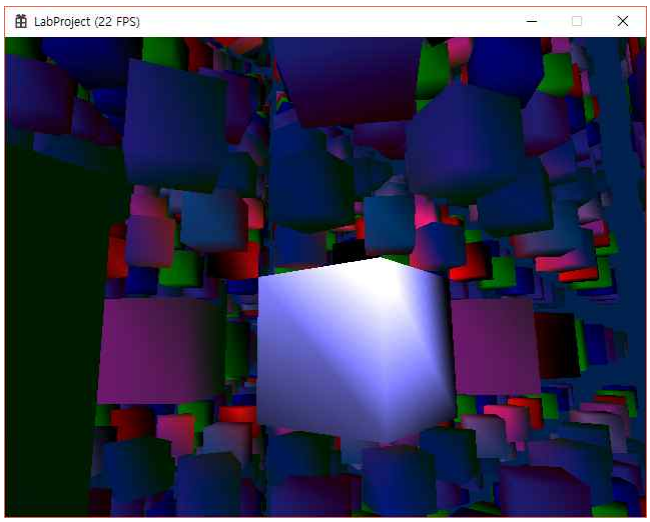
□ 예제 프로그램 19: LabProject18(서술자 테이블: Descriptor Table)

예제 프로그램 LabProject17을 기반으로 여러 개의 직육면체들을 조명 처리를 하여 렌더링할 수 있도록 구현한다. LabProject17는 루트 서술자를 사용하지만 LabProject18은 서술자 테이블을 사용한다.



① 새로운 프로젝트의 생성

먼저 새로운 프로젝트 LabProject18을 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 “LabProject18”을 입력하고 『확인』을 선택한다.

❶ 파일 탐색기에서 프로젝트 “LabProject17” 폴더의 다음 파일을 선택하여 프로젝트 “LabProject18” 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 “LabProject18”에 추가한다. 오른쪽 마우스 버튼으로 『LabProject18』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 “LabProject18”에 추가된다.

② LabProject18.cpp 파일 수정하기

이제 “LabProject18.cpp" 파일의 내용을 “LabProject17.cpp" 파일의 내용으로 바꾸도록 하자. “LabProject17.cpp" 파일의 내용 전체를 “LabProject18.cpp" 파일로 복사한다. 이제 “LabProject17.cpp" 파일에서 “LabProject17”을 “LabProject18”으로 모두 바꾼다.
그리고 “LABPROJECT17”을 “LABPROJECT18”로 모두 바꾼다.

③ “Scene.h” 파일 수정하기

❶ “CScene” 클래스에 다음을 추가한다.

```
protected:
   ID3D12DescriptorHeap             *m_pd3dCbvSrvDescriptorHeap = NULL;

   D3D12_CPU_DESCRIPTOR_HANDLE      m_d3dObjectsCbvCPUDescriptorHandle;
   D3D12_GPU_DESCRIPTOR_HANDLE      m_d3dObjectsCbvGPUDescriptorHandle;
   D3D12_CPU_DESCRIPTOR_HANDLE      m_d3dLightsCbvCPUDescriptorHandle;
   D3D12_GPU_DESCRIPTOR_HANDLE      m_d3dLightsCbvGPUDescriptorHandle;
   D3D12_CPU_DESCRIPTOR_HANDLE      m_d3dMaterialsCbvCPUDescriptorHandle;
   D3D12_GPU_DESCRIPTOR_HANDLE      m_d3dMaterialsCbvGPUDescriptorHandle;
```

④ “Scene.cpp” 파일 수정하기

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

```
ID3D12RootSignature *CScene::CreateGraphicsRootSignature(ID3D12Device *pd3dDevice)
{
```

```
   ID3D12RootSignature *pd3dGraphicsRootSignature = NULL;

   D3D12_DESCRIPTOR_RANGE pd3dDescriptorRanges[2];

   pd3dDescriptorRanges[0].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_CBV;
   pd3dDescriptorRanges[0].RegisterSpace = 0;
   pd3dDescriptorRanges[0].OffsetInDescriptorsFromTableStart = 0;
   pd3dDescriptorRanges[0].BaseShaderRegister = 2; //Game Objects
   pd3dDescriptorRanges[0].NumDescriptors = 1;

   pd3dDescriptorRanges[1].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_CBV;
   pd3dDescriptorRanges[1].RegisterSpace = 0;
   pd3dDescriptorRanges[1].OffsetInDescriptorsFromTableStart = 0;
   pd3dDescriptorRanges[1].BaseShaderRegister = 3; //Materials, Lights
   pd3dDescriptorRanges[1].NumDescriptors = 2;

   D3D12_ROOT_PARAMETER pd3dRootParameters[4];

   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_DESCRIPTOR_TABLE;
   pd3dRootParameters[2].DescriptorTable.NumDescriptorRanges = 1;
   pd3dRootParameters[2].DescriptorTable.pDescriptorRanges = &pd3dDescriptorRanges[0];
   pd3dRootParameters[2].ShaderVisibility = D3D12_SHADER_VISIBILITY_ALL;

   pd3dRootParameters[3].ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE;
   pd3dRootParameters[3].DescriptorTable.NumDescriptorRanges = 1;
   pd3dRootParameters[3].DescriptorTable.pDescriptorRanges = &pd3dDescriptorRanges[1];
   pd3dRootParameters[3].ShaderVisibility = D3D12_SHADER_VISIBILITY_ALL;

   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_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);
}
```

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

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

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

   D3D12_DESCRIPTOR_HEAP_DESC d3dDescriptorHeapDesc;
   d3dDescriptorHeapDesc.NumDescriptors = 21 * 21 * 21 + 3;
   d3dDescriptorHeapDesc.Type = D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV;
   d3dDescriptorHeapDesc.Flags = D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE;
   d3dDescriptorHeapDesc.NodeMask = 0;
   pd3dDevice->CreateDescriptorHeap(&d3dDescriptorHeapDesc,
__uuidof(ID3D12DescriptorHeap), (void **)&m_pd3dCbvSrvDescriptorHeap);

   D3D12_CPU_DESCRIPTOR_HANDLE d3dCbvCPUDescriptorStartHandle =
m_pd3dCbvSrvDescriptorHeap->GetCPUDescriptorHandleForHeapStart();
   D3D12_GPU_DESCRIPTOR_HANDLE d3dCbvGPUDescriptorStartHandle =
m_pd3dCbvSrvDescriptorHeap->GetGPUDescriptorHandleForHeapStart();
   m_d3dMaterialsCbvCPUDescriptorHandle = d3dCbvCPUDescriptorStartHandle;
   m_d3dMaterialsCbvGPUDescriptorHandle = d3dCbvGPUDescriptorStartHandle;
   m_d3dLightsCbvCPUDescriptorHandle.ptr = d3dCbvCPUDescriptorStartHandle.ptr +
::gnCbvSrvDescriptorIncrementSize;
   m_d3dLightsCbvGPUDescriptorHandle.ptr = d3dCbvGPUDescriptorStartHandle.ptr +
::gnCbvSrvDescriptorIncrementSize;
   m_d3dObjectsCbvCPUDescriptorHandle.ptr = d3dCbvCPUDescriptorStartHandle.ptr +
::gnCbvSrvDescriptorIncrementSize;
   m_d3dObjectsCbvGPUDescriptorHandle.ptr = d3dCbvGPUDescriptorStartHandle.ptr +
::gnCbvSrvDescriptorIncrementSize;

   m_pShaders[0].BuildObjects(pd3dDevice, pd3dCommandList,
m_d3dObjectsCbvCPUDescriptorHandle, m_d3dObjectsCbvGPUDescriptorHandle);

   BuildLightsAndMaterials();

   CreateShaderVariables(pd3dDevice, pd3dCommandList);
}
```

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

```
   if (m_pd3dCbvSrvDescriptorHeap) m_pd3dCbvSrvDescriptorHeap->Release();
```

❹❺ “CScene” 클래스의 CreateShaderVariables() 함수를 다음과 같이 수정한다.

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

   D3D12_CONSTANT_BUFFER_VIEW_DESC d3dcbvDesc;
   d3dcbvDesc.BufferLocation = m_pd3dcbLights->GetGPUVirtualAddress();
   d3dcbvDesc.SizeInBytes = ncbLightsBytes;
   pd3dDevice->CreateConstantBufferView(&d3dcbvDesc, m_d3dLightsCbvCPUDescriptorHandle);

   d3dcbvDesc.BufferLocation = m_pd3dcbMaterials->GetGPUVirtualAddress();
   d3dcbvDesc.SizeInBytes = ncbMaterialBytes;
   pd3dDevice->CreateConstantBufferView(&d3dcbvDesc,
m_d3dMaterialsCbvCPUDescriptorHandle);
}
```

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

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

   pd3dCommandList->SetDescriptorHeaps(1, &m_pd3dCbvSrvDescriptorHeap);

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

   UpdateShaderVariables(pd3dCommandList);

   pd3dCommandList->SetGraphicsRootDescriptorTable(3,
m_d3dMaterialsCbvGPUDescriptorHandle);

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

```
}
```

⑤ “GameObject.h” 파일 수정하기

❶ “CGameObject” 클래스에 다음을 추가한다.

```
public:
   D3D12_GPU_DESCRIPTOR_HANDLE       m_d3dCbvGPUDescriptorHandle;
```

❷ “CGameObject” 클래스의 SetCbvGPUDescriptorHandle()과 GetCbvGPUDescriptorHandle() 멤버 함수를 다음과 같이 선언한다.

```
   void SetCbvGPUDescriptorHandle(UINT64 nCbvGPUDescriptorHandlePtr) {
m_d3dCbvGPUDescriptorHandle.ptr = nCbvGPUDescriptorHandlePtr; }

   D3D12_GPU_DESCRIPTOR_HANDLE GetCbvGPUDescriptorHandle() {
return(m_d3dCbvGPUDescriptorHandle); }
```

⑥ “GameObject.cpp” 파일 수정하기

❶ “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);
      }
   }

   pd3dCommandList->SetGraphicsRootDescriptorTable(2, m_d3dCbvGPUDescriptorHandle);

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

⑦ “Shader.h” 파일 변경하기

❶ “CObjectsShader” 클래스에 다음 멤버 변수를 선언한다.

```
protected:
   ID3D12DescriptorHeap             *m_pd3dCbvSrvDescriptorHeap = NULL;
```

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

```
   virtual void BuildObjects(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, D3D12_CPU_DESCRIPTOR_HANDLE d3dCbvCPUDescriptorStartHandle,
D3D12_GPU_DESCRIPTOR_HANDLE d3dCbvGPUDescriptorStartHandle);
```

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

```
   virtual void CreateShaderVariables(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, D3D12_CPU_DESCRIPTOR_HANDLE d3dCbvCPUDescriptorStartHandle,
D3D12_GPU_DESCRIPTOR_HANDLE d3dCbvGPUDescriptorStartHandle);
```

⑧ “Shader.cpp” 파일 변경하기

❶ “CObjectsShader” 클래스의 CreateShaderVariables() 함수를 다음과 같이 수정한다.

```
void CObjectsShader::CreateShaderVariables(ID3D12Device *pd3dDevice,
ID3D12GraphicsCommandList *pd3dCommandList, D3D12_CPU_DESCRIPTOR_HANDLE
d3dCbvCPUDescriptorStartHandle, D3D12_GPU_DESCRIPTOR_HANDLE
d3dCbvGPUDescriptorStartHandle)
{
   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);

   D3D12_GPU_VIRTUAL_ADDRESS d3dGpuVirtualAddress =
m_pd3dcbGameObjects->GetGPUVirtualAddress();

   D3D12_CONSTANT_BUFFER_VIEW_DESC d3dcbvDesc;
   d3dcbvDesc.SizeInBytes = ncbGameObjectBytes;
   for (int j = 0; j < m_nObjects; j++)
   {
      d3dcbvDesc.BufferLocation = d3dGpuVirtualAddress + (ncbGameObjectBytes * j);
      D3D12_CPU_DESCRIPTOR_HANDLE d3dCbvCPUDescriptorHandle;
      d3dCbvCPUDescriptorHandle.ptr = d3dCbvCPUDescriptorStartHandle.ptr +
(::gnCbvSrvDescriptorIncrementSize * j);
      pd3dDevice->CreateConstantBufferView(&d3dcbvDesc, d3dCbvCPUDescriptorHandle);
   }
}
```

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

```
void CObjectsShader::BuildObjects(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, D3D12_CPU_DESCRIPTOR_HANDLE d3dCbvCPUDescriptorStartHandle,
D3D12_GPU_DESCRIPTOR_HANDLE d3dCbvGPUDescriptorStartHandle)
{
```

```
   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];

   CreateShaderVariables(pd3dDevice, pd3dCommandList, d3dCbvCPUDescriptorStartHandle,
d3dCbvGPUDescriptorStartHandle);

   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));

pRotatingObject->SetCbvGPUDescriptorHandle(d3dCbvGPUDescriptorStartHandle.ptr +
(::gnCbvSrvDescriptorIncrementSize * i));
            m_ppObjects[i++] = pRotatingObject;
         }
      }
   }
}
```