

Game Programming with DirectX

3D Geometry File Format

FBX

- **FBX File**

- Autodesk FBX
- 응용 프로그램에서 3D 데이터의 호환(Import/Export)을 위한 파일 형식
- FBX 파일(.fbx): Binary 또는 ASCII 형식으로 저장
- 3D 장면(Scene)의 카메라, 조명, 메쉬, NURB, 그리고 다른 요소들을 저장
- Autodesk 3ds Max, Maya, MotionBuilder: FBX 파일을 지원

- **FBX SDK(Software Development Kit)**

- FBX 파일 형식에 대한 문서를 제공하지 않음
- FBX를 다루는 모든 응용 프로그램은 FBX SDK를 사용해야 함
- <http://www.autodesk.com/fbx> (FBX® 2020.2.1 SDK)
- FBX Plug-ins (Autodesk 3ds Max, Autodesk Maya): FBX SDK로 작성
- 호환성(Interoperability)
Content Developers
Designers
- Autodesk 제품들은 FBX를 지원
Autodesk 3ds Max, Maya, MotionBuilder, Softimage, Mudbox, ...
- FBX 파일 버전 7.3, 7.2, 7.1, 7.0, 6.1, 6.0을 임포트(Import)할 수 있음
- FBX 파일 버전 7.3, 7.2, 7.1, 7.0, 6.1을 익스포트(Export)할 수 있음
- Collada DAE (.dae)
- Alias OBJ (.obj)

- Sample FBX File

```

; FBX 7.1.0 project file
; Copyright (C) 1997-2010 Autodesk Inc. and/or its licensors.
; All rights reserved.
;

FBXHeaderExtension: {
; header information: global file information.
    FBXHeaderVersion: 1003
    FBXVersion: 7100
    CreationTimeStamp: {
        Version: 1000
        Year: 2010
        Month: 1
        Day: 19
        Hour: 16
        Minute: 30
        Second: 28
        Millisecond: 884
    }
    Creator: "FBX SDK/FBX Plugins version 2011.2"
    SceneInfo: "SceneInfo::GlobalInfo", "UserData" {
        ...
    }
    GlobalSettings: {
        Version: 1000
        Properties70: {
            P: "UpAxis", "int", "Integer", "", 1
            P: "UpAxisSign", "int", "Integer", "", 1
            P: "FrontAxis", "int", "Integer", "", 2
            P: "FrontAxisSign", "int", "Integer", "", 1
            ...
        }
    }
}

```

```

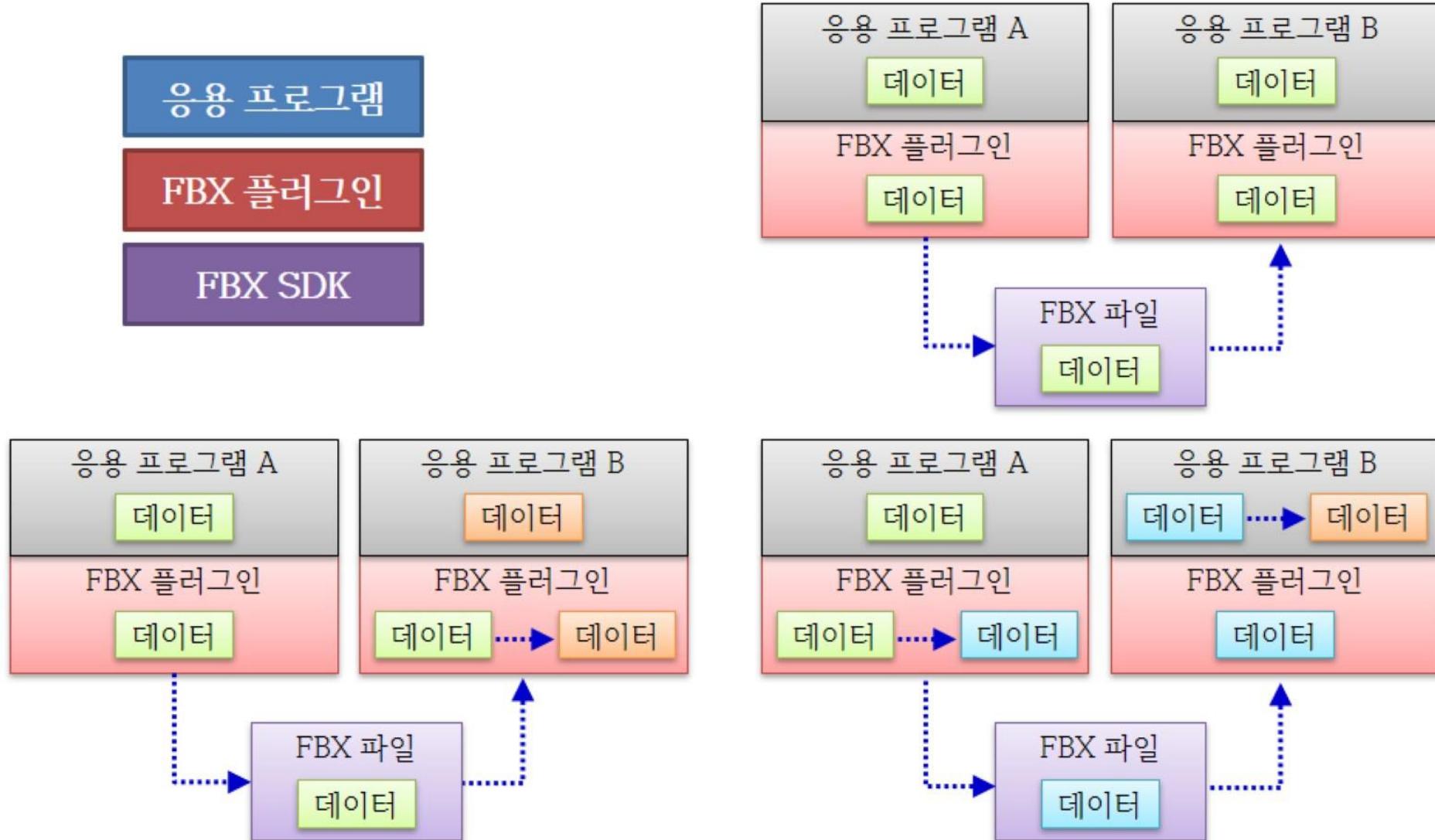
; Object definitions
;-----
Definitions: {
    Version: 100
    Count: 2251
    ObjectType: "GlobalSettings" {
        Count: 1
    }
    ObjectType: "Model" {
        Count: 86
        PropertyTemplate: "FbxNode" {
            Properties70: {
                P: "QuaternionInterpolate", "bool", "", "", 0
                P: "RotationOffset", "Vector3D", "Vector", "", 0, 0, 0
                P: "RotationPivot", "Vector3D", "Vector", "", 0, 0, 0
                P: "ScalingOffset", "Vector3D", "Vector", "", 0, 0, 0
                P: "ScalingPivot", "Vector3D", "Vector", "", 0, 0, 0
            ...
        }
        ObjectType: "Material" {
            Count: 1
            PropertyTemplate: "FbxSurfacePhong" {
                Properties70: {
                    P: "ShadingModel", "KString", "", "", "Phong"
                    P: "MultiLayer", "bool", "", "", 0
                    P: "EmissiveColor", "ColorRGB", "Color", "", 0, 0, 0
                    P: "EmissiveFactor", "double", "Number", "", 1
                    P: "AmbientColor", "ColorRGB", "Color", "", 0.2, 0.2, 0.2
                ...
            }
            Model: 21883936, "Model::Humanoid:Hips", "LimbNode" {
                ...
            }
        }
    }
}

```

FBX

- **FBX 응용 프로그램**

- FBX SDK를 사용하는 이유: 호환성(Interoperability)

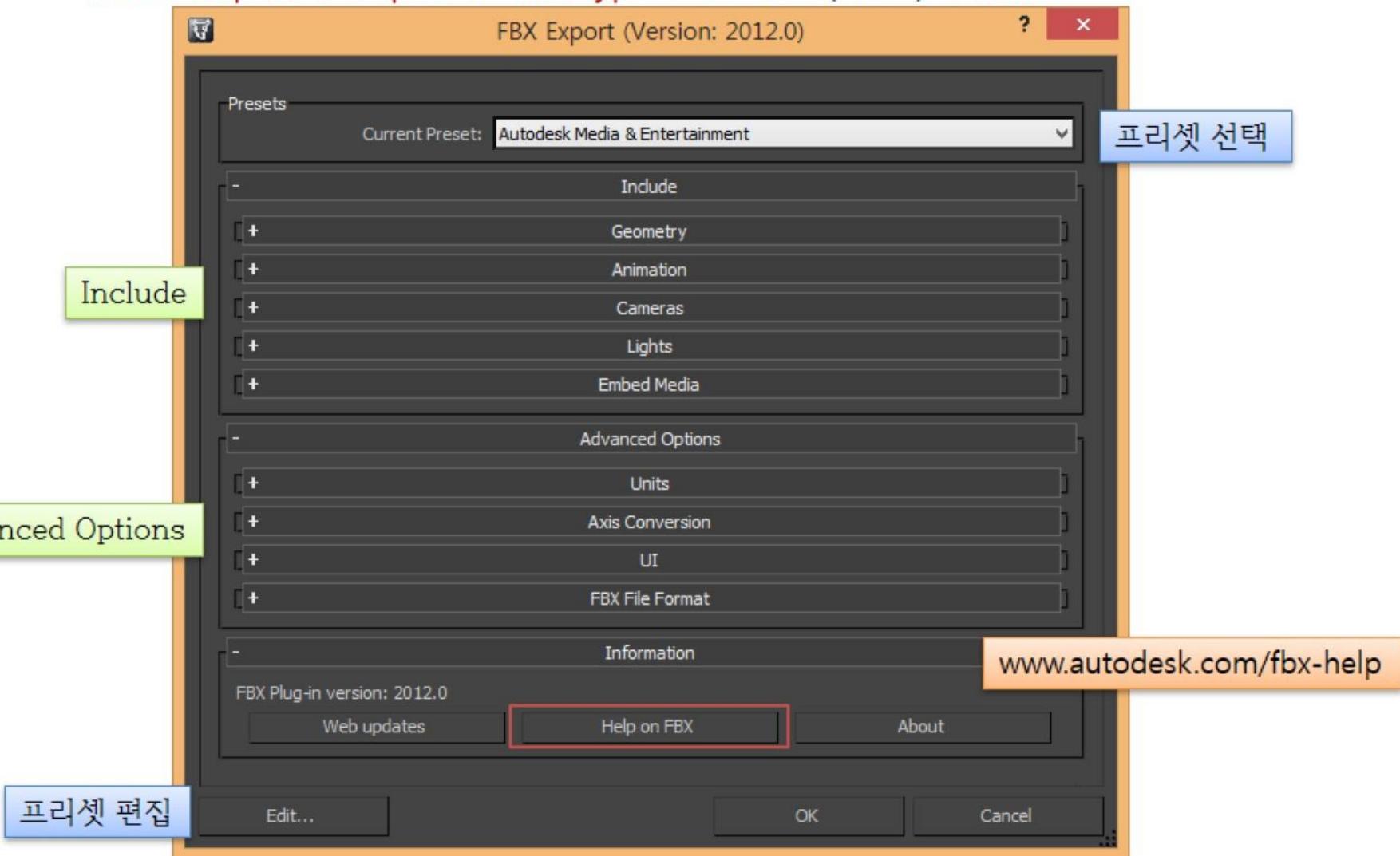


FBX File Export

- **FBX 플러그-인(Plug-in)**

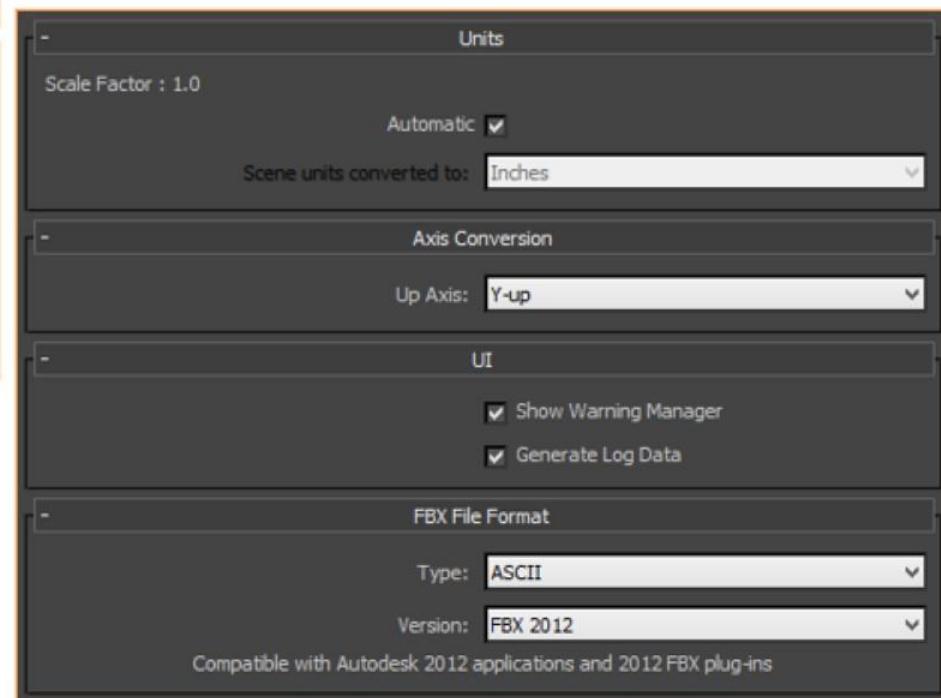
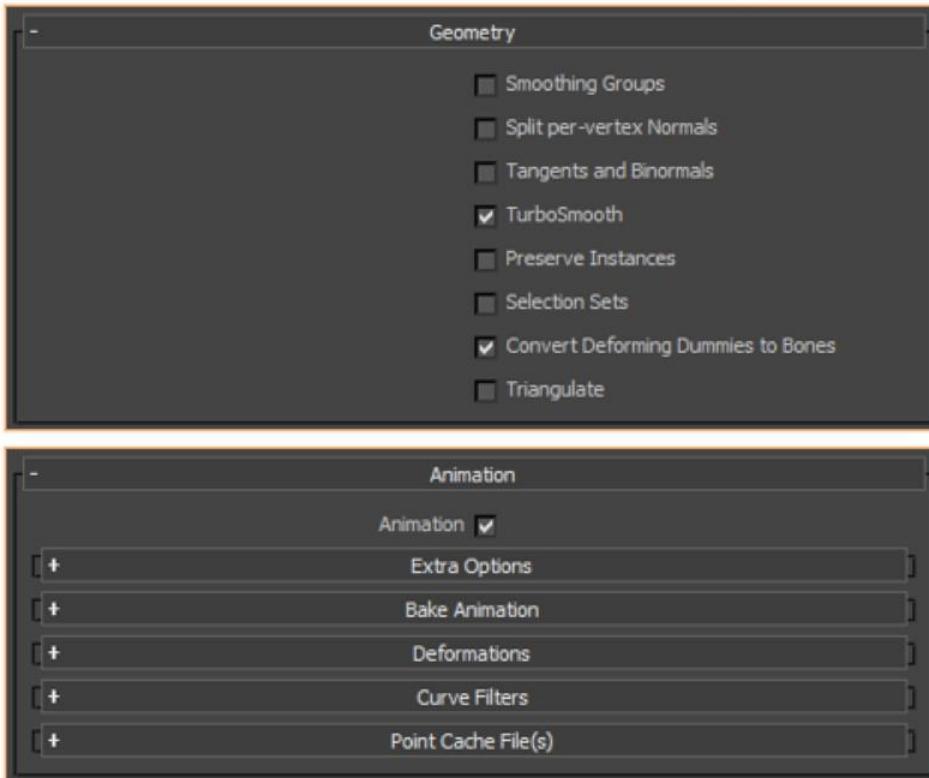
- 3ds Max FBX Export Dialog Box

File ▶ Export ▶ Export ▶ File Type: Autodesk (*.FBX) ▶ Save



FBX File Export

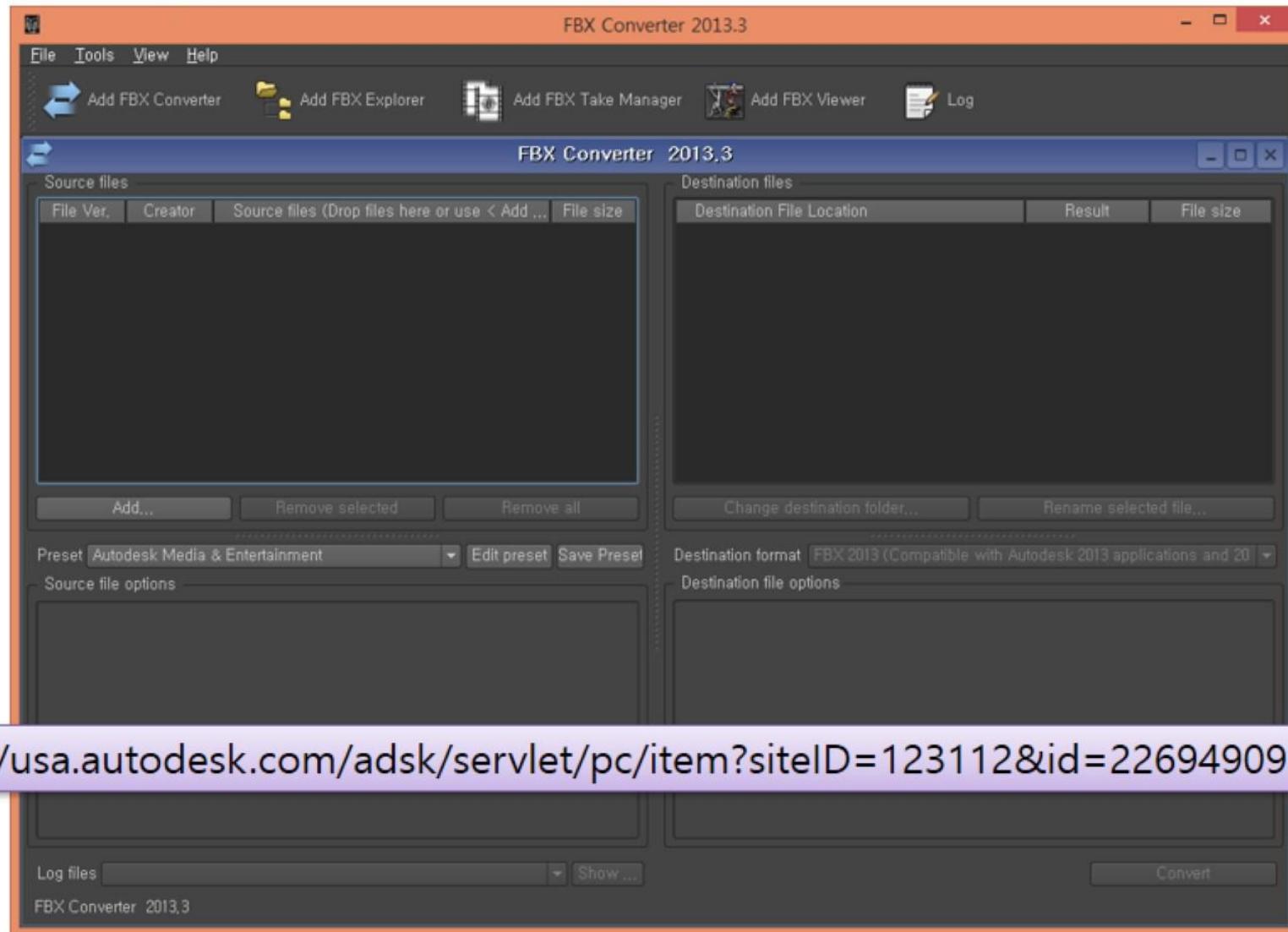
- **FBX 플러그-인(Plug-in)**
 - 3ds Max FBX Export Dialog Box



FBX File Export

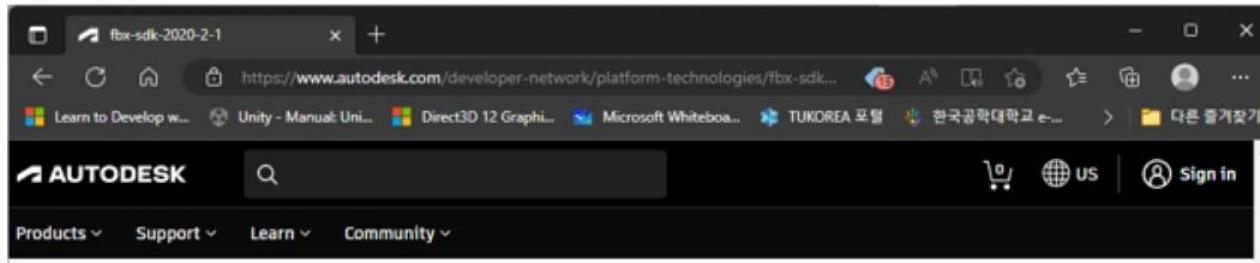
- **FBX 파일 형식 변환기(FBX Converter)**

- C:\Program Files\Autodesk\FBX\FBX Converter\2013.3\FBXConverterUI.exe



FBX SDK

- **FBX 다운로드**



https://www.autodesk.com/developer-network/platform-technologies/fbx-sdk-2020-2-1?us_oa=dotcom-us&us_si=223b7941-4870-4446-aea4-31a8f97c0023&us_st=fbx%20sdk



FBX Software Development Kit

The Autodesk® FBX® SDK is a free, easy-to-use, C++ software development platform and API toolkit that allows application and content vendors to transfer existing content into the FBX format with minimal effort.

- What's new in FBX SDK 2020.2.1
- About FBX SDK 2020.0
- About FBX SDK Extensions 2020.0

Windows

- [FBX SDK 2020.2.1 VS2015](#) (exe - 116.80MB)
- [FBX SDK 2020.2.1 VS2017](#) (exe - 157.60MB)
- [FBX SDK 2020.2.1 VS2019](#) (exe - 152.79MB)

Previous Versions

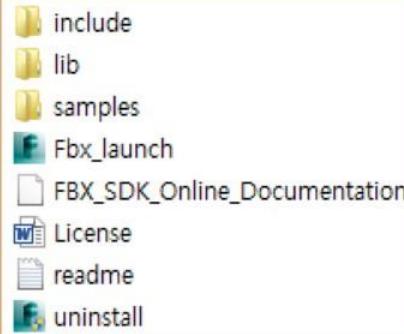
- [SDK Archive](#)
- [Plug-in and Converter Archives](#)

FBX SDK

• FBX SDK 설치

C:\Program Files\Autodesk\FBX\FBX SDK\2020.2.1

Visual Studio 2019



#include "fbxsdk.h"

프로젝트 속성 ▶ 구성 속성

VC++ 디렉터리 ▶ 포함 디렉터리

VC++ 디렉터리 ▶ 라이브러리 디렉터리

C/C++ ▶ 전처리기 ▶ 전처리기 정의

링커 ▶ 입력

MSVCRT/LIBCMT 라이브러리 링크 옵션

C/C++ ▶ 코드 생성 ▶ 런타임 라이브러리

FBX 라이브러리("추가 종속성"):

동적 링크: libfbxsdk.lib (libfbxsdk.dll)

정적 링크: libfbxsdk-md.lib

정적 링크(다중 쓰레드): libfbxsdk-mt.lib

정적 링크:

"특정 기본 라이브러리 무시": LIBMCT

"추가 종속성": wininet.lib

동적 링크: 전처리 정의(FBXSDK_SHARED) 추가

포함 디렉터리(Include Directory)

<FBX 설치 경로>/include

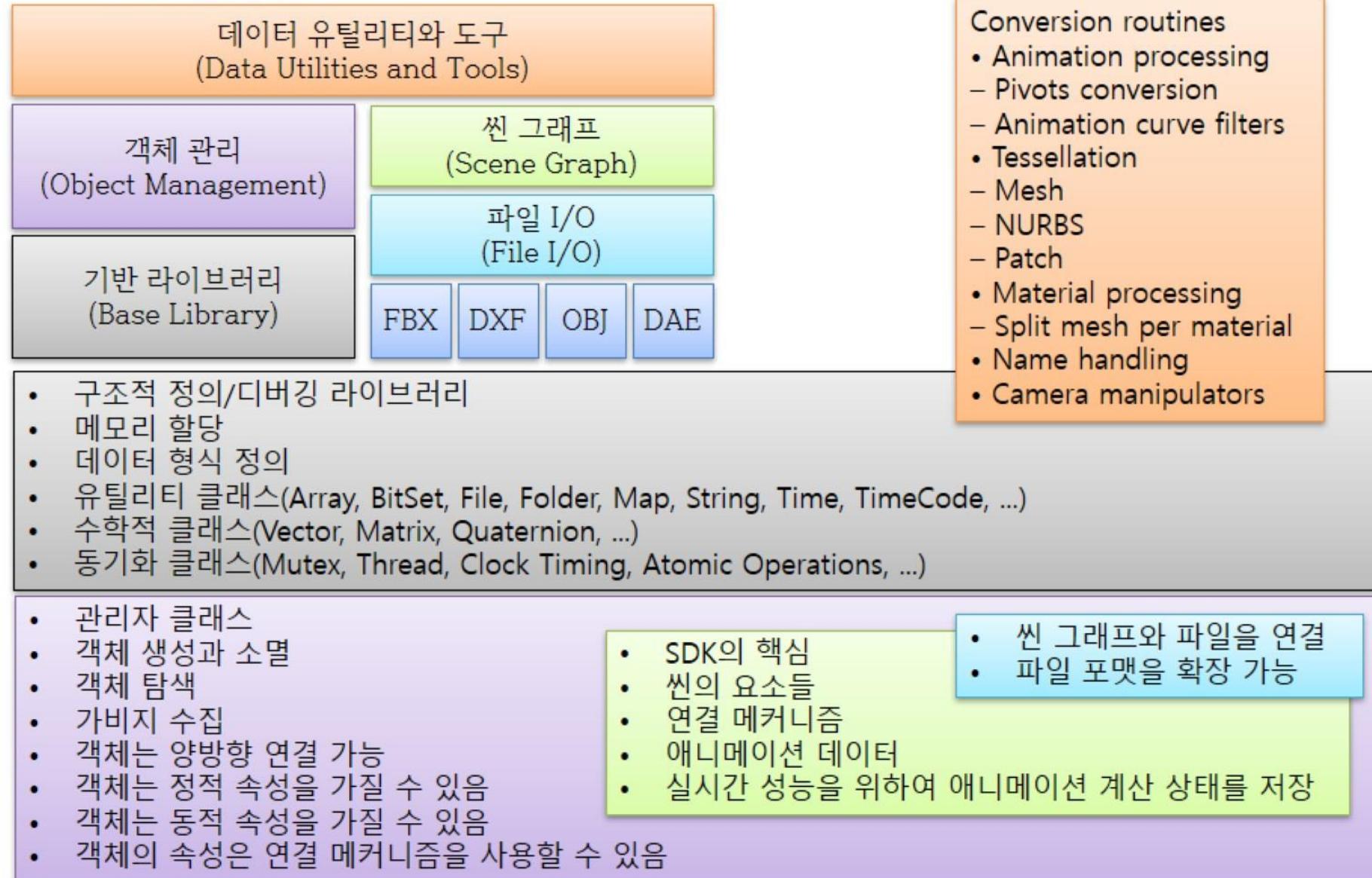
라이브러리 디렉터리(Library Directory)

<FBX 설치 경로>/lib/vs2015/x86/debug

라이브러리 옵션	설명
/MT	다중쓰레드, 정적 라이브러리
/MTd	다중쓰레드, 디버깅 정적 라이브러리
/MD	다중쓰레드, 동적 라이브러리
/MDd	다중쓰레드, 디버깅 동적 라이브러리

FBX SDK

• FBX SDK 구조



FBX SDK

- **FBX SDK가 지원하는 씬 요소**

- FBX SDK는 씬의 다음 요소들을 생성/수정할 수 있음

Scene	FbxScene
Mesh	FbxMesh
LOD(Level Of Detail) groups	FbxLodGroup
Cameras	FbxCamera
Lights, gobos	FbxLight, FbxGobo
NURBS	FbxNurbs, FbxNurbsCurve, FbxNurbsSurface, FbxTrimNurbsSurface
Texture mapping	FbxTexture
Material mapping	FbxSurfaceMaterial
Constraints	FbxConstraint
Vertex cache animation	FbxDeformer
Scene settings	FbxGlobalSettings, FbxAxisSystem
Transformation data	FbxNode
Markers	FbxMarker
Skeleton segments	FbxSkeleton
Animation curves	FbxAnimCurve
Rest, bind poses	FbxPose
Fbx	대부분의 FBX SDK 클래스 이름은 Fbx로 시작(예: FbxNode , FbxScene)
m	멤버 변수는 소문자 “m”으로 시작(예: FbxTakeInfo::mImportname)

FBX SDK

- FBX SDK 프로그램 작성

- ① **#include <fbxsdk.h>**
- ② FBX SDK 메모리 관리 객체(**FbxManager**) 를 생성
FbxManager::Create(), **FbxManager::Destroy()**
- ③ FBX 파일의 내용을 쓴 객체로 임포트(**FbxIOSettings**, **FbxImporter**, **FbxScene**)
- ④ 쓴 객체의 계층구조를 탐색(**FbxScene**, **FbxNode**, **FbxNodeAttribute**)
- ⑤ 쓴의 원소들의 정보를 액세스(**FbxNode**, **FbxNodeAttribute**, **FbxString**)

```
FbxManager *pfbxManager = FbxManager::Create();
FbxIOSettings *pfbxIOSettings = FbxIOSettings::Create(pfbxManager, IOSROOT);
pfbxManager->SetIOSettings(pfbxIOSettings);

FbxImporter *pfbxImporter = FbxImporter::Create(pfbxManager, "");
pfbxImporter->Initialize(pstrFilename, -1, pfbxManager->GetIOSettings());

FbxScene *pfbxScene = FbxScene::Create(pfbxManager, "SceneName");
pfbxImporter->Import(pfbxScene);
pfbxImporter->Destroy();

FbxNode *pfbxRootNode = pfbxScene->GetRootNode();
if (pfbxRootNode) {
    for (int i = 0; i < pfbxRootNode->GetChildCount(); i++) ProcessNode(pfbxRootNode->GetChild(i));
}

pfbxManager->Destroy();
```

FBX SDK

- FBX SDK 프로그램 작성

```
void ProcessNode(FbxNode *pfbxNode) {  
    PrintTabs();  
    const char *pstrnodeName = pfbxNode->GetName();  
    FbxDouble3 fbvxTranslation = pfbxNode->LclTranslation.Get();  
    FbxDouble3 fbvxRotation = pfbxNode->LclRotation.Get();  
    FbxDouble3 fbvxScaling = pfbxNode->LclScaling.Get();  
    printf("<Node name='%s' Translation='(%f, %f, %f)' Rotation='(%f, %f, %f)' Scaling='(%f, %f, %f)'>\n",  
        pstrnodeName, fbvxTranslation[0], fbvxTranslation[1], fbvxTranslation[2], fbvxRotation[0], fbvxRotation[1],  
        fbvxRotation[2], fbvxScaling[0], fbvxScaling[1], fbvxScaling[2]);  
    gnTabs++;
```

```
    for (int i = 0; i < pfbxNode->GetNodeAttributeCount(); i++) {  
        PrintAttribute(pfbxNode->GetNodeAttributeByIndex(i));  
    }
```

```
    for (int j = 0; j < pfbxNode->GetChildCount(); j++) ProcessNode(pfbxNode->GetChild(j));  
    gnTabs--;
```

```
    PrintTabs();
```

```
    printf("</Node>\n");  
}
```

```
int gnTabs = 0;
```

```
void PrintTabs() { for (int i = 0; i < gnTabs; i++) printf("\t"); }
```

```
void PrintAttribute(FbxNodeAttribute *pfbxAttribute) {  
    FbxString strType = GetAttributeTypeName(pfbxAttribute->GetAttributeType());  
    FbxString strName = pfbxAttribute->GetName();  
    PrintTabs();  
    printf("<Attribute type='%s' name='%s'>\n", strType.Buffer(), strName.Buffer());  
}
```

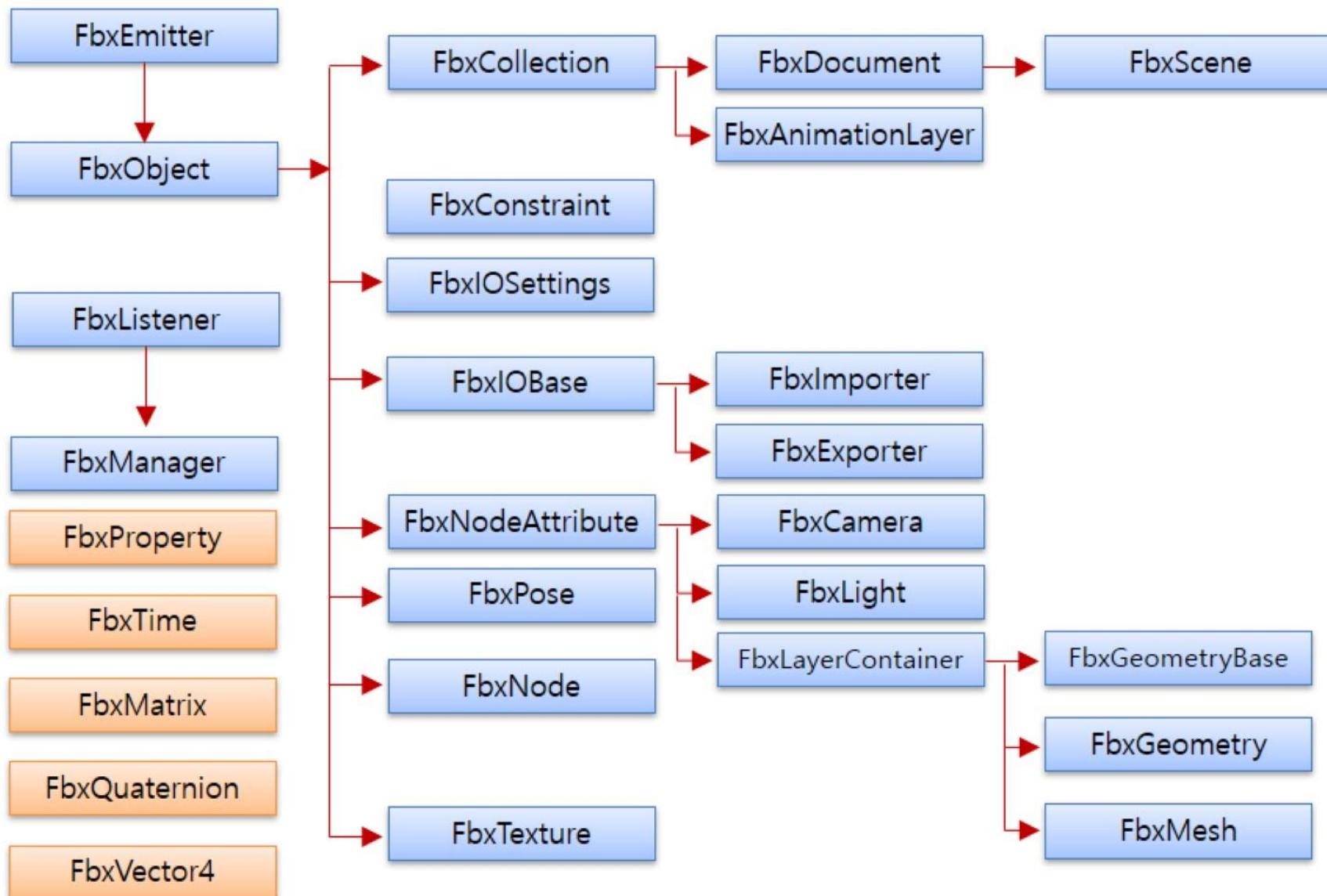
```
FbxString GetAttributeTypeName(FbxNodeAttribute::EType fbxType) {
```

```
    if (fbxType == FbxNodeAttribute::eMesh) return("Mesh");  
    if (fbxType == FbxNodeAttribute::eCamera) return("Camera");
```

```
    ...  
}
```

FBX SDK

- FBX 클래스 계층도



FBX SDK

- **FBX 자료형(Data Types)**

typedef bool	FbxBool;
typedef signed char	FbxChar;
typedef unsigned char	FbxUChar;
typedef signed short	FbxShort;
typedef unsigned short	FbxUShort;
typedef signed int	FbxInt;
typedef unsigned int	FbxUInt;
typedef float	FbxFloat;
typedef double	FbxDouble;
typedef FbxBool*	FbxBoolPtr;
typedef FbxChar*	FbxCharPtr;
typedef FbxUChar*	FbxUCharPtr;
typedef FbxShort*	FbxShortPtr;
typedef FbxUShort*	FbxUShortPtr;
typedef FbxInt*	FbxIntPtr;
typedef FbxUInt*	FbxUIntPtr;
typedef FbxFloat*	FbxFloatPtr;
typedef FbxDouble*	FbxDoublePtr;
typedef FbxInt	FbxEnum;
typedef FbxObject*	FbxReference;

#define FBXSDK_CHAR_MIN	-128
#define FBXSDK_CHAR_MAX	127
#define FBXSDK_UCHAR_MIN	0
#define FBXSDK_UCHAR_MAX	255
#define FBXSDK_SHORT_MIN	-32768
#define FBXSDK_SHORT_MAX	32767
#define FBXSDK USHORT_MIN	0
#define FBXSDK USHORT_MAX	65535
#define FBXSDK_INT_MIN	0x80000000
#define FBXSDK_INT_MAX	0xffffffff
#define FBXSDK_UINT_MIN	0
#define FBXSDK_UINT_MAX	0xffffffff
#define FBXSDK_LONG_MIN	FBXSDK_INT_MIN
#define FBXSDK_LONG_MAX	FBXSDK_INT_MAX
#define FBXSDK ULONG_MIN	FBXSDK_UINT_MIN
#define FBXSDK ULONG_MAX	FBXSDK_UINT_MAX
#define FBXSDK_FLOAT_MIN	FLT_MIN
#define FBXSDK_FLOAT_MAX	FLT_MAX
#define FBXSDK_FLOAT_EPSILON	FLT_EPSILON
#define FBXSDK_DOUBLE_MIN	DBL_MIN
#define FBXSDK_DOUBLE_MAX	DBL_MAX
#define FBXSDK_DOUBLE_EPSILON	DBL_EPSILON
#define FBXSDK_TOLERANCE	(1.0e-6)

```
template<class T> class FBXSDK_DLL FbxVectorTemplate4 { ... }  
typedef FbxVectorTemplate4<FbxDouble> FbxDouble4;  
typedef FbxVectorTemplate4<FbxDouble4> FbxDouble4x4;
```

FBX 씬 임포트(Import)

- **FbxIOSettings** 클래스

- 씬의 어떤 데이터(원소)를 임포트/익스포트 하는가(옵션)를 설정
(카메라, 조명, 메쉬, 텍스쳐, 재질, 애니메이션, 사용자 속성 등)
FbxIOSettings 객체가 생성될 때 모든 옵션은 기본값을 가짐

```
#include <fbxsdk.h>
#include <fbxfilesdk/fbxio/fbxiosettings.h>

FbxManager *pfbxManager = FbxManager::Create();
FbxIOSettings *pfbxIOSettings = FbxIOSettings::Create(pfbxManager, IOSROOT);
pfbxManager->SetIOSettings(pfbxIOSettings);
FbxImporter *pfbxImporter = FbxImporter::Create(pfbxManager, "");
pfbxImporter->Initialize("Test.fbx", -1, pfbxManager->GetIOSettings());
```

- FbxIOSettings 옵션은 계층 구조를 가짐
FbxIOSettings 옵션 값에 접근하기 위해 경로 또는 경로 상수를 사용
FbxIOSettings 설정 옵션은 "Fbxiosettingspath.h" 파일에 정의
옵션은 XML 파일을 사용하여 저장하고 로드할 수 있음
 ReadXMLFile(), **WriteXMLFile()**, **WriteXmlPropToFile()**

```
pfbxIOSettings->GetBoolProp("Import|IncludeGrp|Animation", true);
pfbxIOSettings->GetBoolProp(IMP_ANIMATION, true);
pfbxIOSettings->SetBoolProp(IMP_ANIMATION, false);
pfbxIOSettings->SetBoolProp(EXP_ANIMATION, true);
pfbxIOSettings->GetBoolProp(EXP_ANIMATION, false);
```

FBX 씬 임포트(Import)

- **FbxIOSettings** 클래스

```
bool GetBoolProp(char *pName, bool pDefaultValue);
void SetBoolProp(char *pName, bool pValue);
double GetDoubleProp(char *pName, double pDefaultValue);
void SetDoubleProp(char *pName, double pValue);
int GetIntProp(char *pName, int pDefaultValue);
void SetIntProp(char *pName, int pValue);
FbxTime GetTimeProp(char *pName, FbxTime pDefaultValue);
void SetTimeProp(char *pName, FbxTime pValue);
FbxString GetStringProp(char *pName, FbxString pDefaultValue);
void SetStringProp(char *pName, FbxString pValue);
FbxProperty GetProperty(char *pName);
FbxProperty GetProperty(FbxProperty& pParent);
```

```
#define IOSN_IMPORT "Import"
#define IOSN_ADV_OPT_GRP "AdvOptGrp"
#define IMP_ADV_OPT_GRP IOSN_IMPORT "|" IOSN_ADV_OPT_GRP
#define IOSN_FILE_FORMAT "FileFormat"
#define IMP_FILEFORMAT IMP_ADV_OPT_GRP "|" IOSN_FILE_FORMAT
#define IOSN_FBX "Fbx"
#define IMP_FBX IMP_FILEFORMAT "|" IOSN_FBX
#define IOSN_MATERIAL "Material"
#define IMP_FBX_MATERIAL IMP_FBX "|" IOSN_MATERIAL
```

```
pfbxIOSettings->SetBoolProp(IMP_FBX_MATERIAL, true);
pfbxIOSettings->SetBoolProp(IMP_FBX_TEXTURE, true);
pfbxIOSettings->SetBoolProp(IMP_FBX_SHAPE, false);
pfbxIOSettings->SetStringProp(IMP_FBX_PASSWORD, fbxstrPassword);
pfbxIOSettings->SetBoolProp(IMP_FBX_PASSWORD_ENABLE, true);
pfbxIOSettings->SetBoolProp(IMP_FBX_ANIMATION, true);
pfbxIOSettings->SetBoolProp(IMP_FBX_GLOBAL_SETTINGS, true);
pfbxIOSettings->SetBoolProp(EXP_FBX_MATERIAL, true);
pfbxIOSettings->SetBoolProp(EXP_FBX_TEXTURE, true);
pfbxIOSettings->SetBoolProp(EXP_FBX_EMBEDDED, true);
```

fbxiosettingspath.h

FbxString fbxstrPassword;

텍스쳐, 사운드 등을 이진 FBX 파일에 포함시킴

FBX 씬 임포트(Import)

- 씬 임포트하기(Importing a Scene)

- FbxImporter 클래스

```
FbxManager *pfbxManager = FbxManager::Create();
FbxIOSettings *pfbxIOSettings = FbxIOSettings::Create(pfbxManager, IOSROOT);
pfbxManager->SetIOSettings(pfbxIOSettings);
pfbxIOSettings->SetBoolProp(IMP_FBX_MATERIAL, true);
pfbxIOSettings->SetBoolProp(IMP_FBX_TEXTURE, true);
FbxImporter *pfbxImporter = FbxImporter::Create(pfbxManager, "");
bool bImportStatus = pfbxImporter->Initialize("Test.fbx", -1, pfbxManager->GetIOSettings());
```

```
int nFileVersion;
pfbxImporter->GetFileVersion(nFileVersion);
FbxScene *pfbxScene = FbxScene::Create(pfbxManager, "SceneName");
pfbxImporter->Import(pfbxScene);
pfbxImporter->Destroy();
```

```
FbxClassId nShaderClassID = pfbxManager->FindFbxFileClass("Shader", "FlatShader");
for (int i = 0; i < pfbxNode->GetSrcObjectCount(nShaderClassID); i++)
{
    FbxObject *pfbxObject = pfbxNode->GetSrcObject(nShaderClassID, i);
}
```

```
bool FbxImporter::Initialize(char *pFileName, int pFileFormat = -1, FbxIOSettings *pIOSettings = NULL);
bool FbxImporter::Import(FbxDocument *pDocument, bool bNonBlocking = false);
void FbxImporter::GetFileVersion(int &pMajor, int &pMinor, int &pRevision);
```

FBX 씬 임포트(Import)

- 씬 익스포트하기(Exporting a Scene)

- FbxExporter 클래스

```
FbxManager *pfbxManager = FbxManager::Create();
FbxIOSettings *pfbxIOSettings = FbxIOSettings::Create(pfbxManager, IOSROOT);
pfbxManager->SetIOSettings(pfbxIOSettings);
FbxExporter *pfbxExporter = FbxExporter::Create(pfbxManager, "");
bool bExportStatus = pfbxExporter->Initialize("Test.fbx", -1, pfbxManager->GetIOSettings());
if (!bExportStatus) printf("Error %s\n", pfbxExporter->GetLastErrorString());
int nFileMajor, nFileMinor, nFileRevision;
FbxManager::GetFileVersion(nFileMajor, nFileMinor, nFileRevision);
FbxScene *pfbxScene = FbxScene::Create(pfbxManager, "SceneName");
...
//씬을 임포트하거나 새로운 씬을 생성
...
pfbxExporter->Export(pfbxScene);
pfbxExporter->Destroy();
```

```
(*(pfbxManager->GetIOSettings())).SetBoolProp(EXP_FBX_EMBEDDED, true);
int nFileFormat = pfbxManager->GetIOPluginRegistry()->GetNativeWriterFormat();
bool bExportStatus pfbxExporter->Initialize(nFilename, nFileFormat, pfbxManager->GetIOSettings());
//씬을 익스포트
```

```
bool FbxExporter::Initialize(char *pFileName, int pFileFormat = -1, FbxIOSettings *pIOSettings = NULL);
bool FbxExporter::Export(FbxDocument *pDocument, bool bNonBlocking = false);
```

FBX 씬 임포트(Import)

- **임베디드 미디어 참조하기(Referencing Embedded Media)**

- **임베디드 미디어 참조하기(Referencing Embedded Media)**

이진 FBX 파일은 임베디드 미디어를 포함할 수 있음

임베디드 미디어는 FBX 파일에서 추출되어 서브 폴더에 복사됨

FBX 파일의 위치에 FBX 파일의 이름과 같은 이름(확장자 .fbm)으로 생성됨

“<path>\MyScene.fbx”에 포함된 미디어는 “<path>\MyScene.fbm”에 저장

- **임베디드되지 않은 미디어 참조하기(Referencing Non-Embedded Media)**

ASCII FBX 파일

- 절대적 경로에서 미디어 파일을 확인

```
FbxString pfbxstrTexPath = gpstrAppPath + "\\Crate.jpg";
FbxFileTexture *pfbxTexture = FbxFileTexture::Create(pfbxScene, "Crate Texture");
pfbxTexture->SetFileName(pfbxstrTexPath.Buffer());
```

- 절대적 경로에 미디어 파일이 존재하지 않으면 FBX SDK는 상대적 경로를 사용
(상대적 경로는 씬을 익스포트할 때 FBX 파일에 자동적으로 저장됨)

```
bool FbxFileTexture::SetFileName(char *pFileName);
bool FbxFileTexture::SetRelativeFileName(char *pFileName);
```

FBX SDK 객체 모델

- **FBX SDK 객체 모델(Object Model)**

- **메모리 관리**

- **메모리 관리**

FBX SDK 관리자 객체(Manager Object): [FbxManager](#)

[FbxManager](#) 객체는 SDK가 사용하는 객체들을 생성, 관리, 소멸

- **객체(Object)**

- **객체(Object)**

FBX SDK의 대부분의 클래스는 [FbxObject](#) 클래스에서 파생

- **노드(Node)**

- **노드(Node)**

FBX 씬 그래프는 [FbxNode](#) 객체들의 트리로 구성

카메라, 메쉬, NURBS, 조명, 그밖의 씬 구성 요소들

씬 구성 요소들은 [FbxNodeAttribute](#) 클래스의 인스턴스

- **입출력 객체(I/O Object)**

- **입출력 객체(I/O Object)**

[FbxImporter](#), [FbxExporter](#), [FbxIO](#)

- **컬렉션(Collection)**

- **컬렉션(Collection)**

대부분의 컨테이너 클래스는 [FbxCollection](#) 클래스에서 파생

[FbxCollection::AddMember\(\)](#), [FbxCollection::RemoveMember\(\)](#),

[FbxCollection::FindMember\(\)](#), [FbxCollection::IsMember\(\)](#),

[FbxCollection::GetMemberCount\(\)](#), [FbxCollection::GetMember\(\)](#)

- 객체 속성(Properties)**

- **객체 속성(Properties)**

[FbxNode](#)와 같은 FBX 객체는 C++ 멤버 변수가 아닌 FBX 속성들을 사용

- **객체 속성(Properties)**

[FbxProperty](#), [FbxProperty::Set\(\)](#), [FbxProperty::Get\(\)](#)

- 연결(Connection)**

- **연결(Connection)**

객체와 객체, 속성과 속성, 객체와 속성을 결합할 수 있음

- **연결(Connection)**

연결을 위한 클래스는 없음, 멤버함수를 사용([FbxObject::GetSrcObject\(\)](#), ...)

- **대부분의 하나의 객체(단일 객체: Singleton)를 사용**

FBX SDK 객체 모델

- **FBX SDK 메모리 관리(Managing Memory)**

- **FBX SDK 관리자(Manager) 생성**

FbxManager 객체는 SDK가 사용하는 객체들을 생성, 관리, 소멸

```
FbxManager *pfbxManager = FbxManager::Create();
```

```
FbxManager *FbxObject::GetFbxManager();
```

- **객체(Object)의 생성**

FBX SDK 객체는 Create(), Destroy() 멤버 함수로 생성

이 멤버 함수는 메모리 할당과 반납을 위해 FbxManager 클래스를 사용

```
FbxScene *pfbxScene = FbxScene::Create(pfbxManager, "SceneName");
```

- **메모리 할당(Memory Allocation)**

FbxManager는 생성되는 객체를 위한 메모리를 자동적으로 할당

메모리 할당 함수는 FbxSetMallocHandler() 함수로 변경 가능

- **FBX 객체 이름**

Create() 함수의 두 번째 파라미터는 문자열

객체가 생성될 때 문자열은 새로운 객체의 이름으로 사용(빈 문자열 가능)

디버깅을 위해 구별되는 문자열 이름을 사용하는 것을 권장

- **씬에서 객체의 생성(Creation)**

FbxScene 객체는 메쉬, 조명, 애니메이션, 캐릭터 등과 같은 씬 요소를 포함

```
FbxNode *pfbxNode = FbxNode::Create(pfbxScene, "Node");
```

```
FbxMesh *pfbxMesh = FbxMesh::Create(pfbxScene, "");
```

- **객체의 소멸(Destruction)**

FBX SDK 객체는 Destroy() 멤버 함수로 명시적으로 파괴해야 함

FbxManager 객체는 소멸되는 객체의 메모리를 반환하고 연결 관계를 갱신

FBX SDK 객체 모델

- **FBX 객체(Objects)**
 - **FBX 객체의 생성과 소멸**

FBX 객체는 `FbxObject` 클래스 또는 파생 클래스의 인스턴스
FBX 객체는 클래스의 `Create()` 함수를 호출하여 생성
 `FbxManager` 객체 또는 `FbxScene` 객체의 참조를 첫번째 파라메터로 사용
 `FbxObject::GetFbxManager()`, `FbxObject::GetScene()`

```
FbxManager *pfbxManager = pfbxImporter->GetFbxManager();
```

FBX 객체는 `Destroy()` 함수를 호출하여 소멸
 `FbxObject::Destroy()`, `FbxManager::Destroy()`, `FbxScene::Destroy()`
 - **속성(Properties)**

FBX SDK는 객체에 속성을 연결하기 위하여 `FbxProperty` 클래스를 사용
 - **컬렉션(Collections)**

`FbxAnimLayer`, `FbxAnimStack`, `FbxScene` 클래스는 `FbxCollection`에서 파생
 `FbxCollection::AddMember()`, `FbxCollection::RemoveMember()`
 `FbxCollection::GetMemberCount()`, `FbxCollection::GetMember()`
 `FbxCollection::FindMember()`
 - **FBX 객체의 복사(Copying)**

FBX 객체는 `FbxObject::Copy()` 멤버 함수를 호출하여 복사할 수 있음
= 연산자는 FBX 객체를 복사하기 위하여 사용할 수 없음
 `FbxObject`를 복사하면 연관된 속성과 값을 복사, 객체 연결 관계는 복사되지 않음

```
FbxMesh *pfbxSourceMesh = FbxMesh::Create(pfbxScene, "");  
FbxMesh *pfbxTargetMesh = FbxMesh::Create(pfbxScene, "");  
pfbxTargetMesh->Copy(pfbxSourceMesh);
```

FBX SDK 객체 모델

- **FBX 속성(Properties)**

- 속성 관리

FbxProperty 템플릿 클래스를 사용(FbxPropertyT<T>)

FbxObject가 생성될 때 기본 FbxProperty들이 자동적으로 초기화됨

자체적인 FbxProperty를 생성하려면 FbxProperty::Create()를 사용

```
FbxProperty p = FbxProperty::Create(pfbxScene, DTDouble3, "Vector3Property");
FbxSet<FbxDouble3>(p, FbxDouble3(1.1, 2.2, 3.3);
```

FbxProperty가 이름을 가지면 FbxProperty::GetName() 함수로 접근할 수 있음

FbxProperty의 인스턴스는 FbxProperty::Destroy()를 호출하여 소멸

FbxProperty 계층 구조의 소멸은 FbxProperty::DestroyRecursively()를 호출

- 속성 데이터(Property Data)

속성 데이터에 대한 접근: FbxProperty::Set(), FbxProperty::Get()

```
FbxDouble3 translation = pfbxNode->LclTranslation.Get();
FbxDouble3 rotation = pfbxNode->LclRotation.Get();
FbxDouble3 scaling = pfbxNode->LclScaling.Get();
```

- 플래그(Flags)

속성은 FbxPropertyFlags::eFbxPropertyFlags 집합을 포함

FbxProperty::GetFlag(), FbxProperty::ModifyFlag()

- 연산자(Operators): 비교(==, !=), 대입 연산자(=)

- 사용자 정의 데이터(User-defined Data)

FbxProperty는 사용자 정의 데이터를 포함할 수 있음

FBX SDK 객체 모델

- **FBX 속성(Properties)**

- **속성 계층(Hierarchy)**

속성들은 계층 구조로 구성될 수 있음

하나의 속성은 하나의 FBX 객체(FbxObject) 또는 다른 속성에 연결될 수 있음

속성에 연결된 객체는 FbxProperty::GetFbxObject()를 호출하여 접근할 수 있음

- FBX 객체의 속성들에 대한 접근(Acessing)

FBX 객체는 여러 개의 속성들을 가질 수 있음

FbxObject::FindProperty()

FbxObject::GetFirstProperty(), FbxObject::GetNextProperty()

- 속성 계층 구조의 탐색(Navigating)

FbxProperty::GetParent(), FbxProperty::GetChild(), FbxProperty::GetSibling()

FbxProperty FbxProperty::Find()

- FBX 객체와 FBX 속성의 연결(Connection)

FbxProperty::ConnectDstObject(), FbxProperty::ConnectSrcProperty()

FbxObject::GetSrcProperty(), FbxProperty::GetDstObject()

FbxProperty::GetSrcProperty(), FbxProperty::GetDstProperty()

```
FbxObjectMetaData *pfbxFamilyMetaData = FbxObjectMetaData::Create(pfbxScene, "Family");
```

```
FbxProperty::Create(pfbxFamilyMetaData, DTString, "Level", "Level").Set(FbxString("Family")); // String
```

```
FbxProperty::Create(pfbxFamilyMetaData, DTString, "Type", "Type").Set(FbxString("Wall")); // String
```

```
FbxProperty::Create(pfbxFamilyMetaData, DTFloat, "Width", "Width").Set(10.0f); // float
```

```
FbxProperty::Create(pfbxFamilyMetaData, DTDouble, "Weight", "Weight").Set(25.0); // double
```

```
FbxProperty::Create(pfbxFamilyMetaData, DTDouble, "Cost", "Cost").Set(1.25); // double
```

FBX SDK 객체 모델

- **연결(Connections)**

- **연결(Connection)**

FBX 객체들 사이의 관계 또는 FBX 속성들 사이의 관계를 관리하는 자료구조

`FbxObject::ConnectSrcObject()`, `FbxObject::ConnectDstObject()`

`FbxProperty::ConnectDstObject()`, `FbxProperty::ConnectSrcProperty()`

- 객체-속성 연결(속성: Source, 객체: Destination)

`FbxObject::GetSrcProperty()`, `FbxProperty::GetDstObject()`

- 객체-객체 연결(자식: Source, 부모: Destination)

객체들 사이의 부모-자식 관계

`FbxObject::GetSrcObject()`, `FbxObject::GetDstObject()`

- 속성-속성 연결(자식: Source, 부모: Destination)

속성들 사이의 부모-자식 관계

`FbxProperty::GetSrcProperty()`, `FbxProperty::GetDstProperty()`

`FbxObject *pObj0`

`FbxProperty *pProp0`

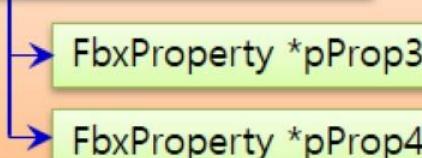


`FbxObject *pObj1`

`FbxProperty *pProp1`

`FbxObject *pObj2`

`FbxProperty *pProp2`



```
int nSrcObjects = pObj0->GetSrcObjectCount();
FbxObject *pObj1 = pObj0->GetSrcObject(0);
FbxObject *pObj2 = pObj0->GetSrcObject(1);
```

```
FbxProperty *pProp0 = pObj0->GetSrcProperty();
```

```
FbxObject *pObj0 = pObj1->GetDstObject();
```

```
FbxProperty *pProp2 = pObj2->GetSrcProperty();
int nProperties = pProp2->GetSrcPropertyCount();
FbxProperty *pProp3 = pProp2->GetSrcProperty(0);
FbxProperty *pProp4 = pProp2->GetSrcProperty(1);
```

FBX SDK 객체 모델

- **연결(Connections)**

- **객체들과 속성들의 연결(Connection)**

연결의 개념은 `FbxProperty`가 `FbxObject`에 동적으로 추가될 수 있게 함
사용자 정의 데이터 형을 정의할 수 있는 신축성을 제공

`FbxProperty::SetUserDataPtr()`, `FbxObject::ConnectSrcProperty()`

`FbxCollection` 클래스는 계층구조에 여러 객체들을 구성하기 위하여 연결에 의존
`FbxScene` 클래스는 `FbxNode` 객체들의 계층구조를 포함

`FbxScene::GetRootNode()`

`FbxNodeAttribute`는 메쉬, 카메라 등의 위치를 나타내기 위해 `FbxNode`에 부착됨

- “객체-객체” 연결(씬에서 노드들 사이의 부모-자식 관계)

```
FbxNode *pfbxParentNode = pfbxScene->GetRootNode();
FbxNode *pfbxChildNode = FbxNode::Create(pfbxScene, "Child");
pfbxParentNode->AddChild(pfbxChildNode);
```

- “객체-객체” 연결(노드와 노드 속성)

`FbxNode`와 `FbxNodeAttribute`의 관계는 `SetNodeAttribute()`로 설정

`FbxNode::SetNodeAttribute(FbxNodeAttribute *pNodeAttribute)`

`FbxMesh` 인스턴스도 같은 방법으로 씬의 노드에 연결될 수 있음

`FbxNodeAttribute`: `FbxNode`의 `Source` 객체

재질(`FbxSurfaceMaterials`)도 `FbxNode`의 `Source` 객체로 연결됨

- “객체-속성” 연결(노드와 변환)

변환 데이터(`FbxTypedProperty`)는 `FbxNode`의 `Source` 속성

`FbxPropertyT<FbxDouble3> LclTranslation`

FBX SDK 객체 모델

- 오류 처리(Error Handling)

- 오류(Error)

FBX 클래스의 함수들은 오류가 발생하면 거짓(false)를 반환

pfbxObject->**GetStatus().GetLastErrorString()** 함수는 오류 메시지 문자열을 반환

pfbxObject->**GetStatus().GetCode()** 함수는 오류를 나타내는 정수 값을 반환

```
bool LoadScene(FbxManager *pfbxManager, FbxDocument *pfbxScene, char *pstrFilename)
{
    FbxImporter *pfbxImporter = FbxImporter::Create(pfbxManager, "");
    if (!pfbxImporter->Initialize(pstrFilename, -1, pfbxManager->GetIOSettings())) {
        FbxString fbxstrError = pfbxImporter->GetStatus().GetErrorString();
        FBXSDK_printf("Error returned: %s\n\n", fbxstrError.Buffer());
        if (pfbxImporter->GetStatus().GetCode() == FbxStatus::eInvalidFileVersion) {
            ...
        }
        return(false);
    }
    if (!pfbxImporter->Import(pfbxScene)) {
        if (pfbxImporter->GetStatus().GetCode() == FbxIO::ePASSWORD_ERROR) {
            FBXSDK_printf("Please enter password: ");
            char strPassword[1024];
            strPassword[0] = 'W0';
            scanf("%s", strPassword);
            FbxString fbxstrPassword(strPassword);
            ...
        }
    }
}
```

FBX SDK 객체 모델

- **FBX 문자열**

- FBX SDK는 내부적으로 UTF-8 문자열을 사용함
UTF-8(8-비트 UCS/Unicode Transformation Format): 가변 길이 문자 인코딩
유니코드의 모든 문자를 표현할 수 있음
- FBX SDK는 UTF-8 문자열을 운영체제의 API의 함수에 맞게 변환함
- ASCII 문자가 아닌 문자열을 사용할 때 UTF-8 문자열로 변환해야 함
- fbxstring.h

```
void FbxWCToAnsi(const wchar_t *pInWideChar, char *&pOutANSI);
void FbxAnsiToWC(const char *pInANSI, wchar_t *&pOutWideChar);
void FbxAnsiToUTF8(const char *pInANSI, char *&pOutUTF8);
void FbxUTF8ToAnsi(const char *pInUTF8, char *&pOutANSI);
void FbxUTF8ToWC(const char *pInUTF8, wchar_t *&pOutWideChar);
void FbxWCToUTF8(const wchar_t *pInWideChar, char *&pOutUTF8);
```

- FbxString 클래스

```
FbxString(const FbxString& pString);
FbxString(const char* pString);
FbxString(char pChar, size_t pNbRepeat=1);
FbxString(const char* pCharPtr, size_t pLength);
size_t GetLen() const;
size_t Size() const;
char& operator[](int nIndex);
char operator[](int nIndex) const;
char* Buffer();
const char* Buffer() const;
```

FBX SDK 객체 모델

- **FBX SDK**

```
template<class T> class FbxArray<T>;
```

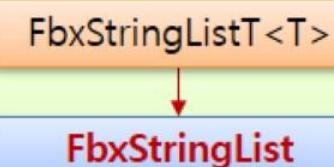
```
int InsertAt(const int nIndex, const T pElement);
int Add(const T pElement);
int AddUnique(const T pElement);
int Size();
int Capacity();
T& operator[](int nIndex);
T GetAt(const int nIndex);
T GetFirst();
T GetLast();
int Find(const T pElement, const int nstartIndex=0);
int FindReverse(const T pElement, const int nstartIndex=FBXSDK_INT_MAX);
bool Reserve(const int nCapacity);
void SetAt(const int nIndex, const T pElement);
void SetFirst(const T pElement);
void SetLast(const T pElement);
T RemoveAt(const int nIndex);
T RemoveFirst();
T RemoveLast();
bool RemoveAt(const T pElement);
bool Resize(const int nSize);
bool Grow(const int nSize);
void Clear();
T *GetArray();
operator T *();
```

FBX SDK 객체 모델

- FBX SDK

```
template<class T> class FbxStringListT<T>;
```

```
int AddItem(T *pItem);
int InsertItemAt(int nIndex, T *pItem);
T *GetItemAt(int nIndex);
int FindItem(T *pItem);
void RemoveLast();
int GetCount(); //원소의 개수
FbxString & operator[](int nIndex);
FbxHandle GetReferenceAt(int nIndex);
void SetReferenceAt(int nIndex, FbxHandle pRef);
char * GetStringAt(int nIndex);
virtual bool SetStringAt(int nIndex, const char *pString);
int Find(Type &pItem);
int FindIndex(const char *pString);
bool Remove(Type &pItem);
void Sort();
int Add(const char *pString, FbxHandle pItem=0);
virtual int InsertAt(int nIndex, const char *pString, FbxHandle pItem=0);
virtual void RemoveAt(int nIndex);
virtual void Clear();
virtual void GetText(FbxString& pText);
virtual int SetText(const char *pList);
```



FBX SDK 객체 모델

- **FBX SDK 확장하기**

- 사용자 데이터(User Data, Custom Data) 설정
FbxObject 객체와 FbxProperty 객체는 사용자 데이터를 가질 수 있음
 - FbxObject 사용자 데이터
FbxObject::SetUserDataPtr(), FbxObject::GetUserDataPtr()
 - FbxProperty 사용자 데이터
FbxProperty::SetUserDataPtr(), FbxProperty::GetUserDataPtr()

```
FbxNodeAttribute *pfbxNodeAttribute = pfbxNode->GetNodeAttribute();
if (pfbxNodeAttribute) {
    if (pfbxNodeAttribute->GetAttributeType() == FbxNodeAttribute::eMesh) {
        FbxMesh *pfbxMesh = pfbxNode->GetMesh();
        if (pfbxMesh && !pfbxMesh->GetUserDataPtr()) {
            FbxAutoPtr<CMesh> pMesh(new CMesh);
            if (pMesh->Initialize(pfbxMesh)) pfbxMesh->SetUserDataPtr(pMesh.Release());
        }
    }
}
```

```
for (int i = 0; i < pfbxNode->GetMaterialCount(); i++) {
    FbxSurfaceMaterial *pfbxMaterial = pfbxNode->GetMaterial(i);
    if (pfbxMaterial && !pfbxMaterial->GetUserDataPtr()) {
        FbxAutoPtr<CMaterial> pMaterial (new CMaterial);
        if (pMaterial->Initialize(pfbxMaterial)) pfbxMaterial->SetUserDataPtr(pMaterial.Release());
    }
}
```

노드와 씬 그래프

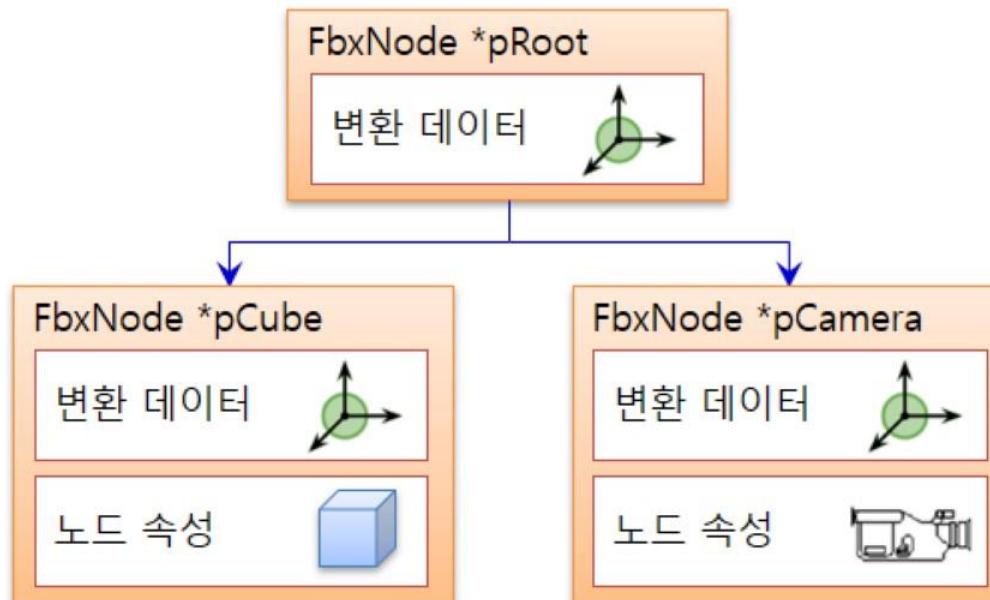
- 노드(Node)와 노드의 속성(Node Attribute)

- 노드(Node)

메쉬, 조명, 카메라와 같은 씬 요소들의 위치와 방향은 일련의 SRT로 표현됨
기하학적 변환 데이터는 **FbxNode**로 추상화됨
씬은 노드들의 부모-자식 계층구조를 포함

- 노드의 속성(Node Attribute)

카메라는 SRT 값뿐 아니라 FOV, 프레임의 가로/세로 크기 등의 데이터를 가짐
이러한 데이터는 **FbxNodeAttribute**의 파생 클래스인 **FbxCamera**로 추상화됨
FbxMesh와 **FbxLight**도 **FbxNodeAttribute**의 파생 클래스
속성 객체는 씬에서 노드의 위치를 나타내기 위하여 **FbxNode**에 연결됨



노드와 씬 그래프

- 노드(FbxNode) 클래스

```
FbxNode *GetParent();
int GetChildCount(bool bRecursive=false);
FbxNode *GetChild(int nIndex);
FbxNode *FindChild(const char *pName, bool bRecursive=true, bool bInitial=false);
int GetNodeAttributeCount();
FbxNodeAttribute *GetNodeAttribute();
FbxNodeAttribute *GetNodeAttributeByIndex(int nIndex);
int GetNodeAttributeIndex(FbxNodeAttribute *pNodeAttribute, FbxStatus *pStatus=NULL);
FbxMarker *GetMarker();
FbxGeometry *GetGeometry();
FbxMesh *GetMesh();
FbxNurbs *GetNurbs();
FbxPatch *GetPatch();
FbxCamera *GetCamera();
FbxLight *GetLight();
int GetMaterialCount();
FbxSurfaceMaterial *GetMaterial(int nIndex);
int GetMaterialIndex(const char *pName);
FbxAMatrix& EvaluateGlobalTransform(FbxTime pTime=FBXSDK_TIME_INFINITE, FbxNode::EPivotSet
pPivotSet=FbxNode::eSourcePivot, bool bApplyTarget=false, bool bForceEval=false);
FbxAMatrix & EvaluateLocalTransform (FbxTime pTime=FBXSDK_TIME_INFINITE, FbxNode::EPivotSet
pPivotSet=FbxNode::eSourcePivot, bool bApplyTarget=false, bool bForceEval=false);
FbxPropertyT<FbxDouble3> LclTranslation;
FbxPropertyT<FbxDouble3> LclRotation;
FbxPropertyT<FbxDouble3> LclScaling;
```

노드와 씬 그래프

- **FBX 씬(Scene)**

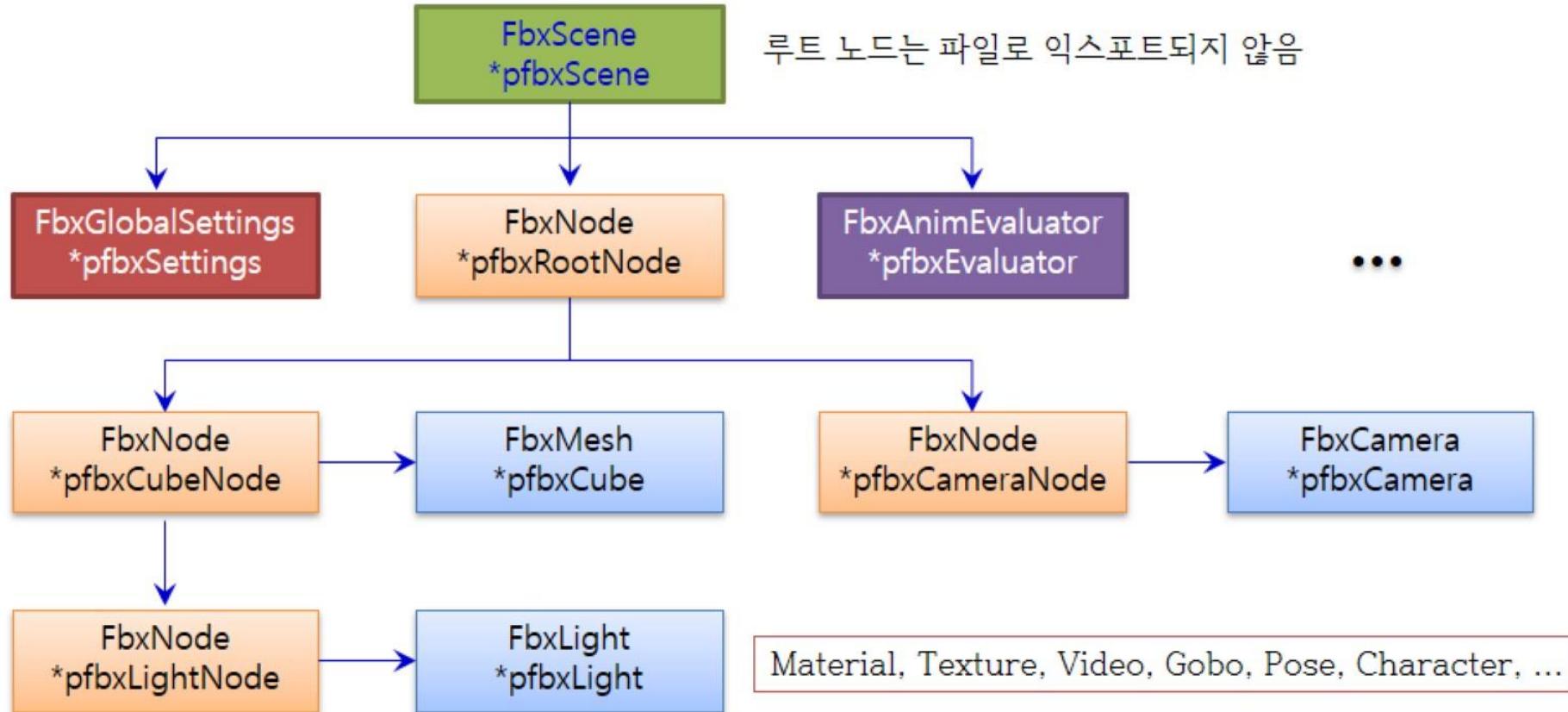
- **씬 그래프의 구성**

- FBX 씬 그래프는 FbxScene 클래스로 추상화됨

- 씬은 노드(FbxNode)들의 계층 구조로 구성됨

- 씬의 루트 노드는 **FbxScene::GetRootNode()** 함수로 접근

- 씬의 요소는 FbxNode와 FbxNodeAttribute의 파생 클래스를 결합하여 정의됨



노드와 씬 그래프

- **FbxScene** 클래스

```
FbxDocumentInfo *GetSceneInfo();
FbxNode *GetRootNode();
void FillPoseArray(FbxArray<FbxPose *>& pPoseArray);
void FillTextureArray(FbxArray<FbxTexture *>& pTextureArray);
void FillMaterialArray(FbxArray<FbxSurfaceMaterial *>& pMaterialArray);
int GetCharacterCount();
FbxCharacter *GetCharacter(int nIndex);
int GetCharacterPoseCount();
int GetPoseCount();
FbxPose *GetPose(int nIndex);
FbxGlobalSettings& GetGlobalSettings();
int GetMaterialCount();
FbxSurfaceMaterial *GetMaterial(int nIndex);
FbxSurfaceMaterial *GetMaterial(char *pName);
int GetTextureCount();
FbxTexture *GetTexture(int nIndex);
FbxTexture *GetTexture(char *pName);
int GetNodeCount();
FbxNode *GetNode(int nIndex);
FbxNode *FindNodeByName(const FbxString& pName);
int GetGeometryCount();
FbxGeometry *GetGeometry(int nIndex);
int GetVideoCount();
FbxVideo *GetVideo(int nIndex);
bool ComputeBoundingBoxMinMaxCenter(FbxVector4& pBBoxMin, FbxVector4 &pBBoxMax, FbxVector4
&pBBoxCenter, bool bSelected=false, const FbxTime &pTime=FBXSDK_TIME_INFINITE, int nAnimLayerId=0);
```

노드와 씬 그래프

- **FBX 씬(Scene)**

- FBX 씬의 생성

씬이 생성되면 씬은 하나의 루트 노드를 가짐

```
FbxManager *pfbxManager = FbxManager::Create();  
FbxScene *pfbxScene = FbxScene::Create(pfbxManager, "Scene Name");  
FbxNode *pfbxRootNode = pfbxScene->GetRootNode();  
FbxNode *pfbxChildNode = FbxNode::Create(pfbxScene, "Child");  
pfbxRootNode->AddChild(pfbxChildNode);
```

- 전역 씬 설정(Global Scene Settings)
씬의 축 시스템, 시스템 단위, 확산(Ambient) 조명, 시간 설정 등을 정의
FbxGlobalSettings (씬이 생성되면 자동으로 하나의 전역 설정을 가짐)
FbxScene::GetGlobalSettings()
 - 애니메이션 계산(Animation Evaluation)
특정한 시간에 씬의 각 노드의 애니메이션(기하학적 변환)을 계산
FbxAnimEvaluator
FbxScene::GetEvaluator()
 - 텍스쳐와 재질 관리(Texture and Material Management)
FbxSurfaceMaterial, **FbxTexture**
FbxScene::GetMaterial()
FbxScene::GetTexture()
 - 캐릭터와 캐릭터 포즈 관리(Character and Character Pose Management)
FbxCharacter, **FbxCharacterPose**
FbxScene::GetCharacter()
FbxScene::GetCharacterPose()

노드와 씬 그래프

- **FBX 씬(Scene)**

- FBX 씬의 병합(Merge)

```
FbxManager *pfbxManager = FbxManager::Create();
FbxScene *pfbxScene = FbxScene::Create(pfbxManager, "First Scene");
FbxIOSettings *pfbxIOSettings = FbxIOSettings::Create(pfbxSdkManager, IOSROOT);
pfbxManager->SetIOSettings(pfbxIOSettings);
FbxImporter *pfbxImporter = FbxImporter::Create(pfbxManager, "");
pfbxImporter->Initialize("First.fbx", -1, pfbxSdkManager->GetIOSettings());
pfbxImporter->Import(pfbxScene);
pfbxImporter->Destroy();

FbxNode *pfbxRootNode = pfbxScene->GetRootNode();
FbxScene *pfbxMergedScene = FbxScene::Create(pfbxManager, "Merged Scene");
int nChildren = pfbxRootNode->GetChildCount();
for (int i = 0; i < nChildren; i++) {
    FbxNode *pfbxChildNode = pfbxRootNode->GetChild(i);
    pfbxMergedScene->GetRootNode()->AddChild(pfbxChildNode);
}
pfbxRootNode->DisconnectAllSrcObject();
int nSceneObjects = pfbxScene->GetSrcObjectCount();
for (int i = 0; i < nSceneObjects; i++) {
    FbxObject *pfbxObject = pfbxScene->GetSrcObject(i);
    if ((pfbxObject == pfbxRootNode) || (*pfbxObject == pfbxScene->GetGlobalSettings())) continue;
    pfbxObject->ConnectDstObject(pfbxMergedScene);
}
pfbxScene->DisconnectAllSrcObject();
```

노드와 씬 그래프

- 씬 축(Axis)과 단위 변환(Unit Conversion)

- 씬 축과 단위 변환

FBX 씬(FbxScene)은 기본적으로 **오른손 좌표계(Y-Up)**로 생성
씬의 축 시스템과 단위는 응용 프로그램에 따라 변경될 필요가 있음

FbxAxisSystem::ConvertScene()

FbxSystemUnit::ConvertScene()

- ConvertScene() 함수 호출은 메쉬의 정점 값을 바꾸지는 않음

노드의 변환과 애니메이션에만 영향을 줌

```
if (pfbxScene->GetGlobalSettings().GetSystemUnit() == FbxSystemUnit::cm)
{
    const FbxSystemUnit::FbxUnitConversionOptions fbxConversionOptions = {
        false, /* mConvertRrsNodes */
        true, /* mConvertAllLimits */
        true, /* mConvertClusters */
        true, /* mConvertLightIntensity */
        true, /* mConvertPhotometricLProperties */
        true /* mConvertCameraClipPlanes */
    };
    FbxSystemUnit::m.ConvertScene(pfbxScene, fbxConversionOptions);
}
```

```
void FbxAxisSystem::ConvertScene(FbxScene *pScene);
void FbxAxisSystem::ConvertScene(FbxScene *pScene, FbxNode *pFbxRoot);
void FbxSystemUnit::ConvertScene(FbxScene *pScene, ConversionOptions &nOptions);
void FbxSystemUnit::ConvertScene(FbxScene *pScene, FbxNode *pFbxRoot, ConversionOptions &nOptions);
```

노드와 씬 그래프

- 씬 축(Axis)과 단위 변환(Unit Conversion)

```
FbxAxisSystem(EUpVector eUpVector, EFrontVector eFrontVector, ECoordSystem eCoordSystem);  
FbxAxisSystem(EPreDefinedAxisSystem eAxisSystem);
```

```
enum EUpVector { eXAxis = 1, eYAxis = 2, eZAxis = 3 };  
enum EFrontVector { eParityEven = 1, eParityOdd = 2 };  
enum ECoordSystem { eRightHanded, eLeftHanded };
```

EUpVector의 부호(1: Up, -1: Down)
EFrontVector(eParityEven: 남은 축의 첫번째)

```
FbxAxisSystem fbxAxisSystemReference = pfbxScene->GetGlobalSettings().GetAxisSystem();  
int nUpSign = 1, nFrontSign = 1;  
EUpVector fbxUpVector = fbxAxisSystemReference.GetUpVector(nUpSign);  
EFrontVector fbxFrontVector = fbxAxisSystemReference.GetFrontVector(nFrontSign);  
ECoordSystem fbxCoordSystem = fbxAxisSystemReference.GetCoordSystem();  
FbxAxisSystem fbxAxisSystem(nUpSign *fbxUpVector, nFrontSign *fbxFrontVector, fbxCoordSystem);
```

```
enum EPreDefinedAxisSystem {  
    eMayaZUp, eMayaYUp, eMax, eMotionBuilder, eOpenGL, eDirectX, eLightwave  
};
```

eMayaZUp	UpVector = ZAxis, FrontVector = -ParityOdd, CoordSystem = RightHanded
eMayaYUp	UpVector = YAxis, FrontVector = ParityOdd, CoordSystem = RightHanded
eMax	UpVector = ZAxis, FrontVector = -ParityOdd, CoordSystem = RightHanded
eMotionBuilder	UpVector = YAxis, FrontVector = ParityOdd, CoordSystem = RightHanded.
eOpenGL	UpVector = YAxis, FrontVector = ParityOdd, CoordSystem = RightHanded
eDirectX	UpVector = YAxis, FrontVector = ParityOdd, CoordSystem = LeftHanded
eLightwave	UpVector = YAxis, FrontVector = ParityOdd, CoordSystem = LeftHanded

노드와 씬 그래프

• 씬 축(Axis)과 단위(Unit) 변환(Conversion)

```
FbxAxisSystem fbxSceneAxisSystem = pfbxScene->GetGlobalSettings().GetAxisSystem();
FbxAxisSystem fbxAxis(FbxAxisSystem::eYAxis, FbxAxisSystem::eParityOdd, FbxAxisSystem::eRightHanded);
if (fbxSceneAxisSystem != fbxAxis) fbxAxis.ConvertScene(pfbxScene);
```

```
FbxSystemUnit fbxSceneSystemUnit = pfbxScene->GetGlobalSettings().GetSystemUnit();
if (fbxSceneSystemUnit.GetScaleFactor() != 1.0) FbxSystemUnit::cm.ConvertScene(pfbxScene);
```

```
FbxAxisSystem(EUpVector eUpVector, EFrontVector eFrontVector, ECoordSystem eCoordSystem);
```

EUpVector의 부호는 카메라를 기준으로 위(+1), 아래(-1)

EFrontVector의 부호는 카메라를 기준으로 앞(+1), 뒤(-1)

시스템 단위의 스케일 팩터(1 unit = 1cm), 스케일 배율(Multiplier, GetMultiplier())

미리 정의된 시스템 단위: FbxSystemUnit mm, dm, cm, m, km, Inch, Foot, Mile, Yard

```
FbxAxisSystem fbxSceneAxisSystem = pfbxScene->GetGlobalSettings().GetAxisSystem();
FbxAxisSystem fbxDirectXAxisSystem(FbxAxisSystem::eDirectX);
if (fbxSceneAxisSystem != fbxDirectXAxisSystem) fbxDirectXAxisSystem.ConvertScene(pfbxScene);
```

```
FbxSystemUnit fbxSceneSystemUnit = pfbxScene->GetGlobalSettings().GetSystemUnit();
if (fbxSceneSystemUnit.GetScaleFactor() != 1.0) FbxSystemUnit::m.ConvertScene(pfbxScene);
```

fbxSceneAxisSystem

```
mUpVector = { mAxis=eYAxis (1) mSign=1 }
mFrontVector = { mAxis=eZAxis (2) mSign=1 }
mCoorSystem = { mAxis=eXAxis (0) mSign=1 }
```

fbxDirectXAxisSystem

```
mUpVector = { mAxis=eYAxis (1) mSign=1 }
mFrontVector = { mAxis=eZAxis (2) mSign=1 }
mCoorSystem = { mAxis=eXAxis (0) mSign=-1 }
```

노드와 씬 그래프

• FBX 노드(Nodes)

- 노드(Node)

노드(FbxNode)는 주로 씬 요소들의 **위치, 회전, 크기** 변환을 나타내기 위하여 사용
씬(FbxScene)은 노드들의 부모-자식 계층구조를 포함(트리(Tree) 구조)
루트 노드는 `FbxScene::GetRootNode()`로 접근
새로운 노드를 생성하여 트리에 동적으로 추가할 수 있음

```
FbxNode *pfbxRootNode = pfbxScene->GetRootNode();
FbxNode *pfbxNode = FbxNode::Create(pfbxScene, "Child");
pfbxRootNode->AddChild(pfbxNode);
```

- 노드 계층 구조(Node Hierarchy)

`FbxNode::GetChild()`, `FbxNode::GetParent()`, `FbxNode::GetChildCount()`

노드의 위치, 회전, 크기 변환은 **부모 좌표계**로 표현(부모 노드에 상대적인 좌표계)

회전과 크기 변환이 적용되는 순서는 `ETransformInheritType`에 설정

변환의 상속은 `FbxNode::SetTransformationInheritType()`으로 설정

씬을 익스포트할 때 루트 노드는 저장되지 않음

- 노드 속성(Attribute)

씬의 **메쉬, 조명, 카메라** 등은 `FbxNodeAttribute`의 파생 클래스로 추상화됨

`FbxNodeAttribute`의 파생 클래스의 예: `FbxMesh`, `FbxLight`, `FbxCamera`

`FbxNodeAttribute`는 노드에 연결됨(속성이 변환 정보를 가질 수 있게 됨)

`FbxNode::SetNodeAttribute(FbxNodeAttribute* pfbxAttribute)`

하나의 노드에 여러 개의 속성이 연결될 수 있음(LOD 처리에 유용)

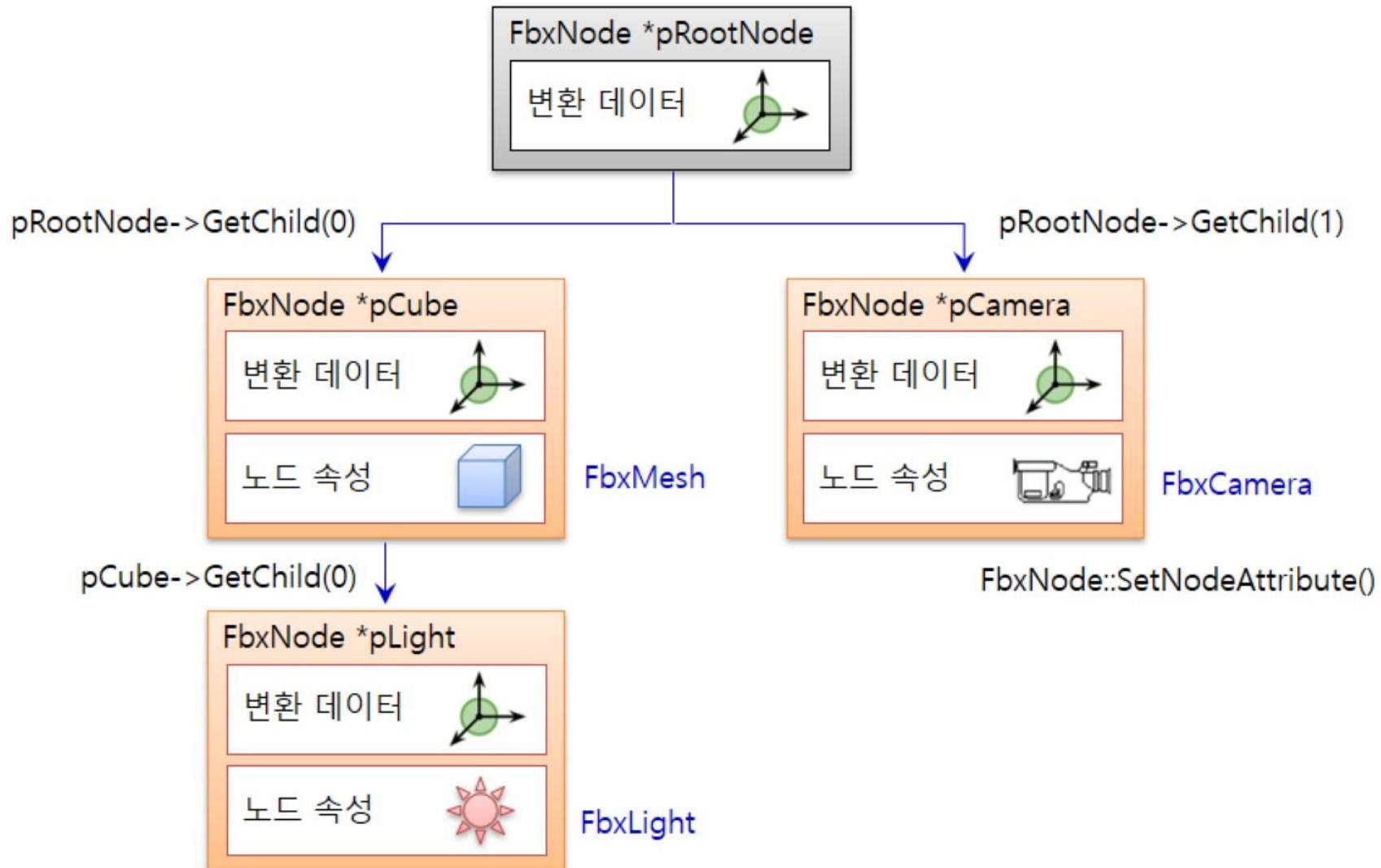
하나의 속성은 여러 노드에 연결될 수 있음(Instancing): 메모리를 줄일 수 있음

`FbxSurfaceMaterial`도 메쉬의 표면 재질을 정의하기 위해 노드에 연결될 수 있음

노드와 씬 그래프

- **FBX 노드(Nodes)**

- 노드 계층 구조(Node Hierarchy)



노드와 씬 그래프

- **FbxNode** 클래스

```
class FbxNode : public FbxObject {  
    FbxNode *GetParent();  
    int GetChildCount(bool pRecursive=false);  
    FbxNode *GetChild(int nIndex);  
    FbxNode *FindChild(const char *pName, bool pRecursive=true, bool pInitial=false);  
    int GetNodeAttributeCount();  
    FbxNodeAttribute *GetNodeAttribute();  
    FbxNodeAttribute *GetNodeAttributeByIndex(int nIndex);  
    FbxGeometry *GetGeometry();  
    FbxMesh *GetMesh();  
    int GetMaterialCount();  
    FbxSurfaceMaterial *GetMaterial(int nIndex);  
    FbxSkeleton *GetSkeleton();  
    FbxCamera *GetCamera();  
    FbxLight *GetLight();  
  
    FbxAnimEvaluator* GetAnimationEvaluator();  
    FbxAMatrix& EvaluateGlobalTransform(FbxTime pTime=FBXSDK_TIME_INFINITE, FbxNode::EPivotSet  
pPivotSet=FbxNode::eSourcePivot, bool pApplyTarget=false, bool pForceEval=false);  
    FbxAMatrix& EvaluateLocalTransform(FbxTime pTime=FBXSDK_TIME_INFINITE, FbxNode::EPivotSet  
pPivotSet=FbxNode::eSourcePivot, bool pApplyTarget=false, bool pForceEval=false);  
    FbxVector4& EvaluateLocalTranslation(FbxTime pTime=FBXSDK_TIME_INFINITE, FbxNode::EPivotSet  
pPivotSet=FbxNode::eSourcePivot, bool pApplyTarget=false, bool pForceEval=false);  
    FbxVector4& EvaluateLocalRotation(FbxTime pTime=FBXSDK_TIME_INFINITE, FbxNode::EPivotSet  
pPivotSet=FbxNode::eSourcePivot, bool pApplyTarget=false, bool pForceEval=false);  
};
```

노드와 씬 그래프

- **FBX 노드(Nodes)**

- **포즈(Pose)**

- **바인드 포즈(Bind Pose)**

링크 변형(Link Deformation)에 포함된 모든 노드들의 변환 행렬을 포함
변형이 없을 때(바인딩 시점의) 노드들의 변환을 제공
행렬은 전역 좌표계(Global Coordinates)로 정의되는 것으로 가정

- **리셋 포즈(Reset Pose)**

- 노드 변환의 스냅샷(Snapshot)**

어떤 시점에 하나의 캐릭터의 모든 노드의 위치를 저장하기 위하여 사용
애니메이션 작업을 위한 참조 위치로 사용할 수 있음

```
class FbxPose : public FbxObject {  
public:  
    bool IsBindPose();  
    bool IsRestPose();  
    int GetCount();  
    FbxNameHandler GetNodeName(int nIndex);  
    FbxNode *GetNode(int nIndex);  
    FbxMatrix& GetMatrix(int nIndex);  
    bool IsLocalMatrix(int nIndex);  
    int Find(const FbxNode *pNode);  
    int Find(const FbxNameHandler &pnodeName, char pCompareWhat=eAllNameComponents);  
    static bool GetPosesContaining(FbxScene* pScene, FbxNode* pNode, PoseList& pPoseList, FbxArray<int>& pIndex);  
    static bool GetBindPoseContaining(FbxScene* pScene, FbxNode* pNode, PoseList& pPoseList,  
        FbxArray<int>& pIndex);  
};
```

```
struct FbxPoseInfo {  
    FbxMatrix mMatrix;  
    bool mMatrixIsLocal;  
    FbxNode* mNode;  
};  
  
typedef FbxArray<FbxNode*> NodeList;  
typedef FbxArray<FbxPose*> PoseList;  
typedef FbxArray<FbxPoseInfo*> PoseInfoList;
```

노드와 씬 그래프

• FBX 노드(Nodes)

- 변환 데이터(Transformation Data)

노드의 변환 데이터는 **부모에 상대적**인 노드의 (위치, 회전, 크기) 변환을 포함
변환 데이터는 일련의 FbxTypedProperty 객체로 표현됨

Lcl: Local

`FbxNode::LclTranslation, FbxNode::LclRotation, FbxNode::LclScaling`

```
FbxDouble3 translation = pfbxNode->LclTranslation.Get();  
FbxDouble3 rotation = pfbxNode->LclRotation.Get();  
FbxDouble3 scaling = pfbxNode->LclScaling.Get();
```

`FbxPropertyT<FbxDouble3> LclTranslation`

```
FbxTransform& FbxNode::GetTransform()  
FbxLimits& FbxNode::GetTranslationLimits()  
FbxLimits& FbxNode::GetRotationLimits()  
FbxLimits& FbxNode::GetScalingLimits()  
Pivots& FbxNode::GetPivots()
```

`template <class T> class FbxPropertyT : public FbxProperty`

`typedef FbxVectorTemplate3<FbxDouble> FbxDouble3
typedef FbxVectorTemplate4<FbxDouble> FbxDouble4`

`FbxConstraintPosition::SetConstrainedObject(FbxObject *pObject)`

변환 데이터는 FbxLimits 객체로 범위를 제한될 수 있음

`FbxNode::GetTranslationLimits(), GetRotationLimits(), GetScalingLimits()`

노드들은 FbxConstraint 객체를 사용하여 제약할 수 있음

씬의 전역 좌표계에 대한 노드의 변환은 변환 행렬로 표현될 수 있음

전역 변환 행렬은 `FbxNode::EvaluateGlobalTransform()` 함수로 접근할 수 있음

지역 변환 행렬은 `FbxNode::EvaluateLocalTransform()` 함수로 접근할 수 있음

- 노드 그룹핑(Grouping)

노드의 인스턴스들은 속성(FbxNodeAttribute)을 갖지 않고 존재할 수 있음

노드는 자식 노드들을 그룹핑하거나 위치를 설정하기 위하여 사용될 수 있음

노드와 씬 그래프

- **FBX 노드(Nodes)**
 - 변환 데이터(Transformation Data)

FbxVectorTemplate3<T>

```
template<class T> class FbxVectorTemplate3
{
public:
    inline FbxVectorTemplate3() { *this = T(0); }
    inline explicit FbxVectorTemplate3(T pValue) { *this = pValue; }
    inline FbxVectorTemplate3(T pData0, T pData1, T pData2) { mData[0] = pData0; mData[1] = pData1; ... }
    inline ~FbxVectorTemplate3() { }
    inline T& operator[](int nIndex) { return mData[nIndex]; }
    inline const T& operator[](int nIndex) const { return mData[nIndex]; }
    inline operator FbxVectorTemplate2<T>& () const { return *((FbxVectorTemplate2<T>*)this); }
    inline FbxVectorTemplate3<T>& operator=(T const &pValue) { mData[0] = pValue; ...; return *this; }
    inline FbxVectorTemplate3<T>& operator=(const FbxVectorTemplate2<T>& pVector) { ... }
    inline FbxVectorTemplate3<T>& operator=(const FbxVectorTemplate3<T>& pVector) { ... }
    inline bool operator==(const FbxVectorTemplate3<T>& pVector) const { ... }
    inline bool operator!=(const FbxVectorTemplate3<T>& pVector) const { return !operator==(pVector); }
    inline T* Buffer() { return mData; }
    inline const T* Buffer() const { return mData; }
};
```

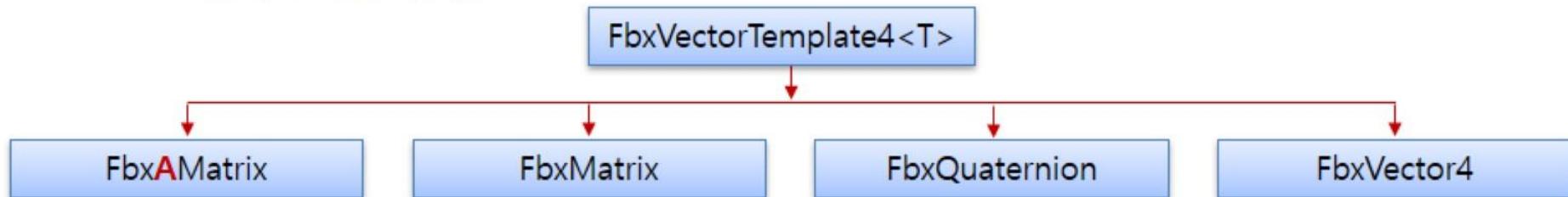
T **mData[3];**

typedef FbxVectorTemplate2<FbxDouble>	FbxDouble2
typedef FbxVectorTemplate3<FbxDouble>	FbxDouble3
typedef FbxVectorTemplate4<FbxDouble>	FbxDouble4
typedef FbxVectorTemplate4<FbxDouble4>	FbxDouble4x4

노드와 씬 그래프

- **FBX 벡터(Vector)와 행렬(Matrix)**

- 벡터와 행렬 클래스



FbxVector4

```
class FbxVector4 : public FbxDouble4 { ... }
```

```
FbxVector4 operator+(double pValue);  
FbxVector4 operator+(FbxVector4& pVector);  
FbxVector4 operator*(FbxVector4& pVector);  
double DotProduct(FbxVector4& pVector);  
FbxVector4 CrossProduct(FbxVector4& pVector);  
double Length();  
double Distance(FbxVector4& pVector);  
void Normalize();
```

FbxAMatrix

```
class FbxAMatrix : public FbxDouble4x4 { ... }
```

아핀 변환 행렬(Affine Transform Matrix)
행렬은 **열우선**으로 정의, 마지막 행은 평행이동을 표현

$$\begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{y} & 1 \end{bmatrix}$$

```
FbxVector4 GetT();  
FbxVector4 GetR();  
FbxQuaternion GetQ();  
FbxVector4 GetS();  
FbxVector4 GetRow(int pY);  
FbxVector4 GetColumn(int pX);  
FbxVector4 MultT(FbxVector4& pVector4); // t' = M * t  
FbxAMatrix operator*(FbxAMatrix& pOther);  
FbxAMatrix Transpose();  
FbxAMatrix Inverse();
```

FbxMatrix

```
class FbxMatrix : public FbxDouble4x4 { ... }
```

00 : (열 0, 행 0) 10: (열 1, 행 0) 23: (열 2, 행 3)
행렬은 **열우선**으로 정의

```
FbxAMatrix Slerp(FbxAMatrix& pOther, double pWeight);  
operator double*();
```

노드와 씬 그래프

- **변환 데이터(Transformation Data)**

- 위치, 회전, 크기 벡터

노드의 기본 SRT 데이터는 LclTranslation, LclRotation, LclScaling 속성으로 접근
노드의 실제 SRT 속성 데이터는 다음에 의존함

- 노드의 기본 SRT 속성(부모 노드에 상대적, 지역좌표계)
- 노드에 적용되는 범위(Limits, FbxNode::GetLimits())
- 노드에 적용되는 제약 조건(Constraints, FbxConstraint)
- 현재 애니메이션 스택의 애니메이션 커브(Animation Curves)

- 전역 및 지역 변환 행렬(Transformation Matrices)

노드의 전역(Global) 및 지역(Local) 변환 행렬

```
FbxAMatrix& mtxGlobalTransform = pfbxNode->EvaluateGlobalTransform();  
FbxAMatrix& mtxLocalTransform = pfbxNode->EvaluateLocalTransform();
```

```
FbxAnimEvaluator *pfbxEvaluator = pfbxScene->GetEvaluator();  
FbxAMatrix& mtxGlobalTransform = pfbxEvaluator->GetNodeGlobalTransform(pfbxNode);  
FbxAMatrix& mtxLocalTransform = pfbxEvaluator->GetNodeLocalTransform(pfbxNode);
```

```
FbxTime fbxTime;  
fbxTime.SetSecondDouble(2.0f);  
FbxAMatrix& mtxGlobalTransform = pfbxNode->EvaluateGlobalTransform(fbxTime);
```

애니메이션 노드의 변환 행렬

```
template<class T> class FbxVectorTemplate4  
typedef FbxVectorTemplate4<FbxDouble4> FbxDouble4x4  
class FbxAMatrix : public FbxDouble4x4
```

```
class FbxMatrix : public FbxDouble4x4
```

노드와 씬 그래프

- **변환 데이터(Transformation Data)**

- **기하학적 변환 속성(Geometric Transformation Properties)**

노드의 기하학적 변환 속성은 FbxNodeAttribute의 지역좌표계 오프셋(Offset)

FbxNode::GeometricTranslation, GeometricRotation, GeometricScaling

기하학적 변환은 노드의 지역(Local) 변환이 계산된 후에 노드의 속성에 적용됨

기하학적 변환은 노드의 계층 구조로 상속되지 않음

```
FbxPropertyT<FbxDouble3> GeometricTranslation;
```

```
FbxDouble3 GeometricTranslation.Get();
```

```
GeometricTranslation.Set(FbxDouble3);
```

```
enum EPivotSet { eSourcePivot, eDestinationPivot };
```

```
void FbxNode::SetGeometricTranslation(EPivotSet pPivotSet, FbxVector4 pVector);
```

```
FbxVector4 FbxNode::GetGeometricTranslation(EPivotSet pPivotSet) const;
```

```
void FbxNode::SetGeometricRotation(EPivotSet pPivotSet, FbxVector4 pVector);
```

```
FbxVector4 FbxNode::GetGeometricRotation(EPivotSet pPivotSet) const;
```

```
void FbxNode::SetGeometricScaling(EPivotSet pPivotSet, FbxVector4 pVector);
```

```
FbxVector4 FbxNode::GetGeometricScaling(EPivotSet pPivotSet) const;
```

기하학적 변환 속성은 FbxNodeAttribute의 지역좌표계와 노드의 지역좌표계의 관계(Offset, Pivot)를 표현

```
FbxAMatrix GetToAttributeNodeTransform(FbxNode *pfbxNode)
```

```
{
```

```
    FbxVector4 T = pfbxNode->GetGeometricTranslation(FbxNode::eSourcePivot);
```

```
    FbxVector4 R = pfbxNode->GetGeometricRotation(FbxNode::eSourcePivot);
```

```
    FbxVector4 S = pfbxNode->GetGeometricScaling(FbxNode::eSourcePivot);
```

```
    return(FbxAMatrix(T, R, S));
```

```
}
```

노드와 씬 그래프

• 변환 행렬의 계산

- FBX SDK와 Maya는 노드를 위한 변환 행렬 계산 공식이 같음

$$\text{WorldTransform} = \text{ParentWorldTransform} * T * R_{\text{off}} * R_p * R_{\text{pre}} * R * R_{\text{post}}^{-1} * R_p^{-1} * S_{\text{off}} * S_p * S * S_p^{-1}$$

- 3ds Max는 다른 공식을 사용(FBX 임포터/익스포터가 자동으로 변환 행렬을 변환)

$$\text{WorldTransform} = \text{ParentWorldTransform} * T * R * S * O_T * O_R * O_S$$

WorldTransform	노드의 변환 행렬	
ParentWorldTransform	부모 노드의 변환 행렬	
T	평행이동(Translation) 행렬	$O_T : \text{FbxNode::GeometricTranslation}$
R_{off}	회전 오프셋(Rotation Offset) 행렬	$O_R : \text{FbxNode::GeometricRotation}$
R_p	회전 피벗(Rotation Pivot) 행렬	$O_S : \text{FbxNode::GeometricScaling}$
R_{pre}	사전 회전(Pre-Rotation) 행렬	
R	회전(Rotation) 행렬, $R = R_x * R_y * R_z$	
R_{post}^{-1}	회전(Rotation) 후 행렬의 역행렬(Inverse of the post-rotation)	
R_p^{-1}	회전 피벗(Rotation Pivot) 행렬의 역행렬(Inverse of the rotation pivot)	
S_{off}	크기변환(Scaling) 오프셋(Offset) 행렬	
S_p	크기변환(Scaling) 피벗(Pivot) 행렬	
S	크기변환(Scaling) 행렬	
S_p^{-1}	크기변환(Scaling) 피벗의 역행렬(Inverse of the scaling pivot)	
O_T	기하학적 변환 평행이동(Geometric transform translation) 행렬	
O_R	기하학적 변환 회전(Geometric transform rotation) 행렬	
O_S	기하학적 변환 크기(Geometric transform scaling) 행렬	

노드와 씬 그래프

- **FBX 노드의 속성(Node Attributes)**

- 노드의 속성 생성

FbxNodeAttribute는 FbxNode와 함께 (위치/회전/크기)변환을 가진 씬 요소를 정의
노드의 속성은 `FbxNode::SetNodeAttribute()` 함수로 설정

노드의 속성은 `FbxNode::GetNodeAttribute()` 함수로 반환
노드의 속성이 설정되어 있지 않으면 NULL을 반환

`FbxNode *pfbxLightNode`

변환 데이터



노드 속성



`FbxLight *pfbxLight`

```
FbxNode *CreateLight(FbxScene *pfbxScene, char *pstrName)
```

```
{
```

```
    FbxLight *pfbxLight = FbxLight::Create(pfbxScene, pstrName);
    pfbxLight->LightType.Set(FbxLight::eSpot);
    pfbxLight->CastLight.Set(true);
```

```
FbxNode *pfbxLightNode = FbxNode::Create(pfbxScene, pstrName);
pfbxLightNode->SetNodeAttribute(pfbxLight);
```

```
return(pfbxLightNode);
```

```
enum EType { ePoint, eDirectional, eSpot, eArea, eVolume };
```

노드와 씬 그래프

- **FBX 노드의 속성(Node Attributes)**

- **노드의 속성(Node Attributes)**

씬의 원소들과 관련된 **FbxNodeAttribute** 파생 클래스

Camera	FbxCamera, FbxCameraStereo
Camera Switcher	FbxCameraSwitcher (MotionBuilder)
Light	FbxLight
Mesh	FbxMesh
Nurb	FbxNurbs, FbxNurbsCurve, FbxNurbsSurface, FbxTrimNurbsSurface
Patch / Parametric Surface	FbxPatch
Level of Detail Group	FbxLodGroup
Marker	FbxMarker
Skeleton	FbxSkeleton

- **노드 속성의 유형**

FbxNodeAttribute (NULL 값과는 다름)

FbxNodeAttribute의 유형(**FbxNodeAttribute::EType**)

FbxNodeAttribute::EType **FbxNodeAttribute::GetAttributeType()**

```
enum FbxNodeAttribute::EType {
    eUnknown, eNull, eMarker, eSkeleton, eMesh, eNurbs, ePatch, eCamera, eCameraStereo,
    eCameraSwitcher, eLight, eOpticalReference, eOpticalMarker, eNurbsCurve, eTrimNurbsSurface,
    eBoundary, eNurbsSurface, eShape, eLODGroup, eSubDiv, eCachedEffect, eLine
};
```

```
int FbxNodeAttribute::GetNodeCount();
FbxNode *FbxNodeAttribute::GetNode(int nIndex=0);
```

노드와 씬 그래프

- FBX 노드의 속성(Node Attributes)

```
void DisplayLight(FbxNode *pfbxNode) {
    FbxLight *pfbxLight = (FbxLight *)pfbxNode->GetNodeAttribute();
    DisplayString("Light Name: ", (char *)pfbxNode->GetName());
    ...
    char *pstrLightTypes[] = { "Point", "Directional", "Spot" };
    DisplayString(" Type: ", pstrLightTypes[pfbxLight->LightType.Get()]);
    DisplayBool(" Cast Light: ", pfbxLight->CastLight.Get());
    if (!(pfbxLight->FileName.Get().IsEmpty())) {
        DisplayString("File Name: \uacfc", pfbxLight->FileName.Get().Buffer(), "\uacfc");
        DisplayBool("Ground Projection: ", pfbxLight->DrawGroundProjection.Get());
        DisplayBool("Volumetric Projection: ", pfbxLight->DrawVolumetricLight.Get());
    }
    ...
}

void DisplayContent(FbxNode *pfbxNode) {
    FbxNodeAttribute::EType fbxAttributeType;
    if (pfbxNode->GetNodeAttribute() != NULL) {
        fbxAttributeType = pfbxNode->GetNodeAttribute()->GetAttributeType();
        if (fbxAttributeType == FbxNodeAttribute::eLight) DisplayLight(pfbxNode);
        ...
    }
    ...
    for (int i = 0; i < pfbxNode->GetChildCount(); i++) DisplayContent(pfbxNode->GetChild(i));
}
```

노드와 씬 그래프

- 조명(Lights)

- 조명의 생성

조명(FbxLight)은 씬에서 다른 객체처럼 생성

기본적으로 FbxLight는 노드의 음수 y-축 방향을 가짐

```
FbxNode *pfbxLightNode = FbxNode::Create(pfbxScene, "LightNode");
FbxLight *pfbxLight = FbxLight::Create(pfbxScene, "Light");
pfbxLightNode->SetNodeAttribute(pfbxLight);
FbxNode *pfbxRootNode = pfbxScene->Get rootNode();
pfbxRootNode->AddChild(pfbxLightNode);
```

씬의 주변(Ambient) 조명은 전역 설정으로 정의

`FbxGlobalSettings FbxScene::GetGlobalSettings()`

`FbxGlobalSettings::SetAmbientColor(FbxColor fbxAmbientColor)`

`FbxColor FbxGlobalSettings::SetAmbientColor()`

- 조명의 유형(Light Type)

조명의 유형은 `FbxLight::LightType` 속성으로 설정

```
pfbxLight->LightType.Set(FbxLight::eSpot);
pfbxLight->LightType.Set(FbxLight::ePoint);
pfbxLight->LightType.Set(FbxLight::eDirectional);
```

```
FbxPropertyT<EType> LightType;
FbxLight::EType {
    ePoint, eDirectional, eSpot, eArea, eVolume
};
```

<code>FbxLight::eSpot</code>	스팟 조명, <code>FbxLight::InnerAngle</code> , <code>FbxLight::OuterAngle</code>
<code>FbxLight::ePoint</code>	점 광원
<code>FbxLight::eDirectional</code>	방향성 광원

노드와 씬 그래프

• 조명(Lights)

- 조명의 방향

스팟 조명과 방향성 조명은 방향(특정 목표를 향하도록)을 가짐

`FbxNode::SetTarget(FbxNode *pfbxNode)`

FbxMarker 노드 속성이 목표 노드에 사용됨

```
FbxNode *pfbxTargetNode = FbxNode::Create(pfbxScene, "TargetNode");
FbxMarker *pfbxMarker = FbxMarker::Create(pfbxScene, "LightMarker");
pfbxTargetNode->SetNodeAttribute(pfbxMarker);
pfbxLightNode->SetTarget(pfbxTargetNode);
```

```
void SetTarget(FbxNode* pNode);
FbxNode* GetTarget() const;
```

FbxNode는 기본적으로 X-축의 양의 방향을 향함

조명은 기본적으로 노드의 Y-축의 음의 방향을 향하므로 조명 노드를 90도 회전

`void FbxNode::SetPostTargetRotation(FbxVector4 pVector)`

- 조명의 색상과 밝기(Color and Intensity)

조명의 색상은 `FbxLight::Color` 속성으로 정의(기본 색상 RGB(1.0, 1.0, 1.0))

조명의 밝기는 `FbxLight::Intensity` 속성으로 정의(기본 밝기 100.0)

```
pfbxLight->Color.Set(FbxDouble3(0.0, 1.0, 0.5));
pfbxLight->Intensity.Set(50.0);
```

FbxProperty 클래스

```
FbxPropertyT<T> & Set(const T &pValue)
T Get() const
FbxPropertyT<T> & operator=(const T &pValue)
operator T () const
```

```
FbxPropertyT<EType> LightType;
FbxPropertyT<FbxBool> CastLight;
FbxPropertyT<FbxDouble3> Color;
FbxPropertyT<FbxDouble> Intensity;
FbxPropertyT<FbxBool> CastShadows;
```

노드와 씬 그래프

• 조명(Lights)

- 조명의 감쇠(Decay, Attenuation)

조명의 감쇠는 `FbxLight::DecayType` 속성으로 정의

```
pfbxLight->DecayType.Set(FbxLight::eQuadratic);
```

```
FbxPropertyT<EDecayType> DecayType;  
FbxPropertyT<FbxDouble> DecayStart;
```

<code>FbxLight::eNone</code>	감쇠 없음
<code>FbxLight::eLinear</code>	선형 감쇠(조명까지의 거리에 따라 선형적으로 감쇠)
<code>FbxLight::eQuadratic</code>	제곱(Quadratic) 감쇠(조명까지의 거리의 제곱에 따라 감쇠)
<code>FbxLight::eCubic</code>	큐빅(Cubic) 감쇠(조명까지의 거리의 세제곱에 따라 감쇠)

```
FbxPropertyT<FbxBool> EnableNearAttenuation;  
FbxPropertyT<FbxDouble> NearAttenuationStart;  
FbxPropertyT<FbxDouble> NearAttenuationEnd;  
FbxPropertyT<FbxBool> EnableFarAttenuation;  
FbxPropertyT<FbxDouble> FarAttenuationStart;  
FbxPropertyT<FbxDouble> FarAttenuationEnd;
```

```
enum EDecayType {  
    eNone, eLinear, eQuadratic, eCubic  
};
```

```
FbxPropertyT<FbxBool> CastShadows;  
FbxPropertyT<FbxDouble3> ShadowColor;
```

- 그림자(Shadows)

그림자는 `FbxLight::CastShadows` 속성으로 활성화

그림자의 색상은 `FbxLight::ShadowColor` 속성으로 정의

기본 그림자 색상은 `RGB(0.0, 0.0, 0.0)`

그림자 텍스쳐는 `FbxLight::SetShadowTexture()` 함수를 사용하여 적용

```
void FbxLight::SetShadowTexture(FbxTexture *pfbxTexture);
```

노드와 씬 그래프

- 카메라(Cameras)

- 카메라 생성

카메라(FbxCamera)는 기본적으로 노드의 X-축의 양의 방향을 향함

```
FbxNode *pfbxCameraNode = FbxNode::Create(pfbxScene, "CameraNode");
FbxCamera *pfbxCamera = FbxCamera::Create(pfbxScene, "Camera");
pfbxCameraNode->SetNodeAttribute(pfbxCamera);
FbxNode *pfbxRootNode = pfbxScene->Get rootNode();
pfbxRootNode->AddChild(pfbxCameraNode);
```

카메라가 생성되면 씬의 기본 카메라로 설정될 수 있음

씬은 명시적으로 설정된 기본 카메라를 가져야 함

```
pfbxScene->GetGlobalSettings().SetDefaultCamera((char *)pfbxCamera->GetName());
```

- 카메라의 방향

카메라의 방향은 FbxNode::SetTarget() 함수를 사용하여 설정할 수 있음

```
FbxNode *pfbxTargetNode = FbxNode::Create(pfbxScene, "TargetNode");
FbxMarker *pfbxMarker = FbxMarker::Create(pfbxScene, "CameraMarker");
pfbxTargetNode->SetNodeAttribute(pfbxMarker);
pfbxCameraNode->SetTarget(pfbxTargetNode);
```

- 기본적으로 FbxCamera::FocusSource 속성은 카메라의 타겟으로 설정

```
pfbxCamera->FocusSource.Set(EFocusDistanceSource::eFocusSrcCameraInterest);
pfbxCamera->FocusSource.Set(EFocusDistanceSource::eFocusSpecificDistance);
pfbxCamera->FocusDistance.Set(100.0);
```

FbxPropertyT<EFocusDistanceSource> FocusSource;
FbxPropertyT<FbxDouble> FocusDistance;

메쉬, 재질, 텍스쳐

- **메쉬(Mesh), 재질(Material), 텍스쳐(Texture)**

- **기하(Geometry)**

FbxGeometry는 기하학적 객체의 기반(Base) 클래스

FbxMesh, FbxNurbs, FbxPatch, FbxLine, FbxBoundary

디포머(Deformer): 스킨(Skin) 디포머, 정점 캐시(Vertex Cache) 디포머, 형상 디포머

- **메쉬(Mesh)**

메쉬(**FbxMesh**)는 정점(Vertices, Control Points)과 레이어들의 집합으로 정의
하나의 **FbxMesh** 객체는 여러 개의 **FbxNode**에 연결될 수 있음(인스턴싱)

씬은 레이어(Layer)와 레이어 원소(**FbxLayerElement**)의 개념을 사용

레이어는 메쉬의 법선 벡터, 텍스쳐, 재질을 정의함

레이어 원소는 법선 맵(Normal Map), 재질 맵(Material Map), 텍스쳐 맵 등을 정의

- **재질(Material)**

재질은 씬에서 기하의 기본적인 렌더링 특성을 정의

확산(Diffuse)/주변(Ambient)/발광(Emissive) 색상 속성

재질(**FbxSurfaceLambert**, **FbxSurfacePhong**)은 **FbxNode**에 연결

int FbxNode::AddMaterial(FbxSurfaceMaterial *pfbxMaterial)

각 재질은 **FbxNode**에서 특정한 인덱스에 연결

새로 생성하는 다각형의 재질을 정의하기 위해 **FbxMesh::BeginPolygon()** 사용

void FbxMesh::BeginPolygon(int pMaterial, int pTexture, int pGroup);

void FbxMesh::AddPolygon(int nIndex, int pTextureUVIndex);

- **텍스쳐(Texture)**

텍스쳐(**FbxFileTexture**, **FbxLayeredTexture**, **FbxProceduralTexture**)는 재질에 연결됨
FbxFileTexture 클래스는 텍스쳐 값을 정의하기 위해 파일에 포함된 데이터를 사용

메쉬, 재질, 텍스쳐

• 레이어(Layer)

- 레이어 메카니즘(Layering Mechanism)

레이어(FbxLayer)는 하나 이상의 레이어 원소를 포함할 수 있음

레이어는 레이어 컨테이너에 포함됨

대부분 하나의 기하를 서술하기 위하여 하나의 레이어가 필요함

- 레이어 원소(Layer Element)

- 법선/접선/종법선 벡터(Normals/Tangents/Binormals)
- 재질(Materials)
- 다각형 그룹(Polygon Groups)
- 텍스쳐 매핑 좌표(Mapping UVs)
- 정점 색상(Vertex Colors)
- 스무딩 정보(Smoothing Informations)
- 정점 주름(Vertex Creases)
- 에지 주름(Edge Creases)
- 사용자 데이터(Custom User Data)
- 가시성(Visibilities)

```
FbxLayer *FbxLayerContainer::GetLayer(int nIndex);
int FbxLayerContainer::CreateLayer();
```

```
FbxMesh *pfbxMesh;
```

```
...
FbxLayer *pfbxLayer0 = pfbxMesh->GetLayer(0);
FbxLayerElementNormal *pfbxNormals = pfbxLayer0->GetNormals();
```

FbxLayerElement

FbxLayerElementNormal

FbxLayerElementTangent

FbxLayerElementBinormal

FbxLayerElementMaterial

FbxLayerElementPolygonGroup

FbxLayerElementUV

FbxLayerElementVertexColor

FbxLayerElementSmoothing

FbxLayerElementCrease

FbxLayerElementUserData

FbxLayerElementHole

FbxLayerElementVisibility

```
FbxLayerElementNormal *FbxLayer::GetNormals();
FbxLayerElementMaterial *FbxLayer::GetMaterials();
```

메쉬, 재질, 텍스쳐

• 레이어(Layer)

– FbxLayerElement 클래스

레이어 원소가 기하 표면에 매핑되는 방법과 매핑 정보가 메모리에 나열되는 방법

FbxLayerElement::EMappingMode(레이어 원소가 메쉬의 표면에 매핑되는 방법)

eNone	레이어 원소의 매핑이 결정되지 않음
eByControlPoint	표면(Surface)의 각 제어점(정점)마다 하나의 매핑 좌표가 있음
eByPolygonVertex	정점이 속한 각 다각형에 대하여 각 정점마다 하나의 매핑 좌표가 있음
eByPolygon	하나의 다각형마다 하나의 매핑 좌표가 있음
eByEdge	메쉬의 각 에지마다 하나의 매핑 좌표가 있음(스무딩 레이어 원소에 사용)
eAllSame	전체 표면에 대하여 하나의 매핑 좌표가 있음

FbxLayerElement::EReferenceMode(매핑 정보가 저장된 위치(좌표들의 배열)에 대한 참조 모드)

eDirect	n번째 원소의 매핑 정보는 FbxLayerElementTemplate::mDirectArray의 n번째에 있음
eIndex	호환성을 위해 제공, FbxLayerElement::eIndexToDirect로 대체
eIndexToDirect	FbxLayerElementTemplate::mIndexArray의 각 원소는 mDirectArray의 인덱스 eByPolygonVertex 매핑 모드의 좌표를 저장할 때 사용, 재질과 텍스쳐도 이 모드로 참조

```
const char *GetName();
EMappingMode FbxLayerElement::GetMappingMode();
EReferenceMode FbxLayerElement::GetReferenceMode();
```

```
static const char* const sTextureNames[];
static const char* const sTextureUVNames[];
static const char* const sNonTextureNames[];
static const FbxDataType sTextureDataTypes[];
static const char* const sTextureChannelNames[];
```

메쉬, 재질, 텍스쳐

- 레이어(Layer)

```
template<class T> class FbxLayerElementTemplate<T>
```

```
    FbxLayerElementArrayTemplate<T>& GetDirectArray();  
    FbxLayerElementArrayTemplate<int>& GetIndexArray();
```

```
class FbxLayerElementArray
```

```
    int GetCount();  
    bool GetAt(int nIndex, void** pItem, EFbxType pValueType);  
    bool GetFirst(void** pItem, EFbxType pValueType);  
    bool GetLast(void** pItem, EFbxType pValueType);  
    int Find(const void* pItem, EFbxType pValueType);  
    int FindAfter(int nAfterIndex, const void* pItem, EFbxType pValueType);  
    int FindBefore(int nBeforeIndex, const void* pItem, EFbxType pValueType);  
    template <class T> inline bool GetAt(int nIndex, T* pItem);
```

```
template<class T> class FbxLayerElementArrayTemplate<T>
```

```
    T GetAt(int nIndex);  
    T GetFirst();  
    T GetLast();  
    int Find(T const &pItem);  
    int FindAfter(int nAfterIndex, T const &pItem);  
    int FindBefore(int nBeforeIndex, T const &pItem);  
    T operator[](int nIndex);  
    FbxLayerElementArray& operator= (const FbxArray<T>& pArrayTemplate);  
    FbxLayerElementArrayTemplate<T>& operator= (FbxLayerElementArrayTemplate<T>& pArrayTemplate);
```

FbxLayerElement

FbxLayerElementTemplate

FbxLayerElementArray

FbxLayerElementArrayTemplate

메쉬, 재질, 텍스쳐

- **메쉬(Mesh)**

- **다각형(Polygon)**

메쉬: 다각형(Polygon)의 집합(기하: Geometry)

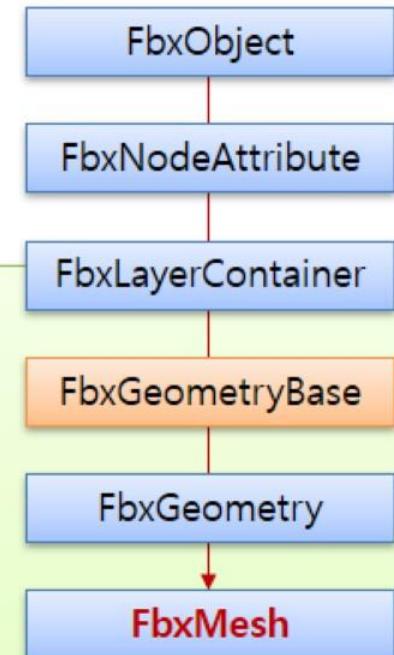
다각형: 정점들의 집합

하나의 메쉬에 여러 형태의 다각형이 혼합될 수 있음

FBX의 메쉬 관련 용어는 표준적인 용어와 다소 다름

- 제어점(Control Point): 정점(Vertex)
- 다각형 정점(Polygon Vertex): 제어점에 대한 인덱스(Index)
- 다각형(Polygon): 다각형 정점들의 집합(최소 3개)

```
class FbxGeometryBase : public FbxLayerContainer {  
    virtual int GetControlPointsCount(); //정점의 개수  
    virtual FbxVector4 GetControlPointAt(int nIndex);  
    virtual FbxVector4 *GetControlPoints(FbxStatus *pStatus=NULL); //정점 배열  
    int GetElementNormalCount();  
    FbxGeometryElementNormal *GetElementNormal(int nIndex=0);  
    int GetElementBinormalCount();  
    FbxGeometryElementBinormal *GetElementBinormal(int nIndex=0);  
    int GetElementTangentCount();  
    FbxGeometryElementTangent *GetElementTangent(int nIndex=0);  
    int GetElementMaterialCount();  
    FbxGeometryElementMaterial *GetElementMaterial(int nIndex=0);  
    int GetElementUVCount();  
    FbxGeometryElementUV *GetElementUV(int nIndex=0);  
    void GetUVSetNames(FbxStringList& pUVSetNameList);  
    FbxGeometryElementVertexColor *GetElementVertexColor(int nIndex=0);  
};
```



메쉬, 재질, 텍스쳐

- **메쉬(Mesh)**

- FbxGeometry 클래스는 다음과 같은 디포머(Deformer)를 지원
 - 스킨 디포머(Skin Deformation Deformer): FbxSkin
 - 정점 캐시 디포머(Vertex Cache Deformer): FbxVertexCacheDeformer
 - 블렌드 형상 디포머(Blend Shape Deformer): FbxBlendShape
 - 기하 가중치 맵(Geometry Weighted Map): FbxGeometryWeightedMap

```
class FbxGeometry : public FbxGeometryBase {  
    virtual FbxNodeAttribute::EType GetAttributeType();  
    int GetDeformerCount();  
    FbxDeformer* GetDeformer(int nIndex, FbxStatus* pStatus=NULL);  
    int GetDeformerCount(FbxDeformer::EDeformerType pType);  
    FbxDeformer* GetDeformer(int nIndex, FbxDeformer::EDeformerType pType, FbxStatus* pStatus=NULL);  
    FbxGeometryWeightedMap* GetSourceGeometryWeightedMap();  
    int GetDestinationGeometryWeightedMapCount();  
    FbxGeometryWeightedMap* GetDestinationGeometryWeightedMap(int nIndex);  
    int GetShapeCount();  
    int GetShapeCount(int nBlendShapeIndex, int nBlendShapeChannelIndex, FbxStatus* pStatus=NULL);  
    FbxShape* GetShape(int nBlendShapeIndex, int nBlendShapeChannelIndex, int nShapeIndex, ...);  
    const FbxShape* GetShape(int nBlendShapeIndex, int nBlendShapeChannelIndex, int nShapeIndex, ...);  
    FbxAnimCurve* GetShapeChannel(int nBlendShapeIndex, int nChannelIndex, FbxAnimLayer* pLayer, ...);  
    FbxAMatrix& GetPivot(FbxAMatrix& pXMatrix);  
    double GetDefaultShape(int nBlendShapeIndex, int nBlendShapeChannelIndex);  
    double GetDefaultShape(FbxBlendShapeChannel* pBlendShapeChannel);  
};
```

FbxGeometryBase
↓
FbxGeometry

```
enum FbxDeformer::EDeformerType { eUnknown, eSkin, eBlendShape, eVertexCache };
```

메쉬, 재질, 텍스쳐

- **메쉬(Mesh)**

```
class FbxMesh : public FbxGeometry {  
    int GetPolygonCount(); //다각형의 개수  
    int GetPolygonSize(int nPolygonIndex); //하나의 다각형의 정점의 개수  
    int GetPolygonGroup(int nPolygonIndex);  
    int GetPolygonVertex(int nPolygonIndex, int nPositionInPolygon); //하나의 다각형의 정점의 인덱스  
    bool GetPolygonVertexNormal(int nPolyIndex, int nVertexIndex, FbxVector4& pNormal);  
    bool GetPolygonVertexNormals(FbxArray<FbxVector4>& pNormals);  
    bool GetPolygonVertexUV(int nPolyIndex, int nVertexIndex, char* pUVSetName, FbxVector2& pUV, bool& bUnmapped);  
    bool GetPolygonVertexUVs(char* pUVSetName, FbxArray<FbxVector2>& pUVs, FbxArray<int>* bUnmappedUVId=NULL);  
    int GetPolygonVertexCount(); //메쉬의 인덱스 개수  
    int* GetPolygonVertices(); //메쉬의 인덱스 데이터  
    int GetPolygonVertexIndex(int nPolygonIndex);  
    int GetTextureUVCount(FbxLayerElement::EType pTypeIdentifier=FbxLayerElement::eTextureDiffuse);  
    int GetUVLayerCount();  
    FbxArray<FbxLayerElement::EType>  GetAllChannelUV(int nLayer);  
    int GetTextureUVIndex(int nPolygonIndex, int nPositionInPolygon, FbxLayerElement::EType pTypeIdentifier=FbxLayerElement::eTextureDiffuse);  
    bool GenerateNormals(bool bOverwrite=false, bool bByCtrlPoint=false, bool bCW=false);  
    bool GetTextureUV(FbxLayerElementArrayTemplate<FbxVector2>** pLockableArray,  
FbxLayerElement::EType pTypeIdentifier=FbxLayerElement::eTextureDiffuse);  
    bool GetMaterialIndices(FbxLayerElementArrayTemplate<int>** pLockableArray);  
    bool GetTextureIndices(FbxLayerElementArrayTemplate<int>** pLockableArray, FbxLayerElement::EType pTextureType);  
};
```

메쉬, 재질, 텍스쳐

- **메쉬(Mesh)**

- **메쉬 생성**

```
FbxNode *pfbxMeshNode = FbxNode::Create(pfbxScene, "MeshNode");
FbxMesh *pfbxMesh = FbxMesh::Create(pfbxScene, "Mesh");
pfbxMeshNode->SetNodeAttribute(pfbxMesh);
FbxNode *pfbxRootNode = pfbxScene->Get rootNode();
pfbxRootNode->AddChild(pfbxMeshNode);
```

- **정점 정의**

FBX SDK의 객체는 **오른손 좌표계(Y-UP)**로 생성(정점은 이에 맞게 정의되어야 함)
씬의 축 변환은 메쉬의 정점에 영향을 주지 않음

큐브(Cube), 100x100x100, 6개의 면, 한 면에 4개의 정점, 전체 24개의 정점

```
pfbxMesh->InitControlPoints(24);
```

```
FbxVector4 *pfbxControlPoints = pfbxMesh->GetControlPoints();
```

```
pfbxControlPoints[0] = FbxVector4(-50, 0, 50); pfbxControlPoints[1] = FbxVector4(50, 0, 50);
pfbxControlPoints[2] = FbxVector4(50, 100, 50); pfbxControlPoints[3] = FbxVector4(-50, 100, 50);
```

```
pfbxControlPoints[4] = FbxVector4(50, 0, 50); pfbxControlPoints[5] = FbxVector4(50, 0, -50);
pfbxControlPoints[6] = FbxVector4(50, 100, -50); pfbxControlPoints[7] = FbxVector4(50, 100, 50);
```

```
pfbxControlPoints[8] = FbxVector4(50, 0, -50); pfbxControlPoints[9] = FbxVector4(-50, 0, -50);
pfbxControlPoints[10] = FbxVector4(-50, 100, -50); pfbxControlPoints[11] = FbxVector4(50, 100, -50);
```

```
pfbxControlPoints[12] = FbxVector4(50, 0, 50); pfbxControlPoints[13] = FbxVector4(-50, 0, 50);
pfbxControlPoints[14] = FbxVector4(-50, 100, 50); pfbxControlPoints[15] = FbxVector4(-50, 100, -50);
```



메쉬, 재질, 텍스쳐

- **메쉬(Mesh)**

- 큐브 생성하기

```
FbxMesh *pfbxMesh = FbxMesh::Create(pfbxScene, "");  
pfbxMesh->InitControlPoints(8);  
FbxVector4 *pfbxvVertices = pfbxMesh->GetControlPoints();  
memcpy((void *)pfbxvVertices, (void *)pControlPoints, 8 * sizeof(FbxVector4));  
FbxGeometryElementMaterial *pfbxMaterialElement = pfbxMesh->CreateElementMaterial();  
pfbxMaterialElement->SetMappingMode(FbxGeometryElement::eAllSame);  
pfbxMaterialElement->SetReferenceMode(FbxGeometryElement::eIndexToDirect);  
pfbxMaterialElement->GetIndexArray().Add(0);  
for (int f = 0, i = 0; f < 6; f++) {  
    pfbxMesh->BeginPolygon();  
    for (int v = 0; v < 4; v++) pfbxMesh->AddPolygon(pIndices[i++]);  
    pfbxMesh->EndPolygon();  
}  
FbxGeometryElementNormal *pfbxNormalElement = pfbxMesh->CreateElementNormal();  
pfbxNormalElement->SetMappingMode(FbxGeometryElement::eByControlPoint);  
pfbxNormalElement->SetReferenceMode(FbxGeometryElement::eDirect);  
for (int n = 0; n < 8; n++)  
    pfbxNormalElement->GetDirectArray().Add(FbxVector4(pNormals[n][0], pNormals[n][1], pNormals[n][2]));  
FbxGeometryElementUV *pfbxUVElement = pfbxMesh->CreateElementUV("UVSet1");  
pfbxUVElement->SetMappingMode(FbxGeometryElement::eByPolygonVertex);  
pfbxUVElement->SetReferenceMode(FbxGeometryElement::eIndexToDirect);  
for (int i = 0; i < 4; i++) pfbxUVElement->GetDirectArray().Add(FbxVector2(pUVs[i][0], pUVs[i][1]));  
for (int i = 0; i < 24; i++) pfbxUVElement->GetIndexArray().Add(pUVIndices[i % 4]);  
pfbxNode->SetNodeAttribute(pfbxMesh);
```

FbxNode *pfbxNode;

typedef double Vector4[4];
typedef double Vector2[2];

int pIndices[24] = { ... };
Vector4 pControlPoints[8] = { { ... }, ... };
Vector4 pNormals[8] = { { ... }, ... };
Vector2 pUVs[14] = { { ... }, ... };
int pUVIndices[24] = { ... };

메쉬, 재질, 텍스쳐

• 메쉬(Mesh)

```
void DisplayControlsPoints(FbxMesh *pfbxMesh) {
    int nControlPoints = pfbxMesh->GetControlPointsCount();
    int nElementNormals = pfbxMesh->GetElementNormalCount();
    FbxVector4 *pfbxvControlPoints = pfbxMesh->GetControlPoints();
    for (int i = 0; i < nControlPoints; i++) {
        FbxVector4 fbxVector4 = pfbxvControlPoints[i]; //fbxVector4 = GetControlPointAt(i);
        for (int j = 0; j < nElementNormals; j++) {
            FbxGeometryElementNormal *pfbxeNormals = pfbxMesh->GetElementNormal(j);
            if (pfbxeNormals->GetMappingMode() == FbxGeometryElement::eByControlPoint) {
                if (pfbxeNormals->GetReferenceMode() == FbxGeometryElement::eDirect) {
                    fbxVector4 = pfbxeNormals->GetDirectArray().GetAt(i);
                }
            }
        }
    }
    int nPolygons = pfbxMesh->GetPolygonCount();
    for (int i = 0, nIndex = 0; i < nPolygons; i++) {
        int nPolygonSize = pfbxMesh->GetPolygonSize(i);
        for (int j = 0; j < nPolygonSize; j++) {
            int nControlPointIndex = pfbxMesh->GetPolygonVertex(i, j);
            fbxVector4 = pfbxvControlPoints[nControlPointIndex];
        }
    }
}
```

int FbxGeometryBase::**GetControlPointsCount()**;
int FbxMesh::**GetPolygonVertexCount()**;

typedef FbxLayerElementNormal **FbxGeometryElementNormal**;
class FbxLayerElementNormal : public **FbxLayerElementTemplate<FbxVector4>**
FbxGeometryElementNormal *FbxGeometryBase::**GetElementNormal(int nIndex=0)**;

int FbxMesh::**GetPolygonCount()**;
int FbxMesh::**GetPolygonSize(int nPolygonIndex)**;

FbxVector4 *FbxGeometryBase::**GetControlPoints(FbxStatus *pStatus=NULL)**;
FbxVector4 FbxGeometryBase::**GetControlPointAt(int nIndex)**;
int FbxMesh::**GetPolygonVertex(int nPolygonIndex, int nPositionInPolygon)**;
int *FbxMesh::**GetPolygonVertices()**;

메쉬, 재질, 텍스쳐

- 메쉬(Mesh)

```
class FbxDeformer : public FbxObject {  
    enum EDeformerType { eUnknown, eSkin, eBlendShape, eVertexCache };  
    virtual EDeformerType GetDeformerType() { return eUnknown; }  
    void SetMultiLayer(bool pMultiLayer);  
    bool GetMultiLayer() const;  
};  
class FbxBlendShape : public FbxDeformer {  
    bool SetGeometry(FbxGeometry* pGeo);  
    FbxGeometry* GetGeometry();  
    bool AddBlendShapeChannel(FbxBlendShapeChannel* RemoveBlendShapeChannel());  
    int GetBlendShapeChannelCount() const;  
    FbxBlendShapeChannel* GetBlendShapeChannel();  
    const FbxBlendShapeChannel* GetBlendShapeChannel() const;  
    EDeformerType GetDeformerType() const;  
    void Reset();  
};  
class FbxVertexCacheDeformer : public FbxDeformer {  
    void SetCache(FbxCache* pCache);  
    FbxCache* GetCache();  
    void SetCacheChannel(char* pName);  
    FbxString GetCacheChannel();  
    void SetActive(bool pValue);  
    bool IsActive();  
    virtual EDeformerType GetDeformerType() { return FbxDeformer::eVertexCache; }  
};  
class FbxSkin : public FbxDeformer {  
    enum EType { eRigid, eLinear, eDualQuaternion, eBlend };  
    void SetSkinningType(EType pType);  
    EType GetSkinningType();  
    void SetDeformAccuracy(double pDeformAccuracy);  
    double GetDeformAccuracy();  
    bool SetGeometry(FbxGeometry* pGeometry);  
    FbxGeometry* GetGeometry();  
    bool AddCluster(FbxCluster* pCluster);  
    FbxCluster* RemoveCluster(FbxCluster* pCluster);  
    int GetClusterCount();  
    FbxCluster* GetCluster(int nIndex);  
    const FbxCluster* GetCluster(int nIndex);  
    EDeformerType GetDeformerType() { return eSkin; }  
    void AddControlPointIndex(int nIndex);  
    int GetControlPointIndicesCount();  
    int* GetControlPointIndices();  
    double* GetControlPointBlendWeights();  
    void SetControlPointWCount(int nCount);  
};
```

메쉬, 재질, 텍스쳐

- **메쉬(Mesh)**

- 법선 벡터 설정

메쉬의 법선 벡터는 FbxLayerElementNormal 클래스로 정의

```
FbxVector4 fbxvNormalXPos(1, 0, 0), fbxvNormalXNeg(-1, 0, 0), fbxvNormalYPos(0, 1, 0);
FbxVector4 fbxvNormalYNeg(0, -1, 0), fbxvNormalZPos(0, 0, 1), fbxvNormalZNeg(0, 0, -1);
FbxLayer *pfbxLayer = pfbxMesh->GetLayer(0);
if (pfbxLayer == NULL) {
    pfbxMesh->CreateLayer();
    pfbxLayer = pfbxMesh->GetLayer(0);
}
FbxLayerElementNormal *pfbxLayerElementNormal= FbxLayerElementNormal::Create(pfbxMesh, "");
pfbxLayerElementNormal->SetMappingMode(FbxLayerElement::eByControlPoint);
pfbxLayerElementNormal->SetReferenceMode(FbxLayerElement::eDirect);
pfbxLayerElementNormal->GetDirectArray().Add(fbxvNormalZPos);
pfbxLayerElementNormal->GetDirectArray().Add(fbxvNormalZPos);
pfbxLayerElementNormal->GetDirectArray().Add(fbxvNormalZPos);
pfbxLayerElementNormal->GetDirectArray().Add(fbxvNormalZPos);
pfbxLayerElementNormal->GetDirectArray().Add(fbxvNormalXPos);
pfbxLayerElementNormal->GetDirectArray().Add(fbxvNormalXPos);
pfbxLayerElementNormal->GetDirectArray().Add(fbxvNormalXPos);
pfbxLayerElementNormal->GetDirectArray().Add(fbxvNormalXPos);
...
pfbxLayer->SetNormals(pfbxLayerElementNormal);
```

```
FbxLayer *FbxLayerContainer::GetLayer(int nIndex);
int FbxLayerContainer::CreateLayer();
```

```
FbxLayerElementArrayTemplate<Type>& FbxLayerElementTemplate::GetDirectArray();
FbxLayerElementArrayTemplate<int>& FbxLayerElementTemplate::GetIndexArray();
void FbxLayer::SetNormals(FbxLayerElementNormal *pNormals);
```

메쉬, 재질, 텍스쳐

- **메쉬(Mesh)**

- **인스턴싱(Instancing)**

하나의 메쉬(FbxMesh) 객체(인스턴스)가 여러 FbxNode 객체(인스턴스)에 연결
하나의 속성(FbxNodeAttribute) 객체를 공유함으로써 메모리를 절약할 수 있음

```
FbxNode *CreateCubeInstance(FbxScene *pfbxScene, char *pstrName, FbxMesh *pfbxCube) {  
    FbxNode *pfbxNode = FbxNode::Create(pfbxScene, pstrName);  
    pfbxNode->SetNodeAttribute(pfbxCube);  
    pfbxNode->LclScaling.Set(FbxVector4(0.3, 0.3, 0.3));  
    pfbxScene->GetRootNode()->AddChild(pfbxNode);  
    return(pfbxNode);  
}
```

```
FbxNodeAttribute* FbxNode::SetNodeAttribute(FbxNodeAttribute *pNodeAttribute);  
FbxNodeAttribute* FbxNode::GetNodeAttributeByIndex(int nIndex);
```

```
FbxMesh *CreateCube(FbxScene *pfbxScene, char *pstrName) {  
    FbxMesh *pfbxMesh = FbxMesh::Create(pfbxScene, pstrName);  
    pfbxMesh->InitControlPoints(24);  
    ...  
    return(pfbxMesh);  
}
```

```
FbxMesh *pfbxCubeMesh = CreateCube(pfbxScene, "Cube");  
FbxNode *pfbxCubeNode01 = CreateCubeInstance(pfbxScene, "Cube Node01", pfbxCubeMesh);  
FbxNode *pfbxCubeNode02 = CreateCubeInstance(pfbxScene, "Cube Node02", pfbxCubeMesh);  
FbxNode *pfbxCubeNode03 = CreateCubeInstance(pfbxScene, "Cube Node03", pfbxCubeMesh);
```

메쉬, 재질, 텍스쳐

- 재질(Material)
 - 재질 클래스

FbxSurfaceMaterial

```
static const char* sShadingModel;
static const char* sMultiLayer;
static const char* sEmissive;
static const char* sEmissiveFactor;
static const char* sAmbient;
static const char* sAmbientFactor;
static const char* sDiffuse;
static const char* sDiffuseFactor;
static const char* sSpecular;
static const char* sSpecularFactor;
static const char* sShininess;
static const char* sBump;
static const char* sNormalMap;
static const char* sBumpFactor;
static const char* sTransparentColor;
static const char* sTransparencyFactor;
static const char* sReflection;
static const char* sReflectionFactor;
static const char* sDisplacementColor;
static const char* sDisplacementFactor;
static const char* sVectorDisplacementColor;
static const char* sVectorDisplacementFactor;
```

FbxSurfaceLambert

```
FbxPropertyT<FbxDouble3> Emissive;
FbxPropertyT<FbxDouble> EmissiveFactor;
FbxPropertyT<FbxDouble3> Ambient;
FbxPropertyT<FbxDouble> AmbientFactor;
FbxPropertyT<FbxDouble3> Diffuse;
FbxPropertyT<FbxDouble> DiffuseFactor;
FbxPropertyT<FbxDouble3> NormalMap;
FbxPropertyT<FbxDouble3> Bump;
FbxPropertyT<FbxDouble> BumpFactor;
FbxPropertyT<FbxDouble3> TransparentColor;
FbxPropertyT<FbxDouble> TransparencyFactor;
FbxPropertyT<FbxDouble3> DisplacementColor;
FbxPropertyT<FbxDouble> DisplacementFactor;
FbxPropertyT<FbxDouble3> VectorDisplacementColor;
FbxPropertyT<FbxDouble> VectorDisplacementFactor;
```

```
static const FbxDouble3 sDiffuseDefault;
```

FbxObject

FbxSurfaceMaterial

FbxSurfaceLambert

FbxSurfacePhong

FbxSurfacePhong

```
FbxPropertyT<FbxDouble3> Specular;
FbxPropertyT<FbxDouble> SpecularFactor;
FbxPropertyT<FbxDouble> Shininess;
FbxPropertyT<FbxDouble3> Reflection;
FbxPropertyT<FbxDouble> ReflectionFactor;
```

메쉬, 재질, 텍스쳐

- 재질(Material)

- 재질 클래스

```
int FbxNode::GetMaterialCount();
FbxSurfaceMaterial *FbxNode::GetMaterial(int nIndex);
```

```
FbxNode *pfbxNode = ...;
int nMaterials = pfbxNode->GetMaterialCount();
int nSurfaceMaterials = pfbxNode->GetSrcObjectCount<FbxSurfaceMaterial>();
FbxSurfaceMaterial *pfbxMaterial;
FbxLayerElementMaterial *pfbxLayerElement;
if (pfbxLayerElement->GetMappingMode() == FbxLayerElement::eAllSame) {
    int nIndex = pfbxLayerElement->GetIndexArray()[0];
    pfbxMaterial = pfbxNode->GetMaterial(nIndex);
}
```

FbxLayerElement

FbxLayerElementTemplate<T>

```
template<class T> class FbxLayerElementTemplate<T>
```

FbxLayerElementTemplate<FbxSurfaceMaterial *>

```
FbxLayerElementArrayTemplate<T>& GetDirectArray();
FbxLayerElementArrayTemplate<int>& GetIndexArray();
bool operator== (const FbxLayerElementTemplate& pOther);
FbxLayerElementTemplate& operator= (FbxLayerElementTemplate const &pOther);
int RemapIndexTo (FbxLayerElement::EMappingMode pNewMapping);
```

FbxLayerElementMaterial

```
FbxGeometryElementMaterial *FbxMesh::GetElementMaterial(int nIndex=0);
```

typedef FbxLayerElementMaterial FbxGeometryElementMaterial;

메쉬, 재질, 텍스쳐

- 재질(Material)

- 재질 생성

```
void CreateMaterials(FbxScene *pfbxScene, FbxMesh *pfbxMesh)
```

```
{  
    for (int i = 0; i < 5; i++)  
    {
```

```
        FbxString fbxstrMaterialName = "Material";
```

```
        fbxstrMaterialName += i;
```

```
        FbxDouble3 fbxBlack(0.0, 0.0, 0.0), fbxRed(1.0, 0.0, 0.0);
```

```
        FbxSurfacePhong *pfbxMaterial = FbxSurfacePhong::Create(pfbxScene, fbxstrMaterialName.Buffer());
```

```
        pfbxMaterial->Emissive.Set(fbxBlack);
```

```
        pfbxMaterial->Ambient.Set(fbxRed);
```

```
        FbxDouble3 fbxColor = FbxDouble3((i > 2) ? 1.0 : 0.0, (i > 0 && i < 4) ? 1.0 : 0.0, (i % 2) ? 0.0 : 1.0);
```

```
        pfbxMaterial->Diffuse.Set(fbxColor);
```

```
        pfbxMaterial->TransparencyFactor.Set(0.0);
```

```
        pfbxMaterial->Shininess.Set(0.5);
```

```
        FbxString fbxstrShadingName = "Phong";
```

```
        pfbxMaterial->ShadingModel.Set(fbxstrShadingName);
```

```
        FbxNode *pfbxNode = pfbxMesh->GetNode();
```

```
        if (pfbxNode) pfbxNode->AddMaterial(pfbxMaterial);
```

```
}
```

```
ExportScene03/main.cxx
```

```
int FbxNode::AddMaterial(FbxSurfaceMaterial *pMaterial);
```

```
int FbxNodeAttribute::GetNodeCount();
```

```
FbxNode *FbxNodeAttribute::GetNode(int nIndex=0);
```

```
FbxSurfaceMaterial *FbxNode::GetMaterial(int nIndex);
```

메쉬, 재질, 텍스쳐

- 재질(Material)

- 재질 생성

```
void DisplayMaterial(FbxGeometry *pfbxGeometry) {  
    int nMaterials = 0;  
    FbxNode *pfbxNode = pfbxNode = pfbxGeometry->GetNode();  
    if (pfbxNode) nMaterials = pfbxNode->GetMaterialCount();  
    if (nMaterials > 0) {  
        FbxPropertyT<FbxDouble3> fbxpDouble3;  
        FbxPropertyT<FbxDouble> fbxpDouble;  
        for (int i = 0; i < nMaterials; i++) {  
            FbxSurfaceMaterial *pfbxMaterial = pfbxNode->GetMaterial(i);  
            FbxImplementation *pfbxImpl = GetImplementation(pfbxMaterial, FBXSDK_IMPLEMENTATION_HLSL);  
            FbxString fbxstrImplementationType = "HLSL";  
            if (pfbxImpl) { ... }  
            else if (pfbxMaterial->GetClassId().Is(FbxSurfacePhong::ClassId)) {  
                fbxpDouble3 = ((FbxSurfacePhong *)pfbxMaterial)->Diffuse;  
                fbxpDouble3 = ((FbxSurfacePhong *)pfbxMaterial)->Specular;  
                fbxpDouble = ((FbxSurfacePhong *)pfbxMaterial)->Shininess;  
            }  
            else if (pfbxMaterial->GetClassId().Is(FbxSurfaceLambert::ClassId)) {  
                fbxpDouble3 = ((FbxSurfaceLambert *)pfbxMaterial)->Diffuse;  
                fbxpDouble3 = ((FbxSurfaceLambert *)pfbxMaterial)->Emissive;  
            }  
        }  
    }  
}
```

```
int FbxNode::GetMaterialCount();  
FbxSurfaceMaterial *FbxNode::GetMaterial(int nIndex);
```

메쉬, 재질, 텍스쳐

- **하드웨어 쉐이더(Hardware Shader)**

- FBX SDK는 CGFX와 DirectX 하드웨어 쉐이더를 지원
CGFX 또는 DirectX 구현(Implementation)으로 재질을 설정할 수 있음
- **FbxImplementation** 클래스
쉐이딩 노드를 구현(쉐이더와 바인딩 테이블(FbxBindingTable)을 정의)
- **FbxBindingTable**
바인딩 테이블은 바인딩들의 집합
바인딩은 FbxObject 객체와 외부 객체(HLSL 쉐이더 파라메터)의 연결을 표현

```
FbxImplementation *pfbxImplement = FbxImplementation::Create(pfbxMyScene, "MyImplementation" );
pMyObject.AddImplementation(pfbxImplement);
pMyObject.SetDefaultImplementation(pfbxImplement);
pfbxImplement->RenderAPI = FBXSDK_RENDERING_API_DIRECTX;
pfbxImplement->RenderAPIVersion = "9.0";
pfbxImplement->Language = FBXSDK_SHADING_LANGUAGE_HSL;
```

```
FbxBindingTable *FbxImplementation::GetTableByTargetName(char *pName);
FbxBindingTable *FbxImplementation::GetRootTable();
FbxBindingTable *FbxImplementation::GetTable(int nIndex);
```

FbxBindingTable

FbxImplementation

```
FbxPropertyT<FbxString> Language;
FbxPropertyT<FbxString> LanguageVersion;
FbxPropertyT<FbxString> RenderAPI;
FbxPropertyT<FbxString> RenderAPIVersion;
FbxPropertyT<FbxString> RootBindingName;
```

```
FbxPropertyT<FbxString> TargetName;
FbxPropertyT<FbxString> TargetType;
FbxPropertyT<FbxString> DescRelativeURL;
FbxPropertyT<FbxString> DescAbsoluteURL;
FbxPropertyT<FbxString> DescTAG;
FbxPropertyT<FbxString> CodeRelativeURL;
FbxPropertyT<FbxString> CodeAbsoluteURL;
FbxPropertyT<FbxString> CodeTAG;
```

메쉬, 재질, 텍스쳐

- 하드웨어 쉐이더(Hardware Shader)

```
FbxSurfaceMaterial *CreateHlslShader(FbxScene *pfbxScene) {  
    FbxSurfaceMaterial *pfbxHlslMaterial = FbxSurfaceMaterial::Create(pfbxScene, "HlslShader" );  
    FbxImplementation *pfbxImplement = FbxImplementation::Create(pfbxScene, FbxString("HLSL"));  
    pfbxHlslMaterial->AddImplementation(pfbxImplement);  
    pfbxHlslMaterial->SetDefaultImplementation(pfbxImplement);  
    pfbxImplement->RenderAPI = FBXSDK_RENDERING_API_DIRECTX;  
    pfbxImplement->RenderAPIVersion = "9.0";  
    pfbxImplement->Language = FBXSDK_SHADING_LANGUAGE_HLSL;  
    pfbxImplement->LanguageVersion = "1.0";  
    pfbxImplement->RootBindingName = "root";  
    FbxBindingTable *pfbxTable = pfbxImplement->AddNewTable("root", "shader");  
    pfbxTable->DescAbsoluteURL = HLSL_SHADERFILE;  
    pfbxTable->DescTAG = "dx9";  
    FbxProperty fbxProp = FbxProperty::Create(pfbxHlslMaterial, FbxDouble3DT, "cubeMap", "cubeMap");  
    fbxProp.ModifyFlag(FbxPropertyAttr::eUserDefined, true);  
    FbxDouble3 fbxvMapVal(0, 1, 0);  
    fbxProp.Set(fbxvMapVal);  
    CreateTableEntry(pfbxTable, fbxProp);  
    FbxFileTexture *pfbxTexture = FbxFileTexture::Create(pfbxScene, "cubeMapTex");  
    pfbxTexture->SetFileName(TEXTURE_FILE);  
    pfbxTexture->SetTextureUse(FbxTexture::eStandard);  
    pfbxTexture->SetMappingType(FbxTexture::eUV);  
    pfbxTexture->ConnectDstProperty(fbxProp);  
    return(pfbxHlslMaterial);  
}  
FbxBindingTable *FbxImplementation::AddNewTable(char* pTargetName, char* pTargetType);
```

메쉬, 재질, 텍스쳐

- 텍스쳐

```
void DisplayTextureNames(FbxProperty &fbxProperty) {  
    int nLayeredTextures = fbxProperty.GetSrcObjectCount<FbxLayeredTexture>();  
    if (nLayeredTextures > 0) {  
        for (int j = 0; j < nLayeredTextures; j++) {  
            FbxLayeredTexture *pfbxLayeredTexture = fbxProperty.GetSrcObject<FbxLayeredTexture>(j);  
            int nTextures = pfbxLayeredTexture->GetSrcObjectCount<FbxTexture>();  
            for (int k = 0; k < nTextures; k++) {  
                char *pstrLayeredTextureName = (char *)pfbxLayeredTexture->GetName();  
            }  
        }  
    }  
    else {  
        int nTextures = fbxProperty.GetSrcObjectCount<FbxTexture>();  
        if (nTextures > 0) {  
            for (int j = 0; j < nTextures; j++) {  
                FbxTexture *pfbxTexture = fbxProperty.GetSrcObject<FbxTexture>(j);  
                if (pfbxTexture) {  
                    char *pstrTextureName = (char *)pfbxTexture->GetName();  
                }  
            }  
        }  
        DisplayTextureNames(pfbxMaterial->FindProperty(FbxSurfaceMaterial::sDiffuse));  
        DisplayTextureNames(pfbxMaterial->FindProperty(FbxSurfaceMaterial::sEmissive));  
        DisplayTextureNames(pfbxMaterial->FindProperty(FbxSurfaceMaterial::sAmbient));  
        DisplayTextureNames(pfbxMaterial->FindProperty(FbxSurfaceMaterial::sSpecular));  
        DisplayTextureNames(pfbxMaterial->FindProperty(FbxSurfaceMaterial::sBump));  
        DisplayTextureNames(pfbxMaterial->FindProperty(FbxSurfaceMaterial::sNormalMap));  
    }  
}
```

메쉬, 재질, 텍스쳐

- **텍스쳐(Texture)**

- 텍스쳐 클래스

- FbxTexture, FbxFileTexture, FbxLayeredTexture, FbxProceduralTexture

- 텍스쳐는 FbxSurfaceMaterial 파생 클래스의 속성에 연결됨

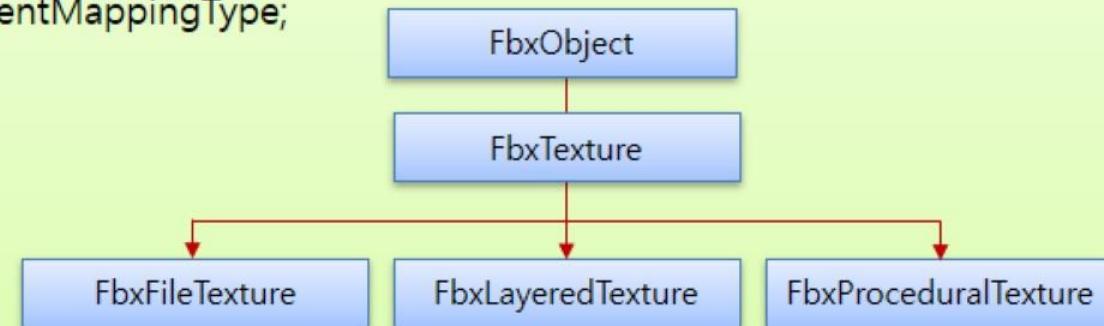
FbxTexture

```
enum EWrapMode { eRepeat, eClamp };
enum ECoordinates { eU, eV, eW };
enum EAlignMode { eLeft, eRight, eTop, eBottom };
enum EAlphaSource { eNone, eRGBIntensity, eBlack };
enum EBlendMode { eTranslucent, eAdditive, eModulate, eModulate2, eOver };
enum EMappingType { eNull, ePlanar, eSpherical, eCylindrical, eBox, eFace, eUV, eEnvironment };
enum ETextureUse { eStandard, eShadowMap, eLightMap, eBumpNormalMap, ... };
```

```
FbxPropertyT<FbxDouble> Alpha;
FbxPropertyT<EUnifiedMappingType> CurrentMappingType;
FbxPropertyT<EWrapMode> WrapModeU;
FbxPropertyT<EWrapMode> WrapModeV;
FbxPropertyT<FbxBool> UVSwap;
FbxPropertyT<FbxBool> PremultiplyAlpha;
FbxPropertyT<FbxDouble3> Translation;
FbxPropertyT<FbxDouble3> Rotation;
FbxPropertyT<FbxDouble3> Scaling;
FbxPropertyT<FbxDouble3> RotationPivot;
FbxPropertyT<FbxDouble3> ScalingPivot;
FbxPropertyT<EBlendMode> CurrentTextureBlendMode;
```

FbxFileTexture

```
FbxPropertyT<FbxBool> UseMaterial;
FbxPropertyT<FbxBool> UseMipMap;
enum EMaterialUse { eModelMaterial, eDefaultMaterial };
```



메쉬, 재질, 텍스쳐

- **텍스쳐(Texture)**

- 파일에서 텍스쳐 생성

```
FbxSurfacePhong *pfbxPhongMaterial = NULL;
FbxNode *pfbxNode = pfbxMesh->GetNode();
if (pfbxNode) pfbxPhongMaterial = pfbxNode->GetSrcObject<FbxSurfacePhong>(0);
if (!pfbxPhongMaterial) pfbxPhongMaterial = CreatePhongMaterial(pfbxScene, pfbxNode, "DiffuseMaterial");
pfbxNode->AddMaterial(pfbxPhongMaterial);
FbxFileTexture *pfbxTexture = FbxFileTexture::Create(pfbxScene, "Diffuse Texture");
pfbxTexture->SetFileName("Scene03.jpg");
pfbxTexture->SetTextureUse(FbxTexture::eStandard);
pfbxTexture->SetMappingType(FbxTexture::eUV);
pfbxTexture->SetMaterialUse(FbxFileTexture::eModelMaterial);
pfbxTexture->SetSwapUV(false);
pfbxTexture->SetTranslation(0.0, 0.0);
pfbxTexture->SetScale(1.0, 1.0);
pfbxTexture->SetRotation(0.0, 0.0);
if (pfbxPhongMaterial) pfbxPhongMaterial->Diffuse.ConnectSrcObject(pfbxTexture);
```

재질의 속성을 텍스쳐에 연결

bool FbxFileTexture::SetFileName(char *pFileName);

FbxPropertyT<FbxDouble3> FbxSurfaceLambert::Diffuse;

bool FbxProperty::ConnectSrcObject(FbxObject *pObj, FbxConnection::EType pType=FbxConnection::eNone)

```
FbxSurfacePhong *CreatePhongMaterial(FbxScene *pfbxScene, char *pstrMaterialName) {
    FbxSurfacePhong *pfbxPhongMaterial = FbxSurfacePhong::Create(pfbxScene, pstrMaterialName);
    pfbxPhongMaterial->Emissive.Set(FbxDouble3(0.0, 0.0, 0.0));
    ...
    return(pfbxPhongMaterial);
}
```

메쉬, 재질, 텍스쳐

• 텍스쳐 레이어(Layered Texture)

- 텍스쳐 레이어는 순차적으로 블렌딩되는 여러 개의 텍스쳐들의 조합
- 텍스쳐 레이어는 [FbxLayeredTexture](#) 클래스로 구현함

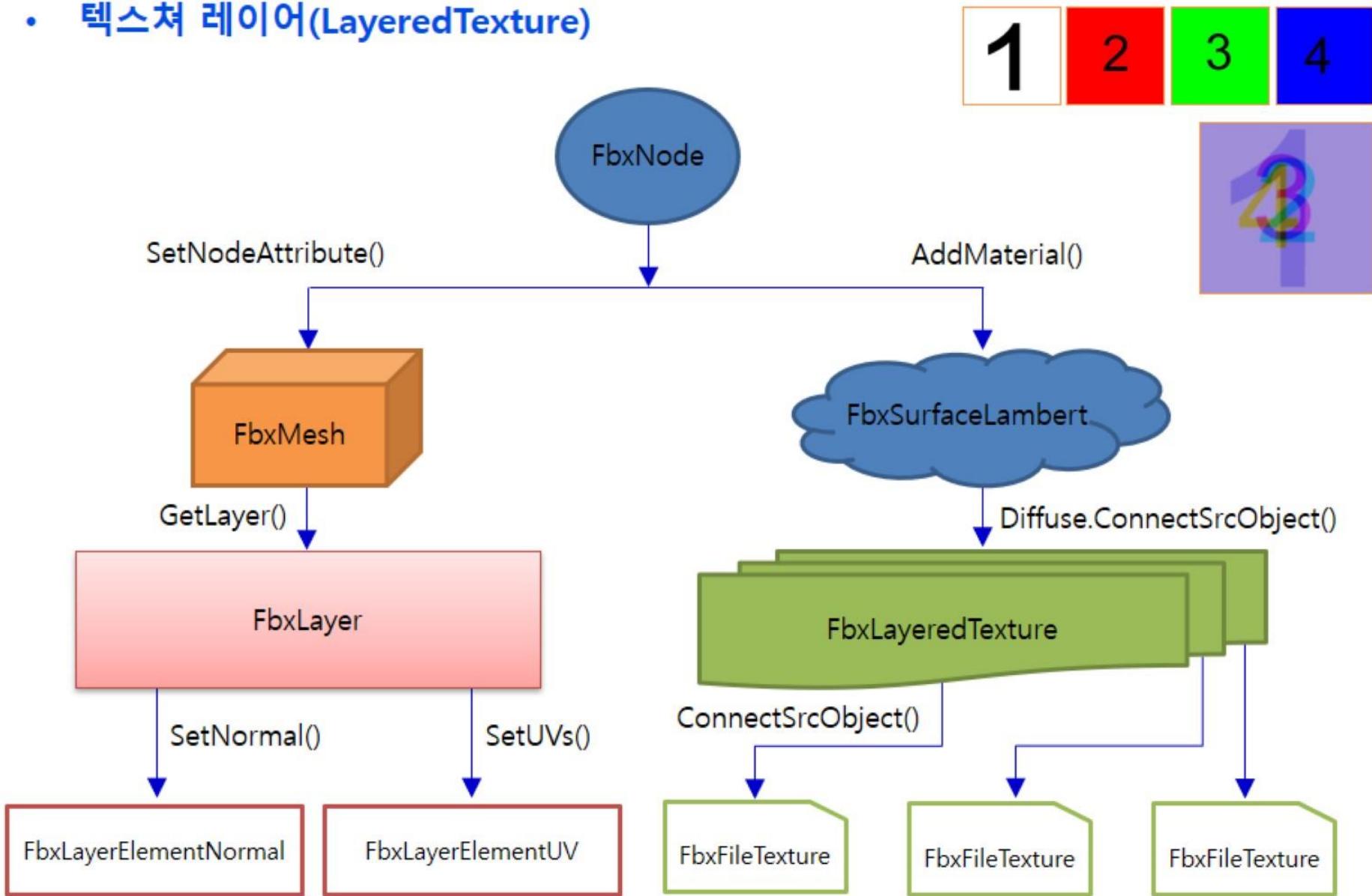
```
bool FbxLayeredTexture::SetTextureBlendMode (int nIndex, EBlendMode pMode);
bool FbxLayeredTexture::GetTextureBlendMode (int nIndex, EBlendMode& pMode);
bool FbxLayeredTexture::SetTextureAlpha (int nIndex, double pAlpha);
bool FbxLayeredTexture::GetTextureAlpha (int nIndex, double &pAlpha);
```

eTranslucent	새로운 텍스쳐와 이전 텍스쳐를 블렌딩(알파값에 따라 투명)
eAdditive	이전 텍스쳐의 색상과 새로운 텍스쳐의 색상을 더함
eModulate	새로운 텍스쳐의 색상과 이전 모든 텍스쳐 레이어의 색상을 곱함
eOver	eTranslucent
eNormal	레이어 1의 색상이 표시(블렌딩 되지 않음)
eDissolve	낮은 레이어의 색상이 표시
eDarken	픽셀을 블렌딩할 때 어두운 색상을 선택
eColorBurn	픽셀을 블렌딩할 때 곱셈 연산(더 밝아지지 않음)
eLinearBurn	픽셀을 블렌딩할 때 곱셈 연산(밝기 증가하도록)
eDarkerColor	픽셀을 블렌딩할 때 나눗셈 연산
eLighten	픽셀을 블렌딩할 때 밝은 색상을 선택
eSubtract	전경 색상에서 배경 색상을 뺄셈(전경 색상의 알파값을 적용)
eDivide	픽셀을 블렌딩할 때 나눗셈 연산
eLuminosity	낮은 레이어의 밝기를 높은 레이어의 밝기로 높임
eColor	낮은 레이어의 색조와 채도를 높은 레이어의 색조와 채도로 변경
eOverlay	레이어는 어두워지게 화면은 더 밝아지게 블렌딩

```
enum EBlendMode {
    eTranslucent, eAdditive,
    eModulate, eModulate2,
    eOver, eNormal, eDissolve,
    eDarken, eColorBurn,
    eLinearBurn, eDarkerColor,
    eLighten, eScreen,
    eColorDodge, eLinearDodge,
    eLighterColor, eSoftLight,
    eHardLight, eVividLight,
    eLinearLight, ePinLight,
    eHardMix, eDifference,
    eExclusion, eSubtract,
    eDivide, eHue,
    eSaturation, eColor,
    eLuminosity, eOverlay,
    eBlendModeCount
}
```

메쉬, 재질, 텍스쳐

- 텍스쳐 레이어(LayeredTexture)



메쉬, 재질, 텍스쳐

- 텍스쳐 레이어(Layered Texture)

```
FbxLayer *pfbxLayer = pfbxMesh->GetLayer(0);
FbxLayerElementNormal *pfbxLayerElementNormal= FbxLayerElementNormal::Create(pfbxMesh, "Normal");
pfbxLayerElementNormal->SetMappingMode(FbxLayerElement::eByControlPoint);
pfbxLayerElementNormal->SetReferenceMode(FbxLayerElement::eDirect);
pfbxLayerElementNormal->GetDirectArray().Add(FbxVector4(0, 0, 1)); ...
pfbxLayer->SetNormals(pfbxLayerElementNormal);

FbxLayerElementUV *pfbxLayerElementUV= FbxLayerElementUV::Create(pfbxMesh, "Diffuse");
pfbxLayerElementUV->SetMappingMode(FbxLayerElement::eByPolygonVertex);
pfbxLayerElementUV->SetReferenceMode(FbxLayerElement::eDirect);
pfbxLayerElementUV->GetDirectArray().Add(FbxVector2(0, 0)); ...
pfbxLayer->SetUVs(pfbxLayerElementUV, FbxLayerElement::eTextureDiffuse);

FbxLayeredTexture *pfbxLayeredTexture = FbxLayeredTexture::Create(pfbxScene, "LayeredTexture");
FbxFileTexture *pfbxTexture01 = FbxFileTexture::Create(pfbxScene, "FileTexture01");
pfbxTexture01->SetFileName("FileTexture01.jpg");
pfbxLayeredTexture->ConnectSrcObject(pfbxTexture01);
pfbxLayeredTexture->SetTextureBlendMode(0, FbxLayeredTexture::eAdditive);
pfbxLayeredTexture->SetTextureAlpha(0, 0.5);
FbxFileTexture *pfbxTexture02 = FbxFileTexture::Create(pfbxScene, "FileTexture02");
pfbxTexture02->SetFileName("FileTexture02.jpg");
pfbxLayeredTexture->ConnectSrcObject(pfbxTexture02);
...
FbxSurfaceLambert *pfbxMaterial = FbxSurfaceLambert::Create(pfbxScene, "Material");
pfbxMaterial->Diffuse.ConnectSrcObject(pfbxLayeredTexture);
pfbxNode->AddMaterial(pfbxMaterial);
```

메쉬, 재질, 텍스쳐

- 법선 벡터와 텍스쳐 좌표

```
FbxGeometryElementNormal *pfbxNormalElement = NULL;
FbxGeometryElementUV *pfbxUVElement = NULL;
if (bHasNormalElement) pfbxNormalElement = pfbxMesh->GetElementNormal(0);
if (bHasUVElement) pfbxUVElement = pfbxMesh->GetElementUV(0);
for (int i = 0, nNormalIndex = 0, nUVIndex = 0; i < m_nVertices; i++) {
    fbxvCurrentPosition = m_pfbxvSourceControlPoints[i];
    m_pd3dxvPositions[i] = D3DXVECTOR3(float(fbxvCurrentPosition[0]), float(fbxvCurrentPosition[1]),
    float(fbxvCurrentPosition[2]));
    if (bHasNormalElement) {
        nNormalIndex = (pfbxNormalElement->GetReferenceMode() == FbxLayerElement::eIndexToDirect) ?
pfbxNormalElement->GetIndexArray().GetAt(i) : i;
        fbxvCurrentNormal = pfbxNormalElement->GetDirectArray().GetAt(nNormalIndex);
        pd3dxvNormals[i] = D3DXVECTOR3(float(fbxvCurrentNormal[0]), float(fbxvCurrentNormal[1]),
        float(fbxvCurrentNormal[2]));
    }
    if (bHasUVElement) {
        nUVIndex = (pfbxUVElement->GetReferenceMode() == FbxLayerElement::eIndexToDirect) ?
pfbxUVElement->GetIndexArray().GetAt(i) : i;
        fbxvCurrentUV = pfbxUVElement->GetDirectArray().GetAt(nUVIndex);
        pd3dxvTextureCoordinates[i] = D3DXVECTOR2(float(fbxvCurrentUV[0]), float(fbxvCurrentUV[1]));
    }
}
```

애니메이션(Animation)

- 애니메이션 자료 구조(Animation Data Structure)

- 애니메이션 클래스

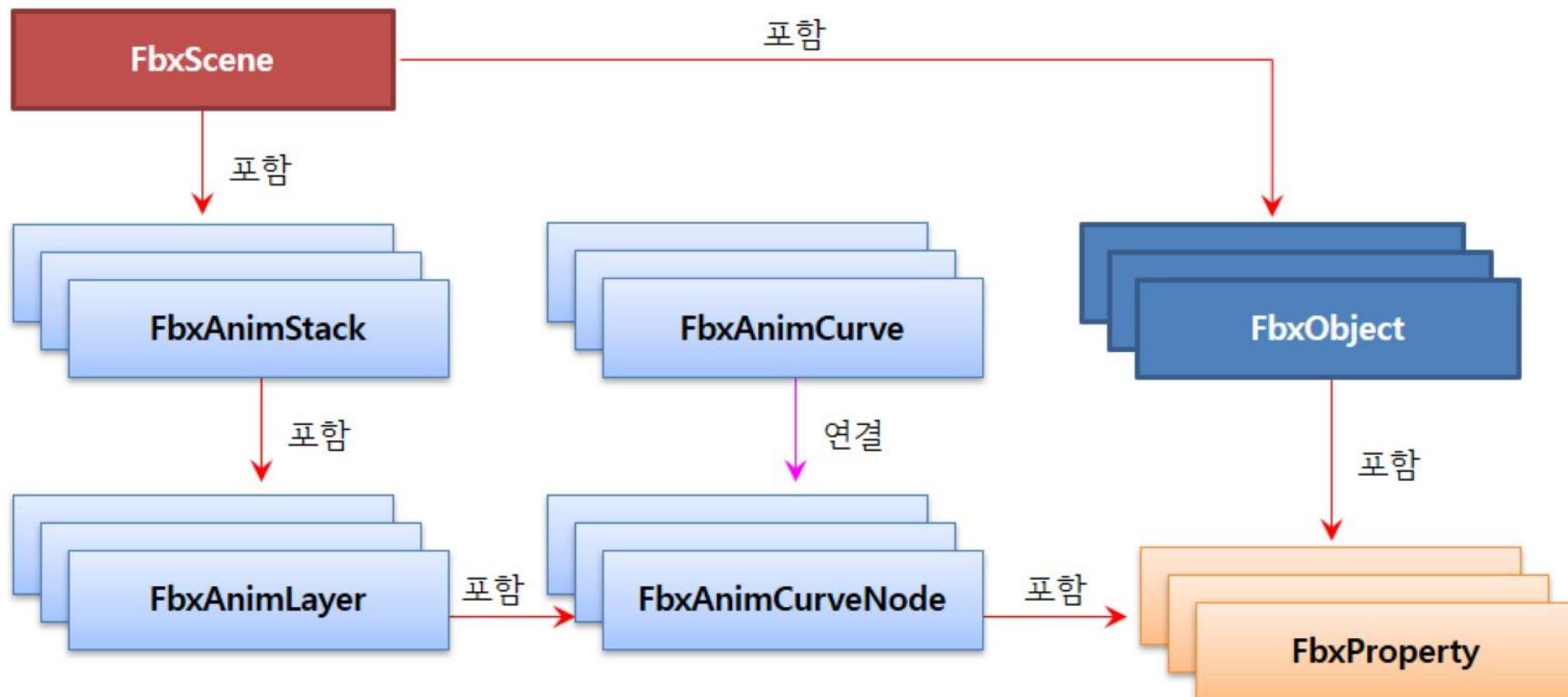
- 애니메이션 스택(FbxAnimStack)

- 애니메이션 레이어(FbxAnimLayer)

- 애니메이션 커브 노드(FbxAnimCurveNode)

- 애니메이션 커브(FbxAnimCurve)

- 애니메이션 커브 키(FbxAnimCurveKey)



애니메이션(Animation)

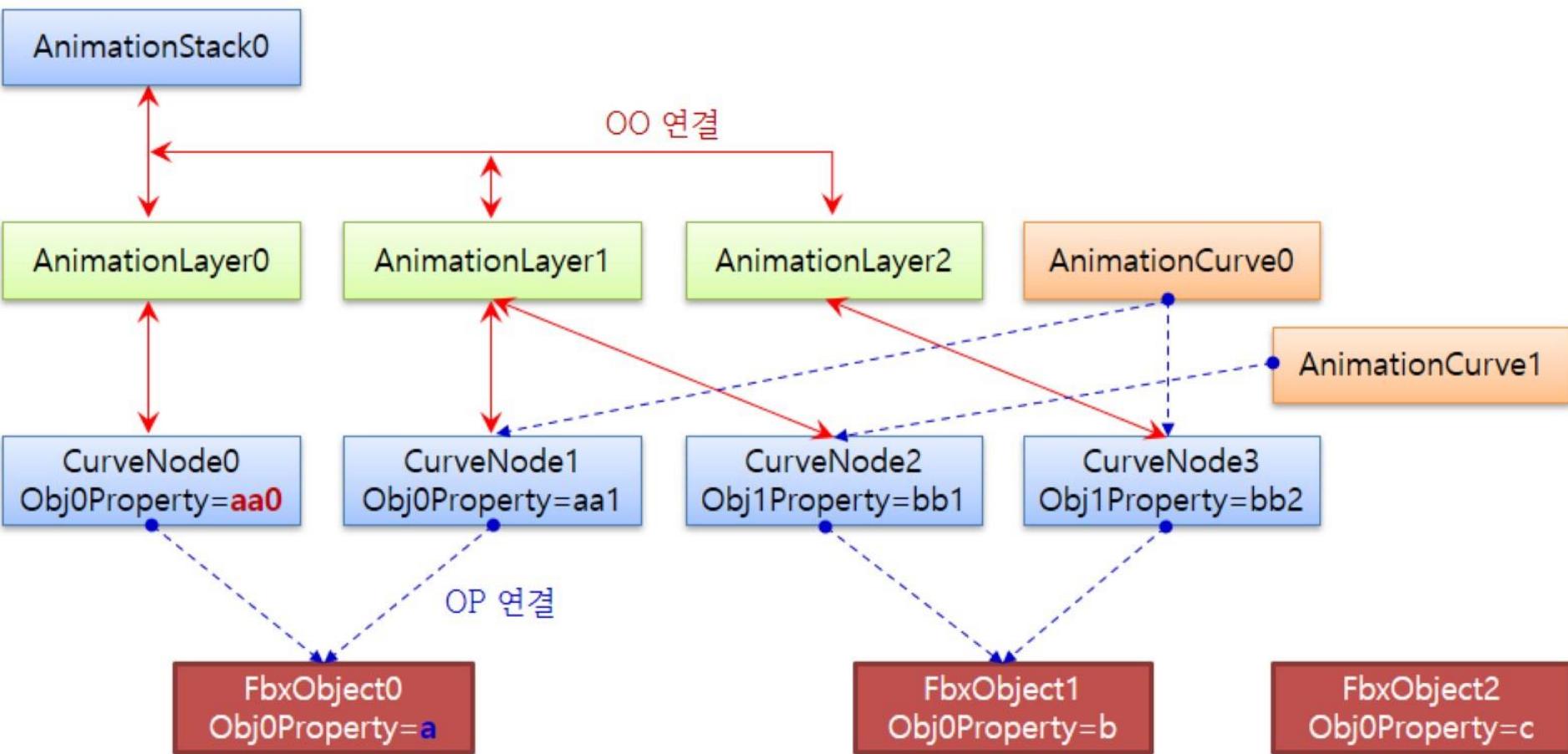
- 애니메이션 자료 구조(Animation Data Structure)

- 애니메이션 클래스

FbxScene	씬은 여러 개의 애니메이션 스택을 가질 수 있음
FbxAnimStack	애니메이션 스택은 하나 이상의 애니메이션 레이어를 포함 애니메이션 스택은 적어도 하나의 애니메이션 레이어를 가져야 함(기본 레이어) 애니메이션 블렌딩을 위해서 두 개 이상의 애니메이션 레이어가 필요
FbxAnimLayer	애니메이션 레이어는 하나 이상의 애니메이션 커브 노드를 포함 상태 플래그, 가중치, 블렌딩 모드 애니메이션 커브 노드는 애니메이션 커브에 연결됨
FbxAnimCurve	애니메이션 커브(FCurve)는 객체의 속성(FbxProperty)이 애니메이션 되는 방법을 정의 하나의 애니메이션 커브가 여러 개의 애니메이션 커브 노드에 연결될 수 있음
FbxAnimCurveBase	애니메이션 키 관리를 위한 기반 클래스
FbxAnimCurveNode	애니메이션 커브 노드는 애니메이션 커브와 FBX 속성을 연결함 하나의 애니메이션 커브와 하나의 FBX 속성을 애니메이션 커브 노드에 연결 애니메이션 커브 노드는 FbxAnimCurveNode::CreateTypedCurveNode()로 생성
FbxAnimCurveKey	애니메이션 커브의 시작과 끝을 나타내는 키 또는 키프레임
FbxObject	하나의 FBX 객체는 FBX 속성들을 포함할 수 있음 FbxNode는 객체의 위치와 관련된 속성들을 포함
FbxProperty	FBX 속성은 객체에 포함된 데이터 씬을 애니메이션하기 위해 FBX 객체에 포함된 적절한 FBX 속성들을 설정
FbxAnimEvaluator	특정 시간에 노드 변환과 속성 값을 계산하기 위하여 사용됨 기본적으로 FbxAnimEvalClassic 클래스를 사용(FbxScene::SetEvaluator()) 씬 노드에 대하여 아핀 변환 행렬을 반환, 속성에 대하여 속성이 가질 수 있는 구조체 반환

애니메이션(Animation)

- 애니메이션 자료 구조(Animation Data Structure)



애니메이션 커브 노드의 값이 FBX 속성의 값을 대체함

애니메이션 레이어(AnimationLayer1)에서 Obj0Property의 값은 AnimationCurve0에서 시간에 따라 결정됨
애니메이션 레이어(AnimationLayer2)에서 Obj1Property의 값은 AnimationCurve1에서 시간에 따라 결정됨

애니메이션(Animation)

- 애니메이션 자료 구조(Animation Data Structure)

- 애니메이션 계산(Evaluation)

씬에서 애니메이션을 계산하기 위하여 **FbxAnimEvaluator** 클래스(추상)를 사용
노드의 변환과 속성값을 시간에 따라 계산함
씬은 기본 **FbxAnimEvalClassic** 객체를 가지고 있음

```
FbxAnimEvaluator *FbxScene::GetEvaluator();
void FbxScene::SetEvaluator(FbxAnimEvaluator *pfbxEvaluator);
```

```
FbxAMatrix& fbxmtxGlobal = pfbxNode->EvaluateGlobalTransform(fbxTime);
```

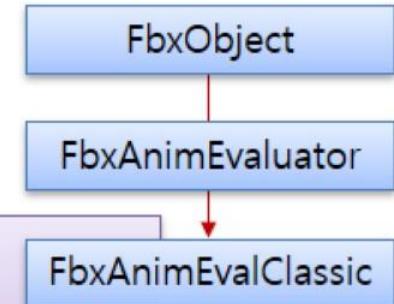
```
FbxAnimEvaluator *pfbxEvaluator = pfbxNode->GetScene()->GetEvaluator();
FbxAMatrix& fbxmtxGlobal = pfbxEvaluator->GetNodeGlobalTransform(pfbxNode, fbxTime);
```

```
FbxColor fbxColor = pfbxMaterial->GetDiffuseColor()->EvaluateValue();
```

FbxAnimEvaluator

```
FbxAnimStack *GetContext ();
FbxAMatrix& GetNodeGlobalTransform(FbxNode *pNode, ...);
FbxAMatrix& GetNodeLocalTransform(FbxNode *pNode, ...);
FbxPropertyValue& GetPropertyValue(FbxProperty& pProperty, FbxTime &pTime, ...);
FbxAnimCurveNode *GetPropertyCurveNode(FbxProperty& pProperty, FbxAnimLayer *pAnimLayer);
```

```
FbxAMatrix FbxNode::EvaluateGlobalTransform(FbxTime pTime=FBXSDK_TIME_INFINITE,
    FbxNode::EPivotSet PivotSet=FbxNode::eSourcePivot, bool bApplyTarget=false, bool bForceEval=false);
FbxAMatrix FbxNode::EvaluateLocalTransform(FbxTime pTime=FBXSDK_TIME_INFINITE,
    FbxNode::EPivotSet PivotSet=FbxNode::eSourcePivot, bool bApplyTarget=false, bool bForceEval=false);
```



애니메이션(Animation)

- 애니메이션 자료 구조(Animation Data Structure)

- 캐시(Cache)

정점 캐시(Vertex Cache)는 정점 애니메이션을 캐시 파일에 저장하는 방법
FBX SDK는 두 가지 점 캐시 파일 형식을 제공함

- eMaxPointCacheV2: 3ds Max Point Cache 2 파일 형식
 - eMayaCache: Maya Cache 파일 형식

캐시 파일의 점 애니메이션을 사용하기 위하여 [FbxCache](#) 클래스를 사용

```
int nVertexCacheDeformers = pfbxGeometry->GetDeformerCount(FbxDeformer::eVertexCache);
for (int i = 0; i < nVertexCacheDeformers; i++) {
    FbxVertexCacheDeformer *pfbxDeformer = static_cast<FbxVertexCacheDeformer *>(pfbxGeometry-
>GetDeformer(i, FbxDeformer::eVertexCache));
    FbxCache *pfbxCache = pfbxDeformer->GetCache();
    if (pfbxCache && pfbxCache->OpenFileForRead()) {
        int nChannel = pfbxCache->GetChannelIndex(pfbxDeformer->GetCacheChannel());
        if (nChannel < 0) continue;
        pfbxCache->GetChannelName(nChannel, fbxstrChannelName);
        FbxCache::EMCDataType eChannelType;
        pfbxCache->GetChannelDataType(nChannel, eChannelType);
        FbxCache::EMCSamplingType eChannelSampling;
        pfbxCache->GetChannelSamplingType(nChannel, eChannelSampling);
        FbxTime fbxtStart, fbxtStop, fbxtRate;
        pfbxCache->GetAnimationRange(nChannel, fbxtStart, fbxtStop);
        pfbxCache->GetChannelSamplingRate(nChannel, fbxtRate);
        unsigned int uChannelSamples;
        pfbxCache->GetChannelSampleCount(nChannel, uChannelSamples);
```

애니메이션(Animation)

- 애니메이션 블렌딩(Animation Blending)

- 애니메이션 블렌딩

하나의 애니메이션에서 다른 애니메이션으로 부드럽게 전이를 할 수 있음

공통의 속성을 공유하는 두 개의 애니메이션을 블렌딩할 수 있음

유사한 움직임을 가지는 애니메이션을 블렌딩하면 최상의 결과를 얻을 수 있음

(걷기 애니메이션 + 뛰기 애니메이션) = 걷기에서 뛰기로 부드럽게 전이

애니메이션 블렌딩을 위해 여러 개의 애니메이션 레이어가 필요

레이어0(걷기를 위한 애니메이션 커브 노드 포함)

레이어1(뛰기를 위한 애니메이션 커브 노드 포함)

애니메이션 레이어는 애니메이션 스택에 추가되는 순서대로 계산됨

애니메이션 레이어는 가중치(블렌드 가중치) 속성을 가짐

가중치 속성은 블렌딩할 때 각 애니메이션이 주는 영향을 결정

레이어0(걷기)의 가중치를 0.99에서 0.01로 변화

레이어1(뛰기)의 가중치를 0.01에서 0.99로 변화

애니메이션 커브 노드는 하나의 애니메이션 커브를 하나의 FBX 속성에 연결

애니메이션 커브 노드를 FBX 속성 값을 바꾸기 위하여 사용할 수 있음

Evaluation

FbxAnimLayer

```
FbxPropertyT<FbxDouble> Weight; //백분율, 기본값(100)
```

```
FbxPropertyT<FbxBool> Mute;
```

```
FbxPropertyT<FbxBool> Solo;
```

```
FbxPropertyT<FbxBool> Lock;
```

```
FbxPropertyT<FbxDouble3> Color;
```

```
FbxPropertyT<FbxEnum> BlendMode;
```

```
FbxPropertyT<FbxEnum> RotationAccumulationMode;
```

```
FbxPropertyT<FbxEnum> ScaleAccumulationMode;
```

```
enum EBlendMode {  
    eBlendAdditive, eBlendOverride, eBlendOverridePassthrough  
};
```

애니메이션(Animation)

- 애니메이션 블렌딩(Animation Blending)

- 애니메이션 블렌드 모드(Blend Mode)

애니메이션이 이전 레이어들의 애니메이션과 어떻게 블렌드되는 가를 조절함
두 개의 애니메이션 레이어는 같은 FBX 속성에 영향을 주어야 함

- 더하기 모드(Additive Mode)

애니메이션 레이어는 애니메이션 스택에서 이전 레이어에 더함
AnimLayer A와 AnimLayer B가 한 노드의 x 값을 조절하는 커브 노드를 포함
⇒ 결과 x 값은 두 레이어의 x 값의 합

- 오버라이드 모드(Override Mode)

애니메이션 레이어는 애니메이션 스택에서 이전 레이어의 애니메이션을 대체
AnimLayer A와 AnimLayer B가 모두 오버라이드 모드
AnimLayer A의 객체의 x 값 = 10, AnimLayerB의 객체의 x 값 = 15
⇒ 결과 애니메이션에서 FBX 객체의 x 값은 15

- 오버라이드-통과 모드(Override-Passthrough Mode)

애니메이션 레이어의 불투명도를 Weight 값으로 조절할 수 있음

`FbxAnimLayer::EBlendMode`

`FbxAnimLayer::ERotationAccumulationMode`

`FbxAnimLayer::EScaleAccumulationMode`

- 애니메이션 블렌드 모드를 적용하지 않기(Bypassing Blend Mode)

애니메이션 레이어는 각 자료형에 대하여 블렌드 모드 우회 플래그를 가짐

`true`: 지정된 자료형의 속성에 대하여 이 레이어를 위한 블렌드 모드는 무시됨

`false`: 지정된 자료형의 속성에 대하여 이 레이어의 블렌드 모드가 사용됨

`FbxAnimLayer::GetBlendModeBypass()`, `FbxAnimLayer::SetBlendModeBypass()`

애니메이션(Animation)

- 애니메이션 데이터 추출(Extracting Animation Data)

- FBX 파일에서 애니메이션 데이터 추출

전체 FBX 파일을 로드하지 않고 씬(FBX 파일)에서 애니메이션 데이터를 추출 가능

씬에서 애니메이션 데이터를 추출하는 절차

① 애니메이션 스택

```
int nAnimStacks = pfbxScene->GetSrcObjectCount<FbxAnimStack>();  
for (int i = 0; i < nAnimStacks; i++) {  
    FbxAnimStack *pfbxAnimStack = pfbxScene->GetSrcObject<FbxAnimStack>(i);
```

② 애니메이션 레이어

```
int nAnimLayers = pfbxAnimStack->GetMemberCount<FbxAnimLayer>();  
for (int j = 0; j < nAnimLayers; j++) {  
    FbxAnimLayer *pfbxAnimLayer = pfbxAnimStack->GetMember<FbxAnimLayer>(j);
```

③ 애니메이션 커브(노드 속성)

```
FbxAnimCurve *pfbxAnimCurve = pfbxNode->LclTranslation.GetCurve(pfbxAnimLayer, "X");  
  
FbxNodeAttribute *pfbxAttribute = pfbxNode->GetNodeAttribute();  
pfbxAnimCurve = pfbxAttribute->Color.GetCurve<FbxAnimCurve>(pfbxAnimLayer, "X");
```

④ 애니메이션 커브 노드

```
FbxProperty fbxProperty = pfbxNode->GetFirstProperty();  
FbxAnimCurveNode *pfbxCurveNode = fbxProperty.GetCurveNode(pfbxAnimLayer);  
int nCurveNodes = pfbxCurveNode->GetCurveCount(0);  
for (int k = 0; k < nCurveNodes; k++) {  
    FbxAnimCurve *pfbxAnimCurve = pfbxCurveNode->GetCurve(0, k);
```

```
}
```

LclTranslation: 채널 3개

```
FbxNode *pfbxNode = pfbxScene->GetRootNode();  
int nNodes = pfbxNode->GetChildCount();  
for (int k = 0; k < nNodes; k++) pfbxNode = pfbxNode->GetChild(k);
```

애니메이션(Animation)

- 애니메이션 데이터 추출(Extracting Animation Data)

- 파일에서 애니메이션 데이터 추출

```
int FbxObject::GetSrcObjectCount<T>();
T *FbxObject::GetSrcObject<T>(int nIndex=0);
int FbxObject::GetMemberCount<T>();
T *FbxObject::GetMember<T>(int nIndex=0);
FbxProperty FbxObject::GetFirstProperty();
FbxProperty FbxObject::GetNextProperty(FbxProperty& fbxProperty);
FbxAnimCurve *FbxProperty::GetCurve(FbxAnimLayer *pAnimLayer, char *pChannel, bool pCreate=false);
FbxAnimCurveNode *FbxProperty::GetCurveNode(FbxAnimStack *pAnimStack, bool pCreate=false)
FbxAnimCurveNode *FbxProperty::GetCurveNode(FbxAnimLayer *pAnimLayer, bool pCreate=false);
```

```
int FbxAnimCurveNode::GetCurveCount(int nChannelId, char *pCurveNodeName=NULL);
FbxAnimCurve *FbxAnimCurveNode::GetCurve(int nChannelId, int Id=0, char *pCurveNodeName=NULL);
unsigned int FbxAnimCurveNode::GetChannelsCount();
```

#define FBXSDK_CURVENODE_TRANSFORM	"Transform"	표준 커브 노드 이름
#define FBXSDK_CURVENODE_TRANSLATION	"T"	
#define FBXSDK_CURVENODE_ROTATION	"R"	
#define FBXSDK_CURVENODE_SCALING	"S"	
#define FBXSDK_CURVENODE_COMPONENT_X	"X"	
#define FBXSDK_CURVENODE_COMPONENT_Y	"Y"	
#define FBXSDK_CURVENODE_COMPONENT_Z	"Z"	
#define FBXSDK_CURVENODE_COLOR	"Color"	
#define FBXSDK_CURVENODE_COLOR_RED	FBXSDK_CURVENODE_COMPONENT_X	
#define FBXSDK_CURVENODE_COLOR_GREEN	FBXSDK_CURVENODE_COMPONENT_Y	
#define FBXSDK_CURVENODE_COLOR_BLUE	FBXSDK_CURVENODE_COMPONENT_Z	

애니메이션(Animation)

- 애니메이션 데이터 추출(Extracting Animation Data)

- 파일에서 애니메이션 데이터 추출

ImportScene/DisplayAnimation.cxx

```
void DisplayAnimation(FbxScene *pfbxScene) {
    for (int i = 0; i < pfbxScene->GetSrcObjectCount<FbxAnimStack>(); i++) {
        FbxAnimStack *pfbxAnimStack = pfbxScene->GetSrcObject<FbxAnimStack>(i);
        DisplayAnimationStack(pfbxAnimStack, pfbxScene->GetRootNode());
    }
}

void DisplayAnimationStack(FbxAnimStack *pfbxAnimStack, FbxNode *pfbxNode) {
    for (int i = 0; i < pfbxAnimStack->GetMemberCount<FbxAnimLayer>(); i++) {
        FbxAnimLayer *pfbxAnimLayer = pfbxAnimStack->GetMember<FbxAnimLayer>(i);
        DisplayAnimationLayer(pfbxAnimLayer, pfbxNode);
    }
}

void DisplayAnimationLayer(FbxAnimLayer *pfbxAnimLayer, FbxNode *pfbxNode) {
    DisplayChannels(pfbxNode, pfbxAnimLayer);
    for (int i = 0; i < pfbxNode->GetChildCount(); i++) {
        DisplayAnimationLayer(pfbxAnimLayer, pfbxNode->GetChild(i));
    }
}

void DisplayChannels(FbxNode *pfbxNode, FbxAnimLayer *pfbxAnimLayer) {
    FbxAnimCurve *pfbxAnimCurve = pfbxNode->LclRotation.GetCurve(pfbxAnimLayer, "X");
    if (pfbxAnimCurve) DisplayAnimCurve(pfbxAnimCurve);
    FbxNodeAttribute *pfbxNodeAttribute = pfbxNode->GetNodeAttribute();
    FbxLight *pfbxLight = pfbxNode->GetLight();
    if (pfbxLight) pfbxAnimCurve = pfbxLight->Intensity.GetCurve(pfbxAnimLayer);

    void DisplayAnimCurve(FbxAnimCurve *pfbxCurve) {
        for (int i = 0; i < pfbxCurve->KeyGetCount(); i++) {
            float fKeyValue = static_cast<float>(pfbxCurve->KeyGetValue(i));
            FbxTime fbxKeyTime = pfbxCurve->KeyGetTime(i);
        }
    }
}
```

애니메이션(Animation)

- 애니메이션 계산(Evaluating Animation)

- 애니메이션 계산

씬을 렌더링하기 위해 시간에 따라 객체의 애니메이션 속성을 계산해야 함

씬에서 애니메이션을 계산하는 절차

```
FbxScene *pfbxScene;
FbxNode *pfbxCameraNode;
pfbxCameraNode.LclTranslation.Set(fbxDouble3(1.1, 2.2, 3.3));
```

① 이밸유에이터(Evaluator) 객체

```
FbxAnimEvaluator *pfbxSceneEvaluator = pfbxScene->GetEvaluator();
```

② 시간 객체 생성

```
FbxTime fbxTime;
fbxTime.SetSecondDouble(0.0);
```

③ 노드 객체 생성

```
FbxNode *pfbxMeshNode = pfbxNode->LclTranslation.GetCurve(pfbxAnimLayer, "X");
```

```
FbxMatrix& fbxGlobalMatrix = pfbxSceneEvaluator->GetNodeGlobalTransform(pfbxMeshNode, fbxTime);
FbxMatrix& fbxLocalMatrix = pfbxSceneEvaluator->GetNodeLocalTransform(pfbxMeshNode, fbxTime);
```

④ 애니메이션 커브 노드

```
FbxProperty fbxProperty = pfbxNode->GetFirstProperty();
FbxAnimCurveNode *pfbxCurveNode = fbxProperty.GetCurveNode(pfbxAnimLayer);
int nCurveNodes = pfbxCurveNode->GetCurveCount(0);
for (int k = 0; k < nCurveNodes; k++) {
    FbxAnimCurve *pfbxAnimCurve = pfbxCurveNode->GetCurve(0, k);
}
```

애니메이션(Animation)

- 노드의 애니메이션(Animating a Node)

- 애니메이션 블렌딩을 사용하지 않으면 애니메이션을 위한 설정은 간단

```
pfbxCameraNode.LclTranslation.Set(fbxDouble3(1.1, 2.2, 3.3))
```

- 최소 하나의 애니메이션 스택과 레이어 필요

```
FbxAnimStack *pfbxAnimStack = FbxAnimStack::Create(pfbxScene, "My stack");
FbxAnimLayer *pfbxAnimLayer = FbxAnimLayer::Create(pfbxScene, "Layer0");
pfbxAnimStack->AddMember(pfbxAnimLayer);
```

- 커브 노드가 애니메이션 데이터와 속성을 연결

```
FbxAnimCurveNode *pfbxAnimCurveNode =
    pfbxCameraNode->LclTranslation.GetCurveNode(pfbxAnimLayer, true);
```

- 커브 노드는 애니메이션 커브의 컨테이너

```
FbxTime fbxTime;
FbxAnimCurve *pfbxTranXCurve = pfbxTranXCurve = pfbxCameraNode-
>LclTranslation.GetCurve<FbxAnimCurve>(pfbxAnimLayer, KFCURVENODE_T_X, true);
pfbxTranXCurve ->KeyModifyBegin();
fbxTime.SetSecondDouble(0.0);
int nKeyIndex = pfbxTranXCurve ->KeyAdd(fbxTime);
pfbxTranXCurve ->KeySet(IKeyIndex, fbxTime, 0.0, FbxAnimCurveDef::eINTERPOLATION_LINEAR);
fbxTime.SetSecondDouble(20.0);
nKeyIndex = pfbxTranXCurve ->KeyAdd(fbxTime);
pfbxTranXCurve ->KeySet(IKeyIndex, fbxTime, 500.0, FbxAnimCurveDef::eINTERPOLATION_LINEAR);
pfbxTranXCurve ->KeyModifyEnd();
```

애니메이션(Animation)

- 애니메이션 자료 구조(Animation Data Structure)

- 애니메이션 계산(Evaluation)

씬에서 애니메이션을 계산하기 위하여 **FbxAnimEvaluator** 클래스(추상)를 사용
노드의 변환과 속성값을 시간에 따라 계산함
씬은 기본 **FbxAnimEvalClassic** 객체를 가지고 있음

```
FbxAnimEvaluator *FbxScene::GetEvaluator();  
void FbxScene::SetEvaluator(FbxAnimEvaluator *pfbxEvaluator);
```

```
FbxAMatrix& fbxmtxGlobal = pfbxNode->EvaluateGlobalTransform(fbxTime);
```

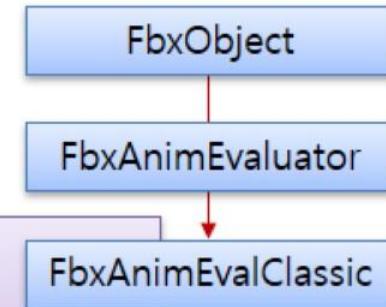
```
FbxAnimEvaluator *pfbxEvaluator = pfbxNode->GetScene()->GetEvaluator();  
FbxAMatrix& fbxmtxGlobal = pfbxEvaluator->GetNodeGlobalTransform(pfbxNode, fbxTime);
```

```
FbxColor fbxColor = pfbxMaterial->GetDiffuseColor()->EvaluateValue();
```

FbxAnimEvaluator

```
FbxAnimStack *GetContext();  
FbxAMatrix& GetNodeGlobalTransform(FbxNode *pNode, ...);  
FbxAMatrix& GetNodeLocalTransform(FbxNode *pNode, ...);  
FbxPropertyValue& GetPropertyValue(FbxProperty& pProperty, FbxTime &pTime, ...);  
FbxAnimCurveNode *GetPropertyCurveNode(FbxProperty& pProperty, FbxAnimLayer *pAnimLayer);
```

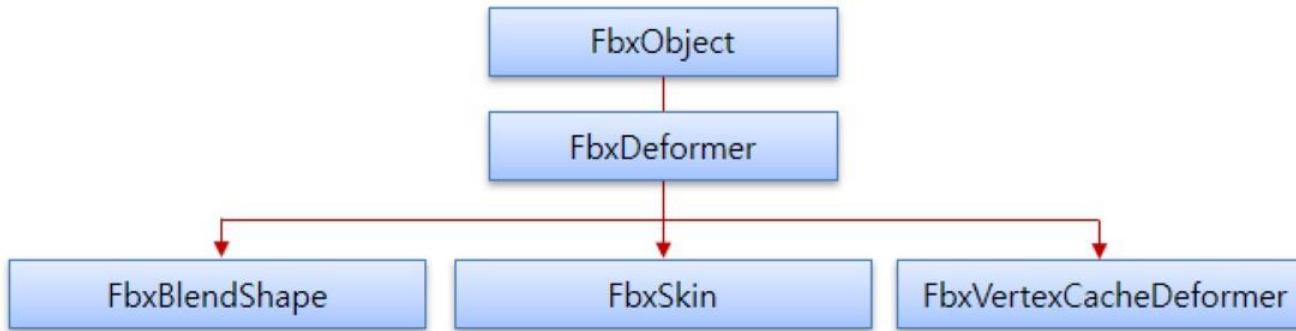
```
FbxAMatrix FbxNode::EvaluateGlobalTransform(FbxTime pTime=FBXSDK_TIME_INFINITE,  
FbxNode::EPivotSet PivotSet=FbxNode::eSourcePivot, bool bApplyTarget=false, bool bForceEval=false);  
FbxAMatrix FbxNode::EvaluateLocalTransform(FbxTime pTime=FBXSDK_TIME_INFINITE,  
FbxNode::EPivotSet PivotSet=FbxNode::eSourcePivot, bool bApplyTarget=false, bool bForceEval=false);
```



애니메이션(Animation)

- **FbxGeometry** 클래스

- 제어점(Control Point)의 변형(Deformation)을 지원하는 기하 객체의 기반 클래스
제어점을 포함하는 클래스: FbxMesh, FbxNurbs, FbxPatch
- FbxGeometry 클래스는 다음과 같은 디포머(Deformer)를 지원
 - 스킨 디포머(Skin Deformation Deformer): FbxSkin
 - 정점 캐시 디포머(Vertex Cache Deformer): FbxVertexCacheDeformer
 - 블렌드 형상 디포머(Blend Shape Deformer): FbxBlendShape
 - 기하 가중치 맵(Geometry Weighted Map): FbxGeometryWeightedMap
- 디포머는 기하(FbxGeometry)에 연결되어 형상(Shape)을 결정



FbxGeometry

```
int GetDeformerCount(); //모든 디포머의 개수
FbxDeformer *GetDeformer(int nIndex, FbxStatus *pStatus=NULL);
int GetDeformerCount(FbxDeformer::EDeformerType nType); //특정한 유형의 디포머의 개수
FbxDeformer *GetDeformer(int nIndex, FbxDeformer::EDeformerType nType, FbxStatus *pStatus=NULL);

EDeformerType FbxDeformer::GetDeformerType();
```

```
enum FbxDeformer::EDeformerType {
    eUnknown, eSkin, eBlendShape, eVertexCache
};
```

애니메이션(Animation)

- **디포머(Deformer) 클래스: FbxDeformer**

- 기하(Geometry)에 연결된 디포머는 기하의 모양을 바꾸기 위하여 동작(애니메이션) 디포머의 하위 객체가 애니메이션 되면 기하도 따라서 애니메이션 됨
- 디포머 형식

FbxDeformer::eSkin	스킨 디포머(FbxSkin)
FbxDeformer::eBlendShape	블렌드 형상 디포머(FbxBlendShape)
FbxDeformer::eVertexCache	정점 캐쉬 디포머(FbxVertexCacheDeformer)

```
FbxDeformer(FbxManager &pManager, const char *pName);  
EDeformerType FbxDeformer::GetDeformerType();
```

```
int nSkinDeformers = pGeometry->GetDeformerCount(FbxDeformer::eSkin);  
for (i = 0; i < nSkinDeformers; i++)  
{  
    FbxSkin *pfbxSkinDeformer = (FbxSkin *)(pGeometry->GetDeformer(i, FbxDeformer::eSkin));  
    int nClusters = pfbxSkinDeformer->GetClusterCount();  
    for (j = 0; j < nClusters; j++)  
    {  
        FbxCluster *pfbxCluster = pfbxSkinDeformer->GetCluster(j);  
        int nLinkMode = pfbxCluster->GetLinkMode();  
        int nIndices = pfbxCluster->GetControlPointIndicesCount();  
        int *pnIndices = pfbxCluster->GetControlPointIndices();  
        double *pfWeights = pfbxCluster->GetControlPointWeights();  
        ...  
    }  
}
```

애니메이션(Animation)

• 클러스터(Cluster, Link)

- 기하의 제어점(정점)의 부분집합에서 가중치에 따라 동작하는 개체
클러스터가 동작하는 각 제어점에 대하여 클러스터 동작의 세기를 가중치와 곱함
클러스터의 링크 노드는 클러스터의 제어점들에 영향을 주는 노드(뼈대)를 나타냄
링크 노드가 움직이면 클러스터의 제어점들이 따라서 움직임
클러스터는 일반적으로 스킨(Skin)의 일부분임
- 링크(Link): 제어점들에 대한 인덱스와 가중치들의 배열을 가짐
- 링크 모드(Link Mode)
링크가 하나의 제어점(위치)에 어떻게 영향을 주는 가를 설정

`ELinkMode FbxCluster::GetLinkMode();`

하나의 제어점에 설정된 가중치들 사이의 관계를 설정

하나의 제어점에 설정된 가중치들은 기하에 연결된 링크들의 집합에 분배됨

eNormalize	하나의 제어점에 설정된 가중치의 합은 1.0으로 정규화
eAdditive	하나의 제어점에 설정된 가중치들의 합을 유지 가중치에 따라 하나의 제어점은 비례하여 이동함
eTotalOne	eNormalize와 유사(단, 가중치의 합은 1.0, 정규화되지 않음)

- 골격(Skeleton): 뼈대(Bone)들의 집합
각 뼈대(Bone)는 하나의 FBX의 노드로 표현됨
기하를 노드(뼈대)들에 연결하기 위하여 하나의 스킨(Skin: FbxSkin)을 생성
스킨은 여러 개의 클러스터를 가지며 각 클러스터는 하나의 뼈대에 대응됨
각 뼈대 노드는 메쉬의 어떤 제어점들에 영향을 줌(영향의 정도는 가중치로 표현)
뼈대의 영향을 전혀 받지 않는 메쉬의 제어점들은 클러스터에 속하지 않음

애니메이션(Animation)

• FbxCluster 클래스

- 클러스터(링크: Link)

하나의 뼈대가 영향을 주는 기하의 제어점(정점)의 부분집합과 가중치를 표현

클러스터의 링크 노드는 클러스터의 제어점에 영향을 주는 노드(뼈대)를 나타냄

링크 노드가 움직이면 클러스터의 정보에 따라 제어점들이 따라서 움직임

ELinkMode GetLinkMode();

EType GetSubDeformerType();

FbxNode *GetLink(); //제어점에 영향을 주는 노드(뼈대: Bone)

FbxNode *GetAssociateModel(); //eAdditive 모드일 때

int GetControlPointIndicesCount(); //뼈대가 영향을 주는 제어점 인덱스와 가중치의 개수

int *GetControlPointIndices(); //뼈대가 영향을 주는 제어점 인덱스의 배열

double *GetControlPointWeights(); //뼈대가 영향을 주는 제어점 가중치의 배열

FbxAMatrix& GetTransformMatrix(FbxAMatrix& pMatrix);

FbxAMatrix& GetTransformLinkMatrix(FbxAMatrix& pMatrix);

FbxAMatrix& GetTransformAssociateModelMatrix(FbxAMatrix& pMatrix);

FbxAMatrix& GetTransformParentMatrix(FbxAMatrix& pMatrix);

```
enum FbxCluster::ELinkMode {  
    eNormalize, eAdditive, eTotalOne  
};
```

- **TransformMatrix**: 링크 노드를 포함하는 (기하)노드의 초기(바인딩 시점) 전역 변환 행렬
- **TransformLinkMatrix**: 링크 노드의 초기(바인딩 시점) 변환 행렬
- **TransformAssociateModelMatrix**: 모델의 초기 변환 행렬

몇 개의 뼈대(링크)와 연결된 메쉬의 경우:

- **Transform**: 바인딩 시점의 메쉬의 전역 변환(Global Transform)
- **TransformLink**: 바인딩 시점의 뼈대(링크)의 전역 변환(Global Transform)
- **TransformAssociateModel**: 바인딩 시점의 연결된 모델의 전역 변환

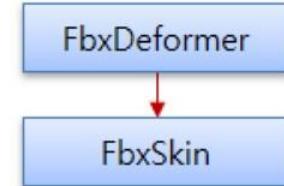
전형적으로 링크 노드는 스킨이 부착되는 뼈대임

연결된 모델(Associate Model) 노드는 스킨이 부착된 뼈대의 부모 노드임

애니메이션(Animation)

• 스킨 디포머 클래스: FbxSkin

- 클러스터(FbxCluster)를 포함하는 스킨 디포머 클래스
각 클러스터는 기하의 제어점들의 부분집합에 적용됨(다른 가중치를 가짐)
메쉬가 스킨을 가지면 스킨은 빠대가 메쉬를 변형하는 방법을 표현함
빠대가 움직이면 클러스터가 메쉬를 변형하기 위하여 동작함
- 스킨은 제어점에 대한 인덱스 배열과 블렌드 가중치를 가짐
- 스키닝의 유형



eRigid	오직 하나의 조인트가 각 제어점에 영향을 줌
eLinear	선형 스키닝(Linear Smooth Skinning)
eDualQuaternion	듀얼 쿼터니언 스키닝(Dual Quaternion Smooth Skinning)
eBlend	선형 스키닝과 듀얼 쿼터니언 스키닝을 블렌드 가중치에 따라 블렌딩

```
double GetDeformAccuracy(); //100: 최대 정밀도, 1: 최소 정밀도
```

```
FbxGeometry *GetGeometry();
int GetClusterCount();
FbxCluster *GetCluster(int nIndex);
EDeformerType GetDeformerType(); //eSkin
EType GetSkinningType();
```

```
int GetControlPointIndicesCount(); //대부분 NULL
int *GetControlPointIndices();
double *GetControlPointBlendWeights();
```

```
enum FbxSkin::EType {
    eRigid, eLinear, eDualQuaternion, eBlend
};
```

```
enum FbxDeformer::EDeformerType {
    eUnknown, eSkin, eBlendShape, eVertexCache
};
```

```
int nSkinDeformers = pfbxMesh->GetDeformerCount(FbxDeformer::eSkin);
FbxSkin *pfbxSkinDeformer = (FbxSkin *)pfbxMesh->GetDeformer(0, FbxDeformer::eSkin);
```

애니메이션(Animation)

- 스킨 디포머 클래스: FbxSkin

```
int nSkinDeformers = pfbxMesh->GetDeformerCount(FbxDeformer::eSkin);
int *pnInfluences = new int[m_nVertices];
for (int i = 0; i < nSkinDeformers; i++) {
    FbxSkin *pfbxSkinDeformer = (FbxSkin *)pfbxMesh->GetDeformer(i, FbxDeformer::eSkin);
    m_nClusters = (pfbxSkinDeformer) ? pfbxSkinDeformer->GetClusterCount() : 0;
    if (m_nClusters > 0) {
        m_pd3dxmtxClusterTransforms = new XMFLOAT4x4[m_nClusters];
        ::ZeroMemory(pnInfluences, m_nVertices * sizeof(int));
        for (int j = 0; j < m_nClusters; j++) {
            D3DXMatrixIdentity(&m_pd3dxmtxClusterTransforms[j]);
            FbxCluster *pfbxCluster = pfbxSkinDeformer->GetCluster(j);
            if (!pfbxCluster->GetLink()) continue;
            int nIndices = pfbxCluster->GetControlPointIndicesCount();
            int *pnControlPointIndices = pfbxCluster->GetControlPointIndices();
            double *pfControlPointWeights = pfbxCluster->GetControlPointWeights();
            for (int k = 0; k < nIndices; k++) {
                int nVertexID = pnControlPointIndices[k];
                if ((nVertexID < 0) || (nVertexID >= m_nVertices)) continue;
                float fWeight = (float)pfControlPointWeights[k];
                m_ppnClusterIDs[nVertexID][pnInfluences[nVertexID]] = j;
                m_ppfClusterWeights[nVertexID][pnInfluences[nVertexID]] = fWeight;
                pnInfluences[nVertexID]++; //정점에 영향을 주는 뼈대의 개수
            }
        }
        CLUSTERIDS *m_ppnClusterIDs;
        CLUSTERWEIGHTS *m_ppfClusterWeights;
    }
    int m_nClusters;
    XMFLOAT4x4 *m_pd3dxmtxClusterTransforms;
}
```

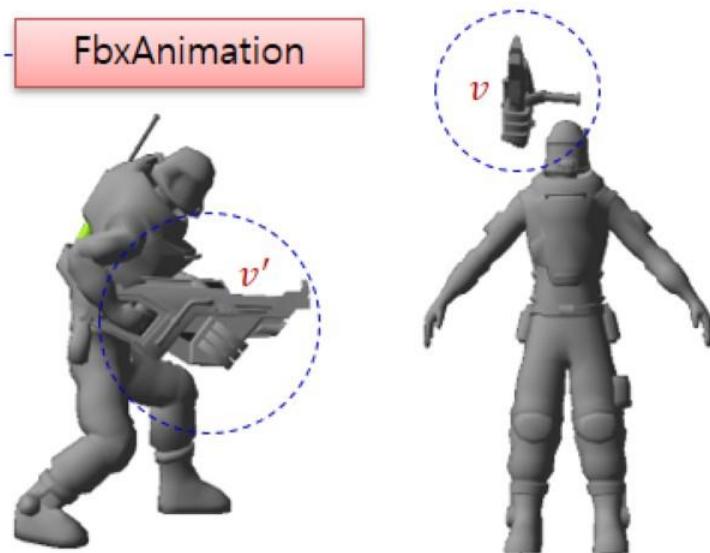
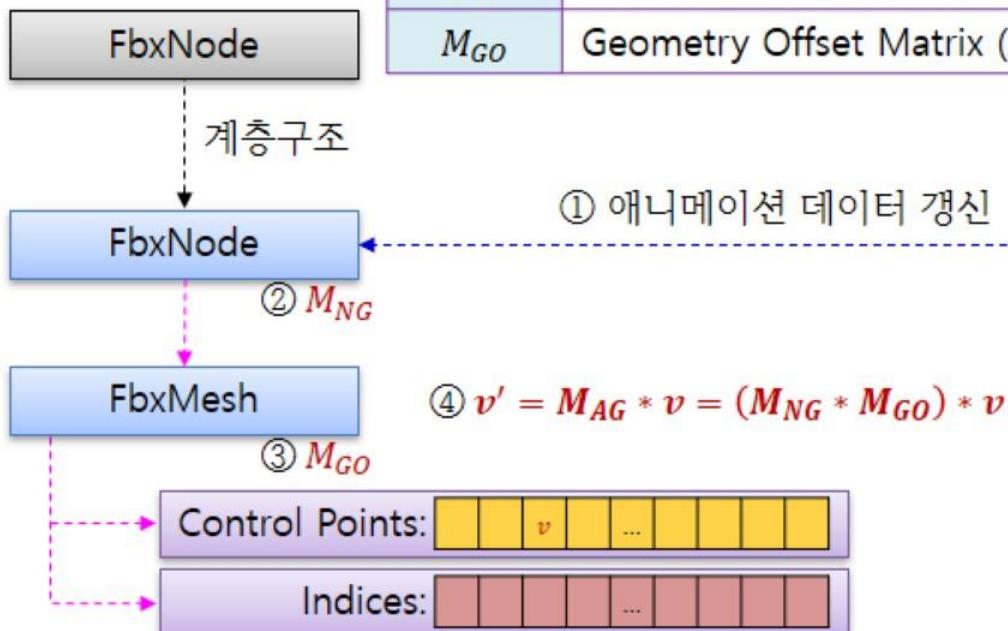
애니메이션(Animation)

- **FBX 스킨 디포머**

- 스킨 메쉬가 아닌 경우

$$M_{AG} = (M_{NG} * M_{GO})$$

M_{AG}	Attribute Global Matrix (AttributeToRoot Matrix)
M_{NG}	Node Global Matrix (NodeToRoot Matrix)
M_{GO}	Geometry Offset Matrix (AttributeToNode Matrix)



```
FbxAMatrix fbxmtxNodeGlobal = pfbxNode->EvaluateGlobalTransform(fbxTime);  
FbxAMatrix fbxmtxGeometryOffset = GetAttributeNodeTransform(pfbxNode);  
FbxAMatrix fbxmtxAttributeGlobal = fbxmtxNodeGlobal * fbxmtxGeometryOffset;
```

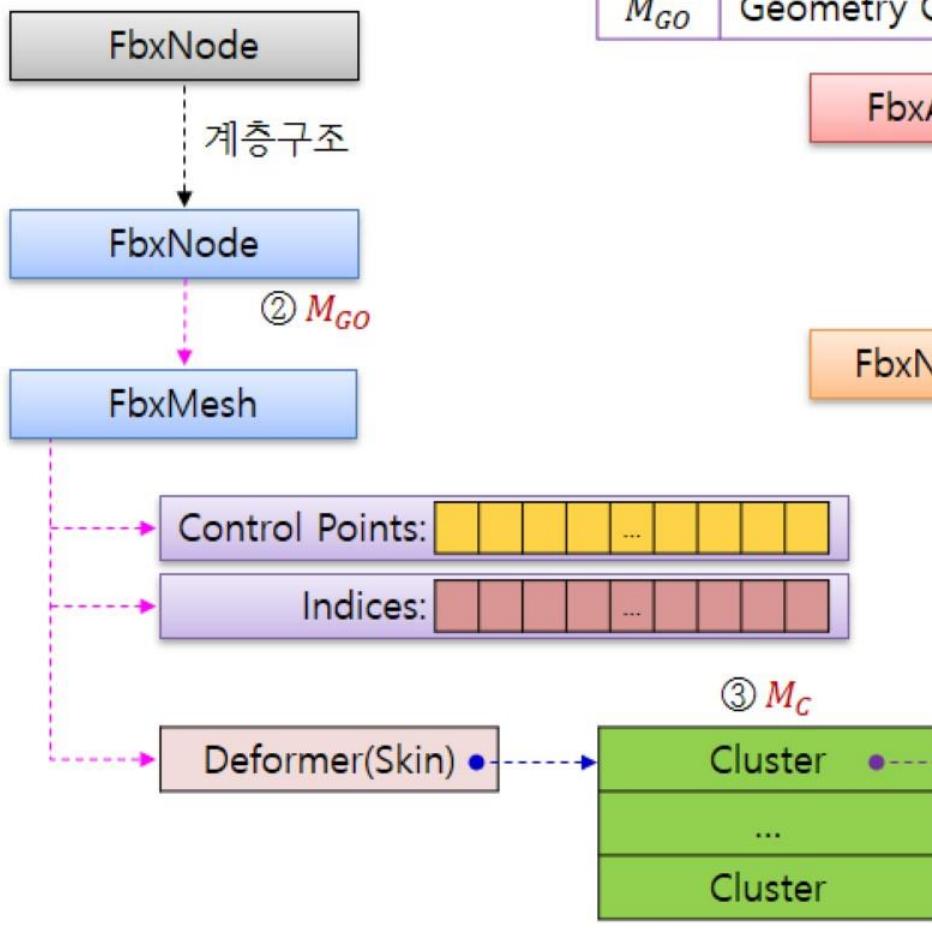
애니메이션(Animation)

- **FBX 스킨 디포머**

- 스킨 메쉬의 경우

$$M_{LK}^{-1} * M_C = \text{Offset Matrix}$$

M_{BG}	Cluster Link Node Global Matrix(BoneToRoot Matrix)
M_{LK}	Cluster Link Transform Matrix(BindPoseLinkNodeToRoot Matrix)
M_C	Cluster Transform Matrix(BindPoseMeshToRoot Matrix)
M_{GO}	Geometry Offset Matrix(MeshToNode Matrix)



FbxAnimation

$$v' = (M_{BG} * M_{LK}^{-1} * M_C * M_{GO}) * v$$

① 애니메이션 데이터 생성

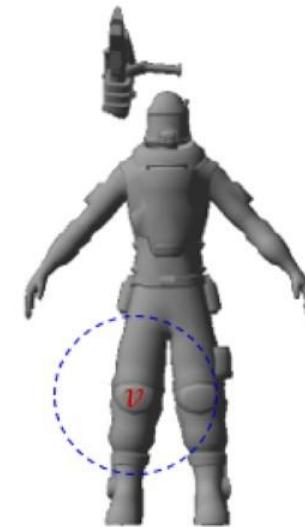
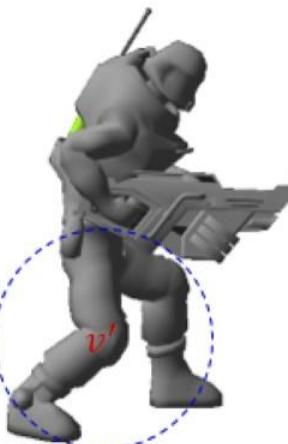
⑤ M_{BG}

④ M_{LK}

Link Node(Bone)

Control Point Indices: [Yellow Boxes]

Control Point Weights: [Yellow Boxes]



애니메이션(Animation)

- FBX 스킨 디포머

```
void ComputeClusterDeformation(FbxMesh *pfbxMesh, FbxCluster *pfbxCluster, FbxTime& rfbxTime,
FbxAMatrix& fbxmtxVertexTransform)
{
    FbxCluster::ELinkMode nClusterMode = pfbxCluster->GetLinkMode();
    if ((nClusterMode == FbxCluster::eNormalize) || (nClusterMode == FbxCluster::eTotalOne))
    {
        FbxAMatrix fbxmtxGeometryOffset = GetAttributeNodeTransform(pfbxMesh->GetNode());
        FbxAMatrix fbxmtxClusterTransform;
        pfbxCluster->GetTransformMatrix(fbxmtxClusterTransform);

        FbxAMatrix fbxmtxClusterLinkTransform;
        pfbxCluster->GetTransformLinkMatrix(fbxmtxClusterLinkTransform);

        FbxAMatrix fbxmtxLinkNodeGlobal = pfbxCluster->GetLink()->EvaluateGlobalTransform(rfbxTime);

        fbxmtxVertexTransform = fbxmtxLinkNodeGlobal * fbxmtxClusterLinkTransform.Inverse() *
        fbxmtxClusterTransform * fbxmtxGeometryOffset;
    }
}
```

M_{BG}	Cluster Link Node Global Matrix(BoneToRoot Matrix)
M_{LK}	Cluster Link Transform Matrix(BindPoseLinkNodeToRoot Matrix)
M_C	Cluster Transform Matrix(BindPoseMeshToRoot Matrix)
M_{GO}	Geometry Offset Matrix(MeshToNode Matrix)

$$\mathbf{v}' = (M_{BG} * M_{LK}^{-1} * M_C * M_{GO}) * \mathbf{v}$$

애니메이션(Animation)

- **FbxShape, FbxBlendShapeChannel, FbxBlendShape 클래스**

- 제어점의 집합에 대한 변형을 표현(Maya의 클러스터 디포머와 유사)
- 기하에 형상(FbxShape)을 추가하면 기하와 형상은 같은 토플로지를 가짐
- 각 블렌드 형상 채널의 블렌드 효과는 가산적(Additive)임
 하나의 블렌드 형상 디포머의 블렌드 효과는 모든 채널의 블렌드 효과의 합임
 각 블렌드 형상 채널의 블렌드 효과는 블렌드 형상 채널의 DeformPercent로 조절

```
FbxBlendShapeChannel *FbxShape::GetBlendShapeChannel();
FbxGeometry *FbxShape::GetBaseGeometry();
int FbxShape::GetControlPointIndicesCount();
int *FbxShape::GetControlPointIndices();
```

```
FbxBlendShape *FbxBlendShapeChannel::GetBlendShapeDeformer();
int FbxBlendShapeChannel::GetTargetShapeCount();
FbxShape *FbxBlendShapeChannel::GetTargetShape(int nIndex);
int FbxBlendShapeChannel::GetTargetShapeIndex(FbxShape *pShape);
double *FbxBlendShapeChannel::GetTargetShapeFullWeights();
FbxPropertyT<FbxDouble> FbxBlendShapeChannel:: DeformPercent;
EType FbxBlendShapeChannel::GetSubDeformerType();
```

```
FbxGeometry *FbxBlendShape::GetGeometry();
int FbxBlendShape::GetBlendShapeChannelCount();
FbxBlendShapeChannel *FbxBlendShape::GetBlendShapeChannel(int nIndex);
EDeformerType FbxBlendShape::GetDeformerType();
```

```
enum FbxSubDeformer::EType { eUnknown, eCluster, eBlendShapeChannel };
enum FbxDeformer::EDeformerType { eUnknown, eSkin, eBlendShape, eVertexCache };
```

애니메이션(Animation)

- **FbxGeometryConverter** 클래스

- 기하 노드의 속성(FbxMesh, FbxNurbs, FbxPatch)을 전환하기 위한 기능 제공
- 삼각형화(Triangulation)와 너브(NURBS)와 패치 표면(Patch Surface)의 전환

```
FbxGeometryConverter(FbxManager* pManager);
```

```
FbxNode *MergeMeshes(FbxArray<FbxNode *> &pMeshNodes, char *pnodeName, FbxScene *pScene);
```

```
bool Triangulate(FbxScene *pScene, bool bReplace, bool bLegacy=false); //다각형을 삼각형으로 분해
```

```
FbxNodeAttribute *Triangulate(FbxNodeAttribute *pNodeAttribute, bool bReplace, bool bLegacy=false);
```

```
FbxNurbs *ConvertPatchToNurbs(FbxPatch *pPatch);
```

```
bool ConvertPatchToNurbsInPlace(FbxNode *pNode);
```

```
FbxNurbsSurface *ConvertPatchToNurbsSurface(FbxPatch *pPatch);
```

```
bool ConvertPatchToNurbsSurfaceInPlace(FbxNode *pNode);
```

```
FbxNurbsSurface *ConvertNurbsToNurbsSurface(FbxNurbs *pNurbs);
```

```
FbxNurbs *ConvertNurbsSurfaceToNurbs(FbxNurbsSurface *pNurbs);
```

```
bool ConvertNurbsToNurbsSurfaceInPlace(FbxNode *pNode);
```

```
bool ConvertNurbsSurfaceToNurbsInPlace(FbxNode *pNode);
```

```
FbxNurbs *FlipNurbs(FbxNurbs *pNurbs, bool bSwapUV, bool bSwapClusters);
```

```
FbxNurbsSurface *FlipNurbsSurface(FbxNurbsSurface *pNurbs, bool bSwapUV, bool bSwapClusters);
```

```
bool EmulateNormalsByPolygonVertex(FbxMesh *pMesh);
```

```
bool ComputeEdgeSmoothingFromNormals(FbxMesh *pMesh);
```

```
bool ComputePolygonSmoothingFromEdgeSmoothing(FbxMesh *pMesh, int nIndex=0);
```

```
bool ComputeEdgeSmoothingFromPolygonSmoothing(FbxMesh *pMesh, int nIndex=0);
```

```
bool SplitMeshesPerMaterial(FbxScene *pScene, bool bReplace); //매쉬마다 하나의 재질을 가지도록
```

```
bool SplitMeshPerMaterial(FbxMesh *pMesh, bool bReplace);
```

애니메이션(Animation)

- 구현 방법
 - FBX SDK를 Direct3D SDK와 함께 사용
 - Exporter를 구현하여 필요한 데이터를 추출
- 좌표계
 - 오른손 좌표계
 - FbxAxisSystem::ConvertScene()
- 정점 와인딩 순서
 - 반시계 방향
- 행렬
 - 열우선 행렬
 - 행렬의 곱 순서
- 예제 프로젝트
 - ImportScene
 - ViewScene