# 6.1 顶点和顶点布局

5.5.1节已经讲过，在Direct3D中，顶点由空间位置和各种附加属性组成，Direct3D可以让我们灵活地建立属于我们自己的顶点格式；换句话说，它允许我们定义顶点的分量。要创建一个自定义的顶点格式，我们必须先创建一个包含顶点数据的结构体。例如，下面是两种不同类型的顶点格式；一个由位置和颜色组成，另一个由位置、法线和纹理坐标组成。

struct Vertex1

{

XMFLOAT3 Pos;

XMFLOAT4 Color;

};

struct Vertex2

{

XMFLOAT3 Pos;

XMFLOAT3 Normal;

XMFLOAT2 Tex0;

XMFLOAT2 Tex1;

};

在定义了顶点结构体之后，我们必须设法描述该顶点结构体的分量结构，使Direct3D知道该如何使用每个分量。这一描述信息是以输入布局（**ID3D11InputLayout**）的形式提供给Direct3D的。输入布局是一个**D3D11\_INPUT\_ELEMENT\_DESC**数组。**D3D11\_INPUT\_ELEMENT\_DESC**数组中的每个元素描述了顶点结构体的一个分量。比如，当顶点结构体包含两个分量时，对应的**D3D11\_INPUT\_ELEMENT\_DESC**数组会包含两个元素。我们将**D3D11\_INPUT\_ELEMENT\_DESC**称为输入布局描述（input layout description）。**D3D11\_INPUT\_ELEMENT\_DESC**结构体定义如下：

typedef struct D3D11\_INPUT\_ELEMENT\_DESC {

LPCSTR SemanticName;

UINT SemanticIndex;

DXGI\_FORMAT Format;

UINT InputSlot;

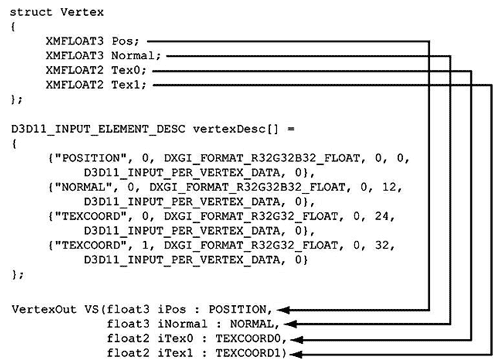
UINT AlignedByteOffset

D3D11\_INPUT\_CLASSIFICATION InputSlotClass;

UINT InstanceDataStepRate;

} D3D11\_INPUT\_ELEMENT\_DESC;

**1．SemanticName**：一个与元素相关的字符串。它可以是任何有效的语义名。语义（semantic）用于将顶点结构体中的元素映射为顶点着色器参数（参见图6.1）。

****

**图6.1 顶点结构体中的每个元素分别由D3D11\_INPUT\_ELEMENT\_DESC数组中的对应元素描述。语义名和语义索引提供了一种将顶点元素映射为顶点着色器参数的方法。**

**2．SemanticIndex**：附加在语义上的索引值。图6.1说明了使用该索引的原因；举例来说，当顶点结构体包含多组纹理坐标时，我们不是添加一个新的语义名，而是在语义名的后面加上一个索引值。在着色器代码中没有指定索引的语义默认索引为0，例如，在图6.1中的**POSITION**相当于**POSITION0**。

**3．Format**：一个用于指定元素格式的DXGI\_FORMAT枚举类型成员；下面是一些常用的格式：

* DXGI\_FORMAT\_R32\_FLOAT // 1D 32-bit float scalar
* DXGI\_FORMAT\_R32G32\_FLOAT // 2D 32-bit float vector
* DXGI\_FORMAT\_R32G32B32\_FLOAT // 3D 32-bit float vector
* DXGI\_FORMAT\_R32G32B32A32\_FLOAT // 4D 32-bit float vector
* DXGI\_FORMAT\_R8\_UINT // 1D 8-bit unsigned integer scalar
* DXGI\_FORMAT\_R16G16\_SINT // 2D 16-bit signed integer vector
* DXGI\_FORMAT\_R32G32B32\_UINT // 3D 32-bit unsigned integer vector
* DXGI\_FORMAT\_R8G8B8A8\_SINT // 4D 8-bit signed integer vector
* DXGI\_FORMAT\_R8G8B8A8\_UINT // 4D 8-bit unsigned integer vector

**4．InputSlot**：指定当前元素来自于哪个输入槽（input slot）。Direct3D支持16个输入槽（索引依次为 0到15），通过这些输入槽我们可以向着色器传入顶点数据。例如，当一个顶点由位置元素和颜色元素组成时，我们既可以使用一个输入槽传送两种元素，也可以将两种元素分开，使用第一个输入槽传送顶点元素，使用第二个输入槽传送颜色元素。Direct3D可以将来自于不同输入槽的元素重新组合为顶点。在本书中，我们只使用一个输入槽，但是在本章结尾的练习2中我们会引导读者做一个使用两个输入槽的练习。

**5．AlignedByteOffset**：对于单个输入槽来说，该参数表示从顶点结构体的起始位置到顶点元素的起始位置之间的字节偏移量。例如在下面的顶点结构体中，元素**Pos**的字节偏移量为0，因为它的起始位置与顶点结构体的起始位置相同；元素**Normal**的字节偏移量为12，因为必须跳过由**Pos**占用的字节才能到达**Normal**的起始位置；元素**Tex0**的字节偏移量为24，因为必须跳过由**Pos**和**Normal**占用的字节才能到达**Tex0**的起始位置；元素**Tex1**的字节偏移量为32，因为必须跳过由**Pos**，**Normal**和**Tex0**占用的字节才能到达**Tex1**的起始位置。

struct Vertex2

{

XMFLOAT3 Pos; // 0-byte offset

XMFLOAT3 Normal; // 12-byte offset

XMFLOAT2 Tex0; // 24-byte offset

XMFLOAT2 Tex1; // 32-byte offset

};

**6．InputSlotClass**：目前指定为**D3D11\_INPUT\_PER\_VERTEX\_DATA**；其他选项用于高级实例技术。

**7．InstanceDataStepRate**：目前指定为0；其他值只用于高级实例技术。

对于前面的两个示例顶点结构体Vertex1和Vertex2来说，对应的输入布局描述为：

D3D11\_INPUT\_ELEMENT\_DESC desc1[]=

{

{"POSITION", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 0,

D3D11\_INPUT\_PER\_VERTEX\_DATA, 0},

{"COLOR", 0, DXGI\_FORMAT\_R32G32B32A32\_FLOAT, 0, 12,

D3D11\_INPUT\_PER\_VERTEX\_DATA, 0}

};

D3D11\_INPUT\_ELEMENT\_DESC desc2[]=

{

{"POSITION", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 0,

D3D11\_INPUT\_PER\_VERTEX\_DATA, 0},

{"NORMAL", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT,0, 12,

D3D11\_INPUT\_PER\_VERTEX\_DATA, 0},

{"TEXCOORD", 0, DXGI\_FORMAT\_R32G32\_FLOAT, 0, 24,

D3D11\_INPUT\_PER\_VERTEX\_DATA, 0}

{"TEXCOORD", 1, DXGI\_FORMAT\_R32G32\_FLOAT, 0, 32,

D3D11\_INPUT\_PER\_VERTEX\_DATA, 0}

};

指定了输入布局描述之后，我们就可以使用**ID3D11Device::CreateInputLayout**方法获取一个表示输入布局的**ID3D11InputLayout**接口的指针：

HRESULT ID3D11Device::CreateInputLayout(

const D3D11\_INPUT\_ELEMENT\_DESC \*pInputElementDescs,

UINT NumElements,

const void \*pShaderBytecodeWithInputSignature,

SIZE\_T BytecodeLength,

ID3D11InputLayout \*\*ppInputLayout);

**1．pInputElementDescs**：一个用于描述顶点结构体的D3D11\_INPUT\_ELEMENT\_DESC数组。

**2．NumElements**：**D3D11\_INPUT\_ELEMENT\_DESC**数组的元素数量。

**3．pShaderBytecodeWithInputSignature**：指向顶点着色器参数的字节码的指针。

**4．BytecodeLength**：顶点着色器参数的字节码长度，单位为字节。

**5．ppInputLayout**：返回创建后的**ID3D11InputLayout**指针。

我们需要进一步解释一下第3个参数的含义。本质上，顶点着色器以一组顶点元素作为它的输入参数——也就是所谓的输入签名（input signature）。自定义顶点结构体中的元素必须被映射为与它们对应的顶点着色器参数，图6.1解释了这一问题。通过在创建输入布局时传入顶点着色器签名，Direct3D在创建时就可以验证输入布局是否与输入签名匹配，并建立从顶点结构体到着色器参数之间的映射关系。一个**ID3D11InputLayout**对象可以在多个参数完全相同的着色器中重复使用。

假设有下列输入参数和顶点结构：

VertexOut VS(float3 Pos:POSITION, float4 Color:COLOR,

float3 Normal: NORMAL){ }

struct Vertex

{

XMFLOAT3 Pos ;

XMFLOAT4 Color;

};

这样会产生错误，VC++的调试输出窗口会显示以下信息：

D3D11:ERROR:ID3D11Device::CreateInputLayout:The provided input

signature expects to read an element with SemanticName/Index:

'NORMAL'/0, but the declaration doesn't provide a matching name.

假如顶点结构和输入参数与输入元素匹配，但类型不同：

VertexOut VS(int3 Pos:POSITION, float4 Color:COLOR) { }

struct Vertex

{

XMFLOAT3 Pos;

XMFLOAT4 Color;

} ;

这样做是可行的，因为Direct3D允许输入寄存器中的字节被重新解释。但是，VC ++调试输出窗口会显示以下信息：

D3D11:WARNING:ID3D11Device::CreateInputLayout:The provided input

signature expects to read an element with SemanticName/Index:

'POSITION'/0 and component(s) of the type 'int32'. However,the

matching entry in the InputLayout declaration, element[0],

specifies mismatched format:'R32G32B32\_FLOAT'.This is not an error,

since behavior is well defined :The element format determines what

data conversion algorithm gets applied before it shows up in a

shader register.Independently,the shader input signature defines

how the shader will interpret the data that has been placed in its

input registers,with no change in the bits stored.It is valid for

the application to reinterpret data as a different type once it is

in the vertex shader,so this warning is issued just in case reinte rpretation was not intended by the author.

下面的代码说明了该如何调用**ID3D11Device::CreateInputLayout**方法。注意，这些代码涉及了一些我们还未讨论的内容（比如**ID3D11Effect**）。本质上，一个effect可以封装一个或多个pass，而每个pass都会与一个顶点着色器相连。所以，我们可以从effect中得到有关pass的描述信息（**D3D11\_PASS\_DESC**），然后再从中得到顶点着色器的输入签名。

ID3D11Effect\* mFX;

ID3D11EffectTechnique\* mTech;

ID3D11InputLayout\* mVertexLayout;

/\* ...create the effect... \*/

mTech = mFX->GetTechniqueByName("Tech");

D3D11\_PASS\_DESC PassDesc;

mTech->GetPassByIndex(0)->GetDesc(&PassDesc);

HR(md3dDevice->CreateInputLayout(vertexDesc, 4,

PassDesc.pIAInputSignature, PassDesc.IAInputSignatureSize,

&mVertexLayout));

创建了输入布局对象之后，它不会自动绑定到设备上。我们必须调用下面的语句来实现绑定：

ID3D11InputLayout\* mVertexLayout;

/\* ...create the input layout... \*/

md3dImmediateContext->IASetInputLayout(mVertexLayout);

如果你打算用一个输入布局来绘制一些物体，然后再使用另一个的布局来绘制另一些物体，那你必须按照下面的形式来组织代码：

md3dImmediateContext->IASetInputLayout(mVertexLayout1);

/\* ...draw objects using input layout 1... \*/

md3dImmediateContext->IASetInputLayout(mVertexLayout2);

/\* ...draw objects using input layout 2... \*/

换句话说，当一个**ID3D11InputLayout**对象被绑定到设备上时，如果不去改变它，那么它会始终驻留在那里。