# 6.13 动态顶点缓冲

到目前为止，我们一直使用的是静态缓冲（static buffer），它的内容是在初始化时固定下来的。相比之下，动态缓冲（dynamic buffer）的内容可以在每一帧中进行修改。当实现一些动画效果时，我们通常使用动态缓冲区。例如，我们要模拟一个水波效果，并通过函数*f*(*x* ,*z* *,t*)来描述水波方程，计算当时间为*t*时，*xz*平面上的每个点的高度。在这一情景中，我们必须使用“山峰与河谷”中的那种三角形网格，将每个网格点代入*f*(*x*, *z* , *t*)函数得到相应的水波高度。由于该函数依赖于时间*t*（即，水面会随着时间而变化），我们必须在很短的时间内（比如1/30秒）重新计算这些网格点，以得到较为平滑的动画。所以，我们必须使用动态顶点缓冲区来实时更新三角形网格顶点的高度。

前面提到，为了获得一个动态缓冲区，我们必须在创建缓冲区时将Usage标志值指定为**D3D11\_USAGE\_DYNAMIC**；同时，由于我们要向缓冲区写入数据，所以必须将CPU访问标志值指定为**D3D11\_CPU\_ACCESS\_WRITE**。

D3D11\_BUFFER\_DESC vbd;

vbd.Usage = D3D11\_USAGE\_DYNAMIC;

vbd.ByteWidth = sizeof(Vertex) \* mWaves.VertexCount();

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

vbd.CPUAccessFlags = D3D11\_CPU\_ACCESS\_WRITE;

vbd.MiscFlags = 0;

HR(md3dDevice->CreateBuffer(&vbd, 0, &mWavesVB));

**然后，使用ID3D11Buffer::Map**函数获取缓冲区内存的起始地址指针，并向它写入数据：

HRESULT ID3D11DeviceContext::Map(

ID3D11Resource \*pResource ,

UINT Subresource,

D3D11\_MAP MapType,

UINT MapFlags,

D3D11\_MAPPED\_SUBRESOURCE \*pMappedResource);

**1．pResource**：指向要访问的用于读/写的资源的指针。缓冲是一种Direct3D 11资源，其他类型的资源，例如纹理资源，也可以使用这个方法进行访问。

**2．Subresource**：包含在资源中的子资源的索引。后面我们会看到如何使用这个索引，而缓冲不包含子资源，所以设置为0。

**3．MapType**：常用的标志有以下几个：

* **D3D11\_MAP\_WRITE\_DISCARD**：让硬件抛弃旧缓冲，返回一个指向新分配缓冲的指针，通过指定这个标志，可以让我们写入新分配的缓冲的同时，让硬件绘制已抛弃的缓冲中的内容，可以防止绘制停顿。
* **D3D11\_MAP\_WRITE\_NO\_OVERWRITE**：我们只会写入缓冲中未初始化的部分；通过指定这个标志，可以让我们写入未初始化的缓冲的同时，让硬件绘制前面已经写入的内容，可以防止绘制停顿。
* **D3D11\_MAP\_READ**：表示应用程序（CPU）会读取GPU缓冲的的一个副本到系统内存中。

**4．MapFlags**：可选标志，这里不使用，所以设置为0；具体细节可参见SDK文档。

**5．pMappedResource**：返回一个指向**D3D11\_MAPPED\_SUBRESOURCE**的指针，这样我们就可以访问用于读/写的资源数据。

**D3D11\_MAPPED\_SUBRESOURCE**结构定义如下：

typedef struct D3D11\_MAPPED\_SUBRESOURCE{

void \*pData;

UINT Row Pitch;

UINT DepthPitch;

}D3D11\_MAPPED\_SUBRESOURCE

**1．pData**：指向用于读/写的资源内存的指针，你必须将它转换为资源中存储的数据的格式。

**2．RowPitch**：资源中一行数据的字节大小。例如，对于一个2D纹理来说，这个大小为一行的字节大小。

**3．DepthPitch**：资源中一页数据的大小。例如，对于一个3D纹理来说，这个大小为3D纹理中一个2D图像的字节大小。

**RowPitch**和**DepthPitch**的区别是针对2D和3D资源（类似于2D和3D数组）而言的。顶点/索引缓冲本质上是1D数组，**RowPitch**和**DepthPitch**的值是相同的，都等于顶点/索引缓冲的字节大小。

下面的代码展示如何在水波演示程序中更新顶点缓冲：

D3D11\_MAPPED\_SUBRESOURCE mappedData;

HR(md3dImmediateContext->Map(mWavesVB, 0, D3D11\_MAP\_WRITE\_DISCARD, 0, &mappedData));

Vertex\* v = reinterpret\_cast<Vertex\*>(mappedData.pData);

for(UINT i = 0; i < mWaves.VertexCount(); ++i)

{

v[i].Pos = mWaves[i];

v[i].Color = XMFLOAT4(0.0f, 0.0f, 0.0f, 1.0f);

}

md3dImmediateContext->Unmap(mWavesVB, 0);

当你完成缓冲区的更新操作之后，必须调用**ID3D11Buffer::Unmap**函数。

当使用动态缓冲区时，必然会有一些额外开销，因为这里存在一个从CPU内存向GPU内存回传数据的过程。 所以，在实际工作中应尽可能多使用静态缓冲区，少使用动态缓冲区。在Direct3D的最新版本中已经引入了一些新特性用于减少对动态缓冲区的需求。例如：

1．可以在顶点着色器中实现简单动画。

2．通过渲染到纹理（render to texture）和顶点纹理推送（vertex texture fetch）功能，可以实现完全运行在GPU上的水波模拟动画。

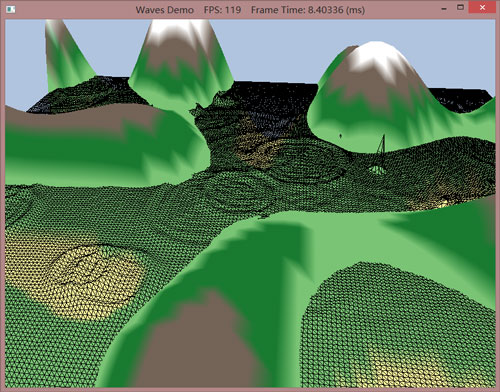
3．几何着色器为GPU提供了创建和销毁图元的能力，在以前没有几何着色器时，这些工作都是由CPU来完成的。

索引缓冲区可以是动态的。不过，在水波演示程序中，三角形的拓扑结构始终不变，只有顶点高度会发生变化；所以，这里只需要让顶点缓冲区变为动态缓冲区。

本章的水波演示程序使用了一个动态缓冲区来实现简单的水波效果。本书不会将重点放在水波模拟算法的实现细节上（有兴趣的读者可以参见[Lengyel02]），我们只是用它来说明动态缓冲区的用法：在CPU上更新模拟数据，然后调用**Map/Unmap**方法更新顶点缓冲区。

**注意**：在水波演示程序中，我们以线框模式渲染水波；是因为我们现在还没有讲到灯光的用法，在实体填充模式下，很难看出水波的运动效果。

**注意**：我们再次强调，这个示例应该在GPU上使用更高级的方式实现，比如渲染到纹理和顶点纹理推送。但是由于我们还没有讲到些技术，所以现在只能在CPU上实现，暂时使用动态顶点缓冲区来更新顶点。

****

**6.19 水波演示程序截图**