Interactive Visualizations with Cesium and Unreal Engine

Daniel Aláez Gómez
Area of Languages and Computer Systems
Department of Statistics, Computer Science and Mathematics
Public University of Navarra

Abstract

This course provides a practical and conceptual introduction to creating interactive geospatial visualizations using Cesium and Unreal Engine. It is designed for students, researchers, and professionals who wish to integrate real-world data into immersive virtual environments. The main objectives are to:

- Understand the fundamental concepts of real-time graphics rendering, 3D engines, and geospatial data integration.
- Learn how to install, configure, and use Unreal Engine and essential plugins, including Cesium for Unreal.
- Build a digital twin of the real world, combining terrain, buildings, and dynamic data sources.
- Implement interactive visualizations using Blueprints, visual scripting, and real-time data APIs.
- Develop critical skills for digital twin and smart city applications, including optimization for VR and collaborative environments.

Through hands-on exercises and guided projects, participants will gain the necessary skills to visualize, interact with, and analyze spatial data in a photorealistic, interactive, and extensible 3D environment. The associated project files can be found on GitHub: https://github.com/UPNAdrone/TwIN_Unreal_Cesium.

Contents

1 Part 1: Theory

1.1 Introduction to Unreal Engine as a Real-Time Graphics Rendering Engine

1.1.1 History and Evolution of Unreal Engine

Unreal Engine represents one of the most significant developments in real-time graphics rendering technology. Developed by Epic Games since 1998, it has evolved from a specialized game engine into a comprehensive, multi-purpose platform that serves diverse industries including simulation, cinema, architecture, urban planning, and professional training. The engine's architecture has been fundamentally redesigned across major versions to address the increasing demands for realism, performance, and accessibility.

The evolutionary trajectory of Unreal Engine demonstrates a consistent focus on democratizing high-end graphics technology while pushing the boundaries of real-time rendering capabilities. Each major version has introduced paradigm-shifting technologies that have subsequently influenced industry standards.

- **UE1 (1998):** The inaugural version introduced the first integrated visual editor, establishing the foundation for Unreal's user-friendly development environment. It featured a revolutionary level editor that allowed designers to create complex 3D environments without extensive programming knowledge, setting a precedent for visual development tools in the industry.
- **UE3 (2006):** This version marked a significant leap in rendering technology with advanced lighting systems including dynamic shadows, per-pixel lighting, and sophisticated material systems. The introduction of multiplatform support enabled developers to target consoles, PC, and mobile devices from a single codebase, fundamentally changing game development workflows.
- **UE4 (2014):** Perhaps the most transformative version, UE4 introduced Blueprints, a visual scripting system that enabled non-programmers to create complex game logic and interactions. The decision to provide source code access under a royalty-based licensing model democratized access to AAA-quality game engine technology. Additionally, UE4 introduced physically-based rendering (PBR), advanced particle systems, and a modern material editor.
- **UE5 (2021):** The current generation introduces revolutionary technologies: Nanite virtualized geometry system eliminates traditional polygon budgets by streaming geometric detail at the pixel level, Lumen provides fully dynamic global illumination that responds to scene changes in real-time, and World Partition enables seamless streaming of massive open worlds. These technologies collectively enable unprecedented levels of visual fidelity while maintaining real-time performance.

1.1.2 Fundamental Concepts of Graphics Engines

Understanding the core principles underlying graphics engines is essential for effective utilization of Unreal Engine in geospatial visualization contexts. These concepts form the theoretical foundation that enables the creation of photorealistic, interactive 3D environments.

- **Real-time rendering:** Unlike pre-rendered graphics, real-time rendering generates images instantaneously (typically at 30-120 frames per second) in response to user interaction. This process requires sophisticated optimization techniques to maintain consistent frame rates while handling complex scenes. The fundamental challenge lies in balancing visual quality with computational efficiency, particularly when rendering large-scale geospatial datasets. Real-time rendering is essential for interactive applications where user agency and immediate feedback are paramount, such as virtual tours, training simulations, and interactive urban planning visualizations.

- **Graphics pipeline:** The graphics pipeline represents a sequential series of computational stages that transform 3D scene data into 2D rasterized images displayed on screen. The pipeline typically includes: (1) *Application stage*, where scene geometry and objects are prepared; (2) *Geometry processing*, involving vertex shaders that transform 3D coordinates, perform vertex-level calculations, and prepare primitives for rasterization; (3) *Rasterization*, converting geometric primitives into fragments (potential pixels); (4) *Fragment processing*, where pixel shaders determine final pixel colors through texture sampling, lighting calculations, and material evaluation; (5) *Output merging*, handling depth testing, alpha blending, and final frame buffer composition. Understanding this pipeline is crucial for optimizing performance in large-scale geospatial visualizations.
- **Materials and shaders:** Materials define the visual properties of surfaces through shader programs—specialized code executed on the GPU. Modern physically-based rendering (PBR) materials simulate real-world light-surface interactions using properties such as albedo (base color), metallicness, roughness, and normal maps. Shaders implement complex algorithms for calculating how light interacts with surfaces, including Fresnel reflections, subsurface scattering, and anisotropic highlights. In geospatial contexts, materials enable realistic representation of diverse terrain types, building facades, and natural features while maintaining performance through efficient shader optimization.
- **Lighting:** Lighting simulation encompasses multiple techniques for achieving photorealism. *Direct lighting* calculates illumination from light sources directly visible to surfaces, while *indirect lighting* (global illumination) accounts for light bouncing between surfaces, creating realistic ambient illumination and color bleeding. Modern engines employ techniques such as lightmaps for static lighting, dynamic shadow mapping for real-time shadows, screen-space reflections for reflective surfaces, and volumetric lighting for atmospheric effects. In geospatial visualizations, accurate lighting is crucial for temporal realism—simulating different times of day and weather conditions enhances the immersive quality of virtual environments.

1.1.3 Architecture of Unreal Engine 5

Unreal Engine 5 employs a component-based architecture that promotes modularity, reusability, and efficient resource management. This architecture is particularly well-suited for geospatial applications where large-scale data must be organized and managed effectively.

- **Scenes and levels:** Levels serve as the fundamental organizational unit for 3D content, functioning as discrete containers for all scene elements including geometry, actors, lighting, and environmental effects. In UE5, the World Partition system enables seamless management of massive open worlds by automatically dividing levels into streaming cells, loading and unloading content based on viewer proximity. This spatial partitioning is essential for geospatial visualizations covering extensive geographic areas, as it allows the engine to maintain performance while rendering city-scale or regional datasets. Levels can reference external assets, support multiple streaming levels, and implement level-specific gameplay logic or visual configurations.

- **Actors:** Actors represent the fundamental interactive entities within Unreal Engine, serving as the base class for all objects that can be placed in a level. Every actor can contain multiple *Components*—modular pieces of functionality such as mesh renderers, collision volumes, audio sources, or custom scripted behaviors. This component-based design enables flexible composition: a single actor might combine a static mesh component for visual representation, a collision component for physics interaction, and a Cesium Globe Anchor component for geospatial positioning. Actors support inheritance hierarchies, allowing specialized types (Pawns for controllable entities, Characters for humanoid actors) while maintaining a consistent interface for level management and interaction.

- **Meshes:** Static meshes represent 3D geometry data stored in optimized formats that can be efficiently rendered by the GPU. Unreal Engine supports multiple mesh formats (FBX, OBJ, glTF) and provides extensive import options for controlling LOD generation, collision mesh creation, and material assignment. With Nanite technology in UE5, meshes can contain billions of polygons while maintaining real-time performance through virtualized geometry streaming. Skeletal meshes extend this concept to support animation through bone hierarchies, enabling animated characters, vehicles, or environmental elements. For geospatial applications, meshes typically represent buildings, terrain features, infrastructure elements, or 3D photogrammetry reconstructions.

- **Blueprints:** Blueprints constitute Unreal Engine's visual scripting system, providing a node-based interface for creating game logic, interactions, and system behaviors without traditional programming. Blueprints compile to bytecode executed by the engine's virtual machine, offering performance comparable to C++ for most use cases while dramatically reducing development time. The system supports multiple blueprint types: *Class Blueprints* define reusable actor types with custom behaviors, *Level Blueprints* contain level-specific logic, *Function Libraries* provide reusable utility functions, and *Interface Blueprints* define contracts for polymorphic behavior. For geospatial applications, Blueprints enable rapid prototyping of data visualization systems, real-time data integration workflows, and interactive exploration mechanics.

1.1.4 Ultra-realism in Unreal Engine 5

Unreal Engine 5 introduces several revolutionary technologies that collectively enable unprecedented levels of visual realism while maintaining real-time performance. These systems are particularly valuable for geospatial visualizations where accurate representation of real-world environments is paramount.

- **Nanite:** Nanite represents a paradigm shift in geometry rendering, implementing a virtualized micropolygon geometry system that eliminates traditional polygon budgets. The system automatically streams and renders geometric detail at the pixel level, meaning that a mesh can contain billions of triangles while only rendering the detail actually visible to the viewer. Nanite achieves this through hierarchical level-of-detail (HLOD) clustering, where geometry is organized into clusters that can be independently streamed and rendered. This technology is transformative for photogrammetry-based geospatial visualizations, as it allows direct import of high-resolution 3D reconstructions without manual optimization or decimation. The system automatically handles LOD transitions, culling, and occlusion, ensuring optimal performance regardless of scene complexity.

- **Lumen:** Lumen provides fully dynamic global illumination that responds to scene changes in real-time without requiring lightmap baking. The system combines multiple techniques:

hardware-accelerated software ray tracing for direct lighting, signed distance field (SDF) global illumination for indirect lighting, and screen-space techniques for reflections and ambient occlusion. Lumen automatically adapts to dynamic lighting conditions, moving objects, and material changes, making it ideal for geospatial visualizations that simulate different times of day, weather conditions, or dynamic scene modifications. The system provides physically accurate light transport, including multi-bounce indirect illumination, color bleeding, and realistic shadow penumbrae, significantly enhancing the perceptual realism of virtual environments.

- **MetaHuman:** While primarily designed for character creation, MetaHuman demonstrates Unreal Engine's capabilities for generating highly realistic digital assets. The system provides a cloud-based authoring tool that generates photorealistic human characters with extensive customization options, realistic facial animation, and high-quality hair rendering. For geospatial applications, this technology illustrates the engine's capacity for creating realistic avatars in virtual city environments, training simulations, or collaborative visualization platforms. The underlying principles of high-fidelity asset generation apply broadly to other types of realistic content creation.
- **VSM (Virtual Shadow Maps):** Traditional shadow mapping techniques struggle with large-scale scenes due to resolution limitations and performance constraints. Virtual Shadow Maps address these limitations by implementing a virtualized shadow map system that provides high-resolution shadows across extensive areas. The system uses a sparse representation that allocates shadow map resolution dynamically based on viewer proximity and importance, ensuring that nearby objects receive high-quality shadows while distant areas use lower resolution efficiently. This technology is essential for geospatial visualizations where accurate shadow representation across city-scale or regional environments enhances spatial understanding and temporal realism (e.g., shadows moving throughout the day).

1.1.5 Applications of the Unreal Engine

- **Video games:** Development for multiple platforms, including VR and mobile.
- **Cinema and TV:** Virtual previsualization and visual effects.
- **Architecture:** Visualization of designs and interactive tours.
- **Simulation:** Training and technical analysis.
- **Geospatial environments:** Realistic visualizations thanks to integration with Cesium.

Glossary

- **Actor:** Interactive object within the scene.
- **Blueprint:** Visual tool for game logic and mechanics.
- **Level:** Part or scenario within Unreal.
- **Shader:** Program to define appearance under illumination.
- **Mesh:** 3D model constructed with vertices and faces.
- **Rendering:** Process of creating images from 3D models.
- **Real-time:** Immediate response to user interaction.

1.2 Cesium for Unreal: Creating Ultra-Realistic Worlds

1.2.1 What is Cesium and Why Integrate It?

Cesium represents a comprehensive ecosystem for 3D geospatial visualization, originally architected for web-based applications but now extending to desktop and game engine environments. The platform addresses fundamental challenges in geospatial visualization: accurate coordinate system handling, efficient streaming of massive datasets, and seamless integration of diverse geospatial data sources. Cesium's core innovation lies in its implementation of the 3D Tiles specification—an open standard for streaming heterogeneous 3D geospatial content that enables efficient transmission and rendering of city-scale or global datasets.

The integration of Cesium with Unreal Engine creates a powerful synergy: Unreal Engine provides state-of-the-art rendering capabilities, photorealistic lighting, and advanced visual effects, while Cesium contributes precise geospatial positioning, efficient data streaming, and access to global geospatial datasets. This combination enables the creation of photorealistic virtual environments that accurately represent real-world locations at any scale, from individual buildings to entire continents.

Why Use Cesium for Unreal?

- **Access to real-world geospatial data:** Cesium provides access to extensive global datasets including high-resolution terrain models, satellite imagery, building footprints, and 3D city models. The Cesium World Terrain dataset offers global elevation data at multiple resolutions, while Cesium Ion hosts curated collections of 3D content including photogrammetry reconstructions, building models, and infrastructure data. This eliminates the need to manually source, process, and integrate geospatial data, significantly accelerating development workflows.
- **Seamless integration of GIS data and photogrammetry:** Cesium for Unreal supports direct import of standard GIS formats (GeoJSON, KML, Shapefiles) and photogrammetry outputs (OBJ, FBX, glTF, LAS/LAZ point clouds). The system automatically handles coordinate system transformations, ensuring that imported data aligns correctly with the global coordinate system. This capability is essential for projects combining multiple data sources, such as combining municipal GIS data with drone-captured photogrammetry or integrating LiDAR surveys with satellite imagery.
- **Facilitates real-world simulations and visualizations:** The combination enables creation of digital twins—accurate virtual representations of real-world systems. Applications span multiple domains: smart city planning and visualization, architectural previsualization using real site contexts, heritage preservation through detailed 3D documentation, emergency response training in accurate geographic environments, urban planning with realistic visualization of proposed developments, and environmental monitoring through temporal visualization of geospatial data. The geospatial accuracy ensures that measurements, spatial relationships, and geographic context remain faithful to reality, making these virtual environments suitable for professional analysis and decision-making.

1.2.2 Principles of Geospatial Data

Effective utilization of Cesium for Unreal requires understanding fundamental geospatial concepts that differ from traditional 3D graphics coordinate systems. These principles ensure accurate representation and proper integration of real-world geographic data.

- **Projection and coordinate systems:** The Earth's curved surface cannot be perfectly represented on a flat plane without distortion. Various map projections (Mercator, UTM, etc.) attempt to minimize distortion for specific use cases, but each introduces trade-offs. Cesium uses the World Geodetic System 1984 (WGS84) as its fundamental coordinate system—a geodetic datum that defines a reference ellipsoid approximating Earth's shape. WGS84 uses a three-dimensional coordinate system: latitude (angular distance north/south of the equator, -90° to +90°), longitude (angular distance east/west of the Prime Meridian, -180° to +180°), and ellipsoidal height (elevation above the WGS84 ellipsoid). This geodetic coordinate system enables precise positioning anywhere on Earth and serves as the foundation for transforming data from other coordinate systems (UTM, State Plane, local grids) into Cesium's global coordinate space.

- **Coordinate system transformations:** Geospatial data often originates in local or regional coordinate systems optimized for specific geographic areas. Cesium for Unreal automatically handles coordinate transformations when importing data, converting from source coordinate systems (identified through spatial reference system identifiers or embedded metadata) to WGS84. Understanding these transformations is crucial when working with high-precision data, as transformation errors can result in misalignment of several meters or more. The system supports both automatic detection of coordinate systems and manual specification when metadata is incomplete or incorrect.

- **Levels of Detail (LOD):** LOD systems are essential for rendering large-scale geospatial datasets efficiently. As the viewer's distance from an object increases, the visual difference between high and low detail becomes negligible, while the computational cost of rendering high detail remains significant. Cesium implements hierarchical LOD systems where data is organized into multiple resolution levels, automatically selecting appropriate detail based on viewing distance, screen space error (the projected pixel error of using a lower LOD), and performance targets. This enables rendering of datasets spanning multiple orders of magnitude in scale—from individual building details to continental terrain—while maintaining consistent frame rates. The LOD system operates transparently, streaming appropriate detail levels as the viewer navigates through the virtual environment.

- **3D Tiles specification:** 3D Tiles is an open specification developed by Cesium for streaming and rendering massive heterogeneous 3D geospatial datasets. Unlike traditional 3D formats designed for single objects, 3D Tiles organizes content into a spatial hierarchy (typically an octree or k-d tree) where each node represents a spatial region and contains either geometry data or references to child nodes. This hierarchical structure enables efficient culling (not loading data outside the view frustum), LOD selection, and progressive loading. The specification supports multiple content types: *Batched 3D Models* for large numbers of similar objects (e.g., buildings), *Instanced 3D Models* for repeated geometry with transformations, *Point Clouds* for LiDAR data, and *Vector Data* for polylines and polygons. Each tile can reference external resources (geometry, textures) or embed data directly, providing flexibility for different use cases and optimization strategies.

1.2.3 Basic Workflow with Cesium for Unreal

1. **Installation of the Cesium plugin in Unreal Engine**
   Download from the Marketplace or the official Cesium website.

2. **Addition of the Cesium Globe**
   Insert the "Cesium World Terrain" object into the Unreal level. This loads a terrestrial globe based on real elevation data.

3. **Loading additional data:**
    - Detailed terrains.
    - Urban models (buildings, infrastructure).
    - Custom data from Cesium Ion (cloud platform for own models).
4. **Positioning and synchronization**
   The camera and actors can be positioned in real geographic coordinates (latitude, longitude, altitude).
5. **Preview and adjustment of detail levels**
   Data is displayed according to scale and distance, facilitating the handling of large scenarios.

1.2.4 Organization and Management of Large-Scale Data

- **Progressive streaming:** Cesium only loads necessary data according to what is seen and camera distance, reducing resource usage.
- **Tiling (division into tiles):** Geospatial data is organized into "tiles" (small pieces) to facilitate efficient loading and rendering.
- **Automatic updates:** If sources in Cesium Ion are updated, they are reflected in the Unreal project.

Glossary

- **3D Tiles:** Open format for transmitting large-scale 3D geospatial data.
- **Streaming:** Progressive and dynamic sending of data according to visualization needs.
- **Cesium Ion:** Online platform to host and process own 3D data or use Cesium's data.
- **WGS84:** Global reference system for geographic coordinates.
- **LOD:** Levels of Detail, key for optimizing performance in large scenes.

1.3 Photogrammetry with Drones and Cesium Ion

1.3.1 Fundamentals of Photogrammetry

Photogrammetry represents a measurement science that extracts three-dimensional information from two-dimensional images through the principles of triangulation and bundle adjustment. The technique exploits the geometric relationships between multiple overlapping photographs of the same subject, captured from different viewpoints, to reconstruct spatial geometry with high accuracy. Modern computational photogrammetry, particularly when combined with unmanned aerial vehicle (UAV) platforms, has revolutionized the acquisition of geospatial data, enabling rapid, cost-effective generation of detailed 3D models at scales ranging from individual objects to entire landscapes.

The photogrammetric process involves several distinct stages, each requiring careful execution to achieve optimal results:

- **Image capture and planning:** Successful photogrammetry requires systematic image acquisition following specific overlap requirements. Typically, forward overlap (between consecutive images along a flight path) should exceed 60-80%, while side overlap (between parallel flight lines) should be 40-60%. This overlap ensures sufficient common features between images for reliable matching. Flight planning software calculates optimal flight paths, altitude, and camera settings based on desired ground sample distance (GSD—the physical size represented by each pixel), terrain characteristics, and camera specifications. Modern drones incorporate GPS receivers and inertial measurement units (IMUs) that record precise camera positions and orientations (exterior orientation parameters) for each image, significantly improving reconstruction accuracy and georeferencing precision.

- **Image processing and feature detection:** The reconstruction process begins with feature detection algorithms (such as SIFT, SURF, or more modern deep learning-based detectors) that identify distinctive points in each image. These features are then matched across image pairs, establishing correspondences between images. The matching process uses descriptors that encode local image appearance around each feature point, enabling robust matching despite variations in lighting, viewpoint, and scale. Advanced algorithms filter incorrect matches using geometric constraints (epipolar geometry) and statistical outlier detection.

- **3D reconstruction:** The matched features enable calculation of camera poses (positions and orientations) through bundle adjustment—a nonlinear optimization process that simultaneously refines camera parameters and 3D point positions to minimize reprojection errors. This process generates a sparse point cloud representing the 3D positions of matched features. Dense reconstruction algorithms (multi-view stereo) then generate dense point clouds by searching for correspondences across all images for every pixel, resulting in millions or billions of 3D points. Surface reconstruction algorithms convert these point clouds into polygonal meshes, while texture mapping projects original photographs onto the mesh surfaces, creating photorealistic 3D models that preserve the visual appearance of the captured subject.

- **Georeferencing and coordinate systems:** Georeferencing establishes the relationship between the reconstructed 3D model and real-world geographic coordinates. This process uses ground control points (GCPs)—surveyed points with known coordinates visible in the images—or direct geotagging from onboard GPS receivers. The georeferencing transformation accounts for coordinate system differences, datum transformations, and potential systematic errors. Accurate georeferencing is essential for integrating photogrammetry models with other geospatial data and for measurements within the virtual environment. Modern workflows often combine GPS-tagged images with GCPs to achieve centimeter-level accuracy, particularly important for engineering and surveying applications.

Advantages and Applications: Photogrammetry offers several advantages over alternative 3D data acquisition methods: it captures both geometry and appearance simultaneously, provides high spatial resolution (sub-centimeter GSD achievable with appropriate flight parameters), covers large areas efficiently, and produces visually realistic results. Applications span diverse domains: *heritage documentation* preserves cultural sites through detailed 3D records, *topographic mapping* generates digital elevation models and orthomosaics for cartography, *agricultural monitoring* assesses crop health and growth through temporal analysis, *engineering surveys* provide as-built documentation and volume calculations, *urban planning* visualizes proposed developments in existing contexts, and *disaster assessment* rapidly documents damage for response planning. The integration of photogrammetry with real-time rendering engines like Unreal enables these detailed reconstructions to serve as the foundation for immersive, interactive virtual environments.

1.3.2 Processing and Formats of 3D Models

- **Point cloud:** Initial representation, with millions of points in 3D space.
- **Textured mesh:** Conversion of the cloud into a polygonal surface, to which a texture is applied to simulate the real appearance.
- **Common formats:**
  - OBJ, FBX, LAS: used in editing and modeling.
  - glTF: open format for 3D models optimized for web and graphics engines.
  - 3D Tiles: Preferred format for Cesium, adapts models for dynamic visualization and large-scale streaming.

1.3.3 Cesium Ion: Loading and Optimization Pipeline

Cesium Ion is Cesium's cloud platform that facilitates the transformation and hosting of 3D models to quickly integrate them into Unreal.

Main Steps:

1. **Model upload:** Export your photogrammetry from reconstruction software (such as Agisoft Metashape, RealityCapture, or Pix4D) in a compatible format (OBJ, FBX, LAS, glTF).
2. **Automatic conversion:** Cesium Ion converts the model to 3D Tiles, segmenting it into "tiles" to optimize streaming and interactive visualization.
3. **Hosting:** The model is saved in the cloud, ready to be used from any project.
4. **Integration in Unreal Engine:** Configure your Cesium Ion account in the Cesium for Unreal plugin and add the resource as a new "Cesium 3D Tileset" in the level.
5. **Visualization and adjustment:** Position and scale the model, verify its alignment, and adjust detail levels according to use.

1.3.4 Limitations and Best Practices

- **Model weight:** Very dense models can hinder streaming and smooth visualization. It is recommended to simplify the mesh and optimize textures before exporting.
- **Level of detail:** Adjust the balance between visual quality and performance, especially for large scenarios.
- **Georeferencing errors:** Ensure that coordinate information is correctly embedded or assigned before uploading to Cesium Ion.
- **Division into multiple tilesets:** For extensive areas or multiple independent elements, it is preferable to segment your model into several files.

1.3.5 Overview of Photogrammetry and LiDAR Software for Drones

There are various tools, both commercial and open source, to convert aerial images and LiDAR data captured by drones into point clouds and georeferenced 3D models. Each solution differs in ease of use, integration with automatic workflows, format compatibility, and precision.

Main Commercial Options:

- **Agisoft Metashape:** Very popular, intuitive interface, supports very detailed models, exports to most standard formats (OBJ, LAZ, LAS, glTF).
- **Pix4D Mapper:** Specialized in topography and civil engineering applications; integrated automations for agricultural mapping, inspection, thermal analysis, and volumes.
- **RealityCapture:** Stands out for its processing speed and handling of large projects, integrates images and laser data.
- **DroneDeploy:** All-in-one cloud platform with automated processing and results ready for analysis or GIS/3D integration.
- **Autodesk ReCap:** Although focused on static scanners, supports drone LiDAR and point clouds for architectural modeling and BIM.

Open Source and Academic Options:

- **OpenDroneMap (WebODM):** Free platform, recommended for advanced users, granular control over processing, exports to formats compatible with Cesium Ion (OBJ, LAS, etc.).
- **MicMac:** Project from the French IGN, very powerful but requires technical experience; oriented to researchers.
- **COLMAP:** More generalist, excellent for small projects or research, processes images not only from drones.

Specific Workflow for LiDAR:

- **CloudCompare:** Does not generate models from images, but is the de facto standard in editing and analysis of 3D LiDAR point clouds.
- **LASTools:** Set of utilities for conversion, cleaning, and classification of LAS/LAZ data.

1.3.6 OpenDroneMap and WebODM

OpenDroneMap (ODM) is a set of open-source tools that allows reconstructing 3D models and orthomosaics from images taken with drones (and also ground cameras). It is oriented to flexible workflows, allows fine-tuning of parameters, and prioritizes transparency and reproducibility in geospatial data science.

WebODM is the web interface of ODM, facilitating project management, visualization, and export. It stands out for being installable locally or in the cloud, controlling processing and privacy of your data.

Main Advantages of WebODM:

- Allows uploading massive sets of photographs and GNSS/LiDAR data.
- Supports batch processing and advanced configuration.
- Generates orthophotos, Digital Surface Model (DSM/DTM), point clouds, textured 3D meshes, and vegetation maps.
- Exports to widely compatible formats (OBJ, LAS, LAZ, GeoTIFF, etc.).
- Plugins such as cesium ion uploader simplify the workflow: automate direct upload of models to Cesium Ion for visualization and use in Unreal.

Integration with Cesium Ion:

1. Process your project and export in OBJ, LAS, or compatible format.
2. Use the Cesium Ion plugin/webhook, or perform manual upload from your account panel at ion.cesium.com.
3. Cesium Ion will convert and host the model as 3D Tiles, ready for integration in Unreal Engine.

1.3.7 Basic Workflow with OpenDroneMap and WebODM

If you work with Windows or Mac, the simplest way is to use the graphical GUI installer, which has a one-time cost of $64 or $99 respectively. Another way is to do it through Docker, following the steps indicated on GitHub: `https://github.com/OpenDroneMap/WebODM/?tab=readme-ov-file#manual-installation-docker`

If you work with Ubuntu, the installation procedure is simpler: `https://docs.opendronemap.org/installation/#linux`

Once installed: `./webodm.sh start`, and you can access the GUI through the browser by entering the indicated route (e.g., `http://localhost:8000`).

1. Prepare the dataset: Make sure to have enough photos with sufficient overlap (recommended minimum: 70% front, 60% lateral). Ideally, images should contain EXIF metadata (GPS, height, yaw, pitch, roll). If no coordinates are provided, you'll need a compatible Ground Control Points (GCP) file.
2. Upload the image dataset. Access the WebODM GUI. Click "New project" and provide a name for it. Inside the project, click "Add Task". Upload the images dataset and the GCP file (if available).
3. Click on "Start Task". WebODM will exec the ODM pipeline. This may take a while.
4. Once finished, you can review and download the results as: orthoimage, point cloud, DEM/DSM (GeoTIFF), quality reports. The resulting orthophoto can be downloaded and further analyzed with your preferred GIS software.

Useful Resources:

- `https://github.com/OpenDroneMap/WebODM`
- `https://www.opendronemap.org`
- `https://cesium.com/learn/ion/integrations-learn/integrations-webodm/`

Glossary

- **Photogrammetry:** Technique for obtaining 3D models from photographs.
- **Point cloud:** Spatial set of 3D points extracted from images.
- **Mesh:** Surface structure made of polygons.
- **Tileset:** Organized set of "tiles" (small pieces of the model) used for efficient visualization.
- **Georeferencing:** Assign a real position in geographic space to a model.
- **Streaming:** Dynamic and segmented download of data according to visualization needs.

1.4 Real-Time Data Integration with Blueprints and Dynamic Models

1.4.1 Concept of Real-Time Data in Virtual Environments

Real-time data integration represents a fundamental capability for creating dynamic, responsive virtual environments that accurately reflect the current state of real-world systems. Unlike static visualizations that represent a fixed moment in time, real-time data integration enables virtual environments to function as living digital twins—continuously updated representations that maintain synchronization with their physical counterparts. This capability transforms virtual environments from passive visualizations into active monitoring, analysis, and decision-support tools.

Real-time data encompasses information that is captured, processed, and updated with minimal latency, typically ranging from milliseconds to seconds depending on the application requirements. The integration of such data into virtual environments presents unique challenges: data must be acquired from diverse sources (APIs, databases, sensor networks, streaming services), parsed and validated, transformed into appropriate formats, and then mapped to visual representations within the 3D environment. The system must handle varying data rates, potential network interruptions, data quality issues, and the need to maintain performance while processing continuous data streams.

Examples and Use Cases:

- **Transportation systems:** Real-time vehicle tracking (buses, trains, aircraft) enables visualization of current traffic patterns, arrival predictions, and system status. Integration with traffic management APIs provides current congestion levels, incident reports, and route optimization data.
- **Environmental monitoring:** Meteorological data streams provide current weather conditions, forecasts, and historical trends. Air quality sensors, water level monitors, and environmental sensors enable real-time visualization of environmental conditions across geographic regions.
- **Smart city infrastructure:** IoT sensor networks monitor energy consumption, waste management, parking availability, and utility usage. Real-time visualization of this data enables urban planners and citizens to understand current city conditions and identify optimization opportunities.
- **Emergency response:** Integration with emergency services data provides real-time visualization of incidents, resource deployment, and response status. This enables command centers to coordinate responses more effectively and provides situational awareness for training scenarios.

- **Social and economic data:** Population density, economic indicators, and social media feeds can be visualized in real-time, providing insights into urban dynamics and human behavior patterns.

Advantages: Real-time data integration significantly enhances the value of virtual environments by ensuring that visualizations remain current and relevant. This temporal accuracy is essential for applications where decisions must be based on current conditions rather than historical snapshots. The integration creates more immersive experiences by introducing dynamic elements that respond to real-world changes, enhancing user engagement and the perceived realism of virtual environments. For training applications, real-time data enables simulation of current scenarios, making training more relevant and effective. For monitoring and analysis, real-time visualization facilitates rapid identification of patterns, anomalies, and trends that might not be apparent in static representations.

1.4.2 Blueprints: Visual Logic in Unreal for Dynamic Interaction

What are Blueprints? Blueprints constitute Unreal Engine's comprehensive visual scripting system, providing a node-based programming interface that enables creation of complex game logic, interactions, and system behaviors without traditional text-based coding. The system employs a graph-based paradigm where nodes represent functions, variables, flow control structures, and operations, connected by execution wires (defining execution order) and data wires (passing values between nodes). Blueprints compile to optimized bytecode executed by Unreal's virtual machine, providing performance comparable to C++ for most use cases while dramatically reducing development time and lowering the barrier to entry for non-programmers.

The Blueprint system supports multiple node types: *Event nodes* (BeginPlay, Tick, custom events) initiate execution flows, *Function nodes* encapsulate reusable logic, *Variable nodes* store and retrieve data, *Flow control nodes* (branches, loops, sequences) manage execution paths, and *Operation nodes* perform calculations and transformations. The visual interface provides immediate feedback, type checking, and debugging capabilities, making it an effective tool for both rapid prototyping and production development.

Basic Structure and Workflow:

- **Event-driven architecture:** Blueprints operate on an event-driven model where execution begins at specific event nodes. *BeginPlay* fires when an actor enters the level, *Tick* executes every frame (typically 60+ times per second), and custom events can be triggered by other Blueprints, timers, or external inputs. This event-driven approach naturally aligns with real-time data integration workflows, where data updates trigger visual changes.

- **Node graphs and execution flow:** Blueprint graphs consist of nodes connected by execution wires (white lines) that define the order of operations, and data wires (colored lines) that pass values between nodes. The visual representation makes complex logic flows immediately comprehensible, facilitating collaboration between programmers and designers. Execution flows can branch conditionally, loop iteratively, or execute in parallel through sequence nodes.

- **Data types and structures:** Blueprints support comprehensive data types including primitives (integers, floats, booleans, strings), vectors and rotators for 3D mathematics, object references, arrays, maps (key-value pairs), and custom structures. This type system enables handling of complex data structures commonly encountered in real-time data integration, such as JSON objects, coordinate systems, and sensor readings.

Integration with Actors and Real-Time Data: Any actor in the scene can have an associated Blueprint class that defines its behavior, properties, and responses to events. For real-time data integration, Blueprints serve as the bridge between external data sources and visual representations. A typical workflow involves: (1) Blueprint nodes request data from external APIs or services, (2) received data is parsed and validated, (3) data values are mapped to actor properties (position, scale, color, visibility), (4) actors are created, updated, or destroyed based on data state. This pattern enables dynamic visualization systems where the virtual environment automatically reflects changes in the underlying data sources, creating responsive, living visualizations that maintain synchronization with real-world systems.

1.4.3 Architecture for Real-Time External Data Integration

1. **Obtain external data**
   - Sources: REST APIs (HTTP), WebSockets for streams, local files, databases, or direct sensors.
2. **Data reading in Unreal**
   - Official or third-party plugins, native nodes ("HTTP Request", "JSON Parse"), or custom C++ code.
3. **Processing and mapping**
   - Convert and associate received data to Blueprint variables.
   - Example: update the position of virtual vehicles according to coordinates received from an API.
4. **Dynamic visualization**
   - Change properties of actors in the scene: position, scale, color, visibility, etc.
   - Generate new objects or update existing ones.

1.4.4 Practical Example: Visualization of a Data Flow

Case: Real-time vehicle visualization using data from an API.

- Make a periodic HTTP request to obtain vehicle locations (GeoJSON or CSV type).
- Parse the JSON and extract relevant information (ID, latitude, longitude, state).
- Convert coordinates to the Unreal system (normally transforming lat/long to X/Y/Z on the Cesium globe).
- Create or move actors that represent vehicles, showing their state in real time.
- Use Blueprints to animate the update and respond to events (appearance, disappearance, color changes according to state).

Glossary

- **Blueprint:** Visual programming system of Unreal Engine (logic and events).
- **API:** Interface that allows programmatic access to external data.
- **WebSocket:** Protocol for receiving real-time and continuous data.
- **JSON:** Common format for structured data transmission (JavaScript Object Notation).
- **Tick:** Event in Unreal that occurs in each frame, useful for periodic checks or updates.

Illustrative Example Suppose you want to monitor urban bus traffic in real time. You use a public API from the city council, receive bus positions and states every 10 seconds, and through Blueprints, move and update small 3D models representing each bus on the real map in Cesium, showing its route, number, state, and color.

1.5 Immersive Worlds and Virtual Reality (VR) with Unreal Engine

1.5.1 Fundamentals of Virtual Reality

Virtual Reality (VR) is a technology that allows the user to completely immerse themselves in a three-dimensional digital environment, perceived as real thanks to devices such as VR headsets, gloves, and motion controllers.

Differences with Other Technologies:

- **Augmented Reality (AR):** Superimposes digital elements on the physical world.
- **Mixed Reality (MR):** Combines elements of VR and AR interacting with the physical and virtual environment.

Key Advantage of VR: Offers a total immersive experience, ideal for complex simulations, architectural tours, training, and interactive geospatial analysis.

1.5.2 Preparation of Environments for VR in Unreal

- **Compatible devices:** HTC Vive, Oculus Quest, Valve Index, Windows Mixed Reality, among others.
- **Performance optimization:**
  - Minimize graphic load: reduce the number of polygons and excessive textures.
  - Adjust the "Frame Rate" (minimum recommended: 72-90 fps in VR).
- **Project configuration in Unreal:**
  - Use VR templates to start quickly.
  - Activate "VR Preview" to test the environment directly.
  - Associate controller inputs to actions in Blueprints.
- **Scaling and navigation:**
  - Ensure realistic scale of the environment relative to the user.
  - Implement intuitive navigation methods (teleportation, "walk", physical interaction).

1.5.3 Interaction Design and User Experience

- **Visual and haptic feedback:**
  Highlight interactive objects, use vibration/controllers for feedback.
- **VR-adapted interfaces:**
  Floating menus, information panels, and HUDs integrated into three-dimensional space.
- **Ergonomics and accessibility:**
  Consider visual fatigue, seated/standing use, and control adjustment.
- **Safety:**
  Implement play boundaries ("guardian boundaries") and signals to avoid collisions in real space.

1.5.4 Geospatial Digital Worlds and Immersion

The integration of Cesium with Unreal allows creating georeferenced virtual worlds navigable in VR:

- Realistic location of terrains and real buildings.
- 1:1 scale visualization of cities, rural areas, infrastructure.
- Direct interaction with models and data ("select building", "measure distance", "change information layers").

Typical Applications: Virtual city tours, emergency training, machinery simulation on real maps, collaborative urban analysis, virtual tourism, and heritage preservation.

1.5.5 Challenges and Opportunities

- **Optimization:** Maintain high graphic realism without sacrificing performance (key in VR).
- **Collaborative and multi-user:** Possibility of sharing tours or analysis in real time with several users in different locations.
- **Leveraging real-time data:** Use of dynamic information (GPS position, traffic, sensors) to enrich the experience.

Glossary

- **VR (Virtual Reality):** Immersive digital environment accessible through specialized devices.
- **Tracking:** Tracking of user position and movement.
- **Frame Rate:** Images per second, crucial for smoothness and comfort in VR.
- **HUD:** Head-Up Display, information presented on screen within the environment.
- **Guardian boundaries:** Safety limits to prevent the user from leaving the safe real area.

2 Part 2: Practice

2.1 Unreal Engine 5 Installation

- Project tested with Unreal Engine 5.3-5.6.
- Minimum specifications:
  https://dev.epicgames.com/documentation/en-us/unreal-engine/hardware-and-software-specif
- Recommended: at least 50 GB of free disk space, preferably 100 GB.

Installation Steps:

1. Access https://www.unrealengine.com/es-ES/download
2. Download the Epic Games installer and install
3. From Epic Games Launcher, in the side menu select Engine, and download the latest stable version of Unreal Engine 5.
4. Once installed, click start.

2.2 Recommended Plugins

The following plugins are free and facilitate the integration of the digital twin in Unreal, allowing the import of data from external sources:

1. Cesium for Unreal
2. Blueprint WebSockets
3. VaRest Rest API with Blueprints

2.2.1 Installing the Cesium for Unreal Plugin

1. Open the Cesium for Unreal plugin page in the Unreal Engine Marketplace.
2. Sign in if necessary, click the Free button to install the plugin in your Unreal Engine account, then press Install to Engine to add it to your Unreal Engine installation.
3. Restart Unreal Engine
4. Within the Unreal project, in Editor > Plugins > Cesium for Unreal > Select the checkbox to activate it.
5. An error may appear after restarting the editor, add the dependency and accept.

2.3 First Project

1. Open Unreal Engine 5 and select a new empty project (blank project). Give it a name and choose a path to save the files.
2. When the project is loaded, activate the Cesium for Unreal plugin. Go to Edit → Plugins, search for "Cesium" and check the corresponding box. You may need to restart Unreal Engine after activating the plugin for the first time.

3. After restarting the editor with the plugin activated, an error related to the Water Body Collision profile may appear. Click the link Add entry to DefaultEngine.ini to fix it.

4. Create a new level (File → New Level) and, when prompted, select "Empty Level" to ensure there are no pre-existing objects.

5. Disable World Bounds Checks. This is in World Settings (Window → World Settings), under the World → Advanced category.

6. Open the Cesium panel: Window → Cesium.

7. Add lighting. Cesium for Unreal includes a ready-made system, CesiumSunSky, that simulates sun and atmosphere.

   (a) In the Cesium panel, Quick Add Basic Actors section, look for Cesium SunSky and click to add it to the level. You will see a sky gradient in the viewport window.
   (b) If after adding Cesium SunSky the view becomes completely white, it may be due to the realistic intensity of the light. To fix it: activate Auto Exposure and Extend default luminance range in project settings (Edit → Project Settings → Engine → Rendering → Default Settings).
   (c) Another option is to reduce the intensity of the sunlight: select the CesiumSunSky actor, and in its Directional Light component, reduce the Intensity to a more reasonable value (for example 10.0) if desired.

8. Save the level with Ctrl+S. You can also set this new level as the project's default map, in Edit → Project Settings → Maps & Modes → Default Maps

2.4 Connecting to Cesium Ion

1. From the Cesium panel, click Connect to Cesium ion.

2. A browser window will open. If you haven't signed in, do so.

3. It will ask for permission for Cesium for Unreal to access your resources in Cesium ion. Accept and return to Unreal Engine.

4. Now you will create a default access token for your project. Click Token in the Cesium panel.

5. In the token window: select Create a new token, you can rename it if you want, and then press Create New Project Default Token.

2.5 Creating a Globe

1. From the Cesium panel, add "Cesium World Terrain + Bing Maps Aerial imagery".

2. You will see the terrain appear in the level.

3. In the Outliner, along with CesiumSunSky, you will see several Cesium actors: Cesium World Terrain, CesiumCameraManager, CesiumCreditSystemBP and CesiumGeoreference.

4. If Cesium World Terrain is not selected, do so now to see more details in the Details panel.

2.6 Pamplona Photogrammetry

There are two alternatives for adding high-resolution photogrammetry of Pamplona and its surroundings to your Unreal Engine project:

1. **Use Google Photorealistic 3D Tiles (Recommended for immediate use):** Cesium for Unreal enables direct integration of Google's high-resolution photogrammetry via Cesium Ion. This provides detailed 3D city visualizations, including Pamplona, without needing any additional datasets.
2. **(Future) Load the Pamplona photogrammetry tile from Gobierno Abierto de Navarra (Open Government of Navarra):** This resource is not yet available, but will be accessible through their open data portal at `https://gobiernoabierto.navarra.es/es` once released.

To Load Google Photorealistic 3D Tiles in Cesium for Unreal:

1. Open the Cesium panel in your Unreal Engine project (`Window` → `Cesium`).
2. In the Cesium panel, find the section labeled **Quick Add Basic Actors**.
3. Click **Google Photorealistic 3D Tiles**. This will add the tileset to your level as a `Cesium3DTileset` actor, georeferenced to the entire globe.
4. To focus on Pamplona, use the Cesium Georeference actor: in the Outliner, select `CesiumGeoreference`. In the Details panel, set the latitude and longitude to center the level on Pamplona: `Latitude: 42.81687`, `Longitude: -1.64323`, `Height: 500`.
5. Google Photorealistic 3D Tiles will appear automatically (you may need to have logged into Cesium Ion and accepted the Google Photorealistic 3D Tiles terms and conditions; see the Cesium documentation).
6. To blend Google Photorealistic 3D Tiles with global terrain, you can add `Cesium World Terrain + Bing Maps Aerial`. Adjust its vertical position (e.g., set Z to `-1000.0`) if you want parts of both datasets visible or to avoid overlap.

Note: Google Photorealistic 3D Tiles is currently the best option for up-to-date detailed city models in Cesium for Unreal, especially since a public API or direct download link for the Pamplona dataset is not available yet.

To Load Pamplona Photogrammetry from Gobierno Abierto de Navarra (when available): The photogrammetry data in 3D Tiles format will be available through the Gobierno Abierto de Navarra open data portal. While this resource is currently being prepared and will be available soon, the following procedure describes how to access it once it becomes available:

1. Access the open data portal of Gobierno Abierto de Navarra at `https://gobiernoabierto.navarra.es/es` and navigate to the "Datos Abiertos" (Open Data) section.
2. Locate the Pamplona photogrammetry dataset in 3D Tiles format. The dataset will include access credentials or a Cesium Ion asset ID for integration.

3. In the Cesium side panel within Unreal Engine, select "Blank 3D Tiles Tileset". It will be added to the level scheme as a Cesium3DTileset within the CesiumGeoreference tree.
4. In the CesiumGeoreference > All options, paste the latitude, longitude, and reference height of Pamplona (`42.81687, -1.64323, 500`).
5. Select the Cesium3DTileset from the element tree again, and in the options navigate to Cesium. Enter the Ion Asset ID and Access Token provided by Gobierno Abierto de Navarra (these will be available in the dataset documentation once published).
6. The Pamplona photogrammetry provided by Gobierno Abierto de Navarra should load. This dataset represents the city of Pamplona with high-resolution 3D reconstruction, enabling detailed visualization of urban structures, buildings, and terrain.
7. If you want to have a long-distance perspective of the mountains surrounding the basin, you can add Cesium World Terrain + Bing Maps Aerial to the project at this point. To prevent the overlay from showing the Bing map above the Gobierno Abierto de Navarra photogrammetry, you can select it and in options > Transform, reduce the Z location to `-1000.0`, for example.

Note: The Gobierno Abierto de Navarra initiative promotes transparency, citizen participation, and open data access. The photogrammetry dataset will be made available as part of this commitment to open government practices, allowing researchers, educators, and developers to utilize high-quality geospatial data for visualization, analysis, and educational purposes. Please check the portal regularly for updates on dataset availability.

2.7 Photorealistic Visual Effects

The following actions will allow you to add greater realism to the scene:

- Add to the project, from the top bar with a box icon and a green plus > Visual Effects > Exponential height fog. By adjusting the parameters, you achieve that atmospheric blurring effect with distance.
- Add to the project, from the same button, volumetric clouds. Then, from the options, you can reduce the minimum and maximum height of the cloud layer to achieve a realistic effect.
- Finally, by selecting Cesium SunSky from the element tree, in the Date And Time options you can configure the time zone, date, and time of day to achieve appropriate lighting.

2.8 OpenSky Integration with Cesium for Unreal

2.8.1 Project Preparation

- Create a project in Unreal Engine 5.
- Install and configure Cesium for Unreal (from the Marketplace).
- Add to the level:
  - Cesium World Terrain.
  - CesiumGeoreference.

- Install VaRest Plugin (for HTTP requests and JSON parsing).
- Confirm that when running Play the globe is seen correctly.

Reference coordinates: Pamplona, Plaza del Castillo 42.81675737875049, -1.643017139525693

2.8.2 Blueprint Manager

- Create a Blueprint BP_OpenSkyManager (Actor type).
- This will be responsible for:
  - Calling the OpenSky REST API.
  - Parsing aircraft positions.
  - Spawning and updating actors that represent them.
- Drag an instance of BP_OpenSkyManager to the level (necessary for BeginPlay to execute).

2.8.3 Aircraft Management Variable

In BP_OpenSkyManager create a variable:

- Name: AircraftMap
- Type: Map
  - Key = String → will store the unique icao24 of the aircraft.
  - Value = Reference to BP_AirplaneMarker (the actor that represents it).

  This avoids duplicates: each aircraft is managed by its identifier.

2.8.4 PollOpenSky Event

In the Event Graph of BP_OpenSkyManager:

1. Custom Event → PollOpenSky.
2. Node Construct VaRest Request.
3. Set URL with the query to OpenSky (example: `https://opensky-network.org/api/states/all?...`).
4. Execute (GET) → connect to Bind Event to OnRequestComplete.
5. Create a Custom Event OnOpenSkyResponse connected to the callback.

Example coordinates:

- 42.14880566, -2.64935531
- 43.50958300, -0.63844790
- URL: `https://opensky-network.org/api/states/all?lamin=42.14880566&lomin=-2.64935531&lamax=43.50958300&lomax=-0.63844790`

2.8.5 JSON Parsing

Within OnOpenSkyResponse:

1. Get Response Object.
2. Get Array Field with "states".
3. ForEachLoop over that array.
4. Each Array Element → As Array.
5. Extract data with Get (index) + As String / As Number:
   - Index 0 = icao24.
   - Index 1 = callsign.
   - Index 5 = lon.
   - Index 6 = lat.
   - Index 13 (or 7) = alt.
   - Index 10 = heading.

2.8.6 Create/Update Aircraft

In each iteration of the loop:

- Check if AircraftMap.Contains(icao24)
  - True → Find in the map → call UpdateFromState on the existing actor.
  - False → SpawnActor BP_AirplaneMarker → Add to the map (Key = icao24, Value = actor) → call UpdateFromState.

  This way, new actors are only created if the aircraft did not exist.

2.8.7 UpdateFromState

With the inputs of lat, lon, alt, and icao:

1. Format Text to convert it to a text line → Print String
2. Make Vector (X = Lon, Y = Lat, Z = Alt) → Return value
3. Get Cesium Globe Anchor → Move to Longitude Latitude Height
4. Connect Return Value of the vector with Move to Longitude Latitude Height

We can apply multipliers to reduce the coordinate range and visualize everything in a smaller sector:

```
((Lon, Lat, Alt) + (1.64323, -42.81687, -550.0)) × 0.01 + (1.64323, 42.81687, 550.0)
```

2.8.8 Aircraft Blueprint

Create BP_AirplaneMarker:

- Components:
  - CesiumGlobeAnchor (for global positioning).
  - StaticMesh (representation of the aircraft, a cube works for testing).
- Variables:
  - LastSeen (Float).
- Custom Event UpdateFromState(lon, lat, alt, heading, callsign):
  - Set LastSeen = Get Game Time in Seconds.
  - Call CesiumGlobeAnchor → MoveToLongitudeLatitudeHeight(lon, lat, alt).
  - Adjust rotation with Make Rotator(0,0,heading).

2.8.9 Automatic Timer

In BeginPlay of BP_OpenSkyManager:

- Set Timer by Event:
  - Event = PollOpenSky.
  - Time = 10 seconds.
  - Looping = true.

  This way every 10s the API is queried and positions are updated.

2.8.10 Result

When running Play, BP_OpenSkyManager queries OpenSky every 10s.

- Aircraft appear as BP_AirplaneMarker on the globe.
- Their positions and rotations are updated.
- Aircraft that disappear are automatically deleted.

2.8.11 Cleaning Up Old Aircraft

After the ForEachLoop:

1. Get Keys from AircraftMap.
2. ForEachLoop → Find actor → check CurrentTime - LastSeen.
3. If greater than 30s (or another value):
   - Destroy Actor.
   - Remove from AircraftMap.

   This eliminates aircraft that are no longer in the response.

3 License and Usage Rights