ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

UDC 004.4 **doi: 10.32620/reks.2025.1.19**

Oleksii VYNOKUR[1], Iryna PEROVA[1], Polina ZHERNOVA[2]

[1] ***Kharkiv National University of Radio Electronics, Kharkiv, Ukraine***
[2] ***American University Kyiv, Kyiv, Ukraine***

ANALYTICAL REVIEW OF VISUALIZATION METHODS FOR LAUNCH AND LANDING OF SPACECRAFT WITH CONSIDERATION OF 64-BIT SYSTEM BOUNDARY VALUE ISSUES

*The **subject matter** of this article is the contemporary software solutions used for modeling and visualizing spacecraft missions, specifically during the stages of launch, flight, and landing. The **goal** of this article is to critically evaluate popular game engines like Unity and Unreal Engine 5, along with specialized flight simulation software such as OpenRocket and Orbiter, focusing on their application in space simulations. The **tasks** are as follows: to investigate and assess the capabilities of the Unity and Unreal Engine 5 game engines in the context of space missions; to identify the limitations of 64-bit floating-point precision for large-scale space simulations and propose the potential for transitioning to 128-bit systems; to evaluate specialized tools like OpenRocket and Orbiter regarding their use for simulating spacecraft behavior; to analyze the existing limitations in integrating real-time data and suggest directions for further research and development. **The obtained results of the article:** It was established that Unity and Unreal Engine 5, although primarily developed for the gaming industry, can be adapted for aerospace simulations. However, due to the limitations of 64-bit precision, they are prone to visualization artifacts and computational errors that compromise the accuracy of the simulations. The transition to 128-bit systems was identified as a promising approach for enhancing the precision and flexibility of space mission modeling. This shift would allow better handling of the extensive scales and detailed aspects of space simulations. Specialized tools like OpenRocket and Orbiter demonstrated high capabilities in modeling aerodynamic characteristics and space missions. Nevertheless, they also face limitations in handling large-scale phenomena or integrating real-time data. The need for further research and development of new algorithms and data structures to ensure high precision and support for large datasets was identified. Additionally, improving the integration of real-time data and user interfaces is necessary to make these tools more accessible. **Conclusions.** The development of 128-bit systems for space simulations is critically important for enhancing the accuracy and realism of the modeling. The Unity and Unreal Engine 5 game engines although having the potential for adaptation to aerospace simulations, require significant improvements in handling large scales and detailed aspects. The tools OpenRocket and Orbiter have significant potential in specialized areas but also need enhancement to expand their capabilities. Further research and development are necessary to create new solutions that will increase the accuracy and functionality of the software for simulating space missions, as well as to develop new hardware such as more powerful processors and increased memory.*

Keywords: *Software; Launch vehicles; LEO; Visualization of calculations; limit states of computer systems; aircraft landing; game engines; simulation of physical phenomena.*

Introduction

Space mission simulations play a crucial role in advancing the efficacy and safety of space missions. The main tools in this field include game engines such as Unity [1] and Unreal Engine 5 [2], and specialized flight simulation software such as OpenRocket [3] and Orbiter [5]. Each of these tools has its own advantages and limitations that should be considered when planning missions. Game engines, while not traditionally associated with the space industry, offer significant visualization and interactivity capabilities, but their use is limited due to the lack of accuracy in large-scale computations that are critical to spaceflight [7, 1].

Motivation. The current 64-bit architecture [8, 9] imposes limitations on the accuracy and scale of space simulations, leading to issues such as visualization artifacts and computational errors. There is a pressing need to explore and develop alternative approaches that can overcome these limitations. This research aims to critically analyze existing tools and propose enhancements to improve spacecraft launch and landing simulations. This paper discusses the impact of the limitations of the current 64-bit architecture [8, 9] on the accuracy and scale

of space simulations. Problems, such as visualization artifacts and computational errors, often arise as a result of the limitations of these systems. Potential solutions and alternative approaches are also discussed, including the development of 128-bit computing systems, which can provide significant improvements in data processing and space mission simulation.

State of the Art. Existing mathematical modeling and flight simulation software, such as MatLab and OpenRocket [3], suffer from disadvantages in terms of visualization and accuracy. They face challenges in adding more parameters that describe the physical phenomena experienced by the spacecraft during flight, such as calculations of support points, interaction of support materials with surfaces, and stability of the structure relative to its geometry. Despite their visualization strengths, game engines are hindered by the limitations of 64-bit architecture in handling complex simulations [7 - 9].

Objective. This study aims to evaluate and compare various tools and technologies used for spacecraft landing and launch simulations. The focus is on identifying the most effective methods for enhancing these simulations, particularly through the use of game engines and the development of advanced computational systems like 128-bit architecture.

Approach and Structure. The paper begins with a detailed examination of each tool and technique, analyzing their strengths and weaknesses. It then discusses the potential of 128-bit computing systems to address the current limitations. The structure of the paper is as follows:

Case Studies. Evaluation of recent advancements in landing systems and simulation methods, including innovative approaches in mechanical adaptation, real-time force sensing, and deep learning for powered landing guidance.

Review of Simulation Tools. Analysis of the game engines and specialized simulation software, highlighting their capabilities and limitations.

Impact of the 64-bit architecture. Discussion on how the current architecture affects the simulation accuracy and scale, with examples of common issues.

Potential Solutions. Exploration of advanced computing systems and their benefits for space mission simulations.

This introduction sets the stage for a more detailed look at each of the tools and techniques that will be analyzed in the following sections of the paper, intending to identify the most effective ways to improve spacecraft launch and landing simulations.

Mathematical modeling or flight simulation software such as MatLab and OpenRocket [3] have several disadvantages in terms of the visualization and accuracy of the representation of the flight and landing process. One of the main ones is the difficulty in adding more parameters describing the physical phenomena experienced by the aircraft during the flight. These include: calculations of the support points of the landing aircraft, the interaction of the support materials and the surfaces with which they interact, and calculations of the stability of the structure relative to its geometry, etc. Accordingly, game engines are a good option for speeding up and refining the results of aircraft landing simulations. However, both simulation programs and game engines have the problem of calculations on a large scale. Existing computers use a 64-bit architecture [8, 9], which provides 3 types of data for processing: float - 4 bytes, double - 8 bytes, decimal - 16 bytes [8, 9]. Game engines, in turn, do not use the decimal type out of necessity, as conventional games do not perform complex simulations on a planet scale, which leads to difficulties in calculating the trajectories of even suborbital flights [7].

As highlighted in the introduction, the advancements in spacecraft landing systems and simulation technologies are crucial for enhancing the efficacy and safety of space missions. Significant research efforts have been directed toward developing innovative landing systems and simulation methods, addressing the unique challenges presented by space exploration. In the review [24], the authors proposed a novel landing leg with electromagnetic damping and anchoring capabilities for small celestial bodies where microgravity and unpredictable surface conditions prevail. This study integrates a comprehensive approach combining mechanical adaptation and real-time force sensing to stabilize the landing process and prevent post-landing drift, reflecting a substantial evolution from traditional passive buffering methods like honeycomb aluminum used in lunar and Martian missions.

Survey [25] discusses a new spacecraft attitude recovery method that leverages platform vibration to mitigate errors and losses in attitude data critical for spacecraft maneuvering. This paper provides an extensive review of the challenges in attitude data acquisition and presents a validated recovery strategy using the ZY302 satellite, highlighting the method's applicability and high correlation with true attitude data.

In the context of simulation technologies, the paper [26] introduced a real-time simulation framework for rocket control using the visual programming environments LabVIEW and X-Plane. The authors focus on the development of a robust simulation interface that facilitates comprehensive testing and iterative enhancements of rocket control algorithms, a critical aspect in reducing the risk associated with live launches.

Furthermore, [27] explored agile mission planning for emergency space launches. This paper systematically constructs a flexible and responsive mission planning framework that incorporates multitask and multiplatform

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

considerations. This method is pivotal for rapid deployment missions, enabling efficient resource scheduling and on-the-fly adjustments to launch plans.

The use of deep learning for improving the powered landing guidance on reusable rockets is examined in [28]. The authors developed a network-based framework that categorizes initial flight conditions to optimize thrust profiles and landing trajectories, significantly advancing the adaptability and accuracy of autonomous guidance systems.

The paper structure. In Section 1, "Materials and Methods of Research," we explore the fundamental concepts and methods required for spacecraft trajectory calculations. This section begins with a detailed discussion of the ballistic trajectories, examining the factors affecting flight and landing in both two-dimensional and three-dimensional spaces. We then cover methods for visualizing these trajectories using simulation tools. Furthermore, the section highlights the use of Tsiolkovsky's formula for velocity change, the role of the Earth's curvature in trajectory planning, and the challenges of designing landing supports. The integration of game engines for modeling spacecraft landing processes and the issues related to boundary values in 64-bit systems are also discussed.

In Section 2, "Results and Discussion," we review software for visualizing spacecraft trajectories, focusing on OpenRocket, Kerbal Space Program, Unity, Unreal Engine 5, and Orbiter. Each tool's capabilities are assessed for accuracy, visualization quality, and handling complex physics such as air resistance and object geometry. In addition, limitations such as artifact generation in Unity and outdated graphic engines in Orbiter are discussed.

The paper concludes with the Conclusions section, which outlines the key findings from the review of available tools for spacecraft flight and landing simulations. It emphasizes the limitations of the current solutions for non-standard scenarios and proposes potential improvements. Two main development paths are suggested: optimizing game engine algorithms for more efficient use of virtual space and upgrading older software like Orbiter to modern graphics standards. The section also points to future research directions, including integrating machine learning to enhance the analysis and simulation of landing sequences.

1. Materials and methods of research

Let's take a look at the simplest 2D visualization of a spacecraft flight, as well as what is needed for a more accurate calculation of both flight and landing in 3-dimensional space, taking into account more parameters.

1.1. Ballistic Trajectory

The ballistic trajectory of a spacecraft is a fundamental concept in space dynamics that determines the path of an object in space under the influence of external forces alone, without any additional propulsive maneuvering after the initial launch phase. In the context of the Earth, this usually means gravity and, in the initial stages, the aerodynamic drag of the atmosphere.

At the simplest level, the ballistic trajectory can be thought of as a parabolic motion when atmospheric drag is ignored, or as a more complex flight under the influence of aerodynamic drag. It is the result of the initial speed and launch angle, which together determine the range, altitude and duration of the flight.

Equation for calculating the flight range (excluding air resistance):

$$D = \left(v_0^2 \sin(2\theta)\right) \div g, \quad (1)$$

where D is the flight range;

v_0 is the initial velocity;

θ is the launch angle relative to the horizon;

g is the free fall acceleration (9.81m/s^2 on the Earth's surface).

Maximum height reached by the object:

$$H = \left(v_0^2 \sin^2(\theta)\right) \div 2g, \quad (2)$$

where H is the maximum height.

The simulation software allows you to visually model the trajectory, considering various external factors, and assess potential risks and options during the critical phases of the mission.

1.2. Methods for visualizing 2D trajectories

The use of two-dimensional graphs to show key driving parameters, such as distance, speed, and angle of ascent, allows you to analyze the effectiveness of the planned route and make the necessary adjustments.

The simulation software allows you to visually model the trajectory, taking into account various external factors, and assess potential risks and options during critical phases of the mission (Fig. 1).

1.3. Cialkowski's formula

In modern launch vehicles, the largest mass fraction is occupied by fuel, not payloads or control systems. The mass fraction of fuel ranges from 90-97% [29] of the total weight of the aircraft. Accordingly, we need to use the Tsiolkovsky formula in our calculations. It expresses the

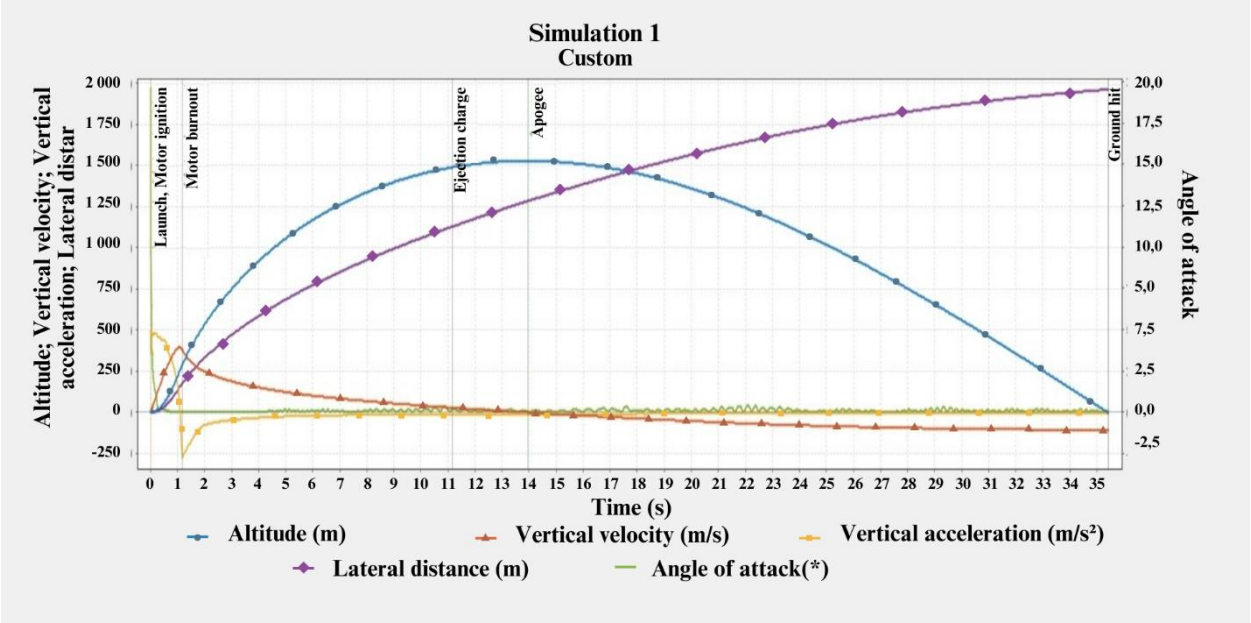

Fig. 1. An example of using Open Rocket to calculate a ballistic trajectory

relationship between the mass of fuel burned and the rocket's ability to change its speed. The Tsiolkovsky formula is a key equation in astrodynamics that allows us to calculate the required change in velocity (Δv Δv) that is needed for a spacecraft to reach a certain orbit or perform a manoeuvre in space [33].

$$\Delta v = I_{sp} \times g_0 \times \ln\left(m_0 \div m_f\right), \qquad (3)$$

where: Δv – required speed change (m/s);

I_{sp} – specific impulse response of the engine (sec);

g_0 – standard acceleration of free fall on the Earth's surface (9.81m/s^2);

m_0 – initial mass of the rocket (kg);

m_f – final mass of the rocket after fuel combustion (kg).

There are two ways to visualize this formula: a graph of Δv versus fuel weight and a simulation of the specific impulse response.

1.4. Curvature of the Earth's surface

The curvature of the Earth is a crucial factor in the planning and analysis of spacecraft flight paths. This aspect is of particular importance when calculating the orbital parameters, in-flight course correction, and when designing missions with a return to Earth. Considering the surface curvature allows for more accurate prediction and adaptation of trajectories to achieve the required mission objectives.

The curvature of the Earth's surface affects the calculation of orbits and trajectories because it requires spacecraft control systems to make corrections to compensate for this curvature. For example, without taking the curvature into account, planning a return to the launch point on Earth would not be possible because the spacecraft would simply fly past the target point.

The visualization techniques for this parameter include:

– Global Earth models, which use three-dimensional Earth models to simulate trajectories, allow us to visually consider the curvature of the Earth's surface and analyze potential spacecraft trajectories on a global scale;

– Globe mapping for projecting a flight path on a physical or virtual globe helps to determine the actual distances and orientation of the trajectory relative to the Earth's surface, contributing to a better understanding of mission parameters;

– Orbit analysis software such as Ansys STK (Systems Tool Kit) [34], which is proprietary and expensive, or Orbiter [5] - which has some quite old code bases and was released to the public in April 2021 [5] - allows the integration of physical and astro-dynamic parameters of the Earth, including its curvature, to accurately model flight paths;

– Demonstration of ways to correct trajectories to account for the Earth's curvature, such as using gravity maneuvers or speed changes to achieve the required orbital parameters.

1.5. Problem of calculation of aircraft geometry supports at landing and physics of their interactions with the surface

The design of a spacecraft landing system is a complex task that requires consideration of numerous factors to ensure the safety and stability of the vehicle during landing.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Main challenges:

– Load distribution - the pores should be designed to effectively distribute the weight of the vehicle on the surface, minimizing the risk of sinking or tipping over during landing.

– Surface adaptation - pores must adapt to the variety of surfaces they may encounter, which requires flexibility in design and material use.

– Shock absorption - landing can be accompanied by strong impacts, so the support system must include shock absorption elements to protect the vehicle and its equipment.

Modern engineering software tools allow for the creation of detailed three-dimensional models of the support system, which simplifies the analysis of potential problems and design improvements [34].

Simulations can demonstrate how the supports interact with different types of surfaces under different landing conditions, allowing for load distribution analysis and identification of potential failure points. The use of dynamic simulations helps to assess how the support system absorbs shocks and ensures the stability of the vehicle when it comes into contact with the surface. In turn, the virtual environment allows engineers to test different configurations and landing conditions, finding optimal solutions without risking the real vehicle.

1.6. Use of a game engine to calculate the supports of the aircraft geometry during landing and the physics of their interactions with the surface

Game engines such as Unreal Engine [2] and Unity [1] has opened up new possibilities for spacecraft engineering and design, including simulations of landing processes. Thanks to their powerful visualization and physics capabilities, these engines allow for detailed, realistic simulations that can mimic the complex interactions between the aircraft and the surface during landing.

Game engines integrate advanced physics engines [35] to accurately simulate the gravitational forces, friction, damping, and other physical properties necessary for landing analysis.

Modern game engines are capable of producing high quality visual effects, including realistic lighting and shadows, which allows you to visually analyze the behavior of the vehicle under different lighting conditions during landing.

Advantages of using game engines:

– game engines allow users to interact with the model in real time, changing parameters and conditions to analyze different landing scenarios. This contributes to a better understanding of the potential problems and the effectiveness of the solutions developed;

– the use of game engines provides access to various modeling, visualization, and analysis tools that allow for detailed calculation of every aspect of the landing process, from motion dynamics to surface impact;

– the use of game engines for preliminary modeling can significantly reduce costs [2, 7] as it allows potential problems to be identified and addressed early in the design process, avoiding costly mistakes and reducing the need for physical tests;

– game engines allow simulating a variety of landing scenarios on any surface, from Earth to Mars or even asteroids, with the ability to easily scale the design to meet specific mission needs.

The use of game engines for modeling and visualizing landing processes opens up new perspectives for designing and analyzing space missions, allowing developers to create more efficient, safe and reliable lander systems.

1.7. The problem of boundary values of 64-bit systems

The limitations of precision and number size in 64-bit computing systems pose challenges not only for traditional scientific and engineering applications, but also for game engines such as Unity and Unreal Engine, especially when working with space simulations and large-scale virtual environments.

Unity uses the float type for physics calculations and rendering, which provides sufficient accuracy for most game projects and virtual reality applications [1]. However, the float type has limited accuracy, especially when working with very large or small values, which is typical for cosmic distances or simulating microscopic processes. This can lead to rounding errors and other inaccuracies in the visualization and physical calculations.

Unreal Engine 5, despite its advanced technology foundation, also faces challenges in handling large and small values, even when using double-precision numbers [7]. Although double precision increases the range and accuracy of the numbers that can be represented, game engines often encounter problems when rendering large-scale scenes or accurately modeling physical processes on a cosmic scale. This can include errors in calculations, inaccuracies in object mapping, and other artifacts that make it difficult to create virtual environments that require high accuracy and realism (Fig. 2).

Developing and using adaptive coordinate systems that can change scale depending on the context of the scene, allowing for high accuracy calculations in different parts of the virtual world [2]. Review and optimize algorithms to minimize the impact of accuracy constraints, including the use of high fidelity algorithms

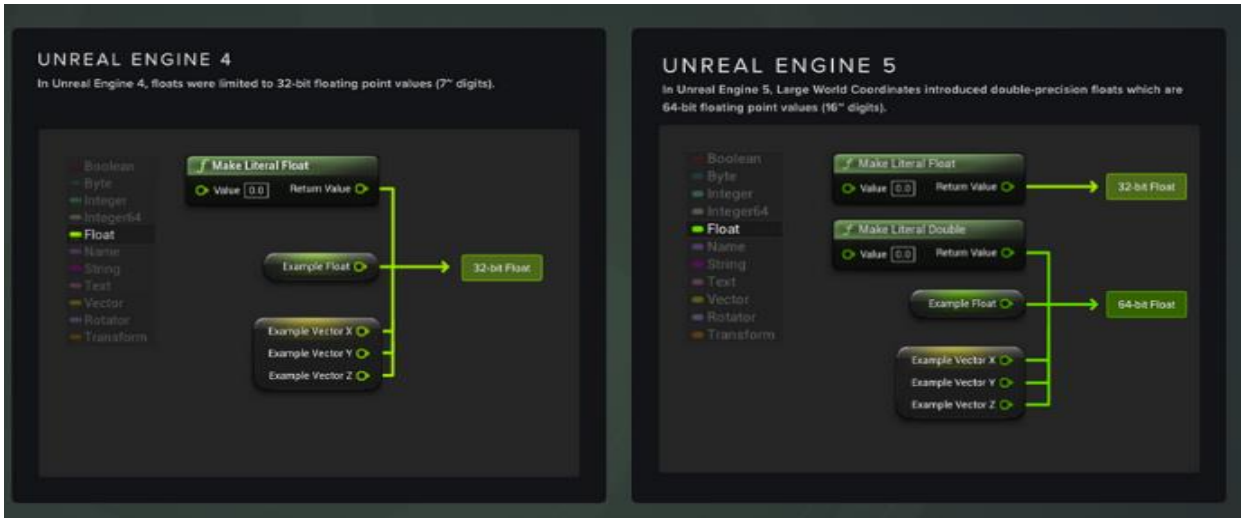

Fig. 2. Changes in the new version of the Unreal Engine 5 engine added the ability to use the double precision method [7]

where appropriate. In the context of overcoming the limitations of number precision and magnitude, 128-bit computing systems represent a potential future development offering an even greater range of number representation and increased computing accuracy (Table 1). Although 128-bit systems are currently in the early stages of research and development, their potential implementation could radically change the way data is processed in research, engineering, and the gaming industry. This transition will require significant efforts in the development of software and hardware solutions but promises to significantly improve the ability to model complex systems and processes that require high computational accuracy on a large scale or when working with very small values.

Table 1

Bit capacity of the system and the number of its possible states

Bits	States
1	2
2	4
4	16
8	256
166	65,536
32	4,294,967,296
64	18,446,744,073,709,551,616
128	340,282,366,920,938,463,463,374,607, 431,768,211,456

2. Results and Discussion

As mentioned above, the best way to visualize the calculations of the aircraft launch is to use software that graphically displays the trajectory, inclination and, for example, air friction. Let's move on to reviewing the available solutions.

2.1. OpenRocket

One of the simplest programs for visualizing rocket flight is OpenRocket. It is a free program for simple calculations of small amateur solid rockets. It has a large list of settings for both the appearance of the rocket and the fuel components it will have (Fig. 3).

The software has both a schematic and 3D visualization of the finished rocket and takes into account its geometry, air friction, temperature and pressure differences at different stages of flight (Fig. 4).

To visualize the calculations, this program uses simple continuous charts that show both simple indicators such as flight altitude, and more complex ones such as rotation about all axes according to the geometry and solid propellant impulse (Fig. 5).

2.2. Kerbal Space Program

Kerbal Space Program is a space mission simulator with the ability to design and control an aircraft. Unlike OpenRocket, the results of the design can be viewed immediately after completion in 3D, as well as participate in the process of controlling the aircraft by launching it into orbit and editing the mission according to the current situation (Fig. 6). The program has an interactive interface with a lot of information about the mission status and the most realistic visualization.

It also has two types of mission status view: local, when we see the aircraft in our immediate vicinity, and global, where we see only the flight path and orbit (Fig. 7). Unfortunately, the program has a small list of aircraft modifications and provides the possibility of landing only with parachutes, which is problematic for medium and large launch vehicles.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

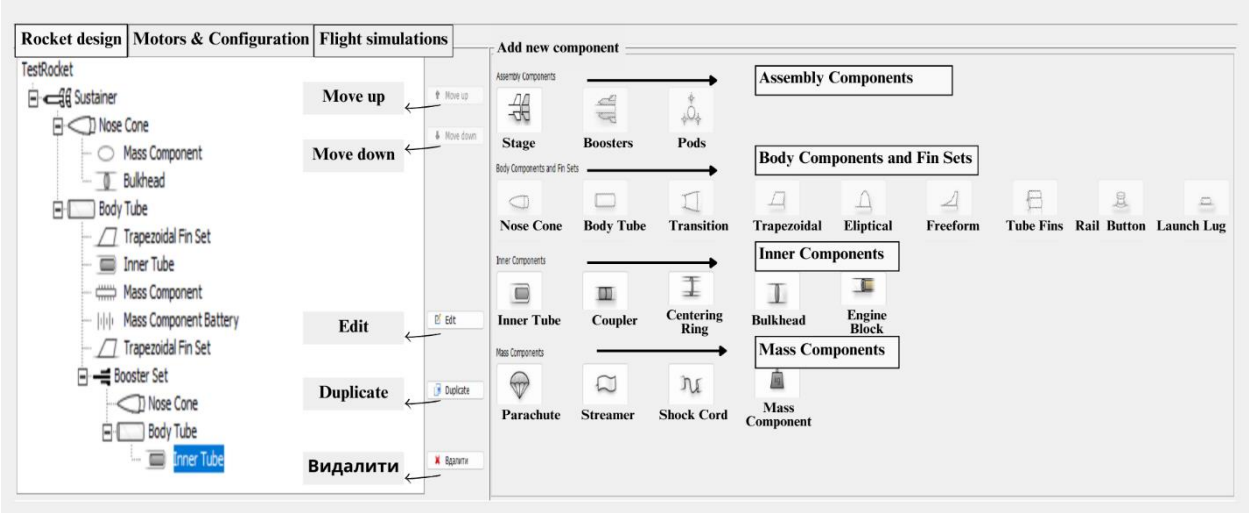

Fig. 3. A set of design tools in the OpenRocket software

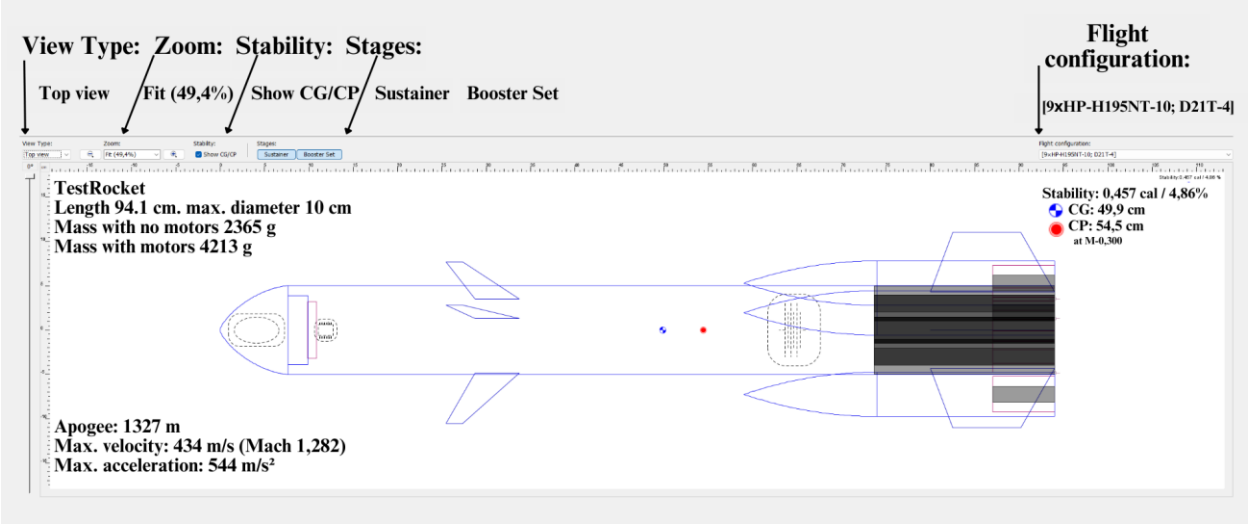

Fig. 4. 2D visualization of a rocket in OpenRocket software

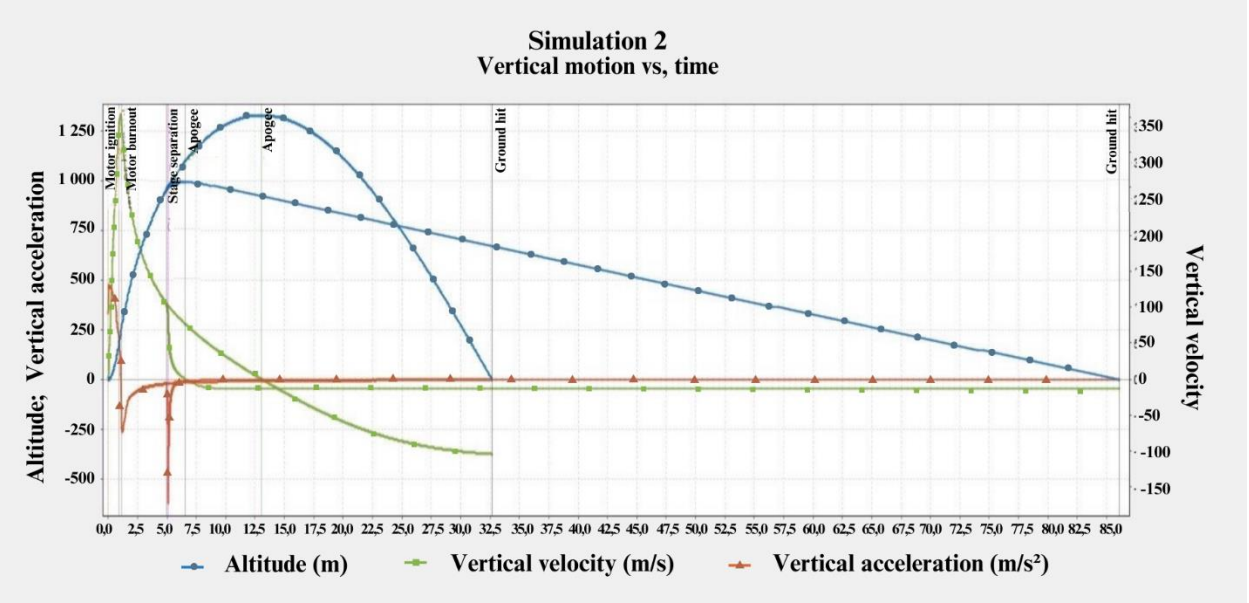

Fig. 5. Graph of the main parameters and flight stages in OpenRocket

2.3. Unity

Unity is a game engine with full integration of object and light physics [1]. The software allows you to create custom scenarios, write your own personal flight and landing control systems, calculate the physical interaction of materials, and simulate the physics of the launch vehicle supports, taking into account the center of mass of the structure.

In this example, we have a sphere that emulates the Earth in an appropriate 1:1 size. The launch considered the geometry of the planet, its gravity, air resistance according to the flight altitude, structural streamlining, and the center of mass of all booster modules (Fig. 8).

Fig. 6. Launch of a carrier rocket in the Kerbal Space Program

Fig. 7. Kerbal Space Program interface when switching to a large-scale view

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

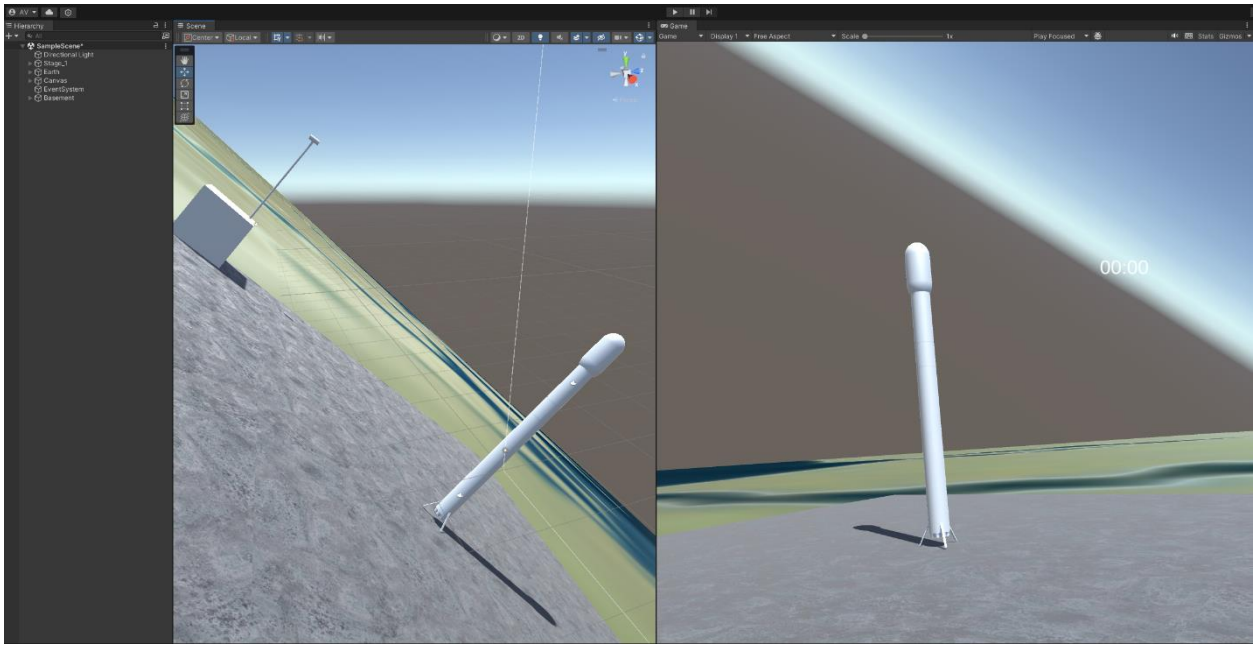

Fig. 8. A designed two-stage rocket on the Unity desktop

Unfortunately, when trying to simulate the real size of the globe, we get undesirable artifacts when moving away from the zero coordinate point. Fig. 9 shows an example of the so-called glitch effect, when at each frame the accuracy of calculating the coordinates of the vertices of objects has a large error, which increases with distance from the center of the coordinates (Fig. 9). This problem is observed in all 64-bit systems that do not have ways to optimize and avoid the problem of boundary values.

Since the visualization was unsuccessful, it was decided to conduct a similar experiment with a scale of 1:100. An example can be seen in Fig. 10 This experiment was successful and the visualization had no problems. This solution is a good way to simulate both the flight and landing of an aircraft and limits the number of parameters to the hardware capacity and conditions of the experiment.

Among the disadvantages is the complexity of calculations on a real-world scale; the engine uses the C# float type to calculate the orientation of objects, which has accuracy limits ranging from ±1.5 x 10−45 till ±3.4 x 1038 [8].

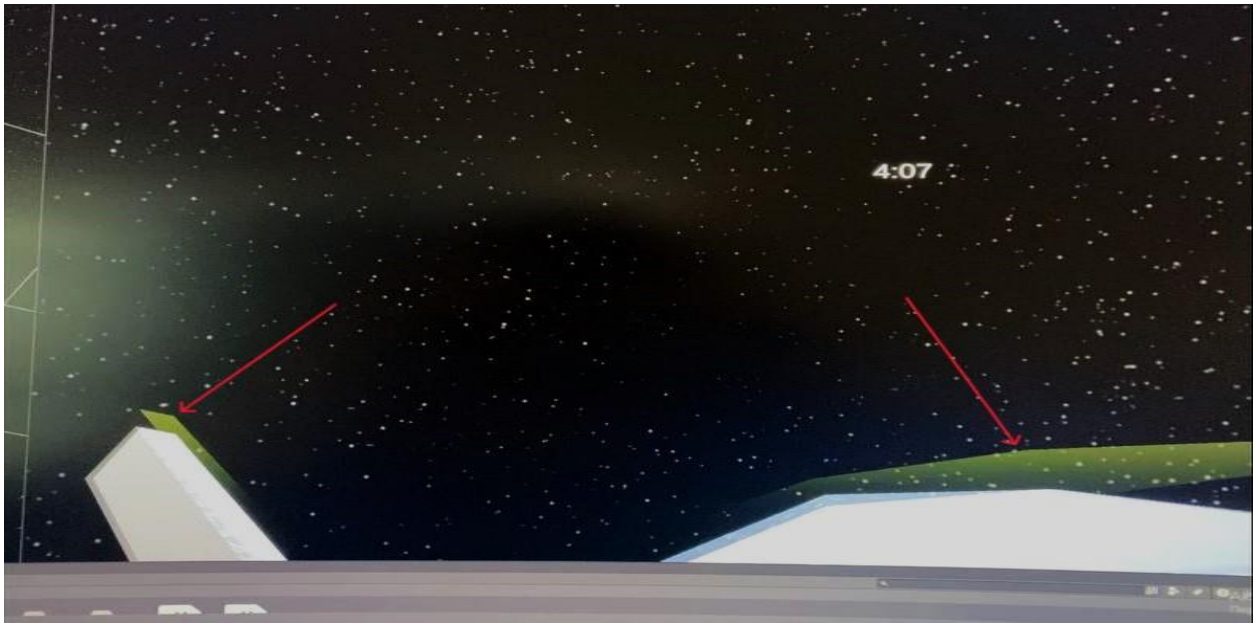

Fig. 9. The glitch effect of the vertices of game objects
at a distance from the initial coordinates

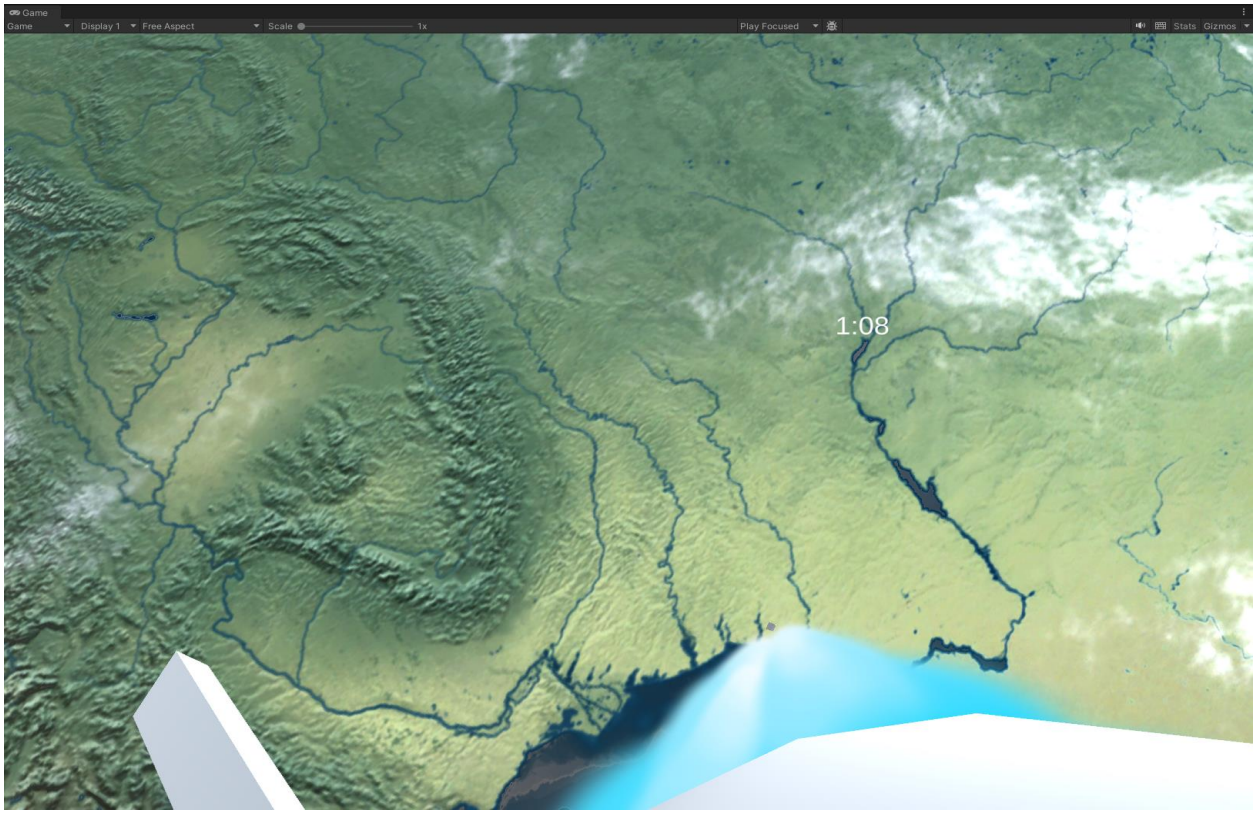

Fig. 10. Scenario with a scale of 1:100

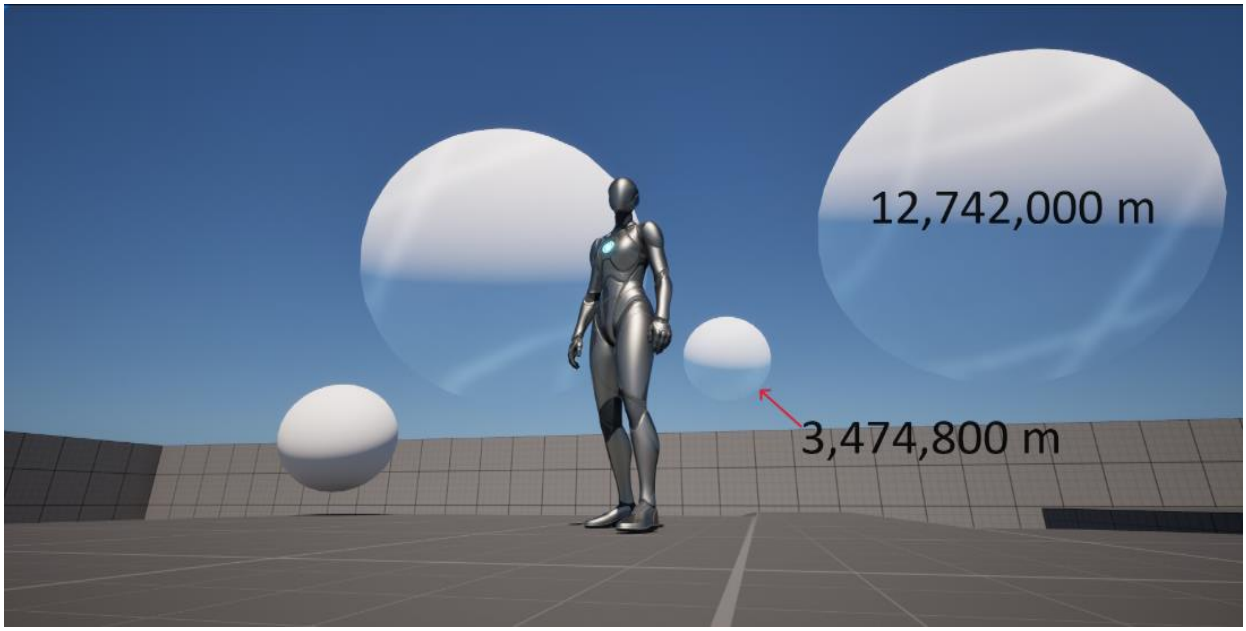

Fig. 11. An example of rendering objects on a real scale in Unreal Engine

2.4. Unreal Engine 5

Unreal Engine 5 is a more advanced game engine that is more professionally oriented. The engine allows you to easily manipulate a large virtual volume without any discomfort. Fig. 11 shows an example of objects that have realistic sizes of the Moon (3474800 m in diameter) and the Earth (12742000 m in diameter) by the standards of the game space. At the same time, this does not prevent the emulation of the physical interaction of objects or their animation without the glitch effects observed in the Unity game engine. This improvement is the result of the new version of the engine using the double precision method, namely its implementation in the double type

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

(Fig. 11). Despite this, some visual and error issues still occur, such as light and shadows being rendered with noise, errors when editing large spaces, and the application sometimes crashing and closing due to overloads and errors in object calculations (Fig. 12)

2.5. A few words about Orbiter

The software is quite advanced for a free open source solution. The software allows you to simulate take-off, orbit correction and landing scenarios, but unfortunately does not allow you to describe the vertical landing of the first stage of the launch vehicle. Unfortunately, the software uses an old graphics engine - DirectX7 [9] (1999) and supports only 32-bit architecture, which imposes its limitations [5] (Fig. 13).

2.6. JSBSim and FlightGear

JSBSim is a powerful tool for aerospace flight dynamics simulation, offering high-precision calculations. Unlike game engines, it focuses on aerodynamic accuracy without graphical limitations.

FlightGear, an open-source flight simulator, integrates with JSBSim to visualize real-time flight data.

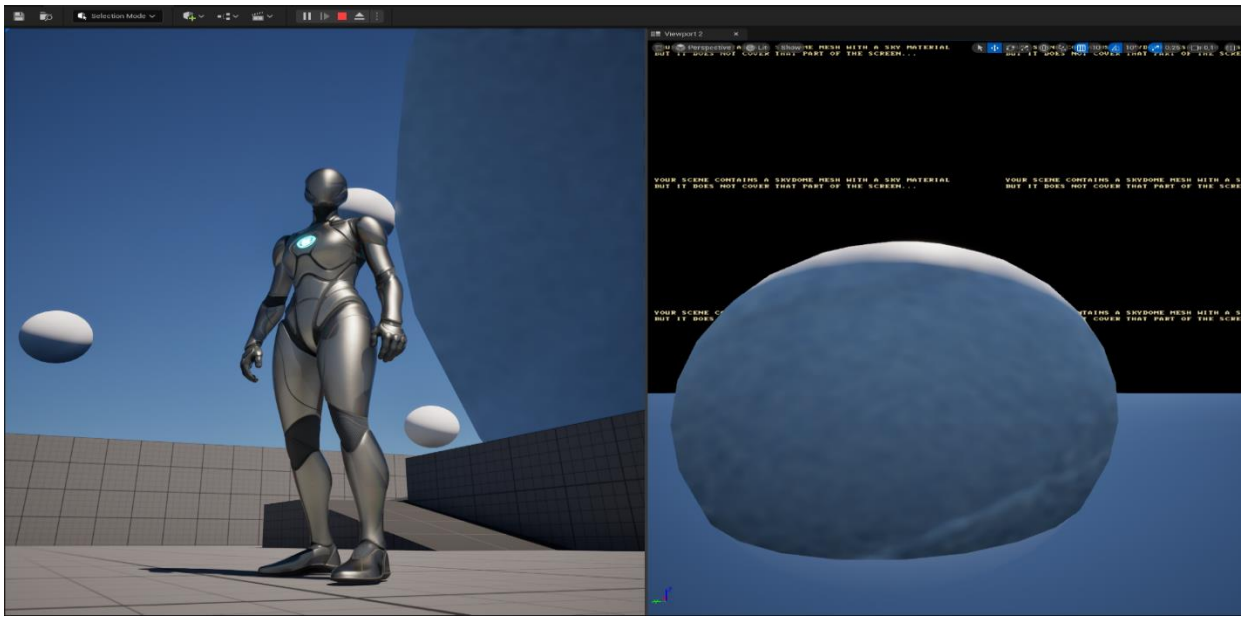

Fig. 12. An example of errors in rendering objects on a real scale

Fig. 13. Orbiter simulator interface during the LEO reentry scenario

This combination enables realistic space mission modeling, trajectory analysis, and the optimization of flight control systems. Numerical integration in JSBSim minimizes errors in long-duration simulations, which is essential for space missions. Numerical integration in JSBSim minimizes errors in long-duration simulations, which is essential for space missions. Integrating JSBSim with real telemetry data also requires additional setup, making the simulations complex.

Despite these challenges, JSBSim and FlightGear provide an effective solution for spacecraft flight simulation, requiring further improvements for enhanced accuracy and flexibility. To better understand the strengths and weaknesses of the different simulation tools, the following table (Table 2) provides a comparative analysis of the key characteristics, including physical accuracy, visualization quality, reinforcement learning (RL) integration, performance, and availability. This overview helps highlight the most suitable tools for various space and aeronautics applications.

In addition to the qualitative comparison, a numerical evaluation system was used to rank the simulators based on key attributes. Each simulator is rated on a scale from 0 to 3 in five categories: physical accuracy, visualization, RL integration, performance, and availability. The total score, as shown in Table 3, provides an overall assessment of each simulator's capabilities.

Table 2

Comparison of Flight Simulators

Simulator	Physical Accuracy	Visualization	RL Integration	Performance	Availability
JSBSim + FlightGear	High – JSBSim provides a highly accurate flight dynamics model used in FlightGear [32].	Medium – Offers 3D visualization (supports panoramic view ~190° across three screens), but graphics lag behind modern engines [4].	High – Ready-to-use interfaces (OpenAI Gym) enable RL-agent training [11].	High – Can run in real-time or faster; JSBSim can operate without graphics for speedup [12].	Open-source – Fully free solution (distributed under GPL) [14].
Orbiter	High – Uses realistic Newtonian physics for orbital flights, including precise orbital mechanics [20].	Medium – 3D planetary and spacecraft rendering (DirectX engine), but outdated graphics [14].	Low – No direct RL support; however, it has an API for creating modules and external flight control [15].	High – Efficient engine working on moderate systems; time acceleration available for long-duration maneuvers.	Freeware (closed) – Proprietary software for Windows (free to use but with closed-source code) [16].
Kerbal Space Program	Medium – Features realistic orbital mechanics, but other physical aspects are simplified [18].	Medium – Full 3D graphics with a cartoonish style (green Kerbal characters, etc.); not photorealistic [17].	Medium – No direct RL support, but modding/API (e.g., kRPC) allows RL agent training [23].	Low – Performance limited by Unity engine; physics simulation speed-up >4× is not possible, and complex scenes slow down the game.	Commercial – Paid game with closed-source code (available via digital stores, has an active modding community).
OpenRocket	High (for rockets) – Accurately models rocket aerodynamics and flight trajectories, validated up to ~Mach 1.5. [30].	Low – lacks real-time 3D visualization (results are presented as graphs and numerical reports) [22].	Low – no interactive interaction; lacks RL tools (calculations are performed offline).	High – resource-efficient; calculations are executed very quickly, allowing thousands of simulations to be run	Open-source – free open-source software (source code available to the community) [13].

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Simulator	Physical Accuracy	Visualization	RL Integration	Performance	Availability
Unreal Engine 5	Medium – built-in Chaos physics engine is simplified for gaming tasks [10]; high-precision physics modules (e.g., AGX Dynamics plugin) are available if needed [18].	High – supports photorealistic graphics with modern lighting and detailing effects (high-quality scene rendering possible) [31].	Medium – no built-in RL integrations, but plugins and frameworks (e.g., Microsoft's AirSim) enable RL applications in UE5 [6].	Low – very resource-intensive; parallel or accelerated execution of multiple simulations is difficult (game engines are not optimized for massive physics calculations) [22].	Conditionally free – the engine can be freely downloaded with access to the source code, but it is proprietary (not open-source; royalty-based license for commercial use) [20].
Unity	Medium – built-in PhysX engine is not designed for high-precision physics but allows integration of custom modules	High – supports realistic 3D graphics with shaders and visualization, but requires optimization for complex scenes	Medium – lacks built-in RL support, but extensions (ML-Agents) enable agents in interactive environments	Medium – performance depends on scene complexity; multithreading is limited, but GPU computing (Compute Shaders) is available	Conditionally free – a free version is available for personal use, but commercial use requires a paid subscription

Table 3

Quantitative Evaluation of Simulators

Simulator	Physical Accuracy	Visualization	RL Integration	Performance	Availability	Total Score
JSBSim + FlightGear	3	2	3	3	1	12
Orbiter	3	2	1	3	0	9
Kerbal Space Program	2	2	2	1	0	7
OpenRocket	3	1	1	3	1	9
Unreal Engine 5	2	3	2	1	0	8
Unity	2	3	2	2	0	9

Conclusions

A review of the available tools for modeling the full sequence of flight and landing of a spacecraft, considering for the curvature of the earth's surface, air friction, changes in the density of the atmosphere, fuel consumption and the geometry of the aircraft and its supports, was carried out. Their main advantages and disadvantages are also analyzed, and flight and landing simulation experiments are carried out, as well as experiments with different dimensions of virtual space. As demonstrated in the Quantitative Evaluation of Simulators, JSBSim + FlightGear provides the most well-rounded solution due to its high accuracy and reinforcement learning integration, making it the best option for scientific applications. Other tools, such as Orbiter and OpenRocket, also show strong results but have limitations in RL integration and visualization. Game engines like Unity and Unreal Engine 5 stand out in terms of visualization but require further optimization to be viable for aerospace research. These findings highlight the need for hybrid approaches that combine high-accuracy physics models with advanced visualization capabilities.

As a result of the analysis of existing specialized software solutions, it can be concluded that there are no ready-made solutions for the full visual simulation of non-standard solutions. OpenRocket is not suitable for

small or medium-sized launch vehicles, has a very simple editor, and visualization is limited to 2D graphs. Other programs have prepared parameters and scenarios and do not have a hot-landing option.

Game engines, on the other hand, are highly customizable, allowing you to describe any number of flight and landing parameters, including hot landing, taking into account the geometry of the supports and the physics of the interacting materials. However, unfortunately, they have their limitations when visualizing large virtual spaces due to the limitations of 64-bit architecture. 128-bit systems are currently only theoretical and do not have real working prototypes, so it is not economically feasible to continue research until a working system is available.

According to the available research, two development options are proposed.

The first is to improve the algorithms for customized scenarios in game engines to reduce the virtual space used by the relative coordinate method used in the Kerbal Space Program simulator. The Unreal Engine 5 engine is particularly interesting as it uses double-precision numbers. An alternative solution would be to upgrade the existing free Orbiter solution to a new graphic engine such as DX11 and integrate a hot landing scenario.

In future research, it is planned to use the selected tools and their integration to model the behaviour of launch vehicle components and develop a system for recognizing and analyzing video data using machine learning for aircraft landing.

Contributions of authors: conceptualization and methodology –**Oleksii Vynokur**; formulation of tasks – **Iryna Perova**; analysis, writing – original draft preparation – **Oleksii Vynokur**; review and editing –**Polina Zhernova**.

Conflict of interest

The authors declare that they have no conflict of interest concerning this research, whether financial, personal, authorship or otherwise, that could affect the research and its results presented in this paper.

Financing

The study was conducted without financial support.

Data availability

The manuscript has no associated data.

Use of artificial intelligence

The authors confirm that they did not use artificial intelligence technologies when creating the current work.

Acknowledgments

We wish to thank Kharkiv National University of Radio Electronics for providing the resources necessary for this study. Our appreciation extends to the editorial team of the journal RECS at the National Aerospace University "Kharkiv Aviation Institute" for considering and publishing our article.

All the authors have read and agreed to the published version of the manuscript.

References

1. Unity Engine. *Unity Technologies*, 2024. Available at: https://unity.com (accessed 01 March 2024).
2. Day, S., Smallwood, W. K., & Kuhn, J. Simulating Industrial Control Systems Using Node-RED and Unreal Engine 4. *National Cyber Summit (NCS) Research Track, Springer*, 2021, vol. 310. Available at: https://link.springer.com/chapter/10.1007/978-3-030-84614-5_2 (accessed 01 March 2024).
3. Durnyak, B. *OpenRocket Technical Documentation*, 2013.
4. Berndt, J. *JSBSim Development Team JSBSim: An Open Source Flight Dynamics Model in C++*, 2011.
5. Orbiter is Now Open Source. *Orbiter Forum*, 2021. Available at: https://www.orbiter-forum.com/threads/orbiter-is-now-open-source.40023/ (accessed 04 March 2024).
6. *Unreal Engine Forums. How to speed up the engine.* Unreal Engine Forums, 2018. Available at: https://forums.unrealengine.com/t/how-to-speed-up-the-engine-for-reinforcement-learning/426647 (accessed 19 December 2024).
7. *Large World Coordinates in Unreal Engine 5*. Epic Games, 2022. Available at: https://dev.epicgames.com/documentation/en-us/unreal-engine/large-world-coordinates-in-unreal-engine-5?application_version=5.0 (accessed 04 March 2024).
8. *Float and Double Type (C#* reference)*. Microsoft Docs, 2022. Available at: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/floating-point-numeric-types (accessed 07 March 2024).
9. *Direct3D*. Microsoft, 2021. Available at: https://learn.microsoft.com/en-us/windows/win32/direct3d (accessed 07 March 2024).
10. *Epic Games. Physics in Unreal Engine*. Epic Games Developer Hub. Available at: https://dev.epicgames.com/documentation/en-us/unreal-engine/physics-in-unreal-engine (accessed 14 December 2024).
11. Gor-Ren. *Pre-built OpenAI Gym interfaces for RL training with JSBSim/FlightGear*. GitHub, 2018 Available at: https://github.com/Gor-Ren/gym-jsbsim# (accessed 14 December 2024).
12. *JSBSim-Team. Frequently Asked Questions*. GitHub, 2022. Available at: https://github.com/JSBSim-Team/jsbsim/wiki/Frequently-Asked-Questions (accessed 14 December 2024).
13. *OpenRocket. Open-source software with publicly available source code*. GitHub, 2023. Available at:

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

https://github.com/openrocket/openrocket (accessed 19 December 2024).

14. *FlightGear open-source flight simulator*, FlightGear, GitLab, 2015. Available at: https://gitlab.com/flightgear/flightgear#:~:text (accessed 15 December 2024).

15. *Making a Space Flight Sim*, 2012. GameDev.net. Available at: https://www.gamedev.net/forums/topic/626934-making-a-space-flight-sim/#:~:text (accessed 15 December 2024).

16. *Orbiter 2005 A Real-Physics Spaceflight Simulation*. SimHQ.net, 2006. Available at: https://www.simhq.net/_air6/air_211a.html# (accessed 15 December 2024).

17. *Whiteastercom. Kerbal Space Program - complex environment for Reinforcement Learning*. Medium.com., 2018. Available at: https://medium.com/@whiteastercom/kerbal-space-program-complex-environment-for-reinforcement-learning-12318db065f5#:~:text= (accessed 17 December 2024).

18. *Algoryx brings high-fidelity physics simulation to Unreal Engine*. Unreal Engine, Epic Games, 2023. Available at: https://www.unrealengine.com/en-US/spotlights/algoryx-brings-high-fidelity-physics-simulation-to-unreal-engine (accessed 19 December 2024).

19. *How realistic is Kerbal Space Program*. Space Stack Exchange, 2014 Available at: https://space.stackexchange.com/questions/4505/ (accessed 14 December 2024).

20. *Question: Is Unreal Engine Open Source?* DragonflyDB. Available at: https://www.dragonflydb.io/faq/is-unreal-engine-open-source (accessed 19 December 2024).

21. *Orbiter*. OrbiterWiki, 2024 Available at: https://www.orbiterwiki.org/wiki/Orbiter#: (accessed 14 December 2024).

22. *What is OpenRocket*. OpenRocket Wiki, 2023. Available at: https://wiki.openrocket.info/Introduction#:~:text (accessed 18 December 2024).

23. *Kerbal Space Program Differential Game Challenge*. MIT Lincoln Laboratory, 2024. Available at: https://www.ll.mit.edu/conferences-events/2024/01/kerbal-space-program-differential-game-challenge#:~:text (accessed 17 December 2024).

24. Zhao, Z., Xiao, T., Tang, Z., Gao, X., Liu, X., Zhang, W., & Liu, B. Development of a Landing Leg with Active Buffering and Anchoring Functions Applied to the Small Body Landing Mechanism. *2020 International Conference on Mechatronics and Automation (ICMA)*, Beijing, China, IEEE, 2020, pp. 695-699. DOI: 10.1109/ICMA49215.2020/9.

25. Mo, F., Ye, F., Xie, J., Zhu, H., Liu, R., & Jin, J. A Novel Spacecraft Attitude Recovery Method Based on Platform Vibration. *2019 9th International Conference on Recent Advances in Space Technologies (RAST)*, Istanbul, Turkey, IEEE, 2019, pp. 117-122. DOI: 10.1109/RAST.2019.8767849.

26. Peng, C. C., Chan, C. Y., Lin, J. H., & Hsieh, T. Y. Spacecraft 6-DoF Localization in a GPS denied Environment. *International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, Penghu, Taiwan, IEEE, 2021, pp. 1-2. DOI: 10.1109/ICCE-TW52618.2021.9603214.

27. Cantri, F. M., Bisri, M. H., & Irwanto, H. Y. Realtime Simulation for Rocket Using Visual Programming. *2022 8th Information Technology International Seminar (ITIS)*, Surabaya, Indonesia, IEEE, 2022, pp. 150-155. DOI: 10.1109/ITIS57155.2022.10010182.

28. Wang, J., Ma, H., Li, H., & Hongbo, C. Real-time guidance for powered landing of reusable rockets via deep learning. *Neural Computing & Applications*, *Springer*, 2023, vol. 35. Available at: https://link.springer.com/article/10.1007/s00521-022-08024-4 (accessed 10 March 2024).

29. Abate, M., Anandapadmanaban, E., Bao, L., Challani, S., Gaughan, J., Jiang, A., Lingineni, A., Vora, A., Yang, C., & Zhao, D. *Correlation Between Simulated, Calculated, and Measured Model Rocket Flight*. 2014. Available at: http://ftp.demec.ufpr.br/foguete/bibliografia/Abate_et_al_2014.pdf (accessed 12 March 2024).

30. Niskanen, S. *Development of an Open Source model rocket simulation software*. MSc thesis. Helsinki University of Technology, 2009 Available at: https://openrocket.sourceforge.net/thesis.pdf#:~:text (accessed 18 December 2024).

31. Chander, S. *Rendering & Lighting a Photorealistic Abandoned Scene in Unreal Engine 5*. 80 Level, 2023. Available at: https://80.lv/articles/rendering-lighting-a-photorealistic-abandoned-scene-in-unreal-engine-5/ (accessed 19 December 2024).

32. Wood, A., Sydney, A., Chin, P., Thapa, B., & Ross, R. *GymFG: A Framework with a Gym Interface for FlightGear*. 2020. ArXiv. Available at: https://arxiv.org/pdf/2004.12481# (accessed 14 December 2024).

33. Eerland, W., Box, S., & Sobester, A. Cambridge rocketry simulator-a stochastic six-degrees-of-freedom rocket flight simulator. *Journal of Open Research Software*, 2017, vol. 5, no 1, pp. 1-6. Available at: https://openresearchsoftware.metajnl.com/articles/10.5334/jors.137 (accessed 12 March 2024).

34. *Ansys STK*. Ansys, 2024. Available at: https://www.ansys.com/products/missions/ansys-stk (accessed 03 March 2024).

35. Bykerk, T., & Karl, S. Preparatory CFD Studies for Subsonic Analyses of a Reusable First Stage Launcher during Landing within the RETPRO Project. *Aerospace Europe Conference 2023*, Goettingen, 2023, pp. 1-10. Available at: https://elib.dlr.de/194477/3/ELIB-Eintrag-2023-BykerkT-194477-Paper-Published.pdf (accessed 07 March 2024).

Received 17.06.2024, Accepted 17.02.2025

ОГЛЯД МЕТОДІВ ВІЗУАЛІЗАЦІЇ ДЛЯ РОЗРАХУНКІВ ЗАПУСКУ ТА ПОСАДКИ КОСМІЧНИХ АПАРАТІВ ТА ПРОБЛЕМА ГРАНИЧНИХ ЗНАЧЕНЬ 64-БІТНИХ СИСТЕМ

О. О. Винокур, І. Г. Перова, П. Є. Жернова

Предмет статті – сучасні програмні рішення, що використовуються для моделювання та візуалізації космічних місій, зокрема на етапах запуску, польоту та посадки. Метою статті є критична оцінка популярних ігрових рушіїв, таких як Unity та Unreal Engine 5, а також спеціалізованого програмного забезпечення для симуляції польотів, як-от OpenRocket і Orbiter, з фокусом на їх застосування у космічних симуляціях. Завдання дослідження включають: вивчення та оцінку можливостей ігрових рушіїв Unity та Unreal Engine 5 у контексті космічних місій; виявлення обмежень точності 64-бітових чисел з плаваючою комою для великомасштабних космічних симуляцій та пропозицію потенціалу переходу на 128-бітові системи; оцінку спеціалізованих інструментів, таких як OpenRocket та Orbiter, щодо їх використання для моделювання поведінки космічних апаратів; аналіз існуючих обмежень в інтеграції даних у реальному часі та пропозицію напрямків для подальших досліджень та розробок. Результати. Було встановлено, що Unity та Unreal Engine 5, хоча і розроблені в першу чергу для ігрової індустрії, можуть бути адаптовані для аерокосмічних симуляцій. Проте через обмеження 64-бітової точності вони схильні до візуальних артефактів та обчислювальних помилок, які компрометують точність симуляцій. Перехід на 128-бітові системи було визначено як перспективний підхід до підвищення точності та гнучкості моделювання космічних місій. Цей перехід дозволив би краще обробляти великі масштаби та детальні аспекти космічних симуляцій. Спеціалізовані інструменти, такі як OpenRocket та Orbiter, продемонстрували високі можливості у моделюванні аеродинамічних характеристик та космічних місій. Водночас вони також стикаються з обмеженнями при обробці великомасштабних явищ або інтеграції даних у реальному часі. Виявлено потребу у подальших дослідженнях та розробці нових алгоритмів і структур даних для забезпечення високої точності та підтримки великих наборів даних. Крім того, необхідно покращити інтеграцію даних у реальному часі та користувацькі інтерфейси, щоб зробити ці інструменти більш доступними. Висновки. Розробка 128-бітових систем для космічних симуляцій є критично важливою для підвищення точності та реалістичності моделювання. Ігрові рушії Unity та Unreal Engine 5, хоча мають потенціал для адаптації до аерокосмічних симуляцій, потребують значного покращення в обробці великих масштабів та детальних аспектів. Інструменти OpenRocket та Orbiter мають значний потенціал у спеціалізованих галузях, але також потребують вдосконалення для розширення своїх можливостей. Необхідні подальші дослідження та розробки для створення нових рішень, які підвищать точність і функціональність програмного забезпечення для моделювання космічних місій, а також розробки нового апаратного забезпечення, такого як більш потужні процесори та збільшена пам'ять.

Ключові слова: програмне забезпечення; ракети-носії; ННО; візуалізація розрахунків; граничні стани комп'ютерних систем; посадка літаків; ігрові рушії; моделювання фізичних явищ.

Винокур Олексій Олександрович – здобувач ступеня доктора філософії з комп’ютерних наук, Харківський національний університет радіоелектроніки, Харків, Україна.

Перова Ірина Геннадіївна – д-р техн. наук, проф., каф. системотехніки, Харківський національний університет радіоелектроніки, Харків, Україна.

Жернова Поліна Євгеніївна – канд. техн. наук, старший викладач, факультет цифрових технологій, American University Kyiv; провідний спеціаліст з розвитку талантів, керівник проєкту, Познань, Польща.

Oleksii Vynokur – PhD Student in Computer Science, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine,
e-mail: avinokur4@gmail.com, ORCID: 0009-0001-4328-3886.

Iryna Perova – Dr. Tech. Sc., Professor, Department of System Engineering, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine,
e-mail: rikywenok@gmail.com, ORCID: 0000-0003-2089-5609, Scopus Author ID: 57189383519,
Researcher ID: V-7479-2017.

Polina Zhernova – PhD, Senior Lecturer, Faculty of Digital Technologies, American University Kyiv; Lead Talent Development Specialist, Project Manager, Poznan, Poland,
e-mail: polina.zhernova@gmail.com, Scopus Author ID: 57202212660.