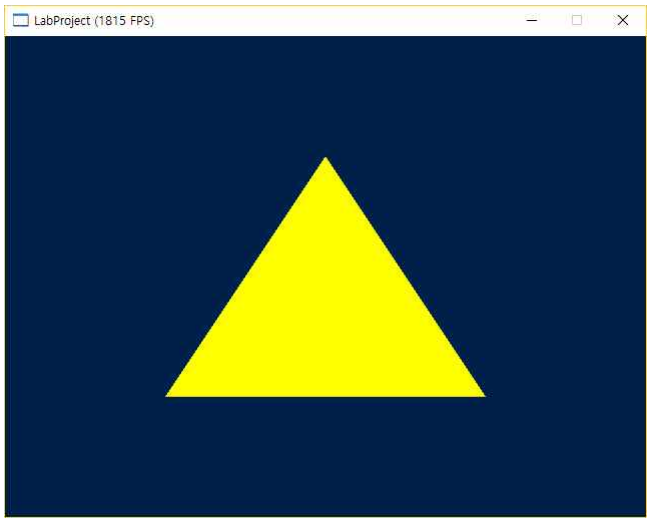
## □ 예제 프로그램 6: LabProject05(삼각형 그리기)

LabProject04 프로젝트를 기반으로 다음과 같이 삼각형을 그려본다.




① 새로운 프로젝트의 생성

먼저 새로운 프로젝트 LabProject05를 생성하자. “LabProjects" 솔루션을 열고 솔루션 탐색기에서 마우스 오른쪽 버튼으로 『솔루션 LabProjects』를 선택하고 메뉴에서 『추가』, 『새 프로젝트』를 차례로 선택한다. 그러면 『새 프로젝트 대화상자』가 나타난다. 그러면 프로젝트 이름 “LabProject05”를 입력하고 『확인』을 선택한다.

② LabProject05.cpp 파일 수정하기

이제 “LabProject05.cpp" 파일의 내용을 “LabProject04.cpp" 파일의 내용으로 바꾸도록 하자. “LabProject04.cpp" 파일의 내용 전체를 “LabProject05.cpp" 파일로 복사한다. 이제 “LabProject05.cpp" 파일에서 “LabProject04”을 “LabProject05”로 모두 바꾼다.
그리고 “LABPROJECT04”을 “LABPROJECT05”로 모두 바꾼다.

③ ”GameFramework.h“ 파일 수정하기

❶ 다음을 추가한다.

```
#include "Scene.h“
```

❷ MoveToNextFrame() 멤버 함수를 다음과 같이 선언한다.

```
public:
   void MoveToNextFrame();
```

❸ 후면버퍼마다 현재의 펜스 값을 관리하기 위하여 m_nFenceValue 멤버 변수를 다음과 같이 수정한다.

```
UINT64 m_nFenceValues[m_nSwapChainBuffers];
```

❹ 씬을 위한 멤버 변수를 다음과 같이 선언한다.

```
CScene *m_pScene;
```

④ "GameFramework.cpp" 파일 수정하기

❶ CGameFramework 클래스의 생성자에 다음을 추가한다.

```
CGameFramework::CGameFramework()
{
   ...
   for (int i = 0; i < m_nSwapChainBuffers; i++) m_nFenceValues[i] = 0;
   m_pScene = NULL;
   ...
}
```

❷ BuildObjects() 멤버 함수를 다음과 같이 수정한다.

```
void CGameFramework::BuildObjects()
{
   m_pScene = new CScene();
   if (m_pScene) m_pScene->BuildObjects(m_pd3dDevice);

   m_GameTimer.Reset();
}
```

❸ ReleaseObjects() 멤버 함수를 다음과 같이 수정한다.

```
void CGameFramework::ReleaseObjects()
{
   if (m_pScene) m_pScene->ReleaseObjects();
   if (m_pScene) delete m_pScene;
}
```

❹ AnimateObjects() 멤버 함수를 다음과 같이 수정한다.

```
void CGameFramework::AnimateObjects()
{
   if (m_pScene) m_pScene->AnimateObjects(m_GameTimer.GetTimeElapsed());
}
```

❺ WaitForGpuComplete() 멤버 함수를 다음과 같이 수정한다.

```
void CGameFramework::WaitForGpuComplete()
{
   UINT64 nFenceValue = ++m_nFenceValues[m_nSwapChainBufferIndex];
   HRESULT hResult = m_pd3dCommandQueue->Signal(m_pd3dFence, nFenceValue);
   if (m_pd3dFence->GetCompletedValue() < nFenceValue)
```

```
    {
        hResult = m_pd3dFence->SetEventOnCompletion(nFenceValue, m_hFenceEvent);
        ::WaitForSingleObject(m_hFenceEvent, INFINITE);
    }
}
```

❻ WaitForGpuComplete() 멤버 함수를 다음과 같이 정의한다.

```
void CGameFramework::MoveToNextFrame()
{
    m_nSwapChainBufferIndex = m_pdxgiSwapChain->GetCurrentBackBufferIndex();

    UINT64 nFenceValue = ++m_nFenceValues[m_nSwapChainBufferIndex];
    HRESULT hResult = m_pd3dCommandQueue->Signal(m_pd3dFence, nFenceValue);
    if (m_pd3dFence->GetCompletedValue() < nFenceValue)
    {
        hResult = m_pd3dFence->SetEventOnCompletion(nFenceValue, m_hFenceEvent);
        ::WaitForSingleObject(m_hFenceEvent, INFINITE);
    }
}
```

❼ FrameAdvance() 멤버 함수를 다음과 같이 수정한다.

```
void CGameFramework::FrameAdvance()
{
    m_GameTimer.Tick(0.0f);

    ProcessInput();

     AnimateObjects();

    HRESULT hResult = m_pd3dCommandAllocator->Reset();
    hResult = m_pd3dCommandList->Reset(m_pd3dCommandAllocator, NULL);

    m_pd3dCommandList->RSSetViewports(1, &m_d3dViewport);
    m_pd3dCommandList->RSSetScissorRects(1, &m_d3dScissorRect);

    D3D12_RESOURCE_BARRIER d3dResourceBarrier;
    ::ZeroMemory(&d3dResourceBarrier, sizeof(D3D12_RESOURCE_BARRIER));
    d3dResourceBarrier.Type = D3D12_RESOURCE_BARRIER_TYPE_TRANSITION;
    d3dResourceBarrier.Flags = D3D12_RESOURCE_BARRIER_FLAG_NONE;
    d3dResourceBarrier.Transition.pResource =
m_ppd3dRenderTargetBuffers[m_nSwapChainBufferIndex];
    d3dResourceBarrier.Transition.StateBefore = D3D12_RESOURCE_STATE_PRESENT;
    d3dResourceBarrier.Transition.StateAfter = D3D12_RESOURCE_STATE_RENDER_TARGET;
    d3dResourceBarrier.Transition.Subresource = D3D12_RESOURCE_BARRIER_ALL_SUBRESOURCES;
    m_pd3dCommandList->ResourceBarrier(1, &d3dResourceBarrier);

    D3D12_CPU_DESCRIPTOR_HANDLE d3dRtvCPUDescriptorHandle =
m_pd3dRtvDescriptorHeap->GetCPUDescriptorHandleForHeapStart();
    d3dRtvCPUDescriptorHandle.ptr += (m_nSwapChainBufferIndex *
m_nRtvDescriptorIncrementSize);
    D3D12_CPU_DESCRIPTOR_HANDLE d3dDsvCPUDescriptorHandle =
m_pd3dDsvDescriptorHeap->GetCPUDescriptorHandleForHeapStart();
```

```
   m_pd3dCommandList->OMSetRenderTargets(1, &d3dRtvCPUDescriptorHandle, FALSE,
&d3dDsvCPUDescriptorHandle);

   float pfClearColor[4] = { 0.0f, 0.125f, 0.3f, 1.0f };
   m_pd3dCommandList->ClearRenderTargetView(d3dRtvCPUDescriptorHandle, pfClearColor, 0,
NULL);

   m_pd3dCommandList->ClearDepthStencilView(d3dDsvCPUDescriptorHandle,
D3D12_CLEAR_FLAG_DEPTH | D3D12_CLEAR_FLAG_STENCIL, 1.0f, 0, 0, NULL);

   if (m_pScene) m_pScene->Render(m_pd3dCommandList);

   d3dResourceBarrier.Transition.StateBefore = D3D12_RESOURCE_STATE_RENDER_TARGET;
   d3dResourceBarrier.Transition.StateAfter = D3D12_RESOURCE_STATE_PRESENT;
   d3dResourceBarrier.Transition.Subresource = D3D12_RESOURCE_BARRIER_ALL_SUBRESOURCES;
   m_pd3dCommandList->ResourceBarrier(1, &d3dResourceBarrier);

   hResult = m_pd3dCommandList->Close();

   ID3D12CommandList *ppd3dCommandLists[] = { m_pd3dCommandList };
   m_pd3dCommandQueue->ExecuteCommandLists(_countof(ppd3dCommandLists),
ppd3dCommandLists);

   WaitForGpuComplete();

   m_pdxgiSwapChain->Present(0, 0);

   MoveToNextFrame();

   m_GameTimer.GetFrameRate(m_pszFrameRate + 12, 37);
   ::SetWindowText(m_hWnd, m_pszFrameRate);
}
```

⑤ ”CScene” 클래스 생성하기

게임 프로그램의 기본적 요소를 추가하기 위해 하나의 클래스(Class)를 만들도록 하자. 클래스의 이름은 “CScene”이다. 솔루션 탐색기에서 오른쪽 마우스 버튼으로 ”LabProject05“를 선택하고 『추가』, 『클래스』를 차례로 선택한다. 클래스 추가 대화상자가 나타나면 ”설치된 템플릿“에서 『C++』을 선택하고 『C++ 클래스』를 선택한 후 『추가』 버튼을 누른다. ”일반 C++ 클래스 마법사“가 나타나면 『클래스 이름』에 ”CScene”을 입력한다. 그러면 “.h 파일”의 이름이 “Scene.h” 그리고 “.cpp 파일”의 이름이 “Scene.cpp”가 된다. 『마침』을 선택하면 클래스 마법사가 두 개의 파일을 프로젝트에 추가하여 준다.

⑥ ”Scene.h” 클래스 선언하기

❶ “Scene.h” 파일에 다음을 추가한다.

```
#pragma once

#include "Timer.h"
```

❷ ”CScene” 클래스를 다음과 같이 수정한다.

```
class CScene
{
public:
    CScene();
    ~CScene();

   bool OnProcessingMouseMessage(HWND hWnd, UINT nMessageID, WPARAM wParam, LPARAM
lParam);
   bool OnProcessingKeyboardMessage(HWND hWnd, UINT nMessageID, WPARAM wParam, LPARAM
lParam);

   void CreateGraphicsRootSignature(ID3D12Device* pd3dDevice);
   void CreateGraphicsPipelineState(ID3D12Device* pd3dDevice);

   void BuildObjects(ID3D12Device *pd3dDevice);
   void ReleaseObjects();

   bool ProcessInput();
    void AnimateObjects(float fTimeElapsed);

   void PrepareRender(ID3D12GraphicsCommandList* pd3dCommandList);
    void Render(ID3D12GraphicsCommandList *pd3dCommandList);

   ID3D12RootSignature *m_pd3dGraphicsRootSignature = NULL;
//루트 시그너쳐를 나타내는 인터페이스 포인터이다.
   ID3D12PipelineState *m_pd3dPipelineState = NULL;
//파이프라인 상태를 나타내는 인터페이스 포인터이다.
};
```

⑦ ”Scene.cpp” 클래스 구현하기

“Scene.cpp” 파일을 다음과 같이 수정한다.

❶ 생성자를 다음과 같이 수정한다.

```
CScene::CScene()
{
}
```

❷ CreateGraphicsRootSignature() 함수를 다음과 같이 정의한다.

```
void CScene::CreateGraphicsRootSignature(ID3D12Device* pd3dDevice)
{
//루트 시그너쳐를 생성한다.
   D3D12_ROOT_SIGNATURE_DESC d3dRootSignatureDesc;
   ::ZeroMemory(&d3dRootSignatureDesc, sizeof(D3D12_ROOT_SIGNATURE_DESC));
   d3dRootSignatureDesc.NumParameters = 0;
   d3dRootSignatureDesc.pParameters = NULL;
   d3dRootSignatureDesc.NumStaticSamplers = 0;
   d3dRootSignatureDesc.pStaticSamplers = NULL;
   d3dRootSignatureDesc.Flags =
D3D12_ROOT_SIGNATURE_FLAG_ALLOW_INPUT_ASSEMBLER_INPUT_LAYOUT;
```

```
   ID3DBlob *pd3dSignatureBlob = NULL;
   ID3DBlob *pd3dErrorBlob = NULL;
   ::D3D12SerializeRootSignature(&d3dRootSignatureDesc, D3D_ROOT_SIGNATURE_VERSION_1,
&pd3dSignatureBlob, &pd3dErrorBlob);
   pd3dDevice->CreateRootSignature(0, pd3dSignatureBlob->GetBufferPointer(),
pd3dSignatureBlob->GetBufferSize(), __uuidof(ID3D12RootSignature), (void
**)&m_pd3dGraphicsRootSignature);
   if (pd3dSignatureBlob) pd3dSignatureBlob->Release();
   if (pd3dErrorBlob) pd3dErrorBlob->Release();
}
```

❸ CreateGraphicsPipelineState() 함수를 다음과 같이 정의한다.

```
void CScene::CreateGraphicsPipelineState(ID3D12Device* pd3dDevice)
{
//정점 셰이더와 픽셀 셰이더를 생성한다.
   ID3DBlob *pd3dVertexShaderBlob = NULL;
   ID3DBlob *pd3dPixelShaderBlob = NULL;

   UINT nCompileFlags = 0;
#if defined(_DEBUG)
   nCompileFlags = D3DCOMPILE_DEBUG | D3DCOMPILE_SKIP_OPTIMIZATION;
#endif

   D3DCompileFromFile(L"Shaders.hlsl", NULL, NULL, "VSMain", "vs_5_1", nCompileFlags, 0,
&pd3dVertexShaderBlob, NULL);
   D3DCompileFromFile(L"Shaders.hlsl", NULL, NULL, "PSMain", "ps_5_1", nCompileFlags, 0,
&pd3dPixelShaderBlob, NULL);

//래스터라이저 상태를 설정한다.
   D3D12_RASTERIZER_DESC d3dRasterizerDesc;
   ::ZeroMemory(&d3dRasterizerDesc, sizeof(D3D12_RASTERIZER_DESC));
   d3dRasterizerDesc.FillMode = D3D12_FILL_MODE_SOLID;
   d3dRasterizerDesc.CullMode = D3D12_CULL_MODE_BACK;
   d3dRasterizerDesc.FrontCounterClockwise = FALSE;
   d3dRasterizerDesc.DepthBias = 0;
   d3dRasterizerDesc.DepthBiasClamp = 0.0f;
   d3dRasterizerDesc.SlopeScaledDepthBias = 0.0f;
   d3dRasterizerDesc.DepthClipEnable = TRUE;
   d3dRasterizerDesc.MultisampleEnable = FALSE;
   d3dRasterizerDesc.AntialiasedLineEnable = FALSE;
   d3dRasterizerDesc.ForcedSampleCount = 0;
   d3dRasterizerDesc.ConservativeRaster = D3D12_CONSERVATIVE_RASTERIZATION_MODE_OFF;

//블렌드 상태를 설정한다.
   D3D12_BLEND_DESC d3dBlendDesc;
   ::ZeroMemory(&d3dBlendDesc, sizeof(D3D12_BLEND_DESC));
   d3dBlendDesc.AlphaToCoverageEnable = FALSE;
   d3dBlendDesc.IndependentBlendEnable = FALSE;
   d3dBlendDesc.RenderTarget[0].BlendEnable = FALSE;
   d3dBlendDesc.RenderTarget[0].LogicOpEnable = FALSE;
   d3dBlendDesc.RenderTarget[0].SrcBlend = D3D12_BLEND_ONE;
   d3dBlendDesc.RenderTarget[0].DestBlend = D3D12_BLEND_ZERO;
   d3dBlendDesc.RenderTarget[0].BlendOp = D3D12_BLEND_OP_ADD;
```

```
    d3dBlendDesc.RenderTarget[0].SrcBlendAlpha = D3D12_BLEND_ONE;
    d3dBlendDesc.RenderTarget[0].DestBlendAlpha = D3D12_BLEND_ZERO;
    d3dBlendDesc.RenderTarget[0].BlendOpAlpha = D3D12_BLEND_OP_ADD;
    d3dBlendDesc.RenderTarget[0].LogicOp = D3D12_LOGIC_OP_NOOP;
    d3dBlendDesc.RenderTarget[0].RenderTargetWriteMask = D3D12_COLOR_WRITE_ENABLE_ALL;

//그래픽 파이프라인 상태를 설정한다.
    D3D12_GRAPHICS_PIPELINE_STATE_DESC d3dPipelineStateDesc;
    ::ZeroMemory(&d3dPipelineStateDesc, sizeof(D3D12_GRAPHICS_PIPELINE_STATE_DESC));
    d3dPipelineStateDesc.pRootSignature = m_pd3dGraphicsRootSignature;
    d3dPipelineStateDesc.VS.pShaderBytecode = pd3dVertexShaderBlob->GetBufferPointer();
    d3dPipelineStateDesc.VS.BytecodeLength = pd3dVertexShaderBlob->GetBufferSize();
    d3dPipelineStateDesc.PS.pShaderBytecode = pd3dPixelShaderBlob->GetBufferPointer();
    d3dPipelineStateDesc.PS.BytecodeLength = pd3dPixelShaderBlob->GetBufferSize();
    d3dPipelineStateDesc.RasterizerState = d3dRasterizerDesc;
    d3dPipelineStateDesc.BlendState = d3dBlendDesc;
    d3dPipelineStateDesc.DepthStencilState.DepthEnable = FALSE;
    d3dPipelineStateDesc.DepthStencilState.StencilEnable = FALSE;
    d3dPipelineStateDesc.InputLayout.pInputElementDescs = NULL;
    d3dPipelineStateDesc.InputLayout.NumElements = 0;
    d3dPipelineStateDesc.SampleMask = UINT_MAX;
    d3dPipelineStateDesc.PrimitiveTopologyType = D3D12_PRIMITIVE_TOPOLOGY_TYPE_TRIANGLE;
    d3dPipelineStateDesc.NumRenderTargets = 1;
    d3dPipelineStateDesc.RTVFormats[0] = DXGI_FORMAT_R8G8B8A8_UNORM;
    d3dPipelineStateDesc.DSVFormat = DXGI_FORMAT_D24_UNORM_S8_UINT;
    d3dPipelineStateDesc.SampleDesc.Count = 1;
    d3dPipelineStateDesc.SampleDesc.Quality = 0;
    pd3dDevice->CreateGraphicsPipelineState(&d3dPipelineStateDesc,
__uuidof(ID3D12PipelineState), (void **)&m_pd3dPipelineState);

    if (pd3dVertexShaderBlob) pd3dVertexShaderBlob->Release();
    if (pd3dPixelShaderBlob) pd3dPixelShaderBlob->Release();
}
```

❹ BuildObjects() 함수와 ReleaseObjects() 함수를 다음과 같이 정의한다.

```
void CScene::BuildObjects(ID3D12Device *pd3dDevice)
{
}

void CScene::ReleaseObjects()
{
    if (m_pd3dGraphicsRootSignature) m_pd3dGraphicsRootSignature->Release();
    if (m_pd3dPipelineState) m_pd3dPipelineState->Release();
}
```

❺ OnProcessingMouseMessage() 함수와 OnProcessingKeyboardMessage() 함수를 다음과 같이 정의한다.

```
bool CScene::OnProcessingMouseMessage(HWND hWnd, UINT nMessageID, WPARAM wParam, LPARAM
lParam)
{
    return(false);
}
```

```
bool CScene::OnProcessingKeyboardMessage(HWND hWnd, UINT nMessageID, WPARAM wParam,
LPARAM lParam)
{
   return(false);
}
```

❻ ProcessInput() 함수와 AnimateObjects() 함수를 다음과 같이 정의한다.

```
bool CScene::ProcessInput()
{
   return(false);
}

void CScene::AnimateObjects(float fTimeElapsed)
{
}
```

❼ Render() 함수를 다음과 같이 정의한다.

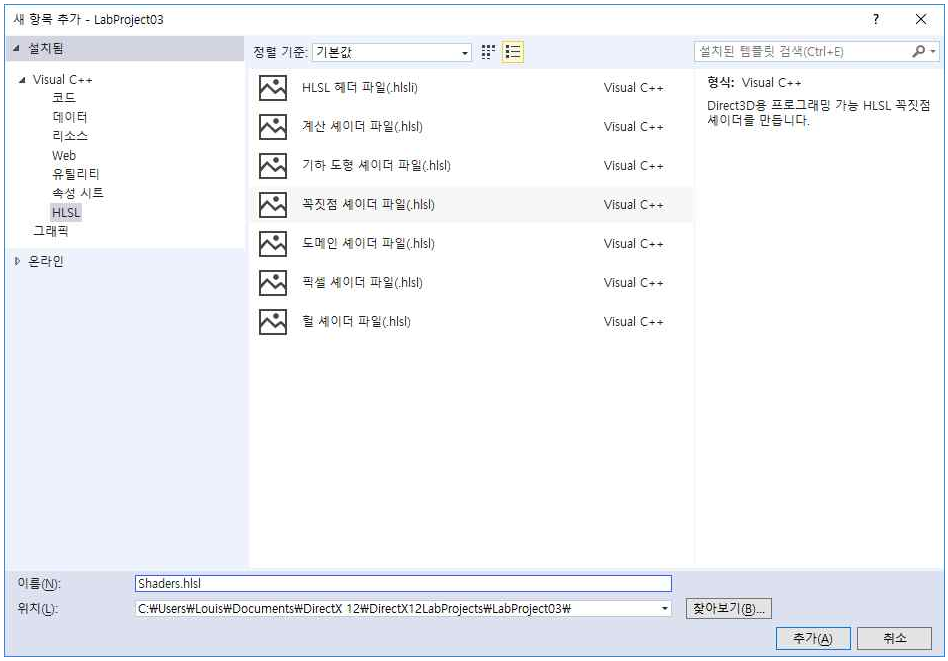
```
void CScene::PrepareRender(ID3D12GraphicsCommandList *pd3dCommandList)
{
//그래픽 루트 시그너쳐를 설정한다.
   pd3dCommandList->SetGraphicsRootSignature(m_pd3dGraphicsRootSignature);
//파이프라인 상태를 설정한다.
   pd3dCommandList->SetPipelineState(m_pd3dPipelineState);
//프리미티브 토폴로지(삼각형 리스트)를 설정한다.
   pd3dCommandList->IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
}
```

❽ Render() 함수를 다음과 같이 정의한다.

```
void CScene::Render(ID3D12GraphicsCommandList *pd3dCommandList)
{
//정점 3개를 사용하여 렌더링한다.
   pd3dCommandList->DrawInstanced(3, 1, 0, 0);
}
```

⑧ "Shaders.hlsl" 파일 생성하기

셰이더 프로그램을 포함하는 파일을 생성하도록 하자. 솔루션 탐색기에서 오른쪽 마우스 버튼으로 "LabProject05"를 선택하고 『추가』, 『새 항목』를 차례로 선택한다. 다음 그림과 같이 "새 항목 추가" 대화상자가 나타나면 "Visual C++"에서 『HLSL』을 선택하고 『꼭지점 셰이더 파일』을 선택한 후 『이름』에 "Shaders.hlsl"을 입력하고 『추가』 버튼을 누른다.




❶ 정점 쉐이더 함수 VSMain()을 다음과 같이 수정한다.

```
//정점 셰이더를 정의한다.
float4 VSMain(uint nVertexID : SV_VertexID) : SV_POSITION
{
   float4 output;

//프리미티브(삼각형)를 구성하는 정점의 인덱스(SV_VertexID)에 따라 정점을 반환한다.
//정점의 위치 좌표는 변환이 된 좌표(SV_POSITION)이다. 즉, 투영좌표계의 좌표이다.
   if (nVertexID == 0) output = float4(0.0, 0.5, 0.5, 1.0);
   else if (nVertexID == 1) output = float4(0.5, -0.5, 0.5, 1.0);
   else if (nVertexID == 2) output = float4(-0.5, -0.5, 0.5, 1.0);

   return(output);
}
```

❷ 픽셀 쉐이더 함수 PSMain()을을 다음과 같이 수정한다.

```
//픽셀 셰이더를 정의한다.
float4 PSMain(float4 input : SV_POSITION) : SV_TARGET
{
//프리미티브의 모든 픽셀의 색상을 노란색으로 반환한다.
   return(float4(1.0f, 1.0f, 0.0f, 1.0f));    //Yellow
}
```

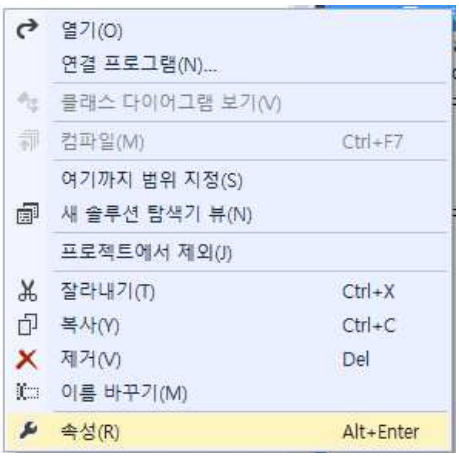
⑨ 프로젝트 빌드 하기

프로젝트 “LabProject05”를 빌드를 하면 다음과 같은 컴파일 오류가 Visual Studio 출력창에 출력될 것이다. 이것은 확장자자 "hlsl"인 파일은 Visual Studio가 기본적으로 HLSL 컴파일러인 fxc.exe로 컴파일

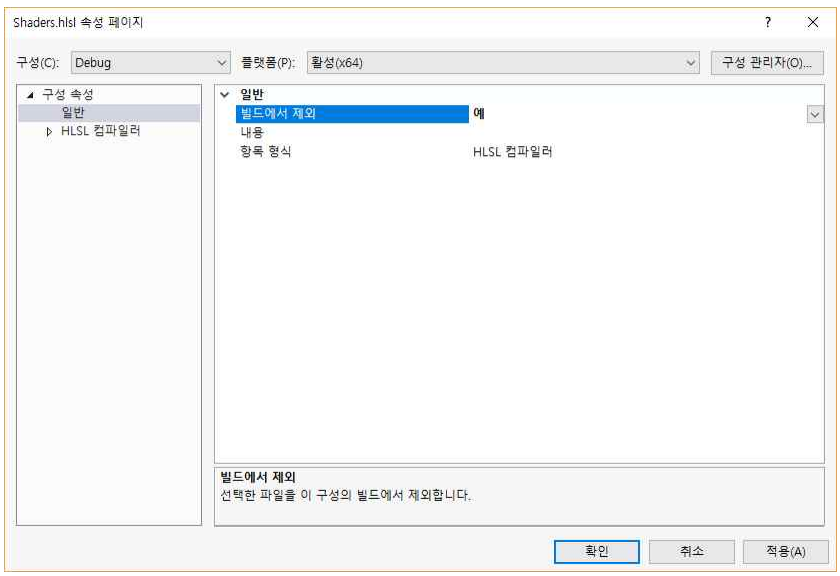
을 하기 때문이다.

```
1>------ 빌드시작: 프로젝트: LabProject05, 구성: Debug x64 ------
1>FXC : error X3501: 'main': entrypoint not found
1>
1>  compilation failed; no code produced
========== 빌드: 성공0, 실패1, 최신0, 생략0 ==========
```

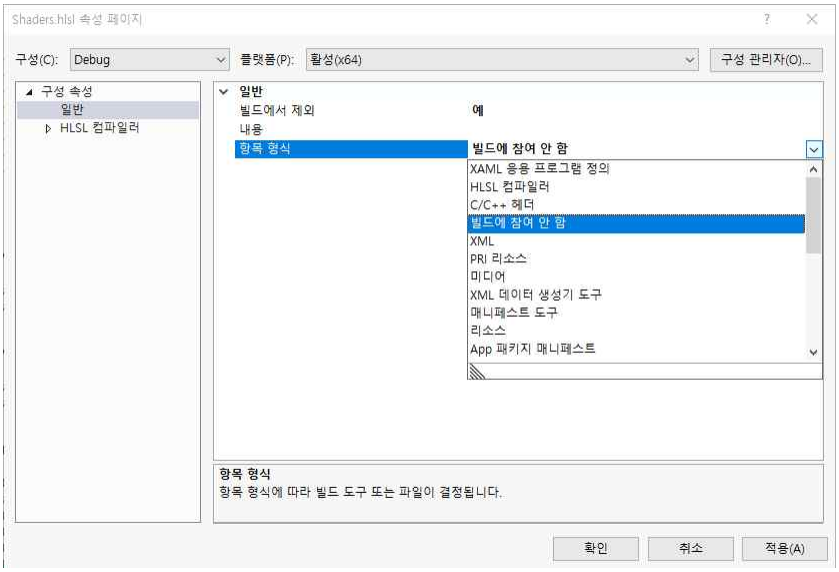
이러한 오류를 발생시키지 않으려면 새로 생성한 “Shaders.hlsl" 파일을 Visual Studio가 컴파일하지 않도록 설정하면 된다. 솔루션 탐색기에서 “Shaders.hlsl" 파일을 오른쪽 마우스 버튼으로 누르고 다음 그림과 같이 ”속성“ 메뉴를 선택한다.




그러면 선택한 파일의 ”속성 페이지”가 나타난다. 여기에서 “빌드에서 제외”를 선택하면 오른쪽에 화살표가 나타난다. 이 화살표를 누르면 “예”, “아니요”를 선택할 수 있는데 “**예**”를 선택한다. 이것은 “Shaders.hlsl" 파일을 Visual Studio가 컴파일하지 않도록 하는 것이다.




또는 다음 그림과 같이 ”항목 형식“에서 ”**빌드에 참여 안 함**“으로 선택하여도 된다.


Shaders.hlsl 속성 페이지
구성(C): Debug
플랫폼(P): 활성(x64)
구성 관리자(O)...
구성 속성
일반
HLSL 컴파일러
일반
빌드에서 제외
예
내용
항목 형식
빌드에 참여 안 함
XAML 응용 프로그램 정의
HLSL 컴파일러
C/C++ 헤더
빌드에 참여 안 함
XML
PRI 리소스
미디어
XML 데이터 생성기 도구
매니페스트 도구
리소스
App 패키지 매니페스트
항목 형식
항목 형식에 따라 빌드 도구 또는 파일이 결정됩니다.
확인
취소
적용(A)