备忘录_模板

笔记本: DirectX 12
创建时间: 2022/8/22 14:03 **更新时间:** 2022/8/27 14:19
作者: handsome小赞
URL: https://blog.csdn.net/zangle260/article/details/42174607

- 深度/模板缓冲可用格式

> 1. DXGI_FORMAT_D32_FLOAT_S8X24_UINT：此格式用一个 32 位浮点数来指定深度缓冲区，并以另一个 32 位无符号整数来指定模板缓冲区。其中，无符号整数里的 8 位用于将模板缓冲区映射到范围[0, 255]，另外 24 位不可用，仅作填充占位。
> 2. DXGI_FORMAT_D24_UNORM_S8_UINT：指定一个无符号的 24 位深度缓冲区，并将其映射到范围[0, 1]内。另外 8 位（无符号整数）用于令模板缓冲区映射至范围[0, 255]。
>
> 在 D3DApp 应用框架中，当要创建深度缓冲区时就要像下面那样来指定它的格式：
>
> ```
> DXGI_FORMAT mDepthStencilFormat = DXGI_FORMAT_D24_UNORM_S8_UINT;
> depthStencilDesc.Format = mDepthStencilFormat;
> ```

- 资源数据清理

> 在绘制每一帧画面之初，用一下方法来重置模板缓冲区中的局部数据（也可用于清理深度缓冲区）
>
> ```
> void ID3D12GraphicsCommandList::ClearDepthStencilView(
>   D3D12_CPU_DESCRIPTOR_HANDLE DepthStencilView,
>   D3D12_CLEAR_FLAGS ClearFlags,
>   FLOAT Depth,
>   UINT8 Stencil,
>   UINT NumRects,
>   const D3D12_RECT *pRects);
> ```
>
> 1. DepthStencilView：待清理的深度/模板缓冲区视图的描述符。
> 2. ClearFlags：指定为 D3D12_CLEAR_FLAG_DEPTH 仅清理深度缓冲区，指定为 D3D12_CLEAR_FLAG_STENCIL 只清理模板缓冲区，指定为 D3D12_CLEAR_FLAG_DEPTH | D3D12_CLEAR_FLAG_STENCIL 则同时清理这两种缓冲区。
> 3. Depth：将此浮点值设置到深度缓冲区中的每一个像素。此浮点数 x 务必满足 $0 \leqslant x \leqslant 1$。
> 4. Stencil：将此整数值设置到模板缓冲区中的每一个像素。此整数 n 必须满足 $0 \leqslant n \leqslant 255$。
> 5. NumRects：数组 pRects 中所指引的矩形数量。
> 6. pRects：一个 D3D12_RECT 类型数组，它标定了一系列深度/模板缓冲区内要清理的区域。若指定为 nullptr 则清理整个深度/模板缓冲区。
>
> 我们在演示程序中的每一帧都已经调用此方法，形如：
>
> ```
> mCommandList->ClearDepthStencilView(DepthStencilView(),
>   D3D12_CLEAR_FLAG_DEPTH | D3D12_CLEAR_FLAG_STENCIL,
>   1.0f, 0, 0, nullptr);
> ```

- 模板测试

> 通过模板缓冲区来阻止对后台缓冲区特定区域的绘制行为
> 模板测试会随着像素的光栅化过程而执行（即在输出合并阶段进行）

```
if( StencilRef & StencilReadMask ◁ Value & StencilReadMask )
  accept pixel
else
  reject pixel
```

1. 左运算数(left-hand-side, LHS)由程序中定义的**模板参考值**(stencil reference value)StencilRef 与程序内定义的**掩码值**(masking value) StencilReadMask 通过 AND (与) 运算来加以确定。
2. 右运算数 (right-hand-side, RHS) 由正在接受模板测试的特定像素位于模板缓冲区中的对应值 Value 与程序中定义的掩码值 StencilReadMask 经过 AND 计算来加以确定。

可以发现，左运算数与右运算数中的 StencilReadMask 是同一个值。接下来，模板测试用程序中所选定的**比较函数** (comparison function) ◁对左运算数与右运算数进行比对，从而得到布尔类型的返回值。如果测试结果为 true，就将当前受检测的像素写入后台缓冲区 (即假设此像素已通过深度测试)；如果测试结果为 false，则禁止此像素向后台缓冲区的写操作。当然，如果一个像素因模板测试失败而被丢弃，它的相关数据也不会被写入深度缓冲区。

运算符 “◁” 是 D3D12_COMPARISON_FUNC 枚举类型所定义的比较函数之一[①]：

- 描述深度/模板状态 (*深度信息的相关设置、模板信息的相关设置*)

> D3D12_DEPTH_STENCIL_DESC

- 深度信息的相关设置

> 深度信息的相关设置如下。
>
> 1. DepthEnable：设置为 true，则开启深度缓冲；设置为 false，则禁用。当深度测试被禁止时，物体的绘制顺序就变得极为重要，否则位于遮挡物之后的像素片段也将被绘制出来 (回顾 4.1.5 节)。如果深度缓冲被禁用，则深度缓冲区中的元素便不会被更新，DepthWriteMask 项的设置也不会起作用。
> 2. DepthWriteMask：可将此参数设置为 D3D12_DEPTH_WRITE_MASK_ZERO 或者 D3D12_DEPTH_WRITE_MASK_ALL，但两者不能共存。假设 DepthEnable 为 true，若把此参数设置为 D3D12_DEPTH_WRITE_MASK_ZERO 便会禁止对深度缓冲区的写操作，但仍可执行深度测试；若将该项设为 D3D12_DEPTH_WRITE_MASK_ALL，则通过深度测试与模板测试的深度数据将被写入深度缓冲区。这种控制深度数据读写的能力，为某些特效的实现提供了良好的契机。
> 3. DepthFunc：将该参数指定为枚举类型 D3D12_COMPARISON_FUNC 的成员之一，以此来定义深度测试所用的比较函数。此项一般被设为 D3D12_COMPARISON_FUNC_LESS，因而常常执行如 4.1.5 节中所述的深度测试。即，若给定像素片段的深度值小于位于深度缓冲区中对应像素

> 的深度值，则接受该像素片段 (离摄像机近的物体遮挡距摄像机远的物体)。当然，也正如我们所看到的，Direct3D 也允许用户根据需求来自定义深度测试。

- 模板信息的相关设置

模板信息的相关设置如下。

1. StencilEnable：设置为 true，则开启模板测试；设置为 false，则禁用。
2. StencilReadMask：该项用于以下模板测试：

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

若采用该项的默认值，则不会屏蔽任何一位模板值。

```
#define      D3D12_DEFAULT_STENCIL_READ_MASK   ( 0xff )
```

3. StencilWriteMask：当模板缓冲区被更新时，我们可以通过写掩码（write mask）来屏蔽特定位的写入操作。例如，如果我们希望防止前 4 位数据被改写，便可以将写掩码设置为 0x0f。而默认配置是不会屏蔽任何一位模板值的：

```
#define D3D12_DEFAULT_STENCIL_WRITE_MASK ( 0xff )
```

4. FrontFace：填写一个 D3D12_DEPTH_STENCILOP_DESC 结构体实例，以指示根据模板测试与深度测试的结果，应对正面朝向的三角形要进行何种模板运算。
5. BackFace：填写一个 D3D12_DEPTH_STENCILOP_DESC 结构体实例，以指出根据模板测试与深度测试的结果，应对背面朝向的三角形要进行何种模板运算。

```
typedef struct D3D12_DEPTH_STENCILOP_DESC {
 D3D12_STENCIL_OP StencilFailOp;       // 默认值为: D3D12_STENCIL_OP_KEEP
 D3D12_STENCIL_OP StencilDepthFailOp; // 默认值为: D3D12_STENCIL_OP_KEEP
 D3D12_STENCIL_OP StencilPassOp;       // 默认值为: D3D12_STENCIL_OP_KEEP
 D3D12_COMPARISON_FUNC StencilFunc;    // 默认值为: D3D12_COMPARISON_FUNC_ALWAYS
} D3D12_DEPTH_STENCILOP_DESC;
```

1. StencilFailOp：枚举类型 D3D12_STENCIL_OP 中的成员之一，描述了当像素片段在模板测试失败时，应该怎样更新模板缓冲区。
2. StencilDepthFailOp：枚举类型 D3D12_STENCIL_OP 中的成员之一，描述了当像素片段通过模板测试，却在深度测试失败时，应如何更新模板缓冲区。
3. StencilPassOp：枚举类型 D3D12_STENCIL_OP 中的成员之一，描述了当像素片段通过模板测试与深度测试时，该怎样更新模板缓冲区。
4. StencilFunc：枚举类型 D3D12_COMPARISON_FUNC 中的成员之一，定义了模板测试所用的比较函数。

```
typedef
enum D3D12_STENCIL_OP
{
  D3D12_STENCIL_OP_KEEP      = 1,
  D3D12_STENCIL_OP_ZERO      = 2,
  D3D12_STENCIL_OP_REPLACE   = 3,
  D3D12_STENCIL_OP_INCR_SAT  = 4,
  D3D12_STENCIL_OP_DECR_SAT  = 5,
  D3D12_STENCIL_OP_INVERT    = 6,
  D3D12_STENCIL_OP_INCR      = 7,
  D3D12_STENCIL_OP_DECR      = 8
} D3D12_STENCIL_OP;
```

1. D3D12_STENCIL_OP_KEEP：不修改模板缓冲区，即保持当前的数据。
2. D3D12_STENCIL_OP_ZERO：将模板缓冲区中的元素设置为 0。
3. D3D12_STENCIL_OP_REPLACE：将模板缓冲区中的元素替换为用于模板测试的模板参考值（StencilRef）。注意，只有当我们将深度/模板缓冲区状态块绑定到渲染流水线时，才能够设定 StencilRef 值（见 11.3.3 节）。
4. D3D12_STENCIL_OP_INCR_SAT：对模板缓冲区中的元素进行递增（increment）操作。如果递增值超出最大值（例如，8 位模板缓冲区的最大值为 255），则将此模板缓冲区元素限定为最大值。
5. D3D12_STENCIL_OP_DECR_SAT：对模板缓冲区中的元素进行递减（decrement）操作。如果递减值小于 0，则将该模板缓冲区元素限定为 0。
6. D3D12_STENCIL_OP_INVERT：对模板缓冲区中的元素数据按二进制位进行反转。
7. D3D12_STENCIL_OP_INCR：对模板缓冲区中的元素进行递增操作。如果递增值超出最大值（例如，对于 8 位模板缓冲区而言，其最大值为 255），则环回至 0。

8. D3D12_STENCIL_OP_DECR：对模板缓冲区中的元素进行递减操作。如果递减值小于 0，则环回至可取到的最大值。

注意 通过观察能够看出，对正面朝向三角形与背面朝向三角形所进行的模板运算可以是互不相同的。由于在执行背面剔除后背面朝向的多边形并不会得到渲染，所以在这种情况下对 BackFace 的设置便是无足轻重的。然而，我们有时却需要针对特定的图形学算法或透明几何体（例如铁丝网盒，我们能透过它看到其背后的面）的处理来渲染背面朝向的多边形。而此时对 BackFace 的设置则又变得特别重要。

- 创建和绑定深度/模板状态

  - **设置模板参考值**
    `ID3D12GraphicsCommandList::OMSetStencilRef`
  - **绑定**
    将填写完整的 `D3D12_DEPTH_STENCIL_DESC` 赋予 PSO，
    `D3D12_GRAPHICS_PIPLINE_STATE_DESC::DepthStencilState`

- 实现平面镜效果

  - **镜像概述**
    将需要镜像的物体反射到镜面的背面，作为场景中的另一个“实物”而已，并且通过模板冲区技术，只允许在镜子中看见。
    步骤：
    1. 将地板、墙壁以及骷髅头实物照常渲染到后台缓冲区（不包括镜子）。注意，此步骤不修改模板缓冲区。
    2. 清理模板缓冲区，将其整体置零。
    3. 仅将镜面渲染到模板缓冲区中。若要禁止其他颜色数据写入到后台缓冲区，可用下列设置所创建的**混合状态**。
       `D3D12_RENDER_TARGET_BLEND_DESC::RrenderTargetWriteMask = 0;`
       再通过以下配置来禁止向深度缓冲区的写操作：
       `D3D12_DEPTH_STENCIL_DESC::DepthWriteMask = D3D12_DEPTH_WRITE_MASK_ZERO;`
       注意 保证先绘制骷髅头实物，后将镜面渲染至模板缓冲区的顺序的顺序是很重要的。这样，深度测试的失败会令镜面的像素被骷髅头实物的像素所遮挡，也就不必再对模板缓冲区进行二次修改了。
    4. 将骷髅头的镜像渲染至后台缓冲区及模板缓冲区中。
       因为只有通过模板测试的像素才能渲染至后台缓冲区，所以，我们便这样设置：StencilRef 设为 1，模板运算符采用 D3D12_COMPARION_RUNC_EQUAL。这样，只有模板缓冲区中元素值为 1 的骷髅头镜像部分才能得以渲染。由于镜面只有可见部分所对应的模板缓冲区中的元素的值为 1 ，所以仅有这一范围内的骷髅头镜像能被渲染出来。
    5. 最后，我们像往常那样将镜面渲染到后台缓冲区中。但是，为了能“透过”镜面观察骷髅头镜像，我们需要运用透明混合技术来渲染镜面。

- **定义镜像的深度 / 模板状态**
  禁止对渲染目标的写操作（用于在绘制镜面时标记模板缓冲区内镜面部分的像素）

```
CD3DX12_BLEND_DESC mirrorBlendState(D3D12_DEFAULT);

D3D12_DEPTH_STENCIL_DESC mirrorDSS;
mirrorDSS.DepthEnable = true;
mirrorDSS.DepthWriteMask = D3D12_DEPTH_WRITE_MASK_ZERO;
mirrorDSS.DepthFunc = D3D12_COMPARISON_FUNC_LESS;
mirrorDSS.StencilEnable = true;
mirrorDSS.StencilReadMask = 0xff;
mirrorDSS.StencilWriteMask = 0xff;

mirrorDSS.FrontFace.StencilFailOp = D3D12_STENCIL_OP_KEEP;

mirrorDSS.FrontFace.StencilDepthFailop = D3D12_STENCIL_OP_KEEP;
mirrorDSS.FrontFace.StencilPassOp = D3D12_STENCIL_OP_REPLACE;
mirrorDSS.FrontFace.StencilFunc = D3D12_COMPARISON_FUNC_ALWAYS;
// 我们不渲染背面朝向的多边形，因而对这些参数的设置并不关心
mirrorDSS.BackFace.StencilFailOp = D3D12_STENCIL_OP_KEEP;
mirrorDSS.BackFace.StencilDepthFailOp = D3D12_STENCIL_OP_KEEP;
mirrorDSS.BackFace.StencilPassOp = D3D12_STENCIL_OP_REPLACE;
mirrorDSS.BackFace.StencilFunc = D3D12_COMPARISON_FUNC_ALWAYS;

D3D12_GRAPHICS_PIPELINE_STATE_DESC markMirrorsPsoDesc = opaquePsoDesc;
markMirrorsPsoDesc.BlendState = mirrorBlendState;
markMirrorsPsoDesc.DepthStencilState = mirrorDSS;
ThrowIfFailed(md3dDevice->CreateGraphicsPipelineState(
  &markMirrorsPsoDesc,
  IID_PPV_ARGS(&mPSOs["markStencilMirrors"])));
```

  用于渲染模板缓冲区中反射镜像的PSO（用于绘制镜面的可见部分内的不被前侧实物遮挡的骷髅头镜像）
- **绘制场景**
  注意 在绘制镜子范围内的镜像时，必须使用两个单独的渲染过程常量缓冲区
- **绕序与镜像**
  通过 irectX::XMMatrixReflect 得到转换镜像的矩阵，乘上透视投影后的齐次剪裁空间的矩阵即可得到镜像。

```
XMVECTOR mirrorPlane = XMVectorSet(0.0f, 0.0f, 1.0f, 0.0f); // xy plane
XMMATRIX R = XMMatrixReflect(mirrorPlane);
XMStoreFloat4x4(World, skullWorld * R);
```

- **修改绕序**
  注意 实际物体的外向法线在镜像中则变为了内向法线。这就需要修改PSO光栅化属性来改变绕序的约定，使得逆时针绕序的三角形看作是正面。

```
drawReflectionsPsoDesc.RasterizerState.FrontCounterClockwise = true
```

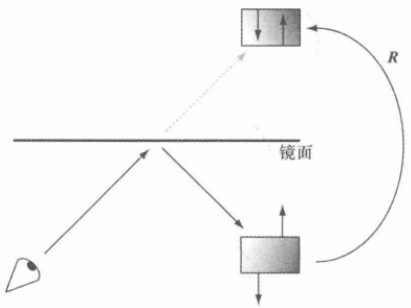

图 11.5　多边形的法线不会随着反射操作而调转过来，这使得镜像的法线都变为内向朝向

- **实现平面阴影**
  1. **方向光阴影**

- 平面方程 p * n + d = 0； p 为平面上一点，n 为法线，d 为原点到平面的距离
- 光线 r(t) = p + tL
- 光线 r(t) 与阴影平面 (n,d) 交点 s

$$s = r(t_s) = p - \frac{n \cdot p + d}{n \cdot L} L$$

- 方向光阴影矩阵

$$s' = [p_x \quad p_y \quad p_z \quad 1] \begin{bmatrix} n \cdot L - L_x n_x & -L_y n_x & -L_z n_x & 0 \\ -L_x n_y & n \cdot L - L_y n_y & -L_z n_y & 0 \\ -L_x n_z & -L_y n_z & n \cdot L - L_z n_z & 0 \\ -L_x d & -L_y d & -L_z d & n \cdot L \end{bmatrix}$$

通过矩阵乘法求出第 i 个投影顶点的坐标 s' 后，需要进行透视除法

最后 s = s'

注意 世界变换后，由于透视除法还没执行，因而几何体的阴影还未被投射到阴影平面上。此时若 Sw = n · L < 0，则 w坐标将变为负值。（在透视投影处理过程中，我们一般是将 z坐标复制到 w坐标，若 w坐标为负值，则表明此点位于视锥体外而应该将其裁剪掉，裁剪操作在透视除法之前的齐次空间内执行）。所以我们用指向无穷远处光源的方向向量 L' = -L 来取代光线方向向量 L

2. **点光阴影**
   - 交点 s

$$s = r(t_s) = p - \frac{n \cdot p + d}{n \cdot (p - L)} (p - L)$$

$$S_{\text{point}} = \begin{bmatrix} n \cdot L + d - L_x n_x & -L_y n_x & -L_z n_x & -n_x \\ -L_x n_y & n \cdot L + d - L_y n_y & -L_z n_y & -n_y \\ -L_x n_z & -L_y n_z & n \cdot L + d - L_z n_z & -n_z \\ -L_x d & -L_y d & -L_z d & n \cdot L \end{bmatrix}$$

3. **通用阴影矩阵**
   1. 如果 Lw = 0，则 L 表示指向无穷远处光源的方向向量（即与方向光光线传播方向相反的向量）
   2. 如果 Lw = 1，则 L 表示点光的位置

   - $$S = \begin{bmatrix} n \cdot L + dL_w - L_x n_x & -L_y n_x & -L_z n_x & -L_w n_x \\ -L_x n_y & n \cdot L + dL_w - L_y n_y & -L_z n_y & -L_w n_y \\ -L_x n_z & -L_y n_z & n \cdot L + dL_w - L_z n_z & -L_w n_z \\ -L_x d & -L_y d & -L_z d & n \cdot L \end{bmatrix}$$

   - 若 w = 0 表示方向光，w = 1 表示点光

```
DirectX::XMMatrixShadow (
  FXMVECTOR  ShadowPlane,
```

```
        FXMVERTOR  LightPosition
    ) ;
```

- **使用模板缓冲区防止双重混合**

  > 避免三角形多次重叠部分混合多次，颜色更暗。

  1. 首先，保证参与渲染阴影的模板缓冲区中的阴影范围像素都已被清理为0。由于“Stencil Demo”演示程序只向地面投射一种阴影，所以这样做也是合情合理的，而且我们仅需修改阴影的模板缓冲区中的像素即可。
  2. 设置模板测试，使之仅接受模板缓冲区中元素为0的像素。如果通过模板测试，则将相应模板缓冲区增值为1。

- **编写阴影部分的代码**
  - 阴影的材质
  - 创建防止双重混合的 深度/模板状态来设置PSO

    > D3D12_DEPTH_STENCIL_DESC

  - 接着用 StencilRef 值为0的阴影PSO来绘制骷髅头阴影
  - 更新阴影的世界矩阵

    > DirectX::XMMatrixShadow

    ```
    XMVECTOR shadowPlane = XMVectorSet(0.0f, 1.0f, 0.0f, 0.0f);
    // xz plane
    XMVECTOR toMainLight = -
    XMLoadFloat3(&mMainPassCB.Lights[0].Direction);
    XMMATRIX S = XMMatrixShadow(shadowPlane, toMainLight);
    XMMATRIX shadowOffsetY = XMMatrixTranslation(0.0f, 0.001f,
    0.0f);
    XMStoreFloat4x4(&mShadowedSkullRitem->World, skullWorld * S
    * shadowOffsetY);
    ```