3ds max Game Export Interface Programming Guide

Introduction

처음 생겨났을 때부터 3ds 맥스는 풍부한 API 를 제공하여 응용프로그램의 거의 모든 관점을 제어할 수 있도록 해 왔다. 고성능의 렌더러와 옷감 시뮬레이션과 같은 Stunning 플러그인을 강력하고 확장적인 API 를 사용해 작성할 수 있었다..

3d 개발자 커뮤니티가 성숙되자, 단순히 응용프로그램으로부터 데이터를 추출하는 것에만 관심을 가지고 3ds 맥스를 위한 플러그인을 제작하는 새로운 부류의 개발자들이 생겨 났다. export 개발자의 여명기가 도래했다.

이 개발자들은 이제 3ds 맥스 API 의 본질적인 가파른 학습 곡선에 직면하게 되었다. 상업 플러그인 개발자들에게 있어 이러한 도전은 개발 사이클에 학습 기간을 통합함으로써 고무적인 것이 될 수 있었지만, 빡빡한 일정에 묶인 많은 게임 개발자들에게 있어 이러한 도전은 보통 3ds 맥스 API 와 관련된 지식을 가진 프로그래머를 양산하게 만들었다

게임 개발자들은 단순하고 종합적인 API 를 원하며, 이를 통해 3ds 맥스 게임 요소 데이터를 빠르게 추출해 그것들을 게임 파이프라인의 다른 영역으로 빨리 옮기기를 원한다. 또한 그들은 신입사원이 export 코드에 접근하고 유지보수하기가 용이하도록 작성될 수 있기를 원한다.

Discreet 의 Sparks 개발자 지원팀이 만든 3ds 맥스를 위한 고수준 C++ API 인 3ds max Game Export Interface 의 세계로 들어가 보자.

3ds max Game Export Interface Overview

3ds 맥스 게임 익스포터 인터페이스는 현재 존재하는 3ds 맥스 API 의 상위에 만들어졌으며, 3ds 맥스로부터 데이터를 추출하기 위한 거의 대부분의 관점을 위한 wrapper 인터페이스를 제공한다.

이 익스포트 인터페이스는 다음 3ds 맥스 오브젝트들을 지원한다

Mesh
Spline
Lights and Cameras
Controllers – including Euler, constraints and Biped
IK Chains
Skin deformers
Materials
Texture – including Bitmap Texture
Custom Attributes
Custom Node Data
ParamBlocks

이 인터페이스는 개발자가 3ds 맥스를 사용하는 프로그래밍에 대한 경험이 매우 적더라도 매우 적은 노력으로 exporter 를 작성할 수 있도록 설계되어 있다. custom 인터페이스를 사용하여 개발자는 중요한 맥스 요소에 그것들의 작동 방식에 대해 알지 않고서도 접근할 수 있다. 데이터를 획득하고 사용하는 데 있어서 중요한 것은 대부분의 개발자들은 무엇을 원하는 지는 알고 있어도 어디서부터 찾아야 하는 지는 모른다는 것이다.

Programming with the 3ds max Game Export Interface

3ds 맥스 게임 익스포트 인터페이스는 프로젝트에서 사용되기 위한 라이브러리와 헤더를 가진 DLL 형태로 제공된다. DLL 은 3ds 맥스 디렉토리의 루트에 위치해야만 한다. 이 인터페이스는 MS XML 구현을 사용하므로 IE 5+ 가 시스템에 설치되어 있어야만 한다. IE 5+ 가 3ds 맥스 설치 시디에 포함되어 있다. 프로퍼티(property)를 지원하기 위해서는 제공된 iGameProp.xml 이 plugcfg 디렉토리에 위치해야 한다.

A Game Export 인터페이스와 표준 3ds 맥스 API 사이의 간단한 비교 :

컨트롤러 데이터를 원래 3ds 맥스 API 로 추출하기 :

```
Control * cont = node->GetTMController()->GetPositionController()

if (!cont)          // Bug out if no controller.
          return;

int i;
IKeyControl *ikc = NULL;

ikc = GetKeyControlInterface(cont);

// TCB point3
if (ikc && cont->ClassID() == Class_ID(TCBINTERP_POINT3_CLASS_ID, 0)) {
          for (i=0; i<ikc->GetNumKeys(); i++) {
          ITCBPoint3Key key;
          ikc->GetKey(i, &key);
          // do as you wish with the data

}
```

export 인터페이스를 통해서 컨트롤러 데이터 추출하기 :

```
IGameKeyTab poskeys;
IGameControl * pGameControl = gameNode->GetIGameControl();
pGameControl->GetBezierKeys(poskeys,IGAME_POS));
```

보면 알겠지만 새로운 인터페이스를 통해 데이터에 접근하는 것이 훨씬 간단하고 3ds 맥스에 의해 사용되는 내부 구조체에 대한 이해를 더 적게 요구한다.

Initialising the Game Export Interfaces

익스포트 인터페이스를 사용하기 전에, 메인 IGameScene 에 대한 포인터를 획득함으로써 초기화해야 한다. 이것은 아래 코드를 통해 이루어진다.

```
IGameScene * pIgame = GetIGameInterface();
```

초기화 동안에 좌표계를 지정할 수 있다.
익스포트 인터페이스는 개발자가 Max, OpenGL, DirectX 와 같은 좌표계 표준을 정의할 수 있도록 하는 시스템을 사용한다. 기본적으로 변환 관리자는 오른손 좌표계를 사용하는 3ds 맥스 시스템을 기본값으로 설정한다. 이는 z 축이 위쪽, y 축이 화면쪽이다.

또한 개발자는 오른손이나 왼손으로 정의되지 않고 특정한 축 방향을 설정하기 위한 기능을 가진 사용자 좌표계를 정의할 수도 있다. 변환 관리자는 모든 Matrix 및 Coordinate 데이터가 다른 처리를 하지 않고도 올바른 포맷으로 반환되도록 보증할 것이다.

```
//pIGame 은 IGameScene 의 포인터

IGameConversionManager * cm = GetConversionManager();
cm->SetCoordSystem(IGAME_D3D);
pIgame->InitialiseIGame(true);  // true- we want selected only
pIgame->SetStaticFrame(0);
```

위의 코드는 변환 관리자를 설정해서 내부 데이터베이스가 DirectX 포맷으로 변환되어 사용될 수 있도록 준비하게 한다. It is now a simple case of enumerating the IGameNodes and extracting the data you need.

다음 코드는 사용자 정의 좌표계를 설정하는 방법을 보여준다

```
UserCoord WhackySystem = {
        1,      //Right Handed
        1,      //X axis goes right
        4,      //Y Axis goes in
        3,      //Z Axis goes down.
        0,      //U Tex axis is left
        1,    //V Tex axis is Down
};

IGameConversionManager * cm = GetConversionManager();
cm->SetUserCoordSystem(WhackySystem);
```

INodes 집합에 기반하는 Export 인터페이스도 초기화해야 한다 :
만약 3ds 맥스 씬에 있는 모든 노드로부터 데이터를 수집하기를 원하지 않는다면, 당신이 관심을 가지고 있는 INode(s)를 제공함으로써 단일 노드나 노드의 리스트에 기반해 IGame 을 초기화할 수 있다. 이것은 지 지정된 노드에 대한 포인터만 유지하는 IGameScene 을 당신에게 제공할 것이다.

```
Tab < INode *> interestingNodes;
Interface * ip = GetCOREInterface();

// 선택된 노드에 대한 접근을 제공하기 위해서 정규 맥스 sdk 를 사용한다.

for(int i=0;i<ip->GetSelNodeCount();i++)
{
        interestingNodes.Append(1,&ip->GetSelNode(i);
}

//pIGame is an IGameScene pointer
IGameScene * scene = pIgame->InitialiseIGame(interestingNodes,false);
```

Extracting the Data

IGameScene

IGameScene 오브젝트는 모든 최상위 Node 및 Material 에 대한 리스트와 카운트를 유지한다. Node 나 Material 인 각 최상위 인터페이스는 자식 노드의 리스트를 유지한다.

IGameScene 은 최상위 Node 에 대한 카운트 뿐만 아니라 씬 내의 모든 Node 의 카운트를 유지한다. 이것은 현존하는 3ds 맥스 API 를 사용해 전처리를 수행해야 할 때를 위한 것이다.

IGameNode

IGameNode 는 실제 Object 의 컨테이너인데, 그 Object 나 씬 내의 Material 은 임의의 Controller 와 연관되어 있다

다음 코드는 IGameScene 을 통해서 루트 레벨에서 Material 에 접근하는 방법을 보여 준다. IGameScene::GetRootMaterialCount() 를 한 번 호출해서 씬 내의 Material 의 개수를 획득하고, GetRootMaterial 을 통해 Material 리스트로의 직접 인덱싱을 허용하는 단순한 열거형을 획득한다.

```
//pIGame 은 IGameScene 의 포인터
void IGameExporter::ExportMaterials()
{
        TSTR buf;
        if(exportMaterials)
        {
                int matCount = pIgame->GetRootMaterialCount();
                buf.printf("%d",matCount);
                for(int j =0;j<matCount;j++)
                {
                        IGameMaterial * mat = pIgame->GetRootMaterial(j);
                        if(mat)
                                DumpMaterial(matNode,mat,j);
                }
        }
}
```

Nodes 는 유사한 방식으로 접근된다. 이 예제에서 부모(최상위) Nodes 가 접근되고 추출된다. ExportNodeInfo() 는 게임 인터페이스의 요소가 아니라는 데 주의하라.

```
for(int loop = 0; loop <pIgame->GetTopLevelNodeCount();loop++)
{
        IGameNode * pGameNode = pIgame->GetTopLevelNode(loop);

        if(pGameNode->IsTarget())
                continue;
        ExportNodeInfo(pGameNode);
        pIgame->ReleaseIGameNode(&pGameNode); // free the memory

}
```

IGameNode 는 단순히 오브젝트 컨테이너이기 때문에, GetIGameObject 가 반드시 호출되어야 한다. 이와 유사하게 Node 의 Material 이나 Conrollrer 에 대한 접근도 GetNodeMaterial() 과 GetIGameControl() 로 이루어 진다. 그리고 나서 개발자는 그것에 대한 사용이 끝났을 때 그 오브젝트와 관련된 메모리를 모두 해제하기 위해서 ReleaseIGameObject() 를 호출할 수 있다. IGameObject 의 메모리를 해제하는 것은 Controller 나 Material 데이터에는 영향을 미치지 않는다는 점에 주의하라.

IExportEntity

모든 Game Interface 를 통해서 추출될 수 있는 3ds 맥스 오브젝트들은 IExportEntity 로부터 상속된다.

이 인터페이스는 단일한 파라미터 접근 뿐만 아니라 전용 API 와 함께 지원되는 특별한 ExportEntity 가 어떤 것인지 질의하는 방식도 제공한다. IsEntitySupported() 메서드가 이를 위해 사용된다 – 지원되는 Entity 는 true 를 반환할 것이다. IExportEntity 는 이 메서드를 사용해서 Entity 가 직접적으로 Bitmap Texture 와 같은 게임 인터페이스들 내부로부터 지원되는지 여부를 결정하며, 이 Entity 에 대한 API 를 통한 직접 접근을 제공할 것이다.

다른 예제에서 IGameMesh 오브젝트는 Tri Object 에 대한 직접 지원을 제공한다. 이 경우 IGameMesh 는 모든 geometry-based 오브젝트를 위한 wrapper 이며, IsEntitySupported() 를 호출하는 것은 Tri Object 로 변환될 수 있는 모든 오브젝트에 대해 true 를 산출하고, 그렇지 않는 것들에 대해서는 false 를 산출할 것이다. 이 메서드가 false 를 반환하면, 그것은 단순히 이 오브젝트와 함께 사용되는 전용 API 가 없다는 것을 의미할 뿐이다. 개발자는 IPropertyContainer 인터페이스를 사용해서 여전히 프로퍼티들을 조회(iterate through)할 수 있으며, 그 방식으로 데이터에 접근할 수 있다.

IGameObjects

IGameNodes 는 IGameObjects 의 컨테이너처럼 행동한다 – 그 Objects 를 적절한 프로퍼티를 가지고 있는 실제 물리적 엔터티인 것으로 생각해도 된다.

IGameObjects 는 오브젝트 자체의 유형을 있는 그대로 Camera 나 Light 등으로 정의한다. 그리고 나서 개발자는 아래에 설명한 것처럼 그것의 올바른 인터페이스로 그 오브젝트를 변환해줘야만 한다.

IGameExporter 샘플로부터 코드를 가지고 왔다(명확함을 위해 편집을 했음).

```
// child 는 IGameNode 포인터
IGameObject * obj = child->GetIGameObject();
switch(obj->GetIGameType())
{
        case IGameObject::IGAME_MESH:
                if(exportGeom )
                {
                        IGameMesh * gM = (IGameMesh*)obj;
                        if(gM->InitializeData())
                        {
                                DumpMesh(gM,geomData);
                        }
                        else
                        {
                                DebugPrint("Bad Object₩n")
                        }
                }
                break;
        case IGameObject::IGAME_SPLINE:
                if(exportSplines)
                {
                        IGameSpline * sp = (IGameSpline*)obj;
                        sp->InitializeData();
                        DumpSpline(sp,splineData);
                }
                break;
}
child->ReleaseIGameObject();
```

이 예제에서 우리는 GetIGameType() 을 IGameObject 에 대해 호출하고 IGameObject 를 관련 인터페이스로 변환한다. IGameObject Types 와 관련 인터페이스의 리스트는 아래의 표에 나와 있다. 이 표는 부록에서도 볼 수 있다.

IGame Object Type	IGame Interface
IGAME_UNKNOWN	IGameGenObject
IGAME_LIGHT	IGameLight
IGAME_MESH	IGameMesh
IGAME_SPLINE	IGameSpline
IGAME_CAMERA	IGameCamera
IGAME_HELPER	IGameHelperObject
IGAME_BONE	IGameHelperObject
IGAME_IKCHAIN	IGameIKChain

메모리를 절약하기 위해 IGameObject 는 InitializeData 멤버를 가진다. 이것은 IGame 에 실제 Object 변환을 시작할 것을 통보한다. Editable Meshes 와 같은 오브젝트들은 많은 메모리를 소비할 수 있으며 이 변환은 개발자의 요구가 있을 대 발생한다. 예를 들어 단일 노드 변환 데이터만이 요구되는 경우, InitializeData 는 호출될 필요가 없을 것이다.

IGameMesh

IGameMesh 오브젝트는 법선, 매핑 채널, 정점 정보, 텍스처 좌표 등을 포함하는 모든 메시 관련 데이터에 대한 완벽한 접근을 제공한다.

It will also take care of such things as mirroring, which is again something that can cause unexpected results when exporting as the normals will appear to be flipped.

IGameMesh 포인터를 한 번 획득하면 데이터를 추출하기 위해서 다양한 메서드들을 사용할 수 있다. (매핑 채널 데이터를 제외한) 대부분의 면 데이터가 FaceEx 라 불리는 확장 면 클래스에서 제공된다. 이것은 3ds 맥스 클래스인 Face 의 확장이며 정점 색상, 알파, 법선, 조명을 위한 부가적인 인덱싱을 포함한다. 매핑 채널은 따로 다뤄진다.

다음 예제는 주어진 메시의 정점 및 면 데이터를 추출하는 방법에 대해 보여 준다. 다른 데이터형들도 비슷한 방식으로 추출된다.

```
int numVerts = gm->GetNumberOfVerts();
for(int i = 0;i<numVerts;i++)
{
	TSTR data;
	Point3 v;
	if(gm->GetVertex(i,v))
	{
		//use the data here
	}

}
int numFaces = gm->GetNumberOfFaces();
for(int f=0;f<numFaces;f++)
{
	FaceEx * face = gm->GetFace(f);
	// use data here
}
```

IGameMesh 는 smoothing 그룹에 기반한 Face 데이터에 대한 접근도 제공한다. 모든 개별 smoothing 그룹에 대해서 면의 리스트가 반환될 수 있다. 또한 이것은 면으로부터의 Material ID 에 기반한 전역 재질 리스트에 대한 재질 색인을 가진 Material ID basis 당 면에 대해서도 잘 작동한다. 이러한 유형의 접근은 하드웨어가 multi-material 에 대한 개념을 가지고 있지 않아서 메시가 면에서의 재질에 기반해 더 작은 청크로 분할될 필요가 있을 때 데이터를 실시간으로 디스플레이 하기 위해서 중요하다. 이것은 부록에서 더 많이 다루고 있다.

오브젝트 상에서 검색되는 모든 매핑 데이터가 수집되고 저장된다. 사용자는 단지 어떤 채널이 데이터를 가지고 있는지 질의하고, 다른 데이터 채널과 유사한 방식으로 그것들에 접근하기만 하면 된다. 일반적으로 사용자는 GetMapFace() 와 GetMapFaceIndex() 를 호출해 모든 데이터에 접근한다.

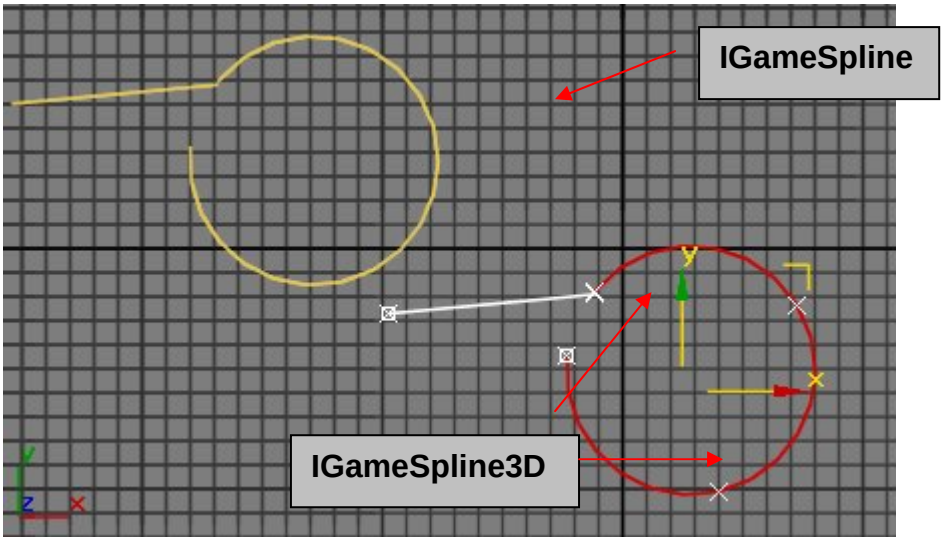
```
//gm 은 IGameMesh 의 포인터
if(exportMappingChannel)
{
	Tab<int> mapNums = gm->GetActiveMapChannelNum();
	int mapCount = mapNums.Count();

	for(int i=0;i < mapCount;i++)
	{

		int vCount = gm->GetNumberOfMapVerts(mapNums[i]);
		buf.printf("%d",vCount);
		for(int j=0;j<vCount;j++)
		{
			vert = NULL;
			Point3 v;
			if(gm->GetMapVertex(mapNums[i],j,v))
			{
				//use data here
			}

		}
		int fCount = gm->GetNumberOfFaces();

		for(int k=0;k<fCount;k++)
		{
			DWORD  v[3];
			gm->GetMapFaceIndex(mapNums[i],k,v);
			//use data here
		}

	}

}
```

IGameSpline

모든 스플라인 기반 IGameObject 는 IGameSpline 오브젝트로 변환될 수 있다. 이 Object 는 하나 혹은 그 이상의 IGameSpline3D 오브젝트로서 표현되는 실제 스플라인 데이터를 위한 컨테이너로서 행동하며, 이는 얼마나 많은 스플라인이 원래의 3ds 맥스 스플라인을 구성하는지에 달려있다.

knot 과 tangent 데이터는 IGameSpline3d Object 에 저장된다. 기본 knot 데이터와 함께 IGameSpline3D 또한 knot 혹은 tangent 핸들의 애니메이션을 저장할 것이다.

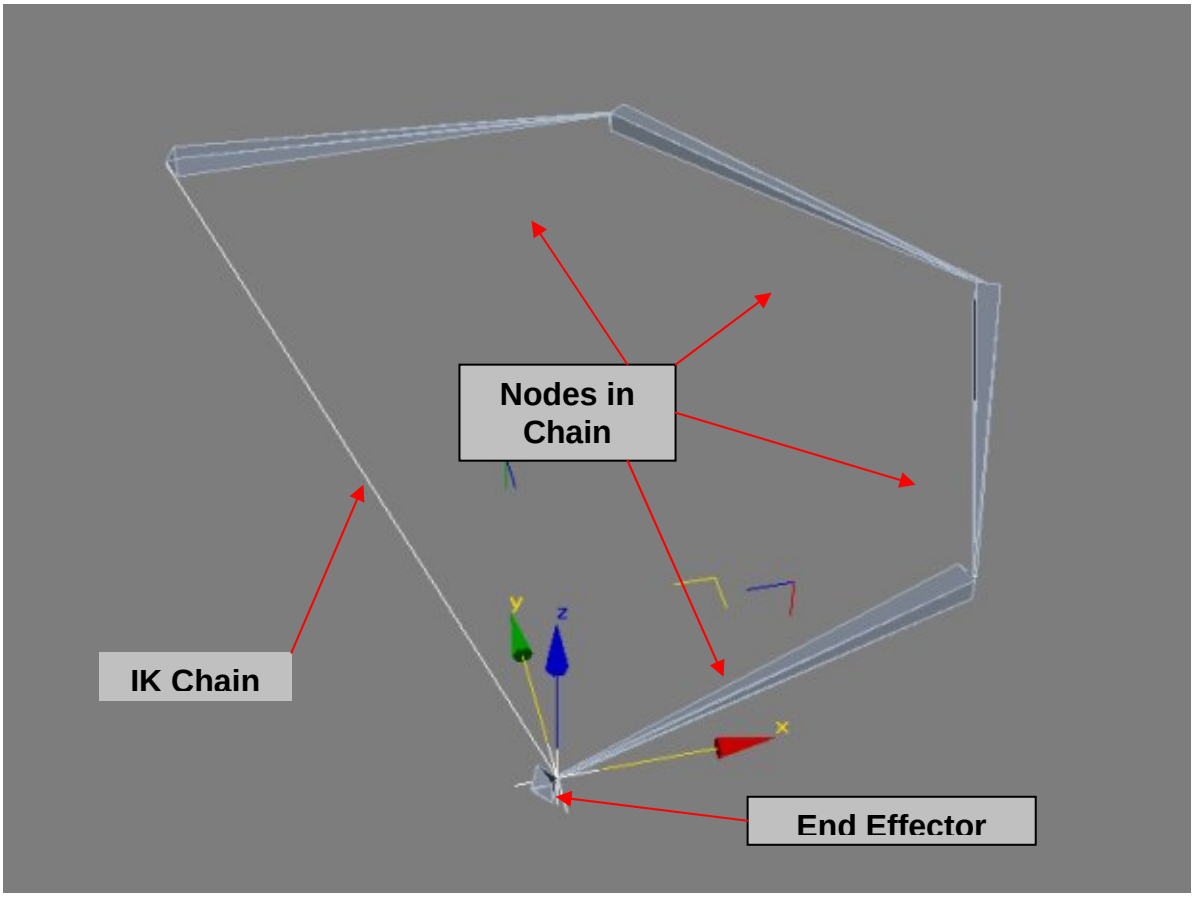
아래의 다이어그램은 IGameSpline 컨테이너 Object 와 그것이 포함하는 IGameSpline3D Object 들의 관계를 나타낸다



스플라인은 path 기반 애니메이션을 위한 Constraints 와 혼합되어 사용되기도 하며, 그것들은 일반적으로 Constraint 데이터의 일부분으로서 추출된다.

IGameIKChain

IGameIKChain 은 Inverse kinematic(IK) 체인에 대한 직접 접근을 제공한다. 이것은 캐릭터가 IK 설정에 근거해 추출될 수 있다는 것을 의미한다. 각 IK 체인은 여러 개의 연결된 노드로 구성된다. 그리고 나서 이들 노드는 IK 나 Forward Kinematic(FK) 모드로 실행되는 end effector 의 제어하에 놓이게 된다. 이 모드는 IGameIKChain 인터페이스를 통해서 접근할 수 있는 On/Off 컨트롤러에 의해서 결정된다.



IK 가 활성활될 때 IGameNode 로부터 획득한 IGameControl 은 end effector 를 위한 컨트롤러가 될 것이다. IK 가 비활성화될 때 Forward kinematics 는 IK 체인을 구성하고 있는 노드로부터의 IGameController 에 의해 계산될 수 있다. IK Enabled 컨트롤러와 함께 Seivel angle 데이터에 대한 직접 접근이 IGameProperty 로서 제공된다.

IGameCamera 과 IGameLight

모든 Light 와 Camera 프로퍼티는 IGameProperty 로서 추출된다. 이것은 모든 애니메이션된 조명 오브젝트와 multiplier 데이터, 그리고 조명을 위한 include/exclude 데이터를 포함한다. 모든 3ds 맥스 조명 및 카메라는 이 인터페이스를 통해 추출되며, 그것들은 공통 데이터를 "공유"한다. 대상에 대한 노드와 함께 조명 및 카메라 유형도 제공된다.

IGameSupportObject

Bone 을 포함하는 모든 Helper 오브젝트들은 IGameSupportObject 로서 wrap 된다. 이것은 매우 단순한 인터페이스이며, 대부분 메인 IPropertyContainer 인터페이스에 대한 접근을 제공한다. 그러나 Helper 의 기하도형에 대한 부가적인 접근이 IGameSupportObject 에 의해 유지되는 IGameMesh 포인터를 통해서 제공된다.

IGameGenObject

게임 인터페이스에 알려지지 않은 모든 오브젝트는 IGameGenObject 로서 wrap 된다. 그 Object 의 기본 프로퍼티에 대해서만 접근이 가능하다. Object 프로퍼티에 대한 접근은 다음 장에 설명되어 있다.

Modifiers

3ds 맥스의 모디파이어 스택은 매우 위압적일 수 있으며, API 내부적에서 특히 혼란스럽게 여겨질 수 있다. 개발자가 그것에 대해서 제대로 이해하지 않고서 접근하는 것은 스택을 불안정한 상태로 두고 떠나는 잠재적인 요소가 있어서 위험하다. 이것은 개발자가 버그를 수정하기 어렵다는 것만을 의미하는 것이 아니라, 이해의 부족 때문에 샘플 코드의 이면에 있는 기능을 확장하기 더 어렵다는 것을 의미하기도 한다.

3ds 맥스 게임 익스포터 인터페이스는 상속 오브젝트나 내부 작동에 대한 지식을 알지 않고도 IGameObject 인터페이스를 통해서 모디파이어에 대한 직접 접근할 수 있도록 해 준다. IGameObject 는 오브젝트에 적용된 모든 모디파이어의 리스트와 함께 스택에서 사용된 이름 및 내부 이름에 대한 접근을 유지하며, 결국 개발자는 간단하게 모디파이어가 필요한지 여부를 결정할 수 있다.

이에 대한 확장으로서 IGameModifier 는 "max skin" 과 "physique" 의 스킨 변형 모디파이어를 직접적으로 지원한다. 심지어 두 모디파이어는 자신만의 API 를 제공하며, 둘 다 서로 다르다. IGame 은 두 스킨 모디파이어에 대한 단일한 접근을 제공하는데, IGameModifier::IsSkin() 을 사용하며, 개발자는 IGameSkin 을 변환해서 weights 와 bone bindings 에 직접 접근할 수 있다.

IGameModifier 는 여전히 원래의 3ds 맥스 모디파이어에 대한 접근을 제공한다. 왜냐하면 일부 데이터는 IGame 이 추출할 수 없는 방식으로 정의된 사용자 데이터이기 때문이다. 이것은 보통 LocalModData 나 Per Face Data 의 형태이며, 반드시 그것을 작성한 개발자에 의해 직접적으로 접근되어야만 한다.

```
int numMod = igameObj->GetNumModifiers();
if(numMod > 0)
{
        for(int i=0;i<numMod;i++)
        {
                IGameModifier * m = igameObj->GetIGameModifier(i);
                // 여기에서 데이터에 접근.....
        }
}
```

IGameModifier 인터페이스는 모디파이어에 의해 영향을 받은 모든 노드에 대한 리스트도 제공한다. 그래서 익스포트시에 당신은 Physique 와 같은 모디파이어에 기반한 노드들을 익스포트할지 선택할 수 있다.

Controllers

IGameControl 은 3ds 맥스의 모든 표준 키프레임 컨트롤러에 대한 직접 접근을 제공한다. 이것은 게임 개발자들에게 특히 유용한데, 그들은 자신들의 게임 엔진에 의해 정의된 대로 추출하고 있는 컨트롤러의 유형이 무엇인지 알고 있기 때문다.

이를 위해 표준 3ds 맥스 API 는 위치 컨트롤러에 대한 질의를 하고, 요청된 key 인터페이스를 소유하고 있는지 여부를 검사하고, 그리고 나서 컨트롤 타입에 기반한 키를 추출할 것을 요구할 것이다. 3ds 맥스 게임 익스포터 인터페이스는 이것을 자동적으로 다루는데, 이는 개발자로 하여금 간단하게 익스포트를 위한 모든 Bezier 위치 키의 리스트를 요청할 수 있도록 해 준다. 다시 말하지만 이 수준의 추상성은 데이터가 3ds 내부에 대한 세부 지식 없이도 빨리 접근될 수 있음을 의미한다.

기본 Key 프레임 컨트롤러와 함께 IGameControl 도 컨트롤러를 샘플링할 수 있다. 이것은 Key 인터페이스를 지원하지 않는 Biped 와 같은 특정 컨트롤러에 있어서 중요하다. 각 IGameControl 인터페이스는 개발자가 IGameControl 컨트롤러가 변환스타일에 기반해 어떤 유형으로 저장되었는지를 알아내도록 허용하는 컨트롤 유형을 유지한다. 예를 들어 Control 이 Unsupported 컨트롤러나 Biped 이면, 개발자는 그것을 원하는 비율로 샘플링할 수 있다. IGame 은 두 개의 샘플링 유형인 Full 과 Quick 를 지원한다. Full 은 지정된 개발자에 의해서 샘플링율로 전체 애니메이션 범위의 애니메이션을 샘플링할 것이다. Quick 는 단지 키가 존재하는 컨트롤러만을 샘플링할 것이다. 이것은 IGAME_TM 샘플링이나 키 설정을 지원하지 않는 컨트롤러에 대해서는 작동하지 않을 것이라는 제약이 있다.

프로그래밍 로직을 쉽게하기 위해서 IGameControl Get[...]Kyes() 메서드는 불리언 형태로서 성공값을 반환한다.

이는 모든 키 메서드를 통한 단순 iteration 을 허용한다. 즉 만약 모두 false 를 반환한다면, 어떠한 키도 발견되지 않은 것이고, 그 다음에 샘플링이 요청될 수 있다.

IGameController 유형 및 관련 의미를 알기 위해서는 아래의 표를 살펴 보라. 이 표는 부록에서도 볼 수 있다.

IGame Controller Type	Max Controller
IGAME_UNKNOWN	Procedural Controller / 3rd Party

IGAME_MAXSTD	Linear, TCB, Bezier
IGAME_BIPED	Biped Controller
IGAME_ROT_CONSTRAINT	Orientation and Look At Constraint
IGAME_POS_CONSTRAINT	Path and Position Constraint.
IGAME_LINK_CONSTRAINT	A Matrix3 TM constraint
IGAME_LIST	A List Controller

Constraints 도 IGameControl 에 의해 직접적으로 지원된다. 스킨 접근과 마찬가지로 모든 Constraints 는 단일한 인터페이스를 통해 지원된다. Constraints 는 Controller 유형으로서 식별된다; 만약 발견되면 개발자는 단순히 개별 컨트롤러 유형에 대한 Constraint 인터페이스를 획득할 필요가 있다. 다음 코드는 이를 실제로 보여 준다.

```
if(exportControllers)
{
	IGameKeyTab posKeys,rotKeys,
	IGameControl * pGC = node->GetIGameControl();
	DWORD T = pGC->GetControlType(IGAME_POS);
	// 위치
	if(T==IGAME_MAXSTD && pGC->GetBezierKeys(poskeys,IGAME_POS))
		DumpBezierKeys(IGAME_POS,poskeys,prsNode);

	else if(T==IGAME_POS_CONSTRAINT && !forceSample)
	{
		IGameConstraint * cnst = pGC->GetConstraint(IGAME_POS);
		DumpConstraints(prsNode,cnst);
	}

	// 회전
	else if(T==IGAME_MAXSTD && pGC->GetTCBKeys(rotkeys,IGAME_ROT)
	{
		DumpTCBKeys(IGAME_ROT,rotkeys,prsNode);
	}

	else if(T==IGAME_ROT_CONSTRAINT && !forceSample)
	{
		IGameConstraint * cnst = pGC->GetConstraint(IGAME_ROT);
		DumpConstraints(prsNode,cnst);
	}

	else
		DumpSampleKeys(child,prsNode);  // usually Biped
	}
}
```

IGameKey

추출된 모든 키는 단일 저장소 클래스에 저장되는데, 이는 모든 Key 유형에 대한 리스트를 유지한다 : Bezier, TCB, Linear, Sampled. IGameKey 에 대한 클래스 정보는 부록을 참조하라.

데이터에 대한 접근은 컨트롤러 유형에 달려 있다. 아래 코드는 Position 에 대한 모든 Bezier Key 를 획득한다 ;

```
IGameKeyTab BezKeys;
If(pGC->GetBezierKeys(BezKeys,IGAME_POS))
{
	for(int i = 0;i<BezKeys.Count();i++)
	{
		if(Type==IGAME_POS || Type==IGAME_POINT3)
		{
			Point3 k = BezKeys[i].bezierKey.pval;
			//use data here
		}
	}
}
```

이 샘플에서는 Bezier Position 데이터에 관심을 가지고 있기 때문에, GetBezierKeys() 메서드가 IGAME_POS 플래그와 함께 사용되었다. 만약 성공하면 BezKeys 는 bezierKey 멤버에 접근할 수 있는 키의 리스트를 보유하게 될 것이다. 이 리스트는 IGameKey 에 의해 유지된다.

Materials 및 Textures

Material 은 IGameMaterial 로서 전역으로 저장되며, 각 노드는 이 전역 배열에 대한 인덱스를 제공하고 있다. Standard Max 재질(StdMtl2) 는 Ambient, Diffuse 색상, Glossiness, Opacity 와 같은 기본 재질 데이터를 획득하기 위한 메소드를 사용해 직접적으로 지원된다. 다른 재질은 요소는 IPropertyContainer 에 대한 접근을 제공할 것이며, 이를 통해 Parameter 추출이 가능하다. 이 경우 API 접근을 위한 사용자 집합을 정의하기 위해 xml 프로퍼티 파일을 사용하는 것이 최선이다. 이에 대해서는 다음 장에서 설명하고 있다.

Multi Material 은 SubObject 레벨 상에서 사용되고 있는 sub 재질들과 Material ID 에 대한 접근과 함께 지원된다. 재질들은 Bitmap Texture 에 대한 직접 지원과 함께 그것이 사용하고 있는 모든 텍스처에 대한 접근을 제공한다. 클리핑 데이터를 포함하는 Bitmap Texture 의 모든 프로퍼티가 지원된다.

```
// mat 은 IGameMaterial 에 대한 포인터.
TSTR buf;
int texCount = mat->GetNumberOfTextureMaps();

for(int i=0;i<texCount;i++)
{
	IGameTextureMap * tex = mat->GetIGameTextureMap(i);
	TCHAR * name = tex->GetTextureName();
	if(tex->IsEntitySupported())	//its a bitmap texture
	{
		IGameProperty * prop = tex->GetClipHData();
		DumpProperties(bitmapTexture,prop);

		prop = tex->GetClipUData();
		DumpProperties(bitmapTexture,prop);

		prop = tex->GetClipVData();
		DumpProperties(bitmapTexture,prop);
```

```
                prop = tex->GetClipWData();
                DumpProperties(bitmapTexture,prop);

        }

}
```

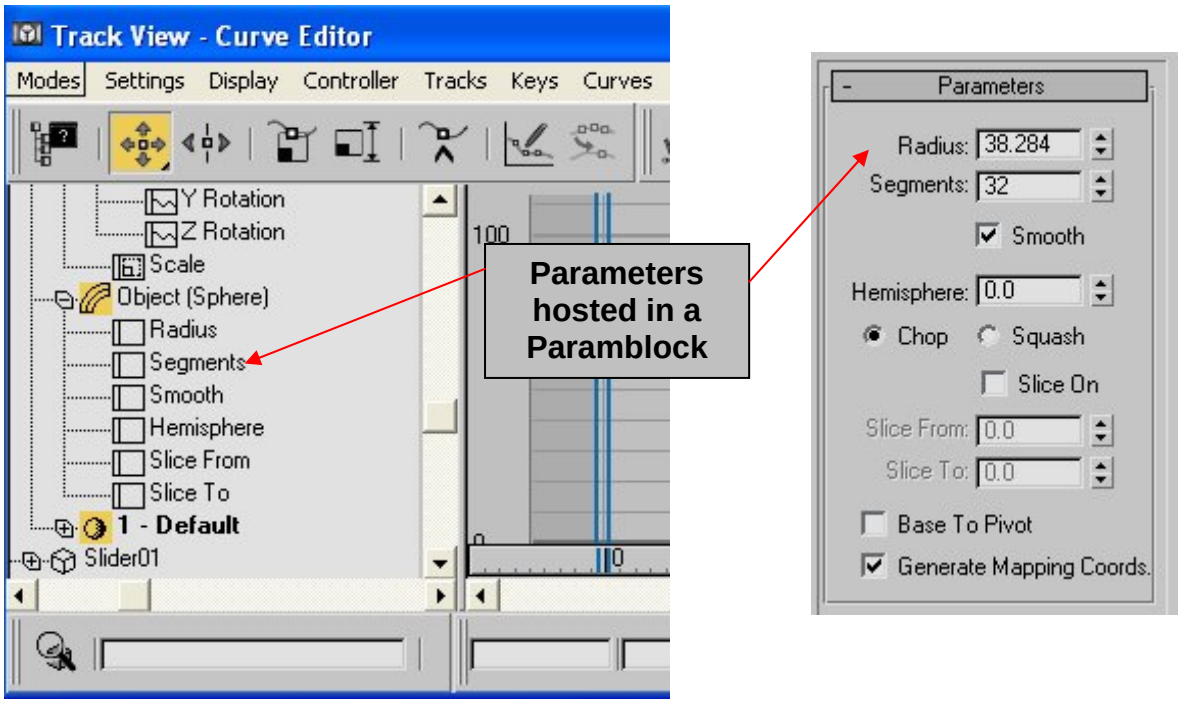
Texture 도 변환을 다루기 위해서 텍스처에 의해 사용된 Coordinate 롤아웃 데이터에 대한 접근을 제공한다. 전체 변환이나 변환을 구성하는 개별 요소에 접근할 수 있다.

Parameter Blocks 과 User Defined Data 지원

Parameter blocks 혹은 ParamBlocks 은 3ds 맥스 프로그래밍에 있어 핵심 부분이다:

3ds 맥스에서 거의 대부분의 파라미터들은 ParamBlock 로서 등록된다. 이것은 개발자가 핵심 응용프로그램을 제어하기 위한 UI 와 데이터를 신경쓰지 않도록 해 준다. 예를 들어 TrackView 에는 많은 편집 가능한 데이터가 ParamBlock 시스템에 의해 구체화(driven)되었다.

아래 예제는 다른 위치에서 같은 paramblock 에 접근하는 것의 강력함에 대해 설명하고 있다. 이 경우 데이터가 모디파이어 다이얼로그에서 디스플레이되고 있음에도 불구하고, TrackView 는 그것의 ParamBlock 에 접근함으로써 모디파이어의 파라미터에 직접적으로 접근한다.



IGame 을 위한 프로그래밍을 할 때 ParamBlock 프로퍼티는 IPropertyContainer 인터페이스를 통해 획득된다 :

게임 익스포트 인터페이스 데이터베이스에 저장된 각 IExportEntity 는 IPropertyContainer 를 지원한다. 이 인터페이스는 실제적으로 Entity 상의 프로퍼티를 검색할 책임이 있다 – 그것은 ParamBlocks 가 저장될 수 있는 모든 다양한 위치를 돌면서, if applicable, Custom Object Data 와 Custom Attributes 를 획득할 것이다. 개발자는 플러그인 지정 파라미터를 추출하고자 할 때 이 인터페이스를 요청해야 한다.

IGameGenObjects 도 IPropertyContainer 를 지원하는데, 게임 인터페이스에 알려지지 않은 3ds 맥스 오브젝트의 프로퍼티에 대한 접근을 부여한다.

IGameScene 은 두 개의 약간 다른 방식으로 플러그인이나 IGameObjects 에 유지되는 ParamBlocks 로부터 파라미터를 추출한다 :

XML Defined

이 메서드는 개발자가 Custom Attributes, Custom Object Data, 개발자 정의 플러그인 데이터를 열거없이 직접 추출할 수 있도록 해 준다.

Simple Enumeration

이 메서드는 ParamBlock 에 대한 직접 제어를 허용하는데, read/write 를 포함한다. 그리고 다양한 엔터티의 모든 프로퍼티에 대한 열거를 하는 가장 빠른 방식이기도 하다.

이들 메서드는 모두 IGameProperty 로서 wrap 되어 있는 원하는 파라미터 데이터를 제공할 것이다. 어떠한 메서드를 사용하느냐는 제어의 수준과 추출할 데이터의 양에 달려 있다.

예를 들어 XML Defined route 는 모든 paramblock 파라미터와 함께 3ds 맥스 오브젝트의 Object Properties 다디얼로그에 할당되어 있는 Custom Attributes 와 Custom Object Data 도 제공할 것이다. Custom Attributes 는 이 route 를 통해서 지원될 수도 있다. 개발자들은 게임 익스포트 인터페이스를 초기화할 때, 사용할 파일의 이름을 지정할 수 있다. 즉 이것은 게임 지정 데이터를 유지하는 것을 허용한다.

XML Defined

customisation 을 허용하고 알려지지 않은 프로퍼티를 지원하기 위해서, 익스포트 인터페이스는 Properties 파일을 사용한다. 이것은 XML 파일인데, 그 내부에 개발자가 추출하고자 하는 파라미터들을 기술한다. 이 시스템을 사용하는 것의 이점은 새로운 프로퍼티들을 추가할 때, 그것을 명시적으로 지원하기 위해서 IGame 이 갱신될 필요가 없다는 것이다. IGameProp.xml 파일의 명세(description)는 부록에서 찾아볼 수 있다.

XML 의 형식은 매우 단순하며, 손으로 편집하는 것이 가능하다. 그러나 IGame Property Editor 유틸리티를 사용해서 새로운 Properties 및 Custom data 를 실패 가능성이 없는 방식으로 추가할 수 있다. 저장된 데이터는 프로퍼티에 대한 충분한 정보를 제공하므로, IGame 은 원래의 ParamBlock 으로부터 그것을 추출할 수 있다. XML 파일도 IPropertyContainer 인터페이스로부터 파라미터를 질의하는데 사용될 수 있는 파라미터에 대한 단일 식별자를 저장한다. 이것은 개발자가 자신만의 API 를 생성해서 특정 부가 유효화(validation)을 수행할지도 모르는 프로퍼티들을 추출할 수 있음을 의미한다. XML Property Definition 파일의 예가 아래에 나와 있다.

```
<ExportEntity>
<!--
 StdMat2 Properties
  -->
<simplename>StdMat2</simplename>
  <classid>0x0000002,0</classid>
```

```
  <shared>false</shared>
<Property>
<id>1</id>
<simplename>Ambient</simplename>
<name>ambient</name>
  <type>point3</type>
  <paramblock2>true</paramblock2>
  </Property>
<Property>
<id>2</id>
<simplename>Diffuse</simplename>
  <name>diffuse</name>
  <type>point3</type>
  <paramblock2>true</paramblock2>
  </Property>
</ExportEntity>
```

위의 초기화에서 IGameScene 은 이 파일을 로드하여 씬 순회동안에 그것과 엔터티들을 매치시키기 위해서 파싱을 할 것이다. It makes a match based on the ClassID of the node and the SubAnimName for IParamBlock or the internal name stored in the ParamBlockDesc2 for IParamBlock2 (shown red in the source below).

```
static ParamBlockDesc2 aniso_param_blk ( aniso_params, _T("shaderParameters"),  0,
&anisoCD, P_AUTO_CONSTRUCT, 0,
        // params
        an_ambient, _T("ambient"), TYPE_RGBA, P_ANIMATABLE, IDS_AMBIENT,
                p_default, Color(0, 0, 0),
                end,
        an_diffuse, _T("diffuse"), TYPE_RGBA, P_ANIMATABLE, IDS_DIFFUSE,
                p_default, Color(0.5f, 0.5f, 0.5f),
                end,
```

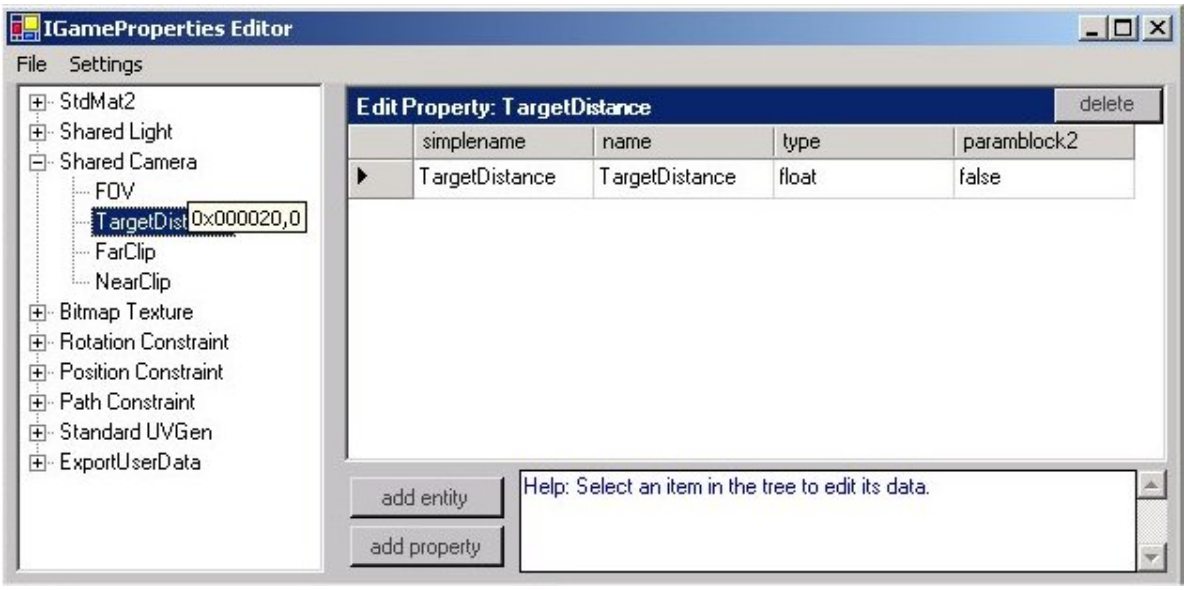
그리고 나서 개발자는 이름이나 ID 에 의해 프로퍼티를 요청할 수 있다.

IPropertyContainer 인터페이스에 대한 포인터가 주어지면, 다음과 같이 wrapper 함수를 작성할 수 있다.

```
IPropertyContainer * pc = entity-> GetIPropertyContainer ();
IGameProperty * prop = pc->QueryProperties(_T("Ambient"));

또는

#define AMBIENT 1
IGameProperty * prop = pc->QueryProperties(AMBIENT);
```

IGame Property Editor 툴을 사용하는 것은 #define 으로 표현되는 모든 ID 들을 include 파일에 저장할 수 있도록 해 준다. 게임 익스포트 인터페이스의 많은 메서드들은 단순히 파라미터들에 대한 wrapper 이며, 정확하게 같은 방식으로 구현되었다. 실제 ID 는 에디터에 의해 생성되며 일부 정의는 자신만의 검색을 위해서 그것들을 사용하게 될 내부 코드이므로 변경이 불가능하다. 이것은

Materials, BitmapTextures, 조명 및 카메라를 포함한다.



XML 기반 추출의 또 다른 중요한 관점은 Custom Object 데이터가 IGameProperty 로서 추출되고 표현될 수 있다는 것이다. 데이터가 정의되면, 그것은 다른 프로퍼티와 정확하게 같은 방식으로 접근이 가능하다.

Custom Object 데이터는 다음 리스트에서 볼 수 있는 다른 ParamBlock 파라미터들과는 다소 다르게 정의된다.

```
<ExportUserData>
<!--
 Node User Data 에 대한 접근
<!--
 id 는 파일에 대해 유일해야 한다

  <id>100</id>
  <keyName>IGameTest</keyName>
  <type>float</type>
  </UserProperty>
<UserProperty>
<id>101</id>
<simplename>IGameTestString</simplename>
  <keyName>IGameTestString</keyName>
  <type>string</type>
  </UserProperty>
  </ExportUserData>
```

User Property 를 반환하는 IGameProperty 사이의 차이는 단지 컨트롤러 인터페이스가 NULL 이라는 것 뿐이다.

Simple Enumeration

모든 ExportEntity 는 열거되는 자체 파라미터를 소유할 수 있다. 개발자는 단지 사용할 IPropertyContainer 에 대한 콜백을 등록하기만 하면 된다. 그러면 모든 ParamBlock 기반 파라미터 요소들은 콜백을 통해 개발자에게 반환될 것이다.

DumpProperty 샘플은 이 기법을 사용해서 오브젝트 상에서 찾을 수 있는 모든 프로퍼티의 이름을 추출한다. 이것은 모디파이어, 재질, IPropertyContainer 인터페이스를 지원하는 모든 것을 포함한다. 다음 코드는 콜백을 설정하는 방법을 보여 준다.

```
class MyProc : public PropertyEnum
{
        DumpProperties * dp;

public:

        MyProc(DumpProperties * d) {dp = d;}

        bool Proc(IGameProperty * prop)
        {
                TCHAR Buf[256];
                _tcscpy(Buf,prop->GetName());

// NULL 이면 그것의 IParamBlock 때문이며,
// 그것은 애니메이션되지 않는다. 즉 이름을 가지고 있지 않다.
                if(_tcscmp(Buf, _T(""))!=0){
                        //use the Property here
                }
                return false;     // 계속 진행
        }
};
```

이 콜백은 다음코드를 사용하여 설정되는데, 여기에서 material 은 IGameMaterial 오브젝트에 대한 포인터이다.

```
IPropertyContainer *pCont = material->GetIPropertyContainer();
MyProc * proccy = new MyProc(this);
pCont->EnumerateProperties(proccy);
```

이 기법은 초기화 처리 중에 발견되는 모든 파라미터에 대한 접근을 제공할 것이다. 이 기법은 모든 파라미터에 대한 기반 ParamBlock 에 대한 접근을 원할 때 유용하다. 즉 그 데이터에 대한 읽기 및 쓰기 권한을 제공한다. 또한 이것은 다양한 엔터티로부터 모든 파라미터를 획득하는 가장 빠른 방법이기도 하다. 개발자는 저수준 접근을 사용하고자 결정했다면 원래의 3ds 맥스 API 가 사용하는 정확한 방식으로 ParamBlock 에 접근할 책임이 있다.

에러 기록

게임 익스포트 인터페이스는 표준 "GetLastError" 기법을 사용하는데, 이는 치명적인 문제에 대한 피드백을 제공하기 위해서 윈도우즈 API 가 사용하는 것과 유사하다. 대부분의 IGame 함수는 호출 코드로부터 검사할 수 있는 전역 에러를 설정할 것이다. 세부적인 에러 문자열을 에러가 발생한 이후에 기술하는 것도 가능하다. 이러한 접근은 개발자로 하여금 에러 검사가 자신의 코드에서

얼마나 많이 수행될 것인지를 결정할 수 있게 해 준다. 예를 들어 GetLastIGameError 를 사용하는 것은 IG_NODE_NOT_FOUND 가 반환되는 결과를 낳을 수 있고, GetLastIGameErrorText 를 사용하는 것은 "No Target Node found" 를 산출할 것이다. 즉 개발자가 target-based 가 아닌 조명이나 카메라에 대한 target 노드를 요청했을 때를 의미한다.

등록될 수 있는 콜백도 존재하는데, 이를 통해 발생할 수 있는 모든 Error 를 통지받게 된다. 이 이벤트는 전역 에러가 설정될 때 발생해, 문제가 실제로 반환될 때 메서드 호출 이전에 잡히도록 해 준다. 이 기법은 익스포트가 완료된 이후에 사용자를 위한 전체 에러 덤프를 제공하기에 편리한 방식이다.

```
class MyErrorProc : public IGameErrorCallBack
{
public:
	void ErrorProc(IGameError error)
	{
		TCHAR * buf = GetLastIGameErrorText();
		DebugPrint("ErrorCode = %d ErrorText = %s₩n", error,buf);
	}
};

MyErrorProc pErrorProc;
SetErrorCallBack(&pErrorProc);
```

현존하는 코드의 통합

유연성을 제공하기 위해서 게임 익스포트 인터페이스는 현존하는 파이프라인 및 코드에 맞춰질 수 있어야 한다. 이것은 모든 IGame 인터페이스로부터 원래의 3ds 맥스 컨테이너에 대한 접근을 제공함으로써 이루어 진다.

IGame 은 다양한 초기화 옵션을 제공해서 통합을 쉽게 만들어 준다. IGameScene 은 상황에 따라 단일 노드나 노드의 테이블에 의해서 초기화될 수 있다. 이러한 방식으로 플로그인의 메인 노드 열거가 3ds 맥스 오브젝트에 대한 접근이 가능하다는 것을 아는 채로 적절한 게임 인터페이스로 대체될 수 있다.(원문 : In this way, a plug-in's main node enumeration can be replaced with an game interface equivalent, knowing that access to the original 3ds max object is available.)

```
	for(int loop = 0; loop <pIgame->GetTopLevelNodeCount();loop++)
	{
		INode * pNode = pIgame->GetTopLevelNode(loop)->GetMaxNode();

		//.. Carry on as before

	}
```

IGameObject 를 직접적으로 사용하기 위한 또 다른 route 가 존재할 수 있다. 그것은 공용 생성자를 가지며, IGameObject 인터페이스에 대한 접근 에 모디파이어를 더하는 것이 이러한 방식으로 구현될 수 있다. 스킨 모디파이어에 대한 접근이나 단지 파라미터에 대한 접근이 요구될 때 이것은 간단한 route 이다.

더우기 IGameSkin 인터페이스는 IGameNodes 나 더 많은 통합을 위해서 열려 있는 원래의 3ds 맥스 INode 에 기반한 데이터 접근을 제공한다.

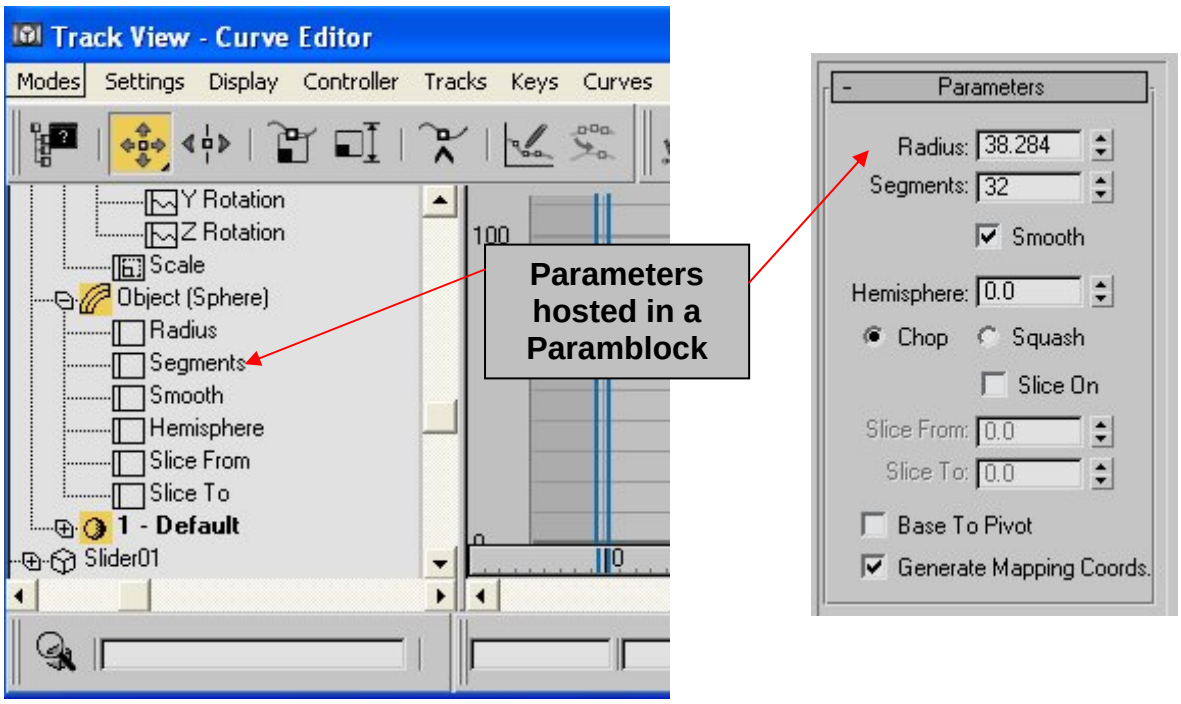
부록

Pararameter Blocks

Parameter blocks 혹은 ParamBlocks 은 3ds 맥스 프로그래밍에 있어 핵심 부분이다:

3ds 맥스에서 거의 대부분의 파라미터들은 ParamBlock 로서 등록된다. 이것은 개발자가 핵심 응용프로그램을 제어하기 위한 UI 와 데이터를 신경쓰지 않도록 해 준다. 예를 들어 TrackView 에는 많은 편집 가능한 데이터가 ParamBlock 시스템에 의해 구체화(driven)되었다.

아래 예제는 다른 위치에서 같은 paramblock 에 접근하는 것의 강력함에 대해 설명하고 있다. 이 경우 데이터가 모디파이어 다이얼로그에서 디스플레이되고 있음에도 불구하고, TrackView 는 그것의 ParamBlock 에 접근함으로써 모디파이어의 파라미터에 직접적으로 접근한다



이 파라미터 저장 메커니즘의 문제는 개발자가 ParamBlock 의 개별 아이템을 어디서부터 검색해야 할지 알지 못한다는 것이다. 이것은 SDK 가 플러인을 제대로 사용하는 데 대한 소스 코드를 제공하고 있기 때문에 내장 플러그인을 사용하면 큰 문제가 되지 않는다. 그러나 이것은 문서화되지 않은 서드 파티 플러그인에 있어서는 문제가 될 수 있다. 이 문제는 개발자들에게 혼란을 일으키고 3ds 맥스 SDK 웹보드에 수많은 질문이 올라 오는 것에 대한 책임이 있다 – "어떠헥 하면 YYY 오브젝트의 XXX 파라미터를 얻을 수 있을까요". 게임 익스포트 인터페이스는 서드 파티 플러그인 제어에 대한 단순한 프로퍼티 기반 접근을 사용해 이 모든 데이터에 대한 접근을 제공한다.

ParamBlock 의 hold 를 획득하는 방식이 항상 명확한 것은 아니다. 비록 paramblock 획득을 위한 맥스 sdk 메서드들이 존재하기는 하지만, 모든 플로그인이 이러한 방식으로 자신의 플로그인을 외포하기로 한 것은 아니다. 그러나 대부분의 paramblock 은 reference 시스템에 저장된다. 이것은 다시 개발자를 또 다른 복잡함으로 유도하고 종종 맥스 sdk 영역에 대해 잘 못 이해하도록 만든다. 그러나 IGame 은 외부에서 접근할 수 있는 pramblock 을 저장하는 모든 장소를 검사함으로써 개발자를 위해서 이를 다루어 준다. 이를 위해 IGame 은 플러그인에 의해 유지되는 모든 reference 를 돌아 보고, IParamBlock 과 IParamBlock2 를 위한 다양한 SuperClassID 들을 검색한다. 그것은 내포된 Reference Target 을 검사해서 그것들이 새로운 paramblock 을 포함하는지 여부를 확인하기도 할 것이다.

IGameProperty XML format.

IGame 이 가능한 한 확장 가능하게 만들기 위해서, 새로운 프로퍼티가 지원될 필요가 있다. 이것은 사용자 정의 재질이나 오브젝트가 표준 인터페이스를 통해서 지원될 수 있도록 허용할 것이다. IGame 에서 모든 프로퍼티 접근은 IGameProp.xml 파일을 통해서 정의된다. IGameProperty 를 반환하는 메서드는 이 파일에 엔트리를 가지게 도리 것이다. 차이라면 IGame 이 이들 중 일부를 "알고 있다"는 것이다. 그리고 프로퍼티의 획득을 위한 단순한 wrapper 를 제공한다는 것이다. 다른 프로퍼티들에 대해 개발자는 유일 ID 를 사용해서 Export Entity 에 의해서 유지되는 프로퍼티에 대한 검색을 수행할 수 있다.

파일 구문

```
<?xml version="1.0" encoding="utf-8" ?>
<!--
IGame 익스포터와 함께 사용되는 프로퍼티 파
  -->
<!--
IGame exposed APIs 를 사용하기 위해서는 아래에 정의된 프로퍼티의 ID 를 변경하지 마십시오.
그것들은 Property Container 를 요청하는 데 사용되며,
일부 메서드들은 이것을 사용해서 이것의 wrapper 를 제공합니다.

각 Proeprty 의 ID 는 paramBlock 에서 인덱싱되지 않지만,
단일 식별자는 다양한 검색에 사용됩니다.
  -->
<IGameProperties>
<ExportEntity>
<!--
 StdMat2 Properties
  -->
<simplename>StdMat2</simplename>
  <classid>0x0000002,0</classid>
  <shared>false</shared>
<Property>
<id>1</id>
<simplename>Ambient</simplename>
<name>ambient</name>
  <type>point3</type>
  <paramblock2>true</paramblock2>
  </Property>
<Property>
<id>2</id>
<simplename>Diffuse</simplename>
  <name>diffuse</name>
  <type>point3</type>
  <paramblock2>true</paramblock2>
  </Property>
</ExportEntity>
<ExportUserData>
<!--
 Node User 데이터에 접근
  <!--
id 는 반드시 파일에 대해서 단일해야 함
```

```
  <id>100</id>
  <keyName>IGameTest</keyName>
  <type>float</type>
  </UserProperty>
<UserProperty>
<id>101</id>
<simplename>IGameTestString</simplename>
  <keyName>IGameTestString</keyName>
  <type>string</type>
  </UserProperty>
  </ExportUserData>
  </IGameProperties>
```

Export Entity 구문

IGame 에서 지원되기 위한 각 Export Entity 는 반드시 <ExportEntity> 부분을 포함해야만 한다. 이것은 부모 및 프로퍼티 공유 여부를 정의한다.

<simplename> ExportEntity 의 세부 명세를 위해서 사용될 수 있는 필드. 이 필드는 IGame 과 관련이 없으며, 단지 IGameProp.xml 파일을 편집하고 시험하는 동안 ExportEntity 를 쉽게 식별하기 위해서만 사용된다.

<classid> 이것은 ParamBlock 소유자의 ClassID 이다.

<shared> 이것은 IGame 에 Property 가 공유되는지 여부를 알려 준다. Shared 프로퍼티의 예는 Light 이다. 여러 개의 서로 다른 조명이 존재하지만 그것들 모두는 공통 프로퍼티를 공유한다. 이 경우 classid 는 그룹의 "SuperClassID" 이다.

각 Property 는 <Property> 블록을 포함한다. 이것은 Paramblock 을 검색하는데 사용되는 데이터이다.

<id> Parameter 를 위한 단일 ID. 이것은 <ExportEntity> 블록 내의 프로퍼티들에 대해서만 유일하면 된다.

<simplename> Property 의 세부 명세를 위해서 사용될 수 있는 필드. 이 필드는 IGame 과 관련이 없으며, 단지 IGameProp.xml 파일을 편집하고 시험할 때 Property 에 대한 쉬운 식별을 위해서만 사용된다.

<name> 파라미터의 이름. IParamBlock 에 대해 이것은 SubAnimName 이며, IParamBlcok2 에 대해 이것은 내부 이름이다.

<type> ParamBlock 유형 – 현재 flaot, point3, int, string 이 지원된다.

<paramblock2> 이 프로퍼티가 ParamBlock2 인지 여부.

Export User Data 구문

맥스의 모든 노드는 object properties 다이얼로그 박스를 통해서 그것에 추가되는 사용자 데이터를 소유할 수 있다. IGame 은 이 데이터를 IGameProperty 로서 추출한다. 그것은 <ExportUserData> 필드에서 찾을 수 있는 데이터를 사용하여 발견된 데이터를 올바르게 추출한다. 사용자 데이터의 각 조각은 <UserProperty> 에 포함된다.

<id> Parameter 에 대한 단일 ID. 이것은 파일에 대해 유일해야만 한다.

<simplename> UserProperty 의 세부 명세를 위해서 사용될 수 있는 필드. 이 필드는 IGame 과 관련이 없으며, IGameProp.xml 파일을 편집하고 시험하는 동안 UserProperty 를 쉽게 식별하기 위해서만 사용된다..

<keyName> 데이터 청크의 이름. 이것은 공백을 포함할 수 없다.

<type> 데이터형. 사용자 데이터는 int, float, bool, string 을 지원한다.

IGameObject Interfaces

IGameObject Types 및 관련 인터페이스.

IGame Object Type	IGame Interface
IGAME_UNKNOWN	IGameGenObject
IGAME_LIGHT	IGameLight
IGAME_MESH	IGameMesh
IGAME_SPLINE	IGameSpline
IGAME_CAMERA	IGameCamera
IGAME_HELPER	IGameHelperObject
IGAME_BONE	IGameHelperObject
IGAME_IKCHAIN	IGameIKChain

3ds max Mesh Representation

다음 다이어그램 집합은 매우 상위 레벨에서 기본 3ds 맥스 메시를 보여 주기 위한 목적으로 제공된다. 이 정보는 IGame 으로부터 데이터를 추출할 때 도움을 줄 것이다.

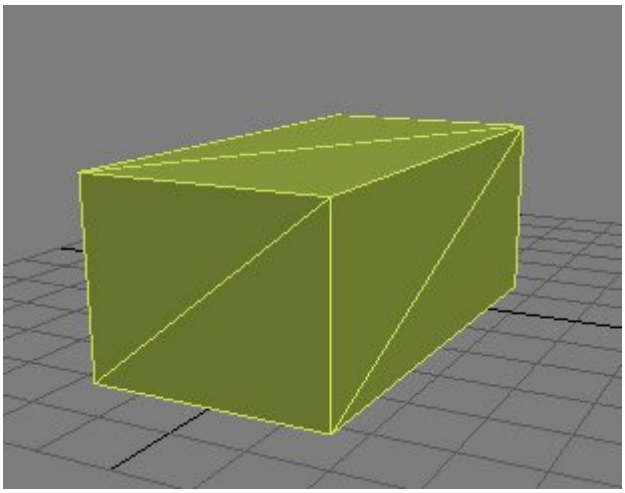


Figure 1 보이는 면과 정점을 가진 박스

Figure 1 은 매우 간단한 박스이지만, 이것은 실제로는 복잡한 오브젝트이다. 당신은 면과 정점을 명확하게 보고 있지만, 법선은 보고 있지 않다. 박스는 각진 모서리를 가지고 있기 때문에, 각 모서리의 측면 상에 있는 면들이 따로 떨어져서 90 도의 법선을 가지고 있다는 사실이 중요하다. 즉 박스는 실제적으로 3 개의 법선을 가지고 있으며, 각 면의 법선은 그것과 연결되어 있다(역주 : 각 정점에 대해서 3 개의 법선이 유지된다는 의미이다. 아래 그림에 보면 정점 당 세 개의 법선이 존재한다). 이 데이터와 법선 색인은 3ds 맥스 내부의 smoothing 그룹을 사용해 제어된다. 이것이 올바르게 제어되고 smoothing 그룹에 기반한 면에 대해 올바른 법선이 계산된다는 것이 중요하다. Figure2 와 3 은 뷰포트에서 이것이 작동하는 것을 보여 준다.

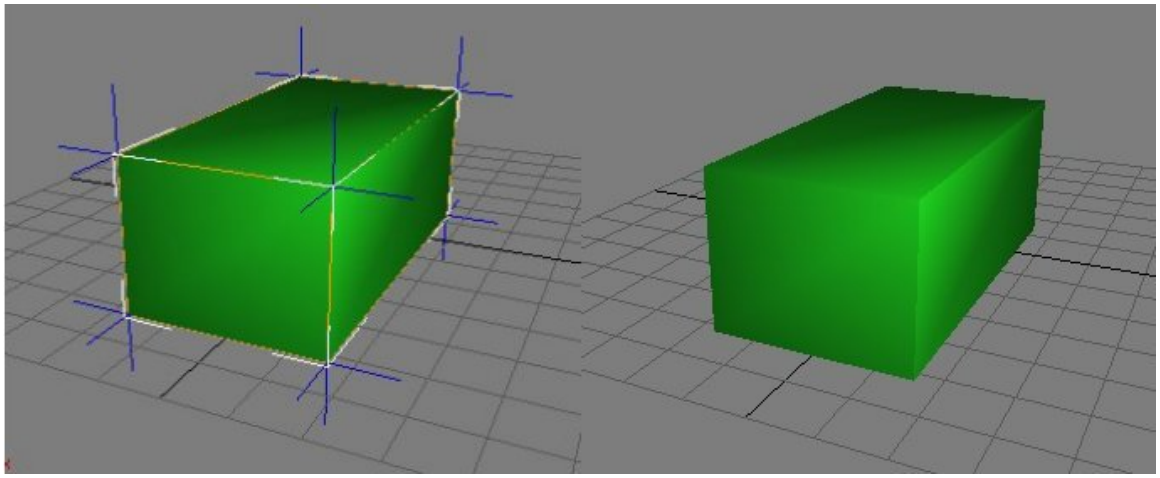


Figure 2 면에 대해서 단일한 법선

파란색으로 칠해 진 세개의 법선들은이 각 정점 위치에 명확하게 보인다. 이는 오른쪽에 있는 박스가 각진 모서리로 보이도록 해 준다. 만약 하나의 정점을 둘러싼 면이 법선을 공유하는 것이 허용된다면, 이것은 같은 smoothing 그룹에 존재하며, 매우 다른 모양으로 렌더링 될 것이다.

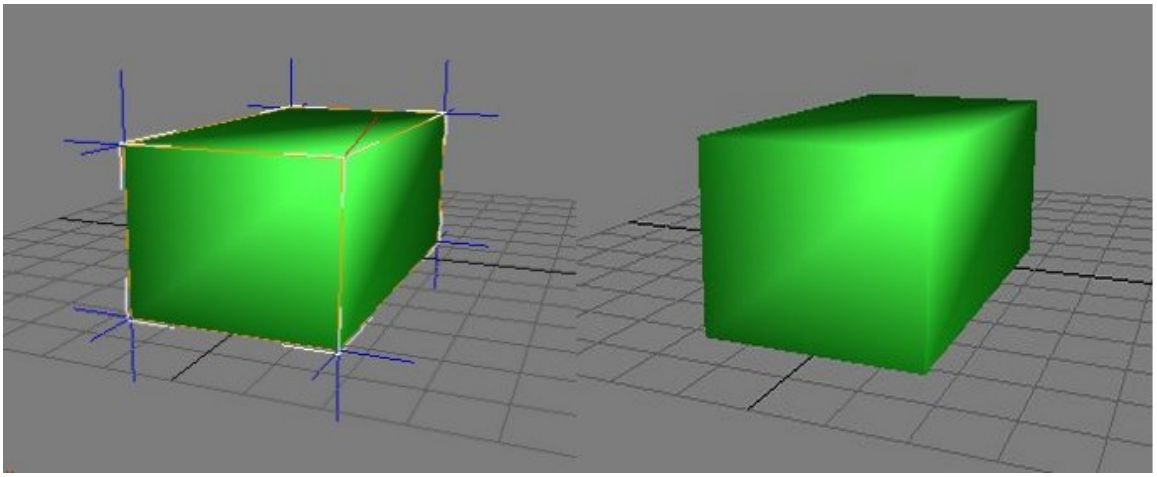


Figure 3 Shared Vertex Normal

(전면 우 상단에) 빨간색의 공유 법선이 보일 것이다. 그리고 명백히 오른족의 이미지에서 보이듯이 이 정점 주변의 각진 모서리들이 없어졌다.

각 면은 다중 Material ID 를 소유할 수도 있다. 이것은 면이 연결된 다른 면과는 다른 텍스처를 소유할 수 있도록 해 준다. 3ds 맥스에서 이것은 Multi₩SubObject 재질을 적용함으로써 이루어지며, 여기에서 재질은 면의 Material ID 에 맞게 적용된다. multi₩SubObject 재질은 사용자가 선택된 면 상의 재질을 "drag and drop" 하면 자동적으로 생성된다. 재질을 획득한 각 면은 자신의 Material ID 를 갱신하고 새로운 multi 재질이 그것을 반영하기 위해서 갱신될 것이다. 다시 말하지만 이것을 이해하는 것은 매우 중요하다. 왜냐하면 이러한 종류의 설정은 단일 Material ID 를 사용하는 각 면이 다른 것과 독립적으로 렌더링될 것을 요구하기 때문이다. 메시는 매우 단순하게 쪼개져서 자신만의 재질을 가지고 독립적으로 그려진다. IGame 은 smoothing 그룹이나 Material ID 에 기반해서 면을 추출하는 기능을 제공한다.

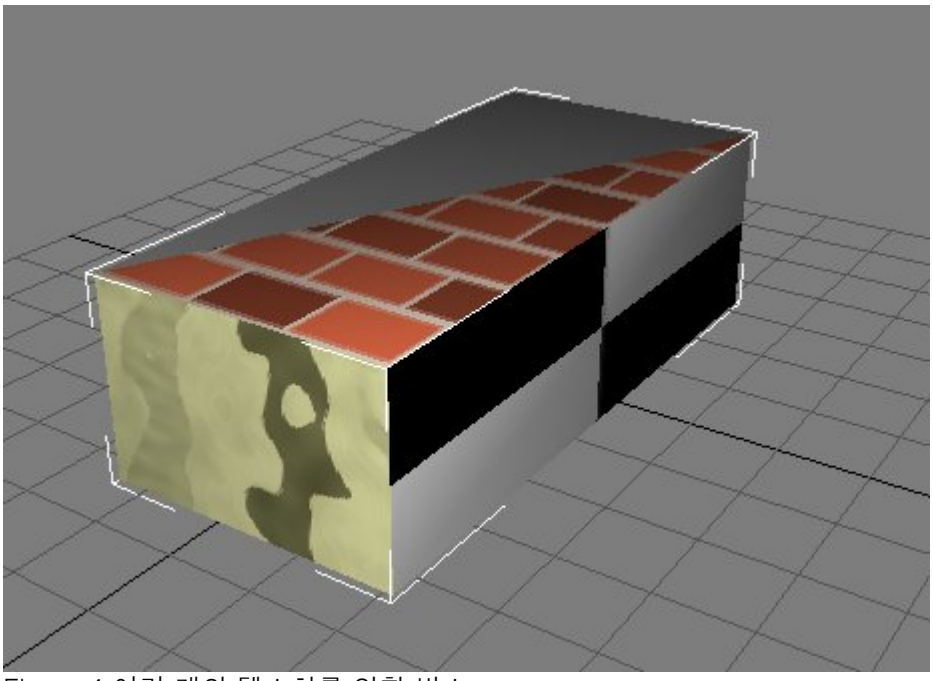


Figure 4 여러 개의 텍스처를 입힌 박스

IGameKey Definition

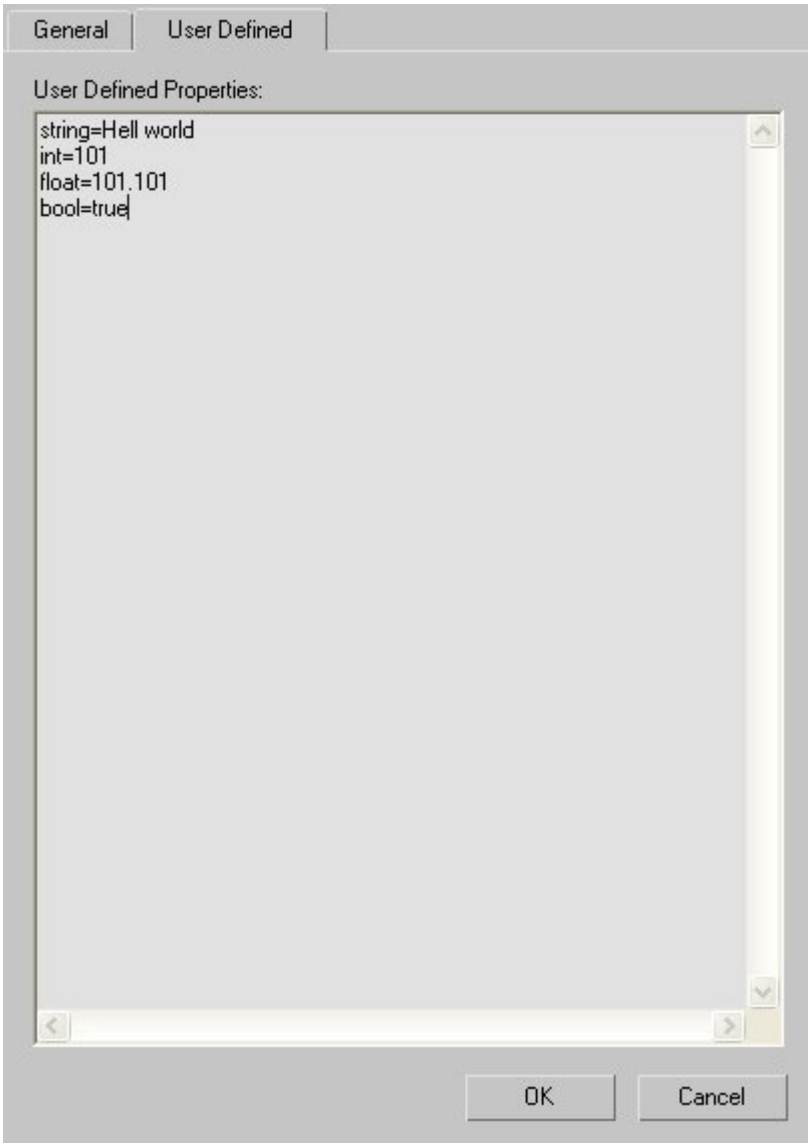
```
//! class IGameTCBKey
/*! A generic TCB key class for IGame
*/
class IGameTCBKey {
	public:
		//! access to basic TCB data
		float tens, cont, bias, easeIn, easeOut;
		//! float based value, used with IGAME_FLOAT
		float fval;
		//! Point3 based value, used iwth IGAME_POINT3 | IGAME_POS
		Point3 pval;
		//! Angle Axis based value where quarternions can be extracted, used with
IGAME_ROT
		AngAxis aval;
		//! ScaleValue based value, used with IGAME_SCALE
		ScaleValue sval;
	};

//! class IGameBezierKey
/*! A generic Bezier Key class for IGame
*/
class IGameBezierKey {
	public:
		//! float based In and out tangents, used with IGAME_FLOAT
		float fintan, fouttan;

		//! float value, used with IGAME_FLOAT
		float fval;
		//! float based tangent lengths used with IGAME_FLOAT
		float finLength, foutLength;

		//! Point3 based In and out tangents, used with IGAME_POINT3 | IGAME_POS
		Point3 pintan, pouttan;

		//! Point3 value, used with IGAME_POINT3 | IGAME_POS
		Point3 pval;
		//! Point3 based tangent lengths used with IGAME_POINT3 | IGAME_POS
		Point3 pinLength, poutLength;
		//! Quaternion based value, used with IGAME_ROT
		Quat qval;
		//! ScaleValue , used with IGAME_SCALE
		ScaleValue sval;

	};

//! class IGameLinearKey
/*! A generic Linear Key class for IGame
*/
class IGameLinearKey  {
	public:
		//! float based value, using IGAME_FLOAT
```

```
		float fval;
		//! Point3 based value, using IGAME_POS |IGAME_POINT3
		Point3 pval;
		//! Quaternion based value, using IGAME_ROT
		Quat qval;
		//!Scale value, using IGAME_SCALE
		ScaleValue sval;
	};


//! class IGameSampleKey
/*! A generic Sample Key class for IGame.  This is used for unkown controllers or controllers
that
simply need to be sampled, this can includes Biped
*/
class IGameSampleKey{
	public:
		//! Point3 value, used with IGAME_POINT3 and IGAME_POS
		Point3 pval;
		//! float value, used with IGAME_FLOAR
		float fval;
		//! Quaternion value, used with IGAME_ROT
		Quat qval;
		//! Scale value, used with IGAME_SCALE
		ScaleValue sval;
		//! GMatrix, used with IGAME_TM
		GMatrix gval;

};


//!class IGameKey
/*! A simple container class for direct Key access of all the available Key types

*/
class IGameKey
{
	public:
		//! The time the key was set
		TimeValue t;
		//! Flags various selection states for the key.
		DWORD flags;
		//!The TCB Keys
		IGameTCBKey tcbKey;
		//!The Bezier keys
		IGameBezierKey bezierKey;
		//!The linear keys
		IGameLinearKey linearKey;
		//!The sampled keys
		IGameSampleKey sampleKey;
};
```

Object Properties

프로퍼티 관리자는 (맥스 sdk 에서 node 프로퍼티로 참조되는) Object Properties 에 대한 확장을 허용한다. 그것은 개발자가 오브젝트 상에 저장되는 데이터의 정의를 제공하는 것을 허용하는 XML 프로퍼티 파일을 통해서 이를 지원한다. 제공되는 데이터형은 INT, BOOL, FLOAT, STRING 이다.

사용자는 이 데이터를 맥스 내부에서 추가할 수 있는데, 오브젝트에서 오른쪽을 클릭하고 Quad 메뉴에서 Properties 를 선택한다. 다음 윈도우즈 다이얼로그 윈도우가 디스플레이 된다.



그리고 나서 사용자는 key 에 type 을 입력하고 "=" 다음에 데이터를 입력하면 된다. key 는 공백문자를 포함해서는 안 된다.

xml 파일에서 개발자는 다음 엔트리를 추가해서, Property Manager 가 데이터를 파싱하고 IGameProperty 인터페이스를 통해 그것에 대한 접근을 제공하게 할 수 있다.

```
<ExportUserData>
<!--
 Access to the Node User Data
  <!--
```

The id must be unique to the file

```
  <id>100</id>
  <keyName>IGameTest</keyName>
  <type>float</type>
  </UserProperty>
<UserProperty>
<id>101</id>
<simplename>IGameTestString</simplename>
  <keyName>string</keyName>
  <type>string</type>
  </UserProperty>
<UserProperty>
<id>102</id>
<simplename>IGameTestInt</simplename>
  <keyName>int</keyName>
  <type>int</type>
  </UserProperty>
<UserProperty>
<id>103</id>
<simplename>IGameTestFLoat</simplename>
  <keyName>float</keyName>
  <type>float</type>
  </UserProperty>
<UserProperty>
<id>104</id>
<simplename>IGameTestBool</simplename>
  <keyName>bool</keyName>
  <type>bool</type>
  </UserProperty>
  </ExportUserData>
```