im18620660608

OpenGL笔记十四：GLSL语法

前言

期待您移步上篇：OpenGL笔记十三：GLSL加载纹理颠倒六种方案

概述

GLSL 全称 OpenGL Shading Language，是用来在 OpenGL 中着色编程的语言，即开发人员写的自定义程序代码。是执行在 GPU上的，代替了固定的渲染管线的一部分，使渲染管线中不同层次具有可编程性。
GLSL 其使用 C 语言作为基础高阶着色语言，避免了使用汇编语言或硬件规格语言的复杂性。
GLSL 的变量命名方式与 **C** 语言类似，可使用字母，数字以及下划线，不能以数字开头。还需要注意的是，变量名不能以 `gl_` 作为前缀，这个是 **GLSL** 保留的前缀，用于 **GLSL** 的内部变量。

数据类型

1.基本数据类型

类型	描述
void	表示空类型，作为函数的返回类型，表示这个函数不返回值。
bool	布尔类型，true or false，以及能产生布尔型的表达式。
int	有符号整型
uint	无符号整型
float	浮点型

2.纹理采样类型

类型	描述
sampler1D	用于内建的纹理函数中引用指定的1D纹理的句柄。只可以作为一致变量或者函数参数使用
sampler2D	二维纹理
sampler3D	三维纹理
samplerCube	盒纹理 cube mapped texture
sampler1DShadow	一维深度纹理
sampler2DShadow	二维深度纹理

3.向量类型

类型	描述
vec2, vec3, vec4	n维浮点数向量 n-component floating point vector
ivec2,ivec3,ivec4	n维整数向量 signed integer vector
uvec2,uvec3,uvec4	n维无符号整数向量 unsigned integer vector
bvec2,bvec3,bvec4	n维布尔向量 Boolean vector

- 向量声明-4分量float 类型向量

```
vec4 V1;
```

- 声明并对其构造初始化

```
vec4 V2 = vec4(1.0, 2.0, 3.0, 4.0);
```

- 向量运算

```
vec4 v1;
vec4 v2 = vec4(1.0,2.0,3.0,4.0);
vec4 v3 = vec4(1.0,2.0,3.0,4.0);
//注意：接下来假设所有参与运算的变量已定义并赋值。
v1 = v2 + v3;
v1 = v4;
v1 += vec4(1.0,1.0,1.0,1.0);
v1 = v1 * v2;
v1 *= 5.0;
```

- 向量元素的获取（成分选择）
  4分向量对应的意义可分为：{x,y,z,w}、{r,g,b,a} 、 {s,t,p,q} 。分别对应于 顶点坐标，颜色，纹理坐标。
  因此就可以单独获取、赋值某单一元素：

```
//通过x,y,z,w来获取向量中元素值
v1.x = 3.0f;
v1.xy = vec2(3.0f,4.0f);
v1.xyz = vec3(3,0f,4,0f,5.0f);

//通过颜色r,g,b,a
v1.r = 3.0f;
v1.rgba = vec4(1.0f,1.0f,1.0f,1.0f);

//纹理坐标s,t,p,q
v1.st = vec2(1.0f,0.0f);
```

不可以混合使用

```
v1.st = v2.xt;//这是不可以的，Error !!!
```

- 向量swizzle
  向量支持swizzle操作，2个及2个以上的向量元素交换。

```
v1.rgba = v2.bgra;
v2.bgra = v1.rgba;
v1.rgba = v2.brga;
```

- 向量赋值、一次性对所有分量操作

随笔档案

阅读排行榜

推荐排行榜

```
//赋值
v1.r = v2.b;
v1.g = v2.g;
v1.b = v2.r;
v1.a = v2.a;

//向量还支持一次性对所有分量操作
v1.x = v2.x +5.0f;
v1.y = v2.y +4.0f;
v1.z = v2.z +3.0f;

v1.xyz = v2.xyz + vec3(5.0f,4.0f,3.0f);
```

4.矩阵类型

类型	描述
mat2、mat2x2	2x2的浮点数矩阵类型
mat3、mat3x3	3x3的浮点数矩阵类型
mat4、mat4x4	4x4的浮点矩阵
mat2x3	2列3行的浮点矩阵(OpenGL的矩阵是列主顺序的)
mat2x4	2列4行的浮点矩阵
mat3x2	3列2行的浮点矩阵
mat3x4	3列4行的浮点矩阵
mat4x2	4列2行的浮点矩阵
mat4x3	4列3行的浮点矩阵

- 创建矩阵：

```
mat4 m1,m2,m3;
```

- 构造单元矩阵：

```
mat4 m4 = mat4(1.0f,0.0f,0.0f,0.0f
                    0.0f,1.0f,0.0f,0.0f,
                    0.0f,0.0f,1.0f,0.0f,
                    0.0f,0.0f,0.0f,1.0f);

//或者
mat4 m5 = mat4(1.0f);
```

5.结构体

结构体可以组合基本类型和数组来形成用户自定义的类型。在定义一个结构体的同时，你可以定义一个结构体实例，或者后面再定义。

```
struct fogStruct {
 vec4 color;
 float start;
 float end;
 vec3 points[3]; // 固定大小的数组是合法的
} fogVar;
```

通过 = 为结构体赋值。使用 ==，!= 来判断两个结构体是否相等。

```
fogVar = fogStruct(vec4(1.0,0.0,0.0,1.0),0.5,2.0);
vec4 color = fogVar.color;
float start = fogVar.start;
```

6.数组

```
float floatArray[4];
vec4 vecArray[2];
//注意
float a[4] = float[](1.0,2.0,3.0,4.0);
vec2 c[2] = vec2[2](vec2(1.0,2.0),vec2(3.0,4.0));
```

修饰符

*存储变量限定符

限定符	描述
none	(默认的可省略)本地变量，可读可写，外部不可见，函数的输入参数既是这种类型
const	常量值必须在声明时初始化，它是只读的不可修改的
attribute	表示只读的顶点数据，只能存在于vertex shader中,一般用于保存顶点或法线数据,它可以在数据缓冲区中读取数据。数据来自当前的顶点状态或者顶点数组。它必须是全局范围声明的，不能再函数内部。一个attribute可以是浮点数类型的标量，向量，或者矩阵。不可以是数组或则结构体。
varying	顶点着色器的输出，主要负责在 vertex 和 fragment 之间传递变量。例如颜色或者纹理坐标，（插值后的数据）作为片段着色器的只读输入数据。必须是全局范围声明的全局变量。可以是浮点数类型的标量，向量，矩阵。不能是数组或者结构体。
centorid varying	在没有多重采样的情况下，与varying是一样的意思。在多重采样时，centorid varying在光栅化的图形内部进行求值而不是在片段中心的固定位置求值。
uniform	一致变量。在着色器执行期间一致变量的值是不变的， 一般用来放置程序传递给shader的变换矩阵，材质，光照参数等等。与const常量不同的是，这个值在编译时期是未知的是由着色器外部初始化的。一致变量在顶点着色器和片段着色器之间是共享的。它也只能在全局范围进行声明。
invariant	(不变量)用于表示顶点着色器的输出和任何匹配片段着色器的输入，在不同的着色器中计算产生的值必须是一致的。所有的数据流和控制流，写入一个invariant变量的是一致的。编译器为了保证结果是完全一致的，需要放弃那些可能会导致不一致值的潜在的优化。除非必要，不要使用这个修饰符。在多通道渲染中避免z-fighting可能会使用到。

- 函数参数限定符
  GLSL 允许自定义函数，但参数默认是以值形式（in 限定符）传入的，也就是说任何变量在传入时都会被拷贝一份，若想以引用方式传参，需要增加函数参数限定符。

限定符	描述
< none: default >	默认使用 in 限定符
in	用在函数的参数中，表示这个参数是输入的，在函数中改变这个值，并不会影响对调用的函数产生副作用。（相当于C语言的传值），这个是函数参数默认的修饰符
out	用在函数的参数中，表示该参数是输出参数，值是会改变的。
inout	用在函数的参数，表示这个参数即是输入参数也是输出参数。

其中使用 inout 方式传递的参数便与其他 OOP 语言中的引用传递类似，参数可读写，函数内对参数的修改会影响到传入参数本身。

```
vec4 getPosition(out vec4 p){
    p = vec4(0.,0.,0.,1.);
    return v4;
}

void doubleSize(inout float size){
    size= size * 3.0  ;
}
```

GLSL 中的运算

- **逐分量运算**

> vec，mat 这些类型其实是由 float 复合而成的，当它们与float 运算时，其实就是在每一个分量上分别与 float 进行运算，这就是所谓的 逐分量运算。GLSL 里，大部分涉及 vec，mat 的运算都是逐分量运算，但也并不全是。下文中就会讲到特例。
> 逐分量运算 是线性的，这就是说 vec 与 float 的运算结果是还是 vec。

注意 GLSL 中没有隐式转换。
因此 int 与 vec，mat 之间是不可运算的，因为 vec 和 mat 中的每一个分量都是 float 类型的，无法与 int 进行逐分量计算。

```
vec3 a  = vec3(1.0, 2.0, 3.0);
mat3 m  = mat3(1.0);
float s = 10.0;

vec3 b  = s * a; // vec3(10.0, 20.0, 30.0)
vec3 c  = a * s; // vec3(10.0, 20.0, 30.0)
mat3 m2 = s * m; // = mat3(10.0)
mat3 m3 = m * s; // = mat3(10.0)
```

- **vec(向量) 与 vec(向量)运算**
  两向量间的运算首先要保证操作数的**阶数**都相同，否则不能计算。例如: vec3*vec2 和 vec4+vec3等等都是不行的。
  它们的计算方式是两操作数在同位置上的分量分别进行运算，其本质还是逐分量进行的，这和上面所说的 float 类型的逐分量运算可能有一点点差异，相同的是 vec 与 vec 运算结果还是 vec，且阶数不变。

- vec(向量) 与 mat(矩阵)
  要保证操作数的阶数相同(这里的矩阵是方阵)，且 vec 与 mat 间只存在乘法运算。

```
vec2 v = vec2(10., 20.);
mat2 m = mat2(1., 2.,  3., 4.);
vec2 w = m * v; // = vec2(1. * 10. + 3. * 20., 2. * 10. + 4. * 20.)

vec2 v = vec2(10., 20.);
mat2 m = mat2(1., 2.,  3., 4.);
vec2 w = v * m; // = vec2(1. * 10. + 2. * 20., 3. * 10. + 4. * 20.)
```

精度限定

GLSL 在进行光栅化着色的时候，会产生大量的浮点数运算，这些运算可能是当前设备所不能承受的，所以 GLSL 提供了 3 种浮点数精度： `highp` 、 `mediump` 、 `lowp` ，我们可以根据不同的设备来使用合适的精度。
在变量前面加上 highp、mediump、lowp 即可完成对该变量的精度声明：

```
lowp float color;
varying mediump vec2 Coord;
lowp ivec2 foo(lowp mat3);
highp mat4 m;
```

在顶点着色器(vertexment shader)里默认是 highp 精度的。
在片元着色器(fragment shader)中是没有默认精度的。可以统一：precision mediump float;设置精度，也可以单独对每一个变量各自设置精度。

控制语句

在语法上，GLSL 与 C 非常相似， 也有 if else、for、while、do while，使用 continue 跳入下一次循环，break 结束循环。

```
for (l = 0; l < numLights; l++) {
    if (!lightExists[l]);
        continue;
    color += light[l];
}

while (i < num) {
    sum += color[i];
    i++;
}

do {
    color += light[lightNum];
    lightNum--;
} while (lightNum > 0);
```

GLSL 还多了一种特殊的控制语句 discard，它会立即跳出片元着色器，并不在向下任何语句。也就不执行后面的片段着色操作，片段也不会写入帧缓冲区。注 **GLSL 函数中没有递归!**

```
    if (true) {
       discard;
    }
```

期待您移步下篇：OpenGL笔记十五：GLSL光照计算

链接：https://www.jianshu.com/p/328e07b88a9f

分类: qt / open - GLSL

imxiangzi
粉丝 - 3 关注 - 0

« 上一篇： OpenGL学习（十）-- 着色语言 GLSL 语法介绍
» 下一篇： GLSL基础语法介绍

posted @ 2023-03-12 08:46 imxiangzi 阅读(212) 评论(0) 收藏 举报

支持 Markdown

自动补全

提交评论　　退出　订阅评论　我的博客

[Ctrl+Enter快捷键提交]

【推荐】100%开源！大型工业跨平台软件C++源码提供，建模，组态！

【推荐】国内首个AI IDE，深度理解中文开发场景，立即下载体验Trae

【推荐】Flutter适配HarmonyOS 5知识地图，实战解析+高频避坑指南

【推荐】凌霞软件回馈社区，携手博客园推出1Panel与Halo联合会员

【推荐】轻量又高性能的 SSH 工具 IShell：AI 加持，快人一步

相关博文：

· OpenGL学习（十）-- 着色语言 GLSL 语法介绍

· GLSL基础语法介绍

· GLSL 学习

· GLSL 语法简介

· GLSL语言基础

阅读排行：

· C#/.NET/.NET Core优秀项目和框架2025年4月简报

· 如何把ASP.NET Core WebApi打造成Mcp Server

· 排行榜的5种实现方案！

· 在 .NET 中使用 Sqids 快速的为数字 ID 披上神秘短串，轻松隐藏敏感数字！

· 突破Excel百万数据导出瓶颈：全链路优化实战指南