



# コンピュータアニメーション特論

## 第8回 キャラクタアニメーション(1)

九州工業大学 情報工学研究院 尾下真樹

# 今日の内容

- 人体モデルの基礎
- 人体モデル(骨格・姿勢・動作)の表現
  - 骨格モデルの表現
  - 姿勢・動作の表現
  - 形状モデルの表現
- 人体モデルの作成
- 動作データの作成
  - キーフレームアニメーション
  - モーションキャプチャ



# キャラクター・アニメーション

- CGにより表現された人体モデル(キャラクター)のアニメーションを実現するための技術
- キャラクター・アニメーションの用途
  - オフライン・アニメーション(映画など)
  - オンライン・アニメーション(ゲームなど)
    - どちらの用途でも使われる基本的な技術は同じ(データ量や詳細度が異なる)
    - 後者の用途では、インタラクティブな動作を実現するための工夫が必要になる
- 人体モデル・動作データの処理技術



Motion Playback

0.05 (5)

**キャラクタ アニメーション**  
**Human Animation**



# 全体の内容

- 人体モデル(骨格・姿勢・動作)の表現
- 人体モデル・動作データの作成方法
- サンプルプログラム、動作再生
- 順運動学、人体形状変形モデル
- 姿勢補間、キーフレーム動作再生、動作補間
- 動作接続・遷移、動作変形
- 逆運動学、モーションキャプチャ
- 動作生成・制御



# 今日の内容

- 人体モデルの基礎
- 人体モデル(骨格・姿勢・動作)の表現
  - 骨格モデルの表現
  - 姿勢・動作の表現
  - 形状モデルの表現
- 人体モデルの作成
- 動作データの作成
  - キーフレームアニメーション
  - モーションキャプチャ



# キャラクター・アニメーションの実現方法

- オフライン・アニメーション制作
  - 通常は市販のアニメーション制作ソフトウェアを利用
- オンライン・アニメーション生成
  - ゲームエンジン(ミドルウェア)の利用
    - Unity, Unreal 等(市販のコンピュータゲーム等でも利用されている)
    - 基本的には、アニメーション制作ソフトウェアで作成されたキャラクターモデルや動作データを再生する機能を提供
    - 提供されている機能以上の高度な動作生成・変形は困難
  - 自分でソフトウェアライブラリを開発
    - 高度な処理も自由に追加できる
    - キャラクターモデルや動作データは、他のソフトウェアで作成されたファイルを読み込んで使用する必要がある



# 前提とする基礎知識

- 位置・向き(回転)の表現と補間
  - vecmath ライブラリの使用方法
  - 「キーフレームアニメーション」の講義の内容
- 幾何形状モデルの表現、可変長配列の扱い
  - Standard Template Library (STL) の使用方法
  - 「幾何形状モデルの読み込み」の講義の内容
- OpenGL + GLUT、オブジェクト指向設計
  - サンプルプログラムを理解するために必要



# 用語についての注意

- キャラクタアニメーション分野の技術は、用語が統一されていないものがあるので、要注意
  - 例：前のスライドで出ていた、「動作接続」「動作遷移」「動作補間」等
  - 同じ概念・技術が別の名前で呼ばれたり、異なる概念・技術が同じ名前で呼ばれたりすることがある
  - 日本語／英語の両方とも
  - 本授業では、なるべく分かりやすい用語で統一した上で、別の呼び方についても補足説明する





# 人体モデルの基礎

# 人間の各要素の表現

本授業ではこの部分を扱う

- 人体の表現

- 身体の表現

- 全身の骨格・形状の表現

- 顔の表現

- 細かい表情変化を表現するためには身体とは別のモデルが必要

- 付属物の表現

- 髪の毛や衣服など

- シミュレーションによる動きの計算



# 人体モデルの表現

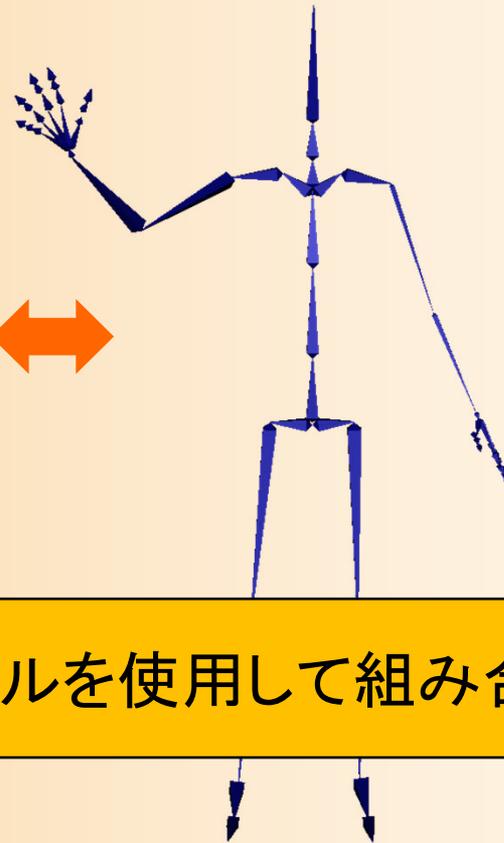
形状モデル  
(ポリゴンモデル)

骨格モデル  
(多関節体)

描画用



姿勢・動作の  
表現・処理用



形状と骨格に別のモデルを使用して組み合わせ



# 骨格モデルの表現(1)

- 人間を多関節体として扱う
  - 全関節の回転角度により、人間の姿勢を表せる
  - 骨格モデルの姿勢に応じて、形状モデルを変形
  - 人間の骨格をモデル化するためには、40～200程度の自由度が必要になる
    - 基本的な関節だけで40程度
    - 手の指や足の指なども入れると200自由度くらい必要



# 骨格モデルの表現(2)

- 多関節体モデルによる表現

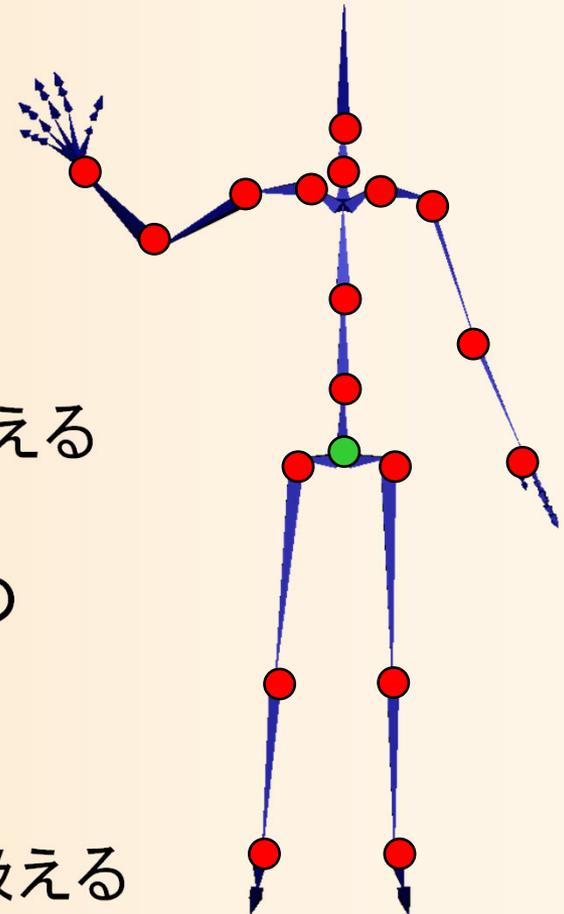
- 複数の体節(部位)が  
関節で接続されたモデル

- 体節

- 多関節体の各部位、剛体として扱える
- 複数の関節が接続されており、  
体節の長さや体節内での各関節の  
接続位置は固定

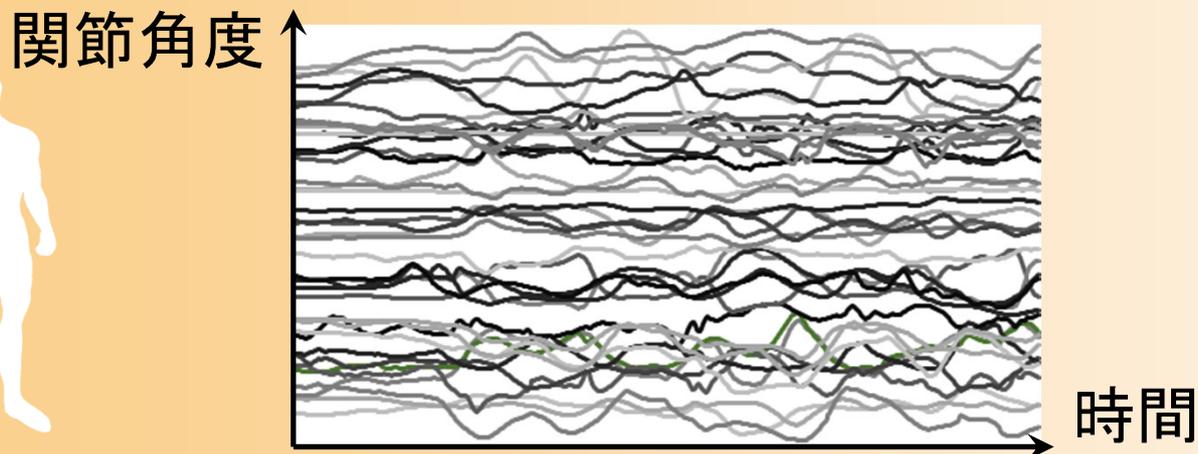
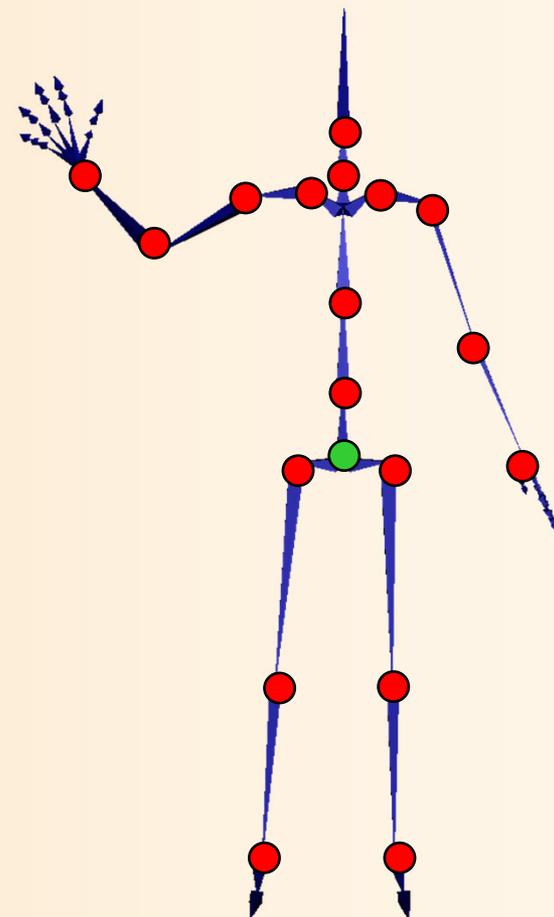
- 関節

- 2つの体節の間を接続、点として扱える
- 関節の回転により姿勢が変化する

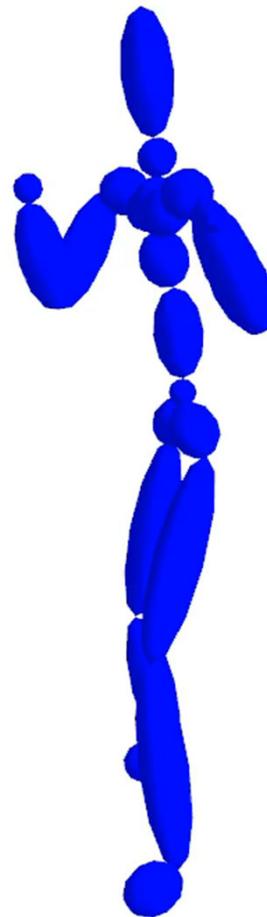


# 姿勢・動作データの表現

- 多関節体の姿勢の表現
  - 各関節の回転角度 (40~自由度)
  - 腰の位置・向き (6自由度)
- 多関節体の動作の表現
  - 姿勢の時間変化により表せる



# 動作データの例



# 姿勢データの表現

- 多関節体の姿勢の表現

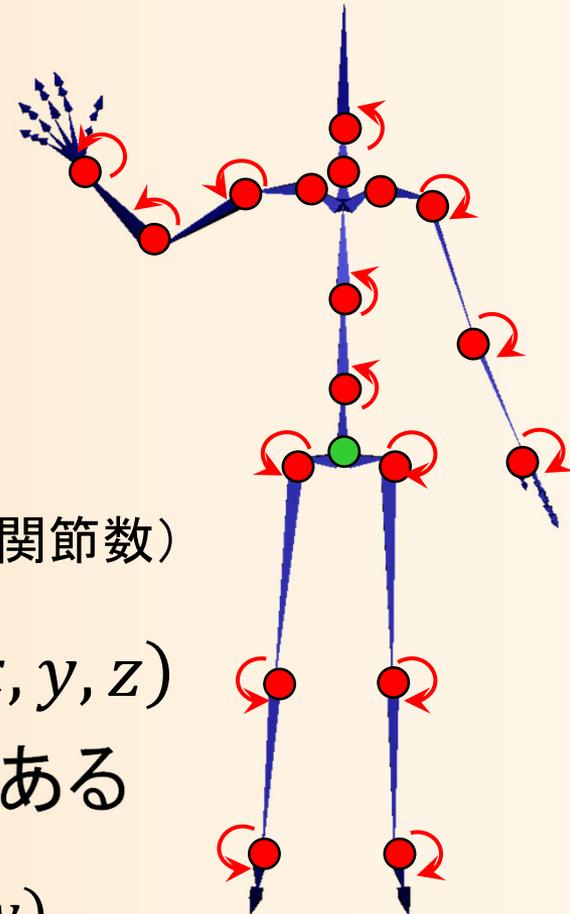
- 各関節の回転 (40~自由度)
- 腰の位置・向き (6自由度)

$$\mathbf{p} = \left( \underbrace{\mathbf{v}_r, \mathbf{o}_r}_{\text{腰の位置・向き}}, \underbrace{\mathbf{r}_1, \dots, \mathbf{r}_n}_{\text{全関節の回転}(n \text{ は関節数})} \right)$$

腰の位置・向き 全関節の回転 ( $n$  は関節数)

- 位置は3次元ベクトルで表現 ( $x, y, z$ )
- 向き・回転は複数の表現方法がある

$$\begin{pmatrix} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{pmatrix} (\theta_1, \theta_2, \theta_3) (x, y, z, w)$$



# 関節の回転の表現

- キーフレームアニメーションの講義で学習した、3種類の向き(回転)の表現方法のいずれかを使用

- 回転行列による表現

- 変数が多い、変数間の制約がある

$$\begin{pmatrix} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{pmatrix}$$

- オイラー角による表現

- 同じ向きの表現が複数ある、補間に向かない、特異点がある

$$(\theta_1, \theta_2, \theta_3)$$

- 四元数(回転軸・角度)による表現

- 共役解がある、変数間の制約がある

$$(x, y, z, w)$$



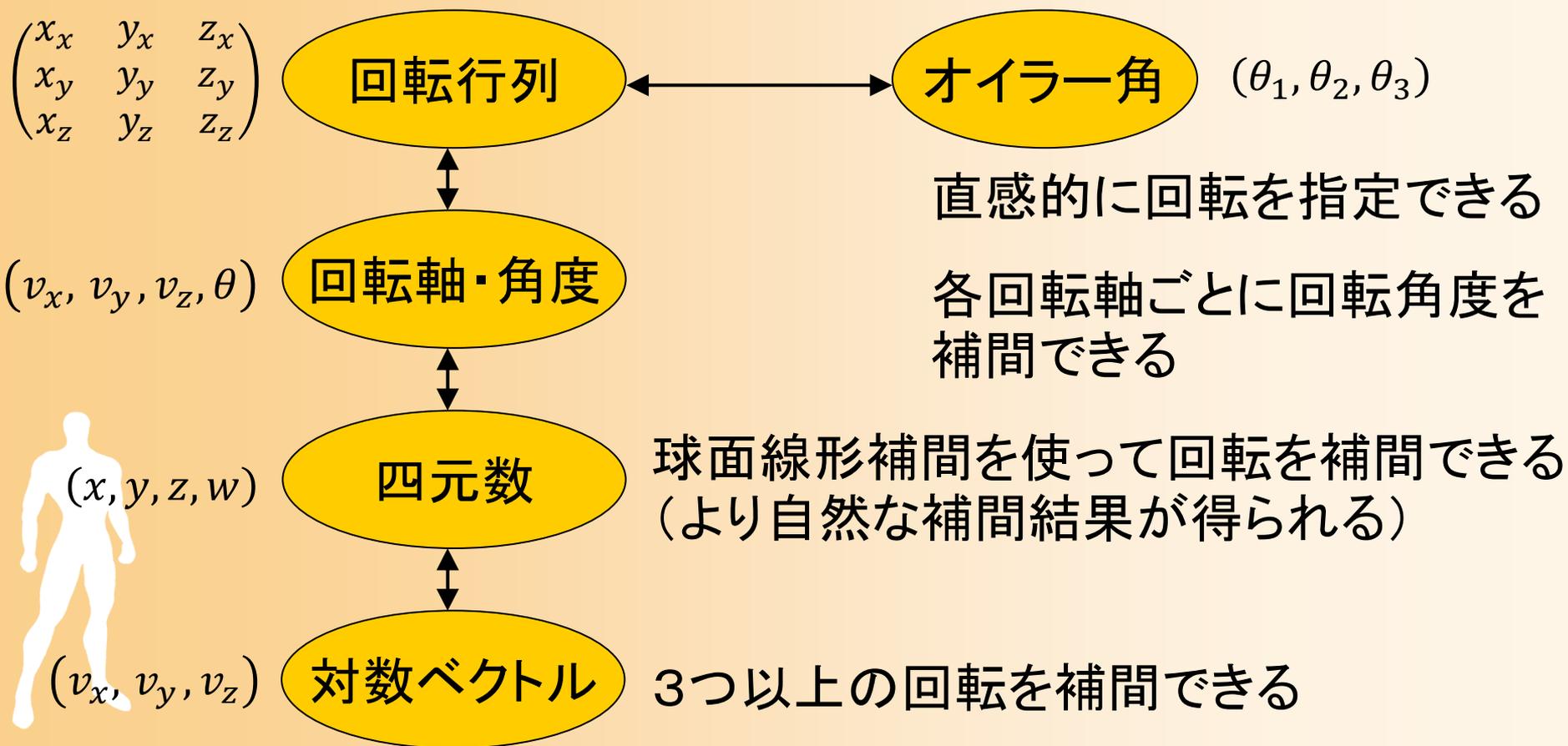
# 関節の回転の表現

- キーフレームアニメーションの講義で学習した、3種類の向き(回転)の表現方法のいずれかを使用
  - 回転行列による表現
  - オイラー角による表現
  - 四元数(回転軸・角度)による表現
    - これらの表現は互いに変換可能なので、必要に応じて変換できる
    - 複数の表現を持たせるようにしても良い
    - レンダリング時には回転行列が必要になる



# 向きの表現方法と相互変換

- 回転行列による表現方法が基本

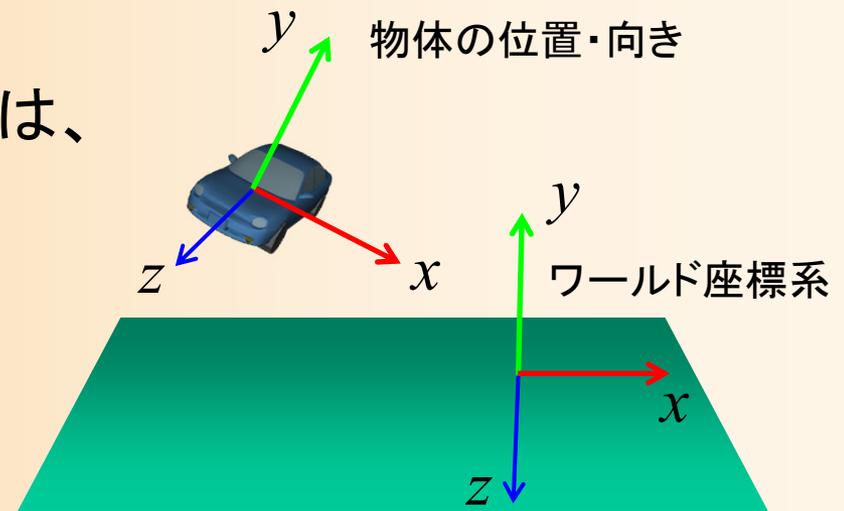


# 回転行列による表現

- 回転行列(3×3行列)による表現

- 各列が、ワールド座標系における、モデル座標系のX軸・Y軸・Z軸の方向ベクトルを表す
- 各列の長さは1で、互いに直交する必要がある
- 向きが一意に決まる
  - 一つの向きの表現方法は、一通りしかない

$$\mathbf{M} = \begin{pmatrix} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{pmatrix}$$



# オイラー角による表現

- 各軸周りの回転角度 ( $\Theta$ ) の組で向きを表現
  - 回転行列の積によって全体の向きを計算

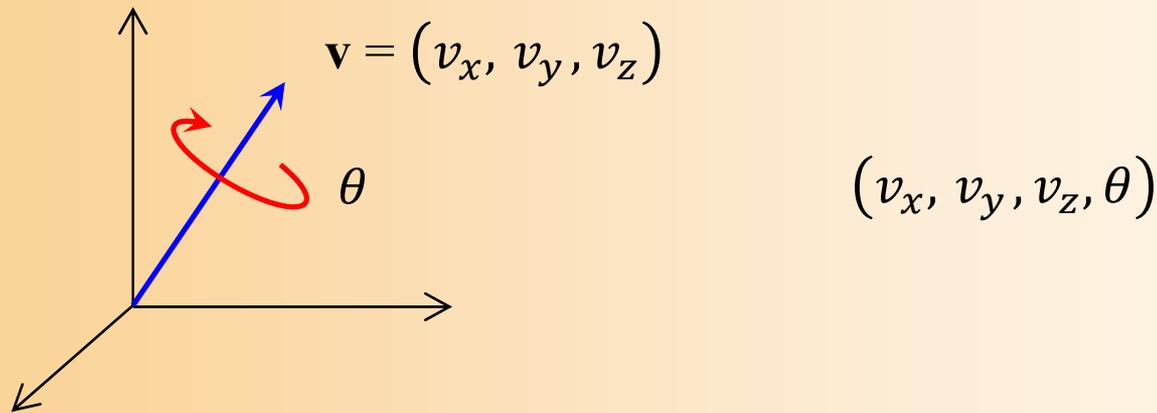
$$\begin{aligned} \mathbf{M} &= R_z(\theta_2) \cdot R_x(\theta_1) \cdot R_y(\theta_0) \\ &= \begin{pmatrix} \cos \theta_2 & -\sin \theta_2 & 0 \\ \sin \theta_2 & \cos \theta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_1 & -\sin \theta_1 \\ 0 & \sin \theta_1 & \cos \theta_1 \end{pmatrix} \begin{pmatrix} \cos \theta_0 & 0 & \sin \theta_0 \\ 0 & 1 & 0 \\ -\sin \theta_0 & 0 & \cos \theta_0 \end{pmatrix} \end{aligned}$$

- ※ 回転行列の適用順序によって向きが変わる
  - 適切な軸と順序をあらかじめ決めておく必要がある
  - 方位角 (y軸周りの回転) → 仰角 (x軸周りの回転) → 回転角 (z軸周りの回転) がよく使われる



# 回転軸と回転角度による表現

- 回転軸と回転角度による向きの表現



- 任意の回転軸を用いることで、一つの回転のみで、どのような向きも表現できる
- 回転軸は長さ1の単位ベクトルとする

$$|\mathbf{v}| = 1$$



# 四元数による表現

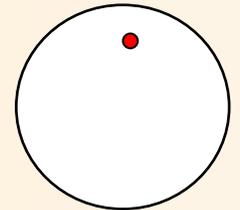
- 単位四元数 (クォータニオン)

- 回転軸と回転角度による表現から変換

$$(v_x, v_y, v_z, \theta)$$

$$(x, y, z, w) = \left( v_x \sin \frac{\theta}{2}, v_y \sin \frac{\theta}{2}, v_z \sin \frac{\theta}{2}, \cos \frac{\theta}{2} \right)$$

- 4次元の球上の点として表せる



- 単位四元数を使うメリット

- 球面線形補間 (2つの向きの間を最短距離で補間する方法) が適用できる



# 関節の自由度の種類

- 3自由度関節

- 人間の大部分の関節は3自由度

- 1自由度関節

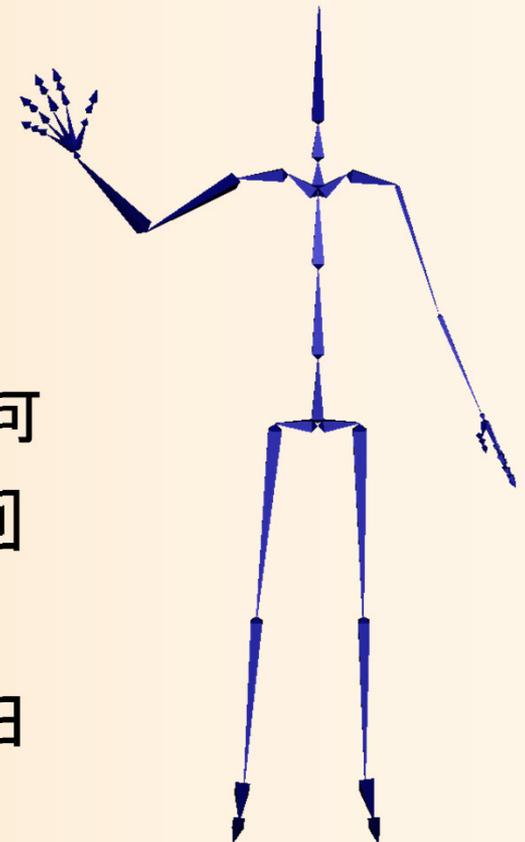
- ひじやひざなどの関節

- 実際は2自由度以上も多少は回転可

- オイラー角表現の場合は1つの回転角度で表現できる

- 回転行列・四元数の場合は1自由度になるように制約を適用する

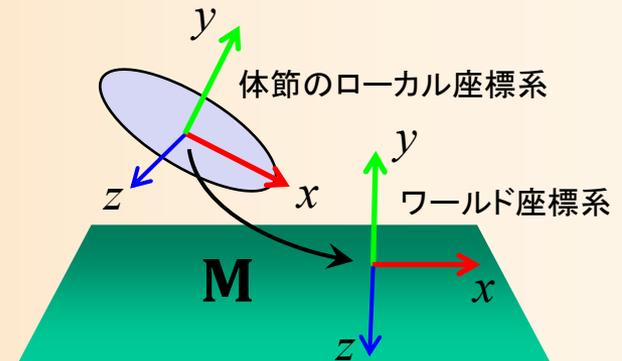
- 全関節を3自由度とするならば省略可



# 位置・向きの変換

- 腰や他の体節の位置・向きの変換
  - 以下のどちらの変換方法でも構わない
  - 位置と向きを別々に変換
    - 位置の変換: 3次元ベクトルによる変換  $(x, y, z)$
    - 向きの変換: 関節の回転と同じ変換
      - 回転行列、オイラー角、四元数など
  - 位置と向きを合わせた  $4 \times 4$  変換行列として変換

$$\begin{pmatrix} R_{11} & R_{12} & R_{13} & x \\ R_{21} & R_{22} & R_{23} & y \\ R_{31} & R_{32} & R_{33} & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



# 姿勢データの表現

- 多関節体の姿勢の表現

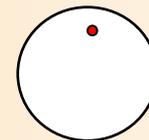
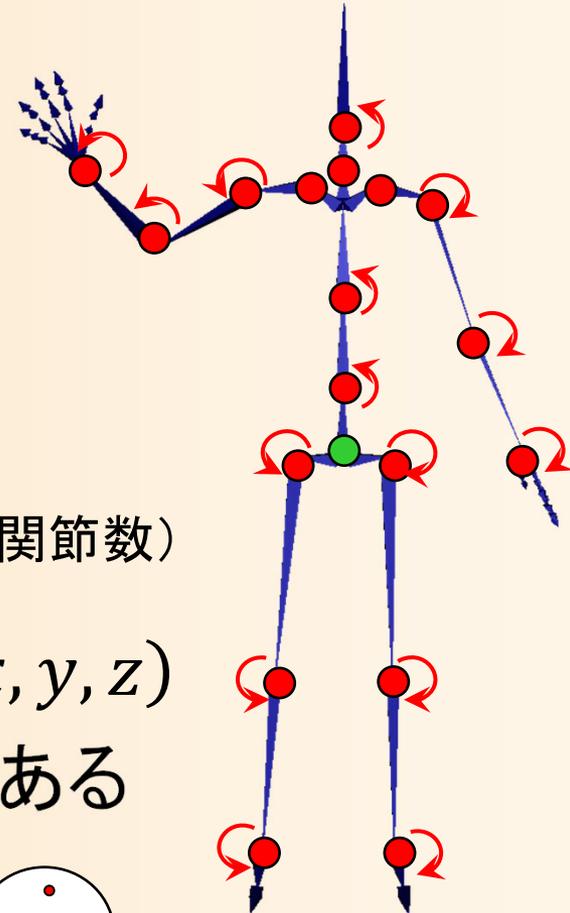
- 各関節の回転(40~自由度)
- 腰の位置・向き(6自由度)

$$\mathbf{p} = \left( \underbrace{\mathbf{v}_r, \mathbf{o}_r}_{\text{腰の位置・向き}}, \underbrace{\mathbf{r}_1, \dots, \mathbf{r}_n}_{\text{全関節の回転}(n \text{ は関節数})} \right)$$

腰の位置・向き 全関節の回転( $n$  は関節数)

- 位置は3次元ベクトルで表現 ( $x, y, z$ )
- 向き・回転は複数の表現方法がある
  - 単位四元数がよく用いられる

$$\mathbf{v}_r, \mathbf{r}_i = (x, y, z, w)$$



# 動作データの表現方法

- 一定間隔動作データ
  - 一定間隔の全フレームの姿勢データを持つ方法
  - 姿勢データの配列により表現可能
    - 30～120 fps 程度の多くの姿勢データが必要
- キーフレーム動作データ
  - キーフレームの姿勢データのみを持ち、中間の姿勢は補間によって求める方法
  - (時刻、姿勢データ)の組の配列により表現可能
    - 各関節ごとに別のキー時刻を使用する方法もある
    - データ量は少なく済む、微妙な姿勢変化は表せない



# 動作データの表現方法

- 一定間隔動作データ
  - 一定間隔の全フレームの姿勢データを持つ方法
  - 姿勢データの配列により表現可能
    - 30～120 fps 程度の多数の姿勢データが必要

$$\mathbf{p} = \left( \underbrace{\mathbf{v}_r, \mathbf{o}_r}_{\text{腰の位置・向き}}, \underbrace{\mathbf{r}_1, \dots, \mathbf{r}_n}_{\text{全関節の回転}} \right)$$

腰の位置・向き    全関節の回転

$$\mathbf{m} = \left( \mathbf{p}_1, \dots, \mathbf{p}_m \right) \quad m \text{ はフレーム数}$$



# 動作データの表現方法

- キーフレーム動作データ
  - キーフレームの姿勢データのみを持ち、中間の姿勢は補間によって求める方法
  - (時刻、姿勢データ)の組の配列により表現可能

$$\mathbf{p} = ( \underbrace{\mathbf{v}_r, \mathbf{o}_r}_{\text{腰の位置・向き}}, \underbrace{\mathbf{r}_1, \dots, \mathbf{r}_n}_{\text{全関節の回転}} )$$

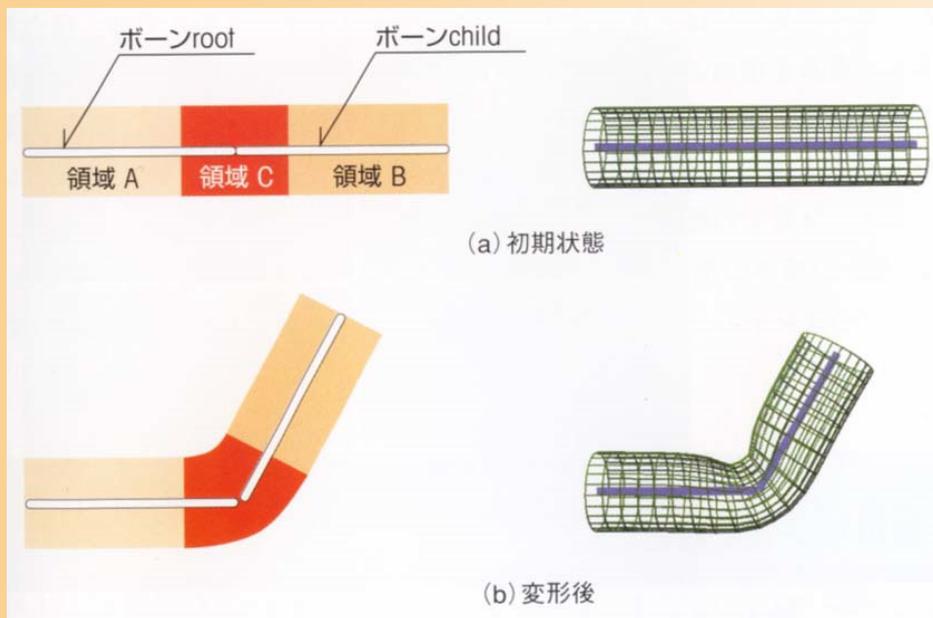
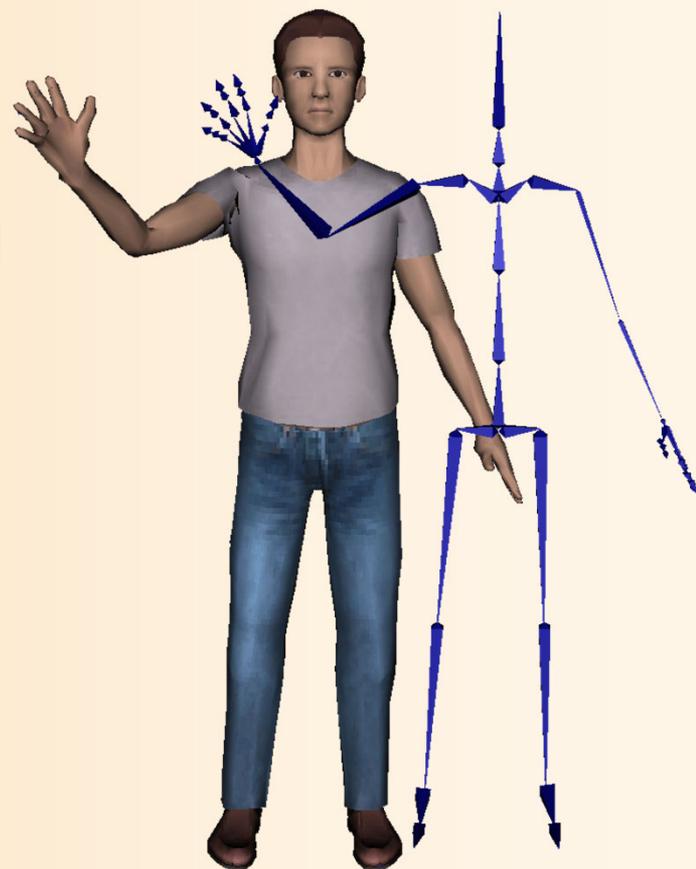
腰の位置・向き 全関節の回転

$$\mathbf{m} = ( (t_1, \mathbf{p}_1), \dots, (t_k, \mathbf{p}_k) ) \quad k \text{ はキーフレーム数}$$



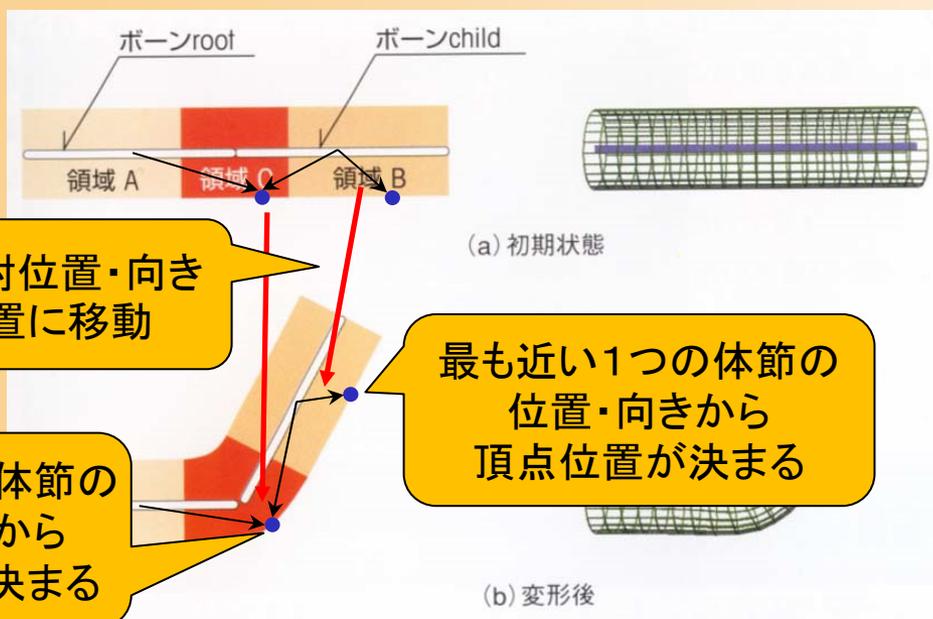
# 形状変形モデル(ワンスキンモデル)

- 人間の全身の形状を1つのポリゴンモデルとして作成
- 骨格モデルの変形に応じてポリゴンモデルの各頂点を移動



# 形状変形モデル(ワンスキンモデル)

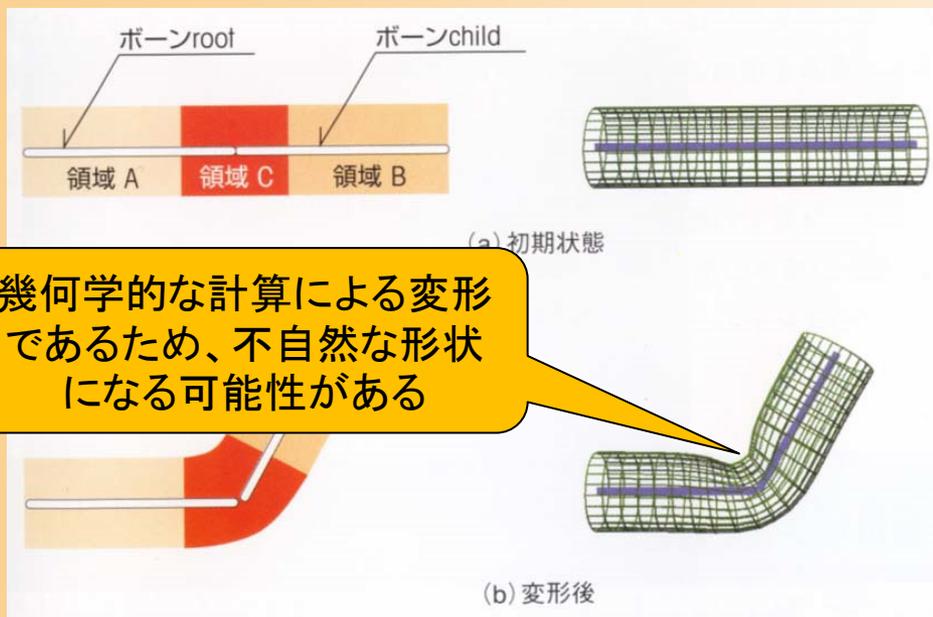
- 人間の全身の形状を1つのポリゴンモデルとして作成
- 骨格モデルの変形に応じてポリゴンモデルの各頂点を移動



「3DCGアニメーション」図4.16

# 形状変形モデル(ワンスキンモデル)

- 人間の全身の形状を1つのポリゴンモデルとして作成
- 骨格モデルの変形に応じてポリゴンモデルの各頂点を移動

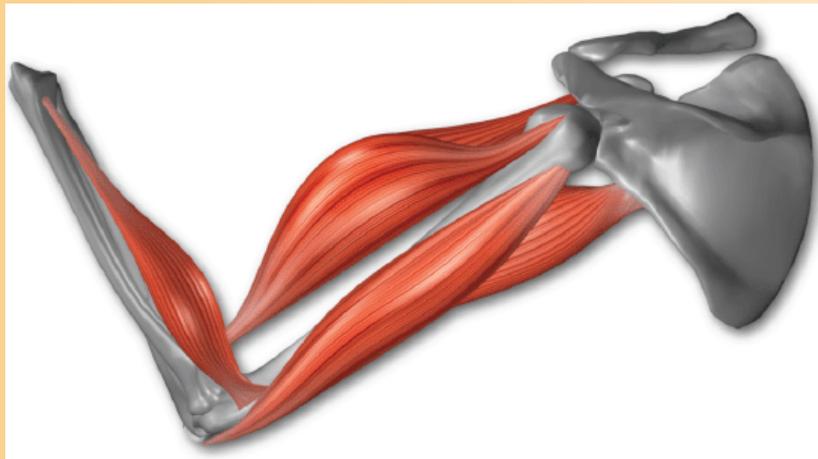


幾何学的な計算による変形であるため、不自然な形状になる可能性がある



# より高度な形状変形モデル(1)

- 筋肉モデルにもとづく変形
    - 人間の筋肉をモデリング
    - 骨格の動きに応じて筋肉を伸縮
    - 筋肉の伸縮に応じて皮膚を変形
- ※ ワンスキンモデルよりもリアルな変形を実現できる

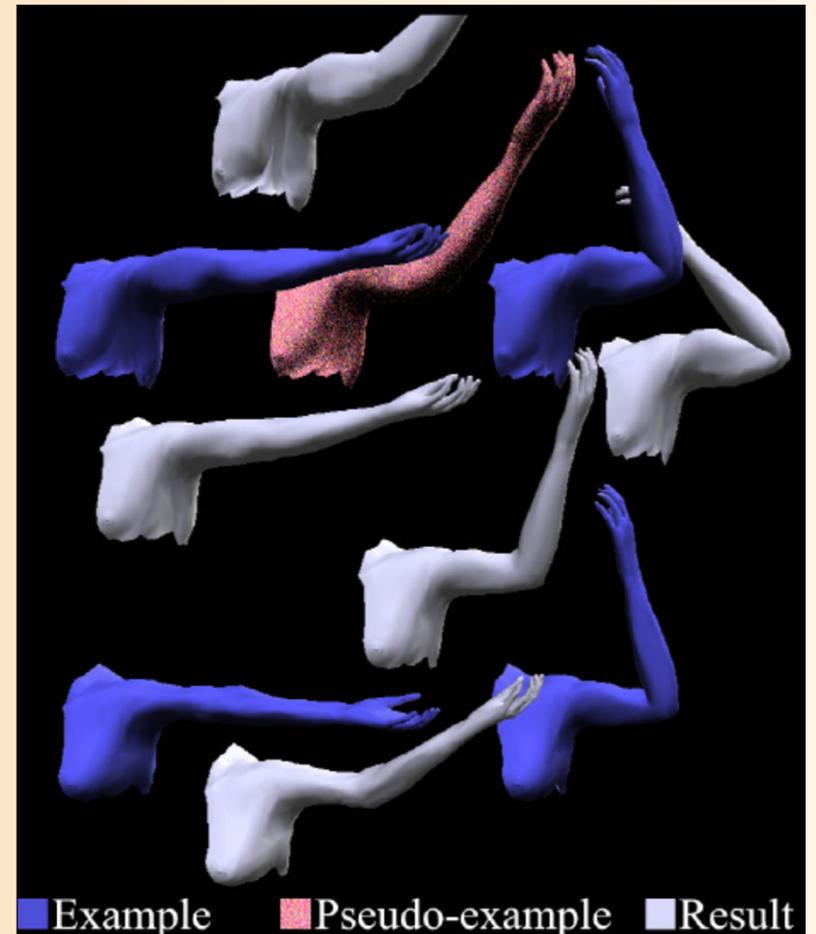


[Softimage XSI]



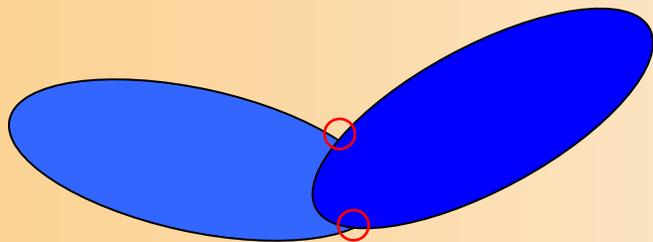
# より高度な形状変形モデル(2)

- 形状補間
  - あらかじめ入力された複数のサンプル形状を適切な重みでブレンドすることで、変形を実現
  - 姿勢に応じて、適切なブレンドの重みを計算する必要がある
  - 顔の表情変形で利用



# 剛体の組み合わせによる形状表現

- 各体節を幾何形状モデルとして表現
  - 関節部分ではポリゴンがめり込むが、全体としてはつながって見える
    - 正確な陰面消去が必要
    - 境界が不自然になる
  - 昔はよく使われていた
  - ロボットなどの表現には有効



# 今日の内容

- 人体モデルの基礎
- 人体モデル(骨格・姿勢・動作)の表現
  - 骨格モデルの表現
  - 姿勢・動作の表現
  - 形状モデルの表現
- 人体モデルの作成
- 動作データの作成
  - キーフレームアニメーション
  - モーションキャプチャ





# 骨格・姿勢・動作の表現

# 骨格モデルの表現(復習)

- 多関節体モデルによる表現

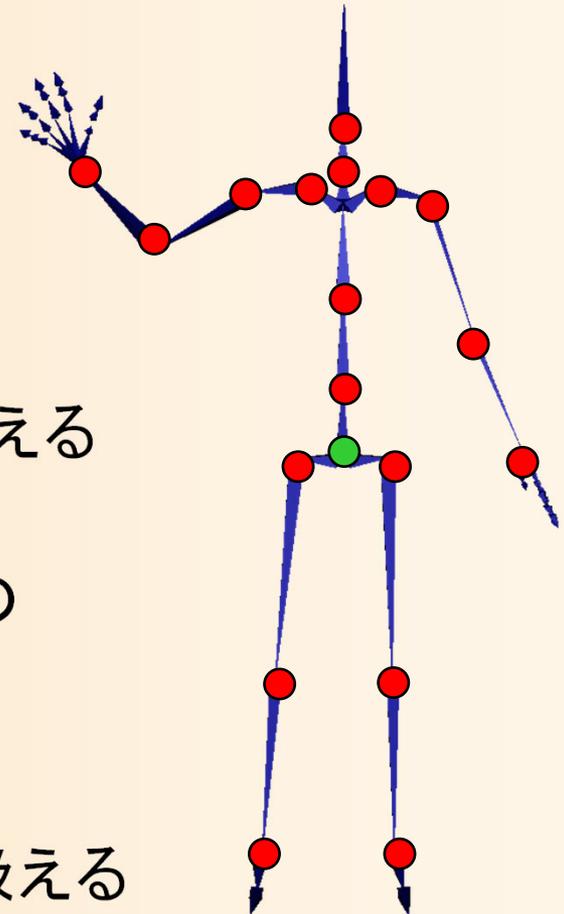
- 複数の体節(部位)が  
関節で接続されたモデル

- 体節

- 多関節体の各部位、剛体として扱える
- 複数の関節が接続されており、  
体節の長さや体節内での各関節の  
接続位置は固定

- 関節

- 2つの体節の間を接続、点として扱える
- 関節の回転により姿勢が変化する



# 骨格モデルの表現方法

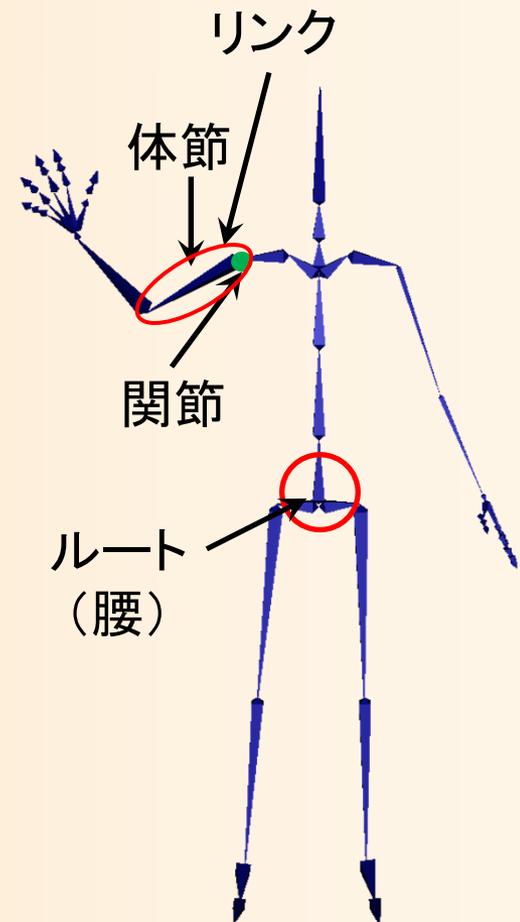
- 多関節体モデルの表現

- 体節・関節の集合により表現

- **ルート体節** (通常は腰) から複数の**末端体節** (手・足・頭) に向かって、体節・関節を順番に接続したツリー構造により表現

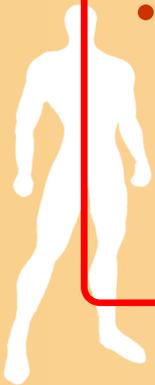
- **体節 + 関節 = リンク**

- 1つの体節と1つの関節 (ルート側の関節) をまとめて、リンクとして扱う方法もある
  - **ボーン** と呼ばれることもある
  - リンクのことを関節や体節と呼ぶこともあるので要注意



# 骨格モデルの表現方法

- 具体的にどのようなデータ構造で表現するかは、いくつかの方法がある
  - 以降の説明では、以下の3通りの表現方法を段階的に説明
- 表現方法1
  - 骨格と姿勢をまとめる、体節と関節をまとめる
- 表現方法2
  - 骨格と姿勢を分ける、体節と関節をまとめる
- 表現方法3
  - 骨格と姿勢を分ける、体節と関節を分ける
  - 本授業では、この表現方法を推奨
    - 一般的には、上の表現方法2も広く使われている



# 骨格モデルの表現方法の違い

- 骨格・姿勢の情報をまとめるか、分けるか
- 骨格情報の中で、体節と関節をまとめるか、分けるか
- 姿勢情報での、関節の回転の表現方法
- 骨格情報での、末端位置の情報の持たせ方
  - 体節に付属情報として持たせる or 仮想的な関節として扱う
- 関節可動範囲、筋力等の追加情報の有無



# 骨格モデルの表現方法1(1)

- 最も単純な表現方法

- 各リンクを表すデータ構造を定義

- リンク = 体節 + 親側の関節

- リンクに含まれる情報

- 骨格情報 (固定情報)

- 隣接リンク

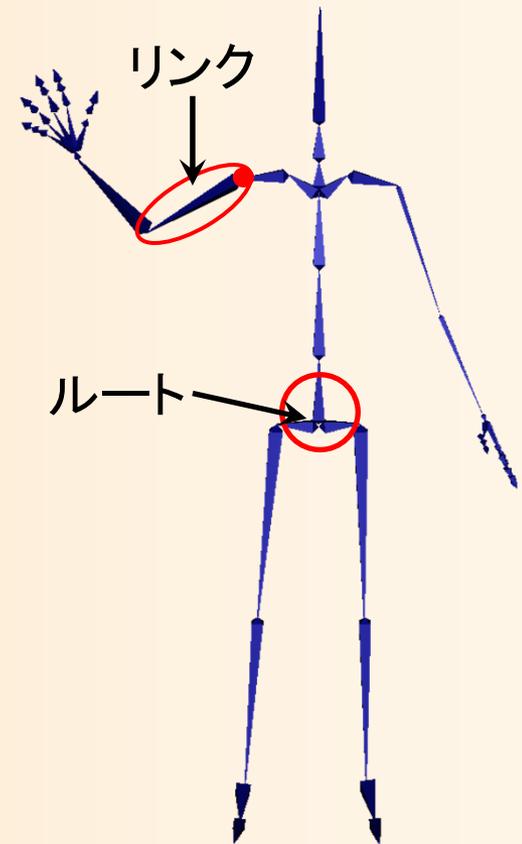
- 隣接リンクとの相対位置

- (スキニング情報)

- 姿勢情報 (動作によって変化)

- 親リンクに対する相対回転

- ルートの位置・向き (ルートリンクのみ)



# 骨格モデルの表現方法1(2)

- 最も単純な表現方法

- 各リンクを表すデータ構造を定義

- リンクに含まれる情報

- 骨格情報（固定情報）

- 隣接リンク

- » 親リンク（ルート側、常に一つ）

- » 子リンク（末端側、複数の場合がある）

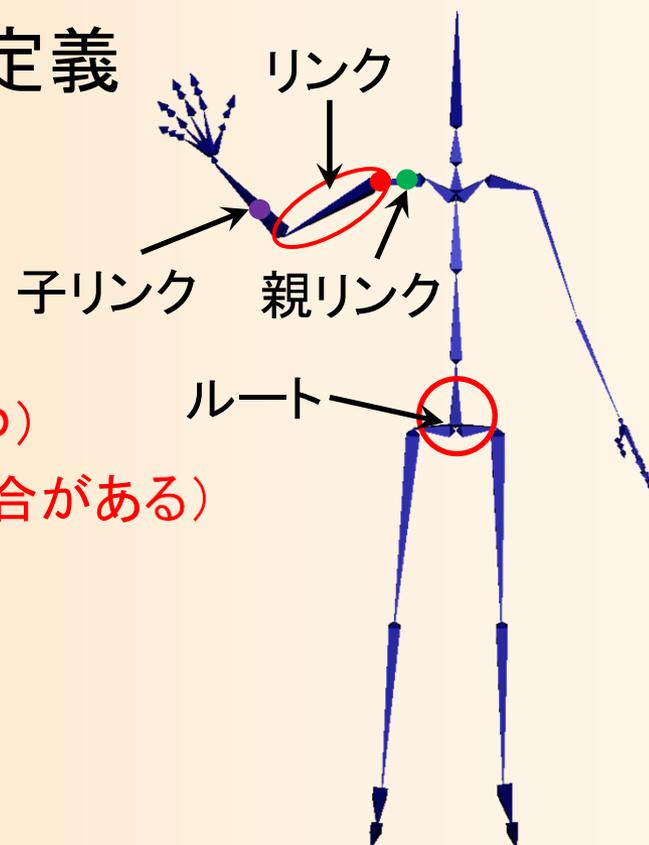
- 隣接リンクとの相対位置

- （スキニング情報）

- 姿勢情報（動作によって変化）

- 親リンクに対する相対回転

- ルートの位置・向き（ルートリンクのみ）



# 骨格モデルの表現方法1(3)

- 最も単純な表現方法

- 各リンクを表すデータ構造を定義
- リンクに含まれる情報

- 骨格情報 (固定情報)

- 隣接リンク

- » 親リンク (ルート側、常に一つ)

- » 子リンク (末端側、複数の場合がある)

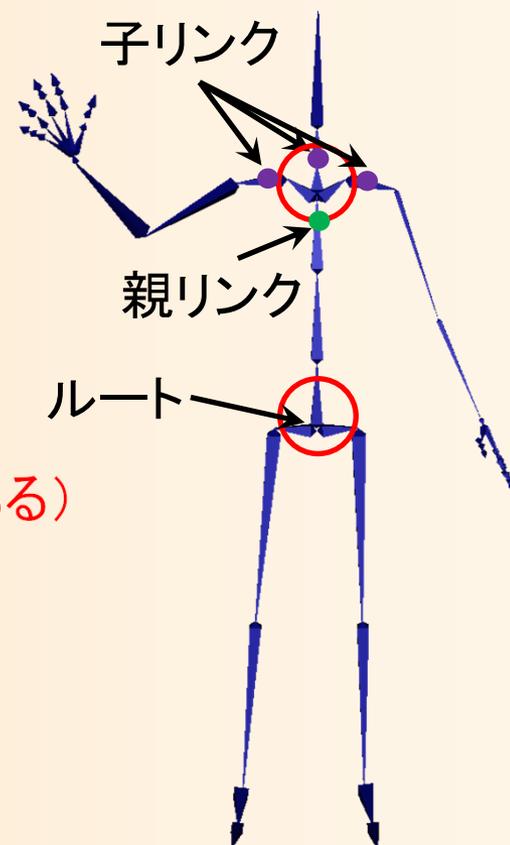
- 隣接リンクとの相対位置

- (スキニング情報)

- 姿勢情報 (動作によって変化)

- 親リンクに対する相対回転

- ルートの位置・向き (ルートリンクのみ)



# 骨格モデルの表現方法1(4)

- 最も単純な表現方法

- 各リンクを表すデータ構造を定義

- リンクに含まれる情報

- 骨格情報（固定情報）

- 隣接リンク

- 隣接リンクとの相対位置

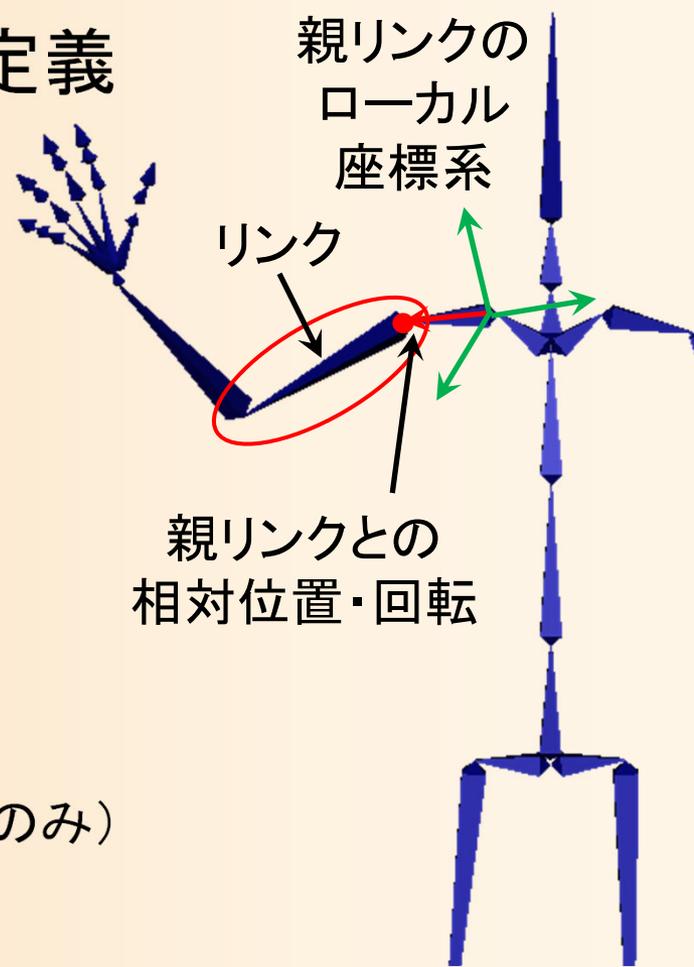
- » 親リンクの座標系での位置

- (スキニング情報)

- 姿勢情報（動作によって変化）

- 親リンクに対する相対回転

- ルートの位置・向き(ルートリンクのみ)



# 骨格モデルの表現方法1(5)

- 最も単純な表現方法

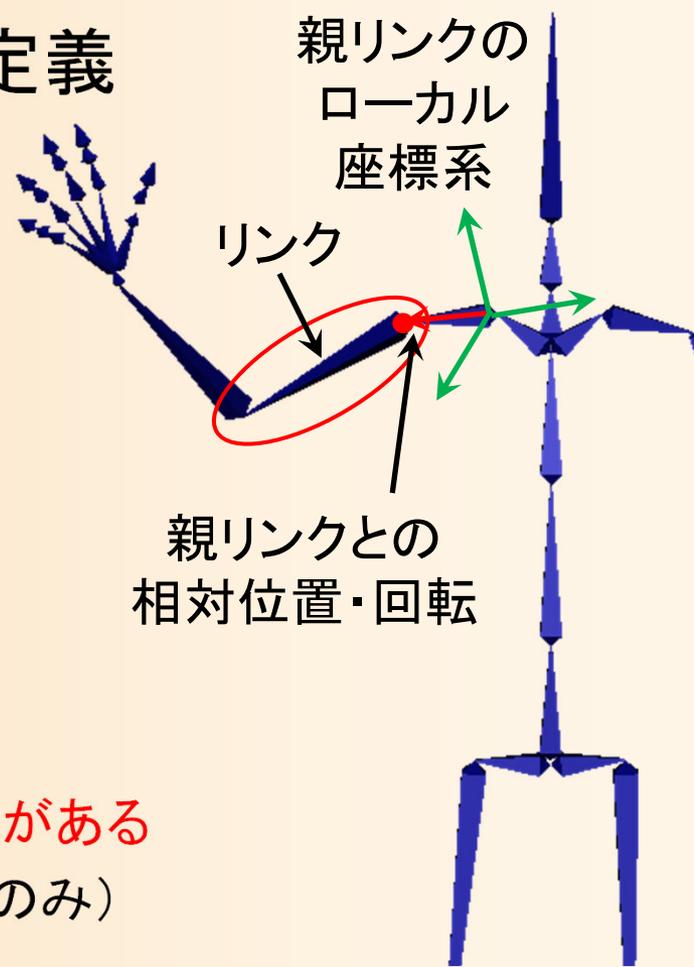
- 各リンクを表すデータ構造を定義
- リンクに含まれる情報

- 骨格情報（固定情報）

- 隣接リンク
- 隣接リンクとの相対位置
- (回転自由度・軸の情報)
- (スキニング情報)

- 姿勢情報（動作によって変化）

- 親リンクに対する相対回転
  - » 回転の表現には複数の方法がある
- ルートの位置・向き（ルートリンクのみ）



# 骨格モデルの表現方法1(6)

- 最も単純な表現方法

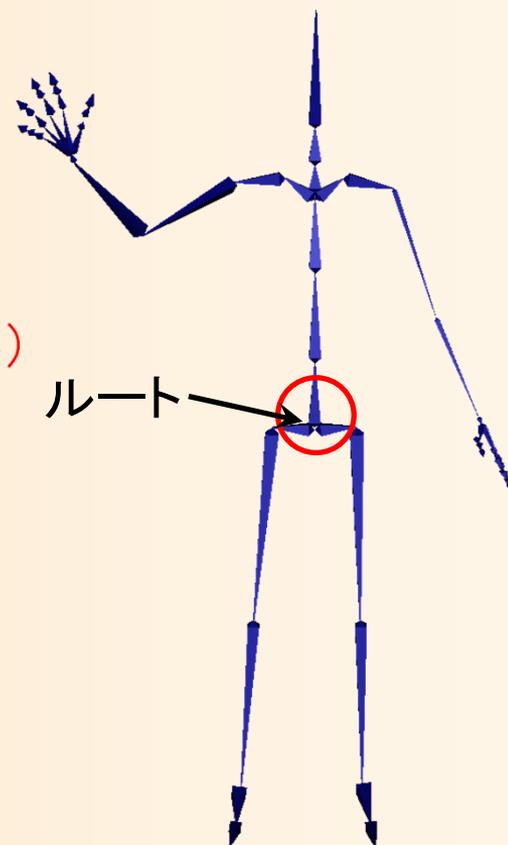
- 各リンクを表すデータ構造を定義
- リンクに含まれる情報

- 骨格情報 (固定情報)

- 隣接リンク (ルートリンクは子リンクのみ)
- 隣接リンクとの相対位置  
(ルートリンクは持たない)
- (スキニング情報)

- 姿勢情報 (動作によって変化)

- 親リンクに対する相対回転  
(ルートリンクは持たない)
- ルートの位置・向き (ルートリンクのみ)



# 骨格モデルの表現方法1(7)

// 多関節体の各リンクを表す構造体

```
struct Link
```

```
{
```

```
    // 親リンク(常に一つ、ルートはなし)
```

```
    Link *      parent;
```

```
    // 子リンク(複数になることがある)
```

```
    vector< Link * >  children;
```

```
    // 親リンクからの接続位置(ローカル座標系)
```

```
    Point3f     offset;
```

```
    // ルートリンクかどうかのフラグ
```

```
    bool        is_root;
```

```
    // ルートリンクの位置・向き(ルートのみが持つ)
```

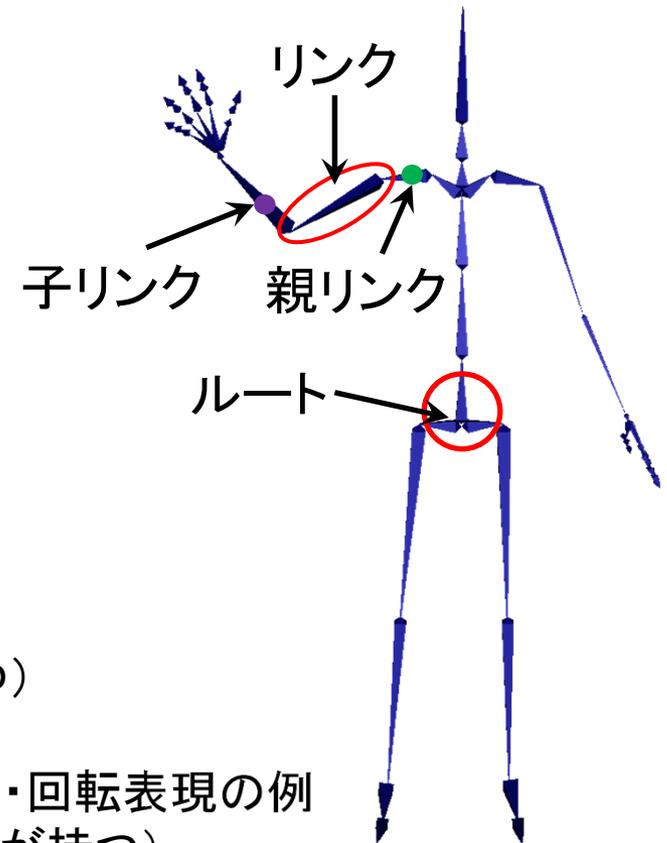
```
    Point3f     root_pos;
```

```
    Matrix3f    root_ori; // 回転行列による向き・回転表現の例
```

```
    // 親リンクに対する相対的な回転(ルート以外が持つ)
```

```
    Matrix3f    link_rot; // 回転行列による向き・回転表現の例
```

```
};
```



# 骨格モデルの表現方法1(8)

- 多関節体の骨格・姿勢の表現
  - リンクの集合(ツリー構造)により表現
    - どちらの方法(あるいは組み合わせ)でも可能

```
// 多関節体の骨格・姿勢を表す構造体
struct SkeletonAndPosture
{
    // 全リンクの配列(0番目の要素をルートリンクとする)
    vector< Link * > links;
};
```

```
// 多関節体の骨格・姿勢を表す構造体
struct SkeletonAndPosture
{
    // ルートリンク(リンクのツリー構造により表現)
    Link *          root_link;
};
```



# 骨格モデルの表現方法2(1)

- 骨格情報と姿勢情報を分ける方法
  - 骨格情報を表すデータ構造(固定の情報)
  - 姿勢情報を表すデータ構造
    - 腰の位置・向き(6自由度)
    - 各関節の回転(n自由度)
  - こちらの方法の方が使いやすい
    - 同一骨格の複数の姿勢データの扱いが容易になる
      - あるキャラクタの多数の姿勢データを扱う場合
      - 骨格が同じキャラクタが複数登場する場合も、骨格データは1つを共有すれば済む



# 骨格モデルの表現方法2(2)

```
// 多関節体の各リンクを表す構造体(骨格情報のみ)
```

```
struct Link
```

```
{
```

```
    // 親リンク(常に一つ、ルートはなし)
```

```
    Link *          parent;
```

```
    // 子リンク(複数になることがある)
```

```
    vector< Link * > children;
```

```
    // 親リンクからの接続位置(ローカル座標系)
```

```
    Point3f        offset;
```

```
};
```

```
// 多関節体の骨格を表す構造体
```

```
struct Skeleton
```

```
{
```

```
    // 全リンクの配列(0番目の要素をルートリンクとする)
```

```
    vector< Link * > links;
```

```
};
```



# 骨格モデルの表現方法2(3)

- 骨格情報と姿勢情報を分ける方法
  - 姿勢情報から骨格情報を参照
  - 全リンクの回転は $3 \times 3$ 回転行列の配列で表す

// 多関節体の姿勢を表す構造体

```
struct Posture
```

```
{
```

```
    Skeleton * body;
```

```
    Point3f    root_pos;        // ルートの位置
```

```
    Matrix3f   root_ori;       // ルートの向き(回転行列表現)
```

```
    Matrix3f * link_rotations; // 各リンクの相対回転(回転行列表現)
```

```
                                // [リンク番号] リンク数分の配列
```

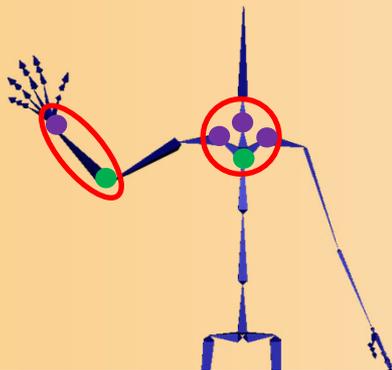
```
};
```



# 骨格モデルの表現方法3(1)

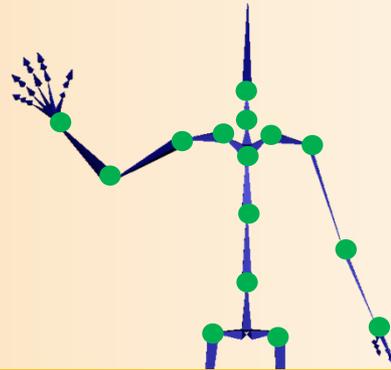
- 骨格情報の中で、関節・体節を分ける

関節・体節をまとめる



体節に、一つの親関節と複数の子関節の情報を含める  
(体節を順番に辿るとき、方向により異なる処理が必要)

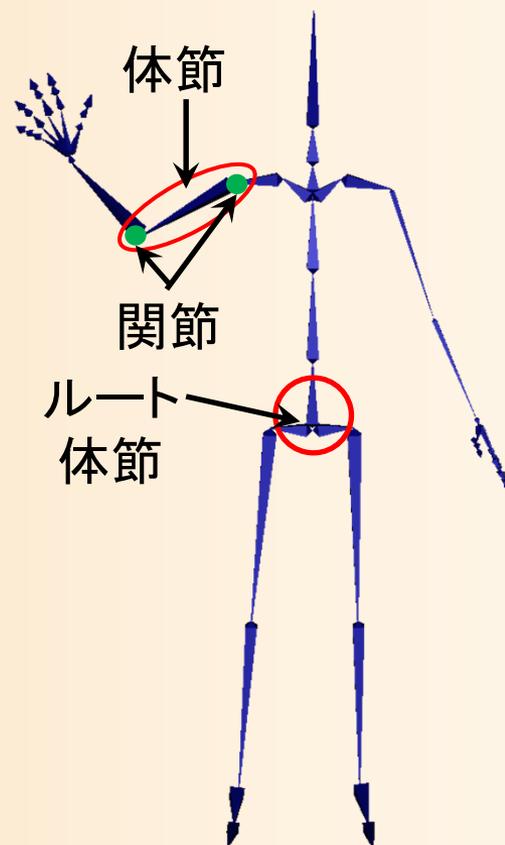
関節・体節を分ける



体節に、複数の関節へのリンクを含める  
(体節に順番に辿るとき、どの方向も同じ処理が可能)

# 骨格モデルの表現方法3(2)

- 骨格情報の中で、体節と関節のデータ構造を分ける
  - 体節に含まれる情報
    - 接続される複数の関節
    - 各関節の接続位置
      - 体節のローカル座標系での位置
  - 関節に含まれる情報
    - 接続される2つの体節
      - ルート側・末端側の体節



# 骨格モデルの表現方法3(3)

- 骨格情報の中で、体節と関節のデータ構造を分ける

```
// 多関節体の体節を表す構造体
struct Segment
{
    // 接続関節
    vector< Joint * >    joints;
    // 各関節の接続位置(体節のローカル座標系)
    vector< Point3f >    joint_positions;
};
```

```
// 多関節体の関節を表す構造体
struct Joint
{
    // 接続体節
    Segment *           segments[ 2 ];
};
```

```
// 多関節体の骨格を表す構造体
struct Skeleton
{
    // 体節・関節の配列
    vector< Segment * > segments;
    vector< Joint * >    joints;
};
```

# 骨格モデルの表現方法3(4)

- 骨格情報と姿勢情報のデータ構造を分ける

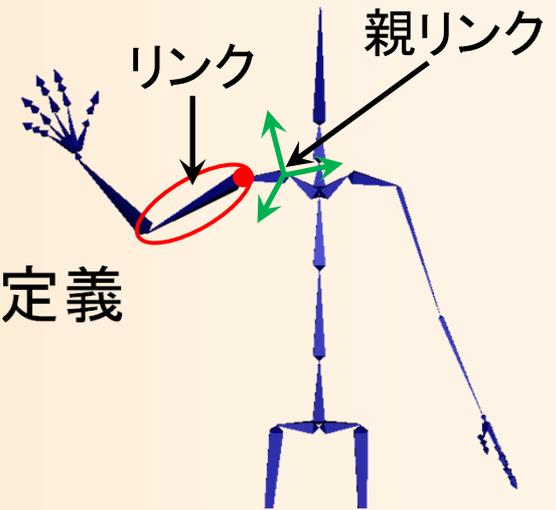
```
// 多関節体の姿勢を表す構造体
struct Posture
{
    Skeleton * body;
    Point3f    root_pos;        // ルートの位置
    Matrix3f   root_ori;       // ルートの向き(回転行列表現)
    Matrix3f * joint_rotations; // 各関節の回転(回転行列表現)
                                   // [関節番号] 関節数分の配列
};
```



# 骨格モデルの表現方法の比較

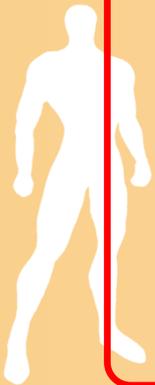
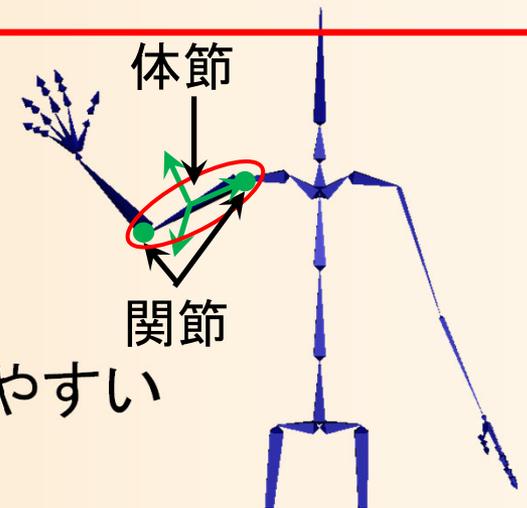
- 表現方法2: 体節と関節をまとめる

- データ構造は単純
- 親リンクの座標系で子リンクの位置を定義
- 運動学・動力学計算の処理の記述はやや面倒になる
- 一般的には、こちらの表現が多い



- 表現方法3: 体節と関節を分ける

- データ構造はやや複雑
- 体節の座標系で関節の位置を定義
- 運動学・動力学計算の処理は記述しやすい
- 本授業では、こちらの表現を採用



# 骨格モデルの詳細情報(1)

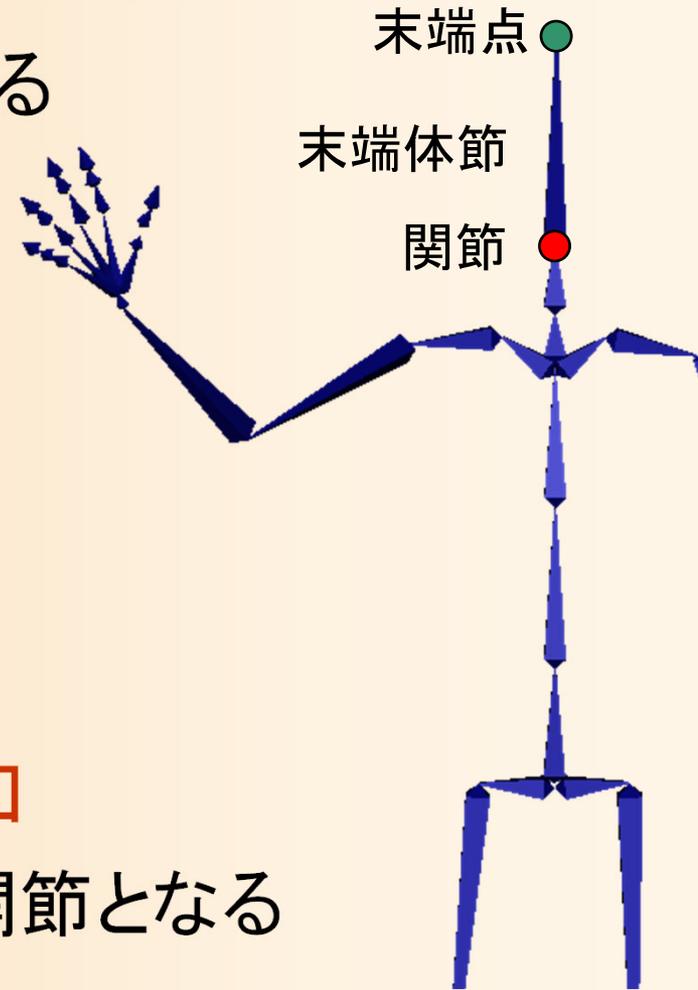
- 関節の回転やルートの向きの表現方法
  - 回転行列( $3 \times 3$ 行列)以外の表現方法もある
    - 四元数の場合、関節数分の配列(回転行列と同様)
    - オイラー角の場合、全関節の自由度の総和分の配列
      - 自由度が異なる関節を混在させる場合は、やや複雑になる
- 関節回転の可動範囲の定義(詳細は省略)
  - 可動範囲を定義して姿勢を制限する方法もある
- 筋力モデル(詳細は省略)
  - 動作解析・生成のため筋力を考慮する方法もある



# 骨格モデルの詳細情報(2)

- 末端点の情報の表現方法
  - 末端点も必要になることがある
- 方法1: 体節に情報を追加

```
// 多関節体の体節を表す構造体
struct Segment
{
    ...
    // 体節の末端位置
    bool        has_site;
    Point3f     site_position;
};
```



- 方法2: 仮想的な関節を追加
  - 1つの体節のみに接続する関節となる



# 骨格・姿勢の表現方法のまとめ

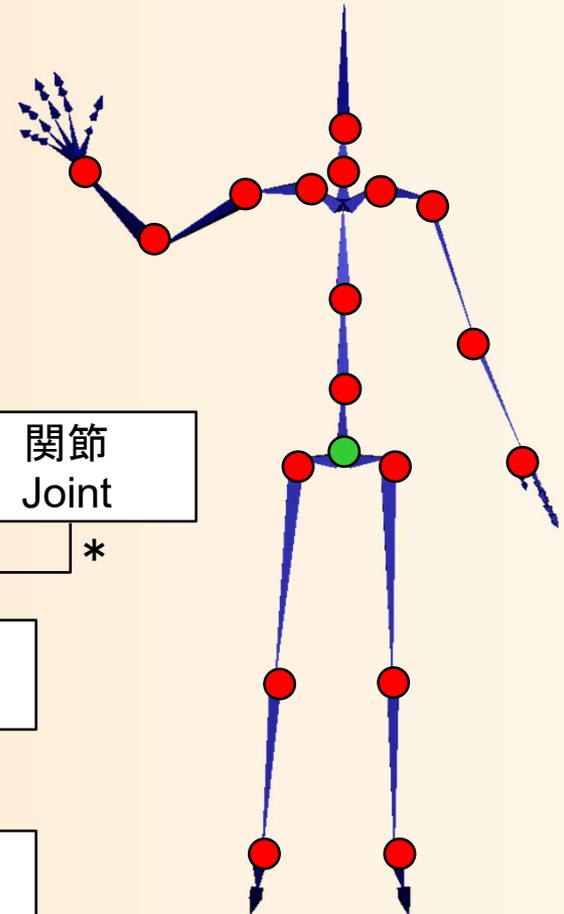
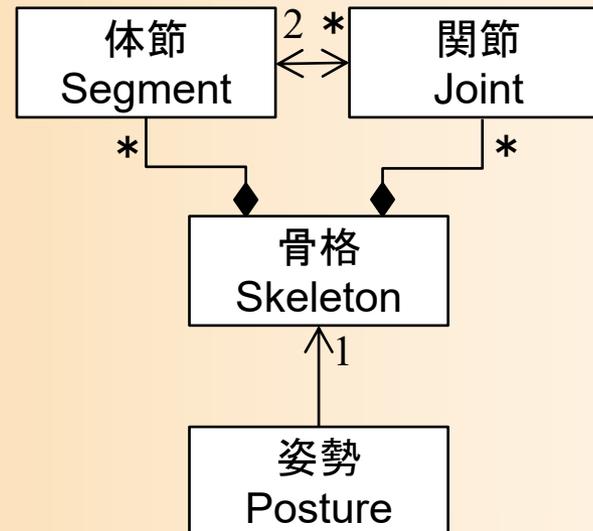
- 骨格情報と姿勢情報を分ける
- 骨格情報の中で、体節と関節を分ける

// 多関節体の体節を表す構造体  
struct Segment

// 多関節体の関節を表す構造体  
struct Joint

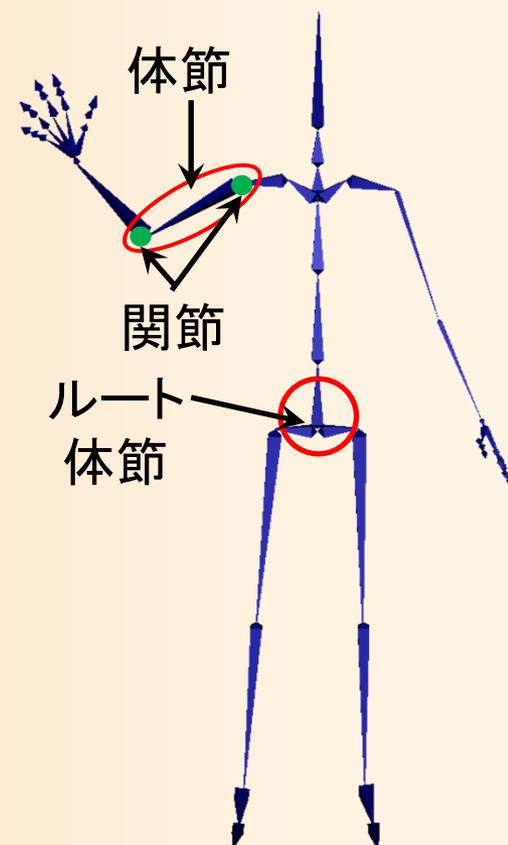
// 多関節体の骨格を表す構造体  
struct Skeleton

// 多関節体の姿勢を表す構造体  
struct Posture



# 骨格の表現方法のまとめ

- 骨格情報の中で、体節と関節のデータ構造を分ける
  - 体節に含まれる情報
    - 接続される複数の関節
    - 各関節の接続位置
      - 体節のローカル座標系での位置
  - 関節に含まれる情報
    - 接続される2つの体節
      - ルート側・末端側の体節



# 骨格モデルの表現方法

- 骨格情報の中で、体節と関節のデータ構造を分ける

```
// 多関節体の体節を表す構造体
struct Segment
{
    // 接続関節
    vector< Joint * >    joints;
    // 各関節の接続位置(体節のローカル座標系)
    vector< Point3f >    joint_positions;
};
```

```
// 多関節体の関節を表す構造体
struct Joint
{
    // 接続体節
    Segment *            segments[ 2 ];
};
```

```
// 多関節体の骨格を表す構造体
struct Skeleton
{
    // 体節・関節の配列
    vector< Segment * > segments;
    vector< Joint * >    joints;
};
```

# 姿勢の表現方法のまとめ

- 多関節体の姿勢の表現

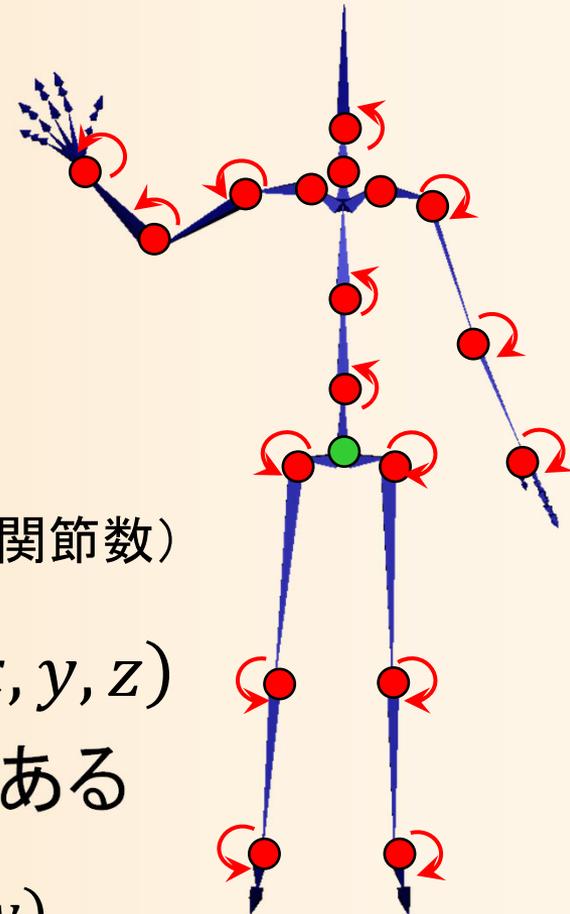
- 各関節の回転 (40~自由度)
- 腰の位置・向き (6自由度)

$$\mathbf{p} = \left( \underbrace{\mathbf{v}_r, \mathbf{o}_r}_{\text{腰の位置・向き}}, \underbrace{\mathbf{r}_1, \dots, \mathbf{r}_n}_{\text{全関節の回転}(n \text{ は関節数})} \right)$$

腰の位置・向き 全関節の回転 ( $n$  は関節数)

- 位置は3次元ベクトルで表現 ( $x, y, z$ )
- 向き・回転は複数の表現方法がある

$$\begin{pmatrix} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{pmatrix} (\theta_1, \theta_2, \theta_3) (x, y, z, w)$$



# 姿勢データの表現方法

- 骨格情報を参照
  - 関節数は骨格情報により決まる
- 向き・回転の表現には回転行列を使用

```
// 多関節体の姿勢を表す構造体
struct Posture
{
    Skeleton * body;
    Point3f    root_pos;        // ルートの位置
    Matrix3f   root_ori;       // ルートの向き(回転行列表現)
    Matrix3f * joint_rotations; // 各関節の回転(回転行列表現)
                                     // [関節番号] 関節数分の配列
};
```



# 形状モデルの表現

- キャラクタの形状変形モデルに必要な情報

- 骨格構造の情報
- 全身の幾何形状データ
- 骨格構造の各体節から幾何形状の各頂点への重み

- $m \times n$  の行列データ  
(体節数 $m$ 、頂点数 $n$ )



- 通常はアニメーションソフトを使って作成したデータを利用

# 形状モデルの表現方法

- 形状変形のための重みの情報は、行列(2次元配列)により表現できる

```
// ワンスキンモデルを表す構造体
struct OneSkinModel
{
    // 骨格情報
    Skeleton * skeleton;
    // 幾何形状情報
    Obj * skin_shape;
    // 変形のためのウェイト情報
    float ** weights; // [頂点番号][体節番号] の2次元配列
    // 初期姿勢での各体節の変換行列の逆行列
    Matrix4f * init_seg_frames; // [体節番号]
};
```

初期状態の姿勢  
から計算



# 動作データの表現方法(復習)

- 一定間隔動作データ
  - 一定間隔の全フレームの姿勢データを持つ方法
  - 姿勢データの配列により表現可能
    - 30～120 fps 程度の多くの姿勢データが必要
- キーフレーム動作データ
  - キーフレームの姿勢データのみを持ち、中間の姿勢は補間によって求める方法
  - (時刻、姿勢データ)の組の配列により表現可能
    - 各関節ごとに別のキー時刻を使用する方法もある
    - データ量は少なく済む、微妙な姿勢変化は表せない



# 動作データの表現方法(復習)

- 一定間隔動作データ
  - 一定間隔の全フレームの姿勢データを持つ方法
  - 姿勢データの配列により表現可能
    - 30～120 fps 程度の多数の姿勢データが必要

$$\mathbf{p} = \left( \underbrace{\mathbf{v}_r, \mathbf{o}_r}_{\text{腰の位置・向き}}, \underbrace{\mathbf{r}_1, \dots, \mathbf{r}_n}_{\text{全関節の回転}} \right)$$

腰の位置・向き      全関節の回転

$$\mathbf{m} = \left( \mathbf{p}_1, \dots, \mathbf{p}_m \right) \quad m \text{ はフレーム数}$$



# 動作データの表現方法(復習)

- キーフレーム動作データ
  - キーフレームの姿勢データのみを持ち、中間の姿勢は補間によって求める方法
  - (時刻、姿勢データ)の組の配列により表現可能

$$\mathbf{p} = ( \underbrace{\mathbf{v}_r, \mathbf{o}_r}_{\text{腰の位置・向き}}, \underbrace{\mathbf{r}_1, \dots, \mathbf{r}_n}_{\text{全関節の回転}} )$$

腰の位置・向き 全関節の回転

$$\mathbf{m} = ( (t_1, \mathbf{p}_1), \dots, (t_k, \mathbf{p}_k) ) \quad k \text{ はキーフレーム数}$$



# 動作データの表現方法

- 一定間隔データとキーフレームデータ

```
// 動作データ(一定間隔)
```

```
struct Motion
```

```
{
```

```
    int        num_frames; // 全フレーム数
```

```
    float      interval;   // フレーム間の時間間隔
```

```
    Posture *  frame_poses; // 姿勢配列 [フレーム番号]
```

```
};
```

```
// 動作データ(キーフレーム)
```

```
struct KeyframeMotion
```

```
{
```

```
    int        num_keyframes; // キーフレーム数
```

```
    float *    key_times;     // 各キー時刻の配列 [キーフレーム番号]
```

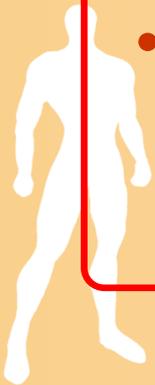
```
    Posture *  key_poses;     // 各キー姿勢の配列 [キーフレーム番号]
```

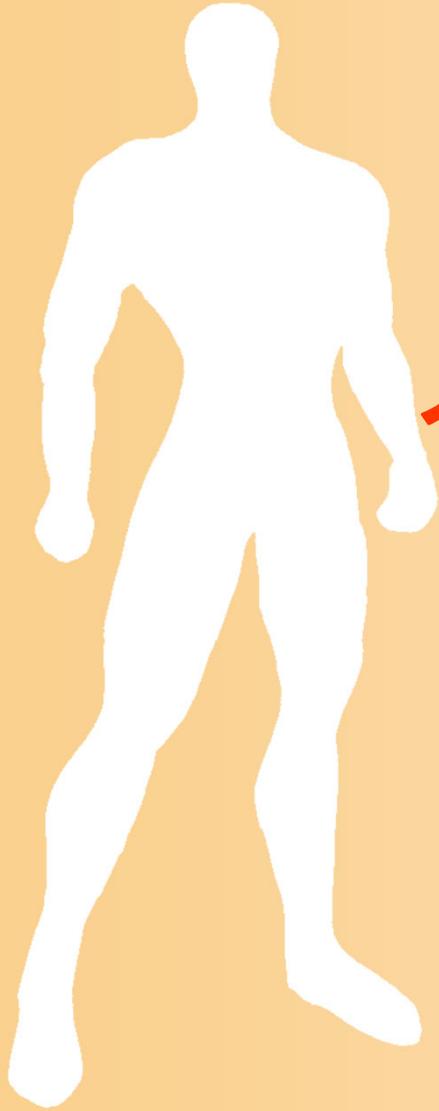
```
};
```



# 今日の内容

- 人体モデルの基礎
- 人体モデル(骨格・姿勢・動作)の表現
  - 骨格モデルの表現
  - 姿勢・動作の表現
  - 形状モデルの表現
- 人体モデルの作成
- 動作データの作成
  - キーフレームアニメーション
  - モーションキャプチャ

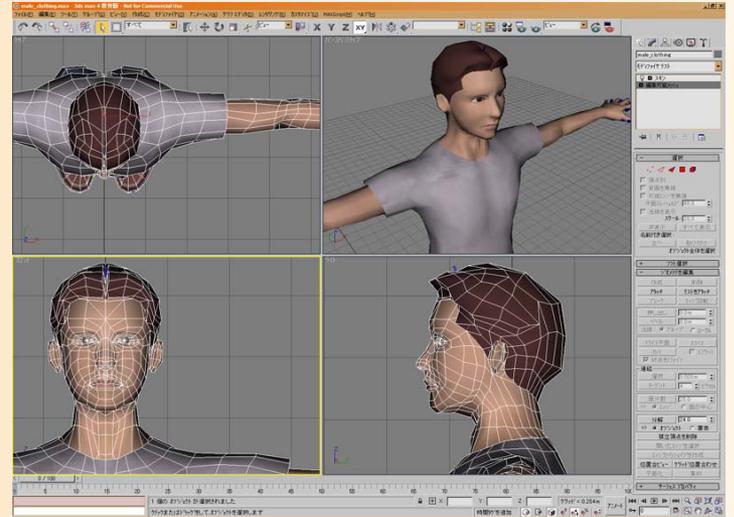




# 人体モデルの作成

# 人体モデルの作成方法

- 人体モデル  
(=骨格+形状モデル)  
の作成方法
- 市販のアニメーション  
制作ソフトウェアを使用  
してデザイナーが作成
- 自分のプログラムで使用するときには、アニメーション制作ソフトウェアから出力したファイルを読み込んで使用





# 人体モデルの作成手順

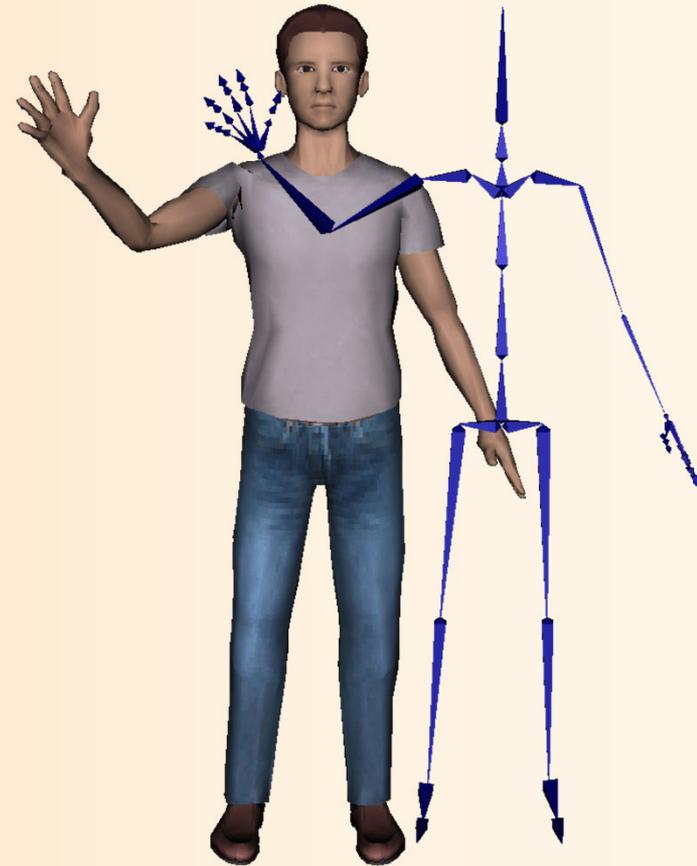
- 標準的な作成手順

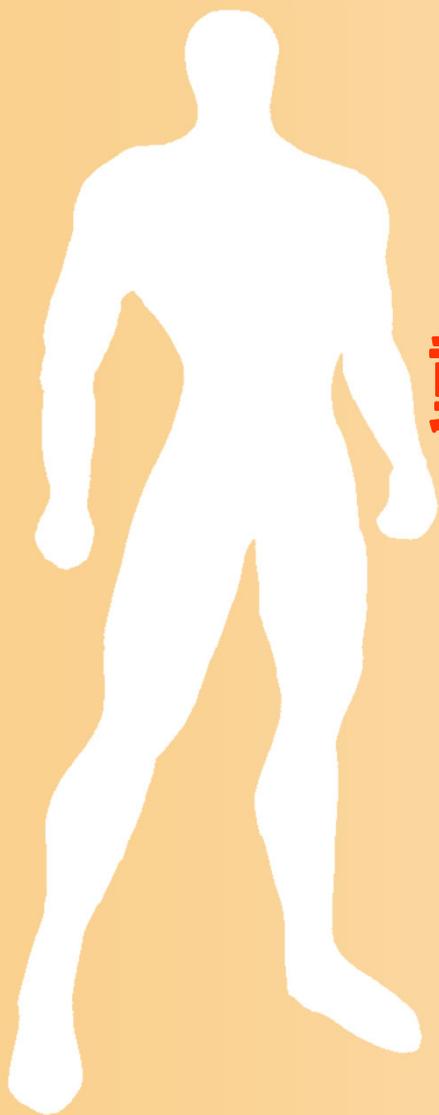
1. 立ち姿勢の形状モデルを作成

- 幾何形状＋テクスチャ画像

2. 骨格モデルを作成して、形状モデルと対応付ける

- 形状変形のための、各頂点に対する各体節の重みを調整
- 形状＋骨格モデルのことを  
**リグ (Rig)**、対応付けの作業を  
**リギング (Rigging)**、と呼ぶ





# 動作データの作成

# 動作データの作成

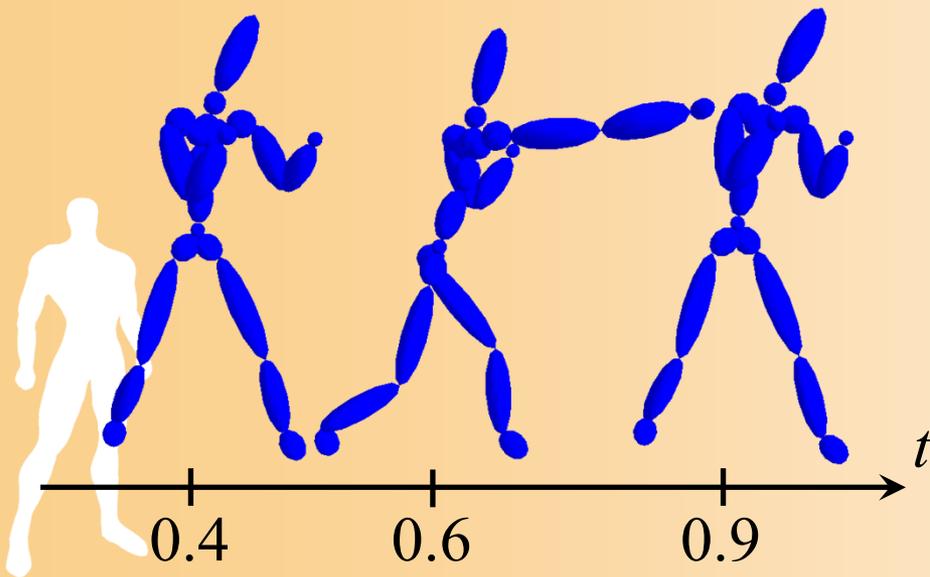
- 動作データの主な作成方法
  - キーフレームアニメーション
  - モーションキャプチャ
  - (動力学シミュレーションによる方法)
    - 3番目の方法は、やや特殊な方法なので、後述



# 動作データの作成方法

## キーフレームアニメーション

- 動作のキーとなる姿勢を手作業で作成
  - 専門知識や時間・労力が必要となる



## モーションキャプチャ

- 人間の身体にセンサを付けて、人間の動作を計測・取得
  - 取得した動作データの加工が困難



# 動作データの作成・表現方法

- 動作データの作成方法

- モーションキャプチャ
- キーフレームアニメーション

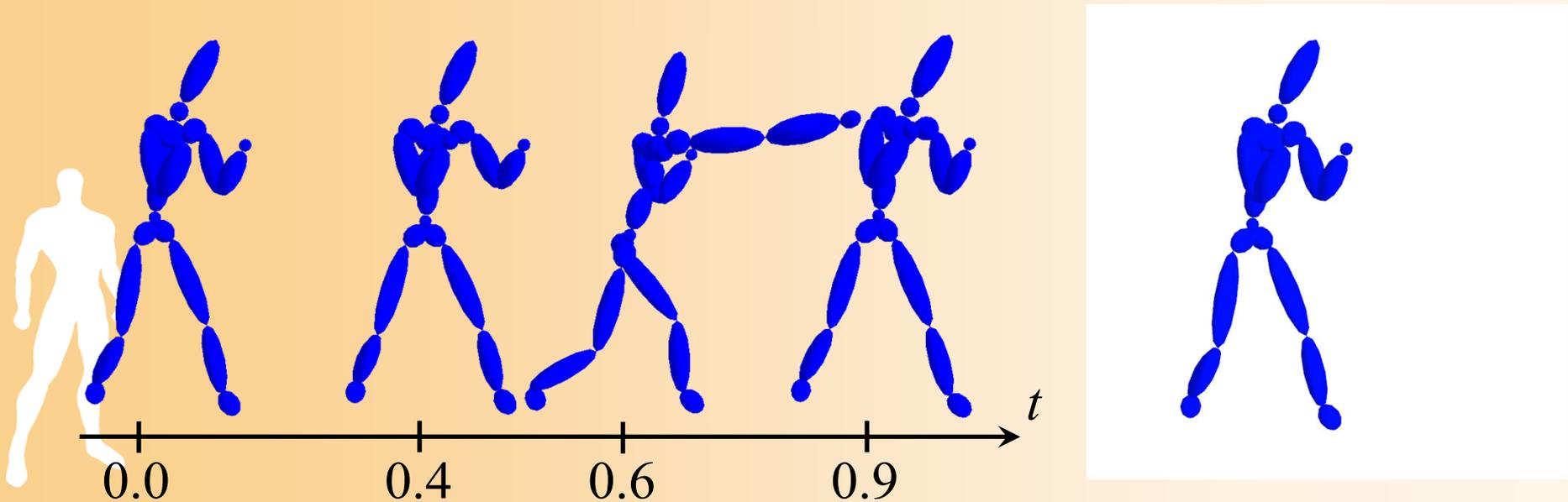
- 動作データの表現方法

- 一定間隔データ
    - モーションキャプチャデータは通常こちらの方法で表現
  - キーフレームデータ
    - 手作業での編集にはこちらの表現の方が向いている
- ※ アニメーションソフトの機能で一定間隔データとキーフレームデータの間の変換も可能



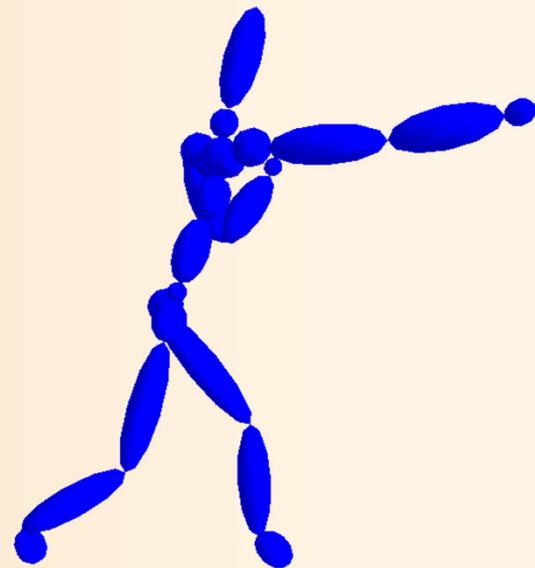
# キーフレームアニメーション

- 動作のキーとなる姿勢を手作業で作成
  - キー姿勢の時刻・姿勢(キーフレーム)を作成
    - 各関節の回転を操作、各部位の位置を操作
- キー姿勢の間を自動的に補間して動作再生



# キーフレームの姿勢の作成

- キーフレームにおけるキー姿勢の作成
  - 基本的にはアニメーターが手作業で作成
  - かなりの時間・労力がかかる
- 基本的な姿勢の作成方法
  - 1つの関節の回転を操作
  - 1つの部位(関節・体節)の位置を操作
    - 部位の位置に合わせて関節の回転を自動計算(後述する逆運動学を使用)



# モーションキャプチャ

- 人間の身体にセンサをつけて、人間の動作を計測・取得する方法
- モーションキャプチャ機器の方式
  - 光学式、慣性式、磁気式、RGB-Dカメラ式、等
  - 計測に使用するセンサーの種類の違いにより、さまざまな方式がある
    - センサーから得た計測データを、動作データに変換する処理が必要となる
    - 計測の時間間隔は、センサーの種類によって異なる
    - センサーの種類によっては、骨格モデルも推定可能



# モーションキャプチャ機器の種類(1)

## • 光学式

- 現在、主に使われている方式
- 全身に多数のマーカを付けて、複数の赤外線カメラで撮影し、マーカの位置から姿勢を推定
- 演技者の身体の拘束が少ない
- 高い精度で動作取得が可能
- 10～20台の赤外線カメラと広いスペースが必要



Optitrack © NaturalPoint



# モーションキャプチャ機器の種類(1)

## • 光学式

- 現在、主に使われている方式
- 全身に多数のマーカを付けて、複数の赤外線カメラで撮影し、マーカの位置から姿勢を推定



Optitrack © NaturalPoint

## • 自発光学式

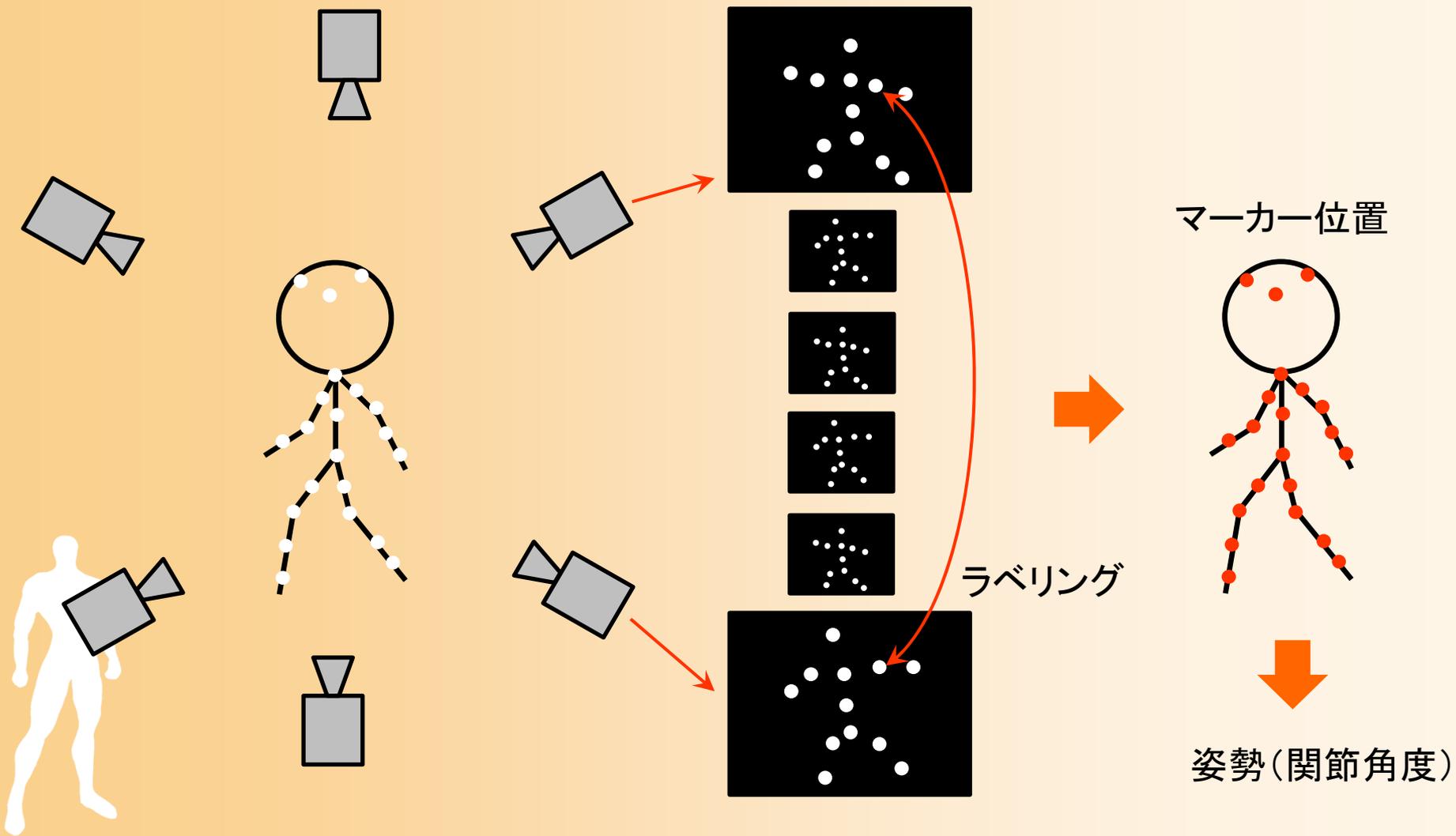
- 光学式の拡張版
- 各マーカが順番に発光
- ラベリング処理が容易になる



Visualeyez © Phoenix Tech.



# 光学式モーションキャプチャ



# モーションキャプチャ機器の種類(2)

- 慣性式

- 加速度・ジャイロ・地磁気センサーの組み合わせにより、各部位の向きを計測
- 小型・安価に実現可能



Perception Neuron © Noitom



Xsens MVN © Xsens

# モーションキャプチャ機器の種類(3)

- 磁気式

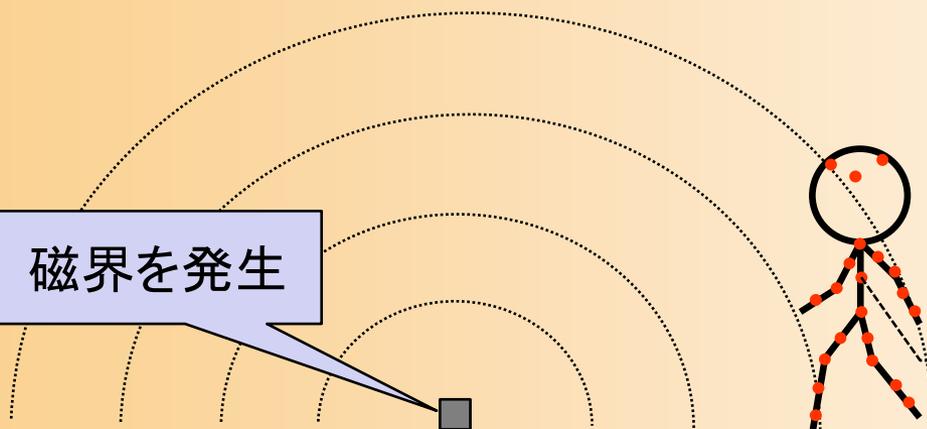
- 磁界を発生させて、体の各部位につけたセンサの位置・向きを計算
- 高速・高精度
- ケーブルの拘束のため動きづらい



trackSTAR © Ascension



磁界を発生



各コイルに流れる誘導電流の大きさから位置を計測

# モーションキャプチャ機器の種類(3)

## • 磁気式

- 磁界を発生させて、体の各部位につけたセンサの位置・向きを計算
- 高速・高精度
- ケーブルの拘束のため動きづらい



trackSTAR © Ascension

## • 機械式

- 体の各関節に回転角度を計測するための計測器を装着
- 計測器の重さのため動きづらい



Gypsy © Meta Motion



# モーションキャプチャ機器の種類(4)

- RGB-Dカメラ方式

- 各ピクセルの奥行きを計測
- 人体部位・姿勢の推定
  - 完全な姿勢は求められない
- 他の方式よりも精度は低い
- Kinect が代表的

- 事前に学習したモデルを使って、デプス画像中の各部位の位置を推定

Shotten et al., “Real-Time Human Pose Recognition in Parts from a Single Depth Image”, IEEE CVPR 2011.



Kinect © Microsoft



# モーションキャプチャ機器の種類(5)

- マーカレスカメラ方式

- 部位の長さが既知であれば、通常のカメラのみでも姿勢推定は可能
- RGB-Dカメラを使うよりも、さらに精度は落ちる
- OpenPose など



OpenPose [Chao 2019]

- 機械学習により画像中の関節点を推定

Zhe Cao et al., “OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields”, IEEE Trans on PAMI, 2019



# モーションキャプチャ機器の比較

方式 (製品例)	光学式 (Optitrack)	慣性式 (Perception Neuron)	RGB-Dカメラ方式 (Kinect)
精度	高	中	低
スーツ着用	要	要	不要
専用スペース	必要	不要	不要
動作(姿勢)取得	○	○	△
骨格モデル推定	○	×	△
指の姿勢取得	△※1	○	△※2
顔の表情取得	△※1	×	△※2
最大秒間フレーム数	120	60	30
価格	数百～数千万円	数十～一千万円	数万円

※1 全身の姿勢とは別にキャプチャを行う必要がある

※2 限定された状態の識別しかできない

# 動作データの作成方法の比較

- モーションキャプチャ

- 人間の動きをそのまま取り込めるので、一見手軽そう
- 実際には、ノイズや、演技者とキャラクタの骨格の違いなどのため、かなり編集が必要になる
- そのままではキーフレームがないので修正がしづらい

- キーフレームアニメーション

- ゼロから作成しなければならないため大変
- キーフレームが設定されているので、修正はしやすい
- 人間らしい細部の動きや自然さを実現するのは難しい

- 両者の使い分けや組み合わせが必要



# 動作データの作成・利用方法

- 既存のソフトウェアを使用して作成
  - キーフレームアニメーションにより動作を作成する場合は、アニメーション制作ソフトウェアを利用
  - モーションキャプチャにより動作を作成する場合は、使用機器に対応したソフトウェアを利用
    - さらに修正を行う場合は、アニメーション制作ソフトウェアを利用
  - 通常は、人体モデルと合わせて、動作データを作成する
    - 演技者の骨格にもとづく動作データを、キャラクタの骨格にもとづく動作データに変換する必要がある  
(モーションリターゲットング(後述)の技術(機能)を利用)
- アニメーション編集ソフトウェアからファイルに出力した動作データを読み込んで使用





# 動作データの入手方法

- モーションキャプチャによる動作取得は難しい
  - 機器はあっても、取得後の動作データの修正が大変
- 研究用の無料の動作データ集
  - CMU motion capture library
    - <http://mocap.cs.cmu.edu/motcat.php>
  - Human 3.6M (点群データを含む)
    - <http://vision.imar.ro/human3.6m/>
  - 豊橋技科大 モーションキャプチャデータライブラリ
    - <https://galaxy.val.cs.tut.ac.jp/ToMoLoW/index.html>
- 有料の動作データ集
  - 購入可能な動作データは少ない



# まとめ

- 人体モデルの基礎
- 骨格・姿勢・動作の表現
  - 骨格モデルの表現
  - 姿勢・動作の表現
  - ワンスキンモデル
- 人体モデルの作成
- 動作データの作成
  - キーフレームアニメーション
  - モーションキャプチャ



# 次回予告

- 人体モデル(骨格・姿勢・動作)の表現
- 人体モデル・動作データの作成方法
- サンプルプログラム、動作再生
- 順運動学、人体形状変形モデル
- 姿勢補間、キーフレーム動作再生、動作補間
- 動作接続・遷移、動作変形
- 逆運動学、モーションキャプチャ
- 動作生成・制御

