УДК 004.42+004.94 http://doi.org/10.15407/pp2023.03.30

A.A. Bernatovych, I.V. Stetsenko

MODULE-BASED SCIENTIFIC PHYSICS ENGINE ARCHITECTURE

The ever-evolving landscape of digital simulations demands innovative frameworks to achieve both realism and efficiency. The research highlights the issues of modern software architecture for physics simulation. The new architecture was developed to mitigate scalability and flexibility issues. The essence of the proposed architecture resides in the convergence of three pivotal concepts: the modular physics engine, the physics pipeline, and the Entity Component System (ECS) pattern. The modular physics engine represents a paradigm shift in simulation design. Compartmentalizing functionalities into modules, this approach fosters flexibility and reusability, enabling efficient construction of specialized simulations. The physics pipeline orchestrates simulations through structured stages, analogous to graphics pipelines. It guides dynamic forces, collisions, and interactions, optimizing resource use and integrating custom systems for accuracy. Complementing these, the ECS pattern decouples data and behavior, facilitating the construction of user defined physical pipeline comprised of loosely coupled modules. Combined with the modular physics engine and physics pipeline, ECS forms a comprehensive approach for complex physics simulations.

Keywords: Computer Simulation, Software Architecture, Physics Engine, Physics Pipeline.

Introduction

Physics simulation plays a pivotal role in advancing scientific understanding and exploration across various disciplines. Its significance extends beyond mere computational modeling, as it enables researchers to delve into complex physical phenomena, unveiling insights that would otherwise remain hidden. Many natural systems are characterized by complex interdependencies and behaviors that are challenging to comprehend through traditional analytical methods alone. Physics simulations allow scientists to model these systems, providing a platform to observe emergent properties, intricate dynamics, and phenomena that may be challenging to observe directly in the real world.

Simulations offer a controlled environment where scientists can rigorously test hypotheses and validate theories. By accurately reproducing experimental conditions, simulations enable researchers to explore the outcomes of diverse scenarios, aiding in the confirmation or refinement of theoretical frameworks. Furthermore, certain physical phenomena that manifest in extreme environments, such as outer space or subatomic scales, are inaccessible for direct observation. Physics simulations offer a virtual laboratory to study these conditions, enabling the investigation of phenomena beyond the limitations of current technology.

The main problem of modern physics engines is the solidity of the system they provide. In most cases, the user cannot manipulate parts of the engine. There are cases when the user would benefit from turning on and off some parts of the simulation pipeline, available in several engines. Still, none of the popular engines allow to change simulation stages or implement a substitute for them [1].

In some cases, only rigid body simulation is needed, which allows the user to remove every other part of the engine, effectively reducing any additional computational or memory cost generated by unnecessary stages [2]. A more complex simulation is needed in other cases, but only for one specific simulation part. For example, the fluid simulation may be implemented ‘outside’ the engine, still reusing all other stages.

1. Physics model architecture overview

Physics simulation works by employing mathematical models and computational techniques to replicate the behavior of real-world

physical systems in a virtual environment. The process involves several key steps that collectively simulate the dynamics, interactions, and behaviors of objects based on fundamental physical principles.

The first step involves creating a mathematical representation of the physical system to be simulated. This includes defining the objects, their properties (such as mass, size, and material properties), and the forces or interactions that affect them. Central to the simulation are the equations of motion, encapsulating the manner in which objects behave. These equations, often built on well-established principles like Newton's second law of motion (), delineate how forces induce changes in an object's acceleration, velocity, and position over time. Numerical integration comes into play next. Equations of motion are numerically integrated over small time intervals, enabling projections of an object's future position and velocity. Diverse numerical integration techniques, including Euler integration, Runge-Kutta methods, and Verlet integration, facilitate these predictions.

Crucial in the simulation is collision detection. Algorithms are deployed to discern instances where objects intersect or closely approach one another. This insight underpins the subsequent calculation of collision effects and assures the accuracy of the simulation. Upon collision, algorithms ascertain alterations in velocity, direction, and deformation for objects involved. This computation considers factors such as elasticity, mass, and impact velocity. The presence of various forces, like gravity, friction, and springs, necessitates force calculation. These forces are determined based on object attributes and their interactions with both other objects and the environment.

Progressing in discrete time steps is intrinsic to the simulation process. Each step entails solving equations of motion to predict the new state of objects at the forthcoming time increment. The choice of time step magnitude strikes a balance between simulation accuracy and computational demands. In intricate simulations, multiple iterations often occur within each time step. Rigid body simulations, for instance, might require iterations to resolve contact forces and constraints. The simulation operates as a perpetual feedback loop, continuously updating object states through calculated forces, collisions, and interactions. With each iteration, the simulation advances incrementally, painting an evolving picture of dynamic behavior.

In the end, the outcomes of the simulation can be observed either in real-time visualization or through subsequent post-processing, generating visual depictions of object movement, interactions, and intricate dynamic phenomena.

The proposed architecture of the Physics Engine allows users to create their simulation stages and construct an engine that suits the needs of developer or scientist the best. Figure 1 represents the structure of the proposed Physics Engine.

The main structural elements of Physics Engine are Physics Models, which represent the world with all entities in it, and rules of physics simulation created by the combination of simulation stages. The modular architectural design has been further developed to address the shortcomings found in widely adopted layered and Service-Oriented Architecture (SOA) frameworks. By incorporating a more refined level of granularity and emphasizing the division of services, these architectural approaches have effectively facilitated the creation and enhancement of intricate software systems that might have otherwise posed significant challenges during design and development. The use of it might provide such benefits as scalability, extensibility by third-party, and reusability and lead to the overall reduction of core application size [3].

The Entity Component System (ECS) pattern aligns seamlessly with the idea of encapsulating specific functionalities within individual modules. Instead of conventional tightly-coupled module interdependencies, ECS promotes the separation of concerns. This separation is achieved through the entity-based structure, where components encapsulate specific attributes and behaviors while systems orchestrate the behavior of entities possessing certain combinations of components [4].

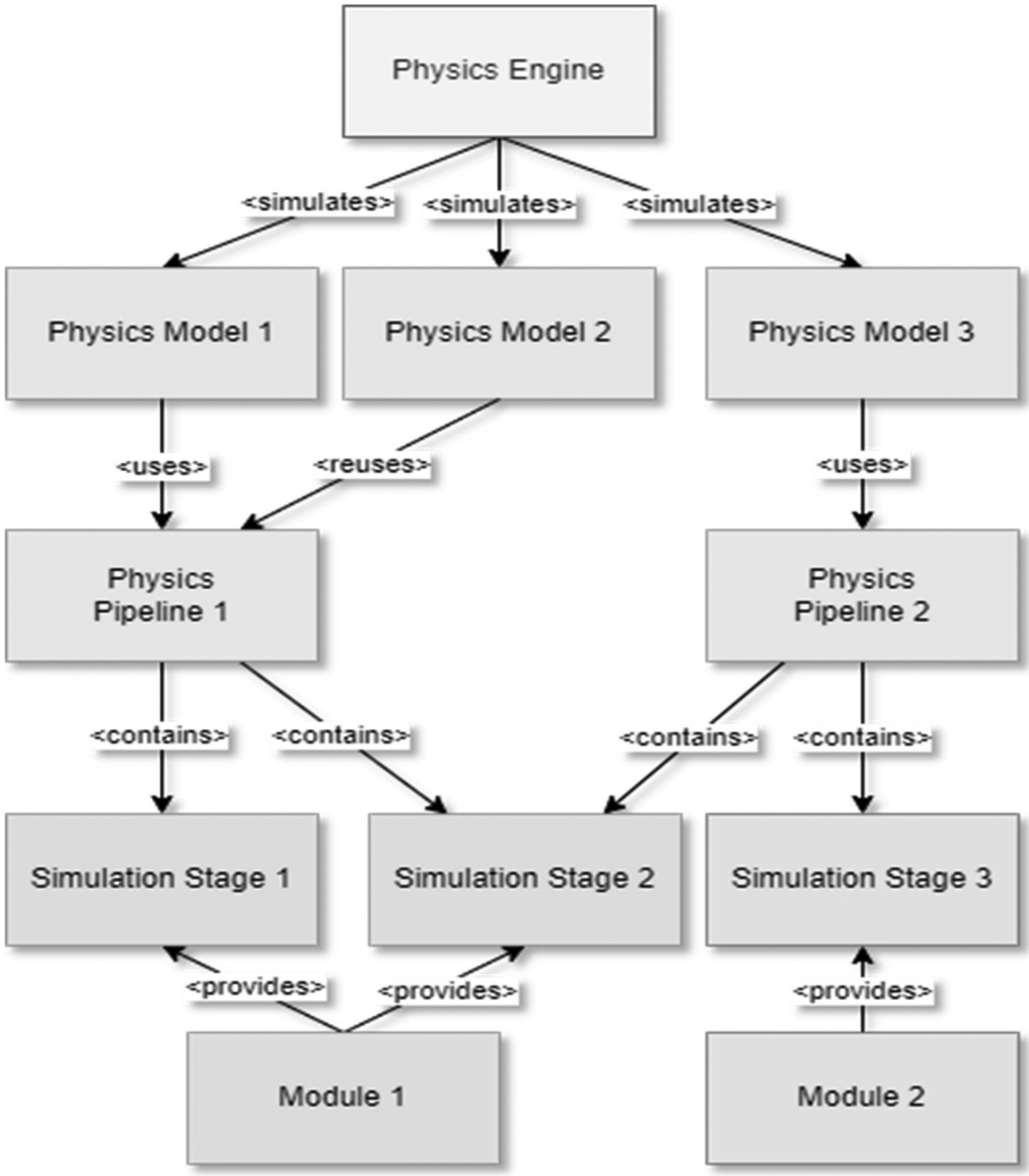

Fig. 1. Structure diagram of the proposed Physics Engine

In the ECS-based simulation environment, entities take center stage as the fundamental building blocks that compose the simulated environment. Each entity represents an object or element within the world. By adopting this entity-centric perspective, the world is broken down into discrete entities, allowing for a more modular and intuitive representation.

Components, another essential aspect of ECS, define the attributes and behaviors of these entities. In the context of world definition, components encapsulate various aspects of an entity's identity and behavior. Properties such as position, orientation, mass, and collision characteristics are captured within components. This separation of data from behavior enhances clarity, maintainability, and the ability to adapt the simulation to different scenarios.

The systems in ECS orchestrate the interaction and behavior of entities based on their components. These systems are responsible for processing specific aspects of the simu-

lation, such as physics calculations or collision detection. They ensure that entities with certain combinations of components interact in a meaningful and realistic manner, shaping the dynamics of the simulated world.

1.1 Physics module

The Physics Module represents a dynamic and extensible class library, capable of being loaded at runtime to enrich the functionality and configurability of the Physics Model. This innovative approach offers a powerful way to introduce new components and stages seamlessly without modifying the core simulation code. The module acts as a repository of reflection data [5], storing crucial information about the various components and simulation stages essential for configuring the Physics Model.

With the ability to load the Physics Module dynamically, researchers and developers can introduce custom components and stages tailored to specific physics simulation requirements. These components may include specialized collision detection algorithms, advanced material properties, or sophisticated force fields that address unique scenarios not covered by the default simulation setup. Similarly, the stages within the module can extend the simulation pipeline, introducing additional data processing steps that are required for diverse use cases.

Moreover, the reflection data stored within the Physics Module provides a comprehensive overview of the available components and stages, making it easier for developers to query and configure the simulation programmatically. The module acts as a knowledge repository, allowing developers to discover available options, examine their properties, and dynamically assemble the Physics Model to suit specific simulation scenarios.

1.2 Physics model simulation update

As there is no simple way of continuous simulation of a physics system, the simulation process is divided into small time steps with fixed . The update flow of the model is defined by a physics pipeline that configures the order of execution for each computation stage (Fig.2).

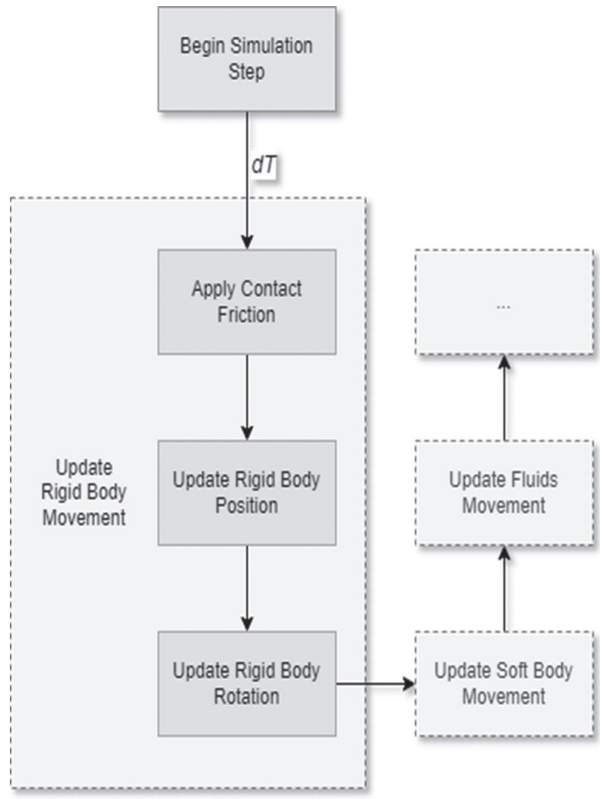

Fig. 2. Diagram of Physics Processing Pipeline

Termed the ‘Physics Pipeline’, this alternative approach integrates physics simulation seamlessly into the computational framework, aligning it with the stages traditionally found in the graphics pipeline [6]. While the graphics pipeline focuses on rendering visual elements, the Physics Pipeline extends this concept to encompass the simulation of dynamic forces, collisions, and other physical phenomena that unfold over time. The pipeline has a fixed entry point that determines the start of the simulation step. Other stages are chained to the entry point. Each stage computes and updates the physical properties of an entity. Stages are processing the entire set of all entities based on the components that they contain. For example, the “update of rigid body position” stage will only update entities with attached rigid body components.

Extending the concept further, the utilization of physical pipelines introduces a profound level of efficiency and resourcefulness by enabling their preservation and subsequent application across various Physics Models. By encapsulating a specific set of simulation

stages, configurations, and behaviors within a pipeline, developers can effortlessly transport this blueprint from one simulation context to another. This affords a consistent and systematic means of orchestrating simulations, irrespective of the specific model being employed. This flexibility proves especially impactful when dealing with comparable or interconnected physics phenomena across diverse contexts. Consider scenarios in which the fundamental principles dictating motion, interactions, and dynamics remain consistent, while the simulation setting or parameters undergo adjustments. In such instances, the ability to employ a well-established and refined physical pipeline for various purposes significantly speeds up the development process, reduces unnecessary repetition, and maintains a coherent framework for simulations.

Accelerating the physics pipeline is a crucial endeavor to enhance the efficiency and performance of simulations. This can be achieved through a series of strategic optimizations and advancements that target different stages of the pipeline.

Utilizing a uniform and contiguous collection of components holds the potential to yield significant advantages, greatly enhancing cache locality and reducing memory access times. By organizing components within a contiguous layout, the memory footprint aligns more closely with the access patterns of the processing units. This coherence allows for more efficient data retrieval, reducing the need for costly memory fetch operations. As a result, cache hits become more frequent, optimizing the utilization of high-speed cache memory and diminishing the latency associated with memory access. The resulting reduction in memory access times and enhanced cache locality synergistically contribute to an overall enhancement in computational efficiency.

Furthermore, the utilization of parallel computing architectures, particularly multicore CPUs, emerges as a formidable technique to accelerate the execution of physics computations. By proficiently distributing computational responsibilities across numerous processing units, simulations are poised to achieve substantial enhancements in speed and responsiveness. Introducing asynchronous processing allows independent stages of the physics pipeline to run concurrently. For example, collision detection can be performed in parallel with constraint solving, minimizing idle time and optimizing resource utilization.

Notably, the synergistic integration of spatial partition algorithms with parallelization yields an even greater leap in acceleration. Through the implementation of spatial partitioning techniques, such as octrees or grids, the computational load associated with collision detection and interaction calculations can be significantly alleviated. By excluding entities situated far apart in space from collision evaluations, unnecessary computations are circumvented, yielding a marked enhancement in overall performance.

2. Results

Proposed architecture of the physics engine allows users to configure their simulation more flexibly due to the following principles:

– **Incorporation of Additional Processing Stages.** Even in cases where certain features may not be inherently present in the default implementation, the modular nature of the approach permits the incorporation of such features as separate libraries that can be linked and utilized at runtime.

– **Selective Removal of Unnecessary Stages.** In the context of prototype models aiming to substantiate hypotheses, streamlined calculations might suffice. The modular setup ensures that users are only charged for the components they employ.

– **Dynamic Replacement of Existing Stages.** In scenarios demanding more intricate computational methodologies, the capability to replace pre-existing processing stages proves to be remarkably advantageous.

The adoption of the ECS pattern introduces novel prospects for extending entity attributes, a prospect that was previously constrained by the rigid definitions of simulated objects.

The following sections describe the new approaches of using physical engine, which become possible due to new architectural design.

2.1 Fully customized physics model

With the architecture detailed in this article, the concept of constructing custom simulation pipelines becomes an achievable and empowering prospect. This dynamic architecture embraces the utilization of separate simulation systems, allowing for the precise tailoring of pipelines to match the specific requirements of the Physics Model (Fig.3).

One of the important advantages of this approach lies in its flexibility. By incorporating separate simulation systems, developers gain the ability to create simulation pipelines that are finely tuned to their intended purposes. This means that components of the simulation that are not directly relevant to the current Physics Model can be selectively excluded from the pipeline. This granularity enhances performance by reducing unnecessary computations and optimizing the simulation for efficiency.

Moreover, the modularity inherent in this architecture empowers developers to effortlessly experiment with different combinations of simulation systems. This experimentation can lead to the discovery of optimal pipeline configurations that produce accurate and efficient results. It also simplifies the process of incorporating new simulation systems or improving existing ones, as modifications can be made to individual systems without disrupting the entire pipeline.

Custom simulation pipelines crafted using this architecture can be thought of as carefully assembled toolsets. Each system within the pipeline represents a specialized tool designed to address a particular aspect of the simulation. This modularity and adaptability ensure that the simulation remains versatile, accommodating a wide range of scenarios and use cases. Furthermore, this approach aligns seamlessly with the notion of data-driven design. Each simulation system processes data in a data-oriented manner, promoting efficient memory access patterns and enhancing overall performance. The data-driven nature of the architecture also enables better parallelization, harnessing the full potential of modern multicore processors.

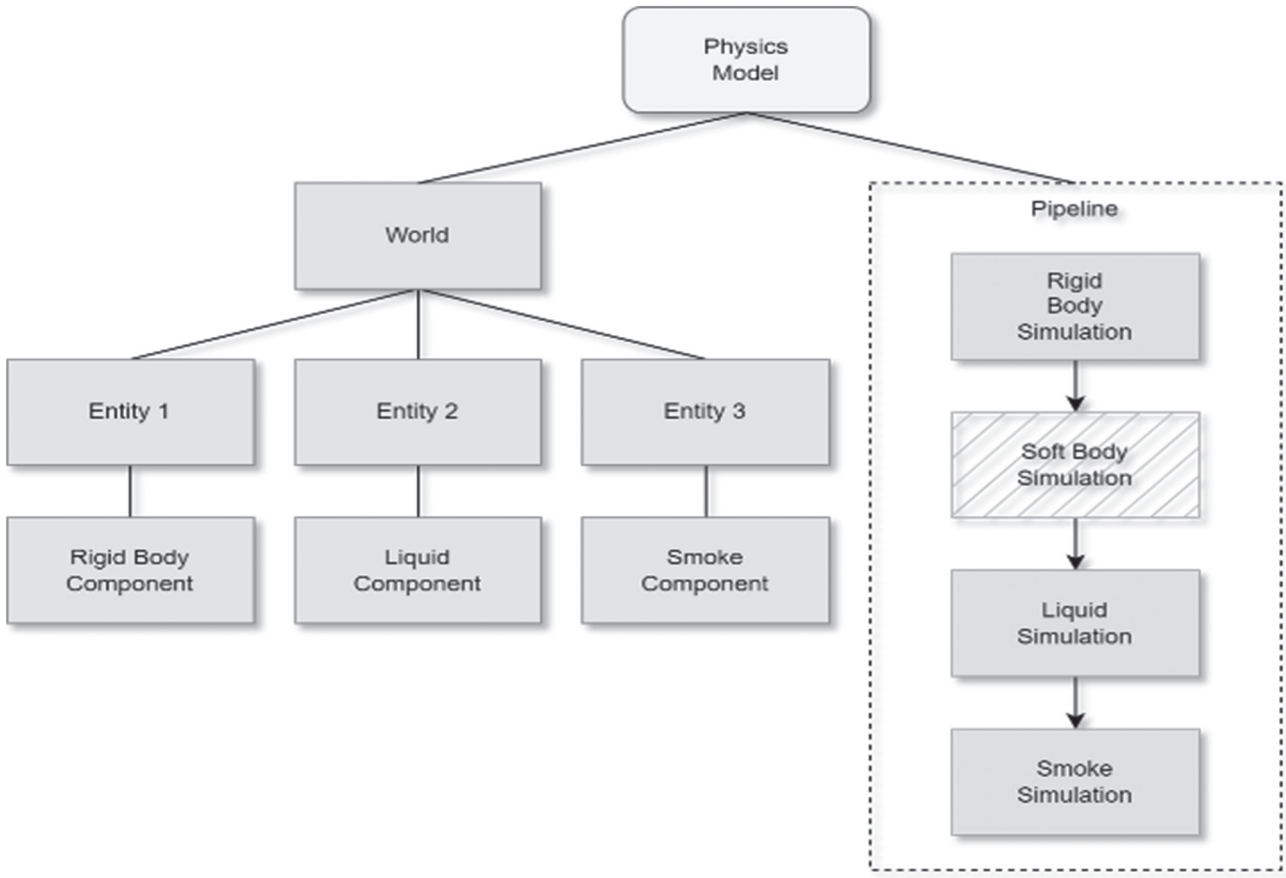

Fig. 3. Diagram of fully customized Physics Model

2.2 Partial physics model override

The framework described in this article provides a means to enhance existing physics engines by employing separate custom simulation systems [7]. As an example, it allows us to modify or extend certain aspects of the physics simulation within well-established engines to better suit our needs.

To illustrate, consider physics engines like PhysX, which come with their own built-in capabilities for simulating rigid bodies and fluids. By implementing the ideas from this framework, we can potentially replace or enhance these default simulations using dedicated custom simulation systems. This means that instead of solely relying on the pre-existing rigid body and fluid simulations within PhysX, we have the option to integrate our custom systems that might offer improved performance or more precise results, depending on the context.

To put this into action, we would create wrapper classes that encompass the existing physics engines. These wrappers manage the core operations of the engine while allowing for the seamless integration of our custom simulation systems. Additionally, each engine could incorporate a component that handles different types of simulations, excluding those that we plan to customize (Fig.4).

In practice, when a simulation event occurs, the customized engine takes precedence. If there is a need to adjust a specific simulation, the related entity is directed to the custom engine's simulation process. After that, the entity is then passed to the custom simulation system that we have designed for the targeted enhancements.

This approach ensures a cohesive and integrated simulation process. By blending the strengths of the native engine with the adaptability of our custom systems, we can fine-tune or expand specific simulation behaviors to better align with specific requirements. This framework effectively offers a way to bridge the capabilities of established engines and the flexibility of custom systems to create physics simulations that are more tailored and versatile.

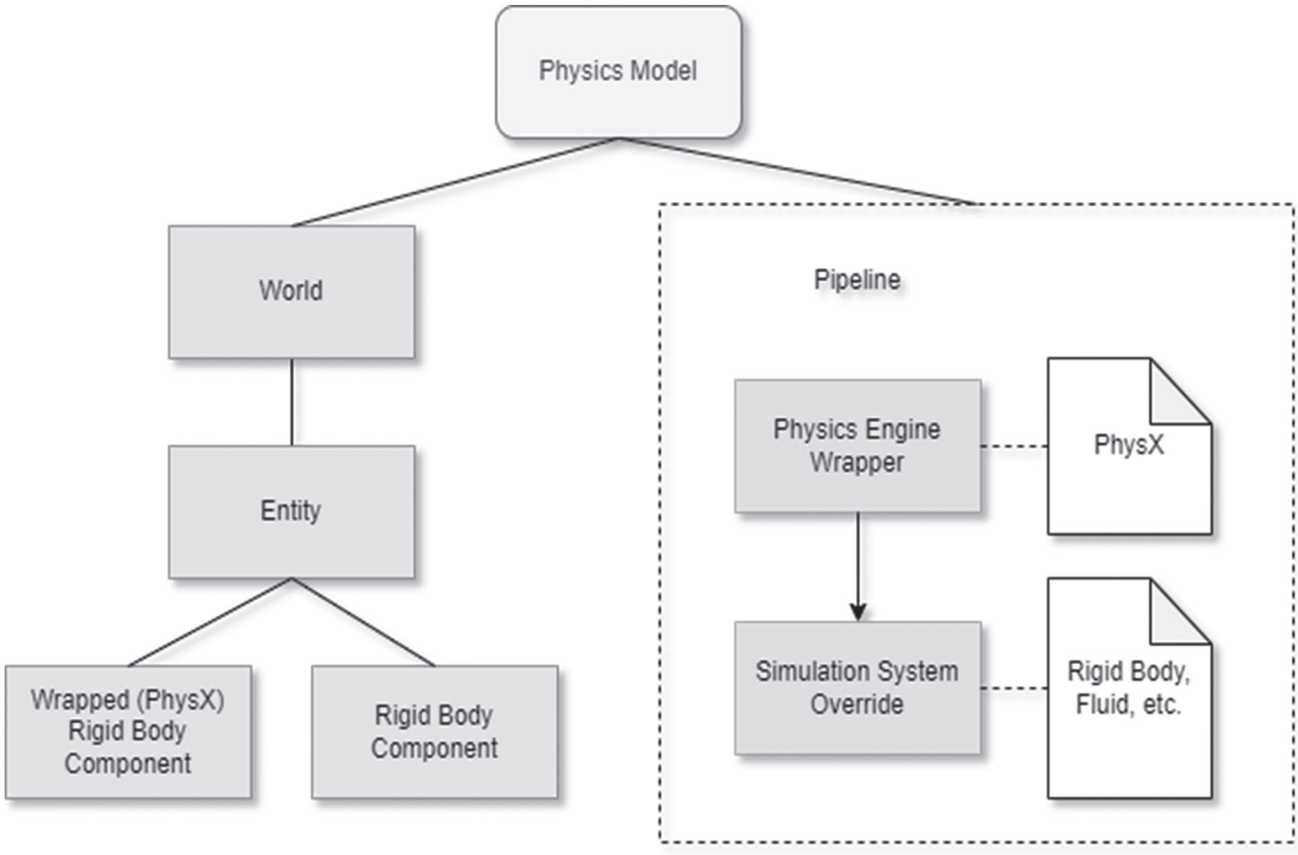

Fig. 4. Example of overriding Rigid Body and Fluid Simulation for PhysX

2.3 Selective physics model

The concept of allowing diverse systems to perform identical simulations for distinct Entities while employing varying computational methods introduces a realm of intriguing trade-offs between precision and performance. This innovative approach acknowledges that achieving the highest level of precision might involve computationally intensive methods, whereas opting for more efficient techniques could involve a certain level of approximation.

For instance, consider a scenario where a range of Entities exists within a virtual environment, each with unique attributes and behavior characteristics. This approach allows for a deliberate selection of simulation methods based on the specific requirements of each Entity. Some Entities might seamlessly integrate with Components from Rigid Body Simulation System 1, which is renowned for its computational precision but may entail higher computational costs. In contrast, other Entities could opt for Components from Rigid Body Simulation System 2, leveraging its more efficient methods, albeit with a slight compromise on precision (Fig.5).

This adaptive strategy offers a nuanced balance that developers can leverage to optimize their simulations based on the intricacies of the Entities involved. Entities with complex physical behaviors that necessitate accurate results might benefit from the more precise simulation approach. Meanwhile, Entities with less intricate interactions could benefit from the computational efficiency of an alternative method.

3. Related works

The landscape of physics simulation and computational modeling has seen a plethora of notable contributions that have enriched our understanding of complex physical phenomena. This section provides an overview of pertinent works that have significantly impacted the field, highlighting their methodologies and insights.

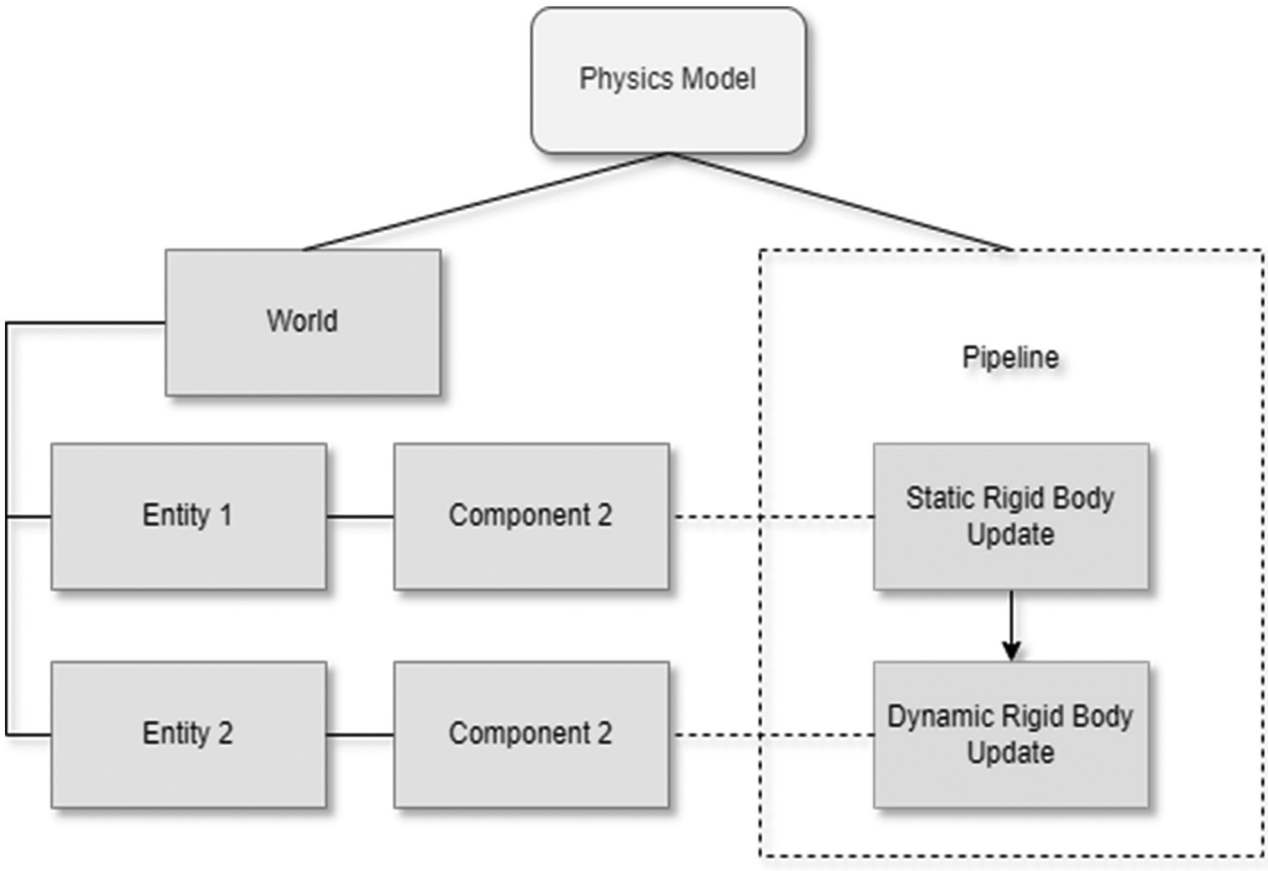

Fig. 5. Example of selective Rigid Body Simulation

Prominent physics engines have demonstrated innovative approaches to simulation. The Bullet Physics Engine [8] stands out for its robust collision detection and rigid body dynamics capabilities, establishing it as a standard choice across various applications. The NVIDIA PhysX engine [9] has gained recognition for real-time physics simulations and GPU acceleration, empowering interactive simulations in gaming and virtual environments.

The Entity Component System architecture has revolutionized entity management in simulations. The Artemis Framework [10] showcased ECS benefits in game development, facilitating efficient entity composition and behavior control. The EnTT library [11] furthered this trend by offering a high-performance ECS solution with a focus on usability and extensibility.

Exploring physics laws-centric perspectives, the xPheve physics engine [12] prioritizing a faithful replication of natural physics laws. This distinctive approach aims to achieve simulations with heightened authenticity, aligning more closely with real-world phenomena. While the xPheve approach aims to achieve authenticity through physics laws replication, it might face performance trade-offs due to the potentially computationally intensive nature of directly simulating complex physics equations. Achieving a high level of precision might require significant computational resources, potentially impacting real-time simulations. Computational-centric approaches often allow developers to optimize and fine-tune simulations for specific use cases. The xPheve approach, focused on physics laws, might provide less control over optimizing performance for specific scenarios, potentially leading to suboptimal outcomes in certain situations.

Conclusions

In conclusion, the Physical Module is a valuable addition to the physics simulation toolkit, providing a flexible, scalable, and dynamic means of expanding the functionality and configurability of the Physical Model. By loading the module at runtime and leveraging its reflection data, developers can introduce custom components and stages, ensuring that the simulation system remains adaptable, innovative, and on the cutting edge of scientific and technological advancements.

The Entity Component System pattern revolutionizes how the world is defined in a physics simulation. Through entities, components, and systems, ECS offers a structured and efficient framework for capturing the essence of the simulated environment. This data-driven approach enhances the realism, adaptability, and complexity of the simulated world, making ECS an indispensable tool for constructing sophisticated and immersive physics simulations. Components play a vital role in the functioning of physics simulation modules by providing essential data representation, promoting modularity, enabling decoupling of logic, facilitating customization and extensibility, supporting efficient data processing, and fostering a component-based architecture. By leveraging components effectively, physics simulation modules can achieve high performance, flexibility, and maintainability, making them essential tools for exploring and understanding complex physical phenomena in diverse research and application domains.

Following the course set by the article, the future steps in the research is to investigate the practical application of this architecture in different contexts, which could unveil challenges and the innovative solutions devised to overcome them would offer valuable practical wisdom. Furthermore, examining potential limitations and trade-offs associated with this architecture is essential for a well-rounded discussion. This might encompass considerations related to computational overhead, memory management, and the intricacies of coordinating modular components within a dynamic system. As this architecture benefits from modularity, fostering an open-source community for collaborative development could be instrumental. Future research might focus on creating platforms that facilitate sharing components, best practices, and continuous improvement.

References

1. Bernatovych, A., Stetsenko. I. (2023) Methods and software of physical simulation. *Adaptive Systems of Automatic Control* 42(1), pp. 130-40. doi: 10.20535/1560-8956.42.2023.279104 (In Ukrainian).
2. Longshaw, S., Turner, M., Finch, E., Gawthorpe, R. (2010) 'Analysing the use of Real-time Physics Engines for Scientific Simulation: Exploring the Theoretical and Practical Benefits for Discrete Element Modelling', *ACME 2010 Proceedings of the 18th Annual Conference Annual Conference of the Association of Computational Mechanics in Engineering*, Southampton, Association of Computational Mechanics in Engineering, 29-31 Marth 2010. pp. 199-202. doi: 10.13140/2.1.3212.7048.
3. Mbugua, S. T., Korongo, J. and Mbuguah, S. (2022) On Software Modular Architecture: Concepts, Metrics and Trends. *International Journal of Computer & Organization Trends* 12(1), 3-10. doi: 10.14445/22492593/IJCTT-V12I1P302.
4. Wiebusch, D., Latoschik, M. E. 'Decoupling the entity-component-system pattern using semantic traits for reusable realtime interactive systems', *2015 IEEE 8th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, Arles, France, 2015, pp. 25-32, doi: 10.1109/SEARIS.2015.7854098.
5. Smith, B. C. (1982) *Procedural reflection in programming languages*. Thesis (Ph.D.). Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science.
6. Liu, F., et el. (2009) 'CUDA renderer: A programmable graphics pipeline', *ACM SIGGRAPH Conference and Exhibition on Computer Graphics and Interactive Techniques in Asia*. 16-19 December 2009. doi: 10.1145/1667146.1667189.
7. Zheng, Z, et el. (2020) Hybrid Framework for Simulating Building Collapse and Ruin Scenarios Using Finite Element Method and Physics Engine. *Applied Sciences* 10(12):4408.
8. He, H., et el. (2019). 'Simulation of Realistic Particles with Bullet Physics Engine', *E3S Web of Conferences,* 92, 14004. doi: 10.1051/e3sconf/20199214004.
9. NVIDIA.developer. PhysX. Available at https://developer.nvidia.com/physx-sdk (Accessed 19 August 2023).
10. Artemis Framework. Available at https://github.com/bsimser/artemis-framework (Accessed 19 August 2023).
11. EnTT. Available at https://github.com/skypjack/entt (Accessed 19 August 2023)
12. Nourian, S., et el. (2006) 'XPHEVE: An Extensible Physics Engine for Virtual Environments', *2006 Canadian Conference on Electrical and Computer Engine*ering, Ottawa, ON, Canada, 2006, pp. 1546-1549, doi: 10.1109/CCECE.2006.277848.

Одержано: 17.08.2023

Про авторів:

Бернатович Анатолій Олександрович,
аспірант Національного Технічного
Університету України
«КПІ імені Ігоря Сікорського».
Кількість наукових публікацій
в українських виданнях – 2.
https://orcid.org/0000-0002-8403-6363

Стеценко Інна Вячеславівна,
доктор технічних наук, професор,
професор кафедри інформатики
та програмної інженерії НТУУ
"КПІ імені Ігоря Сікорського".
Кількість наукових публікацій
в українських виданнях – понад 130.
Кількість наукових публікацій
в зарубіжних виданнях – 15.
Індекс Гірша – 12.
http://orcid.org/ 0000-0002-4601-0058

Місце роботи:

Національний технічний університет
України «Київський політехнічний
інститут імені Ігоря Сікорського»,
03056, м. Київ,
проспект Берестейський 37.
Тел.: (044) 236-9651
e-mail: abern.kpi@gmail.com,
stiv.inna@gmail.com