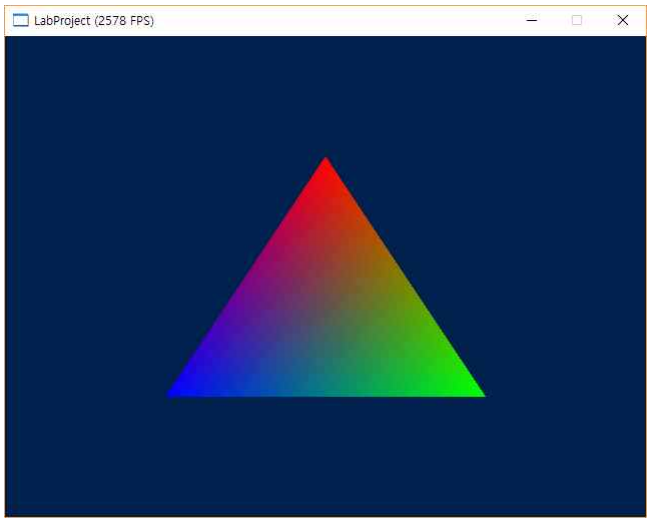
□ 예제 프로그램 8: LabProject07(회전하는 삼각형 그리기)

LabProject06 프로젝트를 기반으로 다음과 같이 회전하는 삼각형을 그려본다.



① 새로운 프로젝트의 생성

먼저 새로운 프로젝트 LabProject07를 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 “LabProject07”를 입력하고 『확인』을 선택한다.

❶ 파일 탐색기에서 프로젝트 “LabProject06” 폴더의 다음 파일을 선택하여 프로젝트 “LabProject07” 폴더에 복사한다.

- GameFramework.h
- GameFramework.cpp
- Mesh.h
- Mesh.cpp
- Object.h
- Object.cpp
- Scene.h
- Scene.cpp
- Shader.h
- Shader.cpp
- stdafx.h
- Timer.h

- Timer.cpp
- Shaders.hlsl

❷ 위에서 복사한 파일을 Visual Studio 솔루션 탐색기에서 프로젝트 “LabProject07”에 추가한다. 오른쪽 마우스 버튼으로 『LabProject07』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 “LabProject07”에 추가된다.

② LabProject07.cpp 파일 수정하기

이제 “LabProject07.cpp" 파일의 내용을 “LabProject06.cpp" 파일의 내용으로 바꾸도록 하자. “LabProject06.cpp" 파일의 내용 전체를 “LabProject07.cpp" 파일로 복사한다. 이제 “LabProject07.cpp" 파일에서 “LabProject06”을 “LabProject07”로 모두 바꾼다.
그리고 “LABPROJECT06”을 “LABPROJECT07”로 모두 바꾼다.

③ “stdafx.h” 파일 수정하기

“stdafx.h” 파일에 벡터와 행렬 연산을 위한 함수들을 다음과 같이 추가한다.

```
//3차원 벡터의 연산
namespace Vector3
{
   inline XMFLOAT3 XMVectorToFloat3(XMVECTOR& xmvVector)
   {
      XMFLOAT3 xmf3Result;
      XMStoreFloat3(&xmf3Result, xmvVector);
      return(xmf3Result);
   }

   inline XMFLOAT3 ScalarProduct(XMFLOAT3& xmf3Vector, float fScalar, bool bNormalize =
true)
   {
      XMFLOAT3 xmf3Result;
      if (bNormalize)
         XMStoreFloat3(&xmf3Result, XMVector3Normalize(XMLoadFloat3(&xmf3Vector)) *
fScalar);
      else
         XMStoreFloat3(&xmf3Result, XMLoadFloat3(&xmf3Vector) * fScalar);
      return(xmf3Result);
   }

   inline XMFLOAT3 Add(const XMFLOAT3& xmf3Vector1, const XMFLOAT3& xmf3Vector2)
   {
      XMFLOAT3 xmf3Result;
      XMStoreFloat3(&xmf3Result, XMLoadFloat3(&xmf3Vector1) +
XMLoadFloat3(&xmf3Vector2));
```

```
      return(xmf3Result);
   }

   inline XMFLOAT3 Add(XMFLOAT3& xmf3Vector1, XMFLOAT3& xmf3Vector2, float fScalar)
   {
      XMFLOAT3 xmf3Result;
      XMStoreFloat3(&xmf3Result, XMLoadFloat3(&xmf3Vector1) + (XMLoadFloat3(&xmf3Vector2)
* fScalar));
      return(xmf3Result);
   }

   inline XMFLOAT3 Subtract(XMFLOAT3& xmf3Vector1, XMFLOAT3& xmf3Vector2)
   {
      XMFLOAT3 xmf3Result;
      XMStoreFloat3(&xmf3Result, XMLoadFloat3(&xmf3Vector1) -
XMLoadFloat3(&xmf3Vector2));
      return(xmf3Result);
   }

   inline float DotProduct(XMFLOAT3& xmf3Vector1, XMFLOAT3& xmf3Vector2)
   {
      XMFLOAT3 xmf3Result;
      XMStoreFloat3(&xmf3Result, XMVector3Dot(XMLoadFloat3(&xmf3Vector1),
XMLoadFloat3(&xmf3Vector2)));
      return(xmf3Result.x);
   }

   inline XMFLOAT3 CrossProduct(XMFLOAT3& xmf3Vector1, XMFLOAT3& xmf3Vector2, bool
bNormalize = true)
   {
      XMFLOAT3 xmf3Result;
      if (bNormalize)
         XMStoreFloat3(&xmf3Result,
XMVector3Normalize(XMVector3Cross(XMLoadFloat3(&xmf3Vector1),
XMLoadFloat3(&xmf3Vector2))));
      else
         XMStoreFloat3(&xmf3Result, XMVector3Cross(XMLoadFloat3(&xmf3Vector1),
XMLoadFloat3(&xmf3Vector2)));
      return(xmf3Result);
   }

   inline XMFLOAT3 Normalize(XMFLOAT3& xmf3Vector)
   {
      XMFLOAT3 m_xmf3Normal;
      XMStoreFloat3(&m_xmf3Normal, XMVector3Normalize(XMLoadFloat3(&xmf3Vector)));
      return(m_xmf3Normal);
   }

   inline float Length(XMFLOAT3& xmf3Vector)
   {
      XMFLOAT3 xmf3Result;
      XMStoreFloat3(&xmf3Result, XMVector3Length(XMLoadFloat3(&xmf3Vector)));
      return(xmf3Result.x);
   }

   inline float Angle(XMVECTOR& xmvVector1, XMVECTOR& xmvVector2)
```

```
    {
        XMVECTOR xmvAngle = XMVector3AngleBetweenNormals(xmvVector1, xmvVector2);
        return(XMConvertToDegrees(acosf(XMVectorGetX(xmvAngle))));
    }

    inline float Angle(XMFLOAT3& xmf3Vector1, XMFLOAT3& xmf3Vector2)
    {
        return(Angle(XMLoadFloat3(&xmf3Vector1), XMLoadFloat3(&xmf3Vector2)));
    }

    inline XMFLOAT3 TransformNormal(XMFLOAT3& xmf3Vector, XMMATRIX& xmmtxTransform)
    {
        XMFLOAT3 xmf3Result;
        XMStoreFloat3(&xmf3Result, XMVector3TransformNormal(XMLoadFloat3(&xmf3Vector),
xmmtxTransform));
        return(xmf3Result);
    }

    inline XMFLOAT3 TransformCoord(XMFLOAT3& xmf3Vector, XMMATRIX& xmmtxTransform)
    {
        XMFLOAT3 xmf3Result;
        XMStoreFloat3(&xmf3Result, XMVector3TransformCoord(XMLoadFloat3(&xmf3Vector),
xmmtxTransform));
        return(xmf3Result);
    }

    inline XMFLOAT3 TransformCoord(XMFLOAT3& xmf3Vector, XMFLOAT4X4& xmmtx4x4Matrix)
    {
        return(TransformCoord(xmf3Vector, XMLoadFloat4x4(&xmmtx4x4Matrix)));
    }
}

//4차원 벡터의 연산
namespace Vector4
{
    inline XMFLOAT4 Add(XMFLOAT4& xmf4Vector1, XMFLOAT4& xmf4Vector2)
    {
        XMFLOAT4 xmf4Result;
        XMStoreFloat4(&xmf4Result, XMLoadFloat4(&xmf4Vector1) +
XMLoadFloat4(&xmf4Vector2));
        return(xmf4Result);
    }

    inline XMFLOAT4 Multiply(XMFLOAT4& xmf4Vector1, XMFLOAT4& xmf4Vector2)
    {
        XMFLOAT4 xmf4Result;
        XMStoreFloat4(&xmf4Result, XMLoadFloat4(&xmf4Vector1) *
XMLoadFloat4(&xmf4Vector2));
        return(xmf4Result);
    }

    inline XMFLOAT4 Multiply(float fScalar, XMFLOAT4& xmf4Vector)
    {
        XMFLOAT4 xmf4Result;
        XMStoreFloat4(&xmf4Result, fScalar * XMLoadFloat4(&xmf4Vector));
        return(xmf4Result);
```

```
    }
}

//행렬의 연산
namespace Matrix4x4
{
    inline XMFLOAT4X4 Identity()
    {
        XMFLOAT4X4 xmmtx4x4Result;
        XMStoreFloat4x4(&xmmtx4x4Result, XMMatrixIdentity());
        return(xmmtx4x4Result);
    }

    inline XMFLOAT4X4 Multiply(XMFLOAT4X4& xmmtx4x4Matrix1, XMFLOAT4X4& xmmtx4x4Matrix2)
    {
        XMFLOAT4X4 xmmtx4x4Result;
        XMStoreFloat4x4(&xmmtx4x4Result, XMLoadFloat4x4(&xmmtx4x4Matrix1) *
XMLoadFloat4x4(&xmmtx4x4Matrix2));
        return(xmmtx4x4Result);
    }

    inline XMFLOAT4X4 Multiply(XMFLOAT4X4& xmmtx4x4Matrix1, XMMATRIX& xmmtxMatrix2)
    {
        XMFLOAT4X4 xmmtx4x4Result;
        XMStoreFloat4x4(&xmmtx4x4Result, XMLoadFloat4x4(&xmmtx4x4Matrix1) * xmmtxMatrix2);
        return(xmmtx4x4Result);
    }

    inline XMFLOAT4X4 Multiply(XMMATRIX& xmmtxMatrix1, XMFLOAT4X4& xmmtx4x4Matrix2)
    {
        XMFLOAT4X4 xmmtx4x4Result;
        XMStoreFloat4x4(&xmmtx4x4Result, xmmtxMatrix1 * XMLoadFloat4x4(&xmmtx4x4Matrix2));
        return(xmmtx4x4Result);
    }

    inline XMFLOAT4X4 Inverse(XMFLOAT4X4& xmmtx4x4Matrix)
    {
        XMFLOAT4X4 xmmtx4x4Result;
        XMStoreFloat4x4(&xmmtx4x4Result, XMMatrixInverse(NULL,
XMLoadFloat4x4(&xmmtx4x4Matrix)));
        return(xmmtx4x4Result);
    }

    inline XMFLOAT4X4 Transpose(XMFLOAT4X4& xmmtx4x4Matrix)
    {
        XMFLOAT4X4 xmmtx4x4Result;
        XMStoreFloat4x4(&xmmtx4x4Result,
XMMatrixTranspose(XMLoadFloat4x4(&xmmtx4x4Matrix)));
        return(xmmtx4x4Result);
    }

    inline XMFLOAT4X4 PerspectiveFovLH(float FovAngleY, float AspectRatio, float NearZ,
float FarZ)
    {
        XMFLOAT4X4 xmmtx4x4Result;
        XMStoreFloat4x4(&xmmtx4x4Result, XMMatrixPerspectiveFovLH(FovAngleY, AspectRatio,
```

```
NearZ, FarZ));
      return(xmmtx4x4Result);
   }

   inline XMFLOAT4X4 LookAtLH(XMFLOAT3& xmf3EyePosition, XMFLOAT3& xmf3LookAtPosition,
XMFLOAT3& xmf3UpDirection)
   {
      XMFLOAT4X4 xmmtx4x4Result;
      XMStoreFloat4x4(&xmmtx4x4Result, XMMatrixLookAtLH(XMLoadFloat3(&xmf3EyePosition),
XMLoadFloat3(&xmf3LookAtPosition), XMLoadFloat3(&xmf3UpDirection)));
      return(xmmtx4x4Result);
   }
}
```

④ "CCamera" 클래스 생성하기

게임 프로그램의 기본적 요소를 추가하기 위해 하나의 클래스(Class)를 만들도록 하자. 클래스의 이름은 "CCamera"이다. 솔루션 탐색기에서 오른쪽 마우스 버튼으로 "LabProject07"을 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 "CCamera"를 입력한다. 그러면 ".h 파일"의 이름이 "Camera.h" 그리고 ".cpp 파일"의 이름이 "Camera.cpp"가 된다. 『마침』을 선택하면 클래스 마법사가 두 개의 파일을 프로젝트에 추가하여 준다.

⑤ "CCamera" 클래스 선언하기

"Camera.h" 파일을 다음과 같이 수정한다.

❶ "Camera.h" 파일의 앞 부분에 다음을 추가한다.

```
#pragma once

#define ASPECT_RATIO  (float(FRAME_BUFFER_WIDTH) / float(FRAME_BUFFER_HEIGHT))
```

❷ 카메라 상수 버퍼를 위한 구조체를 다음과 같이 추가한다.

```
struct VS_CB_CAMERA_INFO
{
   XMFLOAT4X4 m_xmf4x4View;
   XMFLOAT4X4 m_xmf4x4Projection;
};
```

❸ "CCamera" 클래스를 다음과 같이 선언한다.

```
class CCamera
{
protected:
```

```
//카메라 변환 행렬
   XMFLOAT4X4 m_xmf4x4View;
//투영 변환 행렬
   XMFLOAT4X4 m_xmf4x4Projection;

//뷰포트와 씨저 사각형
   D3D12_VIEWPORT m_d3dViewport;
   D3D12_RECT m_d3dScissorRect;

public:
   CCamera();
   virtual ~CCamera();

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

   void GenerateViewMatrix(XMFLOAT3 xmf3Position, XMFLOAT3 xmf3LookAt, XMFLOAT3 xmf3Up);
   void GenerateProjectionMatrix(float fNearPlaneDistance, float fFarPlaneDistance, float
fAspectRatio, float fFOVAngle);

   void SetViewport(int xTopLeft, int yTopLeft, int nWidth, int nHeight, float fMinZ =
0.0f, float fMaxZ = 1.0f);
   void SetScissorRect(LONG xLeft, LONG yTop, LONG xRight, LONG yBottom);

   virtual void SetViewportsAndScissorRects(ID3D12GraphicsCommandList *pd3dCommandList);
};
```

⑥ "CCamera" 클래스 구현하기

"Camera.cpp" 파일을 다음과 같이 수정한다.

❶ "CCamera" 클래스의 생성자를 다음과 같이 수정한다.

```
CCamera::CCamera()
{
   m_xmf4x4View = Matrix4x4::Identity();
   m_xmf4x4Projection = Matrix4x4::Identity();
   m_d3dViewport = { 0, 0, FRAME_BUFFER_WIDTH , FRAME_BUFFER_HEIGHT, 0.0f, 1.0f };
   m_d3dScissorRect = { 0, 0, FRAME_BUFFER_WIDTH , FRAME_BUFFER_HEIGHT };
}
```

❷ SetViewport() 함수와 SetScissorRect() 함수를 다음과 같이 정의한다.

```
void CCamera::SetViewport(int xTopLeft, int yTopLeft, int nWidth, int nHeight, float
fMinZ, float fMaxZ)
{
   m_d3dViewport.TopLeftX = float(xTopLeft);
   m_d3dViewport.TopLeftY = float(yTopLeft);
   m_d3dViewport.Width = float(nWidth);
```

```
   m_d3dViewport.Height = float(nHeight);
   m_d3dViewport.MinDepth = fMinZ;
   m_d3dViewport.MaxDepth = fMaxZ;
}

void CCamera::SetScissorRect(LONG xLeft, LONG yTop, LONG xRight, LONG yBottom)
{
   m_d3dScissorRect.left = xLeft;
   m_d3dScissorRect.top = yTop;
   m_d3dScissorRect.right = xRight;
   m_d3dScissorRect.bottom = yBottom;
}
```

❸ GenerateProjectionMatrix() 함수와 GenerateViewMatrix() 함수를 다음과 같이 정의한다.

```
void CCamera::GenerateProjectionMatrix(float fNearPlaneDistance, float fFarPlaneDistance,
float fAspectRatio, float fFOVAngle)
{
   m_xmf4x4Projection = Matrix4x4::PerspectiveFovLH(XMConvertToRadians(fFOVAngle),
fAspectRatio, fNearPlaneDistance, fFarPlaneDistance);
}

void CCamera::GenerateViewMatrix(XMFLOAT3 xmf3Position, XMFLOAT3 xmf3LookAt, XMFLOAT3
xmf3Up)
{
   m_xmf4x4View = Matrix4x4::LookAtLH(xmf3Position, xmf3LookAt, xmf3Up);
}
```

❹ CreateShaderVariables(), UpdateShaderVariables(), ReleaseShaderVariables() 함수를 다음과 같이 정의한다.

```
void CCamera::CreateShaderVariables(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList)
{
}

void CCamera::UpdateShaderVariables(ID3D12GraphicsCommandList *pd3dCommandList)
{
   XMFLOAT4X4 xmf4x4View;
   XMStoreFloat4x4(&xmf4x4View, XMMatrixTranspose(XMLoadFloat4x4(&m_xmf4x4View)));
//루트 파라메터 인덱스 1의
   pd3dCommandList->SetGraphicsRoot32BitConstants(1, 16, &xmf4x4View, 0);

   XMFLOAT4X4 xmf4x4Projection;
   XMStoreFloat4x4(&xmf4x4Projection,
XMMatrixTranspose(XMLoadFloat4x4(&m_xmf4x4Projection)));
   pd3dCommandList->SetGraphicsRoot32BitConstants(1, 16, &xmf4x4Projection, 16);
}

void CCamera::ReleaseShaderVariables()
{
}
```

❺ SetViewportsAndScissorRects() 함수를 다음과 같이 정의한다.

```
void CCamera::SetViewportsAndScissorRects(ID3D12GraphicsCommandList *pd3dCommandList)
{
   pd3dCommandList->RSSetViewports(1, &m_d3dViewport);
   pd3dCommandList->RSSetScissorRects(1, &m_d3dScissorRect);
}
```

⑦ "GameFramework.h" 파일 수정하기

❶ CGameFramework 클래스에서 다음 멤버 변수를 삭제한다.

```
   D3D12_VIEWPORT m_d3dViewport;
   D3D12_RECT m_d3dScissorRect;
```

❷ CGameFramework 클래스에 다음을 추가한다.

```
public:
   CCamera *m_pCamera = NULL;
```

⑧ "GameFramework.cpp" 파일 수정하기

❶ BuildObjects() 함수를 다음과 같이 수정한다.

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

//카메라 객체를 생성하여 뷰포트, 씨저 사각형, 투영 변환 행렬, 카메라 변환 행렬을 생성하고 설정한다.
   m_pCamera = new CCamera();
   m_pCamera->SetViewport(0, 0, m_nWndClientWidth, m_nWndClientHeight, 0.0f, 1.0f);
   m_pCamera->SetScissorRect(0, 0, m_nWndClientWidth, m_nWndClientHeight);
   m_pCamera->GenerateProjectionMatrix(1.0f, 500.0f, float(m_nWndClientWidth) /
float(m_nWndClientHeight), 90.0f);
   m_pCamera->GenerateViewMatrix(XMFLOAT3(0.0f, 0.0f, -2.0f), XMFLOAT3(0.0f, 0.0f, 0.0f),
XMFLOAT3(0.0f, 1.0f, 0.0f));

//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();
}
```

❷ OnResizeBackBuffers() 함수에서 다음을 삭제한다.

```
   m_d3dViewport.TopLeftX = 0;
   m_d3dViewport.TopLeftY = 0;
   m_d3dViewport.Width = static_cast<float>(m_nWndClientWidth);
   m_d3dViewport.Height = static_cast<float>(m_nWndClientHeight);
   m_d3dViewport.MinDepth = 0.0f;
   m_d3dViewport.MaxDepth = 1.0f;

   m_d3dScissorRect = { 0, 0, m_nWndClientWidth, m_nWndClientHeight };
```

❷ FrameAdvance() 함수에서 다음을 삭제한다.

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

⑨ "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 게임 객체들의 집합이다. 게임 객체는 셰이더를 포함한다.
    CGameObject **m_ppObjects = NULL;
    int m_nObjects = 0;

    ID3D12RootSignature *m_pd3dGraphicsRootSignature = NULL;
};
```

⑩ "Scene.cpp" 파일 수정하기

❶ BuildObjects() 멤버 함수를 다음과 수정한다.

```
void CScene::BuildObjects(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList)
{
    m_pd3dGraphicsRootSignature = CreateGraphicsRootSignature(pd3dDevice);

    CTriangleMesh *pMesh = new CTriangleMesh(pd3dDevice, pd3dCommandList);

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

    CRotatingObject *pRotatingObject = new CRotatingObject();
    pRotatingObject->SetMesh(pMesh);

    CDiffusedShader *pShader = new CDiffusedShader();
    pShader->CreateShader(pd3dDevice, m_pd3dGraphicsRootSignature);
    pShader->CreateShaderVariables(pd3dDevice, pd3dCommandList);

    pRotatingObject->SetShader(pShader);

    m_ppObjects[0] = pRotatingObject;
}
```

❷ ReleaseObjects() 멤버 함수를 다음과 같이 수정한다.

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

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

❸ ReleaseUploadBuffers() 멤버 함수를 다음과 같이 수정한다.

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

❹ CreateGraphicsRootSignature() 멤버 함수를 다음과 같이 정의한다.

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

   D3D12_ROOT_PARAMETER pd3dRootParameters[2];
   pd3dRootParameters[0].ParameterType = D3D12_ROOT_PARAMETER_TYPE_32BIT_CONSTANTS;
   pd3dRootParameters[0].Constants.Num32BitValues = 16;
   pd3dRootParameters[0].Constants.ShaderRegister = 0;
   pd3dRootParameters[0].Constants.RegisterSpace = 0;
   pd3dRootParameters[0].ShaderVisibility = D3D12_SHADER_VISIBILITY_VERTEX;
   pd3dRootParameters[1].ParameterType = D3D12_ROOT_PARAMETER_TYPE_32BIT_CONSTANTS;
   pd3dRootParameters[1].Constants.Num32BitValues = 32;
   pd3dRootParameters[1].Constants.ShaderRegister = 1;
   pd3dRootParameters[1].Constants.RegisterSpace = 0;
   pd3dRootParameters[1].ShaderVisibility = D3D12_SHADER_VISIBILITY_VERTEX;

   D3D12_ROOT_SIGNATURE_FLAGS d3dRootSignatureFlags =
D3D12_ROOT_SIGNATURE_FLAG_ALLOW_INPUT_ASSEMBLER_INPUT_LAYOUT |
D3D12_ROOT_SIGNATURE_FLAG_DENY_HULL_SHADER_ROOT_ACCESS |
D3D12_ROOT_SIGNATURE_FLAG_DENY_DOMAIN_SHADER_ROOT_ACCESS |
D3D12_ROOT_SIGNATURE_FLAG_DENY_GEOMETRY_SHADER_ROOT_ACCESS |
D3D12_ROOT_SIGNATURE_FLAG_DENY_PIXEL_SHADER_ROOT_ACCESS;
   D3D12_ROOT_SIGNATURE_DESC d3dRootSignatureDesc;
   ::ZeroMemory(&d3dRootSignatureDesc, sizeof(D3D12_ROOT_SIGNATURE_DESC));
   d3dRootSignatureDesc.NumParameters = _countof(pd3dRootParameters);
   d3dRootSignatureDesc.pParameters = pd3dRootParameters;
   d3dRootSignatureDesc.NumStaticSamplers = 0;
   d3dRootSignatureDesc.pStaticSamplers = NULL;
   d3dRootSignatureDesc.Flags = d3dRootSignatureFlags;

   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);
}
```

❺ AnimateObjects() 멤버 함수를 다음과 같이 수정한다.

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

❻ Render() 멤버 함수를 다음과 같이 수정한다.

```
void CScene::Render(ID3D12GraphicsCommandList *pd3dCommandList, CCamera *pCamera)
{
   pCamera->SetViewportsAndScissorRects(pd3dCommandList);
   pd3dCommandList->SetGraphicsRootSignature(m_pd3dGraphicsRootSignature);

   if (pCamera) pCamera->UpdateShaderVariables(pd3dCommandList);

//씬을 렌더링하는 것은 씬을 구성하는 게임 객체(셰이더를 포함하는 객체)들을 렌더링하는 것이다.
   for (int j = 0; j < m_nObjects; j++)
   {
      if (m_ppObjects[j]) m_ppObjects[j]->Render(pd3dCommandList, pCamera);
   }
}
```

⑪ ”GameObject.h” 파일 수정하기

❶ “CGameObject” 클래스에 다음을 추가한다.

```
public:
   void Rotate(XMFLOAT3 *pxmf3Axis, float fAngle);
```

❷ ”CRotatingObject” 클래스를 다음과 같이 선언한다.

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

private:
   XMFLOAT3 m_xmf3RotationAxis;
   float m_fRotationSpeed;

public:
   void SetRotationSpeed(float fRotationSpeed) { m_fRotationSpeed = fRotationSpeed; }
   void SetRotationAxis(XMFLOAT3 xmf3RotationAxis) { m_xmf3RotationAxis =
xmf3RotationAxis; }
```

```
    virtual void Animate(float fTimeElapsed);
};
```

⑫ "GameObject.cpp" 파일 수정하기

❶ "CGameObject" 클래스의 Rotate() 함수를 다음과 같이 정의한다.

```
void CGameObject::Rotate(XMFLOAT3 *pxmf3Axis, float fAngle)
{
    XMMATRIX mtxRotate = XMMatrixRotationAxis(XMLoadFloat3(pxmf3Axis),
XMConvertToRadians(fAngle));
    m_xmf4x4World = Matrix4x4::Multiply(mtxRotate, m_xmf4x4World);
}
```

❷ "CGameObject" 클래스의 Render() 함수를 다음과 같이 수정한다.

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

    if (m_pShader)
    {
//게임 객체의 월드 변환 행렬을 셰이더의 상수 버퍼로 전달(복사)한다.
        m_pShader->UpdateShaderVariable(pd3dCommandList, &m_xmf4x4World);
        m_pShader->Render(pd3dCommandList, pCamera);
    }

    if (m_pMesh) m_pMesh->Render(pd3dCommandList);
}
```

❸ "CRotatingObject" 클래스의 생성자와 소멸자를 다음과 같이 정의한다.

```
CRotatingObject::CRotatingObject()
{
    m_xmf3RotationAxis = XMFLOAT3(0.0f, 1.0f, 0.0f);
    m_fRotationSpeed = 90.0f;
}

CRotatingObject::~CRotatingObject()
{
}
```

❹ "CRotatingObject" 클래스의 Animate() 함수를 다음과 같이 정의한다.

```
void CRotatingObject::Animate(float fTimeElapsed)
{
    CGameObject::Rotate(&m_xmf3RotationAxis, m_fRotationSpeed * fTimeElapsed);
}
```

⑬ "Shader.h" 파일 수정하기

❶ "Shader.h" 파일에 다음을 추가한다.

```
#include "Camera.h"

//게임 객체의 정보를 셰이더에게 넘겨주기 위한 구조체(상수 버퍼)이다.
struct CB_GAMEOBJECT_INFO
{
   XMFLOAT4X4 m_xmf4x4World;
};
```

❷ "CShader" 클래스를 다음과 같이 수정한다.

```
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
*pd3dGraphicsRootSignature);

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

   virtual void UpdateShaderVariable(ID3D12GraphicsCommandList *pd3dCommandList,
XMFLOAT4X4 *pxmf4x4World);

   virtual void OnPrepareRender(ID3D12GraphicsCommandList *pd3dCommandList);
   virtual void Render(ID3D12GraphicsCommandList *pd3dCommandList, CCamera *pCamera);
```

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

❸ "CDiffusedShader" 클래스를 다음과 같이 선언한다.

```
class CDiffusedShader : public CShader
{
public:
   CDiffusedShader();
   virtual ~CDiffusedShader();

   virtual D3D12_INPUT_LAYOUT_DESC CreateInputLayout();

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

   virtual void CreateShader(ID3D12Device *pd3dDevice, ID3D12RootSignature
*pd3dGraphicsRootSignature);
};
```

⑭ "Shader.cpp" 파일 수정하기

❶ CShader 클래스의 CreateVertexShader() 함수와 CreatePixelShader() 함수를 다음과 같이 수정한다.

```
D3D12_SHADER_BYTECODE CShader::CreateVertexShader(ID3DBlob **ppd3dShaderBlob)
{
   D3D12_SHADER_BYTECODE d3dShaderByteCode;
   d3dShaderByteCode.BytecodeLength = 0;
   d3dShaderByteCode.pShaderBytecode = NULL;

   return(d3dShaderByteCode);
}

D3D12_SHADER_BYTECODE CShader::CreatePixelShader(ID3DBlob **ppd3dShaderBlob)
{
   D3D12_SHADER_BYTECODE d3dShaderByteCode;
   d3dShaderByteCode.BytecodeLength = 0;
   d3dShaderByteCode.pShaderBytecode = NULL;

   return(d3dShaderByteCode);
}
```

❷ CShader 클래스의 CreateInputLayout() 함수를 다음과 같이 수정한다.

```
D3D12_INPUT_LAYOUT_DESC CShader::CreateInputLayout()
{
   D3D12_INPUT_LAYOUT_DESC d3dInputLayoutDesc;
   d3dInputLayoutDesc.pInputElementDescs = NULL;
```

```
   d3dInputLayoutDesc.NumElements = 0;

   return(d3dInputLayoutDesc);
}
```

❸ CShader 클래스의 CreateShader() 함수를 다음과 같이 수정한다.

```
//그래픽스 파이프라인 상태 객체를 생성한다.
void CShader::CreateShader(ID3D12Device *pd3dDevice, ID3D12RootSignature
*pd3dGraphicsRootSignature)
{
   ID3DBlob *pd3dVertexShaderBlob = NULL, *pd3dPixelShaderBlob = NULL;

   D3D12_GRAPHICS_PIPELINE_STATE_DESC d3dPipelineStateDesc;
   ::ZeroMemory(&d3dPipelineStateDesc, sizeof(D3D12_GRAPHICS_PIPELINE_STATE_DESC));
   d3dPipelineStateDesc.pRootSignature = pd3dGraphicsRootSignature;
   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;
   d3dPipelineStateDesc.Flags = D3D12_PIPELINE_STATE_FLAG_NONE;
   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;
}
```

❹ CShader 클래스의 CreateShaderVariables(), UpdateShaderVariables(), UpdateShaderVariable() 함수를 다음과 같이 정의한다.

```
void CShader::CreateShaderVariables(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList)
{
}

void CShader::UpdateShaderVariables(ID3D12GraphicsCommandList *pd3dCommandList)
{
}

void CShader::UpdateShaderVariable(ID3D12GraphicsCommandList *pd3dCommandList, XMFLOAT4X4
```

```
*pxmf4x4World)
{
   XMFLOAT4X4 xmf4x4World;
   XMStoreFloat4x4(&xmf4x4World, XMMatrixTranspose(XMLoadFloat4x4(pxmf4x4World)));
   pd3dCommandList->SetGraphicsRoot32BitConstants(0, 16, &xmf4x4World, 0);
}

void CShader::ReleaseShaderVariables()
{
}
```

❺ CShader 클래스의 BuildObjects(), ReleaseObjects(), AnimateObjects(), ReleaseUploadBuffers() 함수를 삭제한다.

❻ CShader 클래스의 Render() 함수를 다음과 같이 수정한다.

```
void CShader::Render(ID3D12GraphicsCommandList *pd3dCommandList, CCamera *pCamera)
{
   OnPrepareRender(pd3dCommandList);
}
```

❼ CDiffusedShader 클래스의 생성자와 소멸자를 다음과 같이 정의한다.

```
CDiffusedShader::CDiffusedShader()
{
}

CDiffusedShader::~CDiffusedShader()
{
}
```

❽ CDiffusedShader 클래스의 CreateInputLayout() 함수를 다음과 같이 정의한다.

```
D3D12_INPUT_LAYOUT_DESC CDiffusedShader::CreateInputLayout()
{
   UINT nInputElementDescs = 2;
   D3D12_INPUT_ELEMENT_DESC *pd3dInputElementDescs = new
D3D12_INPUT_ELEMENT_DESC[nInputElementDescs];

   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);
}
```

❾ CDiffusedShader 클래스의 CreateVertexShader() 함수와 CreatePixelShader() 함수를 다음과 같이 정의한다.

```
D3D12_SHADER_BYTECODE CDiffusedShader::CreateVertexShader(ID3DBlob **ppd3dShaderBlob)
{
   return(CShader::CompileShaderFromFile(L"Shaders.hlsl", "VSDiffused", "vs_5_1",
ppd3dShaderBlob));
}

D3D12_SHADER_BYTECODE CDiffusedShader::CreatePixelShader(ID3DBlob **ppd3dShaderBlob)
{
   return(CShader::CompileShaderFromFile(L"Shaders.hlsl", "PSDiffused", "ps_5_1",
ppd3dShaderBlob));
}
```

❿ CDiffusedShader 클래스의 CreateShader() 함수를 다음과 같이 정의한다.

```
void CDiffusedShader::CreateShader(ID3D12Device *pd3dDevice, ID3D12RootSignature
*pd3dGraphicsRootSignature)
{
   m_nPipelineStates = 1;
   m_ppd3dPipelineStates = new ID3D12PipelineState*[m_nPipelineStates];

   CShader::CreateShader(pd3dDevice, pd3dGraphicsRootSignature);
}
```

⑮ "Shaders.hlsl" 파일 수정하기

"Shaders.hlsl" 파일의 내용을 다음과 같이 수정한다.

```
//게임 객체의 정보를 위한 상수 버퍼를 선언한다.
cbuffer cbGameObjectInfo : register(b0)
{
   matrix gmtxWorld : packoffset(c0);
};

//카메라의 정보를 위한 상수 버퍼를 선언한다.
cbuffer cbCameraInfo : register(b1)
{
   matrix gmtxView : packoffset(c0);
   matrix gmtxProjection : packoffset(c4);
};

//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 VSDiffused(VS_INPUT input)
{
   VS_OUTPUT output;
//정점을 변환(월드 변환, 카메라 변환, 투영 변환)한다.
   output.position = mul(mul(mul(float4(input.position, 1.0f), gmtxWorld), gmtxView),
gmtxProjection);
   output.color = input.color;

   return(output);
}

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