1. Shader需要的数据大体可以分为3类：
   1. 顶点流

渲染对象网格数据，如位置、法线、颜色等

* 1. 材质

与渲染对象有关的Shader常量，如底色、金属色、粗糙度等

* 1. 环境

与场景相关的Shader常量，如光源数据，屏幕可视范围等

1. Shader需要先定义一个包括基本渲染流程的Shader模板，再使用#if、#else、#include等预处理指令和预先定义的宏参数组合不同的功能，形成一个完整的Shader。

实现这一步需要定义一种诺亚专有的shader文件格式和该文件格式的解析器，通过解析并组合该文件动态生成hlsl文件，将不同的hlsl存入Cache文件中并组织为hash表，每次通过宏参数的组合进行hash来获取shader

1. C++代码与着色器代码进行参数传递的思路：

由于不同类型的效果可能需要不同的参数，所以在C++部分需要一定的动态性。而在着色器中，处理某种效果的流程是静态的（通过预处理指令组合）。所以在Shader中可以将某种效果需要的所有参数定义到一个结构体中，C++部分通过宏参数动态生成一个UniformList，Uniform中定义了参数的名称、类型和大小，通过UniformList生成一个UniformBuffer，再从渲染对象中获取相应的参数值存入UniformBuffer中，最后统一将UniformBuffer传递给Shader

（现在诺亚的做法是将uniform一个个单独传递，效率很低。）

UE4中影响常量参数表的因素：

材质的作用域 // 这个变量的使用场合不明，诺亚中应该用不到，可以忽略

材质的混合模式

材质的着色模型

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 不透明 | 半透明 | 叠加 | 乘积 | 蒙版 |
| BaseColor（纹理） |  |  |  |  |  |
| EmissiveColor（纹理） |  |  |  |  |  |
| Normal（纹理） |  |  |  |  |  |
| Metallic（纹理） |  |  |  |  |  |
| Specular（纹理） |  |  |  |  |  |
| Roughness（纹理） |  |  |  |  |  |
| TwoSided |  |  |  |  |  |
| Opacity（纹理） |  |  |  |  |  |
| OpacityMask（纹理） |  |  |  |  |  |
| OpacityMaskClipValue |  |  |  |  |  |

初步构思诺亚影响常量参数表的因素

材质的混合模式

材质的着色模型

材质的属性表达式（在着色模型的基础上将各个常量属性变为多样化的可变属性）

参数的具体传递流程：

1. 需要一张表定义各个材质属性所拥有的特殊输入表达式，以及每个表达式所需要的参数列表
2. 材质文件中定义了每个材质属性采用的表达式（或常量），根据表达式的组合拼接所需要的最终参数列表
3. shader根据材质的表达式组合生成相应的着色器代码
4. 从材质文件中获取相应的参数，并根据参数列表将参数值存入UniformBuffer中
5. 将UniformBuffer赋值给shader
6. 材质的属性定义

与Shader一样，材质也需要一个匹配固定渲染流程的Material模板，再通过它拓展出需要各种不同参数的Material，Material中需要定义组合Shader所需要的宏参数

1. 关于环境光

UE4采用的渲染模型是PBR，它定义了场景光照和材质如何相互作用。但是PBR采用了预烘焙的全局光照（辐射度 + SH + CubeMap），诺亚中的场景并不是3D的，实现的成本太高，所以依然需要沿用Phong模型的环境光以及漫反射模型。高光可以使用UE4的实现方式。