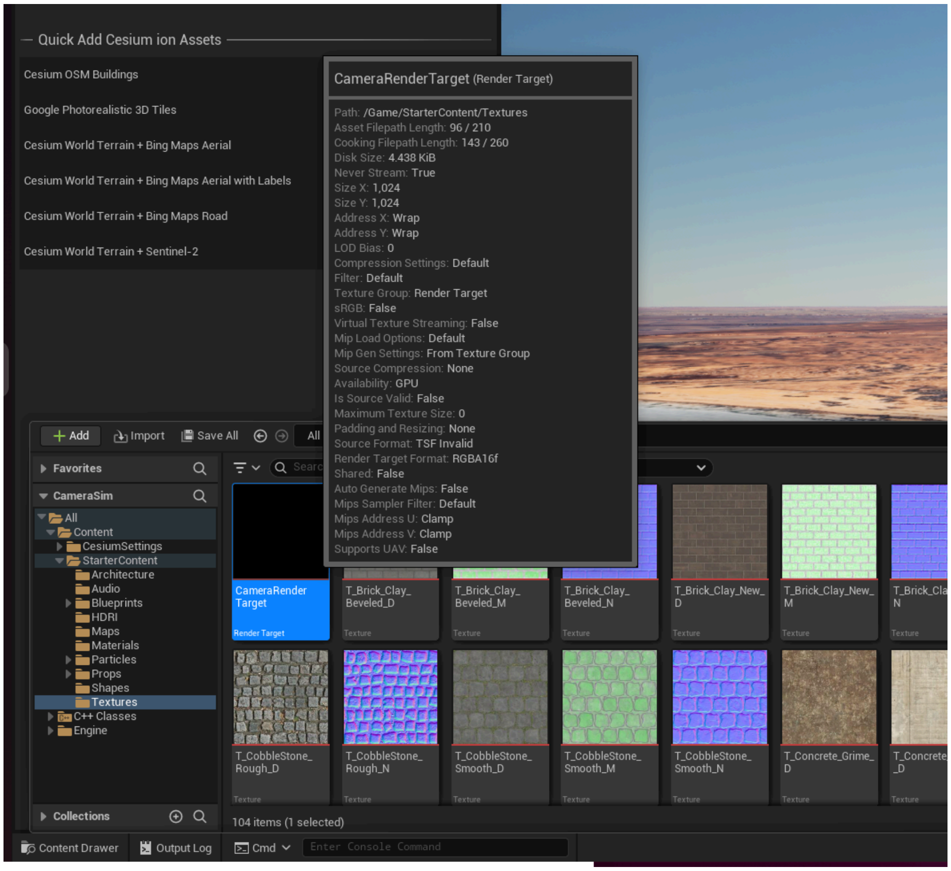
Things to Do Once

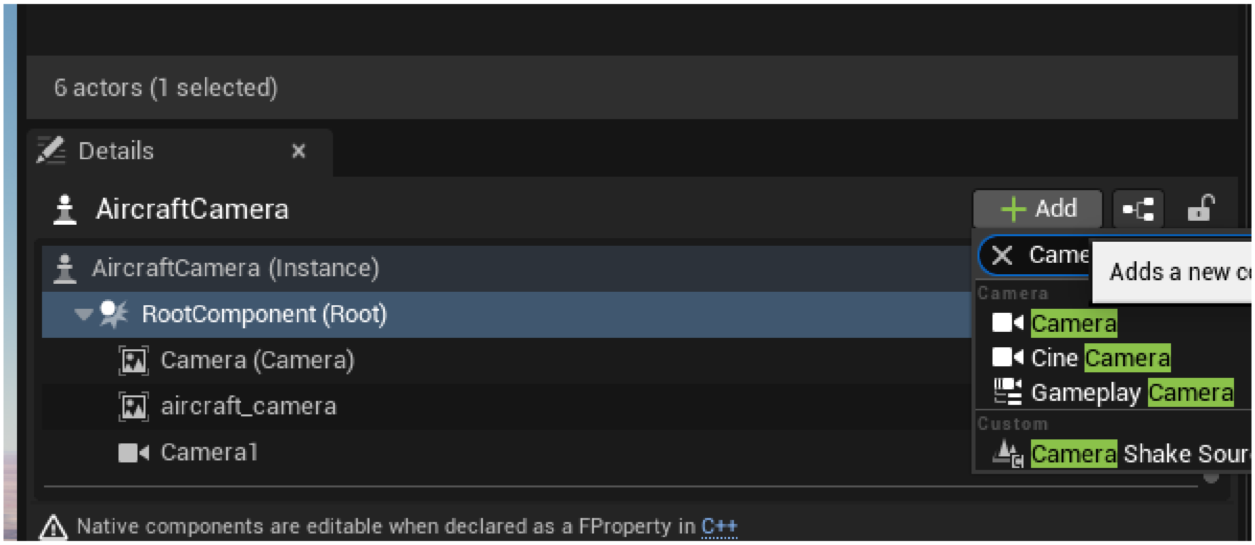
1) Install Unreal Engine 5
2) Download Cesium and extract within the UE5 installation, Linux_Unreal_Engine_5.5.1/Engine/Plugins/Marketplace
   - Done correctly, there should be a Engine/Plugins/Marketplace/CesiumForUnreal directory
   - Note that you may need to create the Marketplace Directory
3) Create a new project (blank game, c++, make sure ‘starter content’ is ticked)
4) Ensure that the Cesium Plugin is enabled (Edit -> Plugins, tick Cesium for Unreal), probably need to restart UE5 afterwar
5) follow the guide to setup cesium ion account
6) Create a new Empty level, add the ‘Google Photorealistic 3D Tiles’ and ‘Cesium SunSky’, and set the origin location to the location of your flight simulation. **Note you will need to create a new level for each location that you want to test.**
7) Adjust the date and time settings so that there is light
8) Ensure that the origin height is set to 0
9) Add a TextureRenderTarget within the Unreal Engine project (in All/Content/StarterContent/Textures). Set the size of this target to 1920x1080. Call it ‘CameraRenderTarget’. Ensure the format is RGBA16f



CameraRenderTarget (Render Target)

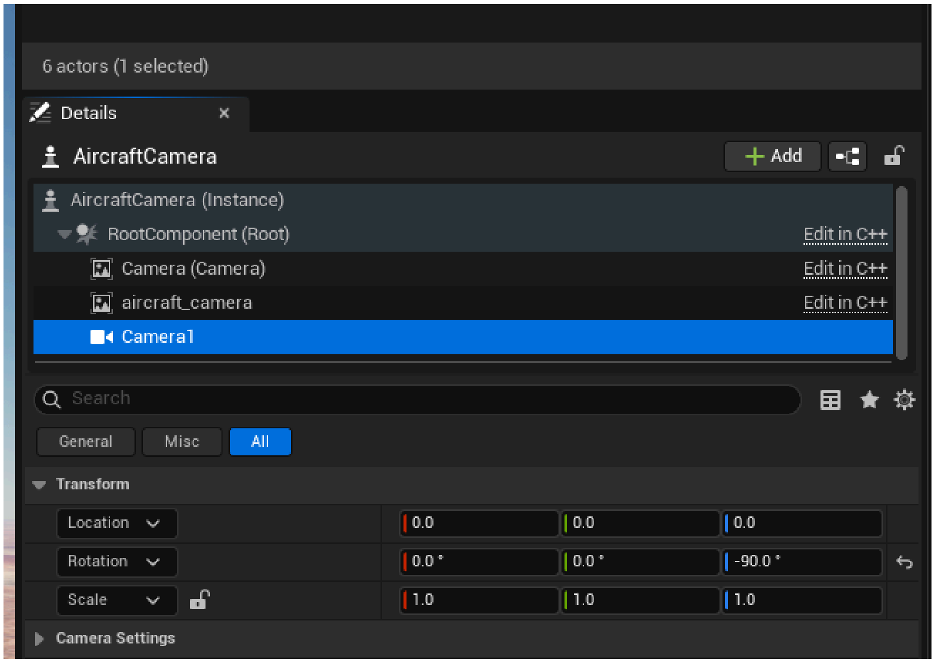
Path: /Game/StarterContent/Textures
Asset Filepath Length: 96 / 210
Cooking Filepath Length: 143 / 260
Disk Size: 4.438 KiB
Never Stream: True
Size X: 1,024
Size Y: 1,024
Address X: Wrap
Address Y: Wrap
LOD Bias: 0
Compression Settings: Default
Filter: Default
Texture Group: Render Target
sRGB: False
Virtual Texture Streaming: False
Mip Load Options: Default
Mip Gen Settings: From Texture Group
Source Compression: None
Availability: GPU
Is Source Valid: False
Maximum Texture Size: 0
Padding and Resizing: None
Source Format: TSF Invalid
Render Target Format: RGBA16f
Shared: False
Auto Generate Mips: False
Mips Sampler Filter: Default
Mips Address U: Clamp
Mips Address V: Clamp
Supports UAV: False

10) Create a bogus C++ Actor (this is to get UnrealEngine to recognise that you have custom c++ code, not sure how else to do this) Tools->New c++ class -> Actor -> Call it whatever)
11) Copy “AircraftCamera.h, AircraftCamera.cpp, JSBSim.h, JSBSim.cpp, Geo.h, Geo.cpp” into the project source code directory (Project/Source/Project/*)
12) In AircraftCamera.cpp, replace #import “CameraSim.h” with your project name eg. #import “TimProject.h”
13) In AircraftCamera.h, replace CAMERASIM_API with your project name eg. class TIMPROJECT_API AAircraftCamera : public APawn
14) Compile the code
15) Drag and drop ‘AircraftCamera’ into the scene
16) In the ‘AircraftCamera’ actor, add a Camera component



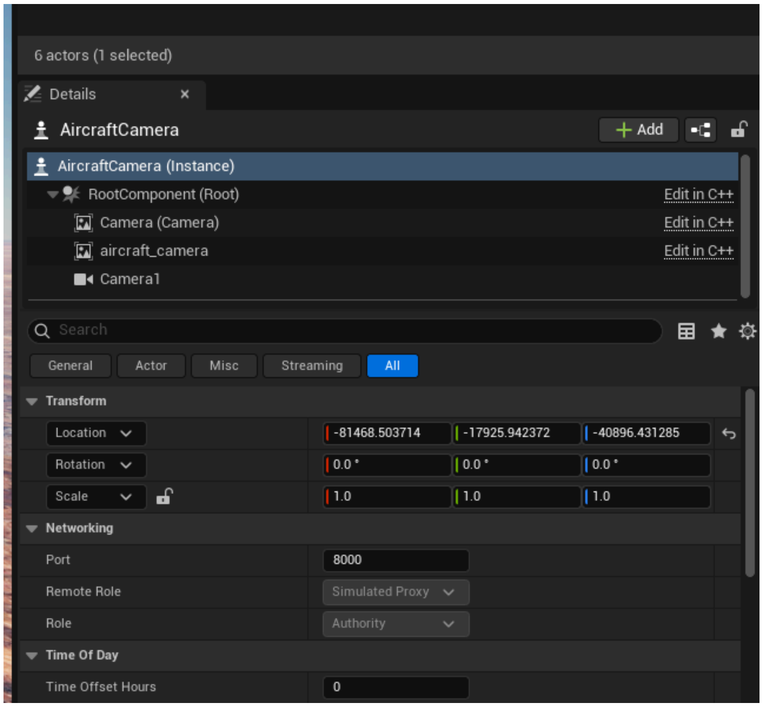
Native components are editable when declared as a FProperty in C++

17) Set the rotation of **all** Camera sub-components to (0, 0, -90)



Location 0.0 0.0 0.0
Rotation 0.0 ° 0.0 ° -90.0 °
Scale 1.0 1.0 1.0

18) Set the port for the JSBSim interface to whatever is coming out of JSBSim



AircraftCamera

- AircraftCamera (Instance)
  - RootComponent (Root)
    - Camera (Camera)
    - aircraft_camera
    - Camera1

Transform

Location	-81468.503714	-17925.942372	-40896.431285
Rotation	0.0 °	0.0 °	0.0 °
Scale	1.0	1.0	1.0

Networking

Port	8000
Remote Role	Simulated Proxy
Role	Authority

Time Of Day

Time Offset Hours	0

19) Create an empty file `sudo touch /usr/share/ue5camsim.data` change its user permissions to 666 (R/W)

20) Within the JSBSim aircraft model (eg. Wanderer.xml) add the following lines

```
<output name="127.0.0.1" type="SOCKET" protocol="UDP" port="8000" rate="30"
precision="9">
        <position> ON </position>
        <property> position/lat-geod-deg </property>
</output>
```

Things to Do Every Time

21) Launch SITL with this JSBSim model (ensure that the UDP output has been added to the aircraft descriptor)

22) Within Unreal Engine, Press play

23) Run UnrealEngineCamera.py as an example for how to pipe the data into Python