リアルタイムリグ

DCCツールと実機で動作する補助関節のセットアップ

SQUARE ENIX®

佐々木　隆典
リードテクニカルアーティスト
株式会社スクウェア・エニックス　技術開発部

アジェンダ

背景と動機	• なぜ補助骨システムが必要か • リグデザインを整理
開発プロセス	• 開発の流れ • 基礎知識としての数学の話 • 要求分析と仕様作成
システムの詳細	• オペレータの基本構成 • ベース関節指定機能 • コンストレイン機能
DCCツールごとの実装と問題点	• DCCツールごとの実装紹介 • 階層スケーリングの対応 • チェーンによる問題
まとめ	

背景と動機

- なぜ補助骨システムが必要か
  - リグデザインを整理

開発の歴史

2000年: スクウェア・ビジュアルワークス

- 映像制作業務の一貫で、Maya で “2AxesDriver” という回転を分解・合成する為の expression を作成したのが始まり。
- ここから伝わって、派生したツールが他社でも使われている。

2006年: 他社

- ゲームランタイムを考慮した「補助骨システム」を Softimage 上に開発。

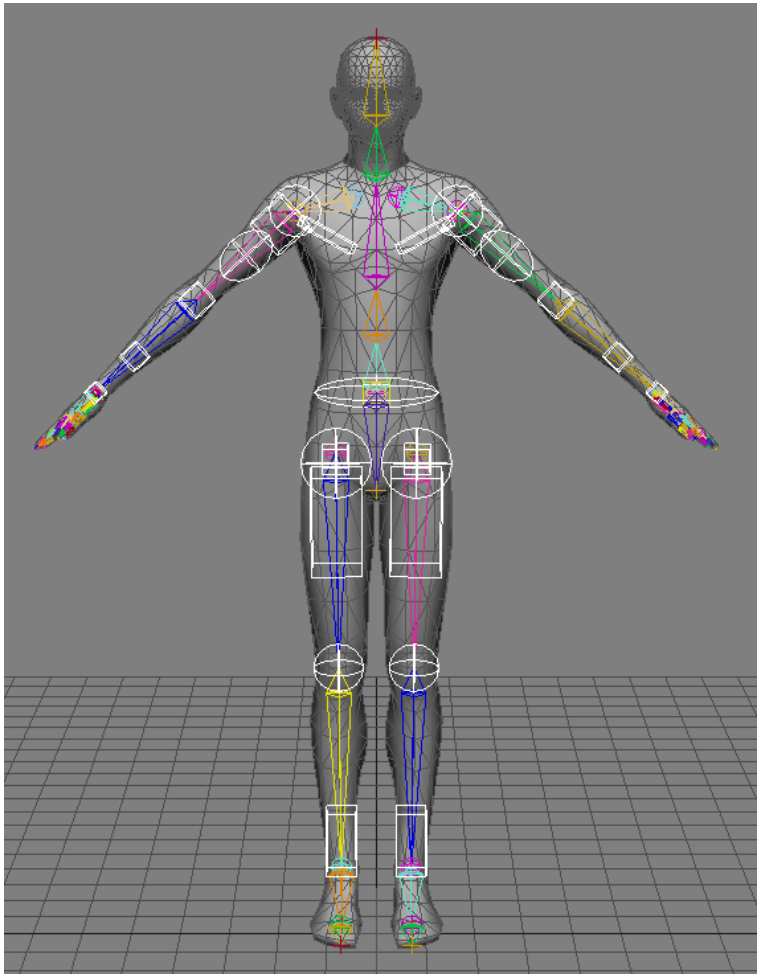
2009年: スクウェア・エニックス

- より発展させて、柔軟なセットアップを可能にした補助骨システム “KineDriver” を Softimage 上に開発。
- MotionBuilder 版も開発。

『補助骨』とは

- 実際の骨格とは何の関係もない
- 主にデフォーム品質の向上を目的に挿入される
- とはいえ、用途はいろいろ

メイン骨に連動させたい
（直接アニメートしない）



補助骨とリグ

「ある関節がこう動いたら、別の関節がこう動く」
というロジックを作りたい

そのような『リグ』をDCCツール上で組む

ゲームランタイムは
どうすべき？

ゲームランタイム上の補助骨

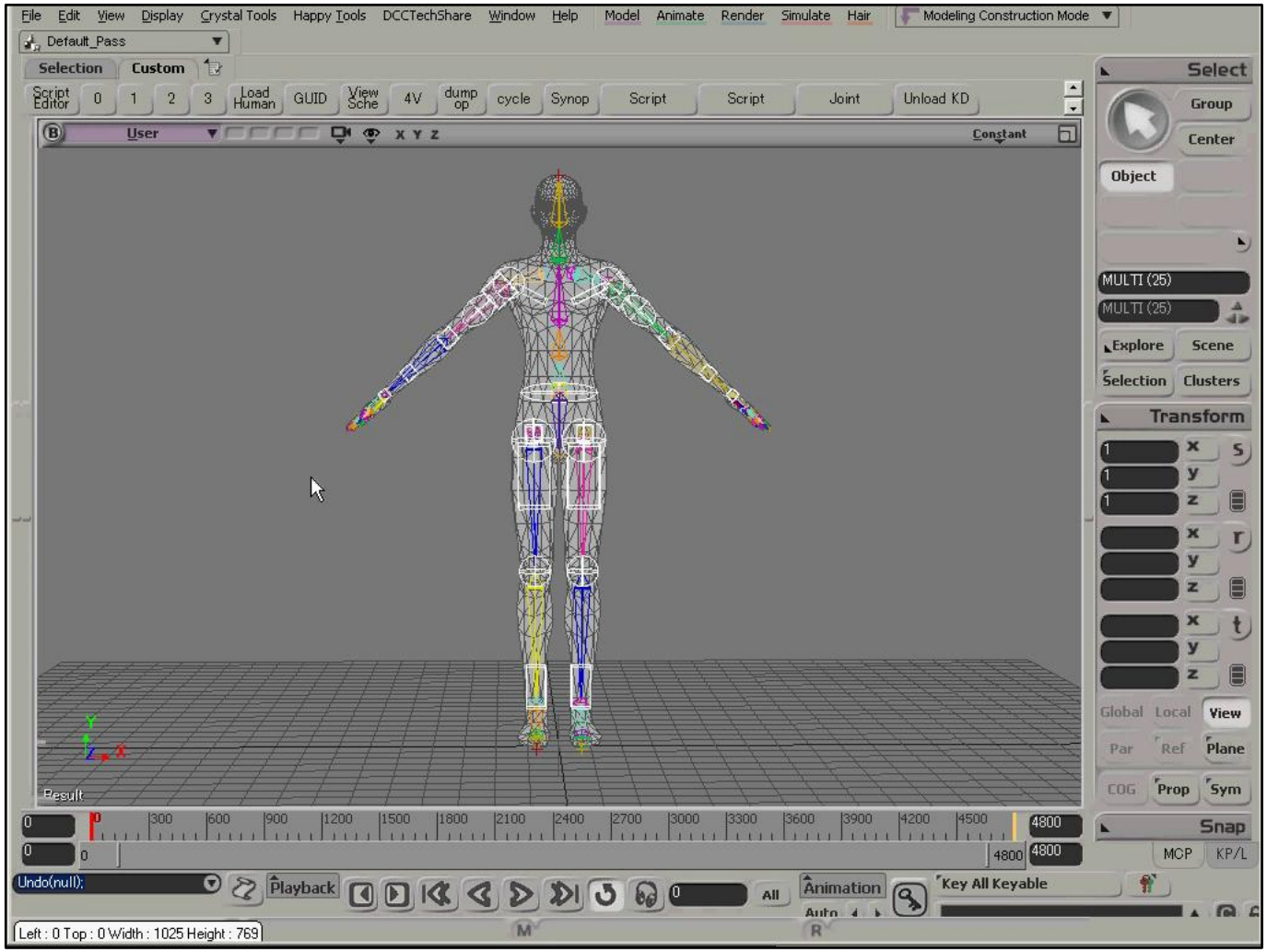
- 実機にはシンプルな骨構造のみを出力？
  - 複雑なリグはDCCツール上だけで済ませたい
  - 補助骨の動きはプロットしてモーションデータにする
- なるべくならそう考えたいが・・・、そうとも言えなくなってきている
  - 必要なモーション数の増加
  - 必要な補助骨の増加
  - 実機ランタイム処理の高度化

実機で補助骨を動かしたい！

- 期待される効果
  - モーションデータ量の大幅な削減
  - ランタイムでのモーション合成処理の高品質化
- 応用効果
  - 補助骨以外の多様なパラメータも関節に連動させる

    例) 法線マップの皺が浮き出るなど

デモ：概要



背景と動機

- なぜ補助骨システムが必要か
- リグデザインを整理

リグ構造について考える

何を実機で動作させる必要があるのか？

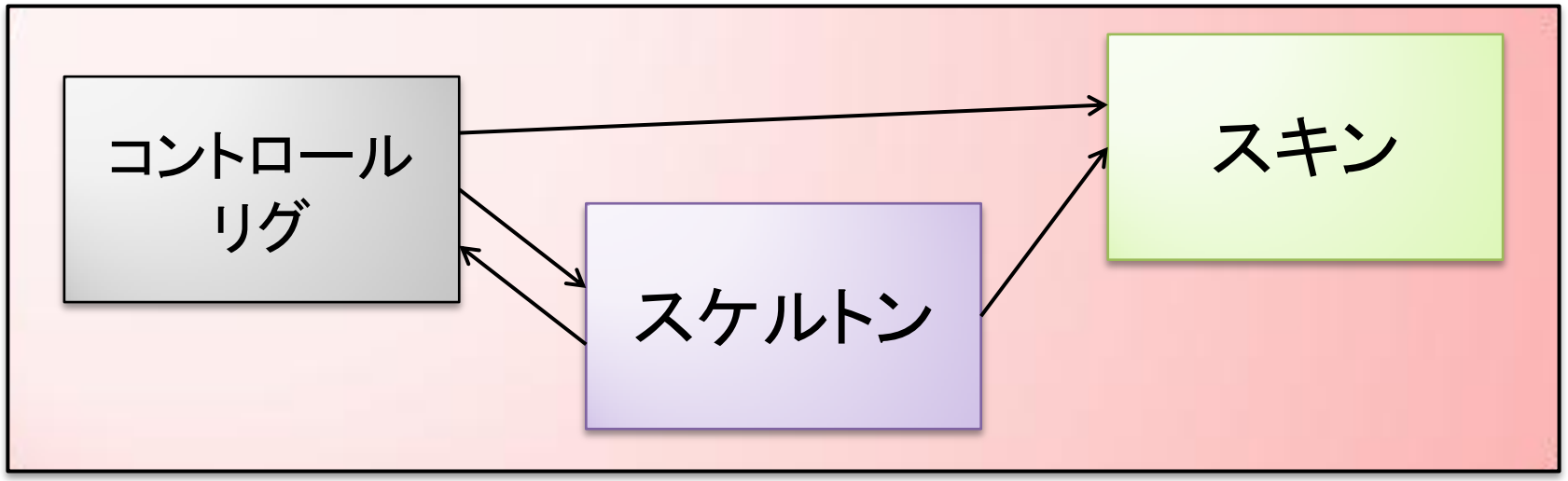
- デフォームリグ
  - 直接スキンをバインドするスケルトン
  - 補助骨や揺れ骨なども含む
- コントロールリグ
  - モーション作成の為の付加構造
  - いわばアニメータ用のツール

実機にもエキスポート

勝手な命名です ^_^;

リグの構造

整理することを考えていないリグ・・・



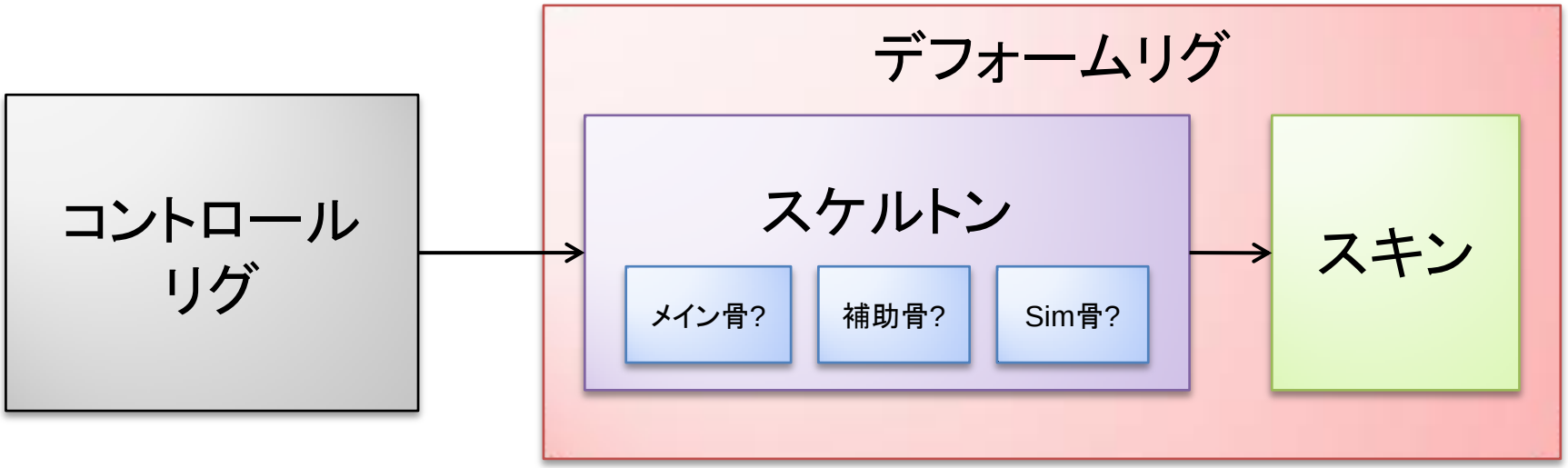
(同じ階層下に混在)

(役割・依存関係もあいまい)

切り離せない・・・

リグの構造

リグ階層と依存関係を分離しよう・・・

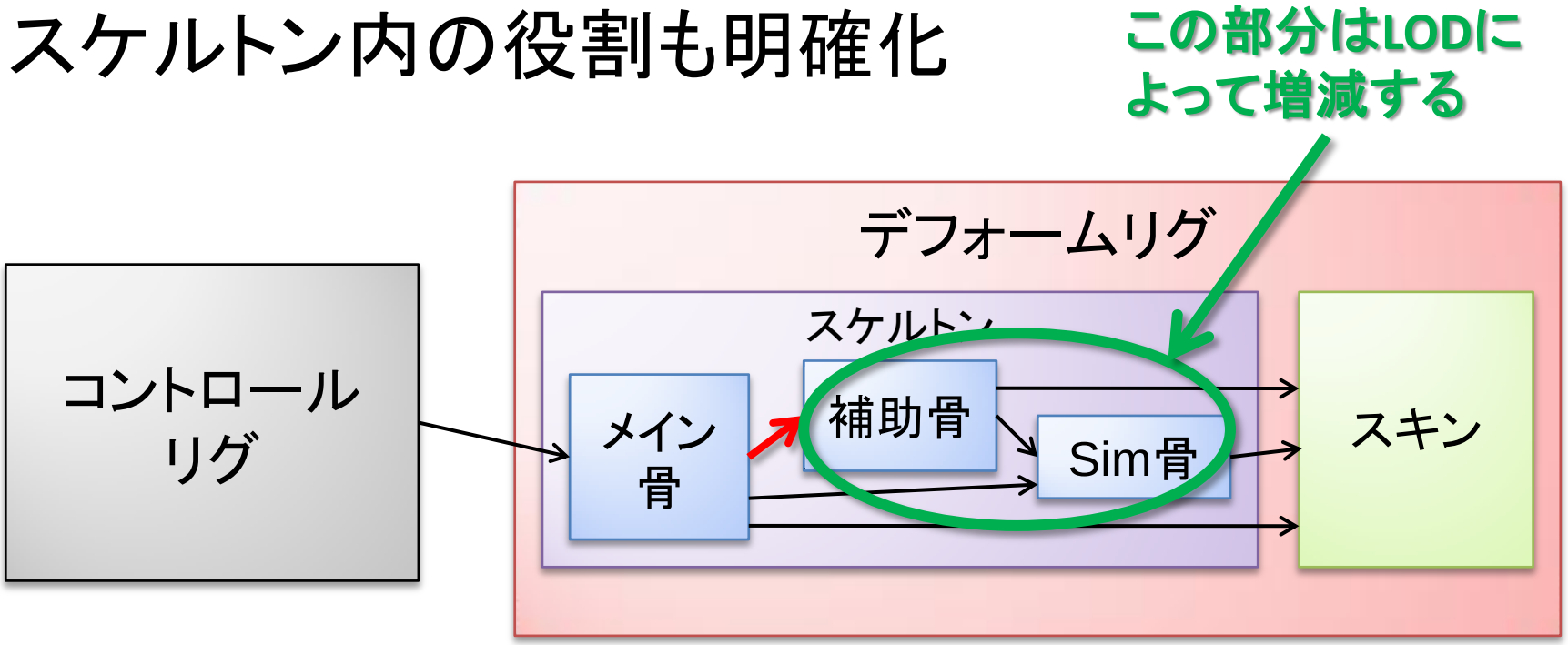


(プロットしたら不要)
(用途毎に切り替え可能)

(これだけで存在可能)

リグの構造: 結論

スケルトン内の役割も明確化

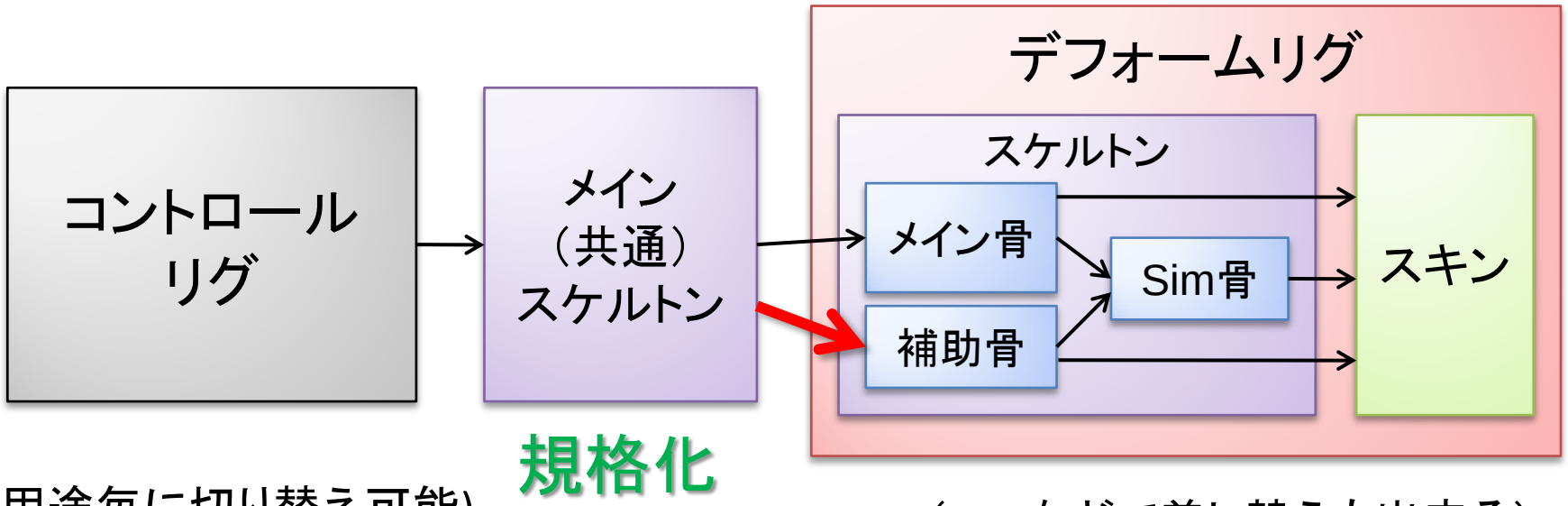


(DCCツール上だけの構造)

(実機でも動作させる構造)

(参考)映像制作の場合の構造例

補助骨連動ロジックすら外に出した



(用途毎に切り替え可能)

(プロットしたら不要)

(さらにプロット可能)

（LODなどで差し替えも出来る）

(これだけで存在可能)

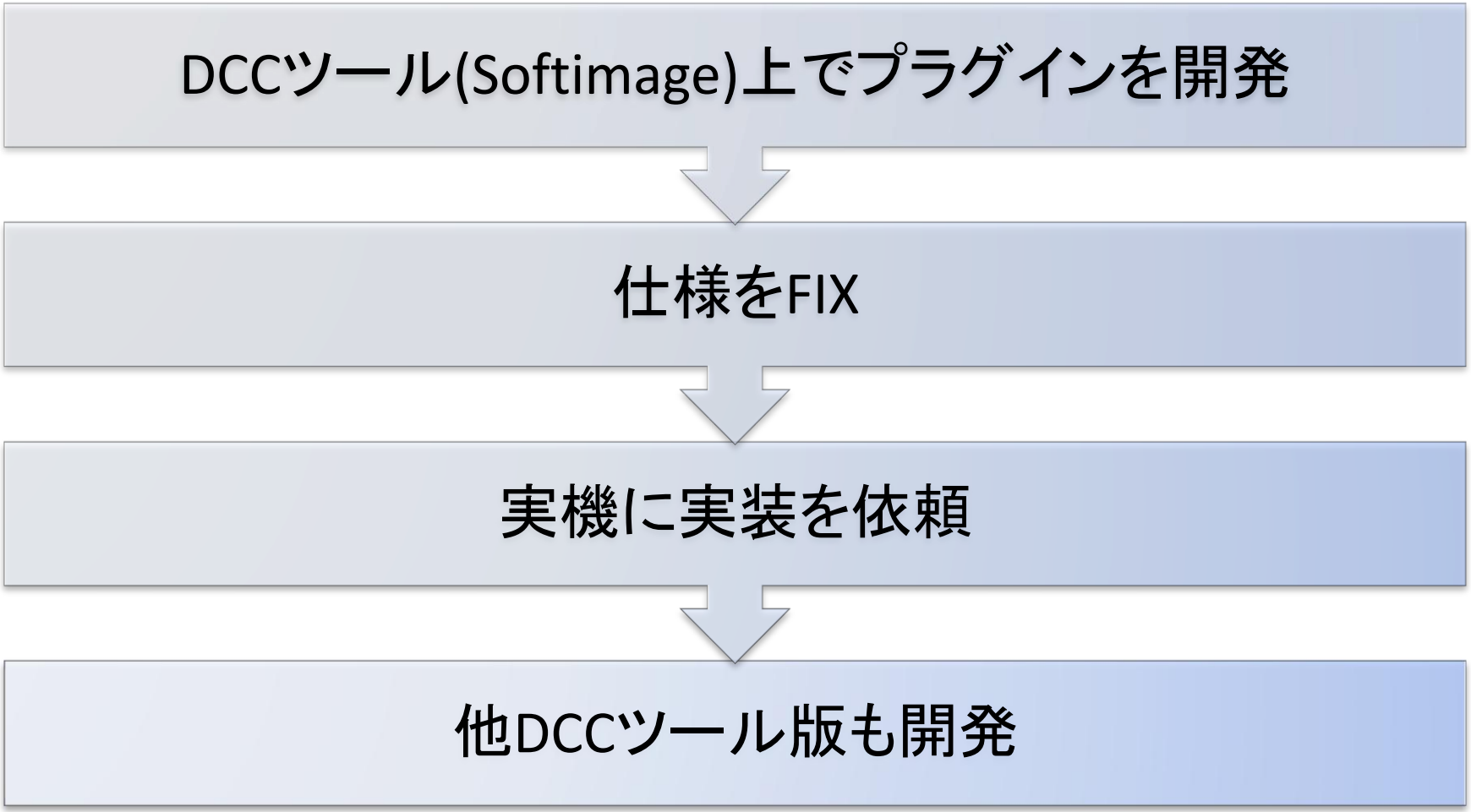
開発プロセス

- 開発の流れ
  - 基礎知識としての数学の話
    - 要求分析と仕様作成

“KineDriver” で目指した環境

- 細かな調整をアーティスト自身が行える
- 表現したい事をより高品質に適える
- それなりの柔軟性（ノードベースのビジュアルプログラミングとまではせず、必要機能を見極めて搭載）
- DCCツール上でセットアップ
- DCCツールと実機で同じように動作
- 特定DCCツールやゲームエンジンに依存しない
- フレーム独立（モーションヒストリ非依存）

開発アプローチ



実装

- 仕様がFixするまではスクリプトオペレータで開発してテスト（Mayaで例えると expression か Python ノードプラグイン）。FixしたらDLLプラグイン化。
- XMLファイルに情報をエキスポート/インポート。実機やDCCツール間のデータコンバート。
- Softimage都合の部分とコア部分を明確に分離してコーディング。実機側は、コア部分のソースコードをプログラマに示して実装を依頼。

開発プロセス

- 開発の流れ
- 基礎知識としての数学の話
- 要求分析と仕様作成

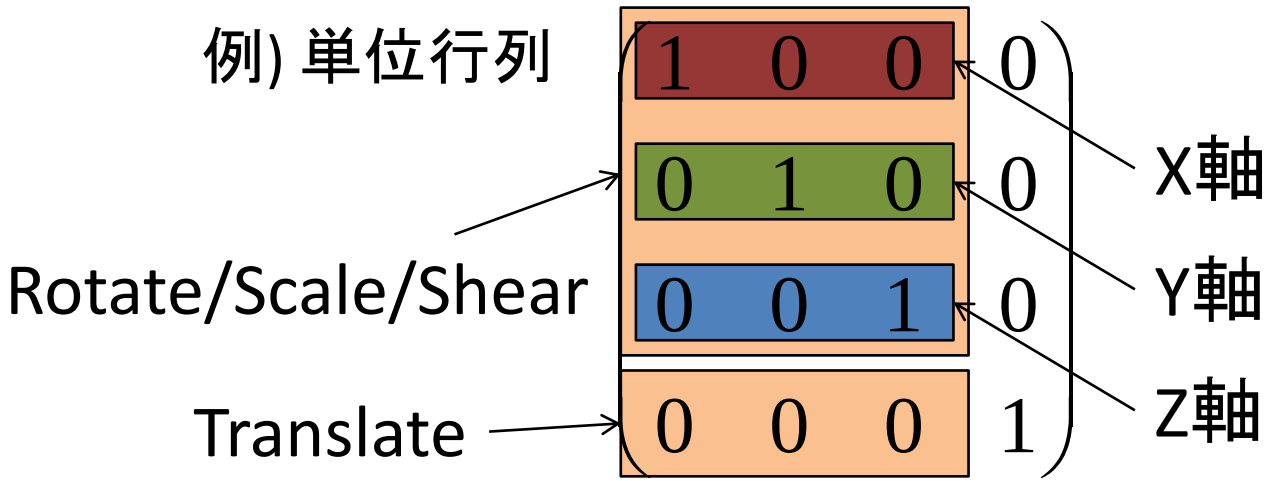
少しだけ「リギング数学」

- リグを作るのも数学を使えると便利
- Softimage
  - ICE Kinematics
  - Script Operator
- Maya
  - Dependency Graph
  - Expression

用語と簡単な
理屈だけ
数式は無し！

トランスフォーム情報の内部表現

- 4x4 マトリックス（行列）がよく使われる



- 乗算で合成する。親子の合成も。

(Transform) = (Shear) * (Scale) * (Rotate) * (Translate)

(子のGlobal) = (子のLocal) * (親のGlobal)

(子のLocal) = (子のGlobal) * (親のGlobal)$^{-1}$ ← 逆行列

行列以外の回転表現について

オイラー角 (rotX, rotY, rotZ)

- 意味を理解し易い
- ジンバルロックなど、問題が多く、扱い難い面も

任意軸周りの回転 (θ, (x,y,z))

- 全ての回転姿勢は一軸回転で表せる

クォータニオン (w,x,y,z)

- 意味不明な4個の数
- 合成や真っ直ぐな補間(slerp)が出来るなど、扱い易い

オイラー角を使用すべきかどうか

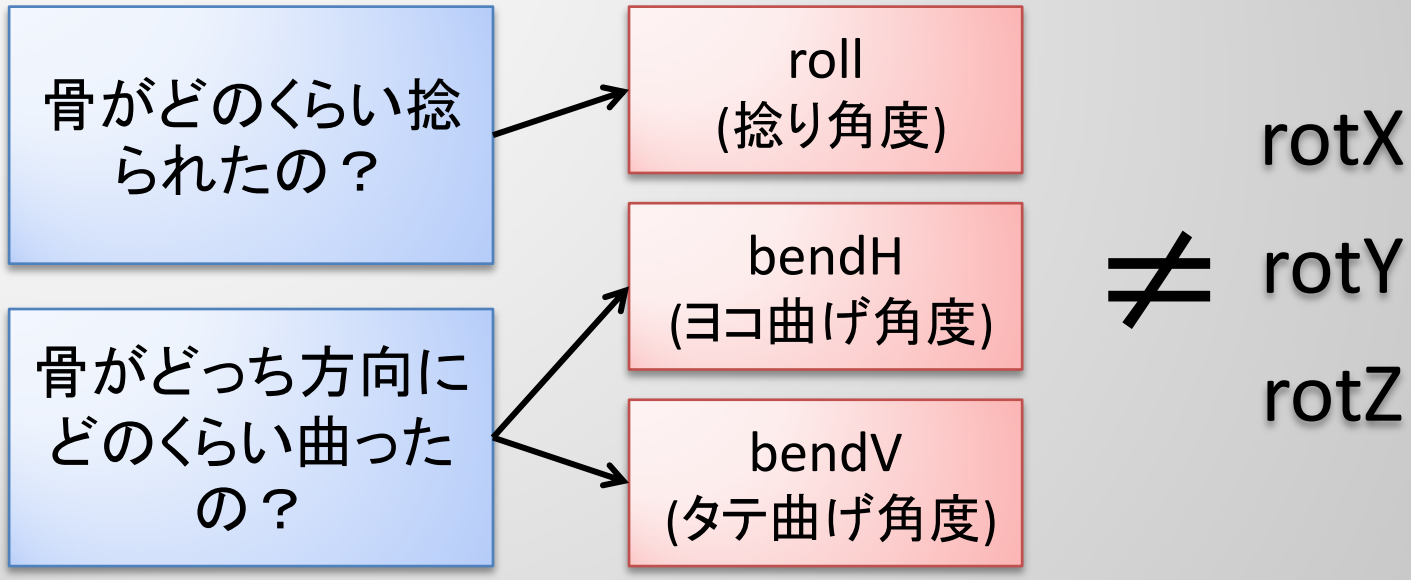
- 真っ直ぐな補間が出来ない
  - 「この回転の半分くらい追従」といった事が出来ない。
- 実機では、クォータニオンやマトリックスで持っている
  - オイラー角で表現された関節連動の式を実機で評価する為には、オイラー角に分解して、式を評価、またクォータニオンに戻す処理が必要になる。
- 一つの状態を表す値の組み合わせが複数存在する
  - 例えば (0,90,60) と (-30,90,30) は同じポーズ
  - オイラー角で表現された式を確実に再現するのは困難

『本当に表現したいこと』を考えると、もっと適した表現形式がある筈

『曲げ』と『捻り』で回転を表現

回転をスケーリングしたり合成したりするだけならばクォータニオンでやればいい。その機能は必要。

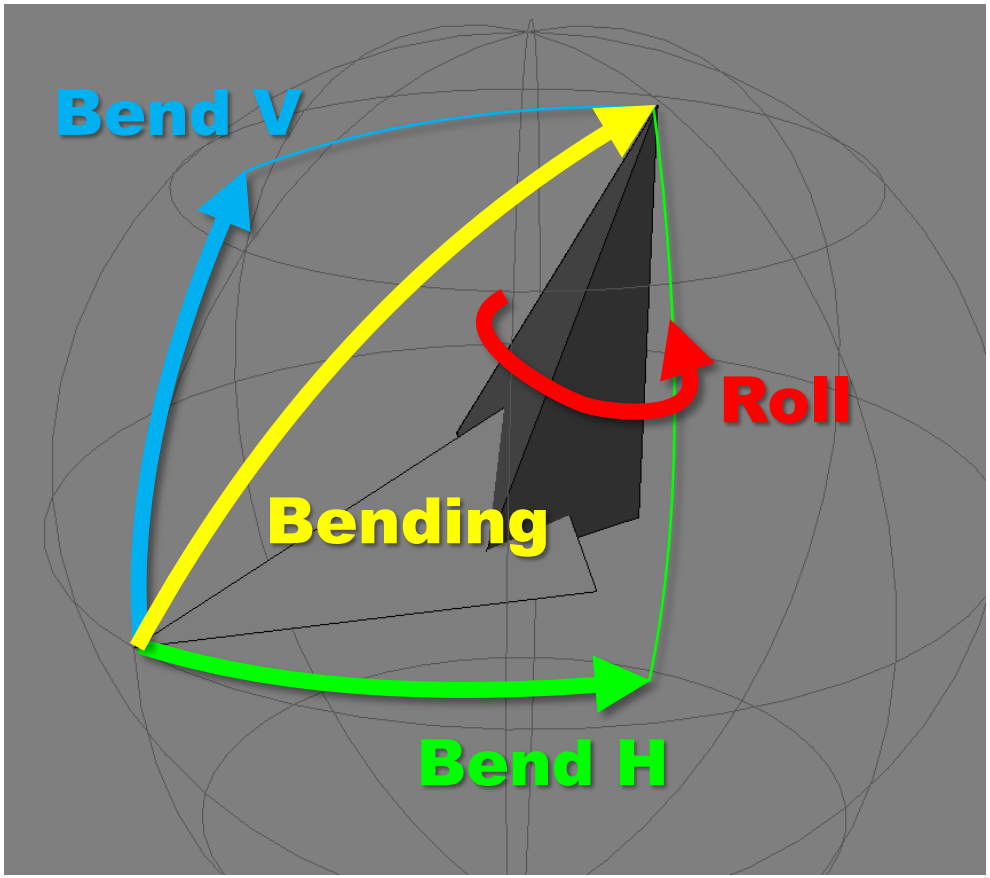
回転を要素で扱いたいことも多い。



『曲げ』と『捻り』の分解イメージ

1. 「曲げ」とは、骨ベクトルが目標方向へ向く為の最短経路の回転。
2. 「捻り」とは、その残りの回転角度。
3. さらに、任意のアップベクトルを決め、曲げ角度を縦横2方向に分解。

※合成処理はその逆を考える。



『曲げ』と『捻り』の算出

- exponential map を使用すると、目的に近い値が得られる。

分解: $v = \ln(q)$

合成: $q = \exp(v)$

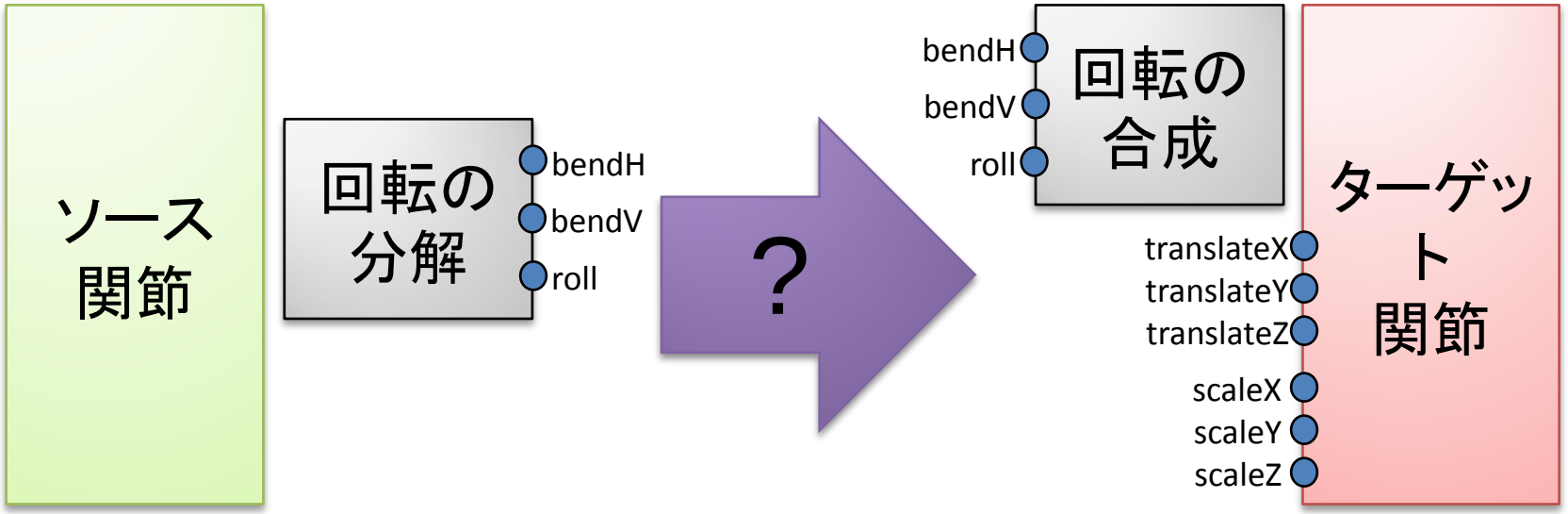
$v = (roll, bendH, bendV)$

開発プロセス

- 開発の流れ
- 基礎知識としての数学の話
- 要求分析と仕様作成

要求分析の為のテスト環境

- 回転の「分解」と「合成」を簡易に実装
  (オフラインの映像制作ならツールはこれで完了だが・・・)



それを使って人体リグを試作してもらう
(どのような関節連動を組みたいのか、ニーズを調査)

試作データを分析

- Softimage の expressionでかなり頑張って作ってくれた。

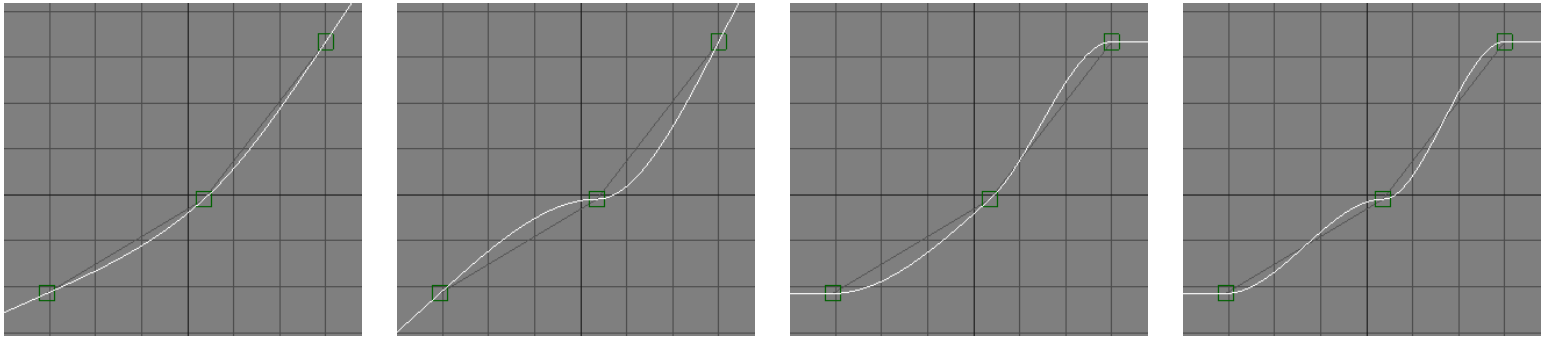
```
-0.836482 + MIN( ( ( cos( MIN( 150, MAX( 0,
pc0_set.l_femur.HappyJointRotation.bendV ) ) / 150 * 180 - 180 ) / 2 + 0.5 ) *
-1.5 + ( cos( MAX( -60, MIN( 0, pc0_set.l_femur.HappyJointRotation.bendV ) ) /
-60 * 180 - 180 ) / 2 + 0.5 ) * -0.05 ) * cos( MIN( 120, MAX( 0,
pc0_set.l_femur.HappyJointRotation.bendH ) ) / 120 * 90 ) + ( cos( MIN( 90,
MAX( 0, pc0_set.l_femur.HappyJointRotation.bendH ) ) / 90 * 180 - 180 ) / 2 +
0.5 ) * -0.7 * cos( MIN( 120, abs( pc0_set.l_femur.HappyJointRotation.bendV ) )
/ 120 * 90 ), ( cos( MIN( 150, MAX( 0, pc0_set.r_femur.HappyJointRotation.bendV
) ) / 150 * 180 - 180 ) / 2 + 0.5 ) * -0.2 * cos( MAX( -120, MIN( 0,
pc0_set.r_femur.HappyJointRotation.bendH ) ) / -120 * 90 ) )
```

とある一か所（殆どこんな感じ）・・・さすがにこれでは大変！

- Softimageのexpressionはパラメータ間の「式」を記述する機能なので、三次元ベクトル計算などはほぼ出来ない。
- 回転を取扱い易くしただけではまだ足りない。表現したい事を汲み取って、合理的な機能として提供する必要がある。

用意すべきパラメータ接続手段

- 単なる直結や反転
- 単にスケールやオフセット、リミットをかけたい
  ⇒リニアに限定した処理
- 動き出しとリミットを EaseIn/EaseOut したい
  ⇒入出力の対応を2～3点指定する簡易Bezier Curve



- より複雑なケースが稀にあるかも
  ⇒自由なFCurve (Non-weighted Bezier Curve)

その他の必要な機能

さらに複雑なエクスプレッションの意義は？

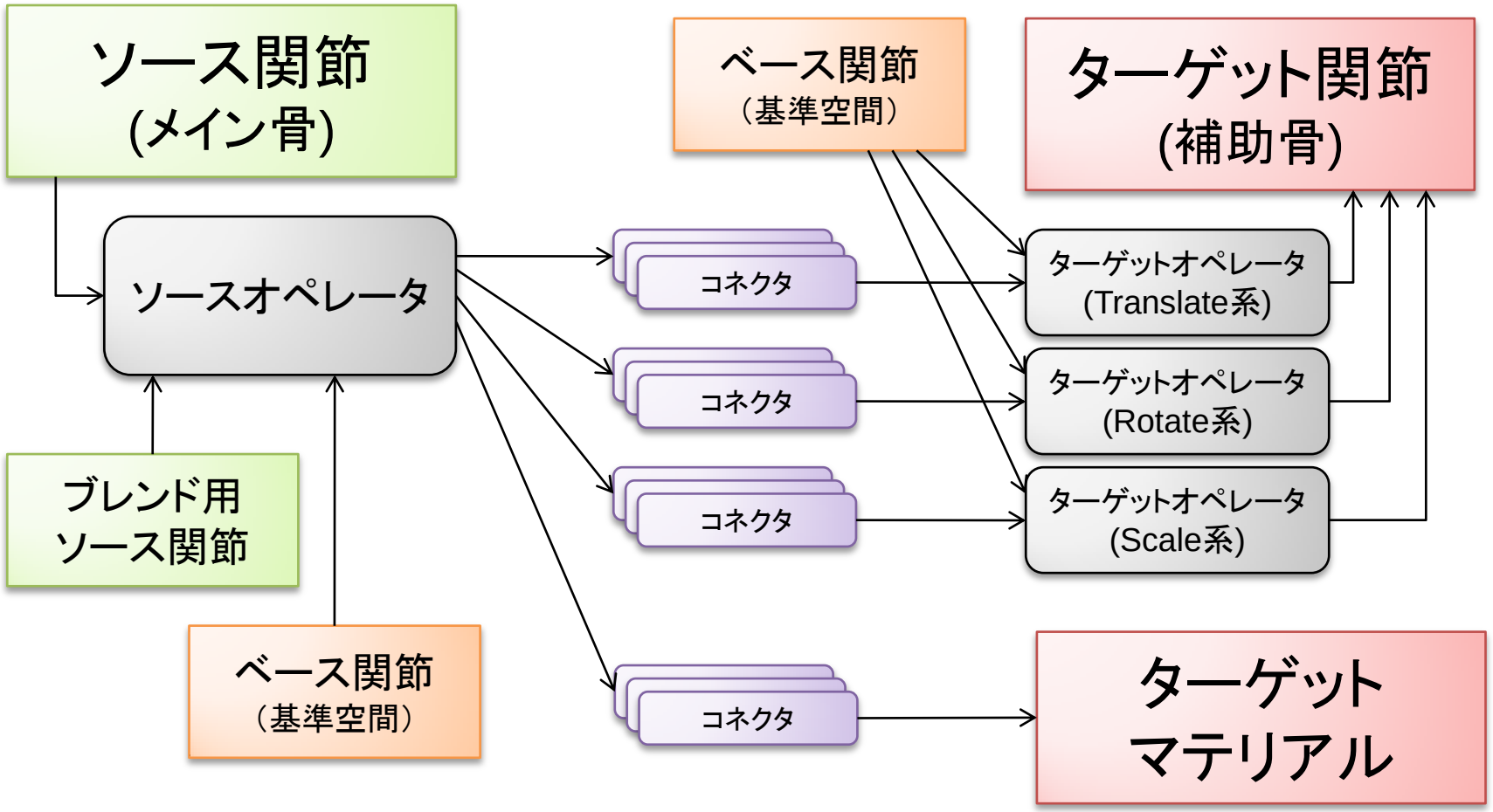
分解した角度で足したり引いたりしてもらうより、三次元の合理的な機能を提供したい。

- 「複数関節の回転のブレンド」をしている
  - ソース関節のブレンド機能を設ける。
- 「複数関節の回転の合成」をしている
  - どこかの関節基準での位置や回転を扱いたいということ。
    例) 背骨基準で肘の位置・・・、腰基準での首の向き・・・など
  - 「ベース関節」(基準空間)指定機能を設ける。

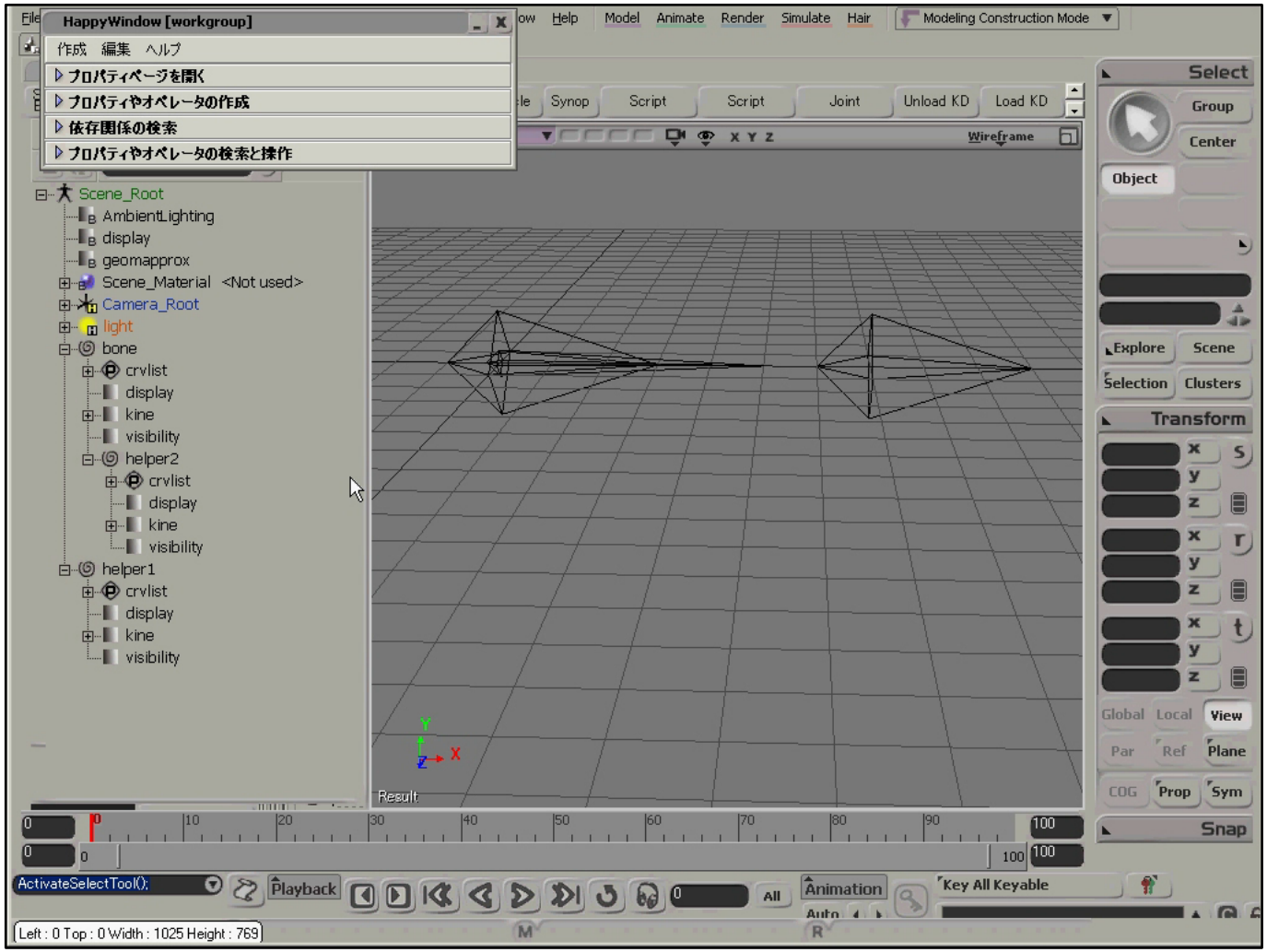
システムの詳細

- オペレータの基本構成
  - ベース関節指定機能
  - コンストレイン機能

基本構成図



デモ：オーサリング



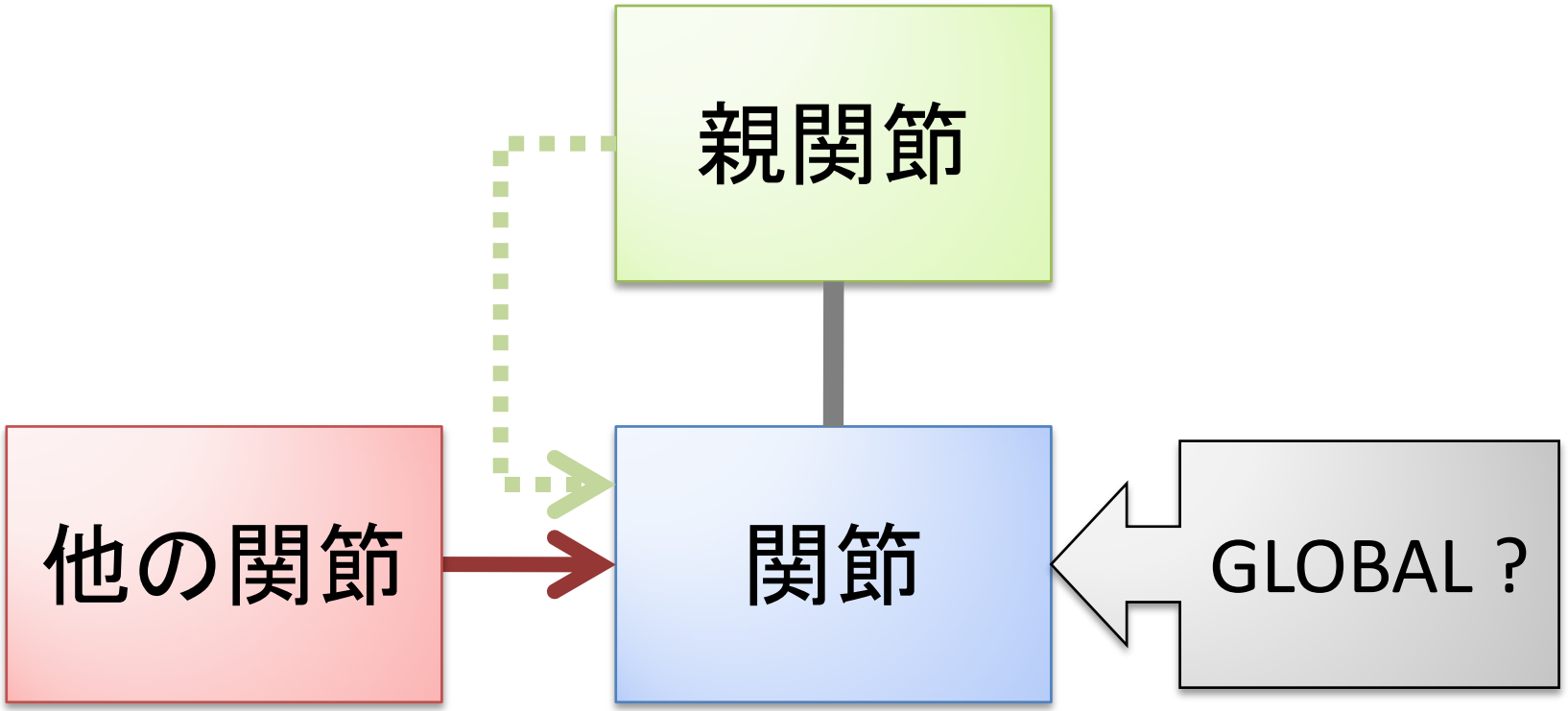
オペレータ依存順の解決

- 補助骨が別の補助骨を動かすという組み方も許容。
- 通常、DCCツールでは、依存関係が考慮されて値の評価時に再計算必要な箇所のみが計算される。
- 一方、通常、ゲームエンジンでは、毎フレーム、シーン全体が評価される。
- 補助骨システムのエキスポータは、DCCツール上の依存グラフを解析して、毎フレーム計算させる順番にソートしてオペレータ情報をXMLファイルに書き出す。
- ゲーム側実装は、ファイルに書かれた順に実行する。

システムの詳細

- オペレータの基本構成
- **ベース関節指定機能**
- コンストレイン機能

ベース関節指定機能



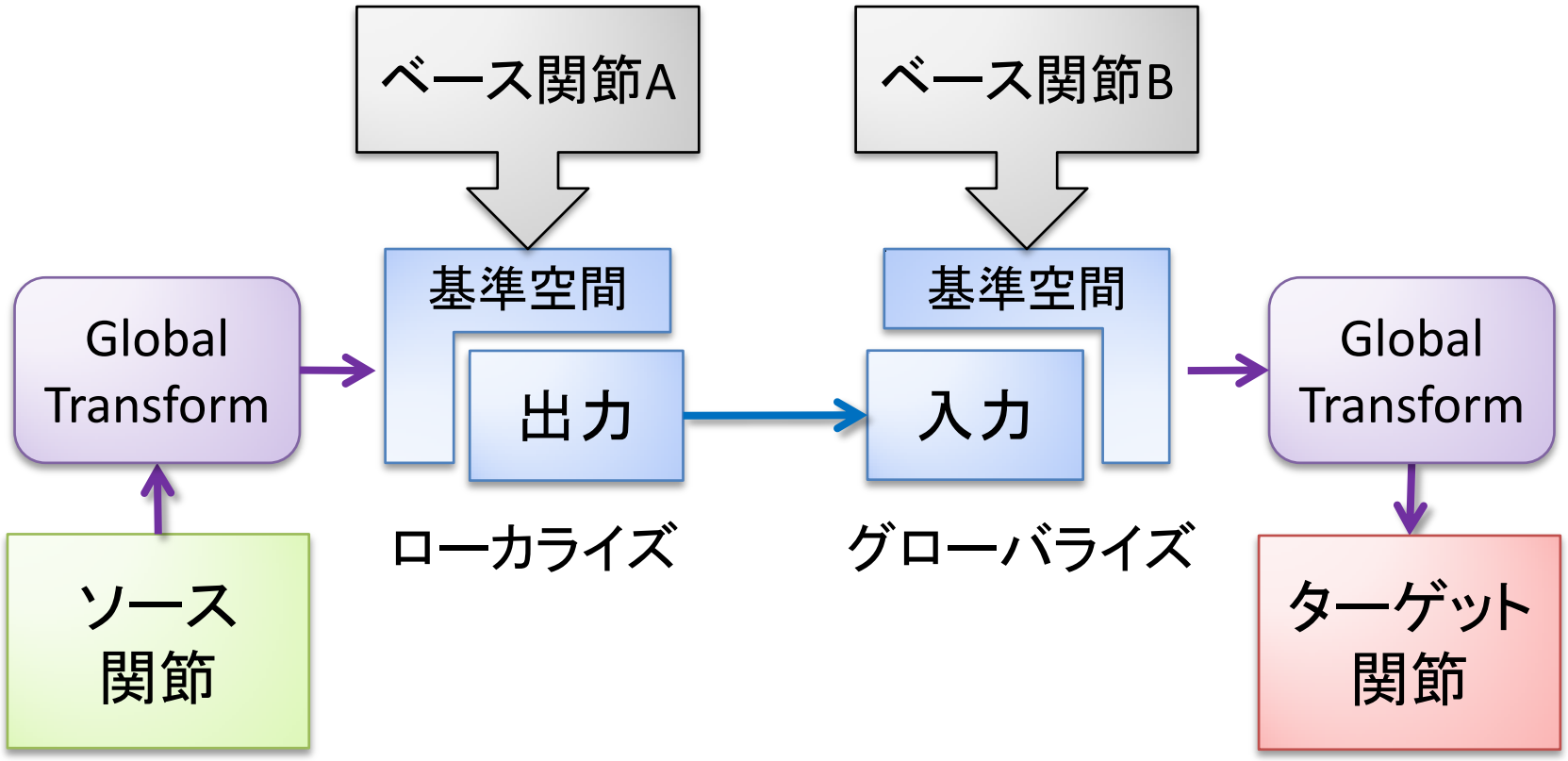
どの空間で操作する？

基準空間機能の意義

- リギングでは、どこかのノードの座標系基準での、あるノードの位置や回転が欲しいという状況はよくある。
  例) 背骨基準で手首や肘の位置を得て肩の補助骨を制御
- 別の空間に属すモノ同士を、それぞれの空間基準で動きを合わせたりする事が容易。
  例) 右腕の動きをそのまま反転して左腕に適用
- ソースとターゲットの基準が GLOBAL ならコンストレイン機能そのものでもある。
- プログラマブルにしなくても、これで結構なニーズに対応できる事がわかった。意外とDCCツールに備わっていない機能。

実装や考え方も簡単な割に効果が大きい

基準空間の計算イメージ

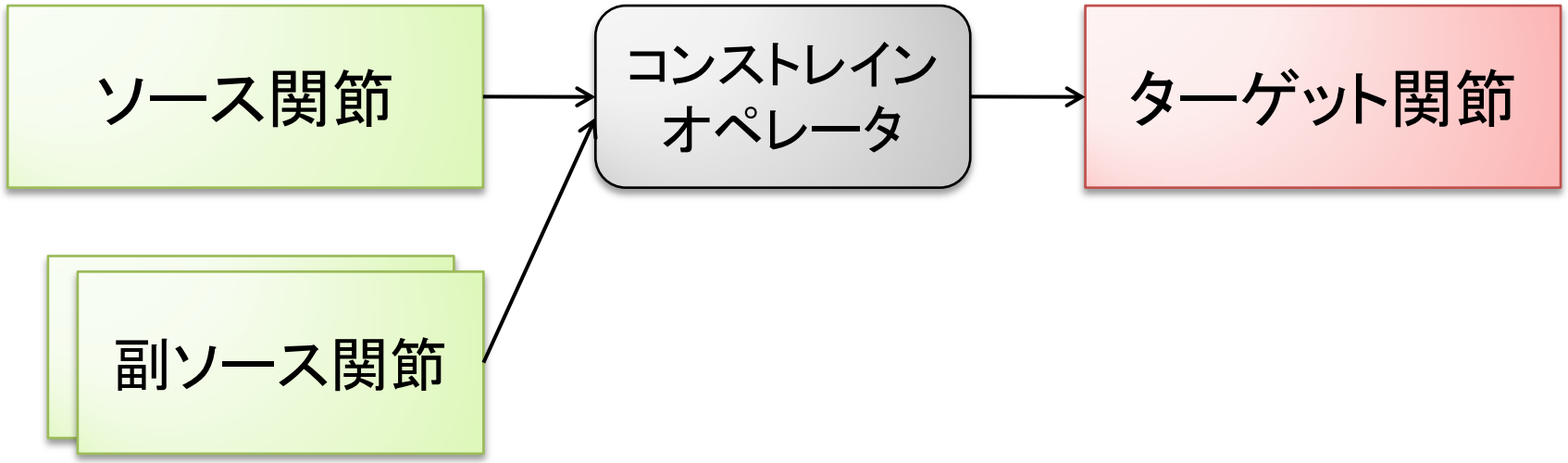


システムの詳細

- オペレータの基本構成
- ベース関節指定機能
- **コンストレイン機能**

コンストレインオペレータ

Position, Orientation, Direction を実装。



「ソースオペレータやコネクタを必要とせずに、ソース関節のトランスフォームを直接参照する特殊なターゲットオペレータ」という位置づけ。

なぜプラグイン？

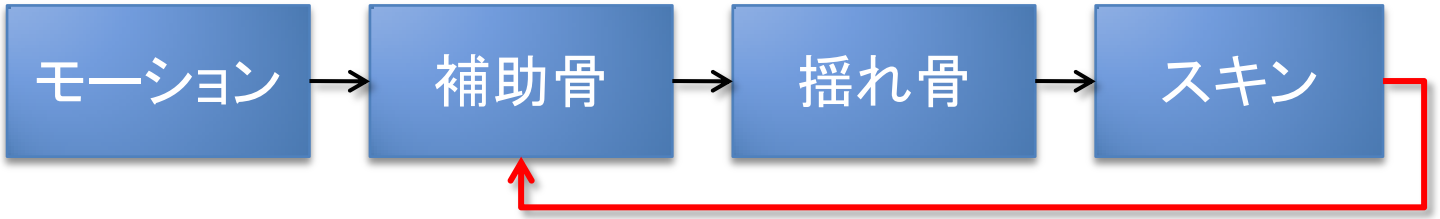
扱い易さと、他DCCツールと実機を考慮した設計

Softimageのコンストレインスタックは柔軟で強力ではあるが、実機や他DCCツールに移植が困難になる。

- 複数ソース設定を1オペレータでサポート。Mayaの仕様に近く、ウェイト設定が簡単。
  - 例えば、3つのソースを等分にブレンドする場合、Softimageでは 1.0、0.5、0.333… とウェイトを設定する必要があるが、もっと直観的に 1.0、1.0、1.0 などと指定出来る。
- Direction コンストレインに アップオブジェクトを指定しなくても、そのままで安定して動く。

コンストレインの応用

- メッシュ表面上への拘束機能も実現したい。
- しかし、これを盛り込むと、実機への実装が難しくなる。

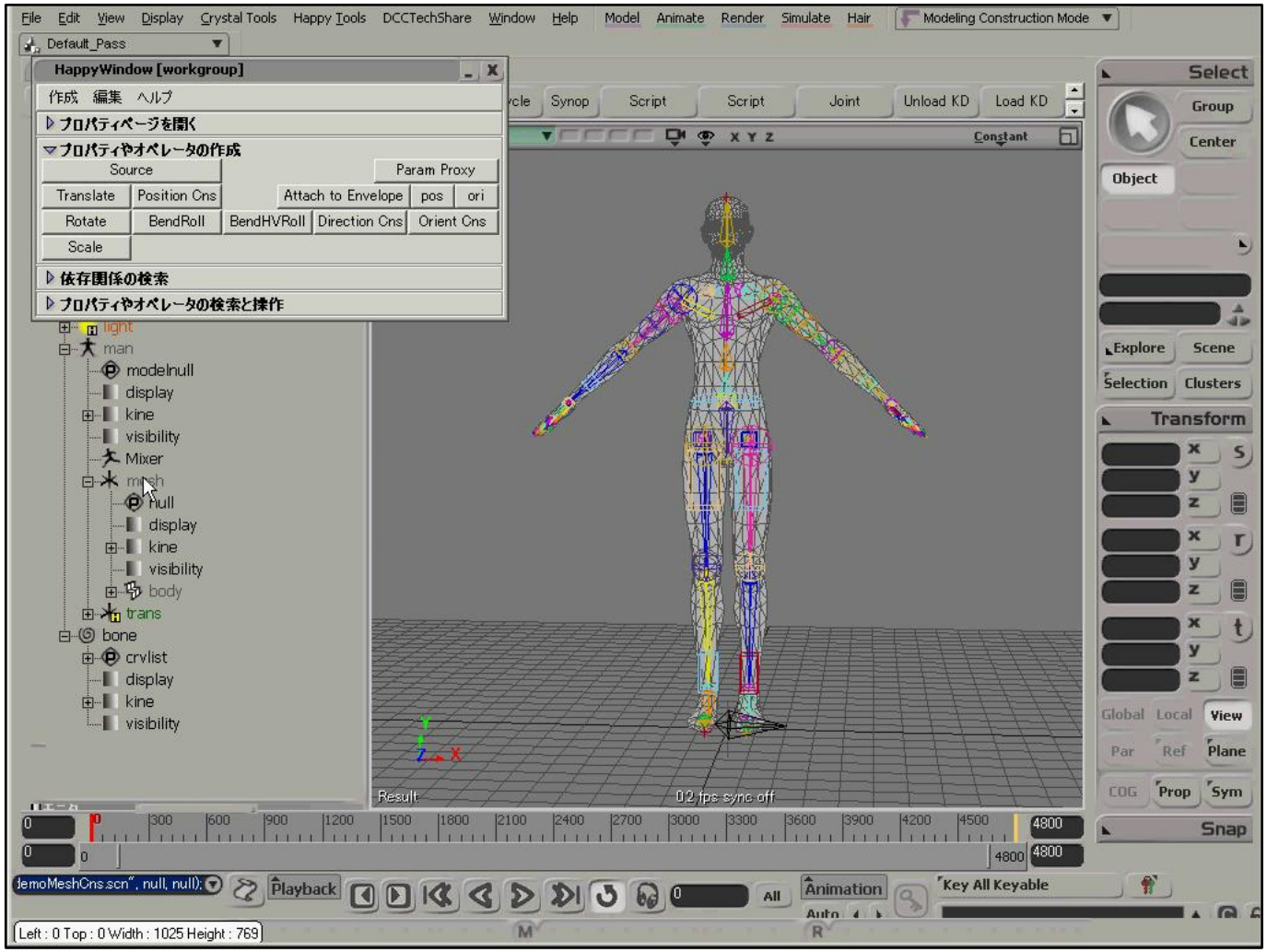


処理フェーズが明確に分かれなくなってしまう！

通常コンストレインで擬似的に実現

スキン上の拘束位置における複数の影響関節をソースとし、ウェイトとオフセットを細かく設定

デモ：メッシュへの疑似拘束



DCCツールごとの実装と問題点

- DCCツールごとの実装紹介
  - 階層スケーリングの対応
  - チェーンによる問題

Softimage版 実装の問題

- カスタムオペレータで実装
  - ノード間の繋がりを俯瞰できず分かりずらい
  - 現状なら、ICE Kinematics で実装すべき
- 思いのほか重たいものに・・・
  - キャラの Frame Rate が 1/3 くらいになってしまう
  - 計算そのものは軽い筈・・・
  - 原因は、カスタムオペレータのオーバーヘッド
    - パラメータやマトリックスの Get/Set に要する処理時間

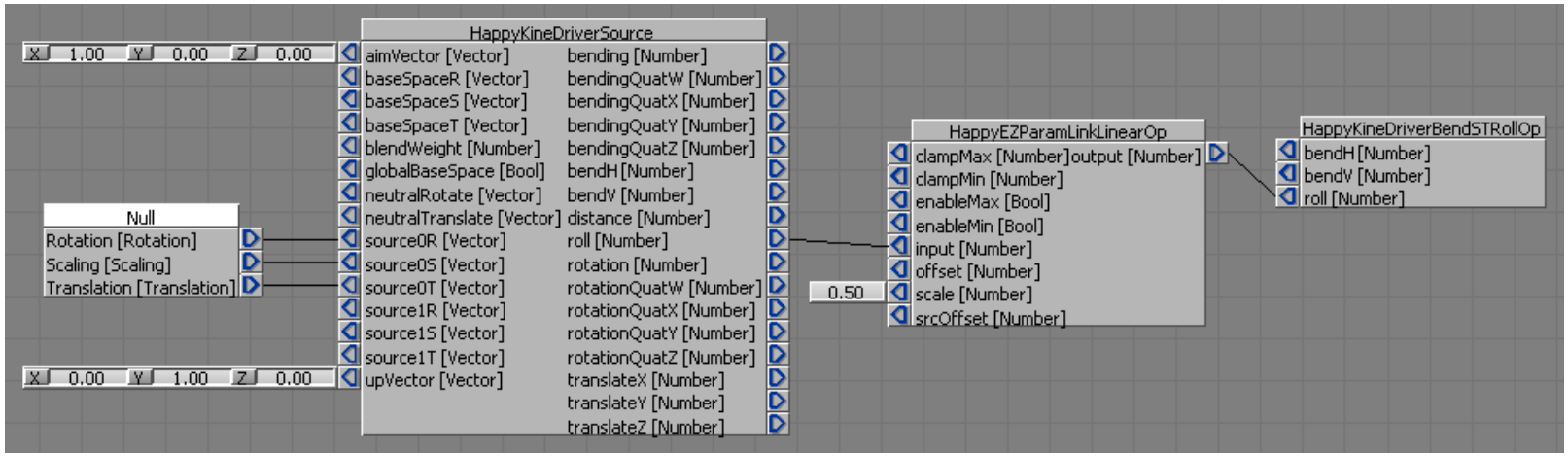
カスタムオペレータの多用はお勧め出来ない

Maya版(予定)

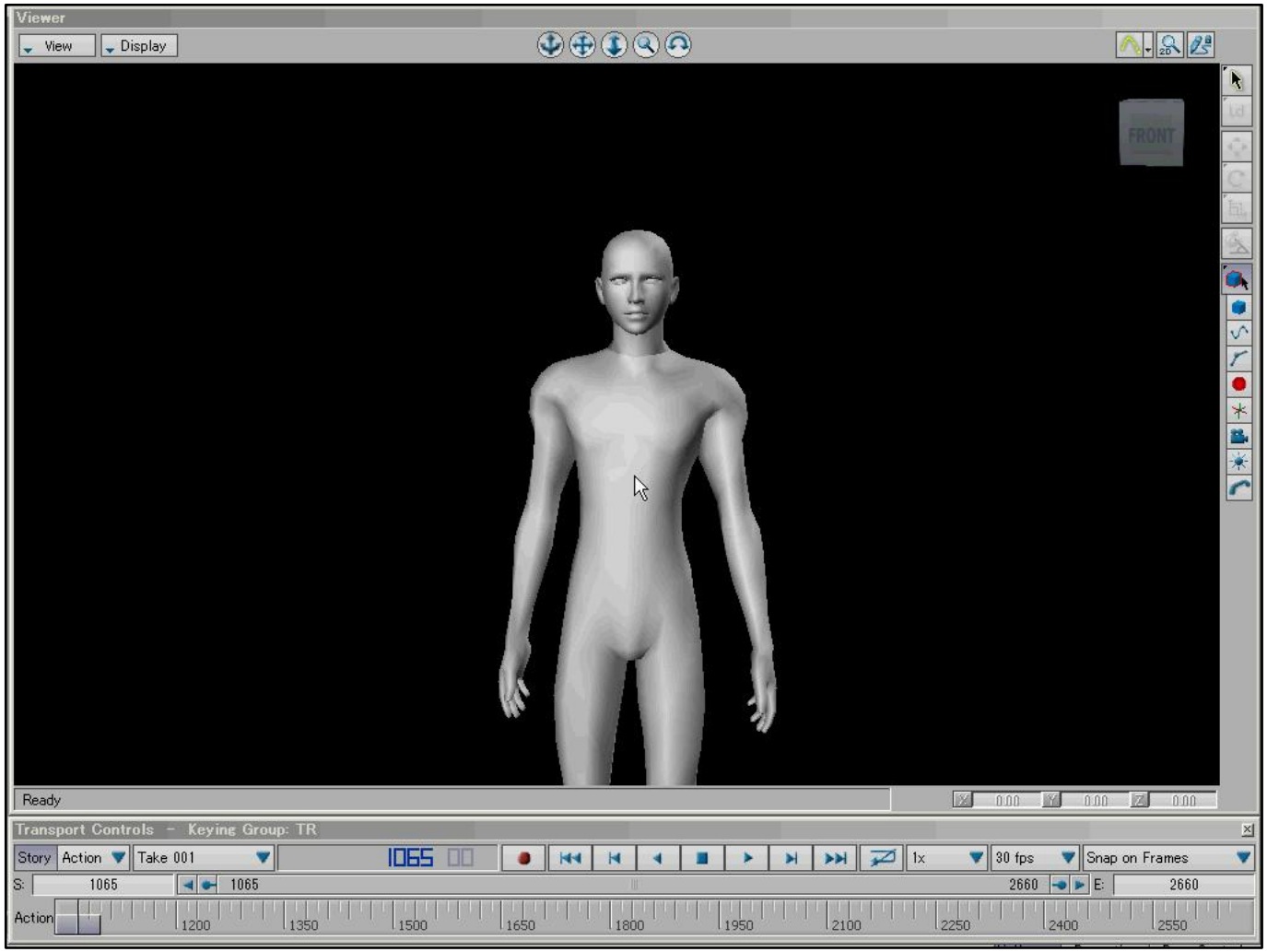
- Dependency Graph 上に素直に実装可能。
  - 素直に実装出来れば、信頼性も高く、高速。
- 今後の社内ニーズに応じて開発。

MotionBuilder版

- 編集環境はあまり考慮せず
  Softimage上で作成したものをインポートして再生。
- 「コンストレイン」と「リレーション」として実装
  - ターゲットオペレータはコンストレイン。
  - その他はリレーションボックス（ノード）。
  - 高速化の為、リレーション空間は一つに統一。



デモ：MotionBuilder版



3dsmax版

- 社内での利用率が低い為、今のところは考慮していない。

DCCツールごとの実装と問題点

- DCCツールごとの実装紹介
- 階層スケーリングの対応
  - チェーンによる問題

「階層スケーリング」とは？

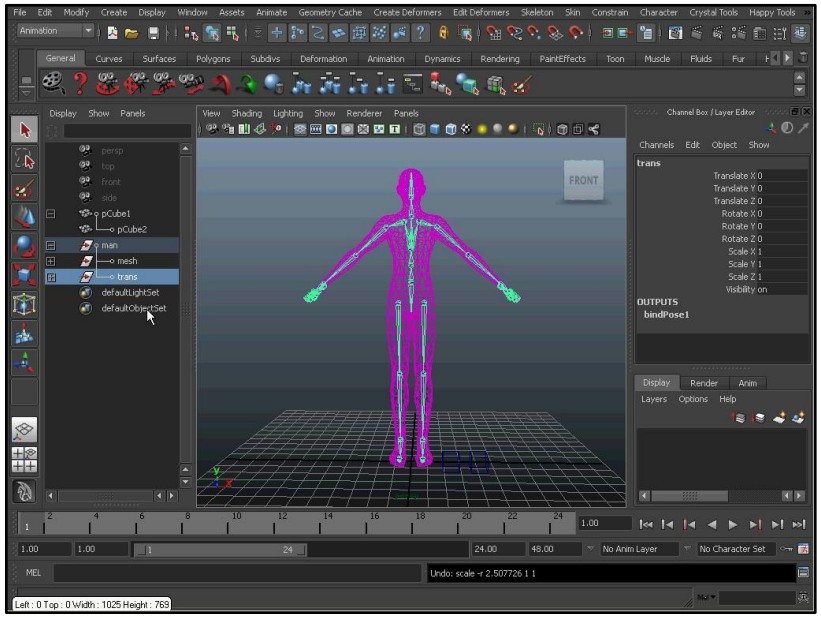
- スケールと回転を混ぜずに別々に乗算していく考え方。Softimageスケーリングとも呼ぶ。
- DCCツールの対応状況
  - Softimage: 階層スケーリング（本当は両対応）
  - Maya: 普通のスケーリング
  - MotionBuilder: 両対応（fbx作成元に合わせる）

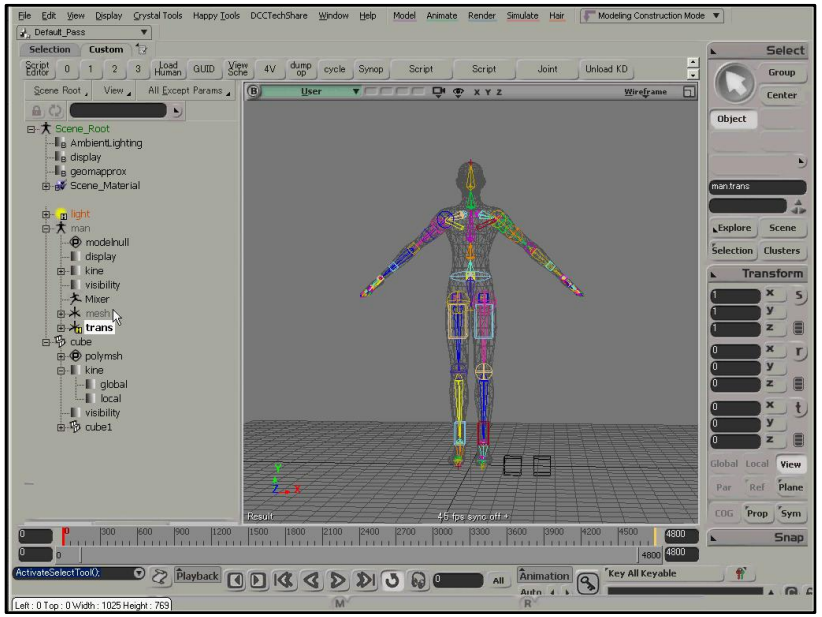
DCCツール間でリグやモーションの互換を考える場合、意識する必要がある。

「階層スケーリング」の意義

- 利点
  - Shear による歪みが発生しない
  - アニメータがコントロールし易い
- 欠点
  - モデル全体のスケール変更が面倒
  - 行列演算の計算量が多い
  - ローカライズの計算も小難しい（逆行列を掛けるだけじゃダメ）

デモ：階層スケーリングとShear





“KineDriver”における対応

基本前提

- “KineDriver”は階層スケーリングを採用。
- DCCツールがどちら仕様であろうが、”KineDriver”の考え方は変えず。Global Matrix を一致させれば良い。

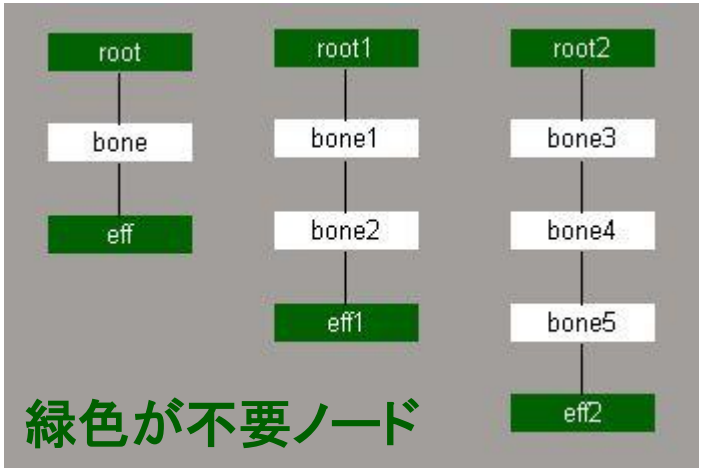
- Maya版(予定)
  - Shear アトリビュートで吸収される筈。
- MotionBuilder版
  - そもそも Shear プロパティを持っていない！
  - 階層スケーリングの骨でしか動作しない仕様とした。

DCCツールごとの実装と問題点

- DCCツールごとの実装紹介
- 階層スケーリングの対応
- チェーンによる問題

「チェーン」とは？

- Softimageでスケルトンといえば「チェーン」
  - IK機能がパッケージされたボーンの連なり
  - IKする単位で1チェーンとなる
  - 1チェーンにつき、実機では不要なノードが2個挿入される（ChainRoot と Effecter）
- Nullノードでスケルトンを組むという考え方もあるが、弊社ではチェーンが使用される。



チェーンによる問題

それまで、チェーンが問題になることはなかった

- 実機にコンバートする際に余計なノードは削除
- モーションデータのコンバートでも考慮

しかし “KineDriver” では問題になる

- 削除されるノードがソースにされていたら？
- 削除されるノードが基準空間に指定されていたら？
- 削除されるノードの子供がソースやターゲットだったら？（これが特に厄介）

今回の対応

Nullによるスケルトン構造を導入すべき？

↓

キャラモデラーの作業効率の点で断念

- MayaやMotionBuilderのように、チェーンでないボーンエレメントが標準で在ると良いのだが・・・

↓

対処

- 殆どの事態は、実機実装とコンバータで吸収
  - 処理の複雑化と、プログラマの負担増の問題は残る
- 解決出来ない事態は、制限事項としてルール化
  - パターンが多く、ルールが複雑
  - 多くはオペレーション制限で封じる
  - 封じ切れない問題はファイルエキスポート時にエラーに

まとめ

まとめ

- 成功点
  - 原理が簡単な割に自由度の高い仕組み
  - 設計方針からぶれずに目標を達せた
  - 設計も汚くはないと思う
  - テクニカルな布教も少し出来た
- 反省点
  - 初めて触る人には難しいらしい
  - Softimage版のパフォーマンス
  - チェーン問題の為、運用ルールが複雑に

ご清聴ありがとうございました

ご質問をお受けします。

連絡先:
佐々木 隆典　ryusukes@square-enix.com

Twitter ハッシュタグ: #hojobone