II. Direct3D Foundations
11. Stenciling

Game Graphic Programming
Kyung Hee University
Software Convergence
Prof. Daeho Lee

Stencil Demo

Depth/Stencil Formats

- The formats used for depth/stencil buffering are as follows:
  - **`DXGI_FORMAT_D32_FLOAT_S8X24_UINT`**
    - This specifies a 32-bit floating-point depth buffer, with 8 bits (unsigned integer) reserved for the stencil buffer mapped to the [0, 255] range and 24 bit not used for padding.
  - **`DXGI_FORMAT_D24_UNORM_S8_UINT`**
    - This specifies an unsigned 24-bit depth buffer mapped to the [0, 1] range with 8 bits (unsigned integer) reserved for the stencil buffer mapped to the [0, 255] range.

  - The **`D3DApp`** framework in this lecture, when we create the depth buffer, we specify:

```
DXGI_FORMAT mDepthStencilFormat
 = DXGI_FORMAT_D24_UNORM_S8_UINT;
depthStencilDesc.Format = mDepthStencilFormat;
```

Depth/Stencil Clearing (1)

- The stencil buffer should be reset to some value at the beginning of each frame. This is done with the following method (which also clears the depth buffer):

```
void ID3D12GraphicsCommandList::ClearDepthStencilView(
  D3D12_CPU_DESCRIPTOR_HANDLE DepthStencilView,
  D3D12_CLEAR_FLAGS ClearFlags,   FLOAT Depth,
  UINT8 Stencil,  UINT NumRects,  const D3D12_RECT *pRects);
```

  - **`DepthStencilView`**: Descriptor to the view of the depth/stencil buffer we want to clear.
  - **`ClearFlags`**: Specify **`D3D12_CLEAR_FLAG_DEPTH`** to clear the depth buffer only; specify **`D3D12_CLEAR_FLAG_STENCIL`** to clear the stencil buffer only; specify **`D3D12_CLEAR_FLAG_DEPTH | D3D12_CLEAR_FLAG_STENCIL`** to clear both.
  - **`Depth`**: The float-value to set each pixel in the depth buffer to; it must be a floating point number x such that $0 \leq x \leq 1$. (**far: 1**)
  - **`Stencil`**: The integer value to set each pixel of the stencil buffer to; it must be an integer n such that $0 \leq n \leq 255$.
  - **`NumRects`**: The number of rectangles in the array **`pRects`** points to.
  - **`pRects`**: An array of **`D3D12_RECT`**s marking rectangular regions on the depth/stencil buffer to clear; specify **`nullptr`** to clear the entire depth/stencil buffer.

Depth/Stencil Clearing (2)

- We have already been calling this method every frame in our demos. For example:

```
mCommandList->ClearDepthStencilView(DepthStencilView(),
        D3D12_CLEAR_FLAG_DEPTH | D3D12_CLEAR_FLAG_STENCIL,
        1.0f, 0, 0, nullptr);
```

The Stencil Test (1)

- We can use the stencil buffer to block rendering to certain areas of the back buffer. The decision to block a particular pixel from being written is decided by the **stencil test**.

```
if(StencilRef&StencilReadMask
    ● StencilBufferValue&StencilReadMask)
// write to back buffer, StencilPassOp
// (assuming the depth test also passes)
     accept pixel
else
     reject pixel
```

●: <, ==, ≤, >, !=, ≥, …

The Stencil Test (2)

```
typedef enum D3D12_COMPARISON_FUNC {
  D3D12_COMPARISON_FUNC_NONE,
  D3D12_COMPARISON_FUNC_NEVER = 1,       // always false
  D3D12_COMPARISON_FUNC_LESS = 2,        // ● → <
  D3D12_COMPARISON_FUNC_EQUAL = 3,       // ● → ==
  D3D12_COMPARISON_FUNC_LESS_EQUAL = 4,        // ● → <=
  D3D12_COMPARISON_FUNC_GREATER = 5,           // ● → >
  D3D12_COMPARISON_FUNC_NOT_EQUAL = 6,         // ● → !=
  D3D12_COMPARISON_FUNC_GREATER_EQUAL = 7,     // ● → >=
  D3D12_COMPARISON_FUNC_ALWAYS = 8       // always true
} ;
```

```
// Depth test example:
D3D12_DEPTH_STENCIL_DESC DSS;
DSS.DepthEnable = true;
DSS.DepthFunc = D3D12_COMPARISON_FUNC_LESS;
```

D3D12_DEPTH_STENCIL_DESC

- **D3D12_DEPTH_STENCIL_DESC**
  - This describes depth-stencil state.

```
typedef struct D3D12_DEPTH_STENCIL_DESC {
  BOOL                       DepthEnable;
  D3D12_DEPTH_WRITE_MASK     DepthWriteMask;
  D3D12_COMPARISON_FUNC      DepthFunc;
  BOOL                       StencilEnable;
  UINT8                      StencilReadMask;
  UINT8                      StencilWriteMask;
  D3D12_DEPTH_STENCILOP_DESC FrontFace;
  D3D12_DEPTH_STENCILOP_DESC BackFace;
} D3D12_DEPTH_STENCIL_DESC;
```

Depth Settings

- **`DepthEnable`**: Specify **`true`** to enable the depth buffering. When depth testing is disabled, the draw order matters, and a pixel fragment will be drawn even if it is behind an occluding object. If depth buffering is disabled, elements in the depth buffer are not updated either, regardless of the **`DepthWriteMask`** setting.
- **`DepthWriteMask`**: This can be either **`D3D12_DEPTH_WRITE_MASK_ZERO`** or **`D3D12_DEPTH_WRITE_MASK_ALL`**, but not both. Assuming **`DepthEnable`** is set to **`true`**, **`D3D12_DEPTH_WRITE_MASK_ZERO`** disables writes to the depth buffer, but depth testing will still occur. **`D3D12_DEPTH_WRITE_MASK_ALL`** enables writes to the depth buffer; new depths will be written provided the depth and stencil tests both pass. The ability to control depth reads and writes becomes necessary for implementing certain special effects.
- **`DepthFunc`**: Specify one of the members of the **`D3D12_COMPARISON_FUNC`** enumerated type to defi ne the depth test comparison function. Usually, this is always **`D3D12_COMPARISON_LESS`** so that the usual depth test is performed. That is, a pixel fragment is accepted provided its depth value is less than the depth of the previous pixel written to the back buffer. But as you can see, Direct3D allows you to customize the depth test if necessary.

Stencil Settings (1)

- **`StencilEnable`**: Specify **`true`** to enable the stencil test.
- **`StencilReadMask`**: This is used in the stencil test:

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

  - The default does not mask any bits:
    - **`#define D3D12_DEFAULT_STENCIL_READ_MASK ( 0xff )`**
- **`StencilWriteMask`**: When the stencil buffer is being updated, we can mask off certain bits from being written to with the write mask.
  - **`The default value does not mask any bits:`**
    - **`#define D3D12_DEFAULT_STENCIL_WRITE_MASK ( 0xff )`**
- **`FrontFace`**: A filled-out **`D3D12_DEPTH_STENCILOP_DESC`** structure indicating how the stencil buffer works for front-facing triangles.
- **`BackFace`**: A filled-out **`D3D12_DEPTH_STENCILOP_DESC`** structure indicating how the stencil buffer works for back-facing triangles.

Stencil Settings (2)

-
  ```
  typedef struct D3D12_DEPTH_STENCILOP_DESC {
  D3D12_STENCIL_OP StencilFailOp;
  D3D12_STENCIL_OP StencilDepthFailOp;
  D3D12_STENCIL_OP StencilPassOp;
  D3D12_COMPARISON_FUNC StencilFunc;
  } D3D12_DEPTH_STENCILOP_DESC;
  ```
  - This describes stencil operations that can be performed based on the results of the stencil test.
  - **StencilFailOp**: A member of the **D3D12_STENCIL_OP** enumerated type describing how the stencil buffer should be updated when the stencil test fails for a pixel fragment.
  - **StencilDepthFailOp**: A member of the **D3D12_STENCIL_OP** enumerated type describing how the stencil buffer should be updated when the stencil test passes but the depth test fails for a pixel fragment.
  - **StencilPassOp**: A member of the **D3D12_STENCIL_OP** enumerated type describing how the stencil buffer should be updated when the stencil test and depth test both pass for a pixel fragment.
  - **StencilFunc**: A member of the **D3D12_COMPARISON_FUNC** enumerated type to defi ne the stencil test comparison function.

Stencil Settings (3)

```
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
} ;
```

- This identifies the stencil operations that can be performed during depth-stencil testing.
- `D3D12_STENCIL_OP_KEEP`: Specifies not to change the stencil buffer.
- `D3D12_STENCIL_OP_ZERO`: Specifies to set the stencil buffer entry to zero.
- `D3D12_STENCIL_OP_REPLACE`: Specifies to replace with the stencil-reference.
- `D3D12_STENCIL_OP_INCR_SAT`: Specifies to increment the stencil buffer entry. (and clamping)
- `D3D12_STENCIL_OP_DECR_SAT`: Specifies to decrement the stencil buffer entry.
- `D3D12_STENCIL_OP_INVERT`: Specifies to invert the bits of the stencil buffer entry.
- `D3D12_STENCIL_OP_INCR`: Specifies to increment the stencil buffer entry.
- `D3D12_STENCIL_OP_DECR`: Specifies to decrement the stencil buffer entry.

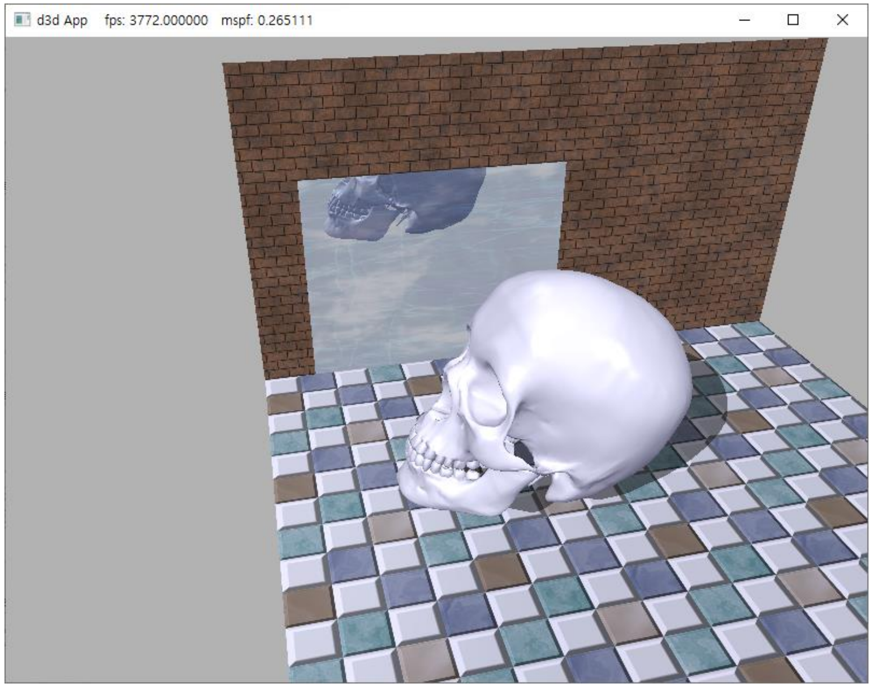
Creating and Binding a Depth/Stencil State

- Creating and binding a depth/stencil state
  - Once we have fully filled out a **`D3D12_DEPTH_STENCIL_DESC`** instance describing our depth/stencil state, we can assign it to the **`D3D12_GRAPHICS_PIPELINE_STATE_DESC::DepthStencilState`** field of a PSO.

- Setting the stencil reference value.

```
ID3D12GraphicsCommandList::OMSetStencilRef(
    [in] UINT StencilRef
);
```

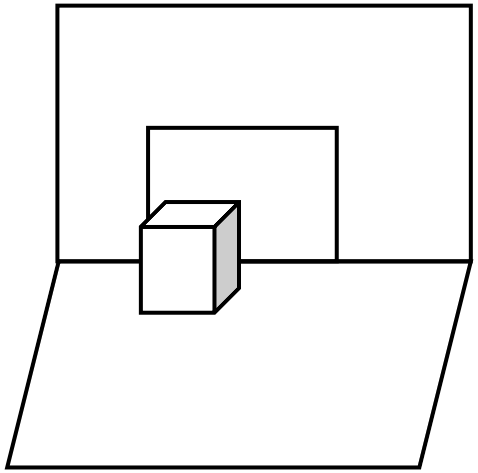
Mirror Overview (1)

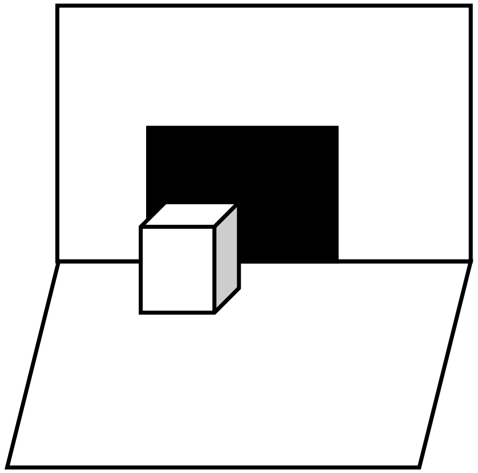


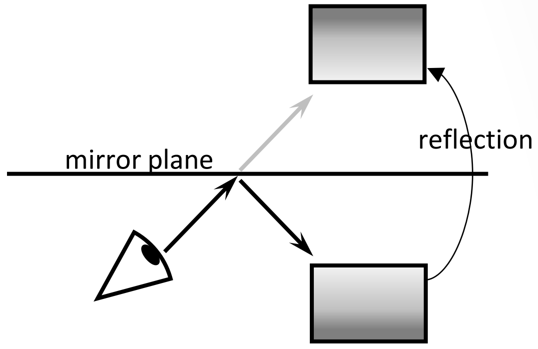
Kyung Hee University
nize@khu.ac.kr

Mirror Overview (2)

- 1. Render the background items.
- 2. Clear the stencil buffer to 0.
- 3. Render the mirror only to the stencil buffer.
- 4. Render the reflected item to the back buffer and stencil buffer. (Render to the back buffer if the stencil test passes.)
- 5. Render the mirror to the back buffer. (with transparency blending)


back buffer


stencil buffer



Defining the Mirror Depth/Stencil States (1)

- We need three additional PSOs.
  - The first is used when drawing the mirror to mark the mirror pixels on the stencil buffer. (1)
  - The second is used to draw the reflected item so that it is only drawn into the visible parts of the mirror. (2)
  - The third is used to draw the mirror to the back buffer. (3)

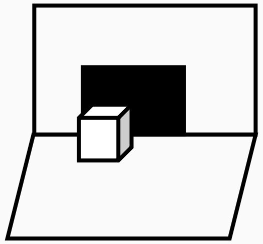
    - **`D3D12_GRAPHICS_PIPELINE_STATE_DESC opaquePsoDesc;`**
    - (1) **`D3D12_GRAPHICS_PIPELINE_STATE_DESC markMirrorsPsoDesc;`**
    - (2) **`D3D12_GRAPHICS_PIPELINE_STATE_DESC drawReflectionsPsoDesc;`**
    - (3) **`D3D12_GRAPHICS_PIPELINE_STATE_DESC transparentPsoDesc;`**

Defining the Mirror Depth/Stencil States (2)

-
```
D3D12_GRAPHICS_PIPELINE_STATE_DESC markMirrorsPsoDesc;
 CD3DX12_BLEND_DESC mirrorBlendState(D3D12_DEFAULT);
 mirrorBlendState.RenderTarget[0].RenderTargetWriteMask = 0;
 // Turn off render target writes
 // default: D3D12_COLOR_WRITE_ENABLE_ALL

 D3D12_DEPTH_STENCIL_DESC mirrorDSS;
 mirrorDSS.DepthEnable = true;
 mirrorDSS.DepthWriteMask = D3D12_DEPTH_WRITE_MASK_ZERO;
 // Disable writing to the depth buffer
 // default: D3D12_DEPTH_WRITE_MASK_ALL
 mirrorDSS.DepthFunc = D3D12_COMPARISON_FUNC_LESS;
 mirrorDSS.StencilEnable = true;
 mirrorDSS.StencilReadMask = 0xff;
 mirrorDSS.StencilWriteMask = 0xff;
```

Defining the Mirror Depth/Stencil States (3)

```
mirrorDSS.FrontFace.StencilFailOp = D3D12_STENCIL_OP_KEEP;
mirrorDSS.FrontFace.StencilDepthFailOp
        = D3D12_STENCIL_OP_KEEP;
mirrorDSS.FrontFace.StencilPassOp = D3D12_STENCIL_OP_REPLACE;

// Replacing with 1 (StencilRef)
//      → set the stencil to 1 on the mirror
// mCommandList->OMSetStencilRef(1);
// mCommandList->SetPipelineState(
//      mPSOs["markStencilMirrors"].Get());
// DrawRenderItems(mCommandList.Get(),
//      mRitemLayer[(int)RenderLayer::Mirrors]);
mirrorDSS.FrontFace.StencilFunc
        = D3D12_COMPARISON_FUNC_ALWAYS;
```



Defining the Mirror Depth/Stencil States (4)

```
        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"])));
```

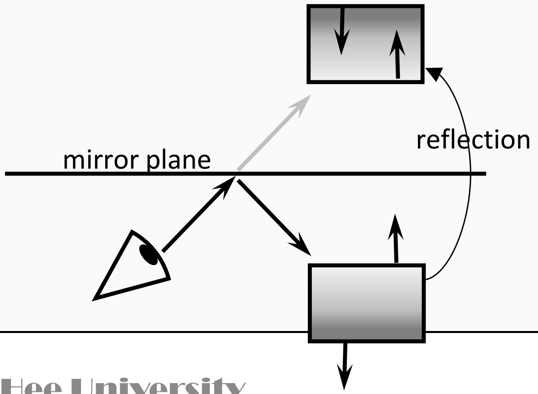
Defining the Mirror Depth/Stencil States (5)

- D3D12_GRAPHICS_PIPELINE_STATE_DESC drawReflectionsPsoDesc;

```
D3D12_DEPTH_STENCIL_DESC reflectionsDSS;
reflectionsDSS.DepthEnable = true;
reflectionsDSS.DepthWriteMask = D3D12_DEPTH_WRITE_MASK_ALL;
reflectionsDSS.DepthFunc = D3D12_COMPARISON_FUNC_LESS;
reflectionsDSS.StencilEnable = true;
reflectionsDSS.StencilReadMask = 0xff;
reflectionsDSS.StencilWriteMask = 0xff;

reflectionsDSS.FrontFace.StencilFailOp = D3D12_STENCIL_OP_KEEP;
reflectionsDSS.FrontFace.StencilDepthFailOp = D3D12_STENCIL_OP_KEEP;
reflectionsDSS.FrontFace.StencilPassOp = D3D12_STENCIL_OP_KEEP;
reflectionsDSS.FrontFace.StencilFunc = D3D12_COMPARISON_FUNC_EQUAL;
```

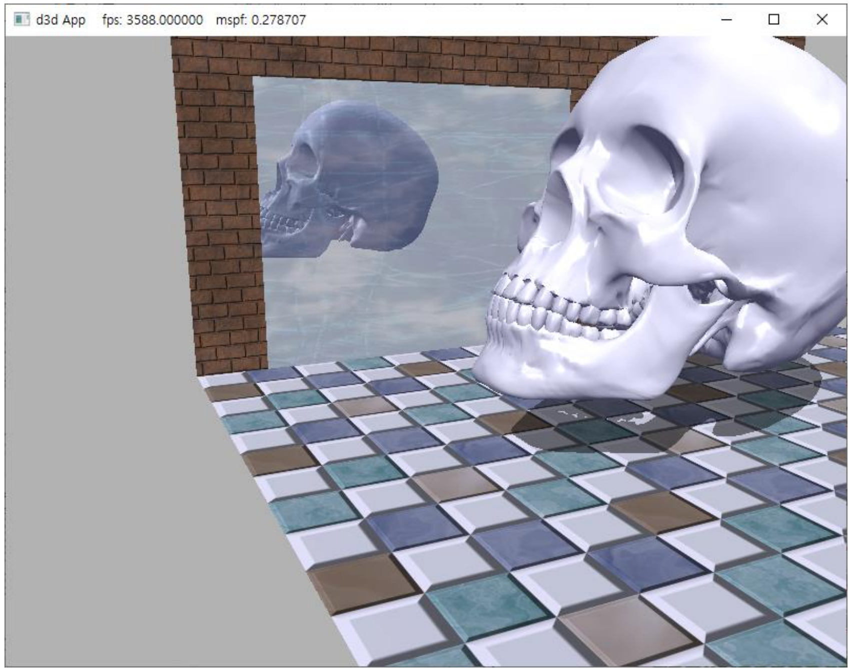
Defining the Mirror Depth/Stencil States (6)

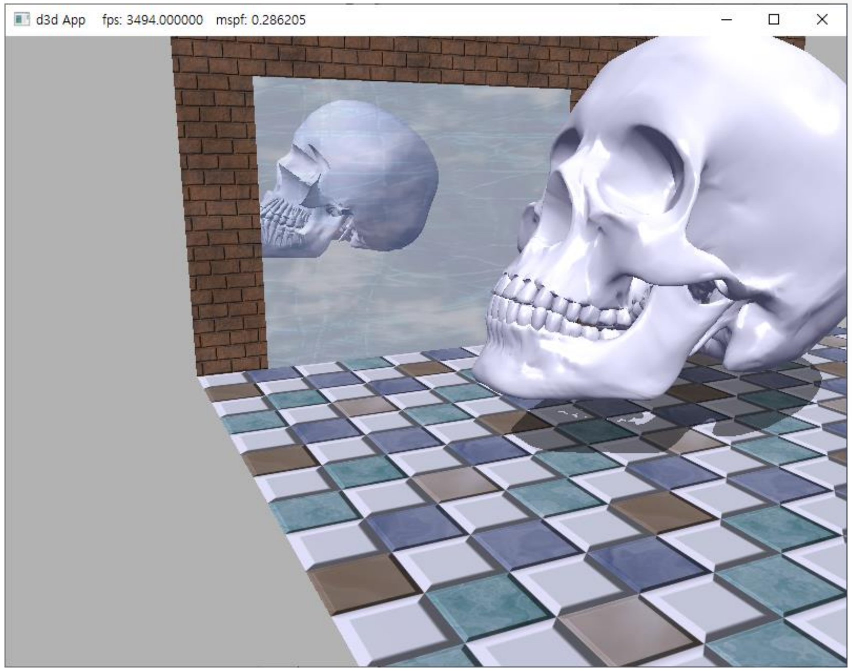
```
reflectionsDSS.BackFace.StencilFailOp = D3D12_STENCIL_OP_KEEP;
reflectionsDSS.BackFace.StencilDepthFailOp = D3D12_STENCIL_OP_KEEP;
reflectionsDSS.BackFace.StencilPassOp = D3D12_STENCIL_OP_KEEP;
reflectionsDSS.BackFace.StencilFunc = D3D12_COMPARISON_FUNC_EQUAL;

D3D12_GRAPHICS_PIPELINE_STATE_DESC drawReflectionsPsoDesc
        = opaquePsoDesc;
drawReflectionsPsoDesc.DepthStencilState = reflectionsDSS;
drawReflectionsPsoDesc.RasterizerState.CullMode = D3D12_CULL_MODE_BACK;
drawReflectionsPsoDesc.RasterizerState.FrontCounterClockwise = true;
ThrowIfFailed(md3dDevice->CreateGraphicsPipelineState(
        &drawReflectionsPsoDesc,
        IID_PPV_ARGS(&mPSOs["drawStencilReflections"])));
```



By default, **Direct3D treats triangles with a clockwise winding order (with respect to the viewer) as front-facing**.

Defining the Mirror Depth/Stencil States (7)





```
// …FrontCounterClockwise = true;
```

Reflected Item

```
XMMATRIX XM_CALLCONV XMMatrixReflect(
  [in] FXMVECTOR ReflectionPlane
) noexcept;
```

- This builds a transformation matrix designed to reflect vectors through a given plane.

```
auto reflectedSkullRitem = std::make_unique<RenderItem>();
*reflectedSkullRitem = *skullRitem;
reflectedSkullRitem->ObjCBIndex = 3;
mReflectedSkullRitem = reflectedSkullRitem.get();
mRitemLayer[(int)RenderLayer::Reflected].push_back(
   reflectedSkullRitem.get());

XMVECTOR mirrorPlane = XMVectorSet(0.0f, 0.0f, 1.0f, 0.0f);
XMMATRIX R = XMMatrixReflect(mirrorPlane);
XMStoreFloat4x4(&mReflectedSkullRitem->World,skullWorld*R);
```

Drawing the Scene (1)

```
void StencilApp::Draw(const GameTimer& gt)
    // Draw opaque items--floors, walls, skull.
    auto passCB = mCurrFrameResource->PassCB->Resource();
    mCommandList->SetGraphicsRootConstantBufferView(2,
        passCB->GetGPUVirtualAddress());
    DrawRenderItems(mCommandList.Get(),
    mRitemLayer[(int)RenderLayer::Opaque]);

    // Mark the visible mirror pixels in the stencil buffer with the value 1
    mCommandList->OMSetStencilRef(1);
    mCommandList->SetPipelineState(mPSOs["markStencilMirrors"].Get());
    DrawRenderItems(mCommandList.Get(),
    mRitemLayer[(int)RenderLayer::Mirrors]);
```

Drawing the Scene (2)

```
    // Draw the reflection into the mirror only (only for pixels where
    //  the stencil buffer is 1).
    // Note that we must supply a different per-pass constant buffer--one
    //  with the lights reflected.
    mCommandList->SetGraphicsRootConstantBufferView(2,
        passCB->GetGPUVirtualAddress() + 1 * passCBByteSize);
    mCommandList->SetPipelineState(mPSOs["drawStencilReflections"].Get());
    DrawRenderItems(mCommandList.Get(),
    mRitemLayer[(int)RenderLayer::Reflected]);
```

```
PassConstants StencilApp::mMainPassCB;
PassConstants StencilApp::mReflectedPassCB;
void UpdateMainPassCB(const GameTimer& gt);
void UpdateReflectedPassCB(const GameTimer& gt);

void StencilApp::BuildFrameResources() {
   for(int i = 0; i < gNumFrameResources; ++i) {
      mFrameResources.push_back(std::make_unique<FrameResource>(md3dDevice.Get(),
         2, (UINT)mAllRitems.size(), (UINT)mMaterials.size()));
   } // passCount: 2
}
```

Drawing the Scene (3)

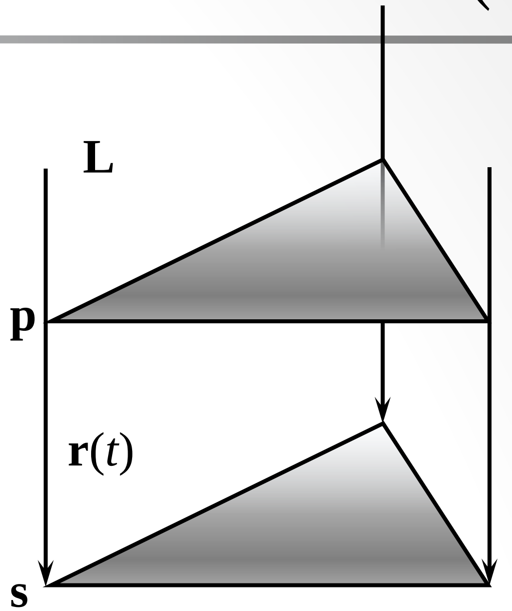
```
    // Restore main pass constants and stencil ref.
    mCommandList->SetGraphicsRootConstantBufferView(2,
        passCB->GetGPUVirtualAddress());
    mCommandList->OMSetStencilRef(0);

    // Draw mirror with transparency so reflection blends through.
    mCommandList->SetPipelineState(mPSOs["transparent"].Get());
    DrawRenderItems(mCommandList.Get(),
        mRitemLayer[(int)RenderLayer::Transparent]);
```

Drawing the Scene (4)

```
PassConstants StencilApp::mMainPassCB;
PassConstants StencilApp::mReflectedPassCB;
void StencilApp::UpdateReflectedPassCB(const GameTimer& gt) {
  mReflectedPassCB = mMainPassCB;
  XMVECTOR mirrorPlane = XMVectorSet(0.0f, 0.0f, 1.0f, 0.0f); // xy plane
  XMMATRIX R = XMMatrixReflect(mirrorPlane);
  XMStoreFloat4x4(&mReflectedSkullRitem->World, skullWorld * R);

  for(int i = 0; i < 3; ++i) { // # of lights
    XMVECTOR lightDir = XMLoadFloat3(&mMainPassCB.Lights[i].Direction);
    XMVECTOR reflectedLightDir = XMVector3TransformNormal(lightDir, R);
       // Reflect the lighting.
    XMStoreFloat3(&mReflectedPassCB.Lights[i].Direction, reflectedLightDir);
  }

  // Reflected pass stored in index 1
  auto currPassCB = mCurrFrameResource->PassCB.get();
  currPassCB->CopyData(1, mReflectedPassCB);
}
```

Parallel Light Shadows (1)

- Shadow projection
  - Projection plane equation: $\mathbf{n}\cdot\mathbf{p}(t)+d=0$



$$\mathbf{n}\cdot\mathbf{p}(t)+d=0$$
$$\mathbf{n}\cdot(\mathbf{p}+t\mathbf{L})+d=0$$
$$\mathbf{n}\cdot\mathbf{p}+t\mathbf{n}\cdot\mathbf{L}+d=0$$
$$t\mathbf{n}\cdot\mathbf{L}=-\mathbf{n}\cdot\mathbf{p}-d$$
$$t=-\frac{\mathbf{n}\cdot\mathbf{p}+d}{\mathbf{n}\cdot\mathbf{L}}$$

$$\mathbf{r}(t)=\mathbf{p}+t\mathbf{L}$$

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

Parallel Light Shadows (2)

- Shadow projection

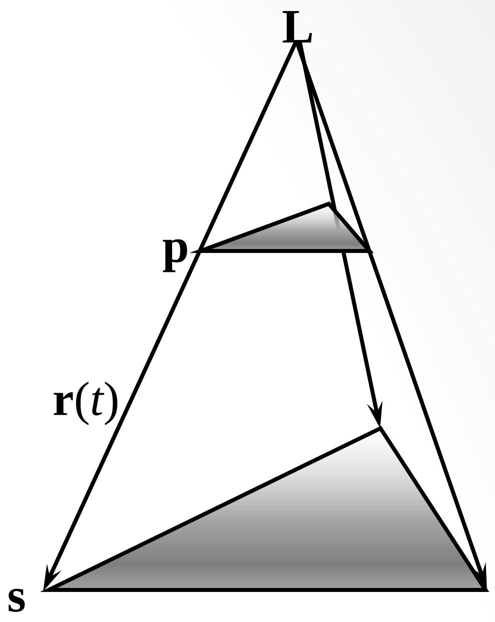
$$\mathbf{s}' = \begin{pmatrix} p_x & p_y & p_z & 1 \end{pmatrix} \begin{pmatrix} \mathbf{n}\cdot\mathbf{L} - L_x n_x & -L_y n_x & -L_z n_x & 0 \\ -L_x n_y & \mathbf{n}\cdot\mathbf{L} - L_y n_y & -L_z n_y & 0 \\ -L_x n_z & -L_y n_z & \mathbf{n}\cdot\mathbf{L} - L_z n_z & 0 \\ -L_x d & -L_y d & -L_z d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

Point Light Shadows

- Shadow projection

$$\mathbf{r}(t) = \mathbf{p} + t(\mathbf{p} - \mathbf{L})$$

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



$$\mathbf{s}' = \begin{pmatrix} p_x & p_y & p_z & 1 \end{pmatrix} \begin{pmatrix} \mathbf{n}\cdot\mathbf{L} + d - L_x n_x & -L_y n_x & -L_z n_x & -n_x \\ -L_x n_y & \mathbf{n}\cdot\mathbf{L} + d - L_y n_y & -L_z n_y & -n_y \\ -L_x n_z & -L_y n_z & \mathbf{n}\cdot\mathbf{L} + d - L_z n_z & -n_z \\ -L_x d & -L_y d & -L_z d & \mathbf{n}\cdot\mathbf{L} \end{pmatrix}$$

General Shadow Matrix

- L_w : 0 → parallel light
- L_w: 1 → point light

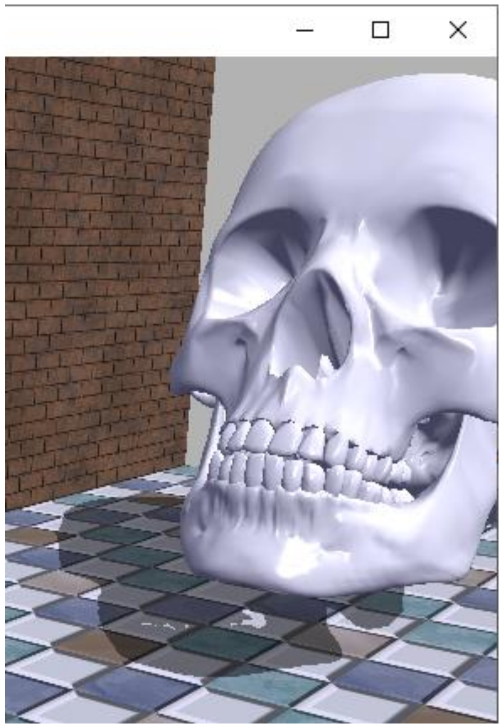
$$\mathbf{S} = \begin{pmatrix} \mathbf{n}\cdot\mathbf{L} + dL_w - L_x n_x & -L_y n_x & -L_z n_x & -L_w n_x \\ -L_x n_y & \mathbf{n}\cdot\mathbf{L} + dL_w - L_y n_y & -L_z n_y & -L_w n_y \\ -L_x n_z & -L_y n_z & \mathbf{n}\cdot\mathbf{L} + dL_w - L_z n_z & -L_w n_z \\ -L_x d & -L_y d & -L_z d & \mathbf{n}\cdot\mathbf{L}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

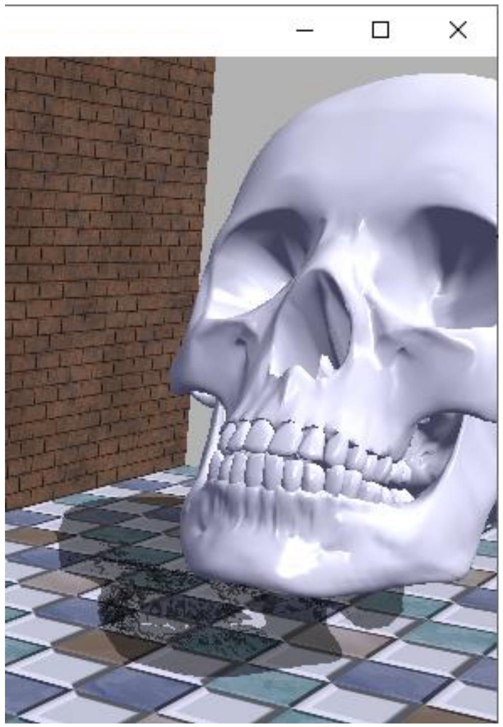
```
XMMATRIX XM_CALLCONV XMMatrixShadow(
  [in] FXMVECTOR ShadowPlane,
  [in] FXMVECTOR LightPosition
) noexcept;
```

- This builds a transformation matrix that flattens geometry into a plane
- **`LightPosition`**: 4D vector describing the light's position. If the light's w-component is 0.0f, the ray from the origin to the light represents a directional light. If it is 1.0f, the light is a point light.

Preventing Double Blending (1)

- Using the stencil buffer to prevent double blending
  - When we flatten out the geometry of an object onto the plane to describe its shadow, it is possible (and in fact likely) that two or more of the flattened triangles will overlap.





Preventing Double Blending (2)

- Using the Stencil Buffer to Prevent Double Blending
  - 1. Assume the stencil buffer pixels where the shadow will be rendered have been cleared to 0. This is true in our mirror demo because we are only casting a shadow onto the ground plane, and we only modified the mirror stencil buffer pixels.
  - 2. Set the stencil test to only accept pixels if the stencil buffer has an entry of 0. If the stencil test passes, then we increment the stencil buffer value to 1.

Shadow Code (1)

```
// shadow transparent & black material
auto shadowMat = std::make_unique<Material>();
shadowMat->Name = "shadowMat";
shadowMat->MatCBIndex = 4;
shadowMat->DiffuseSrvHeapIndex = 3;
shadowMat->DiffuseAlbedo = XMFLOAT4(0.0f, 0.0f, 0.0f, 0.5f);
shadowMat->FresnelR0 = XMFLOAT3(0.001f, 0.001f, 0.001f);
shadowMat->Roughness = 0.0f;
```

Shadow Code (2)

- **D3D12_GRAPHICS_PIPELINE_STATE_DESC shadowPsoDesc;**

```
D3D12_DEPTH_STENCIL_DESC shadowDSS =
CD3DX12_DEPTH_STENCIL_DESC(D3D12_DEFAULT);
shadowDSS.DepthEnable = true;
shadowDSS.DepthWriteMask = D3D12_DEPTH_WRITE_MASK_ALL;
shadowDSS.DepthFunc = D3D12_COMPARISON_FUNC_LESS;
shadowDSS.StencilEnable = true;
shadowDSS.StencilReadMask = 0xff;
shadowDSS.StencilWriteMask = 0xff;

shadowDSS.FrontFace.StencilFailOp = D3D12_STENCIL_OP_KEEP;
shadowDSS.FrontFace.StencilDepthFailOp
       = D3D12_STENCIL_OP_KEEP;
shadowDSS.FrontFace.StencilPassOp = D3D12_STENCIL_OP_INCR;
shadowDSS.FrontFace.StencilFunc = D3D12_COMPARISON_FUNC_EQUAL;
                      // ref(0) == stencilValue(0)
                      //     → stencilValue(1)
```

Shadow Code (3)

```
// We are not rendering backfacing polygons,
//   so these settings do not matter.
shadowDSS.BackFace.StencilFailOp = D3D12_STENCIL_OP_KEEP;
shadowDSS.BackFace.StencilDepthFailOp = D3D12_STENCIL_OP_KEEP;
shadowDSS.BackFace.StencilPassOp = D3D12_STENCIL_OP_INCR;
shadowDSS.BackFace.StencilFunc = D3D12_COMPARISON_FUNC_EQUAL;

D3D12_GRAPHICS_PIPELINE_STATE_DESC shadowPsoDesc
       = transparentPsoDesc;
shadowPsoDesc.DepthStencilState = shadowDSS;
ThrowIfFailed(md3dDevice->CreateGraphicsPipelineState(
       &shadowPsoDesc, IID_PPV_ARGS(&mPSOs["shadow"])));
```

Shadow Code (4)

```
// Update shadow world matrix.
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);
```