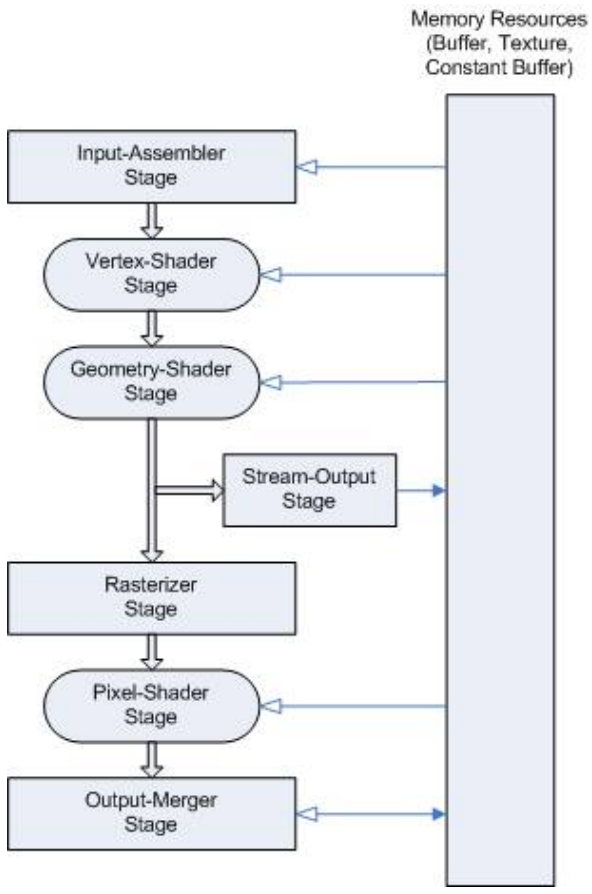
Rendering Pipeline

전체적인 파이프라인 순서



Input Assembler 스테이지

해당 스테이지에서는 버퍼의 기본 데이터(점, 선, 삼각형)을 읽고 다른 파이프 라인에서 데이터를 사용할 수 있게 기본 데이터로 조립합니다.

기본 데이터는 ID3D11DeviceContext::IASetPrimitiveTopology 함수를 통해 형식을 지정할 수 있습니다.

기본 데이터로 조립하는 동안 IA는 시스템 생성 값을 연결하여 셰이더를 보다 효율적으로 만듭니다.
시스템 생성값은 semantics라고 불리는 텍스트 문자열입니다.

각 세 개의 셰이더 단계(VS, GS, PS)는 시스템 생성 값을 사용합니다.

IA 시작하기

1. input buffer 만들기

   input buffer에는 두가지 타입인 vertex buffer과 index buffer가 있습니다.
   vertext buffer는 IA에게 사용할 vertex 데이터를 제공합니다.

```
// m_pDevice는 ID3D11Device*

struct Vertex // vertex 데이터
{
    Vector3 position;
};
```

```
Vertex vertices[] = // buffer 생성하기
{
    Vector3(-0.5,-0.5,0.5), // v0
    Vector3(0,0.5,0.5),     // v1
    Vector3(0.5,-0.5,0.5),  // v2
};

// 버퍼 내용 설정하기
D3D11_BUFFER_DESC vbDesc = {};
m_VertexCount = ARRAYSIZE(vertices);
vbDesc.ByteWidth = sizeof(Vertex) * m_VertexCount;  // 버텍스 버퍼의 크기 (Byte).
vbDesc.CPUAccessFlags = 0;                  // 0 == CPU 엑세스 필요하지 않음
vbDesc.BindFlags = D3D11_BIND_VERTEX_BUFFER;    // 정적 버퍼로 사용.
vbDesc.MiscFlags = 0;
vbDesc.Usage = D3D11_USAGE_DEFAULT;     // CPU 접근 불가, GPU에서 읽기/쓰기로 가능한 버퍼로 생성

// 정점 버퍼 생성
D3D11_SUBRESOURCE_DATA vbData = {};
vbData.pSysMem = vertices; // 버퍼를 생성할 때 복사할 데이터의 주소 설정
m_pDevice->CreateBuffer(&vbDesc, &vbData, &m_pVertexBuffer);


// IA에 바인딩하기
m_pDeviceContext->IASetVertexBuffers(0, 1, &m_pVertexBuffer, &m_VertexBufferStride, &m_VertexBufferOffse
t);
```

index buffer는 vertex buffer에서 각 각 vertex의 인덱스를 제공합니다. (index buffer는 선택적으로 추가할 수 있습니다.)

```
unsigned int indices[] = { 0, 1, 2 };

// 버퍼 내용 설정하기
D3D11_BUFFER_DESC psDesc = {};
m_IndexCount = ARRAYSIZE(indices);
psDesc.ByteWidth = sizeof(unsigned int) * m_IndexCount;
psDesc.CPUAccessFlags = 0;
psDesc.BindFlags = D3D11_BIND_INDEX_BUFFER;
psDesc.MiscFlags = 0;
psDesc.Usage = D3D11_USAGE_DEFAULT;

// 인덱스 버퍼 생성
m_pDevice->CreateBuffer(&psDesc, &psData, &m_pIndexBuffer);


// IA에 바인딩하기
m_pDeviceContext->IASetIndexBuffer(m_pIndexBuffer, DXGI_FORMAT_R32_UINT, 0);
```

2. input-Layout 객체 만들기
   vectex shader가 사용할 vertex의 각 요소를 알려주는 역할을 합니다.

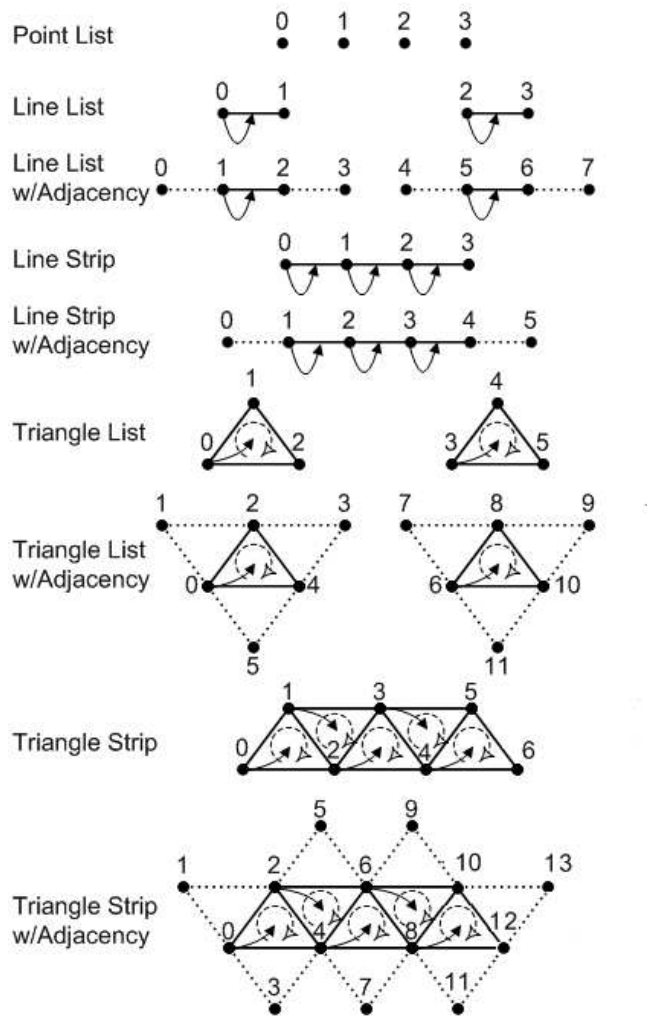
   첫 번째는 식별할 semantic 텍스트 문자열,
   두 번째는 텍스쳐 데이터,
   세번째는 일반 데이터를 식별합니다.

```
D3D11_INPUT_ELEMENT_DESC layout[] =
{
    {"POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0, D3D11_INPUT_PER_VERTEX_DATA, 0},
    {"COLOR", 0, DXGI_FORMAT_R32G32B32A32_FLOAT, 0, 12, D3D11_INPUT_PER_VERTEX_DATA, 0}
};
```

3. primitive topologies 형식 지정하기
   해당 함수는 파이프 라인에서 어떻게 렌더링할 지 설정합니다.

```
m_pDeviceContext->IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
```

- 출력 방식 리스트



Vertex Shader 스테이지

정점 셰이더(VS) 단계는 입력 어셈블러에서 전달된 정점을 처리하여 변환, 스키닝, 모핑, 정점별 조명과 같은 정점별 연산을 수행합니다.

정점 셰이더는 항상 단일 입력 정점에 대해 동작하고 단일 출력 정점을 생성합니다.

VertexShader.hlsl

```
struct VS_INPUT
{
    float3 pos : POSITION;
    float4 col : COLOR;
};

struct PS_INPUT
{
    float4 pos : SV_POSITION;
    float4 col : COLOR;
};

PS_INPUT main(VS_INPUT input)
{
    PS_INPUT output;

    output.pos = float4(input.pos, 1.0f); // 위치 결정
    output.col = input.col; // 색상 결정
```

```
    return output;
}
```

1. Vertex Shader 생성하기

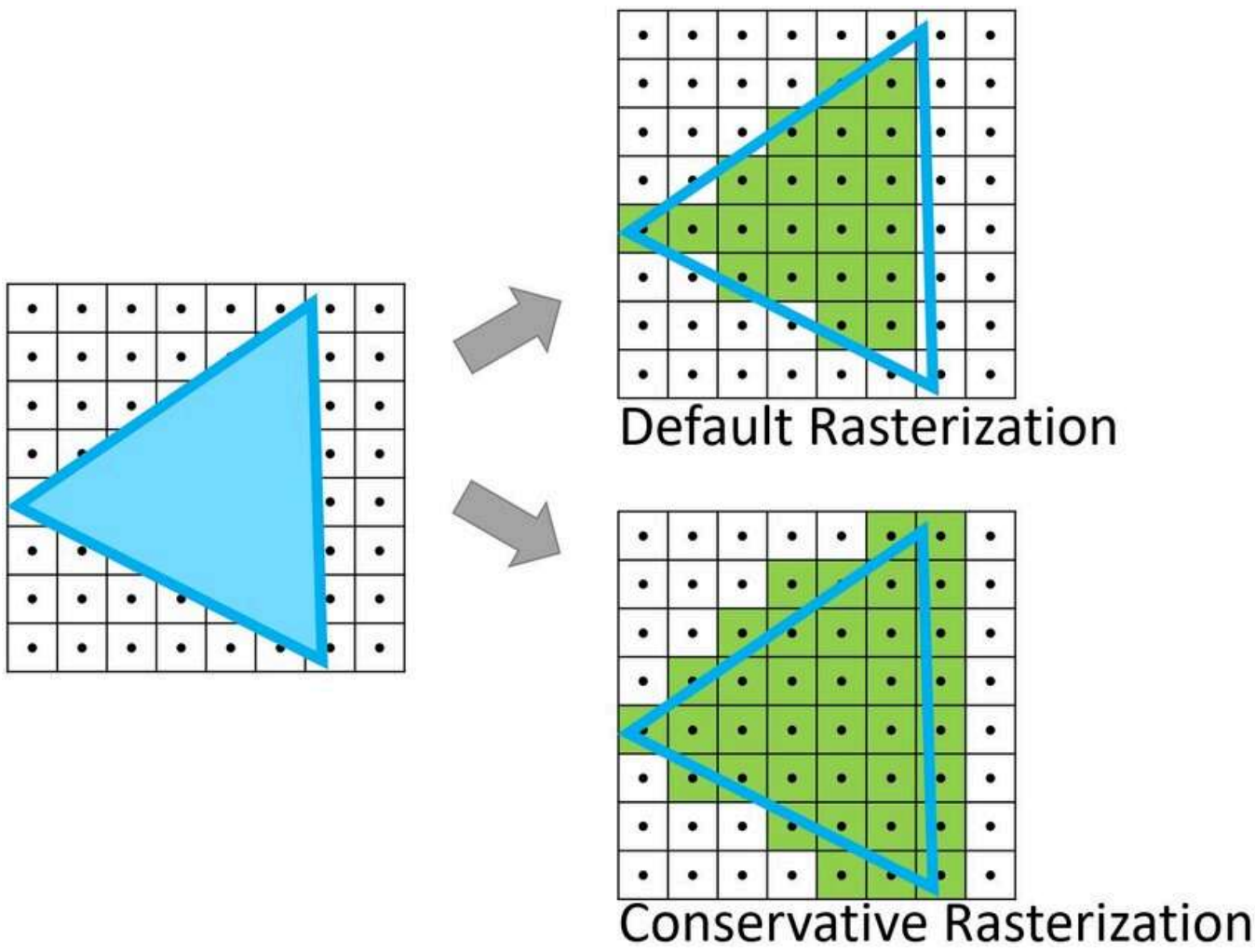
```
    // 버텍스 셰이더 생성
    ID3DBlob* vertexShaderBuffer = nullptr;
    HR_T(CompileShaderFromFile(L"BasicVertexShader.hlsl", "main", "vs_4_0", &vertexShaderBuffer));
    HR_T(m_pDevice->CreateVertexShader(vertexShaderBuffer->GetBufferPointer(), vertexShaderBuffer->GetBuffe
rSize(), NULL, &m_pVertexShader));
```

2. Vertex Shader 설정하기

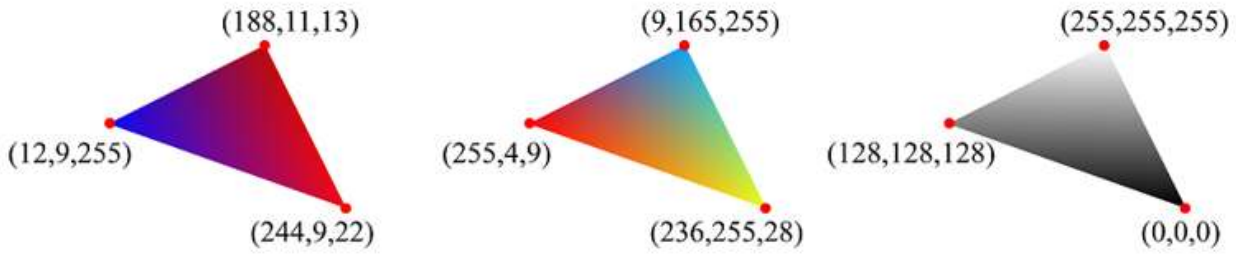
```
// m_pVertexShader는 ID3D11VertexShader*

m_pDeviceContext->VSSetShader(m_pVertexShader, nullptr, 0);
```

래스터라이저 스테이지

래스터화 단계는 실시간 3D 그래픽을 표시하기 위해 벡터 정보(형태 또는 기본 요소로 구성)를 래스터 이미지(픽셀로 구성)로 변환합니다.



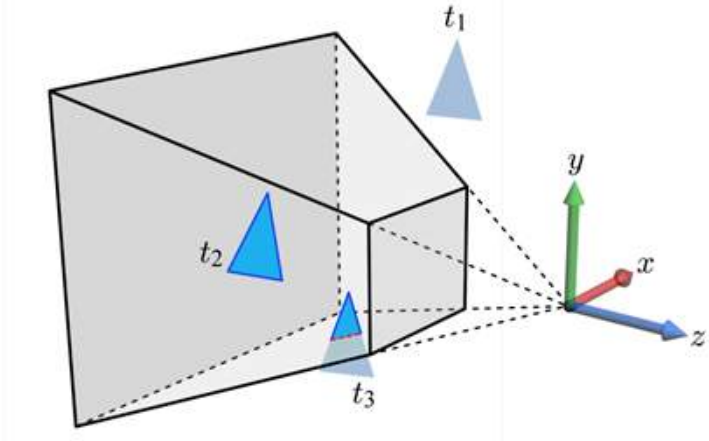
래스터화 동안 각 기본 요소는 픽셀로 변환되며 각 기본 요소에 걸쳐 꼭짓점별 값을 보간합니다.



색깔이 보간된 삼각형 그림

래스터화에는 시야 절두체로 꼭짓점 클리핑, z로 나누기 수행을 통한 원근 제공, 기본 요소를 2D 뷰포트로 매핑, 픽셀 셰이더 호출 방법 결정이 포함됩니다.

픽셀 셰이더 사용은 선택적이지만 래스터라이저 단계는 항상 클리핑, 원근 분할을 수행하여 점을 같은 공간으로 변환하고, 꼭짓점을 뷰포트에 매핑합니다.



래스터라이저 단계로 들어오는 꼭짓점(x,y,z,w)은 같은 유형의 클립 공간에 있는 것으로 간주됩니다. 이 좌표 공간에서 X축은 오른쪽, Y축은 위쪽, Z축은 카메라에서 먼 쪽을 가리킵니다.

1. 뷰포트 설정하기

```
    // 뷰포트 설정하기
    D3D11_VIEWPORT viewport = {};
    viewport.TopLeftX = 0;
    viewport.TopLeftY = 0;
    viewport.Width = (float)m_ClientWidth;
    viewport.Height = (float)m_ClientHeight;
    viewport.MinDepth = 0.0f;
    viewport.MaxDepth = 1.0f;

    // 뷰포트 설정
    m_pDeviceContext->RSSetViewports(1, &viewport);
```

Pixel Shader 스테이지

Pixel Shader는 상수 변수, 텍스쳐 데이터, 보간된 꼭짓점 값 및 기타 데이터를 결합하여 각 픽셀에 출력할 색을 반환한다.

PixelShader.hlsl

```
struct PS_INPUT
{
    float4 pos : SV_POSITION;
```

```
    float4 col : COLOR;
};

float4 main(PS_INPUT input) : SV_TARGET
{
    return input.col; // 정점 색상 그대로 출력
}
```

1. 픽셀 셰이더 설정하기

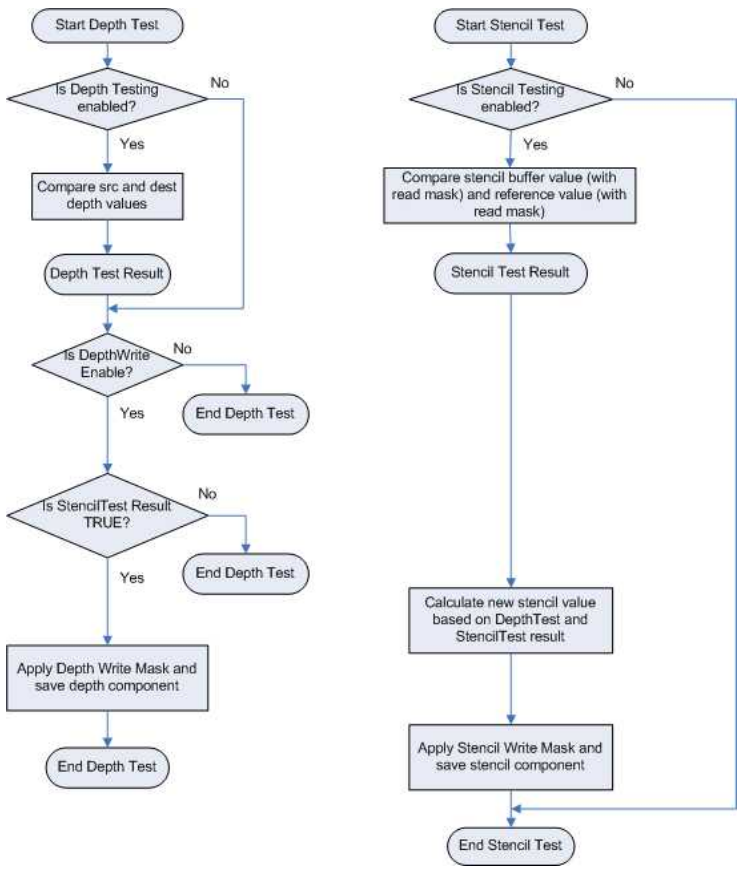
```
// m_pPixelShader는 ID3D11PixelShader*

// 픽셀 세이더 생성
ID3DBlob* pixelShaderBuffer = nullptr;
CompileShaderFromFile(L"BasicPixelShader.hlsl", "main", "ps_4_0", &pixelShaderBuffer);
m_pDevice->CreatePixelShader(pixelShaderBuffer->GetBufferPointer(), pixelShaderBuffer->GetBufferSize(), NUL
L, &m_pPixelShader);

// Pixel Shader 바인딩
m_pDeviceContext->PSSetShader(m_pPixelShader, nullptr, 0);
```

Output Merger 스테이지

Output Merger는 파이프 라인 상태, 픽셀 셰이더에서 생성된 픽셀 데이터, 렌더링 대상의 내용 및 깊이 / 스텐실 버퍼의 내용을 조합하여 최종 렌더링될 픽셀 색을 생성합니다.

OM 단계에서는 어떤 픽셀(깊이 스텐실 테스트 포함)이 보이는지 확인하고 최종 색을 블랜딩합니다.

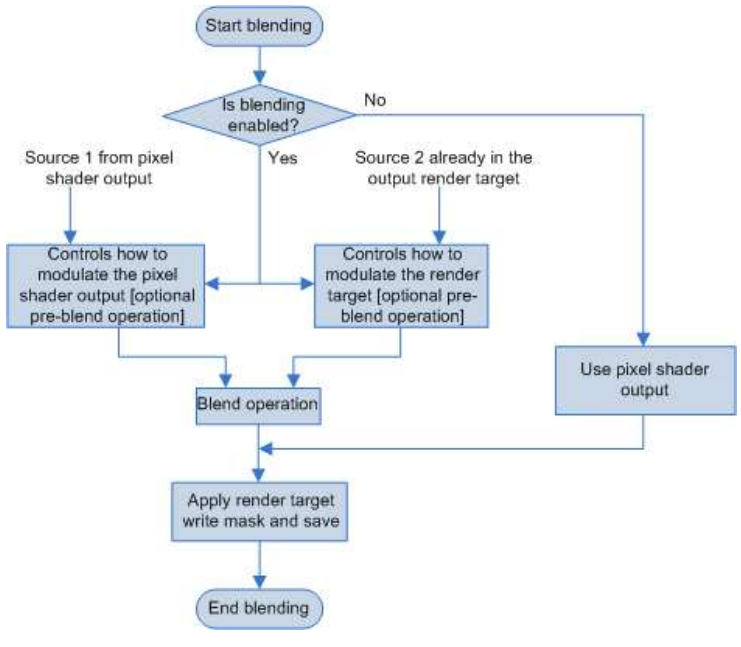
1. Depth-Stencil 테스트 (z버퍼링)



깊이 버퍼를 사용하여 픽셀을 결정합니다.

만약 애플리케이션에서 지정하지 않으면 OM단계에서 기본 값을 사용합니다.

2. Blending



블랜딩은 하나 이상의 픽셀 값을 결합하여 최종 픽셀 색을 만듭니다.

참고 문서

https://learn.microsoft.com/ko-kr/windows/win32/direct3d10/d3d10-graphics-programming-guide-pipeline-stages?source=recommendations

https://microsoft.github.io/DirectX-Specs/d3d/archive/D3D11_3_FunctionalSpec.htm#Rendering Pipeline Overview Changes

https://jidon333.github.io/blog/Rendering-pipeline