# 深度模板(depth stencil)缓冲及应用：镜子

# 什么是stencil

模板缓冲区的用法就像是模板一样，它可以挡住某些像素片段，不让它们存入后台缓冲。

比如喷油漆时使用的图案模板，先把模板贴在汽车上或者其他什么地方，然后开始喷油漆。在模板镂空的地方会有油漆喷到汽车上，而没有镂空的地方会挡住油漆。在喷完之后，揭下模板，图案就喷涂在汽车上了。

# 创建和清空缓冲

在初始化时创建ID3D11DepthStencilView，在每帧开始时重置该buffer为初始值。具体见[初始化和渲染管线](https://ac.virtuosgames.com:8443/pages/viewpage.action?pageId=84024154)。

# 模板测试

我们可以使用模板缓冲区来判断一个特定像素是否可以写入后台缓冲区，这个过程称为模板测试（stencil test）。

模板测试在输出合并阶段进行，在深度测试之前。

其实现过程为

|  |
| --- |
| if( StencilRef & StencilReadMask ⊴ Value &StencilReadMask)  accept pixel  else  reject pixel |

# ID3D11DepthStencilState

通过该接口来控制depth和stencil的使用。

首先需要填充一个**D3D11\_DEPTH\_STENCIL\_DESC**

|  |
| --- |
| typedef struct D3D11\_DEPTH\_STENCIL\_DESC  {  BOOL DepthEnable;  D3D11\_DEPTH\_WRITE\_MASK DepthWriteMask;  D3D11\_COMPARISON\_FUNC DepthFunc;  BOOL StencilEnable;  UINT8 StencilReadMask;  UINT8 StencilWriteMask;  D3D11\_DEPTH\_STENCILOP\_DESC FrontFace;  D3D11\_DEPTH\_STENCILOP\_DESC BackFace;  } D3D11\_DEPTH\_STENCIL\_DESC; |

DepthEnable：是否启用深度测试。

DepthWriteMask： D3D11\_DEPTH\_WRITE\_MASK\_ZERO表示禁用深度缓冲区的写入功能，但深度测试依然有效。D3D11\_DEPTH\_WRITE\_MASK\_ALL表示启用深度缓冲区的写入功能；当深度/模板测试都通过时，新的深度值会被写入深度缓冲区。

DepthFunc：一个用于描述深度测试函数的D3D11\_COMPARISON\_FUNC枚举类型成员。我们一般使用D3D11\_COMPARISON\_LESS实现普通的深度测试

StencilEnable：是否启用模板测试

StencilReadMask：在模板测试时使用的掩码StencilReadMask。

默认掩码不屏蔽任何二进制位：

#define    D3D11\_DEFAULT\_STENCIL\_WRITE\_MASK    ( 0xff )

StencilWriteMask： 当更新模板缓冲区时，我们可以通过掩码来屏蔽某些二进制位，不让它们存入模板缓冲区。

FrontFace/BackFace：一个**D3D11\_DEPTH\_STENCILOP\_DESC**结构体，用于描述模板缓冲区如何处理朝前/朝后的三角形

|  |
| --- |
| typedef struct D3D11\_DEPTH\_STENCILOP\_DESC  {  D3D11\_STENCIL\_OP StencilFailOp;  D3D11\_STENCIL\_OP StencilDepthFailOp;  D3D11\_STENCIL\_OP StencilPassOp;  D3D11\_COMPARISON\_FUNC StencilFunc;  } D3D11\_DEPTH\_STENCILOP\_DESC; |

StencilFailOp：描述当一个像素片段的模板测试失败时，应如何更新模板缓冲区。

StencilDepthFailOp：描述当一个像素片段的模板测试成功而深度测试失败时，应如何更新模板缓冲区。

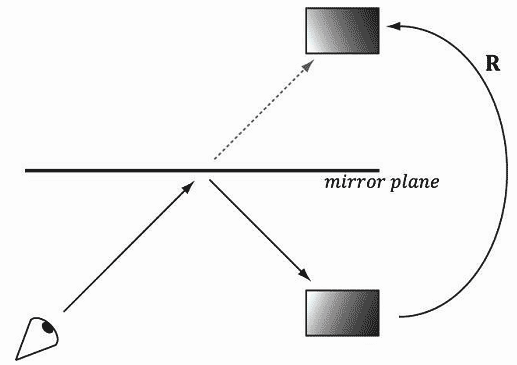
StencilPassOp：描述当一个像素片段的模板测试和深度测试均成功时，应如何更新模板缓冲区。

StencilFunc：指定模板测试时使用的比较函数⊴。

填充好D3D11\_DEPTH\_STENCIL\_DESC之后，使用**ID3D11Device::CreateDepthStencilState**来创建ID3D11DepthStencilState并使用**ID3D11DeviceContext::OMSetDepthStencilState**来指定状态，这里的第二个参数既是StencilRef。

# 平面镜像的实现

要绘制一个物体的映像，我们必须在镜子平面上对它进行反射，但是反射后的物体映像会被渲染到镜子之外的区域。为了使映像只应该显示在镜子里面，我们可以使用模板缓冲区来解决一问题。



**详细步骤**

1.将其他物体（不包括镜子）渲染到后台缓冲区。注意，这一步不会修改模板缓冲区。

2.将模板缓冲区清为0。

|  |
| --- |
| m\_d3dContext->ClearDepthStencilView(m\_depthStencilView.Get(), D3D11\_CLEAR\_STENCIL, 1.0f, 0); |

下图展示了此时的后台缓冲区和模板缓冲区。

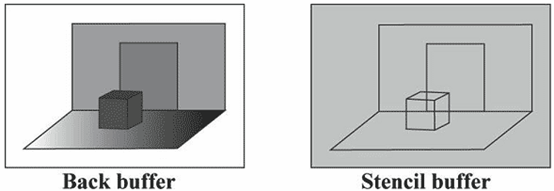


图: 将场景渲染到后台缓冲区，并将模板缓冲区清为0（由浅灰色表示）。模板缓冲区上的黑色轮廓线用于说明后台缓冲区像素与模板缓冲区像素之间的对应关系——它们并不代表绘制在模板缓冲区上的任何数据。

3． 把镜子渲染到模板缓冲区。将模板测试函数设为 D3D10\_COMPARISON\_ALWAYS（始终成功），并指定当模板测试成功时将模板缓冲区元素替换 （D3D10\_STENCIL\_OP\_REPLACE）为1。将StencilDepthFailOp设为D3D10\_STENCIL\_OP\_KEEP，当深度测试失败时不更新模板缓冲区（如果物体挡住 了镜子的某一部分，那么就会发生这种情况）。由于我们只将镜子绘制到模板缓冲区， 所以在模板缓冲区中只有镜子的可视区域对应的像素为 1，而其他像素均为 0。下图展示了更新后的模板缓冲区。实际上，我们是给镜子在模板缓冲区中的可视区域做了标记。

|  |
| --- |
| // Render the mirror to get the stencil    CD3D11\_DEPTH\_STENCIL\_DESC mirrorDesc(D3D11\_DEFAULT);  mirrorDesc.DepthEnable = TRUE;  mirrorDesc.StencilEnable = TRUE;  mirrorDesc.FrontFace.StencilPassOp = D3D11\_STENCIL\_OP\_REPLACE;  mirrorDesc.FrontFace.StencilFunc = D3D11\_COMPARISON\_ALWAYS;    ComPtr<ID3D11DepthStencilState> mMirrorState;  HRESULT hr = m\_d3dDevice->CreateDepthStencilState(&mirrorDesc, mMirrorState.GetAddressOf());  DX::ThrowIfFailed(hr);  m\_d3dContext->OMSetDepthStencilState(mMirrorState.Get(), stencilRef);    mirror->Render(); |

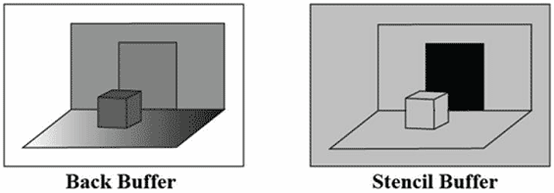


图:  把镜子渲染到模板缓冲区，这个操作的实际上是在模板缓冲区中标记了镜子的可视区域。实心黑色区域的模板元素值为1。注意，被盒子挡住的区域不会设为1，因为这一部分根本无法通过深度测试（盒子挡住了镜子前面的这一部分）。

3.反射物体

需要注意，当三角形被反射到平面上时，它的环绕顺序和平面法线都不会翻转。所以，在反射之后，原来向外的法线会变成向内的法线（参见图10.5）。为了纠正一错误，我们必须告诉Direct3D将逆时针环绕的三角形视为朝前的三角形，将顺时针环绕的三角形视为朝后的三角形（与我们在5.10.2节的约定恰好相反）。这样可以有效地翻转法线方向，使它们在反射之后仍然向外。我们通过设置光栅器状态来翻转环绕顺序。

|  |
| --- |
| CD3D11\_RASTERIZER\_DESC reflectRasterizerDesc(D3D11\_DEFAULT);  reflectRasterizerDesc.FrontCounterClockwise = TRUE;    ComPtr<ID3D11RasterizerState> mReflectRasterizerState;  hr = m\_d3dDevice->CreateRasterizerState(&reflectRasterizerDesc, mReflectRasterizerState.GetAddressOf());  DX::ThrowIfFailed(hr);  m\_d3dContext->RSSetState(mReflectRasterizerState.Get());    // Reflect plane  XMFLOAT4 plane(0.f, 0.f, -1.f, 0.f);  XMVECTOR mirrorPlane = XMLoadFloat4(&plane);  XMMATRIX reflectMatrix = XMMatrixReflect(mirrorPlane);    ......    // Reflect objects  XMMATRIX world = XMLoadFloat4x4((\*it)->m\_world);  XMMATRIX reflectWorld = XMMatrixMultiply(world, reflectMatrix);  XMStoreFloat4x4((\*it)->m\_world, reflectWorld);    (\*it)->Update(m\_timer);  (\*it)->Render();    XMStoreFloat4x4((\*it)->m\_world, world);    ......    m\_d3dContext->RSSetState(nullptr); |

当绘制映像时，我们还需要在镜子平面上反射光源。否则光照会显得很不真实。

|  |
| --- |
| // Reflect light  XMVECTOR pos = XMLoadFloat3(&m\_spotLight.Position);  XMVECTOR reflectPos = XMVector3Transform(pos, reflectMatrix);  XMStoreFloat3(&m\_spotLight.Position, reflectPos);    XMVECTOR dir = XMLoadFloat3(&m\_spotLight.Direction);  XMVECTOR reflectDir = XMVector3Transform(dir, reflectMatrix);  XMStoreFloat3(&m\_spotLight.Direction, reflectDir);    m\_cbPerFrame.spotLight = m\_spotLight;  m\_d3dContext->UpdateSubresource(m\_constantBufferPerFrame.Get(), 0, nullptr, &m\_cbPerFrame, 0, 0);    ......    // Restore the light  XMStoreFloat3(&m\_spotLight.Direction, dir);  XMStoreFloat3(&m\_spotLight.Position, pos); |

4.现在我们将反射后的物体映像渲染到后台缓冲区和模板缓冲区。

此时，只有通过了模板测试的像素片段才能渲染到后台缓冲区中。这次，我们要将模板测试函数设为D3D11\_COMPARISON\_EQUAL，使模板元素为1时测试成功。通过一方式，映像只会渲染到模板元素为1的区域中。由于在模板缓冲区中只有镜子的可视区域的模板元素为 1，所以映像只会被渲染到镜子里面。

|  |
| --- |
| // Use stencil to render other objects    // Because all the objects is behind the wall, so close depth test here or they won't draw.  CD3D11\_DEPTH\_STENCIL\_DESC reflectObjecDesc(D3D11\_DEFAULT);  reflectObjecDesc.DepthEnable = FALSE;  reflectObjecDesc.StencilEnable = TRUE;  reflectObjecDesc.FrontFace.StencilPassOp = D3D11\_STENCIL\_OP\_KEEP;  reflectObjecDesc.FrontFace.StencilFunc = D3D11\_COMPARISON\_EQUAL;    ComPtr<ID3D11DepthStencilState> mReflectObjectState;  hr = m\_d3dDevice->CreateDepthStencilState(&reflectObjecDesc, mReflectObjectState.GetAddressOf());  DX::ThrowIfFailed(hr);  m\_d3dContext->OMSetDepthStencilState(mReflectObjectState.Get(), stencilRef);    ......    m\_d3dContext->OMSetDepthStencilState(nullptr, 0); |

为了能显示镜子之后的镜像，我们需要使用透明混合绘制镜子和镜像。

|  |
| --- |
| // Enable blend to render the object  CD3D11\_BLEND\_DESC reflectBlendDesc(D3D11\_DEFAULT);  reflectBlendDesc.RenderTarget[0].BlendEnable = TRUE;  reflectBlendDesc.RenderTarget[0].SrcBlend = D3D11\_BLEND\_BLEND\_FACTOR;  reflectBlendDesc.RenderTarget[0].DestBlend = D3D11\_BLEND\_INV\_BLEND\_FACTOR;  reflectBlendDesc.RenderTarget[0].BlendOp = D3D11\_BLEND\_OP\_ADD;  reflectBlendDesc.RenderTarget[0].SrcBlendAlpha = D3D11\_BLEND\_ONE;  reflectBlendDesc.RenderTarget[0].DestBlendAlpha = D3D11\_BLEND\_ZERO;  reflectBlendDesc.RenderTarget[0].BlendOpAlpha = D3D11\_BLEND\_OP\_ADD;    ComPtr<ID3D11BlendState> mReflectBlendState;  hr = m\_d3dDevice->CreateBlendState(&reflectBlendDesc, mReflectBlendState.GetAddressOf());  DX::ThrowIfFailed(hr);  float blendFactors[] = { 0.85f,0.85f,0.85f,1.f };  m\_d3dContext->OMSetBlendState(mReflectBlendState.Get(), blendFactors, UINT\_MAX);    ......    m\_d3dContext->OMSetBlendState(nullptr, blendFactors, UINT\_MAX); |