图书馆代码系列教程

题目：HLSL 着色器教程-基础语法篇

类型：高级代码

作者：南宫萧

图书馆编号：HC000003

所属组(选填)：

完成日期：2025 年 5 月 12

基于版本：1.16.5

联系方式（选填）：B051813@163.com

目录

1.基础语法：从数据到着色阶段

1.1 核心数据类型

标量：float（32 位浮点数，最常用）、half（16 位浮点数，节省显存）、int（32 位整数）、bool（布尔值）、uint（无符号整数）

示例：float brightness = 1.0;

向量：用类型名+维度表示，如 float2（二维浮点向量，等价于 float[2]）、int3（三维整型向量）、bool4（四维布尔向量）

示例：

```
float3 pos = float3(1.0, 2.0, 3.0);
float x = pos.x;       // 取 x 分量（1.0）
float4 color = pos.xyz1;  // 扩展为四维向量（1.0,2.0,3.0,1.0）
```

矩阵：类型名+行 x 列表示，如 float2x2（2x2 浮点矩阵）、float4x4（4x4 齐次变换矩阵）。

运算：矩阵乘法用 mul(matA, matB)，矩阵与向量相乘需注意顺序（mul(vec, mat)等价于行向量乘矩阵）。

资源类型（需配合 API 绑定）：

Texture2D/TextureCube：二维纹理 / 立方体纹理（用于采样颜色）。

SamplerState：采样器（控制纹理采样方式，如过滤模式、寻址

模式）。

1.2 变量修饰符

HLSL 通过修饰符控制变量的作用域和行为：

static：仅在当前文件内可见（类似 C 语言）。

const：常量（编译期确定值，不可修改）。

uniform：着色器常量（由 CPU 通过 API 传入，同一批次绘制中所有像素 / 顶点共享）。

示例：uniform float4x4 WorldViewProj;（模型 - 视图 - 投影矩阵，由 CPU 传入）。

1.3 函数与输入输出结构

HLSL 的函数需明确**输入输出参数**，且着色器（顶点 / 像素着色器）需定义输入输出结构，通过 ** **语义（Semantic）** ** 标记数据用途。

（1）语义（Semantic）

语义是 HLSL 的核心机制，用于告诉渲染管线“数据代表什么”（如顶点位置、纹理坐标、颜色等）。

POSITION：顶点位置（齐次坐标，通常为 float4）。

TEXCOORD0/TEXCOORD1：纹理坐标（float2/float3，支持多

组）。

COLOR0：顶点颜色（float4，RGBA 格式）。

NORMAL：顶点法线（float3，用于光照计算）。

示例：顶点着色器输入结构

```
struct VertexInput {
  float4 pos : POSITION;  // 顶点位置（语义标记）
  float2 uv : TEXCOORD0;  // 纹理坐标
};
```

（2）顶点着色器（Vertex Shader）

顶点着色器的职责是将输入的顶点数据（如模型空间坐标）变换到裁剪空间（齐次坐标），输出给光栅化阶段。

示例：基础顶点着色器

```
struct VertexOutput {
    float4 pos : SV_POSITION;  // SV_前缀表示系统值（裁剪空间位置）
    float2 uv : TEXCOORD0;     // 传递给像素着色器的纹理坐标
};

VertexOutput VS(VertexInput input) {
    VertexOutput output;
    // 用世界-视图-投影矩阵变换顶点位置
    output.pos = mul(input.pos, WorldViewProj);
    output.uv = input.uv;
    return output;
}
```

2.进阶语法：从光照到计算着色器

2.1 语义的深度应用

系统值语义（SV_前缀）:

SV_POSITION：裁剪空间顶点位置（顶点着色器必须输出）。

SV_TARGET：输出颜色（像素着色器必须输出）。

SV_VertexID：顶点 ID（几何着色器中用于访问顶点数据）。

自定义语义:

可自定义语义名（如 MY_UV），但需确保输入输出结构在管线中匹配。

示例：

```
struct MyVertexOutput {
    float3 worldPos : WORLD_POS;  // 自定义语义（世界空间位置）
    float3 normal : NORMAL;       // 顶点法线
};
```

2.2 内置函数

数学函数：dot(a,b)（点积）、cross(a,b)（叉积）、length(v)（向量长度）、normalize(v)（归一化）

示例：计算法线与光照方向的夹角

```
float ndotl = dot(normalize(input.normal), lightDir);
```

纹理采样函数：Sample(sampler, uv)（基础采样）、SampleLevel(sampler, uv, mipLevel)（指定 Mip 层级采样）、SampleGrad（自定义梯度采样，用于各向异性过滤）

几何变换函数：TransformObjectToWorld(pos)（模型空间转世界空间，需配合 SV_InstanceID 实例化）

2.3 几何着色器（Geometry Shader）

几何着色器在顶点着色器之后、光栅化之前执行，可动态生成或修改图元（如点、线、三角形）。

典型用途：粒子系统（将点扩展为四边形）、草丛生成（将线扩展为三角形）。

示例：将点扩展为四边形

```
[maxvertexcount(4)]  // 输出最多 4 个顶点（Untiy 引擎限制）
void GS(point VertexOutput input[1], inout TriangleStream<VertexOutput>
output) {
    float4 pos = input[0].pos;
    // 生成四边形的四个顶点（屏幕空间偏移）
    output.Append(pos + float4(-10, -10, 0, 0));
    output.Append(pos + float4(10, -10, 0, 0));
    output.Append(pos + float4(10, 10, 0, 0));
    output.Append(pos + float4(-10, 10, 0, 0));
}
```

2.4 计算着色器（Compute Shader）

计算着色器是 HLSL 的“通用并行计算”模块，不直接参与渲染管线，一般用于物理模拟（如流体、布料），全局光照（如光追中的 BVH 构建），图像后处理（如模糊、锐化）

核心概念：

线程组（Thread Group）：计算着色器的并行单位，由[numthreads(X,Y,Z)]定义每个组的线程数。

共享内存（groupshared）：线程组内线程共享的内存（速度远高于全局内存）。

示例：图像模糊计算着色器

```
Texture2D<float4> InputTex;
RWTexture2D<float4> OutputTex;  // 可读写纹理（需 UAV 绑定）

[numthreads(8,8,1)]  // 每个线程组 8x8=64 个线程
void CS (uint3 id : SV_DispatchThreadID) {  // 线程全局 ID（x,y,z）
    // 累加周围 9 个像素的平均值（简单模糊）
    float4 sum = 0;
    for(int dx=-1; dx<=1; dx++) {
        for(int dy=-1; dy<=1; dy++) {
            sum += InputTex.SampleLevel(samplerLinear, id.xy + float2(dx,
dy), 0);
        }
    }
    OutputTex[id.xy] = sum / 9;  // 写入输出纹理
}
```

2.5 光照模型实现

HLSL 的核心用途之一是实现光照模型（如 Phong、PBR）。

示例：PBR(基础物理渲染)

```
// 输入：顶点法线、视角方向、光照方向、材质参数（金属度、粗糙度）
float4 PBR_Lighting(float3 normal, float3 viewDir, float3 lightDir, float
metallic, float roughness) {
    // 计算半程向量（视角与光照的中间方向）
    float3 halfDir = normalize(viewDir + lightDir);

    // 法线分布函数（NDF）：GGX（粗糙度控制分布宽度）
    float a = roughness * roughness;
    float a2 = a * a;
    float ndoth = max(dot(normal, halfDir), 0.0);
    float denom = ndoth * ndoth * (a2 - 1.0) + 1.0;
    float NDF = a2 / (PI * denom * denom);

    // 菲涅尔项（Fresnel）：金属表面反射率随角度变化
    float F0 = lerp(0.04, 1.0, metallic);  // 非金属默认 0.04，金属 1.0
    float F = F0 + (1.0 - F0) * pow(1.0 - max(dot(halfDir, viewDir), 0.0),
5.0);

    // 最终光照颜色
    float ndotl = max(dot(normal, lightDir), 0.0);
    return lightColor * (NDF * F) * ndotl;  // 输出光照贡献
}
```

3. 与 Direct3D 的集成流程

编写 HLSL 代码：保存为.hlsl 文件（或内联字符串）。

编译着色器：使用 fxc（DirectX SDK 工具）或 dxc（最新的 HLSL

编译器）编译为字节码（.cso 文件）。

命令示例：dxc /T vs_6_0 /E VS main.hlsl -Fo vertexShader.cso（编译顶点着色器）。

API 绑定：通过 ID3D11Device::CreateVertexShader/CreatePixelShader 创建着色器对象，并绑定到渲染管线。

传递参数：通过 ID3D11DeviceContext::UpdateSubresource 更新 uniform 常量缓冲区（如 WorldViewProj 矩阵）。