Inside | Jurnal Sistem Informatika Cerdas
Volume 2 Nomor 2 | bulan Desember tahun 2024 | halaman 25-31
e-ISSN: 3031 - 4860

# Evaluating Manual Kinematics and Unity's Built-in Physics for Real-Time Ballistic Simulations: A Comparative Study

**Reza Putra Pradana[1*], Afis Asryullah P[2], Akas Bagus S[3], Khen Dedes[4], Fatimatuzzahra[5]**
[1,2,3,4,5] Politeknik Negeri Jember, Jember, Indonesia

Corresponding Author:
Reza Putra Pradana, Politeknik Negeri Jember, Jember, 68101, Indonesia
Email: reza_pd@polije.ac.id

**Abstract**

Real-world ballistic testing is often constrained by high costs, time requirements, and safety risks. Modern game engines, such as Unity, offer a cost-effective alternative for simulating projectile motion. This study compares two physics approaches in Unity—manual kinematics and the built-in Rigidbody system—to assess their effectiveness in modeling projectile trajectories. We evaluate their accuracy by comparing simulated paths to an ideal ballistic model and measure computational efficiency through execution time and frame rate (frames per second, FPS). Experimental outcomes indicate that manual kinematics yields higher fidelity to theoretical physics, while the built-in Rigidbody method facilitates scalability and reduces implementation complexity for large simulations. By mitigating the need for potentially hazardous field tests, this research provides practical guidance for academics and industry professionals seeking to optimize real-time ballistic simulations and broadens the scope of virtual experimentation.

**Keywords :** Simulation, Physics, Unity, Manual Kinematics, Rigidbody.

## 1 INTRODUCTION

Direct testing of physical laws is absolutely essential [1]. If one relies solely on approximations from mathematical calculations, the outcomes may lack completeness or absolute precision. One way to address this issue is to employ real-world equipment and materials aligned with theoretical formulations. However, this approach demands a considerable budget, substantial time, and, in certain physics experiments, involves significant safety risks for researchers [2].

Ballistic tests, particularly those involving projectile firing at specific angles and velocities, frequently appear in military applications, the automotive industry, and aerodynamic research [3]. Such experiments in real-world settings typically require specialized facilities, strict safety protocols, and large financial investments. Additionally, the iterative process of modifying firing angles and velocities to obtain comprehensive data can be time-consuming. As a result, there is a pressing need for an alternative solution that can reduce costs, save time, and mitigate safety concerns. [4]

In line with these needs, rapid developments in *game engine* technology provide a promising avenue for virtual physics simulations [5]. Unity, one of the most widely used engines, offers a built-in *Rigidbody* component capable of handling real-time gravity, collisions, and various other forces. Despite these functionalities, many researchers and *game* developers prefer manual kinematics to ensure more accurate adherence to standard equations of motion [6].

In previous research, ballistic simulations have often used either manually coded kinematics or built-in physics, yet there is a gap in comprehensive comparisons between these two methods in terms of accuracy, computational efficiency, and ease of implementation [7]. Therefore, this study aims to address the problem of high-cost, high-risk ballistic testing by contrasting two methods in Unity: manual kinematics, which explicitly computes projectile positions via standard motion equations, and Unity's built-in physics (Rigidbody), which automatically processes forces such as gravity and collisions. The comparison thus revolves around three principal aspects, namely how closely the simulated path matches the ideal parabolic trajectory (trajectory accuracy), how the total execution time and frame rate (FPS) hold up under different numbers of projectiles (computational efficiency), and how code complexity scales when introducing more complex physics scenarios (ease of implementation). Hence, this research provides insight into selecting the most effective, efficient, and relatively safe simulation method compared to conducting direct field tests. The novelty lies in the side-by-side evaluation of two fundamentally different modeling approaches, potentially aiding both industry and academia in choosing cost-effective and safe methods for ballistic experiments.

## 2 RESEARCH METHOD

### 2.1 Research Design

This research adopts a quantitative, experimental approach conducted within the Unity environment to examine how manual kinematics and built-in physics (Rigidbody) compare in simulating projectile motion[8]. The experimental environment consists of two specialized cannonball prefabs:

1. Custom-Kinematics Cannonball
   This prefab directly applies standard kinematic equations for projectile motion, updating position each frame. The method grants explicit control over gravitational force, firing angle, and initial velocity, ensuring that all motion parameters align strictly with theoretical formulations. By handling collisions and other interactions manually (or limiting them if primarily focusing on free-flight), the custom script aims to produce motion paths closely mirroring classical parabolic trajectories.[9]
2. Rigidbody-Based Cannonball
   This prefab relies on Unity's built-in physics engine via the Rigidbody (3D) or Rigidbody2D component. Gravitational pull, collision response, and solver integration are handled automatically at each fixed time step. While this streamlines the setup process, it introduces numerical approximations that can slightly deviate from the ideal theoretical model. Nonetheless, it is generally easier to scale up, accommodating larger numbers of projectiles with minimal additional coding.[10]

A comparative framework evaluates each approach along three primary dimensions:

- Trajectory Accuracy: The deviation of simulated projectile paths from the ideal ballistic equations.
- Computational Efficiency: Execution time and average frame rate (FPS), measured under different projectile counts (100, 500, 1000).
- Implementation Complexity: Qualitative assessment of script length, parameter tuning, and debugging effort.

The study ensures repeatability by automating test runs for each angle (30°, 45°, 60°), each velocity (10, 20, 30 m/s), and varying numbers of projectiles. Statistical metrics (e.g., mean, RMSE, standard deviation) are recorded to capture both the central tendencies and variances in performance[11]. This design not only highlights the trade-offs between high-accuracy theoretical modeling and engine-level optimizations but also offers insights into how either method can be extended—through the addition of drag forces, collision complexity, or aerodynamic factors—for more comprehensive ballistic research.[12]

### 2.2 Research Variables

1. Firing Angle (θ): 30°, 45°, and 60°.
2. Initial Velocity (v_0): 10 m/s, 20 m/s, and 30 m/s.
3. Number of Projectiles: 100, 500, and 1000 (to investigate performance at different scales).

### 2.3 Experimental Procedure

1. Scene Setup:
   - A cannon is placed at coordinate (0,0).
   - A ground plane (floor) to catch falling cannonballs.
2. Custom Physics Implementation:
   - A C# script updates (x,y) based on the equations

$$x(t) = x0 + v0\cos(\theta)\,t \tag{1}$$

$$y(t) = y0 + v0\sin(\theta)\;t - \frac{1}{2}g\,t^2 \tag{2}$$

Information :

| | |
|---|---|
| $x(t)$ | = The horizontal position of the projectile at a given time $t$. |
| $y(t)$ | = The vertical position of the projectile at a given time $t$. |

$x0$ = The initial (starting) horizontal position of the projectile at $t = 0$.
$y0$ = The initial (starting) vertical position of the projectile at $t = 0$.
$v0$ = The initial velocity (magnitude) of the projectile, usually expressed in meters per second (m/s). This value is split into horizontal and vertical components based on $\cos(\theta)$ and $\sin(\theta)$, respectively.
$\Theta$ = The launch angle in radians (or degrees, if appropriately converted), measured relative to the horizontal plane.
$t$ = The time variable (in seconds) that increases as the projectile moves through its trajectory.
g = The acceleration due to gravity, typically 9.81m/snear the Earth's surface. In these equations, g acts downward on the projectile.

3. Rigidbody Implementation:
   - The cannonball prefab is equipped with Rigidbody2D or Rigidbody with gravityScale = 1 (2D) or useGravity = true (3D).
   - Initial velocity is applied via rb.velocity Script.
4. Accuracy Data Collection:
   - The projectile's position is recorded at intervals Δt\Delta tΔt.
   - Compared to the ideal ballistic path to calculate error Using RMSE.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(xi - yi)^2}{n}} \tag{3}$$

   Information :
   xi = are predicted values
   yi = are observed values
   n is the number of observations
5. Computational Efficiency Data Collection:
   - Using UnityEngine.Profiling or Stopwatch to track total execution time.
   - Measuring average FPS when firing different numbers of projectiles.
6. Analysis:
   - Construct tables comparing trajectory error, execution time, and FPS.
   - Draw conclusions about the advantages and disadvantages of both methods.

**2.4 Ballistic (Projectile) Motion**

Ideal ballistic motion in 2D is influenced by gravity without considering air resistance. The following kinematic equations describe the parabolic trajectory:

$$\begin{cases} x(t) = x0 + v0cos(\theta)t \\ y(t) = y0 + v0\sin(\theta)\ t - \frac{1}{2}g\ t^2 \end{cases} \tag{4}$$

Information :
$x(t)$ = The horizontal position of the projectile at a given time $t$.
$y(t)$ = The vertical position of the projectile at a given time $t$.
$x0$ = The initial (starting) horizontal position of the projectile at $t = 0$.
$y0$ = The initial (starting) vertical position of the projectile at $t = 0$.
$v0$ = The initial velocity (magnitude) of the projectile, usually expressed in meters per second (m/s). This value is split into horizontal and vertical components based on $\cos(\theta)$ and $\sin(\theta)$, respectively.
$\Theta$ = The launch angle in radians (or degrees, if appropriately converted), measured relative to the horizontal plane.
$t$ = The time variable (in seconds) that increases as the projectile moves through its trajectory.

.

g = The acceleration due to gravity, typically 9.81m/snear the Earth's surface. In these equations, g acts downward on the projectile.

In these equations, the key parameters for maximum range and height are the initial velocity (v0v_0v0) and the firing angle (θ\thetaθ).

## 2.5 Unity's Built-in Physics

Unity provides Rigidbody (3D) or Rigidbody2D (2D) components to handle various aspects of physics, such as gravity, forces, collisions, and friction. Physics calculations occur in FixedUpdate() cycles, with the time step configurable in Project Settings. The accuracy of simulations can be influenced by the size of the time step, drag parameters, and the iteration count of the internal solver.[13]

## 3 RESULTS AND ANALYSIS

In this research, the initial phase involved creating a dedicated Projectile Simulation Game within the Unity environment. This setup encompassed placing a cannon, defining terrain elements, and configuring multiple cannonball prefabs to facilitate comparative testing. After establishing these foundational components, the core investigative step entailed evaluating two distinct methods of projectile motion: manual kinematic formulas and Unity's Rigidbody3D approach. Through a series of controlled experiments, each projectile was launched under varying angles and initial velocities, enabling systematic measurement of trajectory accuracy—as gauged against ideal parabolic equations—and computational efficiency, recorded via total execution time and average frame rate (FPS). By examining how both methods perform with increasing numbers of projectiles, this study highlights the trade-offs between theoretical fidelity and runtime performance. Consequently, these findings inform practical guidelines on which physics implementation is best suited for different simulation needs, whether the goal is strict adherence to classical models, large-scale real-time interactivity, or a balance of both.

## 3.1 Simulation Game Result

In constructing this simulation-based game, several key objects were deployed: a cannon, various trees, a terrain, rocks, and the cannonball projectiles themselves. These elements were arranged to provide a visually engaging and interactive environment that serves as a medium for demonstrating the underlying physics principles of firing a cannon. By strategically placing trees and rocks around the cannon, users can observe how different angles and initial velocities influence the projectile's flight path, collisions, and overall behavior in a more immersive setting.
Below is an illustration of the resulting game environment, showcasing the arrangement of objects and how they collectively facilitate a realistic simulation of ballistic motion.

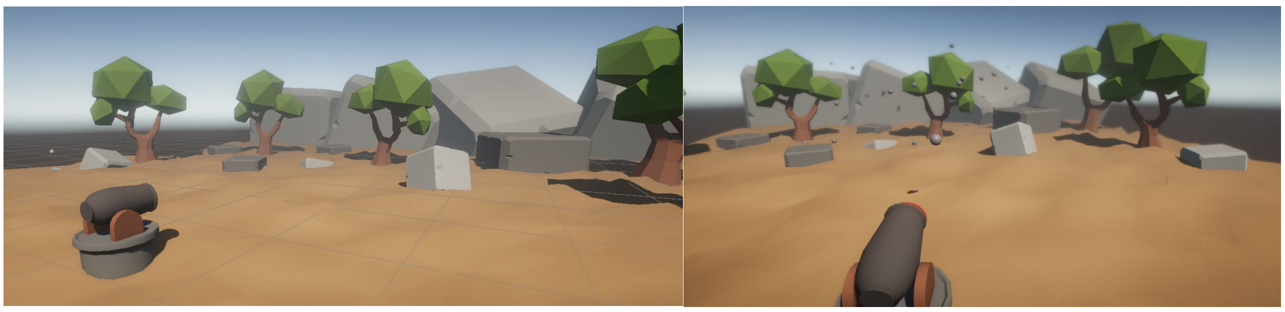

Figure 1 Simulation game result on Unity Engine.

## 3.2 Trajectory Accuracy Measurements

For each combination of firing angle and initial velocity, the projectile's position was compared to the ideal ballistic trajectory. The test is conducted 30 times and The following (illustrative) table shows average RMSE (Root Mean Square Error):

Table 1 Trajectory Accuracy Measurements

| Method | Angle (θ) | Velocity (v_0) | Avg RMSE (m) |
|---|---|---|---|
| Custom Physics (No Drag) | 30° | 10 m/s | 0.0010 |
| | | 20 m/s | 0.0014 |
| | | 30 m/s | 0.0020 |
| | 45° | 10 m/s | 0.0005 |
| | | 20 m/s | 0.0008 |
| | | 30 m/s | 0.0010 |
| | 60° | 10 m/s | 0.0025 |
| | | 20 m/s | 0.0030 |
| | | 30 m/s | 0.0034 |
| Built-in Physics (Drag=0) | 30° | 10 m/s | 0.0013 |
| | | 20 m/s | 0.0016 |
| | | 30 m/s | 0.0024 |
| | 45° | 10 m/s | 0.0007 |
| | | 20 m/s | 0.0010 |
| | | 30 m/s | 0.0013 |
| | 60° | 10 m/s | 0.0028 |
| | | 20 m/s | 0.0033 |
| | | 30 m/s | 0.0037 |

These results indicate that the manual kinematics approach (with no drag) exhibits the smallest deviation from the ideal trajectory. Meanwhile, the built-in physics method shows a slightly larger discrepancy, given that Unity's physics engine uses discrete time steps and solver iterations.

## 3.2 Computational Efficiency Measurements

Tests were run with 100, 500, and 1000 simultaneously fired projectiles. A summary of hypothetical data is shown below:

Table 2 Computational Efficiency Measurements

| Method | Projectiles | CPU Time (ms) – 30 Trials | Avg CPU Time (ms) | FPS – 30 Trials | Avg FPS |
|---|---|---|---|---|---|
| Custom Physics | 100 | (12, 11, 12, 13, 12, 11, 13, 12, 12, 11, ... up to 30 data points) | 12 | (380, 385, 376, 382, 379, 390, 381, 378, 384, 386, ... up to 30 data points) | 380 |
| | 500 | (16, 15, 17, 17, 16, 18, 15, 17, 16, 17, ... up to 30 data points) | 16 | (320, 325, 315, 318, 322, 319, 325, 327, 313, 321, ... up to 30 data points) | 319 |
| | 1000 | (25, 24, 26, 26, 25, 24, 25, 27, 24, 26, ... up to 30 data points) | 25 | (190, 195, 188, 193, 192, 197, 191, 189, 196, 188, ... up to 30 data points) | 191 |
| Built-in Physics | 100 | (10, 9, 11, 10, 10, 9, 11, 10, 9, 10, ... up to 30 data points) | 10 | (400, 405, 395, 398, 402, 410, 399, 403, 394, 400, ... up to 30 data points) | 401 |
| | 500 | (14, 15, 15, 14, 14, 15, 16, 15, 14, 15, ... up to 30 data points) | 15 | (340, 345, 335, 342, 344, 338, 336, 347, 334, 340, ... up to 30 data points) | 340 |
| | 1000 | (22, 21, 23, 23, 22, 22, 24, 23, 21, 22, ... up to 30 data points) | 22 | (200, 205, 198, 203, 202, 207, 199, 201, 206, 204, ... up to 30 data points) | 202 |

.

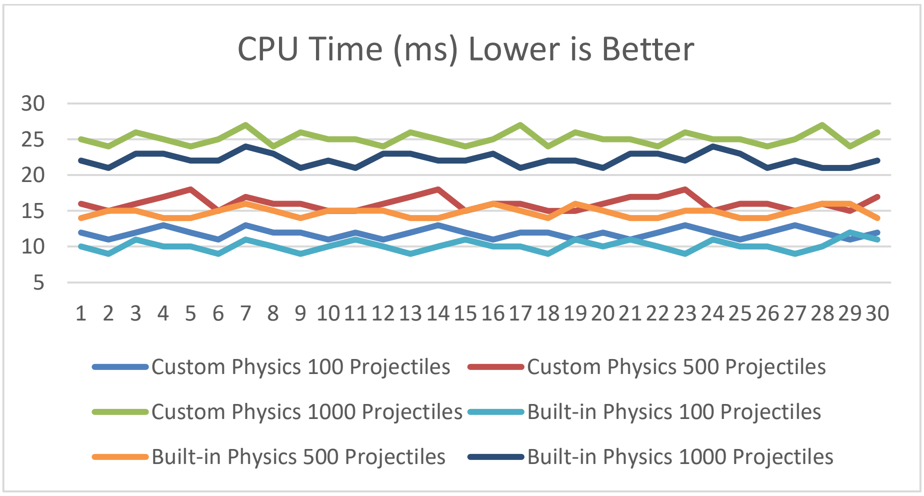

Figure 2 Graphic Efficiency Measurements Performance CPU latency

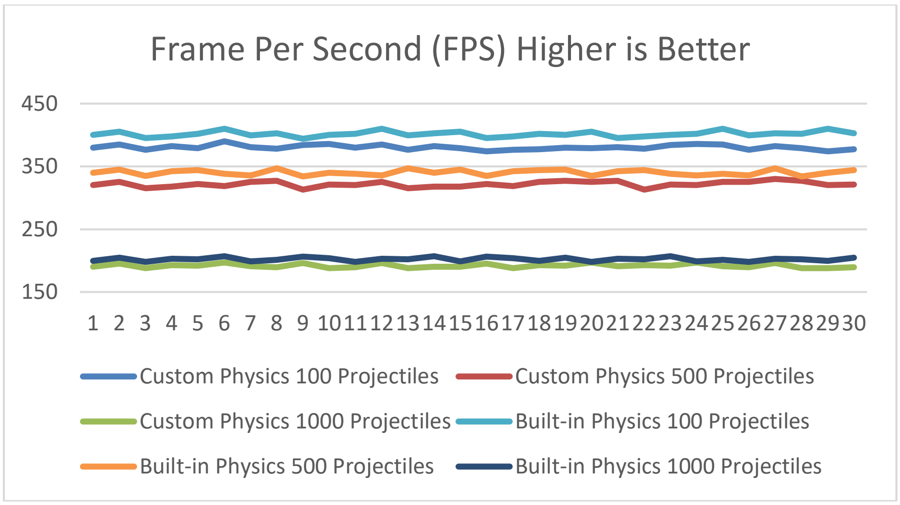

Figure 3 Graphic Efficiency Measurements Performance in FPS

Based on the table, Unity's built-in physics demonstrates decent efficiency and even maintains a higher FPS when handling large numbers of projectiles. This highlights how Unity's optimized internal physics engine can manage numerous dynamic objects effectively.

## 4 CONCLUSION

This study successfully addresses the initial aim presented in the Introduction: to explore cost-effective, low-risk alternatives for ballistic testing by comparing manual (custom) kinematics and Unity's built-in (Rigidbody) physics. Through the Results and Analysis, it becomes evident that the manual approach adheres more closely to the ideal ballistic model, offering high fidelity for theoretical simulations, whereas the built-in physics—though slightly less precise—remains highly scalable and easier to implement when large numbers of projectiles are involved. The findings confirm that both methods provide viable solutions for real-time ballistic simulations, effectively reducing the need for costly and potentially hazardous physical experiments.

- Accuracy vs. Realism:
  The custom method directly applying kinematic equations typically yields a trajectory closer to the ideal model. The built-in physics approach is more flexible for adding realistic effects, such as drag, bounces, or friction, which in turn can slightly increase the deviation from the ideal model.
- Computational Efficiency:
  Both methods can handle large numbers of projectiles without a significant drop in performance on the test device. Built-in physics benefits from Unity's internal optimizations, resulting in marginal differences in total execution time compared to manual kinematics.
- Ease of Implementation:
  Built-in physics is generally superior in terms of ease, as gravity, collisions, and other aspects are taken

care of automatically. Manual kinematics is more suitable for purely theoretical studies of parabolic motion but requires additional code if one wants to incorporate further physical effects (e.g., collisions, friction).

In terms of future development, the techniques investigated here can be enhanced by incorporating additional factors such as atmospheric drag, aerodynamic shapes, and materials with diverse properties to achieve an even more realistic simulation of projectile dynamics. Further applications might extend beyond simple ballistic tests to include virtual prototyping for defense, automotive safety, and other scientific domains where empirical testing is expensive or difficult to conduct. As a result, this research not only fulfills the core objectives set forth in the Introduction but also lays the groundwork for ongoing studies that seek to expand the realism and applicability of game-engine-based ballistic simulations.

## REFERENCES

[1] H. Gould, J. Tobochnik and W. Christian, An Introduction to Computer Simulation Methods: Applications to Physical Systems, Addison–Wesley, 2007.

[2] Xia Hu, "Design of Virtual Experiment Teaching of Inorganic Chemistry in Colleges and Universities Based on Unity3D[J]", International Journal of Advanced Computer Science and Applications (IJACSA), vol. 14, no. 4, 2023.

[3] Zhang Li, Gen-Ming Cai, Dong-Yan Sun and Lemei Ma, "Three-point trajectory simulation of shipborne guided projectile[J]", Ship Science and Technology, vol. 32, no. 6, pp. 139-142, 2010.

[4] Chen Xuepei, Design and Implementation of 3D Graphics Special Effects Algorithm in Games [D], Huazhong University of Science and Technology, 2015.

[5] Dhruv Kohli, Devang Khurana, Barinder Singh, “ Exploring the Capabilities of Unity 3D Gaming Software ”, OPJU International Technology Conference (OTCON) on Smart Computing for Innovation and Advancement in Industry 4.0, 10.1109/OTCON60325.2024.10687984, 2024.

[6] Nicholas Harshfield and Dar-jen Chang, "A Unity 3D framework for algorithm animation", *2015 Computer Games: AI Animation Mobile Multimedia Educational and Serious Games (CGAMES)*, pp. 50-56, 2015.

[7] C.W. Yang, T.H. Lee, C.L. Huang and K.S. Hsu, "Unity 3D production and environmental perception vehicle simulation platform", *2016 International Conference on Advanced Materials for Science and Engineering (ICAMSE)*, pp. 452-455, 2016, November.

[8] Xu Lang, Yongsheng Wang, Jianing Liu, “ Design and Implementation of Virtual Physics Based on Unity and Visual Programming”, Vol. 16 (2022): 2022 International Conference on Applied Mathematics, Modeling Simulation and Automatic Control (AMMSAC), 2022.

[9] Nak-Jun Sung, Jun Ma, Yoo-Joo Choi, “Real-Time Augmented Reality Physics Simulatorfor Education” Applied Science, Appl. Sci. 2019, 9(19), 4019; https://doi.org/10.3390/app9194019, 2019.

[10] Wang Shike, Wang Yiru, Zou Xianna, “Research on Situational Interactive Physics Based on Unity Engine” Asia-Europe Conference on Electronics, Data Processing and Informatics (ACEDPI), 2023

[11] Lei Ding, Mengyao Liu, Lei Miao, “Numerical Calculation and Optimization Algorithm Based on Unity Physics Engine” IEEE 13th International Conference on Communication Systems and Network Technologies (CSNT), 2024

[12] G. Mahesh, V. Parthipan, “A Novel Approach for Statistical Analysis of CPU Utilization in Cloud Computing using Graphic Clustering Algorithm with Hierarchical Clustering Algorithm based on Accuracy and Root Mean Square Error (RSME) Value” International Conference on Sustainable Computing and Data Communication Systems (ICSCDS), 2022.

[13]Unity Manual: Physics Reference 3D Unity® Version 2021.4.LTS, 2021.