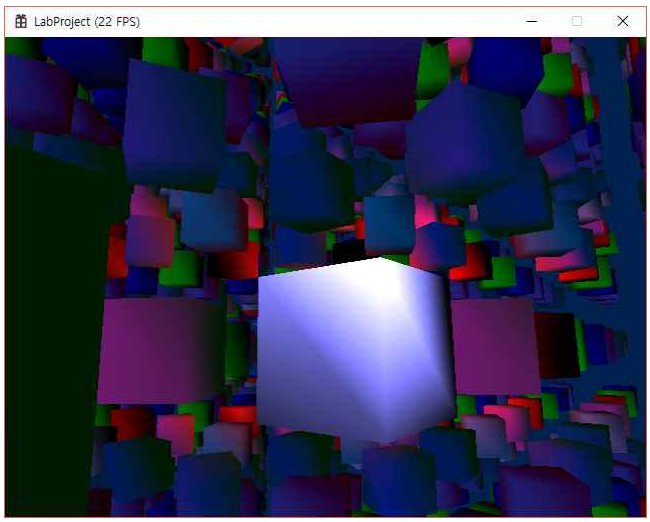
□ 예제 프로그램 18: LabProject17(조명 처리)

예제 프로그램 LabProject11를 기반으로 여러 개의 직육면체들을 조명 처리를 하여 렌더링할 수 있도록 구현한다.



① 새로운 프로젝트의 생성

먼저 새로운 프로젝트 LabProject17을 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 “LabProject17”을 입력하고 『확인』을 선택한다.

❶ 파일 탐색기에서 프로젝트 “LabProject11” 폴더의 다음 파일을 선택하여 프로젝트 “LabProject17” 폴더에 복사한다.

- GameFramework.h
- GameFramework.cpp
- Mesh.h
- Mesh.cpp
- Camera.h
- Camera.cpp
- Player.h
- Player.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 “LabProject17”에 추가한다. 오른쪽 마우스 버튼으로 『LabProject17』를 선택하고 『추가』, 『기존 항목』를 차례로 선택한다. 그러면 “기존 항목 추가” 대화 상자가 나타난다. 위의 파일들을 마우스로 선택(Ctrl+선택)하여 『추가』를 누르면 선택된 파일들이 프로젝트 “LabProject17”에 추가된다.

② LabProject17.cpp 파일 수정하기

이제 “LabProject17.cpp" 파일의 내용을 “LabProject11.cpp" 파일의 내용으로 바꾸도록 하자. “LabProject11.cpp" 파일의 내용 전체를 “LabProject17.cpp" 파일로 복사한다. 이제 “LabProject11.cpp" 파일에서 “LabProject11”를 “LabProject17”으로 모두 바꾼다.
그리고 “LABPROJECT11”를 “LABPROJECT17”으로 모두 바꾼다.

③ “stdafx.h” 파일 수정하기

❶ “stdafx.h” 파일에 다음을 추가한다.

```
#define MAX_LIGHTS                  8
#define MAX_MATERIALS               8

#define POINT_LIGHT                 1
#define SPOT_LIGHT                  2
#define DIRECTIONAL_LIGHT           3
```

④ “Mesh.h” 파일 수정하기

❶ “CIlluminatedVertex” 클래스를 다음과 같이 선언한다.

```
class CIlluminatedVertex : public CVertex
{
protected:
   XMFLOAT3           m_xmf3Normal;

public:
   CIlluminatedVertex() { m_xmf3Position = XMFLOAT3(0.0f, 0.0f, 0.0f); m_xmf3Normal =
XMFLOAT3(0.0f, 0.0f, 0.0f); }
```

```
   CIlluminatedVertex(float x, float y, float z, XMFLOAT3 xmf3Normal = XMFLOAT3(0.0f,
0.0f, 0.0f)) { m_xmf3Position = XMFLOAT3(x, y, z); m_xmf3Normal = xmf3Normal; }
   CIlluminatedVertex(XMFLOAT3 xmf3Position, XMFLOAT3 xmf3Normal = XMFLOAT3(0.0f, 0.0f,
0.0f)) { m_xmf3Position = xmf3Position; m_xmf3Normal = xmf3Normal; }
   ~CIlluminatedVertex() { }
};
```

❷ “CMeshIlluminated” 클래스를 다음과 같이 선언한다.

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

public:
   void CalculateTriangleListVertexNormals(XMFLOAT3 *pxmf3Normals, XMFLOAT3
*pxmf3Positions, int nVertices);
   void CalculateTriangleListVertexNormals(XMFLOAT3 *pxmf3Normals, XMFLOAT3
*pxmf3Positions, UINT nVertices, UINT *pnIndices, UINT nIndices);
   void CalculateTriangleStripVertexNormals(XMFLOAT3 *pxmf3Normals, XMFLOAT3
*pxmf3Positions, UINT nVertices, UINT *pnIndices, UINT nIndices);
   void CalculateVertexNormals(XMFLOAT3 *pxmf3Normals, XMFLOAT3 *pxmf3Positions, int
nVertices, UINT *pnIndices, int nIndices);
};
```

❸ “CCubeMeshIlluminated” 클래스를 다음과 같이 선언한다.

```
class CCubeMeshIlluminated : public CMeshIlluminated
{
public:
   CCubeMeshIlluminated(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, float fWidth = 2.0f, float fHeight = 2.0f, float fDepth = 2.0f);
   virtual ~CCubeMeshIlluminated();
};
```

⑤ “Mesh.cpp” 파일 수정하기

“Mesh.cpp” 파일을 다음과 같이 수정한다.

❶ “CMeshIlluminated” 클래스의 생성자와 소멸자를 다음과 같이 정의한다.

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

CMeshIlluminated::~CMeshIlluminated()
{
}
```

❷ “CMeshIlluminated” 클래스의 CalculateTriangleListVertexNormals() 함수를 다음과 같이 정의한다.

```
void CMeshIlluminated::CalculateTriangleListVertexNormals(XMFLOAT3 *pxmf3Normals,
XMFLOAT3 *pxmf3Positions, int nVertices)
{
   int nPrimitives = nVertices / 3;
   UINT nIndex0, nIndex1, nIndex2;
   for (int i = 0; i < nPrimitives; i++)
   {
      nIndex0 = i*3+0;
      nIndex1 = i*3+1;
      nIndex2 = i*3+2;
      XMFLOAT3 xmf3Edge01 = Vector3::Subtract(pxmf3Positions[nIndex1],
pxmf3Positions[nIndex0]);
      XMFLOAT3 xmf3Edge02 = Vector3::Subtract(pxmf3Positions[nIndex2],
pxmf3Positions[nIndex0]);
      pxmf3Normals[nIndex0] = pxmf3Normals[nIndex1] = pxmf3Normals[nIndex2] =
Vector3::CrossProduct(xmf3Edge01, xmf3Edge02, true);
   }
}
```

❸ “CMeshIlluminated” 클래스의 CalculateTriangleListVertexNormals() 함수를 다음과 같이 정의한다.

```
void CMeshIlluminated::CalculateTriangleListVertexNormals(XMFLOAT3 *pxmf3Normals,
XMFLOAT3 *pxmf3Positions, UINT nVertices, UINT *pnIndices, UINT nIndices)
{
   UINT nPrimitives = (pnIndices) ? (nIndices / 3) : (nVertices / 3);
   XMFLOAT3 xmf3SumOfNormal, xmf3Edge01, xmf3Edge02, xmf3Normal;
   UINT nIndex0, nIndex1, nIndex2;
   for (UINT j = 0; j < nVertices; j++)
   {
      xmf3SumOfNormal = XMFLOAT3(0.0f, 0.0f, 0.0f);
      for (UINT i = 0; i < nPrimitives; i++)
      {
         nIndex0 = pnIndices[i*3+0];
         nIndex1 = pnIndices[i*3+1];
         nIndex2 = pnIndices[i*3+2];
         if (pnIndices && ((nIndex0 == j) || (nIndex1 == j) || (nIndex2 == j)))
         {
            xmf3Edge01 = Vector3::Subtract(pxmf3Positions[nIndex1],
pxmf3Positions[nIndex0]);
            xmf3Edge02 = Vector3::Subtract(pxmf3Positions[nIndex2],
pxmf3Positions[nIndex0]);
            xmf3Normal = Vector3::CrossProduct(xmf3Edge01, xmf3Edge02, false);
            xmf3SumOfNormal = Vector3::Add(xmf3SumOfNormal, xmf3Normal);
         }
      }
      pxmf3Normals[j] = Vector3::Normalize(xmf3SumOfNormal);
   }
}
```

❹ “CMeshIlluminated” 클래스의 CalculateTriangleStripVertexNormals() 함수를 다음과 같이 정의한다.

```
void CMeshIlluminated::CalculateTriangleStripVertexNormals(XMFLOAT3 *pxmf3Normals,
XMFLOAT3 *pxmf3Positions, UINT nVertices, UINT *pnIndices, UINT nIndices)
{
   UINT nPrimitives = (pnIndices) ? (nIndices - 2) : (nVertices - 2);
   XMFLOAT3 xmf3SumOfNormal(0.0f, 0.0f, 0.0f);
   UINT nIndex0, nIndex1, nIndex2;
   for (UINT j = 0; j < nVertices; j++)
   {
      xmf3SumOfNormal = XMFLOAT3(0.0f, 0.0f, 0.0f);
      for (UINT i = 0; i < nPrimitives; i++)
      {
         nIndex0 = ((i % 2) == 0) ? (i + 0) : (i + 1);
         if (pnIndices) nIndex0 = pnIndices[nIndex0];
         nIndex1 = ((i % 2) == 0) ? (i + 1) : (i + 0);
         if (pnIndices) nIndex1 = pnIndices[nIndex1];
         nIndex2 = (pnIndices) ? pnIndices[i + 2] : (i + 2);
         if ((nIndex0 == j) || (nIndex1 == j) || (nIndex2 == j))
         {
            XMFLOAT3 xmf3Edge01 = Vector3::Subtract(pxmf3Positions[nIndex1],
pxmf3Positions[nIndex0]);
            XMFLOAT3 xmf3Edge02 = Vector3::Subtract(pxmf3Positions[nIndex2],
pxmf3Positions[nIndex0]);
            XMFLOAT3 xmf3Normal = Vector3::CrossProduct(xmf3Edge01, xmf3Edge02, true);
            xmf3SumOfNormal = Vector3::Add(xmf3SumOfNormal, xmf3Normal);
         }
      }
      pxmf3Normals[j] = Vector3::Normalize(xmf3SumOfNormal);
   }
}
```

❹ “CMeshIlluminated” 클래스의 CalculateVertexNormals() 함수를 다음과 같이 정의한다.

```
void CMeshIlluminated::CalculateVertexNormals(XMFLOAT3 *pxmf3Normals, XMFLOAT3
*pxmf3Positions, int nVertices, UINT *pnIndices, int nIndices)
{
   switch (m_d3dPrimitiveTopology)
   {
      case D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST:
         if (pnIndices)
            CalculateTriangleListVertexNormals(pxmf3Normals, pxmf3Positions, nVertices,
pnIndices, nIndices);
         else
            CalculateTriangleListVertexNormals(pxmf3Normals, pxmf3Positions, nVertices);
         break;
      case D3D_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP:
         CalculateTriangleStripVertexNormals(pxmf3Normals, pxmf3Positions, nVertices,
pnIndices, nIndices);
         break;
      default:
         break;
   }
```

❺ “CCubeMeshIlluminated” 클래스의 생성자와 소멸자를 다음과 같이 정의한다.

```
CCubeMeshIlluminated::CCubeMeshIlluminated(ID3D12Device *pd3dDevice,
ID3D12GraphicsCommandList *pd3dCommandList, float fWidth, float fHeight, float fDepth) :
CMeshIlluminated(pd3dDevice, pd3dCommandList)
{
   m_nVertices = 8;
   m_nStride = sizeof(CIlluminatedVertex);
   m_nOffset = 0;
   m_nSlot = 0;
   m_d3dPrimitiveTopology = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST;

   m_nIndices = 36;
   UINT pnIndices[36];

   pnIndices[0] = 3; pnIndices[1] = 1; pnIndices[2] = 0;
   pnIndices[3] = 2; pnIndices[4] = 1; pnIndices[5] = 3;
   pnIndices[6] = 0; pnIndices[7] = 5; pnIndices[8] = 4;
   pnIndices[9] = 1; pnIndices[10] = 5; pnIndices[11] = 0;
   pnIndices[12] = 3; pnIndices[13] = 4; pnIndices[14] = 7;
   pnIndices[15] = 0; pnIndices[16] = 4; pnIndices[17] = 3;
   pnIndices[18] = 1; pnIndices[19] = 6; pnIndices[20] = 5;
   pnIndices[21] = 2; pnIndices[22] = 6; pnIndices[23] = 1;
   pnIndices[24] = 2; pnIndices[25] = 7; pnIndices[26] = 6;
   pnIndices[27] = 3; pnIndices[28] = 7; pnIndices[29] = 2;
   pnIndices[30] = 6; pnIndices[31] = 4; pnIndices[32] = 5;
   pnIndices[33] = 7; pnIndices[34] = 4; pnIndices[35] = 6;

   m_pd3dIndexBuffer = ::CreateBufferResource(pd3dDevice, pd3dCommandList, pnIndices,
sizeof(UINT) * m_nIndices, D3D12_HEAP_TYPE_DEFAULT, D3D12_RESOURCE_STATE_INDEX_BUFFER,
&m_pd3dIndexUploadBuffer);

   m_d3dIndexBufferView.BufferLocation = m_pd3dIndexBuffer->GetGPUVirtualAddress();
   m_d3dIndexBufferView.Format = DXGI_FORMAT_R32_UINT;
   m_d3dIndexBufferView.SizeInBytes = sizeof(UINT) * m_nIndices;

   XMFLOAT3 pxmf3Positions[8];

   float fx = fWidth*0.5f, fy = fHeight*0.5f, fz = fDepth*0.5f;

   pxmf3Positions[0] = XMFLOAT3(-fx, +fy, -fz);
   pxmf3Positions[1] = XMFLOAT3(+fx, +fy, -fz);
   pxmf3Positions[2] = XMFLOAT3(+fx, +fy, +fz);
   pxmf3Positions[3] = XMFLOAT3(-fx, +fy, +fz);
   pxmf3Positions[4] = XMFLOAT3(-fx, -fy, -fz);
   pxmf3Positions[5] = XMFLOAT3(+fx, -fy, -fz);
   pxmf3Positions[6] = XMFLOAT3(+fx, -fy, +fz);
   pxmf3Positions[7] = XMFLOAT3(-fx, -fy, +fz);

   XMFLOAT3 pxmf3Normals[8];
   for (int i = 0; i < 8; i++) pxmf3Normals[i] = XMFLOAT3(0.0f, 0.0f, 0.0f);

   CalculateVertexNormals(pxmf3Normals, pxmf3Positions, m_nVertices, pnIndices,
```

```
m_nIndices);

   CIlluminatedVertex pVertices[8];
   for (int i = 0; i < 8; i++) pVertices[i] = CIlluminatedVertex(pxmf3Positions[i],
pxmf3Normals[i]);

   m_pd3dVertexBuffer = CreateBufferResource(pd3dDevice, pd3dCommandList, pVertices,
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;
}

CCubeMeshIlluminated::~CCubeMeshIlluminated()
{
}
```

⑥ “GameObject.h” 파일 수정하기

❶ 재질을 위한 “MATERIAL” 구조체와 “CMaterial” 클래스를 다음과 같이 선언한다.

```
struct MATERIAL
{
   XMFLOAT4                    m_xmf4Ambient;
   XMFLOAT4                    m_xmf4Diffuse;
   XMFLOAT4                    m_xmf4Specular; //(r,g,b,a=power)
   XMFLOAT4                    m_xmf4Emissive;
};

class CMaterial
{
public:
   CMaterial();
   virtual ~CMaterial();

private:
   int                         m_nReferences = 0;

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

//재질의 기본 색상
   XMFLOAT4                    m_xmf4Albedo = XMFLOAT4(1.0f, 1.0f, 1.0f, 1.0f);

//재질의 번호
   UINT                        m_nReflection = 0;
//재질을 적용하여 렌더링을 하기 위한 쉐이더
   CShader                     *m_pShader = NULL;

   void SetAlbedo(XMFLOAT4& xmf4Albedo) { m_xmf4Albedo = xmf4Albedo; }
```

```
   void SetReflection(UINT nReflection) { m_nReflection = nReflection; }
   void SetShader(CShader *pShader);
};
```

❷ “CGameObject” 클래스의 m_pShader 멤버 변수를 삭제하고 다음을 추가한다.

```
   CShader                     *m_pShader = NULL;
   CMaterial                   *m_pMaterial = NULL;
//게임 객체가 쉐이더를 가지지 않고 재질을 가진다.
```

❸ “CGameObject” 클래스의 SetMaterial() 멤버 함수를 다음과 같이 선언한다.

```
   void SetMaterial(CMaterial *pMaterial);
   void SetMaterial(UINT nReflection);
```

⑦ “GameObject.cpp” 파일 수정하기

❶ “CMaterial” 클래스의 생성자와 소멸자를 다음과 같이 정의한다.

```
CMaterial::CMaterial()
{
   m_xmf4Albedo = XMFLOAT4(1.0f, 1.0f, 1.0f, 1.0f);
}

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

❷ “CMaterial” 클래스의 SetShader() 함수를 다음과 같이 정의한다.

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

❸ “CGameObject” 클래스의 소멸자에 다음을 추가한다.

```
   if (m_pMaterial) m_pMaterial->Release();
```

❹ “CGameObject” 클래스의 SetShader() 함수를 다음과 같이 정의한다.

```
void CGameObject::SetShader(CShader *pShader)
{
   if (!m_pMaterial)
   {
      m_pMaterial = new CMaterial();
      m_pMaterial->AddRef();
   }
   if (m_pMaterial) m_pMaterial->SetShader(pShader);
}
```

❺ “CGameObject” 클래스의 SetMaterial() 함수를 다음과 같이 정의한다.

```
void CGameObject::SetMaterial(CMaterial *pMaterial)
{
   if (m_pMaterial) m_pMaterial->Release();
   m_pMaterial = pMaterial;
   if (m_pMaterial) m_pMaterial->AddRef();
}

void CGameObject::SetMaterial(UINT nReflection)
{
   if (!m_pMaterial) m_pMaterial = new CMaterial();
   m_pMaterial->m_nReflection = nReflection;
}
```

❻ “CGameObject” 클래스의 Render() 함수를 다음과 같이 변경한다.

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

   if (m_pMaterial)
   {
      if (m_pMaterial->m_pShader)
      {
         m_pMaterial->m_pShader->Render(pd3dCommandList, pCamera);
         m_pMaterial->m_pShader->UpdateShaderVariable(pd3dCommandList, &m_xmf4x4World);
      }
   }

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

⑧ “Scene.h” 파일 변경하기

❶ “Scene.h” 파일에 조명과 재질을 위한 구조체를 다음과 같이 선언한다.

```
struct LIGHT
{
   XMFLOAT4            m_xmf4Ambient;
```

```
    XMFLOAT4            m_xmf4Diffuse;
    XMFLOAT4            m_xmf4Specular;
    XMFLOAT3            m_xmf3Position;
    float               m_fFalloff;
    XMFLOAT3            m_xmf3Direction;
    float               m_fTheta; //cos(m_fTheta)
    XMFLOAT3            m_xmf3Attenuation;
    float               m_fPhi; //cos(m_fPhi)
    bool                m_bEnable;
    int                 m_nType;
    float               m_fRange;
    float               padding;
};

struct LIGHTS
{
    LIGHT               m_pLights[MAX_LIGHTS];
    XMFLOAT4            m_xmf4GlobalAmbient;
};

struct MATERIALS
{
    MATERIAL            m_pReflections[MAX_MATERIALS];
};
```

❷ “CScene” 클래스에 다음 멤버 함수를 선언한다.

```
public:
//씬의 모든 조명과 재질을 생성
    void BuildLightsAndMaterials();

//씬의 모든 조명과 재질을 위한 리소스를 생성하고 갱신
    virtual void CreateShaderVariables(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList);
    virtual void UpdateShaderVariables(ID3D12GraphicsCommandList *pd3dCommandList);
    virtual void ReleaseShaderVariables();
```

❸ “CScene” 클래스에 다음 멤버 변수를 선언한다.

```
public:
    CPlayer                     *m_pPlayer = NULL;

protected:
//씬의 조명
    LIGHTS                      *m_pLights = NULL;

//조명을 나타내는 리소스와 리소스에 대한 포인터이다.
    ID3D12Resource              *m_pd3dcbLights = NULL;
    LIGHTS                      *m_pcbMappedLights = NULL;

//씬의 객체들에 적용되는 재질
    MATERIALS                   *m_pMaterials = NULL;
```

```
//재질을 나타내는 리소스와 리소스에 대한 포인터이다.
   ID3D12Resource              *m_pd3dcbMaterials = NULL;
   MATERIAL                    *m_pcbMappedMaterials = NULL;
```

⑨ “Scene.cpp” 파일 변경하기

❶ “CScene” 클래스의 BuildLightsAndMaterials() 함수를 다음과 같이 정의한다.

```
void CScene::BuildLightsAndMaterials()
{
   m_pLights = new LIGHTS;
   ::ZeroMemory(m_pLights, sizeof(LIGHTS));

   m_pLights->m_xmf4GlobalAmbient = XMFLOAT4(0.1f, 0.1f, 0.1f, 1.0f);

   m_pLights->m_pLights[0].m_bEnable = true;
   m_pLights->m_pLights[0].m_nType = POINT_LIGHT;
   m_pLights->m_pLights[0].m_fRange = 100.0f;
   m_pLights->m_pLights[0].m_xmf4Ambient = XMFLOAT4(0.1f, 0.0f, 0.0f, 1.0f);
   m_pLights->m_pLights[0].m_xmf4Diffuse = XMFLOAT4(0.8f, 0.0f, 0.0f, 1.0f);
   m_pLights->m_pLights[0].m_xmf4Specular = XMFLOAT4(0.1f, 0.1f, 0.1f, 0.0f);
   m_pLights->m_pLights[0].m_xmf3Position = XMFLOAT3(130.0f, 30.0f, 30.0f);
   m_pLights->m_pLights[0].m_xmf3Direction = XMFLOAT3(0.0f, 0.0f, 0.0f);
   m_pLights->m_pLights[0].m_xmf3Attenuation = XMFLOAT3(1.0f, 0.001f, 0.0001f);
   m_pLights->m_pLights[1].m_bEnable = true;
   m_pLights->m_pLights[1].m_nType = SPOT_LIGHT;
   m_pLights->m_pLights[1].m_fRange = 50.0f;
   m_pLights->m_pLights[1].m_xmf4Ambient = XMFLOAT4(0.1f, 0.1f, 0.1f, 1.0f);
   m_pLights->m_pLights[1].m_xmf4Diffuse = XMFLOAT4(0.4f, 0.4f, 0.4f, 1.0f);
   m_pLights->m_pLights[1].m_xmf4Specular = XMFLOAT4(0.1f, 0.1f, 0.1f, 0.0f);
   m_pLights->m_pLights[1].m_xmf3Position = XMFLOAT3(-50.0f, 20.0f, -5.0f);
   m_pLights->m_pLights[1].m_xmf3Direction = XMFLOAT3(0.0f, 0.0f, 1.0f);
   m_pLights->m_pLights[1].m_xmf3Attenuation = XMFLOAT3(1.0f, 0.01f, 0.0001f);
   m_pLights->m_pLights[1].m_fFalloff = 8.0f;
   m_pLights->m_pLights[1].m_fPhi = (float)cos(XMConvertToRadians(40.0f));
   m_pLights->m_pLights[1].m_fTheta = (float)cos(XMConvertToRadians(20.0f));
   m_pLights->m_pLights[2].m_bEnable = true;
   m_pLights->m_pLights[2].m_nType = DIRECTIONAL_LIGHT;
   m_pLights->m_pLights[2].m_xmf4Ambient = XMFLOAT4(0.1f, 0.1f, 0.1f, 1.0f);
   m_pLights->m_pLights[2].m_xmf4Diffuse = XMFLOAT4(0.3f, 0.3f, 0.3f, 1.0f);
   m_pLights->m_pLights[2].m_xmf4Specular = XMFLOAT4(0.0f, 0.0f, 0.0f, 0.0f);
   m_pLights->m_pLights[2].m_xmf3Direction = XMFLOAT3(1.0f, 0.0f, 0.0f);
   m_pLights->m_pLights[3].m_bEnable = true;
   m_pLights->m_pLights[3].m_nType = SPOT_LIGHT;
   m_pLights->m_pLights[3].m_fRange = 60.0f;
   m_pLights->m_pLights[3].m_xmf4Ambient = XMFLOAT4(0.1f, 0.1f, 0.1f, 1.0f);
   m_pLights->m_pLights[3].m_xmf4Diffuse = XMFLOAT4(0.5f, 0.0f, 0.0f, 1.0f);
   m_pLights->m_pLights[3].m_xmf4Specular = XMFLOAT4(0.0f, 0.0f, 0.0f, 0.0f);
   m_pLights->m_pLights[3].m_xmf3Position = XMFLOAT3(-150.0f, 30.0f, 30.0f);
   m_pLights->m_pLights[3].m_xmf3Direction = XMFLOAT3(0.0f, 1.0f, 1.0f);
   m_pLights->m_pLights[3].m_xmf3Attenuation = XMFLOAT3(1.0f, 0.01f, 0.0001f);
   m_pLights->m_pLights[3].m_fFalloff = 8.0f;
   m_pLights->m_pLights[3].m_fPhi = (float)cos(XMConvertToRadians(90.0f));
```

```
   m_pLights->m_pLights[3].m_fTheta = (float)cos(XMConvertToRadians(30.0f));

   m_pMaterials = new MATERIALS;
   ::ZeroMemory(m_pMaterials, sizeof(MATERIALS));

   m_pMaterials->m_pReflections[0] = { XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f), XMFLOAT4(1.0f,
0.0f, 0.0f, 1.0f), XMFLOAT4(1.0f, 1.0f, 1.0f, 5.0f), XMFLOAT4(0.0f, 0.0f, 0.0f, 1.0f) };
   m_pMaterials->m_pReflections[1] = { XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f), XMFLOAT4(0.0f,
1.0f, 0.0f, 1.0f), XMFLOAT4(1.0f, 1.0f, 1.0f, 10.0f), XMFLOAT4(0.0f, 0.0f, 0.0f, 1.0f) };
   m_pMaterials->m_pReflections[2] = { XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f), XMFLOAT4(0.0f,
0.0f, 1.0f, 1.0f), XMFLOAT4(1.0f, 1.0f, 1.0f, 15.0f), XMFLOAT4(0.0f, 0.0f, 0.0f, 1.0f) };
   m_pMaterials->m_pReflections[3] = { XMFLOAT4(0.5f, 0.0f, 1.0f, 1.0f), XMFLOAT4(0.0f,
0.5f, 1.0f, 1.0f), XMFLOAT4(1.0f, 1.0f, 1.0f, 20.0f), XMFLOAT4(0.0f, 0.0f, 0.0f, 1.0f) };
   m_pMaterials->m_pReflections[4] = { XMFLOAT4(0.0f, 0.5f, 1.0f, 1.0f), XMFLOAT4(0.5f,
0.0f, 1.0f, 1.0f), XMFLOAT4(1.0f, 1.0f, 1.0f, 25.0f), XMFLOAT4(0.0f, 0.0f, 0.0f, 1.0f) };
   m_pMaterials->m_pReflections[5] = { XMFLOAT4(0.0f, 0.5f, 0.5f, 1.0f), XMFLOAT4(0.0f,
0.0f, 1.0f, 1.0f), XMFLOAT4(1.0f, 1.0f, 1.0f, 30.0f), XMFLOAT4(0.0f, 0.0f, 0.0f, 1.0f) };
   m_pMaterials->m_pReflections[6] = { XMFLOAT4(0.5f, 0.5f, 1.0f, 1.0f), XMFLOAT4(0.5f,
0.5f, 1.0f, 1.0f), XMFLOAT4(1.0f, 1.0f, 1.0f, 35.0f), XMFLOAT4(0.0f, 0.0f, 0.0f, 1.0f) };
   m_pMaterials->m_pReflections[7] = { XMFLOAT4(1.0f, 0.5f, 1.0f, 1.0f), XMFLOAT4(1.0f,
0.0f, 1.0f, 1.0f), XMFLOAT4(1.0f, 1.0f, 1.0f, 40.0f), XMFLOAT4(0.0f, 0.0f, 0.0f, 1.0f) };
}
```

❷ “CScene” 클래스의 BuildObjects() 함수를 다음과 같이 수정한다.

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

   m_nShaders = 1;
   m_pShaders = new CObjectsShader[m_nShaders];
   m_pShaders[0].CreateShader(pd3dDevice, m_pd3dGraphicsRootSignature);
   m_pShaders[0].BuildObjects(pd3dDevice, pd3dCommandList);

   BuildLightsAndMaterials();

   CreateShaderVariables(pd3dDevice, pd3dCommandList);
}
```

❸ “CScene” 클래스의 ReleaseObjects() 함수에 다음을 추가한다.

```
   if (m_pLights) delete m_pLights;
   if (m_pMaterials) delete m_pMaterials;
```

❹ “CScene” 클래스의 CreateGraphicsRootSignature() 함수를 다음과 같이 수정한다.

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

   D3D12_ROOT_PARAMETER pd3dRootParameters[5];
```

```
   pd3dRootParameters[0].ParameterType = D3D12_ROOT_PARAMETER_TYPE_CBV;
   pd3dRootParameters[0].Descriptor.ShaderRegister = 0; //Player
   pd3dRootParameters[0].Descriptor.RegisterSpace = 0;
   pd3dRootParameters[0].ShaderVisibility = D3D12_SHADER_VISIBILITY_VERTEX;

   pd3dRootParameters[1].ParameterType = D3D12_ROOT_PARAMETER_TYPE_CBV;
   pd3dRootParameters[1].Descriptor.ShaderRegister = 1; //Camera
   pd3dRootParameters[1].Descriptor.RegisterSpace = 0;
   pd3dRootParameters[1].ShaderVisibility = D3D12_SHADER_VISIBILITY_ALL;

   pd3dRootParameters[2].ParameterType = D3D12_ROOT_PARAMETER_TYPE_CBV;
   pd3dRootParameters[2].Descriptor.ShaderRegister = 2; //GameObject
   pd3dRootParameters[2].Descriptor.RegisterSpace = 0;
   pd3dRootParameters[2].ShaderVisibility = D3D12_SHADER_VISIBILITY_ALL;

   pd3dRootParameters[3].ParameterType = D3D12_ROOT_PARAMETER_TYPE_CBV;
   pd3dRootParameters[3].Descriptor.ShaderRegister = 3; //Materials
   pd3dRootParameters[3].Descriptor.RegisterSpace = 0;
   pd3dRootParameters[3].ShaderVisibility = D3D12_SHADER_VISIBILITY_ALL;

   pd3dRootParameters[4].ParameterType = D3D12_ROOT_PARAMETER_TYPE_CBV;
   pd3dRootParameters[4].Descriptor.ShaderRegister = 4; //Lights
   pd3dRootParameters[4].Descriptor.RegisterSpace = 0;
   pd3dRootParameters[4].ShaderVisibility = D3D12_SHADER_VISIBILITY_ALL;

   D3D12_ROOT_SIGNATURE_FLAGS d3dRootSignatureFlags =
D3D12_ROOT_SIGNATURE_FLAG_ALLOW_INPUT_ASSEMBLER_INPUT_LAYOUT | ...;
   ...
```

❺ “CScene” 클래스의 CreateShaderVariables() 함수를 다음과 같이 정의한다.

```
void CScene::CreateShaderVariables(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList)
{
   UINT ncbElementBytes = ((sizeof(LIGHTS) + 255) & ~255); //256의 배수
   m_pd3dcbLights = ::CreateBufferResource(pd3dDevice, pd3dCommandList, NULL,
ncbElementBytes, D3D12_HEAP_TYPE_UPLOAD,
D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER, NULL);

   m_pd3dcbLights->Map(0, NULL, (void **)&m_pcbMappedLights);

   UINT ncbMaterialBytes = ((sizeof(MATERIALS) + 255) & ~255); //256의 배수
   m_pd3dcbMaterials = ::CreateBufferResource(pd3dDevice, pd3dCommandList, NULL,
ncbMaterialBytes, D3D12_HEAP_TYPE_UPLOAD,
D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER, NULL);

   m_pd3dcbMaterials->Map(0, NULL, (void **)&m_pcbMappedMaterials);
}
```

❻ “CScene” 클래스의 UpdateShaderVariables() 함수를 다음과 같이 정의한다.

```
void CScene::UpdateShaderVariables(ID3D12GraphicsCommandList *pd3dCommandList)
{
   ::memcpy(m_pcbMappedLights, m_pLights, sizeof(LIGHTS));
   ::memcpy(m_pcbMappedMaterials, m_pMaterials, sizeof(MATERIALS));
}
```

❼ “CScene” 클래스의 ReleaseShaderVariables() 함수를 다음과 같이 정의한다.

```
void CScene::ReleaseShaderVariables()
{
   if (m_pd3dcbLights)
   {
      m_pd3dcbLights->Unmap(0, NULL);
      m_pd3dcbLights->Release();
   }
   if (m_pd3dcbMaterials)
   {
      m_pd3dcbMaterials->Unmap(0, NULL);
      m_pd3dcbMaterials->Release();
   }
}
```

❽ “CScene” 클래스의 AnimateObjects() 함수를 다음과 같이 수정한다.

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

//조명의 위치와 방향을 플레이어의 위치와 방향으로 변경한다.
   if (m_pLights)
   {
      m_pLights->m_pLights[1].m_xmf3Position = m_pPlayer->GetPosition();
      m_pLights->m_pLights[1].m_xmf3Direction = m_pPlayer->GetLookVector();
   }
}
```

❾ “CScene” 클래스의 Render() 함수를 다음과 같이 수정한다.

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

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

   UpdateShaderVariables(pd3dCommandList);
```

```
//조명 리소스에 대한 상수 버퍼 뷰를 쉐이더 변수에 연결(바인딩)한다.
   D3D12_GPU_VIRTUAL_ADDRESS d3dcbLightsGpuVirtualAddress =
m_pd3dcbLights->GetGPUVirtualAddress();
   pd3dCommandList->SetGraphicsRootConstantBufferView(4, d3dcbLightsGpuVirtualAddress);

//재질 리소스에 대한 상수 버퍼 뷰를 쉐이더 변수에 연결(바인딩)한다.
   D3D12_GPU_VIRTUAL_ADDRESS d3dcbMaterialsGpuVirtualAddress =
m_pd3dcbMaterials->GetGPUVirtualAddress();
   pd3dCommandList->SetGraphicsRootConstantBufferView(3,
d3dcbMaterialsGpuVirtualAddress);

   for (int i = 0; i < m_nShaders; i++)
   {
      m_pShaders[i].Render(pd3dCommandList, pCamera);
   }
}
```

⑩ “Shader.h” 파일 변경하기

❶ “Shader” 파일에 다음 구조체를 추가한다.

```
//객체를 렌더링할 때 적용하는 상수 버퍼 데이터
struct CB_GAMEOBJECT_INFO
{
   XMFLOAT4X4                  m_xmf4x4World;
//객체에 적용될 재질 번호
   UINT                        m_nMaterial;
};

//플레이어 객체를 렌더링할 때 적용하는 상수 버퍼 데이터
struct CB_PLAYER_INFO
{
   XMFLOAT4X4                  m_xmf4x4World;
};
```

❷ “CShader” 클래스에 다음 멤버 함수를 추가한다.

```
   void UpdateShaderVariable(ID3D12GraphicsCommandList *pd3dCommandList, XMFLOAT4X4
*pxmf4x4World) { }
   void UpdateShaderVariable(ID3D12GraphicsCommandList *pd3dCommandList, MATERIAL
*pMaterial) { }
```

❸ “CObjectsShader” 클래스에 다음 멤버 함수를 선언한다.

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

❹ “CObjectsShader” 클래스에 다음 멤버 변수를 선언한다.

```
protected:
//쉐이더 객체에 포함되어 있는 모든 게임 객체들에 대한 리소스와 리소스 포인터
   ID3D12Resource              *m_pd3dcbGameObjects = NULL;
   UINT8                       *m_pcbMappedGameObjects = NULL;
```

❺ “CPlayerShader” 클래스를 다음과 같이 수정한다.

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

   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);

   virtual void CreateShaderVariables(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList);
   virtual void ReleaseShaderVariables();
   virtual void UpdateShaderVariables(ID3D12GraphicsCommandList *pd3dCommandList) { }
   virtual void UpdateShaderVariable(ID3D12GraphicsCommandList *pd3dCommandList,
XMFLOAT4X4 *pxmf4x4World);

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

protected:
//플레이어 객체에 대한 리소스와 리소스 포인터
   ID3D12Resource              *m_pd3dcbPlayer = NULL;
   CB_PLAYER_INFO              *m_pcbMappedPlayer = NULL;
};
```

⑪ “Shader.cpp” 파일 변경하기

❶ “CShader” 클래스의 CompileShaderFromFile() 함수를 다음과 같이 수정한다.

```
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

   ID3DBlob *pd3dErrorBlob = NULL;
   HRESULT hResult = ::D3DCompileFromFile(pszFileName, NULL,
D3D_COMPILE_STANDARD_FILE_INCLUDE, pszShaderName, pszShaderProfile, nCompileFlags, 0,
```

```
ppd3dShaderBlob, &pd3dErrorBlob);

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

   return(d3dShaderByteCode);
}
```

❷ “CObjectsShader” 클래스의 CreateInputLayout() 함수를 다음과 같이 수정한다.

```
D3D12_INPUT_LAYOUT_DESC CObjectsShader::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] ={ "NORMAL", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 12,
D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, 0 };

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

   return(d3dInputLayoutDesc);
}
```

❸ “CObjectsShader” 클래스의 CreateVertexShader() 함수와 CreatePixelShader() 함수를 다음과 같이 수정한다.

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

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

❹ “CObjectsShader” 클래스의 CreateShaderVariables() 함수, UpdateShaderVariables() 함수 그리고 ReleaseShaderVariables() 함수를 다음과 같이 수정한다.

```
//객체의 정보를 저장하기 위한 리소스를 생성하고 리소스에 대한 포인터를 가져온다.
void CObjectsShader::CreateShaderVariables(ID3D12Device *pd3dDevice,
ID3D12GraphicsCommandList *pd3dCommandList)
{
```

```
   UINT ncbGameObjectBytes = ((sizeof(CB_GAMEOBJECT_INFO) + 255) & ~255); //256의 배수
   m_pd3dcbGameObjects = ::CreateBufferResource(pd3dDevice, pd3dCommandList, NULL,
ncbGameObjectBytes * m_nObjects, D3D12_HEAP_TYPE_UPLOAD,
D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER, NULL);

   m_pd3dcbGameObjects->Map(0, NULL, (void **)&m_pcbMappedGameObjects);
}

//객체의 월드변환 행렬과 재질 번호를 상수 버퍼에 쓴다.
void CObjectsShader::UpdateShaderVariables(ID3D12GraphicsCommandList *pd3dCommandList)
{
   UINT ncbGameObjectBytes = ((sizeof(CB_GAMEOBJECT_INFO) + 255) & ~255); //256의 배수
   XMFLOAT4X4 xmf4x4World;
   for (int j = 0; j < m_nObjects; j++)
   {
      XMStoreFloat4x4(&xmf4x4World,
XMMatrixTranspose(XMLoadFloat4x4(&m_ppObjects[j]->m_xmf4x4World)));
      CB_GAMEOBJECT_INFO *pbMappedcbGameObject = (CB_GAMEOBJECT_INFO
*)(m_pcbMappedGameObjects + (j * ncbGameObjectBytes));
      ::memcpy(&pbMappedcbGameObject->m_xmf4x4World, &xmf4x4World, sizeof(XMFLOAT4X4));
      pbMappedcbGameObject->m_nMaterial = m_ppObjects[j]->m_pMaterial->m_nReflection;
   }
}

void CObjectsShader::ReleaseShaderVariables()
{
   if (m_pd3dcbGameObjects)
   {
      m_pd3dcbGameObjects->Unmap(0, NULL);
      m_pd3dcbGameObjects->Release();
   }
}
```

❺ “CObjectsShader” 클래스의 BuildObjects() 함수를 다음과 같이 수정한다.

```
void CObjectsShader::BuildObjects(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList)
{
   CCubeMeshIlluminated *pCubeMesh = new CCubeMeshIlluminated(pd3dDevice,
pd3dCommandList, 12.0f, 12.0f, 12.0f);

   int xObjects = 10, yObjects = 10, zObjects = 10, i = 0;

   m_nObjects = (xObjects * 2 + 1) * (yObjects * 2 + 1) * (zObjects * 2 + 1);

   m_ppObjects = new CGameObject*[m_nObjects];

   float fxPitch = 12.0f * 2.5f;
   float fyPitch = 12.0f * 2.5f;
   float fzPitch = 12.0f * 2.5f;

   CRotatingObject *pRotatingObject = NULL;
   for (int x = -xObjects; x <= xObjects; x++)
   {
```

```
      for (int y = -yObjects; y <= yObjects; y++)
      {
         for (int z = -zObjects; z <= zObjects; z++)
         {
            pRotatingObject = new CRotatingObject();
            pRotatingObject->SetMaterial(i % MAX_MATERIALS);
            pRotatingObject->SetMesh(pCubeMesh);
            pRotatingObject->SetPosition(fxPitch*x, fyPitch*y, fzPitch*z);
            pRotatingObject->SetRotationAxis(XMFLOAT3(0.0f, 1.0f, 0.0f));
            pRotatingObject->SetRotationSpeed(10.0f * (i % 10));
            m_ppObjects[i++] = pRotatingObject;
         }
      }
   }

   CreateShaderVariables(pd3dDevice, pd3dCommandList);
}
```

❻ “CObjectsShader” 클래스의 Render() 함수를 다음과 같이 수정한다.

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

   UpdateShaderVariables(pd3dCommandList);

   UINT ncbGameObjectBytes = ((sizeof(CB_GAMEOBJECT_INFO) + 255) & ~255);
   D3D12_GPU_VIRTUAL_ADDRESS d3dcbGameObjectGpuVirtualAddress =
m_pd3dcbGameObjects->GetGPUVirtualAddress();

   for (int j = 0; j < m_nObjects; j++)
   {
      if (m_ppObjects[j])
      {
         pd3dCommandList->SetGraphicsRootConstantBufferView(2,
d3dcbGameObjectGpuVirtualAddress + (ncbGameObjectBytes * j));
         m_ppObjects[j]->Render(pd3dCommandList, pCamera);
      }
   }
}
```

❼ “CPlayerShader” 클래스의 CreateShaderVariables() 함수, ReleaseShaderVariables() 함수, 그리고 UpdateShaderVariable() 함수를 다음과 같이 정의한다.

```
void CPlayerShader::CreateShaderVariables(ID3D12Device *pd3dDevice,
ID3D12GraphicsCommandList *pd3dCommandList)
{
   UINT ncbElementBytes = ((sizeof(CB_PLAYER_INFO) + 255) & ~255); //256의 배수
   m_pd3dcbPlayer = ::CreateBufferResource(pd3dDevice, pd3dCommandList, NULL,
ncbElementBytes, D3D12_HEAP_TYPE_UPLOAD,
D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER, NULL);
```

```
   m_pd3dcbPlayer->Map(0, NULL, (void **)&m_pcbMappedPlayer);
}

void CPlayerShader::ReleaseShaderVariables()
{
   if (m_pd3dcbPlayer)
   {
      m_pd3dcbPlayer->Unmap(0, NULL);
      m_pd3dcbPlayer->Release();
   }
}

void CPlayerShader::UpdateShaderVariable(ID3D12GraphicsCommandList *pd3dCommandList,
XMFLOAT4X4 *pxmf4x4World)
{
   XMFLOAT4X4 xmf4x4World;
   XMStoreFloat4x4(&xmf4x4World, XMMatrixTranspose(XMLoadFloat4x4(pxmf4x4World)));
   ::memcpy(&m_pcbMappedPlayer->m_xmf4x4World, &xmf4x4World, sizeof(XMFLOAT4X4));
}
```

❽ “CPlayerShader” 클래스의 CreateVertexShader(), CreatePixelShader() 함수를 다음과 같이 정의한다.

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

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

❾ “CPlayerShader” 클래스의 Render() 함수를 다음과 같이 정의한다.

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

   D3D12_GPU_VIRTUAL_ADDRESS d3dGpuVirtualAddress =
m_pd3dcbPlayer->GetGPUVirtualAddress();
   pd3dCommandList->SetGraphicsRootConstantBufferView(0, d3dGpuVirtualAddress);
}
```

⑫ “Player.cpp” 파일 변경하기

❶ “CAirplanePlayer” 클래스의 생성자를 다음과 같이 수정한다.

```
CAirplanePlayer::CAirplanePlayer(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
```

```
*pd3dCommandList, ID3D12RootSignature *pd3dGraphicsRootSignature)
{
   CAirplaneMeshDiffused *pAirplaneMesh = new CAirplaneMeshDiffused(pd3dDevice,
pd3dCommandList, 20.0f, 20.0f, 4.0f, XMFLOAT4(0.0f, 0.5f, 0.0f, 0.0f));
   SetMesh(pAirplaneMesh);

   m_pCamera = ChangeCamera(SPACESHIP_CAMERA/*THIRD_PERSON_CAMERA*/, 0.0f);
   SetPosition(XMFLOAT3(0.0f, 0.0f, -50.0f));

   CreateShaderVariables(pd3dDevice, pd3dCommandList);

   CPlayerShader *pShader = new CPlayerShader();
   pShader->CreateShader(pd3dDevice, pd3dGraphicsRootSignature);
   pShader->CreateShaderVariables(pd3dDevice, pd3dCommandList);
   SetShader(pShader);
}
```

⑬ “GameFramework.cpp” 파일 변경하기

❶ “CGameFramework” 클래스의 BuildObjects() 함수를 다음과 같이 수정한다.

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

   m_pScene = new CScene();
   if (m_pScene) m_pScene->BuildObjects(m_pd3dDevice, m_pd3dCommandList);

   CAirplanePlayer *pAirplanePlayer = new CAirplanePlayer(m_pd3dDevice,
m_pd3dCommandList, m_pScene->GetGraphicsRootSignature());
   m_pScene->m_pPlayer = m_pPlayer = pAirplanePlayer;
   m_pCamera = m_pPlayer->GetCamera();

   m_pd3dCommandList->Close();
   ID3D12CommandList *ppd3dCommandLists[] = { m_pd3dCommandList };
   m_pd3dCommandQueue->ExecuteCommandLists(1, ppd3dCommandLists);

   WaitForGpuComplete();

   if (m_pScene) m_pScene->ReleaseUploadBuffers();
   if (m_pPlayer) m_pPlayer->ReleaseUploadBuffers();

   m_GameTimer.Reset();
}
```

⑭ “Shaders.hlsl” 파일 변경하기

❶ 상수 버퍼 선언 부분을 다음과 같이 수정한다.

```
//플레이어 객체의 데이터를 위한 상수 버퍼
cbuffer cbPlayerInfo : register(b0)
{
   matrix      gmtxPlayerWorld : packoffset(c0);
```

```
};

//카메라 객체의 데이터를 위한 상수 버퍼(스펙큘러 조명 계산을 위하여 카메라의 위치 벡터를 추가)
cbuffer cbCameraInfo : register(b1)
{
   matrix      gmtxView : packoffset(c0);
   matrix      gmtxProjection : packoffset(c4);
   float3      gvCameraPosition : packoffset(c8);
};

//게임 객체의 데이터를 위한 상수 버퍼(게임 객체에 대한 재질 번호를 추가)
cbuffer cbGameObjectInfo : register(b2)
{
   matrix      gmtxGameObject : packoffset(c0);
   uint        gnMaterial : packoffset(c4);
};
```

❷ 상수 버퍼 선언 부분 다음에 조명 계산 쉐이더 함수를 포함하고 있는 파일을 포함하기 위하여 다음을 추가한다.

```
#include "Light.hlsl"
```

❸ 플레이어 객체를 렌더링하기 위한 정점 쉐이더와 픽셀 쉐이더 함수를 다음과 같이 추가한다.

```
VS_DIFFUSED_OUTPUT VSPlayer(VS_DIFFUSED_INPUT input)
{
   VS_DIFFUSED_OUTPUT output;

   output.position = mul(mul(mul(float4(input.position, 1.0f), gmtxPlayerWorld),
gmtxView), gmtxProjection);
   output.color = input.color;

   return(output);
}

float4 PSPlayer(VS_DIFFUSED_OUTPUT input) : SV_TARGET
{
   return(input.color);
}
```

❹ 조명을 사용하여 객체를 렌더링하기 위한 정점 쉐이더와 픽셀 쉐이더 함수를 다음과 같이 추가한다.

```
//정점 조명을 사용
#define _WITH_VERTEX_LIGHTING

//정점 쉐이더의 입력 정점 구조
struct VS_LIGHTING_INPUT
{
   float3 position : POSITION;
   float3 normal : NORMAL;
};

//정점 쉐이더의 출력 정점 구조
```

```
struct VS_LIGHTING_OUTPUT
{
   float4 position : SV_POSITION;
   float3 positionW : POSITION;

#ifdef _WITH_VERTEX_LIGHTING
   float4 color : COLOR;
#else
   float3 normalW : NORMAL;
#endif
};

//정점 쉐이더 함수
VS_LIGHTING_OUTPUT VSLighting(VS_LIGHTING_INPUT input)
{
   VS_LIGHTING_OUTPUT output;

   output.positionW = (float3)mul(float4(input.position, 1.0f), gmtxGameObject);
   output.position = mul(mul(float4(output.positionW, 1.0f), gmtxView), gmtxProjection);
   float3 normalW = mul(input.normal, (float3x3)gmtxGameObject);
#ifdef _WITH_VERTEX_LIGHTING
   output.color = Lighting(output.positionW, normalize(normalW));
#else
   output.normalW = normalW;
#endif
   return(output);
}

//픽셀 쉐이더 함수
float4 PSLighting(VS_LIGHTING_OUTPUT input) : SV_TARGET
{
#ifdef _WITH_VERTEX_LIGHTING
   return(input.color);
#else
   float3 normalW = normalize(input.normalW);
   float4 color = Lighting(input.positionW, normalW);
   return(color);
#endif
}
```

⑮ “Light.hlsl” 파일 생성하기

❶ “Light.hlsl” 파일 생성하고 다음을 추가한다.

```
#define MAX_LIGHTS             8
#define MAX_MATERIALS          8

#define POINT_LIGHT            1
#define SPOT_LIGHT             2
#define DIRECTIONAL_LIGHT      3

#define _WITH_LOCAL_VIEWER_HIGHLIGHTING
#define _WITH_THETA_PHI_CONES
//#define _WITH_REFLECT
```

❷ 재질과 조명을 위한 다음의 구조체를 선언한다.

```
struct MATERIAL
{
   float4              m_cAmbient;
   float4              m_cDiffuse;
   float4              m_cSpecular; //a = power
   float4              m_cEmissive;
};

struct LIGHT
{
   float4              m_cAmbient;
   float4              m_cDiffuse;
   float4              m_cSpecular;
   float3              m_vPosition;
   float               m_fFalloff;
   float3              m_vDirection;
   float               m_fTheta; //cos(m_fTheta)
   float3              m_vAttenuation;
   float               m_fPhi; //cos(m_fPhi)
   bool                m_bEnable;
   int                 m_nType;
   float               m_fRange;
   float               padding;
};
```

❸ 재질과 조명을 위한 상수 버퍼를 다음과 같이 선언한다.

```
cbuffer cbMaterial : register(b3)
{
   MATERIAL       gMaterials[MAX_MATERIALS];
};

cbuffer cbLights : register(b4)
{
   LIGHT          gLights[MAX_LIGHTS];
   float4         gcGlobalAmbientLight;
};
```

❹ 방향성 조명 효과 계산을 위한 DirectionalLight() 함수를 다음과 같이 정의한다.

```
float4 DirectionalLight(int nIndex, float3 vNormal, float3 vToCamera)
{
   float3 vToLight = -gLights[nIndex].m_vDirection;
   float fDiffuseFactor = dot(vToLight, vNormal);
   float fSpecularFactor = 0.0f;
   if (fDiffuseFactor > 0.0f)
   {
      if (gMaterials[gnMaterial].m_cSpecular.a != 0.0f)
      {
```

```
#ifdef _WITH_REFLECT
         float3 vReflect = reflect(-vToLight, vNormal);
         fSpecularFactor = pow(max(dot(vReflect, vToCamera), 0.0f),
gMaterials[gnMaterial].m_cSpecular.a);
#else
#ifdef _WITH_LOCAL_VIEWER_HIGHLIGHTING
         float3 vHalf = normalize(vToCamera + vToLight);
#else
         float3 vHalf = float3(0.0f, 1.0f, 0.0f);
#endif
         fSpecularFactor = pow(max(dot(vHalf, vNormal), 0.0f),
gMaterials[gnMaterial].m_cSpecular.a);
#endif
      }
   }

   return((gLights[nIndex].m_cAmbient * gMaterials[gnMaterial].m_cAmbient) +
(gLights[nIndex].m_cDiffuse * fDiffuseFactor * gMaterials[gnMaterial].m_cDiffuse) +
(gLights[nIndex].m_cSpecular * fSpecularFactor * gMaterials[gnMaterial].m_cSpecular));
}
```

❺ 점 조명 효과 계산을 위한 PointLight() 함수를 다음과 같이 정의한다.

```
float4 PointLight(int nIndex, float3 vPosition, float3 vNormal, float3 vToCamera)
{
   float3 vToLight = gLights[nIndex].m_vPosition - vPosition;
   float fDistance = length(vToLight);
   if (fDistance <= gLights[nIndex].m_fRange)
   {
      float fSpecularFactor = 0.0f;
      vToLight /= fDistance;
      float fDiffuseFactor = dot(vToLight, vNormal);
      if (fDiffuseFactor > 0.0f)
      {
         if (gMaterials[gnMaterial].m_cSpecular.a != 0.0f)
         {
#ifdef _WITH_REFLECT
            float3 vReflect = reflect(-vToLight, vNormal);
            fSpecularFactor = pow(max(dot(vReflect, vToCamera), 0.0f),
gMaterials[gnMaterial].m_cSpecular.a);
#else
#ifdef _WITH_LOCAL_VIEWER_HIGHLIGHTING
            float3 vHalf = normalize(vToCamera + vToLight);
#else
            float3 vHalf = float3(0.0f, 1.0f, 0.0f);
#endif
            fSpecularFactor = pow(max(dot(vHalf, vNormal), 0.0f),
gMaterials[gnMaterial].m_cSpecular.a);
#endif
         }
      }
      float fAttenuationFactor = 1.0f / dot(gLights[nIndex].m_vAttenuation, float3(1.0f,
fDistance, fDistance*fDistance));
```

```
      return(((gLights[nIndex].m_cAmbient * gMaterials[gnMaterial].m_cAmbient) +
(gLights[nIndex].m_cDiffuse * fDiffuseFactor * gMaterials[gnMaterial].m_cDiffuse) +
(gLights[nIndex].m_cSpecular * fSpecularFactor * gMaterials[gnMaterial].m_cSpecular)) *
fAttenuationFactor);
   }
   return(float4(0.0f, 0.0f, 0.0f, 0.0f));
}
```

❻ 스팟 조명 효과 계산을 위한 SpotLight() 함수를 다음과 같이 정의한다.

```
float4 SpotLight(int nIndex, float3 vPosition, float3 vNormal, float3 vToCamera)
{
   float3 vToLight = gLights[nIndex].m_vPosition - vPosition;
   float fDistance = length(vToLight);
   if (fDistance <= gLights[nIndex].m_fRange)
   {
      float fSpecularFactor = 0.0f;
      vToLight /= fDistance;
      float fDiffuseFactor = dot(vToLight, vNormal);
      if (fDiffuseFactor > 0.0f)
      {
         if (gMaterials[gnMaterial].m_cSpecular.a != 0.0f)
         {
#ifdef _WITH_REFLECT
            float3 vReflect = reflect(-vToLight, vNormal);
            fSpecularFactor = pow(max(dot(vReflect, vToCamera), 0.0f),
gMaterials[gnMaterial].m_cSpecular.a);
#else
#ifdef _WITH_LOCAL_VIEWER_HIGHLIGHTING
            float3 vHalf = normalize(vToCamera + vToLight);
#else
            float3 vHalf = float3(0.0f, 1.0f, 0.0f);
#endif
            fSpecularFactor = pow(max(dot(vHalf, vNormal), 0.0f),
gMaterials[gnMaterial].m_cSpecular.a);
#endif
         }
      }
#ifdef _WITH_THETA_PHI_CONES
      float fAlpha = max(dot(-vToLight, gLights[nIndex].m_vDirection), 0.0f);
      float fSpotFactor = pow(max(((fAlpha - gLights[nIndex].m_fPhi) /
(gLights[nIndex].m_fTheta - gLights[nIndex].m_fPhi)), 0.0f), gLights[nIndex].m_fFalloff);
#else
      float fSpotFactor = pow(max(dot(-vToLight, gLights[i].m_vDirection), 0.0f),
gLights[i].m_fFalloff);
#endif
      float fAttenuationFactor = 1.0f / dot(gLights[nIndex].m_vAttenuation, float3(1.0f,
fDistance, fDistance*fDistance));

      return(((gLights[nIndex].m_cAmbient * gMaterials[gnMaterial].m_cAmbient) +
(gLights[nIndex].m_cDiffuse * fDiffuseFactor * gMaterials[gnMaterial].m_cDiffuse) +
(gLights[nIndex].m_cSpecular * fSpecularFactor * gMaterials[gnMaterial].m_cSpecular)) *
fAttenuationFactor * fSpotFactor);
   }
```

```
   return(float4(0.0f, 0.0f, 0.0f, 0.0f));
}
```

❼ 한 점에 대한 조명 효과 계산을 위한 Lighting() 함수를 다음과 같이 정의한다.

```
float4 Lighting(float3 vPosition, float3 vNormal)
{
   float3 vCameraPosition = float3(gvCameraPosition.x, gvCameraPosition.y,
gvCameraPosition.z);
   float3 vToCamera = normalize(vCameraPosition - vPosition);

   float4 cColor = float4(0.0f, 0.0f, 0.0f, 0.0f);
   for (int i = 0; i < MAX_LIGHTS; i++)
   {
      if (gLights[i].m_bEnable)
      {
         if (gLights[i].m_nType == DIRECTIONAL_LIGHT)
         {
            cColor += DirectionalLight(i, vNormal, vToCamera);
         }
         else if (gLights[i].m_nType == POINT_LIGHT)
         {
            cColor += PointLight(i, vPosition, vNormal, vToCamera);
         }
         else if (gLights[i].m_nType == SPOT_LIGHT)
         {
            cColor += SpotLight(i, vPosition, vNormal, vToCamera);
         }
      }
   }
   cColor += (gcGlobalAmbientLight * gMaterials[gnMaterial].m_cAmbient);
   cColor.a = gMaterials[gnMaterial].m_cDiffuse.a;

   return(cColor);
}
```