the graphics fidelity of virtual worlds. However, these advances are approaching the limits of what is reasonable [25]. The combination of these aspects has put focus on the intelligence, or lack thereof, within virtual worlds. More often than not, this intelligence is painted in a negative light. Classical examples of this include artificial intelligence (AI) driven non-player character (NPC) obliviousness to the changing environment or incapability to choose effective tactics in combat. Equally, if not more, problematic can be the lack of appropriate challenge presented by such NPCs often seen as incapable of exploiting obvious weaknesses in the player's tactics or inability to track line of sight

L. T. De Paolis et al. (Eds.): XR Salento 2022, LNCS 13445, pp. 37–47, 2022.
https://doi.org/10.1007/978-3-031-15546-8_3

exploiting targets. These and many other observations led to the realization that existing methods and techniques for computer game AI development and implementation are incapable to efficiently scale with the growing size and complexity of virtual worlds.

This incapability should be considered alongside the various issues concerning the interactivity of virtual worlds. The most prevalent examples are the various implementations of world borders, an artificial and sometimes blatantly obvious limitation of where the players can go and explore. The inability to interact with seemingly functional objects or the limitation of being able to break down walls, chop trees or dig ground only in very specific, designer approved and specially prepared places. Not to mention the impossibility to climb over various kinds of fallen trees, rocks or mountains that would be perfectly reasonable maneuvers in real life. These issues have been prevalent for decades and are often accepted as design or hardware limitations. While there are games that successfully address select issues [9,11,21], these often come with a significant increase in development time and a shift or two in design philosophy often making the solutions one of the selling points and core elements of the game [11,21].

Both of these limitations, in the realms of intelligence and interactivity, often stem from the practice of faking various real-life systems with minimal effort. The fakes will often look akin to the real-life counterparts when viewed or interacted with from angles or in ways anticipated by the designers. The illusion tends to dissipate rapidly when used in an unusual way, giving rise to various bugs and exploits. Fixing such issues often requires patching each individual location or redesigning the involved systems. While the first fix may be attractive in smaller projects, its efficiency in terms of development time falls short in larger ones. On the other hand, the second fix may be overkill for smaller projects, but the only reasonable option for larger ones. Additionally, system redesign often places requirements on or reveals flaws in other systems resulting in a cascade of redesigns. All of these issues intensify when considering moddable games. In such cases, the original designers do not control all the potential content of the game. This requires robust, expressive, performant and well-documented systems and limits the original designers' capability to redesign existing systems requiring more refined implementation from the start.

This methodology of faking systems and various related techniques comes from a time when visual spectacle was one of the main goals and computational resources were much more limited, especially on consumer grade hardware. In the time since the spectacle has been polished to such a degree that further improvements are marginal at best [25], and as a result, focus is being shifted to game design. In addition, the research and advances in algorithm and simulation design are more accessible than ever before. As such, this paper identifies key requirements and proposes an architecture of a data driven simulation framework for generation and operation of scalable virtual worlds. The rest of this document is laid out as follows: the Sect. 2 offers a brief overview of the most notable problems observed and expands on several requirements for the proposed framework. Section 3 provides overview of the main concepts of the proposed framework and Sect. 4 follows up with a discussion of the more obscure yet

fundamental aspects of the architecture. Finally, the Sect. 5 draws conclusions and lays out future work.

2 Problems and Requirements

One of the main reasons for the aforementioned issues is the lack of data; this is an internal consequence especially visible in moddable games. In the intelligence department, this is often seen as NPC unawareness of certain objects, states or entire systems that are represented to the player. A prime example are various animation states or even animation systems in general. Another area to consider is the lack of NPC memory of objects or states, especially in combination with spatial awareness, an example scenario would be tracking or intercepting a target through a complex environment like a city. One final aspect to consider would be awareness of other NPCs, such capabilities would aid in both, strategical analysis for long-term planning and tactical analysis for more complex combat encounters. All of this should be considered in an environment that can present partial or even misleading information, such as limited visibility or use of camouflage, to both players and NPCs.

Meanwhile, in terms of interactivity, one of the hardest problems to solve would be the world borders as these represent the fundamental spatial limitations of the area of content, removing these would require either a planet's worth of content [11,20,30] or procedural generation [11,21]. The problem of both terrain and object destructibility seems simple enough until potential implications on structural stability or object functionality are considered [9,21]. The climbing problem would be simple to solve on its own, albeit it would require redesign of world borders and various other systems to account for the player's location in unusual places [23].

This is only half of the story. One also needs to consider where to acquire the data required and how to use it. In most of the aforementioned problems most of the complexity lies in the use of the data while its acquisition provides only minor to no hindrance. An exception to this rule is the world border problem. Here the complexity shifts almost exclusively to data acquisition. In terms of intelligence, the availability of all the aforementioned data would lead to a severe increase in requirements for computational power and AI complexity. These would increase further because of multiplying possibilities to consider in terms of interactivity.

While the computational power necessary for reasonably accurate solutions to aforementioned problems is available on consumer grade hardware, the issues often lie in its inefficient use. For example, several of the most popular game engines offer limited multithreading support, often restricting engine access to a single thread [8,13]. While this simplifies engine design, it complicates, hinders and requires special considerations in the use of multithreading in games [28,33]. Such design decisions may have been reasonable in previous decades, but with the rise and focus on CPU core counts in the recent years [14,19], it is becoming a problem. Another issue is the object-oriented programming paradigm often employed in game design. While it is the bread and butter of

modern programming practices [17] and easy to use, it often leaves performance on the table when it comes to systems and algorithms working on large amounts of objects [10].

This leads us to the first major requirement of the proposed framework, the ability to use as much performance as possible while still being approachable by developers used to the old ways. The second requirement is maximal modularity. This should aid in both system development and moddability of the resulting games, as well as portability of designed systems. The third and final major requirement is maximal usability outside the gaming industry. This effectively means that the framework should be generic enough to be usable for simulation and computation in other industries and scientific pursuits. One minor requirement should be addressed: the portability of the framework itself. This means that the framework should be developed to maximize simplicity, and minimize and compartmentalize the reliance on and interaction with the engine. This is of secondary priority though and perfect portability is most likely unachievable, albeit such development practices should aid in porting of both the framework itself and various systems developed within it to other game engines.

3 Framework Architecture

The overall architecture of the proposed framework consists of three main concepts: worlds, simulations and layers (see Fig. 1). Each world encapsulates a system of simulations and functions as an ordered directed acyclic graph. Each simulation is a process that executes some kind of computation receiving inputs and producing outputs. The input and output data of a simulation is partitioned into layers, each layer functions as a container holding data of a similar nature.

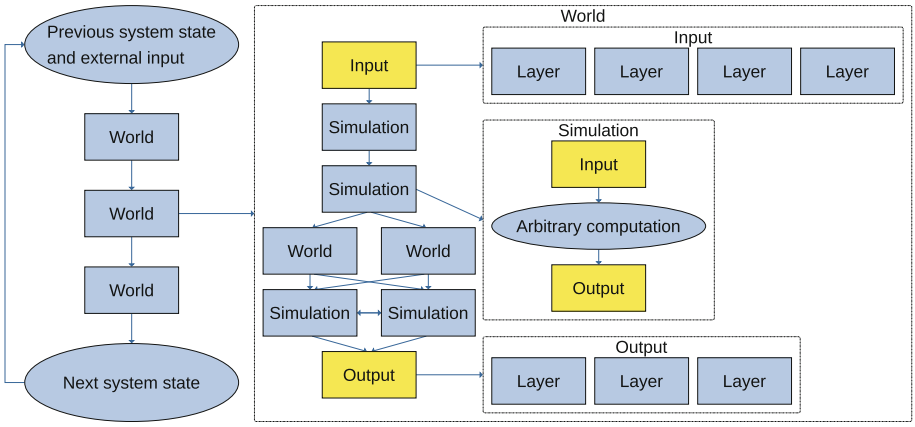

Fig. 1. An overview of the architecture depicting the looping operation of the entire system, nesting of worlds, parallel execution of simulations without interdependencies and linear execution of a distributed simulation.

A world functions similarly to a higher order simulation, it has inputs and outputs, represents an arbitrary computational process and can be executed. This means that a world can be used as a simulation itself allowing for greater flexibility given by a nested approach to the overall system architecture. In this sense, a world represents a higher order process that facilitates access to and orchestration of other processes. To this end, it provides access to the underlying simulations to external processes, executes the simulations when requested and manages the computational graph. In addition, the world is responsible for passing the data, in form of layers, along the computation path while individual simulations are responsible for interpreting and linking of their relevant input and output layers. Such division of responsibility should allow for addition and replacement of individual simulations while still maintaining the possibility to use custom communication formats between knowledgeable simulations.

The simulations themselves can represent any computational process. This includes, but is not limited to, classical simulations using agent, event or stock-based approaches as well as various hybrids. The computation itself can use any means necessary to arrive at the result, for instance, classical algorithms executed on the CPU, parallel algorithms executed on the GPU or special purpose schemes such as ECS. Note that this flexibility extends beyond single processes or even machines, individual simulations could be executed in a distributed fashion with or without knowledge of encompassing worlds. This is aided by the worlds only passing around layers, effectively pointers to arbitrary data collections. Since simulations and layers do data linking and interpretation, this allows for more performant use schemes amongst knowledgeable simulations by exploiting access to the raw data. Additionally, this allows for smart layers hiding the complexities and potentially distributed nature of the underlying data by providing a generic access interface.

The layers provide a data exchange interface that satisfies patterns for generic and more specialized raw access to the underlying data. Moreover, the generic access can hide various peculiarities of the underlying data storage such as distribution over the network or processes and internal data structure complexities as well as the use of various caches to aid efficiency. Meanwhile, the raw access serves as a more direct route to the underlying data storage exposing all the peculiarities and controls of the distribution process. This effectively means that the underlying data storage can use any means, regions or entities from a structural standpoint and RAM, VRAM, HDD, SSD or network resources from a technology standpoint.

Some general notes need to be addressed. First, while this architecture allows mixing of various technologies through generic interfaces, it also emphasizes performance gains by using a single technology throughout facilitated by the raw access interfaces. It is probable that these smart layer interfaces will aid in the use of various linking techniques amongst knowledgeable simulations increasing the overall performance by cutting out unnecessary back and forth transformations. Second, this architecture uses similar principles to already established technologies such as ECS [1,18,32] and GP GPU [31] and can be considered a

more generic approach encompassing both. This framework will aid in practical use and combination of various technologies exploiting benefits of each while minimizing drawbacks, as well as providing common terminology and metrics for discussing various technicalities.

4 Design Considerations

In addition to the aforementioned general structure, several other aspects need to be considered, the first of which is the temporal integration methodology. It must be noted that one of the expected main use cases of the proposed framework involves various simulations of drastically different temporal scales. This effectively means that a world may be executing varied simulations concerning everything from cosmological scale mega structure and galaxy formation to the processes of individual cells and tissues. Such a variety of processes has a matching variety of effective temporal resolutions, meaning that a scheduler is required to execute all simulations at the right times to maintain both simulated process integrity and overall system efficiency. Additionally, each world and simulation may have its own scheduler. This is aided by the nested overall structure, but also means that each simulation must use the same generic scheduling interface. This also implies that each individual simulation may use its own internal scheduling methodology as long as it conforms to the external interface. To achieve this at least two scheduling strategies will be provided, a more discrete one using the closest discrete bins based on powers of two [29] and a more arbitrary one using an unbalanced binary tree with a smart iterator. Of course, any other strategy may be implemented and used, as long as it conforms to the external interface.

Similarly, the spatial integration methodology also requires consideration. In this case, the same foreword can be referred to for a visualization of the vast differences in the spatial scale of simulated processes. However, this case is more complicated than temporal integration since various simulations may use vastly different methods for partitioning the simulated region. So far various methods have been observed, chiefly amongst them are the classical grid-based approaches [6], various octree-based implementations [2], use of Voronoi tessellation and Delaunay triangulation [29] and simple collections of discrete entities [29]. It should be noted that the proposed framework needs to be able to support use cases in both two and three-dimensional space as well as different coordinate systems local to larger entities such as planets and moons. Together all of this means that the external interface of layers needs to support generic value lookups based on coordinates and a reference coordinate system while providing some additional information such as coordinate system extents and translation functions between local and global coordinate systems. These considerations raise concerns regarding potential loss of precision across different coordinate system interfaces. As an extreme example, the coordinates necessary for an astrophysical simulation spanning hundreds of parsecs cannot be put on the same 64-bit float as the ones measured in nanometers and used in bacterial simulations, certainly not without loss of precision. Moreover, while indefinite

precision numerical schemes that could allow for unified storage exist [26], these are almost exclusively implemented in software and prohibitively expensive in terms of computational power [12]. Additionally, the highest performance hardware available to consumers prioritizes support of 32-bit floats [15,24] leading to a necessary choice between performance and precision.

So far, we have discussed the proposed framework's support for two types of dynamic resolutions, temporal and spatial. Both of these concepts have for decades been used as ways to improve simulation efficiency by cutting out unnecessary detail [2]. In a sense, this goal is the same as that of level of detail (LOD) systems used in computer games, also, for a considerable amount of time [27]. These LOD systems increase the performance of games mainly by limiting the resolution of graphical assets used for far away objects while maintaining acceptable graphical fidelity. As such we consider the concepts of temporal and spatial resolutions to be just single aspects of larger concepts, temporal level of detail (T-LOD) and spatial level of detail (S-LOD) respectively. This assumption aids in two aspects, firstly it provides umbrella terms to group various not only resolution-based techniques together and, secondly, it provides a flexible terminology to measure and manipulate the fidelity of simulations. The regard of these concepts, alongside the terminology, have paved the way for another: functional level of detail (F-LOD). This is the concept of changing the executed simulation logic based on the required fidelity of the region under consideration. While, at first glance, this may seem similar to S-LOD, F-LOD merely builds on top of it. Where S-LOD partitions space into regions of varying sizes, F-LOD decides what logic should be executed upon them, meanwhile T-LOD determines how often this execution should happen. Note that some games [4,16,22] and simulations [34] have used similar techniques, but to our knowledge, there has been no attempt of consolidating these approaches under a single banner.

In the previous discussion, we briefly mentioned fidelity, and while the concept itself is clear, it expresses how close a replica is to the original, the question lays in the measurement of perceived fidelity. Perceived fidelity is of importance since it effectively measures what level of fidelity can be expressed given the observer's interface. The key factors to consider here are the limitations of the observer's interface. These introduce a point beyond which any increase in fidelity does not produce any noticeable effect [7]. In classic LOD systems, this is measured using geometric and pixel errors relative to the highest fidelity assets available. The system tries to minimize these errors while remaining within bounds of its limited resources [5]. This raises an important question: how should the errors be measured in temporal, spatial and functional aspects of various simulations? While the exact measurement unit is unimportant, what is of paramount importance is the relative stability and consistency of the measurements used by various simulations. While the exact considerations and guidelines of this system remain unclear and a subject to further study what is clear is that significant considerations should be directed to the observer attachment to certain aspects of the virtual world. In this sense attachment effectively measures how noticeable would be the loss of the aspect under consideration. Again, this is of paramount

importance since the focus of the entire framework is to provide believable and consistent virtual worlds on a massive scale within limited resources. Under such conditions exploration is inevitable, which will lead to the generation of new or LOD increase of the existing regions of the world. This will also lead to pressure on the limited resources and the need to lower the LOD or entirely leave something out. One of the first things to leave would be something of the least consequence for the observers. The currently unresolved question is how to measure this potential consequence, this attachment in a stable and consistent manner. Again, this is a subject for further study.

The preceding outline paints the proposed framework as a system of simulations for partial or complete generation and evolution of interactive environments with potential to use both random and hard-coded initial conditions. This brings up the discussion regarding two of the main algorithm design considerations: evolution and generation. In this sense the evolution is the application of iterative logic to the simulation state advancing it to the next one in time. This methodology can be employed to both create content and provide interactivity. Meanwhile, generation is the application of initialization logic to create a simulation state. This methodology can only be used to create content. The evolution methodology is mostly used for simulations in more scientific endeavors due to its temporal causality and problem domain complexity while the generation methodology is mostly applied to computer games due to the lack of computational resources for complex simulations and necessity for perfect realism. In this context let us recall that the proposed framework may need to increase LOD or generate entirely new regions in response to observer exploration. As such at least parts of the region state, if not the entire region, would need to be created using the generation methodology. This effectively means that to ensure consistency with already existing, potentially evolved regions the generation method used would need to rapidly bring the state of the region to the desired point in time or LOD while still maintaining the desired fidelity. Again, depending on the complexity of the simulations and temporal interdependencies of the region states involved it may not be feasible to evolve the region due to computational resources or temporal logistics. This means that each simulation with multiple LODs or indefinite bounds would need at least two algorithms, one for evolving the regions normally and the other for increasing LOD or generating new regions on the spot. While for simpler simulations with fewer time steps repeated application of evolution logic to generate new regions could be performant enough, that cannot be said for more complex simulations, worlds that are advanced significantly far enough in time or LOD increase. Moreover, it may be difficult to provide matching generation algorithms for more complex simulations, as such this is a subject for further study.

Altogether it can be seen that the proposed framework would specialize in the creation and operation of partial virtual worlds in a sense that these worlds would not be complete due to the constraints of computational resources. As such these worlds would have more realized regions and ones that are more imaginary in the sense of higher and lower simulation LODs respectively. Effectively the

created virtual worlds would not be complete realizations, rather they would be more akin to a moving window serving as an interface with everything out of its bounds, potentially blurred. In this regard the key challenges lie in designing a system that could adapt to observer interactions in a performant enough way to keep up with the required fidelity while remaining flexible enough to support moddability and incorporation of both standard and cutting-edge technologies.

5 Conclusion

This paper outlined and exemplified some systematic problems concerning common patterns employed in the design methodology of computer game systems. For brevity, we classified these into two avenues: intelligence and interactivity. We noted several performance, modularity and portability requirements that should be addressed to construct a framework to successfully alleviate these issues. Next, we proposed such a framework employing data driven development methodology that should fulfil the noted requirements and sketched its core architecture in terms of worlds, simulations and layers. We also noted several considerations to take into account during the proposed framework's development phase, namely, temporal, spatial and functional integration, perceived fidelity and observer attachment and the mixed use of both evolution and generation algorithms.

In addition, we noted several areas for further study, specifically, the measurements of error terms in temporal, spatial and functional integration, the evaluation of observer attachment and matching evolution and generation algorithms. Albeit it must be noted that these are largely overarching conceptual directions, as such most of the following research should concern implementation technicalities. In this regard we already foresee a number of potential avenues chiefly concerning massively parallel algorithm design, implementation and notation as well as other, more miscellaneous questions.

References

1. Andrea Catania: Godex, March 2022. https://github.com/GodotECS/godex. Accessed 29 Mar 2022
2. Barnes, J.H., Hut, P.: A hierarchical o(n log n) force-calculation algorithm. Nature **324**, 446–449 (1986)
3. Barr, M., Copeland-Stewart, A.: Playing video games during the Covid-19 pandemic and effects on players' well-being. Games Cult. **17**(1), 122–139 (2022). https://doi.org/10.1177/15554120211017036
4. Bay 12 Games: Slaves to Armok: God of blood Achapter II: Dwarf fortress. https://www.bay12games.com/dwarves. Accessed 1 Mar 2022
5. CesiumGS: 3D Tiles, February 2022. https://github.com/CesiumGS/3d-tiles/blob/main/3d-tiles-reference-card.pdf. Accessed 1 Mar 2022
6. Chan, K.H., Im, S.K.: Fast Grid-Based Fluid Dynamics Simulation with Conservation of Momentum and Kinetic Energy on GPU, pp. 299–310, December 2017. https://doi.org/10.1007/978-3-319-71598-8_27

7. Deering, M.: The Limits Of Human Vision, October 2000
8. Gerke Max Preussner: East coast devcon 2014: Concurrency & parallelism in ue4 - tips for programming with many CPU cores, April 2015. https://www.slideshare.net/GerkeMaxPreussner/concurrency-parallelism-in-ue4-tips-for-programming-with-many-cpu-cores. Accessed 29 Mar 2022
9. Ghost Ship Games: Deep rock galactic. https://store.steampowered.com/app/548430/Deep_Rock_Galactic. Accessed 1 Mar 2022
10. Ali, H.: Why OOP is not performance efficient, May 2020. https://www.linkedin.com/pulse/why-we-need-move-from-oop-hadid-ali. Accessed 29 Mar 2022
11. Hello Games: No Man's Sky. https://www.nomanssky.com. Accessed 1 Mar 2022
12. Turner-Trauring, I.: Massive memory overhead: Numbers in python and how NumPy helps, October 2021. https://pythonspeed.com/articles/python-integers-memory. Accessed 29 Mar 2022
13. Bonastre, J.: Why should I use threads instead of coroutines?, November 2016. https://support.unity.com/hc/en-us/articles/208707516-Why-should-I-use-Threads-instead-of-Coroutines. Accessed 29 Mar 2022
14. Rupp, K.: 42 years of microprocessor trend data, February 2018. https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data. Accessed 29 Mar 2022
15. Carbotte, K.: Nvidia's new Titan v pushes 110 teraflops from a single chip, December 2017. https://www.tomshardware.com/news/nvidia-titan-v-110-teraflops,36085.html. Accessed 29 Mar 2022
16. Ludeon Studios: Rimworld. https://store.steampowered.com/app/294100/RimWorld. Accessed 1 Mar 2022
17. Medi Madelen Gwosdz: If everyone hates it, why is OOP still so widespread?, September 2020. https://stackoverflow.blog/2020/09/02/if-everyone-hates-it-why-is-oop-still-so-widely-spread. Accessed 29 Mar 2022
18. Michele Caini: EnTT, March 2022. https://github.com/skypjack/entt. Accessed 29 Mar 2022
19. Bailey, M.: Parallel programming: Moore's law and multicore, March 2022. https://web.engr.oregonstate.edu/~mjb/cs475/Handouts/moores.law.and.multicore.2pp.pdf.Accessed 29 Mar 2022
20. Mobius Digital: Outer wilds, https://store.steampowered.com/app/753640/Outer_Wilds. Accessed 1 Mar 2022
21. Mojang Studios: Minecraft. https://www.minecraft.net. Accessed 1 Mar 2022
22. Netcore Games: Tales of Maj'eyal, https://te4.org. Accessed 1 Mar 2022
23. Nintendo: The legend of Zelda: Breath of the wild. https://www.zelda.com/breath-of-the-wild. Accessed 1 Mar 2022
24. NVIDIA Corporation & affiliates: CUDA C++ Programming Guide, March 2022. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. Accessed 29 Mar 2022
25. Gallaga, O.L.: Is it really worth upgrading your PC for 'ultra' gaming graphics?, February 2021. https://debugger.medium.com/is-it-really-worth-upgrading-your-pc-for-ultra-gaming-graphics-634c61c6f75f. Accessed 29 Mar 2022
26. Python Software Foundation: Built-in Types, March 2022. https://docs.python.org/3/library/stdtypes.html. Accessed 29 Mar 2022
27. RasterGrid Kft.: GPU based dynamic geometry LOD, October 2010. https://www.rastergrid.com/blog/2010/10/gpu-based-dynamic-geometry-lod. Accessed 29 Mar 2022
28. Meredith, R.: Simple multithreading for unity, July 2017. https://richardmeredith.net/2017/07/simple-multithreading-for-unity. Accessed 29 Mar 2022

29. Springel, V.: E pur si muove:galilean-invariant cosmological hydrodynamical simulations on a moving mesh. **401**(2), 791–851 (2010). https://doi.org/10.1111/j.1365-2966.2009.15715.x
30. System Era Softworks: Astroneer. https://store.steampowered.com/app/361420/ASTRONEER. Accessed 1 Mar 2022
31. Unity Technologies: Compute shaders. March 2022. https://docs.unity3d.com/2022.2/Documentation/Manual/class-ComputeShader.html. Accessed 29 Mar 2022
32. Unity Technologies: Entities overview, March 2022. https://docs.unity3d.com/Packages/com.unity.entities@0.50/manual/index.html. Accessed 29 Mar 2022
33. Blanco, V.: Multithreading overview March 2021. https://vkguide.dev/docs/extra-chapter/multithreading. Accessed 29 Mar 2022
34. Weinberger, R., Springel, V., Pakmor, R.: The Arepo public code release. Astrophys. J. Suppl. Ser. **248**(2), 32 (2020). https://doi.org/10.3847/1538-4365/ab908c