GPU着色器脚本编程

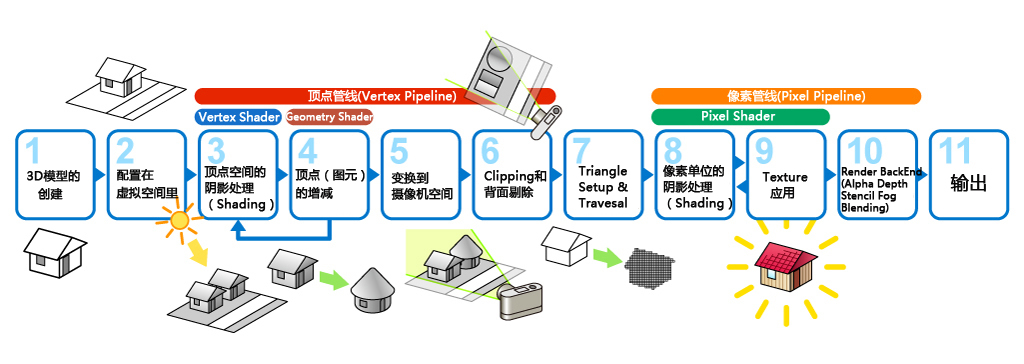
一、

1. 针对Ogre对GPU编程相关的封装和脚本的使用，从中总结出了些知识点供后面的研究和使用，这里是根据查阅相关资料和编写代码获取到的部分总结，可能存在不足之处。

这里主要讲解

1. 如何去调试着色器脚本程序。

二、渲染流程

[](http://bbs.lanlong.net/data/attachment/portal/201402/04/165032zh7fwfygtpsg34pm.jpg)

a.应用程序阶段：将几何体数据(顶点坐标/颜色/法向量/纹理坐标、纹理等)通过数据总线传送到图形硬件。

b.几何阶段：主要负责顶点坐标变换、光照、裁剪、投影以及屏幕映射，该阶段基于GPU进行运算，该阶段的末端得到了经过变换和投影之后的顶点坐标、颜色、以及纹理坐标。

c.光栅化阶段：基于几何阶段的输出数据，为像素正确配色，以便绘制完整图像，该阶段进行的都是单个像素的操作，每个像素的信息存储在颜色缓冲器(color buffer或者frame buffer)中。

值得注意的是，光照计算属于几何阶段，因为光照计算涉及视点、光源和物体的世界坐标，所以通常放在世界坐标系中进行计算；而雾化以及涉及物体透明度的计算属于光栅化阶段，因为上述两种计算都需要深度值信息(Z值)，而深度值是在几何阶段中计算，并传递到光栅阶段的。

模型顶点经过世界坐标变换、视图坐标变换、投影变换、光照计算、 图元组装、裁剪、屏幕映射。

在上图中，我们可以使用脚本进行干预的阶段在 Vertex Shader Geometry Shader Pixel Shader三个阶段。 Vertex Shader 是对输入的每个顶点进行变换和光照，Geometry Shader是对每个图元的操作(如点、线段、三角形)。Piexl Shader是对光栅化后的每个像素进行处理。

三、顶点、片段着色器

1、顶点着色器脚本 cg脚本

vertex\_program cgProjectTextureScene\_vs cg

{

//源程序文件

source ProjectTexture.cg

//函数入口

entry\_point RenderSceneVertMain

//编译器版本

profiles vs\_2\_0 arbvp1 vs\_2\_x vs\_3\_0 vp40

//参数

default\_params

{

//使用模型世界、相机视图矩阵

param\_named\_auto mWorldView worldview\_matrix

//使用当前脚本的相机的投影矩阵

param\_named\_auto mProj projection\_matrix

//使用当前脚本的相机的视图矩阵的逆，用于计算点的世界空间坐标 从主相机视图里面进行反算

param\_named\_auto invsView inverse\_view\_matrix

//RTT world\_matrix

//param\_named\_auto WorldMatrix0 custom 0

//param\_named\_auto WorldMatrix1 custom 1

//param\_named\_auto WorldMatrix2 custom 2

//param\_named\_auto WorldMatrix3 custom 3

//自定义参数，外面计算后传入

//RTT相机的视图矩阵 view matrix

param\_named\_auto viewMatrix0 custom 4

param\_named\_auto viewMatrix1 custom 5

param\_named\_auto viewMatrix2 custom 6

param\_named\_auto viewMatrix3 custom 7

//RTT相机的投影矩阵 project\_matrix

param\_named\_auto prjMatrix0 custom 8

param\_named\_auto prjMatrix1 custom 9

param\_named\_auto prjMatrix2 custom 10

param\_named\_auto prjMatrix3 custom 11

}

}

G3DGpuProgramParams.cpp中涉及到的参数对应含义表

如 world\_matrix 表示世界矩阵，用于计算模型的世界坐标，view\_matrix 表示视图矩阵，用于将世界坐标转换到相机的视图坐标系下，projection\_matrix 投影矩阵，将相机中模型视图坐标投影到近裁剪面上。如上面的参数param\_named\_auto mWorldView worldview\_matrix定义了一个世界视图矩阵类型的变量mWorldView，应用程序会将模型的世界矩阵、视图矩阵进行相乘变成世界视图矩阵传入到GPU中用于计算。param\_named\_auto viewMatrix0 custom 4 这里定义了一个自定义变量，序号为4，这里的参数是设置到渲染对象列表中,通过Renderable->setCustomParameter进行设置，序号也为当前变量设置的序号。这里的cgProjectTextureScene\_vs脚本直接被Ogre的资源管理器进行解析，可直接设置到通道Pass里用作顶点着色器（像素着色器同样处理）。根据上面的脚本，在C++层程序代码中用下面的方法进行设置参数：



custom 类型的参数需要将参数设置到Renderable参数列表里面。上面一段脚本定义了源文件名、函数名、编译器版本、参数列表。而真正执行的顶点程序是在指定的源文件中的RenderSceneVertMain。

渲染场景 顶点脚本 函数入口

void RenderSceneVertMain(

float4 iPos : POSITION, // GPU ？输入的 本地坐标下的坐标 POSITION 是一种坐标的语义

float3 iNormal : NORMAL, //法向量

float2 iTex : TEXCOORD0, //纹理坐标

out float4 oPos : POSITION, //顶点着色输出参数

out float2 Tex : TEXCOORD0, //输出纹理坐标

out float4 vPos : TEXCOORD1, //输出的 视图坐标下的坐标 在片段着色器里面，用统样的定义 float4 vPos : TEXCOORD1 去掉了 out 申明 ， 获取这里的输出

out float3 vNormal : TEXCOORD2, //输出的 法向量

out float4 vPosLight : TEXCOORD3, //输出的 点在RTT相机下的投影坐标

out float4 vPosInPrjView:TEXCOORD4,

uniform float4x4 mWorldView, //应用程序输入的参数 世界 \* 视图 矩阵

uniform float4x4 mProj, //应用程序输入的投影矩阵

uniform float4x4 invsView, //应用程序输入的视图矩阵的逆

uniform float4 viewMatrix0, //

uniform float4 viewMatrix1,

uniform float4 viewMatrix2,

uniform float4 viewMatrix3,

uniform float4 prjMatrix0,

uniform float4 prjMatrix1,

uniform float4 prjMatrix2,

uniform float4 prjMatrix3)

{

//

// 世界、视图矩阵，转换到相机的视图坐标系下

//

vPos = mul( mWorldView,iPos);

//

// 相机的投影矩阵，获取投影坐标

//

oPos = mul( mProj ,vPos);

//

// 顶点法向量 计算到视图坐标系下

//

vNormal = mul((float3x3)mWorldView,iNormal);

//

// 保存纹理坐标

//

Tex = iTex;

//

// Transform the position to light projection space, or the

// projection space as if the camera is looking out from

// the spotlight.

//

// 外面传入的 RTT 相机的视图矩阵、投影矩阵 用于计算当前点在RTT 相机下的世界坐标、投影坐标,

// 在片段里面获取计算得到纹理坐标和像素在RTT相机下的深度值

float4x4 viewMatrix = float4x4( viewMatrix0.x,viewMatrix0.y,viewMatrix0.z,viewMatrix0.w,

viewMatrix1.x,viewMatrix1.y,viewMatrix1.z,viewMatrix1.w,

viewMatrix2.x,viewMatrix2.y,viewMatrix2.z,viewMatrix2.w,

viewMatrix3.x,viewMatrix3.y,viewMatrix3.z,viewMatrix3.w);

float4x4 projMatrix = float4x4(prjMatrix0.x,prjMatrix0.y,prjMatrix0.z,prjMatrix0.w,

prjMatrix1.x,prjMatrix1.y,prjMatrix1.z,prjMatrix1.w,

prjMatrix2.x,prjMatrix2.y,prjMatrix2.z,prjMatrix2.w,

prjMatrix3.x,prjMatrix3.y,prjMatrix3.z,prjMatrix3.w);

//通过主相机反算得到 世界坐标下的世界坐标，因为没有传入世界矩阵同事iPos是本地坐标系的坐标，需要这样反算

//

float4 temWorld = mul(invsView , vPos);

//转到视图坐标下

vPosInPrjView = mul(viewMatrix ,temWorld);

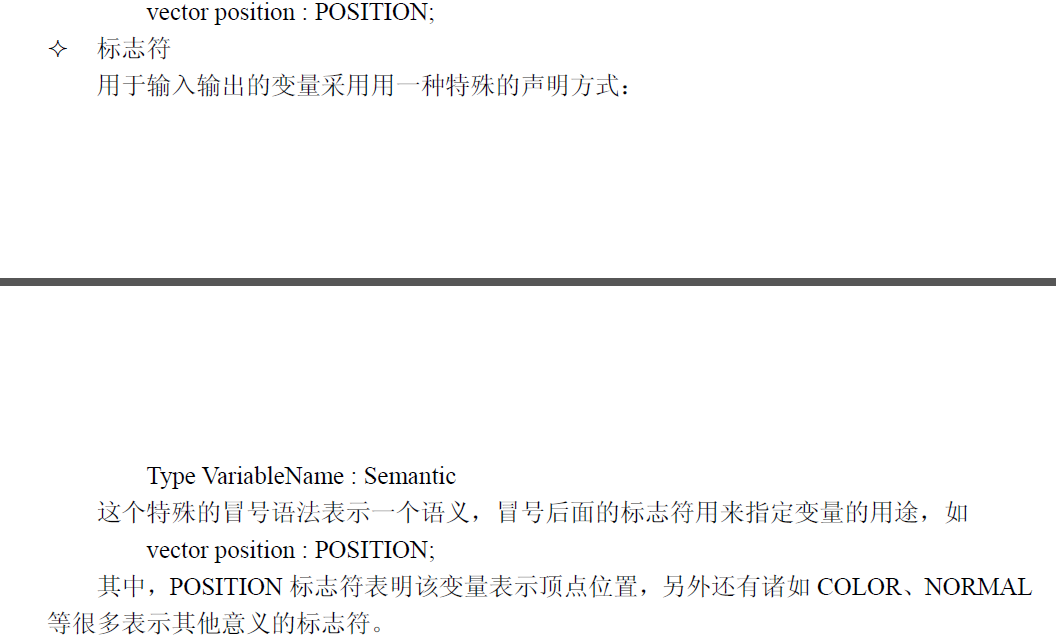
//转到投影空间里

vPosLight = mul(projMatrix ,vPosInPrjView);

//vPosLight = mul(mProj,vPosInPrjView);

}

其中对冒号“：”的使用，其他一些文章是如下解释的



2、输入/输出和语义绑定

float4 iPos : POSITION, // GPU ？输入的本地坐标下的坐标POSITION 是一种坐标的语义

float3 iNormal : NORMAL, //法向量

float2 iTex : TEXCOORD0, //纹理坐标

out float4 oPos : POSITION, //顶点着色输出参数

out float2 Tex : TEXCOORD0, //输出纹理坐标

out float4 vPos : TEXCOORD1, //输出的 视图坐标下的坐标 在片段着色器里面，用统样的定义 float4 vPos : TEXCOORD1 去掉了 out 申明 ， 获取这里的输出

out float3 vNormal : TEXCOORD2, //输出的 法向量

out float4 vPosLight : TEXCOORD3, //输出的 点在RTT相机下的投影坐标

out float4 vPosInPrjView:TEXCOORD4

uniform float4x4 mWorldView, //应用程序输入的参数 世界 \* 视图 矩阵

uniform float4x4 mProj, //应用程序输入的投影矩阵

uniform float4x4 invsView, //应用程序输入的视图矩阵的逆

uniform float4 viewMatrix0, //

uniform Cg 语言将输入数据流分为两类  
a. Varying inputs,即数据流输入图元信息的各种组成要素。 从应用程序输入到 GPU 的数据除了顶点位置数据，还有顶点的法向量数据，纹理坐标数据等。 Cg 语言提供了一组语义词，用以表明参数是由顶点的哪些数据初始化的。  
b. Uniform inputs，表示一些与三维渲染有关的离散信息数据，这些数据通常由应用程序传入，并通常不会随着图元信息的变化而变化，如材质对光的反射信息、运动矩阵等。 Uniform 修辞一个参数，表示该参数的值由外部应用程序初始化并传入；例如在参数列表中写：uniform float brightness,uniform float4x4 modleWorldProject 表示从“外部”传入一个 float 类型数据，和一个 4 阶矩阵。“外部”的含义通常是用 OpenGL 或者 DirectX 所编写的应用程序。使用 Uniform 修辞的变量， 除了数据来源不同外， 与其他变量是完全一样的。需要注意的一点是： uniform 修辞的变量的值是从外部传入的，所以在 Cg 程序（顶点程序和片段程序）中通常使用 uniform 参数修辞函数形参，不容许声明一个用 uniform 修辞的局部变量！否则编译时会出现错误提示信息

in / out / inout（输入/输出修饰符，用来修饰形参。输入/输出修辞符通常和语义词一起使用，表示顶点着色程序和片段着色程序的输入输出）

in: 修辞一个形参只是用于输入，进入函数体时被初始化，且该形参值的改变不会影响实参值，这是典型的值传递方式。

out: 修辞一个形参只是用于输出的，进入函数体时并没有被初始化，这种类型的形参一般是一个函数的运行结果。  
inout: 修辞一个形参既用于输入也用于输出，这是典型的引用传递。

void myFunction(out float x); //形参 x，只是用于输出

void myFunction(inout float x); //形参 x，即用于输入时初始化，也用于输出数据

void myFunction(in float x); //形参 x，只是用于输入

void myFunction(float x); /等价与 in float x，这种用法和 C\C++完全一致

语义词与语义绑定

语义词，表示输入图元的数据含义（是位置信息，还是法向量信息），也表明这些图元数据存放的硬件资源（寄存器或者纹理缓冲区）。顶点着色程序和片段着色程序中 Varying inputs 类型的输入，必须和一个语义词相绑定，这称之为绑定语义（ binding semantics）。

由于 Cg 语言并不支持指针机制，且图形硬件处理过程中，数据通常暂存在寄存器中，故而在 Cg 语言中，通过引入语义绑定（ binding semantics）机制，指定数据存放的位置，实际上就是将输入\输出数据和寄存器做一个映射关系（在 OpenGL Cg profiles 中是这样的，但在DirectX-based Cg profiles 中则并没有这种映射关系）。根据输入语义，图形处理器从某个寄存器取数据；然后再将处理好的数据，根据输出语义，放到指定的寄存器。

记住这一点：语义，是两个处理阶段（顶点程序、片段程序）之间的输入\输出数据和寄存器之间的桥梁，同时语义通常也表示数据的含义，如 POSITION，一般表示参数种存放的数据是顶点位置。

语义，分为输入语义和输出语义；输入语义和输出语义是有区别的。虽然一些参数经常会使用相同的绑定语义词， 例如： 顶点 Shader 的输入参数， POSITION指应用程序传入的顶点位置， 而输出参数使用 POSITION 语义就表示要反馈给硬件光栅器的裁剪空间位置，光栅器把 POSITION 当成一个位置信息。虽然两个语义都命名为 POSITION，但却对应着图形流水线上不同的寄存器。

顶点着色程序的输入语义：

语义词关键字：

POSITION (输入本地坐标系顶点，输出屏幕奇次坐标)

NORMAL   法向量

TANGENT 切线方向(指定了一个与纹理坐标方向一致的)

BINORMAL 法向和切线方向的叉乘方向

TEXCOORD0---TEXCOORD7 纹理坐标，也用于将数据从顶点着色器传入到片段着色器，TEXCOORD0 一般是纹理坐标

BLENDWEIGHT

PSIZE

BLENDINDICES

语义词 POSITION0 等价于 POSITION， 其他的语义词也有类似的等价关系。in float4 modelPos: POSITION    表示该参数中的数据是顶点位置坐标（通常位于模型空间），属于输入参数，语义词 POSITION 是输入语义，如果在 OpenGL 中则对应为接受应用程序传递的顶点数据的寄存器（图形硬件上）。  
  
in float4 modelNormal: NORMAL  表示该参数中的数据是顶点法向量坐标（通常位于模型空间），属于输入参数，语义词 NORMAL 是输入语义，如果在 OpenGL 中则对应为接受应用程序传递的顶点法向量的寄存器（图形硬件上）。

b.顶点着色程序的输出语义：

语义词关键字：

POSITION           PSIZE，

FOG                    COLOR0-COLOR1

TEXCOORD0-TEXCOORD7

顶点程序的输出数据被传入到片断程序中，所以顶点着色程序的输出语义词，通常也是片段程序的输入语义词，不过语义词POSITION除外。  
顶点着色程序必须声明一个输出变量，并绑定POSITION语义词，该变量中的数据将被用于，且只被用于光栅化！

为了保持顶点程序输出语义和片段程序输入语义的一致性， 通常使用相同的struct类型数据作为两者之间的传递，这是一种非常方便的写法，推荐使用。  
struct VertexScreen

{

float4 oPosition : POSITION;

float4 objectPos : TEXCOORD0;

float4 objectNormal : TEXCOORD1;

};

注意：当使用struct结构中的成员变量绑定语义时，需要注意到顶点着色程序中使用的POSITION语义词，是不会被片段程序所使用的。

如果需要从顶点着色程序向片段程序传递数据，例如顶点投影坐标、光照信息等，则可以声明另外的参数，绑定到TEXCOORD系列的语义词进行数据传递，实际上TEXCOORD系列的语义词通常都被用于从顶点程序向片段程序之间传递数据，当然，你也可以选择不使用struct结构，而直接在函数形参中进行语义绑定。无论使用何种方式，都要记住vertex program中的绑定语义（ POSITION除外）的输出形参中的数据会传递到fragment program中绑定相同语义的输入形参中。

c.片段着色程序的输出语义：

片段着色程序的输出语义词较少，通常是COLOR。这是因为片段着色程序运行完毕后，就基本到了GPU流水线的末端了。 片段程序必须声明一个out向量（三元或四元），绑定语义词COLOR，这个值将被用作该片断的最终颜色值。

语义绑定方法：

a.绑定语义放在函数的参数列表的参数声明后面中

void mian\_v(float4 position\_obj : POSITION,

float3 normal\_obj : NORMAL,

out float4 oPosition : POSITION,

out float4 oColor : COLOR,

uniform float4x4 modelViewProj){………………}

b.绑定语义可以放在结构体（ struct）的成员变量后面

struct Output {float4 position : POSITION;float3 color : COLOR;};

c.绑定语义词可以放在函数声明的后面

float4 main\_v( float4 position: POSITION,out float4 oposition : POSITION,uniform float4x4 modelViewProj):COLOR

{

oposition = mul(modelViewProj,position);

float4 ocolor = float4(1.0,0,0,0);

return ocolor;

}

通过下面的定义可以在Ogre里面申明GPU顶点着色器脚本，会被资源管理进行加载，应用程序可以直接设置材质的参数

pass->setVertexProgram("ProjectTextureSceneVS")

vertex\_program ProjectTextureSceneVS unified

{

delegate cgProjectTextureScene\_vs //cg

//delegate glProjectTextureScene\_vs //glsl

}

1. 片段着色器

片段着色脚本是在光栅化阶段完成后进行的，对输出到屏幕上的每个像素进行处理的程序脚本。

void RenderScenePixMain(

float2 Tex : TEXCOORD0,

float4 vPos : TEXCOORD1,

float3 vNormal : TEXCOORD2,

float4 vPosLight : TEXCOORD3,

float4 vPosInPrjView:TEXCOORD4,

float4 oColor:TEXCOORD5,

out float4 Color : COLOR,

uniform float4 vLightPos,

uniform float4 vLightDir,

uniform sampler2D sourceTex:TEXUNIT0,

uniform sampler2D samShadow:TEXUNIT1,

uniform sampler2D prjTexture:TEXUNIT2){…..}

上面的Tex是从顶点着色器传改过来的，其中用的TEXCOORD0语义进行传递的。vPos、vNormal、vPosLight、vPosInPrjView、oColor等用同样的方式进行传递。uniform vLightPos 、vLightDir是供应用程序传入。sampler2D定义纹理对象，标识为TEXUNIT0等语义，在Ogre C++程序里面创建纹理单元时，第一个纹理单元对应sourceTex，第二个纹理单元对应samShadow，第三个纹理单元对应prjTexture。