Unity vs. Blender Physics Simulation Performance Analysis

Paola Rojas Domínguez

***Abstract*—This report presents a comparative analysis between Blender and Unity in the context of robotic simulations, with a focus on unmanned aerial vehicle (UAV) applications. The study highlights Blender's strengths in 3D asset modeling, geometry creation, and animation, while Unity demonstrates superior performance in real-time physics, interaction, and control logic integration. By examining workflows combining both platforms, the report emphasizes that the most effective approach relies on using Blender for detailed modeling and Unity for dynamic simulation. Furthermore, while USD-based connectors enable seamless transfer of geometry and textures, physical properties and behaviors must be defined within the simulation environment. Overall, Blender and Unity prove to be complementary tools that, when integrated, provide a robust framework for robotics development.**

***Index Terms*—Omniverse, Drones, Unity, Blender, UAV, 3D-models.**

I. Introduction

UMANNED aerial vehicle (UAV) simulation demands both precise 3D modeling and a physics environment capable of responding in real time to control inputs. In this sense, Unity and Blender represent two complementary approaches. Unity has established itself as a game engine optimized for interactive simulation and dynamic behavior programming, while Blender has positioned itself as the benchmark tool for high-quality modeling and animation, incorporating a physics engine based on Bullet Physics.

This report compares both platforms through various case studies, both compiling various third-party projects and a proprietary project: a 3D-modeled drone that performs basic takeoff and hover maneuvers. The purpose is to identify the advantages and limitations of each software when developing projects that integrate modeling, animation, and physics simulation in robotics applications.

II. Comparing projects in Unity and Blender

A. Tornado Simulator

We compare the performance of Unity and Blender in a tornado simulation project. We observe that both engines offer particular advantages and limitations in the context of simulating complex natural phenomena [1].

Blender, despite recording the simulation at 20 frames per second, demonstrates fluid and efficient execution thanks to its robust graphics and physics computation system, making it suitable for visual experiments and rapid prototyping. Unity, for its part, demonstrates a high capacity to handle large numbers of particles, in the range of 800 to 900 without compromising speed, and stands out for its optimization in interactive and real-time environments. However, the transition between the two engines requires code rewriting and detailed configuration of rigid bodies and particle systems, which represents a technical challenge but also an opportunity to generate more dynamic simulations.

The application of external forces, such as wind, to objects with distinct physical properties (trees, particles, or rigid elements) reveals the importance of proper mathematical and physical parameterization, since the objects' dynamic responses directly depend on these calculations. In this sense, the project illustrates how mastery of the fundamentals of physics, trigonometry, and calculus becomes essential for obtaining realistic and optimized results, underscoring that the quality of the simulation depends not only on the engine's capabilities, but also on the developer's technical knowledge. Overall, the comparison shows that Blender is more accessible for the initial creation of visual models and simulations, while Unity is positioned as a more robust platform for the implementation of interactive and scalable environments, reaffirming the complementarity of both tools in the simulation and computer graphics development workflow.

B. Blender vs Unity comparison in four categories

A detailed comparison between Blender and Unity will now be made in four key categories, providing a technical overview of the strengths and unique features of each platform [2].

In the modeling field, Blender proves to be a specialized tool with a high degree of control over geometry, digital sculpting, and procedural rendering, giving it a considerable advantage for the creation of detailed visual content. In contrast, Unity is not designed as a modeling software, but as a runtime engine, so it relies heavily on models imported from other platforms, standing out more for its compatibility and resource integration than for its native construction capabilities.

Regarding texturing, Blender offers a robust shading pipeline and node-based materials, allowing for very precise physical and artistic control of surface appearance. Unity, although it incorporates advanced systems such as the Shader Graph and supports PBR (Physically Based Rendering) materials, focuses its strength on optimizing these materials for real-time execution, prioritizing efficiency over absolute detail.

In the category of animation and game engine, the gap widens: Blender has solid animation tools for characters and simulations, but lacks a native interaction system. Unity, on the other hand, has established itself as a real-time engine

with a powerful scripting, physics, and logic control system, allowing it to bring animations to an interactive and scalable environment.

Finally, in the ”What has it produced?” category, the video shows how Blender has been adopted in multiple high-quality audiovisual productions and short films (for example, the Blender Foundation’s open projects), while Unity has established itself as a benchmark in the video game and interactive simulation industry, supported by commercial titles and applications in sectors such as augmented reality and virtual reality. Overall, the comparison shows that Blender and Unity do not directly compete in the same fields, but rather complement each other’s workflows: Blender as a tool for artistic and technical asset creation and Unity as a platform for real-time execution, interaction, and deployment.

III. 3D Drone Modeling

In order to compare both software packages in a more robotics-related context, a drone model was downloaded from the Sketchfab platform, specifically the resource called UAV Drone Shop (War Thunder) Props. This model was used in both environments to perform a simple test that consisted of simulating take-off and forward, backward, and lateral movements. This procedure established a common ground on which the capabilities of Unity and Blender were compared.

A. Unity

The first implementation was carried out in Unity. The model in `.obj` format was imported directly into the environment, and a C# script was developed that allowed the drone to be controlled in real time using the keyboard. The space bar activated takeoff, while the `W`, `A`, `S`, and `D` keys executed forward, left, backward, and right movements, respectively. Likewise, the rotation of the propellers was programmed to visually simulate the flight action.

```
using UnityEngine;

public class DroneController :
 ↪  MonoBehaviour
{
    public float thrust = 30f;
    private Rigidbody rb;

    [Header("Hélices")]
    public GameObject[] propellers;
    public float spinSpeed = 1000f;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    void FixedUpdate()
    {
        // Aplicar fuerza vertical
        if (Input.GetKey(KeyCode.Space))
        {
            rb.AddForce(Vector3.up *
             ↪  thrust);

            // Girar hélices
            foreach (GameObject prop in
             ↪  propellers)
            {
                prop.transform.Rotate(Vec
                 ↪  tor3.forward *
                 ↪  spinSpeed *
                 ↪  Time.deltaTime);
            }
        }

        // Flechas para moverlo
        if (Input.GetKey(KeyCode.W))
         ↪  rb.AddForce(Vector3.forward *
         ↪  thrust);
        if (Input.GetKey(KeyCode.S))
         ↪  rb.AddForce(Vector3.back *
         ↪  thrust);
        if (Input.GetKey(KeyCode.A))
         ↪  rb.AddForce(Vector3.left *
         ↪  thrust);
        if (Input.GetKey(KeyCode.D))
         ↪  rb.AddForce(Vector3.right *
         ↪  thrust);
    }
}
```

An initial problem was that the propellers did not rotate on their own axis but rather around the entire drone. To solve this problem, the model was opened in Blender, where the origin of each propeller was adjusted to the corresponding geometric center and again exported in the `.fbx` format. Once re-imported into Unity, the behavior was as expected: the propellers rotated correctly and the drone responded in real time to user input. This result confirmed Unity’s ability to combine 3D modeling, dynamic animations, and physics simulation into a single workflow, enabling direct experimentation with user controls and evaluation of model stability in an interactive physics environment.

The results obtained can be seen in the following YouTube video: 3D Drone Model - Unity

B. Blender

The second implementation was carried out in Blender using the same model, already corrected for the propeller pivots. Unlike Unity, Blender does not natively offer an environment for real-time interactive simulation, so the strategy consisted of developing an animation using the internal animation engine and a Python script. The drone was configured to execute a controlled takeoff and subsequently trace a square-shaped trajectory in the air. This procedure did not require user interaction during execution, as movements were defined as keyframes on the timeline.

```
import bpy
from math import radians
```

```
# CONFIGURACIÓN DEL DRON
drone_collection =
↪ bpy.data.collections["Collection"]

# Cuerpo y hélices
cuerpo = bpy.data.objects["Cuerpo"]
helices = [obj for obj in
↪ drone_collection.objects if "Helice"
↪ in obj.name]

# Parámetros
velocidad_mov = 0.05          # avance por
↪ frame
giro_helice = radians(50)     # rotación por
↪ frame
frames_por_lado = 50          # duración de
↪ cada lado
altura = 1.0                  # altura fija
↪ de vuelo

# Origen inicial
origen = (0.0, 0.0, 0.0)

# HANDLER
def frame_handler(scene):
    frame = scene.frame_current
    lado = (frame // frames_por_lado) % 4
    paso = frame % frames_por_lado

    # Calcular posición base
    x, y, z = origen

    if lado == 0:    # Adelante (eje Y+)
        x = origen[0]
        y = origen[1] + paso *
        ↪ velocidad_mov
    elif lado == 1: # Derecha (eje X+)
        x = origen[0] + paso *
        ↪ velocidad_mov
        y = origen[1] + frames_por_lado *
        ↪ velocidad_mov
    elif lado == 2: # Atrás (eje Y-)
        x = origen[0] + frames_por_lado *
        ↪ velocidad_mov
        y = origen[1] + (frames_por_lado -
        ↪ paso) * velocidad_mov
    elif lado == 3: # Izquierda (eje X-)
        x = origen[0] + (frames_por_lado -
        ↪ paso) * velocidad_mov
        y = origen[1]

    z = altura

    # Mover cuerpo del dron
    cuerpo.location = (x, y, z)

    # Girar hélices sobre su propio eje
    for h in helices:
        h.rotation_euler.z += giro_helice

# CONECTAR HANDLER
bpy.app.handlers.frame_change_pre.clear()
bpy.app.handlers.frame_change_pre.append(
↪ frame_handler)
```

Although this approach does not offer the flexibility to modify parameters during the simulation, it does allow for highly detailed visualization of the trajectory and recording of animation sequences, which are useful in preview or presentation stages. In this regard, Blender demonstrated its strength in producing precise, high-quality animations, although it was limited in terms of real-time physics simulation.

In this YouTube video, you can see how the animation turned out: 3D Drone Model - Blender

IV. Conclusion

The comparative analysis conducted in this report highlights the complementary roles of Blender and Unity in developing robotics-oriented simulations. Blender excels as a modeling tool, offering advanced capabilities for geometry creation, texture, and animation, making it the preferred environment for creating detailed and accurate 3D assets. Unity, on the other hand, demonstrates clear advantages in real-time physics simulation, user interaction, and dynamic behavior scripting, essential for testing UAV maneuvers or control algorithms.

Therefore, the most effective workflow is to model and prepare assets in Blender and then import them into Unity to implement real-time physics and interaction logic. Current connectors between Blender and Unity, particularly through the USD format, facilitate the transfer of geometry and textures but do not preserve physical behaviors or control scripts. Therefore, while the USD export workflow is valuable for resource portability, defining forces, constraints, and simulation parameters must be done directly on the target platform, whether Unity or Omniverse.

In conclusion, Blender and Unity are not competitors but complementary tools. Blender provides the artistic and structural foundation for UAV models, while Unity provides the interactive and physical realism necessary for robotic simulation, enabling a robust and efficient development process.

References

[1] Sci Fi Animator. (2013, December 23). *Blender and Unity Tutorial Game Engine Physics Comparison Test* [Video]. YouTube. https://www.youtube.com/watch?v=MX1gvI-Z9As

[2] Sorcerer. (2020, May 31). *Unity vs Blender — Comparison 2020* [Video]. YouTube. https://www.youtube.com/watch?v=2ke4qJryzxw

[3] Sketchfab. (s.f.). *Sketchfab - The best 3D viewer on the web.* https://sketchfab.com/

[4] Omniverse Unity Connector — NVIDIA NGC. (n.d.). NVIDIA NGC Catalog. Retrieved August 26, 2025, from https://catalog.ngc.nvidia.com/orgs/nvidia/teams/omniverse/resources/omni_unity_connector