**虚幻引擎心得**

一.地形系统的深入浅出

使用虚幻引擎的地貌系统可以创建广袤的、开放的世界环境，这得益于地貌系统强劲的LOD系统和高效的内存使用，可以创建大于terrain地形系统几个数量级的地形地貌。虚幻引擎现可处理最大精度为 8192x8192 的高度图，支持巨型室外世界场景，无需对引擎或工具进行修改即可快速便捷地创建出辽阔庞大的室外环境。

1.地形比静态网格体更适合创建巨大地形

地形的顶点数据为每个顶点 4 字节，静态网格体以 12 字节矢量储存位置，切线 X 和 Z 矢量每个均装入 4 字节，16 位或 32 位浮点 UV 每个顶点为 24 或 28 字节。这意味着在相同的顶点密度下，静态网格体占用的内存是地形的6或7倍。地形还将其数据保存为纹理，可为远处场景流式输出未使用的LOD层级，并在靠近远处场景时从背景磁盘中进行加载。地形使用恒定的高度场，因此可更高效地存储碰撞数据。

2. 地形系统在 GPU中的存储方式

地形系统在 GPU 内存中将把地形渲染数据存储为纹理，使数据可在顶点着色器中进行查找。数据被装入 32 位纹理，高度以 R 和 G 通道的形式占据 16 位，X 和 Y 偏移在 B 和 A 通道中存储为 8 位数值。

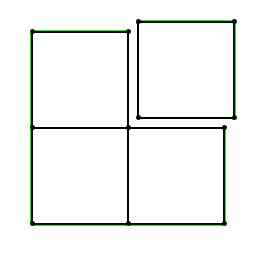
地形被划分为多个组件，即为虚幻引擎的渲染、可视性计算和碰撞的基础单元。地形中的组件尺寸均相同，且为方形。地形创建时将决定地形组件的尺寸，选择取决于需要创建的地形尺寸和细节。

3.地形的组成

（1）组件

地形由组件组成，最大支持规格为32 x 32，组件的比例可以任意搭配；组件由分段组成，分段的规格为1 x 1或2 x 2 ；分段由方格组成，支持方格的规格为7 x 7到255 x 255；地形的维度基于每个分段中的四边形数量、每个组件中的分段数量、以及地形中的组件数量。地形 Actor 拥有颜色编码，可以明确分辨每个分段的作用。地形的边缘以黄色高亮，每个组件的边缘为浅绿色，分段边缘为中绿色，方格四边形为深绿色。

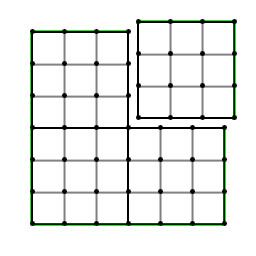
简单的地形如下图所示（轮廓为绿色），它包含四个组件。每个组件由单个四边形组成。已经分离出一个组件，用于展示组件相交位置顶点的复制方式。



（2）组件分段

组件可被划分为 1 个或 4 个（2x2）子分段。这些分段是地形 LOD 计算的基础单元。使用 4 个（2x2）子分段选项获得的高度图与在一个子分段上使用 4 次组件所获得的高度图相同，但使用的组件次数越少，运行性能越好。每个分段的尺寸（以顶点数为单位）必须为 2 的幂次方（最大为 256x256）。因此不同的 LOD 层级可存储在纹理的 mipmap 中。这将使一个组件中四边形的数量为 2 的幂次方减 1（如每个组件有一个分段）或 2 的幂次方减 2（如每个组件有四个分段）。

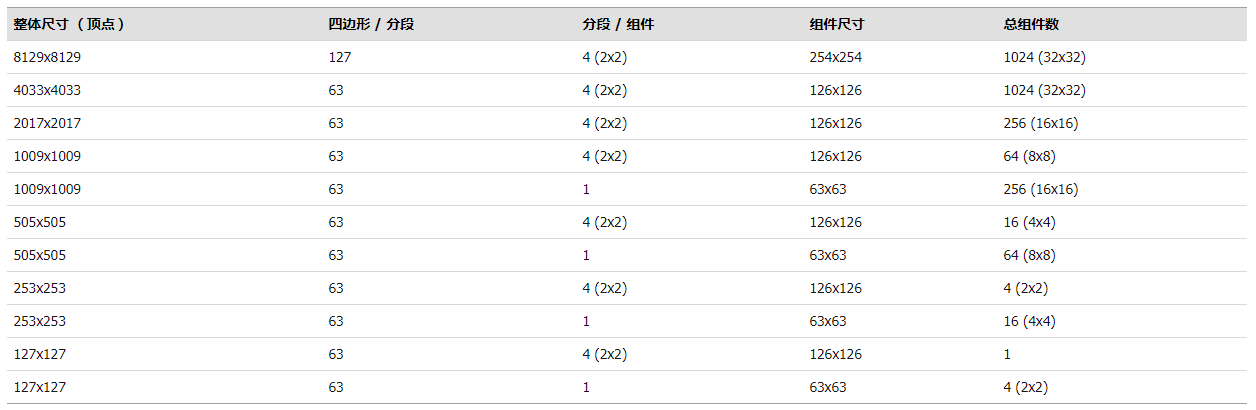
一个单独组件如下图所示（轮廓为绿色），它包含四个分段。每个分段由 9 个（3x3）四边形组成。可再次看到分段相交处复制的顶点。



（3）组件尺寸和组件总数量的合理搭配

对组件尺寸和组件总数量进行选择是为保证性能平衡的必要手段。组件尺寸越小，LOD 的过渡越快，并对更多地形构成遮挡，但需要的组件数量越多。每个组件均有渲染线程 CPU 处理消耗，每个分段均为绘制调用，因此需要将它们的数量控制到最低。

官方推荐地形尺寸如下图所示。



4.地形系统创建地形的方式

创建地形有两种方式，一种为自定义地形，一种为导入高度图地形。自定义地形方便程序人员快速搭建地形进行测试，虚幻引擎提供了一套强大的工具集，包括雕刻、平滑、平整、斜坡、腐蚀、水力侵蚀、噪点、重新拓扑、可见性等功能，完全满足工作的需要。但通常为了使地形更加逼真或者达到理想效果，会由专业美术人员使用GlobalMapper、World Machine等工具制作高度图来作为地形。高度图的格式为16bit，灰度，以小字节序排列的RAW格式。

5.高度图的分辨率对地形的影响

斜坡与平地的交汇处会因分辨率的过大变得突兀，此处的过大指的是一手获得的卫图与处理后的高度图的分辨率差值。当分辨率差距过大造成过渡不平滑时，可以在处理高度图时进行平滑处理。处理后的高度图最高点和最低点不会改变，但是过度的平滑处理会使得除最高点和最低点之外的所有点变平，在进行平滑处理时应注意实时观察。如果除最高点和最低点外有重要的高山或盆地不建议使用平滑处理，因为会改变它们的海拔。如果显得突兀的边缘较少的话，可以用引擎自带的地形修改器进行修改。

6.生成的地形无法与高度图中的点一一对应

因为地形在软件中转换过两次，转换中势必存在误差，还有在UE4中计算Scale时省略过小数字产生的误差，在测量时产生的误差，但是可以保证99%的还原，此结论是经过多次细致的测量得到的。最高点与最低点完全可以对上，其他的点要取决于在World Machine里是否进行过平滑处理，如果处理过那么就无法对应上，没有处理过要看分辨率，当地形的分辨率大于等于卫片的分辨率就可以对应上，当地形的分辨率小于卫片的分辨率就无法对应。

7.地形的动态加载（流关卡）

使用单一地形存在帧率较低，内存占用较高，不能对某一地段的精度进行修改等问题。同时在程序实际运行时我们可能并不需要加载所有地形，在地形较大视野较小时更是如此，作为一个成熟的游戏引擎，虚幻引擎为此提供了关卡流送体积域和世界构成器 。

（1）关卡流送体积域

关卡流送体积域的功能使得在游戏过程中加载或卸载关卡成为可能，并能控制关卡是否可见。这样便能将整个游戏世界分拆成一个个小的区块，在特定时刻，只有那些相关的区块才需要占据资源，并得到渲染显示。 如果制作的恰当，能够做到一个非常大的，无缝的管卡之间的切换，这能使玩家觉得仿佛置身在一个庞大的世界中。

关卡流送体积域的原理十分简单：流送关卡的加载/卸载请求基于视点是否处于关卡相关的关卡流送体积域中而发出。

具体而言，关卡流送体积域可以两种方式使用：

游戏中，玩家视点处于体积域中时，关卡流送体积域将使关卡加载；玩家视点处于体积域外时，关卡将卸载。

编辑器中，关卡流送体积域可基于透视视口摄像机的位置自动隐藏/取消隐藏关卡，用于预览关卡流送。

简单的实例如下图所示。

**流送前**



流送后



基于体积域的关卡流送易于使用，不要求脚本编写，是控制关卡流送的理想方式。此外，基于体积域的关卡流送和基于脚本的流送相比更易于维护：加载系统的需求发生变化时，调整体积域的大小即可对关卡加载/卸载行为进行修改。

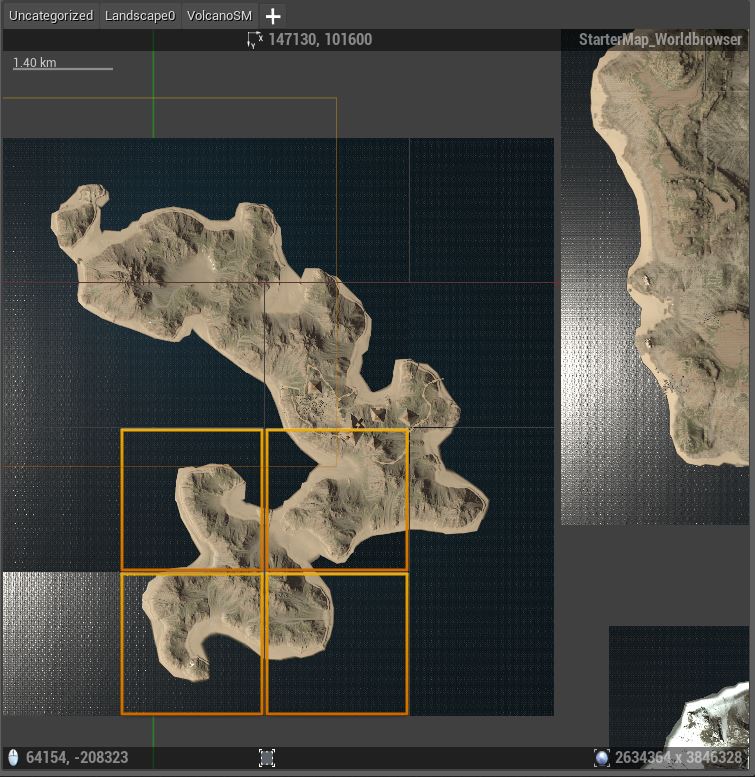
关卡流送体积域因为细腻灵活，不要求脚本编写，而更适合于较小区域的动态加载，例如从一个房间到另一个房间。但凡事有利就有弊，在面临大地形流送时关卡流送体积域就变得捉襟见肘，体积域在庞大的地形面前会因摆放问题变得不方便，没有统一管理器也使得无法从大局上进行规划，此时就应该使用世界构成器来进行流送。

（2）****世界构成器****

世界构成器 的设计是为了简单的管理大型世界。其目标之一是避免使用永久关卡来保存那些可以用流数据来处理的信息， 因为那样做会使得在一些具有较多关卡设计师的项目中，需要关卡设计师们同时并行工作的时候成为瓶颈。永久关卡不在保存任何流关卡信息，代替的做法是检索一个文件夹，并将文件夹中检索到的所有关卡都作为流关卡。每个流关卡在数据包的头上都存贮了一些信息， 世界构成器可以直接读取这些信息而不必将该关卡加载到内存中。初始状态时，除了永久关卡以外的其他所有关卡都从内存中卸载掉，可以随时手动的加载或卸载世界的某个部分。

世界构成器的每一个关卡都被视做图层，默认情况下，所有关卡都分配给 未分类 图层。图层存放了关于动态载入距离的信息， 所以分配给特定图层的关卡将继承该图层的动态载入距离设置。被分配到没有载入距离设置图层的关卡 则永远不会因为距离而被动态载入，这样可以通过蓝图或C++来对何时加载进行控制。图层不能被删除也不能被编辑，如果想修改动态载入距离设置， 必须使用期望的设置创建新图层并将关卡分配给它。在下次打开关卡时，空的图层将会被自动删除，关卡每次仅可以分配给一个图层。

下图为世界构成器，桔黄色的为图层。



**世界构成器**以关卡为操作单位，加载大地图、多分段时，世界构成器 更易于编写和判断。

二.编程心得

对虚幻引擎的结构有了初步、大体的认识，可以帮助理解引擎的设计思路，理清代码结构顺序。同时，养成良好的编码习惯，使得代码可读性更强，逻辑更清晰，对于蓝图和C++编程是十分有必要的。

1.基类Object

Object是虚幻引擎的其他类的基类。它不能被加入到世界（无法在关卡中生成或放置），但它带有数据和函数，由此可以使每一个对象可追踪，提供通用的属性和接口，统一的内存分配释放，统一的序列化模型，统计功能，调试的便利，为反射提供便利，UI编辑的便利。虚幻引擎以此构建了一个Object运行的世界。

2.组件化开发的思想

虚幻引擎包含着组件化开发的思想，这样做使得功能划分更佳清晰，可以按组件分配开发任务；更好排查问题，某个组件出现问题，直接对组件进行处理；项目可维护性更强，提高开发效率。UactorCompoent就这么应运而生了。UactorCompoent作为功能的载体，具备一定程度的嵌套组装能力。

虚幻引擎有了组件那么必定有容器，在虚幻引擎中AActor充当容器，无论是实体还是移动，它的功能都是由Components拼装而成。它可以通过像静态网格组件这种方式通过图像来呈现。

为了方便使用，在Actor的基础上衍生出了Pawn。简单的说Pawn是一个Actor，但是可以使用PlayerController或AIController控制。例如Pawn可以是由玩家控制的一个人或者RTS游戏由AI在控制的单位。Pawn又衍生出Character，Character在Pawn之上增加MovementComponent，使得它可以使用导航和移动，还增加了SkeletalMeshComponent，这使它可以有更复杂更易于管理的表现形式。

对于游戏引擎而言，“显示”指的是游戏的UI，或是手柄上的输入和震动，是与玩家直接交互的载体；“数据”指的是Mesh，Material，Actor，Level等各种元素组织起来的内存数据表示；“算法”可以是各种渲染算法，物理模拟，AI寻路，。抽象这三个变化，并归纳关系，就是典型的MVC模式了。作为上帝的Controller就承担着“算法”的功能，Controller不能是一个Component，一是因为Component的层级太低，表达的是功能的概念而非逻辑；二是Component必须依附于Actor存在，而我们的Controller希望能独立存在。 Actor具有配置动态生成、输入事件响应、Tick、可继承、可容纳Component、可在世界里出现、可在网络间同步的能力，只需要再加个控制Pawn的能力就能达到“算法”的要求，所以Controller由Actor分化而来。

3. 编码规范

良好的编码规范在保证代码在高质量完成需求的同时具备良好的可读性、可维护性。如成员变量前要加下划线，首字母要大写；成员函数要首字母大写并且有意义，注意语态；系统头文件应用#include <xxx.h>；自定义同文件应用#include "xxx.h"；头文件对变量函数声明，CPP文件进行定义；局部变量小写，开头不加下划线等要点。

三.性能分析

1.背景和方法

我制作了一个可以机动、瞄准、射击开火、寻路、搜索敌人的坦克，在较小的场景中使用这样的几个低模坦克可以达到一百二十帧，当在一百平方公里贴上高精度卫片随机地点生成两百个高模坦克时只有不到二十帧。为了提高性能，我开始对性能进行分析，采用控制因素的方法,把多因素的问题变成多个单因素的问题,而只改变其中的某一个因素,从而研究这个因素对事物影响,分别加以研究,最后再综合解决。

2.性能测试工具

（1）stat unit 命令

首先要确定瓶颈是在CPU还是GPU，这需要在控制台上输入stat unit 命令。

如图所示



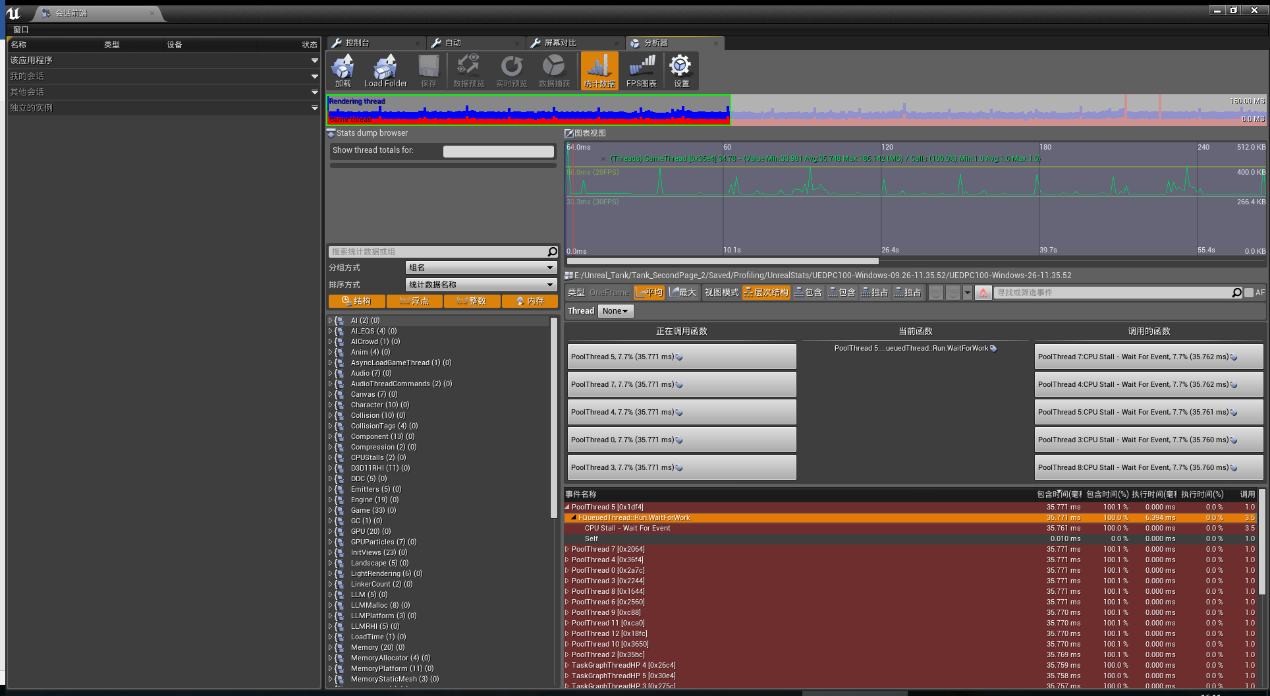
Frame时间是产生一帧花的总时间，由于逻辑线程（Game）和渲染线程（Draw）在一帧结束的时候需要同步，一帧花的时间经常跟其中的一个线程花的时间 相近。GPU时间测量了显卡渲染当前场景花的时间。由于 GPU时间是跟当前帧同步的，所以它跟一帧花的时间也基本差不多。

如果一帧花的时间跟逻辑线程的时间比较接近，那么瓶颈在逻辑线程，相反如果跟渲染线程的时间比较接近，那么瓶颈在渲染线程。如果两个时间 都不接近，但跟GPU时间比较接近，那么瓶颈在显卡上。

（2）如果在逻辑线程

通过~打开控制台里面输入"stat startfile"，让它运行一会至少10s来获取一个多帧的平均值。如果时长过长，那么生成的文件就会很大。通过stat stopfile来结束性能分析，一个后缀为ue4stats的文件会在工程的路径下产生。

这个时候你就可以使用UnrealFrontEnd（跟UE4Editor在同级目录）来打开分析的结果，或者在UE4Edtior里面通过window-->Developper ToolsàSession Frontend，打开后切换到Profiler面板，通过load来打开ue4stats文件，如图所示。



打开后你就可以自己来查看耗费时间的地方了，如果要查看卡顿，可以在时间线上查看高峰的地方，通过选择Maximum而不是Average，这样它就会显示一些峰值。

（3）stat engine



这个命令可以查看实时的渲染面数，这个属性对于CPU的计算很关键。

3.以“随机地点生成两百个可对打低模坦克”为控制因素

在一百平方公里贴上高精度卫片情况下，帧率达到60。

在一百平方公里贴上高精度卫片随机地点生成两百个可对打低模坦克时，帧率达到68左右。此时逻辑线程为18毫秒左右，渲染线程为9毫秒左右。此时的逻辑线程耗时大于渲染线程，最终帧数与逻辑线程耗时相近，根据短板效应可以主要考虑逻辑线程。使用性能分析工具发现此时的计算面为18万，基准图耗时2.7毫秒，动态原始绘图2.7毫秒，移动组件消耗了8.4毫秒，数据更新消耗3.6毫秒。由此可见，两百个低模坦克因网格体、特效和运行逻辑所带来的计算对性能影响还是十分明显的。

4.以“高/低模坦克”为控制因素

在一百平方公里贴上高精度卫片随机地点生成两百个可对打高模坦克时，帧率达到20左右。此时逻辑线程为42毫秒左右，渲染线程为14毫秒左右。此时的逻辑线程耗时大于渲染线程，最终帧数与逻辑线程耗时相近，根据短板效应可以主要考虑逻辑线程。使用Profiler性能分析工具发现此时的计算面为一亿三千面，基准图耗时4.7毫秒，动态原始绘图4.7毫秒，移动组件消耗了36毫秒，数据更新消耗22毫秒。

经过分析带来此问题的原因是因为一个高模大约一万多面，低模只有十几个面；从结构上比较，高模比低模多出两个履带，十八个轮子；高模有精细的UV贴图，低模只有简单的单色贴图。

5.提升性能的方法

为了提高性能，经过测试发现，减少屏幕上的UI可以提高帧数，具体提高多少帧应以UI处理前后的复杂程度而定；独立窗口播放要比编辑器视口播放提高7帧左右；将实体炮弹及时销毁可以大幅提高性能，将炮弹从之前的不销毁改为一秒后销毁，帧数上提高6帧左右；添加距离场修改物体将会大大减少面数，减少逻辑线程时间。注意用来观察数据的渲染视图也会产生较大的开销，大约为8毫秒左右，在考虑问题时应将此纳入其中。

优化后帧数提升到30帧左右，但实际上开销最大的还是运动组件方面，34毫秒的总帧时间，运动组件占到了27毫秒。通过性能分析工具对运行数据进行采集，然后逐帧分析，追踪到引擎中的代码，发现在这其中包含着物理模拟、组件碰撞、碰撞更新，这几个开销较大，目前还在尝试在此方面进行优化。

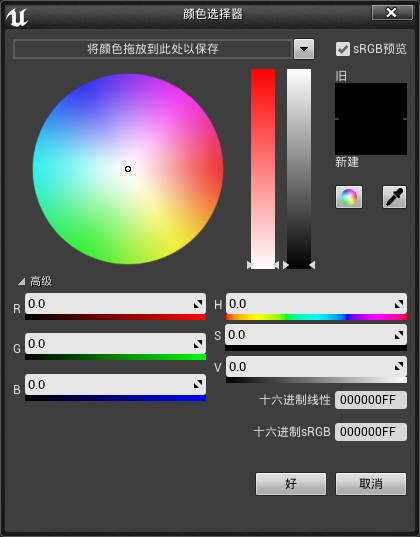
总的来说，我认为对于程序的优化应从以下几个方面入手：减少不必要的静态/动态网格体，减少每次绘制调用的元素，将光源设为不投射阴影，减少过多的顶点，合理使用光照减少光线复杂性，合理使用线框，减少着色器复杂性，优化逻辑减少运算。

四.材质浅析

1.HSV

HSV(Hue, Saturation, Value)是根据颜色的直观特性创建的一种颜色空间, 也称六角锥体模型,虚幻引擎使用的就是这种颜色表示方式。

这个模型中颜色的参数分别是：色调（H），饱和度（S），明度（V）。



(1)色调H

用角度度量，取值范围为0°～360°，从红色开始按逆时针方向计算，红色为0°，绿色为120°,蓝色为240°。它们的补色是：黄色为60°，青色为180°,品红为300°；

(2)饱和度S

饱和度S表示颜色接近光谱色的程度，可以看成是某种光谱色与白色混合的结果。（光谱色指的是从380～770nm[可见光](https://baike.baidu.com/item/%E5%8F%AF%E8%A7%81%E5%85%89)波长范围内的[单色光](https://baike.baidu.com/item/%E5%8D%95%E8%89%B2%E5%85%89)所对应的[颜色](https://baike.baidu.com/item/%E9%A2%9C%E8%89%B2)，[自然界](https://baike.baidu.com/item/%E8%87%AA%E7%84%B6%E7%95%8C)的[彩虹](https://baike.baidu.com/item/%E5%BD%A9%E8%99%B9)的颜色就是典型的光谱色。）其中光谱色所占的比例愈大，颜色接近光谱色的程度就愈高，颜色的饱和度也就愈高。饱和度高，颜色则深而艳。光谱色的白光成分为0，饱和度达到最高。通常取值范围为0%～100%，值越大，颜色越饱和。

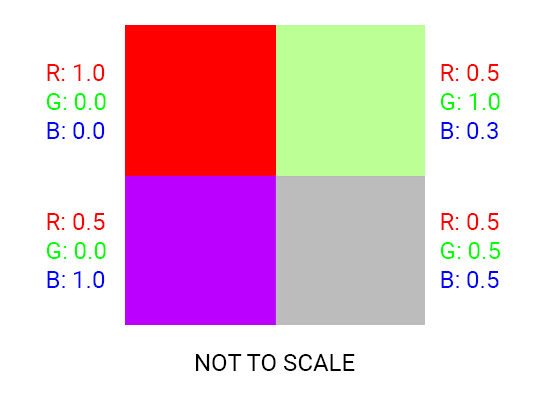
(3)明度V

明度表示颜色明亮的程度，对于光源色，明度值与发光体的光亮度有关；对于物体色，此值和物体的透射比或反射比有关。通常取值范围为0%（黑）到100%（白）。

2.Texture

Texture是一个图像，图像是像素的集合。在彩色图像中，像素的颜色由红色（R），绿色（G）和蓝色（B）通道决定。

以下是RGB值标记的2×2图像示例。

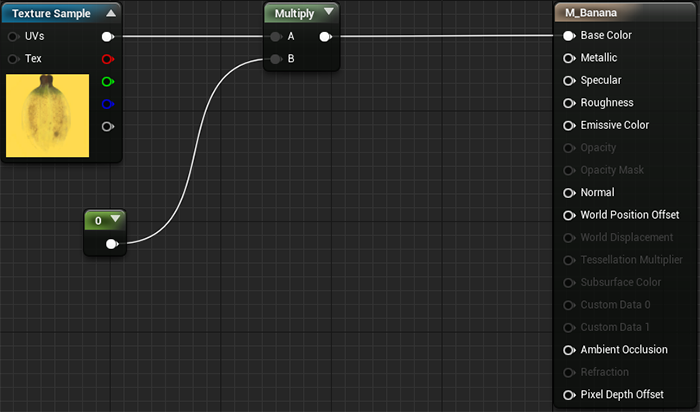


注意：在虚幻引擎中，RGB通道的范围为0.0到1.0。然而，在大多数其他应用中，RGB通道的范围是0到255.这些只是显示相同信息的不同方法，并不意味着虚幻引擎的颜色范围更小。

3.Multiply节点

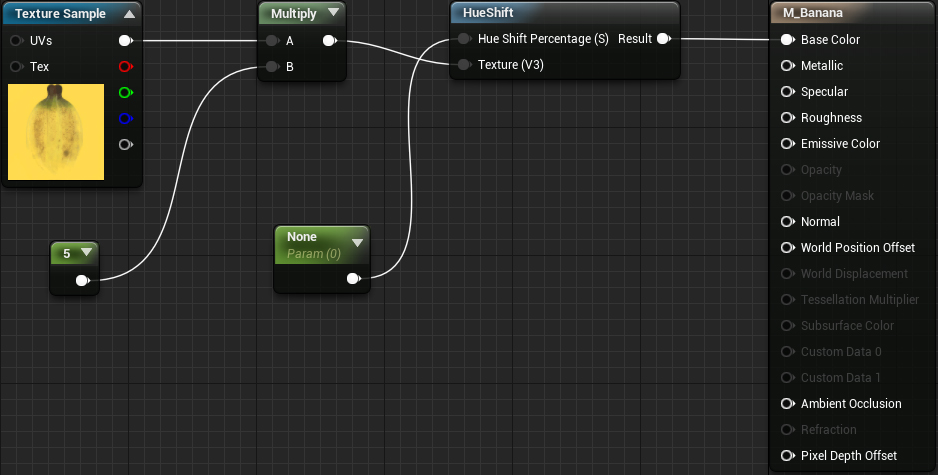
Multiply节点会将输入的数相乘，它可以更改像素的亮度，而不影响其色调或饱和度。

如下图所示。



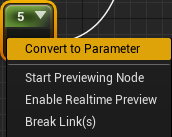
4.HueShift节点

HueShift节点可以改变纹理色调，也就是HSV中的H。



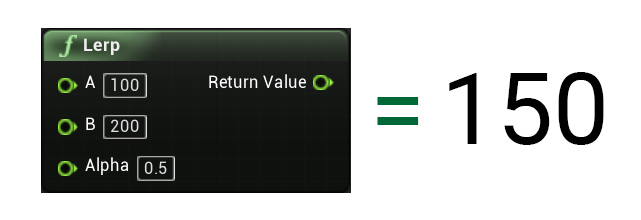
5.将节点参数在材质实例中暴露

在常量参数上右击，选择Convert to Parameter就可转换为标量参数。



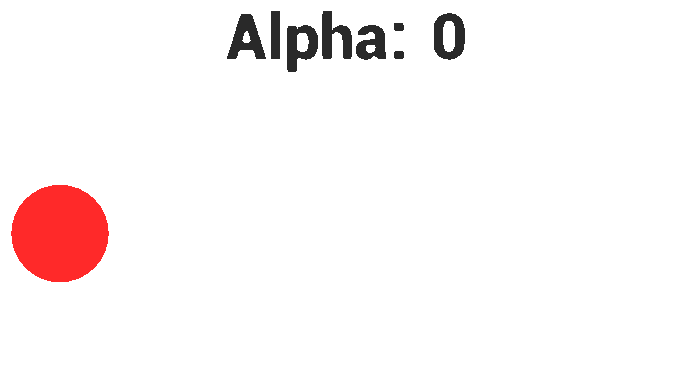
6.线性插值

线性插值是找到A和B之间的值的一种方式。例如，您可以使用线性插值来查找介于100到200之间的值。



Alpha视为A和B之间的百分比。Alpha为0将返回A，Alpha为1将返回B。

您可以随着时间的推移增加Alpha，从而平滑地将值从A点移动到B点。

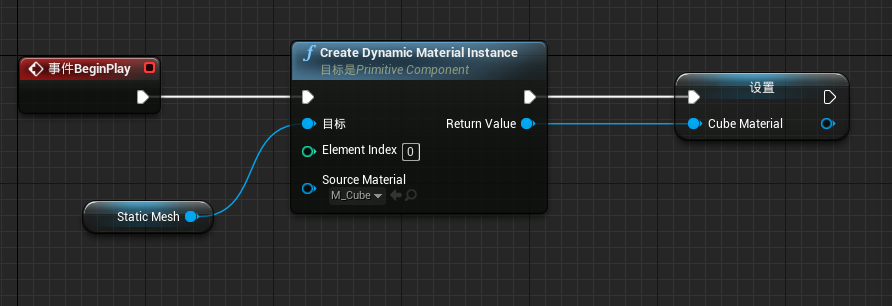


7.动态材质实例

常规材质实例不能在运行中改变数据，但可以在运行过程中动态编辑，可以使用Blueprints或C ++来执行此操作。

动态材质实例可以让你的材质更加炫酷，例如更改对象的不透明度以使其不可见。

动态材质实例不能预先创建，必须在运行中动态创建，需要给其指定使用的目标和从哪个材质创建而来，具体创建方法如图所示



创建完成后可以设置动态材质实例中的某个值，如图所示

