# 6.2 顶点缓冲

为了让GPU访问顶点数组，我们必须把它放置在一个称为缓冲（buffer）的特殊资源容器中，该容器由**ID3D11Buffer**接口表示。

用于存储顶点的缓冲区称为顶点缓冲（vertex buffer）。Direct3D缓冲不仅可以存储数据，而且还说明了如何访问数据以及数据被绑定到图形管线的那个阶段。要创建一个顶点缓冲，我们必须执行以下步骤：

1．填写一个**D3D11\_BUFFER\_DESC**结构体，描述我们所要创建的缓冲区。

2．填写一个**D3D11\_SUBRESOURCE\_DATA**结构体，为缓冲区指定初始化数据。

3．调用**ID3D11Device::CreateBuffer**方法来创建缓冲区。

**D3D11\_BUFFER\_DESC**结构体的定义如下：

typedef struct D3D11\_BUFFER\_DESC{

UINT ByteWidth;

D3D11\_USAGE Usage;

UINT BindFlags;

UINT CPUAccessFlags;

UINT MiscFlags;

UINT StructureByteStride;

} D3D11\_BUFFER\_DESC;

**1．ByteWidth**：我们将要创建的顶点缓冲区的大小，单位为字节。

**2．Usage**：一个用于指定缓冲区用途的**D3D11\_USAGE**枚举类型成员。有4个可选值：

（a）**D3D10\_USAGE\_DEFAULT**：表示GPU会对资源执行读写操作。在使用映射API（例如**ID3D11DeviceContext::Map**）时，CPU在使用映射API时不能读写这种资源，但它能使用**ID3D11DeviceContext::UpdateSubresource**。**ID3D11DeviceContext::Map**方法会在6.14节中介绍。

（b）**D3D11\_USAGE\_IMMUTABLE**：表示在创建资源后，资源中的内容不会改变。这样可以获得一些内部优化，因为GPU会以只读方式访问这种资源。除了在创建资源时CPU会写入初始化数据外，其他任何时候CPU都不会对这种资源执行任何读写操作，我们也无法映射或更新一个immutable资源。

（c）**D3D11\_USAGE\_DYNAMIC**：表示应用程序（CPU）会频繁更新资源中的数据内容（例如，每帧更新一次）。GPU可以从这种资源中读取数据，使用映射API（**ID3D11DeviceContext::Map**）时，CPU可以向这种资源中写入数据。因为新的数据要从CPU内存（即系统RAM）传送到GPU内存（即显存），所以从CPU动态地更新GPU资源会有性能损失；若非必须，请勿使用**D3D11\_USAGE\_DYNAMIC**。

（d）**D3D11\_USAGE\_STAGING**：表示应用程序（CPU）会读取该资源的一个副本（即，该资源支持从显存到系统内存的数据复制操作）。显存到系统内存的复制是一个缓慢的操作，应尽量避免。使用**ID3D11DeviceContext::CopyResource**和**ID3D11DeviceContext::CopySubresourceRegion**方法可以复制资源，在12.3.5节会介绍一个复制资源的例子。

**3．BindFlags**：对于顶点缓冲区，该参数应设为**D3D11\_BIND\_VERTEX\_BUFFER**。

**4．CPUAccessFlags**：指定CPU对资源的访问权限。设置为0则表示CPU无需读写缓冲。如果CPU需要向资源写入数据，则应指定**D3D11\_CPU\_ACCESS\_WRITE**。具有写访问权限的资源的Usage参数应设为**D3D11\_USAGE\_DYNAMIC**或**D3D11\_USAGE\_STAGING**。如果CPU需要从资源读取数据，则应指定**D3D11\_CPU\_ACCESS\_READ**。具有读访问权限的资源的Usage参数应设为**D3D11\_USAGE\_STAGING**。当指定这些标志值时，应按需而定。通常，CPU从Direct3D资源读取数据的速度较慢。CPU向资源写入数据的速度虽然较快，但是把内存副本传回显存的过程仍很耗时。所以，最好的做法是（如果可能的话）不指定任何标志值，让资源驻留在显存中，只用GPU来读写数据。

**5．MiscFlags**：我们不需要为顶点缓冲区指定任何杂项（miscellaneous）标志值，所以该参数设为0。有关**D3D11\_RESOURCE\_MISC\_FLAG**枚举类型的详情请参阅SDK文档。

**6．StructureByteStride**：存储在结构化缓冲中的一个元素的大小，以字节为单位。这个属性只用于结构化缓冲，其他缓冲可以设置为0。所谓结构化缓冲，是指存储其中的元素大小都相等的缓冲。

**D3D11\_SUBRESOURCE\_DATA**结构体的定义如下：

typedef struct D3D11\_SUBRESOURCE\_DATA {

const void \*pSysMem;

UINT SysMemPitch;

UINT SysMemSlicePitch;

} D3D11\_SUBRESOURCE\_DATA;

**1．pSysMem**：包含初始化数据的系统内存数组的指针。当缓冲区可以存储n个顶点时，对应的初始化数组也应至少包含n个顶点，从而使整个缓冲区得到初始化。

**2．SysMemPitch**：顶点缓冲区不使用该参数。

**3．SysMemSlicePitch**：顶点缓冲区不使用该参数。

下面的代码创建了一个只读的顶点缓冲区，并以中心在原点上的立方体的8顶点来初始化该缓冲区。之所以说该缓冲区是只读的，是因为当立方体创建后相关的几何数据从不改变——始终保持为一个立方体。另外，我们为每个顶点指定了不同的颜色；这些颜色将用于立方体着色，我们会在本章随后的小节中对此进行讲解。

// 定义在d3dUtil.h中的Colors命名空间

//

// #define XMGLOBALCONST extern CONST \_\_declspec(selectany)

// 1. extern so there is only one copy of the variable, and not a separate

// private copy in each .obj.

// 2. \_\_declspec(selectany) so that the compiler does not complain about

// multiple definitions in a .cpp file (it can pick anyone and discard

// the rest because they are constant--all the same).

namespace Colors

{

XMGLOBALCONST XMVECTORF32 White = {1.0f, 1.0f, 1.0f, 1.0f};

XMGLOBALCONST XMVECTORF32 Black = {0.0f, 0.0f, 0.0f, 1.0f};

XMGLOBALCONST XMVECTORF32 Red = {1.0f, 0.0f, 0.0f, 1.0f};

XMGLOBALCONST XMVECTORF32 Green = {0.0f, 1.0f, 0.0f, 1.0f};

XMGLOBALCONST XMVECTORF32 Blue = {0.0f, 0.0f, 1.0f, 1.0f};

XMGLOBALCONST XMVECTORF32 Yellow = {1.0f, 1.0f, 0.0f, 1.0f};

XMGLOBALCONST XMVECTORF32 Cyan = {0.0f, 1.0f, 1.0f, 1.0f};

XMGLOBALCONST XMVECTORF32 Magenta = {1.0f, 0.0f, 1.0f, 1.0f};

XMGLOBALCONST XMVECTORF32 Silver = {0.75f, 0.75f, 0.75f, 1.0f};

XMGLOBALCONST XMVECTORF32 LightSteelBlue = {0.69f, 0.77f, 0.87f, 1.0f};

}

// 创建顶点缓冲

Vertex vertices[] =

{

{ XMFLOAT3(-1.0f, -1.0f, -1.0f), (const float\*)&Colors::White },

{ XMFLOAT3(-1.0f, +1.0f, -1.0f), (const float\*)&Colors::Black },

{ XMFLOAT3(+1.0f, +1.0f, -1.0f), (const float\*)&Colors::Red },

{ XMFLOAT3(+1.0f, -1.0f, -1.0f), (const float\*)&Colors::Green },

{ XMFLOAT3(-1.0f, -1.0f, +1.0f), (const float\*)&Colors::Blue },

{ XMFLOAT3(-1.0f, +1.0f, +1.0f), (const float\*)&Colors::Yellow },

{ XMFLOAT3(+1.0f, +1.0f, +1.0f), (const float\*)&Colors::Cyan },

{ XMFLOAT3(+1.0f, -1.0f, +1.0f), (const float\*)&Colors::Magenta }

};

D3D11\_BUFFER\_DESC vbd;

vbd.Usage = D3D11\_USAGE\_IMMUTABLE;

vbd.ByteWidth = sizeof(Vertex) \* 8;

vbd.BindFlags = D3D11\_BIND\_VERTEX\_BUFFER;

vbd.CPUAccessFlags = 0;

vbd.MiscFlags = 0;

vbd.StructureByteStride = 0;

D3D11\_SUBRESOURCE\_DATA vinitData;

vinitData.pSysMem = vertices;

ID3D11Buffer \* mVB;

HR(md3dDevice->CreateBuffer(&vbd, &vinitData, &mVB));

Vertex类型和颜色由以下结构定义：

struct Vertex

{

XMFLOAT3 Pos;

XMFLOAT4 Color;

};

在创建顶点缓冲区后， 我们必须把它绑定到设备的输入槽上，只有这样才能将顶点送入管线。这一工作使用如下方法完成：

void ID3D11DeviceContext::IASetVertexBuffers(

UINT StartSlot,

UINT NumBuffers,

ID3D10Buffer \*const \*ppVertexBuffers,

const UINT \*pStrides,

const UINT \*pOffsets);

**1．StartSlot**：顶点缓冲区所要绑定的起始输入槽。一共有16个输入槽，索引依次为0到15。

**2．NumBuffers**：顶点缓冲区所要绑定的输入槽的数量，如果起始输入槽为索引k，我们绑定了n个缓冲，那么缓冲将绑定在索引为*I*k，*I*k+1……*I*k+n-1的输入槽上。

**3．ppVertexBuffers**：指向顶点缓冲区数组的第一个元素的指针。

**4．pStrides**：指向步长数组的第一个元素的指针（该数组的每个元素对应一个顶点缓冲区，也就是，第i个步长对应于第i个顶点缓冲区）。这个步长是指顶点缓冲区中的元素的字节长度。

**5．pOffsets**：指向偏移数组的第一个元素的指针（该数组的每个元素对应一个顶点缓冲区，也就是，第i个偏移量对应于第i个顶点缓冲区）。这个偏移量是指从顶点缓冲区的起始位置开始，到输入装配阶段将要开始读取数据的位置之间的字节长度。当希望跳过顶点缓冲区前面的一部分数据时，可以使用该参数。

因为**IASetVertexBuffers**方法支持将一个顶点缓冲数组设置到不同的输入槽中，因此这个方法看起来有点复杂。但是，大多数情况下我们只使用一个输入槽。本章最后的练习部分你会遇到使用两个输入插槽的情况。

顶点缓冲区会一直绑定在输入槽上时。如果不改变输入槽的绑定对象，那么当前的顶点缓冲区会一直驻留在那里。所以，当使用多个顶点缓冲区时，你可以按照下面的形式组织代码：

ID3D11Buffer\* mVB1; // stores vertices of type Vertex1

ID3D11Buffer\* mVB2; // stores vertices of type Vertex2

/\*...Create the vertex buffers...\*/

UINT stride = sizeof(Vertex1);

UINT offset = 0;

md3dImmediateContext->IASetVertexBuffers(0, 1, &mVB1, &stride, &offset);

/\* ...draw objects using vertex buffer 1... \*/

stride = sizeof(Vertex2);

offset = 0;

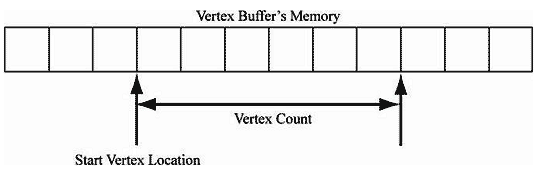
md3dImmediateContext->IASetVertexBuffers(0, 1, &mVB2, &stride, &offset);

/\* ...draw objects using vertex buffer 2... \*/

把顶点缓冲区指定给输入槽并不能实现顶点的绘制；它只是绘制前的准备工作（准备把顶点传送到管线）。顶点的实际绘制工作由**ID3D11DeviceContext::Draw**方法完成：

void ID3D11DeviceContext::Draw(UINT VertexCount, UINT StartVertexLocation);

这两个参数定义了在顶点缓冲区中所要绘制的顶点的范围，如图6.2所示。

****

**图6.2 StartVertexLocation 指定了在顶点缓冲区中所要绘制的第一个顶点的索引（从0开始）。VertexCount指定了所要绘制的顶点的数量。**