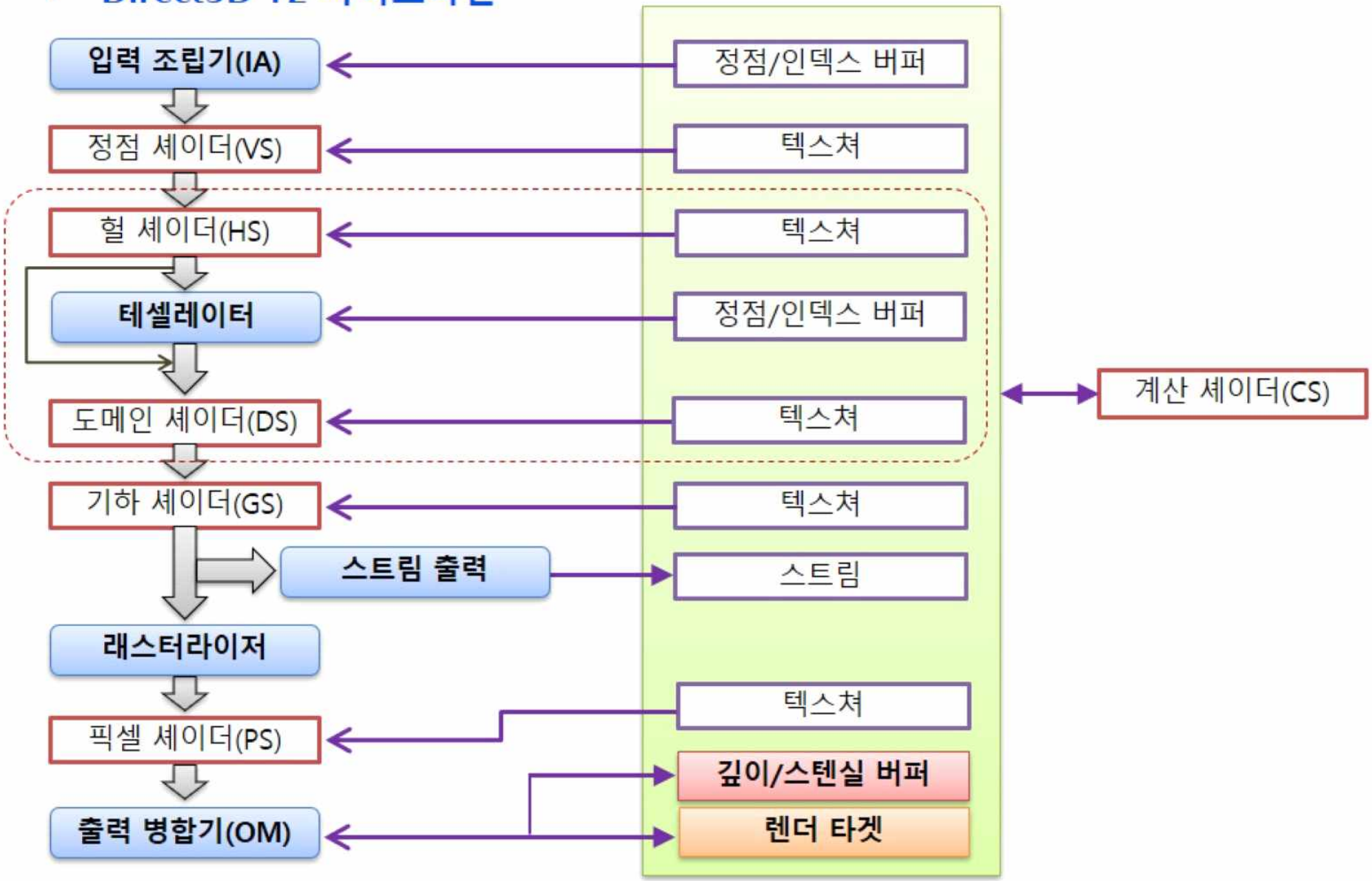
Game Programming with DirectX

Direct3D Graphics Pipeline
(Shader Samples)
Blurring

Direct3D 파이프라인

- Direct3D 12 파이프라인



블러링(Blurring)

- 블러링(Blurring)

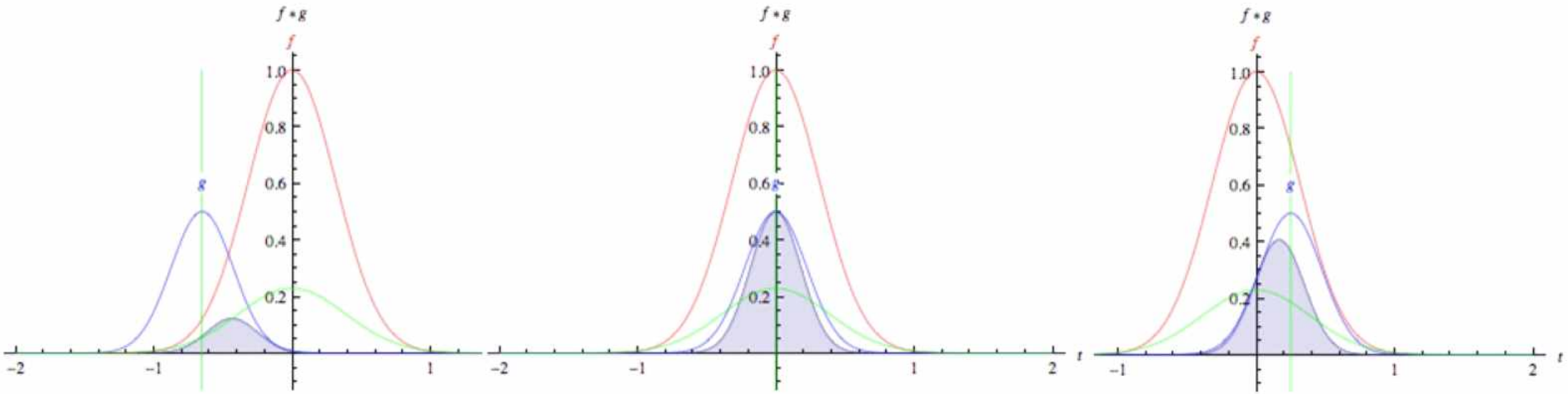
블러링(Blurring)

- **컨벌루션(Convolution)**
  하나의 함수가 다른 함수 위에서 움직이면서 겹치는 양을 표현하는 적분
  - 유한 범위 $[0, t]$에서 함수 f와 g 의 컨벌루션

$$[f * g](t) = \int_0^t f(\tau)\, g(t-\tau)\, d\tau$$

  - 무한 범위 $[-\infty, +\infty]$에서 함수 f와 g의 컨벌루션

$$f * g = \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau = \int_{-\infty}^{\infty} g(\tau)\, f(t-\tau)\, d\tau$$



  - 디지털 이미지 컨벌루션

$$y(i) = \sum_{k=1}^{n} f(i-k)x(k) \qquad (f * g)(i,j) = \sum_{r=-m}^{m} \sum_{c=-n}^{n} f(r,c)\, g(i+r, j+c)$$

블러링(Blurring)

- **컨벌루션(Convolution)**
  - 컨벌루션 매스크(커널: Kernel)
    1x3, 3x1, 3x3, 5x5, ...

k_{00}	k_{01}	k_{02}
k_{10}	k_{11}	k_{12}
k_{20}	k_{21}	k_{22}

p_0	p_1	p_2
p_3	p_{ij}	p_4
p_5	p_6	p_7

$$R_{ij} = k_{00}p_0 + k_{01}p_1 + k_{02}p_2 + k_{10}p_3 + k_{11}p_{ij} + k_{12}p_4 + k_{20}p_5 + k_{21}p_6 + k_{22}p_7$$

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

스무딩(Smooth)

0.0	0.2	0.0
0.2	0.2	0.2
0.0	0.2	0.0

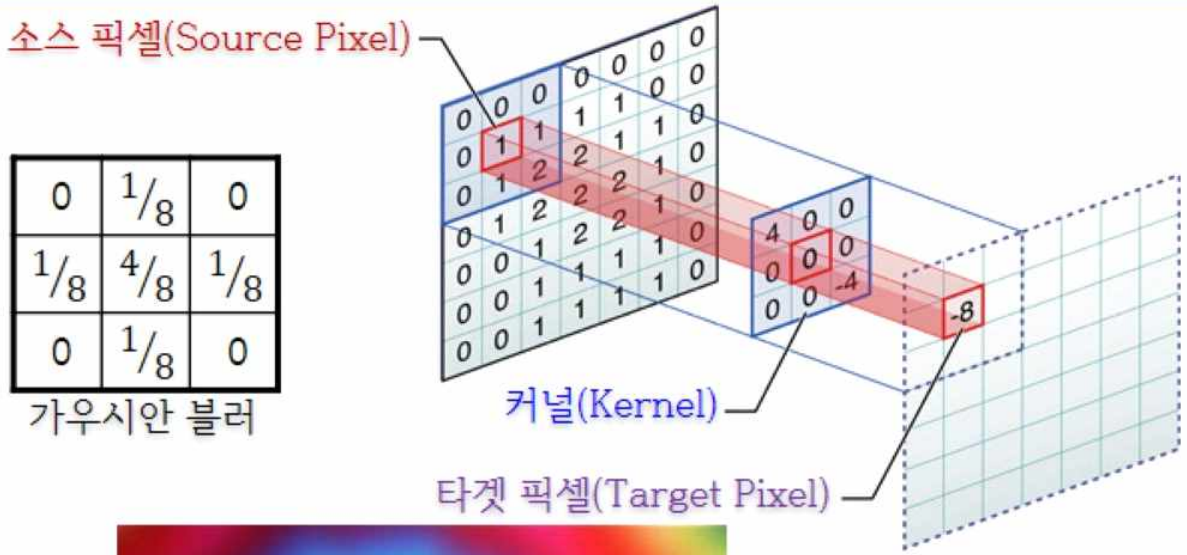
블러 매스크

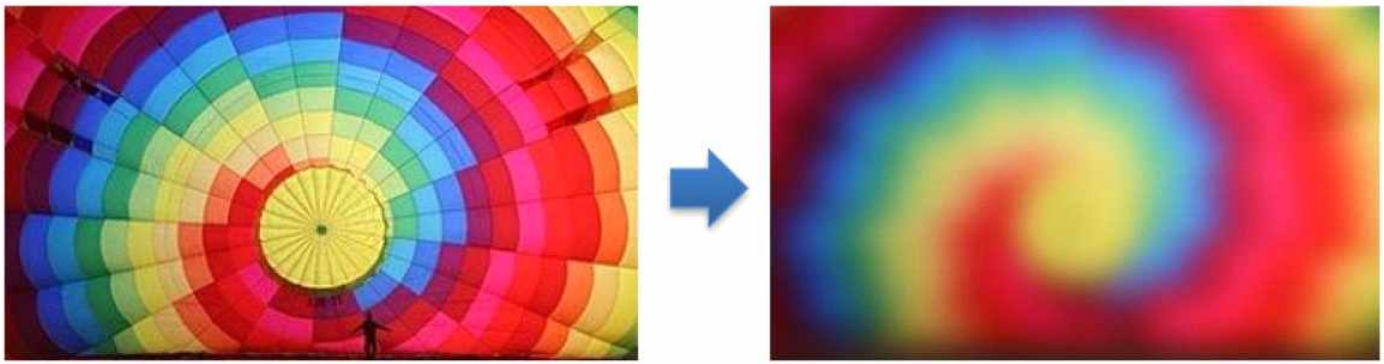
1/3	0.0	0.0
0.0	1/3	0.0
0.0	0.0	1/3

모션 블러 매스크

0	1/8	0
1/8	4/8	1/8
0	1/8	0

가우시안 블러



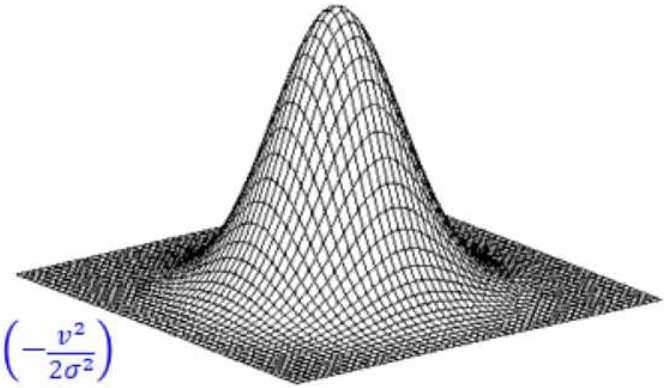


블러링(Blurring)

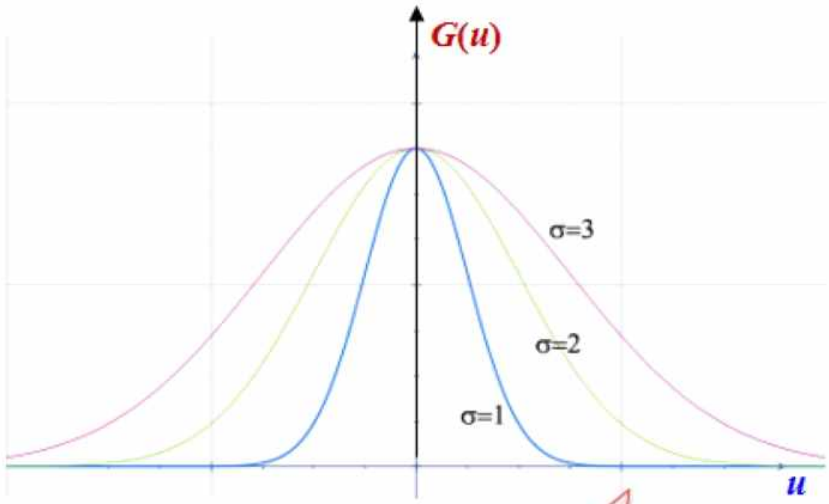
- **블러링(Blurring)**
  - 가우시안(Gaussian) 매스크
    가우시안 함수(Gaussian Function)

$$G(u) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{u^2}{2\sigma^2}\right)$$

$$G(u,v) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{u^2+v^2}{2\sigma^2}\right) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{u^2}{2\sigma^2}\right)\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{v^2}{2\sigma^2}\right)$$



$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$



$$Blur_G(I,u,v) = Blur_G(Blur_G(I,v),u)$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 4 & 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 4 \\ 1 \end{bmatrix}$$

$$G(u,v) = G(u)\,G(v)$$

블러링(Blurring)

- 블러링(Blurring)

```
[numthreads(256, 1, 1)]
void HorzBlurCS(int3 vGroupThreadID : SV_GroupThreadID, int3 vDispatchThreadID : SV_DispatchThreadID)
{
    if (vGroupThreadID.x < 5)
    {
        int x = max(vDispatchThreadID.x - 5, 0);
        gTextureCache[vGroupThreadID.x] = gtxtInput[int2(x, vDispatchThreadID.y)];
    }
    else if (vGroupThreadID.x >= 256-5)
    {
        int x = min(vDispatchThreadID.x + 5, gtxtInput.Length.x-1);
        gTextureCache[vGroupThreadID.x+2*5] = gtxtInput[int2(x, vDispatchThreadID.y)];
    }
    gTextureCache[vGroupThreadID.x+5] = gtxtInput[min(vDispatchThreadID.xy, gtxtInput.Length.xy-1)];

    GroupMemoryBarrierWithGroupSync();

    float4 cBlurredColor = float4(0, 0, 0, 0);
    for (int i = -5; i <= 5; i++)
    {
        int k = vGroupThreadID.x + 5 + i;
        cBlurredColor += gfWeights[i+5] * gTextureCache[k];
    }

    gtxtRWOutput[vDispatchThreadID.xy] = cBlurredColor;
}
```

```
Texture2D<float4> gtxtInput;
RWTexture2D<float4> gtxtRWOutput;

groupshared float4 gTextureCache[(256+2*5)];
```

```
static float gfWeights[11] = { 0.05f, 0.05f, 0.1f, 0.1f, 0.1f, 0.2f, 0.1f, 0.1f, 0.1f, 0.05f, 0.05f };
```

블러링(Blurring)

- 블러링(Blurring)

```
[numthreads(1, 256, 1)]
void VertBlurCS(int3 vGroupThreadID : SV_GroupThreadID, int3 vDispatchThreadID : SV_DispatchThreadID)
{
   if (vGroupThreadID.y < 5)
   {
      int y = max(vDispatchThreadID.y - 5, 0);
      gTextureCache[vGroupThreadID.y] = gtxtInput[int2(vDispatchThreadID.x, y)];
   }
   else if (vGroupThreadID.y >= 256-5)
   {
      int y = min(vDispatchThreadID.y + 5, gtxtInput.Length.y-1);
      gTextureCache[vGroupThreadID.y+2*5] = gtxtInput[int2(vDispatchThreadID.x, y)];
   }
   gTextureCache[vGroupThreadID.y+5] = gtxtInput[min(vDispatchThreadID.xy, gtxtInput.Length.xy-1)];

   GroupMemoryBarrierWithGroupSync();

   float4 cBlurredColor = float4(0, 0, 0, 0);
   for (int i = -5; i <= 5; i++)
   {
      int k = vGroupThreadID.y + 5 + i;
      cBlurredColor += gfWeights[i+5] * gTextureCache[k];
   }

   gtxtRWOutput[vDispatchThreadID.xy] = cBlurredColor;
}
```

```
Texture2D<float4> gtxtInput;
RWTexture2D<float4> gtxtRWOutput;

groupshared float4 gTextureCache[(256+2*5)];
```

```
static float gfWeights[11] = { 0.05f, 0.05f, 0.1f, 0.1f, 0.1f, 0.2f, 0.1f, 0.1f, 0.1f, 0.05f, 0.05f };
```

Direct3D 파이프라인

- **텍스쳐에 렌더링(Render to Texture)**
  - **후면 버퍼(Back Buffer)**
    DXGI가 생성한 2D 텍스쳐(버퍼)
    IDXGISwapChain::Present() 함수로 화면으로 프리젠트됨
    후면 버퍼에 렌더링하기 위하여 후면 버퍼를 사용하여 렌더 타겟 뷰를 생성/연결
  - **다중 렌더 타겟(MRT: Multiple Render Targets)**
    Direct3D 디바이스에 새로운 2D 텍스쳐를 생성하여 렌더 타겟으로 지정할 수 있음
    전체 8개의 렌더 타겟을 지정할 수 있음(8개의 렌더 타겟 슬롯)
    모든 렌더 타겟은 크기(가로, 세로, 깊이 등)가 같아야 함
    각 렌더 타겟은 다른 형식(DXGI Format)을 가져도 됨
    하나의 렌더 타겟 뷰는 여러 렌더 타겟 슬롯에 연결될 수 없음
- **시스템-값 시멘틱(System-Value Semantics)**

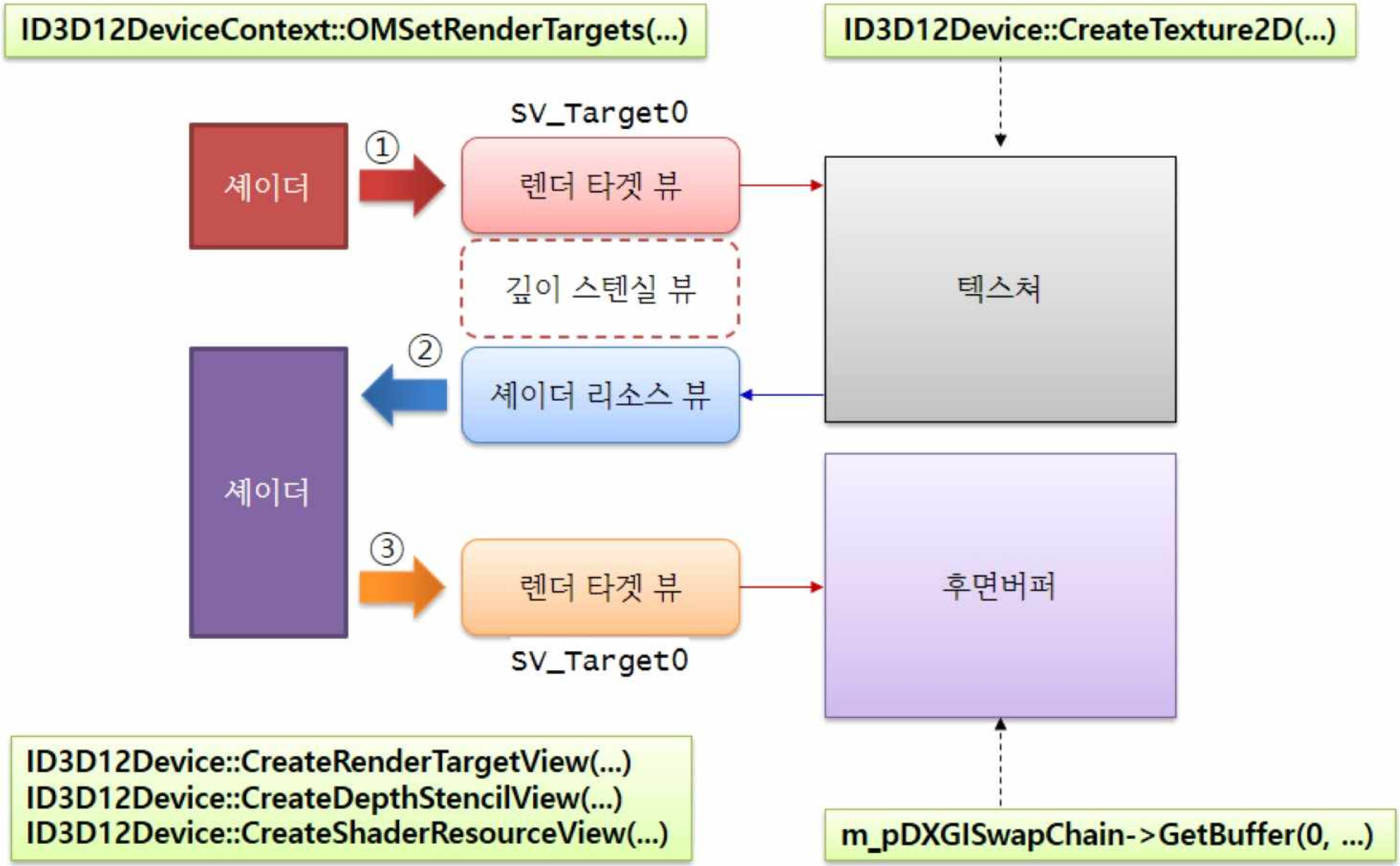
SV_Target[n]	렌더 타겟으로 사용할 출력을 지정, 픽셀 셰이더의 출력으로 사용, n=0~7	float

```
struct PS_OUTPUT
{
    float4 color : SV_Target0;
    float4 normalDepth : SV_Target1; //RGB:Normal, A:Depth
};
```

```
PS_OUTPUT PS(VS_INPUT input)
{
    PS_OUT output = (PS_OUT)0;
    output.color = input.color;
    output.normalDepth = float4(input.normal.xyz, input.position.w);
    return(output);
}
```

Direct3D 파이프라인

- 렌더 타겟(Render Target)



백열광 효과(Glow Effect)

- **백열광 효과(Glow Effect)**
  - 씬을 블러링한 결과와 씬을 결합하여 렌더링
    ① 씬을 텍스쳐로 렌더링
    ② 화면 크기의 사각형을 렌더링
    렌더 타겟을 리소스로 설정(gtxtRenderTarget)
    렌더 타겟에서 밝은 부분을 추출하여 렌더링(gtxtPowered)
    ③ 화면 크기의 사각형을 렌더링
    렌더 타겟을 리소스로 설정(gtxtPowered)
    렌더링 결과를 블러링(gtxtBlurred)
    ④ 블러링 결과(gtxtBlurred)와 씬의 렌더링 결과(gtxtRenderTarget)를 합성

```
Texture2D<float4> gtxtRenderTarget : register(t0);

float4 PSPower(VS_OUTPUT input) {
   float4 cColor = gtxtRenderTarget.Sample(gssDefault, input.uv);
   cColor = pow(cColor, gfPower);
   return(cColor);
}
```

```
static float gfPower = 4.0f;
```

```
Texture2D<float4> gtxtBlurred : register(t3);

float4 PS(VS_OUTPUT input) {
   float4 cBlurred = gtxtBlurred.Sample(gssDefault, input.uv);
   float4 cColor = gtxtRenderTarget.Sample(gssDefault, input.uv);
   return(cBlurred + cColor);
}
```