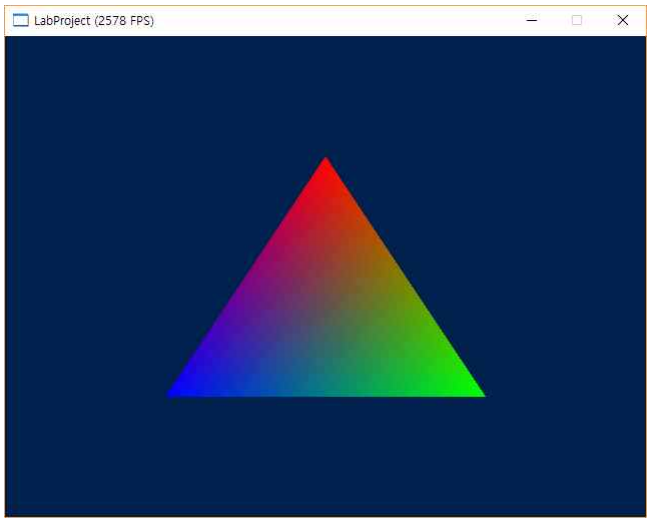
□ 예제 프로그램 7: LabProject06(삼각형 그리기)

LabProject05 프로젝트를 기반으로 다음과 같은 삼각형을 그려본다.



① 새로운 프로젝트의 생성

먼저 새로운 프로젝트 LabProject06를 생성한다. “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 “LabProject06”를 입력하고 『확인』을 선택한다.

❶ 파일 탐색기에서 프로젝트 “LabProject05” 폴더의 다음 파일을 선택하여 프로젝트 “LabProject06” 폴더에 복사한다.

- GameFramework.h
- GameFramework.cpp
- Scene.h
- Scene.cpp
- stdafx.h
- Timer.h
- Timer.cpp

❷ 위에서 복사한 파일을 Visual Studio 솔루션 탐색기에서 프로젝트 “LabProject06”에 추가한다. 오른쪽 마우스 버튼으로 『LabProject06』을 선택하고 『추가』, 『기존 항목』를 차례로 선택한다. 그러면 “기존 항목 추가” 대화 상자가 나타난다. 다음 파일들을 마우스로 선택(Ctrl+선택)하여 『추가』를 누르면 선택된 파일들이 프로젝트 “LabProject06”에 추가된다.

- GameFramework.h
- GameFramework.cpp
- Scene.h
- Scene.cpp
- Timer.h
- Timer.cpp

② LabProject06.cpp 파일 수정하기

이제 “LabProject06.cpp" 파일의 내용을 “LabProject05.cpp" 파일의 내용으로 바꾸도록 하자. “LabProject05.cpp" 파일의 내용 전체를 “LabProject06.cpp" 파일로 복사한다. 이제 “LabProject06.cpp" 파일에서 “LabProject05”을 “LabProject06”로 모두 바꾼다.
그리고 “LABPROJECT05”을 “LABPROJECT06”으로 모두 바꾼다.

③ ”stdafx.h“ 파일과 ”stdafx.cpp“ 파일 수정하기

❶ ”stdafx.h“ 파일에 다음을 추가한다.

```
extern ID3D12Resource *CreateBufferResource(ID3D12Device *pd3dDevice,
ID3D12GraphicsCommandList *pd3dCommandList, void *pData, UINT nBytes, D3D12_HEAP_TYPE
d3dHeapType = D3D12_HEAP_TYPE_UPLOAD, D3D12_RESOURCE_STATES d3dResourceStates =
D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER, ID3D12Resource **ppd3dUploadBuffer =
NULL);
```

❷ ”stdafx.cpp“ 파일에 다음을 추가한다.

```
/*버퍼 리소스를 생성하는 함수이다. 버퍼의 힙 유형에 따라 버퍼 리소스를 생성하고 초기화 데이터가 있으면 초기화
한다.*/
ID3D12Resource *CreateBufferResource(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, void *pData, UINT nBytes, D3D12_HEAP_TYPE d3dHeapType,
D3D12_RESOURCE_STATES d3dResourceStates, ID3D12Resource **ppd3dUploadBuffer)
{
   ID3D12Resource *pd3dBuffer = NULL;

   D3D12_HEAP_PROPERTIES d3dHeapPropertiesDesc;
   ::ZeroMemory(&d3dHeapPropertiesDesc, sizeof(D3D12_HEAP_PROPERTIES));
   d3dHeapPropertiesDesc.Type = d3dHeapType;
   d3dHeapPropertiesDesc.CPUPageProperty = D3D12_CPU_PAGE_PROPERTY_UNKNOWN;
   d3dHeapPropertiesDesc.MemoryPoolPreference = D3D12_MEMORY_POOL_UNKNOWN;
   d3dHeapPropertiesDesc.CreationNodeMask = 1;
   d3dHeapPropertiesDesc.VisibleNodeMask = 1;

   D3D12_RESOURCE_DESC d3dResourceDesc;
   ::ZeroMemory(&d3dResourceDesc, sizeof(D3D12_RESOURCE_DESC));
   d3dResourceDesc.Dimension = D3D12_RESOURCE_DIMENSION_BUFFER;
   d3dResourceDesc.Alignment = 0;
```

```
   d3dResourceDesc.Width = nBytes;
   d3dResourceDesc.Height = 1;
   d3dResourceDesc.DepthOrArraySize = 1;
   d3dResourceDesc.MipLevels = 1;
   d3dResourceDesc.Format = DXGI_FORMAT_UNKNOWN;
   d3dResourceDesc.SampleDesc.Count = 1;
   d3dResourceDesc.SampleDesc.Quality = 0;
   d3dResourceDesc.Layout = D3D12_TEXTURE_LAYOUT_ROW_MAJOR;
   d3dResourceDesc.Flags = D3D12_RESOURCE_FLAG_NONE;

   D3D12_RESOURCE_STATES d3dResourceInitialStates = D3D12_RESOURCE_STATE_COPY_DEST;
   if (d3dHeapType == D3D12_HEAP_TYPE_UPLOAD) d3dResourceInitialStates =
D3D12_RESOURCE_STATE_GENERIC_READ;
   else if (d3dHeapType == D3D12_HEAP_TYPE_READBACK) d3dResourceInitialStates =
D3D12_RESOURCE_STATE_COPY_DEST;

   HRESULT hResult = pd3dDevice->CreateCommittedResource(&d3dHeapPropertiesDesc,
D3D12_HEAP_FLAG_NONE, &d3dResourceDesc, d3dResourceInitialStates, NULL,
__uuidof(ID3D12Resource), (void **)&pd3dBuffer);

   if (pData)
   {
      switch (d3dHeapType)
      {
      case D3D12_HEAP_TYPE_DEFAULT:
      {
         if (ppd3dUploadBuffer)
         {
//업로드 버퍼를 생성한다.
            d3dHeapPropertiesDesc.Type = D3D12_HEAP_TYPE_UPLOAD;
            pd3dDevice->CreateCommittedResource(&d3dHeapPropertiesDesc,
D3D12_HEAP_FLAG_NONE, &d3dResourceDesc, D3D12_RESOURCE_STATE_GENERIC_READ, NULL,
__uuidof(ID3D12Resource), (void **)ppd3dUploadBuffer);

//업로드 버퍼를 매핑하여 초기화 데이터를 업로드 버퍼에 복사한다.
            D3D12_RANGE d3dReadRange = { 0, 0 };
            UINT8 *pBufferDataBegin = NULL;
            (*ppd3dUploadBuffer)->Map(0, &d3dReadRange, (void **)&pBufferDataBegin);
            memcpy(pBufferDataBegin, pData, nBytes);
            (*ppd3dUploadBuffer)->Unmap(0, NULL);

//업로드 버퍼의 내용을 디폴트 버퍼에 복사한다.
            pd3dCommandList->CopyResource(pd3dBuffer, *ppd3dUploadBuffer);

            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 = pd3dBuffer;
            d3dResourceBarrier.Transition.StateBefore = D3D12_RESOURCE_STATE_COPY_DEST;
            d3dResourceBarrier.Transition.StateAfter = d3dResourceStates;
            d3dResourceBarrier.Transition.Subresource =
D3D12_RESOURCE_BARRIER_ALL_SUBRESOURCES;
            pd3dCommandList->ResourceBarrier(1, &d3dResourceBarrier);
         }
```

```
            break;
        }
        case D3D12_HEAP_TYPE_UPLOAD:
        {
            D3D12_RANGE d3dReadRange = { 0, 0 };
            UINT8 *pBufferDataBegin = NULL;
            pd3dBuffer->Map(0, &d3dReadRange, (void **)&pBufferDataBegin);
            memcpy(pBufferDataBegin, pData, nBytes);
            pd3dBuffer->Unmap(0, NULL);
            break;
        }
        case D3D12_HEAP_TYPE_READBACK:
            break;
        }
    }
    return(pd3dBuffer);
}
```

④ "CShader", "CMesh", "CGameObject" 클래스 생성하기

게임 프로그램의 기본적 요소를 추가하기 위해 클래스(Class)들을 만들도록 하자. 클래스의 이름은 "CShader", "CMesh", 그리고 "CGameObject"이다.
솔루션 탐색기에서 오른쪽 마우스 버튼으로 "LabProject06"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 "CShader"를 입력한다. 그러면 ".h 파일"의 이름이 "Shader.h" 그리고 ".cpp 파일"의 이름이 "Shader.cpp"가 된다. 『마침』을 선택하면 클래스 마법사가 두 개의 파일을 프로젝트에 추가하여 준다. 같은 방법으로 "CMesh" 클래스와 "CGameObject" 클래스를 생성한다.

⑤ "CMesh" 클래스 선언하기

"Mesh.h" 파일을 다음과 같이 수정한다.

❶ "Mesh.h" 파일에 다음을 추가한다.

```
#pragma once
```

❷ "CVertex" 클래스와 "CDiffusedVertex" 클래스를 다음과 같이 선언한다.

```
//정점을 표현하기 위한 클래스를 선언한다.
class CVertex
{
protected:
//정점의 위치 벡터이다(모든 정점은 최소한 위치 벡터를 가져야 한다).
    XMFLOAT3 m_xmf3Position;
```

```
public:
   CVertex() { m_xmf3Position = XMFLOAT3(0.0f, 0.0f, 0.0f); }
   CVertex(XMFLOAT3 xmf3Position) { m_xmf3Position = xmf3Position; }
   ~CVertex() { }
};

class CDiffusedVertex : public CVertex
{
protected:
//정점의 색상이다.
    XMFLOAT4 m_xmf4Diffuse;

public:
   CDiffusedVertex() { m_xmf3Position = XMFLOAT3(0.0f, 0.0f, 0.0f); m_xmf4Diffuse =
XMFLOAT4(0.0f, 0.0f, 0.0f, 0.0f); }
   CDiffusedVertex(float x, float y, float z, XMFLOAT4 xmf4Diffuse) { m_xmf3Position =
XMFLOAT3(x, y, z); m_xmf4Diffuse = xmf4Diffuse; }
   CDiffusedVertex(XMFLOAT3 xmf3Position, XMFLOAT4 xmf4Diffuse) { m_xmf3Position =
xmf3Position; m_xmf4Diffuse = xmf4Diffuse; }
   ~CDiffusedVertex() { }
};
```

❸ ”CMesh” 클래스를 다음과 같이 선언한다.

```
class CMesh
{
public:
    CMesh(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList *pd3dCommandList);
    virtual ~CMesh();

private:
   int m_nReferences = 0;

public:
   void AddRef() { m_nReferences++; }
   void Release() { if (--m_nReferences <= 0) delete this; }

   void ReleaseUploadBuffers();

protected:
   ID3D12Resource *m_pd3dVertexBuffer = NULL;
   ID3D12Resource *m_pd3dVertexUploadBuffer = NULL;

   D3D12_VERTEX_BUFFER_VIEW m_d3dVertexBufferView;

   D3D12_PRIMITIVE_TOPOLOGY m_d3dPrimitiveTopology = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST;
   UINT m_nSlot = 0;
   UINT m_nVertices = 0;
   UINT m_nStride = 0;
   UINT m_nOffset = 0;

public:
   virtual void Render(ID3D12GraphicsCommandList *pd3dCommandList);
};
```

❹ “CTriangleMesh” 클래스를 다음과 같이 선언한다.

```
class CTriangleMesh : public CMesh
{
public:
    CTriangleMesh(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList *pd3dCommandList);
    virtual ~CTriangleMesh() { }
};
```

⑥ ”CMesh” 클래스 구현하기

“Mesh.cpp” 파일을 다음과 같이 수정한다.

❶ CMesh 클래스의 소멸자를 다음과 같이 수정한다.

```
CMesh::~CMesh()
{
    if (m_pd3dVertexBuffer) m_pd3dVertexBuffer->Release();
    if (m_pd3dVertexUploadBuffer) m_pd3dVertexUploadBuffer->Release();
}
```

❷ CMesh 클래스의 ReleaseUploadBuffers() 함수를 다음과 같이 정의한다.

```
void CMesh::ReleaseUploadBuffers()
{
//정점 버퍼를 위한 업로드 버퍼를 소멸시킨다.
    if (m_pd3dVertexUploadBuffer) m_pd3dVertexUploadBuffer->Release();
    m_pd3dVertexUploadBuffer = NULL;
};
```

❸ CMesh 클래스의 Render() 함수를 다음과 같이 정의한다.

```
void CMesh::Render(ID3D12GraphicsCommandList *pd3dCommandList)
{
//메쉬의 프리미티브 유형을 설정한다.
    pd3dCommandList->IASetPrimitiveTopology(m_d3dPrimitiveTopology);
//메쉬의 정점 버퍼 뷰를 설정한다.
    pd3dCommandList->IASetVertexBuffers(m_nSlot, 1, &m_d3dVertexBufferView);
//메쉬의 정점 버퍼 뷰를 렌더링한다(파이프라인(입력 조립기)을 작동하게 한다).
    pd3dCommandList->DrawInstanced(m_nVertices, 1, m_nOffset, 0);
}
```

❸ CTriangleMesh 클래스의 생성자를 다음과 같이 정의한다.

```
CTriangleMesh::CTriangleMesh(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList) : CMesh(pd3dDevice, pd3dCommandList)
{
```

```
//삼각형 메쉬를 정의한다.
   m_nVertices = 3;
   m_nStride = sizeof(CDiffusedVertex);
   m_d3dPrimitiveTopology = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST;

   CDiffusedVertex pVertices[3];
   pVertices[0] = CDiffusedVertex(XMFLOAT3(0.0f, 0.5f, 0.0f), XMFLOAT4(1.0f, 0.0f, 0.0f,
1.0f));
   pVertices[1] = CDiffusedVertex(XMFLOAT3(0.5f, -0.5f, 0.0f), XMFLOAT4(0.0f, 1.0f, 0.0f,
1.0f));
   pVertices[2] = CDiffusedVertex(XMFLOAT3(-0.5f, -0.5f, 0.0f), XMFLOAT4(Colors::Blue));

//삼각형 메쉬를 리소스(정점 버퍼)로 생성한다.
   m_pd3dVertexBuffer = ::CreateBufferResource(pd3dDevice, pd3dCommandList, pVertices,
m_nStride * m_nVertices, D3D12_HEAP_TYPE_DEFAULT,
D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER, &m_pd3dVertexUploadBuffer);

//정점 버퍼 뷰를 생성한다.
   m_d3dVertexBufferView.BufferLocation = m_pd3dVertexBuffer->GetGPUVirtualAddress();
   m_d3dVertexBufferView.StrideInBytes = m_nStride;
   m_d3dVertexBufferView.SizeInBytes = m_nStride * m_nVertices;
}
```

⑦ ”CGameObject” 클래스 선언하기

“GameObject.h” 파일을 다음과 같이 수정한다.

❶ “GameObject.h” 파일에 다음을 추가한다.

```
#pragma once

#include "Mesh.h"

class CShader;
```

❷ ”CGameObject” 클래스를 다음과 같이 선언한다.

```
class CGameObject
{
public:
   CGameObject();
    virtual ~CGameObject();

private:
   int m_nReferences = 0;

public:
   void AddRef() { m_nReferences++; }
   void Release() { if (--m_nReferences <= 0) delete this; }

protected:
```

```
    XMFLOAT4X4 m_xmf4x4World;
    CMesh *m_pMesh = NULL;

    CShader *m_pShader = NULL;

public:
    void ReleaseUploadBuffers();

    virtual void SetMesh(CMesh *pMesh);
    virtual void SetShader(CShader *pShader);

    virtual void Animate(float fTimeElapsed);

    virtual void OnPrepareRender();
    virtual void Render(ID3D12GraphicsCommandList *pd3dCommandList);
};
```

⑧ ”CGameObject” 클래스 구현하기

“GameObject.cpp” 파일을 다음과 같이 수정한다.

❶ “GameObject.cpp” 파일에 다음을 추가한다.

```
#include "Shader.h“
```

❷ CGameObject 클래스의 생성자와 소멸자를 다음과 같이 수정한다.

```
CGameObject::CGameObject()
{
    XMStoreFloat4x4(&m_xmf4x4World, XMMatrixIdentity());
}

CGameObject::~CGameObject()
{
    if (m_pMesh) m_pMesh->Release();
    if (m_pShader)
    {
        m_pShader->ReleaseShaderVariables();
        m_pShader->Release();
    }
}
```

❸ CGameObject 클래스의 SetShader() 함수와 SetMesh() 함수를 다음과 같이 정의한다.

```
void CGameObject::SetShader(CShader *pShader)
{
    if (m_pShader) m_pShader->Release();
    m_pShader = pShader;
    if (m_pShader) m_pShader->AddRef();
}
```

```
void CGameObject::SetMesh(CMesh *pMesh)
{
   if (m_pMesh) m_pMesh->Release();
   m_pMesh = pMesh;
   if (m_pMesh) m_pMesh->AddRef();
}
```

❹ CGameObject 클래스의 ReleaseUploadBuffers() 함수를 다음과 같이 정의한다.

```
void CGameObject::ReleaseUploadBuffers()
{
//정점 버퍼를 위한 업로드 버퍼를 소멸시킨다.
   if (m_pMesh) m_pMesh->ReleaseUploadBuffers();
}
```

❺ CGameObject 클래스의 Animate() 함수를 다음과 같이 정의한다.

```
void CGameObject::Animate(float fTimeElapsed)
{
}
```

❻ CGameObject 클래스의 OnPrepareRender() 함수를 다음과 같이 정의한다.

```
void CGameObject::OnPrepareRender()
{
}
```

❼ CGameObject 클래스의 Render() 함수를 다음과 같이 정의한다.

```
void CGameObject::Render(ID3D12GraphicsCommandList *pd3dCommandList)
{
   OnPrepareRender();

//게임 객체에 셰이더 객체가 연결되어 있으면 셰이더 상태 객체를 설정한다.
   if (m_pShader) m_pShader->Render(pd3dCommandList);

//게임 객체에 메쉬가 연결되어 있으면 메쉬를 렌더링한다.
   if (m_pMesh) m_pMesh->Render(pd3dCommandList);
}
```

⑨ "CShader" 클래스 선언하기

"Shader.h" 파일을 다음과 같이 수정한다.

❶ "Shader.h" 파일에 다음을 추가한다.

```
#pragma once
```

```
#include "Object.h"
```

❷ ”CShader” 클래스를 다음과 같이 수정한다.

```
//셰이더 소스 코드를 컴파일하고 그래픽스 상태 객체를 생성한다.
class CShader
{
public:
    CShader();
    virtual ~CShader();

private:
   int m_nReferences = 0;

public:
   void AddRef() { m_nReferences++; }
   void Release() { if (--m_nReferences <= 0) delete this; }

   virtual D3D12_INPUT_LAYOUT_DESC CreateInputLayout();
   virtual D3D12_RASTERIZER_DESC CreateRasterizerState();
   virtual D3D12_BLEND_DESC CreateBlendState();
   virtual D3D12_DEPTH_STENCIL_DESC CreateDepthStencilState();

   virtual D3D12_SHADER_BYTECODE CreateVertexShader(ID3DBlob **ppd3dShaderBlob);
   virtual D3D12_SHADER_BYTECODE CreatePixelShader(ID3DBlob **ppd3dShaderBlob);

   D3D12_SHADER_BYTECODE CompileShaderFromFile(WCHAR *pszFileName, LPCSTR pszShaderName,
LPCSTR pszShaderProfile, ID3DBlob **ppd3dShaderBlob);

   virtual void CreateShader(ID3D12Device *pd3dDevice, ID3D12RootSignature
*pd3dRootSignature);

   virtual void CreateShaderVariables(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList) { }
   virtual void UpdateShaderVariables(ID3D12GraphicsCommandList *pd3dCommandList) { }
   virtual void ReleaseShaderVariables() { }

   virtual void ReleaseUploadBuffers();

   virtual void BuildObjects(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, void *pContext = NULL);
   virtual void AnimateObjects(float fTimeElapsed);
   virtual void ReleaseObjects();

   virtual void OnPrepareRender(ID3D12GraphicsCommandList *pd3dCommandList);
   virtual void Render(ID3D12GraphicsCommandList *pd3dCommandList);

protected:
//셰이더가 포함하는 게임 객체들의 리스트(배열)이다.
   CGameObject **m_ppObjects = NULL;
   int m_nObjects = 0;

//파이프라인 상태 객체들의 리스트(배열)이다.
```

```
    ID3D12PipelineState **m_ppd3dPipelineStates = NULL;
    int m_nPipelineStates = 0;
};
```

⑩ "CShader" 클래스 구현하기

"Shader.cpp" 파일을 다음과 같이 수정한다.

❶ 소멸자를 다음과 같이 수정한다.

```
CShader::~CShader()
{
    if (m_ppd3dPipelineStates)
    {
        for (int i = 0; i < m_nPipelineStates; i++) if (m_ppd3dPipelineStates[i])
m_ppd3dPipelineStates[i]->Release();
        delete[] m_ppd3dPipelineStates;
    }
}
```

❷ CreateRasterizerState() 함수를 다음과 같이 정의한다.

```
//래스터라이저 상태를 설정하기 위한 구조체를 반환한다.
D3D12_RASTERIZER_DESC CShader::CreateRasterizerState()
{
    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;

    return(d3dRasterizerDesc);
}
```

❸ CreateDepthStencilState() 함수를 다음과 같이 정의한다.

```
//깊이-스텐실 검사를 위한 상태를 설정하기 위한 구조체를 반환한다.
D3D12_DEPTH_STENCIL_DESC CShader::CreateDepthStencilState()
{
    D3D12_DEPTH_STENCIL_DESC d3dDepthStencilDesc;
    ::ZeroMemory(&d3dDepthStencilDesc, sizeof(D3D12_DEPTH_STENCIL_DESC));
    d3dDepthStencilDesc.DepthEnable = TRUE;
```

```
    d3dDepthStencilDesc.DepthWriteMask = D3D12_DEPTH_WRITE_MASK_ALL;
    d3dDepthStencilDesc.DepthFunc = D3D12_COMPARISON_FUNC_LESS;
    d3dDepthStencilDesc.StencilEnable = FALSE;
    d3dDepthStencilDesc.StencilReadMask = 0x00;
    d3dDepthStencilDesc.StencilWriteMask = 0x00;
    d3dDepthStencilDesc.FrontFace.StencilFailOp = D3D12_STENCIL_OP_KEEP;
    d3dDepthStencilDesc.FrontFace.StencilDepthFailOp = D3D12_STENCIL_OP_KEEP;
    d3dDepthStencilDesc.FrontFace.StencilPassOp = D3D12_STENCIL_OP_KEEP;
    d3dDepthStencilDesc.FrontFace.StencilFunc = D3D12_COMPARISON_FUNC_NEVER;
    d3dDepthStencilDesc.BackFace.StencilFailOp = D3D12_STENCIL_OP_KEEP;
    d3dDepthStencilDesc.BackFace.StencilDepthFailOp = D3D12_STENCIL_OP_KEEP;
    d3dDepthStencilDesc.BackFace.StencilPassOp = D3D12_STENCIL_OP_KEEP;
    d3dDepthStencilDesc.BackFace.StencilFunc = D3D12_COMPARISON_FUNC_NEVER;

    return(d3dDepthStencilDesc);
}
```

❹ CreateBlendState() 함수를 다음과 같이 정의한다.

```
//블렌딩 상태를 설정하기 위한 구조체를 반환한다.
D3D12_BLEND_DESC CShader::CreateBlendState()
{
    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;

    return(d3dBlendDesc);
}
```

❺ CreateInputLayout() 함수를 다음과 같이 정의한다.

```
//입력 조립기에게 정점 버퍼의 구조를 알려주기 위한 구조체를 반환한다.
D3D12_INPUT_LAYOUT_DESC CShader::CreateInputLayout()
{
    UINT nInputElementDescs = 2;
    D3D12_INPUT_ELEMENT_DESC *pd3dInputElementDescs = new
D3D12_INPUT_ELEMENT_DESC[nInputElementDescs];

//정점은 위치 벡터(POSITION)와 색상(COLOR)을 가진다.
    pd3dInputElementDescs[0] = { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0,
D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, 0 };
```

```
   pd3dInputElementDescs[1] = { "COLOR", 0, DXGI_FORMAT_R32G32B32A32_FLOAT, 0, 12,
D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, 0 };

   D3D12_INPUT_LAYOUT_DESC d3dInputLayoutDesc;
   d3dInputLayoutDesc.pInputElementDescs = pd3dInputElementDescs;
   d3dInputLayoutDesc.NumElements = nInputElementDescs;

   return(d3dInputLayoutDesc);
}
```

❻ CreateVertexShader(), CreatePixelShader(), CompileShaderFromFile() 함수를 다음과 같이 정의한다.

```
//정점 셰이더 바이트 코드를 생성(컴파일)한다.
D3D12_SHADER_BYTECODE CShader::CreateVertexShader(ID3DBlob **ppd3dShaderBlob)
{
   return(CompileShaderFromFile(L"Shaders.hlsl", "VSMain", "vs_5_1", ppd3dShaderBlob));
}

//픽셀 셰이더 바이트 코드를 생성(컴파일)한다.
D3D12_SHADER_BYTECODE CShader::CreatePixelShader(ID3DBlob **ppd3dShaderBlob)
{
   return(CompileShaderFromFile(L"Shaders.hlsl", "PSMain", "ps_5_1", ppd3dShaderBlob));
}

//셰이더 소스 코드를 컴파일하여 바이트 코드 구조체를 반환한다.
D3D12_SHADER_BYTECODE CShader::CompileShaderFromFile(WCHAR *pszFileName, LPCSTR
pszShaderName, LPCSTR pszShaderProfile, ID3DBlob **ppd3dShaderBlob)
{
   UINT nCompileFlags = 0;
#if defined(_DEBUG)
   nCompileFlags = D3DCOMPILE_DEBUG | D3DCOMPILE_SKIP_OPTIMIZATION;
#endif

   ::D3DCompileFromFile(pszFileName, NULL, NULL, pszShaderName, pszShaderProfile,
nCompileFlags, 0, ppd3dShaderBlob, NULL);

   D3D12_SHADER_BYTECODE d3dShaderByteCode;
   d3dShaderByteCode.BytecodeLength = (*ppd3dShaderBlob)->GetBufferSize();
   d3dShaderByteCode.pShaderBytecode = (*ppd3dShaderBlob)->GetBufferPointer();

   return(d3dShaderByteCode);
}
```

❼ CreateShader() 함수를 다음과 같이 정의한다.

```
//그래픽스 파이프라인 상태 객체를 생성한다.
void CShader::CreateShader(ID3D12Device *pd3dDevice, ID3D12RootSignature
*pd3dRootSignature)
{
//그래픽스 파이프라인 상태 객체 배열을 생성한다.
```

```
    m_nPipelineStates = 1;
    m_ppd3dPipelineStates = new ID3D12PipelineState*[m_nPipelineStates];

    ID3DBlob *pd3dVertexShaderBlob = NULL, *pd3dPixelShaderBlob = NULL;

    D3D12_GRAPHICS_PIPELINE_STATE_DESC d3dPipelineStateDesc;
    ::ZeroMemory(&d3dPipelineStateDesc, sizeof(D3D12_GRAPHICS_PIPELINE_STATE_DESC));
    d3dPipelineStateDesc.pRootSignature = pd3dRootSignature;
    d3dPipelineStateDesc.VS = CreateVertexShader(&pd3dVertexShaderBlob);
    d3dPipelineStateDesc.PS = CreatePixelShader(&pd3dPixelShaderBlob);
    d3dPipelineStateDesc.RasterizerState = CreateRasterizerState();
    d3dPipelineStateDesc.BlendState = CreateBlendState();
    d3dPipelineStateDesc.DepthStencilState = CreateDepthStencilState();
    d3dPipelineStateDesc.InputLayout = CreateInputLayout();
    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;
    pd3dDevice->CreateGraphicsPipelineState(&d3dPipelineStateDesc,
__uuidof(ID3D12PipelineState), (void **)&m_ppd3dPipelineStates[0]);

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

    if (d3dPipelineStateDesc.InputLayout.pInputElementDescs) delete[]
d3dPipelineStateDesc.InputLayout.pInputElementDescs;
}
```

❽ BuildObjects(), ReleaseObjects(), AnimateObjects() 함수를 다음과 같이 정의한다.

```
//셰이더 객체가 포함하는 게임 객체들을 생성한다.
void CShader::BuildObjects(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, void *pContext)
{
    CTriangleMesh *pTriangleMesh = new CTriangleMesh(pd3dDevice, pd3dCommandList);

    m_nObjects = 1;
    m_ppObjects = new CGameObject*[m_nObjects];

    m_ppObjects[0] = new CGameObject();
    m_ppObjects[0]->SetMesh(pTriangleMesh);
}

void CShader::ReleaseObjects()
{
    if (m_ppObjects)
    {
        for (int j = 0; j < m_nObjects; j++) if (m_ppObjects[j]) delete m_ppObjects[j];
        delete[] m_ppObjects;
    }
}
```

```
void CShader::AnimateObjects(float fTimeElapsed)
{
   for (int j = 0; j < m_nObjects; j++)
   {
      m_ppObjects[j]->Animate(fTimeElapsed);
   }
}
```

❾ ReleaseUploadBuffers() 함수를 다음과 같이 정의한다.

```
void CShader::ReleaseUploadBuffers()
{
   if (m_ppObjects)
   {
      for (int j = 0; j < m_nObjects; j++) if (m_ppObjects[j])
m_ppObjects[j]->ReleaseUploadBuffers();
   }
}
```

❿ OnPrepareRender(), Render() 함수를 다음과 같이 정의한다.

```
void CShader::OnPrepareRender(ID3D12GraphicsCommandList *pd3dCommandList)
{
//파이프라인에 그래픽스 상태 객체를 설정한다.
   pd3dCommandList->SetPipelineState(m_ppd3dPipelineStates[0]);
}

void CShader::Render(ID3D12GraphicsCommandList *pd3dCommandList)
{
   OnPrepareRender(pd3dCommandList);

   for (int j = 0; j < m_nObjects; j++)
   {
      if (m_ppObjects[j]) m_ppObjects[j]->Render(pd3dCommandList);
   }
}
```

⓫ "Scene.h" 파일 수정하기

❶ CScene 클래스의 선언을 다음과 같이 수정한다.

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

//씬에서 마우스와 키보드 메시지를 처리한다.
   bool OnProcessingMouseMessage(HWND hWnd, UINT nMessageID, WPARAM wParam, LPARAM
lParam);
   bool OnProcessingKeyboardMessage(HWND hWnd, UINT nMessageID, WPARAM wParam, LPARAM
lParam);
```

```
    void BuildObjects(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList);
    void ReleaseObjects();

    bool ProcessInput(UCHAR *pKeysBuffer);
    void AnimateObjects(float fTimeElapsed);
     void Render(ID3D12GraphicsCommandList *pd3dCommandList);

    void ReleaseUploadBuffers();

//그래픽 루트 시그너쳐를 생성한다.
    ID3D12RootSignature *CreateGraphicsRootSignature(ID3D12Device *pd3dDevice);
    ID3D12RootSignature *GetGraphicsRootSignature();

protected:
//씬은 셰이더들의 집합이다. 셰이더들은 게임 객체들의 집합이다.
    CShader **m_ppShaders = NULL;
    int m_nShaders = 0;

    ID3D12RootSignature *m_pd3dGraphicsRootSignature = NULL;
};
```

⑫ "Scene.cpp" 파일 수정하기

❶ BuildObjects() 멤버 함수를 다음과 수정한다.

```
void CScene::BuildObjects(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList)
{
//그래픽 루트 시그너쳐를 생성한다.
    m_pd3dGraphicsRootSignature = CreateGraphicsRootSignature(pd3dDevice);

//씬을 그리기 위한 셰이더 객체를 생성한다.
    m_nShaders = 1;
    m_ppShaders = new CShader*[m_nShaders];

    CShader *pShader = new CShader();
    pShader->CreateShader(pd3dDevice, m_pd3dGraphicsRootSignature);
    pShader->BuildObjects(pd3dDevice, pd3dCommandList, NULL);

    m_ppShaders[0] = pShader;
}
```

❷ ReleaseObjects() 멤버 함수를 다음과 같이 수정한다.

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

    if (m_ppShaders)
```

```
    {
        for (int i = 0; i < m_nShaders; i++)
        {
            m_ppShaders[i]->ReleaseShaderVariables();
            m_ppShaders[i]->ReleaseObjects();
            m_ppShaders[i]->Release();
        }
        delete[] m_ppShaders;
    }
}
```

❸ ReleaseUploadBuffers() 멤버 함수를 다음과 같이 정의한다.

```
void CScene::ReleaseUploadBuffers()
{
    if (m_ppShaders)
    {
        for (int j = 0; j < m_nShaders; j++) if (m_ppShaders[j])
m_ppShaders[j]->ReleaseUploadBuffers();
    }
}
```

❹ GetGraphicsRootSignature() 멤버 함수를 다음과 같이 정의한다.

```
ID3D12RootSignature *CScene::GetGraphicsRootSignature()
{
    return(m_pd3dGraphicsRootSignature);
}
```

❺ CreateGraphicsRootSignature() 멤버 함수를 다음과 같이 정의한다.

```
ID3D12RootSignature *CScene::CreateGraphicsRootSignature(ID3D12Device *pd3dDevice)
{
    ID3D12RootSignature *pd3dGraphicsRootSignature = NULL;

//매개변수가 없는 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
```

```
**)&pd3dGraphicsRootSignature);
   if (pd3dSignatureBlob) pd3dSignatureBlob->Release();
   if (pd3dErrorBlob) pd3dErrorBlob->Release();

   return(pd3dGraphicsRootSignature);
}
```

❻ AnimateObjects() 멤버 함수를 다음과 같이 수정한다.

```
void CScene::AnimateObjects(float fTimeElapsed)
{
   for (int i = 0; i < m_nShaders; i++)
   {
      m_ppShaders[i]->AnimateObjects(fTimeElapsed);
   }
}
```

❼ Render() 멤버 함수를 다음과 같이 수정한다.

```
void CScene::Render(ID3D12GraphicsCommandList *pd3dCommandList)
{
//그래픽 루트 시그너쳐를 파이프라인에 연결(설정)한다.
   pd3dCommandList->SetGraphicsRootSignature(m_pd3dGraphicsRootSignature);

//씬을 렌더링하는 것은 씬을 구성하는 셰이더(셰이더가 포함하는 객체)들을 렌더링하는 것이다.
   for (int i = 0; i < m_nShaders; i++)
   {
      m_ppShaders[i]->Render(pd3dCommandList);
   }
}
```

⑬ "GameFramework.cpp" 파일 수정하기

❶ BuildObjects() 함수를 다음과 같이 수정한다.

```
void CGameFramework::BuildObjects()
{
   m_pd3dCommandList->Reset(m_pd3dCommandAllocator, NULL);

//씬 객체를 생성하고 씬에 포함될 게임 객체들을 생성한다.
   m_pScene = new CScene();
   m_pScene->BuildObjects(m_pd3dDevice, m_pd3dCommandList);

//씬 객체를 생성하기 위하여 필요한 그래픽 명령 리스트들을 명령 큐에 추가한다.
   m_pd3dCommandList->Close();
   ID3D12CommandList *ppd3dCommandLists[] = { m_pd3dCommandList };
   m_pd3dCommandQueue->ExecuteCommandLists(1, ppd3dCommandLists);

//그래픽 명령 리스트들이 모두 실행될 때까지 기다린다.
   WaitForGpuComplete();
```

```
//그래픽 리소스들을 생성하는 과정에 생성된 업로드 버퍼들을 소멸시킨다.
   if (m_pScene) m_pScene->ReleaseUploadBuffers();

   m_GameTimer.Reset();
}
```

⑭ "Shaders.hlsl" 파일 수정하기

❶ "Shaders.hlsl" 파일의 내용을 다음과 같이 수정한다.

```
//정점 셰이더의 입력을 위한 구조체를 선언한다.
struct VS_INPUT
{
   float3 position : POSITION;
   float4 color : COLOR;
};

//정점 셰이더의 출력(픽셀 셰이더의 입력)을 위한 구조체를 선언한다.
struct VS_OUTPUT
{
   float4 position : SV_POSITION;
   float4 color : COLOR;
};

//정점 셰이더를 정의한다.
VS_OUTPUT VSMain(VS_INPUT input)
{
   VS_OUTPUT output;

   output.position = float4(input.position, 1.0f);
   output.color = input.color;

   return(output);
}

//픽셀 셰이더를 정의한다.
float4 PSMain(VS_OUTPUT input) : SV_TARGET
{
   return(input.color);
}
```