Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 104 (2021) 792–797

www.elsevier.com/locate/procedia

54th CIRP Conference on Manufacturing Systems

Scope and delimitation of game engine simulations for ultra-flexible production environments

Liliana Zarco[a,*], Jörg Siegert[a,b], Thilo Schlegel[a,b], Thomas Bauernhansl[a,b]

[a]*Institute of Industrial Manufacturing and Management, University of Stuttgart, Allmandring 35, 70569 Stuttgart, Germany*
[b]*Fraunhofer Institute for Manufacturing Engineering and Automation, Nobelstrasse 12, 70569 Stuttgart, Germany*

* Corresponding author. Tel.: +49 711 685 61872; fax: +49 711 685 51872. *E-mail address:* liliana.zarco@iff.uni-stuttgart.de

Abstract

Game engines are middleware that allows the integration of multiple resources turning them into assets with graphic information, programmed content and physical simulation parameters. These features meet the requirements for modelling, visualizing, simulating and controlling complex production systems. Therefore, an approach based on game engines to manage the user-perceived complexity was proposed. This paper compares 24 game engines and identifies suitable modelling environments for production based on their technical and market-development features. Then, a modular machine was integrated in two game engines. The results of scopes and limitations regarding functionality, scale, physics simulation accurateness, resource redundancies and visualization are presented.

Keywords: ultra-flexible production systems; game engine; production simulation; digital factory

1. Introduction

Ultra-flexible production systems have a large number of variables to control. Various concepts for ultra-flexible production systems have been developed and validated at the IFF. These concepts are based on a standardized coordinate system that specifies the temporal-local state of the order, processes, modules and resources [1]. This system is the basis of the Matrix Fusion Factory (MFF), which enables the fusion of the real factory and the digital factory[2].

Ultra-flexible production environments require platforms capable of representing, modeling, controlling and visualizing highly complex systems. Game engines have evolved rapidly in recent years, proving that they can represent very complex systems in a reliable and user-friendly way. In addition, game engines cover the software control requirements for ultra-flexible production systems: reusability, compounding (hierarchy), resource compatibility, etc. [3].

For these reasons, game engines offer great potentials not only in entertainment but also in other areas of industry and technology. This paper has the following goals:

1. Identify the potentials of game engines for modeling, simulation and control of ultra-flexible production systems.
2. Analyze the limitations for the use of game engines considering functionality, scalability, granularity, accuracy of physical simulations, numbers of variables, redundancy and visualization.
3. Analyze the capabilities of game engines for scenario-based simulations in a production environment.
4. Compare and select suitable game engines for modelling, controlling, visualizing and simulating flexible production systems. A suitable game engine should: (a) allow the integration of 3D models, (b) include a robust physics engine, (c) support artificial intelligence (AI), (d) support networking and online interactions between several clients/users, and (e) have an available license and continuous active producer and open community support in

order to get quick solutions to potentially arising problems and take advantage of community developed tools and experiences.

This paper compiles game engines scopes and their technical features, as well community impacts and market developments; contributing to the determination of potentials and limitations of game engines for modeling, simulation, control and visualization of ultra-flexible modular production systems, and especially for their complexity management.

Section 2 describes different approaches for game engines and videogames attributes and classifications. Then, a discussion about the potentials and limitations of game engines is discussed in Section 3, as well as a summary of 24 game engines and their features. Section 4 presents the use case of modelling a modular robot in Unity3D and Unreal Engine and a performance evaluation to validate the described potentials and limitations. Finally, Section 5 summarizes the conclusions and future works of this research.

2. Classification and attributes of game engines

Game Engines are software platforms with an extensible data-driven architecture that allow to reuse resources for new games or applications without major modifications [4]. The basic functionalities of game engines include the following attributes: graphics rendering, simulation of mechanical behavior, kinematics, momentum, and collision detection generally through a physics engine, scripting integration, multithreading capability, scene management, integration of resources (animations, sounds, images, textures, etc.), communication capability through different protocols, artificial intelligence, networking and multiplayer, as well as the integration of assets, plugins and cross compiling. All these attributes (see Fig. 1) make game engines powerful environments of system representation, control and visualization.

2.1 Game Engines attributes

There are several Game Engines with similar characteristics both for the market and for their technical development features. So far there is no formal classification, but there are the following segments for game engines:

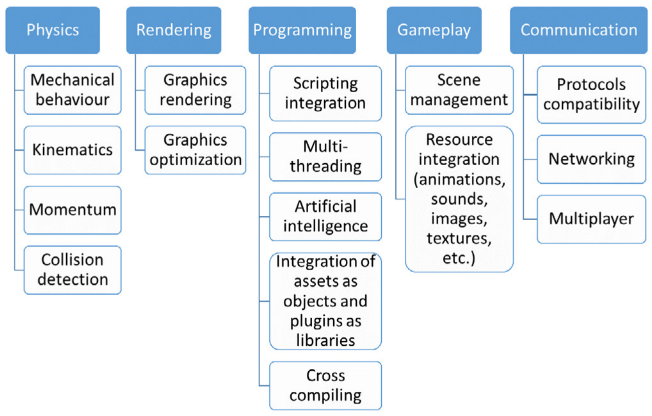

Fig. 1. Main attributes of game engines

- by the **development environment**: 2D, 2.5D and 3D;
- by the **app base**: PC Games, Mobile Games, Console Games like PlayStation (PS) or Xbox, TV Games, Others; and
- by **business model**: Open Source, Freeware, based on income, Fix Priced License.

In literature, there are several comparisons about game engine performance [5–7]. [8] compares five game engines and describes evaluation criteria into five groups:

- basic features of the engine including price;
- support, flexibility, interoperability and usability;
- engine system requirements and installation;
- functionality and the ability to export; and
- multimedia support and working environment.

Furthermore, [9] has presented a robust framework for comparing 24 serious game engines in three attribute classes:

- critical attributes include 2D/3D support, deployment platforms, development platforms and licensing costs/terms;
- required attributes like graphics rendering, audio/visual fidelity, artificial intelligence, physics and networking; and
- project based attributes considering CAD platform support, availability of import/export of assets, world (level) editor availability, content creation, scripting languages, learning curve and accessibility.

Table 1 summarizes four game engine studies [7, 8, 10, 11], includes also additional features from the respective game engine producers' websites and highlights their critical points. Hence, game engines without 3D-model integration, robust physics engines, AI support, networking and online support and available license are not suitable for ultra-flexible production systems modelling. Furthermore, game engines were organized top-down according to their market development and community support (forums, social networks, and learning support available). As a result, Unity3D and Unreal Engine were identified as suitable game engines for modelling, controlling and simulating complex production environments.

2.2 Classification and attributes of videogames

Serious games can be classified by purpose (e.g. decision-making, simulation, knowledge sharing, data collection, motivation and training games), scope or audience (e.g. education, military, marketing, politics, scientific research, etc.), gameplay or game mechanics (avoid, match, destroy, create, manage, analyze, solve, move, select, etc.), audience, (graphic user interface) GUI and deployment device technologies [6, 12].

Gamification mechanics for videogames are [6]: (1) progression mechanics, implies defined goals, skills mastery, accomplishments and personal score; (2) feedback gamification, includes bonuses, challenges and rewards; and (3) behavioral mechanics like exploration, discovering, ownership, community collaboration and competition.

The aforementioned video game characteristics and mechanics of gamification are a highly flexible production environment similar to the requirements for modeling. This relationship is described in section 3.7.

3. Game engine potentials and limitations

Gamification for controlling ultra-flexible production systems through game engines leverages elements of video games and their development environments in the technical context of the factory with the goal of creating precise control and a positive experience for the user. By creating a heightened visual experience, operators can effectively perform tasks and even build new skills. Game engines can exploit resource redundancy, enabling efficient integration of new and diverse production resources into simulations. These resources can be scaled or detailed according to the needs of the simulation. In addition, the integration of metadata and reliable physical simulations enables the generation and evaluation of valuable engineering parameters. Game engine compatibility enables the development of applications for numerous end-user devices, increasing the agility of information in the static and dynamic systems of a factory. In this way, employees can have the information they need at the right time and in the right place.

3.1 Potential of fast development and data redundancy

The term "game engine" arose in the mid-1990s, starting with the well-defined separation between core software components (3D rendering system, collision detection and audio system) and game design (art assets, game worlds and game rules) [4], allowing to create new art and game resources with minimal changes. Nowadays, game developers can reuse significant portions of its core software components and invest in custom software engineering. Game engines enable the implementation and use of design elements and techniques, game mechanics and analytics and software architecture in serious applications to improve user experience, engagement, effectiveness and productivity [6].

Table 1. Summary of meta-analysis and self-research about game engines features and performances

Game Engine Features	2D/ 3D	Work flow editor	World editor	Physics Engine	AI support	Networking online	Programming skills	Scripting	Platform	License
Unity3D	2D/ 3D	Yes	Yes	PhysX, Box2D, Unity Physics and Havok	i.a. Bots and FSM	Yes	Low	C#, JavaScript	PC, Web, Mobile, Xbox, PS	Free, Paid
Unreal Engine	3D	Yes	Yes	Chaos Physics	i.a. FSM	Yes	Low	C++, Blue-prints	PC, Mobile, Xbox, PS	Free, Paid
Cry-Engine	3D	Yes	Yes	Proprietary	Yes	Limited	Low	C#, C++, Lua	PC, Mobile, Xbox, PS	Free, Paid
Game Maker	2D	Yes	Yes	Box2D and Liquid-Fun	Yes	Yes	Low	GLM	PC, Mobile, Xbox, PS	Free, Paid
Neoaxis Engine	3D	No	Yes	Bullet	Yes	Limited	Medium	C#	PC	Free
Game-Salad	2D	Yes	Yes	Rigid-body physics	Yes	Yes	Low	Lua	PC, Mobile, Web	Free, Paid
Cocos2D	2D	No	Yes	Chip-munk	Yes	Limited	Medium	C++, JS	PC, Mobile	Free
Torque	2D/ 3D	No	Yes	PhysX	Yes	Yes	Medium	C++	PC, Web	Paid
Unigine	3D	No	Yes	Proprietary	Yes	Limited	Medium	C#, C++	PC, Web, Mobile, Xbox, PS	Paid
Quake 4	3D	Limited	Yes	Proprietary	Yes	Limited	High	C++	PC	Free
Construct2	2D	Yes	Yes	box2dweb, Cocoon-JS	Yes	Yes	Low	JavaScript	PC, Mobile, Web	Paid
Shiva 3D	2D/ 3D	Yes	Yes	ODE physics engine	Yes	Yes	Medium	Lua, C++, ObjectiveC	PC, Xbox, Mobile, PS, Web	Paid
Cafu (MIT)	3D	Limited	Limited	Proprietary and Bullet	N/A	Yes	Medium	Lua	PC, Mobile	MIT Free
Amazon Lumber-yard	3D	Yes	Yes	PhysX	N/A	Yes	Medium	C++	PC, Web, Mobile, Xbox, PS	Free
Panda-3D (Disney)	3D	5	5	Proprietary, Bullet, PhysX	i. a. FSM	Yes	High	C++, Python	PC	BSD but N/A
Delta 3D	3D	Yes	Yes	ODE	Yes	Limited	Medium	C++	PC	Free
Source Engine	3D	Yes	Yes	Havok, Rubikon	Yes	Limited	Medium	C++	PC, Xbox	Paid
Frost-Bite	3D	N/A	N/A	N/A	Yes	Limited	Low	N/A	PC, Web, Xbox, PS	N/A
SnowDrop	2D/ 3D	N/A	N/A		Yes	Yes	Low	N/A		
Dunia 2	3D	No	Yes		Yes	Yes	Low	N/A		
Fox	3D	Limited	Limited		Yes	Yes	Low	N/A		
Chrome Engine	3D	Limited	Limited		Yes	Yes	Low	N/A		Removing Support
ID tech 5/6	3D	No	Limited		Yes	Yes	High	C++		N/A
Adventure Game Studio	2D	N/A	N/A	None	No	Limited	N/a	JAVA, C#	Windows	

Furthermore, the use of game engines for industrial and engineering applications is increasing, boosting the development of them especially for their technical characteristics and the improvement in physical simulations, as well as the optimization of rendering and latency times.

3.2 Data architecture: potential for scalability and granularity

Game engines allow implementing hierarchical data architectures, reuse resources but also to rearrange its relations. Unified Robot Description Format (URDF) is a standardized XML format for representing a robot model [13]. The URDF has been used for several models, including ROS (Robot Operating System), which is an open source framework for robot programming.

Hence, an IFF data architecture was developed, extending the URDF parameter and including also environment data. Table 2 shows the parameters of the basis control application developed by IFF and highlighted the common parameters between standardized URDF and the IFF description format. In this way it is possible to detail different elements, modules, module accumulations, machines, and environmental components with the appropriate granularity and scale them or reuse the models or resources for other production systems.

Table 2. IFF description format

Category		Parameter
Modul	*ID*	
	Kinematics	Inertial
	CPS	Origin, geometry and material
	Joint	Origin, parent, axis, calibration and dynamics
		Child
	Machine safety	Limit
Environment	*Origin*	
	Collision	Origin and geometry
	Scope	Limit geometry
	Human Safety	Emergency and safety spheres

3.3 Limitations of integrated physic engines

Physics engines are frameworks with reusable libraries that contain the required algorithms and data structures for computing physics effects [14]. In a physics engine, each simulation consists of four steps [15]:

1. Force and torque generators
2. Motion simulator using an object´s mass, acceleration and calculated forces
3. Iterative collision and contact detector between multiple static or movable objects
4. Iterative collision resolver solving overlapping according to physical characteristics and considering friction and restitution

Physics engine requirements could be divided into five aspects: first, the physics engine had to be able to detect collisions between relatively large numbers of objects (between 250 and 500) at interactive frame rates from 60 Hz (~16ms) to 1 kHz (1ms); second, the computed responses of dynamic objects to collisions had to result in realistic behavior; third, the movement constraints between objects should obey their defined limits; and fourth, a physics engine should support the realistic simulation of a screw mechanism, performing a stable collision and friction computation for complex geometric objects [15]. Previous comparisons between physics engines (*Spheres, Bullet, Havok, Newton, ODE* and *PhysX*) considering collision computation performance, accuracy of collision response, fixed constraint stability, 1-DOF constraint stability, interpenetration between rigid bodies and advanced collision and friction have shown that there is no physics engine that generally performs best of any given task [15].

The scope of each game engine with respect to the fidelity of the physical simulation varies greatly from one game engine to another. For example, Unreal integrates *Chaos Physics* for driving its physical simulation calculations and to perform all collision calculations. *Chaos Physics* has the following features: in-engine fracturing tools, dynamic strain evaluation, interactive simulation caching, physics parameter fields, collisions and materials support [16]. Unity3D has an optimized combination, different physics engines using *Box2D* for bi-dimensional environments, *PhysX* for tridimensional physics, *Havok* Physics and a self-developed *Unity Physics* for Unity's Data-Oriented Technology Stack (DOTS) [17]. *PhysX* has one of the best performance for stable constraints and little inter-penetrations, while advanced bounces and friction could lead to unrealistic behavior under certain conditions. On the other hand, *Havok* is highly optimized for speed [15].

3.4 Potential of metadata integration and hardware-in-the-loop

The resource versatility of some game engines like Unity3D allows the creation of a metadata file. In fact, the described IFF description format is also a metadata xml file. These metadata files can also contain further information about the elements, modules, module compounds or production assets, compensating some of the limitations mentioned in section 3.3. By means of metadata files it is possible to observe energy yields of electricity, or physical behaviors that are not observable with the integrated physic engine. The data that can be integrated into the simulations through metadata are, for example for electronic performance indicators, magnetic fields, energy flow, costs and other strategic KPIs.

Using this approach, it is possible to perform accurate virtual commissioning and hardware-in-the-loop simulations to develop and test complex real-time embedded systems [18]. Hence, the connection between the real control system and the simulation system cannot distinguish between a real machine and a simulated one. Therefore, it is also possible to use some real machine parts and simulate the remaining [14].

3.5 Potential and limitations of resource diversity and compatibility

The potential for the integration of diverse resources varies greatly from one game engine to another. The resources that can be integrated are generally three-dimensional models with metadata, sound, animations, scripts, information packages, among others. The compatibility of these resources between game engines has not been studied. Based on the developments at IFF, two limitations regarding resource compatibility have been recognized: the import and export of metadata through format changes for CAD models; and the connection sockets between different programming languages.

The first limitation involves the integration of limited three-dimensional formats for Unity3D and Unreal (e.g. *Filmbox* .fbx). Most of three-dimensional models used for industrial applications are developed in engineering CAD modeling software, e.g. SolidWorks. Since they are not compatible, it is necessary to export the models through a third tool. For this purpose another tool like 3dsMax or Maya should be used, which require the necessary metadata settings for export to .fbx and thus import into Unity3D or Unreal.

The second limitation is the integration of resources such as scripts or APIs developed in different programming languages. For example, in Unity3D it is possible to program scripts in JavaScript (JS) or C#; while Unreal integrates by default APIs for C++, Blueprint and Python. To solve this, both game engines have solutions that allow cross compiling. However, the integration of sockets for cross compiling could increase the latency.

3.6 Versatile end device deployment and personalized information

Game engines allow to compile and run final applications on different platforms. For example, Unity3D supports deployment for Android, iOS, Linux, OSX, PS4, PS5, tvOS, Win, Xbox One, Series XS, Nintendo Switch, Google Stadia, WebGL, AndroidTV, Magic Leap, Oculus Rift, ARCore and Microsoft HoloLens (see **Fehler! Verweisquelle konnte nicht gefunden werden.**). This versatility derives in a big use potential of end devices in production systems. End devices can make data acquisition and information availability more flexible at any required time or place. For example in computers, servers, mobile phones, tablets, or even smart watches, and virtual reality glasses. The available information could be sensor communication, data acquisition, maintenance information, hardware monitoring and personalized information based on user profiles.

3.7 Potential for controlling and visualizing ultra-flexible production modules, environments and networks

The requirements for industrial engineering applications, specifically for the control of cyber-physical production systems have been investigated at the IFF institute previously.

According to the previous analysis, Game Engines have a higher potential than other engineering software or physics engines independently. Game engines have enough flexibility to model and accumulate different production means and their elements and integrate visual elements for intuitive user interaction, as well as the capacity to display complex information in a transparent way [19].

Beyond covering the mentioned requirements, game engines can structure and scale resources hierarchically, allowing the modularization of digital production means. In addition, the integration of an environment in which the means of production can interact among themselves and with the dynamic and static external elements is possible through the creation of scenarios. Game engines allow the creation of complex interactive “worlds” or environments. Furthermore, massive multiplayer online games (MMO) can connect and manage a large number of customers (machines or users) in the same application.

Therefore, it is plausible to use game engines for controlling, planning and simulating several production scenarios.

4. Use Case

Based on the identification of suitable game engines described in Section 2, an application of control and visualization of a modular robot was developed in Unity3D (see Fig. 2) and Unreal (see Fig. 3). To corroborate the scopes and limits investigated and proposed in the previous sections, a simulation of a robot of six degrees of freedom was implemented. This simulation was originally developed in Unity3D and migrated in Unreal for the comparison the performance and resource compatibility. The simulation consists of the control of the six motors that correspond to each degree of freedom of the robot. The user can modify the angle and speed of each of the motors or the final position of the end effector and the total motion speed of the robot through inverse kinematics. The robot must be able to avoid collisions with the base or other objects that may be within reach.

Firstly, both game engines have a project and version management, Unity3D has *Unity Hub*, while Unreal does it through *Epic Games*. When creating a new project, in Unity3D

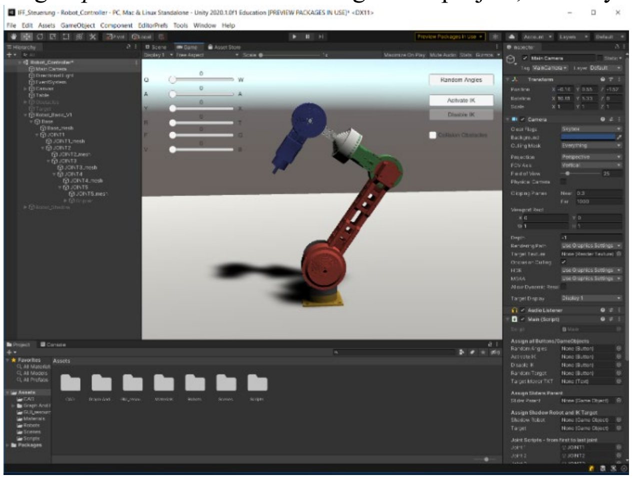

Fig. 2. Robot simulation in Unity3D environment

you can choose between 2D and 3D, while Unreal only has new project or 3D project templates. However, when importing the .fbx into Unreal, the engine separated the hierarchy of elements without grouping them into one object. While Unity3D keeps the hierarchy determined in 3dsMax. Furthermore, the metadata corresponding to the materials have not been assigned in Unreal, while Unity3D automatically assigns them using shaders and own materials. Both game engines are evolving to match even more accurate physics simulations. Table 3 summarizes the scope and limitations of these two game engines, considering highly flexible production resource simulations.

Table 3. Summary of scopes and limitations of Unity3D and Unreal

GAME ENGINE	UNITY3D	UNREAL
Functionality	Both game engines can integrate functions and hierarchies for the accumulation of production modules, as well as the corresponding programming to simulate their behavior	
Scalability	Both game engines allow spatial scaling of the elements, as well as project scaling	
Physics simulation accurateness	Uses an integration strategy of different physics engines for a	Concentrates on collisions and their visual effects

	realistic and relatively accurate simulation	
Resource redundancies	Both game engines allow resource redundancy	
Highly complex system visualization	GUI visual effects, 2D/3D models and environments	High quality of visual effects for 3D models and environments

5. Conclusion and future work

The use case has shown the game engines potential to represent, simulate, control and visualize production modules as an example of industrial robots. Thanks to the flexibility of communication protocols it is possible to merge the virtual model in a Game Engine with its real model. One of the most evident advantages has been the versatility of game engines to create visualization interfaces with various graphic resources, allowing the creation of intuitive environments for the control and monitoring of production modules. However, it is necessary to carry out future studies of physical simulation accurateness and scenario-based control. Therefore, part of the future works is the integration of physics engines.

The control of cyber-physical production systems through game engines is an area of research with high development potential that is in its initial phase. Game engines are investing resources to make their development tools more efficient and accurate. The development and integration of movement algorithms for optimal path-finding and kinematics is possible and needs to be studied.

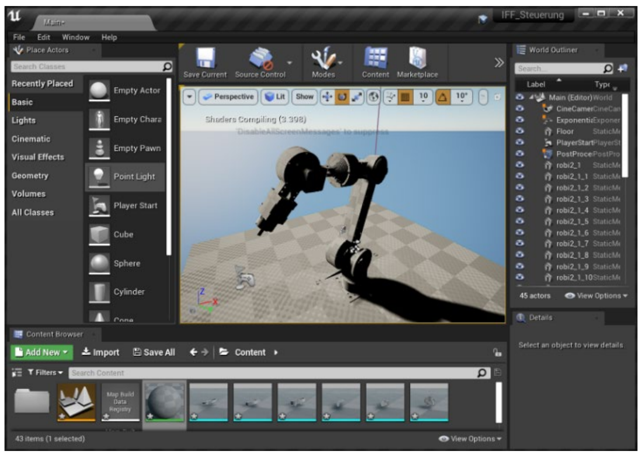

Fig. 3. Robot modelling in Unreal

Finally, the two-way communication between virtual models developed in game engines and reality should ensure optimal latency. Feedback control of this latency and therefore, of the system's bidirectional response times is another emerging area that must be deeply investigated.

References

[1] Schlegel, T., Siegert, J., Bauernhansl, T., 2019. Metrological Production Control for Ultra-flexible Factories *81*, p. 1313.

[2] Siegert, J., Schlegel, T., Zarco, L., Miljanovic, B. *et al.*, 2020. Ultra-flexible Factories: An Approach to Manage Complexity *93*, p. 329.

[3] Zarco, L., Siegert, J., Bauernhansl, T., 2019. Software Model Requirements Applied to a Cyber-Physical Modular Robot in a Production Environment *81*, p. 352.

[4] Gregory, J., 2019. *Game engine architecture*. CRC Press, Taylor & Francis Group, Boca Raton, London, New York.

[5] Xie, J., 2011. The research on mobile game engine, in *Proceedings of 2011 International Conference on Image Analysis and Signal Processing: October 21-23, 2011, Wuhan, China*, IEEE Press, [Piscataway, N.J.], p. 635.

[6] Uskov, A., Sekar, B., 2014 - 2014. Serious games, gamification and game engines to support framework activities in engineering: Case studies, analysis, classifications and outcomes, in *IEEE International Conference on Electro/Information Technology*, IEEE, p. 618.

[7] Vasudevamurt, V.B., Uskov, A., 2015 - 2015. Serious game engines: Analysis and applications, in *2015 IEEE International Conference on Electro/Information Technology (EIT)*, IEEE, p. 440.

[8] Pavkov, S., Frankovic, I., Hoic-Bozic, N., 2017 - 2017. Comparison of game engines for serious games, in *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, IEEE, p. 728.

[9] Ali, Z., Usman, M., 2016 - 2016. A framework for game engine selection for gamification and serious games, in *2016 Future Technologies Conference (FTC)*, IEEE, p. 1199.

[10] Navarro, A., Vicente, J., Rios, O., 2012. Open Source 3D Game Engines for Serious Games Modeling, in *Modeling and Simulation in Engineering*, InTech.

[11] Lang, C., 2011. *Panda3D 1.7 game developer's cookbook: Over 80 recipes for developing 3D games with Panda3D, a full-scale 3D game engine*. Packt, Birmingham, UK.

[12] Ghannem, A., 2014. Characterization of serious games guided by the educational objectives, in *Proceedings, TEEM '14: Second International Conference on Technological Ecosystems for Enhancing Multiculturality Salamanca, Spain, October 1st-3rd, 2014*, ACM, New York, p. 227.

[13] ROS.Org. urdf - ROS Wiki: urdf. https://wiki.ros.org/urdf. Accessed 11 December 2020.

[14] Neher, P., Lechler, A., 2015 - 2015. Using game physics engines for hardware-in-the-loop material flow simulations: benefits, requirements and experiences, in *2015 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, IEEE, p. 1002.

[15] Hummel, J., Wolff, R., Stein, T., Gerndt, A. *et al.*, 2012. An Evaluation of Open Source Physics Engines for Use in Virtual Reality Assembly Simulations, in *Advances in Visual Computing*, Springer Berlin Heidelberg, Berlin, Heidelberg, p. 346.

[16] Epic Games. Physics: Chaos Physics. https://docs.unrealengine.com/en-US/InteractiveExperiences/Physics/index.html. Accessed 11 December 2020.

[17] Technologies, U. Unity - Manual: Physics. https://docs.unity3d.com/Manual/PhysicsSection.html. Accessed 11 December 2020.

[18] Glatt, M., Kull, D., Ravani, B., Aurich, J.C., 2019. Validation of a physics engine for the simulation of material flows in cyber-physical production systems *81*, p. 494.

[19] Siegert, J., Zarco, L., Rossmeissl, T., Schlegel, T., 2019 - 2019. Software Control for a Cyber-Physical System in a Manufacturing Environment based on a Game Engine, in *2019 Global IoT Summit (GIoTS)*, IEEE, p. 1.