ChatGPT

Cesium for Unreal integration checklist (Birmingham, AL)

1 Plugin installation & project setup

1. **Install the Cesium for Unreal plug-in** – The **Quickstart** guide on Cesium's site shows how to install the plug-in from the Unreal Engine Marketplace.
2. Open the Cesium for Unreal plugin page in the Epic Games Marketplace.
3. Sign in with your Epic account if prompted, click **Free** and then **Install to Engine**.
4. After installation, launch Unreal Engine and create a new project (e.g., a **Blank** Game template) and uncheck **Starter Content** [1].
5. Once the project is open, enable the Cesium plug-in via **Edit → Plugins** and search for "Cesium". Tick the check box and restart Unreal if asked [2].
6. On the first run you may see a "Water Body Collision profile" error; click the **Add entry to DefaultEngine.ini** link to resolve it [3].
7. **Project preparation** – Create an empty level and disable world bounds checks so that Cesium actors can exist far from the origin [4].
8. In **World Settings**, uncheck **Enable World Bounds Checks** [4].
9. Add a **Cesium SunSky** actor through the Cesium panel to provide globe-aware lighting [5].
10. Adjust auto-exposure or reduce the sun intensity if the scene appears white [6].
11. **Add Cesium content** – Open the Cesium panel (**Window → Cesium**).
12. Choose **Connect to Cesium ion**. A browser window opens; sign in and allow Unreal to access your Cesium assets [7].
13. From the panel, generate an **Access Token** (per-project) and set it as the default token [8].
14. Click **Cesium World Terrain + Bing Maps Aerial Imagery** to stream high-resolution terrain and imagery into your level [9]. The level now contains Cesium actors including **Cesium World Terrain** and a **CesiumGeoreference** [10].
15. **Verify engine compatibility** – The Cesium plug-in only supports the last three UE versions; older releases are available via the `ue4-main` branch [11]. For Linux or offline use, download release ZIPs from GitHub and extract them to `Engine/Plugins/Marketplace` [12]. Cesium for Unreal is licensed under Apache 2.0 [13].
16. **Project folder structure** – To keep geospatial assets organized, create folders under your project's **Content** directory:

```
Content/
├── CesiumAssets/          # imported Cesium 3D Tilesets and terrain
├── Buildings/             # FBX or glTF exports of Birmingham buildings
├── OSMData/               # raw OSM building footprints or JSON
├── Blueprints/
│   ├── ImportTools/      # Editor Utility Blueprints or Python scripts
│   └── Weather/         # blueprint classes for dynamic weather
├── Materials/             # placeholder materials for buildings/labels
└── Scripts/               # Python scripts used for automation
```

2 Cesium ion account & data pipeline

- **Create or review your Cesium ion account** – Sign up at ion.cesium.com and sign in via your chosen provider.
- **Generate a project-specific access token** – On the Cesium panel, press **Token** → **Create New Project Default Token**, optionally rename it, and click **Create** [8].
- Cesium ion tokens define scopes; tokens created in Unreal only grant permissions to the assets you import, following the best-practice of least privilege [14].
- In ion's **Access Tokens** page you can manage tokens, specify allowed URLs or assets, and restrict scopes such as `assets:read` or `assets:list` [15].
- **Upload custom datasets** – If you convert your building models to 3D Tiles, you can upload them via the Cesium ion dashboard or REST API and stream them like any other tileset. However, for our Birmingham pipeline we import FBX meshes directly into Unreal (see section 4).

3 Terrain & imagery streaming

3.1 Automated download/setup

1. **Add the terrain and imagery dataset** – Use the Cesium panel to add **Cesium World Terrain + Bing Maps Aerial Imagery** [9]. The dataset streams on demand; the plug-in automatically downloads higher-resolution tiles as you zoom in.
2. **Set your level's location** – Select the **CesiumGeoreference** actor in the Outliner. In the Details panel, set **Origin Latitude**, **Origin Longitude** and **Origin Height** to the target coordinates (Birmingham, AL is 33.5186 °N, –86.8104 °W). Changing these values shifts the entire level [16].
3. **Stream OSM Buildings** – Cesium includes a global 3D building dataset. From the Cesium panel, add **Cesium OSM Buildings** to stream photogrammetry-style building meshes for major cities [17]. This provides quick building context if high-resolution custom models aren't available.

3.2 Scripted auto-navigation

Cesium for Unreal provides a **CesiumFlyTo** component that can move a pawn between geodetic coordinates. In the **Transition Between Locations** tutorial:

- Select your **CesiumGeoreference** and set the origin to the starting location [18].
- Add a **Dynamic Pawn** (from the Cesium panel) if one does not already exist [19]. The Dynamic Pawn automatically contains a `CesiumFlyTo` component.

- Open the **Level Blueprint**. Drag the Dynamic Pawn into the graph and call **Fly to Location Longitude Latitude Height (FlyTo)** from its `CesiumFlyTo` component [20].
- Connect a key press (e.g., F) to this function and create a **Make Vector** node for the destination (X = longitude, Y = latitude, Z = height). For example, to fly to Melbourne the tutorial uses `X=144.9631`, `Y=-37.8136`, `Z=2000` [21]. In our case you would use the Birmingham coordinates and a suitable height (e.g., 1000 m) to reposition the camera.
- Optionally adjust flight duration and curves (Height Percentage, Progress, Maximum Height by Distance) on the `CesiumFlyTo` component to control speed and arc [22].

You can trigger this flight automatically from Blueprints or call it via Python (Unreal Editor's Python API) on editor start.

3.3 Performance considerations

- **Bandwidth** – Streaming high-resolution terrain and imagery requires a steady internet connection. Bandwidth usage depends on view distance and movement speed; plan for tens to hundreds of megabytes per session when working at city scale.
- **Storage** – Cached tiles are stored locally under `Saved/CesiumData` in your project. Manage this folder when you work across multiple cities to avoid disk bloat.
- **Level of detail** – In the Cesium3DTileset (Cesium World Terrain) settings, adjust **Maximum Screen Space Error**, **Preload Ancestors**, and **Preload Siblings** to trade off between performance and visual quality.

4 Integrating OSM/Blender building data

4.1 Acquiring OSM building footprints

- The **Overpass API** can query OpenStreetMap for building footprints and metadata within a bounding box. Our `osm_extract.py` script uses Overpass QL with `out geom` to fetch building ways and parse tags.
- If a building has a `height` tag (in metres or with a unit), we parse it; otherwise we estimate height from `building:levels` multiplied by the default 3 m per floor (a common OSM convention for 3D rendering [23]).
- The extracted footprints (lat/lon coordinates with heights and names) are saved to a JSON file for Blender.

4.2 Generating 3D extrusions in Blender

- Run our `blender_export_buildings.py` inside Blender (3.3 LTS or later). The script reads the JSON file, converts WGS-84 coordinates to a local X–Y coordinate system (east-north-up), extrudes each footprint into a prism with the estimated height, assigns a placeholder material, and exports each building as an FBX.
- For Unreal compatibility we set the FBX export settings: **Forward = –Y**, **Up = Z**, and **Global Scale = 100** (Blender units in metres become centimetres in UE). Each building is exported as a separate file.

4.3 Importing FBX models into Cesium scenes

1. **Batch import with Unreal Python** – Unreal Engine supports Editor Python scripting. The snippet below reads a list of FBX files and their geodetic coordinates (from the same JSON used in Blender) and automates the import:

```
import unreal

# adjust these paths
BUILDING_JSON = r"/Game/OSMData/birmingham_buildings.json"
FBX_FOLDER = r"C:/Path/To/ExportedFBX"  # outside content folder
DEST_PATH = "/Game/Buildings"

# load coordinate data
import json
with open(unreal.Paths.convert_relative_path_to_full(BUILDING_JSON), 'r') as f:
    buildings = json.load(f)

asset_tools = unreal.AssetToolsHelpers.get_asset_tools()
editor_util = unreal.EditorLevelLibrary()

for b in buildings:
    fbx_path = f"{FBX_FOLDER}/{b['id']}.fbx"
    # import the mesh
    task = unreal.AssetImportTask()
    task.filename = fbx_path
    task.destination_path = DEST_PATH
    task.automated = True
    task.save = True
    asset_tools.import_asset_tasks([task])

    # spawn actor from imported mesh
    asset_name = f"{DEST_PATH}/{b['id']}"
    static_mesh = unreal.EditorAssetLibrary.load_asset(asset_name)
    actor = editor_util.spawn_actor_from_object(static_mesh, location=(0,0,0),
rotation=(0,0,0))

    # attach CesiumGlobeAnchor and set location
    anchor = unreal.CesiumGlobeAnchorComponent(actor)
    anchor.set_editor_property("latitude", b['latitude'])
    anchor.set_editor_property("longitude", b['longitude'])
    anchor.set_editor_property("height", b['height'])
    anchor.set_editor_property("east_up_south", False)  # East-South-Up frame
```

This script loops through buildings, imports the corresponding FBX as a Static Mesh, spawns it in the level, adds a **Cesium Globe Anchor** component and sets its `latitude`, `longitude` and `height`. Running it

in the **Python Console** (Window → Developer Tools → Python) or as an Editor Utility script will place all buildings automatically.

1. **Blueprint alternative** – Use a **Data Table** containing building IDs and their coordinates. Iterate through the rows in a Blueprint, spawn a Static Mesh Actor, add a Cesium Globe Anchor component and assign the coordinates. The **Placing Objects on the Globe** tutorial explains that adding a CesiumGlobeAnchor to a movable actor makes it georeferenced [24].

4.4 Merging with Cesium's streamed terrain

- After placement, the buildings will rest on the streamed terrain. If they appear to float or sink, adjust their `height` values in the JSON or via the anchor component.
- To avoid z-fighting with the Cesium OSM Buildings dataset, hide Cesium OSM Buildings for Birmingham (toggle the actor's visibility or remove it entirely) and rely on your custom models.

4.5 Common pitfalls & gotchas

- **Coordinate systems** – In Cesium for Unreal, +X points east, +Y south and +Z up relative to the origin. Blender exports should therefore use forward –Y and up Z; our scripts already set these.
- **Scaling** – Always export from Blender using a 100× scale (metres → centimetres). If models appear too large or small, adjust the global scale on import.
- **Precision** – Use double-precision coordinates. Unreal 5 uses double precision internally for georeferenced actors, but if you write custom shader code or physics you may still encounter floating-point jitter [25].
- **Sublevels** – When building multi-city worlds, use georeferenced sublevels and the **CesiumOriginShift** component to rebase the world origin automatically [26].

5 Scene composition & automation scripting

5.1 Batch importing & organizing assets

- The Python snippet above demonstrates bulk import. For even larger datasets, consider writing an **Editor Utility Blueprint** that exposes an array of file paths and coordinates. Use the **Spawn Actor from Class** node to instantiate your building class and the **Add Component** node to attach a **Cesium Globe Anchor** with your lat/lon/height.
- After import, group buildings into folders based on neighborhoods or categories (e.g., Downtown, Medical District). This simplifies toggling visibility or applying materials.

5.2 Annotations & labeling

- Create a new **Widget Blueprint** for labels. Use a `TextBlock` bound to a building's name or status.
- In your building actor class, add a **Widget Component** that points to this widget and set its **Draw Size**.
- To colour-code buildings (e.g., damage states), assign a dynamic material instance to each static mesh. Expose a parameter such as `Color` and set it via Blueprint or Python depending on the building's state.

5.3 Interactive overlays & pins

- You can spawn **Billboard** actors (e.g., static mesh spheres or icons) at specific coordinates with a Cesium Globe Anchor to represent points of interest (businesses, emergency sites).
- Bind user input or data feed events to update the label text, colour and size.

6 Weather & storm overlays

Unreal Engine does not include a dedicated weather system, but there are several approaches to add dynamic weather on top of Cesium's globe:

- **Ultra Dynamic Sky (UDS)** – A popular Unreal Marketplace asset that provides a procedural sky, time-of-day cycle and configurable weather (rain, snow, storm clouds). The UDS website notes that it is "designed to be more dynamic and natural than most sky solutions, offering a great degree of flexibility" [27].
- After purchasing UDS, enable the plug-in in your project. Drag the **Ultra Dynamic Sky** actor into the level. UDS exposes parameters for cloud coverage, precipitation and lightning frequency.
- To integrate with Cesium, place the UDS actor at the level origin and disable the default Cesium SunSky (or use UDS's built-in directional light).
- In Blueprints, adjust UDS parameters (e.g., `WeatherType`, `Rain Intensity`, `Storm Intensity`) based on your simulation scenario.

- **Volumetric clouds & Niagara** – Unreal's built-in volumetric cloud system can create cloud layers. Combine this with Niagara particle systems for rain or hail. Use Blueprint logic to toggle Niagara systems based on a weather variable.

- **Fluid Flux or Brushify** – Third-party plugins provide flooding and erosion effects. For example, Fluid Flux simulates water volumes that can flood low-lying areas. Combine this with Cesium's terrain to demonstrate storm surge scenarios.

Suggested demo scenarios:

1. **Storm command** – Increase UDS rain intensity and wind speed to simulate a severe thunderstorm over downtown Birmingham. Use Niagara lightning effects and enable screen-space rain on the camera.
2. **Flooding** – Use a water plane or Fluid Flux to raise water levels along rivers. Combine with precipitation to visualize flood risk.
3. **Hail** – Use a Niagara emitter for hail particles and adjust UDS to heavy cloud cover. Create sound cues for hail impacts.

7 Workflow documentation & scaling

- **Version control** – Place all scripts (`osm_extract.py`, `blender_export_buildings.py`, Unreal Python scripts) in a Git repository. Use branches to manage updates when the Cesium plug-in or UE version changes.

- **Onboarding doc** – Create a Markdown wiki (e.g., `docs/README.md`) that lists prerequisites (UE version, Blender version, Python 3.10), installation steps, and expected environment variables. Include troubleshooting tips for common errors (e.g., missing water collision profiles, mismatched coordinate systems, streaming artefacts).
- **Agent tasks checklists** – For scaling to new cities, maintain checklists:
- Define bounding box and run `osm_extract.py` to fetch building footprints.
- Review and clean building metadata; estimate heights for missing tags.
- Run Blender script to generate extruded FBX models.
- In Unreal, set CesiumGeoreference to the new city.
- Batch import the FBX assets and attach Cesium Globe Anchor components using the import script.
- Add global terrain/imagery and optional OSM Buildings for context.
- Configure environment (SunSky or weather system) and test navigation.
- **Troubleshooting** – Document issues such as misaligned buildings (check lat/lon order), scale mismatches (verify global scale in FBX export/import), asset streaming failures (refresh Cesium ion token), and jitter (use the OriginShift component). Use the Cesium Community Forum for support.

8 Legal & licensing checklist

- **Cesium for Unreal** – Released under the Apache 2.0 license [13]. The plug-in is free for commercial and non-commercial use.
- **Cesium World Terrain, imagery and OSM Buildings** – When streamed from Cesium ion you must abide by Cesium ion terms of service. Assets may be used in both commercial and non-commercial projects but cannot be redistributed.
- **OpenStreetMap building data** – OSM data is licensed under the Open Database License (ODbL) which requires attribution and share-alike for derivative databases [28]. If you publish or distribute derived building models, include credit such as "© OpenStreetMap contributors, licensed under the ODbL" and share any modified OSM data under the same license.
- **Third-party plugins** – Ultra Dynamic Sky and Fluid Flux are commercial assets; ensure you have appropriate licenses. For marketing exports (images or videos) you must credit data sources (Cesium, Bing Maps, OSM) according to their respective terms.

Summary

This checklist provides step-by-step guidance to integrate Cesium for Unreal into a UE5 project focused on downtown Birmingham. It covers installation, connecting to Cesium ion, streaming global terrain and imagery, scripting navigation to coordinates, importing OSM-derived building models, automating asset placement, adding dynamic weather overlays, documenting the workflow and ensuring legal compliance. By following these steps and using the provided scripts, you can build repeatable geospatial visualizations that blend real-world terrain with custom 3D assets for applications such as emergency response, urban planning or storm simulation.

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [14] [16] [17] Cesium for Unreal Quickstart – Cesium
https://cesium.com/learn/unreal/unreal-quickstart/

[11] [12] [13] cesium-unreal/README.md at main · CesiumGS/cesium-unreal · GitHub
https://github.com/CesiumGS/cesium-unreal/blob/main/README.md

[15] Cesium ion Access Tokens – Cesium
https://cesium.com/learn/ion/cesium-ion-access-tokens/

[18] [19] [20] [21] [22] Transition Between Locations on the Globe – Cesium
https://cesium.com/learn/unreal/unreal-flyto/

[23] Key:building:levels - OpenStreetMap Wiki
https://wiki.openstreetmap.org/wiki/Key:building:levels

[24] [25] [26] Placing Objects on the Globe – Cesium
https://cesium.com/learn/unreal/unreal-placing-objects/

[27] Ultra Dynamic Sky
https://www.ultradynamicsky.com/

[28] Open Database License - OpenStreetMap Wiki
https://wiki.openstreetmap.org/wiki/Open_Database_License