# 纹理贴图映射（texture mapping）

# ID3D11Texture2D

2D纹理是一种数据元素矩阵。2D纹理的用途之一是存储2D图像数据，在纹理的每个元素中保存一个像素颜色。但这不是纹理的唯一用途，例如在法线贴图中，纹理存储的不是颜色，而是3D向量。

类似的有**ID3D11Texture1D**和**ID3D11Texture3D**。

纹理只支持特定格式的数据存储，这些格式由**DXGI\_FORMAT**枚举类型描述。常用的有

1．DXGI\_FORMAT\_R32G32B32\_FLOAT：每个元素包含3个32位浮点分量。

2．DXGI\_FORMAT\_R16G16B16A16\_UNORM：每个元素包含4个16位分量，分量的取值范围在[0,1]区间内。

3．DXGI\_FORMAT\_R32G32\_UINT：每个元素包含2个32位无符号整数分量。

4．DXGI\_FORMAT\_R8G8B8A8\_UNORM：每个元素包含4个8位无符号整数分量，分量的取值范围在[0,1]区间内。

5．DXGI\_FORMAT\_R8G8B8A8\_SNORM：每个元素包含4个8 位有符号整数分量，分量的取值范围在[−1,1] 区间内。

6．DXGI\_FORMAT\_R8G8B8A8\_SINT：每个元素包含4个8位有符号整数分量，分量的取值范围在[−128, 127]区间内。

7．DXGI\_FORMAT\_R8G8B8A8\_UINT：每个元素包含4个8位无符号整数分量，分量的取值范围在[0, 255]区间内。

纹理可以被绑定到渲染管线的不同阶段。比较常见的情况是将纹理作为渲染目标（即，Direct3D渲染到纹理）和着色器资源（即，在着色器中对纹理进行采样）。

对应的**D3D11\_BIND\_FLAG**为： **D3D11\_BIND\_RENDER\_TARGET** | **D3D11\_BIND\_SHADER\_RESOURCE**

资源不能被直接绑定到一个管线阶段；我们只能把与资源关联的资源视图绑定到不同的管线阶段。无论以哪种方式使用纹理，Direct3D始终要求我们在初始化时为纹理创建相关的资源视图。这样有助于提高运行效率。

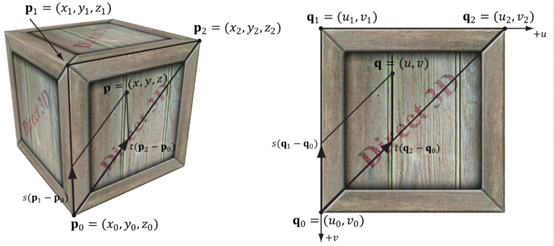
相应的两种视图为：渲染目标视图（**ID3D11RenderTargetView**）和着色器资源视图（**ID3D11ShaderResourceView**）。

# 纹理坐标

Direct3D的纹理坐标系由表示图像水平方向的u轴和表示图像垂直方向的v轴组成，且0≤(u,v)≤1。这样使 Direct3D拥有一个独立于纹理尺寸的坐标空间。例如，无论纹理实际多大，(0.5, 0.5)永远表示中心的纹理元素。

每个顶点拥有一个纹理坐标，用于指定纹理上的点。既每个3D点都有一个相应的2D纹理点。这样，由3个顶点构成的每个3D三角形在纹理空间中都会有一个相应的2D纹理三角形。

|  |
| --- |
| // Basic 32-byte vertex structure.  struct Basic32  {  XMFLOAT3 Pos;  XMFLOAT3 Normal;  XMFLOAT2 Tex;  };  **const** D3D11\_INPUT\_ELEMENT\_DESC InputLayoutDesc::Basic32[3] =  {  {"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}  }; |



# 寻址模式

当给定一个纹理坐标(u,v)∈[0,1]时，纹理函数T返回一个颜色(r,g,b,a)，当纹理坐标超出规定的范围时，Direct3D提供4种不同的方式来获得结果，这称为寻址模式：重复（wrap）、边框颜色（border color）、截取（clamp）和镜像（mirror）。

**重复(D3D11\_TEXTURE\_ADDRESS\_WRAP)：通过在每个整点连接处重复图像来扩展纹理。**

**边框颜色(D3D11\_TEXTURE\_ADDRESS\_BORDER)：通过将每个不在[0,1]区间内的(u,v)映射为程序员指定的某个颜色来扩展纹理。**

**截取(D3D11\_TEXTURE\_ADDRESS\_CLAMP)：通过将每个不在[0,1]区间内的(u,v)映射为颜色T(u0,v0)来扩展纹理。其中，(u0,v0)∈[0,1]，(u0,v0)是与(u,v)距离最近的点。**

**镜像(D3D11\_TEXTURE\_ADDRESS\_MIRROR)：通过在每个整点连接处镜像图像来扩展纹理。**

通过**D3D11\_SAMPLER\_DESC**的**AddressU**，**AddressV**来指定。

重复寻址模式可能是最常用的寻址模式。因为它可以把纹理平铺在物体表面上，使我们在不提供额外数据的情况下有效地提高纹理分辨率（虽然些额外的分辨率是重复的）。当使用平铺时，我们通常希望纹理是无缝的（seamless）。例如砖墙的纹理。

# 使用纹理

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。之后再们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。

关键API：

**D3DX11CreateTextureFromFile**

**ID3D11Device::CreateShaderResourceView**

**D3DX11CreateShaderResourceViewFromFile**

|  |
| --- |
| HRESULT hr = D3DX11CreateShaderResourceViewFromFile(  m\_d3dDevice.Get(), L"CrateGame\\WoodCrate01.dds", nullptr, nullptr, m\_diffuseMapView.GetAddressOf(), nullptr);  DX::ThrowIfFailed(hr);    m\_d3dContext->PSSetShaderResources(0, 1, m\_diffuseMapView.GetAddressOf()); |

# 纹理过滤

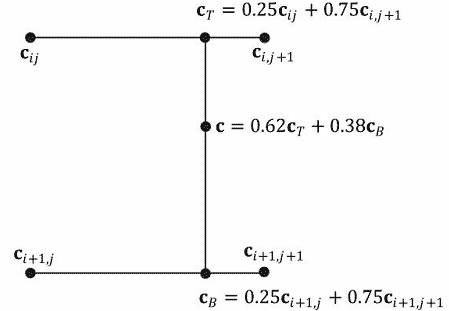
倍增(Magnification)

当观察点离场景中的物体很近时，物体的纹理会被拉大，使得我们将要用很少的纹理元素来覆盖很多的像素。

例如渲染目标的分辨率为1024×1024，物体纹理的分辨率为256×256，则每个纹理元素要覆盖4个像素。

通常使用线性插值来解决这一问题。

2D线性插值也称为双线性插值（bilinear interpolation）。给出一对位于4个纹理元素之间的纹理坐标，我们在u方向上进行两次1D线性插值，然后在v方向上进行一次1D线性插值。



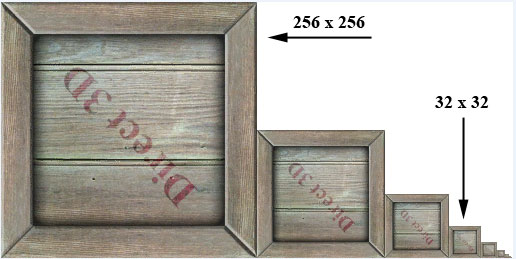
缩减(Minification)

与倍增的情况恰好相反，若物体离观察点较远，则需用较多的纹理元素会被映射为较少的像素。

例如物体在屏幕中占64×64的像素，们要把256×256个纹理元素均匀地缩减为64×64个纹理元素。

多级渐近贴图映射（mipmapping）则用于处理该情况。

在初始化时（或创建资源时），通过对图像进行降阶采样生成纹理的多个缩略版本来创建多级渐近纹理链（mipmap chain）。



在运行时，为纹理映射挑选两个与屏幕几何体分辨率最匹配的多级渐近纹理层（其中，一个比屏幕几何体分辨率大一些，另一个比屏幕几何体分辨率小一些）。然后，在这两个多级渐近纹理层上插值，分别取出一个纹理颜色。最后，在这两个纹理颜色之间进行插值。这样的过滤称为三线性过滤(trilinear filtering)。

各向异性过滤（anisotropic filter）

当多边形的法线向量与摄像机的观察向量夹角过大时（例如，当多边形垂直于观察窗口时），这种过滤可以有效缓解图像的失真问题。

过滤方式通过D3D11\_SAMPLER\_DESC的Filter来指定。

在代码中使用**CreateSamplerState**和相应的**SetSamplers**函数来指定采样规则。在HLSL中使用**Texture2D::Sample**对一个纹理进行采样。**PSSetSamplers**的slot对应shader里的samplers slot。

|  |
| --- |
| CD3D11\_SAMPLER\_DESC samplerDesc(D3D11\_DEFAULT);  HRESULT hr = m\_d3dDevice->CreateSamplerState(&samplerDesc, m\_samplerState.GetAddressOf());  DX::ThrowIfFailed(hr);  m\_d3dContext->PSSetSamplers(0, 1, m\_samplerState.GetAddressOf()); |

Shader

|  |
| --- |
| Texture2D gDiffuseMap : register(t0);  SamplerState gsamLinear : register(s0);    ......    float4 PS(VertexOut pin) : SV\_Target  {  float4 color = gDiffuseMap.Sample(gsamLinear, pin.TextCoor);  **return** color;  } |

# 对纹理进行变换

如前所述，纹理坐标表示纹理平面上的2D点。因此，我们可以像使用其他坐标一样，对纹理坐标进行平移、旋转和缩放。下面是一些会对纹理进行变换的例子：

1．沿着墙体拉伸一幅砖块纹理。该墙体顶点的纹理坐标在[0,1]区间内。我们将每个纹理坐标乘以4，使区间扩大为[0,4]，让纹理在墙体上重复4×4次。

2．在一片晴朗的蓝天上（即，在一个天空球上）拉伸一幅白云纹理。通过一个时间函数控制纹理坐标的平移，形成白云在天上飘动的效果。

3．当实现粒子效果时，有时需要对纹理坐标进行旋转；例如，随着时间的推移旋转一幅火球纹理。

纹理坐标变换与普通坐标变换的实现方式相同。我们指定一个变换矩阵，将纹理坐标向量与该矩阵相乘。例如：

vOut.texC = mul(float4(vIn.texC, 0.0f, 1.0f),gTexMtx);