

HumanIK

GAME パイプライン



オートデスク株式会社
メディア&エンターテインメント

Autodesk®

HumanIK

Motion Builder から派生した
ミドルウェアライブラリ (C++ ベース)

主要な2つの機能 (ソルバー)

フルボディク

リターゲット

Windows, XBox360, PS3, Wii, Linux64, MacOSX
軽量、コンパクト、高速化、最適化されてライブラリ

HumanIK 概要

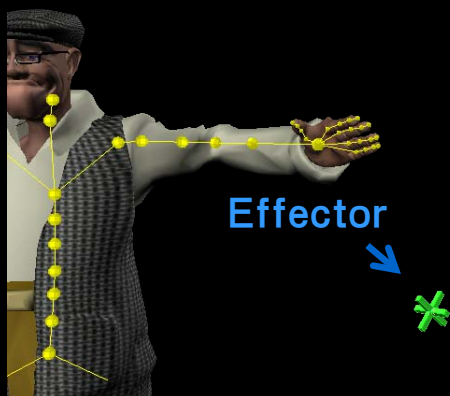
フルボディIK

- ・ IKエフェクターのリーチで全身がコントロール可能
- ・ 各エフェクター:
 - 作用のOn/Off
 - 移動Reachのウエイト
 - 回転Reachのウエイト
 - Pullの値
 - 関節の固さの値

Reach と Pull

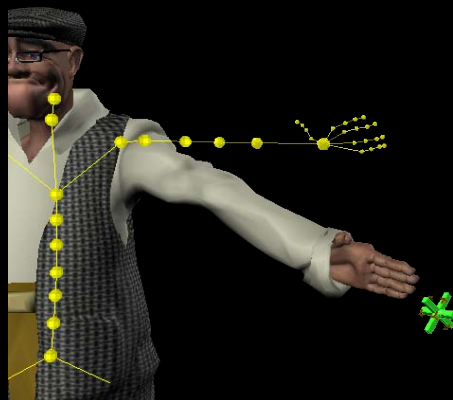
0 % Reach T

0 % Pull



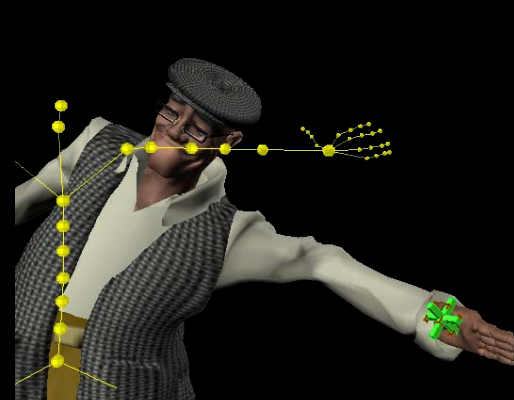
100 % Reach T

0 % Pull



100 % Reach T

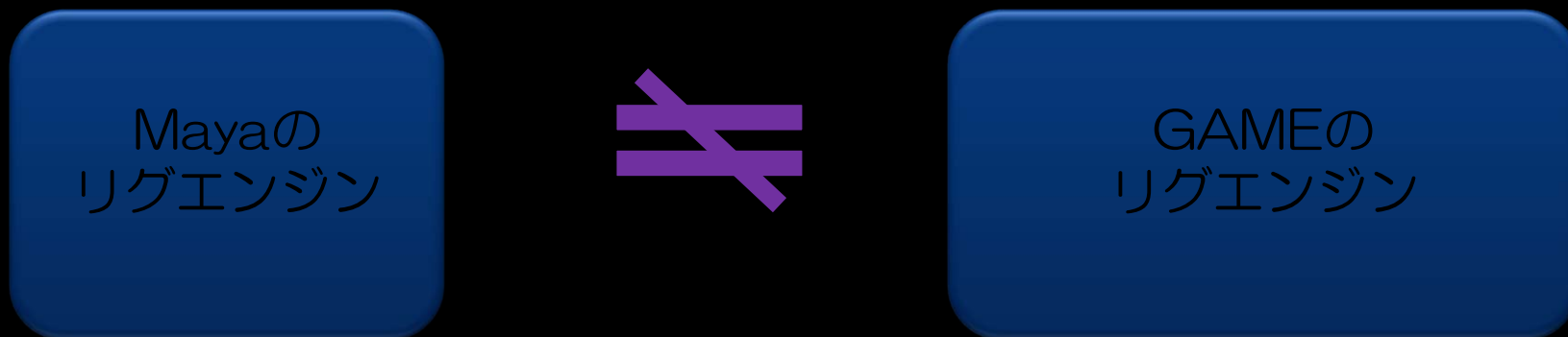
100 % Pull



現在のキャラクターパイプライン

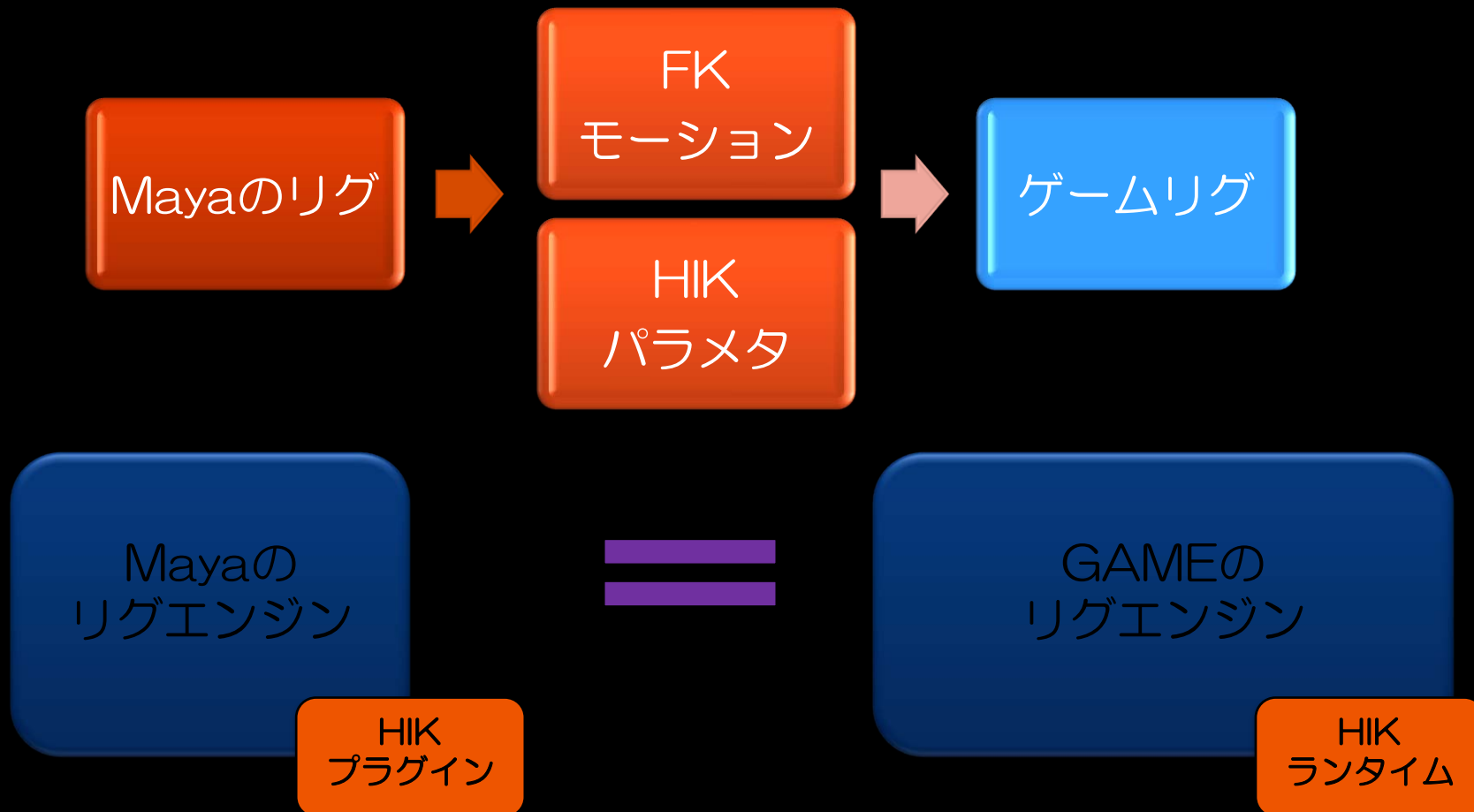


CGソフトのrig と ゲームrig に関連性がない



CGソフト側ではキャラクタの細かな挙動を事前に確認するのが難しい

Maya+HIKベースのパイプライン

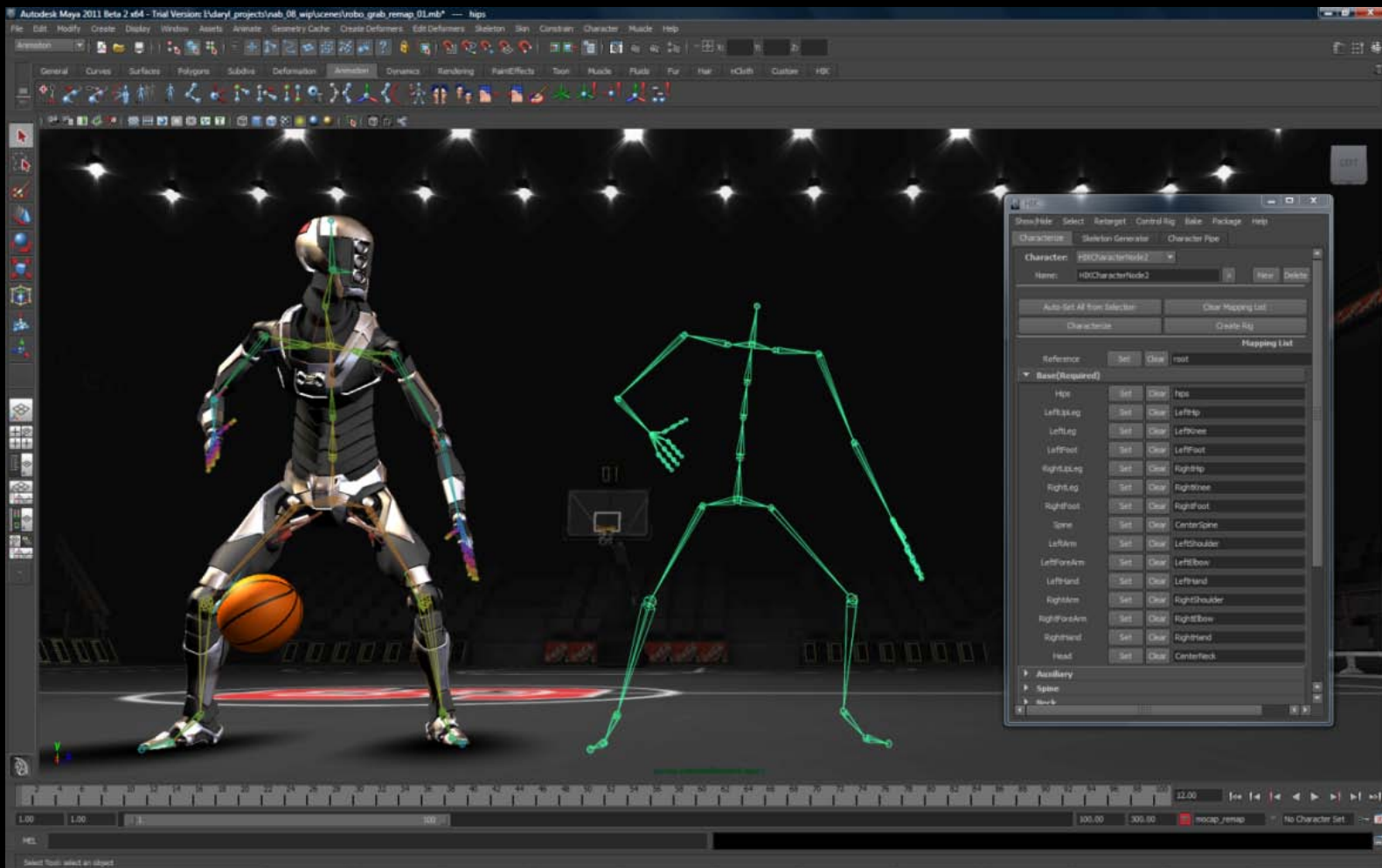


CGソフトで事前にゲームリグの挙動を完全に把握できる

HumanIK

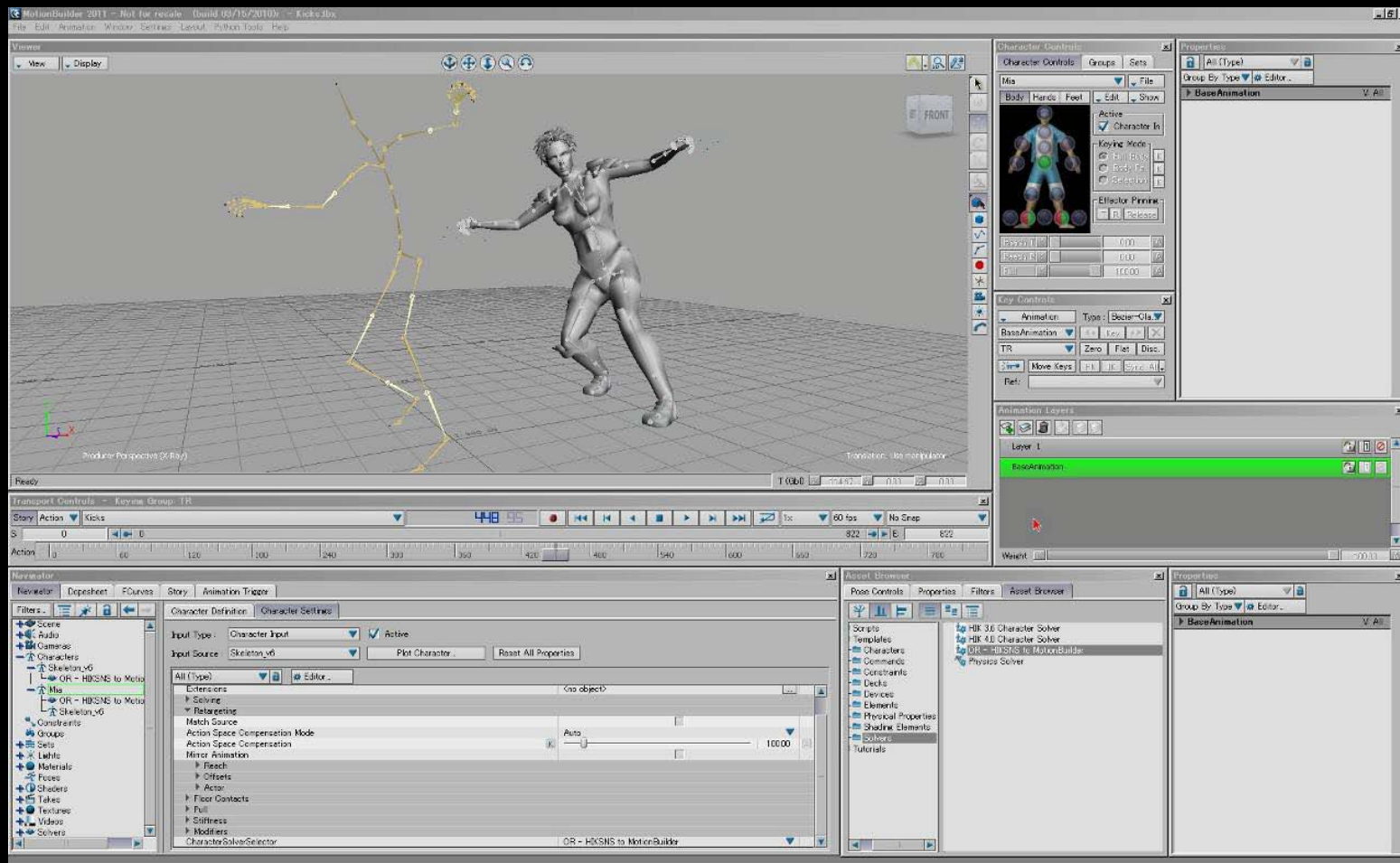
Maya 用 HIK プラグイン

フルボディIK リターゲッティング

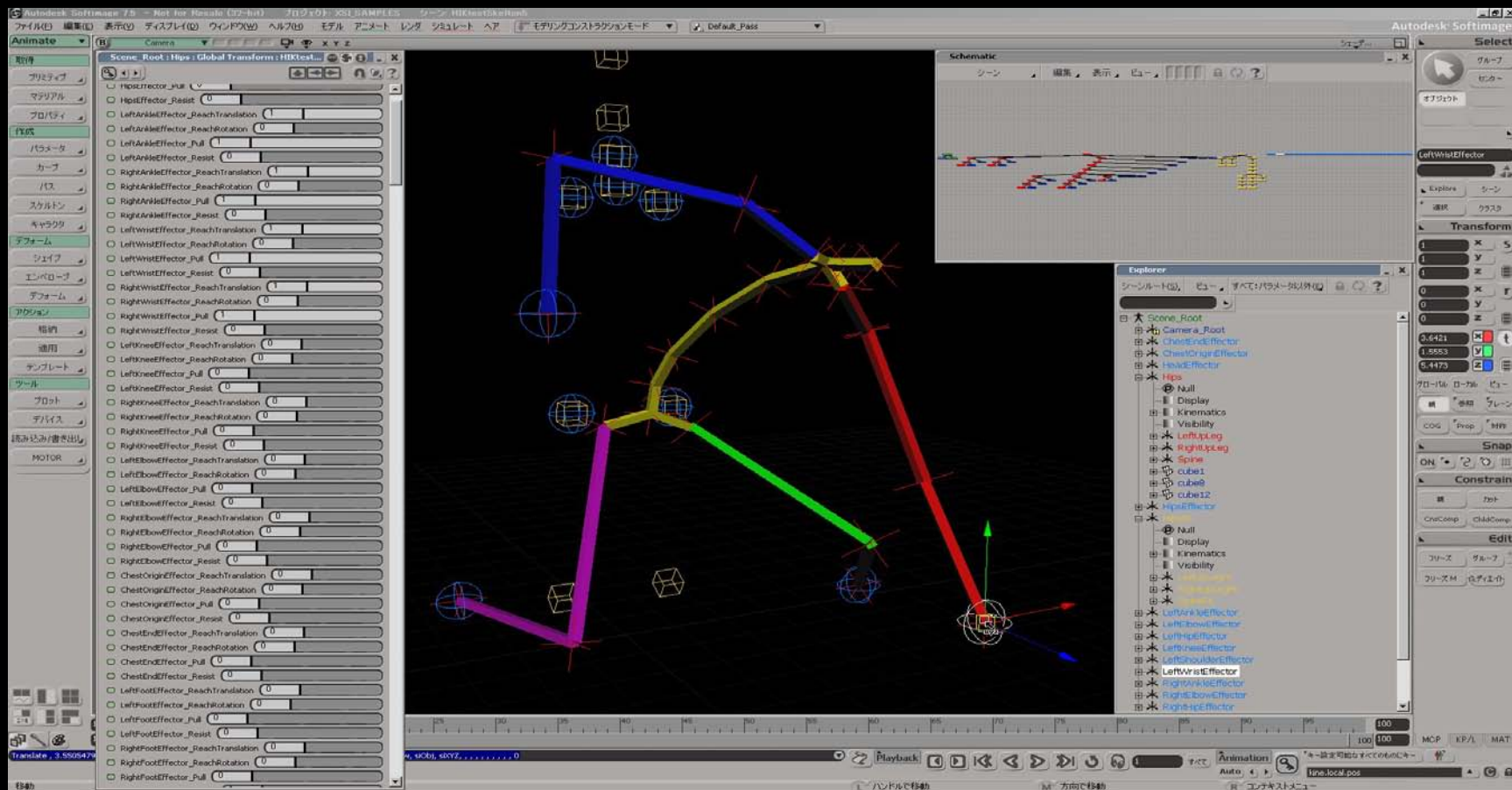


HumanIK Motion Builder

フルボディIK リターゲッティング



HumanIK Softimage 用 HIK プラグイン

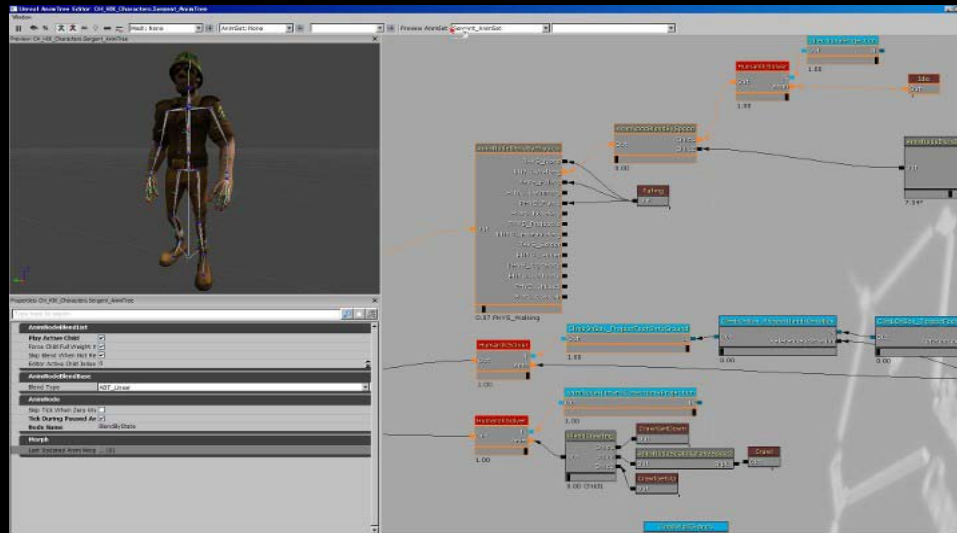


HumanIK

Unreal Engine3 インテグレーション

ノードベースのGUIプログラミングが可能
HumanIKソルバーをUnreal Engine3のノードとして提供
(ソースコードを公開)

Unreal Editor用 キャラクターライゼーションツールを提供
プログラムレスでHIKキャラクターの設定が可能



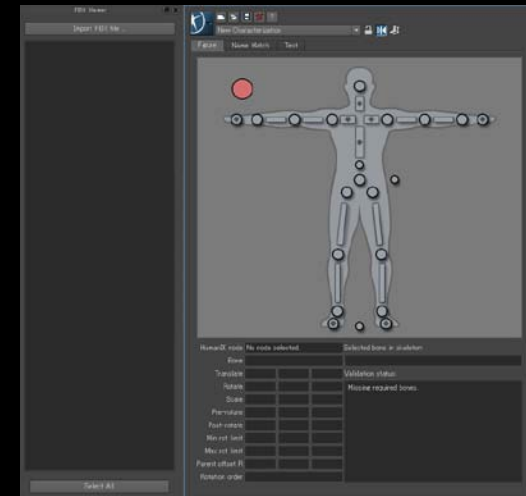
HumanIK

キャラクターライゼーション ツール

HumanIK キャラクタオブジェクトを
プログラミングをせずに作成可能（キャラクターライズ）
GUIで HumanIK スケルトンマッピングが可能

Maya プラグイン
Softimage プラグイン

出力ファイル
.hikc (binary)



HumanIK

.hikcファイルを読み込み HIKランタイムでキャラクターを実態化するには

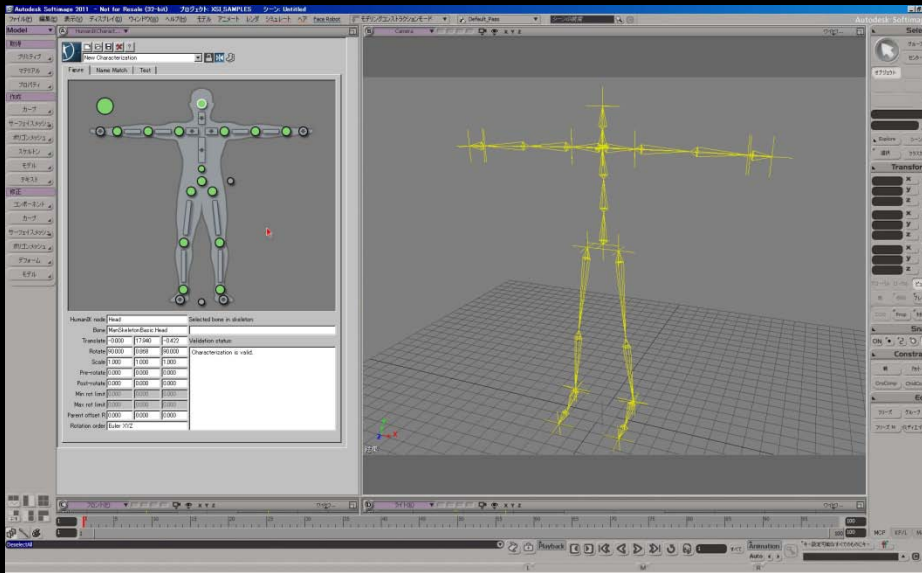
HIKReadFromStream() で .hikcバイナリを読み込む

```
HIKCharacter* HIKReadFromStream(  
    HIKCharacterDefinition* pCharacterDefinition,  
    const char* pStream,  
    size_t pStreamSize,  
    HIKMalloc pMalloc,  
    HIKFree pFree,  
    const char* pValidationString,  
    unsigned int pValidationKey )
```

.hikcファイルをプログラマがGAMEエンジンで読み込むだけ

HumanIK

キャラクタライゼーションツール



```
////////////////////////////////////  
// Initialization: Create all the necessary character, character state and effector state for a retargeting.  
MyCharacter IDstCharacter(gHikDef);  
MyCharacter ISrcCharacter(gHikDef);  
  
// The reason why these states wasn't embedded into MyCharacter class is simple. You may instance it as much state you  
// need for parallel processing. So it's a good idea to store these states in way it will be easy to get them depending  
// on which process need it:  
HIKCharacterState IDstState = HIKCharacterStateCreate(IDstCharacter-mCharacter, &malloc);  
HIKCharacterState ISrcState = HIKCharacterStateCreate(ISrcCharacter-mCharacter, &malloc);  
HIKEffectorSetState IEffState = HIKEffectorSetStateCreate(&malloc);  
HIKPropertySetState IDstPropertyState = HIKPropertySetStateCreate(&malloc);  
HIKPropertySetState ISrcPropertyState = HIKPropertySetStateCreate(&malloc);  
  
////////////////////////////////////  
// Setup the character's geometry (default stance) for the source and destination character.  
int i;  
for(i=0;i<IDstCharacter-mJointsCount;i++)  
{  
    HIKSetCharacterizeNodeStateV(IDstCharacter-mCharacter, IDstCharacter-mJoints[i]-mHikNodeId, &IDstCharacterization[i][0]);  
}  
  
for(i=0;i<ISrcCharacter-mJointsCount;i++)  
{  
    HIKSetChar  
  
////////////////////////////////////  
class MyCharacter  
// Period this class hold all the model driven by your character and the data necessary to drive  
// a HIKCharacter.  
public:  
    MyCharacter* mCharacter;  
    MyHikLinkToModel* mJoints; // This member will be allocated in the constructor depending on the character definition.  
    int mJointsCount; // How much joint this character will drive.  
  
public:  
    MyCharacter(const HIKCharacterDefinition& ICharacterDefinition);  
    MyCharacter(NULL, mJointsCount(0));  
  
    MyCharacter = HIKCharacterCreate(&HikDef, &malloc, AutodeskCustomerString, AutodeskCustomerKey);  
  
    // First step count how much joint we need  
    int i;  
    for(i=0;i<LastNodeID;i++)  
    {  
        if(ICharacterDefinition.mUsedNodes[i] != HIKNodeNotUsed)  
            mJointsCount++;  
    }  
  
    // Allocate the necessary link  
    mJoints = new MyHikLinkToModel[mJointsCount];  
  
    // Define all the joint link.  
    int iJointToller = 0;  
    for(i=0;i<LastNodeID;i++)  
    {  
        if(ICharacterDefinition.mUsedNodes[i] != HIKNodeNotUsed)  
        {  
            mJoints[i].mJointToller = i;  
            // In this case we don't set anything because this is just an example and we don't really  
            // drive any Model  
            mJoints[i].mModel = NULL;  
            iJointToller++;  
        }  
    }  
  
    virtual MyCharacter()  
    {  
        mCharacter = NULL;  
        mJoints = NULL;  
        mJointsCount = 0;  
    }  
};
```

手間のかかるキャラクタライズのプログラムコードが不要

HumanIK

Softimage HumanIK 実装例

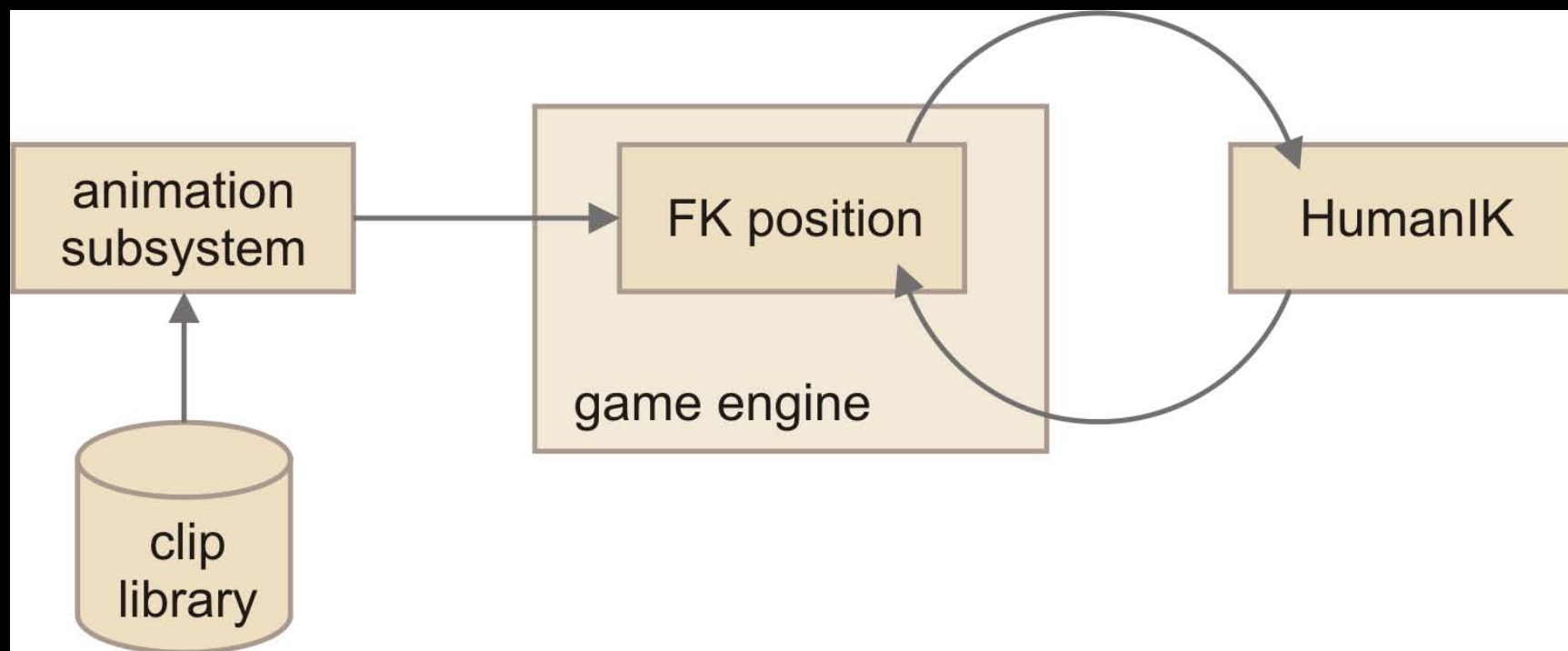
処理の流れ

1. HIKReadFromStream() .hikcファイルのキャラクタを実体化(キャラクタライズ)
2. Softimageのスケルトンから間接のマトリクスを抽出
3. HIKSetNodeState() FKトランスフォームマトリクスをセット
4. HIKSetEffectorState() IKエフェクタのトランスフォームマトリクスをセット
5. HIKSetEffectorFloorState() フロアコンタクトを設定
6. HIKSetTranslationActive() IKエフェクタの移動Reachを設定
7. HIKSetRotationActive() IKエフェクタの回転Reachを設定
8. HIKSetPull() IKエフェクタのPullを設定
9. HIKSetResist() IKエフェクタに関係する間接の硬さを設定
10. HIKSetPropertyValue() プロパティ値を設定 (細かなHIKの設定値)
11. HIKSolveForEffectorSet() フルボディIKの計算
12. HIKGetNodeStatedv() HIKステートから計算した間接のマトリクスを取得
13. Softimageのスケルトンへ流し込む

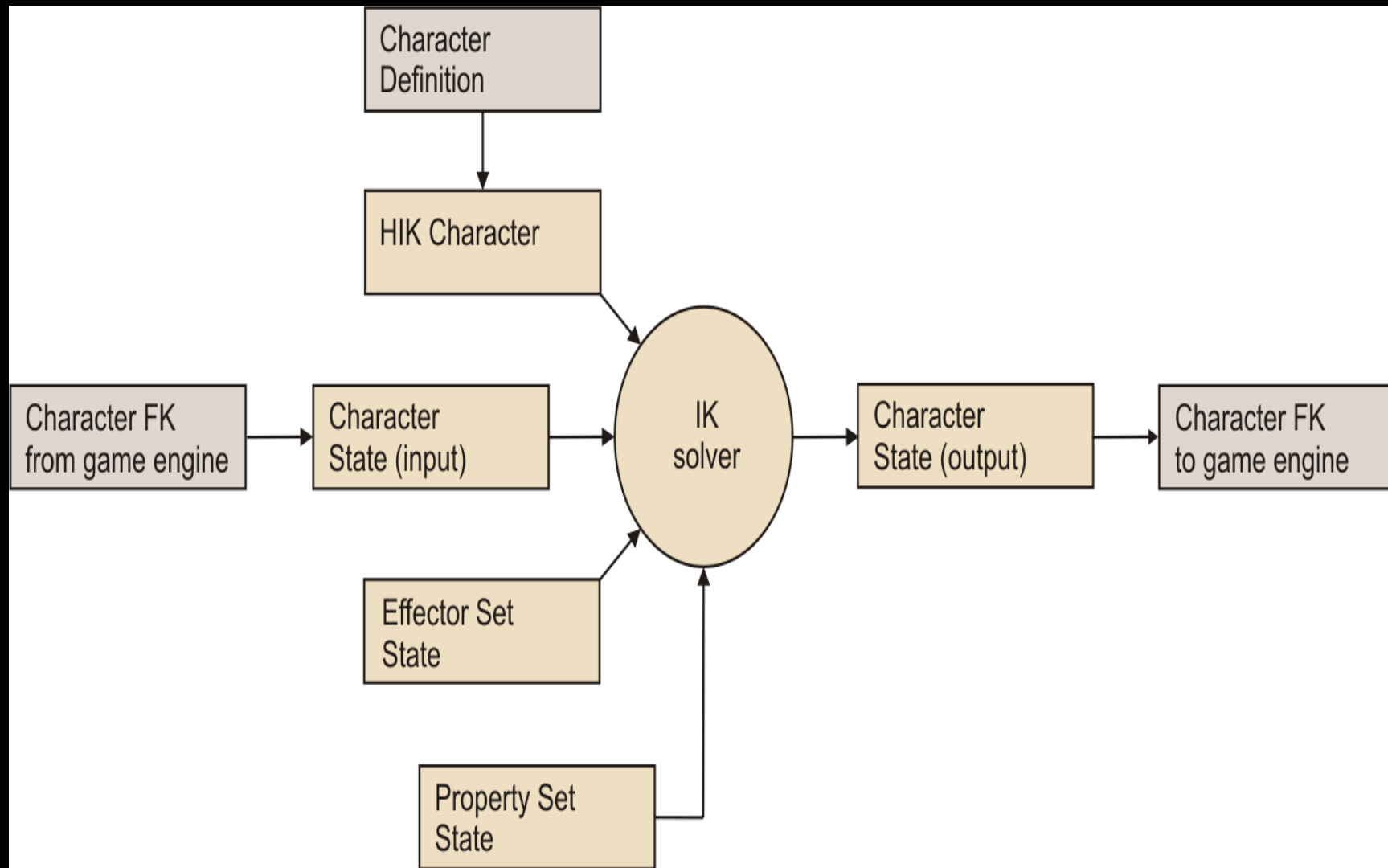


HumanIK アーキテクチャ

既存ゲームエンジンに組み込み使用する



HumanIK アーキテクチャ



Autodesk®