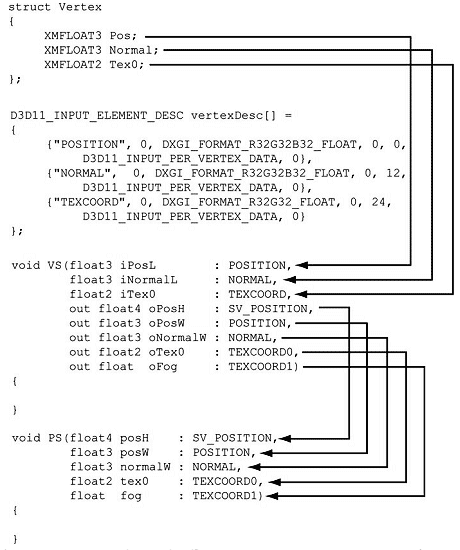
# 6.6 像素着色器示例

5.10.3节说过，由顶点着色器（或几何着色器）输出的顶点属性都已经过了插值处理。这些插值随后会作为像素着色器（pixel shader）的输入数据传入像素着色器。假设这里没有几何着色器，图6.5说明了目前顶点数据的流动过程。

****

**图6.5 D3D11\_INPUT\_ELEMENT\_DESC数组为每个顶点元素指定了一个关联语义，而顶点着色器的每个参数都有一个附加语义。这些语义描述了顶点元素和顶点着色器参数之间的对应关系。同样，顶点着色器的每个输出参数和像素着色器的每个输入参数也都有一个附加语义。这些语义用于将顶点着色器的输出参数映射为像素着色器的输入参数。**

与顶点着色器相似，像素着色器也是一个函数，只不过它处理的数据是像素片段（pixel fragment）。像素着色器的任务是为每个像素片段计算一个颜色值。请注意，像素和像素片段的含义不同，像素片段可能不会被存入后台缓冲区；例如，像素着色器可以对像素片段进行裁剪（在HLSL中的一个**clip**函数，它可以终止对一个像素片段的处理工作），或者当一个像素片段没有通过深度测试或模板测试时，它会被丢弃。所以，后台缓冲区中的一个像素可能会对应多个候选像素片段；这就是我们要区分“像素片段”和“像素”这两个术语的原因，虽然有时这两个术语会交替使用，但是读者应该明白它们在特定环境下的特定含义。

下面是一个简单的像素着色器，它与6.4节给出的顶点着色器对应。出于完整性的考虑，

我们将该顶点着色器再次列了出来。

cbuffer cbPerObject

{

float4x4 gWorldViewProj;

};

void VS(float3 iPosL : POSITION, float4 iColor : COLOR,

out float4 oPosH : SV\_POSITION,

out float4 oColor : COLOR)

{

// 转换到齐次剪裁空间

oPosH = mul(float4(iPosL, 1.0f), gWVP);

// 将顶点颜色直接传递到像素着色器

oColor = iColor;

}

float4 PS(float4 posH: SV\_POSITION,float4 color : COLOR) : SV\_TARGET

{

return color;

}

在本例中，像素着色器只是简单地返回插值颜色。注意，像素着色器的输入参数和顶点着色器的输出参数是对应的；这是一项规定。该像素着色器返回了一个4D颜色值。函数参数列表中的**SV\_TARGET**语义表示返回值与渲染目标视图的格式一致。

我们也可以使用输入/输出结构体的形式重写上面的顶点着色器和像素着色器代码。我们将语义附加到输入/输出结构的成员上，使用return语句用于输出代替输出参数。

cbuffer cbPerObject

{

float4x4 gWorldViewProj;

};

struct VertexIn

{

float3 PosL : POSITION;

float4 Color : COLOR;

};

struct VertexOut

{

float4 PosH : SV\_POSITION;

float4 Color : COLOR;

};

VertexOut VS(VertexIn vin)

{

VertexOut vout;

// 转换到齐次剪裁空间

vout.PosH = mul(float4(vin.PosL, 1.0f), gWorldViewProj);

// 将顶点颜色直接传递到像素着色器

vout.Color = vin.Color;

return vout;

}

float4 PS(VertexOut pin) : SV\_Target

{

return pin.Color;

}