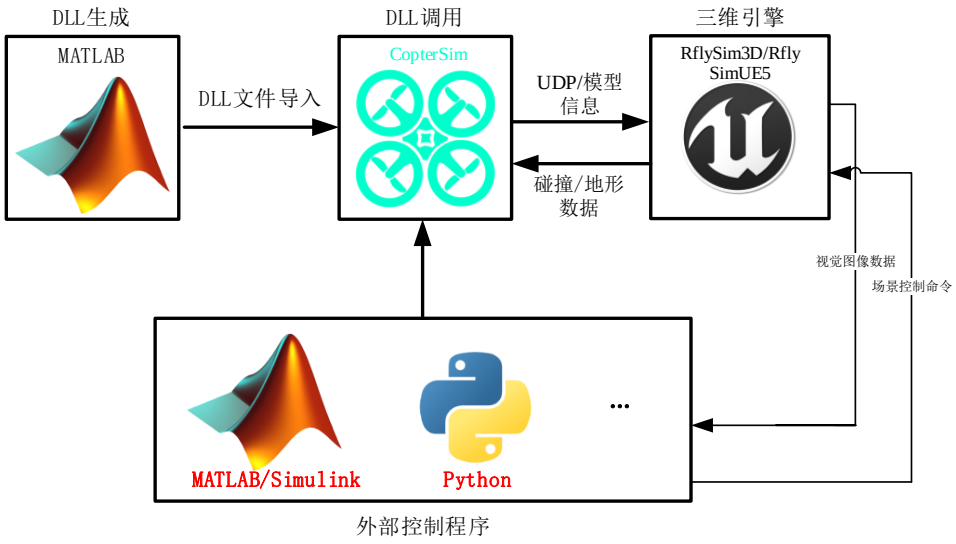
API 说明文件检索大纲

Contents

1、RflySim3D 的构架和基本功能

RflySim3D 在 RflySim 仿真平台中所处的位置如下图所示：



CopterSim 会根据从 Pixhawk（或者 PX4 SITL）传入的电机控制数据解算出无人机当前的状态（主要是位置、姿态数据），随后会将这些数据发送给 RflySim3D，而 RflySim3D 会将这些数据应用至场景里相应的无人机上，从而使我们能更直观的看到无人机的状态。

RflySim3D 使用 UDP 通信，能够接受一些来自外部的命令，例如切换场景、创建无人机、开启 UE 内置的物理碰撞等，命令的细节将在 RflySim3D 接口与使用方法介绍中介绍，总之 RflySim3D 可以接受来自 CopterSim、Python、Simulink 的 UDP 命令，并返回碰撞/地形数据以及视觉图像数据。

RflySim3D 内的场景元素还支持通过 XML 文件进行一些配置和扩展，主要是用 XML 配置无人机的构型（四旋翼、六旋翼、固定翼等）、模型在列表中的优先级、飞机的名字、飞机的初始位置与姿态、各致动器（一般是旋翼）的初始位置、姿态、材质、旋转轴、运动模式，还可以定义摄像机的位置，以及一些障碍组件（例如柱子、圆环）等等。

2. UE4/UE5 开发环境

对应平台例程：[UE 安装与蓝图编程入门实验]

UE4

Visual Studio 配置

虚幻引擎版本	Visual Studio 版本
4.25 或更高版本	VS 2019 (Default)
4.22 或更高版本	VS 2017 / VS 2019
4.15 或更高版本	VS 2017
4.10 - 4.14	VS 2015
4.2 - 4.9	VS 2013

详见设置虚幻引擎的 Visual Studio

UE4 安装及编辑器界面

安装步骤详见安装虚幻引擎

编辑器界面详见虚幻编辑器界面

UE5

Visual Studio 配置

虚幻引擎版本	VS 2019 版本	VS 2022 版本
5.3	16.11.5 或更高版本	17.4 或更高版本， 推荐 17.6（默认值）

虚幻引擎版本	VS 2019 版本	VS 2022 版本
5.2	16.11.5 或更高版本	17.4 或更高版本 （默认值）
5.1	16.11.5 或更高版本 （默认值）	17.4 或更高版本

详见为虚幻引擎 C++项目设置 Visual Studio 开发环境

UE5 安装及编辑器界面

安装步骤详见安装虚幻引擎

编辑器界面详见虚幻编辑器界面

3. 场景开发依赖软件

对应平台例程演示效果 3.1：0.ApiExps/e0_DevToolsUsage/4.SketchUpUsage/Readme.pdf

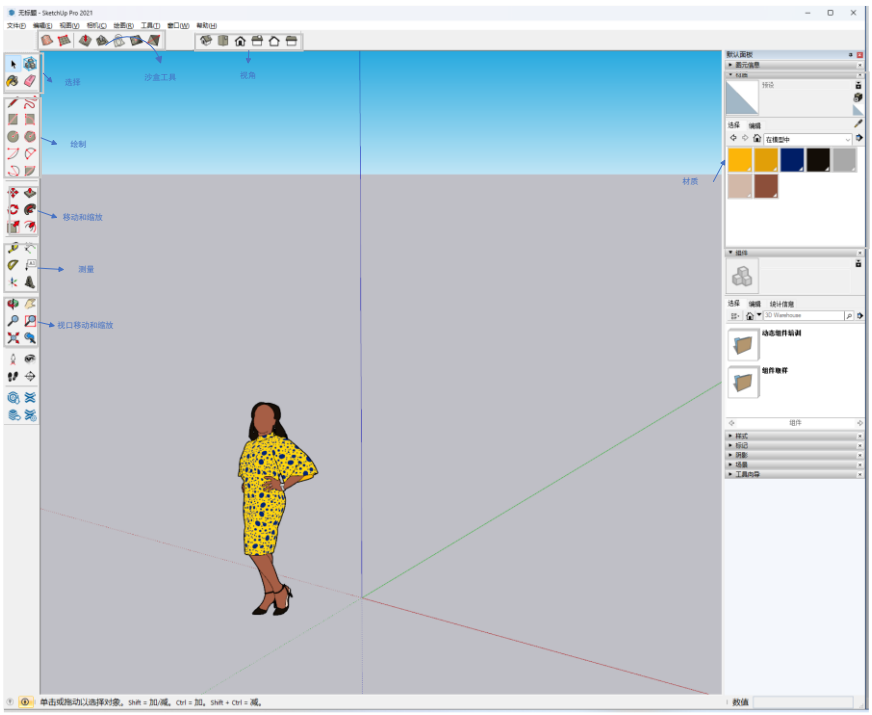
对应平台例程演示效果 3.2：0.ApiExps/e0_DevToolsUsage/2.3dsMaxUsage/Readme.pdf

对应平台例程演示效果 3.3：0.ApiExps/e0_DevToolsUsage/5.TwinmotionUsage/Readme.pdf

对应平台例程演示效果 3.4：0.ApiExps/e0_DevToolsUsage/3.CesiumForUnrealUsage/Readme.pdf

3.1 SketchUp

SketchUp 是由 Trimble Navigation 开发的流行的三维建模软件。它以其简单易用的界面和强大的建模工具而闻名。SketchUp 适用于各种应用领域，包括建筑设计、室内设计、景观设计等。它允许用户快速创建和编辑三维模型，并提供了丰富的插件库以扩展功能。



详细安装及使用教程如下：

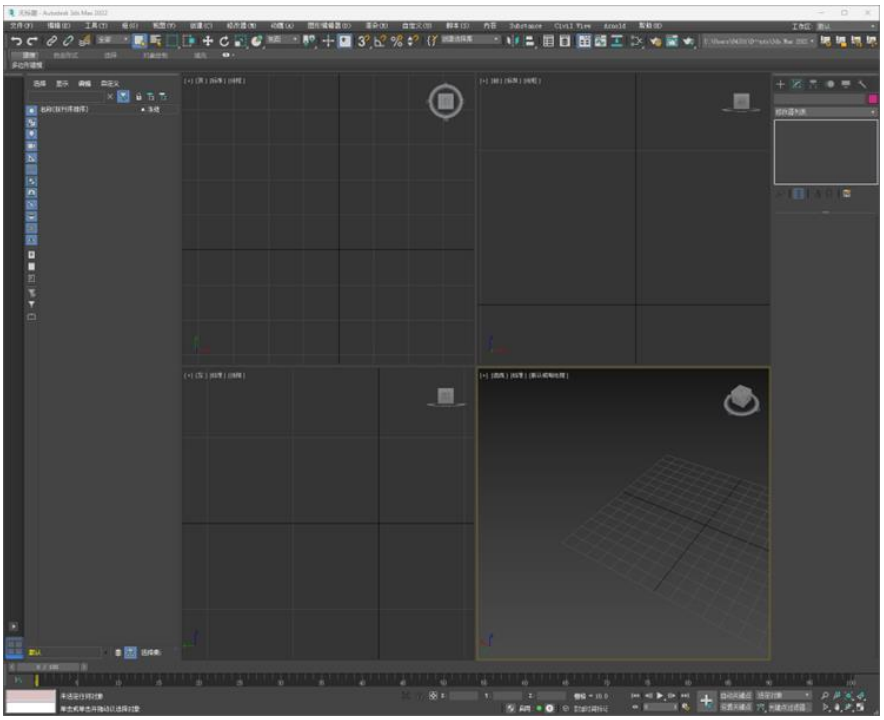
下载 SketchUp | 三维建模软件免费试用: https://www.sketchup.com/zh-CN/try-sketchup#for-personal

SketchUp Campus：https://learn.sketchup.com/collections

【合集】很详细的新手教程！ SketchUp 全套新手基础入门精品教程_哔哩哔哩_bilibili：https://www.bilibili.com/video/BV1Tb411E7Co/?spm_id_from=333.337.search-card.all.click&vd_source=49928ee3b2e585b60a2ebcfbd79ca829

3.2 3DsMax

3ds Max 是由 Autodesk 开发的功能强大的三维建模和动画软件。它提供了精细的建模工具，允许用户创建从简单的立方体到复杂的角色和场景的任何物体。



详细安装及使用方法如下：

3dsMax 官网 https://www.autodesk.com.cn/products/3ds-max/overview?term=1-YEAR&tab=subscription

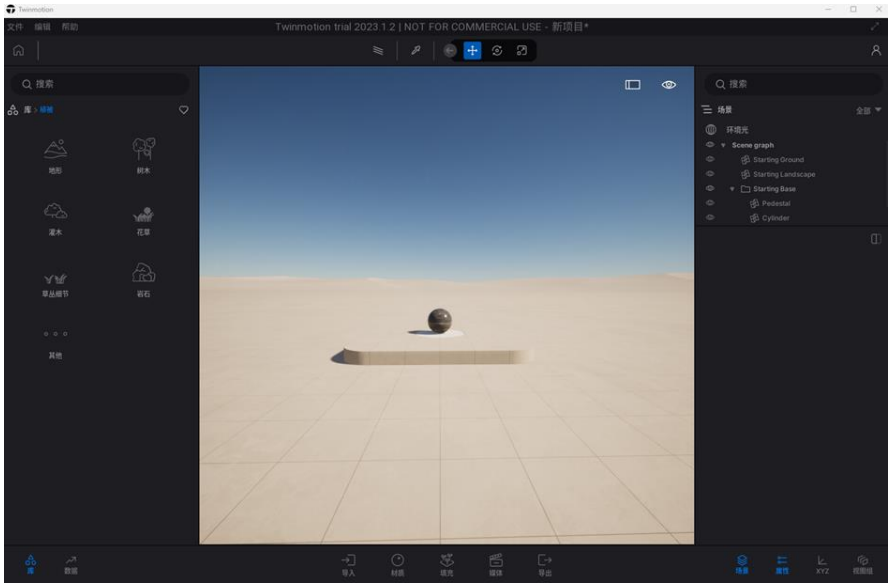
3Ds Max Tutorial: Full Beginner Crash Course (New for 2022) | RedefineFX - YouTube：https://www.youtube.com/watch?v=P-lWkbDXxVU

3.3 Twinmotion

Twinmotion 是一款基于 Unreal Engine 的实时可视化软件，提供了快速渲染和高质量渲染等功能。它内置了大量的建筑场景素材和材质，使用户能够快速创建逼真的场景。Twinmotion 具有直观易用的用户界面，可用于增强场景细节、更换材质和增加植被等。此外，Twinmotion 还支持与其他软件和引擎的集成，例如 Revit、SketchUp 和 Unreal Engine 等。尽管 Twinmotion 提供了一些基础的 3D 元素（如家具，植物，人物，天气，光线等），并允许用户使用这些元素进行场景组装和装饰，但它本质上并不是一个专门的 3D 建模工具。在 RflySim 平台中运用自定义场景时，可以充分利用 Twinmotion 提供的内置材质和物体，使已有的场景更加丰富和逼真。

2022 版的界面：

2023 版的界面：



详细安装及使用方法如下：

https://www.twinmotion.com/learning-resources

【Twinmotion】比 UE5 简单，10 分钟创建沙滩上手一个全新渲染软件_哔哩哔哩_bilibili：https://www.bilibili.com/video/BV1NY411w7MD/?spm_id_from=333.337.search-card.all.click&vd_source=49928ee3b2e585b60a2ebcfbd79ca829

3.4 Cesium

Cesium 是一种用于创建和展示三维地理信息的开源平台。它基于 WebGL 技术，可以在现代的 web 浏览器中实现高性能的三维地球渲染和地理数据可视化。Cesium 具有强大的地理空间分析和可视化功能，可用于实现虚拟地球、卫星图像浏览、地理信息系统等应用，为用户提供了交互性和可扩展性的地理信息展示解决方案。

详细使用方法如下：

Cesium for Unreal – Cesium：https://cesium.com/learn/unreal/

Cesium for Unreal Quickstart – Cesium：https://cesium.com/learn/unreal/unreal-quickstart/

https://www.bilibili.com/video/BV16p4y1t7Mc

3.5 Python

Python 的设计强调代码可读性，以及允许程序员用少量代码表达概念。它提供了丰富的标准库和各种各样的框架、库和工具；支持面向对象编程，但也允许过程式和函数式编程，且支持多种操作系统。在机器学习和人工智能领域被广泛应用，例如 TensorFlow, PyTorch，且可以用于编写各种自动化脚本。在 RflySim 平台安装过程中，会自动安装 python3.8.1 环境。

详细使用方法如下：

Python 基础教程 https://www.runoob.com/python/python-tutorial.html

3.6 Simulink

Simulink 是由 MathWorks 公司开发的一个基于图形的仿真和模型设计环境，常与 MATLAB 一起使用。

详细使用方法如下：

Simulink official tutorial https://ww2.mathworks.cn/products/simulink/getting-started.html

4、快捷键控制接口

对应平台例程演示效果 4：快捷键接口使用演示

为了提高用户交互性和操作便利性，在内置的全局命令基础上，RflySim3D/RflySimUE5 还具有一系列内置快捷键，其中部分的快捷键会与 CopterSim 发生交互。

这些快捷键可用于管理模拟环境和飞机，如弹出帮助菜单、显示或隐藏信息、切换地图和飞机、激活碰撞引擎、显隐小地图等（F1, ESC, S, H, D, M, B, C, P，L）。且可以切换到指定的地图、聚焦到指定飞机、修改模型三维样式（M+数字*, B+数字*, C+数字*）。

F1（帮助）：

弹出帮助菜单提示；

ESC（清除 Copter）：

清除所有飞机(先关闭所有 CopterSim)

S（显/隐 CopterID）:

显示/隐藏飞机 ID；

H（隐/显所有屏幕文字）:

隐藏/显示所有屏幕文字；

D（显/隐 Copter 数据）:

显示/隐藏当前飞机数据；

M（切换地图）:

切换地图(有 CopterSim 下会被切换回 CopterSim 的地图)；

M+数字*（切换到第*号地图）:

切换到第*号地图；

B（切换聚焦 Copter）:

在不同飞机间切换视角焦点；

B+数字*（聚焦到第*号 Copter）:

切换到第*号飞机

C（切换当前 Copter 样式）:

切换当前飞机（最新创建的）三维样式；

C+数字*（切换到第*号三维样式）:

切换到第*号三维样式;

CTRL + C（切换全部 Copter 样式）:

切换全部飞机三维样式

P（激活碰撞引擎）限个人高级版以上:

开启物理碰撞引擎（会与场景物体和地面发生碰撞，本功能仅支持完整版）

L（显/隐小地图）:

显示/隐藏小地图(默认出现的是显示在右下角，在小地图上双击会显示更多可修改的内容)

开启 CopterSim 后，快捷键常用于视角控制和飞机轨迹记录等（V, N, 鼠标操作, T），以及切换到指定的视角、指定运行轨迹的粗细、创建指定编号的障碍物、切换到指定的通信模式（V+数字*, N+数字*, T+数字*, O+数字*，P+数字*）。

V（切换跟随视角）:

飞机上的视角切换，0：跟随视角、1：前视摄像头、2：右视摄像头、等…；

V+数字*（切换到第*号跟随视角）:

切换到第*号视角

N（切换上帝视角）:

切换到飞机上帝视角，

N+数字*（切换到第*号上帝视角）:

切换到第*号上帝视角

0：跟随飞机视角（不随飞机姿态改变视角角度）1：固定地面视角且始终看向当前飞机、2：固定地面向北看视角、3：固定地面向南、等…；

鼠标左键按下拖动（调整视角角度）：

切换视角角度；

鼠标右键按下拖动（调整视角纵向位置）：

切换视角所在纵向 yz 位置

鼠标滚轮（调整视角横向位置）：

切换视角所在横向 x 位置

CTRL+鼠标滚轮（调整所有 Copter 尺寸）：

缩放所有飞机尺寸(多机时便于观察)；

ALT+鼠标滚轮（调整当前视角 Copter 尺寸）：

缩放当前视角飞机尺寸

T（开/关 Copter 轨迹记录）：

开启或关闭飞机轨迹记录功能

T+数字*（更改轨迹粗细为*号）：

开启/更改轨迹粗细为*号

鼠标双击（显示击中点信息）：

显示击中点的位置、尺寸、物体等信息。注：双击后立即按下 N 键，可以快速将视角切换到双击位置，便于物体创建

O+数字*（生成 ClassID 为“*”的物体）:

在鼠标双击处生成样式 ID 为“*”的物体（障碍物）

P+数字（切换通信模式）限个人高级版以上:

开启 P 模式后，RflvSim3D 会将障碍信息高速回传给各个 CopterSim 的 30100 系列端口 。0,1,2 分别对应本地发送，局域网发送，局域网极简发送(只碰撞时发送)的通信模式。

- P0 模式(按下 P+0 键，默认按下 P 键也会触发本模式) 下，RfvSim3D 会将每个飞机的周围环境距离数据高频传输给本电脑 (不会发送局域网) 上所有 CopterSim。
- P1 模式下，RflvSim3D 会将每个飞机周围距离数据高频传输给局域网内每个 CopterSim (通过指定 IP 和端口的方式以提高效率)
- P2 模式下，只有飞机发生碰撞过程中(和 1 秒内)，RfvSim3D 才会将障碍数据低频发送给局域网内的 CopterSim (通过指定 IP 和端口方式) ，因此从数据频率和目标 IP 数来优化通信

I+数字（切换局域网屏蔽状态，该功能仅限完整版以上）:

- I-1 模式(按下 I+-1 键，默认按下 I 键也会触发本模式) 下，会切换当前的局域网屏蔽状态。如果当前局域网是解除屏蔽的，这两个命令会使其变为屏蔽状态；反之亦然。
- I0 模式下，会启动局域网屏蔽。当执行这个命令后，所有局域网上的 RflySim3D 实例将不会接收或发送数据。
- I1 模式下，会解除局域网屏蔽，执行此命令后，局域网上的 RflySim3D 实例将恢复正常通信。

5、命令行控制接口

对应平台例程演示效果 5：命令行接口使用效果演示

RflySim3D 内置了一些全局的命令，可以完成 RflySim3D 相关的大部分功能，在 RflySim3D 的程序窗口中按下键盘的波浪号（~），可以打开控制台终端，这里可以触发 UE 引擎本身的一些内置命令，还可以触发 RflySim3D 平台内置的命令。这些命令相当于一个个的全局函数，具体命令如下：

5.1 RflySim 命令接口

RflyShowTextTime（显示文本）

函数解释

让 UE 显示 txt，持续 time 秒。使用方法如下

```
RflyShowTextTime(String txt, float time);
```

RflyLoad3DFile（执行 TXT 脚本）

函数解释

加载并执行路径下的 TXT 脚本文件。使用方法如下

```
RflyLoad3DFile(FString FileName);
```

RflySim3D 支持 txt 文件作为脚本，这里介绍的命令都可以写成 txt 脚本来使用。

操作示例

例如，我们在 D 盘创建一个“Test.txt”文件，在其中分行输入控制台命令，然后打开 RflySim3D，输入控制台命令：

```
RflyLoad3DFile “D:/Test.txt”
```

RflySetIDLabel（设置 CopterID 标签处显示）

函数解释

设置一个 Copter 的头顶 ID 位置的显示内容（会替换原先按 S 键默认显示的 CopterID）。使用方法如下：

```
RflySetIDLabel(int CopterID,FString Text, FString colorStr, float size);
```

参数解释

- CopterID（int）：要设置 ID 位置内容的 Copter 的 ID。
- Text（FString）：要在 ID 位置显示的文本内容。
- colorStr（FString）：文本的颜色。可以用表示颜色值的字符串进行指定，例如 "FF0000" 表示红色，"00FF00" 表示绿色。
- size（float）：文本的大小或缩放。

操作示例

使用该函数时，需要向参数传递适当的值。操作命令示例如下：

```
RflySetIDLabel 1000 "Hi, here's the test string" FFFF00 20
```

修改 ID 为 1000 的 Copter 上的标签的内容与颜色以及标签字体大小，这个颜色是 16 进制的 RGB 代码，FFFF00 表示红色与绿色的值为 255，蓝色的值为 0，它们混

合后就变成了黄色。

RflySetMsgLabel（设置 CopterID 标签下方显示）

函数解释

设置一个 Copter 头顶的 Message 显示的内容，使用方法如下：

```
RflySetMsgLabel(int CopterID, FString Text, FString colorStr, float size, float time, int flag);
```

参数解释

- CopterID（int）：要设置 Message 显示内容的 Copter 的 ID。
- Text（FString）：要在 Message 显示的文本内容。
- colorStr（FString）：文本的颜色。可以用表示颜色值的字符串进行指定，例如 "FF0000" 表示红色，"00FF00" 表示绿色。
- size（float）：文本的大小。
- time（float）：文本的显示时间，以秒为单位。在指定的时间结束后，消息将消失。
- flag（int）：操作的消息行数。共有 5 个消息标签，当 flag < 0 时表示新增一个消息。

操作示例

```
RflySetMsgLabel 1000 "Hi, here's the test string2" FF00FF 20 5 -1
```

将在 ID 标签下方显示一个紫色的文本字符串。这样使用 RflySetMsgLabel 函数，可以设置 Copter 的 Message 显示内容。您可以指定不同的参数来创建多行的 Message，每行消息可以在指定的时间内显示并消失。通过设置不同的 flag 值，可以操作不同的行数来更新或新增消息内容。

RflyChangeMapbyID（根据 ID 切换地图）

函数解释

根据地图的 ID 切换 RflySim3D 场景地图。使用方法如下：

```
RflyChangeMapbyID(int id)
```

操作示例

例如在控制台终端中输入

```
RflyChangeMapbyID 5;
```

就快速切换到 5 号地图了。

RflyChangeMapbyName（根据名称切换地图）

函数解释

根据地图名切换 RflySim3D 场景地图。使用方法如下：

```
RflyChangeMapbyName(String txt)
```

操作示例

例如在控制台终端中输入

```
RflyChangeMapbyName 3Ddisplay
```

就切换到地图 3DDisplay 了。

RflyCesiumOriPos（修改地图原点）

函数解释

修改 Cesium 地图的原点位置（需要 Cesium 全球大场景），允许设置地图原点的经纬高坐标。使用方法如下：

```
RflyCesiumOriPos(double lat, double lon, double Alt, double SolarTime)
```

参数解释

- lat（double）：地图原点的纬度值。
- lon（double）：地图原点的经度值。
- Alt（double）：地图原点的海拔（高度）值。
- SolarTime (double)：地图原点的太阳时。

操作示例

```
RflyCesiumOriPos 39.9042 116.4074 50 12
```

这将设置地图原点位置为北京的经纬度坐标。

注意

- 使用该函数之前，需要进入使用了 Cesium 插件的地图，并且最好处于联网状态以加载在线地图图像。
- 经纬度范围为[-180, 180]和[-90, 90]。确保提供的经纬度值在有效范围内。

RflyCameraPosAngAdd（偏移相机）

函数解释

给摄像机的位置与角度增加一个偏移值（在当前摄像机位置和角度的基础上进行调整），使用方法如下：

```
RflyCameraPosAngAdd(float x, float y, float z,float roll,float pitch,float yaw)
```

参数解释

- x（float）：摄像机在 x 轴方向上的偏移值。

- y（float）：摄像机在 y 轴方向上的偏移值。
- z（float）：摄像机在 z 轴方向上的偏移值。
- roll（float）：摄像机绕 x 轴旋转的偏移角度（滚转）。
- pitch（float）：摄像机绕 y 轴旋转的偏移角度（俯仰）。
- yaw（float）：摄像机绕 z 轴旋转的偏移角度（偏航）。

操作实例

```
RflyCameraPosAngAdd 0 0 -10 0 -30 0
```

让摄像机向上移动 10 米，并且向下俯视 30 度

RflyCameraPosAng（重设相机）

函数解释

直接设置摄像机的位置和角度，使用方法如下：

```
RflyCameraPosAng(float x, float y, float z, float roll, float pitch, float yaw);
```

参数解释

- x（float）：摄像机在世界坐标系下的 X 轴位置。
- y（float）：摄像机在世界坐标系下的 Y 轴位置。
- z（float）：摄像机在世界坐标系下的 Z 轴位置。
- roll（float）：摄像机绕 X 轴旋转的角度（滚转）。
- pitch（float）：摄像机绕 Y 轴旋转的角度（俯仰）。
- yaw（float）：摄像机绕 Z 轴旋转的角度（偏航）。

操作实例

```
RflyCameraPosAng 0 0 1000 0 0 0
```

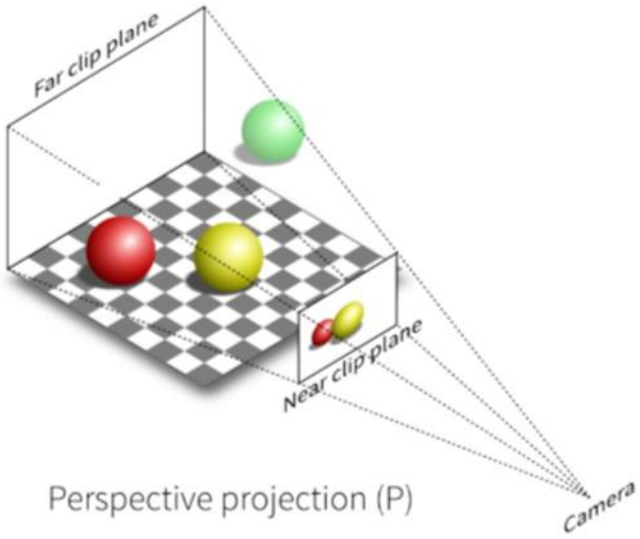
将摄像机设置在世界坐标系原点上方 1000 个单位的高度，角度为默认的正前方。

RflyCameraFovDegrees（设置视域）

函数解释

设置摄像机的视域体 FOV 角度（Field of View 视场角），使用方法如下：

```
RflyCameraFovDegrees(float degrees)
```



此函数会修改摄像机的水平视场角，也就是上图平面上最左侧的中点与最右侧的中点与摄像机形成的角度。一般会默认该角度为 90°。

操作示例

将该角度改为 120°操作命令如下：

```
“RflyCameraFovDegrees 120
```

RflyChange3Dmodel（修改 Copter 样式）

函数解释

修改一个无人机的模型样式，用法如下

```
RflyChange3DModel(int CopterID, int veTypes=0)
```

此命令可以修改指定无人机的三维模型样式

参数解释

- **CopterID**：无人机的 ID，用于指定要修改的无人机。
- **veTypes**：三维模型样式的编号。这个参数是可选的，默认为 0。可以指定不同的数字来对应不同的模型样式。

操作示例

例如这里无人机的 ID 为 1000，那么想要将它的样式变为 1 号，可以使用命令

```
RflyChange3DModel 1000 1
```

RflyChangeVehicleSize（调整 Copter 尺寸）

函数解释

修改一个无人机的缩放大小，用法如下：

```
RflyChangeVehicleSize(int CopterID, float size=0)
```

参数解释

- CopterID：无人机的 ID，用于指定要修改的无人机。
- size：缩放的倍数。这个参数是可选的，默认为 0，表示不进行缩放。通过指定一个正数倍数来使无人机变大，指定一个小于 1 的倍数来使无人机变小。

操作示例

假设你想将无人机 ID 为 1000 的无人机大小增大到原来的 10 倍。你可以使用以下命令：

```
    RflyChangeVehicleSize(1000, 10);
```

RflyMoveVehiclePosAng（偏移 Copter）

函数解释

给无人机的位置与角度设置一个偏移值，isFitGround 设置无人机是否适应地面，用法如下：

```
    RflyMoveVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float rol
l, float pitch, float yaw)
```

参数解释

- CopterID：无人机的 ID，用于指定要设置偏移的无人机。
- isFitGround：是否适应地面高度。如果设置为 1，则无人机的 z 坐标会根据地形的高度进行调整；如果设置为 0，则无人机的 z 坐标保持不变。
- x、y、z：位置偏移的三个坐标轴分量。通过设置这些参数，可以将无人机在三维空间中沿着 x、y、z 轴移动指定的距离。
- roll、pitch、yaw：旋转角度的欧拉角分量。通过设置这些参数，可以使无人机绕其自身的 x、y、z 轴旋转指定的角度。

操作示例

```
    RflyMoveVehiclePosAng 1000 1 -10 -10 -10 0 -20 0
```

可以让飞机移动（-10，-10，-10）米，并且 pitch 角度-20°。如果 isFitGround 设置为 1，则无人机的 z 坐标会根据地形的高度决定。

RflySetVehiclePosAng（重设 Copter）

函数解释

设置无人机的位置与角度，用法如下：

```
    RflySetVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float roll,
 float pitch, float yaw)
```

此命令与 RflyMoveVehiclePosAng 函数类似，但它是直接设置无人机的位置，而不是给出一个增量。

参数解释

- CopterID：无人机的 ID，用于指定要设置位置和角度的无人机。
- isFitGround：是否适应地面高度。如果设置为 1，则无人机的 z 坐标会根据地形的高度进行调整；如果设置为 0，则无人机的 z 坐标保持不变。
- x、y、z：位置坐标的三个分量。通过设置这些参数，可以直接将无人机移动到指定的位置。
- roll、pitch、yaw：旋转角度的欧拉角分量。通过设置这些参数，可以直接将无人机设置为指定的角度。

操作示例

假设你想直接将无人机 ID 为 1000 的无人机设置到位置(-10, -10, -10)米，并将其设置为 pitch 角度为-20°。你可以使用以下命令：

```
RflySetVehiclePosAng(1000, 1, -10, -10, -10, 0, -20, 0);
```

RflySetActuatorPWMs（触发蓝图接口，该功能仅限个人高级版以上）

函数解释

传入 8 个值，并触发目标无人机的蓝图的接口函数

```
RflySetActuatorPWMs(int CopterID, float pwm1, float pwm2, float pwm3, float pwm4, float pwm5, float pwm6, float pwm7, float pwm8);
```

它其实就是直接设置无人机的 8 位电机数据，一般该数据是由 UDP 发送来的，但可以用此命令直接设置。

参数解释

- CopterID：无人机的 ID，用于指定要设置电机数据的无人机。
- pwm1、pwm2、pwm3、pwm4、pwm5、pwm6、pwm7、pwm8：8 个电机数据。这些数据控制着无人机的各个旋翼或其他致动器的旋转速度或其他行为。具体含义需要参考相关蓝图模型的设置。

操作示例

假设你已通过 RflySim3D 创建了一个 ID 为 1000 的无人机，并且希望将该无人机的螺旋桨开始转动，其中前四个电机数据控制着旋转速度为每秒 10 单位。

```
RflySetActuatorPWMs(1000, 10, 10, 10, 10, 0, 0, 0, 0);
```

RflySetActuatorPWMsExt（触发扩展蓝图接口，该功能仅限个人高级版以上）

函数解释

用于传入 16 个值，并触发目标无人机的蓝图接口函数。用法如下

```
RflySetActuatorPWMsExt(int CopterID, float pwm9, float pwm10, float pwm11, float pwm12, float pwm13, float pwm14, float pwm15, float pwm16, float pwm17, float pwm18, float pwm19, float pwm20, float pwm21, float pwm22, float pwm23, float pwm24)
```

参数解释：

- CopterID：无人机的 ID，用于指定目标无人机调用蓝图接口函数。
- pwm9 - pwm24：16 个参数，作为数据传递给目标无人机的蓝图接口函数。具体的含义和作用取决于目标无人机的蓝图设计。

注意

此函数需要完整版本的 RflySim 来使用，并且目标无人机必须是自定义蓝图类的对象。这 16 维数据会直接传递给目标无人机的蓝图类中的 "ActuatorInputsExt" 函数，而 RflySim3D 不会默认处理这些数据。

通过调用 "RflySetActuatorPWMsExt" 函数，你可以传递各种控制数据到目标无人机的蓝图中，从而实现各种控制效果。这个函数提供了更多的参数，以便用于灵活的控制和交互。

RflyDelVehicles（清除 Copter）

函数解释

删除指定 Copter，用法如下

```
RflyDelVehicles(FString CopterIDList);
```

该函数可以根据 ID 删除当前存在于场景的无人机。RflySim3D 并不会主动删除场景中的无人机，哪怕已经没有接收到它们的数据了。（这也是为什么关闭 CopterSim 后无人机并没有从场景中消失），必须明确的要求 RflySim3D 移除这些无人机。

参数解释

- CopterIDList：一个 FString 类型的参数，用于指定要删除的无人机的 ID 列表。多个 ID 之间使用逗号分隔。

操作示例

假设在场景中存在两架无人机，其 ID 分别为 1000 和 1001。如果你想删除这两架无人机，你可以使用以下命令：

```
RflyDelVehicles("1000,1001");
```

注意

在删除无人机之前，应当确保停止向 RflySim3D 发送这些无人机的数据，以避免可能导致无人机被重新创建的情况发生。

RflyScanTerrainH（扫描地形）

函数解释

获取地形数据，用法如下

```
    RflyScanTerrainH(float xLeftBottom(m), float yLeftBottom(m), float xRightTop(m), float yR
ightTop(m), float scanHeight(m), float scanInterval(m))
```

该函数可以扫描三维地形，生成一个 png 的高度图与 txt，CopterSim 程序会需要它才知道 UE 有哪些地形、以及它们的高程。但该函数需要知道地形的大小和高度、扫描的精度。

参数解释

- xLeftBottom、yLeftBottom：地形的左下角坐标，以米为单位。这指定了地形扫描的起始位置。
- xRightTop、yRightTop：地形的右上角坐标，以米为单位。这指定了地形扫描的结束位置。
- scanHeight：地形的最大高度限制，以米为单位。该参数用于指定地形的高度范围，以便生成高度图和文本文件。
- scanInterval：扫描采样间隔，以米为单位。该参数决定了扫描时对地形进行采样的精度，也就是采样点之间的距离。

操作示例

例如，你可以使用以下命令扫描地形：

```
    RflyScanTerrainH(-1000, -1000, 1000, 1000, 200, 1);
```

这个命令指定了地形的左下角坐标为 (-1000, -1000) 米，右上角坐标为 (1000, 1000) 米。地形高度限制为最大不超过 200 米。扫描间隔为每隔 1 米进行采样。

执行该命令后，RflySim3D 将扫描指定范围内的地形，并生成一个高度图的 PNG 图像文件和一个文本文件（TXT 格式），可以在“\PX4PSP\RflySim3D”找到此地图的 2 个同名文件。

RflyReqVehicleData（激活数据回传）

函数解释

激活数据回传模式，在该模式下，一旦 RflySim3D 收到 Copter 的更新数据，它会

再次将相关数据发送出来。用法如下

```
RflyReqVehicleData(FString isEnable);
```

参数解释

如果传入的参数 isEnable 不为 0，则 RflySim3D 会开始发送 Copter 的数据。

默认情况下，RflySim3D 只接收来自外部的 Copter 的信息，而收到的 UDP 数据的结构类似于以下的结构体：

```
struct SOut2SimulatorSimple {
int checkSum;
int copterID;
int vehicleType;
float MotorRPMSMean;
float PosE[3];
float AngEuler[3];
}
```

该结构体的成员变量和类型：

- checkSum：整型变量，表示数据包的校验码。
- copterID：整型变量，表示飞机的 ID 号。
- vehicleType：整型变量，表示飞机的类型。
- MotorRPMSMean：浮点型变量，表示电机转速的平均值。
- PosE：包含 3 个浮点型变量的数组，表示飞机的位置，三个元素分别是 X、Y 和 Z 坐标。
- AngEuler：包含 3 个浮点型变量的数组，表示飞机的欧拉角，三个元素分别是 Roll、Pitch 和 Yaw。

RflySim3D 默认不会发送三维场景中无人机的位置和姿态信息，但是如果有需要的话，可以调用该命令 "RflyReqVehicleData 1"。当 RflySim3D 收到此命令时，它将进入"数据回传模式"，开始使用 UDP 发送请求的 Copter 的数据。在该模式下，一旦 RflySim3D 收到 Copter 的更新数据，它会再次将相关数据发送出来。

具体发送的数据结构如 reqVeCrashData，包含以下字段：

- checksum：数据包校验码（固定值 1234567897）
- copterID：当前飞机的 ID 号
- vehicleType：当前飞机的样式
- CrashType：碰撞物体类型，-2 表示地面，-1 表示场景静态物体，0 表示无碰撞，1 及以上表示被碰飞机的 ID 号
- runnedTime：当前飞机的时间戳
- VelE：当前飞机的速度
- PosE：当前飞机的位置
- CrashPos：碰撞点的坐标
- targetPos：被碰物体的中心坐标
- AngEuler：当前飞机的欧拉角

- MotorRPMS：当前飞机的电机转速
- ray：飞机的前后左右上下扫描线
- CrashedName：被碰物体的名字

操作示例

```
RflyReqVehicleData 1
```

RflySetPosScale（全局缩放）

函数解释

全局位置的缩放，用法如下

```
RflySetPosScale(float scale);
```

参数解释

该函数会影响通过 UDP 传入 RflySim3D 的 Copter 的位置信息，默认 scale==1。

RflySim3D 接收的结构体如下：

```
struct SOut2SimulatorSimple {
 int checkSum;
 int copterID;
 int vehicleType;
 float MotorRPMSMean;
 float PosE[3]; //包含 3 个浮点型变量的数组，表示飞机的位置，三个元素分别是 X、Y 和 Z 坐标。
 float AngEuler[3];
}
```

RflySim3D 在收到位置信息后还会进行一个处理：“PosE[i]= PosE[i]*scale”，进行一个全局的缩放变换。

操作示例

该函数主要用于单位统一，RflySim3D 中的空间单位是 cm，如果外部传入的数据的单位是 m，则可以使用命令

```
RflySetPosScale 100
```

这样就不必额外进行单位转换的逻辑了。

RflyReqObjData（指定回传数据）

函数解释

请求获取三维场景中物体的数据，用法如下

```
RflyReqObjData(int opFlag, FString objName, FString colorStr);
```

该函数类似“RflyReqVehicleData 函数”，它也会回传三维场景中物体的一些数据。但它一次只回传一个指定目标的数据。

参数解释

- opFlag：操作标志，用于指定返回的数据类型。值的含义如下：

0：返回指定 ID 的相机（Camera）数据。

1：返回指定 ID 的飞机（Copter）数据。

2：返回指定名称的物体（Object）数据。

20：清除所有相机

21：清除所有飞机

22：清除所有物体

23：清除所有飞机和物体

24：清除所有相机、物体和飞机

10：删除一个相机

11：删除一个飞机

12：删除一个物体

- objName：物体名称或相机 ID，用于指定要获取数据的目标物体或相机。对于相机数据，可以指定相机的 ID。对于物体数据，可以指定物体的名称。
- colorStr：颜色字符串，可选参数，用于指定数据的颜色。

当 opFlag 为 0 时，该函数会返回指定 ID（objName=seqID）的用于捕获图像的相机数据。相机数据结构如下所示：

```
struct CameraData {
int checksum = 0;    // 校验和，常数 1234567891
int SeqID;           // 相机序号
int TypeID;          // 相机类型
int DataHeight;      // 图像像素高度
int DataWidth;       // 图像像素宽度
float CameraFOV;     // 相机视场角
float PosUE[3];      // 相机的中心位置
float angEuler[3];   // 相机的欧拉角
double timestmp;     // 时间戳
};
```

当 opFlag 为 1 时，该函数会返回指定 ID 的飞机（objName=CopterID）的数据。飞机数据结构如下所示：

```
struct CoptReqData {
int checksum = 0;    // 校验和，常数 1234567891
int CopterID;        // 飞机 ID
float PosUE[3];      // 飞机的中心位置（人为三维建模时指定，姿态坐标轴，不一定在几何中心）
float angEuler[3];   // 飞机的欧拉角
float boxOrigin[3];  // 物体几何中心坐标
float BoxExtent[3];  // 物体外框长宽高的一半
double timestmp;     // 时间戳
};
```

当 opFlag 为 2 时，该函数会返回指定 ID（objName= seqID/ ObjName[32]）的物体的数据。物体数据结构如下所示：

```
struct ObjReqData {
int checksum = 0;              // 校验和，常数 1234567891
int seqID = 0;
```

```
    float PosUE[3];                    // 物体的中心位置（人为三维建模时指定，姿态坐标轴，不一定在几何中心）
    float angEuler[3];                 // 物体的欧拉角
    float boxOrigin[3];                // 物体几何中心坐标
    float BoxExtent[3];                // 物体外框长宽高的一半
    double timestmp;                   // 时间戳
    char ObjName[32] = { 0 };          // 碰物体的名字
    };
```

操作示例

当 opFlag==0 时，它可以根据请求创建一个相机用于捕获图像，而相机的数据可以通过该函数来获得：

```
RflyReqObjData 0 0 FFFFFF
```

当 opFlag==1 时，打开 RflySim3D，鼠标双击地面+按下字母 O+数字 3，创建一个 ID 飞机（ID 默认为 1000），输入命令

```
RflyReqObjData 1 1000 FFFFFF
```

当 opFlag==2 时，在 RflySim3D 中，并不是所有物体都是 Copter 类的对象（例如地形、障碍、建筑等），而此接口可以用于获取这些非 Copter 的物体的信息。

打开 RflySim3D，双击地面，我们可以看到鼠标击中地面后，屏幕上输出了一些信息，其中第一个单词表示击中物体的名字，也就是说“地形的名字为 Landscape_1”

然后我们使用命令

```
RflyReqObjData 2 Landscape_1 FFFFFF
```

表示我们想知道名为“Landscape_1”的对象的信息。

RflyClearCapture（清空图像缓存）

函数解释

清空抓取的图像

```
RflyClearCapture(int seqID)
```

该函数会清空指定 seqID 的图像传感器捕捉的图像的内存，如果 seqID<0 则清空所有相机的图像的内存数据。

参数解释

- seqID：定义的相机编号

```
RflyClearCapture -1
```

RflyDisableVeMove（拒收指定 Copter 数据）

函数解释

拒接接收指定 ID 的 Copter 的信息

```
RflyDisableVeMove(FString CopterIDList, int disable);
```

使用该函数可以让 RflySim3D 根据 ID 拒绝接收一些无人机的数据，相当于禁止了这些 ID 的 Copter 的创建、移动、姿态变化。

参数解释

- CopterIDList：飞机 ID 列表，以逗号分隔。用于指定要禁止接收信息的飞机 ID。
- disable：禁用标志，控制是否禁用指定 ID 的飞机的信息。当值为 1 时，禁用；当值为 0 时，取消禁用。

操作示例

打开在“\Desktop\RflyTools”打开软件在环仿真 SITLRun，使用 QGC 让无人机处于移动状态，然后我们使用命令：

```
RflyDisableVeMove 1 1
```

会发现 QGC 中无人机仍在移动，CopterSim 中的模拟坐标仍在变化，但 RflySim3D 中的无人机停下来了。

RflyChangeViewKeyCmd（模拟快捷键）

函数解释

模拟在按一个快捷键 + 输入一个数字的效果，使用方法如下：

```
RflyChangeViewKeyCmd(String key, int num)
```

参数解释

见 4、快捷键控制接口

操作示例

例如，按快捷键 M+数字就可以切换到指定 ID 的地图，那么可以输入命令切换到 5 号地图。

```
RflyChangeViewKeyCmd M 5
```

RflyEnImgSync（切换传图模式）

函数解释

启用同步或异步发图模式

```
RflyEnImgSync(int isEnable)
```

参数解释

- isEnable：值为 0 禁用异步模式，值为 1 启用同步模式

操作示例

5.2 UE4/UE5 常用内置命令

5.2.1 常规使用

t.Maxfps（限制帧率）

函数解释：

通过设置最大帧率，可以限控制仿真程序的性能表现。

参数解释：

无参数。该命令只用于设置最大帧率，不需要额外的参数进行指定。

操作示例：

```
t.MaxFPS 60
```

上述示例中，将最大帧率设置为 60 帧。

```
t.MaxFPS 30
```

上述示例中，将最大帧率设置为 30 帧。

slomo （修改运行速度）

函数解释：

通过调整运行速度，加快或减慢应用程序的运行。

参数解释：

- 运行速度值：用于指定运行的速度。默认值为 1，表示正常速度。较小的值会减慢速度，较大的值会加快速度。

操作示例：

```
slomo 0.5
```

上述示例中，将运行速度设置为正常速度的一半。

```
slomo 2
```

上述示例中，将运行速度设置为正常速度的两倍。

HighResShot（自定义尺寸截图）

函数解释：

通过指定截图的尺寸，可以获取更高分辨率的截图。

参数解释：

- 截图尺寸值：用于指定截图的宽度和高度。通常使用像素为单位的数值进行指定。

操作示例：

```
HighResShot 1920x1080
```

上述示例中，进行一个宽度为 1920 像素，高度为 1080 像素的截图。截图将以指定的尺寸保存。

stat fps（显示更新率）

函数解释：

通过使用该命令，可以实时监测仿真程序的帧率表现。

参数解释：

无参数。该命令只用于显示帧率信息，不需要额外的参数进行指定。

操作示例：

```
stat fps
```

上述示例中，启用帧率显示。RflySim3D 中将显示当前的帧率信息, 再次输入该命令就会隐藏帧率显示

stat unit（显示各类消耗）

函数解释：

在 RflySim3D 中显示性能统计信息。通过使用该命令，可以实时监测各个系统的性能表现，如 CPU、GPU 和渲染时间等。

参数解释：

无参数。该命令只用于显示性能统计信息，不需要额外的参数进行指定。

操作示例：

```
stat unit
```

上述示例中，启用性能统计信息显示。RflySim3D 中将显示 CPU、GPU 和渲染时间等性能数据。

stat rhi（显示 GPU 上的消耗细则）

函数解释：

在 RflySim3D 中显示渲染接口（Render Hardware Interface）的性能统计信息。通过使用该命令，可以实时监测 RflySim3D 使用的渲染接口的性能表现。

参数解释：

无参数。该命令只用于显示渲染接口的性能统计信息，不需要额外的参数进行指定。

操作示例：

```
stat rhi
```

上述示例中，启用渲染接口性能统计信息显示。RflySim3D 中将显示渲染接口的性能数据，如渲染帧数、GPU 利用率和 GPU 内存使用情况等。

stat game（显示各进程 Tick 反馈时间）

函数解释：

在 RflySim3D 中显示程序逻辑和性能的统计信息。通过使用该命令，可以实时监测 RflySim3D 的逻辑操作和性能表现。

参数解释：

无参数。该命令只用于显示 RflySim3D 逻辑和性能统计信息，不需要额外的参数进行指定。

操作示例：

```
stat game
```

上述示例中，启用 RflySim3D 逻辑和性能统计信息显示。RflySim3D 中将显示 RflySim3D 逻辑帧数、RflySim3D 更新时间和渲染时间等统计数据。

stat gpu（显示帧 GPU 统计）

函数解释：

在 RflySim3D 中显示 GPU（显卡）的性能统计信息。通过使用该命令，可以实时

监测 RflySim3D 中 GPU 的性能表现。

参数解释：

无参数。该命令只用于显示 GPU 的性能统计信息，不需要额外的参数进行指定。

操作示例：

```
stat gpu
```

上述示例中，启用 GPU 的性能统计信息显示。RflySim3D 中将显示 GPU 的使用率、渲染时间和 GPU 内存使用情况等统计数据。

stat Engine（显示帧数，时间，三角面数等）

函数解释：

在 RflySim3D 中显示引擎相关的统计信息。通过使用该命令，可以实时监测引擎的性能表现和其他相关信息。

参数解释：

无参数。该命令只用于显示引擎的统计信息，不需要额外的参数进行指定。

操作示例：

```
stat Engine
```

上述示例中，启用引擎相关的统计信息显示。RflySim3D 中将显示包括帧率、时间、三角面数、贴图内存使用量和物理模拟的统计数据等。

stat scenerendering（显示 Drawcall）

函数解释：

在 RflySim3D 中显示 Draw Calls（绘制调用）。通过使用该命令，可以实时监测 RflySim3D 中场景渲染的性能表现以及其他相关信息。

参数解释：

无参数。该命令只用于显示场景渲染的统计信息，不需要额外的参数进行指定。

操作示例：

```
stat scenerendering
```

上述示例中，启用场景渲染的统计信息显示。RflySim3D 中将显示包括 Draw Calls（绘制调用）、渲染帧数、三角面数、渲染时间、物体数量、光照计算和阴影计算等统计数据。

r.setRes（设置显示分辨率）

函数解释：

设置游戏的显示分辨率。

```
r.setRes [Width]x[Height] [Fullscreen/Windowed]
```

参数解释：

- 分辨率参数：[宽度]x[高度]，用于指定游戏的显示分辨率。例如，1280x720 表示宽度为 1280 像素，高度为 720 像素。
- 显示模式参数：全屏模式（Fullscreen）或窗口模式（Windowed），用于指定游戏的显示模式。

操作示例：

设置分辨率为 1920x1080，全屏模式：

```
r.setRes 1920x1080 Fullscreen
```

设置分辨率为 1280x720，窗口模式：

```
r.setRes 1280x720 Windowed
```

注意

实际支持的分辨率和显示模式取决于显卡、显示器以及 RflySim3D 本身设置。

r.Streaming.PoolSize（修改纹理流送池大小）

函数解释：

设置纹理流送池的大小。纹理流送池是用于动态加载和卸载纹理资源的内存池，以优化性能和资源管理。

参数解释：

- 池大小（Pool Size）：指定纹理流送池的大小。以 MB 为单位来表示大小。

操作示例：

设置流式传输池大小为 4096 MB：

```
r.Streaming.PoolSize 4096MB
```

注意

默认分配 1G 显存, 不知道如何设定，给零值就行 `r.Streaming.PoolSize 0`, 当屏幕左上角出现 "TEXTURE STREAMING POOL OVER ***MiB BUDGET" 时就需要修改这个，显示提示时，会导致部分贴图加载 lod 变化，画面会模糊

5.2.2 LowGPU 场景使用

r.forcelod（LOD 点面数）

函数解释：

更改场景的 Level of Detail (LOD) 设置，降低点面数。LOD 是一种用于管理对象显示细节的技术，可以在不同距离和相机角度下使用不同的模型和纹理细节级别，

以平衡性能和视觉效果。

参数解释：

- LOD 级别：要强制应用的 LOD 级别。通常使用 0（维数最高的级别）到最大 LOD 级别的数值进行表示。

操作示例：

将所有 LOD 强制设置为 0，这将把所有对象的 LOD 强制设置为最高级别，即禁用 LOD。这意味着将始终以最高细节级别呈现对象，无论其使用的距离或相机角度如何。

```
r.ForceLOD 0
```

类似地，将 LOD 设置为 -1，这将关闭所有对象的 LOD，将使用最低的细节级别呈现对象。

```
r.ForceLOD -1
```

注意

在复杂场景中，修改 LOD 设置可能导致地形或其他对象的显示出现变化。

r.ScreenPercentage（渲染分辨率百分比）

函数解释：

控制渲染时使用的分辨率百分比。

参数解释：

- 分辨率百分比（Resolution Percentage）：指定分辨率百分比。通常使用整数值表示，例如 100 表示原始分辨率，50 表示降低到原始分辨率的一半，200 表示增加到原始分辨率的两倍。

操作示例：

将渲染分辨率设置为原始分辨率的 75%：

```
r.ScreenPercentage 75
```

r.ShadowQuality（阴影质量）

函数解释：

控制阴影的质量级别。

参数解释：

- 参数值及对应阴影质量：

  0: 禁用阴影

  1: 低质量阴影

  2: 中等质量阴影

3: 高质量阴影

操作示例：

禁用阴影：完全禁用游戏中的阴影以提高性能。

r.PostProcessAAQuality（抗锯齿质量）

函数解释：

控制后处理抗锯齿效果的质量级别。

参数解释：

- 参数值及对应抗锯齿质量：

  0: 禁用后处理抗锯齿

  1: 低质量后处理抗锯齿

  2: 中等质量后处理抗锯齿

  3: 高质量后处理抗锯齿

操作示例：

禁用后处理抗锯齿：这会完全禁用后处理抗锯齿效果，以提高性能。但图像可能会出现锯齿或边缘不平滑的现象。

```
r.PostProcessAAQuality 0
```

r.SetNearClipPlance（视口近裁剪面）

函数解释：

设置摄像机的近裁剪平面（Near Clip Plane）。

参数解释：

- 近裁剪平面距离：以摄像机空间中的单位来表示，通常为厘米。

操作示例：

以下是将近裁剪平面设置为 10 个单位距离的示例：

```
r.SetNearClipPlane 10
```

r.MipMapLoDBias（纹理层级偏差）

函数解释：

调整纹理的 MipMap 层级偏差（LOD Bias）。MipMap 是一种用于提供不同细节级别的纹理预渲染的技术。

参数解释：

- LOD Bias 值：表示纹理的 MipMap 层级偏差。较高的 LOD Bias 值表示使用更低分辨率的 MipMap 层级，降低纹理显示的细节级别。

操作示例：

以下是将纹理的 MipMap 层级偏差设置为 2 的示例：

```
r.MipMapLODBias 2
```

sg.TextureQuality（纹理质量）

函数解释：

纹理质量级别决定了纹理的分辨率和细节，对游戏的视觉效果和性能都有影响。

参数解释：

- 用于设置纹理质量级别。

操作示例：

以下是将纹理质量级别设置为最低（级别 0）的示例：

```
sg.TextureQuality 0
```

sg.PostProcessQuality（后处理质量）

函数解释：

后期处理（Postprocessing）是在渲染完场景之后对图像进行处理的过程，可以对

图像进行颜色校正、模糊、光照效果等调整，以增强场景的视觉效果。

参数解释：

- 质量级别：决定了后期处理效果的细节和计算复杂度。

操作示例：

以下是将后期处理质量级别设置为最低（级别 0）的示例：

```
sg.PostProcessQuality 0
```

foliage.MaxTrianglesToRender（植被类模型三角面渲染数量）

函数解释：

植被类型的模型三角面渲染数量（Foliage Triangles to Render）是指在仿真场景中绘制植被类型（foliage）模型时，限制三角面数量的设置。

参数解释：

- 用于设置允许绘制的植被模型的三角面数量。

操作示例：

以下是将植被模型的三角面数量设置为最低（0）的示例：

```
foliage.MaxTrianglesToRender 0
```

6. 程序启动脚本接口

对应平台例程演示效果 6.1：0.ApiExps/e3_InitAPI/1.TXTAllCrtlScript/Readme.pdf

对应平台例程演示效果 6.2~6.4：0.ApiExps/e3_InitAPI/2.TXTMapCrtlScript/Readme.pdf

UE 引擎的 FFileHelper 类可以对 txt 文本文件进行读写操作。在确定路径下 txt 文件存在后，可以调用 FFileHelper::LoadFileToString 函数将文件内容读取到一个 FString 对象中，然后对内容进行处理或显示；要读取文本文件的每一行，可以使用 FFileHelper::LoadFileToStringArray 函数。要写入文本文件，可以使用 FFileHelper::SaveStringToFile 函数；如果要将字符串数组写入文本文件并自动换行，可以使用 FFileHelper::SaveStringArrayToFile 函数。在如下链接中有一个简单用例：【UE4 C++】读写 Text 文件 FFileHelper

RflySim3D 利用此功能，开发了如下图的四个 txt 脚本接口

6.1 程序开机启动（RflySim3D.txt）

RflySim3D 能自动识别指定目录下的 txt 脚本，在平台 PX4PSP\RflySim3D 目录下创建一个名为“RflySim3D.txt”脚本，在打开 RflySim3D 时，就会自动执行脚本中的一些控制台命令，每条命令需独占一行。

6.2 切换地图启动（LowGPU.txt）

在平台 PX4PSP\RflySim3D 目录下创建一个名为“地图名称.txt”脚本，同样可以预存一些控制台命令，在 RflySim3D 中切换到该地图时，就会执行对应命令。这里的 LowGPU.txt 中预存的命令主要用于提高渲染效率以适应低性能电脑的仿真需求，详见 5.2.2 LowGPU 场景使用

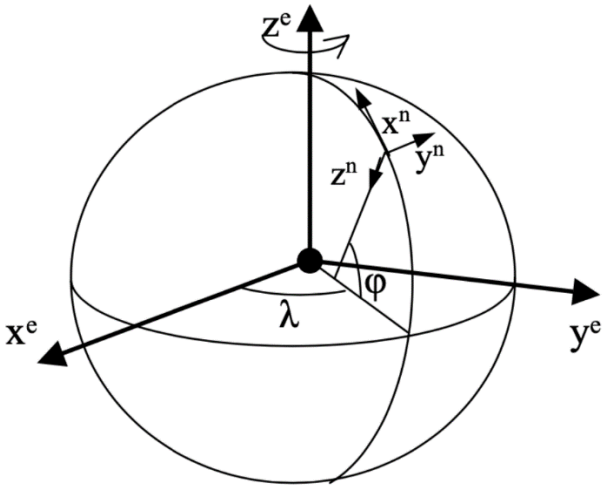
6.3 击中物体日志（ClickLog.txt）

ClickLog.txt 记录了从打开 RflySIm3D 开始（重启 RflySim3D 后之前的击中点信息会消失），用户鼠标双击的值。以在 RflySim3D 中击中 3DDisplay 中地面上一点及一架四旋翼后生成的日志文件为例：

首行记录为最近一次击中物体的当前时间、单位、坐标系以及地图名称信息

```
2023.09.18-16.58.33, Unit:m, Frame: NED, MapName:3DDisplay
```

- 当前时间（精确到秒）：2023.09.18-16.58.33
- 单位（Unit）：长度 m，弧度 rad
- 当地水平面坐标系（Frame）：NED（北东地）



x 轴指向北， y 轴指向东， z 轴指向地

绕 x 轴转动，称为 roll 角

绕 y 轴转动，称为 pitch 角

绕 z 轴转动，成为 yaw 角

对应的 IMU 载体坐标系是前右下

欧拉角旋转顺序：z-y-x

- 地图名称（MapName）：3DDisplay

接下来依次为从前到后所击中点的信息

```
    Landscape_1; Click Point: [1.289, 0.486, -8.103]; UEOri Pos: [-504.000, -504.000, -1.000]; BoxOri Pos: [-24696.000, -24696.000, -48.082]; Size: [51408.000, 51408.000, 103.836]
```

- 击中对象编号：这里的对象是地面 Landscape_1
- 击中点坐标（Click Point）：该点在世界场景中的位置[1.289, 0.486, -8.103]
- 击中对象在 UE 系统中的真实坐标（UEOri Pos）：这里是地面对象 Landscape_1 在 UE 系统中的坐标[-504.000, -504.000, -1.000]
- 击中对象包围盒的中心点（BoxOri Pos）：这里是地面对象 Landscape_1 的包围盒体的中心点在 UE 系统中的坐标[-24696.000, -24696.000, -48.082]
- 击中对象包围盒上的一个顶点（Size）：这里是地面对象 Landscape_1 的包围盒体的一个顶点在 UE 系统中的坐标[51408.000, 51408.000, 103.836]

```
    Copter_1000; ID: 1000; PosE: [-5.746, 0.601, -7.771]; AngEuler: [0.000, 0.000, 0.330], CenterHeight: 16.00
    Copter_1000; Click Point: [-5.915, 0.524, -7.972]; UEOri Pos: [-5.746, 0.601, -7.931]; BoxOri Pos: [0.002, 0.006, 0.016]; Size: [0.549, 0.535, 0.294]
```

- 击中对象编号：这里的对象是自行创建的，所以是 Copter_***的命名规则，此处为 Copter_1000
- 击中对象 ID：1000
- 发送的位置（PosE）：击中对象用于传输的位置信息
- 发送的姿态角（AngEuler）：击中对象用于传输的姿态角信息

- 发送的物体中心到地面高度（CenterHeight）：击中对象用于传输的高度信息
- 击中点坐标（Click Point）：该点在世界场景中的位置[-5.915, 0.524, -7.972]
- 击中对象在 UE 系统中的真实坐标（UEOri Pos）：这里是四旋翼对象 Copter_1000 在 UE 系统中的坐标[-5.746, 0.601, -7.931]
- 击中对象包围盒的中心点（BoxOri Pos）：这里是 Copter_1000 的包围盒体的中心点在 UE 系统中的坐标[0.002, 0.006, 0.016]
- 击中对象包围盒上的一个顶点（Size）：这里是 Copter_1000 的包围盒体的一个顶点在 UE 系统中的坐标[0.549, 0.535, 0.294]

6.4 创建物体日志（CreateLog.txt）

CreateLog.txt 记录按下 O 键，创建的物体的信息（重启 RflySim3D 后之前的创建物体信息会消失），格式和使用 RflyLoad3DFile 命令加载的 txt 相同，故将 CreateLog.txt 改名为***.txt，再次打开 RflySim3D 并切换到***地图，可以看到之前用 O 键和 Rfly**命令创建的场景，会被自动加载。以在 RflySim3D 中创建一架四旋翼后生成的日志文件为例：

首行记录为最近一次创建或移动物体的当前时间、单位、坐标系以及地图名称信息

```
2023.09.19-10.09.25, Unit:m or rad, Frame: NED, MapName:3Ddisplay
```

- 当前时间（精确到秒）：2023.09.19-10.09.25
- 单位（Unit）：长度 m，弧度 rad
- 当地水平面坐标系（Frame）：NED（北东地）
- 地图名称（MapName）：3DDisplay

如下为创建物体的格式和信息

```
CMD:CreateVehicle|Rfly***, copterID, vehicleType, PosE[0], PosE[1], PosE[2], AngEuler[0], AngEuler[1], AngEuler[2]
CreateVehicle, 1000, 3, -10.45, -1.77, -6.97, 0.00, 0.00, -0.83
```

- 创建物体 ID（copterID）：1000
- 创建物体类型（vehicleType）：3（对应四旋翼）
- 创建物体位置坐标（PosE[0], PosE[1], PosE[2]）：该四旋翼在世界场景中的位置-10.45, -1.77, -6.97
- 创建物体姿态角（AngEuler[0], AngEuler[1], AngEuler[2]）：该四旋翼的姿态角 0.00, 0.00, -0.83

注意：

vehicleType 是一个控制三维物体的变量（具体样式由 XML 中的模型类别 classID 和样式 DisplayOrder 决定）.

- vehicleType=3 对应四旋翼。RflySim3D 有多重四旋翼样式可选，输入 ve

hicleType =1003, 2003, 3003 之类来选择具体样式

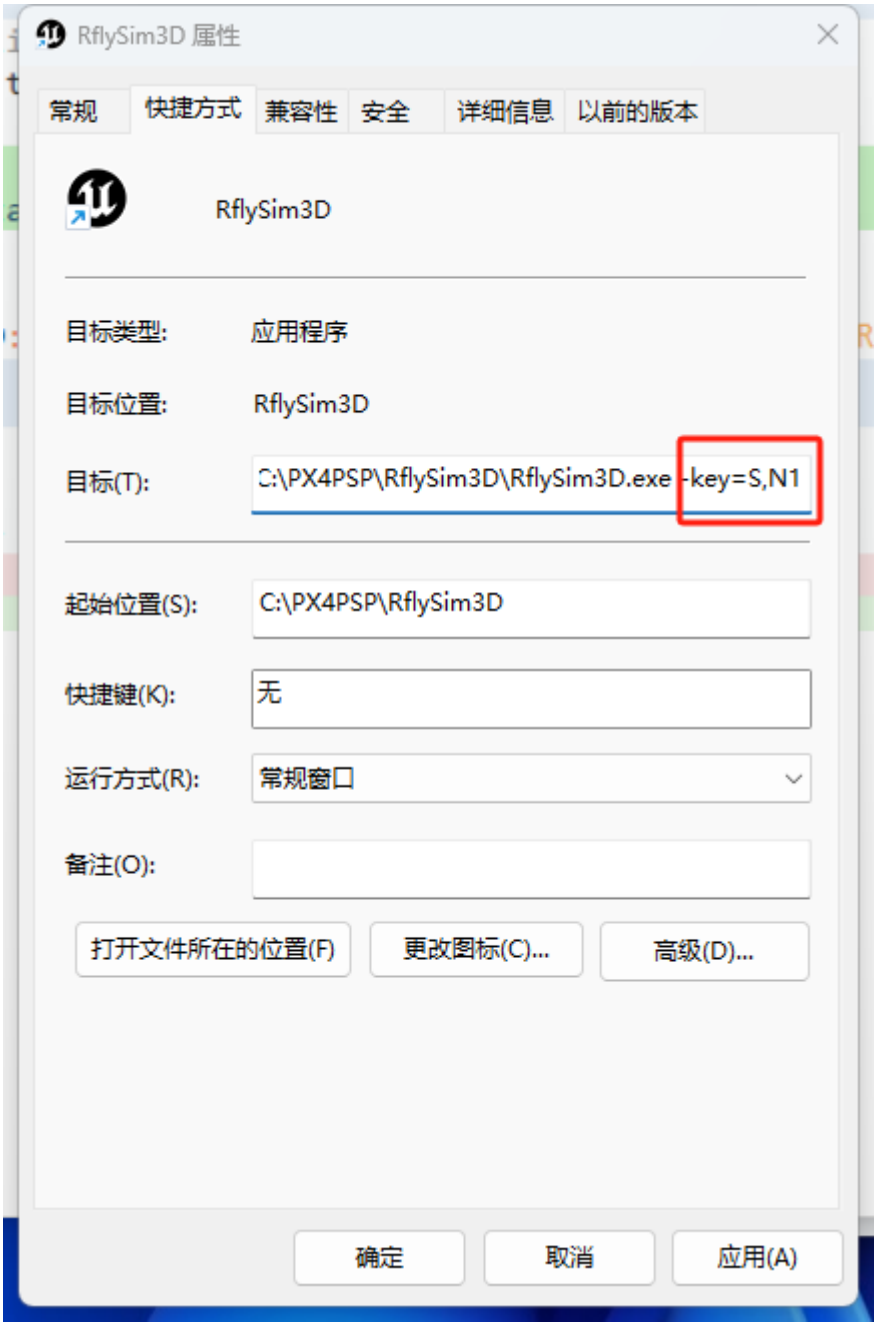
- vehicleType =5 或 6 对应一个六旋翼
- vehicleType =30 对应人，其中 3030 表示样式为 3 的人
- vehicleType =100 对应固定翼.
- vehicleType=150 对应环、方框等靶标
- vehicleType=60 对应发光的等，用于灯光秀显示

6.5 程序启动参数配置命令

BAT 脚本配置快捷键

```
    REM UE4Path
    cd %PSP PATH% RflySimUE5tasklist|find /i "RflySim3D.exe" start %PSPPATH%\RflySimUE5\RflyS
im3D.exe -key=S,N1
    choice /t 5 /d y /n >nul
```

直接配置快捷方式



7. 场景导入接口

对应平台例程演示效果 7.1：UE4 默认场景导入

7.1 普通导入（由 UE 导入 RflySim）

在 UE 项目中已处理完成的场景有三种：UE 默认场景、虚幻商城购买的场景以及从其它 UE 项目中迁移来的场景。这些场景直接在 UE 中烘焙完成即可导入 RflySim3D，而 RflySim3D 中完整的仿真场景需要如下三部分信息。

7.1.1 场景文件(”****.umap”)

需要将烘焙完成后“【UE 项目主文件夹】\Saved\Cooked\WindowsNoEditor\【工程名】\

Content”目录下的场景文件拷贝到 C:\PX4PSP\RflySim3D\RflySim3D\Content 目录下。

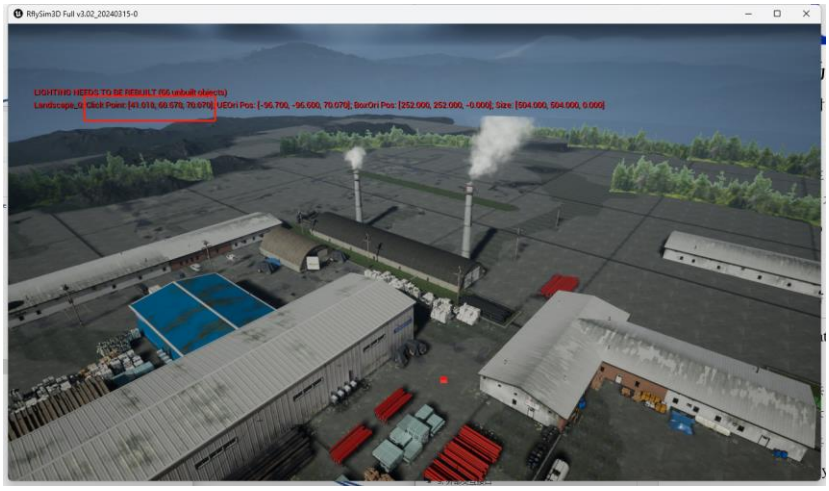
注意：

烘焙完拷贝出来后不能再修改得到文件和文件夹的名字，也不能在\RflySim3D\Content 目录下新建文件夹。重命名只能在 UE 编辑器中进行。

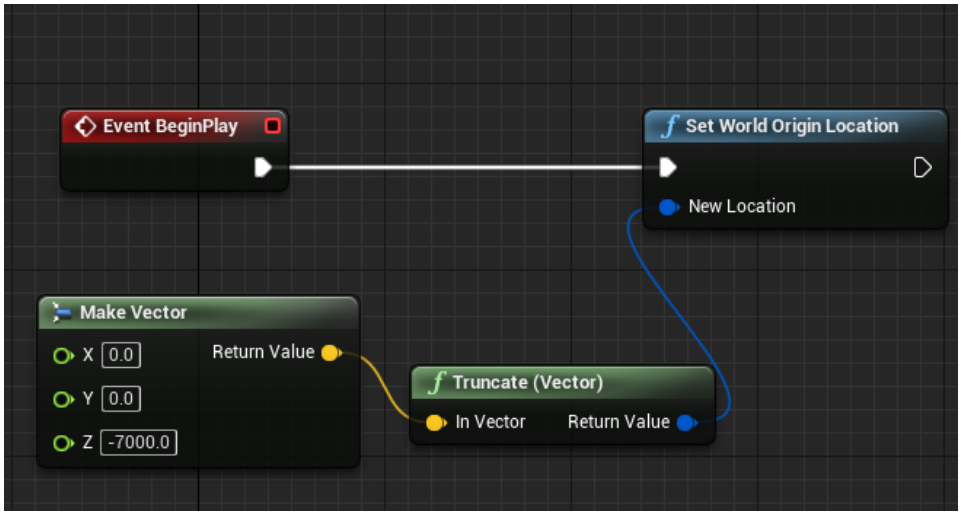
每个”****.umap”（烘焙前后有区别）地图文件对应了一个独立的三维场景，RflySim 3D/RflySimUE5 会自动扫描 PX4PSP\RflySim3D\RflySim3D\Content 目录下所有的.umap。

场景中水平面以下部分的高度不能超过 50m（这是由 CopterSim 中的 dll 模型初始点参考海拔设置的，由*init.m 文件中的 ModelParam_envAlitude 参数指定），为了确保场景最低点在水平面上方，应该将场景中最低点设为场景参考点。下面以 OldFactory 场景为例说明这一过程：

先将场景导入 RflySim3D，在 RflySim3D 中目测该场景最低点，双击读取该点高程信息，这里说明该场景最低点在水平面下方 70m 处。



因此在 UE 中打开该场景的关卡蓝图，如下将场景最低点（水平面下方 7000cm 处）设为参考点，相当于把场景整体上移 70m，保存关卡蓝图，重新把场景打包导入 RflySim3D。



7.1.2 地形高程信息(“****.png”)

场景导入前，在 UE 编辑器中观测并记录场景的大致范围（单位为厘米）；烘焙场景并导入 RflySim3D 后，在 RflySim3D 中利用比”****.umap”场景地形略大的范围（单位为米）使用 RflyScanTerrainH（扫描地形）命令，这将在 PX4PSP\RflySim3D 目录找到对应“****.png”和“****.txt”文件，分别存入了该场景的高程信息和地形校准及范围。将这两个文件拷贝到 PX4PSP\CopterSim\external\map 中即可完成导入。

注意：

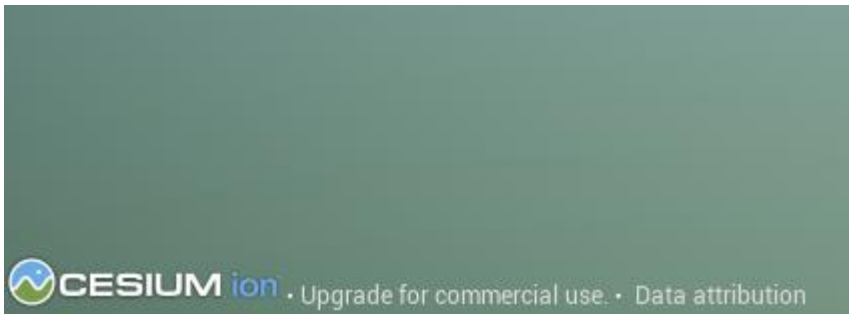
png 地形文件实际上是以图片形式存储的二维矩阵，包含了场景的高程图。以 png 格式存储矩阵能够很好的实现高程矩阵的压缩，便于节省空间。

png 的高程文件并不包含坐标原点、缩放尺度、场景范围等信息，因此需要一个校正文件，RflySim 平台采用 txt 格式输入 9 维数组或 12 维数组（多出经纬高地理基准信息）传入校正信息。

7.1.3 地形校准数据(“****.txt”)

平台中 txt 校正文件存储的是右上角三维坐标点（xy 全为正，z 向上为正）、左下角三维坐标点（xy 全为负，z 向上为正）、第 3 点三维坐标点，单位均为厘米。前两个点的目的是为了确认地形的范围和中心坐标，第 3 点坐标可自行选取，理论上需要尽量在高度上与前两个点有落差，用于校正高度尺度。

除此之外，当在 RflySim3D 中使用 cesium 全球大场景（仅个人高级版以上）时，在 txt 中，还可以上述 9 维数据之后输入三维 GPS 地形数据，按纬度、经度、高度顺序加入，可配置 QGroundControl 中对应的显示基准坐标。对应例程见 2.AdvExps\e3_CusGIS\1.ObliModelMap\Readme.pdf



7.2 Datasmith 导入

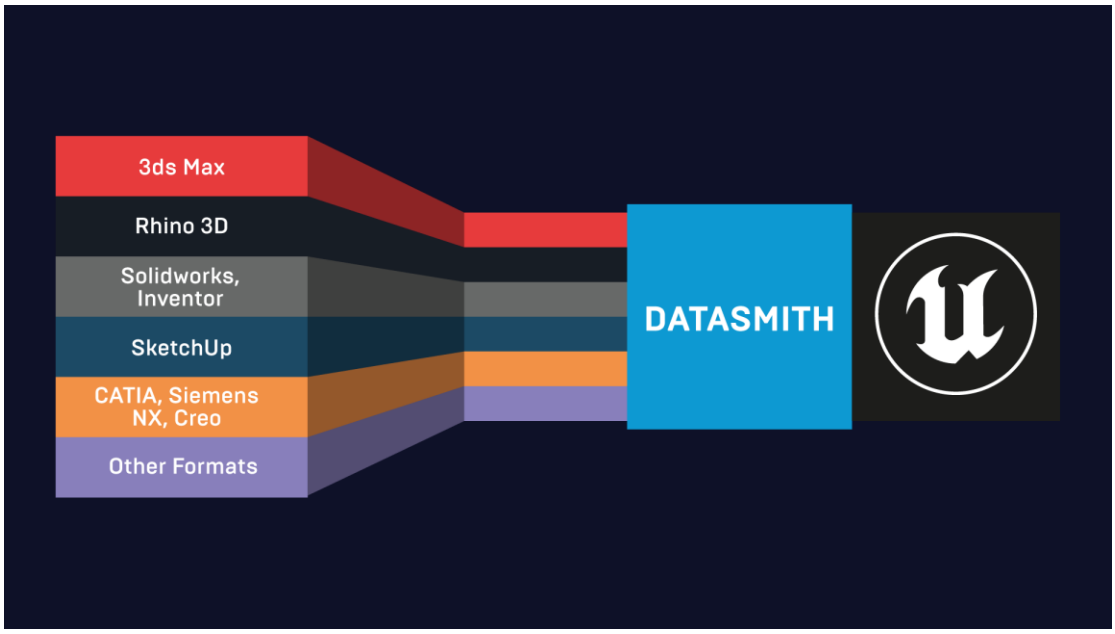
Datasmith 是 Epic Games 公司为虚幻引擎开发的一系列插件集合，用于将各种设计数据（如 CAD、3D 建模软件等）导入虚幻引擎中，以利用其实时渲染和可视化功能创建交互式三维场景。详细使用教程见：Datasmith 概述 | 虚幻引擎文档 (unrealengine.com)

相比于普通场景导入，利用 Datasmith 插件可以支持多种不同的文件格式，包括但不限

于 FBX、DWG、3DS、SketchUp、Rhino、CATIA、SolidWorks 等；Datasmith 保留了导入数据的准确性，包括几何形状、材质、光照、相机、动画等信息；Datasmith 保留原始设计数据中的层次结构和组织方式，以便更容易管理和编辑导入的场景；Datasmith 提供自动化工具，可用于自动处理导入数据，例如自动设置材质、处理 LOD（细节层次）、创建碰撞体等。总而言之，使用 Datasmith 插件可以导入更复杂的场景和模型，且使得导入效果更精确。

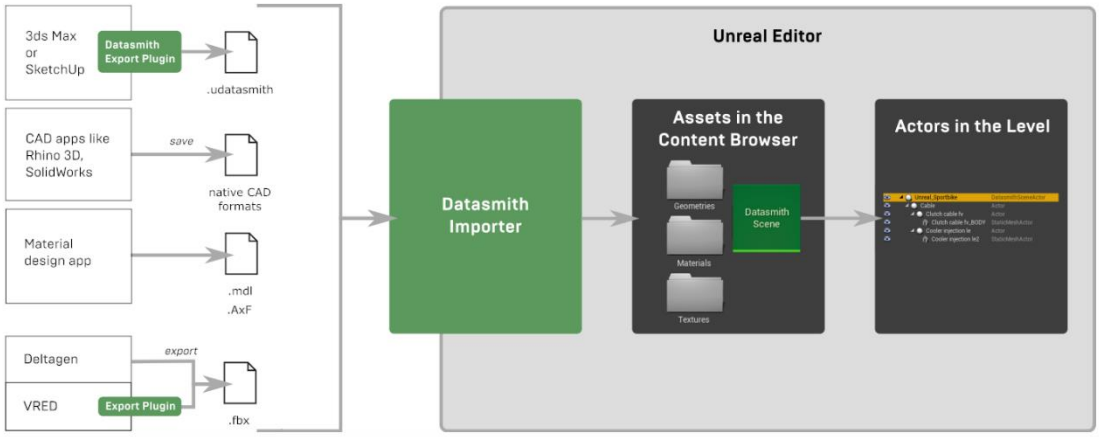
7.2.1 Datasmith For Unreal（导入 UE）

为各种三维处理软件安装对应版本的.udatasmith 文件导出器插件即可导出.udatasmith 文件，所有.udatasmith 文件导出器见 [Datasmith 导出器插件 - Unreal Engine]



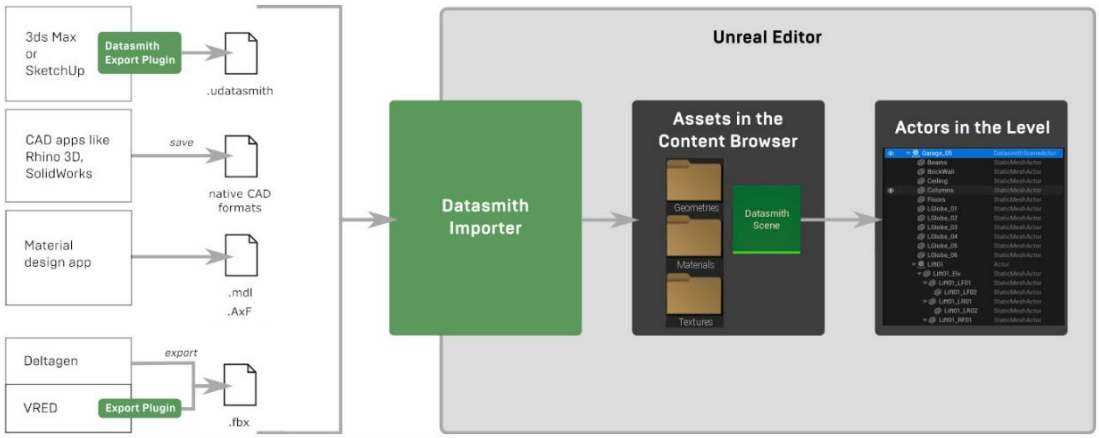
主要工作流程如下：

UE4



在 UE4 中启用如下插件即可将.udatasmith 格式文件导入 UE4。

UE5

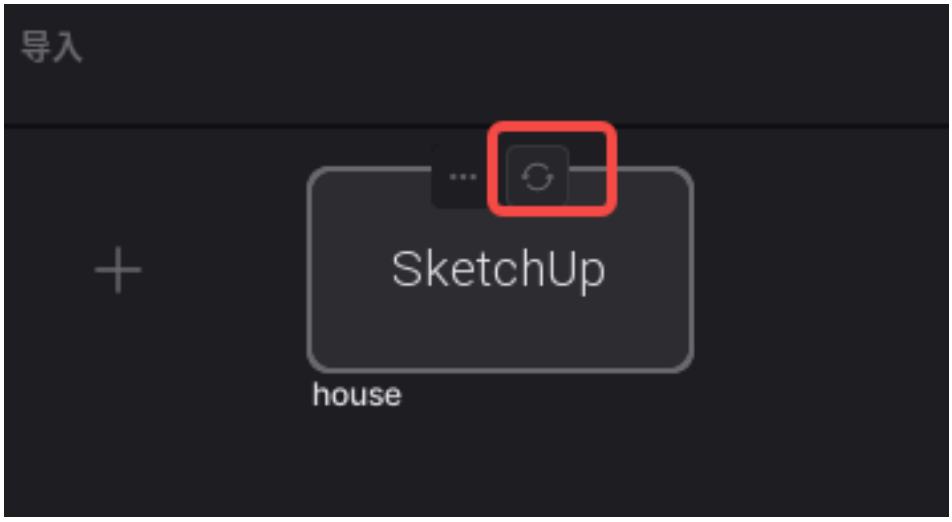


在 UE5 中启用如下插件即可将.udatasmith 格式文件导入 UE5。

7.2.2 Datasmith For Twinmotion（导入 Twinmotion）

只要为三维处理软件下载并安装了对应版本的导出器插件，使用该软件处理模型时就自动支持在 Twinmotion 中实时预览还未编辑完成的模型场景，即一边完成初步粗模设计，一边预览后期材质处理之后的效果。以 sketchup 为例，下载并安装对应版本的导出器插件：[面向 SketchUp Pro 的 Twinmotion Datasmith 导出器插件 - Twinmotion]

在 sketchup 中打开对应场景，然后在 Twinmotion 中选择直链导入并刷新链接。

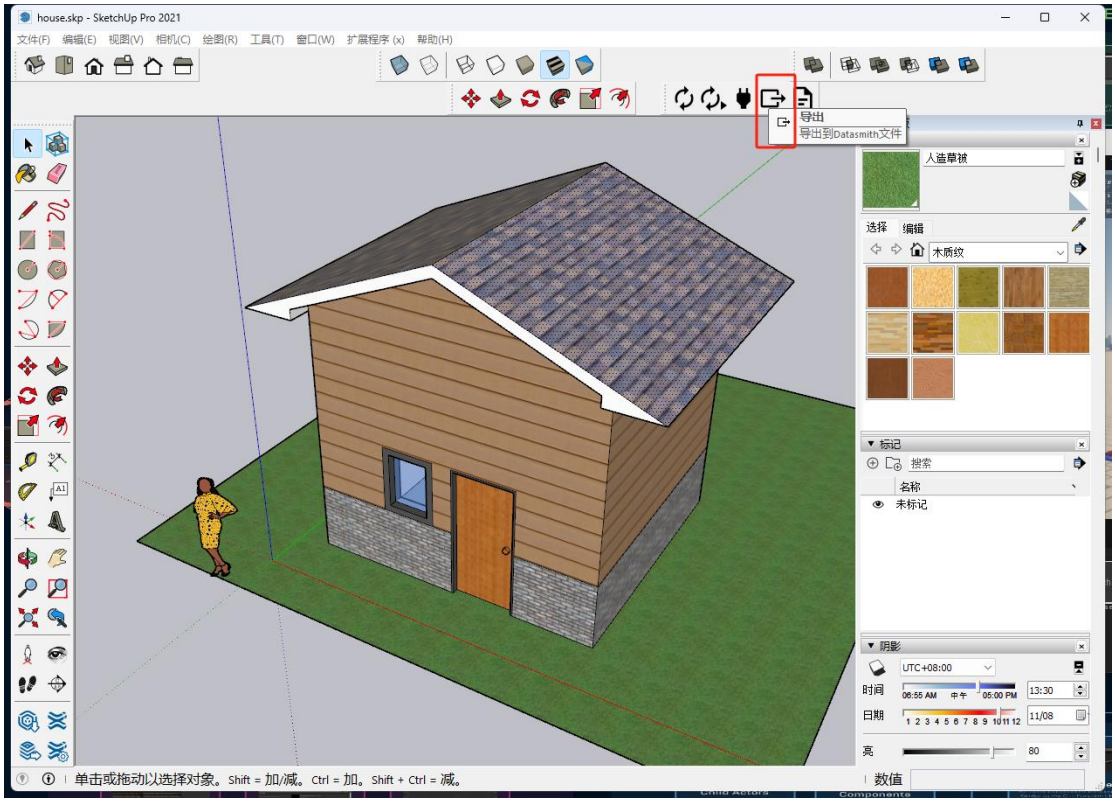


即可在 twinmotion 视口中看到该场景

7.2.3 SKETCHUP PRO 导出器

下载并安装对应版本的导出器插件：面向 SketchUp Pro 的 Twinmotion Datasmith 导出器插件 - Twinmotion，安装时不能开启 SketchUp

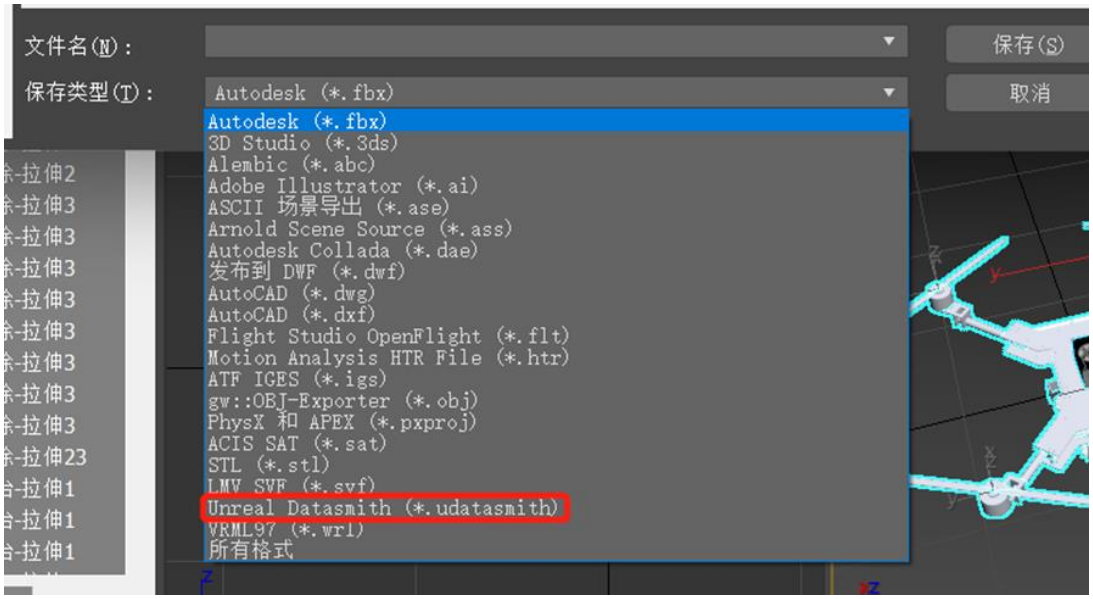
在 SketchUp 中将对应模型导出为.udatasmith 格式



7.2.4 AUTODESK 3DS MAX 导出器

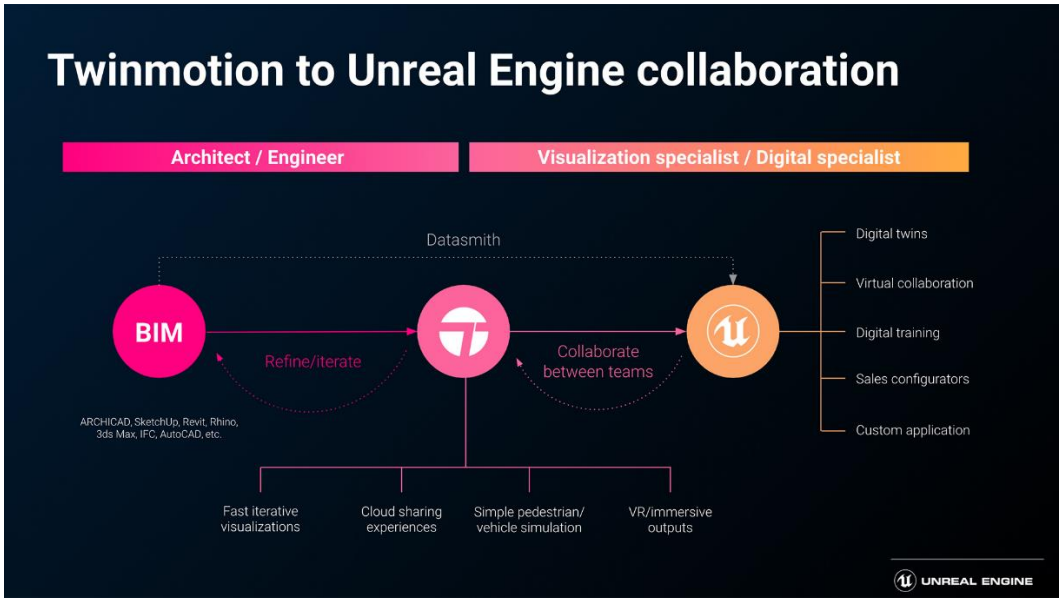
下载并安装对应版本的导出器插件：面向 3ds Max 的 Twinmotion Datasmith 导出器插件 - Twinmotion，安装时不能开启 3DsMax

在 3DsMax 中将对应模型导出为.udatasmith 格式：文件->导出->文件类型会出现 udatasmith 文件类型，选择导出



7.3 Twinmotion 导入

Twinmotion 支持所有主流 CAD、BIM 和建模解决方案的文件格式（Twinmotion 插件 - Twinmotion），并可与其中的大多数实现一键同步。此外，也可以在虚幻引擎中进一步开发 Twinmotion 项目。



7.3.1 Twinmotion 导入 UE4

Twinmotion 版本

UE4 安装 2022.2.3 之前的 Twinmotion 版本，包括 2022.2.3。

插件

UE4 安装“Datasmith Twinmotion 导入器”和“面向虚幻引擎的 Twinmotion 内容”插件

导入方法

UE4 只支持 Twinmotion 2022.2.3 之前导出的.tm 文件

如使用 UE4，需启用如下两个插件

其中，“Datasmith Twinmotion Inporter”会解析.tm 文件；“Twinmotion Content”包含 Twinmotion 的各种材质库以及模型库，会按照材质路径为场景中的模型生成贴图。

导入 RflySim3D 方法

烘焙完成后，需要将烘焙文件需要拷贝至 PX4PSP\RflySim3D 下的三个文件

[项目名] -> Saved -> Cooked -> WindowsNoEditor -> [项目名]-> Content 拷贝至 PX4 PSP -> RflySim3D -> RflySim3D -> Content 下.

[项目名] -> Saved -> Cooked -> WindowsNoEditor -> Engine -> Plugins -> Marketplac e -> TMtoUnrealContent 拷贝至 PX4PSP -> RflySim3D -> Engine -> Plugins -> Marke tplace -> TMtoUnrealContent 下.

[项目名] -> Saved -> Cooked -> WindowsNoEditor -> Engine -> Plugins -> Enterprise -> DatasmithContent 拷贝至 PX4PSP -> RflySim3D -> Engine -> Plugins -> Enterprise -> DatasmithContent 下.

7.3.2 Twinmotion 导入 UE5

Twinmotion 版本

UE5 安装 2023.3.1 之后的版本。

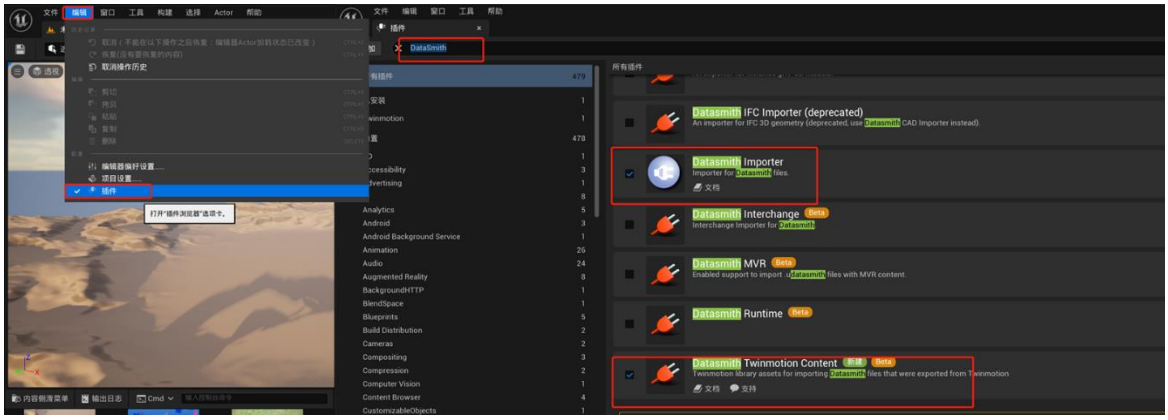
插件

UE5 安装“面向虚幻引擎的 Datasmith Twinmotion 内容”插件

导入方法

UE5 需要使用 2023.1 Preview 1 之后的版本，该版本 Twinmotion 支持 Datasmith 文件导出。

如果使用的是 UE5，可使用 Datasmith 导入的方式来将 Twinmotion 文件导入 UE5 中。（需启用如下插件“Datasmith Inporter”及“Datasmith Twinmotion Content”）



其中，“Datasmith Inporter”会解析.udatasmith 文件；“Datasmith Twinmotion Content”包含 Twinmotion 的各种材质函数，会计算出场景中各类物体表面的材质属性并赋予不同的材质实例。

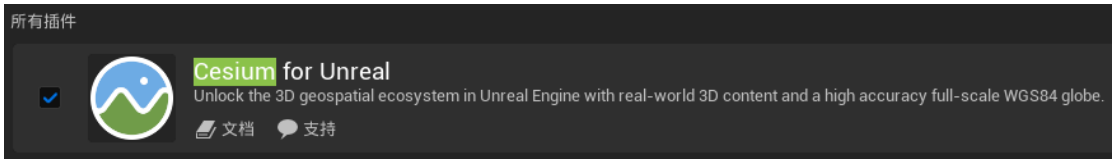
导入 RflySimUE5 方法

[项目名] -> Saved -> Cooked -> WindowsNoEditor -> [项目名]-> Content 拷贝至 PX4 PSP -> RflySimUE5 -> RflySimUE5 -> Content 下.

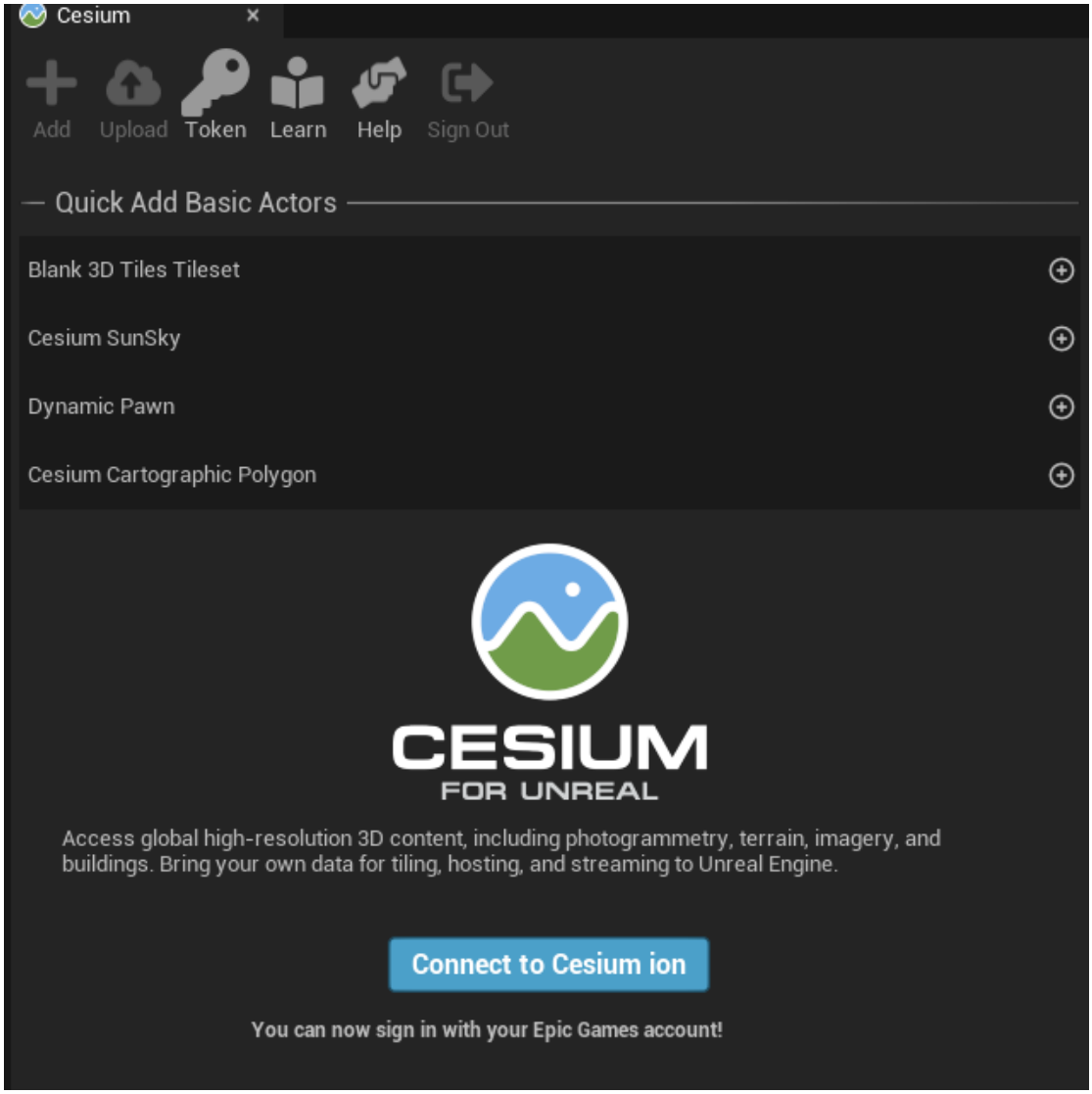
7.4 Cesium 导入

7.4.1 Cesium For Unreal

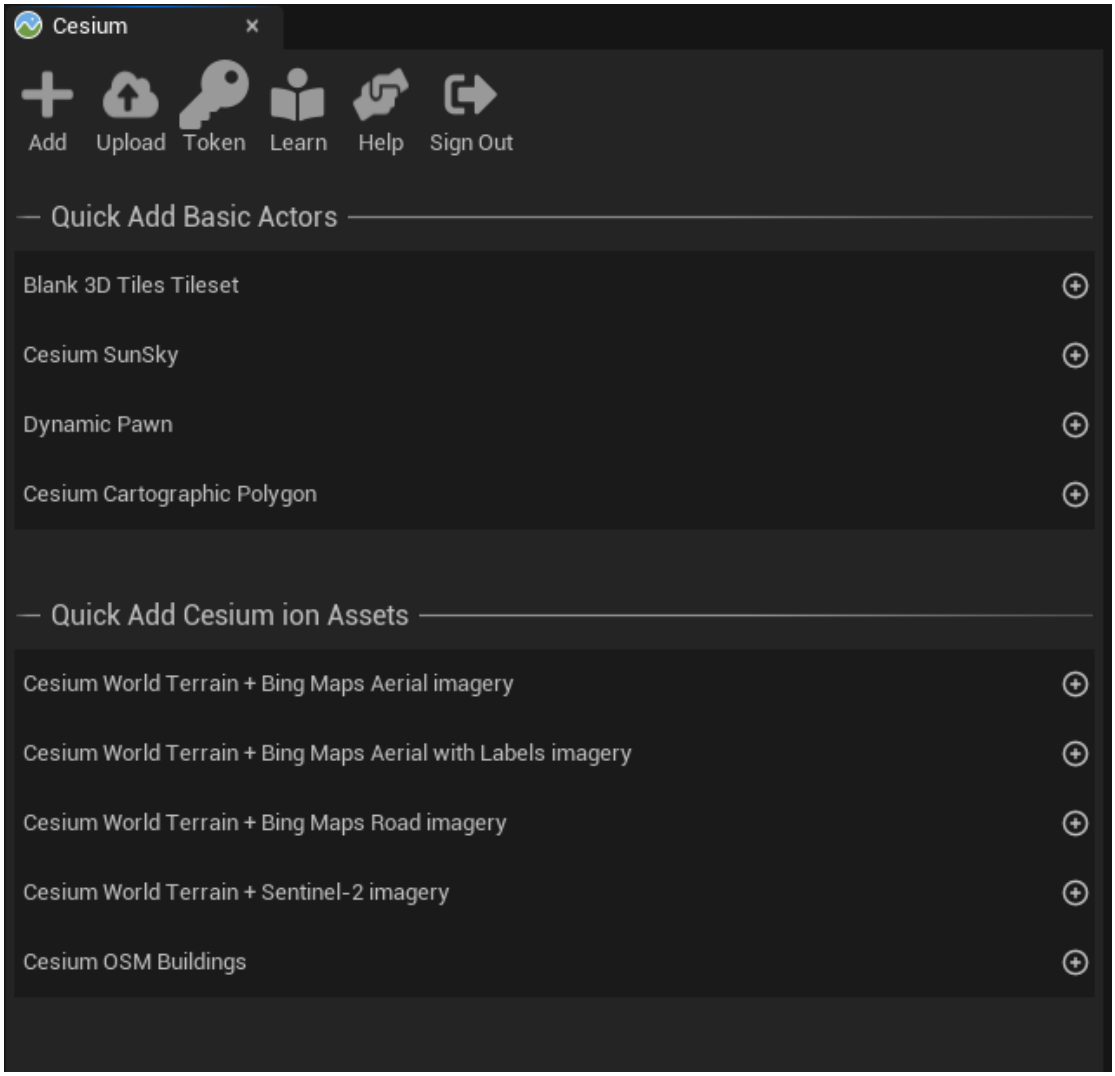
启用如下插件



连接到 Cesium ion，登录 Epic 账号



Cesium for unreal 插件成功启用



8. 模型导入接口

8.1 对应例程效果演示：[1.BasicExps/e1_CusLoadDroneyeeX680/Readme.pdf]

8.1 XML 规则

XML（可扩展标记语言）是一种用于描述数据的标记语言，它类似 HTML，但是其中的标记是可以自己定义的，由于这种灵活性和可扩展性，常常被作为程序的配置文件，XML 文件的格式类似 html 也是由各种标签组合而成，每个标签每个标签由两对<>符号围起来<***></***>，在标签内部存放着字符串、数字或其他标签。而 RflySim3D 也使用它来配置无人机的属性。在 RflySim3D 中，每个 Copter 的三维模型都会有一个 XML 格式的配置文件提供与其相关的信息。

F450_Default.xml

```
<?xml version="1.0"?>
<vehicle>
    <ClassID>3</ClassID>
    <DisplayOrder>1000</DisplayOrder>
    <Name>F450_Default</Name>
```

```
        <Scale>
            <x>1</x>
            <y>1</y>
            <z>1</z>
        </Scale>
        <AngEulerDeg>
            <roll>0</roll>
            <pitch>0</pitch>
            <yaw>0</yaw>
        </AngEulerDeg>
        <body>
            < ModelType>0</ModelType>
            <MeshPath>/Rfly3DSimPlugin/copter/F450Body</MeshPath>
            <MaterialPath></MaterialPath>
            <AnimationPath></AnimationPath>
            <CenterHeightAboveGroundCm>16.3</CenterHeightAboveGroundCm>
            <NumberHeigthAboveCenterCm>20</NumberHeigthAboveCenterCm>
        </body>
        <ActuatorList>
            <Actuator>
                <MeshPath>/Rfly3DSimPlugin/copter/PropellersCCW</MeshPath>
                <MaterialPath></MaterialPath>
                <RelativePosToBodyCm>
                    <x>15.798</x>
                    <y>16.167</y>
                    <z>4.5</z>
                </RelativePosToBodyCm>
                <RelativeAngEulerToBodyDeg>
                    <roll>0</roll>
                    <pitch>0</pitch>
                    <yaw>0</yaw>
                </RelativeAngEulerToBodyDeg>
                <RotationAxisVectorToBody>
                    <x>0</x>
                    <y>0</y>
                    <z>1</z>
                </RotationAxisVectorToBody>
                <RotationModeSpinOrDefect>0</RotationModeSpinOrDefect>
            </Actuator>
            <Actuator>
                <MeshPath>/Rfly3DSimPlugin/copter/PropellersCCW</MeshPath>
                <MaterialPath></MaterialPath>
                <RelativePosToBodyCm>
                    <x>-15.634</x>
                    <y>-16.299</y>
                    <z>4.5</z>
                </RelativePosToBodyCm>
                <RelativeAngEulerToBodyDeg>
                    <roll>0</roll>
                    <pitch>0</pitch>
                    <yaw>0</yaw>
                </RelativeAngEulerToBodyDeg>
                <RotationAxisVectorToBody>
                    <x>0</x>
                    <y>0</y>
```

```
                <z>1</z>
            </RotationAxisVectorToBody>
            <RotationModeSpinOrDefect>0</RotationModeSpinOrDefect>
        </Actuator>
        <Actuator>
            <MeshPath>/Rfly3DSimPlugin/copter/PropellersCW</MeshPath>
            <MaterialPath></MaterialPath>
            <RelativePosToBodyCm>
                <x>16.299</x>
                <y>-15.634</y>
                <z>4.5</z>
            </RelativePosToBodyCm>
            <RelativeAngEulerToBodyDeg>
                <roll>0</roll>
                <pitch>0</pitch>
                <yaw>0</yaw>
            </RelativeAngEulerToBodyDeg>
            <RotationAxisVectorToBody>
                <x>0</x>
                <y>0</y>
                <z>1</z>
            </RotationAxisVectorToBody>
            <RotationModeSpinOrDefect>0</RotationModeSpinOrDefect>
        </Actuator>
        <Actuator>
            <MeshPath>/Rfly3DSimPlugin/copter/PropellersCW</MeshPath>
            <MaterialPath></MaterialPath>
            <RelativePosToBodyCm>
                <x>-16.299</x>
                <y>15.634</y>
                <z>4.5</z>
            </RelativePosToBodyCm>
            <RelativeAngEulerToBodyDeg>
                <roll>0</roll>
                <pitch>0</pitch>
                <yaw>0</yaw>
            </RelativeAngEulerToBodyDeg>
            <RotationAxisVectorToBody>
                <x>0</x>
                <y>0</y>
                <z>1</z>
            </RotationAxisVectorToBody>
            <RotationModeSpinOrDefect>0</RotationModeSpinOrDefect>
        </Actuator>
    </ActuatorList>
    <OnboardCameras>
        <camera>
            <name>Chase_Camera</name>
            <RelativePosToBodyCm>
                <x>-70</x>
                <y>0</y>
                <z>5</z>
            </RelativePosToBodyCm>
            <RelativeAngEulerToBodyDeg>
                <roll>0</roll>
```

```
            <pitch>0</pitch>
            <yaw>0</yaw>
        </RelativeAngEulerToBodyDeg>
    </camera>
    <camera>
        <name>Front_Camera</name>
        <RelativePosToBodyCm>
            <x>10</x>
            <y>0</y>
            <z>0</z>
        </RelativePosToBodyCm>
        <RelativeAngEulerToBodyDeg>
            <roll>0</roll>
            <pitch>0</pitch>
            <yaw>0</yaw>
        </RelativeAngEulerToBodyDeg>
    </camera>
    <camera>
        <name>Back_Camera</name>
        <RelativePosToBodyCm>
            <x>-10</x>
            <y>0</y>
            <z>0</z>
        </RelativePosToBodyCm>
        <RelativeAngEulerToBodyDeg>
            <roll>0</roll>
            <pitch>0</pitch>
            <yaw>180</yaw>
        </RelativeAngEulerToBodyDeg>
    </camera>
    <camera>
        <name>Right_Camera</name>
        <RelativePosToBodyCm>
            <x>0</x>
            <y>10</y>
            <z>0</z>
        </RelativePosToBodyCm>
        <RelativeAngEulerToBodyDeg>
            <roll>0</roll>
            <pitch>0</pitch>
            <yaw>90</yaw>
        </RelativeAngEulerToBodyDeg>
    </camera>
    <camera>
        <name>Left_Camera</name>
        <RelativePosToBodyCm>
            <x>0</x>
            <y>-10</y>
            <z>0</z>
        </RelativePosToBodyCm>
        <RelativeAngEulerToBodyDeg>
            <roll>0</roll>
            <pitch>0</pitch>
            <yaw>-90</yaw>
        </RelativeAngEulerToBodyDeg>
```

```
            </camera>
            <camera>
                <name>Down_Camera</name>
                <RelativePosToBodyCm>
                    <x>0</x>
                    <y>0</y>
                    <z>-10</z>
                </RelativePosToBodyCm>
                <RelativeAngEulerToBodyDeg>
                    <roll>0</roll>
                    <pitch>-90</pitch>
                    <yaw>0</yaw>
                </RelativeAngEulerToBodyDeg>
            </camera>
            <camera>
                <name>Up_Camera</name>
                <RelativePosToBodyCm>
                    <x>0</x>
                    <y>0</y>
                    <z>10</z>
                </RelativePosToBodyCm>
                <RelativeAngEulerToBodyDeg>
                    <roll>0</roll>
                    <pitch>90</pitch>
                    <yaw>0</yaw>
                </RelativeAngEulerToBodyDeg>
            </camera>
        </OnboardCameras>
    </vehicle>
```

载具的三维样式

ClassID（模型类别 ID）

标签表示飞机构型（大样式），目前 ClassID 有下列选择：3（四旋翼）、30（人物）、40（标定板）、100（固定翼飞机）、5 和 6（六旋翼）、60（发光体）、151（方形环）、150（圆形环）、152（球形）、50（小车）

DisplayOrder（同类模型的显示顺序）

表示同一构型（如四旋翼）可选模型列表中的排列优先级（越小排的越靠前，排序为小样式），内置的机型优先级从 1000 开始标号，如果小于 1000 则会取代内置飞机，优先展示最新导入机型。例如，DroneyeeX680.xml 模板中 1015 对应的是排名第 4 的飞机；

注意

飞机的显示样式由大样式和小样式共同决定，在 mav.sendUE4Pos 的命令中直接发送 vehicleType 来直达需要的样式，**格式是：大样式+小样式*100000**。

Name（模型显示名称）

标签表示在 UE 中显示的飞机名字；

Scale（模型显示三维尺寸）

标签表示整机三维缩放的尺寸；

AngEulerDeg（模型初始显示姿态）

表示飞机显示偏转一定角度（单位度），例如输入 yaw=180，则飞机初始姿态会掉个头变成头朝后

Body（模型机体构成）

是机体主标签

ModelType（激活载具蓝图）

<ModelType>取值为 3 时，能够激活载具蓝图(UE 的载具类 WheeledVehiclePawn)

ModelType（网格体类型）

表示是否为带动画的网格，这里一般都需要选 0，除非对于一些自带动画的三维模型（0：StaticMesh 1：Animation 2：Blueprints ）。（PS：曾经这个 xml 的标签名为 isAnimationMesh，如果有旧版的 xml 文件可能会显示为它）

MeshPath（模型三维文件路径）

表示机体的三维文件所在目录，其中/Game/表示 RflySim3D 内容目录，DroneyeeX680/DroneyeeX680Body 表示刚才拷贝出的机身三维模型文件。

MaterialPath（材质路径）

表示材质的路径，如果 UE 里面已经设置过材质这里可以不填，填上的话会用本

材质叠加到飞机上

AnimationPath（动画路径）

CenterHeightAboveGroundCm（模型质心离地高度）

表示质心离地面高度

isMoveBodyCenterAxisCm（模型轴心位置）

表示调整模型三维轴心位置向量，输入逗号分隔的三维向量。

NumberHeigthAboveCenterCm（模型显示标签离地高度）

表示载具模型上的数字标号（CopterID）显示的高度

ActuatorList（执行器显示参数列表）

为作动器（螺旋桨、舵机、轮胎、软管刚体、转台）等外部活动（可定义）组件的列表

Actuator（某执行器）

标签内为某一个作动器的具体参数

MeshPath（网格体路径）

三维模型路径，同 body 标签定义

MaterialPath（材质路径）

材质路径同 body 标签定义

RelativePosToBodyCm（相对机体中心位置）

表示该作动器(例如螺旋桨)的相对机体中心的位置

RelativeAngEulerToBodyDeg（安装角度）

表示作动器的安装角度（单位度），通常给 0 即可，表示和导入 UE 时的姿态相同，不需要再旋转了

RotationModeSpinOrDefect（运动模式）

表示作动器运动模式，0 表示旋转(输入为转动速度, 单位转每分)，1 表示偏转(输入为转动角度, 单位度)，2 表示静止物体，3 表示平移（单位厘米）

RotationAxisVectorToBody（旋转轴）

旋转轴，这里多旋翼的螺旋桨是绕 z 轴旋转，如果是车辆则绕 y 轴旋转，如果是固定翼的螺旋桨则是绕 x 轴旋转。

OnboardCameras（机载摄像头）

标签页是定义了一些观察的机载摄像头，从第 2 个摄像头开始，都是固连在机体上的，因此填写摄像头的安装角度和位置即可定义；第 1 个摄像头是一个跟随观察的视角，它不会跟随飞机俯仰滚转，但是会跟随飞机偏航。

AttatchToOtherActuator（执行器附加功能）

标签定义了执行器的附加到另一个执行器上功能，“附加”后执行器会随着父执行器进行移动与旋转，但它也可以单独旋转，<AttatchToOtherActuator>2</Attatch ToOtherActuator >表示将当前执行器附加到 ActuatorList 中的第二个执行器上。

在“\PX4PSP\RflySim3D\RflySim3D\Plugins\Rfly3DSimPlugin\Content\XML”找到其他无人机的 XML 参考。

8.2 普通导入（静态网格体/骨骼网格体）

8.2.1 静态网格体拼接模型（ModelType 标签取 0）

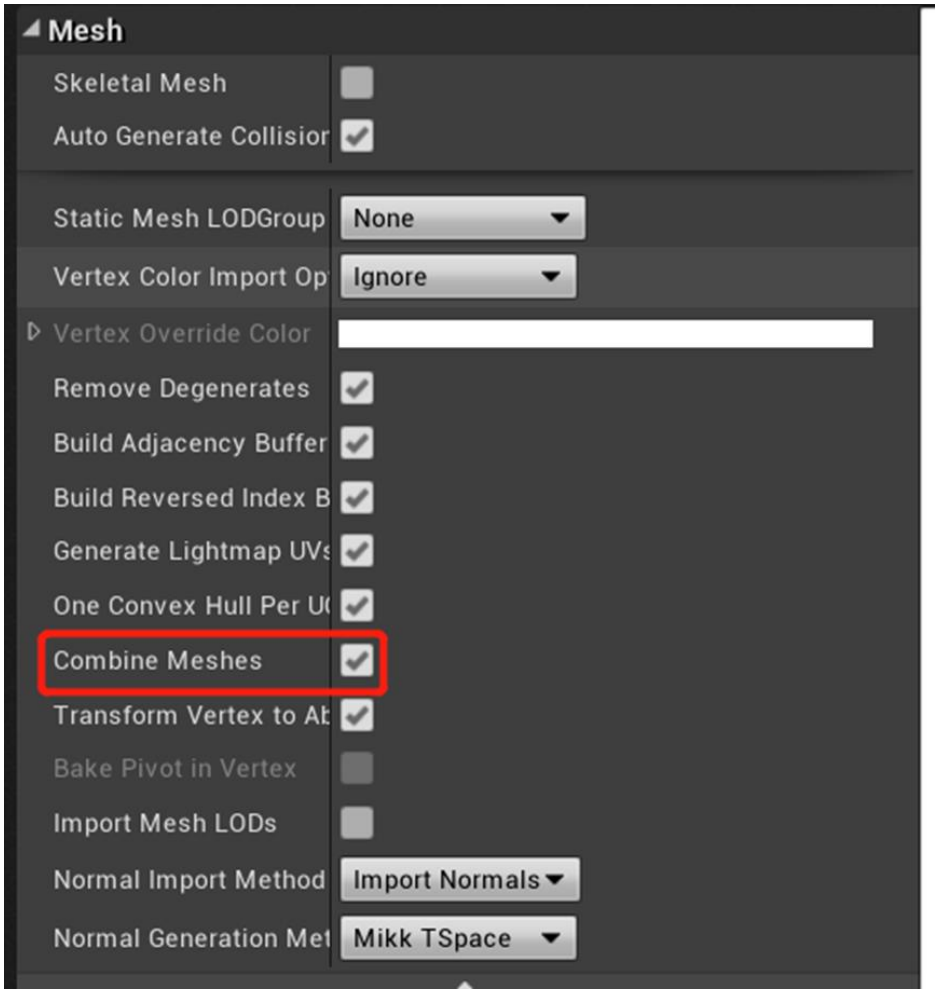
将需要控制的执行器和无需控制的机身一同导入 UE 进行组装和材质替换，烘焙完成后，将[项目名] -> Saved -> Cooked -> WindowsNoEditor -> [项目名]-> Content 下的内容与对应的 XML 文件拷贝至 PX4PSP -> RflySim3D -> RflySim3D-> Content 下

注意

导入 UE 的模型应为 FBX、OBJ 等通用三维格式，导入前应在三维处理软件中分割出需要控制的执行器，例如，固定翼飞机通常要控制数个舵面，则将其余部分一同划归机身。

在将模型导入 UE 时，勾选“Combine Mesh”选项，使得所有组件构成一个整体导入，

否则会将机体分成 N 个部件。



XML 文件中的 ModelType 标签应该取 0，对应静态网格体

8.2.2 自带动画模型（ModelType 标签取 1）

将模型的骨骼网格体和动画序列一同导入 UE，烘焙完成后，将[项目名] -> Saved -> Cooked -> WindowsNoEditor -> [项目名]-> Content 下的内容与对应的 XML 文件拷贝至 PX4PSP -> RflySim3D -> RflySim3D-> Content 下。

注意

这里的模型主要是用作一些移动障碍（如人物模型），可以直接在虚幻商城搜索需要的模型，这样就不用在三维处理软件中进行蒙皮绑骨等操作。

XML 文件中的 ModelType 标签应该取 1，对应自带动画效果的骨骼网格体

8.3 蓝图导入（普通蓝图/载具蓝图）该功能仅限个人高级版以上

8.3.1 使用空白模板 （ModelType 标签取 2）

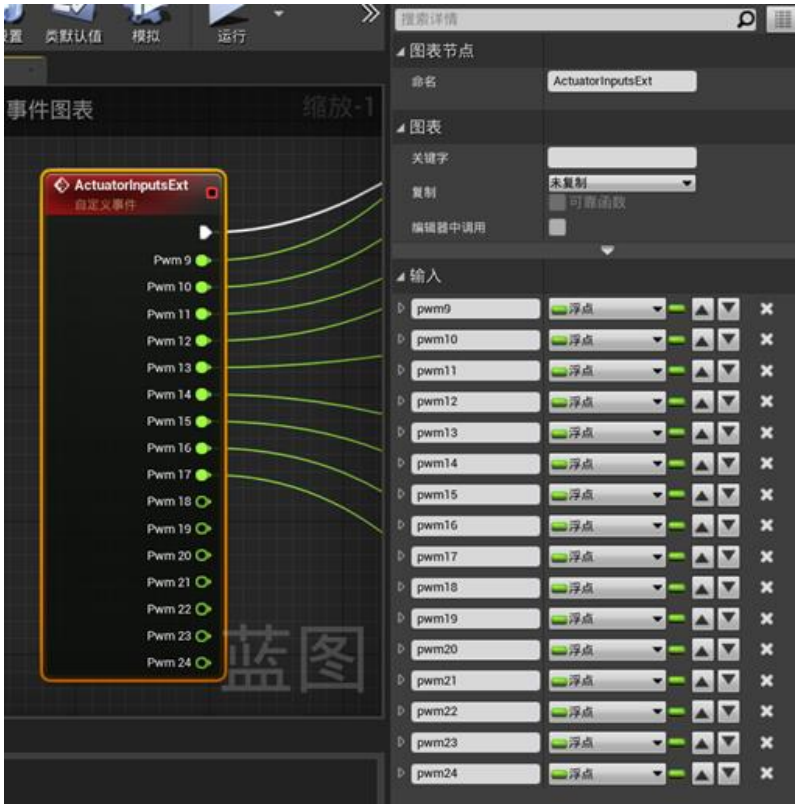
将模型的骨骼网格体和动画序列一同导入 UE，添加替换材质。为模型创建动画蓝图定义各执行器动画播放效果；为模型创建蓝图类，在事件图表中控制相应执行器触发条件及

传输数据接口。烘焙完成后，将[项目名] -> Saved -> Cooked -> WindowsNoEditor -> [项目名] -> Content 下的内容与对应的 XML 文件拷贝至 PX4PSP -> RflySim3D -> RflySim3D-> Content 下。

注意

1.如果直接从虚幻商城获取的载具模型，会带有已经配置好的动画蓝图和碰撞模型，只需在蓝图类中修改蓝图事件。

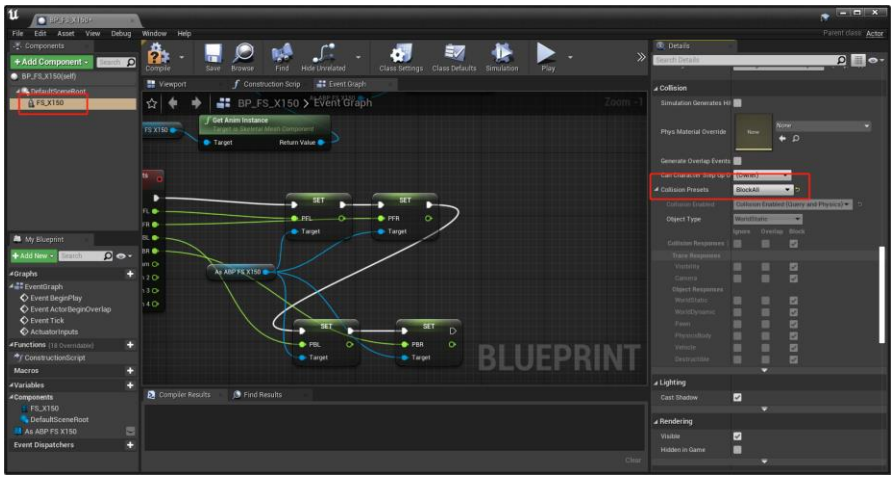
蓝图逻辑接入平台的接口有两个，“ActuatorInputs”和“ActuatorInputsExt”，在蓝图类的事件图表中创建这两个自定义事件。其中，“ActuatorInputs”右侧“输入”端创建 8 维的（pwm1~pwm8）的输入信号，作为基础 8 维的执行器输入信号；“ActuatorInputsExt”添加 pwm9~pwm24 的 16 维输入信号，作为 9~24 维扩展执行器输入信号接收端。



XML 文件中的 ModelType 标签应该取 2，对应自带程序逻辑的蓝图类

2.如果导入 UE 的是自定义的骨骼网格体，除了配置上述两个蓝图接口，还需要启用模型蓝图类的碰撞响应，这是由于骨骼网格体默认导入 Actor，其碰撞预设都是 NoCollision。

找到自定义的蓝图类，选中其中的骨骼网格体组件，在其碰撞预设中选择 BlockAll。

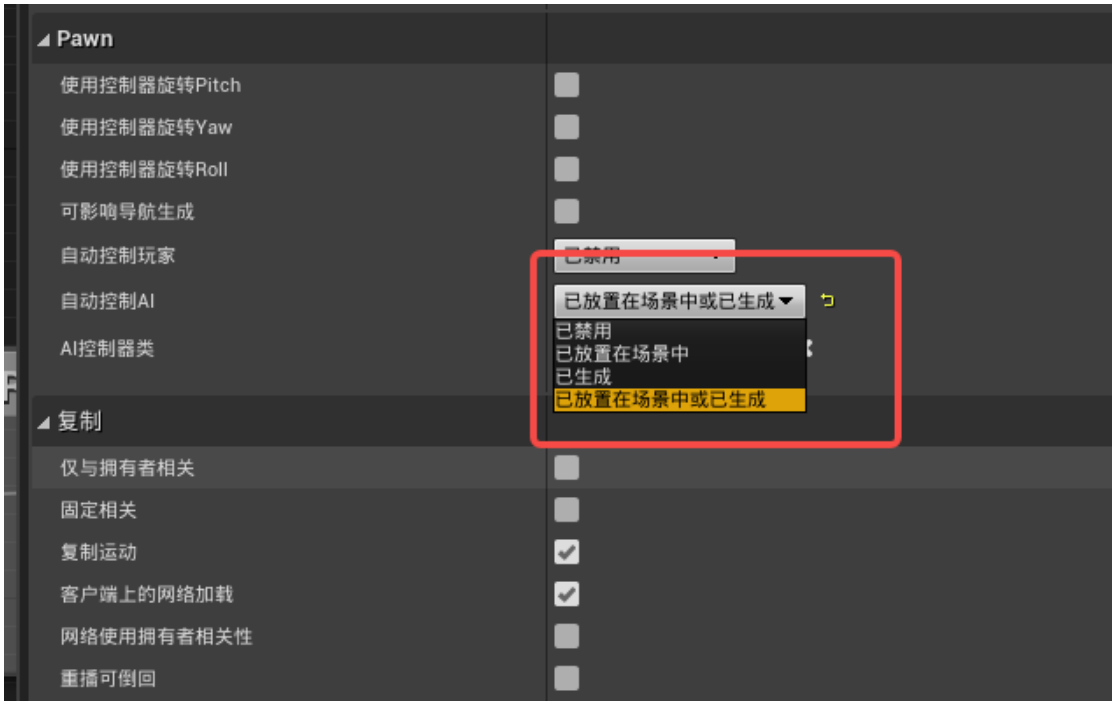


8.3.2 使用载具模板（ModelType 标签取 3）

创建 UE 项目时使用载具模板，其余步骤与使用空白模板蓝图相同

注意

若使用载具模板自带的模型，需要在载具蓝图类中启用自动控制 AI 如下图



XML 文件中使用 ModelType 标签代替 ModelType 标签，对应值取 3，对应载具蓝图类

8.4 蓝图控制规则

三维场景中显示的一个个物体都是一个个的对象，它们也有一个个成员变量、成员函数，我们控制的这些无人机也是这样，这些对象是在 UnrealEngine 中定义的，它们如果是继承于 UE 蓝图类的对象，则平台提供了调用蓝图函数的接口。RflySim3D 提供给蓝图的接口主要包括：“ActuatorInputs”与“ActuatorInputsExt”，这两个接口可以将一组数据传输给场景中指定的无人机，可以用于展示一些自定义的效果（该效果需要使用蓝图系统自己编写），例如打开舱门、爆炸、显示文字、切换飞机材质、使机翼旋转等等任何 UE 蓝图系统能做到的事情。它是更高阶的接口，因此也要求基本掌握 UE 的蓝图使用方法。

所谓蓝图接口，就是从外部调用蓝图的函数，从而触发在蓝图中编写的各种功能。这种调用可以通过之前介绍过的两个 RflySim3D 控制台命令：“RflySetActuatorPWMs”与“RflySetActuatorPWMsExt”来实现，也可使用其他程序（如 Python、Simulink）通过 UDP 发送指定的结构体来实现。

8.4.1 ActuatorInputs 接口

sendUE4Pos 系列 python 命令

使用方法见 sendUE4Pos（创建/更新模型）

RflySetActuatorPWMs 控制台命令

使用方法见 RflySetActuatorPWMs（触发蓝图接口）

UE4DataEncoder（Simulink 子模块）

使用方法见 UE4DataEncoder

8.4.2 ActuatorInputsExt 接口

sendUE4ExtAct（python 命令）

使用方法见 sendUE4ExtAct（触发扩展蓝图接口）

RflySetActuatorPWMsExt 控制台命令

使用方法见 RflySetActuatorPWMsExt（触发扩展蓝图接口）

Ue4ExtMsgEncoder（Simulink 子模块）

使用方法见 Ue4ExtMsgEncoder

9. 外部交互接口

9.1 通信端口

9.1.1 网络通信

RflySim3D 接收

组播 IP 地址和端口 224.0.0.11： 20008

限制了数据源只能来自于本机（127.0.0.1）。目前，CopterSim 在联机选项未勾选的情况下，默认会通过本接口向 RflySIm3D 发数据。

SocketReceiver1 类

接收 224.0.0.11：20008 的 UDP 包（虽然是组播，但设置了只接受本地环回地址的包），还会接收本机 20009 端口的包（不是组播也可以）

组播 IP 地址和端口 224.0.0.10： 20009

在源码 JoinedToGroup 里面设置了数据源为任意，因此可以接受局域网的消息，进行分布式联机仿真。目前，CopterSim 在联机选项勾选的情况下，默认会通过本接口向 RflySIm3D 发数据。

SocketReceiver 类

接收 224.0.0.10：20009 的 UDP 包，还会接收本机 20009 端口的包（不是组播也可以）

本机 20010++1 系列端口(20010~20029)

在一台电脑上，第一个运行的 RflySim3D，会直接监听本接口（20010，窗口名字是 RflySim3D-0）。第 2 个窗口再绑定 20010 时，会发现被占用，因此转而订阅下一个端口（20011，窗口名字是 RflySim3D-1）。依次类推，最多 20 个窗口，也就是最多到 20029，RflySim3D-19。Python 的 UE4 系列函数接口，指定 RflySim3D 窗口的功能就是通过这个实现的。如果要发往 RflySim3D-i，就要用端口 20010+i。

SocketReceiverMap 类

接收本机 20010~20029 端口中的一个（这是因为一台电脑上多个 RflySim3D 会争夺端口，所以根据序号会依次分配这 20 个端口）

RflySim3D 发送

Sendersocket 类

组播 IP 地址和端口 224.0.0.10：20002

Ue4Req 结构体

详见 Ue4Req（接收/回应询问）

UTF8 的字符串

组播 IP 地址和端口 224.0.0.10： 20006（Python、Simulink）

用于 RflySim3D 传出数据给 Python，Simulink 接收。包括以下结构体：

reqVeCrashData（坠机数据结构体）

详见 reqVeCrashData（坠机数据）

CopterSimCrash 结构体

详见 CopterSimCrash

CoptReqData

详见 CoptReqData（回应收到的 ReqObjData）

ObjReqData（物体信息数据结构体）

详见 ObjReqData（回应收到的 ReqObjData）

CameraData（相机数据结构体）

详见 CameraData（回应收到的 ReqObjData）

9999++1 系列端口（回传图像）

回传图片的系列端口，在图像传感器的 json 里面定义，通常是 9999+i 的形式，这个可以通过 json 修改。如果一路图像选择了网络传输的形式，那么它会将压缩后的图像，往这个端口发。

30100++2 系列端口（CopterSim）

1）CopterSim 和 RflySim3D 有一个响应请求机制。CopterSim 发送一个 Ue4Req 结构体给 RflySim3D，收到后会将 reqIndex 置为 1，并返回这个接口体。形成一个握手机制。通常第一个 CopterSim 窗口会通过 20008 发送这个消息，确认 RflySim3D 是否接收到三维数据，如果超时没有反馈，CopterSim 会切换到 20010 端口向 RflySim3D 传输数据。

2）返回 CopterID 号飞机的碰撞数据（Ue4RayTraceDrone 结构体），通过 30100+2*CopterID-2 号端口，发送到指定的 CopterSim，实现碰撞数据的接收。

3）返回坠机数据

Ue4Req 结构体

详见 Ue4Req（接收/回应询问）

Ue4RayTraceDrone 结构体

详见 Ue4RayTraceDrone 结构体（发给各 Copter 所属 CopterSim）

9.1.2 共享内存

UE4CommMemData（32 个视觉传感器）

详见 UE4CommMemData（图像校验数据）

RflySim3DImg_i（第 i 个视觉传感器）

9.1.3 Redis 通信

9.2 数据协议（UDP 接口）

9.2.1 发送

图像

imageHeaderNew（图像数据结构体）

```
struct imageHeaderNew {
 int checksum = 0;//1234567890;
 int PackLen = 0;
 int PackSeq = 0;
 int PackNum = 0;
 double timeStmp = 0;
};
```

作用解释：

imageHeaderNew 结构体的作用是表示图像头部信息，它可以用来存储与图像有关的标识信息。这个结构体可以作为数据的一部分，发送它时，后面会接最多 60000 字节的图像数据（因此每个包大小都为 60000+4×4+8×1），这 60000 字节中可能有多个图像包，每个前面都有一个这个结构体。

参数解释：

- checksum（校验和）：这个参数用于存储一个整数值，通常用于数据的完整性检查。校验和是通过对数据进行算术运算生成的值，用于检测数据在传输或存储过程中是否发生了错误或损坏。
- PackLen（数据包长度）：这个参数表示数据包的长度，通常是一个整数值。它可以用来指示数据包中包含的数据的大小或长度信息。
- PackSeq（数据包序列号）：这个参数用于存储数据包的序列号，通常是一个整数值。序列号可以帮助在接收端恢复数据包的正确顺序，特别是在数据包在网络中传输时可能会乱序的情况下。
- PackNum（数据包编号）：与 PackSeq 类似，这个参数也是一个整数，可能用于标识数据包的编号或其他类似的信息。
- timeStmp（时间戳）：这个参数是一个双精度浮点数，用于存储时间戳信

息。时间戳通常表示一些事件或数据的时间，可能用于记录图像头部信息生成的时间或其他与时间相关的信息。

UE4CommMemData（图像校验数据）

```
struct UE4CommMemData{
 int Checksum;//校验位，设置为 1234567890
 int totalNum;//最大传感器数量
 int WidthHeigh[64];//分辨率宽高序列，包含最多 32 个传感器的
}
```

作用解释

与 imageHeaderNew 类似，目前平台还会在名为：UE4CommMemData 的共享内存中存放图像的校验数据。

UE4CommMemData 结构体的作用是表示与 UE4（Unreal Engine 4）通信以及传感器数据管理相关的内存信息。这个结构体可以用来存储与通信和传感器数据处理相关的配置信息。

参数解释：

- Checksum（校验位）：这个参数用于存储校验位，它通常用于数据完整性检查。校验位的默认值已设置为 1234567890，这是一个预设的校验值。校验位用于验证数据在传输或存储过程中是否受损或被篡改。
- totalNum（最大传感器数量）：这个参数是一个整数，表示系统中可用的最大传感器数量。它可能用于配置或记录系统支持的传感器的数量。
- WidthHeigh（分辨率宽高序列）：这个参数是一个整数数组，最多可以包含 64 个元素。它用于存储传感器的分辨率宽度和高度序列。这个数组可以包含最多 32 个传感器的分辨率信息。

询问+心跳

Ue4Req（接收/回应询问）

```
struct Ue4Req {
 int checksum;
 int reqIndex ;
}
```

作用解释

Ue4Req 结构体的作用是用于发起请求和接收响应，以在 CopterSim 和 RflySim3D 之间建立通信的握手机制。它允许 CopterSim 发送请求给 RflySim3D，并确认 Rfl

ySim3D 是否成功接收请求的方式。接受到 CopterSim 的该结构体后，会返回给该 CopterSim 一个同样的结构体

参数解释

- checksum（校验位）：这个参数是一个整数，用于存储校验位。校验位通常用于数据完整性检查，以确保接收到的数据没有被篡改。在这里，它可能用于验证 Ue4Req 结构体的完整性。
- reqIndex（请求索引）：这个参数也是一个整数，用于表示请求的标识。

  0：没有飞机数据

  1：已有飞机数据

射线检测

reqVeCrashData（坠机数据）

```
struct reqVeCrashData {
 int checksum; //数据包校验码 1234567897
 int copterID; //当前飞机的 ID 号
 int vehicleType; //当前飞机的样式
 int  CrashType;//碰撞物体类型，-2 表示地面，-1 表示场景静态物体，0 表示无碰撞，1 以上表示被碰飞机的 ID 号
 double runnedTime; //当前飞机的时间戳
 float VelE[3]; // 当前飞机的速度
 float PosE[3]; //当前飞机的位置
 float CrashPos[3];//碰撞点的坐标
 float targetPos[3];//被碰物体的中心坐标
 float AngEuler[3]; //当前飞机的欧拉角
 float MotorRPMS[8]; //当前飞机的电机转速
 float ray[6]; //飞机的前后左右上下扫描线
 char CrashedName[20] = {0};//被碰物体的名字
}
```

作用解释：

reqVeCrashData 结构体的作用是用于存储与飞机碰撞相关的数据。这个结构体的字段包括了碰撞的各种细节信息，如碰撞类型、时间戳、位置坐标、速度、姿态等。

参数解释：

- checksum（数据包校验码）：用于验证数据的完整性，以确保数据在传输过程中没有被篡改。其值为 1234567897。
- copterID（飞机的 ID 号）：用于唯一标识不同的飞机。
- vehicleType（飞机的样式）：表示当前飞机的样式或类型。
- CrashType（碰撞物体类型）：表示碰撞物体的类型，-2 表示地面，-1 表示场景静态物体，0 表示无碰撞，而 1 以上的值表示被碰撞的飞机的 ID 号。

- runnedTime（当前飞机的时间戳）：记录碰撞发生的时间。
- VelE（当前飞机的速度）：包含 3 个浮点数的数组，表示当前飞机的速度分量，通常包括东向、北向和上升速度。
- PosE（当前飞机的位置）：包含 3 个浮点数的数组，表示当前飞机的位置坐标，通常包括东向、北向和上升位置。
- CrashPos（碰撞点的坐标）：包含 3 个浮点数的数组，表示碰撞发生的位置坐标。
- targetPos（被碰物体的中心坐标）：包含 3 个浮点数的数组，表示被碰撞物体的中心坐标。
- AngEuler（当前飞机的欧拉角）：包含 3 个浮点数的数组，用于描述当前飞机的欧拉角，通常包括俯仰、偏航和滚转角。
- MotorRPMS（当前飞机的电机转速）：包含 8 个浮点数的数组，用于表示当前飞机的各个电机的转速
- ray（飞机的前后左右上下扫描线）：这是一个包含 6 个浮点数的数组，用于描述飞机进行前后左右上下方向扫描的数据。
- CrashedName（被碰物体的名字）：这是一个长度为 20 的字符数组，用于存储被碰撞物体的名字。

Ue4RayTraceDrone 结构体（发给各 Copter 所属 CopterSim）

```
struct Ue4RayTraceDrone {
 int checksum;
 int CopterID;
 float size;
 float velE[3];
 float ray[6];//前后左右上下
 float posE[3];
}
```

作用解释：

用于存储射线追踪无人机的关键数据。射线追踪无人机通常用于进行环境感知、避障、碰撞检测等任务。这些任务需要获取无人机的状态信息以及射线追踪数据。

参数解释：

- checksum（数据包校验码）：这个整数字段用于存储数据包的校验码，以验证数据的完整性。它有助于确保接收到的数据没有在传输过程中被损坏或篡改。
- CopterID（飞机的 ID 号）：这个整数字段表示当前射线追踪无人机的唯一标识号。每个无人机都应有一个唯一的 ID 号，以便识别和跟踪不同的无人

机。

- size（尺寸）：这个浮点数字段表示无人机的尺寸或大小。尺寸信息可以用于避障和碰撞检测，以确保无人机能够安全地通过狭窄的通道或避免与物体碰撞。
- velE（速度）：这是一个包含 3 个浮点数的数组，用于表示无人机的速度分量。通常，这包括东向速度、北向速度和上升速度。
- ray（射线）：这是一个包含6个浮点数的数组，表示无人机进行前、后、左、右、上和下方向的射线追踪数据。射线追踪数据用于检测周围环境中的物体，以帮助无人机避障和进行碰撞检测。
- posE（位置）：这是一个包含 3 个浮点数的数组，表示无人机的当前位置坐标。通常，这包括东向位置、北向位置和上升位置。

相机

CameraData（回应收到的 ReqObjData）

```
struct CameraData { //56
 int checksum = 0;//1234567891
 int SeqID; //相机序号
 int TypeID;//相机类型
 int DataHeight;//像素高
 int DataWidth;//像素宽
 float CameraFOV;//相机视场角
 float PosUE[3]; //相机中心位置
 float angEuler[3];//相机欧拉角
 double timestmp;//时间戳
}
```

作用解释：

CameraData 结构体的主要作用是存储与摄像机相关的数据，以便进行分析、处理等后续操作。

参数解释：

- checksum（数据包校验码）：这个整数字段用于存储数据包的校验码，以验证数据的完整性。
- SeqID（相机序号）：这个整数字段表示相机的序号或标识，用于区分不同的摄像机设备。如果有多个摄像机，这个字段可以帮助识别是哪个摄像机产生的数据。
- TypeID（相机类型）：这个整数字段表示相机的类型或种类，可以用于标识不同种类的摄像机设备。
- DataHeight（像素高度）：这个整数字段表示图像数据的高度，即图像的

像素行数。这个参数定义了图像的分辨率。

- DataWidth（像素宽度）：这个整数字段表示图像数据的宽度，即图像的像素列数。这个参数也定义了图像的分辨率。
- CameraFOV（相机视场角）：这个浮点数字段表示相机的视场角，通常以度数表示。它描述了相机可以覆盖的水平范围，对于摄像机的视野设置非常重要。
- PosUE（相机中心位置）：这是一个包含 3 个浮点数的数组，表示相机的中心位置坐标。这些坐标表示相机在 UE 坐标系中的位置
- angEuler（相机欧拉角）：这是一个包含 3 个浮点数的数组，用于表示相机的欧拉角，通常包括俯仰、偏航和滚转角度。这些角度描述了相机的朝向。
- timestmp（时间戳）：这个双精度浮点数字段表示数据的时间戳，用于记录数据采集的时间。时间戳通常用于数据同步、时间相关的分析和事件时间记录。

模型

CoptReqData（回应收到的 ReqObjData）

```
struct CoptReqData { //64
 int checksum = 0; //1234567891 作为校验
 int CopterID;//飞机 ID
 float PosUE[3]; //物体中心位置（人为三维建模时指定，姿态坐标轴，不一定在几何中心）
 float angEuler[3];//物体欧拉角
 float boxOrigin[3];//物体几何中心坐标
 float BoxExtent[3];//物体外框长宽高的一半
 double timestmp;//时间戳
};
```

参数解释

- checksum（数据包校验码）：这个整数字段用于存储数据包的校验码，其默认值为 0，但注释中提到了示例校验码值 1234567891。校验码通常用于验证数据的完整性，以确保数据在传输过程中没有被篡改。
- CopterID（飞机 ID）：这个整数字段表示飞机的唯一标识号，用于区分不同的飞机。
- PosUE（物体中心位置）：这是一个包含 3 个浮点数的数组，表示物体的中心位置坐标。
- angEuler（物体欧拉角）：这是一个包含 3 个浮点数的数组，用于表示物体的欧拉角，通常包括俯仰、偏航和滚转角度。这些角度描述了物体的朝向。

- boxOrigin（物体几何中心坐标）：这是一个包含 3 个浮点数的数组，表示包围盒体的几何中心坐标。
- BoxExtent（物体外框长宽高的一半）：这是一个包含 3 个浮点数的数组，表示物体外框（包围盒）的长、宽和高的一半尺寸。这个信息对于描述物体的边界框非常有用。
- timestmp（时间戳）：这个双精度浮点数字段表示数据的时间戳，用于记录数据采集的时间。时间戳通常用于数据同步、时间相关的分析和事件时间记录。

CopterSimCrash

```
struct CopterSimCrash {
 int checksum = 0; //1234567891 作为校验
 int CopterID;//飞机 ID
int TargetID; //目标飞机 ID
};
```

静态物体

ObjReqData（回应收到的 ReqObjData）

```
struct ObjReqData { //96
 int checksum = 0; //1234567891 作为校验
 int seqID = 0;
 float PosUE[3]; //物体中心位置（人为三维建模时指定，姿态坐标轴，不一定在几何中心）
 float angEuler[3];//物体欧拉角
 float boxOrigin[3];//物体几何中心坐标
 float BoxExtent[3];//物体外框长宽高的一半
 double timestmp;//时间戳
 char ObjName[32] = { 0 };//碰物体的名字
};
```

9.2.2 接收

单个飞机数据

SOut2Simulator（单机数据 1）

数据结构

```
//FULL 完整模式，输出到 RflySim3D 的数据
```

```
struct SOut2Simulator {
    int checksum; // 校验码 123456789，必须设定为本值，才会认为是有效数据
    int copterID;  //飞机 ID 序号
    int vehicleType;  //载具样式 ID，对应 UE 的 XML 中 ClassID
    int reserv; //备用标志位，Int 型，将来用于表示碰撞、或大地图等标识
    float VelE[3];   //速度，北东地，单位米/s
    float AngEuler[3];  //欧拉角，滚转俯仰偏航，单位弧度
    float AngQuatern[4]; //姿态四元数向量，w x y z
    float MotorRPMS[8];  //执行器偏转量或转速，旋翼类对应 RPM 转每分
    float AccB[3];       //载具机体 FRD 坐标系下加速度
    float RateB[3];      //载具机体 FRD 坐标系下角速度，pqr，单位 rad/s
    double runnedTime; //时间戳，仿真开始为 0 时刻。
    double PosE[3];   //北东地位置，单位米，z 向下为正
    double PosGPS[3];    //载具纬度、经度、高度向量，单位度和米，高度向上为正
};
```

作用解释

可以使用这些数据来更新单个飞机的状态、计算新的姿态和位置，在下一帧渲染出来。该结构体为FULL完整通信模式下（如UDP_full），输出到RflySim3D的数据。

参数解释

参数名	参数类型	描述	单位
checksum（校验码）	整数	这个整数字段用于验证数据的有效性，必须设定为 123456789，才会被 RflySim3D 认识为有效数据。	无
copterID（飞机 ID）	整数	这个整数字段用于标识飞机的唯一标识符，以便区分不同的飞机。	无
vehicleType（三维样式）	整数	这个整数字段表示飞机的构型类别，以区分不同种类的飞机，例如直升机、固定翼飞机等。	无
reserv（备用标志位）	整数	这个整数字段用于表示一些特殊的情况，例如碰撞、大地图等，目前没有使用。	无
VelE（速度）	浮点数数组	这个浮点数数组表示飞机在北东地坐标系下的速度分量，分别对应北向、东向和地向的速度。	米/秒
AngEuler（姿态欧拉角）	浮点数数组	这个浮点数数组表示飞机在北东地坐标系下的姿态角，分别对应滚转、俯仰和偏航角。	弧度
AngQuatern（姿态四元数）	浮点数数组	这个浮点数数组表示飞机的姿态四元数，分别对应 w、x、y、z 分量。姿态四元数是一种表示三维旋转的数学方法，具有唯一性、连续性和稳定	无

		性的优点。	
MotorRPMS（执行器偏转量或转速）	浮点数数组	这个浮点数数组表示飞机的执行器的输出，对于旋翼类飞机，表示每个旋翼的转速，对于其他类型的飞机，表示其他相应的执行器的偏转量，例如升降舵、方向舵等。	转/分或无
AccB（载具机体加速度）	浮点数数组	这个浮点数数组表示飞机在自身的机体坐标系下的加速度分量，分别对应前后、右左、上下的加速度，也称为 FRD 坐标系。	米/秒^2
RateB（载具机体角速度）	浮点数数组	这个浮点数数组表示飞机在自身的机体坐标系下的角速度分量，分别对应滚转、俯仰和偏航轴的角速度，也称为 pqr。	弧度/秒
runnedTime（时间戳）	双精度浮点数	这个双精度浮点数字段表示飞机已经运行的时间。	秒
PosE（载具北东地位置）	双精度浮点数数组	这个双精度浮点数数组表示飞机在北东地坐标系下的位置分量，分别对应北向、东向和地向的位置。注意地向的位置是向下为正，即高度是负值。	米
PosGPS（载具纬度、经度、高度）	双精度浮点数数组	这个双精度浮点数数组表示飞机的纬度、经度和高度，分别对应纬度、经度和高度。注意高度是向上为正，即海拔是正值。	度和米

//解析方法：对比包长 168，校验 checksum 是否匹配 123456789，然后按 4i24f7d 编码进行解析

//解析示例详见：PX4MavCtrlV4.py/getTrueDataMsg()函数

//发送示例详见：UE4CtrlAPI.py/sendUE4PosFull()函数

SOut2SimulatorSimple（单机数据 2）

```
// Simple 简化模式，输出到 RflySim3D 的数据
struct SOut2SimulatorSimpleTime {
    int checkSum; //校验码：1234567891，必须设定为本值，才会认为是有效数据
    int copterID;  //飞机 ID 序号
    int vehicleType;  //载具样式 ID，对应 UE 的 XML 中 ClassID
    int PosGpsInt[3];   //纬度、精度、高度：lat*10^7,lon*10^7,alt*10^3，int 型发放节省空间
    float MotorRPMS[8]; //执行器偏转量或转速
    float VelE[3]; //速度，北东地，单位米/s
    float AngEuler[3];  //欧拉角，滚转俯仰偏航，单位弧度
    double PosE[3];   //北东地位置，单位米，z 向下为正
```

```
    double runnedTime; //时间戳，仿真开始为 0 时刻。
};
```

各参数含义与“SOut2Simulator”中相同。该结构体为 Simple 简化通信模式下（如 UDP_simple），输出到 RflySim3D 的数据。

//解析方法：对比包长 112，校验 checksum 是否匹配 1234567891，然后按 6i14f4d 编码进行解析

//解析示例详见：PX4MavCtrlV4.py/getTrueDataMsg()函数

//发送示例详见：UE4CtrlAPI.py/sendUE4PosNew()函数

SOut2SimulatorSimpleTimeF（单机数据 3）

```
struct SOut2SimulatorSimpleTimeF {
 int checkSum;
 int copterID;  //Vehicle ID
 int vehicleType;  //Vehicle type
 float PWMs[8];
 float PosE[3];   //NED vehicle position in earth frame (m)
 float VelE[3];
 float AngEuler[3];  //Vehicle Euler angle roll pitch yaw (rad) in x y z
 double runnedTime;
}
```

各参数含义与“SOut2Simulator”中相同

- checksum 是数据校验位，1234567890 表示正常数据，1234567891 表示飞机始终贴合地面

SOut2SimulatorSimpleTime（单机数据 4）

```
struct SOut2SimulatorSimpleTime {
 int checkSum; //1234567890
 int copterID;
 int vehicleType;
 float PWMs[8];
 float VelE[3];
 float AngEuler[3];
 double PosE[3];
 double runnedTime;
}
```

各参数含义与“SOut2Simulator”中相同

- checksum 是数据校验位，1234567890 表示正常数据，1234567891 表示飞机始终贴合地面

SOut2SimulatorSimple1（单机数据 5）

```
struct SOut2SimulatorSimple1 {
 int checkSum;
 int copterID;
 int vehicleType;
 float MotorRPMSMean;
 float PosE[3];
 float AngEuler[3];
 float Scale[3];
}
```

各参数含义与“SOut2Simulator”中相同

- checksum 是数据校验位，1234567890 表示正常数据，1234567891 表示飞机始终贴合地面
- MotorRPMSMean 表示输入 1 维电机数据，会将这个数同时存入 8 维电机中
- Scale[3]表示 xyz 方向的缩放尺度。

短数据包

_netDataShort

```
typedef struct _netDataShort {
 int  tg;
 int  len;
 char  payload[PAYLOAD_LEN_SHORT];
}
```

参数解释

- payload 会被解释为“SOut2Simulator”结构的数据

多机数据

Multi3DData（20 机数据 1）

```
struct Multi3DData {
 uint16 IDs[20];
 uint16 VehileTypes[20];
 float PosE[60];
 float AngEuler[60];
}
```

作用解释

会经过一个 for 循环，依次放入数组中，并在下一帧更新这 20 架飞机数据

参数解释

- IDs（飞机 ID 数组）：这是一个包含 20 个无符号整数（uint16）的数组，用于存储多个飞机的唯一标识号。每个元素对应一个飞机的 ID，可以用于区分不同的飞机。
- VehileTypes（飞机类型数组）：这是一个包含 20 个无符号整数（uint16）的数组，用于存储多个飞机的类型或类别。与每个飞机 ID 对应的元素表示相应飞机的类型。
- PosE（位置矢量数组）：这是一个包含 60 个单精度浮点数（float）的数组，用于存储多个飞机的位置信息。通常，每组连续的 3 个元素表示一个飞机的位置矢量，使用 NED 坐标。因此，这个数组可以存储 20 个飞机的位置信息。
- AngEuler（欧拉角数组）：这是一个包含 60 个单精度浮点数（float）的数组，用于存储多个飞机的欧拉角信息。类似于位置矢量，每组连续的 3 个元素表示一个飞机的欧拉角，通常包括俯仰、偏航和滚转角度。因此，这个数组可以存储 20 个飞机的欧拉角信息。

Multi3DDataNew（20 机数据 2）

```
struct Multi3DDataNew {
 int checksum;
 uint16 IDs[20];
 uint16 VehileTypes[20];
 float PosE[60];
 float AngEuler[60];
 double runnedTime[20];
}
```

与 Multi3DData（20 机数据 1）类似

- checksum 是数据校验位，1234567890 表示正常数据，1234567891 表示飞机始终贴合地面

Multi3DData1（20 机数据 3）

```
struct Multi3DData1 {
 uint16 IDs[20];
 uint16 VehileTypes[20];
 float PosE[60];
 float AngEuler[60];
 uint16 ScaleXYZ[60];
}
```

与 Multi3DData（20 机数据 1）类似

- ScaleXYZ[60]表示 20 个飞机在 xyz 方向的缩放尺度。

Multi3DData1New（20 机数据 4）

```
struct Multi3DData1New {
 int checksum;
 uint16 IDs[20];
 uint16 VehileTypes[20];
 float PosE[60];
 float AngEuler[60];
 uint16 ScaleXYZ[60];
}
```

与 Multi3DData（20 机数据 1）类似

- checksum 是数据校验位，1234567890 表示正常数据，1234567891 表示飞机始终贴合地面
- ScaleXYZ[60]表示 20 个飞机在 xyz 方向的缩放尺度。

Multi3DData2（20 机数据 5）

```
struct Multi3DData2 {
 uint16 IDs[20];
 uint16 VehileTypes[20];
 float PosE[60];
 float AngEuler[60];
 uint16 ScaleXYZ[60];
 float PWMs[160];
}
```

与 Multi3DData（20 机数据 1）类似

- ScaleXYZ[60]表示 20 个飞机在 xyz 方向的缩放尺度。
- PWMs[160]表示 20 个飞机的 8 维电机数据

Multi3DData2New（20 机数据 6）

```
struct Multi3DData2New {
 int checksum;
 uint16 IDs[20];
 uint16 VehileTypes[20];
 float PosE[60];
 float AngEuler[60];
 uint16 ScaleXYZ[60];
 float PWMs[160];
}
```

与 Multi3DData（20 机数据 1）类似

- checksum 是数据校验位，1234567890 表示正常数据，1234567891 表示飞

机始终贴合地面

- ScaleXYZ[60]表示 20 个飞机在 xyz 方向的缩放尺度。
- PWMs[160]表示 20 个飞机的 8 维电机数据

Multi3DData20Time（20 机数据 7）

```
struct Multi3DData20Time {
 int checksum;
 uint16 IDs[20];
 uint16 VehileTypes[20];
 float PosE[60];
 float VelE[60];
 float AngEuler[60];
 float PWMs[160];
 double runnedTime[20];
}
```

与 Multi3DData（20 机数据 1）类似

- checksum 是数据校验位，1234567890 表示正常数据，1234567891 表示飞机始终贴合地面
- PWMs[160]表示 20 个飞机的 8 维电机数据

Multi3DData10Time（10 机数据）

```
struct Multi3DData10Time {
 int checksum;
 uint16 IDs[10];
 uint16 VehileTypes[10];
 float PosE[30];
 float VelE[30];
 float AngEuler[30];
 float PWMs[80];
 double runnedTime[10];
}
```

作用解释

存储 10 架飞机每一帧的数据

参数解释

- checksum（校验和）：这个整数字段用于存储数据的校验和，以确保数据的完整性和正确性。1234567890 表示正常数据，1234567891 表示飞机始终贴合地面
- IDs（飞机 ID 数组）：这是一个包含 10 个无符号整数（uint16）的数组，用于存储多个飞机的唯一标识号。每个元素对应一个飞机的 ID，可以用于区分不同的飞机。

- VehileTypes（飞机类型数组）：这是一个包含 10 个无符号整数（uint16）的数组，用于存储 10 个飞机的类别。与每个飞机 ID 对应的元素表示相应飞机的类型。
- PosE（位置矢量数组）：这是一个包含 30 个单精度浮点数（float）的数组，用于存储多个飞机的位置信息。每个时间点的位置信息包括 10 个飞机，通常每组连续的 3 个元素表示一个飞机的位置矢量，北东地坐标。因此，这个数组可以存储 10 个飞机在同一帧的位置信息。
- VelE（速度矢量数组）：这是一个包含 30 个单精度浮点数（float）的数组，用于存储多个飞机的速度信息。每帧的速度信息包括 10 个飞机，每组连续的 3 个元素表示一个飞机的速度矢量，包括北东地的速度分量。因此，这个数组可以存储 10 个飞机在同一帧的速度信息。
- AngEuler（欧拉角数组）：这是一个包含 30 个单精度浮点数（float）的数组，用于存储多个飞机的欧拉角信息。每帧的欧拉角信息包括 10 个飞机，每组连续的 3 个元素表示一个飞机的欧拉角，通常包括俯仰、偏航和滚转角度。因此，这个数组可以存储 10 个飞机在同一帧的欧拉角信息。
- PWMs（PWM 信号数组）：这是一个包含 80 个单精度浮点数（float）的数组，用于存储多个飞机的 PWM（脉冲宽度调制）信号信息。每帧的 PWM 信号信息包括 10 个飞机，每组连续的 8 个元素表示一个飞机的 PWM 信号。PWM 信号用于控制飞机的电机和舵面等部件。
- runnedTime（已运行时间数组）：这是一个包含 10 个双精度浮点数（double）的数组，用于存储每帧的时间戳。每个元素表示相应时间点的已运行时间，通常以秒为单位。

Multi3DData100（100 机数据 1）

```
struct Multi3DData100 {
 uint16 IDs[100];
 uint16 VehileTypes[100];
 float PosE[300];
 float AngEuler[300];
 uint16 ScaleXYZ[300];
 float MotorRPMSMean[100];
}
```

作用解释

存储 100 架飞机每一帧的数据

参数解释

- IDs（飞机 ID 数组）：这是一个包含 100 个无符号整数（uint16）的数组，用于存储多个飞机的唯一标识号。每个元素对应一个飞机的 ID，可以用于区

分不同的飞机。

- VehileTypes（飞机类型数组）：这是一个包含 100 个无符号整数（uint16）的数组，用于存储多个飞机的类型或类别。与每个飞机 ID 对应的元素表示相应飞机的类型。
- PosE（位置矢量数组）：这是一个包含 300 个单精度浮点数（float）的数组，用于存储多个飞机的位置信息。通常，每组连续的 3 个元素表示一个飞机的位置矢量，包括北向、东向和地向的坐标。因此，这个数组可以存储 100 个飞机的位置信息。
- AngEuler（欧拉角数组）：这是一个包含 300 个单精度浮点数（float）的数组，用于存储多个飞机的欧拉角信息。类似于位置矢量，每组连续的 3 个元素表示一个飞机的欧拉角，通常包括俯仰、偏航和滚转角度。因此，这个数组可以存储 100 个飞机的欧拉角信息。
- ScaleXYZ（比例尺数组）：这是一个包含 300 个无符号整数（uint16）的数组，用于存储每个飞机的比例尺信息。比例尺用于确定模型在模拟中的尺寸。每组连续的 3 个元素表示一个飞机的比例尺信息，包括 X 轴、Y 轴和 Z 轴的比例。
- MotorRPMSMean（电机平均转速数组）：这是一个包含 100 个单精度浮点数（float）的数组，用于存储每个飞机的电机平均转速。每个元素表示相应飞机的电机平均转速，以转每分钟（RPM）为单位。

Multi3DData100New（100 机数据 2）

```
struct Multi3DData100New {
 int checksum;
 uint16 IDs[100];
 uint16 VehileTypes[100];
 float PosE[300];
 float AngEuler[300];
 uint16 ScaleXYZ[300];
 float MotorRPMSMean[100];
}
```

与 Multi3DData100（100 机数据 1）相似

- checksum 是数据校验位，1234567890 表示正常数据，1234567891 表示飞机始终贴合地面

Multi3DData100Time（100 机数据 3）

```
struct Multi3DData100Time {
 int checksum;
```

```
 uint16 IDs[100];
 uint16 VehileTypes[100];
 float PosE[300];
 float VelE[300];
 float AngEuler[300];
 float PWMs[800];
 double runnedTime[100];
}
```

与 Multi3DData100（100 机数据 1）相似，只是多出了时间戳

- checksum 是数据校验位，1234567890 表示正常数据，1234567891 表示飞机始终贴合地面

RflySim3D 的命令行数据

Ue4CMD

```
struct Ue4CMD {
 int checksum;
 char data[252];
}
```

作用解释

存入 1 条 RflySim3D 控制台命令

参数解释

- checksum（校验和）：这是一个整数字段，可能用于存储控制台命令数据的校验和，以确保数据的完整性和正确性。
- data（数据）：这是一个长度为 252 的字符数组，用于存储控制台命令数据。

```
struct Ue4CMD {
 int checksum;
 char data[52];
}
```

Ue4CMDNet

```
struct Ue4CMDNet { //总长度为 96
                int checksum; //校验位，这里应该是 1234567897
                char data[92];
            } i92s
```

作用解释

存入 1 条 RflySim3D 控制台命令

参数解释

- checksum（校验和）：这是一个 4 字节的整数，用于存储控制台命令数据

的校验和，以确保数据的完整性和正确性。

- data（数据）：一个 92 字节的字符数组，用于存储命令数据。这是将要发送给 RflySim3D 的实际命令所存储的地方。

图像数据

VisionSensorReq

```
struct VisionSensorReq {
 uint16 checksum; //数据校验位，12345
 uint16 SeqID; //内存序号 ID
 uint16 TypeID; //传感器类型 ID
 uint16 TargetCopter; //绑定的目标飞机      //可改变
 uint16 TargetMountType; //绑定的类型     //可改变
 uint16 DataWidth;   //数据或图像宽度
 uint16 DataHeight; //数据或图像高度
 uint16 DataCheckFreq; //检查数据更新频率
 uint16 SendProtocol[8]; //传输类型（共享内存、UDP 传输无压缩、UDP 视频串流），IP 地址，端口号，...
 float CameraFOV;  //相机视场角（仅限视觉类传感器）  //可改变
 float SensorPosXYZ[3]; // 传感器安装位置     //可改变
 float SensorAngEular[3]; //传感器安装角度    //可改变
 float otherParams[8]; //预留的八位数据位
}
```

作用解释

用于表示一个视觉传感器的请求信息，包括传感器的各种参数和属性。收到该信息后发送图像数据包：imageHeaderNew

参数解释

- checksum（数据校验位）：这是一个无符号整数（uint16），用于存储数据的校验和，以确保数据的完整性和正确性。
- SeqID（内存序号 ID）：这是一个无符号整数（uint16），用于标识请求的序号或 ID，最多为 32。
- TypeID（传感器类型 ID）：这是一个无符号整数（uint16），用于标识传感器的类型或类别。TypeID 属于[1,6]，456 是激光雷达

  1:RGB 图（免费版只支持 RGB 图），

  2:深度图，

  3:灰度图，

  4：点云相对于机体坐标系，

  5：点云相对于大地坐标系；

  6：相对于机体坐标系下的花瓣式扫描点云
- TargetCopter（绑定的目标飞机）：这是一个无符号整数（uint16），用于

指定传感器绑定的目标飞机。

- TargetMountType（绑定的类型）：这是一个无符号整数（uint16），用于指定传感器的绑定类型。
- DataWidth（数据或图像宽度）：这是一个无符号整数（uint16），表示传感器生成的数据或图像的宽度。
- DataHeight（数据或图像高度）：这是一个无符号整数（uint16），表示传感器生成的数据或图像的高度。
- DataCheckFreq（检查数据更新频率）：这是一个无符号整数（uint16），表示检查数据更新的频率。
- SendProtocol（传输类型）：这是一个包含 8 个无符号整数（uint16）的数组，用于指定数据的传输类型和相关参数。包括传输方式（共享内存、UDP 传输等）、IP 地址、端口号等信息。

  第 1 位表示发送模式

  0：共享内存（免费版只支持共享内存），

  1：UDP 直传 png 压缩，

  2：UDP 直传图片不压缩，

  3：UDP 直传 jpg 压缩

  第 2-5 位表示 IP 地址

  第 6 位表示端口号

- CameraFOV（相机视场角）：这是一个单精度浮点数（float），仅在传感器类型为视觉类传感器时有效，用于表示相机的视场角度。
- SensorPosXYZ（传感器安装位置）：这是一个包含 3 个单精度浮点数（float）的数组，用于表示传感器的安装位置，包括相对于飞机的 X、Y 和 Z 坐标。
- SensorAngEular（传感器安装角度）：这是一个包含 3 个单精度浮点数（float）的数组，用于表示传感器的安装角度，包括俯仰、偏航和滚转角度。
- otherParams（预留的八位数据位）：这是一个包含 8 个单精度浮点数（float）的数组，用于存储其他相关参数预留的数据位。

Ue4EKFInit

```
struct Ue4EKFInit {
 int checksum;
 int CopterID;
 int initCode;
}
```

作用解释

用于表示 扩展卡尔曼滤波器 (EKF) 的初始化信息

参数解释

- checksum（数据校验位）：这是一个整数字段，用于存储数据的校验和，以确保数据的完整性和正确性。
- CopterID（飞机 ID）：这是一个整数字段，用于标识扩展卡尔曼滤波器的初始化是针对哪个飞机或无人机的。
- initCode（初始化代码）：这是一个整数字段，表示扩展卡尔曼滤波器的初始化状态。

蓝图数据

Ue4ExtMsgFloat（扩展蓝图）

```
struct Ue4ExtMsgFloat {
 int checksum;
 int CopterID;
 double runnedTime;
 float pwm[16];
}
```

作用解释

用于传递与 Copter 相关的状态信息，包括时间戳和 PWM 值。会调用指定 Copter 的蓝图的 ActuatorInputsExt 函数

参数解释

- checksum（数据校验位）：这是一个整数字段，用于存储数据的校验和，以确保数据的完整性和正确性。
- CopterID（飞机 ID）：这是一个整数字段，用于标识消息是针对哪个飞机。
- runnedTime（运行时间）：这是一个双精度浮点数（double），表示消息的时间戳，
- pwm（脉冲宽度调制）：这是一个包含 16 个双精度浮点数（double）的数组，用于存储 8 位 PWM 之外的值。

Ue4ExtMsg（扩展蓝图）

```
struct Ue4ExtMsg {
 int checksum;
 int CopterID;
 double runnedTime; //Current  stamp (s)
 float pwm[16];
```

```
}
```

与 Ue4ExtMsgFloat（扩展蓝图）类似

Copter 附加到其他 Copter 的模式

VehicleAttatch25

```
struct VehicleAttatch25 {
 int checksum;//1234567892
 int CopterIDs[25];
 int AttatchIDs[25];
 int AttatchTypes[25];
//0：正常模式，1：相对位置不相对姿态，2：相对位置+偏航，3：相对位置+姿态
}
```

作用解释

描述多个飞机之间的附属关系，最多将 25 架飞机附加到其他最多 25 架飞机上

参数解释

- checksum（数据校验位）：这是一个整数字段，用于存储数据的校验和。
- CopterIDs（飞机 ID 数组）：这是一个包含 25 个整数元素的数组。每个元素对应一个作为附件的 copter，存储其标识号（ID）。
- AttatchIDs（附属目标 ID 数组）：这是一个包含 25 个整数元素的数组。每个元素对应一个被依附的 copter，存储其标识号（ID）。
- AttatchTypes（附件类型数组）：这是一个包含 25 个整数元素的数组。每个元素对应一个 copter，存储附属关系类型。

  0：正常模式（无相对关系）

  1：相对位置不相对姿态

  2：相对位置+偏航

  3：相对位置+姿态

Cesium 地图的 GPS 坐标

Ue4CMDGPS

```
struct Ue4CMDGPS {
 int checksum;
 int CopterID;
 double GPS[3];
}
```

作用解释

修改 cesium 地图中目标飞机的 GPS 坐标

参数解释

- checksum（数据校验位）：这是一个整数字段，用于存储数据的校验和，以确保数据的完整性和正确性。
- CopterID（飞机 ID）：这是一个整数字段，用于标识指令发送给哪个飞行器。
- GPS（GPS 坐标数组）：这是一个包含 3 个双精度浮点数（double）的数组，用于存储 GPS 坐标信息。GPS 坐标包括纬度、经度和海拔高度。

Copter 显示的字体

Ue4CopterMsg

```
struct Ue4CopterMsg {
 int checksum; //1234567899
 int CopterID;
 int dispFlag;
 int RGB[3];
 float dispTime;
 float fontSize;
 char data[120];
};
```

作用解释

用于设置每个 copter 头顶显示的标签内容

参数解释

- checksum（数据校验位）：这是一个整数字段，用于存储数据的校验和，以确保数据的完整性和正确性。
- CopterID（飞机 ID）：这是一个整数字段，用于标识消息是针对哪个飞行器或无人机的。如果 CopterID 的值小于或等于 0，则表示该消息适用于所有飞行器，即所有飞行器都会处理这条消息。
- dispFlag（显示标志）：这是一个整数字段，用于控制消息的显示方式。
- RGB（颜色信息）：这是一个包含 3 个整数元素的数组，用于表示消息的颜色。RGB 数组的三个元素分别代表红色、绿色和蓝色的颜色分量，用于定义消息的文本颜色。
- dispTime（显示时间）：这是一个浮点数字段，表示消息的显示时间，通常以秒为单位。
- fontSize（字体大小）：这是一个浮点数字段，表示消息文本的字体大小。
- data（消息内容）：这是一个字符数组，最多包含 120 个字符，用于存储

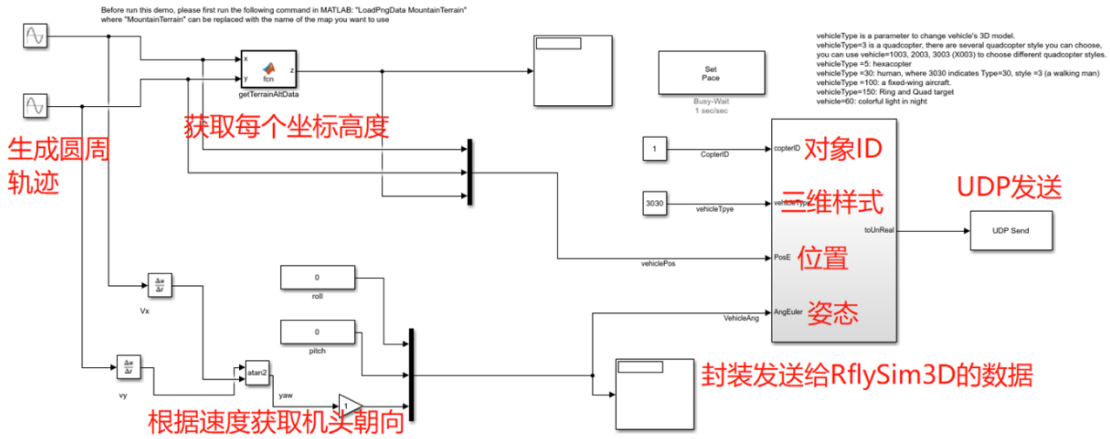
消息的文本内容。如果未指定消息内容（data），则默认应该显示飞机的 ID。

10. 外部接口文件

10.1 Simulink 接口函数

10.1.1 使用说明

只需确保 Simulink 模块中调用的函数与该模块处于同一目录下，确保其被正确调用，运行该模块同时打开 RflySim3D,RflySim3D 收到 UDP 包，会自动执行相应的指令。



10.1.2 模块介绍

数据发送接口

RflyCameraPosAng.m

在 MATLAB 中进入含有该函数的路径

打开 RflySim3D，在 Matlab 命令行中调用该函数即可

```
RflyCameraPosAng(0,0,-10,0,-30,0)
```

作用与 RflySim3D 控制台命令 RflyCameraPosAng（重设相机）相同

RflySendUE4CMD.m

在 MATLAB 中进入含有该函数的路径

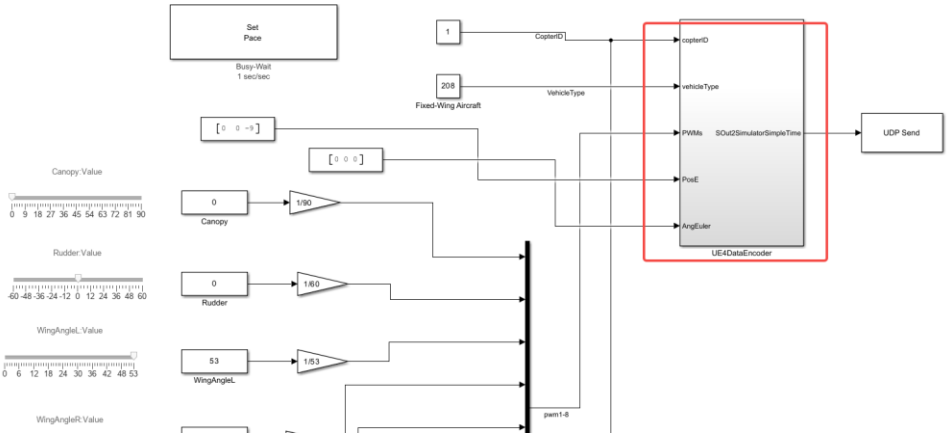
打开 RflySim3D，在 Matlab 命令行中调用该函数即可

```
RflySendUE4CMD(uint8('RflyChangeMapbyName Grasslands'))
```

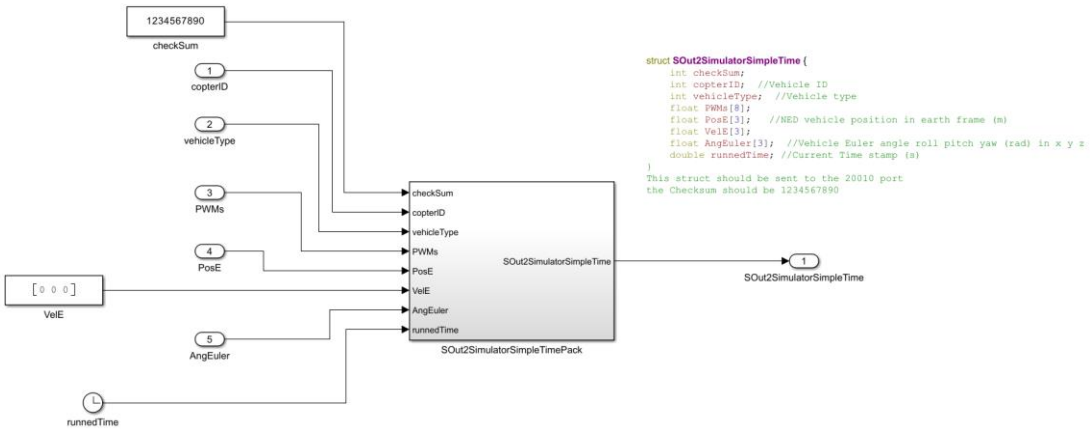
作用与 python 接口中的“sendUE4Cmd”函数相同

UE4DataEncoder

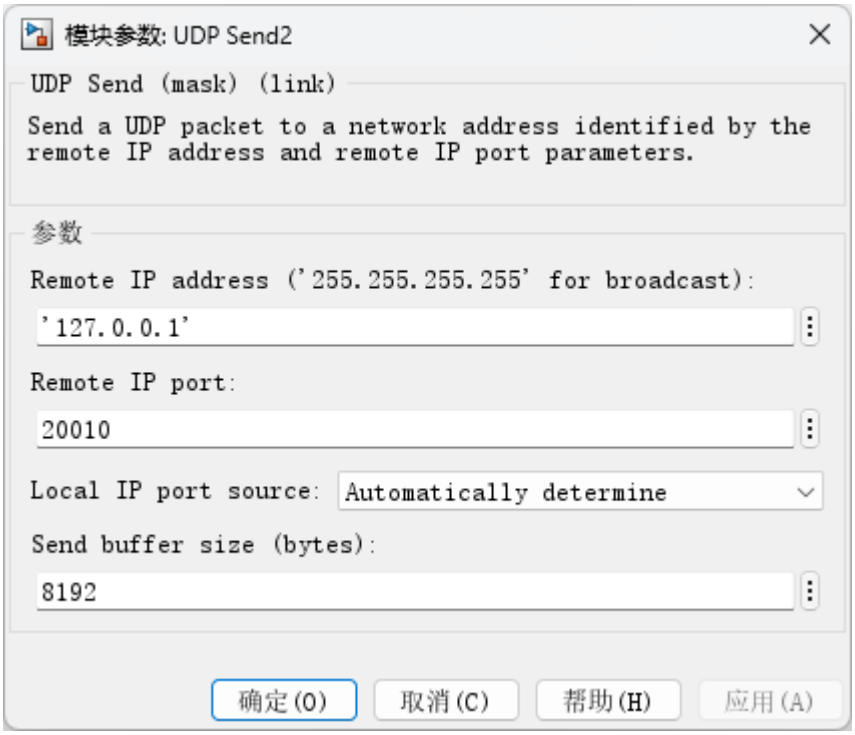
用于传输指定 copter 的 1-8 位电机数据给蓝图接口 8.4.1 ActuatorInputs 接口，参数与 RflySetActuatorPWMs（触发蓝图接口）相同



数据存入 SOut2SimulatorSimpleTime（单机数据 4）结构体

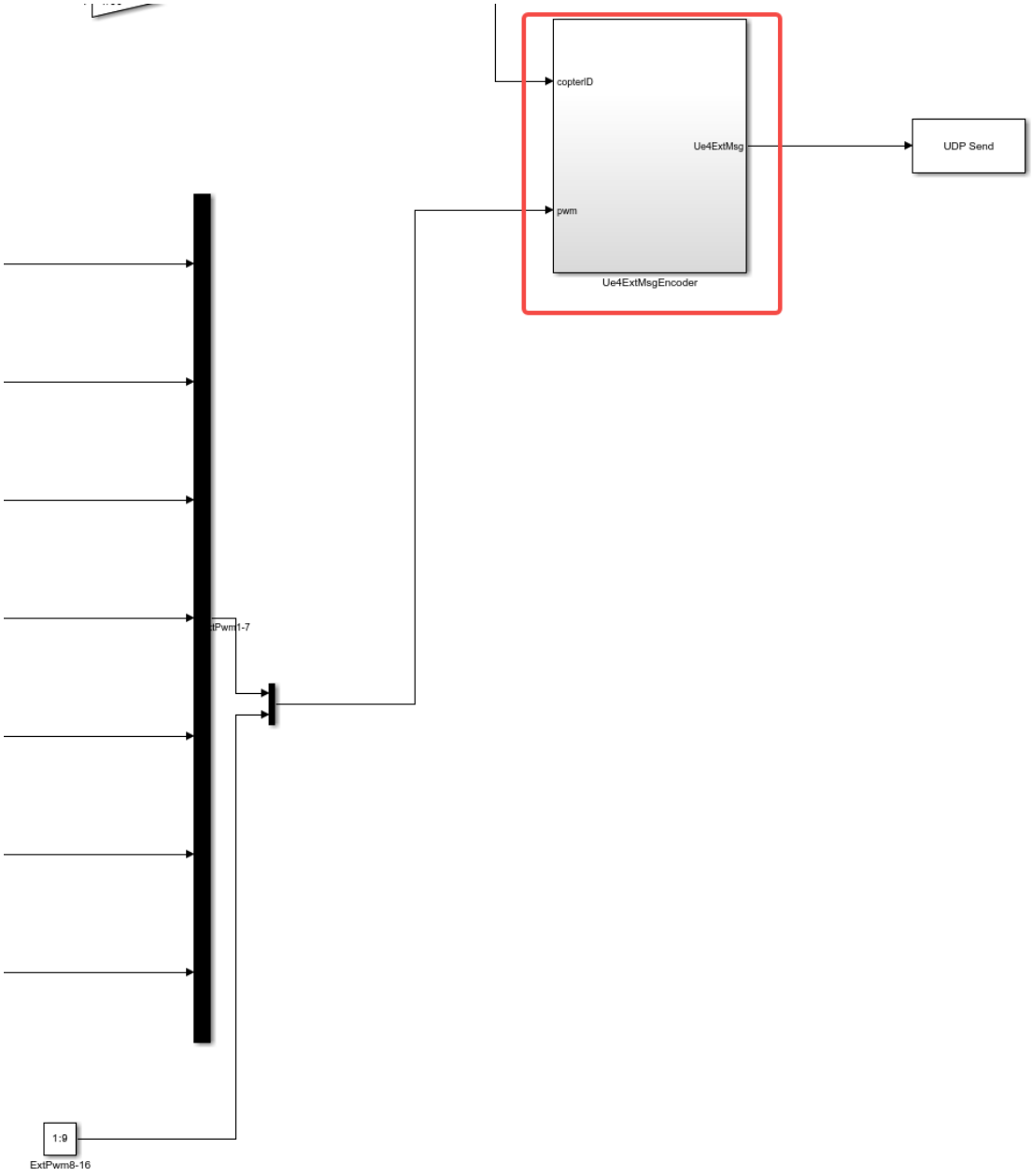


发送到本机 20010++1 系列端口(20010~20029)

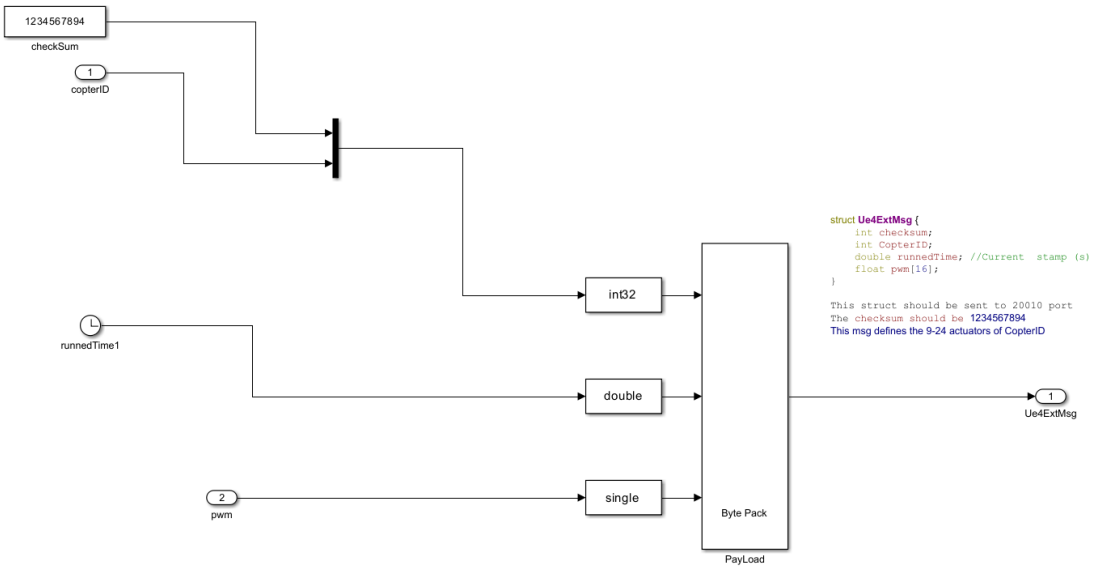


Ue4ExtMsgEncoder

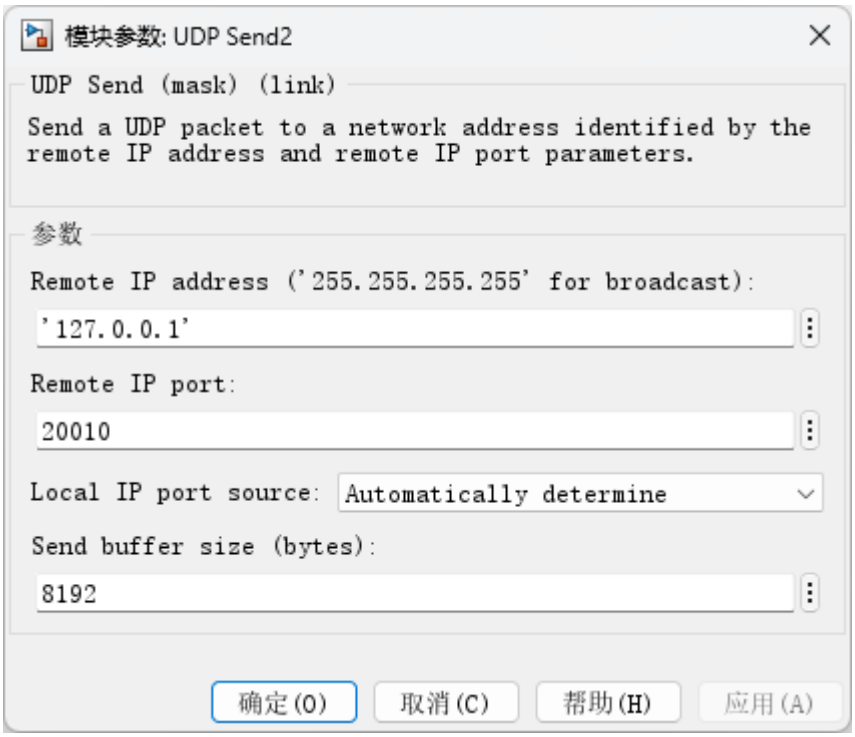
用于传输指定 copter 的 9-24 位电机数据给蓝图扩展接口 8.4.2 ActuatorInputsExt 接口，参数与 RflySetActuatorPWMsExt（触发扩展蓝图接口） 相同



数据存入 Ue4ExtMsg（扩展蓝图）结构体



发送到[本机 20010++1 系列端口(20010~20029)]



获取地形接口

LoadPngData.m

在 MATLAB 中进入含有该函数的路径

打开 RflySim3D，在 Matlab 命令行中调用该函数即可

```
LoadPngData 地图名称
```

该函数会依次搜索：本路径下的“map”文件夹和“【安装路径】\CopterSim\external\map”文件夹，(【安装路径】默认是 C:\PX4PSP，根据平台安装配置决定)，确保地形文件在上述文件夹之一。运行完上述脚本后，会生成一个“地图名称.mat”文件来存储高度图矩阵数据。

可自行查看如下算法，主要是导入 png 为矩阵文件，加入校准数据，再转存为高度图矩阵

```
    function LoadPngData(pngMapName)

    if ~exist('pngMapName','var')
        pngMapName='MountainTerrain';
    end

    filelocPath='map';
    fileLocPng = [filelocPath,'\',pngMapName,'.png'];
    if ~exist(fileLocPng,'file')
        filelocPath='..\..\..\CopterSim\external\map';
        fileLocPng = [filelocPath,'\',pngMapName,'.png'];
        if ~exist(fileLocPng,'file')
            matPath=[userpath,'\Add-Ons\Toolboxes\PX4PSP\code\+codertarget\+pixhawk\+CMAKE_Ut
ils\FirmwareVersion.mat'];
            if exist(matPath,'file')
                AA=load(matPath);
                filelocPath=[AA.RflyPSP_Px4_Base_Dir,'\CopterSim\external\map'];
                fileLocPng = [filelocPath,'\',pngMapName,'.png'];
                if ~exist(fileLocPng,'file')
                    error(['Cannot find file: ',pngMapName,'.png file']);
                end
            else
                error(['Cannot find file: ',pngMapName,'.png file']);
            end

        end

    end

    fileLocTxt = [filelocPath,'\',pngMapName,'.txt'];
    if ~exist(fileLocTxt,'file')
        error(['Cannot find file: ',pngMapName,'.txt']);
    end

    fileID = fopen(fileLocTxt);
    m_readData_cell = textscan(fileID,'%f',9,'Delimiter',',');
    m_readData = m_readData_cell{1};
    [m,n]=size(m_readData);
    if m~=9 || n~=1
        error(['Cannot parse data in ',fileLocTxt]);
    end

    rowmap  = imread(fileLocPng);
    rowmap = double(rowmap)-32768;
    [rows, columns] = size(rowmap);
```

```
PosScaleX = (m_readData(1)-m_readData(4))/(columns-1);
PosScaleY = (m_readData(2)-m_readData(5))/(rows-1);

PosOffsetX = m_readData(4);
PosOffsetY = m_readData(5);

intCol = int32((m_readData(7)-PosOffsetX)/PosScaleX + 1);
intRow = int32((m_readData(8)-PosOffsetY)/PosScaleY + 1);
% constainXY(intRow,intCol);

heightInit = double(rowmap(1,1));
heightFirst = double(rowmap(rows,columns));
heightThird = double(rowmap(intRow,intCol));


if abs(heightThird-heightFirst)<=abs(heightThird-heightInit)
    if abs((heightInit-heightThird))>10
        PosScaleZ = (m_readData(6)-m_readData(9))/((heightInit-heightThird));
    else
        PosScaleZ = 1;
    end
else
    if abs(heightThird-heightFirst)>10
        PosScaleZ = (m_readData(3)-m_readData(9))/((heightFirst-heightThird));
    else
        PosScaleZ = 1;
    end
end

intPosInitZ = heightInit;
PosOffsetZ = m_readData(6);

xMax=abs(m_readData(1)/100);
yMax=abs(m_readData(2)/100);

binmap= -(PosOffsetZ + ((rowmap)-intPosInitZ)*PosScaleZ)/100.0;

save('MapHeightData','binmap','PosOffsetX','PosScaleX','PosOffsetY','PosScaleY');
```

getTerrainAltData.m

在 MATLAB 中进入已有该函数以及“地图名称.mat”文件的路径

打开 RflySim3D，在 Matlab 命令行中调用该函数即可

```
getTerrainAltData(x,y)
```

getTerrainAltData 函数的作用是输入地图的 x，y 坐标，输出当前地形高度 z。通过该函数可以获取地形中任意位置的高度信息，从而可以创建出紧贴地表的运动轨迹。

可自行查看其算法如下：

```
function zz = getTerrainAltData(xin,yin)

%The map for simulation
mapname='MapHeightData';
```

```
persistent binmap;
persistent PosOffsetX;
persistent PosScaleX;
persistent PosOffsetY;
persistent PosScaleY;
if isempty(binmap)

%     if ~exist([mapname,'.mat'],'file')
%         LoadPngData(mapname);
%     end

    SS=load([mapname,'.mat']);
    binmap=SS.binmap;
    PosOffsetX=SS.PosOffsetX;
    PosScaleX=SS.PosScaleX;
    PosOffsetY=SS.PosOffsetY;
    PosScaleY=SS.PosScaleY;

end


intCol = (xin*100-PosOffsetX)/PosScaleX+1;
intRow = (yin*100-PosOffsetY)/PosScaleY+1;

intColInt=floor(intCol);
intRowInt = floor(intRow);
a=intCol-intColInt;
b=intRow-intRowInt;

intRowInt1=intRowInt+1;
intColInt1=intColInt+1;

[m,n]=size(binmap);
if intColInt<1
    intColInt=1;
    intColInt1=1;
    a=0;
end


if intColInt>=n
    intColInt=n;
    intColInt1=intColInt;
    a=0;
end


if intRowInt<1
    intRowInt=1;
    intRowInt1=1;
    b=0;
end
```

```
    if intRowInt>=m
        intRowInt=m;
        intRowInt1=intRowInt;
        b=0;
    end

    zz=binmap(intRowInt,intColInt)*(1-b)*(1-a)+binmap(intRowInt1,intColInt)*b*(1-a)+binmap(in
tRowInt,intColInt1)*(1-b)*a+binmap(intRowInt1,intColInt1)*b*a;
```

其它接口

GenSwarmPos12.m

GenSwarmPos30.m

GenSwarmPos100.m

GenSwarmPos2PC200.m

10.2 Python 接口库文件 UE4CtrlAPI.py

10.2.1 使用说明

示例如下：

```
    # 导入所需库
    import time
    import math
    import sys

    # 导入 RflySim APIs
    import UE4CtrlAPI as UE4CtrlAPI

    # 创建一个新的 MAVLink 通信实例，UDP 发送端口（CopterSim 的接收端口）是 20100
    ue = UE4CtrlAPI.UE4CtrlAPI()

    # sendUE4Cmd: 用于修改场景显示样式的 RflySim3D API
```

```
    # 格式：ue.sendUE4Cmd(cmd,windowID=-1)，其中 cmd 是一个命令字符串，windowID 是接收的窗口编号（假设同时打开多个 RflySim3D 窗口），windowID =-1 表示发送到所有窗口
    # 示例：RflyChangeMapbyName 命令表示切换地图（场景），下面的字符串是地图名称，这里将切换所有打开的窗口切换到草地图
    ue.sendUE4Cmd(b"RflyChangeMapbyName Grasslands")
    time.sleep(2)
```

在导入必要的依赖库和 RflySim3D 的接口函数 UE4CtrlAPI.py 后，还需新建要调用的类对应的通信实例，如这里的 ue，对应 UE4CtrlAPI.py 文件下 UE4CtrlAPI 类对应的实例。

10.2.2 类定义

所有类的初始化方法有两个构造函数，一个是默认的构造函数，将属性初始化为零。另一个是使用给定的参数 iv 进行构造，参数 iv 是一个包含数据的列表，按顺序设置属性的值。

10.2.2.1 PX4SILIntFloat 类（初始化控制模式）

PX4SILIntFloat 是一个用于输出到 CopterSim DLL 模型的数据的类。它包含以下方法：

__init__方法

```
    def __init__(self):
        self.checksum = 0
        self.CopterID = 0
        self.inSILInts = [0, 0, 0, 0, 0, 0, 0, 0]
        self.inSILFLoats = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

    def __init__(self, iv):
        self.checksum = iv[0]
        self.CopterID = iv[1]
        self.inSILInts = iv[2:10]
        self.inSILFLoats = iv[10:30]
```

参数：

- checksum：校验位
- CopterID：消息对应的 Copter ID
- inSILInts：输出到 CopterSim DLL 模型的 SILInts 数据
- inSILInts 的第一个整数用于表征和修改状态，相应位为 1 时表示系统处于相应的状态。例如，第一个位表示仿真模式，当接收到的 inSILInts[0] 的第一个位为 1 时，表示系统进入仿真模式。只有当 inSILInts[0] 的第一个位为 1 时，才会设置一次状态，否则综合模型将继续使用原有状态。原有状态可以来自外部设置值，也可以是内部状态自动转换。例如，接收到起飞命令后，首先切换到起飞模式，起飞完成后自动切换到定点模式。
- inSILFLoats：输出到 CopterSim DLL 模型的 SILFoats 数据。inSILFloats

用于存放实际的数据，其具体含义可以根据 inSILInts 的设定而变化。例如，前 3 个浮点数表示位置，但具体是哪个坐标系下的位置由 inSILInts 控制。具体的协议如下：

inSILFloats[0-2]：位置

inSILFloats[3-5]：速度或遥控器的俯仰、滚转、偏航信号。其中，inSILFloats[3] 可以作为速率使用。

inSILFloats[6-8]：加速度

inSILFloats[9-11]：姿态控制使用的欧拉角，对于用户来说更加直观。

inSILFloats[12-14]：姿态速率

inSILFloats[15]：油门

10.2.2.2 reqVeCrashDat 类（UE 返回碰撞相关数据）

reqVeCrashDat 是这个类是由 RflySim3D 发送的结构体，是当 RflySim3D 开启数据回传模式时（使用“RflyReqVehicleData 1”控制台命令），会发送出来的结构体。里面主要是与碰撞相关的数据，会被发往组播 ip“224.0.0.10：20006”。它包含以下方法：

__init__方法

```
def __init__(self):
      self.checksum = 1234567897
      self.copterID = 0
      self.vehicleType = 0
      self.CrashType = 0
      self.runnedTime = 0
      self.VelE = [0, 0, 0]
      self.PosE = [0, 0, 0]
      self.CrashPos = [0, 0, 0]
      self.targetPos = [0, 0, 0]
      self.AngEuler = [0, 0, 0]
      self.MotorRPMS = [0, 0, 0, 0, 0, 0, 0, 0]
      self.ray = [0, 0, 0, 0, 0, 0]
      self.CrashedName = ""
```

参数

- copterID：表示该结构体是哪个 Copter 的数据。
- vehicleType：表示该 Copter 的样式
- CrashType：表示碰撞物体类型，-2 表示地面，-1 表示场景静态物体，0 表示无碰撞，1 以上表示被碰飞机的 ID 号
- runnedTime：当前飞机的时间戳
- VelE：当前飞机的速度（米/秒）

- PosE：当前飞机的位置（北东地，单位米）
- CrashPos：碰撞点的坐标（北东地，单位米）
- targetPos：被碰物体的中心坐标（北东地，单位米）
- AngEuler：当前飞机的欧拉角（Roll，Pitch，Yaw，弧度）
- MotorRPMS：当前飞机的电机转速
- Ray：飞机的前后左右上下扫描线
- CrashedName：被碰物体的名字

10.2.2.3 CoptReqData（UE 返回模型数据）

__init__（初始化配置）

```
def __init__(self):
        self.checksum = 0 # 1234567891 作为校验
        self.CopterID = 0 #飞机 ID
        self.PosUE = [0,0,0]
        self.angEuler = [0,0,0]
        self.boxOrigin = [0,0,0]
        self.BoxExtent = [0,0,0]
        self.timestmp = 0
        self.hasUpdate=True
```

由 RflySim3D 返回的飞机数据，调用 RflySim3D 的“RflyReqObjData”命令后，RflySim3D 根据该命令的参数，发送的某个 Copter 的信息（向 RflySim3D 发送命令时附带了该 Copter 的 ID）。

参数

- checksum：是数据的校验位，用于检验数据传输过程中是否发生了异常。
- CopterID：表示该消息对应的 copter ID。
- PosUE：表示该 Copter 的位置（米，北东地）
- angEuler：表示该 Copter 的角度（弧度，Roll、Pitch、Yaw）
- boxOrigin：包围盒的几何中心（米，北东地）
- BoxExtent：包围盒的边长的一半（米，X 轴(forward)、Y 轴(right)、Z 轴(up)）
- Timestmp：时间戳
- hasUpdate：这个值不是 RflySim3D 发过来的

10.2.2.4 ObjReqData（UE 返回目标物体数据）

__init__（初始化配置）

```
def __init__(self):
        self.checksum = 0 # 1234567891 作为校验
        self.seqID = 0
        self.PosUE = [0,0,0]
        self.angEuler = [0,0,0]
        self.boxOrigin = [0,0,0]
        self.BoxExtent = [0,0,0]
        self.timestmp = 0
        self.ObjName=''
        self.hasUpdate=True
```

由 RflySim3D 返回的数据，调用 RflySim3D 的“RflyReqObjData”命令后，RflySim3D 根据该命令的参数，发送的某个物体的信息（向 RflySim3D 发送命令时，附带了该物体的 Name）。

参数

- checksum：是数据的校验位，用于检验数据传输过程中是否发生了异常。
- seqID：恒为 0。（Obj 不是 Copter，它没有 ID 可供返回）
- PosUE：表示该物体的位置（米，北东地）
- angEuler：表示该 Obj 的角度（弧度，Roll、Pitch、Yaw）
- boxOrigin：包围盒的几何中心（米，北东地）
- BoxExtent：包围盒的边长的一半（米，X 轴(forward)、Y 轴(right)、Z 轴(up)）
- Timestmp：当前场景已存在时间（秒，场景存在的时间(因为它不是 Copter，没有时间戳)）
- ObjName：该物体的名字
- hasUpdate：这个值不是 RflySim3D 发过来的

10.2.2.5 CameraData（UE 返回相机数据）

__init__（初始化配置）

```
def __init__(self):
        self.checksum = 0 #1234567891 作为校验
        self.SeqID = 0
        self.TypeID = 0
        self.DataHeight = 0
```

```
            self.DataWidth = 0
            self.CameraFOV = 0
            self.PosUE = [0,0,0]
            self.angEuler = [0,0,0]
            self.timestmp = 0
            self.hasUpdate=True
```

由 RflySim3D 返回的数据，调用 RflySim3D 的“RflyReqObjData”命令后，RflySim3D 根据该命令的参数，发送的某个相机（视觉传感器）的信息（向 RflySim3D 发送命令时，附带了该相机（视觉传感器）的 SeqID）。

参数

- checksum：是数据的校验位，用于检验数据传输过程中是否发生了异常。
- seqID：视觉传感器的 ID
- TypeID：视觉传感器的类型（RPG、深度图等）
- DataHeight：数据高度(像素)
- DataWidth：数据宽度(像素)
- CameraFOV：相机水平视场角（单位度）
- PosUE：表示该物体的位置（米，北东地）
- angEuler：表示该 Camera 的角度（弧度，Roll、Pitch、Yaw）
- Timestmp：时间戳
- hasUpdate：这个值不是 RflySim3D 发过来的

10.2.2.6 UE4CtrlAPI 类（UE 场景控制命令）

__init__（初始化配置）

```
    def __init__(self, ip='127.0.0.1'):
            self.ip = ip

            self.udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # Create sock
et
            self.udp_socket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
            self.udp_socketUE4 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # Create s
ocket
            self.udp_socketUE4.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
            self.inSilVect = []
            self.inReqVect = []

            self.stopFlagUE4=True
            self.CoptDataVect=[]
            self.ObjDataVect=[]
            self.CamDataVect=[]
```

这是该类的构造函数

参数

- ip：这是一个字符串变量，表示与 RflySim3D 进行通信的 IP 地址。默认值为 '127.0.0.1'，表示本地主机。
- udp_socket 和 udp_socketUE4：这是两个用于网络通信的 UDP 套接字（socket）变量。其中 udp_socket 是设置为广播模式的套接字，udp_socketUE4 则具有 SO_REUSEADDR 标志。这些套接字用于与 RflySim3D 进行通信，是发送和接收网络数据的主要方式。
- inSilVect、inReqVect、CoptDataVect、ObjDataVect 和 CamDataVect：这些是列表变量，用于存储飞机相关数据。包括飞行器状态、静态物体数据和相机数据。
- stopFlagUE4：这是一个布尔（Boolean）变量，用于控制 RflySim3D 是否处于停止状态。

sendUE4Cmd（控制台命令）

```
def sendUE4Cmd(self,cmd,windowID=-1):
```

用于发送 RflySim3D 控制台命令控制三维场景显示。结构体见 Ue4CMD

参数

- cmd：要发送的命令字符串。这个参数可以是 RflySim3D 的显示样式控制命令、地图切换命令、摄像机移动命令、车辆模型改变命令等。
- windowID：可选参数，表示 RflySim3D 窗口的 ID，默认值为 -1。窗口 ID 可以用于向特定的窗口发送命令，用于控制具体的 RflySim3D 实例。如果 windowID 小于 0，则表示发送命令给所有窗口实例。如果 windowID 大于等于 0，则表示发送命令给指定的窗口 ID。

sendUE4CmdNet（局域网内广播控制台命令，该功能仅限完整版以上支持）

```
def sendUE4CmdNet(self,cmd):
```

用于发送 RflySim3D 控制台命令控制局域网（LAN）内的所有 RflySim3D 实例。结构体见 Ue4CMDNet

参数

- cmd：要发送的命令字符串。这个参数可以是 RflySim3D 的显示样式控制命令、地图切换命令、摄像机移动命令、车辆模型改变命令等。

sendUE4LabelID（设置 Copter 标签内容）

```
def sendUE4LabelID(self,CopterID=0,Txt='',fontSize=30,RGB=[255,0,0],windowID=-1):
```

用于设置指定 copter 的头顶显示标签。结构体见 Ue4CopterMsg

参数

- CopterID（默认值为 0）：表示无人机或飞行器的 ID。这个参数用于指定要显示标签的飞行器。
- Txt：表示要显示的文本。这是标签中实际要显示的内容。
- fontSize（默认值为 30）：表示字体的大小。用于指定显示文本的字体大小。
- RGB（默认为 [255, 0, 0]）：表示标签的颜色。这是一个包含三个整数的列表，分别表示红色、绿色和蓝色的值。
- windowID（默认值为 -1）：表示窗口的 ID。如果指定了窗口 ID，标签将仅在该窗口上显示。如果 windowID 为负数，标签将在所有窗口上显示。

sendUE4LabelMsg（设置 Copter 标签下方内容）

```
def sendUE4LabelMsg(self,CopterID=0,Txt='',fontSize=30,RGB=[255,0,0],dispTime=0,dispFlag=-1,windowID=-1):
```

用于在指定 copter 头顶新建 1 行显示内容。结构体见 Ue4CopterMsg

参数

- CopterID（默认值为 0）：飞行器的 ID，用于指定要显示消息的飞行器。如果 CopterID 小于等于 0，则所有飞行器都会显示消息；如果 CopterID 大于 0，则只有对应的飞行器会显示消息。
- Txt（默认为空字符串）：要显示的文本内容。
- fontSize（默认值为 30）：字体的大小，用于指定文本显示的字体大小。
- RGB（默认为 [255, 0, 0]）：标签的颜色，以 RGB 格式表示。每个分量的值范围为 0-255，分别表示红、绿、蓝的颜色分量。
- dispTime（默认值为 0）：消息的消失时间（单位秒）。如果值小于 0，消息会立即消失；如果值等于 0，消息会一直显示；如果值大于 0，消息会在指定的秒数后消失。
- dispFlag（默认值为 -1）：显示标志，用于控制消息的显示方式。不同的 dispFlag 值对应不同的显示行为，包括逐层累加、清空所有消息、更新指定行（ID 行）和更新 1-5 行消息等。

  dispFlag <0 且 dispTime>=0 表示依次累加的方式添加消息

  dispFlag <0 且 dispTime<0 表示清空所有消息

dispFlag =0 且 dispTime>=0 表示更新 ID 行（第 0 行数据）

dispFlag >=1 and <5 表示更新 1 到 5 行的消息，注意此时 dispTime<0 会删除本消息，若>=0 会更新消息

dispFlag>当前消息数，则切换到累加模式

- windowID（默认值为 -1）：窗口的 ID。如果指定了窗口 ID，标签消息将只在指定的窗口上显示。如果 windowID 为负数，则标签消息将在所有窗口上显示。

sendUE4Attatch（Copter 附加关系）

```
def sendUE4Attatch(
    self: Self@UE4CtrlAPI,
    CopterIDs: Any,
    AttatchIDs: Any,
    AttatchTypes: Any,
    windowID: int = -1
)
```

将指定 copter 附加到其他 copter 上，最多 25 个 copter。结构体见 VehicleAttatch25

参数

- CopterIDs：飞行器的 ID 列表，用于指定要附着到其他飞行器上的飞行器。
- AttatchIDs：附着目标飞行器的 ID 列表，用于指定要将飞行器附着到哪些飞行器上。
- AttatchTypes：附着类型的列表，用于控制附着的方式。每个元素的取值范围为 0-3，分别表示不同的附着方式：

  0：正常模式

  1：相对位置不相对姿态

  2：相对位置加偏航（不相对俯仰和滚转）

  3：相对位置加全姿态（俯仰、滚转和偏航）
- windowID（默认值为 -1）：窗口的 ID。如果指定了窗口 ID，标签消息将只在指定的窗口上显示。如果 windowID 为负数，则标签消息将在所有窗口上显示。

sendUE4Pos（创建/更新模型）

```
(method) def sendUE4Pos(
    self: Self@UE4CtrlAPI,
    copterID: int = 1,
    vehicleType: int = 3,
```

```
        MotorRPMSMean: int = 0,
        PosE: Any = [0, 0, 0],
        AngEuler: Any = [0, 0, 0],
        windowID: int = -1
    ) -> None
```

将位置和角度信息发送到 RflySim3D 以创建新的 3D 模型或更新旧模型的状态。

将最多 8 位电机数据传给蓝图模型的 ActuatorInputs 接口

结构体见 SOut2SimulatorSimple（单机数据 2）

参数

- copterID（默认值为 1）：飞行器的 ID，指定要创建或更新状态的飞行器。
- vehicleType（默认值为 3）：飞行器的类型。根据实际情况，可以指定不同的类型来创建或更新不同类型的飞行器模型。
- MotorRPMSMean（默认值为 0）：电机转速的均值。这个参数用于控制电机的转速，可以影响模型的运动状态。
- PosE（默认值为 [0, 0, 0]）：位置信息，表示飞行器当前的位置。这是一个包含三个元素的列表，分别表示飞行器在 x、y、z 坐标轴上的位置。
- AngEuler（默认值为 [0, 0, 0]）：欧拉角信息，表示飞行器当前的姿态。这是一个包含三个元素的列表，分别表示飞行器的俯仰、滚转和偏航角度。
- windowID（默认值为 -1）：窗口的 ID。如果指定了窗口 ID，位置和角度信息将仅应用于指定的窗口。如果 windowID 为负数，则位置和角度信息将应用于所有窗口。

sendUE4PosNew（创建/更新模型）

```
(method) def sendUE4PosNew(
    self: Self@UE4CtrlAPI,
    copterID: int = 1,
    vehicleType: int = 3,
    PosE: Any = [0, 0, 0],
    AngEuler: Any = [0, 0, 0],
    VelE: Any = [0, 0, 0],
    PWMs: Any = [0] * 8,
    runnedTime: int = -1,
    windowID: int = -1
) -> None
```

与 sendUE4Pos（创建/更新模型）类似，结构体见 SOut2SimulatorSimpleTime（单机数据 4）

参数

- self：表示类的实例对象，用于访问类的成员变量和其他方法。
- copterID（默认值为 1）：飞行器的 ID，用于指定要创建或更新状态的飞行器。

- vehicleType（默认值为 3）：飞行器的类型，根据实际情况可以指定不同的类型来创建或更新不同类型的飞行器模型。
- PosE（默认值为 [0, 0, 0]）：位置信息，表示飞行器当前的位置。这是一个包含三个元素的列表，分别表示飞行器在 x、y、z 坐标轴上的位置。
- AngEuler（默认值为 [0, 0, 0]）：欧拉角信息，表示飞行器当前的姿态。这是一个包含三个元素的列表，分别表示飞行器的俯仰、滚转和偏航角度。
- VelE（默认值为 [0, 0, 0]）：速度信息，表示飞行器当前的速度。这是一个包含三个元素的列表，分别表示飞行器在 x、y、z 轴上的速度。
- PWMs（默认值为 [0，0，0，0，0，0，0，0] ）：PWM（脉宽调制）信息，表示飞行器的电机控制信号。这是一个包含八个元素的列表，用于控制飞行器电机的转速或推力。
- runnedTime（默认值为 -1）：运行时间，以毫秒为单位，表示飞行器的运行时间。
- windowID（默认值为 -1）：窗口的 ID。如果指定了窗口 ID，位置、姿态、速度等信息将仅应用于指定的窗口。如果 windowID 为负数，则这些信息将应用于所有窗口。

sendUE4Pos2Ground（创建/更新模型）

```
(method) def sendUE4Pos2Ground(
    self: Self@UE4CtrlAPI,
    copterID: int = 1,
    vehicleType: int = 3,
    MotorRPMSMean: int = 0,
    PosE: Any = [0, 0, 0],
    AngEuler: Any = [0, 0, 0],
    windowID: int = -1
) -> None
```

将位置和角度信息发送到 RflySim3D 以创建新的 3D 模型或更新旧模型在地面上的状态，校验和=1234567891 用于告诉 UE4 生成始终适合地面结构的对象。将最多 8 位电机数据传给蓝图模型的 ActuatorInputs 接口

结构体见 SOut2SimulatorSimple（单机数据 2）

参数

- copterID（默认值为 1）：飞行器的 ID，指定要创建或更新状态的飞行器。
- vehicleType（默认值为 3）：飞行器的类型。根据实际情况，可以指定不同的类型来创建或更新不同类型的飞行器模型。
- MotorRPMSMean（默认值为 0）：电机转速的均值。这个参数用于控制电机的转速，可以影响模型在地面上的状态。
- PosE（默认值为 [0, 0, 0]）：位置信息，表示飞行器当前在地面上的位置。

这是一个包含三个元素的列表，分别表示飞行器在 x、y、z 坐标轴上的位置。

- AngEuler（默认值为 [0, 0, 0]）：欧拉角信息，表示飞行器当前的姿态。这是一个包含三个元素的列表，分别表示飞行器的俯仰、滚转和偏航角度。
- windowID（默认值为 -1）：窗口的 ID。如果指定了窗口 ID，位置和角度信息将仅应用于指定的窗口。如果 windowID 为负数，则位置和角度信息将应用于所有窗口。

sendUE4PosScale（创建/更新模型）

```
(method) def sendUE4PosScale(
    self: Self@UE4CtrlAPI,
    copterID: int = 1,
    vehicleType: int = 3,
    MotorRPMSMean: int = 0,
    PosE: Any = [0, 0, 0],
    AngEuler: Any = [0, 0, 0],
    Scale: Any = [1, 1, 1],
    windowID: int = -1
) -> None
```

将位置和角度信息发送到 RflySim3D 以创建新的 3D 模型或通过更改比例更新旧模型的状态 。将最多 8 位电机数据传给蓝图模型的 ActuatorInputs 接口

结构体见 SOut2SimulatorSimple1（单机数据 5）

参数

- self：表示类的实例对象，用于访问类的成员变量和其他方法。
- copterID（默认值为 1）：飞行器的 ID，用于指定要创建或更新状态的飞行器。
- vehicleType（默认值为 3）：飞行器的类型，根据实际情况可以指定不同的类型来创建或更新不同类型的飞行器模型。
- MotorRPMSMean（默认值为 0）：电机转速的均值，用于控制电机的转速，影响模型的运动状态。
- PosE（默认值为 [0, 0, 0]）：位置信息，表示飞行器当前的位置。这是一个包含三个元素的列表，分别表示飞行器在 x、y、z 坐标轴上的位置。
- AngEuler（默认值为 [0, 0, 0]）：欧拉角信息，表示飞行器当前的姿态。这是一个包含三个元素的列表，分别表示飞行器的俯仰、滚转和偏航角度。
- Scale（默认值为 [1, 1, 1]）：比例信息，用于更改模型的比例。这是一个包含三个元素的列表，分别表示飞行器在 x、y、z 轴上的比例。
- windowID（默认值为 -1）：窗口的 ID。如果指定了窗口 ID，位置、姿态和比例信息将仅应用于指定的窗口。如果 windowID 为负数，则位置、姿态和比例信息将应用于所有窗口。

sendUE4PosScale2Ground（创建/更新模型）

```
(method) def sendUE4PosScale2Ground(
    self: Self@UE4CtrlAPI,
    copterID: int = 1,
    vehicleType: int = 3,
    MotorRPMSMean: int = 0,
    PosE: Any = [0, 0, 0],
    AngEuler: Any = [0, 0, 0],
    Scale: Any = [1, 1, 1],
    windowID: int = -1
) -> None
```

将位置和角度信息发送到 RflySim3D 以创建新的 3D 模型或通过更改比例来更新旧模型的状态，这里采用校验和=1234567891 告诉 UE4 生成始终适合地面的对象。将最多 8 位电机数据传给蓝图模型的 ActuatorInputs 接口

结构体见 SOut2SimulatorSimple1（单机数据 5）

参数

- self：表示类的实例对象，用于访问类的成员变量和其他方法。
- copterID（默认值为 1）：飞行器的 ID，用于指定要创建或更新状态的飞行器。
- vehicleType（默认值为 3）：飞行器的类型，根据实际情况可以指定不同的类型来创建或更新不同类型的飞行器模型。
- MotorRPMSMean（默认值为 0）：电机转速的均值，用于控制电机的转速，影响模型的运动状态。
- PosE（默认值为 [0, 0, 0]）：位置信息，表示飞行器当前的位置。这是一个包含三个元素的列表，分别表示飞行器在 x、y、z 坐标轴上的位置。
- AngEuler（默认值为 [0, 0, 0]）：欧拉角信息，表示飞行器当前的姿态。这是一个包含三个元素的列表，分别表示飞行器的俯仰、滚转和偏航角度。
- Scale（默认值为 [1, 1, 1]）：比例信息，用于更改模型的比例。这是一个包含三个元素的列表，分别表示飞行器在 x、y、z 轴上的比例。
- windowID（默认值为 -1）：窗口的 ID。如果指定了窗口 ID，位置、姿态和比例信息将仅应用于指定的窗口。如果 windowID 为负数，则位置、姿态和比例信息将应用于所有窗口。

sendUE4PosFull（创建/更新模型）

```
(method) def sendUE4PosFull(
    self: Self@UE4CtrlAPI,
    copterID: Any,
    vehicleType: Any,
```

```
    MotorRPMS: Any,
    VelE: Any,
    PosE: Any,
    RateB: Any,
    AngEuler: Any,
    windowID: int = -1
) -> None
```

将位置和角度信息发送到 RflySim3D 以创建新的 3D 模型或更新旧模型的状态。

将最多 8 位电机数据传给蓝图模型的 ActuatorInputs 接口

结构体见 SOut2Simulator（单机数据 1）

参数

- self：表示类的实例对象，用于访问类的成员变量和其他方法。
- copterID：飞行器的 ID，用于指定要创建或更新状态的飞行器。
- vehicleType：飞行器的类型，根据实际情况可以指定不同的类型来创建或更新不同类型的飞行器模型。
- MotorRPMS（默认值为 [0，0，0，0，0，0，0，0] ）：电机转速信息，用于控制飞行器的电机转速。
- VelE：速度信息，表示飞行器当前水平面坐标系（NED）的速度。
- PosE：位置信息，表示飞行器当前水平面坐标系（NED）的位置。
- RateB：角速度信息，表示飞行器当前机体坐标系的角速度。
- AngEuler：欧拉角信息，表示飞行器当前的姿态。
- windowID（默认值为 -1）：窗口的 ID。如果指定了窗口 ID，上述信息将仅应用于指定的窗口。如果 windowID 为负数，则这些信息将应用于所有窗口。

sendUE4ExtAct（触发扩展蓝图接口）

```
(method) def sendUE4ExtAct(
    self: Self@UE4CtrlAPI,
    copterID: int = 1,
    ActExt: Any = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    windowID: int = -1
) -> None
```

将 8 位电机数据之外的最多 16 位执行器数据传给蓝图模型的扩展蓝图接口 ActuatorInputsExt

参数

- self：表示类的实例对象，用于访问类的成员变量和其他方法。
- copterID（默认值为 1）：飞行器的 ID，用于指定要执行外部动作的飞行器。
- ActExt（默认值为 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]）：外部动作信

息，用于指定飞行器执行的外部动作。这是一个包含 16 个元素的列表，每个元素表示一个外部动作的值。

- windowID（默认值为 -1）：窗口的 ID。如果指定了窗口 ID，外部动作信息将仅应用于指定的窗口。如果 windowID 为负数，则外部动作信息将应用于所有窗口。

sendUE4PosSimple（创建/更新模型）

```
(method) def sendUE4PosSimple(
    self: Self@UE4CtrlAPI,
    copterID: Any,
    vehicleType: Any,
    PWMs: Any,
    VelE: Any,
    PosE: Any,
    AngEuler: Any,
    runnedTime: int = -1,
    windowID: int = -1
) -> None
```

将位置和角度信息发送到 RflySim3D 以创建新的 3D 模型或更新旧模型的状态。将最多 8 位电机数据传给蓝图模型的 ActuatorInputs 接口

结构体见 SOut2SimulatorSimpleTime（单机数据 4）

参数

- self：表示类的实例对象，用于访问类的成员变量和其他方法。
- copterID：飞行器的 ID，用于指定要创建或更新状态的飞行器。这是一个任意类型的参数，表示飞行器的标识。
- vehicleType：飞行器的类型，根据实际情况可以指定不同的类型来创建或更新不同类型的飞行器模型。
- PWMs（默认值[0，0，0，0，0，0，0，0]）：PWM（脉宽调制）信息，表示飞行器的电机控制信号。
- VelE：速度信息，表示飞行器当前的速度。
- PosE：位置信息，表示飞行器当前的位置。
- AngEuler：欧拉角信息，表示飞行器当前的姿态。
- runnedTime（默认值为 -1）：时间戳，以毫秒为单位
- windowID（默认值为 -1）：窗口的 ID。如果指定了窗口 ID，上述信息将仅应用于指定的窗口。如果 windowID 为负数，则这些信息将应用于所有窗口。

sendUE4PosScale100（创建 100 个 Copter）

```
(method) def sendUE4PosScale100(
    self: Self@UE4CtrlAPI,
    copterID: Any,
    vehicleType: Any,
    PosE: Any,
    AngEuler: Any,
    MotorRPMSMean: Any,
    Scale: Any,
    isFitGround: bool = False,
    windowID: int = -1
) -> None
```

将位置和角度信息发送到 RflySim3D，一次创建 100 架飞机。结构体见 Multi3DData100New（100 机数据 2）

参数

- self：表示类的实例对象，用于访问类的成员变量和其他方法。
- copterID：飞行器的 ID，用于指定要创建或更新状态的飞行器。这是一个任意类型的参数，表示飞行器的标识。
- vehicleType：飞行器的类型，根据实际情况可以指定不同的类型来创建或更新不同类型的飞行器模型。
- PosE：位置信息，表示飞行器当前的位置。
- AngEuler：欧拉角信息，表示飞行器当前的姿态。
- MotorRPMSMean（默认值为 0）：电机转速的均值，用于控制电机的转速，影响模型的运动状态。
- Scale：比例信息，用于更改模型的比例。
- isFitGround（默认值为 False）：是否适应地面。如果设置为 True，则模型会根据地面结构进行调整；如果设置为 False，则不会进行调整。
- windowID（默认值为 -1）：窗口的 ID。如果指定了窗口 ID，上述信息将仅应用于指定的窗口。如果 windowID 为负数，则这些信息将应用于所有窗口。

sendUE4PosScalePwm20（创建 20 个 Copter）

```
(method) def sendUE4PosScalePwm20(
    self: Self@UE4CtrlAPI,
    copterID: Any,
    vehicleType: Any,
    PosE: Any,
    AngEuler: Any,
    Scale: Any,
    PWMs: Any,
```

```
    isFitGround: bool = False,
    windowID: int = -1
) -> None
```

将位置和角度信息发送到 RflySim3D，一次创建 20 架飞机。结构体见 Multi3DData2New（20 机数据 6）

参数

- self：表示类的实例对象，用于访问类的成员变量和其他方法。
- copterID：飞行器的 ID，用于指定要创建或更新状态的飞行器。这是一个任意类型的参数，表示飞行器的标识。
- vehicleType：飞行器的类型，根据实际情况可以指定不同的类型来创建或更新不同类型的飞行器模型。
- PosE：位置信息，表示飞行器当前的位置。
- AngEuler：欧拉角信息，表示飞行器当前的姿态。
- Scale：比例信息，用于更改模型的比例。
- PWMs（默认值为 0）：PWM（脉宽调制）信息，表示飞行器的电机控制信号。
- isFitGround（默认值为 False）：是否适应地面。如果设置为 True，则模型会根据地面结构进行调整；如果设置为 False，则不会进行调整。
- windowID（默认值为 -1）：窗口的 ID。如果指定了窗口 ID，上述信息将仅应用于指定的窗口。如果 windowID 为负数，则这些信息将应用于所有窗口。

getUE4Pos（获取 Copter 位置）

```
(method) def getUE4Pos(
    self: Self@UE4CtrlAPI,
    CopterID: int = 1
) -> (list | list[int])
```

获取指定 copter 的位置

getUE4Data（获取 Copter 数据）

```
(method) def getUE4Data(
    self: Self@UE4CtrlAPI,
    CopterID: int = 1
) -> (Any | Literal[0])
```

该函数可以获得 Copter 在 RflySim3D 中的数据，这个数据也是 reqVeCrashData 结构的。

initUE4MsgRec（启用监听）

```
(method) def initUE4MsgRec(self: Self@UE4CtrlAPI) -> None
```

初始化 self.udp_socketUE4，并且启动一个线程 t4(self.UE4MsgRecLoop)开始监听 224.0.0.10：20006，以及自身的 20006 端口。

endUE4MsgRec（终止监听）

```
(method) def endUE4MsgRec(self: Self@UE4CtrlAPI) -> None
```

停止线程 t4(self.UE4MsgRecLoop)的监听

UE4MsgRecLoop（循环）

```
(method) def UE4MsgRecLoop(self: Self@UE4CtrlAPI) -> None
```

UE4 消息侦听循环

函数解释

它是监听 224.0.0.10：20006，以及自身的 20006 端口的处理函数，用于处理 RflySim3D 或 CopterSim 返回的消息，一共有 6 种消息：

1) 长度为 12 字节的 CopterSimCrash：

```
struct CopterSimCrash {
    int checksum;
    int CopterID;
    int TargetID;
}
```

由 RflySim3D 返回的碰撞数据，RflySim3D 中按 P 键时 RflySim3D 会开启碰撞检测模式，如果发生了碰撞，会传回这个数据，P+数字可以选择发送模式（0 本地发送，1 局域网发送，2 局域网只碰撞时发送），默认为本地发送。

2) 长度为 120 字节的 PX4SILIntFloat：

```
struct PX4SILIntFloat{
    int checksum;//1234567897
    int CopterID;
    int inSILInts[8];
    float inSILFLoats[20];
};
```

3) 长度为 160 字节的 reqVeCrashData：

```
struct reqVeCrashData {
    int checksum; //数据包校验码 1234567897
    int copterID; //当前飞机的 ID 号
    int vehicleType; //当前飞机的样式
    int  CrashType;//碰撞物体类型，-2 表示地面，-1 表示场景静态物体，0 表示无碰撞，1 以上表示被碰飞机的 ID 号
    double runnedTime; //当前飞机的时间戳
    float VelE[3]; // 当前飞机的速度
```

```
    float PosE[3]; //当前飞机的位置
    float CrashPos[3];//碰撞点的坐标
    float targetPos[3];//被碰物体的中心坐标
    float AngEuler[3]; //当前飞机的欧拉角
    float MotorRPMS[8]; //当前飞机的电机转速
    float ray[6]; //飞机的前后左右上下扫描线
    char CrashedName[20] = {0};//被碰物体的名字
}
```

前面介绍过这个类了，在开启数据回传的情况下，RflySim3D 会为所有 Copter 发送 reqVeCrashData（数据变化时发送，每秒一次）。

4) 长度为 56 字节的 CameraData：

```
struct CameraData { //56
    int checksum = 0;//1234567891
    int SeqID; //相机序号
    int TypeID;//相机类型
    int DataHeight;//像素高
    int DataWidth;//像素宽
    float CameraFOV;//相机视场角
    float PosUE[3]; //相机中心位置
    float angEuler[3];//相机欧拉角
    double timestmp;//时间戳
};
```

RflySim3D 根据 reqCamCoptObj 函数发送的命令，定时返回的该结构体，该程序接收到后会存放在 self.CamDataVect 中。

5) 长度为 64 字节的 CoptReqData：

```
struct CoptReqData { //64
    int checksum = 0; //1234567891 作为校验
    int CopterID;//飞机 ID
    float PosUE[3]; //物体中心位置（人为三维建模时指定，姿态坐标轴，不一定在几何中心）
    float angEuler[3];//物体欧拉角
    float boxOrigin[3];//物体几何中心坐标
    float BoxExtent[3];//物体外框长宽高的一半
    double timestmp;//时间戳
};
```

RflySim3D 根据 reqCamCoptObj 函数发送的命令，定时返回的该结构体，该程序接收到后会存放在 self.CoptDataVect 中。

6) 长度为 96 字节的 ObjReqData：

```
struct ObjReqData { //96
    int checksum = 0; //1234567891 作为校验
    int seqID = 0;
    float PosUE[3]; //物体中心位置（人为三维建模时指定，姿态坐标轴，不一定在几何中心）
    float angEuler[3];//物体欧拉角
    float boxOrigin[3];//物体几何中心坐标
    float BoxExtent[3];//物体外框长宽高的一半
    double timestmp;//时间戳
    char ObjName[32] = { 0 };//碰物体的名字
};
```

RflySim3D 根据 reqCamCoptObj 函数发送的命令，定时返回的该结构体，该程序接收到后会存放在 self.ObjDataVect 中。

收到的数据可以使用 getCamCoptObj 函数进行获取。

getCamCoptObj（获取物体数据）

```
(method) def getCamCoptObj(
    self: Self@UE4CtrlAPI,
    type: int = 1,
    objName: int = 1
) -> (Any | Literal[0])
```

从 RflySim3D 获得指定的数据，UE4MsgRecLoop 函数开启了监听 RflySim3D 的消息，并将监听到的三种存放到了 3 个列表中，此函数正是在这个列表中搜索数据，因此需要先使用 reqCamCoptObj 函数向 RflySim3D 请求相关的数据才行。

参数

- Type：0 表示相机，1 表示飞机，2 表示物体

reqCamCoptObj（指定窗口数据回传）

```
(method) def reqCamCoptObj(
    self: Self@UE4CtrlAPI,
    type: int = 1,
    objName: int = 1,
    windowID: int = 0
) -> None
```

向 RflySim3D 发送一个数据请求，可以请求场景中物体的数据（并不能创建新的物体，而是获得已存在物体的数据）。可以是视觉传感器、Copter、场景中普通的物体。获得的数据详见 CoptReqData 类、ObjReqData 类、CameraData 类

参数

- type：0 表示相机，1 表示飞机，2 表示物体
- objName：type 表示相机时，seqID；表示飞机时，objName 对应 CopterID；表示物体时，objName 对应物体名字
- windowID：表示想往哪个 RflySim3D 发送消息，默认是 0 号窗口。（不要给所有 RflySim3D 发送该数据，因为返回的值都是一样，没必要额外消耗性能）

10.2.2.7 UEMapServe（UE 获取地形接口）

__init__（初始化配置）

```
def __init__(self,name=''):
```

```
        if name == '':
            self.PosOffsetX=0
            self.PosScaleX=0
            self.PosOffsetY=0
            self.PosScaleY=0
            self.xMax=0
            self.yMax=0
            self.binmap= []
        else:
            self.LoadPngData(name)
```

LoadPngData（加载高度图）

```
(method) def LoadPngData(
    self: Self@UEMapServe,
    name: Any
) -> None
```

按如下方法调用

```
map = UE4CtrlAPI.UEMapServe()
map.LoadPngData("Grasslands")
```

参数

- name：地图名（也是不带扩展名的文件名）

函数解释：

用该函数时会在脚本目录下寻找并读取 name.txt 与 name.png，这是由 RflySim3D 生成的地图文件。（txt 文件指示了地图的大小与范围，png 文件则是地图的高度图），平台中每个地图都会有这两个文件，可以在“C:\PX4PSP\CopterSim\external\map”下找到。

调用该函数后，地图的信息就被读入内存了，相当于调用 getTerrainAltData 函数之前的初始化。

getTerrainAltData（获取地形）

```
(method) def getTerrainAltData(
    self: Self@UEMapServe,
    xin: Any,
    yin: Any
) -> Any
```

查询地图上某个点的高度（单位米）。

参数

- xin：查询点的 X 坐标（米）
- yin：查询点的 Y 坐标（米）

10.3 Redis 接口函数（开发中）

10.3.1 配置文件 RedisConfig.ini

```
[CommMode]
CommunicationMode = 1
[RedisConfig]
host = "192.168.31.79"
port = 6379
db = 0
memkey = "GLOBAL_CAMERA_PARAMETERS_REDIS_KEY"
r3d_rec_key = "GLOBAL_MESSAGE_REDIS_KEY"
r3d_send_key_20006 = "GLOBAL_MESSAGE_R3D_SEND_KEY_20006"
r3d_send_key_30100 = "GLOBAL_MESSAGE_R3D_SEND_KEY_30100"
```

10.3.2 使用说明

11. 常用特效接口汇总（开发中）

11.1 显示标签

快捷键

S（显/隐 CopterID）：

命令行（限个人高级版以上）

RflySetIDLabel（设置 CopterID 标签处显示）

RflySetMsgLabel（设置 CopterID 标签下方显示）

Python 接口（限个人高级版以上）

sendUE4LabelID（设置 Copter 标签内容）

sendUE4LabelMsg（设置 Copter 标签下方内容）

11.2 画线

T（开/关 Copter 轨迹记录）：

T+数字*（更改轨迹粗细为*号）：

11.3 天气

11.4 小地图

L（显/隐小地图）：

11.5 虚拟管道

O+803:O+数字*（生成 ClassID 为“*”的物体）：

11.6 通信特效

O+802:O+数字*（生成 ClassID 为“*”的物体）：

注意，通信特效实际上是一系列粒子的组合，单独创建一个粒子，会由于其生命周期太短而无法观测，需要持续创建。参考 0.ApiExps/e8_RflySim3DEffect/1.Comm

11.7 HelicopterPadDemo（直升机着陆场）

O+800:O+数字*（生成 ClassID 为“*”的物体）：

11.8 CirclePlaneDemo（环形平面）

O+810:O+数字*（生成 ClassID 为“*”的物体）：

11.9 SatelliteDemo（卫星）

O+811:O+数字*（生成 ClassID 为“*”的物体）：

11.10 SatelliteReceiveDemo（卫星接收）

O+812:O+数字*（生成 ClassID 为“*”的物体）：

11.11 BallonDemo（气球）

O+830:O+数字*（生成 ClassID 为“*”的物体）：

11.12 CycleDemo（圆环）

O+831:O+数字*（生成 ClassID 为“*”的物体）：

...

参考资料

[1]. 无