# 10.4 平面镜像的实现

在自然界中有许多物体的表面都非常光滑，可以像镜子一样反射周围的物体。本节介绍了如何在3D应用程序中模拟镜像效果。为简单起见，我们降低了任务难度，只在平面上实现镜像效果。例如，一辆光滑的汽车可以反射周围的物体；但是，车身是一个平滑曲面，而非平面。我们不选择这样的物体。我们将在光滑的大理石地板或挂在墙上的镜子中渲染物体的映像——换句话说，我们只实现平面上的镜像效果。

要在程序中实现镜像效果，必须解决两个问题。首先，我们必须知道如何在一个任意平面上反射物体，正确地绘制该物体的映像。其次，我们只能在镜子里面显示映像；也就是，我们必须以某种方式将一个表面“标记”为镜子，然后在渲染时只在镜子里面绘制物体映像。回顾图10.1，它最先引入了这一概念。

第一个问题可以很容易地通过解析几何来解决，具体请参见附录C。第二个问题可以通过模板缓冲区来解决。

## 10.4.1 概述

**注意**：当绘制映像时，我们还需要在镜子平面上反射光源。否则，映像中的光照会显得很不真实。

图10.2说明了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。不过，这会出现图10.1所示的问题。即，物体映像（在本例中是头骨）会被渲染到镜子之外的区域（例如，墙面）。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，因为模板缓冲区可以阻止像素渲染到后台缓冲区的某些区域上。所以，我们可以使用模板缓冲区来控制头骨的映像，避免映像渲染到镜子之外的区域。下面给出了具体的实现步骤：

****

**图10.2 观察点从镜子中看到的立方体映像。要模拟一效果，我们需要在镜子平面上反射立方体，然后将立方体映像绘制出来。**

1．将地板、墙壁和头骨（不包括镜子）渲染到后台缓冲区。注意，一步不修改模板缓冲区。

2．将模板缓冲区清为0。图10.3展示了此时的后台缓冲区和模板缓冲区。

****

**图10.3 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．把镜子只渲染到模板缓冲区。我们可以通过设置以下混合状态禁止颜色写入到后台缓冲区中：

D3D11\_RENDER\_TARGET\_BLEND\_DESC::RenderTargetWriteMask = 0;

通过以下设置禁止写入到深度缓冲区中：

D3D11\_DEPTH\_STENCIL\_DESC::DepthWriteMask = D3D11\_DEPTH\_WRITE\_MASK\_ZERO;

在将镜子绘制到模板缓冲时，我们将模板测试函数设为**D3D11\_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（**D3D11\_STENCIL\_OP\_REPLACE**）为1（**StencilRef**）。将**StencilDepthFailOp**设为**D3D11\_STENCIL\_OP\_KEEP**，当深度测试失败时不更新模板缓冲区（如果头骨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。图10.4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。

****

**图10.4 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）。**

4．现在我们将头骨映像渲染到后台缓冲区和模板缓冲区。但是要记住，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，头骨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，所以头骨映像只会被渲染到镜子里面。

5．最后，我们将镜子绘制到后台缓冲区。但是，为了为了能显示镜子之后的头骨镜像，我们需要使用透明混合绘制镜子，如果不这样做，那么镜子就会挡住位于它后面的头骨镜像。要实现这个效果，我们只需定义一个镜子用的材质实例，将漫反射分量的alpha通道设置为0.5，这样镜子表示镜子是半透明的，然后就可以像9.5.4节那样绘制半透明的镜子了：

mMirrorMat.Ambient = XMFLOAT4(0.5f, 0.5f, 0.5f, 1.0f);

mMirrorMat.Diffuse = XMFLOAT4(1.0f, 1.0f, 1.0f, 0.5f);

mMirrorMat.Specular = XMFLOAT4(0.4f, 0.4f, 0.4f, 16.0f);

上述设置给出以下混合方程：

**C** = 0.5·**C**src + 0.5·**C**dst

假设我们已将头骨镜像的像素发送到后台缓冲区中，则50%的颜色来自于镜子（源），50%的颜色来自于头骨（目标）。

## 10.4.2 定义深度/模板状态

要实现上述算法，我们必须定义两个深度/模板状态。一个用于在模板缓冲区上绘制镜子，标记镜子的可视区域。另一个用于绘制头骨映像，使映像只显示在镜子里面。

//

// MarkMirrorDSS

//

D3D11\_DEPTH\_STENCIL\_DESC mirrorDesc;

mirrorDesc.DepthEnable = true;

mirrorDesc.DepthWriteMask = D3D11\_DEPTH\_WRITE\_MASK\_ZERO;

mirrorDesc.DepthFunc = D3D11\_COMPARISON\_LESS;

mirrorDesc.StencilEnable = true;

mirrorDesc.StencilReadMask = 0xff;

mirrorDesc.StencilWriteMask = 0xff;

mirrorDesc.FrontFace.StencilFailOp = D3D11\_STENCIL\_OP\_KEEP;

mirrorDesc.FrontFace.StencilDepthFailOp = D3D11\_STENCIL\_OP\_KEEP;

mirrorDesc.FrontFace.StencilPassOp = D3D11\_STENCIL\_OP\_REPLACE;

mirrorDesc.FrontFace.StencilFunc = D3D11\_COMPARISON\_ALWAYS;

// We are not rendering backfacing polygons, so these settings do not matter.

mirrorDesc.BackFace.StencilFailOp = D3D11\_STENCIL\_OP\_KEEP;

mirrorDesc.BackFace.StencilDepthFailOp = D3D11\_STENCIL\_OP\_KEEP;

mirrorDesc.BackFace.StencilPassOp = D3D11\_STENCIL\_OP\_REPLACE;

mirrorDesc.BackFace.StencilFunc = D3D11\_COMPARISON\_ALWAYS;

HR(device->CreateDepthStencilState(&mirrorDesc, &MarkMirrorDSS));

//

// DrawReflectionDSS

//

D3D11\_DEPTH\_STENCIL\_DESC drawReflectionDesc;

drawReflectionDesc.DepthEnable = true;

drawReflectionDesc.DepthWriteMask = D3D11\_DEPTH\_WRITE\_MASK\_ALL;

drawReflectionDesc.DepthFunc = D3D11\_COMPARISON\_LESS;

drawReflectionDesc.StencilEnable = true;

drawReflectionDesc.StencilReadMask = 0xff;

drawReflectionDesc.StencilWriteMask = 0xff;

drawReflectionDesc.FrontFace.StencilFailOp = D3D11\_STENCIL\_OP\_KEEP;

drawReflectionDesc.FrontFace.StencilDepthFailOp = D3D11\_STENCIL\_OP\_KEEP;

drawReflectionDesc.FrontFace.StencilPassOp = D3D11\_STENCIL\_OP\_KEEP;

drawReflectionDesc.FrontFace.StencilFunc = D3D11\_COMPARISON\_EQUAL;

// We are not rendering backfacing polygons, so these settings do not matter.

drawReflectionDesc.BackFace.StencilFailOp = D3D11\_STENCIL\_OP\_KEEP;

drawReflectionDesc.BackFace.StencilDepthFailOp = D3D11\_STENCIL\_OP\_KEEP;

drawReflectionDesc.BackFace.StencilPassOp = D3D11\_STENCIL\_OP\_KEEP;

drawReflectionDesc.BackFace.StencilFunc = D3D11\_COMPARISON\_EQUAL;

HR(device->CreateDepthStencilState(&drawReflectionDesc, &DrawReflectionDSS));

## 10.4.3 绘制场景

下面是draw方法的代码。为了突出重点，我们略去了不相关的细节，比如设置常量缓冲区的值（完整的细节请参见本例源代码）。

//

// 镜子只绘制到模板缓冲

//

activeTech->GetDesc( &techDesc );

for(UINT p = 0; p < techDesc.Passes; ++p)

{

 ID3DX11EffectPass\* pass = activeTech->GetPassByIndex( p );

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

 // Set per object constants.

 XMMATRIX world = XMLoadFloat4x4(&mRoomWorld);

 XMMATRIX worldInvTranspose = MathHelper::InverseTranspose(world);

 XMMATRIX worldViewProj = world\*view\*proj;

 Effects::BasicFX->SetWorld(world);

 Effects::BasicFX->SetWorldInvTranspose(worldInvTranspose);

 Effects::BasicFX->SetWorldViewProj(worldViewProj);

 Effects::BasicFX->SetTexTransform(XMMatrixIdentity());

 // 不写入渲染目标

 md3dImmediateContext->OMSetBlendState(RenderStates::NoRenderTargetWritesBS, blendFactor, 0xffffffff);

 // 将镜子可见部分的像素绘制到模板缓冲中。

 // 但不要将镜子的深度信息写入深度缓冲中，否则会遮挡之后的头骨镜像。

 md3dImmediateContext->OMSetDepthStencilState(RenderStates::MarkMirrorDSS, 1);

 pass->Apply(0, md3dImmediateContext);

 md3dImmediateContext->Draw(6, 24);

 // 恢复之前的状态

 md3dImmediateContext->OMSetDepthStencilState(0, 0);

 md3dImmediateContext->OMSetBlendState(0, blendFactor, 0xffffffff);

}

//

// 绘制头骨镜像

//

activeSkullTech->GetDesc( &techDesc );

for(UINT p = 0; p < techDesc.Passes; ++p)

{

 ID3DX11EffectPass\* pass = activeSkullTech->GetPassByIndex( p );

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

 md3dImmediateContext->IASetIndexBuffer(mSkullIB, DXGI\_FORMAT\_R32\_UINT, 0);

 XMVECTOR mirrorPlane = XMVectorSet(0.0f, 0.0f, 1.0f, 0.0f); // xy plane

 XMMATRIX R = XMMatrixReflect(mirrorPlane);

 XMMATRIX world = XMLoadFloat4x4(&mSkullWorld) \* R;

 XMMATRIX worldInvTranspose = MathHelper::InverseTranspose(world);

 XMMATRIX worldViewProj = world\*view\*proj;

 Effects::BasicFX->SetWorld(world);

 Effects::BasicFX->SetWorldInvTranspose(worldInvTranspose);

 Effects::BasicFX->SetWorldViewProj(worldViewProj);

 Effects::BasicFX->SetMaterial(mSkullMat);

 // 保存之前的光照方向，然后反射光照方向

 XMFLOAT3 oldLightDirections[3];

 for(int i = 0; i < 3; ++i)

 {

 oldLightDirections[i] = mDirLights[i].Direction;

 XMVECTOR lightDir = XMLoadFloat3(&mDirLights[i].Direction);

 XMVECTOR reflectedLightDir = XMVector3TransformNormal(lightDir, R);

 XMStoreFloat3(&mDirLights[i].Direction, reflectedLightDir);

 }

 Effects::BasicFX->SetDirLights(mDirLights);

 // 在反射中剔除顺时针绕行的三角形

 md3dImmediateContext->RSSetState(RenderStates::CullClockwiseRS);

 // 根据模板标记绘制镜子可见部分中的头骨镜像

 md3dImmediateContext->OMSetDepthStencilState(RenderStates::DrawReflectionDSS, 1);

 pass->Apply(0, md3dImmediateContext);

 md3dImmediateContext->DrawIndexed(mSkullIndexCount, 0, 0);

 // 恢复默认状态。

 md3dImmediateContext->RSSetState(0);

 md3dImmediateContext->OMSetDepthStencilState(0, 0);

 // 恢复光照方向

 for(int i = 0; i < 3; ++i)

 {

 mDirLights[i].Direction = oldLightDirections[i];

 }

 Effects::BasicFX->SetDirLights(mDirLights);

}

//

// 将镜子绘制到后台缓冲区，需要镜像透明混合，

// 这样才能显示镜子后面的镜像

//

activeTech->GetDesc( &techDesc );

for(UINT p = 0; p < techDesc.Passes; ++p)

{

 ID3DX11EffectPass\* pass = activeTech->GetPassByIndex( p );

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

 // Set per object constants.

 XMMATRIX world = XMLoadFloat4x4(&mRoomWorld);

 XMMATRIX worldInvTranspose = MathHelper::InverseTranspose(world);

 XMMATRIX worldViewProj = world\*view\*proj;

 Effects::BasicFX->SetWorld(world);

 Effects::BasicFX->SetWorldInvTranspose(worldInvTranspose);

 Effects::BasicFX->SetWorldViewProj(worldViewProj);

 Effects::BasicFX->SetTexTransform(XMMatrixIdentity());

 Effects::BasicFX->SetMaterial(mMirrorMat);

 Effects::BasicFX->SetDiffuseMap(mMirrorDiffuseMapSRV);

 // 镜子

 md3dImmediateContext->OMSetBlendState(RenderStates::TransparentBS, blendFactor, 0xffffffff);

 pass->Apply(0, md3dImmediateContext);

 md3dImmediateContext->Draw(6, 24);

}

## 10.4.4 环绕顺序和反射

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。我们通过设置如下光栅器状态来翻转环绕顺序：

//

// CullClockwiseRS

//

// 注意：把朝前的三角形定义为逆时针方向绕行使我们仍然可以进行背面剔除。

// 如果我们关闭背面剔除，就不得不要考虑D3D11\_DEPTH\_STENCIL\_DESC中的BackFace属性了。

D3D11\_RASTERIZER\_DESC cullClockwiseDesc;

ZeroMemory(&cullClockwiseDesc, sizeof(D3D11\_RASTERIZER\_DESC));

cullClockwiseDesc.FillMode = D3D11\_FILL\_SOLID;

cullClockwiseDesc.CullMode = D3D11\_CULL\_BACK;

cullClockwiseDesc.FrontCounterClockwise = true;

cullClockwiseDesc.DepthClipEnable = true;

HR(device->CreateRasterizerState(&cullClockwiseDesc, &CullClockwiseRS));

****

**图10.5 多边形法线不会随着映像而翻转，在反射之后，原来向外的法线会变成向内的法线。**