Leveraging the Internet to drive a real car in the Virtual Earth 3D Model

Author, co-author (Do NOT enter this information. It will be pulled from participant tab in MyTechZone)

Affiliation (Do NOT enter this information. It will be pulled from participant tab in MyTechZone)

Abstract

Digital mapping tools have become indispensable for road navigation. Applications like Waze and Google Maps harness the power of satellite imagery to provide precise visualization of GPS coordinates. The field advanced significantly in May 2023 with the introduction of dynamic 3D representations of the Earth. Companies such as Cesium now offer Unity3D and Unreal Engine Application Programming Interface that can be applied to geospatial applications. These images are no longer static and offer the opportunity to provide seamless continuous navigation.

Driving simulation has been widely used for training and research. We investigate with this project the potential of this new geospatial database as a tool for scenario development to study manual and autonomous driving. We present an in-vehicle driving simulation integration that employs a real steering wheel and pedals from a stationary vehicle as controls. The visual experience is delivered through the Meta Quest Headset through an overlay in a Mixed Reality environment.

Two case scenarios are examined. The first case involves navigating downtown Denver. The use of photorealistic representations of Denver's buildings offers an immersive experience, although the 3D topology presents some irregularities. These irregularities result from the limited number of polygons used for the digital modeling, especially on flat surfaces like roads and pavements.

The second scenario leverages the hilly landscapes outside Denver. These areas, characterized by arid, treeless terrain typical of Colorado, offer a smooth driving experience. Still, the technology incorporates projection such as phantom cars, flat images of vehicles on the roadway that were captured during satellite data acquisition. We explore opportunities to address these inaccuracies and enhance the environment for a more realistic and immersive driving experience.

Introduction

In recent years, incredible advancements in computing technology, marked by enhanced processing capabilities and increased portability, have opened the door to a multitude of applications. Three key research areas have seen significant progress: ride-hailing, vehicle automation, and electrification. Furthermore, the widespread availability of the Internet and the rise of 5G technologies have facilitated seamless ride-hailing via platforms like Uber and Lyft, while also providing real-time weather and traffic updates through applications such as Waze.

Despite the rapid progress in various technological domains, the realm of driving education has seen limited evolution. Efforts to introduce driving simulation as an educational tool have been met with skepticism from driving instructors. Conventional tabletop driving simulators equipped with game-oriented hardware such as steering wheels have seen limited adoption in driving schools due to their inherent lack of immersion [1]. However, the recent advancements in MR offer a paradigm shift in this field. Achieving increasing levels of immersion has become more cost-effective than ever before. In the following section, we elaborate on a novel approach that involves the use of an actual vehicle to provide a comprehensive driving simulator experience. This innovative platform employs a series of non-invasive sensors attached to the vehicle's controls, including the steering wheel, gas pedal, and brake pedal, to ensure a robust mechanical immersion. The simulation is delivered through a Mixed Reality Headset, affording drivers a realistic 360-degree view of both their vehicle and the surrounding driving environment.

The paper's first section outlines the simulation hardware, while the subsequent sections delve into the software integration of the Cesium 3D geospatial platform. Given a stable internet connection, this software facilitates the incorporation of GPS coordinates, enabling users to navigate within that specific environment.

Driving Simulator Design

Real vehicle as framework

Our driving simulator leveraged an actual stationary vehicle to increase immersion. Previous work on driving simulator development was published in [2], [3], and [4]. The guiding principle was to provide an intuitive experience to users by using an actual car. Two vehicles were used to deliver the driving experience: a 2011 Toyota Prius and a 2018 Tesla 3. None of the vehicles were running during the simulation. The front vehicle tires sat on rotating platforms to free the steering wheel.

Non-invasive sensors, based on Inertial Measurement Unit (IMU) technology were strapped to the vehicle steering wheel, gas, and brake pedals. A calibration procedure allowed us to position the zero for car controls (centered steering wheel, relaxed position for both pedals) and range (steering wheel from -450 degrees to +450 degrees) and pedal deflection range. A dedicated companion app allowed for this calibration (figure 2).

Mixed reality provides the appropriate blending of real and virtual worlds to deliver scenarios inside of this car framework. Virtual windows are overlaid on top of the car windshields and windows.

Non-invasive IMU sensors

Three identical sensors were built to acquire steering wheel angle, gas, and brake pedal information. These sensors, which consist of an IMU board, a small processor, and a rechargeable battery, were encased in a small enclosure. The gas and brake sensors were strapped under the car's pedals. The steering wheel sensor was affixed to the center of the steering wheel. An external dedicated router was used to create a dedicated wifi network for the sensors. Each of the three sensors was assigned a dedicated IP address. The firmware establishes an automatic connection through the TCP protocol server and sends data read by IMU modules at a high frequency (60Hz).

The Mixed Reality Headset

The simulation was developed using the Unity3D engine and relies on the Meta Quest suite. Specifically, Meta's Quest 2 and Quest Pro were used. While the Quest 2 provides a more reliable user experience, its grayscale passthrough is limiting. The Quest Pro features a color passthrough that is more suitable to the Mixed Reality experience sought by the research team. During the simulation, the test users enjoyed a fully immersive view of the driving environment through the car's windshield and windows. Simultaneously, they were able to see their extended hands with an outline that reacts to interactive objects, their body for corporal presence, and the interior details of the vehicle. The sensory experience is realistic: users sit in an actual car seat, wear a seatbelt, and use real car controls. It is visually immersive in the sense that they can see the inside of the vehicle, their hands, and body, while enjoying driving virtual scenarios.

The headset was connected to the wifi network, allowing for data streaming through the internet. The headset was tetherless and did not require any wired connection to a computer. The resolution of the Quest Pro headset is 1800 x 1920 pixels per eye with an average refresh rate of 60Hz.

Implementing virtual objects

The Unity3D engine provided a large number of commercial, free, and open source plug-ins and tools to facilitate the creation of environments and physical controllers for the simulation experience. A virtual car framework was used. All car mirrors were fully functional and behaved as actual physical mirrors to the user wearing the MR Headset.

To deliver enough flexibility to the virtual controls, additional virtual car components were added. A virtual gear allowed the user to intuitively shift from Park to Neutral to Drive to Reverse (figure 1). A virtual turn signal was also implemented on the left side of the steering wheel (figure 3). These virtual objects facilitated intuitive interactions with the driving simulation. The Quest headset's hand tracking ability delivered an intuitive interaction with the virtual objects. The user could simply *grab* the virtual gear to go from Park to Neutral, Reverse or Drive.

To demonstrate virtual objects versatility, an additional virtual red button was incorporated (figure 2). This virtual button, which can also be intuitively pressed through a simple hand gesture, allowed for a reset of the driving scene in case of any undesired state such as an accident.

Figure 1. Mixed Reality Driving. Sensors affixed to the steering wheel and pedals of the vehicle allow the driver to use his real vehicle (on the left) to drive the Google Earth Environment (right).

Software Development

Mixed Reality with Unity 3D

Unity 3D is a widely recognized tool for developing Augmented Reality (AR), Mixed Reality (MR), and Virtual Reality (VR) applications. Its popularity can be attributed to its supportive user community and a diverse range of native add-ons.

This ecosystem of add-ons offers versatility and expedites the creation of Extended Reality (XR) prototypes. These extensions enable the integration of additional processes, such as managing virtual vehicle behavior and conducting complex physics simulations.

Unity 3D is compatible with the Quest platform, which runs on an adapted Android operating system. This compatibility streamlines the process of building and testing XR applications on the Quest headset. Developers can use Unity's tools in conjunction with the Quest's user-friendly Integration SDK to efficiently create and assess immersive experiences. Unity serves as a valuable platform for both XR development and deployment on the Quest, making it a noteworthy tool in the field of AR, MR, and VR application development.

Meta's Oculus Integration SDK for Unity provides a crucial interface for effortless access to Quest device-specific features and a wide array of services offered by Meta. These include Hand Tracking, advanced VR rendering techniques, seamless interaction with virtual objects, spatial sound enhancements, and a cloud-based Text-to-Speech (TTS) service, among others. Notably, Meta's SDK is built upon the robust OpenXR backend API, which not only ensures seamless Quest compatibility but also extends support to various other headset devices with minimal codebase updates.

Network topology and Internet requirements

In the context of network topology and internet requirements for the integration of proprietary sensors into our XR simulation, a specialized intermediary system becomes essential. This consists of a laptop equipped with Wi-Fi and Ethernet network adapters, enabling dual connectivity to fulfill the demands of the Internet Access Point (typically a regular router) and the customized router specifically designed for our sensors. Fortunately, such network configurations are readily supported by most modern laptops.

The companion application, running on this intermediary laptop, serves to manage connections between Metadrive XR and the IMU sensors. Meanwhile, the VR headset is linked to the Internet Access Point, ensuring that internet connectivity needs are met. The network configuration is illustrated in Figure 2, providing a clear visual representation of the system. Our XR application hinges on three primary components: the sensors, securely affixed to the car's steering wheel, gas and brake pedals, the XR headset itself, and the companion application, which facilitates calibration and acts as a communication bridge between the various elements.

These components engage in seamless communication via two distinct local Wi-Fi networks, meticulously arranged in a specialized topology. This network topology comprises a generic Internet Access Point to cater to internet-related requirements, and a dedicated, customized router (as depicted in Figure 2) designed to fulfill the specific needs of the sensors. This dual-network approach ensures the smooth and reliable operation of our XR simulation system.

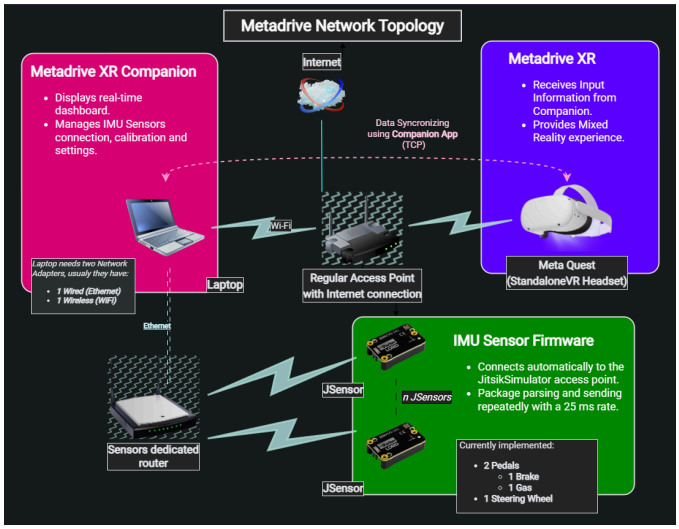

Figure 2.Metadrive network topology

The Cesium Geospatial libraries

In May 2023, Cesium partnered with the Google Maps Platform for an experimental release of Photorealistic 3D Tiles available through a platform. This service offers integration for multiple game engines and simulation platforms such as Unity3D and the Unreal game engines, combines photogrammetry, BIM, and/or other 3D data sources to offer a dynamic 3D representation that can be rendered efficiently for Virtual Reality applications [5], [6], [7]. Drone and flying applications are a natural fit for this type of top-level view of Cesium's 3D terrain representations, leading to spatially realistic flight simulator applications. This technology's suitability for driving simulation – a ground-level view – is still an unexploited topic.

Previous work

Previous efforts to integrate the Cesium workflow in driving applications can be traced to the VectorZero RoadRunner Unreal Engine Plugin developed by Gabby Getz [8], [9]. Given the great need for auto-manufacturers to test autonomous driving algorithms in a myriad of conditions, simulation has become a major tool for testing. The ability to use real terrain with various weather and lighting conditions can accelerate testing and facilitate the development of self-driving cars. The simulation workflow can provide input for cameras and lidars that can feed into the self-driving car's algorithm. Our project is a continuation of this work. It aims at assessing the quality of Digital Twins and the Virtual Driving scene for driving applications.

Software integration and API

Our research team leveraged the Cesium for Unity Application Programming Interface for the development of MetaDrive XR [10]. The general steps necessary for this software integration are detailed in this paper's appendix. The prerequisites for the software integration are:

- An installed version of Unity 2021.3.2f1 or later. The latest version of Unity 2021.3+ LTS is recommended.
- A Cesium ion account to stream terrain and building assets into Unity.

Memory and Processing Requirements

Cesium provides a high-performance service for 3D Tile streaming compatible with a range of devices. These include those built on the Unity Engine whether they run on Windows, macOS, Android, or VR platforms such as the Quest 2 and Quest Pro.

It is important to note that while Cesium SDK supports Quest 2, we must be mindful of performance considerations when integrating vehicle and physics frameworks. To ensure a seamless experience, it is crucial that the simulation maintains a consistent frame rate above 60 frames per second (FPS). This level of performance ensures that essential functions like hand tracking, networking, rendering, and other resource-intensive processes remain unhindered.

Scenario 1: Driving in Downtown Denver

Our initial Cesium geospatial integration targeted Downtown Denver.

Table 1.GPS coordinates for Downtown Denver

Latitude	Longitude	Height
39° 45 '11" N	105° 00' 04" W	1581 m

The software integration leveraged Cesium's API and the implementation was immediate. As a result, the research team could immediately drive in Downtown Denver.

City highlights, such as the Denver Train Station, can easily be recognized (see figure 3). The neighborhood, however, was however challenging for our driving simulation [12], We encountered two types of issues. The first issue, which is the less problematic one, is the rendering of buildings. In Downtown Denver, a large number of building facades were depicted at a low resolution. We attribute this to the difficulty of recording high resolution data on high rise building through satellite imagery. Smaller artifacts, such as trees, traffic lights, stop signs, or parked cars, while visible in the Unity model, present a poor rendering, although they are still recognizable. More importantly for our driving application, roads in between high rise buildings are bumpy and do not present a flat surface for driving. When driving, the car follows this uneven surface, resulting in the feeling of driving over a pothole. This feeling is purely visual, as the user does not experience any actual acceleration/deceleration. Going over this terrain with the Virtual Reality Headset also exacerbates the feeling of motion sickness which often accompanies the VR experience. This uneven terrain is a direct result of the low resolution for flat surfaces (road and pavements) from the satellite imagery in this specific neighborhood. In its present form, and in this specific high rise neighborhood, these roads are impractical for a driving application.

Figure 3.Driving in the Google Earth Model of Downtown Denver, Colorado.

Scenario 2: Driving on Colorado Roads

Given the limitation for driving applications posed by inner city neighborhoods, our research team turned its focus to a less challenging scenario. We chose to drive in Colorado.

Table 2. GPS coordinates for targeted Colorado roads.

Latitude	Longitude	Height
39° 44' 20"	-105° 15' 36"	1789 m

The desert terrain allows for better road resolution. This resulted in smooth driving on a road with far better defined markings [11]. This terrain did not include the presence of phantom cars, 'flat images of cars' that are present on the road but do not have any volume and do not exist as Unity entities. It just appears as if the road itself is imprinted with the image of a car–they can be driven over as if they do not exist. This is in contrast to the parked cars in the Denver model which have an actual collider and create an obstruction to driving. Our findings are therefore, that while 'driving out of the box' is technically possible in some neighborhoods, an editing phase will be needed to fully leverage the satellite images for driving. The Unity 3D environment is fully editable. Manual or automated techniques can be developed to make the environment more realistic and better suited for driving simulation.

Figure 4. Driving on Colorado roads.

Mixed Reality constraints

Our in-car set up leverages a Mixed Reality environment. We discuss here the hardware and software constraints and limitations associated with this framework.

Our preliminary results showed that users are comfortable using the car as a simulator. There is indeed no hardware learning curve, as it comes to the steering wheel and pedals.

Hardware constraints may relate to the specific car used. For our study, a Toyota Prius was used. The vehicle's tires were placed on rotating platforms which freed the steering wheel. This setup might

not be possible with other cars which lock the steering wheel when the car engine is off. For a wide implementation on other vehicles, we propose the use of a clip-on steering wheel, which will free the steering wheel. Our IMU sensors were attached under the pedals. This setup might not be possible with all cars. For some vehicles, the hydraulic system locks the pedals if the engine is off. An additional constraint is the need for a robust wifi network connection. This was discussed above in the Network Topology and Internet Requirements. It is also important that the IMU sensors not introduce latency. A lag of 50 ms or less is required so the driver can easily control the vehicle.

Other constraints relate to the Mixed Reality environment itself. Hand recognition gets degraded when lighting conditions in the vehicle are poor, e.g. by night, or when the headset is in direct sunlight. Our preliminary tests were developed in a garage with regular indoor light. When designing the software, special consideration must be given to the design of the User Interface. The Red Virtual Reset button (figure 1) must be positioned so that there are no physical barriers (such as the physical dashboard) to access it. For our next iteration, we plan to place menus and virtual buttons above the driver's head. Since different cars have different configurations, with the example of the central console, a wide implementation of the Mixed Reality framework requires the use of the commonality between vehicles, for example, the free space under the roof.

Limitations and future work

The proof of concept for driving in Downtown Denver and Colorado roads show that while some environments will be ready for driving 'as is', other environments will require considerable work before they can be used for a proper driving simulation. We anticipate tasks will include cleaning up the environment (phantom cars), smoothing the road (filtering), and adding signals and traffic. In addition, some models such as tunnels will need to be created from scratch as they are absent from satellite imagery.

An alternative approach involves generating 3D representations of semantic road data based on well-established specifications. Several tools, including OpenDRIVE (ASAM), RoadRunner (Mathworks), and Lanelet2 (Autoware), can be instrumental in accomplishing this task. While the transformation from semantic data to a 3D representation can be automated, it is advisable to seek a supervised output to mitigate potential issues similar to those encountered with Cesium. Relying on human-readable semantic descriptions, this method guarantees enhanced control and precision throughout the translation process.

Our immediate future work will involve Roosevelt Blvd in Philadelphia, which has been repeatedly listed by Farm Insurances as one of the most dangerous roadways in the country. Between 2016 and 2021, 77 people died in car crashes on Roosevelt Boulevard, accounting for 11% of the total traffic fatalities in Philadelphia. Several factors make it so deadly; The Roosevelt Boulevard consists of 12 lanes with speed limits of 40 mph and passes through densely populated neighborhoods. The road also has numerous intersections with local streets and connects to major high-speed highways and arterials throughout Philadelphia. Our next steps in Virtual Reality driving will consist of developing a robust model of the Roosevelt Blvd to test for its suitability for driving education.

Creating a digital model of the world is a colossal project. As cars continue to integrate cameras and lidar technology for self-driving and drones become even more ubiquitous, this Earth model will benefit from a myriad of new data sources that will incrementally enhance the 3D Earth Model. We anticipate the growth of the geographical Digital Twin ecosystem which will contribute to the safety and development of our roads.

Summary/Conclusions

Driving simulators can be traced back as far back as 1967 when the drivetrain was introduced [13]. They range from basic tabletop systems such as a steering wheel and set of pedals that are easily deployable to costly alternatives like the National Advanced Driver Simulator (NDAS) that provide high-level mechanical immersion. Such simulators have been widely used in research [13]. While hardware implementations differ widely between products, software implementation has, until now, always required the designing, from scratch, of driving scenarios through dedicated software such as Unity3D or Unreal Engine. In all cases, the driving scenario must be carefully planned and built to suit the specific application.

The advent of a Dynamic Model of the Earth presents phenomenal opportunities, especially relating to urban development and driving safety. While most driving simulator scenarios have used synthetic environments that are not grounded in actual geography, it is now possible to build and even drive virtually real scenarios that originate from Earth satellite images. Our project has ties with the growing field of Digital Twin development. Our research study demonstrated that some environments, like those void of high buildings or trees, are 'ready to drive' out of the box, while others like city environments need a substantial editing phase before they can be integrated into a driving scenario. Our preliminary 'out of the box' driving highlighted the need to develop metrics to assess the smoothness of virtual roads. These metrics will need to be tested in a large number of urban and rural settings to assess the system's versatility and reliability. A second editing phase will be needed to restore traffic lights and stop signs. Finally, a third editing phase will be needed to add traffic so the simulator can truly become a tool for testing manual or autonomous driving. As the Earth Virtual Models get more elaborate, we anticipate they can better be used to complement driving education, increasing the safety of our roads.

References

1. Michael, Despina, Marios Kleanthous, Marinos Savva, Smaragda Christodoulou, Maria Pampaka, and Andreas Gregoriades. "Impact of immersion and realism in driving simulator studies." *International Journal of Interdisciplinary Telecommunications and Networking (IJITN)* 6, no. 1 (2014): 10-25.

2. Qiao, Zhijie, Xiatao Sun, Helen Loeb, and Rahul Mangharam. "Drive Right: Shaping Public's Trust, Understanding, and Preference Towards Autonomous Vehicles Using a Virtual Reality Driving Simulator." In *2023 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1-8. IEEE, 2023.
3. Qiao, Zhijie, Helen Loeb, Venkata Gurrla, Matt Lebermann, Johannes Betz, and Rahul Mangharam. "Drive Right: Autonomous Vehicle Education through an Integrated Simulation Platform." *SAE International Journal of Connected and Automated Vehicles* 5, no. 12-05-04-0028 (2022).
4. Mangharam, Rahul, Helen Loeb, and Zhijie Qiao. "Training Drivers to Automated Vehicles." (2023).
5. Lee, Ahyun, Kang-Woo Lee, Kyong-Ho Kim, and Sung-Woong Shin. "A geospatial platform to manage large-scale individual mobility for an urban digital twin platform." *Remote Sensing* 14, no. 3 (2022): 723.
6. Jeddoub, Imane, Gilles-Antoine Nys, Rafika Hajji, and Roland Billen. "Digital Twins for cities: Analyzing the gap between concepts and current implementations with a specific focus on data integration." *International Journal of Applied Earth Observation and Geoinformation* 122 (2023): 103440.
7. Sermet, Yusuf, and Ibrahim Demir. "GeospatialVR: A web-based virtual reality framework for collaborative environmental simulations." *Computers & geosciences* 159 (2022): 105010.
8. https://www.youtube.com/watch?v=y6pLqsFXV3Y
9. https://cesium.com/blog/2018/09/17/autonomous-driving-bof-siggraph-2018-trip-report/
10. https://cesium.com/learn/unity/unity-quickstart/
11. https://vimeo.com/830296548
12. https://www.youtube.com/watch?v=YFGLZSy5ff4
13. Stack, H. J. (1966). History of Driver Education in theUnited States.
14. Marshall, D., Dow, B., & Brown, T. (2010). Validation of the National Advanced Driving Simulator for the Study of Young Driver Risk for the Center for Child Injury Prevention Science I/UCRC.

Contact Information

Helen Loeb, Ph.D., Corresponding Author
Jitsik LLC
276 Barwynne Rd, Wynnewood PA 19096
Tel: (610) 731-3960
helen.loeb@jitsik.com

Acknowledgments

This work was funded, in part, by a grant from Mobility21 University Transportation Center at the University of Pennsylvania and Carnegie Mellon University (Project 342) which provided support through the US Department of Transportation (USDOT). The Department specifically disclaims responsibility for any analyses, interpretations or conclusions. The content is solely the responsibility of the authors and does not necessarily represent the official views of the US DOT. We thank Rahul Mangharam from University of Pennsylvania, Mike Peretz, and for their efforts.

Definitions/Abbreviations

API	Application Programming Interface
IMU	Inertial Measurement Unit
VR	Virtual Reality
MR	Mixed Reality
BIM	Building Information Modeling

Appendix: Cesium for Unity

Cesium for Unity Quickstart

This is a quickstart guide to building a Cesium for Unity app with Cesium World Terrain and Cesium OSM Buildings.

A Cesium for Unity scene with Cesium World Terrain and Cesium OSM Buildings, set in Chicago.

Prerequisites

- An installed version of Unity 2021.3.2f1 or later. The latest version of Unity 2021.3+ LTS is recommended. For instructions on installing Unity, visit the Unity download page and refer to the Installing Unity guide.
- A Cesium ion account to stream terrain and building assets into Unity. Sign up for a free Cesium ion account if you don't already have one.

Cesium ion is an open platform for streaming and hosting 3D content, and includes global, curated data that you can use to create your own real-world applications.

1 Create a new project and import the Cesium For Unity package

1. Create a new Unity project. Unity recommends creating projects with the Unity Hub, which you can download here. This tutorial was written with Unity Hub 3.3.0.

Create a new project from the Projects tab by clicking the New project button.

A new window should open, allowing you to configure your project. This tutorial uses the 3D (URP) template, but the 3D (HDRP) template will also work. Give your project a name and choose its file location. Then, press Create project. Your new project will open momentarily.

Cesium for Unity works with both the Universal Render Pipeline (URP) and High Definition Render Pipeline (HDRP). However, it does not support Unity's built-in renderer. If you choose the empty 3D project as your template, the datasets loaded by Cesium will not render properly.

2. Once the project has fully loaded, open the Project Settings by going to Edit > Project Settings in the menu.

3. Click the Package Manager section on the left.

4. Add a new Scoped Registry with the following settings and click Save:

Name: Cesium

URL: https://unity.pkg.cesium.com

Scope(s): com.cesium.unity

5. Close the project settings and then open the Package Manager by going to Window > Package Manager in the menu.

6. In the Package Manager, click on the Packages drop-down and select My Registries.

7. Cesium for Unity will appear in the package list. Click on it, and then click Install. Cesium for Unity and its dependencies will be downloaded and installed.

2 Connect to Cesium ion

1. Open the Cesium window by selecting Cesium > Cesium from the menu.

2. Click the Connect to Cesium ion button.

3. A pop-up browser window will open. If you are not logged in, log in to your Cesium ion account. You can also sign in with your Epic Games, Github, or Google account.

4. Once you are logged in, you'll see a prompt asking you to allow Cesium for Unity to access your assets. Select Allow, then return to Unity to continue.

5. Now you'll create a default access token for your project. Every asset that you stream from Cesium ion requires an access token. In this tutorial, you'll set a project-wide access token that all your assets will use.

Click on the Token button at the top of the Cesium window.

6. A new window will appear to configure the token. Select the Create a new token option, and rename the token if you wish. Then, press the Create New Project Default Token button.

The new token you created will be added to your Cesium ion account. If you already have a token in your Cesium ion account that you would like to use, you can select it from the "Use an existing token" drop-down instead of creating a new one.

Tokens created by Cesium for Unity access only the assets that you allow. This follows security best practices for your Cesium ion account. Whenever you use the Cesium panel or Cesium ion Assets panel to add an asset to your scene, Cesium for Unity will automatically update the appropriate permission for the token.
You may choose to manually configure your token and add or remove assets using the Access Tokens page on Cesium ion.

3 Add a globe to your scene

Unity creates a new SampleScene whenever you create a new project. Feel free to rename your scene. Verify that it contains a Main Camera and a Directional Light in the Hierarchy window.

If either of these objects is missing, you can add it from the menu by clicking GameObject > Camera or GameObject > Light > Directional Light, respectively. You may have additional objects in your scene depending on which render pipeline you chose, which is fine.

With these basic objects in your scene, you are ready to add an asset from Cesium ion.

1. From the Cesium window, add "Cesium World Terrain + Bing Maps Aerial imagery" by clicking the button next to that entry.

Terrain will start to appear in the scene.

2. Take a look at the Hierarchy window. You should see two new game objects. One of them, Cesium World Terrain, is the tileset you just created. The parent of the tileset, CesiumGeoreference, is created automatically the first time you add a 3D Tileset to the scene.

3. If Cesium World Terrain is not already selected, select it now. In the Inspector window, you'll see more information about this game object.

This is a Cesium3DTileset component. When attached to a game object, it streams 3D Tiles data into Unity and provides ways to configure that tileset.

You'll learn about many of the available settings in future tutorials. For now, feel free to explore and try out the different settings yourself. Hover over any setting with your mouse to learn more about what it does.

Once you're ready, continue to the next section.

4 Configure the Main Camera

In your scene, you should have a game object named Main Camera with a Camera component attached. If you added a camera to the level yourself, it may be called Camera instead. This camera will capture your scene during Play Mode. You can preview this by clicking the Game tab.

You may notice that the terrain on the horizon is strangely cut off. This clipping can also happen in the Scene view while moving the Editor camera around, though it is usually less obvious.

This happens because cameras in Unity can only see a limited range in front of them. If you view your camera from the Scene view, you can see its range visualized as a frustum, like below.

The start and end of the camera's frustum are defined by the clipping planes. These represent the distances from the camera at which objects will show up. It is clear that the camera cannot see very far in the distance.

Cesium World Terrain is a full-scale globe, so Unity is rendering terrain that spans *hundreds* of thousands of miles, just from this camera view! However, Unity's cameras are not configured to see objects this far by default. Fortunately, you can manually adjust the clipping planes so the camera can properly view the scene in Play Mode.

1. Select your camera from the Hierarchy window, and go to the Camera settings in the Inspector. Find the values of the Clipping Planes.

The Near clipping plane is the minimum distance from the camera where objects start to appear. If this is a large value, objects close to the camera will not show up. On the other hand, the Far clipping plane is the maximum distance from the camera that objects will appear. Anything farther than the Far plane will not appear.

It is clear that the camera cannot see very far in the distance. However, this can be easily changed in the Camera component.

2. Set the Near value to 1 and the Far value to a large value, at least 100,000. This will significantly extend the sight of the camera and prevent the terrain from being clipped in-game.

In your own applications, these values can change depending on your use cases. For example, you can set the Near plane to a larger value if you plan to show data only from far away. Try not to let the Far plane get too far away from the Near plane, or you may run into rendering issues. See the Unity documentation for the Camera component for more details.

Now that your camera can see more of the scene, feel free to position the camera to get the best view of the terrain. You can use the Move and Rotate tools in the Scene view to move the camera to the perfect spot. You can also switch to these tools by pressing the W and E keys on your keyboard respectively.

The Unity Editor's camera works a bit differently than the in-game cameras. You can view its settings by clicking the camera icon from the Scene view.

The editor camera has an option for Dynamic Clipping, which automatically adjusts the Near and Far values of the camera depending on what is visible in your scene. When this option is checked, the Near and Far planes cannot be manually adjusted.

This setting is useful when working with Cesium for Unity, especially when zooming in and out on the globe. However, the automatic adjustments are not always precise, and clipping may still occur. If necessary, you can disable this option and set the Near and Far values yourself. You can also change the Camera Speed if you want to travel around the globe faster in the Editor.

5 Add global 3D buildings to your scene

Now that you've adjusted your camera to render more of the Unity scene, let's return to adding content to the world. Cesium for Unity can visualize more than just terrain, which we can demonstrate with a global dataset of city buildings.

1. Select the CesiumGeoreference game object in the Hierarchy window. This object determines where in the world your Unity scene is set. The scene's current latitude, longitude, and height can be changed with this game object.
2. In the Inspector window, look for the Latitude, Longitude, and Height variables under the Origin (Longitude Latitude Height) header.

These coordinates currently point to the hills outside Denver, Colorado, USA.

3. Change these variables to the coordinates of your favorite city. This tutorial uses the following coordinates for Chicago, Illinois, USA.

Latitude: 41.878101
Longitude: -87.59201
Height: 1000.0

After entering these coordinates, the scene will have shifted to the new location.

4. The city looks very flat because Cesium World Terrain doesn't include building details. Fortunately, the Cesium OSM Buildings dataset can be added to fill the empty space.

In the Cesium window, locate the Cesium OSM Buildings option underneath Quick Add Cesium ion Assets. Click to add the data to the scene.

The buildings should now appear on top of the terrain. You may be able to recognize some of the buildings that make up the Chicago skyline!

6 Explore your scene

Now that you have added global real-world content to your scene, let’s learn how to navigate it. You’ll learn multiple ways to move the Editor camera through your scene. You’ll also learn about the Dynamic Camera, a controller included in the Cesium for Unity package that will help you navigate the Earth's immense size during play mode.

The Editor camera can be controlled by using your keyboard and mouse. With the Scene View selected, use the arrow keys to move the camera forward, backward, left, or right. You can pan the camera by either selecting the hand icon in the Tools menu or holding the middle mouse button, then dragging across the Scene View window.

To look around with the camera, hold the right mouse button and drag the mouse across the Scene View. You can also rotate the camera by holding Alt and dragging with the left mouse button. Lastly, you can use the mouse’s scroll wheel, or hold Alt while dragging with the right mouse button, to zoom in and out of the scene.

If the Editor camera is moving too slowly, you can adjust the Camera Speed as described in the previous step.

For play mode, Unity provides a Character Controller component that you can attach to your camera to move it through the scene. However, you may need a controller that accounts for the scale of real-world data and can efficiently navigate the globe at any altitude. This is where Cesium for Unity’s Dynamic Camera comes in.

Cesium's DynamicCamera is a globe-aware camera controller that can adjust its orientation based on where it is on the globe. The DynamicCamera is able to dynamically adjust its clipping planes so the globe is not clipped as it zooms out. It also offers easier navigation of the globe by allowing users to adjust its movement speed with the mouse wheel, and providing the ability to fly between global locations along a curved path.

1. Using the Cesium panel, add a Dynamic Camera to your scene.

The DynamicCamera should appear under the existing CesiumGeoreference in the Hierarchy window.

2. Disable the Main Camera by selecting it in the Hierarchy window and unchecking the box next to its name in the Inspector. The DynamicCamera will become the new main camera of the scene. You can also remove the Main Camera object from the scene entirely.

3. Since the DynamicCamera is a georeferenced object, it will maintain its position relative to its coordinates on the globe, and not the standard Unity world coordinates. This means that if the CesiumGeoreference’s Origin is changed to a different location, the DynamicCamera will stay behind.

If you'd like to move it to your new location, select it in the Hierarchy window and find its Transform in the Inspector. Change its Position’s X, Y, and Z coordinates to 0 to move it to the Unity origin. You can also right-click the three dots on the Transform component and select Reset from the drop-down menu.

4. You're ready to test out your scene! Press the Play button at the top toolbar.

You can use the W, A, S, and D keys and the mouse to fly around. You can also use the Q and E keys to move the camera vertically with respect to the globe. Use the mouse scroll wheel if you need to change your speed.

You've created your first scene with Cesium for Unity. Feel free to explore and check out the world!