# 8.3 创建和启用纹理

纹理数据通常是存储在磁盘上的图像文件。我们需要将它读取出来，并载入到一个**ID3D11Texture2D**对象中（参见**D3DX11CreateTextureFromFile**）。不过，纹理资源是不能被直接绑定到渲染管线上的；我们需要为纹理创建一个着色器资源视图（**ID3D11ShaderResourceView**），然后将视图绑定到管线上。这个过程可分为两步：

1．调用**D3DX11CreateTextureFromFile**，读取存储在磁盘上的图像文件，创建**ID3D11Texture2D**对象。

2．调用**ID3D11Device::CreateShaderResourceView**，为纹理创建对应的着色器资源视图。

这两步也可通过如下D3DX函数一次完成：

HRESULT D3DX11CreateShaderResourceViewFromFile(

 ID3D11Device \*pDevice,

 LPCTSTR pSrcFile,

 D3DX11\_IMAGE\_LOAD\_INFO \*pLoadInfo,

 ID3DX11ThreadPump \*pPump,

 ID3D11ShaderResourceView \*\*ppShaderResourceView,

 HRESULT \*pHResult

);

**1．pDevice**：创建纹理时使用的Direct3D设备指针。

**2．pSrcFile**：所要载入的图像的文件名。

**3．pLoadInfo**：可选的图像信息；当设为空值时，该函数使用源图像文件信息。例如，我们在这里指定空值，那么源图像的尺寸就是纹理的尺寸；另外，该函数还会生成一个完整的多级渐近纹理链（8.4.2节）。这个默认值很好，因为它所执行的一系列行为通常都是我们想要的。

**4．pPump**：在一个新的线程中载入资源。如果要在当前的工作线程中载入资源，请将它设为空值。在本书中，这个参数总是设为空值。

**5．ppShaderResourceView**：返回纹理的着色器资源视图指针。

**6．pHResult**：当pPump设为空值时，该参数也应设为空值。

该函数可以载入以下格式的图像文件：BMP、JPG、PNG、DDS、TIFF、GIF、WMP（参见**D3DX11\_IMAGE\_FILE\_FORMAT**）。

**注意**：有时，我们所说的“纹理”实际上是指“与它关联的着色器资源视图”。例如，我们会说：“将纹理绑定到管线上”，但实际上是指“将它的视图绑定到管线上”。

例如，要从图像文件WoodCrate01.dds创建一个纹理，我们可以编写如下代码：

ID3D11ShaderResourceView\* mDiffuseMapSRV;

HR(D3DX11CreateShaderResourceViewFromFile(md3dDevice,

L"WoodCrate01.dds", 0, 0, &mDiffuseMapSRV, 0 ));

在纹理载入之后，我们需要把它指定给一个effect变量，使像素着色器中的代码可以访问到这个资源。在.fx文件中，2D纹理对象由**Texture2D**类型表示；例如，我们可以在effect文件中声明一个纹理变量：

// Nonnumeric values cannot be added to a cbuffer.

Texture2D gDiffuseMap;

如注释所述：“非数值对象不能放在常量缓冲区中”，纹理对象必须放在常量缓冲区之外。 我们可以在C++应用程序代码中获取这个**Texture2D**对象的指针（它是一个着色器资源变量）：

ID3D11EffectShaderResourceVariable\* mfxDiffuseMapVar;

mfxDiffuseMapVar = mFX->GetVariableByName("gDiffuseMap")->AsShaderResource();

在我们获取**Texture2D**对象的指针之后，可以通过如下C++接口更新纹理对象：

// Set the C++ texture resource view to the effect texture variable.

mfxDiffuseMapVar->SetResource(mDiffuseMapSRV);

与其他的effect效果变量相同，当我们在绘图调用之间更新效果变量时，一定要调用**Apply**方法：

// set crate texture

mfxDiffuseMapVar->SetResource(mCrateMapSRV);

pass->Apply(0, md3dImmediateContext);

DrawCrate();

// set grass texture

mfxDiffuseMapVar->SetResource(mGrassMapSRV);

pass->Apply(0, md3dImmediateContext);

DrawGrass();

// set brick texture

mfxDiffuseMapVar->SetResource(mBrickMapSRV);

pass->Apply(0, md3dImmediateContext);

DrawBricks();

使用纹理贴图集可以在一次绘制调用中绘制更多的几何体，所以可以提高性能。例如，我们使用了如图8.3所示的包含条板、草地、砖墙纹理的贴图集，通过调整纹理坐标在每个对象上映射不同的子纹理，我们就可以在一次绘制调用中绘制整个几何体（假设每个对象都没有需要改变的其他参数）：

// set texture atlas

mfxDiffuseMap ->SetResource(mAtlasSRV);

pass ->Apply(0, md3dImmediateContext);

DrawCrateGrassAndBricks();

调用绘制需要耗费资源，使用这个技术可以使绘制调用减到最少。

**注意**：纹理资源实际上可以由任何着色器（顶点、几何或像素）使用。但是目前我们只在像素着色器中使用纹理。如前所述，纹理本质上是一种特殊的数组，所以不难想象，这些数组数据也可以在顶点着色器和几何着色器中使用。