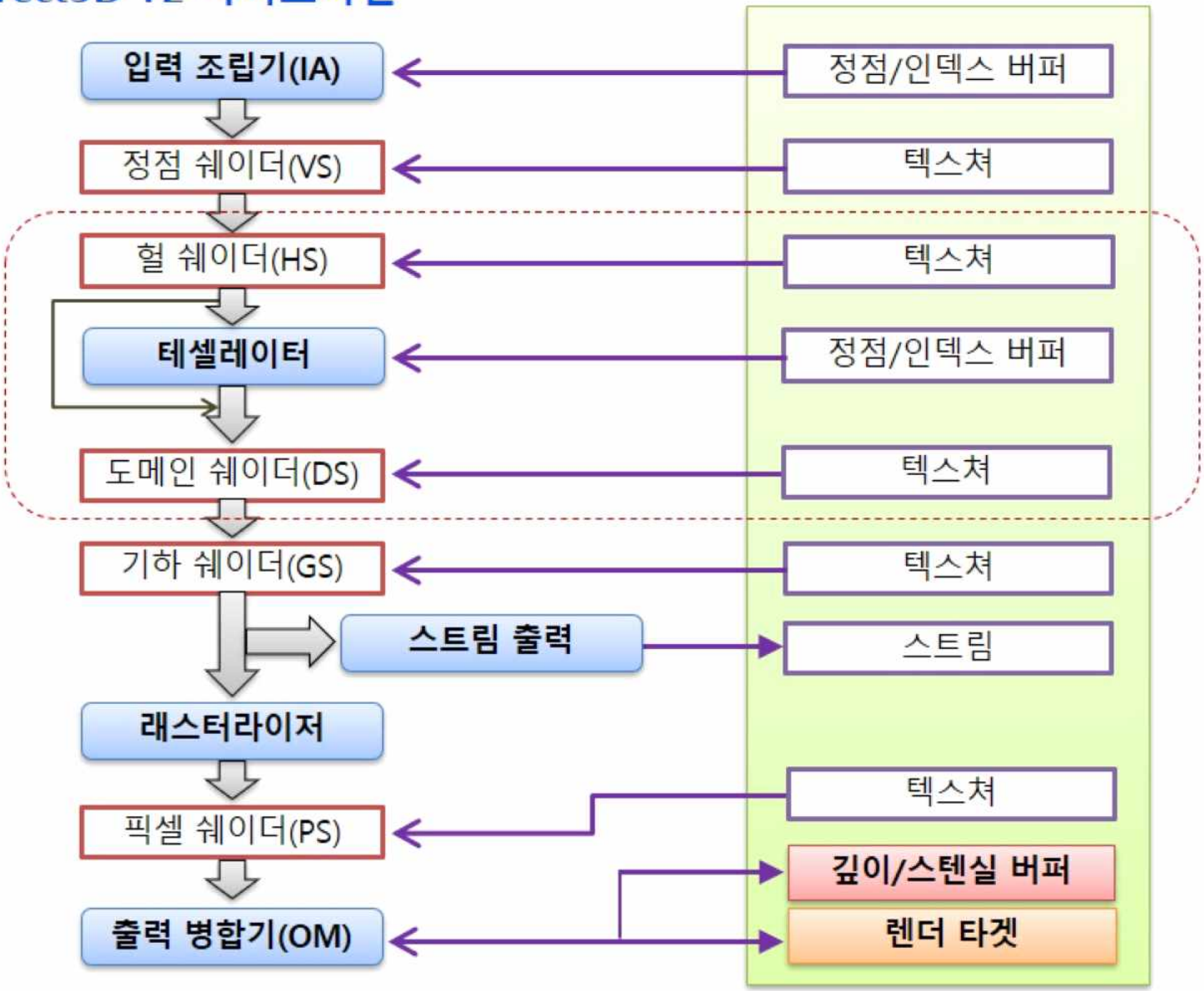
Game Programming with DirectX

Direct3D Graphics Pipeline (Geometry Shader)

Direct3D 파이프라인

- Direct3D 12 파이프라인

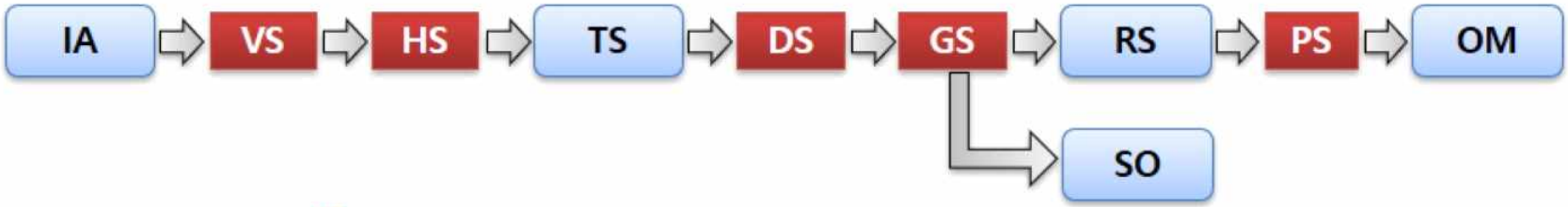


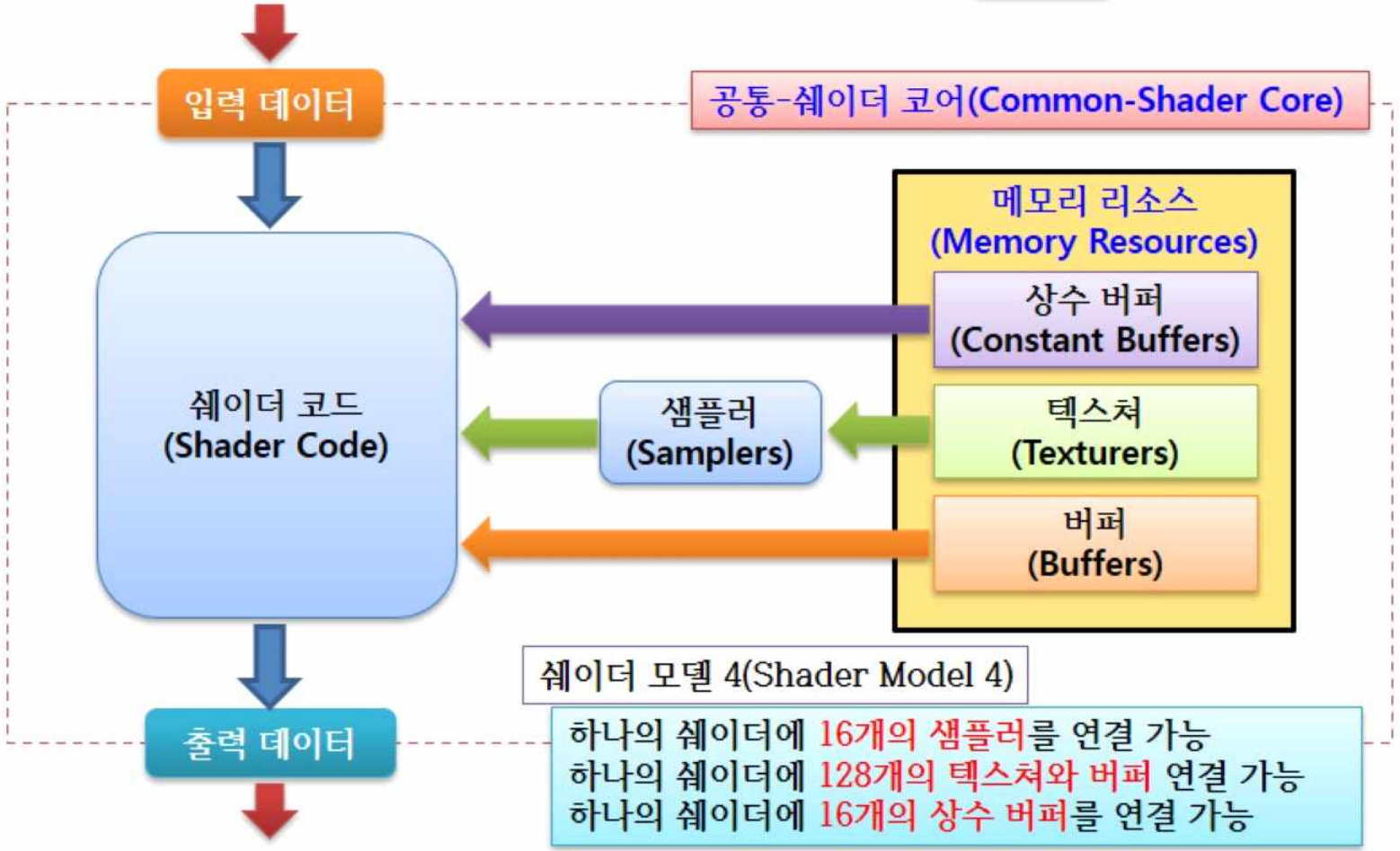
Direct3D 파이프라인

- **Direct3D 12 렌더링 파이프라인**
  - GPU를 사용하여 리소스(정점/인덱스, 텍스쳐)를 2D 이미지로 렌더링하는 과정
  - 파이프라인은 여러 개의 파이프라인 단계(Pipeline Stage)로 구성
    프로그램 가능 단계와 고정 프로그램 단계로 구분
  - **고정 프로그램 단계**
    Direct3D에서 모든 처리가 진행되며 응용 프로그램에서 변경할 수 없는 단계
    - **입력 조립** 단계(**IA**: Input Assembler Stage)
    - **테셀레이터** 단계(**TS**: Tessellator Stage)
    - **스트림 출력** 단계(**SO**: Stream Output Stage)
    - **래스터라이저** 단계(**RS**: Rasterizer Stage)
    - **출력 병합** 단계(**OM**: Output Merger Stage)
  - **프로그램 가능 단계**
    응용 프로그램에서 쉐이더 프로그램을 통하여 제공해야 하는 단계
    - **정점 쉐이더** 단계(**VS**: Vertex Shader Stage)
    - **헐 쉐이더** 단계(**HS**: Hull Shader Stage)
    - **도메인 쉐이더** 단계(**DS**: Domain Shader Stage)
    - **기하 쉐이더** 단계(**GS**: Geometry Shader Stage)
    - **픽셀 쉐이더** 단계(**PS**: Pixel Shader Stage)
  - Direct3D 렌더링 파이프라인의 단계들은 연결되어 **연동**되도록 구성
    일반적으로 각 단계의 출력 데이터가 바로 다음 단계의 입력 데이터가 되도록 구성

Direct3D 파이프라인

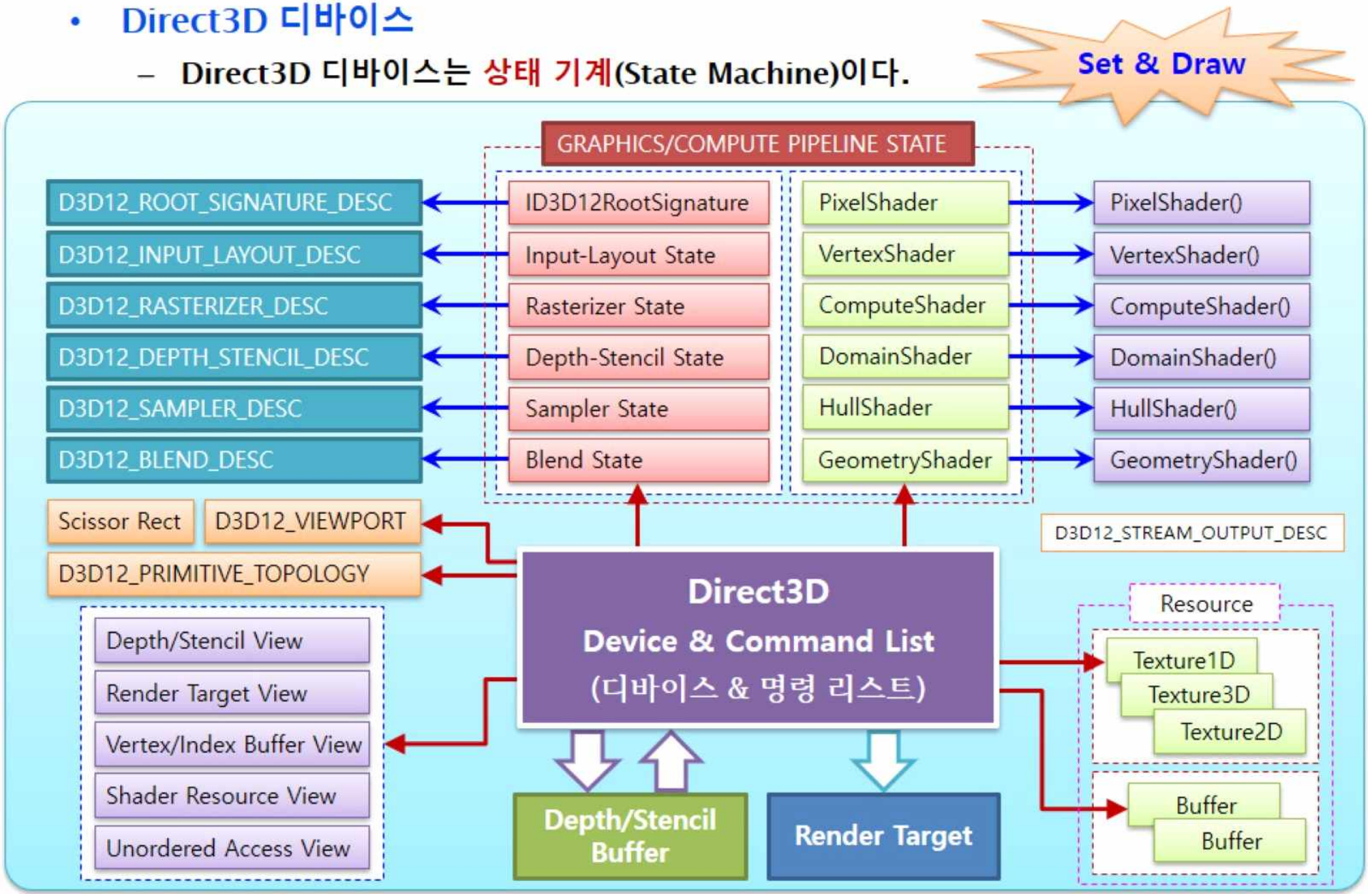
- Direct3D 쉐이더 단계(Shader Stage)





Direct3D 파이프라인

- Direct3D 디바이스
  - Direct3D 디바이스는 상태 기계(State Machine)이다.



Direct3D 파이프라인

- **ID3D12Device**
  - 가상 어댑터(Virtual Adapter)를 나타내는 인터페이스

CheckFeatureSupport

CopyDescriptors	CopyDescriptorsSimple	
CreateCommandAllocator	**CreateCommandList**	**CreateCommandQueue**
CreateCommandSignature	**CreateCommittedResource**	CreateComputePipelineState
CreateConstantBufferView	**CreateDepthStencilView**	**CreateDescriptorHeap**
CreateFence	**CreateGraphicsPipelineState**	**CreateHeap**
CreatePlacedResource	CreateQueryHeap	**CreateRenderTargetView**
CreateReservedResource	CreateRootSignature	**CreateSampler**
CreateShaderResourceView	CreateSharedHandle	CreateUnorderedAccessView
Evict		
GetAdapterLuid	GetCopyableFootprints	GetCustomHeapProperties
GetDeviceRemovedReason	GetNodeCount	**GetDescriptorHandleIncrementSize**
GetResourceAllocationInfo	GetResourceTiling	
MakeResident		
OpenSharedHandle	OpenSharedHandleByName	
SetStablePowerState		

Direct3D 파이프라인

- **ID3D12GraphicsCommandList 인터페이스**

BeginEvent	BeginQuery	
ClearDepthStencilView	**ClearRenderTargetView**	
ClearState	ClearUnorderedAccessViewFloat	ClearUnorderedAccessViewUint
Close		
CopyBufferRegion	CopyResource	CopyTextureRegion
CopyTiles		
DiscardResource		
Dispatch		
DrawIndexedInstanced	**DrawInstanced**	
EndEvent	EndQuery	
ExecuteBundle	ExecuteIndirect	
IASetIndexBuffer	**IASetPrimitiveTopology**	**IASetVertexBuffers**
OMSetBlendFactor	**OMSetRenderTargets**	**OMSetStencilRef**
Reset		
ResolveQueryData	ResolveSubresource	
ResourceBarrier		
RSSetScissorRects	**RSSetViewports**	SetComputeRoot32BitConstant
SetComputeRoot32BitConstants	SetComputeRootConstantBufferView	
SetComputeRootDescriptorTable	SetComputeRootShaderResourceView	
SetComputeRootSignature	SetComputeRootUnorderedAccessView	
SetDescriptorHeaps		
SetGraphicsRoot32BitConstant	SetGraphicsRoot32BitConstants	SetGraphicsRootConstantBufferView
SetGraphicsRootDescriptorTable	SetGraphicsRootShaderResourceView	
SetGraphicsRootSignature	SetGraphicsRootUnorderedAccessView	
SetMarker	**SetPipelineState**	SetPredication
SOSetTargets		

Direct3D 파이프라인

- 그래픽 파이프라인 상태

```
typedef struct D3D12_GRAPHICS_PIPELINE_STATE_DESC {
    ID3D12RootSignature *pRootSignature;
    D3D12_SHADER_BYTECODE VS;
    D3D12_SHADER_BYTECODE PS;
    D3D12_SHADER_BYTECODE DS;
    D3D12_SHADER_BYTECODE HS;
    D3D12_SHADER_BYTECODE GS;
    D3D12_STREAM_OUTPUT_DESC Str
    D3D12_BLEND_DESC BlendState;
    UINT SampleMask;
    D3D12_RASTERIZER_DESC Rasterize
    D3D12_DEPTH_STENCIL_DESC Dep
    D3D12_INPUT_LAYOUT_DESC Input
    D3D12_INDEX_BUFFER_STRIP_CUT_V
    D3D12_PRIMITIVE_TOPOLOGY_TYPE
    UINT NumRenderTargets;
    DXGI_FORMAT RTVFormats[8];
    DXGI_FORMAT DSVFormat;
    DXGI_SAMPLE_DESC SampleDesc;
    UINT NodeMask;
    D3D12_CACHED_PIPELINE_STATE CachedPSO;
    D3D12_PIPELINE_STATE_FLAGS Flags;
} D3D12_GRAPHICS_PIPELINE_STATE_DESC;
```

```
typedef struct D3D12_SHADER_BYTECODE {
    void *pShaderBytecode;
    SIZE_T BytecodeLength;
} D3D12_SHADER_BYTECODE;
```

```
typedef struct D3D
    D3D12_FILL_MO
    D3D12_CULL_MO
    BOOL FrontCour
    INT DepthBias;
    FLOAT DepthBias
    FLOAT SlopeScal
    BOOL DepthClip
    BOOL Multisamp
    BOOL AntialiasedLineEnable;
    UINT ForcedSampleCount;
    D3D12_CONSERVATIVE_RASTERIZATION_MODE ConservativeRaster;
} D3D12_RASTERIZER_DESC;
```

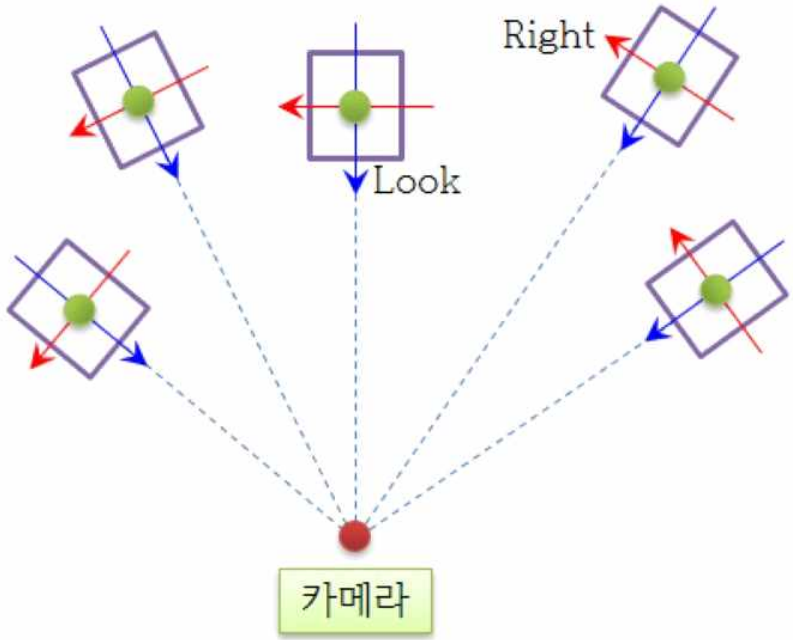
```
typedef struct D3D12_DEPTH_STENCIL_DESC {
    BOOL DepthEnable;
    D3D12_DEPTH_WRITE_MASK DepthWriteMask;
    D3D12_COMPARISON_FUNC DepthFunc;
    BOOL StencilEnable;
    UINT8 StencilReadMask;
    UINT8 StencilWriteMask;
    D3D12_DEPTH_STENCILOP_DESC FrontFace;
    D3D12_DEPTH_STENCILOP_DESC BackFace;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

```
typedef enum D3D12_PIPELINE_STATE_FLAGS {
    D3D12_PIPELINE_STATE_FLAG_NONE,
    D3D12_PIPELINE_STATE_FLAG_TOOL_DEBUG
} D3D12_PIPELINE_STATE_FLAGS;
```

```
typedef
    D3D12
    D3D12
    D3D12_PRIMITIVE_TOPOLOGY_TYPE_LINE,
    D3D12_PRIMITIVE_TOPOLOGY_TYPE_TRIANGLE,
    D3D12_PRIMITIVE_TOPOLOGY_TYPE_PATCH
} D3D12_PRIMITIVE_TOPOLOGY_TYPE;
```

Direct3D 파이프라인

- **나무(Tree) 그리기**
  - 나무(Tree)를 아주 많이 그린다고 가정
    메쉬를 사각형으로 만들고 텍스쳐 이미지를 매핑(알파 블렌딩), 빌보드 처리



하나의 나무마다 정점 버퍼, 월드 변환 행렬
인스턴싱을 사용하면 효율적
다른 방법은?
나무마다 행렬이 필요하지는 않음

Direct3D 파이프라인

- **정점 조립(Input Assembly)**

```
D3D12_INPUT_ELEMENT_DESC d3dInputLayout[] = {
   { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0, D3D12_INPUT_~_PER_VERTEX_DATA, 0 },
   { "COLOR", 0, DXGI_FORMAT_R32G32B32A32_FLOAT, 0, 12, D3D12_INPUT_~_PER_VERTEX_DATA, 0 },
   { "INSTANCE", 0, DXGI_FORMAT_R32G32B32A32_FLOAT, 1, 0, D3D12_INPUT_~_PER_INSTANCE_DATA, 1 },
   { "INSTANCE", 1, DXGI_FORMAT_R32G32B32A32_FLOAT, 1, 16, D3D12_INPUT_~_PER_INSTANCE_DATA, 1 },
   { "INSTANCE", 2, DXGI_FORMAT_R32G32B32A32_FLOAT, 1, 32, D3D12_INPUT_~_PER_INSTANCE_DATA, 1 },
   { "INSTANCE", 3, DXGI_FORMAT_R32G32B32A32_FLOAT, 1, 48, D3D12_INPUT_~_PER_INSTANCE_DATA, 1 }
};
```

```
void ID3D12GraphicsCommandList::DrawIndexedInstanced(
   UINT IndexCountPerInstance, //인덱스의 개수
   UINT InstanceCount, //인스턴스의 개수
   UINT StartIndexLocation, //시작 인덱스의 위치(인덱스)
   INT BaseVertexLocation, //각 정점 인덱스에 더해지는 값
   UINT StartInstanceLocation //인스턴스 인덱스에 더해지는 값
);
```

```
struct VS_INPUT {
   float3 position : POSITION;
   float4 color : COLOR0;
   float4x4 transform : INSTANCE;
};
```

```
VS_OUTPUT VS(VS_INPUT input, uint nInstanceID : SV_InstanceID) {
   VS_OUTPUT output = (VS_OUTPUT)0;
   output.position = mul(float4(input.position, 1), input.transform);
   output.position = mul(output.position, gmtxView);
   output.position = mul(output.position, gmtxProjection);
   output.color = input.color;
   output.instanceID = nInstanceID;
   return(output);
}
```

```
struct VS_OUTPUT {
   float4 position : SV_POSITION;
   float4 color : COLOR0;
   float4 instanceID : INSTANCE;
};
```

```
m_pd3dGraphicsCommandList->DrawInstanced(36, 1000, 0, 0);
```

Direct3D 파이프라인

- **기하 쉐이더 단계(Geometry-Shader Stage)**
  - 기하 쉐이더(GS) 단계는 **하나의 완전한 프리미티브를 구성하는** 정점들을 입력받아 **새로운 정점들을 생성**하여 출력(tristrip, linestrip, pointlist)할 수 있다. 정점-쉐이더의 입력은 에지-인접성 정보를 포함할 수 있다(예, 삼각형에 대하여 세 개의 정점, 인접 정점 프리미티브의 경우 추가적인 세 개의 인접 정점).
  - 기하 쉐이더가 활성화되면 기하 쉐이더는 **각 프리미티브(점, 선분, 삼각형)에 대하여 한번씩 호출**된다. 예를 들어 삼각형 스트립의 경우에는 스트립의 각 삼각형에 대하여 기하 쉐이더가 한번씩 호출된다.
  - 기하 쉐이더는 프리미티브 객체(PointStream, LineStream, TriangleStream)에 새로운 정점들을 순차적으로 추가한다.
  - 출력되는 프리미티브의 개수는 기하 쉐이더에서 자유롭게 변경할 수 있다. 출력할 수 있는 정점들의 최대 개수는 정적으로 선언되어야 한다. 정점들은 항상 스트립으로 출력된다. 기하 쉐이더가 호출될 때마다 새로운 스트립이 시작되며 또한 새로운 스트립은 RestartStrip() HLSL 함수로 생성할 수 있다.
  - 기하 쉐이더 단계는 입력-조립 단계가 생성하는 **SV_PrimitiveID 시멘틱 값을 사용**할 수 있다. 텍스쳐의 로드(Load)와 샘플링 연산을 수행할 수 있다.
  - 기하 쉐이더의 출력은 래스터라이저 단계 또는 스트림 출력 단계를 거쳐 정점 버퍼로 입력될 수 있다.

Direct3D 파이프라인

- **래스터라이저 단계(Rasterizer Stage) 설정**
  - 뷰포트(Viewport) 설정
    뷰포트는 렌더링할 렌더 타겟(후면 버퍼) 영역을 나타내는 구조체임
    하나의 렌더 타겟에 하나의 뷰포트를 설정할 수 있음
    명령 리스트가 Reset() 될 때마다 뷰포트를 다시 설정해야 함
    모든 뷰포트를 동시에 설정해야 함
    뷰포트의 개수가 1보다 크면 기하 쉐이더에서 SV_ViewportArrayIndex를 사용 가능
    최대 개수: D3D12_VIEWPORT_AND_SCISSORRECT_OBJECT_COUNT_PER_PIPELINE(16)
    깊이 값 [0, 1.0]은 [MinDepth, MaxDepth]로 변환됨

```
void ID3D12GraphicsCommandList::RSSetViewports(
   UINT NumViewports, //뷰포트의 개수
   D3D12_VIEWPORT *pViewports //뷰포트들의 배열
);
```

```
typedef struct D3D12_VIEWPORT {
   FLOAT TopLeftX;
   FLOAT TopLeftY;
   FLOAT Width;
   FLOAT Height;
   FLOAT MinDepth; //0~1.0
   FLOAT MaxDepth; //0~1.0
} D3D12_VIEWPORT;
```

  - 시저(Scissor) 사각형 설정
    시저 사각형은 렌더링할 영역(사각형)을 의미함
    모든 시저 사각형을 동시에 설정해야 함
    시저 사각형에 포함되지 않은 영역이 렌더링(래스터라이저)에서 제거됨
    하나의 렌더 타겟에 하나의 시저 사각형을 설정할 수 있음
    명령 리스트가 Reset() 될 때마다 시저 사각형을 다시 설정해야 함

```
void ID3D12GraphicsCommandList::RSSetScissorRects(
   UINT NumRects, //시저 사각형의 개수
   D3D12_RECT *pRects //시저 사각형들의 배열
);
```

Direct3D 파이프라인

- **기하 쉐이더 단계(Geometry-Shader Stage)**
  - 시스템 값 시멘틱
    - **SV_GSInstanceID**
      실행되는 기하 쉐이더의 인스턴스 ID
    - **SV_RenderTargetArrayIndex**
      렌더 타겟이 배열 리소스일 때 렌더 타겟 배열의 인덱스를 나타냄
      프리미티브가 어떤 렌더 타겟(깊이 스텐실 버퍼)으로 출력되는 가를 나타냄
      기하 쉐이더의 출력과 픽셀 쉐이더의 입력으로 사용
    - **SV_ViewportArrayIndex**
      뷰 포트 배열의 인덱스(프리미티브가 어떤 뷰 포트를 사용할 것인 가를 나타냄)
      기하 쉐이더의 출력과 픽셀 쉐이더의 입력으로 사용

```
struct GS_OUTPUT {
    float3 positionL: POSITION;
    float4 position: SV_POSITION;
    uint renderTarget: SV_RenderTargetArrayIndex;
};
```

```
void ID3D12GraphicsCommandList::RSSetViewports(
    UINT NumViewports, //뷰포트의 개수
    D3D12_VIEWPORT *pViewports //뷰포트들의 배열
);
```

```
[maxvertexcount(18)]
void GSSkyBox(triangle VS_OUTPUT input[3], inout TriangleStream<GS_OUTPUT> outStream) {
    for (int i = 0; i < 6; i++) {
        GS_SKYBOX_OUTPUT output;
        output.renderTarget = i;
        ...
    }
}
```

[instance(n)]
n: 각 프리미티브에 대하여 실행되는 기하 쉐이더 인스턴스의 개수(최대 32개까지)
SV_GSInstanceID

Direct3D 파이프라인

- 기하 쉐이더(Geometry Shader)

```
[maxvertexcount(NumVerts)]
void GeometryShaderName(
    PrimitiveType DataType Name[NumElements],
    inout StreamOutputObject<DataType> Name
);
```

[earlydepthstencil]
픽셀 쉐이더를 실행하기 전에 깊이-스텐실 검사를 실행

NumVerts	생성할 정점의 최대 개수
ShaderName	기하 쉐이더 이름(문자열)
PrimitiveType	프리미티브 유형(point, line, triangle, lineadj, triangleadj)
DataType	입력 데이터 형식(HLSL 데이터 형식)
Name	파라메터 이름(문자열)
NumElements	파라메터 원소의 개수
StreamOutputObject	스트림-출력 객체, 템플릿 형태로 사용(PointStream, LineStream, TriangleStream)

PrimitiveType

point	점들의 리스트, [1]
line	선분들의 리스트 또는 스트립, [2]
triangle	삼각형들의 리스트 또는 스트립, [3]
lineadj	인접성 선분들의 리스트 또는 스트립, [4]
triangleadj	인접성 삼각형들의 리스트 또는 스트립, [6]

StreamOutputObject

PointStream	출력 프리미티브가 점인 경우
LineStream	출력 프리미티브가 선분인 경우
TriangleStream	출력 프리미티브가 삼각형인 경우

Append	출력 데이터를 스트림에 추가
RestartStrip	새로운 프리미티브 스트립을 시작

Direct3D 파이프라인

- **스트림-출력 객체(Stream-Output Object)**
  - 기하 쉐이더의 출력 데이터를 순서대로 출력(Stream)하기 위한 템플릿 객체

```
inout StreamOutputObject<DataType> Name;
```

StreamOutputObject

스트림-출력은 4개까지 가능
2개 이상의 스트림을 사용할 때 모두 PointStream이어야 함

PointStream	출력 프리미티브가 점인 경우
LineStream	출력 프리미티브가 선분인 경우
TriangleStream	출력 프리미티브가 삼각형인 경우

Append	출력 데이터를 스트림에 추가
RestartStrip	현재의 프리미티브 스트립을 끝내고 새로운 프리미티브 스트립을 시작

```
Append(StreamDataType);
```

StreamDataType	스트림-출력 객체 선언의 템플릿 데이터 형식(DataType)과 일치

```
RestartStrip();
```
스트림-출력은 항상 프리미티브 스트립을 가정
삼각형 리스트를 출력하려면 각 삼각형에 대하여 이 함수를 호출

```
[maxvertexcount(18)]
void GS_CubeMap(triangle GS_IN input[3], inout TriangleStream<PS_IN> outStream) {
   PS_IN output;
   ...
   outStream.Append(output);
}
```

Direct3D 파이프라인

- 기하 쉐이더(Geometry-Shader)

```
GS_IN VSCubeMap(VS_IN input)
{
    GS_IN output = (GS_IN)0.0f;
    output.position = mul(input.position, gmtxWorld);
    output.uv = input.uv;
    return output;
}
```

```
struct VS_IN {
    float4 position: POSITION;
    float3 normal: NORMAL;
    float2 uv: TEXCOORD;
};
```

```
struct GS_IN {
    float4 position: SV_POSITION;
    float2 uv: TEXCOORD;
};
```

```
[maxvertexcount(18)]
void GSCubeMap(triangle GS_IN input[3], inout TriangleStream<PS_IN> outStream)
{
    for (int f = 0; f < 6; f++)
    {
        PS_IN output;
        output.renderTargetIndex = f;
        for (int v = 0; v < 3; v++)
        {
            output.position = mul(input[v].position, gmtxViews[f]);
            output.position = mul(output.position, gmtxProjection);
            output.uv = input[v].uv;
            outStream.Append(output);
        }
        outStream.RestartStrip();
    }
}
```

```
struct PS_IN {
    float4 position: SV_POSITION;
    float2 uv: TEXCOORD0;
    uint renderTargetIndex: SV_RenderTargetArrayIndex;
};
```

```
float4 PSCubeMap(PS_IN input) : SV_Target {
    return gtxtDiffuse.Sample(gssLinear, input.uv);
}
```

Direct3D 파이프라인

- 기하 쉐이더(Geometry-Shader)

```
[maxvertexcount(8)]
void GS(triangle VS_OUT input[3], inout TriangleStream<GS_OUT> outStream) {
    GS_OUT v[6];
    v[0].position = input[0].position;
    v[0].color = input[0].color;
    v[1].position = input[1].position;
    v[1].color = input[1].color;
    v[2].position = input[2].position;
    v[2].color = input[2].color;
    v[3].position = (input[0].position + input[1].position) * 0.5f;
    v[3].color = (input[0].color + input[1].color + float4(1.0f, 0.0f, 0.0f, 1.0f)) / 3.0f;
    v[4].position = (input[1].position + input[2].position) * 0.5f;
    v[4].color = (input[1].color + input[2].color + float4(0.0f, 1.0f, 0.0f, 1.0f)) / 3.0f;
    v[5].position = (input[0].position + input[2].position) * 0.5f;
    v[5].color = (input[0].color + input[2].color + float4(0.0f, 0.0f, 1.0f, 1.0f)) / 3.0f;
    outStream.Append(v[0]);
    outStream.Append(v[3]);
    outStream.Append(v[5]);
    outStream.Append(v[4]);
    outStream.Append(v[2]);
    outStream.RestartStrip();
    outStream.Append(v[3]);
    outStream.Append(v[1]);
    outStream.Append(v[4]);
}
```

```
struct VS_OUT {
    float4 position : POSITION;
    float4 color : COLOR0;
};
```

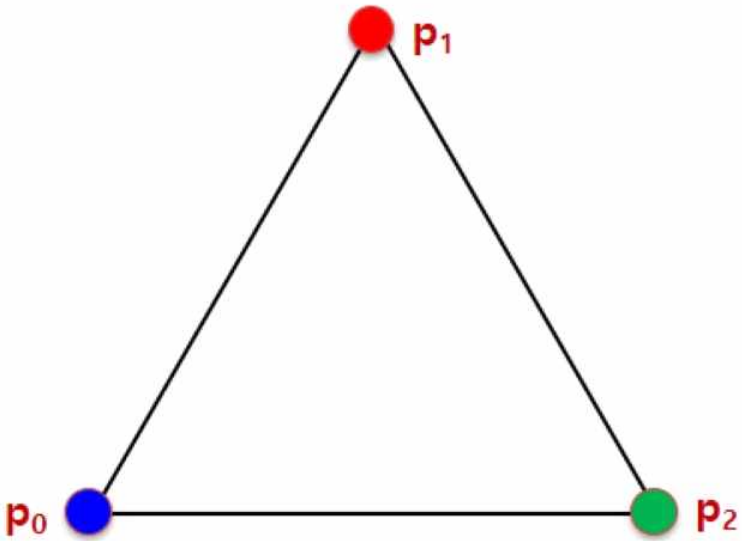
```
struct GS_OUT {
    float4 position : SV_POSITION;
    float4 color : COLOR0;
};
```

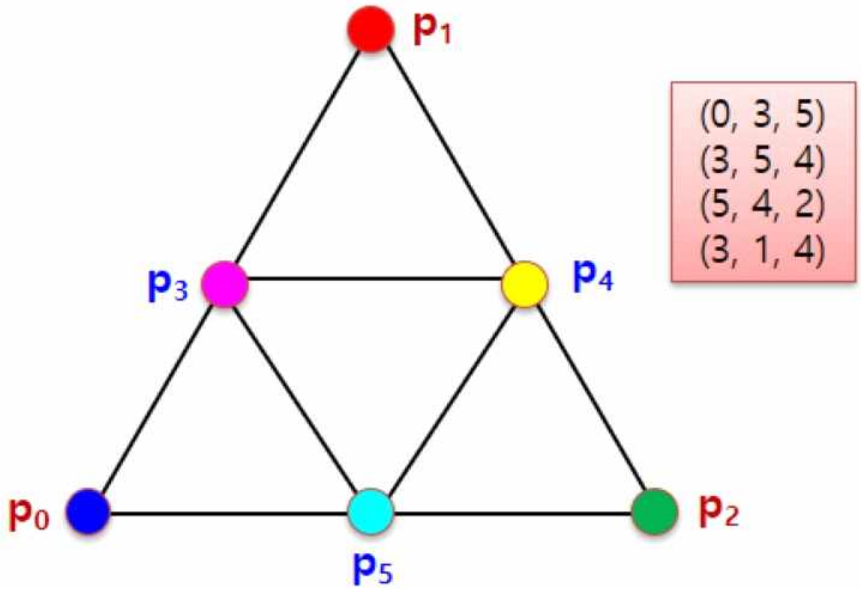
```
VS_OUT VS(float4 position : POSITION, float4 color : COLOR) {
    VS_OUT output = (VS_OUT)0;
    output.position = mul(position, gmtxWorld);
    output.position = mul(output.position, gmtxView);
    output.position = mul(output.position, gmtxProjection);
    output.color = color;
    return(output);
}
```

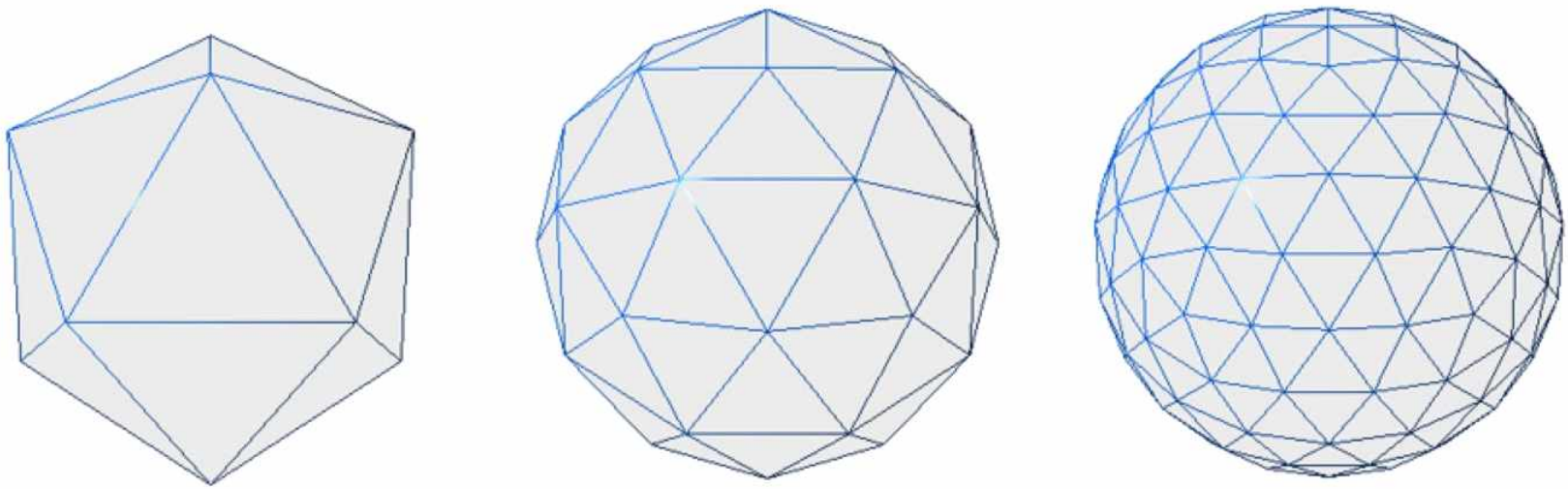
```
float4 PS(GS_OUT input) : SV_Target {
    return(input.color);
}
```

Direct3D 파이프라인

- 기하 쉐이더(Geometry-Shader)







Direct3D 파이프라인

- 기하 쉐이더의 예(나무 빌보드)

```
class CTreeVertex {
public:
    XMFLOAT3 m_xmf3Position;
    XMFLOAT2 m_xmf2Size;
    CTreeVertex(XMFLOAT3 xmf3Position, XMFLOAT2 xmf2Size) { m_xmf3Position = xmf3Position; m_xmf3Size
= xmf3Size; }
    ~CTreeVertex() { }
};
```

```
int cxTerrain = m_pTerrain->m_pHeightMap->GetHeightMapWidth();
int czTerrain = m_pTerrain->m_pHeightMap->GetHeightMapLength();
nTrees = 100;
```

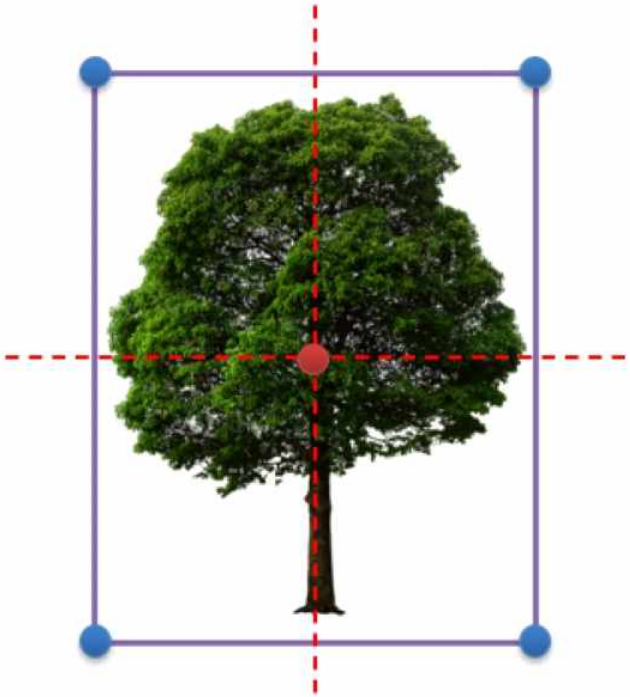
```
D3D12_INPUT_ELEMENT_DESC d3dInputLayout[] =
{
   { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0, ... },
   { "SIZE", 0, DXGI_FORMAT_R32G32_FLOAT, 0, 12, ...  }
};
```

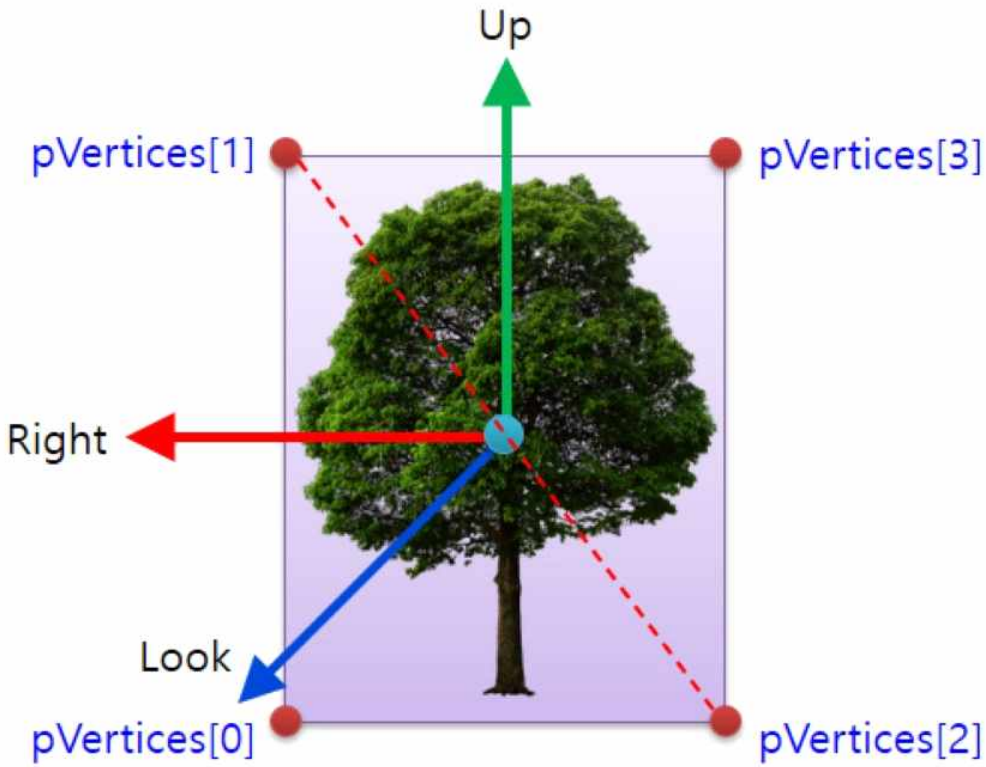
```
m_nStride = sizeof(CTreeVertex);
m_nVertices = nTrees;
XMFLOAT3 xmf3Position;
CTreeVertex *pTreeVertices = new CTreeVertex[nTrees];
for (int i = 0; i < nTrees; i++) {
    float fxTerrain = xmf3Position.x = rand() % cxTerrain;
    float fzTerrain = xmf3Position.z = rand() % czTerrain;
    xmf3Position.y = m_pTerrain->m_pHeightMap->GetHeight(fxTerrain, fzTerrain, false) + 6.0f;
    pTreeVertices[i] = CTreeVertex(xmf3Position, XMFLOAT2(20.0f, 50.0f));
}
m_pd3dVertexBuffer = ::CreateBufferResource(pd3dDevice, pd3dCommandList, pTreeVertices, m_nStride *
m_nVertices, D3D12_HEAP_TYPE_DEFAULT, D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER,
&m_pd3dVertexUploadBuffer);

m_d3dVertexBufferView.BufferLocation = m_pd3dVertexBuffer->GetGPUVirtualAddress();
m_d3dVertexBufferView.StrideInBytes = m_nStride;
m_d3dVertexBufferView.SizeInBytes = m_nStride * m_nVertices;
```

Direct3D 파이프라인

- 기하 쉐이더의 예(나무 빌보드)






```
float fHalfW = input[0].sizeW.x * 0.5f;
float fHalfH = input[0].sizeW.y * 0.5f;
float4 pVertices[4];
pVertices[0] = float4(input[0].centerW + fHalfW * vRight – fHalfH * vUp, 1.0f);
pVertices[1] = float4(input[0].centerW + fHalfW * vRight + fHalfH * vUp, 1.0f);
pVertices[2] = float4(input[0].centerW – fHalfW * vRight – fHalfH * vUp, 1.0f);
pVertices[3] = float4(input[0].centerW – fHalfW * vRight + fHalfH * vUp, 1.0f);
float2 pUVs[4] = { float2(0.0f,1.0f), float2(0.0f,0.0f), float2(1.0f,1.0f), float2(1.0f,0.0f) };
```

Direct3D 파이프라인

- 기하 쉐이더의 예(나무 빌보드)

```
[maxvertexcount(4)]
void GS(point VS_OUT input[1], uint primID : SV_PrimitiveID, inout TriangleStream<GS_OUT> outStream)
{
    float3 vUp = float3(0.0f, 1.0f, 0.0f);
    float3 vLook = gvCameraPosition.xyz - input[0].centerW;
    vLook = normalize(vLook);
    float3 vRight = cross(vUp, vLook);
    float fHalfW = input[0].sizeW.x * 0.5f;
    float fHalfH = input[0].sizeW.y * 0.5f;
    float4 pVertices[4];
    pVertices[0] = float4(input[0].centerW+fHalfW*vRight-fHalfH*vUp, 1.0f);
    pVertices[1] = float4(input[0].centerW+fHalfW*vRight+fHalfH*vUp, 1.0f);
    pVertices[2] = float4(input[0].centerW-fHalfW*vRight-fHalfH*vUp, 1.0f);
    pVertices[3] = float4(input[0].centerW-fHalfW*vRight+fHalfH*vUp, 1.0f);
    float2 pUVs[4] = { float2(0.0f,1.0f), float2(0.0f,0.0f), float2(1.0f,1.0f), float2(1.0f,0.0f) };
    GS_OUT output;
    for (int i = 0; i < 4; i++) {
        output.posW = pVertices[i].xyz;
        output.posH = mul(pVertices[i], gmtxViewProjection);
        output.normalW = vLook;
        output.uv = pUVs[i];
        output.primID = primID;
        outStream.Append(output);
    }
}
```

```
struct VS_IN {
    float3 posW : POSITION;
    float2 sizeW : SIZE;
};
```

```
struct VS_OUT {
    float3 centerW : POSITION;
    float2 sizeW : SIZE;
};
```

```
VS_OUT VS(VS_IN input) {
    VS_OUT output;
    output.centerW = input.posW;
    output.sizeW = input.sizeW;
    return output;
}
```

```
struct GS_OUT {
    float4 posH  : SV_POSITION;
    float3 posW : POSITION;
    float3 normalW : NORMAL;
    float2 uv : TEXCOORD;
    uint primID : SV_PrimitiveID;
};
```

```
cbuffer cbViewProjectionMatrix: register(b0) {
    matrix gmtxViewProjection : packoffset(c0);
};
```

Direct3D 파이프라인

- 기하 쉐이더의 예(나무 빌보드)

```
float4 PS(GS_OUT input) : SV_Target
{
    float4 cIllumination = Lighting(input.posW, input.normalW);
    float3 uvw = float3(input.uv, (input.primID % 4));
    float4 cTexture = gTreeTextureArray.Sample(gSamplerState, uvw);
    float4 cColor = cIllumination * cTexture;
    cColor.a = cTexture.a;
    return(cColor);
}
```

```
struct GS_OUT {
    float4 posH : SV_POSITION;
    float3 posW : POSITION;
    float3 normalW : NORMAL;
    float2 uv : TEXCOORD;
    uint primID : SV_PrimitiveID;
};
```

```
Texture2DArray gTreeTextureArray : register(t4);
SamplerState gSamplerState : register(s0);
```

```
DXGI_FORMAT Object.Sample(
    SamplerState S,
    float Location
    [, int Offset]
);
```

Object	텍스쳐 객체 데이터 형
S	샘플러 상태 객체
Location	텍스쳐 좌표(텍스쳐 객체에 따라 다름)
Offset	텍스쳐 좌표 오프셋, 텍스쳐 좌표에 더해짐

텍스쳐 좌표 데이터 형

Texture1D	float
Texture1DArray, Texture2D	float2
Texture2DArray, Texture3D, TextureCube	**float3**
TextureCubeArray	float4

텍스쳐 좌표 오프셋 데이터 형

Texture1D, Texture1DArray	int
Texture2D, Texture2DArray	int2
Texture3D	int3
TextureCube, TextureCubeArray	x

```
Texture2D gTreeTextureArray[4] : register(t4);

gTreeTextureArray[NonUniformResourceIndex(input.primID) % 4].Sample(gSamplerState, input.uv);
```