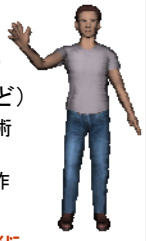
コンピュータグラフィックス特論 II

第12回　キャラクタアニメーション（3）

九州工業大学　尾下 真樹

キャラクタ・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）のアニメーションを実現するための技術
- キャラクタ・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



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
- 順運動学
- 逆運動学
- 姿勢補間
- 動作接続・遷移・補間
- 動作変形・生成・制御

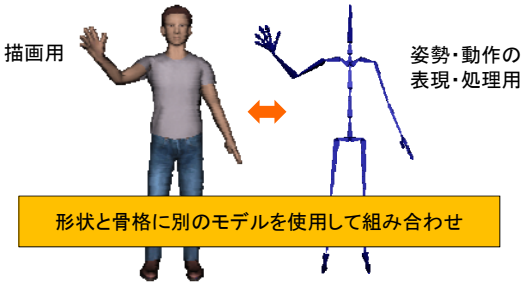
今日の内容

- 前回までの復習
- 逆運動学
- 姿勢補間
- キーフレームアニメーション

前回までの復習

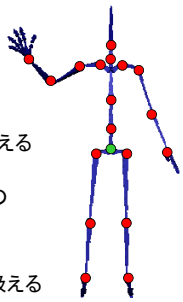
人体モデルの表現

形状モデル（ポリゴンモデル）　骨格モデル（多関節体）



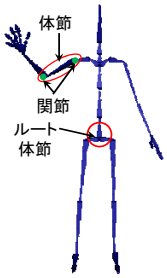
骨格モデルの表現

- 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・姿勢の表現方法

- 骨格情報と姿勢情報を分ける
- 骨格情報の中で、関節・体節を分ける
  - 体節
    - 複数の関節と接続
    - 各関節の接続位置
      - 体節のローカル座標系
  - 関節
    - 2つの体節の間を接続
      - 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、体節と関節を分ける

```
// 人体モデル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;
};

// 人体モデルの関節を表す構造体
struct  Joint
{
    // 接続体節
    Segment *              segments[ 2 ];
};
```

```
// 人体モデル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

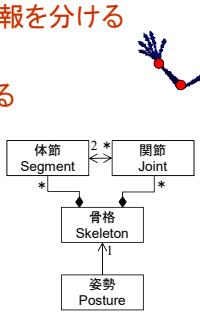
- 骨格情報と姿勢情報を分ける

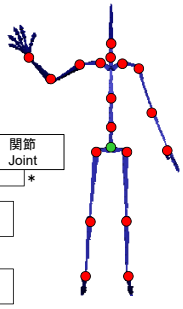
```
// 人体モデルの姿勢を表す構造体
struct  Posture
{
    Skeleton *  body;
    Point3f       root_pos;        // ルートの位置
    Matrix3f      root_ori;        // ルートの向き（回転行列表現）
    Matrix3f *  joint_rotations; // 各関節の回転（回転行列表現）
                                          // [リンク番号] リンク数分の配列
};
```

骨格モデル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

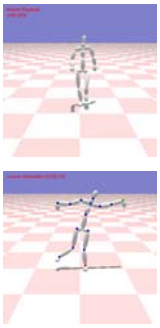
```
// 多関節体の体節を表す構造体
struct  Segment

// 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
```





デモプログラム

- 複数のデモを含むプログラム
  - マウスの中ボタン or m キーで切り替え
- 動作再生
- キーフレーム動作再生
- 順運動学計算
- 逆運動学（CCD-IK）
- 姿勢補間
- 動作補間（2つの動作の補間）
- 動作接続・遷移

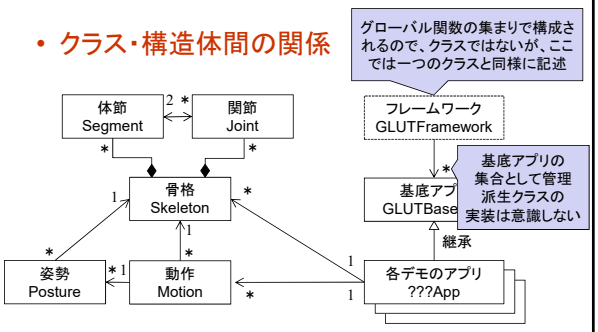


サンプルプログラム

- デモプログラムの一部
  - 骨格・姿勢のデータ構造定義（SimpleHumn.h/cpp）
  - BVH動作クラス（BVH.h/cpp）
  - アプリケーションの基底クラス（GLUTBaseApp.h/cpp）
    - 各イベント処理のためのメソッドの定義を含む
    - 本クラスを派生させて各アプリケーションクラスを定義
  - メイン処理、コールバック関数（GLUTBaseApp.cpp）
    - 全アプリケーションを管理、切替
    - 実行中のアプリケーションのイベント処理を呼び出し
  - 各アプリケーションの実装（human_sample.cpp）
    - 主要な処理を各自で実装（レポート課題）
      - 課題提出をしやすいように全アプリケーションの実装を一つのファイルに集約

クラス図

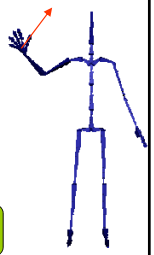
- クラス・構造体間の関係

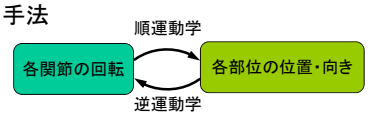


逆運動学

運動学（復習）

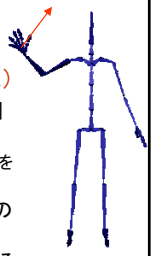
- 運動学（キネマティックス）
  - 多関節体の姿勢表現の基礎となる考え方
  - 人間の姿勢は、全関節の関節回転により表現できる
  - 各関節の回転と、各部位の位置・向きの間の関係を計算するための手法





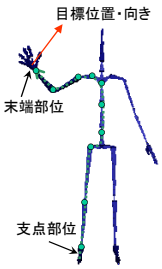
順運動学と逆運動学（復習）

- 順運動学（フォワード・キネマティックス）
  - 多関節体の関節回転から、各部位の位置・向きを計算
  - 回転・移動の変換行列の積により計算
- 逆運動学（インバース・キネマティックス）
  - 指定部位の位置・向きから、多関節体の関節回転を計算
    - 手先などの移動・回転量が与えられた時、それを実現するための関節回転の変化を計算
  - 動きを指定する時、関節回転よりも、手先の位置・向きなどを使った方がやりやすい
  - ロボットアームの軌道計画等にも用いられる



逆運動学

- 逆運動学（インバース・キネマティックス）
  - 入力
    - 現在の姿勢（腰の位置・向き、各関節の回転）
    - 任意の末端部位（エンドエフェクタ）とその目標位置・向き
      - 複数の部位となったり、位置・向きの一部のみを指定する場合がある
    - 支点部位
  - 出力
    - 入力を満たすような姿勢変化を計算

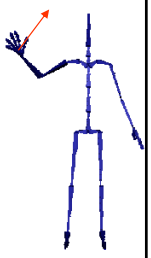


逆運動学計算における問題

- 逆運動学での制約条件に関する問題
  - アンダー・コンストレインツ（少な過ぎる制約）
    - ある部位の位置・向きを指定しただけでは姿勢が一意に決まらない
      - 例：手の位置がきまっても、その手の位置を実現するような全身の関節角度の組は無数に存在する
    - 何らかの評価基準を入れて適当な解を決定する
      - 例：前ステップの姿勢との差がなるべく小さくなる解を選択
  - オーバー・コンストレインツ（多過ぎる制約）
    - 逆に制約が多すぎて、与えられた制約を満足する姿勢がない

逆運動学（IK）の計算方法

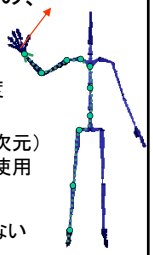
- いくつかの計算方法がある
  - 数値的解法（Numerical）
    - 擬似逆行列を使った計算方法
    - Cyclic Coordinate Descent（CCD）法
    - 粒子法（Particle-IK）
    - 非線形最適化問題による計算方法
  - 解析的解法（Analytical）
  - サンプルデータを用いる方法



IKの数値的解法（1）

- ヤコビ行列の疑似逆行列による方法
  - 各関節の角度を微小変化させたときの、末端部位の微小変化を計算

    $\mathbf{J}\Delta\boldsymbol{\theta} = \Delta\mathbf{p}$
    - $\Delta\boldsymbol{\theta}$ 支点から末端までの各関節の角度変化（n次元）（オイラー角表現が基本）
    - $\Delta\mathbf{p}$ 末端部位の位置・向き変化（3or6 次元）向きにはオイラー角や部位の軸などを使用
  - $\Delta\mathbf{p}$ を満たすような $\Delta\boldsymbol{\theta}$ を計算
    - 通常 n>3or6 なので解は一意に決まらない



IKの数値的解法（2）

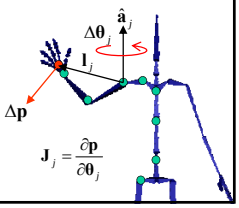
- ヤコビ行列の計算
  - 各関節の角度を微小変化させたときの、末端部位の微小変化

$$\begin{pmatrix} \boxed{\mathbf{J}} \\ \mathbf{J}_j \end{pmatrix} \begin{pmatrix} \\ \Delta\boldsymbol{\theta} \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \Delta\mathbf{p} \\ \\ \end{pmatrix}$$

関節から末端へのベクトル

$$\mathbf{J}_j = \hat{\mathbf{a}}_j \times \mathbf{l}_j$$

j番目の関節の回転に応じた末端の変化　回転軸の向きを表す単位ベクトル　※ 全てワールド座標系で計算



IKの数値的解法（3）

- 擬似逆行列による計算
  - Jは正則ではなので逆行列は計算できない
  - 擬似逆行列を使用（$|\Delta\theta|$が最小となる解を計算）

    $\mathbf{J}\Delta\boldsymbol{\theta} = \Delta\mathbf{p}$

    $\Delta\boldsymbol{\theta} = \mathbf{J}^{+}\Delta\mathbf{p} \qquad \mathbf{J}^{+} = \mathbf{J}^{T}\left(\mathbf{J}\mathbf{J}^{T}\right)^{-1}$
    - 最小二乗法の一種
    - 逆行列が計算できれば擬似逆行列は計算可能
  - $\mathbf{J}$ には、関係のない関節は含まないようにする必要がある（逆行列が計算できなくなる）

IKの数値的解法（4）

- 零空間写像による他の制約の追加

  $\Delta\boldsymbol{\theta} = \mathbf{J}^{+}\Delta\mathbf{p} + \left(\mathbf{I} - \mathbf{J}^{+}\mathbf{J}\right)\Delta\boldsymbol{\theta}'$
  - 第2項は末端部位の位置・向きには影響しない
    - 影響しない範囲で、任意の修正を適用できる
    - 関節回転可能範囲の制約、重心の位置の制約、等
- 重み付き擬似逆行列による回転割合の制御

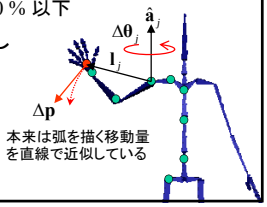
  $\Delta\boldsymbol{\theta} = \mathbf{J}^{+}\Delta\mathbf{p} \qquad \mathbf{J}^{+} = \mathbf{W}^{-1}\mathbf{J}^{T}\left(\mathbf{J}\mathbf{W}^{-1}\mathbf{J}^{T}\right)^{-1}$
  - $\mathbf{W}$ は各回転軸の重みを表す n×n 行列

IKの数値的解法(5)

- ヤコビ行列による計算方法の注意

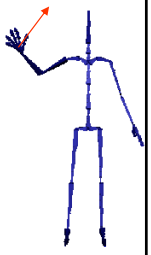
$$\Delta\boldsymbol{\theta} = \mathbf{J}^{+}\Delta\mathbf{p} + \left(\mathbf{I} - \mathbf{J}^{+}\mathbf{J}\right)\Delta\boldsymbol{\theta}'$$

  - $\Delta\mathbf{p}$ の長さは十分小さく取る必要がある
    - リンク全体の長さの 2～10 % 以下
  - 繰り返し計算によって少しずつ姿勢を修正
  - 姿勢が変わる度に $\mathbf{J}$ を計算し直す必要がある



逆運動学(IK)の計算方法

- いくつかの計算方法がある
  - 数値的解法(Numerical)
    - 擬似逆行列を使った計算方法
    - Cyclic Coordinate Descent(CCD)法
    - 粒子法(Particle-IK)
    - 非線形最適化問題による計算方法
  - 解析的解法(Analytical)
  - サンプルデータを用いる方法



CCD法

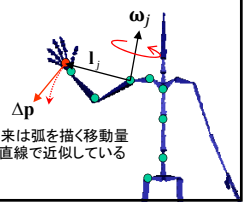
- Cyclic Coordinate Descent(CCD)法
  - 疑似逆行列を使う方法と同様の繰り返し計算で姿勢変化を計算
    - 疑似逆行列を使う方法よりも簡単な方法
  - 各関節の回転を独立に計算
    - 末端関節の位置が目標位置に来るように、各関節の回転軸・回転角度を計算

CCD法

- Cyclic Coordinate Descent(CCD)法
  - 各関節の回転を独立に計算
    - 末端関節の位置が目標位置に来るように、各関節の回転軸・回転角度を計算

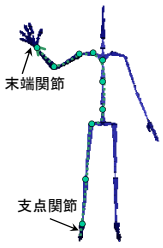
$$\boldsymbol{\omega}_j = \mathbf{l}_j \times \Delta\mathbf{p}$$

      - 回転軸に回転角度をかけたベクトルの形式で求まる
      - 必要に応じて、回転行列などの別の表現に変換する
    - 必要に応じて、各関節の回転量にウェイトを適用することも可能



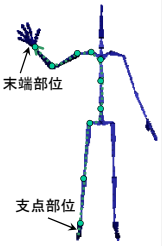
CCD法の計算手順

1. 指定された支点関節・末端関節にもとづき、支点関節から末端関節までの経路を決定
2. 繰り返し計算により、少しずつ姿勢を変化
3. 各繰り返し処理の中で、末端関節から支点関節に向かって順番に処理を適用
   - 末端関節の目標位置を満たすように各関節の回転を計算



CCD法の計算方法(1)

- 末端→支点関節のパス(関節順番)の決定
  - 各体節のどちら側(何番目の関節側)に末端・支点関節があるかが分かれば、パスを決定可能
    - 事前にインデックスを作成しておく
    - あるいは、ルートがどちらにあるかは分かるので、末端・支点関節からそれぞれルートに向かって辿り、合流した体節でパスを結合する
      - 後の説明では、こちらの方法を使用
    - 実装が難しければ、最初は常にルート体節が支点と仮定して作成すると、簡単になる

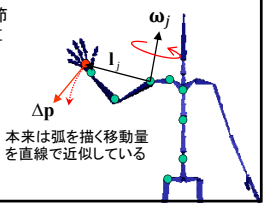


CCD法の計算方法(2)

- 各関節の回転を計算
  - 収束するまで繰り返し計算
    - 末端関節から支点関節に向かって順番に繰り返し
      - 各関節のヤコビ行列にもとづき、各関節の回転を独立に計算
      - 目標を満たすための各関節の回転軸・回転角度を計算

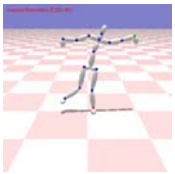
$$\omega_j = \mathbf{l}_j \times \Delta\mathbf{p}$$

      - 末端関節との間に腰(ルート)がある場合は、ルートの移動・回転が必要
      - 各関節の回転量に重みをかけて調整することも可能



プログラミング演習

- 逆運動学計算アプリケーション
  - CCD法による逆運動学計算のアプリケーション
  - マウス操作による関節の選択・移動(実装済み)
    - 以前の授業のピッキング技術(スクリーン座標で判定)により末端・支点関節を選択
    - マウスドラッグに応じて、視線に垂直な超平面上で、末端関節の目標位置を移動
  - CCD法による逆運動学計算(各自実装)



逆運動学計算アプリケーション

- InverseKinematicsCCDApp (一部未実装)
  - CCD法による逆運動学計算のアプリケーション
  - マウス操作による関節の選択・移動(実装済み)
    - 以前の授業のピッキング技術(スクリーン座標で判定)により末端・支点関節を選択
    - マウスドラッグに応じて、視線に垂直な超平面上で、末端関節の目標位置を移動
  - CCD法による逆運動学計算(各自実装)
    - ルート体節を支点とする → 任意の関節を支点とする
    - FindJointPath関数の一部を作成
    - ApplyInverseKinematicsCCD関数の一部を作成

CCD法のプログラミング

- CCD法による逆運動学計算
  - 入力:現在姿勢、支点関節番号、末端関節番号、末端関節の目標位置
  - 出力:逆運動学計算後の姿勢
    - 現在姿勢の入力と計算後の姿勢の出力に、同じ変数を使用

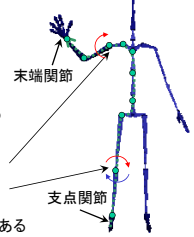
```
// 逆運動学計算(CCD法)
void  ApplyInverseKinematicsCCD( Posture & posture,
  int base_joint_no, int ee_joint_no, Point3f ee_joint_position );
```

CCD法のプログラミング(1)

- 支点関節から末端関節への経路(パス)探索
  - 入力:骨格情報、支点関節番号、末端関節番号
  - 出力:経路に含まれる関節番号+方向のリスト
    - 出力には可変長配列を使用
    - 関節番号+方向(1 or −1)
      - ルートよりも支点側(1)にあるか末端側(−1)にあるかで、関節周りの回転の方向が逆転するため、方向も合わせて出力

末端側:関節回転により末端側が回転

支点側:関節回転により支点側が回転
関節の回転を表すときには、
回転方向を反対にする必要がある



CCD法のプログラミング(1)

- 支点関節から末端関節への経路(パス)探索
  - 入力:骨格情報、支点関節番号、末端関節番号
  - 出力:経路に含まれる関節番号+方向のリスト
    - ルート体節が支点の場合は、支点番号を −1 とする
    - 出力には可変長配列を使用
    - 経路に含まれる関節番号+方向(1 or −1)のリスト
      - ルートよりも支点側(1)にあるか末端側(−1)にあるかで、関節周りの回転の方向が逆転するため、方向も合わせて出力

```
// 末端関節から支点関節へのパスを探索
void  FindJointPath( const Skeleton * body,
   int base_joint_no, int ee_joint_no,
   vector< int > & joint_path, vector< int > & joint_path_signs );
```

CCD法のプログラミング（1）

```
// 末端関節から支点関節への経路（パス）を探索
void  FindJointPath( const Skeleton * body,
    int base_joint_no, int ee_joint_no,
    vector< int > & joint_path, vector< int > & joint_path_signs )
{
    // 末端関節から探索を開始

    // 末端関節からルートに向かうパスを探索
    while（・・・）
        // ルート体節か支点関節に到達したら終了
        // 現在の関節をパスの配列に追加、関節の符号は全て 1 とする

    // 支点関節に到達したら終了（支点がルート体節 or 現在の関節）

    // 支点関節からルートに向かうパスを探索（上と同様の処理）

    // 2つのパスを統合
}
```

CCD法のプログラミング（1）

```
// 末端関節から支点関節への経路（パス）を探索
void  FindJointPath( const Skeleton * body,
    int base_joint_no, int ee_joint_no,
    vector< int > & joint_path, vector< int > & joint_path_signs )
{
    // 末端関節から探索を開始

    // 末端関節からルートに向かうパスを探索
    while（・・・）
        // ルート体節か支点関節に到達したら終了
        // 現在の関節をパスの配列に追加、関節の符号は全て 1 とする

    // 支点
    // 支点関節
    // 2つのパ
}
```

最初はルート体節が支点であると仮定して、ここまでの処理を作成すると良い
うまくいったら、任意の支点関節が指定されても対応できるように、残りの処理も作成する

CCD法のプログラミング（2）

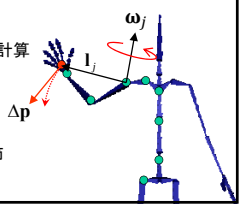
- 各関節の回転を計算
  - 収束するまで繰り返し計算
    - 末端関節から支点関節に向かって順番に繰り返し
      - 目標を満たすための各関節の回転軸・回転角度を計算

$$\boldsymbol{\omega}_j = \mathbf{l}_j \times \Delta\mathbf{p}$$

        - » 関節のローカル座標系で計算
      - 現在の関節の回転に、求めた回転を適用して、姿勢を更新

$$\mathbf{R}_j' = \mathbf{R}(\boldsymbol{\omega}_j)\,\mathbf{R}_j$$

        - » ルートよりも支点側の関節では、順番は逆になる



CCD法のプログラミング（2）

```
// 逆運動学計算（CCD法）
void  ApplyInverseKinematicsCCD( Posture & posture,
  int base_joint_no, int ee_joint_no, Point3f ee_joint_position )
{
    // 末端関節から支点関節へのパスを探索
    FindJointPath( ・・・);

    // CCD法の繰り返し計算（収束するか、一定回数繰り返したら終了）
    for ( int i=0; i<max_iteration; i++ )
    {
        // 末端関節から支点関節に向かって繰り返し
        for ( int j=0; j<joint_path.size(); j++ )
        {
            // 末端関節を目標位置に近づける、現在の関節の回転を計算
            // 回転を適用する際には、回転の方向を考慮

            // 順運動学計算を再計算
```

CCD法のプログラミング（2）

```
// 逆運動学計算（CCD法）
void  ApplyInverseKinematicsCCD( Posture & posture,
  int base_joint_no, int ee_joint_no, Point3f ee_joint_position )
{
    // 末端関節から支点関節へのパスを探索
    FindJointPath( ・・・);

    // CCD法の繰り返し計算（収束するか、一定回数繰り返したら終了）
    for ( int i=0; i<max_iteration; i++ )
    {
        // 末端関節から支点関節に向かって繰り返し
        for ( int j=0; j<joint_path.size(); j++ )
        {
            // 末端関節を目標位置に近づける、現在の関節の回転を計算
            // 回転を適用す
            // 順運動学計算を
```

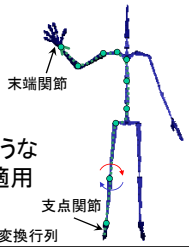
末端関節と現在の関節の間にルート体節がある場合は、ルート体節に移動・回転を適用

CCD法のプログラミング（3）

- ルートよりも支点側にある関節の回転
  - 前のスライドと同じ方法で関節回転を計算
    - 右図の赤矢印の回転が求まる
  - 逆の回転を適用
    - 右図の青矢印の回転を適用
  - そのままでは、ルートが固定されて、支点関節が移動する
  - 支点関節の位置・向きを保つような回転・移動を求めて、ルートに適用

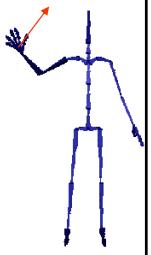
$$\mathbf{M} = \mathbf{M}_{base}\ \mathbf{M}_{base}{}'^{-1}$$

関節回転の適用前・後の支点関節の変換行列



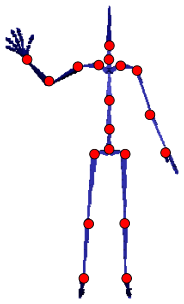
逆運動学(IK)の計算方法

- いくつかの計算方法がある
  - 数値的解法(Numerical)
    - 擬似逆行列を使った計算方法
    - Cyclic Coordinate Descent(CCD)法
    - 粒子法(Particle-IK)
    - 非線形最適化問題による計算方法
  - 解析的解法(Analytical)
  - サンプルデータを用いる方法



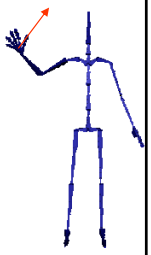
粒子法による解法

- 関節点を粒子とみなして、粒子の位置を計算
  - 粒子シミュレーションの手法により、末端関節の目標位置や粒子間の距離の条件を満たすような粒子の位置を計算
- 関節点(粒子)の位置の制約を満たすように、姿勢(全関節の回転)を計算



逆運動学(IK)の計算方法

- いくつかの計算方法がある
  - 数値的解法(Numerical)
    - 擬似逆行列を使った計算方法
    - Cyclic Coordinate Descent(CCD)法
    - 粒子法(Particle-IK)
    - 非線形最適化問題による計算方法
  - 解析的解法(Analytical)
  - サンプルデータを用いる方法

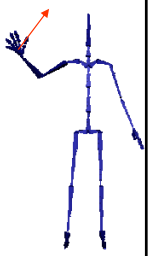


非線形最適化問題による解法

- 最適化問題と考えて、指定部位の位置・向きの条件を満たすような姿勢(関節角度の組み合わせ)を探索
- 疑似逆行列による解法(線形問題)を、より一般化した解法
- 目標関数の定義や最適化問題の解法にさまざまな方法がある
  - 手法によっては計算時間がかかる

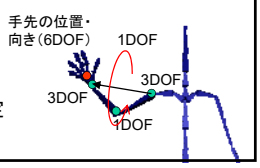
逆運動学(IK)の計算方法

- いくつかの計算方法がある
  - 数値的解法(Numerical)
    - 擬似逆行列を使った計算方法
    - Cyclic Coordinate Descent(CCD)法
    - 粒子法(Particle-IK)
    - 非線形最適化問題による計算方法
  - 解析的解法(Analytical)
  - サンプルデータを用いる方法



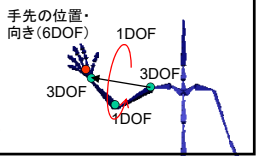
IKの解析的解法(1)

- 人間の骨格に基づく効率的な計算方法
- 手足の解析的なIK計算
  - 肩 or 股関節を支点として、手 or 足の位置・向きを満たす、腕・脚の姿勢を計算
  - 手先・足先の位置・向きは6自由度、
  - 腕・脚は全体で7自由度
  - 残りの自由度は1(ひじ・ひざの回転)のみ
    - 何らかの方法でこれを決定すれば解は一意に決まる



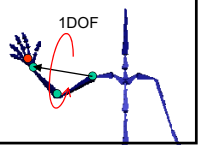
IKの解析的解法(2)

1. 入力:手先の目標位置・向き、肩(胴体)の位置・向き、腕の2本の体節の長さ
2. 目標位置と肩の距離から、ひじの屈伸回転を計算(1DOF)
3. 手先が目標位置に来るように、肩の回転を計算(2DOF)
4. 何らかの方法で、ひじの旋回回転を決定(1DOF)
5. 手先の目標向きに応じて、手首の回転を計算(3DOF)



IKの解析的解法(3)

- ひじ(ひざ)の旋回角度の計算方法の例
  - 現在の姿勢の旋回角度を保持
    - 姿勢を少しだけ変更するような場合であれば、有効
  - 標準的な旋回角度を適用
    - どのような目標位置・向きのときに、どのような旋回角度になるかのデータを用意しておき、そのデータにもとづいて決定

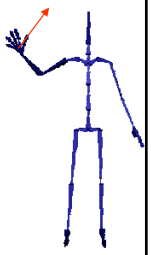


解析的解法の利用

- 解析的解法は広く用いられている
  - 繰り返し計算が不要であるため、一定の速度で高速に計算できる
    - オンラインアプリケーションでの利用には速度が重要
  - 人体モデルに特化した手法
    - 手足以外の関節の回転についても、何らかのルールに従って変化させることで、全身の姿勢変形に利用
    - アニメーション編集ソフトウェアにも組み込まれている
      - Autodesk Human IK
  - 基本的には大きな姿勢変形には向かない
    - 手足の位置を微調整するような姿勢変形に有効

逆運動学(IK)の計算方法

- いくつかの計算方法がある
  - 数値的解法(Numerical)
    - 擬似逆行列を使った計算方法
    - Cyclic Coordinate Descent(CCD)法
    - 粒子法(Particle-IK)
    - 非線形最適化問題による計算方法
  - 解析的解法(Analytical)
  - サンプルデータを用いる方法



サンプルデータを用いる方法

- Style-based IK [Grochow et al. 2004]
  - あらかじめ多数の姿勢データを学習し、制約条件が与えられると、制約条件を満たしつつ、学習データに近いような、適切な姿勢を計算
    - 潜在空間(低次元空間)に学習姿勢を写像
    - 潜在空間の中で姿勢を探索・合成
  - サンプルデータにもとづき冗長性の問題を解決
  - 一つの学習モデルで複数の動作は扱えない
    - 同じ制約条件でも動作によって適切な姿勢は異なる

サンプルデータを用いる方法

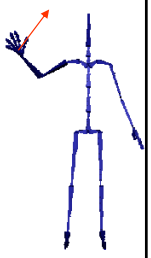
- 姿勢補間を用いる方法もある
  - 少ないパラメタ(決められた動作や末端部位)に対して、十分多数のサンプル姿勢が用意できる場合
  - 複数の姿勢の補間が必要になる
- 姿勢補間については次回説明

逆運動学計算の利用時の注意

- いずれの方法を用いても、入力姿勢から大きく離れた姿勢を生成することは難しい
  - 広い範囲の自然な姿勢の生成は難しい
- 姿勢や動作によっては、適切な末端関節の目標位置を与えることが難しい
  - 動作生成のための軌道を与えることは難しい
- 微小な姿勢の変化であれば対応可能
  - 手先や足先の位置を周囲の環境に応じて少し修正する程度の姿勢変形

逆運動学の計算方法のまとめ

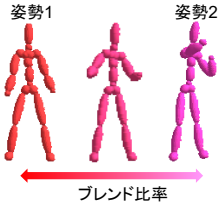
- いくつかの計算方法がある
  - 数値的解法（Numerical）
    - 擬似逆行列を使った計算方法
    - Cyclic Coordinate Descent（CCD）法
    - 粒子法（Particle-IK）
    - 非線形最適化問題による計算方法
  - 解析的解法（Analytical）
  - サンプルデータを用いる方法



姿勢補間

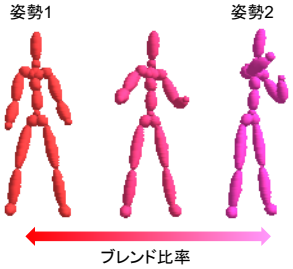
姿勢補間

- 基礎技術
  - 2つの姿勢の補間
    - 混合（Blending）、補間（Interpolation）、合成（Synthesis）などいくつかの呼び方がある
- 姿勢補間の応用例
  - キーフレームアニメーション
  - 動作補間
  - 動作接続・遷移



2つの姿勢の補間

- 2つの入力姿勢を指定された重み（比率）で補間（混合）して、新しい姿勢を生成

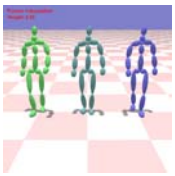


2つの姿勢の補間の計算方法

- 2つの入力姿勢を指定された重み（比率）で補間（混合）して、新しい姿勢を生成
- 腰の位置・向き、各関節の回転を補間
  - 腰の位置の補間
    - 位置の補間手法を適用
  - 腰の向き・各関節の回転の補間
    - 向き・回転の表現方法にもとづいて、適切な補間手法を適用（もしくは別の表現方法に変換して補間）
      - 四元数表現を使った補間
      - オイラー角表現を使った補間

プログラミング演習

- 姿勢補間アプリケーション
  - 2つのサンプル姿勢を補間して、新しい姿勢を生成
  - マウス操作（左右方向の左ドラッグ）に応じて補間の重みを変更
    - 補間の重みに応じて姿勢を更新
  - 2つの姿勢の補間処理（各自作成）



姿勢補間アプリケーション

- PostureInterpolationApp（一部未実装）
  - 2つのサンプル姿勢を設定
    - BVH動作＋時刻を指定して読み込み・設定
  - マウス操作（左右方向の左ドラッグ）に応じて補間の重みを変更
    - 補間の重みに応じて姿勢を更新
  - 補間姿勢＋2つのサンプル姿勢を描画
  - 2つの姿勢の補間処理（各自作成）
    - PostureInterpolation関数を実装

姿勢補間のプログラミング

- 姿勢補間
  - 2つの姿勢＋重みを入力、補間姿勢を出力

```
// 姿勢補間（2つの姿勢を補間）
void  PostureInterpolation(
    const Posture & p0, const Posture & p1, float ratio,
    Posture & p )
{
    // 2つの姿勢の各関節の回転を補間（関節ごとに繰り返し）
    p.joint_rotations[ i ] = ???;
    // 2つの姿勢のルートの向きを補間
    p.root_pos = ???;
    // 2つの姿勢のルートの位置を補間
    p.root_ori = ???;
}
```

姿勢補間の計算方法

- 向き・回転の補間（腰の向き・関節の回転）
  - 四元数を使った球面線形補間（SLERP）
    - vecmathのクラス・メソッドを使って計算可能
    - キーフレームアニメーションの講義の説明を参照
- 位置の補間（腰の位置）
  - 線形補間

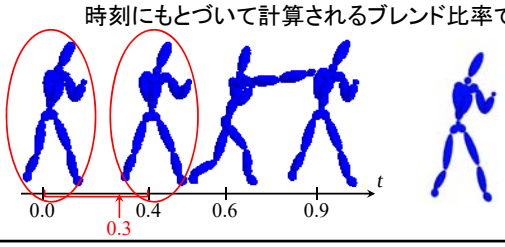
姿勢補間のプログラミング

- 2つの姿勢の補間
  - 関節の回転の補間（各関節に対して繰り返し）
    - 2つの姿勢の関節回転（回転行列表現）を四元数表現に変換
    - 重みに応じて、2つの四元数の補間を計算
    - 計算結果を回転行列表現に変換し、出力姿勢の関節回転として設定
  - 腰の向きの補間
    - 関節の回転の補間と同じ
  - 腰の位置の補間
    - 重みに応じて、線形補間

キーフレームアニメーション

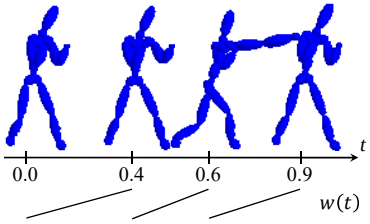
キーフレームアニメーション

- キー姿勢の間を補間して動作を生成
  - (時刻、姿勢データ)の組の配列から動作を生成
  - 各区間で、前後の2つのキーフレームの姿勢を、時刻にもとづいて計算されるブレンド比率で補間



キーフレームアニメーション

- ブレンド比率(姿勢補間の重み)の変化
  - 時間の関数として設定
  - 関節ごとに異なる関数を設定することもできる



プログラミング演習

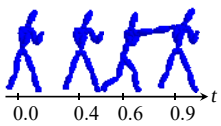
- キーフレーム動作再生アプリケーション
  - キーフレーム動作再生
  - BVH動作＋キー時刻の情報から、キーフレーム動作を初期化
  - 比較用に、元のBVH動作や全キー姿勢を並べて描画
  - キーフレーム動作からの姿勢取得(各自作成)

キーフレーム動作再生アプリケーション

- KeyframeMotionPlaybackApp(一部未実装)
  - 基本的に MotionPlaybackApp と同じ再生処理
  - 初期化処理(Initialize関数)で、キーフレーム動作を初期化(BVH動作＋キー時刻の情報から)
  - アニメーション処理(Animation関数)
    - キーフレーム動作からの姿勢取得を呼び出し
  - 描画処理(Display関数)
  - キーフレーム動作からの姿勢取得(各自作成)
    - GetKeyframeMotionPosture関数の一部を実装
    - 姿勢補間は、前に作成した関数を呼び出して使用

キーフレーム動作の表現方法

```
// キーフレーム動作を表す構造体
struct  KeyframeMotion
{
    // 骨格モデル
    Skeleton *          body;
    // キーフレーム数
    int                 num_keyframes;
    // 各キー時刻の配列［キーフレーム番号］
    float  *            key_times;
    // 各キー姿勢の配列［キーフレーム番号］
    Posture *           key_poses;
};
```



キーフレームアニメーションのプログラミング

- キーフレーム動作からの姿勢取得

```
// 姿勢を取得
void  GetKeyframeMotionPosture(
        const KeyframeMotion & m, float time, Posture & p )
{
    // 指定時刻に対応する区間番号を取得
    for ( int i=0; i<m.num_keyframes; i++ )
        if ( (m.key_times[i] <= time ) && ( time < m.key_times[i+1] ) )
            no = i;
    // 指定時刻に応じて前後の姿勢の補間割合(0.0～1.0)を計算
    float  s = ???;
    // 前後のキー姿勢を補間
    PostureInterpolation( ??? );
}
```

キーフレーム動作と時刻を入力
姿勢を出力

現在時刻にもとづき、区間の前後のキー姿勢のブレンド比率(0.0～1.0)を計算

作成済みの姿勢補間関数を利用

キーフレームアニメーションのプログラミング

1. 指定時刻に対応する、区間番号を取得
   - 全キー時刻の情報（KeyframeMotion 構造体のkey_times配列）を参照し、指定された時刻がi 番目のキー時刻よりも後で、i+1 番目のキー時刻よりも前になるような、i 番目の区間を探索
2. 指定時刻に対応する、ブレンド比率の計算
   - 区間の開始時に0.0 になり、区間の終了時に1.0 になるように、ブレンド比率を計算
   - i番目のキー時刻からの経過時刻を、i番目の区間の長さ（i番目と i+1番目のキー時刻の差）で割ることで計算
3. 前後のキー姿勢をブレンド比率で補間して出力

まとめ

- 前回の復習
- 逆運動学
- 姿勢補間
- キーフレームアニメーション

レポート課題

- キャラクタ・アニメーション基礎技術
  - サンプルプログラム（human_sample.cpp）の未実装部分を作成
    1. 順運動学計算
    2. 姿勢補間
    3. キーフレーム動作再生
    4. 動作接続・遷移
    5. 動作補間
    6. 逆運動学計算（CCD法）
       1. ルート体節を支点とする場合
       2. 任意の関節を支点とする場合

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
- 順運動学
- 逆運動学
- 姿勢補間
- 動作接続・遷移・補間
- 動作変形・生成・制御