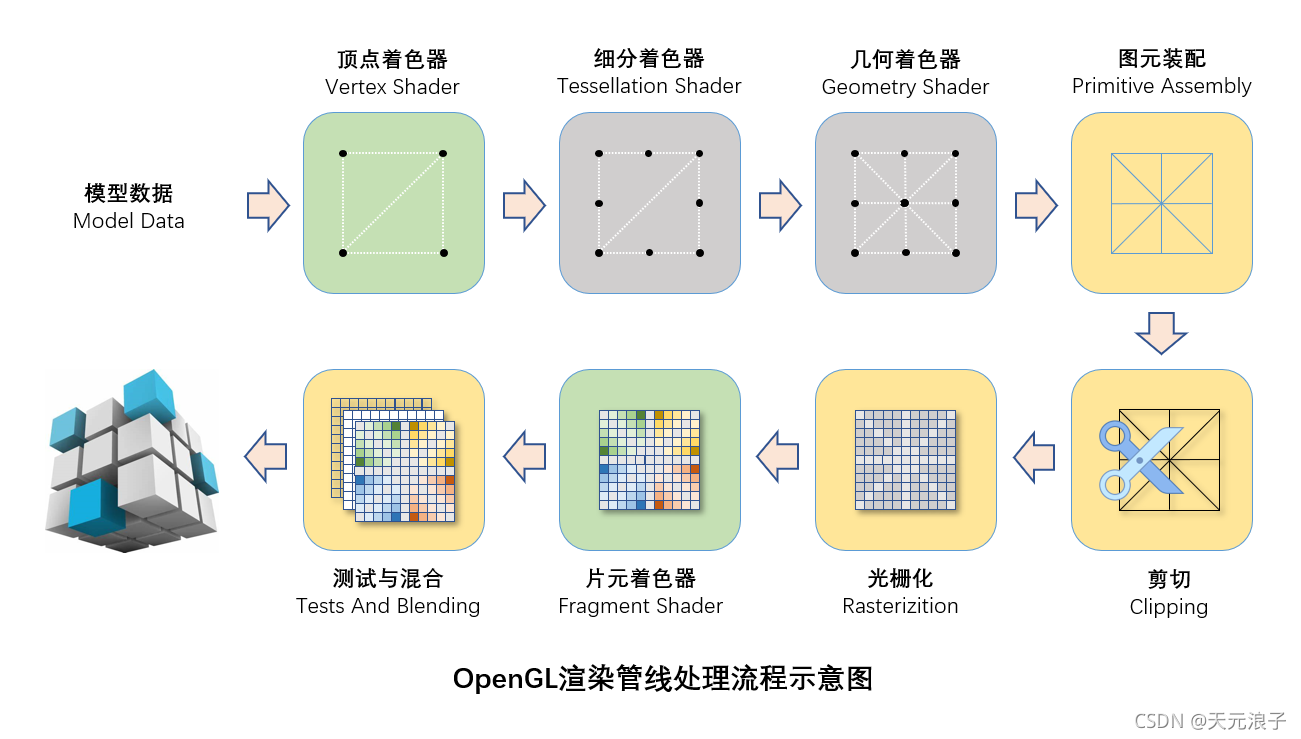
**OpenGL着色器语言GLSL资料汇编**

**文章目录**

* [1 渲染管线中的着色器类型](https://xufive.blog.csdn.net/article/details/121437277#1__2)
* [2 GLSL和OpenGL的版本对应关系](https://xufive.blog.csdn.net/article/details/121437277#2_GLSLOpenGL_21)
* [3 GLSL语法](https://xufive.blog.csdn.net/article/details/121437277#3_GLSL_49)
  + [3.1 数据类型](https://xufive.blog.csdn.net/article/details/121437277#31__50)
    - [3.1.1 基本数据类型](https://xufive.blog.csdn.net/article/details/121437277#311__53)
    - [3.1.2 向量](https://xufive.blog.csdn.net/article/details/121437277#312__64)
    - [3.1.3 矩阵](https://xufive.blog.csdn.net/article/details/121437277#313__85)
    - [3.1.4 结构体](https://xufive.blog.csdn.net/article/details/121437277#314__94)
    - [3.1.5 数组](https://xufive.blog.csdn.net/article/details/121437277#315__106)
  + [3.2 操作符和优先级](https://xufive.blog.csdn.net/article/details/121437277#32__114)
  + [3.3 流控制](https://xufive.blog.csdn.net/article/details/121437277#33__146)
    - [3.3.1 循环结构](https://xufive.blog.csdn.net/article/details/121437277#331__149)
    - [3.3.2 分支结构](https://xufive.blog.csdn.net/article/details/121437277#332__169)
  + [3.4 函数](https://xufive.blog.csdn.net/article/details/121437277#34__193)
* [4 着色器的组成](https://xufive.blog.csdn.net/article/details/121437277#4__203)
  + [4.1 预编译命令](https://xufive.blog.csdn.net/article/details/121437277#41__221)
  + [4.2 全局变量声明](https://xufive.blog.csdn.net/article/details/121437277#42__240)
  + [4.3 主函数](https://xufive.blog.csdn.net/article/details/121437277#43__250)
* [5 存储限定符](https://xufive.blog.csdn.net/article/details/121437277#5__253)
* [6 布局限定符](https://xufive.blog.csdn.net/article/details/121437277#6__270)
* [7 内置变量](https://xufive.blog.csdn.net/article/details/121437277#7__312)
  + [7.1 顶点着色器内置变量](https://xufive.blog.csdn.net/article/details/121437277#71__315)
  + [7.2 细分控制着色器内置变量](https://xufive.blog.csdn.net/article/details/121437277#72__324)
  + [7.3 细分估值着色器内置变量](https://xufive.blog.csdn.net/article/details/121437277#73__336)
  + [7.4 几何着色器内置变量](https://xufive.blog.csdn.net/article/details/121437277#74__348)
  + [7.5 片元着色器内置变量](https://xufive.blog.csdn.net/article/details/121437277#75__362)
  + [7.6 计算着色器内置变量](https://xufive.blog.csdn.net/article/details/121437277#76__378)
* [8 常用内置函数](https://xufive.blog.csdn.net/article/details/121437277#8__389)
  + [8.1 基础函数](https://xufive.blog.csdn.net/article/details/121437277#81__413)
  + [8.2 三角函数](https://xufive.blog.csdn.net/article/details/121437277#82__442)
  + [8.3 矩阵函数](https://xufive.blog.csdn.net/article/details/121437277#83__461)
  + [8.4 向量函数](https://xufive.blog.csdn.net/article/details/121437277#84__470)
  + [8.5 噪声函数](https://xufive.blog.csdn.net/article/details/121437277#85__491)

**1 渲染管线中的着色器类型**

OpenGL的工作流程通常被描述为渲染管线（rendering pipeline），该管线上的每一个环节都依赖相应的着色器（shader）来处理数据。在OpenGL 3.0版本（包括该版本）之前，用户可以使用固定功能管线（fixed-function pilpline）以避免编写着色器程序，但是从3.1版本开始，OpenGL从核心模式中移除了固定功能管线，使用着色器成了用户唯一的选择。

  
上图是根据我自己的理解绘制的渲染管线处理流程示意图。可以看出，整个管线上有四个环节使用四种不同的着色器，分别是顶点着色器、细分着色器、几何着色器和片元着色器，其中细分着色器又分为细分控制着色器和细分估值着色器。渲染管线中的细分着色器、几何着色器并非必需，使用它们是为了获得更好的效果。一般情况下，仅适用顶点着色器和片元着色器就可以生成模型。

除了上述四种着色器外，还有一种独立于渲染管线之外的着色器，叫做计算着色器，用于处理其他着色器程序所创建的和使用的缓存数据。

不同类型的着色器源码经过编译、连接后生成着色器程序，编译时需要为着色器源码指定着色器类型。下面是OpenGL为不同类型的着色器预定义的常量。

| **着色器** | **常量名** | **常量值** |
| --- | --- | --- |
| 顶点着色器 | GL\_VERTEX\_SHADER | 35633 |
| 细分控制着色器 | GL\_TESS\_CONTROL\_SHADER | 36488 |
| 细分估值着色器 | GL\_TESS\_EVALUATION\_SHADER | 36487 |
| 几何着色器 | GL\_GEOMETRY\_SHADER | 36313 |
| 片元着色器 | GL\_FRAGMENT\_SHADER | 35632 |
| 计算着色器 | GL\_COMPUTE\_SHADER | 37305 |

**2 GLSL和OpenGL的版本对应关系**

GLSL（OpenGL Shading Language）是为编写着色器程序而产生的语言，最初随OpenGL 2.0版本一同发布，并随着OpenGL的升级不断更新。下表展示了OpenGL和GLSL的版本对应关系。

| **OpenGL版本号** | **GLSL版本号** | **发布时间** |
| --- | --- | --- |
| 2.0 | 110 | 2004年9月7日 |
| 2.1 | 120 | 2006年7月2日 |
| 3.0 | 130 | 2008年8月11日 |
| 3.1 | 140 | 2009年3月24日 |
| 3.2 | 150 | 2009年8月3日 |
| 3.3 | 330 | 2010年3月11日 |
| 4.0 | 400 | 2010年3月11日 |
| 4.1 | 410 | 2010年7月26日 |
| 4.2 | 420 | 2011年8月8日 |
| 4.3 | 430 | 2012年8月6日 |
| 4.4 | 440 | 2013年7月23日 |
| 4.5 | 450 | 2014年7月 |
| 4.6 | 460 | 2017年7月 |

同样的，对于嵌入式设备来说，每个版本的OpenGL ES，也有与之相对应的GLSL ES版本。

| **OpenGL ES 版本号** | **GLSL ES版本号** |
| --- | --- |
| 2.0 | 100 |
| 3.0 | 300 |
| 3.1 | 310 |
| 3.2 | 320 |

**3 GLSL语法**

**3.1 数据类型**

GLSL的数据类型，大致可以分为基本数据类型、向量、矩阵、结构体和数组等五类。

**3.1.1 基本数据类型**

GLSL的基本数据类型有五种，分别是浮点型、双精度浮点型、整型、无符号整型和布尔型。

| **类型** | **说明** |
| --- | --- |
| float | IEEE32位浮点数 |
| double | IEEE64位浮点数 |
| int | 有符号32整数 |
| uint | 无符号32整数 |
| bool | 布尔型：true, false |

**3.1.2 向量**

五种基本数据类型都可以生成二元、三元和四元向量。

| **基本类型** | **二元向量** | **三元向量** | **四元向量** |
| --- | --- | --- | --- |
| float | vec2 | vec3 | vec4 |
| double | dvec2 | dvec3 | dvec4 |
| int | ivec2 | ivec3 | ivec4 |
| uint | uvec2 | uvec3 | uvec4 |
| bool | bvec2 | bvec3 | bvec4 |

以下关于向量类型的变量声明都是合法的。

vec3 p1 = vec3(1.0, 2.0, 3.0);

vec4 p2 = vec4(p1, 4.0);

ivec3 n = ivec3(p1);

vec4 white = vec4(1.0);

vec3 rgb= vec3(white);

**3.1.3 矩阵**

矩阵类型只有由float和double生成，矩阵阶数限定在2，3，4阶。以下关于矩阵类型的变量声明都是合法的。

mat3 m1 = mat3(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0);

mat3 m2 = mat2(vec3(1.0, 2.0, 3.0), vec3(4.0, 5.0, 6.0), vec3(7.0, 8.0, 9.0));

mat4 m3 = mat4(9.0) // 主对角线元素为9.0，其他元素全为0.0

**3.1.4 结构体**

和C语言类似，GLSL的结构体可以将不同类型的数据组合在一起，成为一个新的类型，并隐式定义一个新类型的构造函数。

struct particle {

float lifetime;

vec3 positon;

vec3 velocity;

};

particle p = particle(3.0, vec3(1.0, 2.0, 3.0), vec3(4.0, 5.0, 6.0));

**3.1.5 数组**

GLSL支持任意类型的数组，包括结构体数组。数组的length()方法返回数据元素的个数。数组不支持负数索引。

float p[3] = float[](1.0, 2.0, 3.0);

int ids[] = float[3](1, 2, 3);

**3.2 操作符和优先级**

下表列出了GLSL支持的操作符，以及各自的优先级。

| **优先级** | **操作符** | **操作对象类型** | **备注** |
| --- | --- | --- | --- |
| 1 | () | 无限制 | 可以嵌套 |
| 2 | [] | 数组、向量、矩阵 | 索引 |
| 2 | f() | 函数 | 函数调用 |
| 2 | . | 结构体 | 访问结构体的域变量或方法 |
| 2 | ++ – | 算术类型 | 后置递增/递减 |
| 3 | ++ – | 算术类型 | 前置递增/递减 |
| 3 | + - | 算术类型 | 一元操作符，正负号 |
| 3 | ~ | 整型 | 一元操作符，按位非 |
| 3 | ! | 布尔型 | 一元操作符，逻辑非 |
| 4 | \* / % | 算术类型 | 乘/除/模 |
| 5 | + - | 算术类型 | 加/减 |
| 6 | << >> | 整型 | 位操作，左移/右移 |
| 7 | < > <= >= | 算术类型 | 关系比较操作 |
| 8 | == != | 无限制 | 相等操作 |
| 9 | & | 整型 | 按位与 |
| 10 | ^ | 整型 | 按位异或 |
| 11 | | | 整型 | 按位或 |
| 12 | && | 布尔型 | 逻辑与 |
| 13 | ^^ | 布尔型 | 逻辑异或 |
| 14 | || | 布尔型 | 逻辑或 |
| 15 | a?b:c | 布尔型?任意:任意 | 三元操作符，分支结构 |
| 16 | = | 无限制 | 赋值 |
| 16 | += -+ \*= /= %= | 算术类型 | 算术赋值 |
| 16 | <<= >>= | 整型 | 位操作赋值 |
| 16 | &= ^= |= | 整型 | 位操作赋值 |
| 17 | , | 无限制 | 操作符序列 |

**3.3 流控制**

和大多数编程语言一样，GLSL也同样支持顺序结构、分支机构和循环结构，也支持break、continue、return等流控制语句。此外，GLSL还有一个专用的流控制语句discard，仅用于片元着色器中，意为丢弃当前片元。

**3.3.1 循环结构**

GLSL完全复制了C语言的循环结构，包括for循环、while循环和do…while循环。

// for循环

for (int i=0; i<100; i++) {

... ...

}

// while循环

while (n < 100) {

... ...

}

// do...while循环

do {

... ...

} while (n < 100)

**3.3.2 分支结构**

GLSL支持if…else结构和switch结构。

// if...else结构

if (condition) {

... ...

} else {

... ...

}

// switch结构

switch (int\_value) {

case n1:

... ...

break;

case n1:

... ...

break;

default:

... ...

}

**3.4 函数**

GLSL支持自定义函数，函数声明的语法和C语言类似，只是形参的类型前面需要加上参数限制符。如果函数类型是void，则函数可以无返回值。另外，函数命名也不能以gl\_作为前缀。下表是四种参数限制符的说明。

| **参数限制符** | **说明** |
| --- | --- |
| in | 将数据复制到函数中（默认） |
| const in | 将只读数据复制到函数中 |
| out | 从函数中获取数据 |
| inout | 将数据复制到函数中，并返回函数中修改的数据 |

**4 着色器的组成**

这是一个最简单的顶点着色器的例子，乍一看，和C语言程序的源码非常类似，连注释都是一样的。

// 1. 预编译命令

#version 330 core

// 2. 全局变量声明

in vec4 a\_Position;

// 3. 主函数

void main() {

gl\_Position = a\_Position;

}

一般而言，一个着色器程序的源码有三部分组成：预编译命令、全局变量声明和主函数。

**4.1 预编译命令**

和C语言的预编译类似，GLSL同样提供了一系列命令来定义宏和常量，或者有条件地生成编译代码块。上面的着色器例子中，#version 330 core就是声明GLSL版本的预编译命令。下表列出了常用的预编译命令。

| **命令** | **说明** |
| --- | --- |
| #define | 定义宏和常量 |
| #undef | 删除宏和常量 |
| #if | 条件编译 |
| #ifdef | 条件编译 |
| #ifndef | 条件编译 |
| #else | 条件编译 |
| #elif | 条件编译 |
| #endif | 条件编译 |
| #error text | 强制编译器将text输出到日志 |
| #pragma options | 设置编译器的特定选项 |
| #extension options | 设置GLSL扩展 |
| #version number | 设置GLSL版本 |
| #line options | 设置诊断行号 |

**4.2 全局变量声明**

GLSL属于强类型语言，所有变量都需要事先声明，并给出确定的数据类型。变量声明的一般格式如下。

数据类型 变量名 = 变量值;

尽管在主函数内部也可以声明变量，但主函数外的变量声明在数据类型前还需要一个存储限定符。比如上面的着色器例子中，a\_Position是预先声明的变量，其数据类型是vec4，数据类型前面的in就是存储限定符。存储限定符是GLSL语言中最具特色的语法，也是初学者最难理解的概念。关于存储限定符的更多说明，详见本文第5节。带存储限定符的变量声明的一般格式如下。

存储限定符 数据类型 变量名;

**4.3 主函数**

和C语言类似，主函数由多行语句组成，无返回值。主函数可以直接使用在主函数外都声明的全局变量，也可以在使用之前的任何时候声明局部变量，也可以使用着色器内置的输入变量（只读）和输出变量。不同类型的着色有着各自不同的内置变量，详见本文第6节。

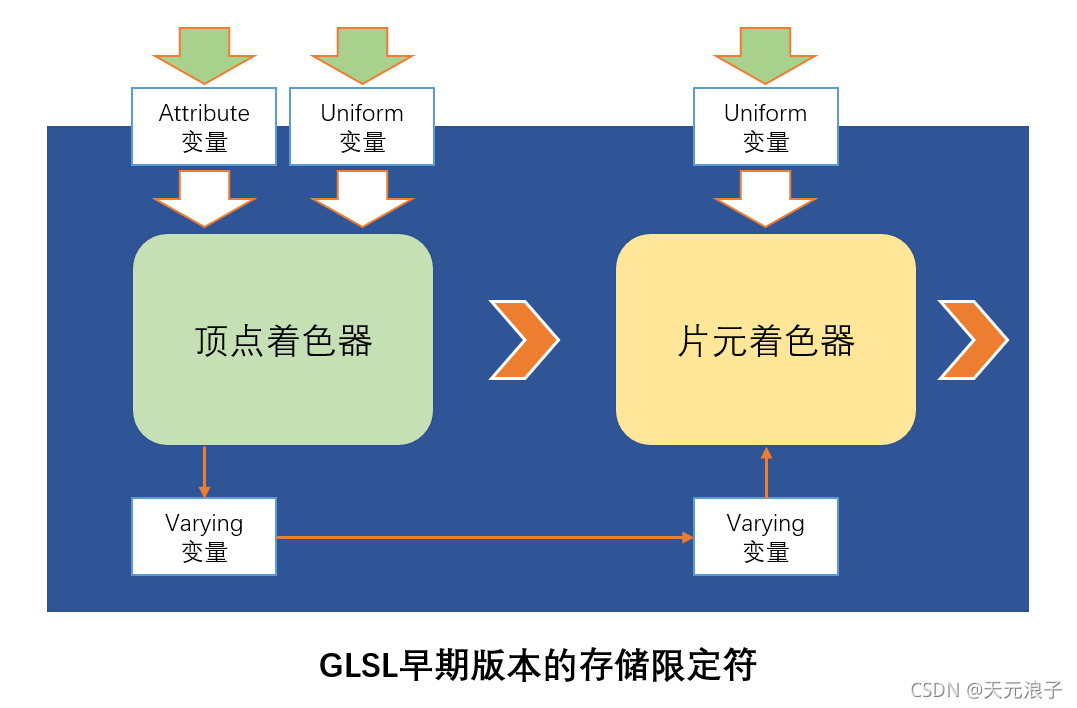
**5 存储限定符**

在OpenGL的渲染管线上，顶点数据不是批量处理的，而是一个顶点一个顶点逐一处理的。也就是说，顶点着色器程序每次只处理一个顶点。同样的，后续的着色器也是逐面片、逐图元、逐片元处理的。

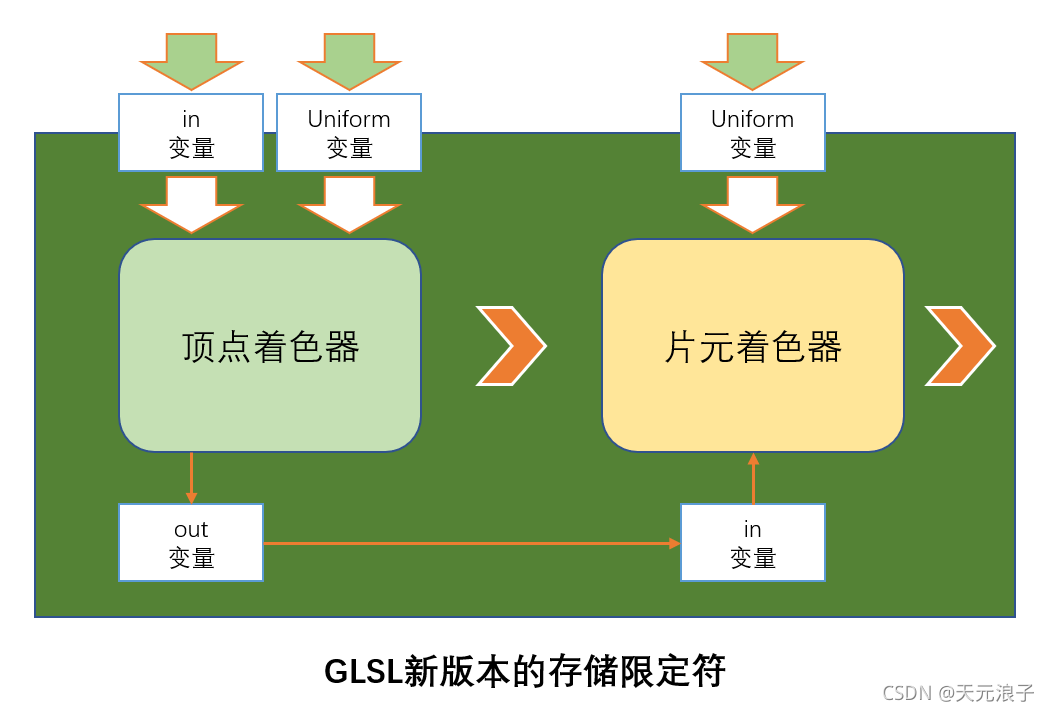
假如一个模型有1万个顶点，则顶点着色器程序需要运行1万次，每次输入的顶点数据（包括法向量、纹理坐标等）是各不相同的。但是也有在这1万运行中保持不变的数据，比如在该模型的此次渲染期间，投影矩阵、视点矩阵不会改变，大多数情况下，灯光参数也不会改变。

对于一个模型的一次渲染，在GLSL140版本之前，将顶点着色器程序每次运行时都会改变的数据称之为attribute，将每次运行保持不变的数据称之为uniform，将着色器之间顺序传递的数据称之为varying——这就是所谓的存储限定符。

我们可以向着色器程序传入attribute类型的数据和uniform类型的数据，但不能传入varying类型的数据。下图描述了attribute、uniform和varying的作用和用法。



也许是为了强化渲染管线数据的数据输入输出概念，自GLSL140版本开始，存储限定符attribute和varying不再被推荐使用，取而代之的是in和out。显然，in和out与attribute和varying并不是一一对应的关系。



除了一直不变的uniform，以及后来增补的in和out之外，还有一个存储限定符const，用来向着色器传递只读变量。

**6 布局限定符**

从GLSL330版本开始，GLSL增加了一个新的概念：布局管理。布局管理使用一个新的限定符layout，用来设置着色器输入/输出变量接口布局。下面的例子使用layout为in变量设置了接口位置，可以在渲染时直接使用这些接口而无需通过访问着色器程序获取它们。

#version 330 core

layout(location=0) in vec4 a\_Position;

layout(location=1) in vec4 a\_Normal;

layout(location=3) in vec4 a\_TexCoord;

如果变量是一个数组，则数组的每一个元素都会占据一个location的编号位置。下面的代码中，u\_Matrix是一个长度为3的数组，所以它将占据5、6、7三个位置。

layout(location=5) uniform mat4 u\_Matrix[3];

对于out变量，layout同样有效，且location的编号独立于in变量。

layout(location=0) out v\_color;

layout(location=1) out v\_TexCoord;

限定符layout的另一个用途是控制UBO（Uniform Buffer Object）数据块的布局，这需要另外几个布局限定符，见下表。

| **布局限定符** | **说明** |
| --- | --- |
| shared | 多程序共享uniform数据块（默认布局方式） |
| packed | uniform数据块占用最小内存空间，同时禁止多程序共享 |
| std140 | 使用标准布局方式设置uniform数据块和buffer块 |
| std430 | 使用标准布局方式设置buffer块 |
| row\_major | 以行主序方式存储uniform数据块中的矩阵 |
| column\_major | 以列主序方式存储uniform数据块中的矩阵 |

下面的代码，共享了一个uniform数据块，以行主序方式存储数据，多个限定符之间以逗号分隔。

layout(shared, row\_major) uniform ublock\_name {...};

需要特别说明的是，尽管在声明uniform数据块时已经提供了数据块的名字，但不能将数据块名视为uniform变量的父名称。这意味着不同的uniform数据块中如果存在相同的uniform变量名，就会造成编译错误。

**7 内置变量**

渲染管线各个环节之间依赖变量实现数据传递，类似于流水线作业：前一道工序的输出，就是后一道工序的输入。除了显式指定in或out属性的变量，渲染管线上各个着色器还有各自的内置变量，这些内置变量也同样具有in和out属性。

**7.1 顶点着色器内置变量**

| **属性** | **数据类型** | **变量名** | **说明** |
| --- | --- | --- | --- |
| in | int | gl\_VertexID | 当前顶点的索引序号 |
| in | int | gl\_InstanceID | 当前图元的实例数量 |
| out | vec4 | gl\_Position | 当前顶点的齐次坐标 |
| out | float | gl\_PointSize | 当前顶点栅格化时点的大小 |
| out | float | gl\_ClipDistance[] | 指定每个平面的裁剪距离 |

**7.2 细分控制着色器内置变量**

| **属性** | **数据类型** | **变量名** | **说明** |
| --- | --- | --- | --- |
| in/out | vec4 | gl\_Position | 当前顶点的齐次坐标 |
| in/out | float | gl\_PointSize | 当前顶点栅格化时点的大小 |
| in/out | float | gl\_ClipDistance[] | 指定每个平面的裁剪距离 |
| in | int | gl\_PatchVerticesIn | 当前面片的顶点数量 |
| in | int | gl\_PrimitiveID | 已处理图元数量 |
| in | int | gl\_InvocationID | 请求分配的面片顶点数量 |
| patch out | float | gl\_TessLevelOuter[4] | 设置面片外部细分层次 |
| patch out | float | gl\_TessLevelInner[2] | 设置面片内部细分层次 |

**7.3 细分估值着色器内置变量**

| **属性** | **数据类型** | **变量名** | **说明** |
| --- | --- | --- | --- |
| in/out | vec4 | gl\_Position | 当前顶点的齐次坐标 |
| in/out | float | gl\_PointSize | 当前顶点栅格化时点的大小 |
| in/out | floa t | gl\_ClipDistance[] | 指定每个平面的裁剪距离 |
| in | int | gl\_PatchVerticesIn | 当前面片的顶点数量 |
| in | int | gl\_PrimitiveID | 已处理图元数量 |
| in | vec3 | gl\_TessCoord | 3个分量代表3个顶点的权重 |
| patch in | float | gl\_TessLevelOuter[4] | 设置面片外部细分层次 |
| patch in | float | gl\_TessLevelInner[2] | 设置面片内部细分层次 |

**7.4 几何着色器内置变量**

| **属性** | **数据类型** | **变量名** | **说明** |
| --- | --- | --- | --- |
| in/out | vec4 | gl\_Position | 当前顶点的齐次坐标 |
| in/out | float | gl\_PointSize | 当前顶点栅格化时点的大小 |
| in/out | float | gl\_ClipDistance[] | 指定每个平面的裁剪距离 |
| in | int | gl\_PatchVerticesIn | 当前面片的顶点数量 |
| in | int | gl\_InvocationID | 请求分配的面片顶点数量 |
| patch out | float | gl\_TessLevelOuter[4] | 设置面片外部细分层次 |
| patch out | float | gl\_TessLevelInner[2] | 设置面片内部细分层次 |
| out | int | gl\_PrimitiveID | 图元标识符 |
| out | int | gl\_Layer | 设置面的层级 |
| out | int | gl\_ViewPortIndex | 绘制图元的视口索引 |

**7.5 片元着色器内置变量**

| **属性** | **数据类型** | **变量名** | **说明** |
| --- | --- | --- | --- |
| in | vec4 | gl\_FragCoore | 当前片元的窗口坐标 |
| in | bool | gl\_FrontFacing | 当前片元的前后面标志 |
| in | float | gl\_ClipDistance[] | 指定每个平面的裁剪距离 |
| in | vec2 | gl\_PointCoore | 当前片元在点图元中的位置 |
| in | int | gl\_PrimitiveID | 图元标识符 |
| in | int | gl\_SampleID | 当前处理的样本数量 |
| in | vec2 | gl\_SamplePosition | 在多样本缓冲中当前样本位置 |
| in | int | gl\_SampleMaskIn[] | 样本掩码 |
| in | int | gl\_Layer | 面的层级 |
| in | int | gl\_ViewPortIndex | 绘制图元的视口索引 |
| out | float | gl\_FragDepth | 设置面片外部细分层次 |
| out | int | gl\_SampleMask[] | 样本掩码 |

**7.6 计算着色器内置变量**

| **属性** | **数据类型** | **变量名** | **说明** |
| --- | --- | --- | --- |
| in | uvec3 | gl\_NumWorkGroups | 每个工作组全局工作项的总数 |
| const | uvec3 | gl\_WorkGroupSize | 局部工作组大小 |
| in | uvec3 | gl\_WorkGroupID | 当前调用所在的全局工作组三维索引 |
| in | uvec3 | gl\_GlobalInvotionID | 当前工作项的全局索引 |
| in | uvec3 | gl\_LocalInvotionID | 当前调用所在的全局工作组中局部工作组的索引 |
| in | uint | gl\_LocalInvotionIndex | gl\_LocalInvotionID的一维标识 |

**8 常用内置函数**

为了描述方便，对内置函数的参数类型和函数返回类型，特作约定如下表。

| **约定类型** | **覆盖的实际类型** |
| --- | --- |
| genType | float vec2 vec3 vec4 |
| genFType | float vec2 vec3 vec4 |
| genDType | double dvec2 dvec3 dvec4 |
| genIType | int ivec2 ivec3 ivec4 |
| genUType | uint uvec2 uvec3 uvec4 |
| genBType | bool bvec2 bvec3 bvec4 |
| vec | vec2 vec3 vec4 |
| dvec | dvec2 dvec3 vec4 |
| ivec | ivec2 ivec3 ivec4 |
| uvec | uvec2 uvec3 uvec4 |
| bvec | bvec2 bvec3 bvec4 |
| gvec | vec ivec uvec |
| mat | 全部单精度浮点型矩阵 |
| dmat | 全部双精度浮点型矩阵 |
| gsampler[…] | sampler[…] isampler[…] usampler[…] |
| gimage[…] | image[…] iimage[…] uimage[…] |

GLSL提供了数以百计的内置函数，门类齐全、功能之强大。本文将内置函数分为等基础函数、三角函数、矩阵函数、向量函数、噪声函数、纹理函数、图像函数、着色器函数和其他函数等九大类别，受限于个人能力，目前只完成了前五个类别，还有纹理函数、图像函数、着色器函数和其他函数等四类，以待后补。

**8.1 基础函数**

| **函数** | **说明** |
| --- | --- |
| genType pow (genType x, genType y) | 返回x的y次方 |
| genType exp (genType x) | 返回e的x次幂 |
| genType exp2 (genType x) | 返回2的x次幂 |
| genType log (genType x) | 返回x的自然对数（x>0） |
| genType log2 (genType x) | 返回以2为底x的对数（x>0） |
| genType sqrt (genType x) | 返回x的算术平方根 |
| genType inversesqrt (genType x) | 返回x的算术平方根的倒数 |
| genFDIType abs (genFDIType x) | 返回x的绝对值 |
| genFDIType sign (genFDIType x) | 若x>0返回1.0，若x<0返回-1.0，否则返回0 |
| genFDType floor (genFDType x) | 向下取整 |
| genFDType ceil (genFDType x) | 向上取整 |
| genFDType round(genFDType x) | 四舍五入取整 |
| genFDType roundEven(genFDType x) | 四舍五入取整（奇入偶舍，3.5和4.5舍入之后都是4.0） |
| genFDType fract (genFDType x) | 返回x-floor(x) |
| genFDType trunc (genFDType x) | 返回的整数与x同号，且绝对值不大于x的绝对值 |
| genFDType mod(genFDType x, (float|double) y) | 取模，返回x-y\*floor(x/y) |
| genFDType modf(genFDType x, out genFDType i) | 返回x的小数部分，整数部分赋予i |
| genFDIUType min (genFDIUType x, genFDIUType y) | 若y<x返回y，否则返回x |
| genFDIUType min (genFDIUType x, (int|uint|float|double) y) | 若y<x返回y，否则返回x |
| genFDIUType max (genFDIUType x, genFDIUType y) | 若y>x返回y，否则返回x |
| genFDIUType max (genFDIUType x, (int|uint|float|double) y) | 若y>x返回y，否则返回x |
| genFDType mix(genFDType x, genFDType x, genFDType a, ) | 返回线性插值结果，即x+(y-x)a |
| genFDType mix(genFDType x, genFDType x, (float|double) a, ) | 返回线性插值结果，即x+(y-x)a |
| genBtype isnan(genFDType x) | 判断x是否是NaN |
| genBtype isinf(genFDType x) | 判断x是否是无穷大 |

**8.2 三角函数**

| **函数** | **说明** |
| --- | --- |
| genType radians (genType degrees) | 度转弧度 |
| genType degrees (genType radians) | 弧度转度 |
| genType sin (genType angle) | 返回angle正弦函数值 |
| genType cos (genType angle) | 返回angle余弦函数值 |
| genType tan (genType angle) | 返回angle正切函数值 |
| genType asin (genType x) | 返回x的反正弦函数值 |
| genType acos (genType x) | 返回x的反余弦弦函数值 |
| genType atan (genType x) | 返回x的反正切函数值, 函数区间(− π / 2 , π / 2 -\pi/2,\pi/2−π/2,π/2) |
| genType atan (genType y, genType x) | 返回y/x的反正切函数值, 函数区间(− π , π -\pi,\pi−π,π) |
| genType sinh (genType x) | 返回x双曲线正弦函数值 |
| genType cosh (genType x) | 返回x双曲线余弦函数值 |
| genType tanh (genType x) | 返回x双曲线正切函数值 |
| genType asinh (genType x) | 返回x的反双曲线正弦函数值 |
| genType acosh (genType x) | 返回x的反双曲线余弦弦函数值 |
| genType atanh (genType x) | 返回x的反双曲线正切函数值) |

**8.3 矩阵函数**

| **函数** | **说明** |
| --- | --- |
| mat matrixCompMult (mat x, mat y) | 矩阵对应元素相乘，矩阵相乘则是重载乘法运算符（\*） |
| mat outerProduct(vecM c, vecN r) | 列向量c外乘行向量r，返回M行N列的矩阵 |
| mat transpose(mat m) | 返回矩阵m的转置矩阵，m保持不变 |
| float determinant(mat m) | 返回矩阵m的行列式 |
| mat inverse(mat m) | 返回矩阵m的逆矩阵，m保持不变 |

**8.4 向量函数**

| **函数** | **说明** |
| --- | --- |
| (float|double) length(genFDType x) | 返回向量的模（L2范式） |
| (float|double) distance(genFDType p0, genFDType p0) | 返回点p0和点p1之间的距离 |
| (float|double) dot(genFDType x, genFDType y) | 返回向量x和y的点积 |
| (vec3|dvec3) cross((vec3|dvec3) x, (vec3|dvec3) y) | 返回向量x和y的差积 |
| genFDType normallize(genFDType x) | 返回向量x的正则化向量（模长为1） |
| genFDType faceforward(genFDType N, genFDType I, genTypeFD Nref) | 如果dot(Nref, I)<0返回N，否则返回-N |
| genFDType reflect (genFDType I, genFDType N) | 返回反射向量，I为入射向量，N为法向量 |
| genFDType refract(genFDType I, genFDType N, (float|double) eta) | 返回折射向量，eta为折射率 |
| bvec lessThan(gvec x, gvec y) | 逐分量比较，返回x<y |
| bvec lessThanEqual(gvec x, gvec y) | 逐分量比较，返回x<=y |
| bvec greaterThan(gvec x, gvec y) | 逐分量比较，返回x>y |
| bvec greaterThanEqual(gvec x, gvec y) | 逐分量比较，返回x>=y |
| bvec equal((gvec|bvec) x, (gvec|bvec) y) | 逐分量比较，返回x==y |
| bvec notEqual((gvec|bvec) x, (gvec|bvec) y) | 逐分量比较，返回x!=y |
| bool any(bvec x) | 若x的分量存在true返回true，否则返回false |
| bool all(bvec x) | 若x的分量均为true返回true，否则返回false |
| bool not(bvec x) | bool向量取反 |

**8.5 噪声函数**

| **函数** | **说明** |
| --- | --- |
| float noise1(genType x) | 返回基于x的一维噪声 |
| float noise2(genType x) | 返回基于x的二维噪声 |
| float noise3(genType x) | 返回基于x的三维噪声 |
| float noise4(genType x) | 返回基于x的四维噪声 |