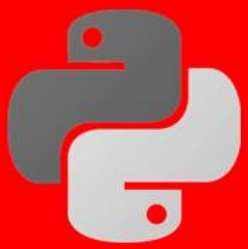


Simplified Chinese Version

# Unreal Engine 5 Programming

Shudong YANG

Leanpub

# 虚幻引擎 5 编程

## Unreal Engine 5 Programming


## Dr. Shudong YANG



**This book is for sale at http://leanpub.com/unreal-engine-5-programming**

**This version was published on 2026-07-31**



This is a **Leanpub** book. Leanpub empowers authors and publishers with the Lean Publishing process. **Lean Publishing** is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

# Citing this Book

If you found this book useful for your blog post, research article or product, I would be grateful if you would cite this book. You can cite the book like this:

Shudong YANG. ***Unreal Engine 5 Programming***[M]. Leanpub, 2026.

If you use the book as a reference, it would be great if you wrote me a line and told me what for. This is of course optional and only serves to satisfy my own curiosity and to stimulate interesting exchanges.

My email is shudong.yang@djtu.edu.cn

# 序

如果你曾经被 C++的指针逼疯过，或者被蓝图的连线绕晕过，那么恭喜你，这本书就是为你写的。

这是一本关于虚幻引擎 5 编程的书。

写这本书的初衷很简单：我自己学 UE 的时候，发现东西散落在各处。官方文档很全，但像一本百科全书；视频教程很生动，但翻找某个知识点时又不太方便。于是我想，能不能写一本书，既有体系的骨架，又有实战的血肉？

希望你读这本书的时候，不只是学会了怎么点按钮，而是慢慢理解了为什么这么设计。毕竟，工具会更新，版本会迭代，但那套底层的思维方式，才是真正能陪你走远的东西。

那么，准备好了吗？

**Shudong YANG**
2026/7/31

# AIGC Statement

During the creation of this book, generative artificial intelligence was used as a supporting tool. Specifically, AI assisted with content structuring, translation and polishing, and language refinement. The author has comprehensively reviewed, revised, and validated all AI-generated auxiliary content to ensure the accuracy and rigour of the final material.

**Shudong YANG**
2026/5/25

# 目录

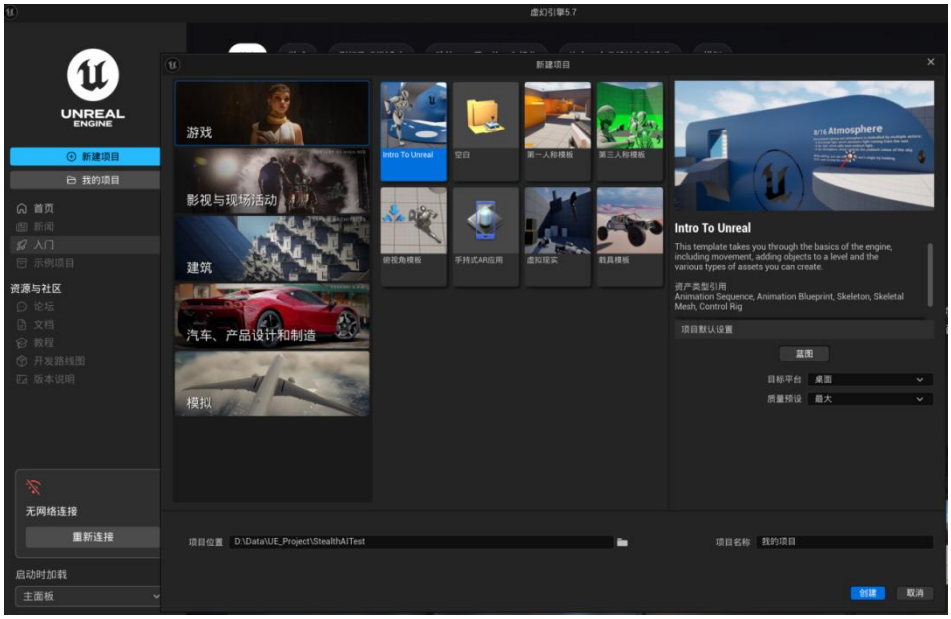

UE5 官方文档[1]中译中

# 1. 虚幻引擎版本演进

习近平总书记在庆祝中国共产党成立百周年大会上的讲话中指出，“以史为鉴，可以知兴替。我们要用历史映照现实、远观未来”。重视历史学习和历史教育，从 UE 历史中获得启迪，从 UE 版本更新[2]中提炼出法宝，是游戏人的一种优良传统。在学习历史中，更好地走向未来。

## 1.1. UE5.7 版本说明（节选，与编程相关）


虚幻引擎 5.7 版本[3]，继续完善 UE5 工具集。此版本优化了渲染、角色与动画、世界构建和程序化内容生成等。




UE5.7 登录页

[1] https://www.unrealengine.com/unreal-engine-5?lang=zh-CN
[2] https://dev.epicgames.com/documentation/unreal-engine/whats-new
[3] https://dev.epicgames.com/documentation/unreal-engine/unreal-engine-5-7-release-notes

## 1.1.1. 程序化内容生成

UE 引擎的程序化内容生成（Procedural Content Generation，以下简称 **PCG**）是一套强大的工具集，能帮开发者快速、自动地在 UE 中构建和迭代资产与整个世界。PCG 的出现改变了传统的游戏世界构建方式，让开发效率得到了质的飞跃。

**PCG Framework** 是 UE 内置的原生程序化生成系统，以 PCG 图表为核心，通过节点和点云数据来定义生成规则。可以在编辑器中实时调整并立刻看到结果。核心工作流基于点（Points）的生成。图表首先在 3D 空间（地形、样条线等）中生成带各类属性（变换、密度、颜色等）的点。

**图表与节点：**类似材质编辑器，通过连接和配置不同功能的节点（如筛选、变换、生成网格体等）构建复杂的生成逻辑。**主要特点：**1）**实时编辑与反馈：**图表修改结果会实时在视口更新，便于快速迭代。2）**可扩展性强：**从路边的栅栏到整个星球的生态群系，都能胜任。3）**运行时动态生成：**支持在游戏运行时动态生成内容，创造更具交互性的世界。4）**强大的调试工具：**提供节点调试信息、可视化密度等，帮助分析和优化生成逻辑。PCG 框架下有两个特色子系统，极大扩展了其能力边界：

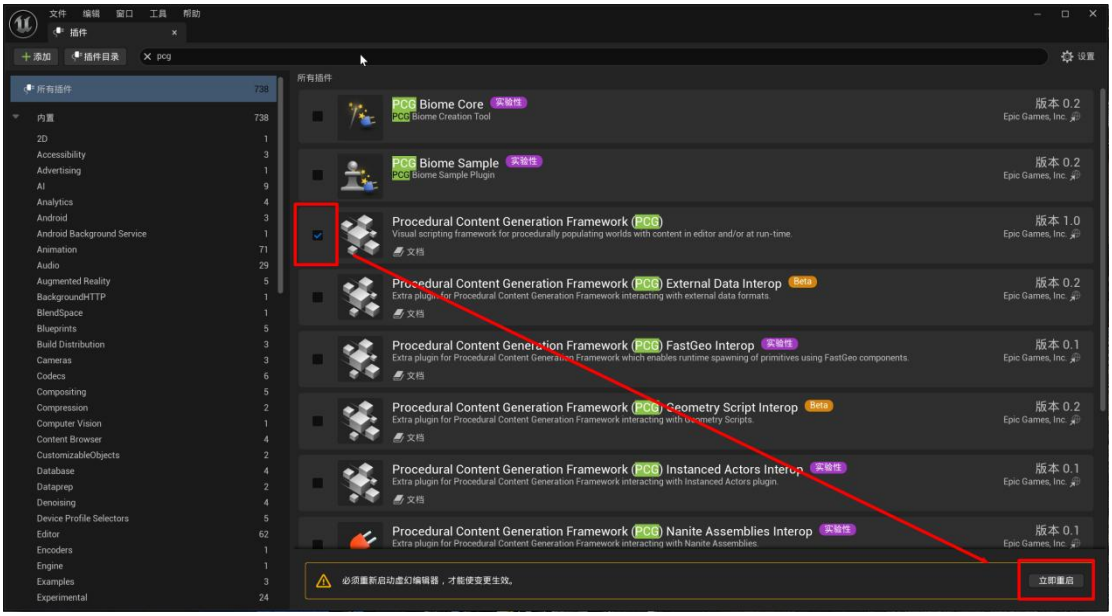
**PCG 子系统**

| 子系统 | 核心功能 | 主要应用场景 |
| --- | --- | --- |
| **Geometry Scripting** (几何体脚本) | 通过蓝图或 Python 直接在 UE 中创建和修改程序化网格体的 API | 关卡设计的快速原型搭建、创建工具进行网格体布尔运算，并能将结果烘焙成静态网格体 |
| **Shape Grammar** (形状语法) | 使用语法规则将符号替换为特定模块，实现**基于规则**的模块化生成 | 非常适合生成墙壁、栅栏、桥梁等沿样条线分布的构建系统。 |

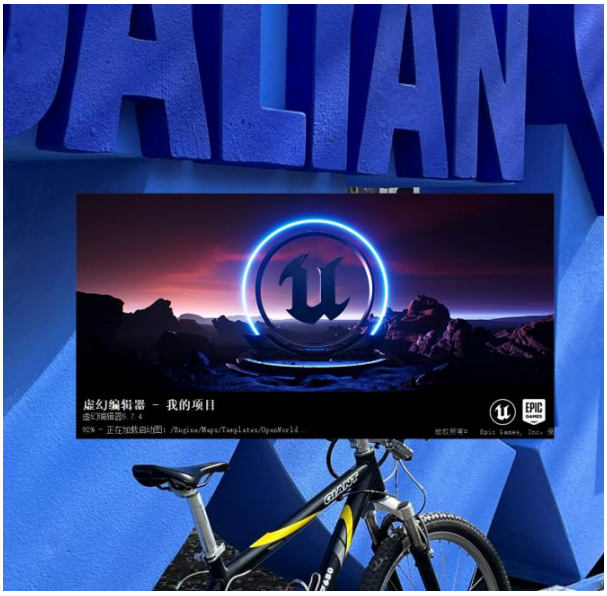
V5.7 版本的 PCG 变更如下：

**（1）PCG 编辑器模式**

虚幻引擎现已新增 **PCG 专用编辑器工具模式**。提供了利用 PCG 框架的可自定义工具库，如绘制样条线、绘制或创建体积，每项工具都有相关的 PCG 图表及其参数，可供实时编辑。



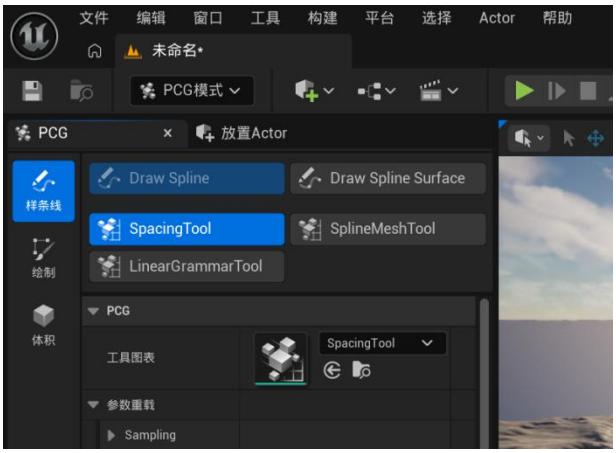
重启 UE：



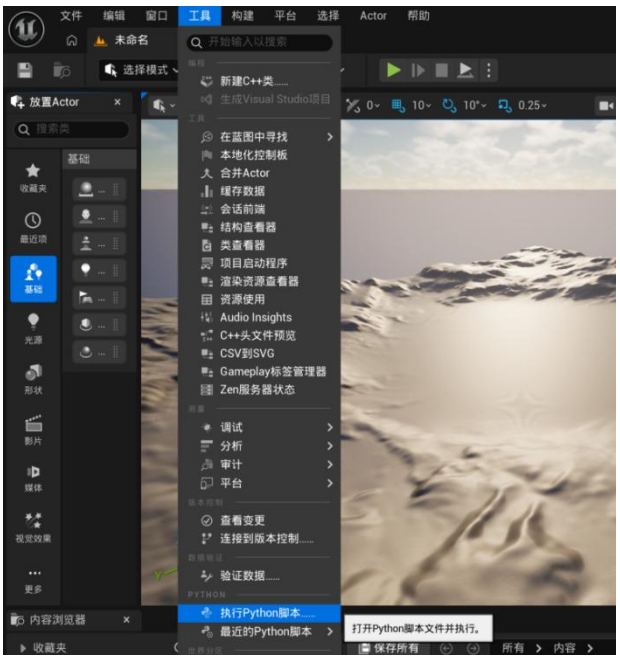

**UE5.7 和 002 号爱车 ATX680 同框**

“选择模式”出现 PCG：


可使用专用 PCG 图表工具，或以其为范例根据自身需求创建、扩展资源库。




另外，还可以调用 Python 接口：




**工具-执行 Python 脚本**

**（2）程序化植被编辑器**

程序化植被编辑器是一项新的实验性插件，可直接在虚幻引擎中创建和编辑植被网格体，并且支持 Nanite[4]植被。在此初始版本中，创建的程序化植被资产将通过植被编辑器[5]进行编辑。使用插件内容文件夹中提供的导入器节点和骨骼预设，可以：

- 通过重力、雕刻和缩放运算符修改树木形状。
- 调整树叶和树枝的分布。
- 分配不同的树叶模型。
- 导出至骨骼网格体或静态网格体，包括对 Nanite 植被组装的支持。
- 通过单个节点图表或多个节点图表创建多种变体，以便为项目构建独特的植被网格体资产库。

**（3）资产工作流**

UE5.7 版本可将 PCG 图表作为独立图表执行，无需世界上下文，也无需通过 PCG 组件生成图表，更新后支持：

- **支持资产创建工作流**，如仅需运行独立 PCG 图表即可通过几何体运算创建网格体、通过 GPU 运算生成纹理，甚至创建 PCG 数据资产。
- 基于 PCG 框架构建的资产创建工具和插件（如新版程序化植被编辑器）可充分利用其完整功能体系，并且未来会持续更新。

要启用此功能，可在图表设置中勾选新增的 Is Standalone Graph 选项。启用后，可通过右键菜单使用生成独立图表（Generate Standalone Graph）功能，并可在图表编辑器中使用独立图表的调试对象进行生成、检查及调试操作。

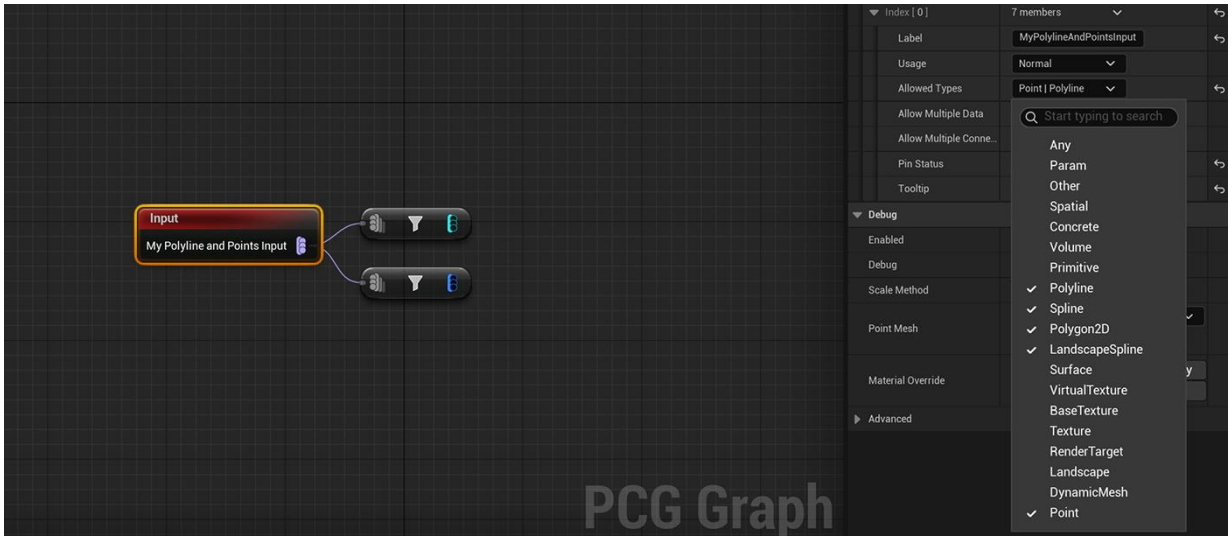
---

[4] **Nanite** 是虚幻引擎 5 引入的革命性**虚拟几何体系统**。彻底改变了高精度模型的渲染方式，能智能地只渲染你最终能从屏幕上感受到的细节。

[5] 基于 PCG 的节点图表编辑器

**（4）自定义数据类型**

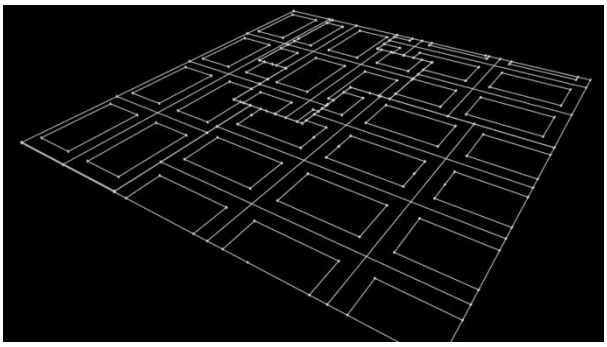
在初始版本中，PCG 框架内置的数据类型列表采用硬编码方式，导致在扩展框架或与外部插件建立互操作性时难以新增数据类型。




UE 5.7 版本现已支持在类型层次结构中添加任意自定义数据类型，从而扩展框架功能，或基于框架核心与功能集构建专属工具。同时，升级了类型选择器，从而更清晰地展示类型关系并支持多选操作。

**（5）Polygon2D 数据类型与运算符**

PCG 框架的数据类型列表新增了 Polygon2D 数据类型。该数据类型通过闭合形状表示区域范围，并且可转换为曲面或样条线数据，适用于采样或特定运算场景。



UE5 的“Polygon2D”并非单一结构，而是围绕 2D 多边形设计的系统，主要包含：1）核心几何数据结构、2）PCG 框架下的数据与功能这两层面。

其中，核心几何数据结构属于引擎的 GeometryCore 模块，使用 C++模板编写，能灵活支持不同数值精度。

- **TPolygon2：基础多边形**，定义为一个由 2D 顶点组成的列表。关键属性：Vertices（顶点数组）。基础操作：能添加顶点、计算面积（Area()）和包围盒（Bounds()），也支持倒角和裁剪。
- **TGeneralPolygon2：带孔洞的多边形**，用于表示内部有“洞”的复杂 2D 形状，比如一片有湖水的森林区域。关键属性：由 Outer（外部边界，TPolygon2 类型）和 Holes（孔洞数组）构成。核心功能：可添加孔洞并判断某点是否在区域内

为支持此数据类型，UE V5.7 版本新增了一组运算符：

- 创建 Polygon2D（Create Polygon 2D）：获取输入的点数据或样条线数据，并转换为 2D 多边形。
- 多边形运算（Polygon Operations）：支持多边形间的运算 （如交集/并集/差集）。 此外，可以使用 样条线路径对 2D 多边形进行切割，实现形状分割与隔离。 例如 在城市生成图表中，通过网格状样条数据切割大范围区域， 生成独立的城市街区。
- 裁剪路径（Clip Paths）：支持将样条线与 2D 多边形进行交集或差集运算。
- 偏移多边形（Offset Polygon）：对 2D 多边形进行偏移放大或缩小，同时正确处理重叠区域。
- 从 Polygon2D 创建曲面：将 2D 多边形转换为可采样的曲面，通过曲面采样节点在其区域内生成分布点。

**（6）GPU 快速几何体互操作**

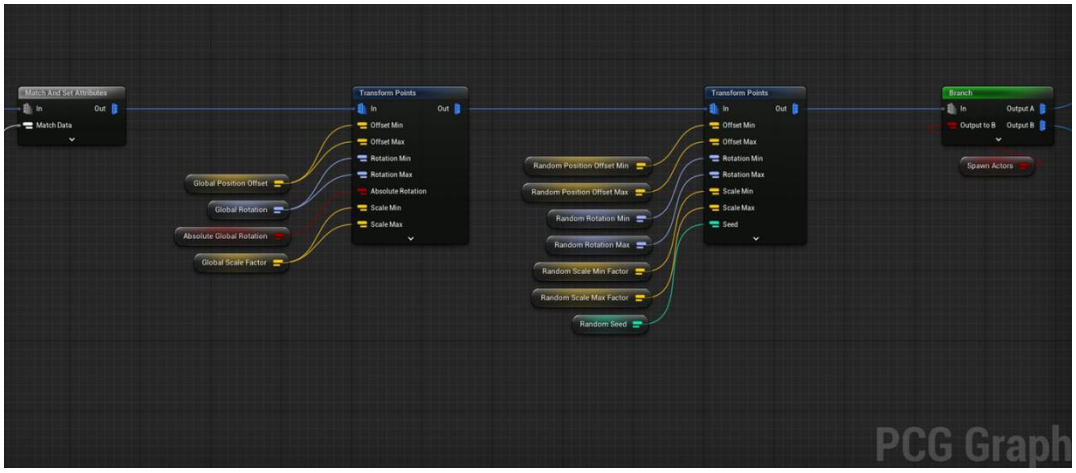
PCG GPU 现采用快速几何体组件，在使用框架生成并放置高密度静态网格体（如地面散布物和草地）时，可进一步提升游戏线程性能。 该功能无需使用分区执行体，并支持动态创建本地 PCG 组件。要启用此优化，请在项目中激活 PCG FastGeo Interop 插件，并将 CVARpcg.RuntimeGeneration.ISM.ComponentlessPrimitives 设为 1。

**（7）GPU 运算符和改进**

PCG Framework GPU 计算功能集将迎来以下新增功能：

- 场景捕捉（Scene Capture）：一个新节点，用于从组件原点执行场景捕捉，获取场景颜色或深度样本并进行散射。
- 纹理保存到资产（Save Texture to Asset）：一个新节点，用于将 PCGTextureData 烘焙成 UTexture2D 资产。
- 支持重载（Support for Overrides）：GPU 节点现已支持参数重载，包括自定义 HLSL 节点的元素总数（Num Elements）。
- 升级现有的本地运算符，支持 GPU 计算：
- 剔除 Actor 边界之外的点（Cull Points Outside Actor Bounds）
- 变换点（Transform Points）

**（8）UX 改进**




UX 又迎来了一系列的改进，方便你在处理 PCG 节点图表时使用，这些改进包括：

- 参数和重载类型着色与验证：现在参数、重载引脚、常量输出引脚及其边缘都会根据其类型显示不同的颜色，从而与蓝图类型颜色的可视化语言相匹配。 当输入可以是任何类型或未定义时，引脚和边缘会使用白色作为显示颜色。
- 参数最小值、最大值和单位值：现在你可以通过最小/最大值和显示单位来配置图表参数。
- 数据视口控制：我们添加了更多图形编辑器内的数据视口控制和选项，帮助你在具体上下文中预览数据。

**（9）节点和运算符**

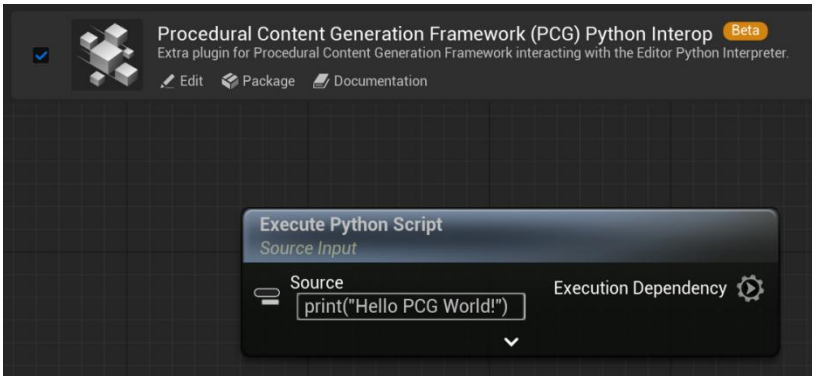
虚幻引擎 5.7 引入了多个本地 PCG 元素和新选项，包括：

- 样条相交（Spline Intersection）：在 3D 中查找相交的样条曲线，并在相交处添加控制点或返回相交点。
- 分割样条（Split Splines）：根据 alpha、距离、关键点或控制点谓词将样条线分割成多个样条线。
- 获取片段（Get Segment）：从点、样条线或多边形 2D 返回线段索引点或样条线。
- 生成样条线网格体组件重载（Spawn Spline Mesh Component Overrides）：现在生成样条线网格体时 已正确遵循组件类，并支持在生成样条线网格体节点中 重载组件属性。
- 新 PCG 蓝图元素类（New PCG Blueprint Element Classes）：新增多项 PCG 元素类，包含基于范围的可重载函数，以支持 5.6 版本原生节点中 引入的数组结构格式变更。采用新元素 可避免额外数据转换，从而在处理蓝图 PCG 元素时 提升运行效率。
- 获取执行上下文（Get Execution Context）：已更新为支持 Is Listen Server、Has Authority 和 Is Builder 上下文。
- 提取属性（Extract Attribute）检索给定索引处的特定属性，并将其输出为新的属性集。
- 导出选定属性（Export Selected Attributes）：将数据导出为二进制或 JSON 格式，用于 ML 应用程序等外部用例。
- 属性重映射（Attribute Remap）：现在可由曲线驱动。
- 按位运算符（Bitwise Operators）：添加了左移和右移。

- 布尔运算符（Boolean Operators）：已添加 NAND、NOR、XNOR、IMPLY、NIMPLY 以及从按位运算自动转换为布尔运算。
- 数学运算符：新增 Add Modulo 运算，可用于偏移和使用指数重新排序。
- 网格体边界和网格体点：添加了对骨骼网格体的支持。


**（10）Python 互操作插件**

5.7 版本添加了新的 PCG 插件，提供了一个新的 **Execute Python Script**（执行 Python 脚本）节点，可在执行图形的同时运行 Python 脚本。可以直接将 Python 脚本作为输入，也可以通过磁盘上的文件来运行，还可以通过重载软对象路径的属性或参数来动态设置文件路径。




注意，UE 5.7 的 Execute Python Script 是一项旨在将自动化与程序化生成深度融入工作流的工具。核心是让用户通过 Python 脚本，来控制或自动化虚幻编辑器中的任务，而非用于游戏运行时逻辑。Execute Python Script 的运行方式与使用场景如下：

| 运行方式 | 主要应用场景 |
| --- | --- |
| File 菜单执行 | 用于快捷执行一次性脚本，如在编辑器中对资源进行批量操作或运行测试 |
| 命令行执行 | 常用于外部工具链驱动的自动化流水线，例如在 CI 系统（持续集成）中自动执行任务 |
| 输出日志控制台 | 为开发者提供即时验证与调试的环境，可交互式执行单行 Python 命令或整个脚本 |

| 运行方式 | 主要应用场景 |
| --- | --- |
| PCG 图表节点 | 主要用于程序化内容生成（PCG）流程中，将 Python 脚本作为算法节点直接集成 |
| 电影渲染队列节点 | 用于自动化电影渲染流程，可在渲染前后执行特定的 Python 回调脚本 |

典型场景有：1）**资源管理与流程自动化**：批量修改纹理属性、自动生成 LOD、导入资产等。通过命令行可以构建全自动的资源管理流程。2）**关卡设计**：在 PCG 图表中嵌入 Python 节点，能创建比内置节点更复杂的自定义算法来生成游戏世界，实现关卡设计的自动化与多样化。3）**自动化测试**：结合 PythonAutomationTest 插件，系统会自动发现并执行以 test_*.py 命名的测试脚本，用于保障编辑器工具或游戏逻辑的稳定性。4）**电影与过场动画管线**：通过“电影渲染队列”节点，可以在渲染前自动检查贴图，或在渲染后通知外部系统，实现专业级的镜头序列自动化。

使用 File 菜单执行脚本是最直接的使用方式，主要分两步：1）**启用插件**：需要在 Edit > Plugins 中启用 Python Editor Script Plugin 插件。该插件内嵌了 Python 3.11.8 环境，无需额外安装 Python。通常建议同时启用 Editor Scripting Utilities 插件，封装了许多底层 API，能大幅简化编辑器操作。2）**运行脚本**：插件启用后，在编辑器的 File 菜单中就能找到 Execute Python Script 选项，点击后选择你的.py 文件即可运行。

## 1.1.2. 角色和动画

**（1）运动匹配选择器集成（实验性）**

**运动匹配（Motion Matching）**是虚幻引擎中一套基于查询的动画姿势选择系统，存在于**“姿势搜索”（Pose Search）**插件中。与传统依赖状态机或混合空间的动画系统不同，运动匹配可以在运行时动态地从**动画数据库**中选择最适合当前情境的动画姿势进行播放，无需在动画序列之间预先设置过渡或混合逻辑。

从运行原理来看，运动匹配主要包含三个时间维度的数据匹配：

| 匹配维度 | 含义 | 数据来源 |
| --- | --- | --- |
| 当前姿势 | 角色当下骨骼位置与速度 | 上一帧姿势 |
| 过去运动 | 最近几帧走过的轨迹和姿势 | Pose History 节点 |
| 未来运动 | 玩家输入预测未来要走到的位置 | Trajectory 节点（由移动组件生成） |

在 UE 5.7 中，Epic 带来了运动匹配集成到选择器的首次迭代！这样可以让用户控制哪些资产在运动匹配中是有效的，而在以前的版本中，这只能在数据库层级实现，而不能在单个资产层级做到。

在 UE 5.7 版本中，可以搜索最合适的姿势，还能通过选择器根据游戏逻辑等额外条件过滤每个资产的结果。

选择器中新增的**姿势搜索（Pose Search）**字段可以管理复杂的情况，并且运动匹配的调试功能也已集成到选择器中。

**（2）混合形状和雕刻工具**

在虚幻引擎的传统角色美术工作流中，1）骨骼蒙皮[6]、2）混合形状（Blend Shapes）[7]、3）网格体雕刻[8]通常需要在不同的数字内容创作软件（如 Maya、Blender 或 ZBrush）中分别完成，再导入 UE 进行整合。UE 5.7 版本对骨骼编辑器工具的更新，打破了这一外部依赖，将行业标准级的雕刻工作流的灵活性直接引入了引擎内部。现在，开发者可以在骨骼网格体上即时移动骨骼、绘制权重和雕刻混合形状，从而实现“骨骼驱动 + 形态变形 + 细节雕刻”的一体化工作流——角色网格可以在三种编辑模式之间无缝切换。

V5.7 版本更新了骨骼编辑器工具，现在你可以无缝创建和编辑变形形状、骨骼和蒙皮权重。

- 现在可以在骨骼网格体上即时移动骨骼、绘制权重和雕刻混合形状。
- 新变形目标查看器。
- 形状可适应网格体拓扑的变化。
- 为动画师配备富有表现力的工具，助力打造影视级动画表演。

**（3）控制绑定物理效果的世界碰撞**

控制绑定物理效果的世界碰撞要解决的核心问题是：用物理模拟来增强动画的真实感。例如，如果角色的外套、头发、项链等是一个纯粹的动画序列，它在角色转身或跳跃时的摆动总是固定的。而 Control Rig 物理让你可以在绑定的骨骼链上开启实时物理模拟。当角色靠近墙壁时，外套会撞到墙壁上被压住或弹回，而不是直接穿模。这就是“世界碰撞”在起作用的时刻，它让模拟的物体对周边的物理环境做出反应。这样，角色的动态就更加“活”了，能从根本上提升角色的表现力和沉浸感。

V5.7 版本将关卡碰撞引入控制绑定物理效果模拟，使用现有的世界几何体和物理对象来创建更真实的接触、推动和交互。

- 世界碰撞支持控制绑定物理效果的单向交互。
- 将关卡网格体和物理对象作为碰撞物。
- 在模拟和回放过程中，驱动与道具、地面和环境的接触。

**（4）控制绑定依赖性查看器**

控制绑定依赖性查看器是 Unreal Engine 5.7 新增的一项 Control Rig 调试工具，专门用于

---

[6] **骨骼蒙皮**是一种让网格体跟随骨骼运动而变形的技术。核心思想是：将角色网格体的每个顶点与一根或多根骨骼绑定，并为每根骨骼分配一个蒙皮权重（0~1 之间的数值）。当骨骼移动或旋转时，系统根据权重加权平均计算出顶点的最终位置，从而实现网格体随骨骼驱动的自然变形。蒙皮是角色动画中最基础的技术，适用于四肢、躯干等大范围运动。

[7] **混合形状**（又称**变形目标**）是一种通过顶点位移实现局部精细变形的技术。先定义一个基础网格体，再预先制作若干目标形状（如微笑、皱眉、肌肉鼓起等）。运行时，通过 0 到 1 的权重值将目标形状的顶点偏移量叠加到基础网格体上，多个混合形状可同时混合使用。混合形状擅长表现面部表情、呼吸起伏等无需骨骼参与的细节形变，通常与骨骼蒙皮配合使用。

[8] **网格体雕刻**是一种直接在网格体表面进行交互式顶点编辑的技术，类似于用虚拟笔刷在模型上“捏”出形状。与基于骨骼或预设目标的变形不同，雕刻是手动或程序化修改网格体几何数据的过程，可以增加凹凸细节、重塑轮廓或改变拓扑。

可视化控制、骨骼与模块之间的数据流和依赖关系，让复杂绑定的调试更直观高效。

在 Control Rig 系统中，多个控制点（Control）、骨骼（Bone）和模块（Module）之间存在复杂的驱动关系。例如：反向运动学[9]控制器可能同时影响脚踝、膝盖和脚掌的旋转。当绑定逻辑出错时，手动逐节点排查极为困难。Dependency Viewer 正是为解决这类问题而设计。可以通过专门的依赖性查看器来了解绑定元素的连接情况，并能随时间更新。将控制、骨骼和模块之间的关系可视化，让调试更简单直观。

- 查看父子项和元数据关系，一目了然。
- 调试绑定和模块，更清晰地了解数据流。
- 熟悉的依赖结构可视化。

**（5）FBIK 和重定向性能**

与 IK 相对的是正向运动学（Forward Kinematics，FK）。在 FK 模式下，动画师需要手动设定每一级骨骼的旋转角度，末端位置由父级向子级逐级传递计算得出。FK 适用于绝大多数场景——当角色的脚踩在地面上且身体重心固定时，用 FK 驱动整体动作自然流畅。然而，当角色需要站在高低起伏的地面上、用手触摸不同位置的物体，或者在与环境发生精确交互时，固定不变的关键帧动画就会穿模或者漂移，这时候就需要 IK 介入。

| 维度 | 正向运动学（FK） | 逆向运动学（IK） |
|---|---|---|
| 计算方向 | 父级→子级，由骨骼根部向末端逐级计算 | 子级→父级，由末端位置逆向推导骨骼旋转 |
| 控制方式 | 动画师手动设置每个关节的旋转角度 | 动画师仅指定末端目标位置，系统自动计算关节角度 |
| 适用场景 | 角色整体运动、挥舞武器、普通行走 | 脚部着地适配地形、手部触摸特定物体、与环境交互 |
| 优点 | 可精确控制每个关节的运动轨迹 | 自动适配环境变化，无需手动调整每帧动画 |
| 缺点 | 无法响应环境变化，容易出现穿模 | 计算成本较高，可能出现非预期的关节扭曲 |

IK 的本质是求解一组关节角度，使得骨骼链的末端能够到达指定的目标位置。由于骨骼链中通常包含多个自由度，同一个末端位置往往对应多组不同的关节角度组合，因此需要借助算法来寻找最优解。针对不同的应用场景和性能要求，业界和虚幻引擎提供了多种 **IK 解算算法**：1）Two Bone IK（双骨骼 IK）；2）CCDIK（循环坐标下降 IK）；3）FABRIK（前后延伸反向运动学）；4）Full Body IK（全身 IK）。

Full Body IK（**FBIK**）是一种全局协调的 IK 系统，专门用于同时管理多个肢体链的相互

[9] **反向运动学**（Inverse Kinematics，简称 IK）是一种通过指定末端执行器的位置来计算中间关节旋转角度的运动学方法。在角色动画的语境下，IK 允许动画师指定角色的手脚等末端部位需要到达的位置，系统自动计算并调整手臂、腿部等骨骼链中所有关节的旋转角度，使末端部位准确到达目标点。

影响。与上述针对单条骨骼链的解算器不同，FBIK 允许开发者定义头、双手、双脚等多个 IK 目标点，系统会对角色全身所有受影响的骨骼进行综合求解，确保每个目标同时到达指定位置的同时，保持角色姿态的自然协调。UE 中的 FBIK 建立在基于位置 IK 框架之上，支持逐骨骼设置、首选角度、挤压和拉伸等高级特性，旨在充当控制绑定中的程序化调整工具，例如复杂地形下的脚部对齐或手臂伸展行为。启用 FBIK 需要在编辑器的插件管理器中开启 FullBodyIK 插件，随后在控制绑定图表中创建 Full Body IK 节点并完成执行器和骨骼设置。

**解算器类型对比总结**

| 解算器 | 适用骨骼链长度 | 核心算法 | 优势 | 典型用途 |
|---|---|---|---|---|
| Two Bone IK | 固定 3 节 | 几何直接求解 | 计算开销最小，无迭代 | 脚部接地、手部握持 |
| CCDIK | 任意长度 | 循环坐标下降迭代 | 支持关节角度约束 | 手指按压、颈部转动 |
| FABRIK | 任意长度 | 双向迭代重定位 | 保持链条自然形态，性能均衡 | 触手、尾巴、多节肢体 |
| FBIK | 多链条协同 | 基于位置全局优化 | 多目标协调，全身自然 | 地形适配、全身程序化调整 |

UE5.7 版本对全身 IK 和 IK 重定向器做了性能改进，并添加了控制功能。这为你提供了在运行时执行重定向的方法。

- 利用 FBIK 改进姿势，并且不需要付出高昂的性能代价（速度提高约 20%）。
- 使用 IK 绑定中的伸展肢体解算器，实现高效的运行时重定向。
- 使用 FK 旋转模式的复制本地（Copy Local）可以加快每个条链的旋转传输速度。
- 在重定向堆栈上启用性能分析。
- 为每个重定向运算符添加 LOD 阈值。

**（6）改进的脚部接触和比例重定向**

在 UE 5.7 中，IK 重定向器（IK Retargeter）针对地面接触和大比例角色进行了专项强化，使动画在不同体型（尤其是身高悬殊的角色）之间迁移时更加自然、稳定。这样就能更好地在各种角色之间传输动画，从而大幅减少了美术师手动修正动画的时间。现在可以执行以下所有操作：

- 确定胯部高度，防止骨盆坠地并保持垂直运动。
- 为双脚添加地板约束，以便在靠近地板时保持垂直位置。
- 使用 FK 平移模式朝向和缩放来传输骨骼平移数据。
- 添加拉伸链运算符，匹配或缩放源重定向链。
- 使用伸展肢体解算器来控制肢体的挤压和伸展。
- 将源缩放的枢轴点更改为不同的骨骼，从而进行动作捕捉或配合游戏玩法。
- 添加过滤骨骼运算符，以便在对较大的角色进行重定向时去除高频噪点。

**（7）空间感知重定向（实验性）**

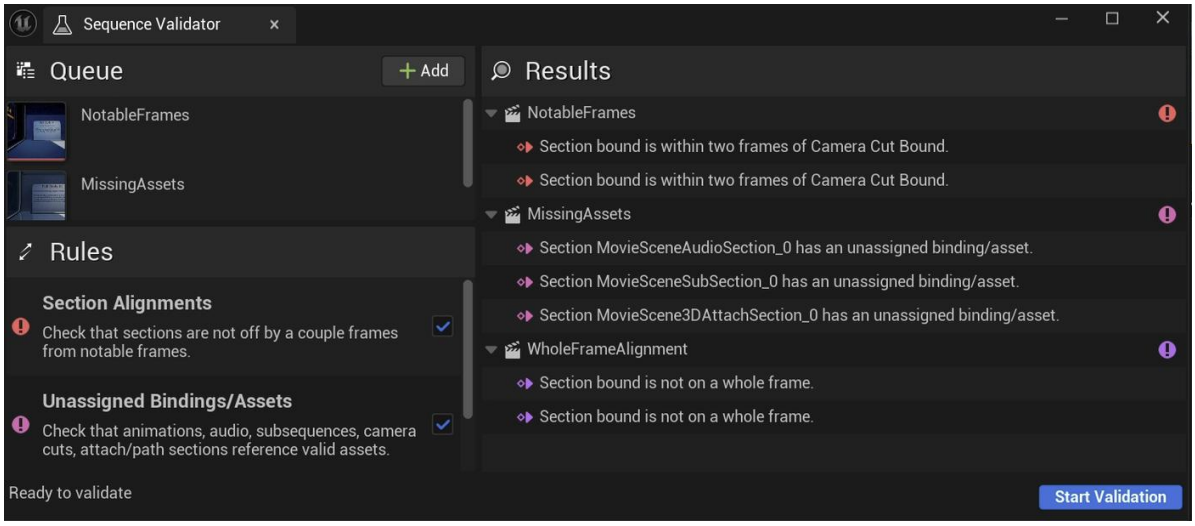
空间感知重定向（Spatially Aware Retargeting）可以看作一个更智能的动画重定向系统。核心作用是解决不同体型角色间，因身体比例差异导致的动画失真和肢体穿插问题。这样也可以减少重定向后清理动画的繁杂工作。

- 添加了形体交互运算符，利用物理原理将 IK 目标推出，以防止自碰撞。
- 添加了相对 IK 运算符，从而根据从信号源到目标的接触长度来移动 IK 目标。

**（8）序列验证器（实验性）**

序列验证器为影片创作者提供了可以在序列中检查是否存在内容错误的方法。这样能够减少在 Sequencer 中工作时可能发生的潜在问题。用它你可以做到：

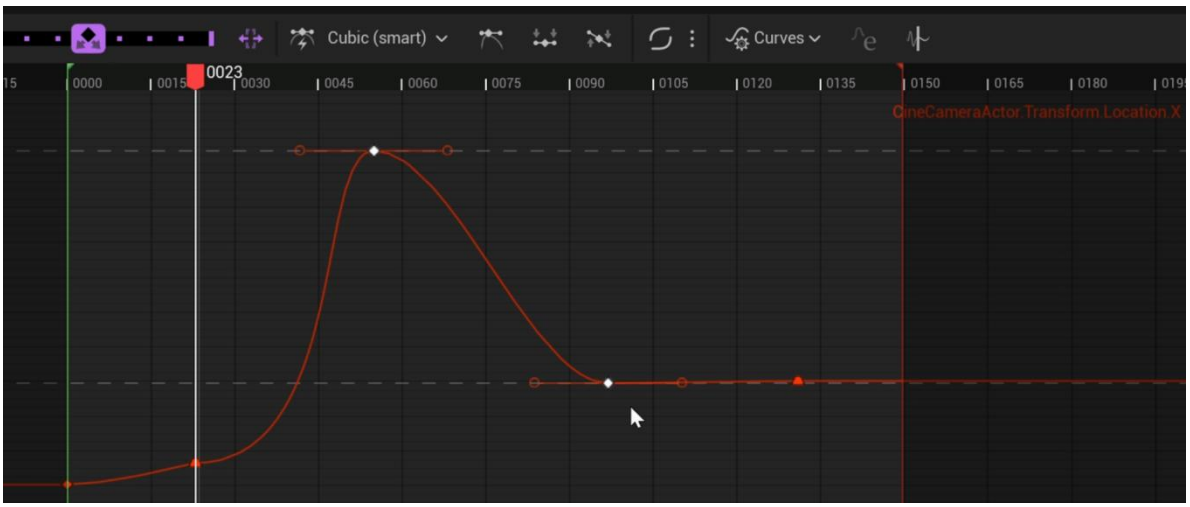
- 对多个序列及其序列层级进行排序，发现潜在问题。
- 检查序列中分段是否偏离了摄像机切换或回放范围。
- 检查各分段是否有未分配的绑定或资产。
- 检查分段的起始和结束是否在完整帧上。
- 查看结果并直接导航到每个序列中的问题所在。




**（8）缓动曲线（测试版）和 UX 改进**

V5.7 版本可以在 Sequencer 中使用动态设计（Motion Design）中的缓动曲线（Ease Curve）工具。可以使用 Ease Curve 插件来启用它。此外，此次更新还带来了更统一的行为，旨在提高可用性，让你可以更快地进行迭代。

- 在 Sequencer、曲线编辑器和 UMG 中使用缓动曲线工具来应用和保存不同的缓动曲线预设。
- 还可以使用缩放锁定功能调整缩放，类似于 Sequencer 序列树中的细节面板。
- 改进了序列层级的时间轴显示，从而同时显示当前序列和父序列的时间。
- 扩展蓝图和 Python 功能，访问轨道状态（单独、静音、停用、锁定和引脚）。
- 改进了变换烘焙，使其包括时间扭曲功能。
- 在导航工具中添加了新的绑定 ID（Binding ID）和序列 ID（Sequence ID）列。




**（9）无数据可变（实验性）**

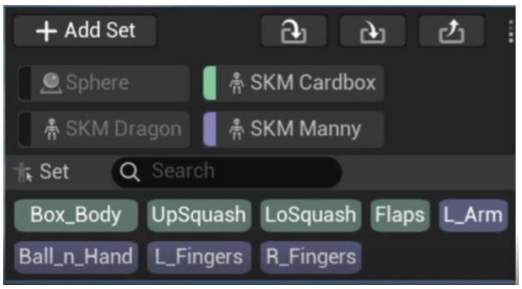
无数据(Dataless)是指输入方式更轻量。传统方式下，所有可能性都会在编译时被固化进资产，导致包体臃肿。而无数据方式只保留操作的指令，不包含任何原始数据，因此资产更小、更灵活。

使用无数据的可自定义 Object，加快迭代速度并简化工作流程。参数输入会在运行时处理而不是编译时，因此编译速度更快，加密效果更好，包体也更小。

- 将可自定义对象转换为无数据对象。
- 通过运行时处理参数，获得更快的编译时间。
- 像使用其它 UE 资产一样使用加密和打包。

**（10）选择集**

动画师需要在整个制作过程中快速访问多个选择。这可能是一个绑定中的多个控制，也可能是不同资产中的多个控制和/或变换。 目前动画师需要依赖 Gui Pickers 等自定义工具来实现这一功能，因此我们希望开发一个更原生的版本，类似于其他工具中好用的插件。




**（11）动画模式 UX 和 QOL 改进**

V5.7 版本对动画模式做出了改进：约束（Constraints）、空间切换（Space Switching）和吸附工具（Snapper）工具现在合并为一个新的约束（Constraint）窗口。同样是关于这个约束，V5.7 版本给创建流程添加了一些新的选项。

- 当前帧：这是以前的标准功能。 它会在当前帧上创建一个 ON 帧，并向前补偿所有帧。
- 从开头（新）（From Start (new)）：获取当前帧的偏移量，并将其作为镜头的第一帧，而不是当前帧。
- 无限（新）（Infinite (new)）：不添加关键帧。 相反，它会将当前帧的偏移量应用于

整个序列。

现在姿势库（Pose Library）、补间工具、约束、选择集（新）和动画图层可根据图标状态打开或关闭相应窗口。 它们在布局中的位置和大小也会被保存。可以将切换这些窗口的操作分配给相应的快捷键。

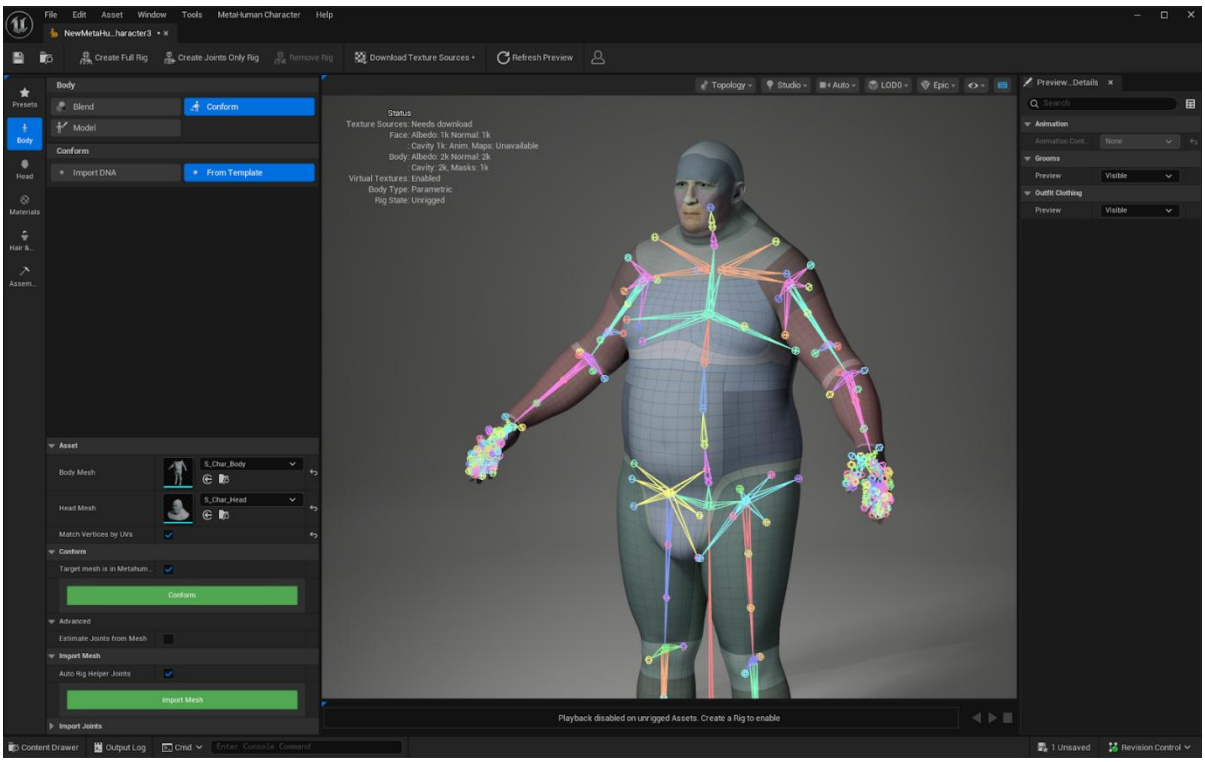
## 1.1.3. MetaHuman

**（1）MetaHuman Creator**

MetaHuman 是虚幻引擎为解决创建超写实数字人类这一复杂课题，而构建的一整套综合性的技术与解决方案体系。整合了创建、驱动与集成数字人所必需的全套工作流。

MetaHuman Creator 则是这套体系中的核心创作工具，用户正是通过 MetaHuman Creator 来直观、高效地完成从零开始创造一个独特数字人类角色的核心任务。MetaHuman Creator 随虚幻引擎 5.6 发布，并支持 Windows 平台，V5.7 版本虚幻引擎中的 MetaHuman Creator 已支持 Linux 和 macOS 平台。

用户可以直接在编辑器中，通过面部雕刻、身体参数调节、发型与服饰搭配等方式，灵活创建细节丰富的数字人类角色。系统会自动生成带有完整骨骼绑定和面部表情的角色资产，方便直接用于动画制作。同时，它也提供了供外部数字内容创作（DCC）软件使用的插件，例如为 Houdini 和 Maya 提供了创建和导入毛发/服装的高级工具集。




V5.7 版本增强了身体网格体的拟合流程，新增了对多种姿势的支持，并引入了拟合模板与模型网格体之间的 UV 空间的顶点对应关系，让你更轻松地获得更精准的结果。参数化身体控制如今也更直观，让你能更轻松地保持人体比例的范围，同时提升精度与控制力。

此版本支持使用 Python 或蓝图对 MetaHuman 角色资产的几乎所有编辑与组装操作进行自动化和批处理。首次通过 API 将 MetaHuman Creator 深度集成至你的管线与创作流程，该 API 支持雕刻、拟合、服装、绑定和纹理功能，简化完整的角色组装流程。可在 MetaHuman Creator 中实时预览编辑效果，在虚幻编辑器中实现交互式运行，或通过计算农场进行离线处理。

**（2）MetaHuman**

MetaHuman 是虚幻引擎（Unreal Engine）推出的高保真数字人类创建系统，允许开发者快速生成、绑定及使用具备影视级真实感的虚拟角色。通过 MetaHuman Creator 从预设库中挑选面部特征、发型、体型，数分钟即可组合出独特角色。自动生成面部绑定（含约 200 个 Blendshape）、身体骨骼及毛发物理系统，支持直接用于动画与表演捕捉。自动生成适合游戏的 LOD 层级，并支持网格体转交到 UE 项目中调整材质与细节。生成的资产可无缝用于游戏、影视虚拟拍摄、VR/AR 等实时交互场景。

虚幻引擎 **Groom 插件**现通过基于关节的工作流程，实现了对发束造型与动画的更精细控制。现在你可以混合模拟与各种美术风格下的运动，充分利用骨骼网格体动画工作流，并通过增强的选择集和绘制工具来获得更大的美术创作自由度。该工具集与最初在 Houdini 中创建的 MetaHuman Groom 完全兼容。

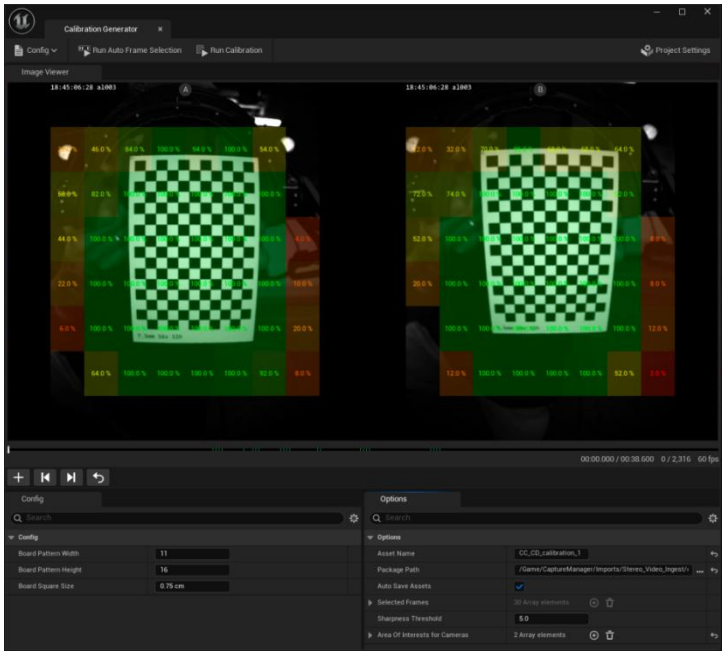
**（3）MetaHuman Animator**

MetaHuman Animator 是 MetaHuman 框架下的一套高保真面部表演捕捉与动画生成工具集。它在 MetaHuman Creator（负责“创造”角色）之后登场，专注于将真实演员的表演高效地转化为角色的生动动画，解决的是“如何让角色活起来”的核心问题。

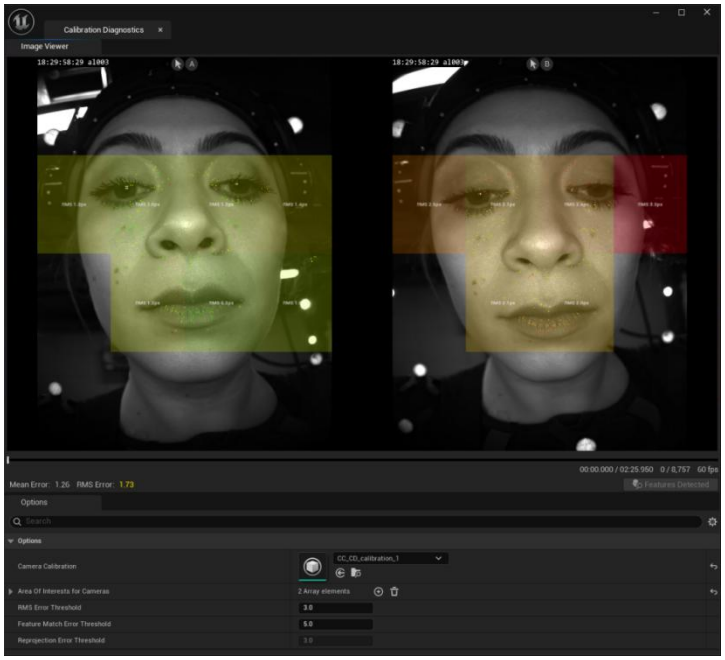
核心功能包括：1）**多元数据解算动画**：支持从三种来源生成面部动画，数据源的多样性大大扩展了其应用场景：2）**实时与离线双模式**：既可在虚幻引擎中实时捕捉并预览动画，也支持对素材进行高质量离线烘焙，达到最高保真度。3）**无缝集成全身动捕**：通过支持时间码同步，可以轻松将面部动画与身体动作捕捉、音频进行对轨，完成完整的角色表演。4）**美术师友好的编辑**：生成的动画数据在语义上是正确的，使用了 MetaHuman 标准的绑定控制点，

并输出平滑的控制曲线，便于美术师在引擎或 Maya 中进行二次调整。

V5.6 版本可通过蓝图接口获取来自 MetaHuman Live Link 源的实时动画。




棋盘格镜头生成的校准数据工作流程（使用立体摄像机设备组捕获）已得到改进，提供更优的自动帧选择功能、辅助手动帧选择的可视化工具，以及跨镜头的可重复配置。




**UE 官网提供图片**

结合全新的 MetaHuman Animator Calibration Diagnostics（实验性），可对照表演素材验证校准效果，更轻松地判断校准方案是否最优。

**（4）MetaHuman 捕获**

**MetaHuman 捕获**是指利用 MetaHuman Animator 工具集，通过 Live Link Face 等入口应用，从各种设备采集真实演员的面部表演数据，并将其转化为 MetaHuman 数字角色动画的完整流程。

V5.7 版本通过扩展支持摄像机的范围，增强了 Live Link Face iOS 与 Android 应用中的实时动画选项。通过 Live Link Face，可以使用外接摄像机在支持的 Android 设备或 iPad 上生成实时面部动画（注：苹果的 iOS 目前不支持外接摄像机）。仅可在 iPad 上通过外接摄像机录制视频。也可在 iOS 和 Android 设备上，通过 Live Link Face 选择设备任意内置摄像机来实时生成动画。仅可在 iOS 上通过内置摄像机录制视频。

**（5）MetaHuman for Houdini**

最新的 MetaHuman for Houdini 更新带来了参数化毛发工具，这种指引驱动型的工作流程可以通过预创作数据来打造发型。该工具集包含毛囊生成器，可处理发际线周围的毛发密度与定位，并根据走向、长度和体积参数生成引导线，同时提供发束插值与高级细节（如簇状感）等造型功能。这也是快速构建并迭代新发型的便捷途径。

此外，为帮助大家学习创建更复杂的发型，Epic 在 Fab 平台发布了新版 Houdini 版 MetaHuman Groom 高级套件作为额外的学习资源。

在 V5.7 版本，MetaHuman for Houdini 已适用 Linux。

**（6）MetaHuman for Maya**

MetaHuman for Maya 的最新更新可在 Linux 上使用，并引入了对 Maya 2026 的支持。

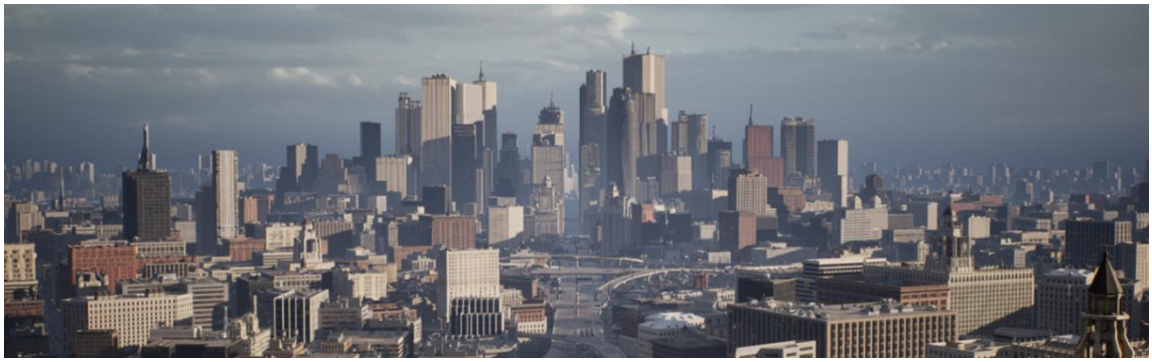
## 1.1.4. 世界构建

**（1）自定义 HLOD**

**LOD**（Level of Detail，细节层次）是一项基础的**实时渲染优化**技术。其原理是为同一模型预先准备多个不同精度的几何体版本，运行时根据模型与摄像机之间的距离或屏幕占比，自动切换显示相应的版本。当物体远离视点时，用低面数、低材质的简化模型替代高精度模型，能够在不明显降低视觉品质的前提下，大幅减少需渲染的三角形数量和绘制调用。在虚幻引擎中，静态网格体的 LOD 可由引擎自动生成，也支持手动定制。开发者通过设置屏幕尺寸等参数来控制各级 LOD 的切换距离。

**HLOD**（Hierarchical Level of Detail，分层细节层次）是虚幻引擎中用于大规模场景优化的重要系统。与针对单个模型的 LOD 不同，HLOD 以场景中的空间单元为对象，将一定范围内的一组静态网格体合并为一个简化的代理模型。在远距离观察时，引擎用代理模型替代原始物体，从而显著降低绘制调用次数与渲染三角面数，有效减轻 CPU 与 GPU 的负担。在虚幻引擎中，HLOD 能够自动生成代理网格，并支持合并材质与纹理烘焙，形成近似于原簇的外观。开发者可在项目设置中启用 HLOD 系统，并通过构建命令为关卡或 World Partition 空间网格生成 HLOD Actor。这些 Actor 会根据视距自动切换，使复杂场景在保持视觉连续性的同时获得流畅的运行性能，特别适用于开放世界与大型场景的开发。

LOD 与 HLOD 的区别在于：LOD 作用于单个模型，而 HLOD 针对的是空间中一组模型的集合。

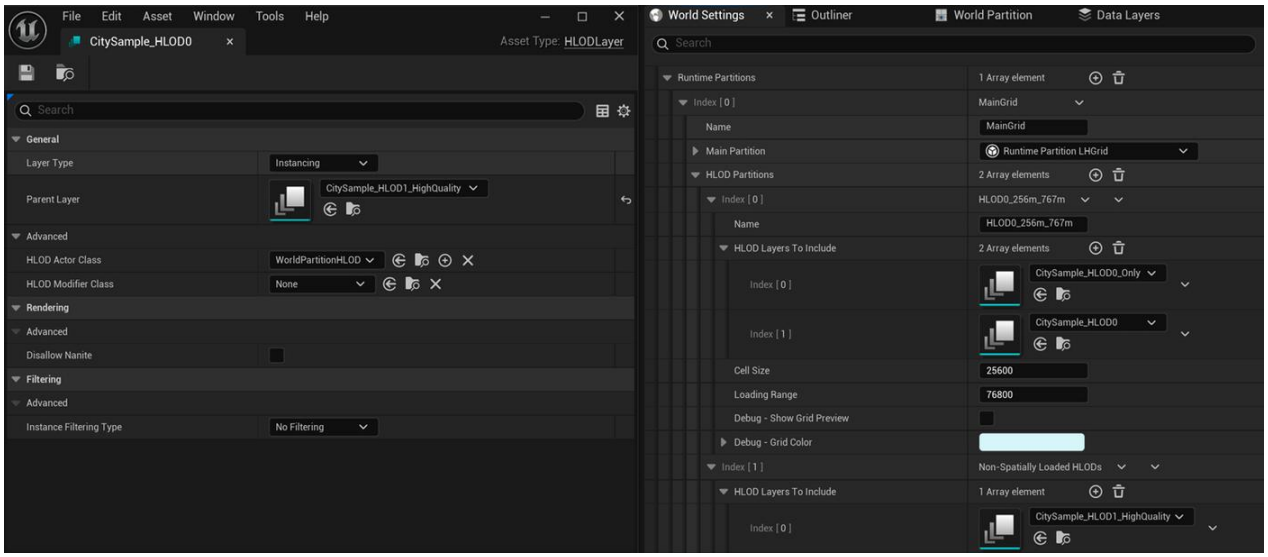


**自定义 HLOD**：在使用 HLOD 生成方法时，输出的结果可能并不总能满足项目的要求，而且会受到网格体组件的约束。自定义 HLOD 支持为单个 Actor 或组提供自定义 HLOD 表示，从而解决了这一限制。可以通过两种方式来使用新的世界分区自定义 HLOD Actor 类：

- 直接注入自定义表示：你可以将自定义表示完整地注入 HLOD 运行时分区，并将其用作父 HLOD 层的输入。
- 仅提供自定义源：你可以将自定义表示作为 HLOD 生成流程的输入，而无需将其添加到世界本身。

**（2）HLOD 配置 UX 改进**

V5.7 版本重新设计了世界场景设置（World Settings） > 世界分区（World Partition）的 HLOD 层配置和 HLOD 层资产，以提高创建或编辑配置时的可用性。




## 1.1.5. 编辑器和 UI 系统

**（1）AI 助手（实验性）**

UE 5.7 引入的 AI 助手（AI Assistant）是一个内置于虚幻编辑器中的智能辅助工具（实验性功能），被直接集成在编辑器内，旨在通过减少上下文切换来提高您的工作效率。可以理解关于引擎的问题，并提供 C++代码片段和分步操作指导。主要目标是将帮助无缝集成到工作流中，并通过 F1 键 快速唤醒。

其核心功能围绕着专用的侧滑面板（Slide-out Panel）展开：

- 知识问答：回答关于 UE 功能、工作流、最佳实践等问题。
- C++代码生成：根据需求生成 C++代码片段，例如查找“如何用 C++将 Actor 移动到指定位置”。
- 分步指导：针对特定任务提供操作步骤，例如“如何为角色设置伤害系统”。
- UI 悬停提示：将鼠标悬停在编辑器 UI 元素上，获取即时说明。
- 资源集成：聚合搜索来自官方文档、社区 Wiki、论坛等渠道的信息

AI 助手由 UEFN 和开发者社区中已有的、专门训练的 AI 来提供支持，可专门用于理解虚幻引擎并帮助你更快地完成任务。

在使用 AI 助手时，建议：1）谨慎验证：鉴于其“实验性”阶段，对于代码或操作建议，务必在理解后进行验证和测试。2）处理 AI 幻觉：如果助手返回与项目实际不符的信息，可以尝试开启新对话或重新表述问题以重置上下文。3）考虑时效性：文档建议助手可能基于 UE 5.7 及相关文档训练，对于更新版本的特性需注意知识时效性。4）关注隐私：Epic 尚未完全公布数据处理与隐私细节，处理敏感项目时请留意官方更新。

**（2）改进蓝图编辑器的复制交互**

**蓝图（Blueprint）**是虚幻引擎中一套基于节点的可视化脚本系统。让开发者在不写传统代码的情况下，通过连接功能节点就能构建游戏逻辑，为程序员、设计师和美术提供了一个协同创作的平台。蓝图的核心是节点（Node）和连线（Wire），在一个图表（Graph）中共同定义了程序的逻辑流和数据流。

- **节点**：每个节点都封装了一个特定功能，比如一个游戏事件（游戏开始、按下按键）、一个动作（播放声音、移动角色）或一次计算。
- **连线**：节点通过连线连接。蓝图中有两种关键连线：
  - 白色执行线：定义了程序的执行顺序，逻辑从左到右流动。
  - 彩色数据线：在不同节点间传递数据（如整数、位置、物体引用等）。

这些图表共同构成了蓝图的“脚本”，驱动 Actor 的行为。一个蓝图可以包含多种类型的图表，以适应不同需求。

在以前的版本中，复制会清除你的选择，导致你要在树状图中手动重新选择项目。现在，在 V5.7 版本蓝图编辑器中复制项目会更流畅，干扰更少。可以在编辑时保持节奏，避免复制元素时的意外点击和返工。

- 可以在蓝图编辑器中复制项目而不会丢失选择。
- 在树状图中工作时，保持激活选择。
- 减少复制多个项目时的重复步骤。

UE 5.7 中更进一步，使蓝图编辑更为高效轻松。

### 1.1.6. 特别鸣谢

UE5.7.4 可能与 Intel NPU[10]冲突，当 UE5.7.4 启动不了时，除了检查驱动程序以外，还要注意运行 UE 的电脑是否有 NPU。特别鸣谢**“UE 大王”王帅博士**提供了 UE5.7.4 启动失败的解决方案：在 Windows11 系统的设备管理器页面，关闭 NPU 即可。


然而，关闭 NPU 之后，腾讯会议的麦克风就不好使了。有卧龙之处，必有凤雏，感谢**“UE 小王子”2024 级刘振元同学**发现这两者的相关性。据初步调查，此现象源于腾讯会议对 NPU 的强依赖架构：其 AI 降噪与非线性回声消除是基于 DNN 模型开发，默认将推理计算移动至 NPU 以换取低时延。因此，强制关闭 NPU 会导致音频预处理链路中断。

综上，适用 UE5.7.4 时关闭 NPU，开腾讯会议前打开 NPU，暂时就这样吧，等 UE5.8 更新。

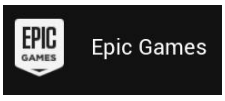
## 1.2. UE5.8 版本说明（节选）

在本书写作过程中，UE5.8.0 发布了。表扬一下 UE5.8.0，彻底解决了 NPU 冲突问题。


还在学习 UE5.8.0 时，Epic 又更新了 UE5.8.1。

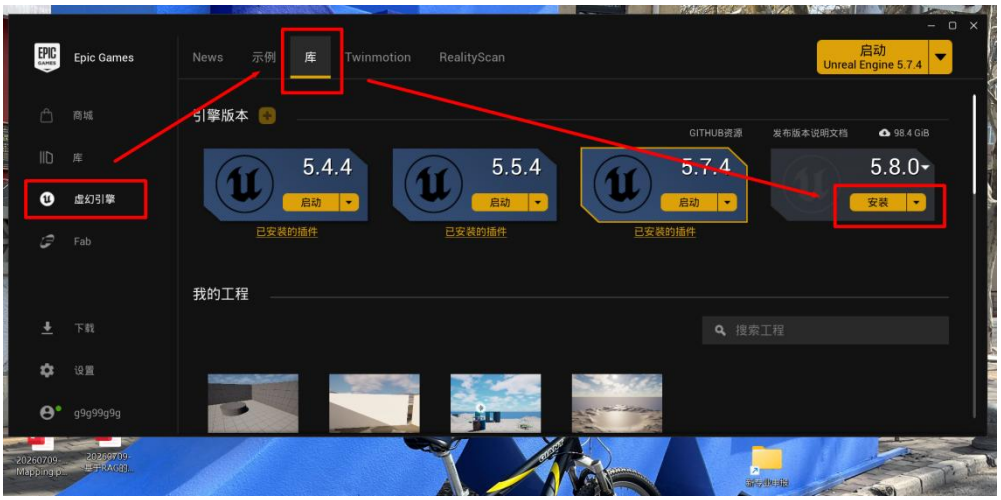
### 1.2.1. 虚幻引擎 5.8 安装

下载 EPIC 平台：

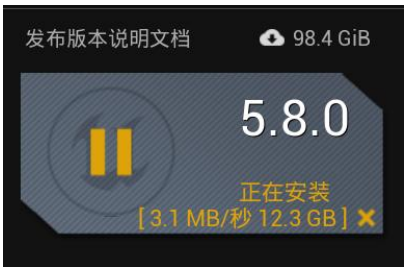



安装 UE5.8：

[10] Intel NPU（神经网络处理单元）是集成在 Intel Core Ultra 等新一代处理器中的低功耗 AI 加速硬件，专为长时间运行的 AI 推理任务优化，如实时视频抠像、超分辨率、人体关键点检测等。结合 OpenVINO™工具套件或 Intel® NPU Acceleration Library，可显著提升模型推理性能并降低功耗。

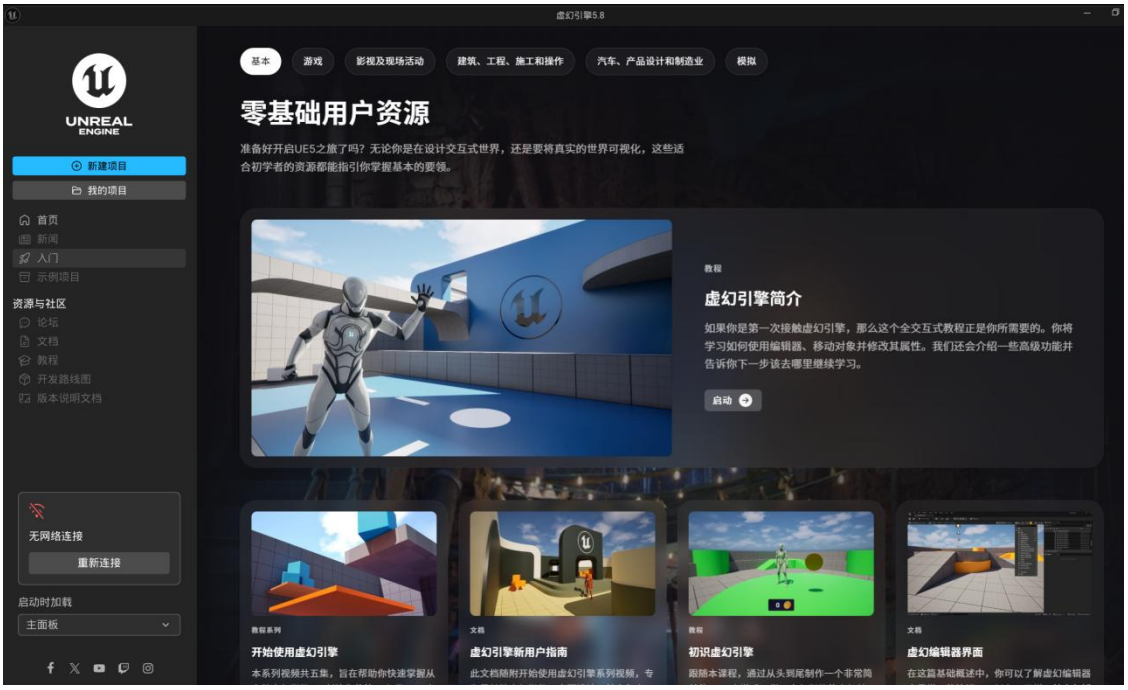



组件选择，默认就行：

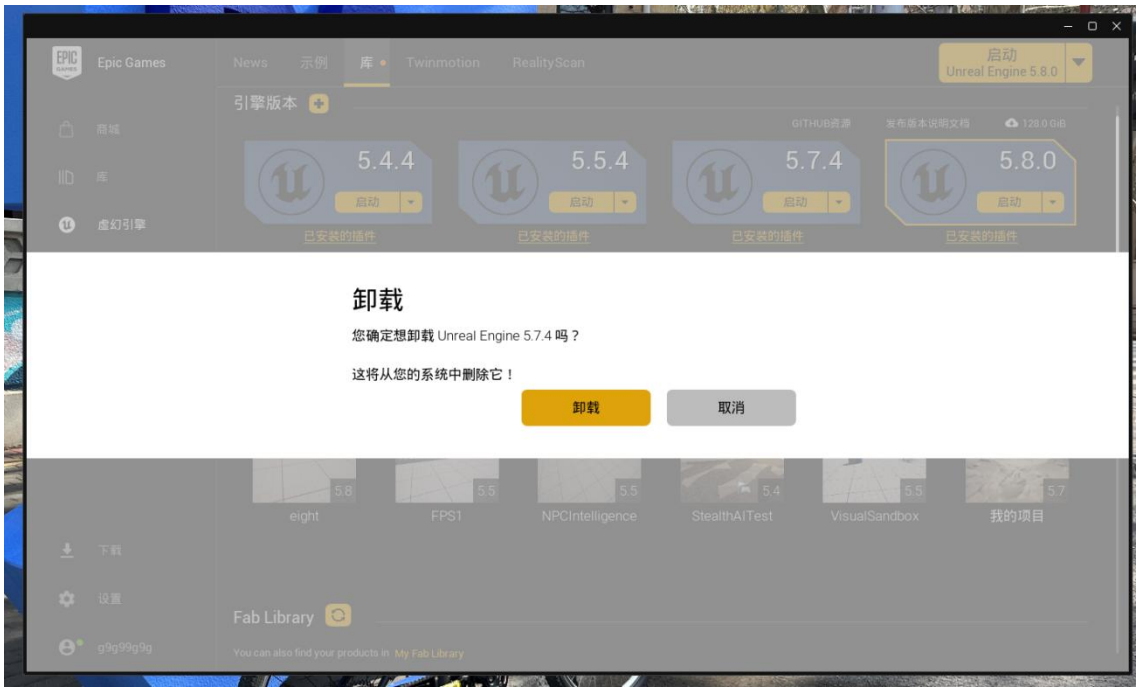




掐指一算，大约需要 8 个小时。

UE5.8 启动页：




UE5.7 可以退休了：




## 1.2.2. 程序化内容生成

### （1）非破坏性手动编辑

在 UE5.8 里，PCG 终于支持**非破坏性手动编辑**了。什么意思呢？就是可以直接上手去改程序化生成出来的东西，选中某些点、排除掉某些点、修改属性，甚至后悔了还能一键恢复。再也不用因为改了一个参数就得从头跑一遍图表了，像在 Word 里用撤销一样贴心。

现在，图表里的每个节点都可以被标记为手动编辑或临时重载。还新增了一个数据重载窗口，把你对当前节点做的所有动手脚都列得一清二楚，方便随时检查和调整。

**（2）复杂属性**

以前 PCG 的元数据属性只能存些简单的数字、向量之类，现在升级了，数组、结构体、集合、映射（也就是字典）统统支持作为元数据值，还能被重载！也就是说，可以在一堆点里存一整个结构体数组，然后提取、修改、遍历它们，就像写程序一样灵活。

为此，新增了两个实用节点：创建复杂常量和添加复杂常量，让你直接在图表里造出这些复杂数据，不用绕道去 C++或蓝图。

**（3）嵌入式子图表**

现在你可以像写蓝图函数一样，在 PCG 图表内部创建子图表。把一段常用的生成逻辑打包成一个子图表，存起来，随时复用，主图表变得清爽又整洁。这才是程序员的“模块化”。

**（4）图表参数编辑器**

PCG 的参数管理终于有了专属的指挥中心，全新的参数层级编辑器。可以在这里给参数分组、排序、加描述、设层级，就像整理文件夹一样直观。再也不用在一大堆参数里瞎找了。

**（5）PCG 性能优化**

运行时：PCG 的 GPU 运行时散射，就是在地面上撒石头、草那种操作，现在速度和地形自带的 GPU 草地系统一样快，而且已经在所有主流平台上被蹂躏测试过了，稳得很。

无 Actor/无组件运行时生成：现在你可以在不创建任何 Actor 或组件的情况下，让 PCG 在运行时默默生成数据，更适合后台计算。

编辑器内优化：运行图表时的 CRC 和缓存流程被优化了，缓存失效的概率降低，效率蹭蹭涨。生成大量静态网格体时，ISM（实例化静态网格体）的复用逻辑更强了，不会重复造轮子。图表执行和调度也做了改进，减少了资源争抢和瓶颈，跑起来更顺畅。

**（6）PCG 用户体验**

UE5.8 继续宠粉，又给 PCG 的工作流加了不少人性化小细节：

- 上下文筛选：当你从任意类型的引脚拖出一根线想添加新节点时，弹出的节点面板会自动只显示能接得上的节点，省得你在一堆不相关的选项里翻找。当然，如果你非要看全部，也可以一键关闭筛选。
- HLSL 内核：现在你可以直接从 HLSL 内核（自定义着色器代码）里创建属性，不用再跑到节点设置里手动添加了，少一步就是快一步。
- HLSL 编辑器：代码编辑器的字体大小终于可以调了，近视眼福音。
- 数据视口：在 PCG 编辑器的数据视口里，你可以直接编辑手动重载，所见即所得，改完立刻看效果。

**（7）PCG 节点和运算符**

UE5.8 又塞进了一批新节点，让生成逻辑更强大：

- 元数据数组操作：最常用的数组操作都安排上了，创建、添加、添加唯一（不重复添加）、包含判断等等，像在玩 Python 列表。
- 对齐点：可以把一个点的位置和旋转，对齐到另一个点的边界，在任意轴和参考系下

都行。比如让一棵树严丝合缝地贴在一块石头的侧面上。

- 场景捕获：场景捕获节点现在支持输入变换，你可以从任何方向、任何位置“拍一张照”来采样场景颜色或深度，GPU 采样更灵活了。
- 创建/添加复杂常量：这两个节点让你在图表里直接创建或添加任意复杂属性类型，配合前面说的复杂属性支持，如虎添翼。
- 将样条线应用于组件：可以把 PCG 里的样条线数据写回到场景中现有的样条线组件上。比如在编辑器里用 PCG 算出一条新的路径，然后直接覆盖掉原来手动画的样条线。
- 传送 Actor 和组件：你可以对场景里任何 Actor 或组件引用施加变换（移动、旋转、缩放）。这特别适合基于关卡的 PCG 工作流，在编辑器里就能对已有物体“瞬移”，不用等运行时。
- 从属性获取类：从一个软 Object 引用里提取出它的类，并以属性形式返回，方便后续逻辑判断。
- 获取编辑器摄像机：把当前编辑器视口里你正看着的摄像机位置、朝向等信息，作为一个“点”返回。这样你可以在编辑器里做“以摄像机为中心”的生成操作。
- 在动态网格体上设置材质：直接把材质应用到动态网格体数据上，不用再转成静态网格体再贴材质了。

**（8）程序化植被编辑器（实验性）**

这个之前 5.7 就有的实验性插件，在 5.8 里迎来了大升级，能力暴涨：

- 生长节点：现在你可以在编辑器里直接“种树”，不用从外部导入配方，像玩模拟经营游戏一样看着植物长出来。
- 种子生成：内置种子生成逻辑，随机性更可控。
- 从 3D 网格体或 2D 图像/图集中提取骨骼：你可以导入一棵树的模型或一张叶子图集，让系统自动提取出“骨骼”结构，用来驱动程序化生长。
- 植被嫁接：支持把不同植物的分支嫁接到一起，创造杂交新品种。
- Object 规避和雕刻：树枝会自动避开周围物体，还会根据环境进行雕刻变形，更自然。
- 修剪 - 分支循环、位移曲面细分：可以给分支做循环修剪，还能用位移曲面细分增加细节。
- 网格化器改进：生成最终网格体的算法更聪明了，结果更干净。
- 可视化模式：新增多种可视化显示模式，方便你调试生长过程。
- 主干材质合成器：可以自动给主干生成材质，省得你手动调。

以上，这就是 UE5.8 在 PCG 方面给咱们带来的新玩意儿。总结一句话：PCG 现在不仅能自动生成，还能手动改，能处理复杂数据，能内嵌子图表，跑得更快，操作更爽，种树也更像真种树了。

## 1.2.3. 框架

框架系统大阅兵：StateTree、Mass、导航、Iris、Mover 和统一输入齐上阵

**（1）StateTree：起始状态和编译器扩展**

UE5.8 给状态树（State Tree）松了绑，现在可以明确指定树的起始状态，不用非得从默认根节点开始跑了。编译器管理也升级了，编辑器会像个监工一样盯着你的树，一旦过时了，比如依赖的资产变了，就醒目地提醒你该重新编译，再也不用自己瞎猜这树到底是不是最新的。

属性绑定更是支持反向操作，能直接把已完成任务里的数据，一键转移到另一个任务里，省去中间商赚差价，数据流转前所未有的丝滑。

**（2）Mass 框架：更快、更模块化，专治各种人多势众**

**Mass**，这个专为海量实体（成千上万个单位）打造的数据驱动系统，在 5.8 里彻底健身了一番。**Mass Signals**（信号）现在成了引擎核心成员，不再是个编外插件。而且得益于基于原型的无锁调度，实体创建可以在游戏线程之外偷偷进行，完全不卡主线程，效率飙升。

彻底重排了 Mass 处理器的执行和依赖解析逻辑，目的只有一个，拼命压榨多核 CPU，让现代硬件跑得更快更安全。更细粒度的调度加上线程安全的观察者通知，为这一波提速保驾护航。

此外，全新的 **MassCore 模块**将核心实体系统从 MassGameplay 大礼包中剥离了出来，让其他开发者想接入时不用买棺还珠，轻量化引入即可。

**（3）导航：按网格体级别控制可行走表面**

AI 走路终于可以挑路了！静态网格体现在能在导航网格生成时，自己定义哪些区域能走、哪些是雷区。设计师可以通过每网格的区域成本（Area Cost）来微调，比如给沙发贴上高成本标签，AI 就会乖乖绕道走，而不会傻乎乎地试图从上面踩过去，穿模既视感瞬间消失。

这项功能通过 NavCollision 的标志位和区域类别设置即可开启，贯穿整个 Recast 导航生成管线，设置起来非常顺手。

**（4）Iris：复制系统正式毕业，进入生产就绪**

好消息！Iris（虚幻的新一代网络复制系统）在 UE5.8 中已经达到了可供授权方使用的生产就绪状态，不再是实验品了。开发团队重点打磨了稳定性和日常易用性：1）协议不匹配时，客户端会收到更清晰的报错信息，不用再对着日志发懵。2）新增的生命周期警告能帮你提前发现对象管理方面的隐患（比如对象销毁时机不对）。3）序列化、剔除距离处理和诊断工具也全面升级，RPC DoS 检测功能也重新上岗，防止恶意刷包攻击。4）更重要的是，Iris 如今更贴合引擎整体发展方向，增加了对场景图集成和 RemoteObjects RPC 路由的支持，接入更顺滑。

**（5）Mover：物理、动画和网络全面大补**

Mover 系统（角色移动的新框架）迎来了范围极广的更新，涵盖核心移动、网络、动画和 ChaosMover 四大块。

- 核心移动：修复了一大波 Bug，新增了对 Detour Crowd 的 NavWalking 支持；站在移动平台（比如摇晃的船）上的附着移动更稳了，不会一卡一滑；根运动和动画蒙太奇处理也更丝滑。
- 分层移动：现在可以提前安排调度，同步效果更佳，同时回滚和预测支持也大幅扩展。
- 网络功能：新增了 Iris 复制的可选支持，并引入了自适应时间膨胀，减少了客户端预测的延迟开销。
- ChaosMover：增加了用于动作匹配工作流的轨迹预测功能，更深入地集成了 Chaos 可视调试器，对模拟驱动的蒙太奇、分层移动、Gameplay 标签和即时移动效果的支持也更广泛。

总的来说，这些改动标志着 Mover 正在逐步脱离实验状态，向稳定架构大步迈进——移动模式、预测、重模拟、动画驱动工作流，都越来越“皮实耐造”。

**（6）统一输入系统：通用 UI 和增强输入合体**

UE5.8 终于把输入工作流给捋顺了，将增强输入和通用输入/UI 合二为一。这意味着处理跨平台输入（键盘、鼠标、手柄、触屏）时，不再需要维护两套重复的数据资产，调试过程也变得更直观。

本次更新还带来了：

- 更可靠的事件处理
- 更好的控件绑定行为
- 扩展的虚拟按键支持
- 各种触发边缘情况的修复

此外，新工具，比如增强输入 UI 的输入调试器和更强的可扩展性，让你排查输入问题时更加得心应手，再也不用靠猜了。

总的来说，UE5.8 在框架层面下了狠功夫：StateTree 更灵活，Mass 更快更省内存，导航更细粒度，Iris 终于可上生产，Mover 稳步成熟，输入系统大一统。每一个都是硬货，够各位同学玩一阵子了。

# 1.3. 虚幻引擎设计哲学

## 1.3.1. The Unreal Way

虚幻引擎官方文档中有一份经典文献，被称作 The Unreal Way（UNREAL 方法），也有人称之为 UNREAL 之禅。这份文档的思想只有一句话：“**虚幻引擎的设计目标是让一个游戏项目能够尽可能快、尽可能容易地启动并运行起来。**”

为了达到这个目标，Epic 鼓励开发者按照 Unreal 方式来做事。这句话听起来有点玄，但背后有一个非常务实的理由，虚幻引擎是一个已经非常成熟的系统。在绝大多数情况下，某个功能之所以要按某种特定方式来实现，都有其实际原因。Epic 提供的解决方案，在虚幻引擎的上下文中，通常就是最好、最实用的那一个。

一个孤立设计的系统，可能比虚幻引擎更先进、更容易使用或更高效。但当你试图让它和代码库的其他部分协同工作，并以可接受的帧率运行时，可能会引发无数令人头疼的问题。因此，官方鼓励开发者尽量使用引擎提供的现成解决方案，最大限度地减少对引擎的修改。这背后是一种务实的哲学，**与其另起炉灶，不如站在巨人的肩膀上**。

### 1.3.2. 隔离与继承

那么，当引擎的默认行为确实无法满足项目需求时，该怎么办？The Unreal Way 给出的答案非常明确，**隔离你的游戏项目，无论是内容还是代码**。具体做法是：在你自己的模块中，尽可能地继承核心组件的子类，并复制示例项目中的功能实现。这样做的好处是双重的：1）**不影响核心引擎**，所有修改都局限在自己的项目模块中，不会污染引擎本身的代码。2）**方便版本合并**，当 Epic 发布新版本的引擎时，可以轻松地将更新合并进来，而不需要和自己的修改打架。

这背后的设计哲学是，引擎是平台，项目是内容。平台应该保持稳定和可升级，内容则可以在平台之上自由生长。正如 The Unreal Way 文档所提醒的，需要权衡“合并更新代码所需的时间和精力”与“大规模子系统替换将带来的理论上的改进”。除非有充分的理由，否则不要轻易修改引擎核心。

### 1.3.3. 模块化

如果说 The Unreal Way 是引擎设计的指导思想，那么模块化（Modularity）就是实现这一思想的技术手段。在虚幻引擎中，模块（Module）是软件架构的基本构建块。模块将特定的编辑器工具、运行时功能、库或其他功能封装在独立的代码单元中。这种设计带来了几个关键优势：1）**可维护性**：每个模块各司其职，代码边界清晰，修改一个模块不会牵一发而动全身。2）**可复用性**：一个写好的模块可以在不同的项目中被复用，就像搭乐高积木一样。3）**可插拔性**：插件（Plugin）本质上就是带资产和描述的模块集合。需要就启用，不需要就禁用，非常灵活。

引擎本身由一组模块构成，而每个游戏项目也是由一个或多个游戏模块构成的。模块化的设计哲学贯穿始终，从引擎底层到游戏逻辑，无一例外。

### 1.3.4. 数据驱动

另一个贯穿虚幻引擎设计始终的重要理念是数据驱动（Data-Driven）。所谓数据驱动，核心思想是把游戏中的各种数值、配置、参数从代码中抽离出来，放到外部数据文件中，如数据表、数据资产。

为什么这样做？因为：1）**迭代更快**：策划调整数值只需要修改 Excel 表格并重新导入，不需要打开蓝图、修改节点、重新编译。2）**职责分离**：程序员负责构建系统，策划/设计师

负责填充数据。各司其职，互不干扰。3）**按需加载**：数据驱动让你能够在需要时才加载所需的数据，优化内存和性能。

从数据表（DataTable）、曲线表（CurveTable）到数据资产（Data Asset），再到更高级的数据注册表（Data Registry），虚幻引擎为数据驱动提供了完整的工具链。这背后是一种**配置优于编码**的设计哲学，能用数据配的，就别写死在代码里。

## 1.3.5. 一切皆对象

虚幻引擎的整个游戏对象模型，是一个经典的单一继承树（Single Inheritance Hierarchy）案例。几乎所有的类都直接或间接继承自一个共同的基类 UObject。这种设计的优点是显而易见的：1）**统一的生命周期管理**：UObject 提供了垃圾回收机制，开发者不需要手动管理内存。2）**统一的反射系统**：UObject 支持属性（UPROPERTY）和函数（UFUNCTION）的反射，使得蓝图可以访问 C++的变量和函数，也使得编辑器可以自动生成 UI 控件。3）**统一的序列化**：UObject 支持将对象状态保存到磁盘（保存关卡、保存资产）和从磁盘加载。

Actor 继承自 UObject，是所有可以放入关卡的对象的基类。Pawn 继承自 Actor，是所有可被控制的实体的基类。Character 继承自 Pawn，是专门为人形角色设计的子类。这种清晰的类继承体系，让开发者能够从抽象到具体，一步步构建出复杂的游戏逻辑。这种设计哲学的核心理念是通过一个强大而统一的基类体系，为所有上层功能提供坚实的地基。

## 1.3.6. C++与蓝图的平衡

虚幻引擎的另一个独特设计哲学，是 C++与蓝图的深度整合与明确分工。这不是二选一，而是各取所长、协同工作。

C++负责造轮子：实现核心算法、复杂逻辑、高性能计算，以及构建供蓝图调用的功能库。C++代码是引擎的“骨架”，坚固、高效、稳定。而蓝图负责搭积木：在 C++构建的基础上，快速迭代游戏逻辑、设计原型、处理事件响应。蓝图是引擎的“血肉”，灵活、直观、易上手。

这种分工的设计哲学基于一个朴素的认知：程序员的时间是宝贵的，CPU 的时间是廉价的。把程序员的精力从繁琐的数值调整和逻辑拼接中解放出来，投入到真正需要深度思考和优化的核心系统上。而设计师和美术师则可以通过蓝图，亲手实现自己的创意，不需要等待程序员排期。

Epic 官方甚至建议，在项目初期，先用蓝图快速搭建原型验证玩法，等玩法稳定后，再将性能瓶颈部分用 C++重写。这是一种先快后优的务实哲学。

## 1.3.7. 工具为人服务

最后，虚幻引擎的设计哲学中还有一个贯穿始终的原则：**开发者效率优先**。无论是在 Slate

UI 框架的设计中，还是在 Niagara 特效系统的设计中，都能看到这一原则的影子。

Slate 的设计目标就是在可能的情况下提高开发者的效率。程序员的時間很宝贵，CPU 很快且便宜，所以宁愿让 CPU 多干点活，也要让程序员少操点心。

Niagara 设计目标之一是不碍用户的事（gets out of the way of the user）。把完全的控制权交给美术师，让工具适应人，而不是让人适应工具。

PCG、世界分区、数据驱动：所有这些系统背后都有一个共同的指向，减少重复劳动，缩短迭代周期，让创作者把精力集中在创意本身。

## 1.4. UE6 猜想

如果说虚幻引擎 5 的设计哲学是**用划时代的图形技术重新定义世界构建方式**，那么虚幻引擎 6 的哲学则是一次深刻的转向，**从如何做出更精美的画面转向如何更聪明地创造、发行和运营游戏**。

UE6 不再仅仅是图形技术的迭代，而是 Epic Games 试图打破游戏孤岛、统一创作者生态、并从根本上改变游戏开发范式的一次雄心勃勃的尝试。它的设计哲学可以归结为三个核心支柱：统一的创作生态、革命性的编程模型，以及 AI 驱动的开发流程。

**（1）UEFN 与 UE5 大一统：重塑创作生态**

UE6 最根本的设计哲学是融合。将两条原本平行的开发线——服务于 AAA 级游戏开发的虚幻引擎 5 和面向创作者社区的**虚幻编辑器（UEFN）**合并为一个统一的产品。

**一次构建，随处部署**是 UE6 大一统哲学的最终愿景。其目标是让开发者能够使用同一套工具，一次性开发出游戏或体验，并将其无缝部署到传统的 PC、主机、移动平台，以及庞大生态系统，甚至是开发者自己的实时服务生态中。

**打破游戏孤岛**：UE6 试图建立一个互联的元宇宙生态，让玩家和内容可以在不同的游戏世界之间自由穿梭。这不仅仅是技术上的统一，更是商业模式上的变革，Epic 将其在《堡垒之夜》中验证成功的 UGC 用户生成内容工具链和生态系统，作为核心能力开放给全行业。

**（2）Verse 与场景图：语言与架构的革命**

为了实现大规模、持久化的在线世界，UE6 在底层架构上进行了重构，有全新的编程语言 Verse 和全新的 Gameplay 框架场景图（Scene Graph）。

Verse 为大规模协作而生的新一代语言，Verse 被定位为 Epic 未来编程模型的基础，是一种专为驱动大规模、持久化游戏世界而设计的编程语言。Verse 从函数式、逻辑式和命令式语言中汲取灵感，旨在解决现代游戏开发中的复杂性和可扩展性问题。

UE6 的一个重大变革是计划逐步淘汰蓝图（Blueprint）系统，转而采用**基于 Verse 的模块化场景图**。虽然 UE6 早期版本会保留蓝图以支持平滑迁移，但 Verse 代表了未来的方向。

**场景图（Scene Graph）**是 Verse 的舞台，是一个在 Verse 基础上从零构建的现代化高级 Gameplay 框架。为开发者提供了基础，以便轻松创建游戏，并在不同游戏之间共享可互操作的组件。

颠覆性的网络模型是 Verse 最具革命性的特性。开发者可以像编写单机游戏代码一样编

写网络逻辑，而 Verse 运行时会在后台自动将工作负载透明地分发到多个服务器上执行。

Verse 中的所有函数都作为原子事务运行，出错时可以回滚和重新模拟，极大地简化了复杂网络环境的开发难度。

极简的存档系统，基于事务和分布式语义，Verse 运行时可以自动同步和保存程序的任何全局状态，开发者无需再手动配置复杂的数据库。

**（3）深度集成的智能助手**

UE6 将 AI 从辅助工具提升为核心开发流程的一部分。

MCP（模型上下文协议）集成，UE6 计划通过 MCP 协议深度集成 Claude、Gemini、Codex 等主流大语言模型。

覆盖全流程的 AI 辅助，AI 的应用场景被明确规划在关卡搭建、角色绑定、粒子系统配置、动画蒙皮、代码分析、自动化测试等耗时且重复的劳动中。

目标是解放创造力，其终极目标是大幅减少内容创作中的繁琐工作，将开发者的时间和精力解放出来，投入到更具创造性的探索和迭代中。

**（4）开放与互操作**

UE6 致力于建立一个更加开放的游戏资产和经济体系。

拥抱开放标准：UE6 计划优先采用 glTF、USD 等开放标准作为内容交换的基础。

可移植的内容与经济：终极目标是实现内容、代码乃至经济系统在不同游戏和生态系统之间的互通。一个典型的例子是，未来玩家在《堡垒之夜》中购买的皮肤，可能成为可在其他 UE6 游戏中使用的跨游戏道具。

总而言之，UE6 的设计哲学不再局限于提供更逼真的画面，而是致力于构建一个更智能、更开放、更统一的开发与运营平台。试图回答一个比如何渲染更根本的问题：在未来的游戏产业中，我们应该如何更高效地协作、创造和连接玩家？

# 2. 欢迎来到虚幻引擎

https://dev.epicgames.com/documentation/unreal-engine/understanding-the-basics-of-unreal-engine

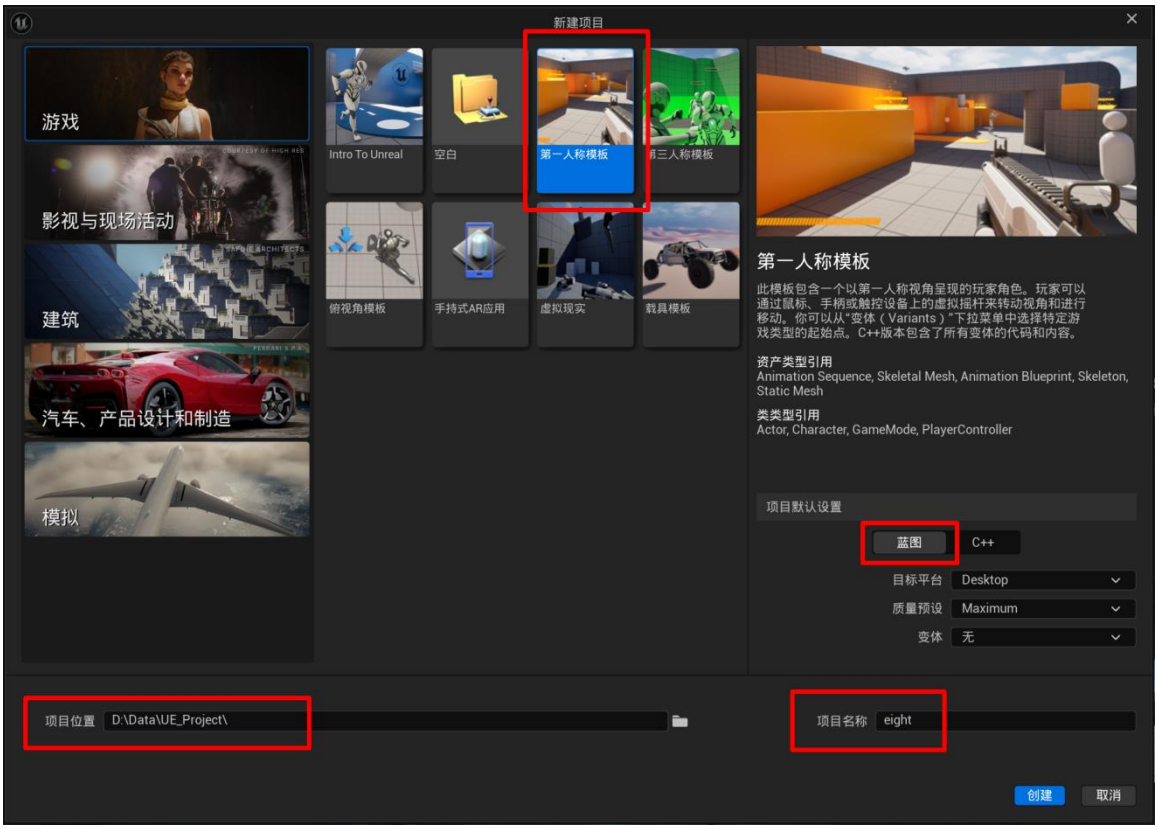
## 2.1. 虚幻引擎基础

### 2.1.1. 虚幻引擎术语

**（1）项目 Project**

虚幻引擎 5 项目（Unreal Engine 5 Project）中包含游戏的所有内容。项目中包含的大量文件夹都在磁盘上，例如 Blueprints 和 Materials。可以按照自己的意愿命名文件夹并将其整理到项目中。虚幻编辑器（Unreal Editor）中的内容浏览器（Content Browser）面板显示与磁盘上的 Project 文件夹相同的目录结构。每个项目都有与其关联的.uproject 文件。.uproject 文件是创建、打开或保存项目的方法。可以创建任意数量的不同项目，然后并行处理它们。

启动 UE5.8 引擎，新建项目，项目名就叫 eight 吧：




**（2）蓝图 Blueprint**

蓝图视觉效果脚本（Blueprint Visual Scripting）系统是完善的 gameplay 脚本系统，在虚幻编辑器中使用基于节点的界面来创建 gameplay 元素。就像许多常用的脚本编写语言，可以用于在引擎中定义以 object 为导向（OO）的类或 object。在使用虚幻引擎时，经常会发现使用蓝图定义的 object 被统称为蓝图。

中译中，可以把蓝图想象成一种不用写代码的**可视化编程工具**。在虚幻编辑器里，只需要像搭积木一样，把一个个代表功能的节点拖拽并连线，就能让游戏里的角色动起来、触发机关或播放音效。这种画流程图一样的操作，就是我们说的蓝图脚本。而且，用这种方式最终做出来的整个东西，比如一把会自动开火的枪、一个能打开的门，在虚幻里也常常直接被叫做蓝图。

**（3）类 Class**

类（Class）定义虚幻引擎中特定 Actor 或 Object 的行为和属性。类是分层的，意味着类从其父类（即派生出类的类，或子类的来源）中继承信息并将该信息传递给其子项。可以在 C++代码或蓝图中创建类。

类就像是一张**设计图纸**，规定了一类东西的共同特征和能做的事。比如敌人这张图纸，会定义敌人有多少血量、移动速度多快，以及可以攻击、巡逻这些行为。

图纸之间还有**遗传**关系：可以先画一张动物的基础图纸，再画一张狗的图纸，让它自动继承动物的所有特征，然后再添加汪汪叫这类属于自己的新能力。在虚幻中，可以用蓝图或者 C++代码来画出这样的图纸，也就是创建类。

**（4）对象 Object**

Object 是虚幻引擎中最基本的类，换而言之，它们就像建造系统的砖块，包含资产的大量基本功能。虚幻引擎中的几乎所有功能都继承自 object（或使用其中的部分功能）。在 C++中，UObject 是所有 object 的基类，可以实施多种功能，例如垃圾回收、用于将变量提供给虚幻编辑器的元数据（UProperty）支持以及用于加载和保存的序列化。

对象是虚幻世界中最基础的**积木块**，几乎所有游戏内容都建立在它之上。提供了很底层却又很通用的能力：比如让你能在编辑器里直接看到和调整数值，能自动把没用的东西清理掉节省内存，还能把当前的游戏状态保存下来、下次再加载回来。当你根据设计图（类）在游戏里真正“摆放”出一个具体的东西时，那个东西就是对象。打个比方，你拿同一张“宝箱”图纸，在场景里放了 10 个宝箱，这 10 个宝箱每一个都是独立的对象，它们可以拥有各自不同的状态，比如有没有被打开过。

**（5）Actor**

Actor 是可以放到关卡中的任何 object，例如摄像机、静态网格体或玩家出生点位置。Actor 支持 3D 变换，例如转换、旋转和缩放。可以通过 gameplay 代码（C++或蓝图）创建（生成）或销毁 Actor。在 C++中，AActor 是所有 Actor 的基类。

Actor 就是你能放进游戏关卡里的任何东西。摄像机、一棵树、一盏灯、一个玩家出生的标记点等全都是 Actor。可以随意移动、旋转、缩放它们。而且，它们可以由蓝图或 C++随时生成或销毁。打个比方：Actor 就像舞台上的道具、灯光、演员，只要是能摆在场景里的，都属于这个范畴。

**（6）类型转换 Casting**

类型转换（Casting）是一种动作，将会提取特定类的 Actor 并尝试将其作为其他类进行处理。是一种尝试把某个 Actor 当成特定种类来用的动作。

类型转换可能成功，也可能失败，如果成功了，你就能使用那个特定种类才有的功能，

即可以在你类型转换到的 Actor 上访问特定于类的功能。例如，如果你要制作一款游戏，在其中具有能够以不同方式影响玩家角色的多种体积类型。其中一个体积是火焰（Fire），可以随着时间降低玩家血量。当角色与关卡中的任何体积重叠时，就可以将该体积类型转换（Cast）到火焰（Fire）上，以尝试访问其损害玩家血量功能。

- 如果类型转换成功，即如果玩家站在火中，则玩家的血量将开始下降。
- 如果类型转换失败，即如果玩家站在任何其他类型的体积中——其血量将不受影响。

类型转换不同于简单地检查 Actor 是否是给定的类，将返回一个二选一的答案（是或否），但不允许你与该类的任何特定功能进行交互。

**（7）组件 Component**

组件（Component）是一种可以添加到 Actor 的功能。将组件添加到 Actor 时，Actor 可以使用组件提供的功能。例如：

- 点光源组件将使 Actor 像点光源一样发光。
- 旋转移动组件将使 Actor 转动。
- 音频组件将使 Actor 能够播放音效。

注意，组件必须连接到 Actor，不能独自存在。必须依附在某个 Actor 身上才有意义。就像一台音响必须装在车（Actor）上才能跟着跑。

**（8）Pawn**

Pawn[11]是 Actor 的子类，作为游戏内的形象或人像，例如游戏中的角色。玩家或游戏的 AI 可以控制 Pawn，将其作为非玩家角色（NPC）。当人类或 AI 玩家控制 Pawn 时，会将其视为 被占有 。相反，当人类或 AI 玩家未控制 Pawn 时，会将其视为未被占有。

**（9）角色 Character**

角色（Character）是计划用作玩家角色的 Pawn Actor 的子类。角色子类包括碰撞设置、双足运动的输入绑定以及用于玩家控制动作的其他代码。

角色是 Pawn 的进一步特化，专门为两脚走路、像人一样运动的玩家角色准备的。自带碰撞处理、行走跳跃的输入绑定，还有各种让玩家直接操控的代码。简单说，如果你要做一个人形主角，直接选“角色”最省事，它已经把走路、跳这些基础能力都打包好了。

**（10）玩家控制器 Player Controller（遥控器）**

玩家控制器（Player Controller）获取玩家输入，并将其转换到游戏内的互动中。每个游戏内部都至少具有一个玩家控制器。玩家控制器通常操控一个 Pawn 或角色作为玩家在游戏中的呈现方式。玩家控制器还是多人游戏的主要网络互动点。在多人游戏期间，服务器具有游戏中每个玩家的玩家控制器的一个实例，因为它还必须对每个玩家进行网络功能调用。每个客户端都只有一个与玩家对应的玩家控制器，并且只能使用其玩家控制器与服务器进行通信。关联的 C++类是 PlayerController。

玩家控制器负责接收你的键盘、鼠标、手柄输入，然后把它翻译成游戏里的动作。每个

[11] 延伸小知识：在国际象棋里，Pawn 是步兵的意思。在 UE 中专门用来代表游戏里的“被控制形象”，比如一个角色、一架飞机。玩家或者 AI 都可以控制它。当它被玩家或 AI 控制时，我们叫它“被占有”；没人控制的时候，就叫“未被占有”。你可以把它想象成棋盘上的一枚棋子：你的手（玩家控制器或 AI 控制器）拿起来，它就是“被占有”的；放下不管，它就只是摆在那里。

游戏至少有一个玩家控制器。在多人游戏中，服务器会为每一个玩家准备一个控制器实例，但你自己的电脑上只有属于你的那一个遥控器。它通常控制着一个 Pawn 或角色，作为你在游戏世界里的化身。这就像你手中的遥控器，你在外面按，里面的角色才动。

**（11）AI 控制器 AI Controller（自动驾驶程序）**

就像玩家控制器操控 Pawn 作为玩家在游戏中的呈现方式，AI 控制器（AI Controller） 操控 Pawn 在游戏中呈现非玩家角色（NPC）。默认情况下，Pawn 和角色都以基本 AI 控制器终结，除非它们被玩家控制器专门操控或者收到指令不允许为自己创建 AI 控制器。关联的 C++类是 AIController 。

和玩家控制器类似，只不过它是专门用来操控非玩家角色（NPC）的“大脑”。默认情况下，一个 Pawn 或角色如果没被玩家控制器占有，就会由 AI 控制器接管，让它表现出巡逻、攻击等行为。除非你特意告诉它“不许自己动”，否则它都会有个基础的 AI 来驱动。

**（12）玩家状态 Player State（记分牌）**

玩家状态（Player State）是游戏参与者在游戏中的状态，例如人类玩家或模拟玩家的机器人。非玩家 AI 作为游戏世界的一部分而存在，没有玩家状态。玩家状态可能包含的玩家信息示例包括：

- 名称
- 当前级别
- 血量
- 得分

对于多人游戏，所有玩家的玩家状态都在所有机器中存在，可以将数据从游戏中复制到客户端以保持内容一致。这不同于玩家控制器，因为玩家控制器仅存在于玩家所使用的机器上。

玩家状态记录的是游戏里每个参与者的公开信息，比如：名字、等级、血量、得分等。只有真正的玩家或模拟玩家的机器人有这个东西，纯 AI 的 NPC 是没有的。在多人游戏中，所有人的玩家状态都会在所有机器上同步，这样你才能看到别人的血条和分数。这就像比赛时场边那块公开展示的记分牌，每个人都能看。

**（13）游戏模式 Game Mode（裁判+规则书）**

游戏模式（Game Mode） 设置要运行的游戏的规则。这些规则可以包括：

- 玩家如何加入游戏。
- 游戏是否可以暂停。
- 任何游戏特定行为，例如获胜条件。

可以在项目设置中设置默认游戏模式，并针对不同的关卡重载游戏模式。无论你选择以何种方式实施，每个关卡都只能有一个游戏模式。在多人游戏中，游戏模式新存在于服务器上，而规则将复制（发送）到每个连接的客户端。关联的 C++类是 GameMode。

**（14）游戏状态 Game State（公共告示板）**

游戏状态（Game State）是一个容器，包含你要在游戏中复制到每个客户端的信息。简而言之，它是每个连接的人的"游戏状态"。游戏状态可能包含的内容示例包括：

- 与游戏得分相关的信息。
- 比赛是否开始。
- 根据世界中的玩家数量确定生成 AI 角色的数量。

对于多人游戏，每个玩家的机器上都有一个本地游戏状态实例。本地游戏状态实例从游戏状态的服务器实例获取更新的信息。关联的 C++类是 GameState。

游戏状态是一个信息容器，存放着需要让所有玩家都知道的东西，比如：比赛开始了吗？当前总比分是多少？根据玩家数量要生成多少个 AI 敌人？每个玩家的机器上都有一个本地的副本，这个副本会不断从服务器那边更新。它好比场馆里的公共大屏幕，显示着比赛时间、全局比分，确保每个人都看到一样的信息。

**（15）笔刷 Brush（基础积木）**

笔刷（Brush）是一种用来定义简单 3D 形状的 Actor，例如立方体或球体。可以将笔刷放置在关卡中以定义关卡几何体（这些几何体称为二进制空间分区或 BSP 笔刷）。例如，可以把它们放在关卡里快速搭建墙壁、地板、天花板这些基础结构（这在引擎里叫 BSP 笔刷）。如果要快速把一片区域围起来，笔刷是最高效的办法，就像用大块积木搭个轮廓。

**（16）体积 Volumes（特殊力场）**

体积（Volumes）是带有边界的 3D 空间，本身可以带有各种效果，根据连接到体积的效果，具有不同的使用方法。例如：

- 阻挡体积（Blocking Volumes） 是可见的，用于阻止 Actor 通过它们。
- 施加伤害体积（Pain Causing Volume） 对与其重叠的任何 Actor 造成持续伤害。
- 触发器体积（Trigger Volumes） 的编程方式为，在 Actor 进入或退出体积时触发事件。

**（17）关卡 Level（地图/小关）**

关卡（Level）是你定义的 gameplay 区域。关卡包含玩家可以看到并与其交互的所有内容，例如几何体、Pawn 和 Actor。虚幻引擎将每个关卡保存为单独的 .umap 文件，这也是为什么你在某些情况下会看到它们被称为 地图（Maps）。这就像超级玛丽里的 1-1、1-2，每一小关都是一个关卡。

**（18）游戏世界 World（大容器）**

世界（World）是构成游戏的所有关卡的容器。处理关卡的流送和动态 Actor 的生成（创建）。世界是一个总容器，把游戏用到的所有关卡都装在一起。它还负责处理关卡的“流送”（从一个区域无缝加载到另一个区域）以及动态生成 Actor。你正在玩的一整部游戏背后，就是一个世界在统筹着所有关卡的运转，让切换场景变得连贯自然。

## 2.1.2. 工具、编辑器、系统

可以把虚幻引擎 5 想象成一个超级游戏制作工厂，里面分成三种东西：小工具、专用工作室、以及整个支撑系统。

- **工具**(Tool)：就像是你手里的螺丝刀、画笔或卷尺。每一样工具只干一件特定的小事，比如在场景里“摆放一棵树”或“刷一片草地”。

- **编辑器**(Editor)：更像是一个配备了全套设备的专业房间。比如你想搭整个游戏关卡，就去“关卡工作室”；想给模型上色画材质，就去“材质画室”。它是很多工具组合在一起的地方。
- **系统**(System)：则是背后的整套规则和流水线。比如“蓝图系统”就是用连线的方式让游戏动起来的整套机制，“材质系统”是让所有表面看起来真实的那套底层技术。有时候，一个房间（编辑器）的名字和它背后的那套机制（系统）名字很像，但你可以这样记：编辑器是你坐在前面操作的那个界面，系统是让一切运转起来的幕后功臣。

另外，这些工具和房间有些是引擎自带的，有些是可以选择的“插件包”——就像手机上的 App，你需要什么功能再打开，不用就关掉，免得桌面太乱。下面就是这间工厂里最重要的几个“工作室”，了解它们是干什么的，以后你就不容易迷路了。

**（1）关卡编辑器：你的主工地**

这是你打开引擎就看到的默认大房间，也是你待得最久的地方。你在这里把所有东西拼成一个可玩的游戏关卡：拖入灯光、敌人、道具，刷上地形，再用蓝图写好规则。简单说，造世界的地方。

**（2）静态网格体编辑器：模型美容室**

当你导入一把椅子或一块石头的 3D 模型后，就在这里查看它、调整它的碰撞体积、设置它的精细度（远处看起来简单点，近处看起来细致点）。给模型做体检和微调的地方。

**（3）材质编辑器：材质画板**

你想让地板看起来是木头还是铁锈？想让它发光或透明？这个节点连线的画板就是专门给模型“画皮”的，最终做出一个个材质球，然后贴到模型上。

**（4）蓝图编辑器：可视化编程台**

不用写代码，像画流程图一样把节点连起来，就能让一扇门自动打开、让角色吃到金币后变大。所有用这种方式做出来的东西，本身也常常被叫做“蓝图”。这是让一切动起来的魔法连线板。

**（5）物理资产编辑器：布娃娃装配间**

如果你想让角色的骨骼网格体被撞到后软趴趴地倒下，或者尾巴能甩来甩去，就在这里给它的骨架加上物理形体。可以用自动化工具快速生成一套基础设置，就像给模型绑上无形的弹簧和气球。

**（6）行为树编辑器：AI 大脑设计器**

想让敌人会巡逻、看到你就追、血量低了就逃跑？这里用另一种节点树的方式给 NPC 编排各种行为逻辑。给怪物写“脑筋急转弯”的地方。

**（7）Niagara 编辑器：特效魔法工坊**

火焰、烟雾、闪电、爆炸、雪花……所有的粒子特效都出自这里。你可以把发射器保存起来，下次直接拿来用或改一改。制造视觉奇观的烟花工厂。

**（8）UMG 虚幻运动图形界面编辑器：UI 装修工具**

游戏里的血条、弹药数、菜单、对话框，都是在这个可视化的工具里拼出来的。拖个按

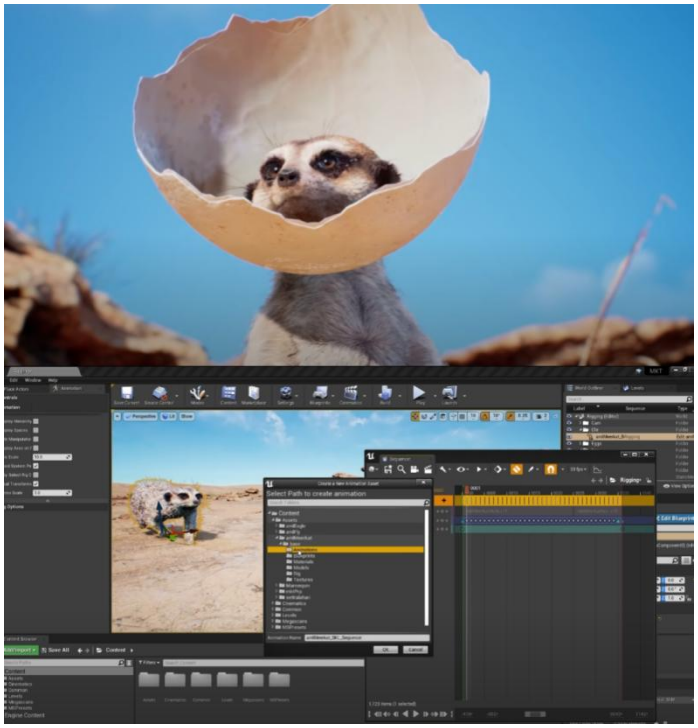
钮，拉个进度条，设计完就能直接看到效果。给游戏做电子屏幕和触控板的地方。

**（9）字体编辑器：字体管理室**

用来导入字体、调整字形显示效果，确保游戏里的文字在不同大小下都清晰可读。管文字怎么长的后勤部。

**（10）Sequencer 编辑器：电影导演剪辑室**

如果你想做过场动画，比如开门的剧情镜头，就用这个多轨道编辑器。把摄像机移动、角色动作、音乐、特效全排进一条时间轴上。你的私人电影导演工作台。



**（11）动画相关的两个重要棚**

- 动画编辑器：可以编辑骨骼、动画蓝图等一切与角色动作相关的资产。
- Control Rig 编辑器：让你直接在引擎里给角色绑骨骼、做操控手柄，不用再导入导出到其他软件。在引擎内直接调动作的操纵室。

**（12）Sound Cue 编辑器：音效调音台**

想把多个声音随机播放、根据环境混音、或者让脚步声踩在不同地面发出不同声响？在这里把各种音效节点连起来，最后输出成一个智能音效。

**（13）媒体编辑器：视频播放设置卡**

不能剪辑视频，但能指定引擎里播放哪段视频文件或网络流，并设置要不要自动播放、循环几次。管“电视机”换台和播放模式的地方。

**（14）针对大型演出和硬件的特殊编辑器**

- **nDisplay 3D 配置编辑器**：如果你要在巨型弧幕、穹顶或拼接墙上显示你的虚幻场景，就需要在这配置让多台显示器同步渲染出整幅无缝画面。
- **DMX 库编辑器**：用于控制真实的舞台灯光、烟雾机、激光等演出设备，通过 DMX 协

议让虚拟世界和现场表演同步。把引擎变成舞台控制台的关键。

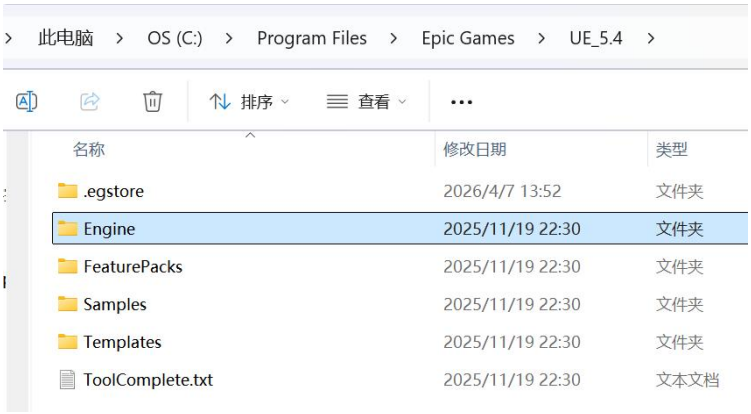
## 2.1.3. 目录结构

接着用之前那种逛工厂的感觉，来把虚幻引擎里的文件夹和目录结构讲清楚。你不需要一开始就背下来，只要知道每样东西大概放在哪个仓库里，以后找东西不迷路就行。

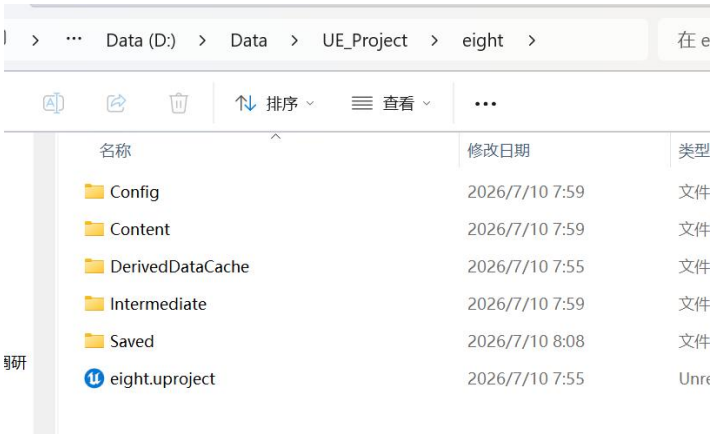
**（1）两大核心区域：你自己的和引擎的**

想象你的电脑上装了一套虚幻引擎。在最高一层的目录里，你会看到两个最主要的“大本营”：

- Engine 文件夹：这是引擎自己的家。里面装着引擎运行需要的所有东西——源代码、自带工具、内置内容等等。正常使用引擎时，你一般不会去改动这里的东西。




- 你创建的游戏项目文件夹：比如你新建了一个叫“eight”的项目，就会有一个对应的 eight 文件夹。所有只属于你这个游戏的文件，全都住在这里。




**（2）还有有哪些重要的房间？**

- Templates（模板库）：新建项目时那些“第一人称”“俯视角”“空白”的项目模板，都存放在这。相当于给你准备好了几种毛坯房户型。
- GenerateProjectFiles.bat：这是个一键生成工具。如果你要用 Visual Studio 写 C++ 代码，就运行它，它会帮你把引擎和项目代码整理成 Visual Studio 能打开的方案文件。
- Default.uprojectdirs：一个不起眼的辅助文件，能帮引擎自动发现在子目录里藏着的项目。

**（3）两边都有的通用仓库**

下面这些文件夹，在引擎自己的家里，和你游戏项目的家里，都可能会看到。它们的名字一样，但装的东西只为自己服务。理解它们，就基本掌握了虚幻的文件结构。

- Binaries（可执行与编译产物）：放编译后生成的可执行文件、动态链接库等。就是代码“变成”程序后出来的成品或半成品。你不需要手动操作这里。
- Build（构建相关文件）：存放编译引擎或游戏时需要的辅助文件，比如给特定平台（手机、主机）打包时用的脚本或设置。
- Config（配置文件）：各种.ini 设置文件，控制着引擎和游戏的行为。你可以把它理解为设置面板的存档。一个很重要的规则：游戏项目 Config 里的设置，会覆盖引擎 Config 里的同名设置。所以你想修改，改你自己项目的 Config 就好。
- Content（内容资产）：这个目录太太太重要了。你在引擎里导入的所有看得见摸得着的东西——模型、贴图、声音、动画、材质——全都存在这里。引擎自带的免费资源在 Engine/Content；你自己的东西全放在你项目的 Content 文件夹。你游戏只能使用你自己 Content 里的东西，但引擎自带 Content 是大家都能用的。
- DerivedDataCache（派生数据缓存）：引擎在加载某些内容时，会自动提前算好一些“中间结果”保存在这，下次打开就快很多。如果删了它，下次打开引擎会花很长时间重新生成，所以别手痒删它，除非在清理缓存时特意操作。
- Intermediate（临时文件）：编译代码或者构建项目时产生的临时文件。在游戏项目里，着色器编译后的文件也会暂存于此。这个文件夹可以删，重建项目时会重新生成。
- Plugins（插件）：插件是一套可以灵活添加或移除的功能包。引擎自带的插件在 Engine/Plugins，你给项目单独启用的插件在项目的 Plugins 文件夹里。
- Saved（存档与记录）：自动保存的关卡、备份的配置、运行日志、崩溃报告等全在这里。测试游戏闪退了？可以来这里翻日志找线索。这个文件夹里的东西也可以删除，引擎下次会重建。
- Source（源代码）：引擎或游戏的所有 C++ 源代码文件。如果你只是用蓝图，几乎不用碰这里。但在你游戏项目的 Source 目录下，代码是按照“模块”来分文件夹的，每个模块里会有：
  - Private：放.cpp 实现文件，内部逻辑。
  - Public：放.h 头文件，对外公开的声明。

**（4）只有引擎才有的专属区域**

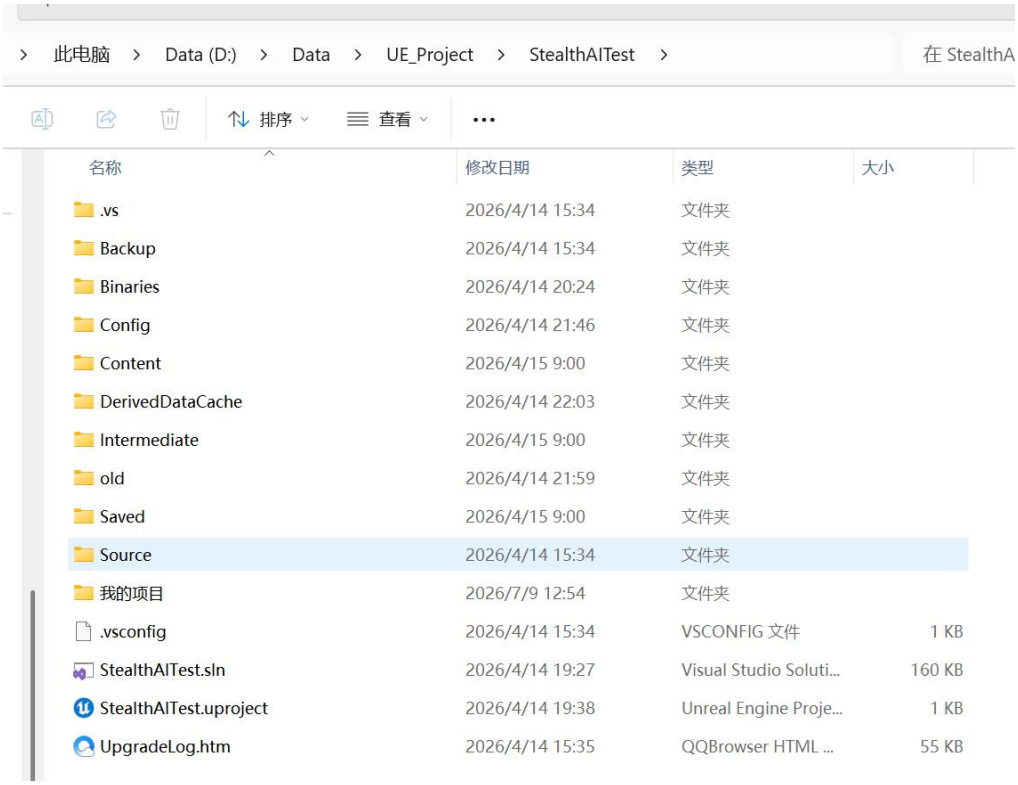
下面这些基本只在 Engine 目录下出现：

- Documentation（文档）：引擎自带文档，有 HTML 网页版，也有源用的 Markdown 文件。
- Extras（额外杂项）：一些辅助和工具文件。
- Programs（配套程序）：存放像 Unreal Frontend、Unreal Header Tool 等引擎配套程序的项目配置和日志。
- Shaders（着色器）：存放引擎着色器的源文件（.usf），控制着所有炫酷的渲染效果。

**（5）你的游戏项目目录一览**

当你打开自己的游戏项目文件夹时，常见到这些：

| *文件夹* | *通俗解释* |
| --- | --- |
| Binaries | 你的游戏编译后的可执行文件。 |
| Config | 专属于你这款游戏的设置，覆盖引擎默认设定。 |
| Content | 你这游戏的所有美术、声音等资产，你的“私家仓库”。 |
| Intermediate | 临时构建文件，可删，重新生成。 |
| Saved | 自动保存、日志、配置备份，可删，会自动重建。 |
| Source | 游戏 C++ 代码，按模块放好。 |




另外，如果你用 Visual Studio 打开项目，会在解决方案浏览器里看到一个虚拟的“External dependencies”文件夹，它其实只是给你看的公有引擎头文件，不用管。

**关于模块 vs.插件：**

- 模块：纯代码的积木块。你创建的项目本身就是一个模块，由.Build.cs 文件来描述。模块只带代码，不能包含资产。
- 插件：带装备的完整扩展包。它有代码，有资产（比如贴图、模型），还有一个.uplugin 描述文件，可以轻松打包出来发给别人，装到别的项目里直接用。想做通用功能或工具包，就做成插件。

刚开始只需要重点关注 Content 和 Config 这两个文件夹，因为你导入的资源和游戏设置都在那里。随着你慢慢开始写 C++，再去熟悉 Source 目录的结构也不迟。把这些文件夹理解成你家厨房里分类放好的调料和食材，以后做起菜来就得心应手了。

## 2.1.4. 测量单位

虚幻引擎就像一个小型的物理世界，要在这个世界里搭场景、做游戏，就必须先定好“尺子”和“秤”该怎么看。不然你把一面墙做成了 10 厘米高，角色 2 米高，那肯定没法玩。所以，引擎里所有东西都有默认的测量单位，搞清楚这个，你的世界才不会乱套。

**（1）默认单位：都用国际通用的那套**

虚幻引擎默认使用的是我们上学时学的国际单位制（SI），具体是这样的：

- 距离/长度：厘米 (cm)，你在引擎里看到的数字，1 基本单位就是 1 厘米。一个 180 的单位，就是 1.8 米。
- 质量：千克(kg)，一个物体的重量单位。
- 时间：用秒(s)或分 (min) 来算。
- 角度：度(deg)，旋转一圈是 360 度。
- 速度：米/秒(m/s)，注意不是厘米/秒哦。
- 温度：摄氏度(C)。
- 力：牛顿(N)，比如推箱子用的力。
- 扭矩：牛顿·米(N·m)，比如拧螺丝的力道。

可以把这套单位想象成游戏世界里的国际普通话，引擎自带的物理、移动等系统，都是按这套单位来计算的。

**（2）想换成英寸、磅、华氏度怎么办？**

如果更习惯用英制单位，比如面向美国市场，或者只是个人偏好，可在编辑器里随时更改显示用的单位。操作很简单：

- 在顶部菜单栏点击 编辑（Edit）→项目设置（Project Settings）。
- 在弹出的窗口左侧，找到 编辑器（Editor）→外观（Appearance）。
- 在右侧往下找，找到单位（Units）分类，展开里面的 高级（Advanced）。
- 这时你就能看到各种“量”及其当前单位了。点击旁边的下拉菜单，就能自由切换，比如把长度从厘米改成英寸(in) 或英尺 (ft)，把质量从千克改成磅 (lb)。

注意：这里改的只是编辑器里数字的显示方式，并不会改变引擎内部的实际物理运算（内部仍然用国际单位）。就好比你可以把车速表调成“英里/小时”显示，但车子本身的机械计算还是以米和秒为基准，只是你眼睛看到的不同了。所以放心调整，不会把物理效果改乱。

**（3）常用单位速查**

虚幻引擎支持的单位非常多，下面挑几类最常用的给你看看，以后需要时再回来翻：

- 长度和距离
  - 公制：微米 (μm)、毫米 (mm)、厘米 (cm)、米 (m)、千米 (km)
  - 英制：英寸 (in)、英尺 (ft)、码 (yd)、英里 (mi)
  - 特别提示：还有“光年 (ly)”，做太空游戏可能会用到。
- 质量和重量

    - 公制：毫克 (mg)、克 (g)、千克 (kg)、吨 (t)
    - 英制：盎司 (oz)、磅 (lb)、英石 (st)
- 速度：米/秒 (m/s)、千米/秒 (km/s)、英里/时 (mph)
- 温度：摄氏度 (C)、华氏度 (F)、开氏度 (K)
- 时间和频率
    - 时间：纳秒 (ns)、毫秒 (ms)、秒 (s)、分 (min)、时 (hr)、天 (d) 等
    - 频率：赫兹 (Hz)、千赫兹 (kHz)、转/分 (rpm)
- 力和压强
    - 力：牛顿 (N)、千克力 (kgf)、磅力 (lbf)
    - 压强：帕斯卡 (Pa)、千帕 (kPa)、兆帕 (MPa)
- 其他
    - 百分比 (%)：0 到 100 的数字。
    - 乘数 (x)：表示倍数，没有实际单位。
    - 像素密度 (PPI)：像素/英寸。

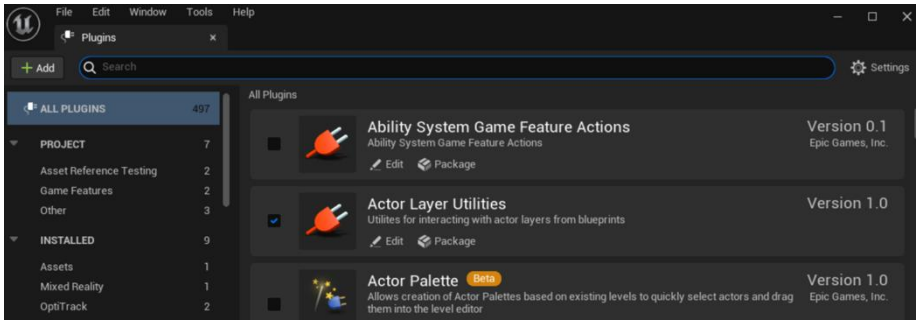
还有流明、勒克斯等光照单位，以及字节 (B) 到太字节 (TB) 等数字信息单位，做存档或光效时可能会用上。

刚入门你只需要记住：虚幻里 1 个单位长度 = 1 厘米，然后默认都用国际单位，这样角色、物理、移动全对得上。以后想改显示单位，就去项目设置的“外观”里调。不用特意背那些缩写，大概知道有哪些种类就够了，用到再查，熟能生巧。

## 2.1.5. 插件

**（1）引擎内安装插件**

要启用虚幻引擎插件，请执行以下步骤：在主菜单中，转到编辑（Edit） > 插件（Plugins）。这将打开插件（Plugins）窗口。



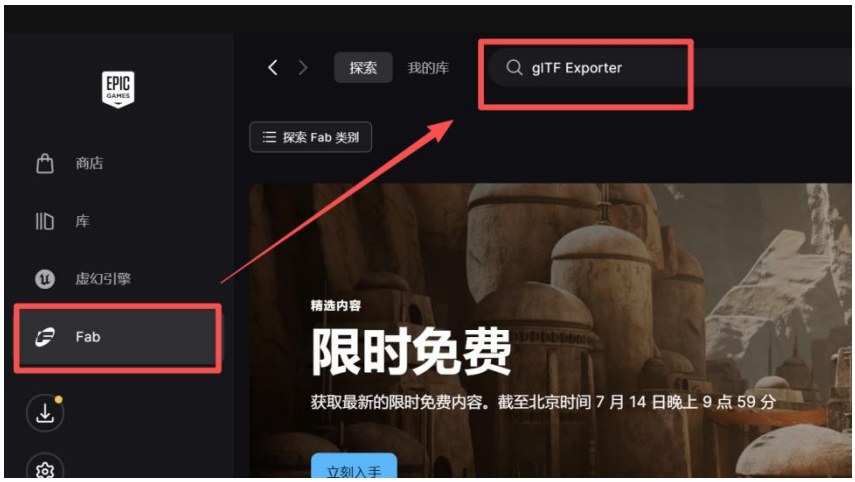

要启用插件，请点击其旁边的复选框。注意，对于非生产就绪的插件，例如测试版插件，可能会看到警告，要求你确认你想启用该插件。

然后重新启动虚幻引擎。第三方插件可能要求执行额外步骤，你才能将其启用。请注意，Epic Games 对第三方插件的内容概不负责。

**（2）从 Fab 安装插件**

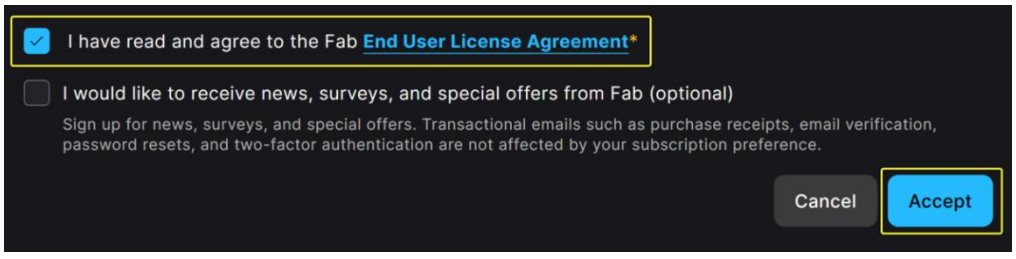
虚幻引擎自带能提供许多功能的插件，同时你也可以从 Fab 安装额外的插件，直接在 Fab 网站上浏览和下载虚幻引擎插件，也可以在 Epic Games 启动器的 Fab 选项卡中访问它们。要从该网站下载插件，请按以下步骤操作：

在 Epic Games 启动器中找到 Fab 选项卡，搜索要安装的插件，并点击缩略图打开列表页面：




要下载免费插件，请按以下步骤操作：在插件的页面上，点击添加到我的库（Add to My Library）：


接受 FabEULA：



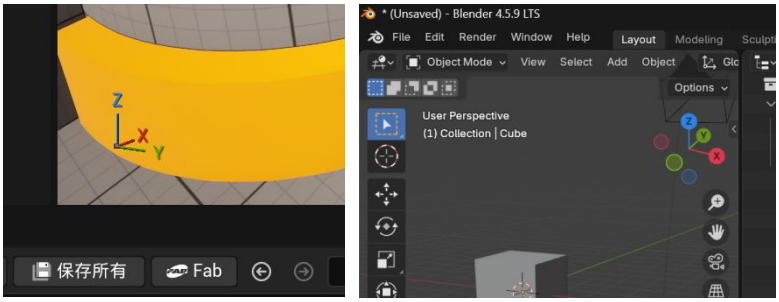

此时，你的免费插件就会出现在你位于 Fab 网站上的 Fab 库以及 Epic Games 启动器中。

## 2.1.6. 坐标系与坐标空间

### （1）坐标系

为什么要先搞懂坐标系？想象一下，你要在游戏里放一把椅子。椅子放在哪？离门多远？朝哪个方向？要不要放大一点？这些问题都需要用数字来精确描述。坐标系[12]就是一套**通用的空间描述语言**，让引擎、美术、程序都能准确理解某个东西到底在哪、朝哪、多大。

如果你从 Maya、Blender、3ds Max 等别的软件往虚幻里导模型，发现模型倒在地上或者朝向不对，十有八九就是坐标系没对齐。




**虚幻与 Blender 坐标系正好相反**

虚幻引擎使用的是左手坐标系，并且 **Z 轴**朝上。什么叫左手坐标系？你可以自己做一个实验：伸出左手，食指指向正前方，是 **X 轴**正方向。伸出中指，指向正右方，那是 **Y 轴**正方向。现在竖起大拇指，指的方向，就是 **Z 轴**正方向，也就是上方。这就是左手定则。虚幻里 **X** 是前，**Y** 是右，**Z** 是上。


**XYZ 左手坐标系**

原点（Origin）就是三根轴交汇的那个点，坐标是(0, 0, 0)。所有物体在世界中的位置，都是相对于原点来计算的。

### （2）坐标空间

坐标系是一个大框架，但同一个物体，在不同的坐标空间里看，坐标值是不一样的。打个比方：你在教室里坐在第三排第四列，这个位置是相对于教室说的。但如果你在火车上，相对于火车来说你的位置没变，相对于地面来说你的位置一直在变。参照系不同，坐标就不

[12] 二维坐标的发明，源自 1630 年代数学家笛卡尔卧病住院时观察天花板而来。后来又有了极坐标、三维坐标等。

同。在虚幻引擎里，最常用的坐标空间有这几个：

**世界空间（World Space）**全局坐标系。整个世界共用一套坐标轴，原点在关卡的正中央(0, 0, 0)。任何 Actor 放在关卡里，它在世界空间中的位置、旋转、缩放，都是相对于这个全局原点来计算的。1）位置：相对于世界原点的偏移量。2）旋转：相对于世界坐标轴的朝向。3）缩放：相对于原始大小的倍数。**什么时候用世界空间？**当你想让一个东西朝世界的正北方向移动，或者放到地图的某个绝对位置时，用的是世界空间。

**局部空间（Local Space / Object Space）。**每个物体自己的坐标系。每个 Actor 都有自己的小世界，原点在自己身上，三根轴跟着自己转。局部空间描述的是：一个东西相对于它爹的位置。1）如果 Actor 没有父级（直接放在关卡里），那么它的局部空间=世界空间。2）如果 Actor 有父级（比如一把剑挂在角色手上），那么它的局部坐标是相对于父级的，剑相对于手的位置是(0, 5, 0)，但手在动，剑的世界坐标也跟着变。**什么时候用局部空间？**当你想让子物体永远保持在父物体的某个相对位置时，用的是局部空间。比如角色的武器、帽子、背包，它们跟着角色动，但相对位置不变。

**视图空间（View Space / Camera Space）。**以摄像机为原点的坐标系。把摄像机放在原点，摄像机的正前方是 Z 轴（或者某个轴，取决于具体实现）。世界空间转成视图空间后，距离保持不变，因为只是平移和旋转，没有缩放。**什么时候用视图空间？**在渲染管线的早期阶段，引擎会把所有物体从世界空间转换到视图空间，方便后续计算哪些东西在摄像机前面、哪些在侧面。

**屏幕空间（Screen Space）。**把 3D 世界投影到 2D 屏幕上之后得到的坐标。坐标值范围通常是[-1, 1]（标准化设备坐标），表示物体在屏幕上的相对位置。**什么时候用屏幕空间？**当你想在屏幕上显示一个血条，让血条始终钉在某人头顶时，你需要把这个人的世界坐标转成屏幕坐标，才知道血条该画在屏幕的哪个位置。

**视口空间（Viewport Space）。**屏幕空间再乘以屏幕分辨率，得到的就是像素坐标。比如(1920, 1080)表示屏幕右下角。**什么时候用视口空间？**画 UI 的时候。UMG 里的控件位置，就是用视口空间（像素）来定位的。

**虚幻引擎各坐标空间一览**

| 空间 | 通俗理解 | 特点 |
| --- | --- | --- |
| 世界空间 | 全局坐标 | 原点在关卡中心，所有物体共用一套坐标系 |

| 空间 | 通俗理解 | 特点 |
| --- | --- | --- |
| 局部空间 | 相对于“爹”的位置 | 每个物体有自己的坐标系，跟着自己转 |
| 视图空间 | 以摄像机为原点 | 距离和世界空间一样，只是换了原点 |
| 屏幕空间 | 3D→2D 投影后的坐标 | 范围[-1, 1]，用于判断物体在屏幕上的相对位置 |
| 视口空间 | 屏幕上的像素坐标 | 比如(1920, 1080)，用于 UI 定位 |

**（3）空间变换**

空间是怎么转换的？从一个空间到另一个空间的转换，叫做空间变换。虚幻引擎里命名非常规范，格式永远是从哪到哪。比如：

- WorldToView：从世界空间到视图空间
- LocalToWorld：从局部空间到世界空间
- TangentToWorld：从切线空间到世界空间

渲染管线中最常见的转换流程：

**局部空间→世界空间→视图空间→屏幕空间→视口空间**

每一步都是矩阵运算，但作为开发者，你不需要手动算这些，引擎帮你全包了。你只需要知道：在哪个空间下操作，得到的效果是什么。

综上，坐标系是虚幻世界的空间语言，左手系 Z 轴朝上是基本语法；而坐标空间就是不同参照系下的方言，世界空间说绝对位置，局部空间说相对位置，屏幕空间说投影位置。搞懂这些，你就掌握了在 3D 世界里定位一切的基础能力。

**（4）切线空间**

在渲染和材质领域，还有一个空间，就是**切线空间**（Tangent Space）——法线贴图的贴身坐标系。切线空间就是模型表面的本地坐标系，让法线贴图能像贴纸一样，贴在任何姿势的皮肤上都不变形。如果说世界空间是北斗定位系统，局部空间是以物体自身为原点，那切线空间就是贴在一张脸皮上的坐标纸，紧紧地贴在模型表面的每一个三角形上，跟着表面走，随着表面转。在模型表面的任意一个点上，切线空间由三个互相垂直的轴组成：

| 轴 | 英文名 | 方向 | 通俗理解 |
| --- | --- | --- | --- |
| 法线 | Normal（N） | 垂直于表面，指向外 | 垂直扎出表面一根针 |
| 切线 | Tangent（T） | 平行于表面，指向纹理坐标的 U 方向（水平） | 沿着纹理贴图横着走 |
| 副切线 | Bitangent/Binormal（B） | 平行于表面，指向纹理坐标的 V 方向（竖直） | 沿着纹理贴图竖着走 |

右手定则：在虚幻引擎中，切线（T）×副切线（B）=法线（N）。只要知道其中两个，就能算出第三个。

为什么需要切线空间？最直接的原因就是法线贴图（Normal Map）。**法线贴图**是一张紫

色的图片，里面存储的不是颜色，而是方向数据，具体来说，是每个像素的法线方向。但有一个关键问题：法线贴图里存的方向，是以谁为参照系的？如果存的是世界空间的方向，那这张贴图只能用在特定朝向的模型上。比如一张朝北的墙，法线全朝北；一旦你把这张贴图贴到朝东的墙上，光照就全错了，因为方向不对了。

解决方案就是切线空间：法线贴图里存储的，是相对于模型表面局部方向的偏移量。也就是说，贴图里存的是相对于这根法线，往左偏了多少、往右偏了多少、往外凸了多少。贴图里 (0, 0, 1)（纯紫色）表示“完全垂直于表面”，没有凹凸。贴图里 (0.5, 0.5, 0.7) 表示“往切线方向和副切线方向稍微偏了一点”，代表表面有凹坑或凸起。这么做的好处是：一张砖墙的法线贴图，可以贴在任何朝向的模型上，贴在平地上、贴在弧形的柱子上、贴在歪着的斜坡上，都能正确显示凹凸效果。因为引擎会在渲染时，把贴图里的相对方向转换到当前模型表面的实际方向上。

## 2.2. 内容浏览器

### 2.2.1. 内容浏览器

内容浏览器，就是游戏项目的数字资产仓库。在游戏里用到的所有东西，包括模型、贴图、声音、材质、蓝图等全都存放在这里。可把内容浏览器看成是一个超级好用的文件管理器，专门为游戏开发设计。


内容浏览器能帮你做什么？这可不只是个文件夹，是一个功能强大的控制中心，主要能干这些事：

- 浏览与管理：像逛超市一样，随意查看项目里的所有资产，并直接拖进关卡里使用。
- 智能搜索：用名字快速搜东西，还能配合高级筛选器，比如“只显示石头材质”，瞬间锁定目标。
- 资产整理：把常用的资产放进“收藏夹”（私有、本地或共享集合），方便随时取用。
- 健康检查：自动帮你找出可能有问题的资产，让你提前处理，避免崩坏。
- 资产搬移：能在不同文件夹之间，甚至不同项目之间，安全地迁移资产，不会把链接搞丢。

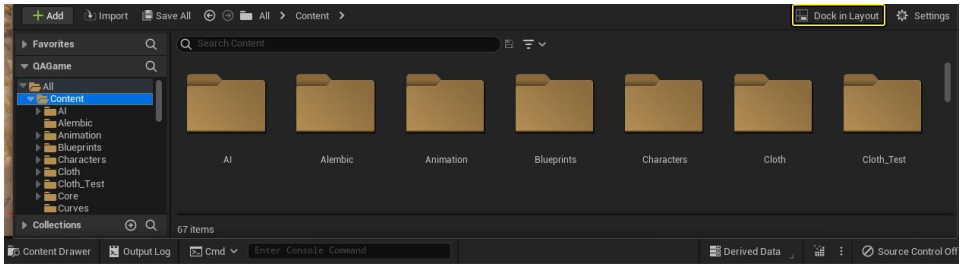
三种方式打开它

- 从菜单：顶部菜单栏点击 窗口（Window）→选择 内容浏览器（Content Browser）。
- 从快捷菜单：点击主工具栏上的 创建（Create） 按钮，也能找到它。
- 从底部栏：点击编辑器最底部工具栏上的 内容侧滑菜单（Content Drawer） 按钮。这会弹出一个临时的浏览器，你可以选择把它固定在界面上。


小技巧：可以同时打开最多 4 个内容浏览器。比如，一个只看静态模型，一个只看材质，拖拽管理非常方便。

认识“内容侧滑菜单”。这是一个特殊的快捷版内容浏览器，设计得非常顺手：

- **呼出方式**：除了点底部按钮，Windows 还可以用快捷键 Ctrl + 空格键，Mac 系统的话用 Cmd + 空格键。这个快捷键用得最多，记住的话效率大增。
- **自动消失**：当你用完，点击其他地方，它就会自动缩回去，不占屏幕空间。
- **让它一直显示**：如果想让它固定住，点它上面的 停靠在布局中（Dock in Layout） 按钮就行。




## 2.2.2. 内容浏览器界面

内容浏览器（Content Browser）分为以下几个区域：