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

2021年度

今日の内容

- 前回までの復習
- 姿勢補間
- キーフレーム動作再生
- 動作補間
- レポート課題(1)

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
- 姿勢補間、キーフレームアニメーション、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

- 前回までの復習
- 姿勢補間
- キーフレーム動作再生
- 動作補間
- レポート課題(1)

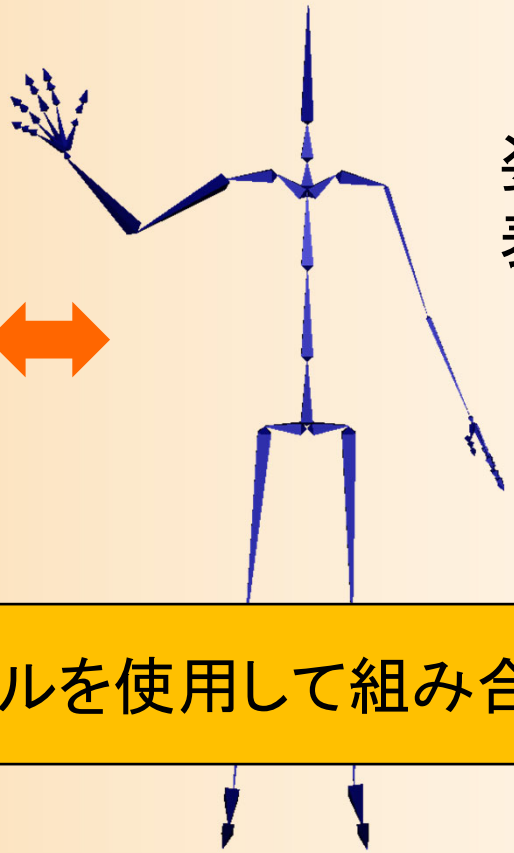
前回までの復習

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）

描画用

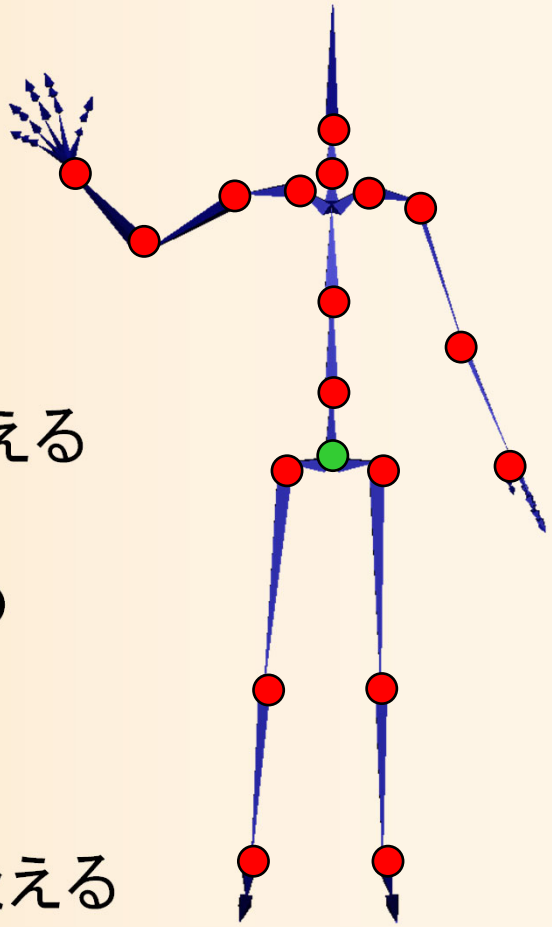


姿勢・動作の
表現・処理用

形状と骨格に別のモデルを使用して組み合わせ

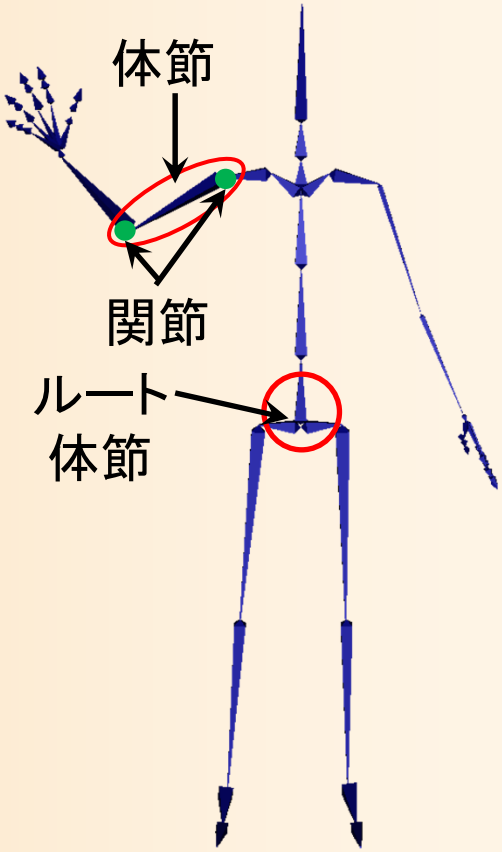
骨格モデルの表現

- 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

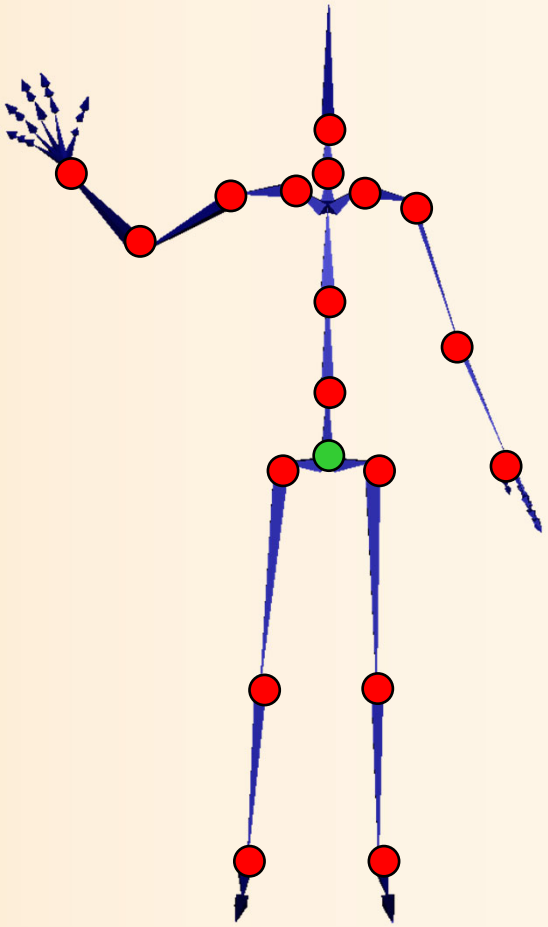
- 骨格情報と姿勢情報を分ける
- 骨格情報の中で、関節・体節を分ける
  - 体節
    - 複数の関節と接続
    - 各関節の接続位置
      - 体節のローカル座標系
  - 関節
    - 2つの体節の間を接続
      - ルート側・末端側の体節



骨格・姿勢・動作のデータ構造

- 骨格・姿勢の構造体定義（SimpleHuman.h/cpp）

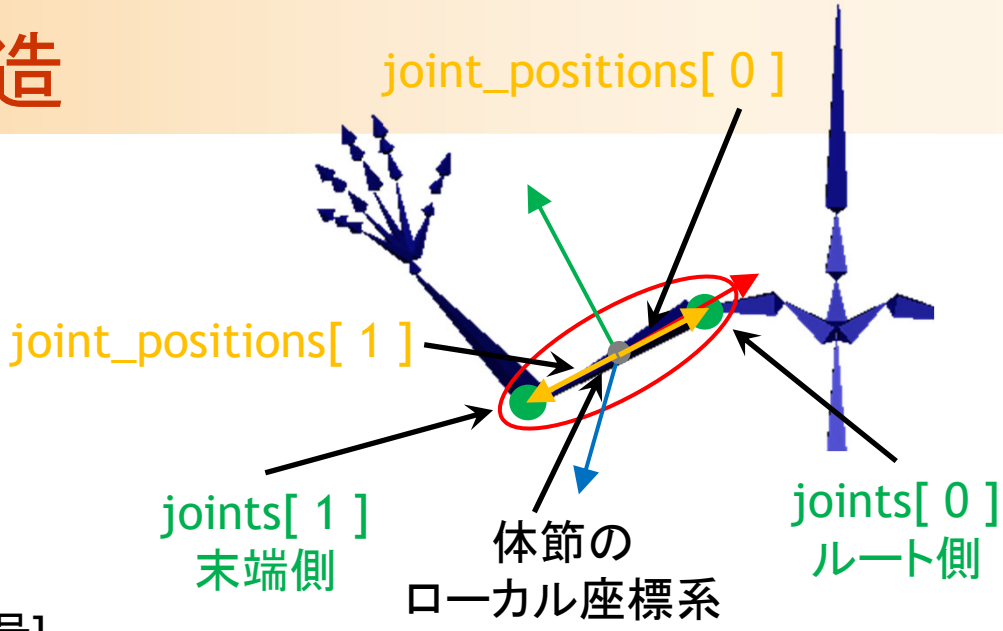
```
// 人体モデルの体節を表す構造体
struct  Segment

// 人体モデルの関節を表す構造体
struct  Joint

// 人体モデルの骨格を表すクラス
class  Skeleton

// 人体モデルの姿勢を表すクラス
class  Posture
```



骨格モデルのデータ構造(1)

- 体節のデータ構造

```
// 人体モデルの体節を表す構造体
Struct  Segment
{
    // 体節番号・名前
    int             index;
    string          name;

    // 体節の接続関節数
    int             num_joints;
    // 接続関節の配列 [接続関節番号]
    Joint **        joints;
    // 各接続関節の接続位置の配列(体節のローカル座標系)[接続関節番号]
    Point3f *       joint_positions;

    // 体節の末端位置
    bool            has_site;
    Point3f         site_position;
};
```



複数の関節と接続
ルート体節以外は、0 番目の
関節が、ルート側の関節とする

骨格モデルのデータ構造(2)

- 関節のデータ構造

```
// 人体モデルの関節を表す構造体
struct  Joint
{
    // 関節番号・名前
    int             index;
    string          name;

    // 接続体節
    Segment *     segments[ 2 ];
};
```

2つの体節の間を接続
0 番目の体節が、ルート側の体節とする

骨格モデルのデータ構造(3)

- 骨格データ構造

```
// 人体モデルの骨格を表すクラス
struct  Skeleton
{
    // 関節数
    int             num_segments;
    // 関節の配列 [関節番号]
    Segment **      segments;

    // 体節数
    int             num_joints;
    // 体節の配列 [体節番号]
    Joint **        joints;

    Skeleton( int s, int j );
    ~Skeleton();
};
```

姿勢のデータ構造

- 姿勢のデータ構造

```
// 人体モデルの姿勢を表すクラス
class  Posture
{
  public:
    const 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;// 各関節の相対回転（回転行列表現）
                                    // [関節番号] ※ 関節数分の配列
  public:
    // コンストラクタ
    Posture( Sleleton * b );
    // 初期化
    void  Init( Skeleton * b );
};
```

動作のデータ構造

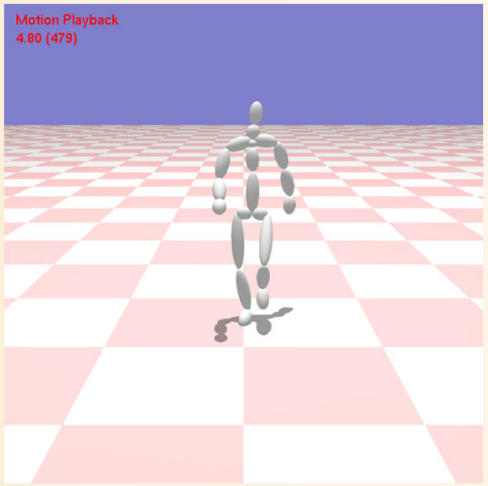
- 動作のデータ構造

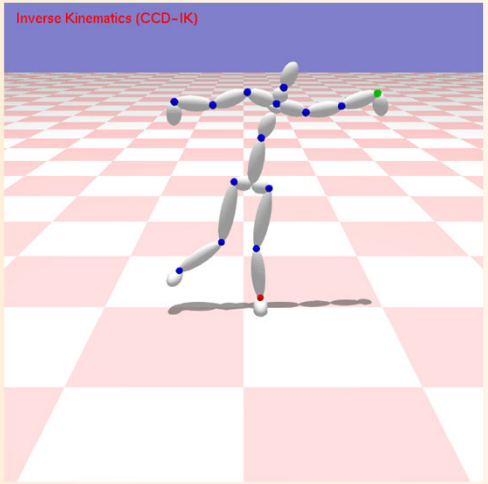
```
// 人体モデルの動作を表すクラス
class  Motion
{
    // 骨格モデル
    const Skeleton *  body;
    // フレーム数
    int                    num_frames;
    // フレーム間の時間間隔
    float                  interval;
    // 全フレームの姿勢 [フレーム番号]
    Posture *              frames;

    // 姿勢を取得
    void  GetPosture( float time, Posture & p ) const;
};
```

時刻を入力として、
その時刻の姿勢を出力

デモプログラム

- 複数のアプリケーションを含む
  - 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
- 姿勢補間
- 動作補間（2つの動作の補間）
- 動作接続・遷移
- 動作変形
- 逆運動学計算（CCD-IK）



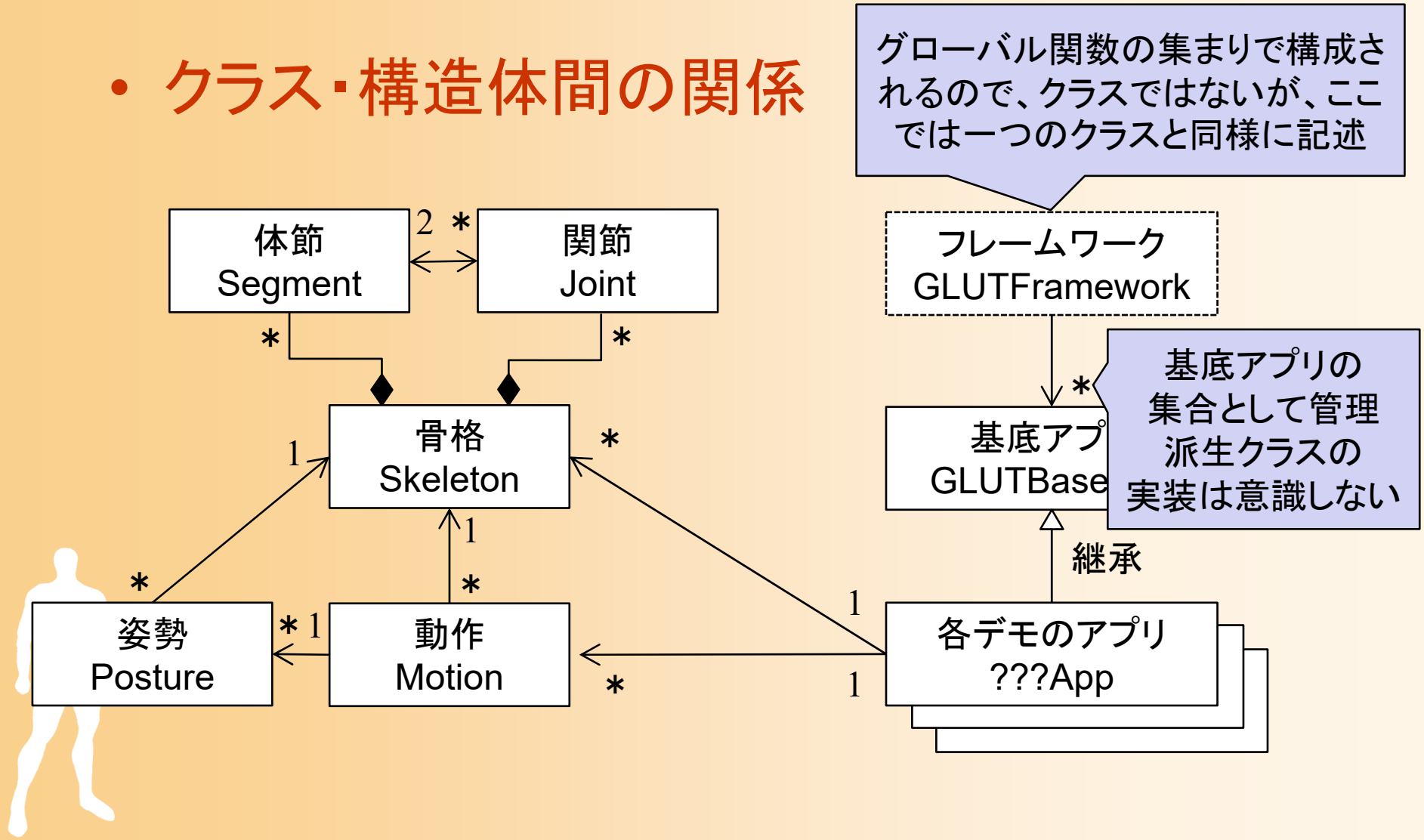


サンプルプログラム

- デモプログラムの一部
  - 骨格・姿勢・動作のデータ構造定義（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）
    - 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
  - コールバック関数（SimpleHumanGLUT.h/cpp）
    - GLUTBaseAppの定義・実装、全アプリケーションを管理・切替
    - アプリケーションのイベント処理を呼び出すGLUTコールバック関数
  - メイン処理（SimpleHumanMain.cpp）
  - 各アプリケーションの定義・実装（???App.h/.cpp）
    - 主要な処理を各自で実装（レポート課題）

クラス図

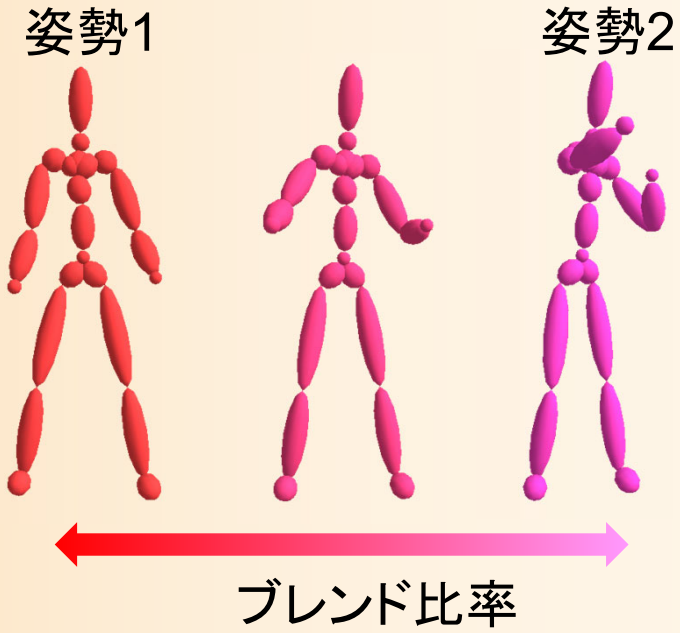
- クラス・構造体間の関係



姿勢補間

姿勢補間

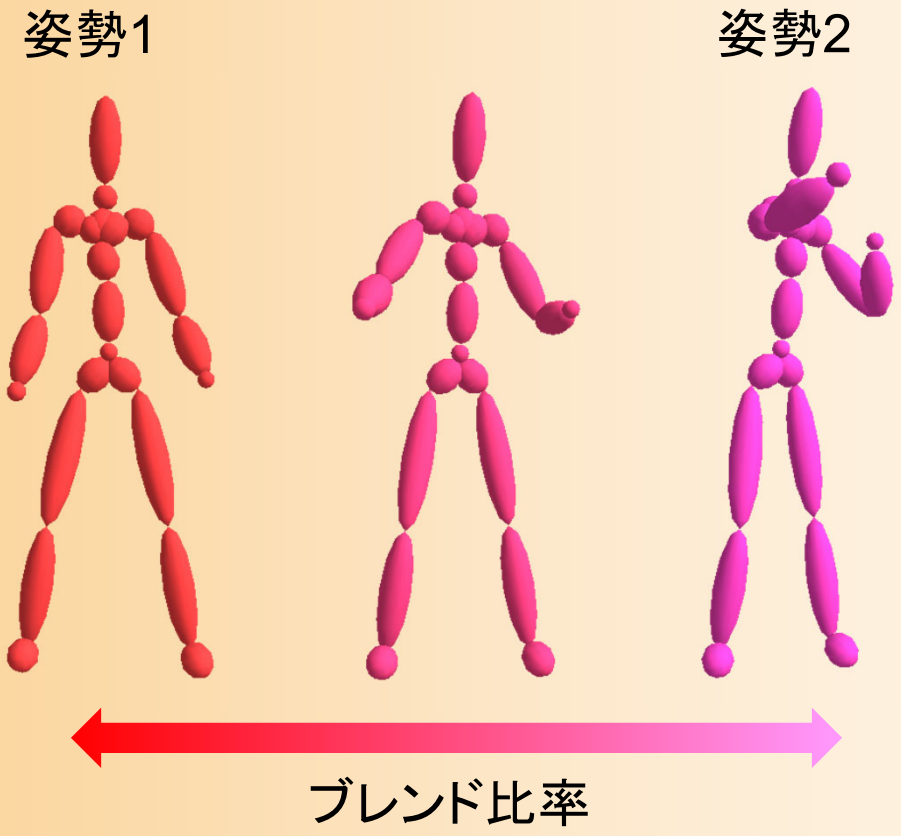
- 基礎技術
  - 2つの姿勢の補間
    - 混合(Blending)、
      補間(Interpolation)、
      合成(Synthesis)など
      いくつかの呼び方がある



- 姿勢補間の応用例
  - キーフレーム動作再生、動作補間、
    動作接続・遷移、動作変形

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

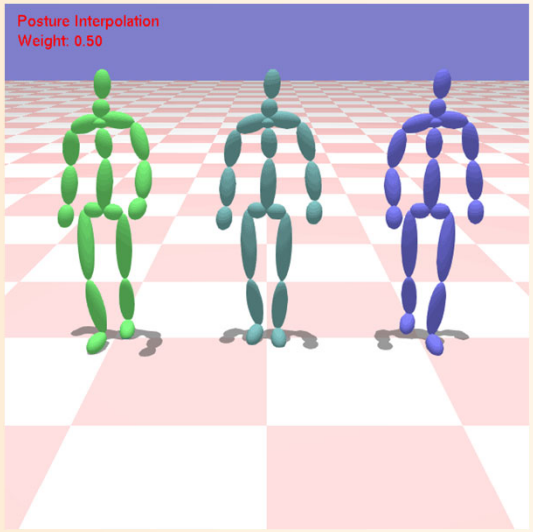
- 2つの入力姿勢を指定された重み(比率)で補間(混合)して、新しい姿勢を生成

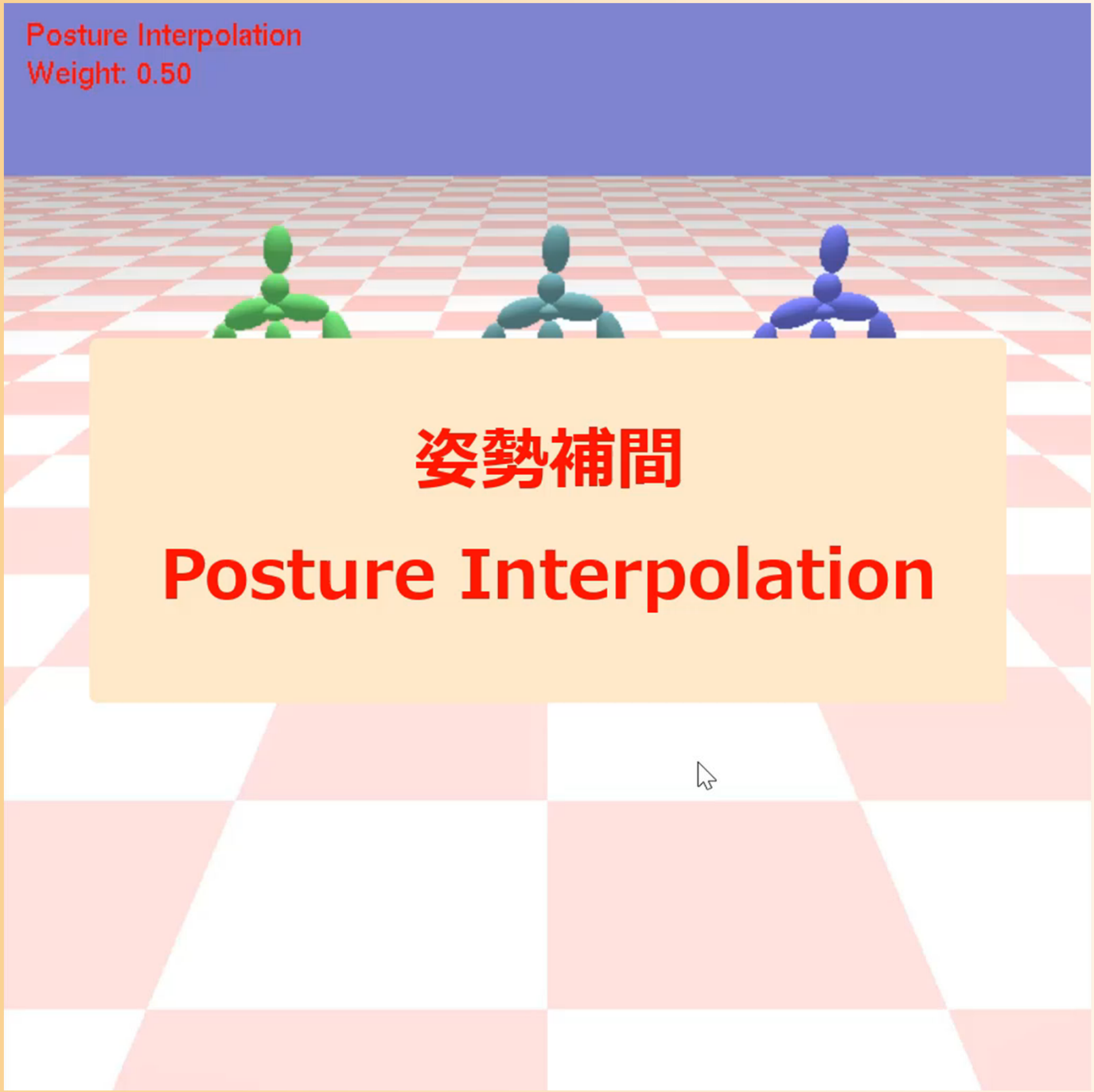
$$\mathbf{p} = w\,\mathbf{p}_1 + (1-w)\,\mathbf{p}_2$$

- 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 or 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
  - キー操作により、補間姿勢の水平位置の固定の有無を切替
  - 2つの姿勢の補間処理（各自作成）




Posture Interpolation
Weight: 0.50
姿勢補間
Posture Interpolation

姿勢補間アプリケーション

- PostureInterpolationApp（一部未実装）
  - 2つの入力姿勢を設定
    - 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つの入力姿勢を描画
  - 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（1）

- 姿勢補間
  - ２つの姿勢＋重みを入力、補間姿勢を出力

重み（0～1）の値に応じて、補間姿勢は p0～p1 の間で変化

```
// 姿勢補間（2つの姿勢を補間）
void  MyPostureInterpolation(
    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）

- 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

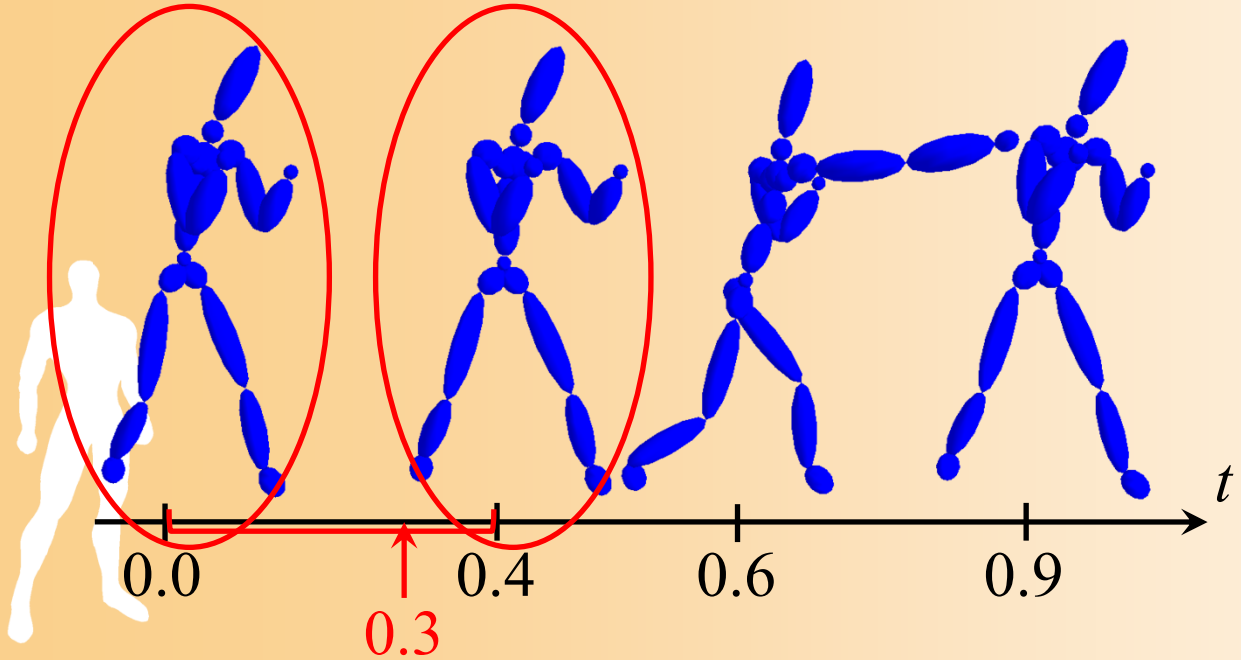
今日の内容

- 前回までの復習
- 姿勢補間
- キーフレーム動作再生
- 動作補間
- レポート課題(1)

キーフレーム動作再生

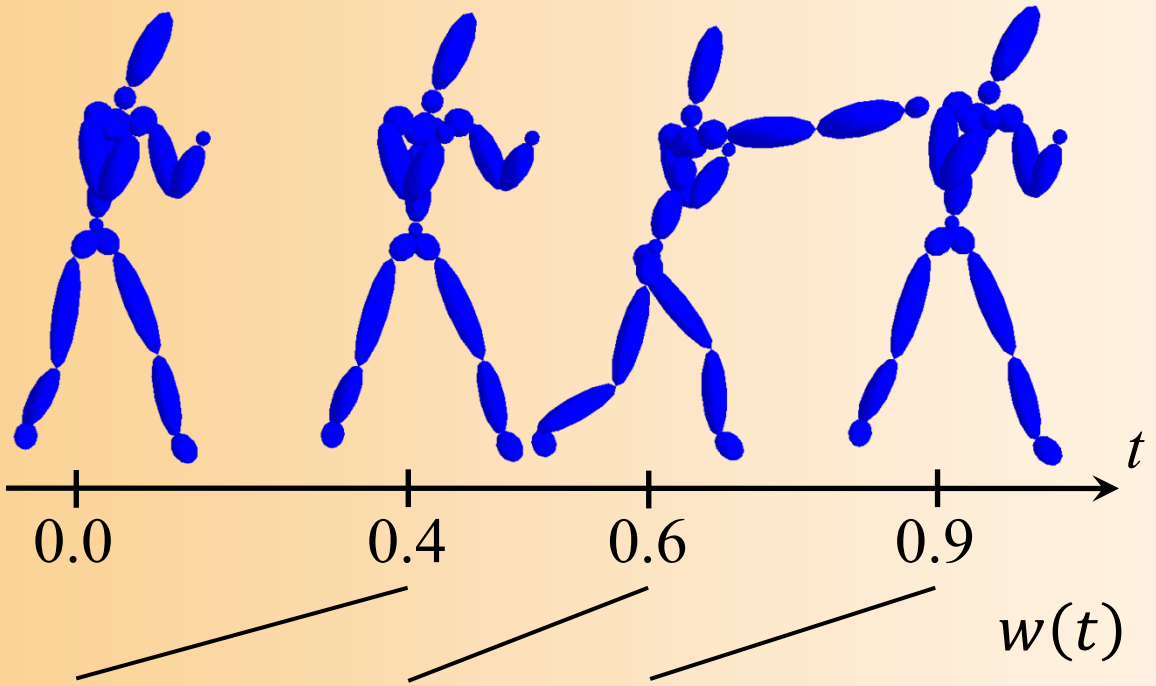
キーフレーム動作再生

- 複数の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



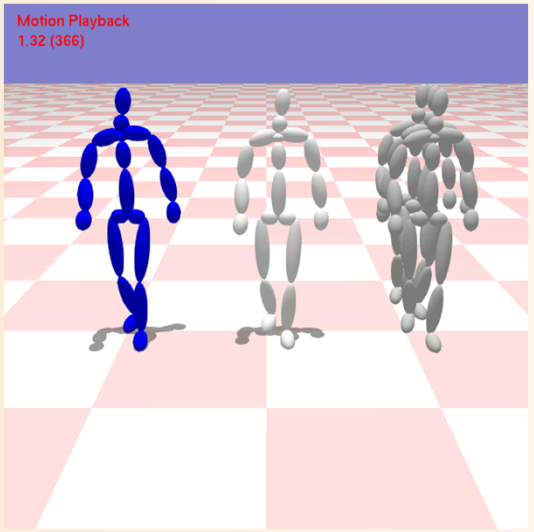
前後のキーフレーム姿勢の補間

- 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
    - キー操作で表示の有無を切替
  - キーフレーム動作からの姿勢取得（各自作成）



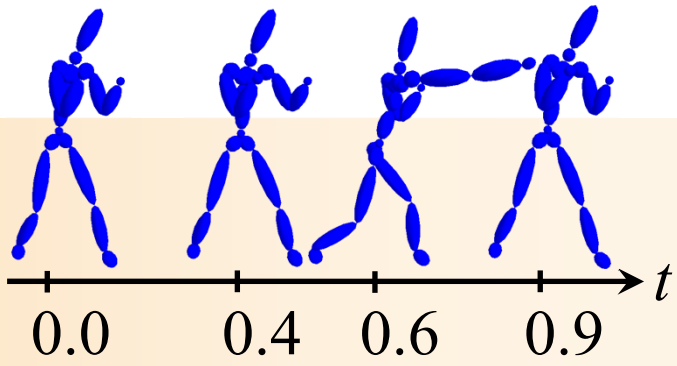

Keyframe Motion Playback
0.04 (238)
キーフレーム動作再生
Keyframe Motion Playback

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生成（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

```
// 人体モデルのキーフレーム動作を表すクラス
class  KeyframeMotion
{
    // 骨格モデル
    const 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;
};
```



キーフレーム動作の初期化

- 本プログラムでは、BVH動作からキー姿勢を抜き出して、キーフレーム動作を生成
  - キー姿勢の数や時刻は、プログラム内に記述

```
void  KeyframeMotionPlaybackApp::Initialize()
{
    const char *  sample_motions = "sample_walking1.bvh";
    const int  num_keytimes = 3;
    const float  sample_keytimes[ num_keytimes ] = { 2.35f, 3.00f, 3.74f };
    ...
    for ( int i = 0; i < num_keytimes; i++ )
        motion->GetPosture( sample_keytimes[ i ], key_poses[ i ] );
    ...
    keyframe_motion->Init( body, num_keytimes, &key_times.front(),
        &key_poses.front() );
}
```

キーフレームアニメーションのプログラミング(1)

- 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
    int  no = ???;
    // 指定時刻が動作の範囲外であれば終了

    // 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
    MyPostureInterpolation( ??? );
}
```

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、区間番号(0 ～ n-1)を取得

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（2）

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 クラスの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

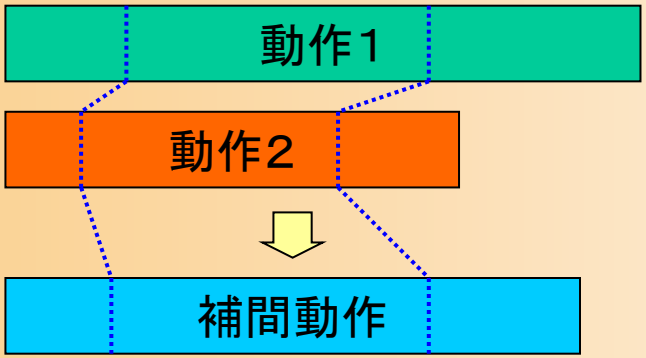
今日の内容

- 前回までの復習
- 姿勢補間
- キーフレーム動作再生
- 動作補間
- レポート課題(1)

動作補間

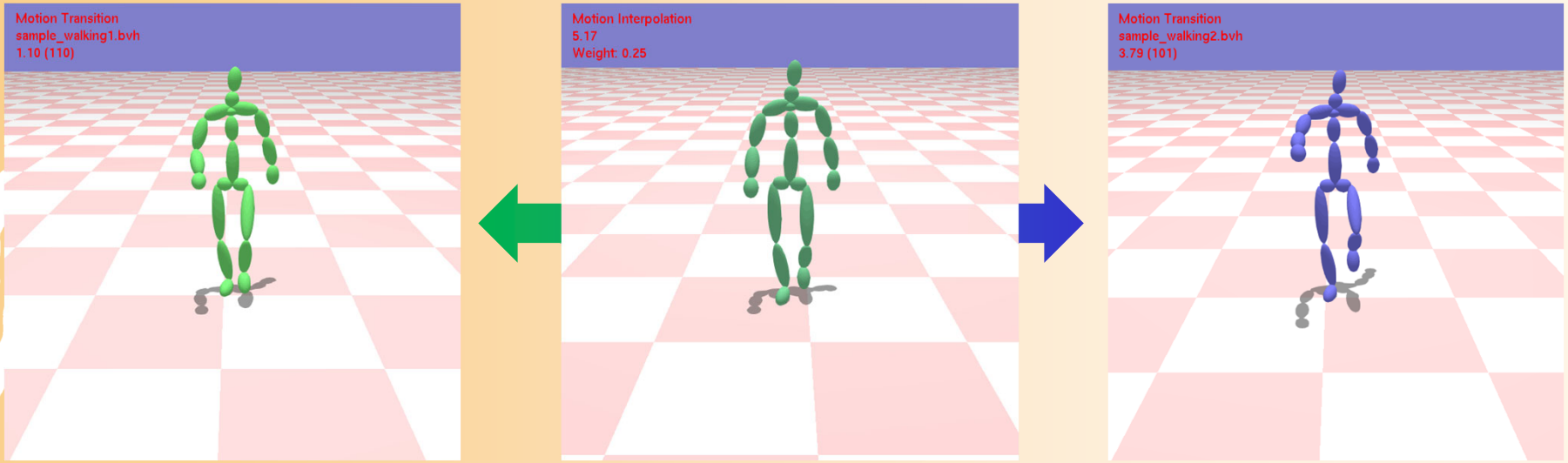
動作補間

- 動作補間（モーション・インターポレーション）
  - 複数の動作を混合して新しい動作を生成
    - 例：ゆっくり走る動作と早く走る動作から、中くらいの速度で走る動作を生成
    - 任意の比率で混合できる（全動作の重みの和を 1 にする）
  - 事前に複数の動作を同期しておく必要がある
    - 同じタイミング（足を着く・離すなど）にキー時刻を設定



デモプログラム

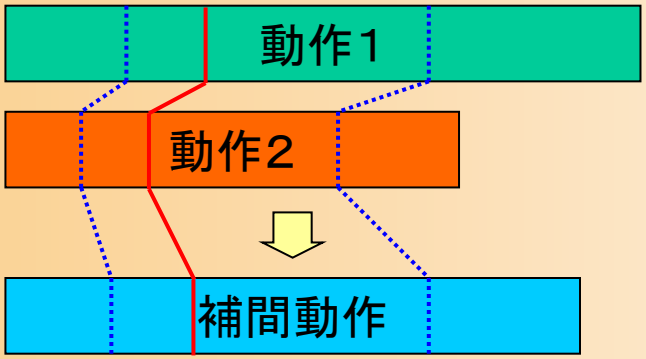
- 動作補間アプリケーション
  - 2つのサンプル動作を補間して再生
  - マウス操作(左右方向の左ドラッグ)に応じて動作補間の重みを変更(動作再生中も変更可能)
  - 動作補間処理




Motion Interpolation
2.79
Weight: 0.00
動作補間
Motion Interpolation

動作補間の計算方法

- 任意の時刻 t の姿勢 $\mathbf{p}$ を計算
  1. 時刻 t に対応する、各動作の時刻 t_i を決定
     - あらかじめ設定された区間・キー時刻情報から計算
  2. 各動作から、時刻 t_i の姿勢 $\mathbf{p}_i$ を取得
  3. 各姿勢 $\mathbf{p}_i$ を重み w_i に応じて補間
     - 3つ以上の動作を補間する場合には、3つ以上の回転の補間手法を用いる必要がある

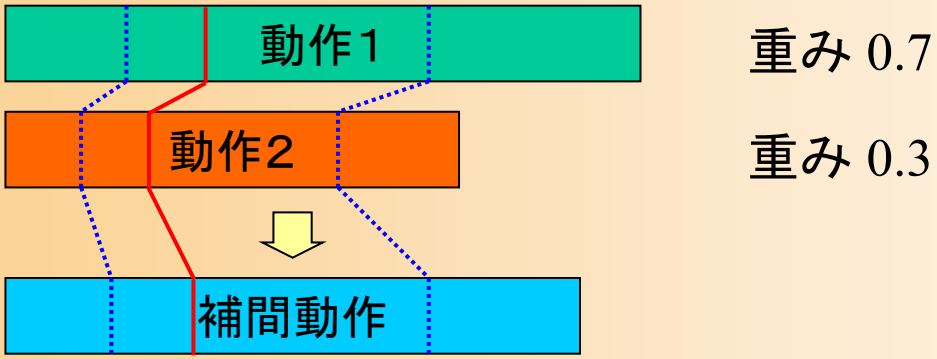


重み 0.7

重み 0.3

動作補間でのタイミングの計算

- 時刻 t に対応する、各動作の時刻 t_i を決定
  1. 補間動作（生成動作）のキー時刻を計算
     - サンプル動作のキー時刻を、現在の補間重みにもとづいて加重平均を取ることで計算
  2. 時刻 t に対応する、補間動作での区間と区間内での正規化時刻（0.0～1.0）を計算
  3. 各動作のキー時刻にもとづき、時刻 t_i を計算



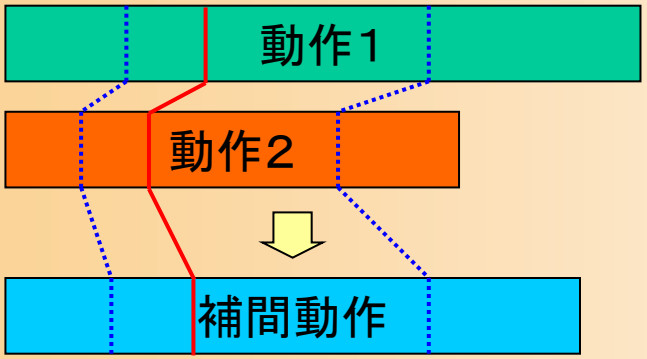
動作補間での姿勢の計算

- 各動作の姿勢 $\mathbf{p}_i$ を重み w_i に応じて補間
  - 動作数が2つの場合は、2つの姿勢に対する姿勢補間により計算

$$\mathbf{p}(t) = w_1\,\mathbf{P}_1(t_1) + w_2\;\mathbf{P}_2(t_2)$$

重みの和は 1 である（$w_1 + w_2 = 1$）ため、以下の姿勢補間で計算できる

$$\mathbf{p}(t) = w_1\,\mathbf{P}_1(t_1) + (1 - w_1)\,\mathbf{P}_2(t_2)$$

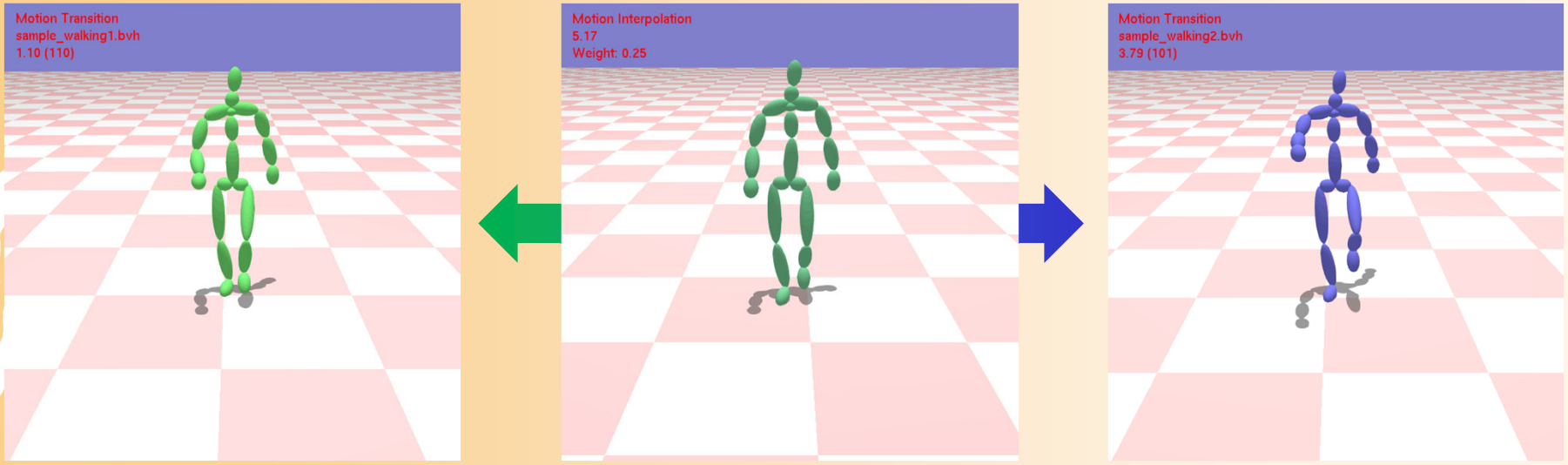


重み 0.7

重み 0.3

プログラミング演習

- 動作補間アプリケーション
  - 2つのサンプル動作を補間して再生
  - マウス操作(左右方向の左ドラッグ)に応じて動作補間の重みを変更(動作再生中も変更可能)
  - 動作補間処理(各自実装)



動作補間アプリケーション

- MotionInterpolationApp（一部未実装）
  - 2つのサンプル動作を補間して再生
    - 2つのサンプル動作（BVH動作）を読み込み・設定
    - 各動作の再生範囲・キー時刻を設定
  - マウス操作（左右方向の左ドラッグ）に応じて動作補間の重みを変更
  - 動作補間処理（各自実装）
    - Animation関数の一部を実装
    - 繰り返し再生には、動作接続・遷移の処理を利用
      - 動作接続・遷移の処理が未作成であれば、繰り返し動作の開始時に、位置が不連続になる

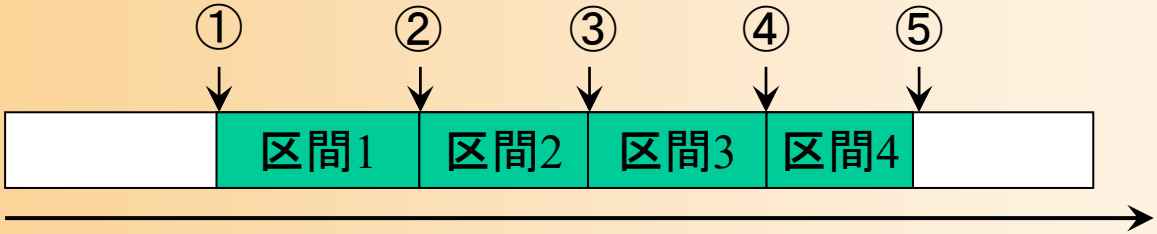
動作補間に用いる動作情報

- 動作のメタ情報を表す構造体

```
// 動作のメタ情報を表す構造体
struct  MotionInfo
{
    // 動作情報
    Motion *        motion;
    // 動作の開始・終了時刻（動作のローカル時間）
    float           begin_time;
    float           end_time;
    // 動作のブレンド区間の終了・開始時刻（動作のローカル時間）
    vector< float >  keytimes;
    ....
}
// 動作データのリスト（メンバ変数）
vector< MotionInfo * >  motion_list;
```

サンプル動作

- デモプログラム（サンプルプログラム）では、異なるスタイルの歩行動作を使用
  - 動作接続・遷移デモと同じサンプル動作を使用
  - 繰り返し歩行動作中の１サイクル分を使用
  - ５つのキー時刻を設定
    - 右足を上げ始める（動作開始）→ 右足を着く（ブレンド区間終了）→ 左足を上げ始める → 左足を着く（ブレンド区間開始）→ 右足を上げ始める（動作終了）



サンプル動作の読み込み

- LoadSampleMotions関数で読み込み
  - 動作に関する情報を関数内に記述
  - BVHファイルを読み込み、動作データの配列を出力

```
// サンプル動作セットの読み込み
const Skeleton *  LoadSampleMotions(
    vector< MotionInfo * > &  motion_list, const Skeleton * body )
{
    const int  num_motions = 3;
    const int  num_keytimes = 5;
    const char *  sample_motions[ num_motions ] = {
        "sample_walking1.bvh", "sample_walking2.bvh",
        "sample_walking3.bvh" };
    const float  sample_keytimes[ num_motions ][ num_keytimes ] = {
        { 2.35f, 3.00f, 3.08f, 3.68f, 3.74f },
        { 1.30f, 2.07f, 2.12f, 2.88f, 2.94f },
        { 1.20f, 2.00f, 2.08f, 2.80f, 2.86f } };
    ...
```

動作補間のプログラミング

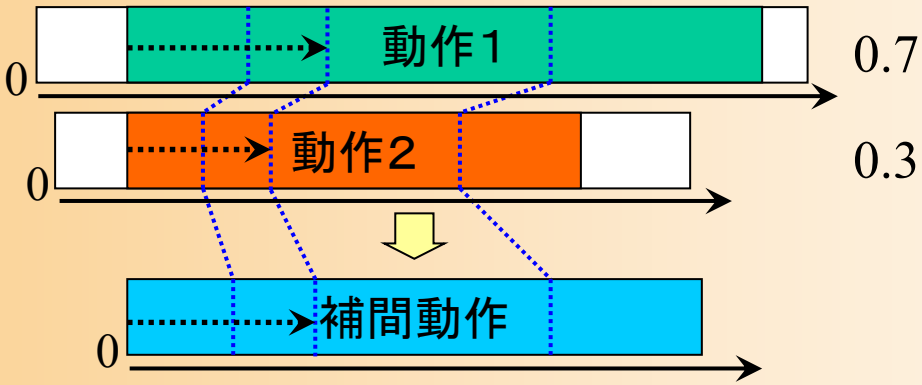
- 動作補間・再生

```
// 動作再生処理（動作補間＋動作接続）
void  MotionInterpolationApp::AnimationWithInterpolation( float delta )
{
    // 現在の重みでの補間動作のキー時刻を計算

    // アニメーションの時間を進める
    // 動作が終了したら、繰り返し再生のための各動作の接続処理

    // 現在の時刻に対応する区間・正規化時間（0.0～1.0）を計算

    // 各動作のローカル時間を計算し、動作の姿勢を取得

    // 各動作の姿勢を現在の重み（ブレンド比率）で姿勢補間
}
```

各自作成

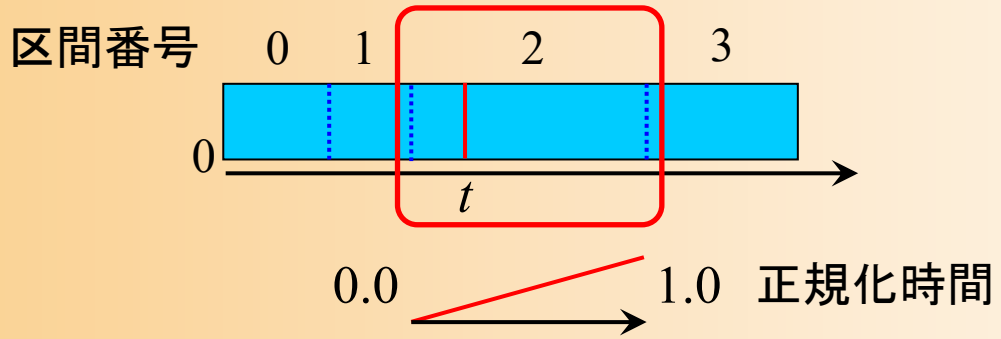
動作補間のプログラミング(1)

1. 現在の時刻に対応する区間番号と正規化時間を計算
    1. 補間動作の各キー時刻を計算(0.0 を開始時刻とする補間動作のキー時刻)
        - 各サンプル動作におけるキー時刻(開始時刻からの経過時間)を、重みに応じて補間することで計算
            - 計算結果を、ローカル変数(vector<float> keytimes)に格納
        - 本来は、補間の重みが変更された時にのみ計算すれば良い



動作補間のプログラミング(2)

1. 現在の時刻に対応する区間番号と正規化時間を計算
    2. 区間番号(0～キー区間数)と正規化時間(0.0～1.0)を計算
        - 上で求めた補間動作の各キー時刻(vector<float> keytimes)にもとづいて、現在時刻 t に対応する区間番号(seg_no)と区間内での正規化時間(seg_time)を計算する



動作補間のプログラミング（3）

2. 各サンプル動作の時間を計算して、姿勢を取得
   - 区間番号（seg_no）と正規化時間（seg_time）と、サンプル動作のキー時刻から、各サンプル動作の現在時刻 motion_time[i] を計算
   - 各サンプル動作の姿勢 motion_posture[i] を取得
3. 各サンプル動作の姿勢を、現在の重み（ブレンド比率）（weight） で姿勢補間
   - 前に作成した2つの姿勢の補間の関数を呼び出し

$$\mathbf{p}(t) = w_1\,\mathbf{P}_1(t_1) + (1 - w_1)\,\mathbf{P}_2(t_2)$$

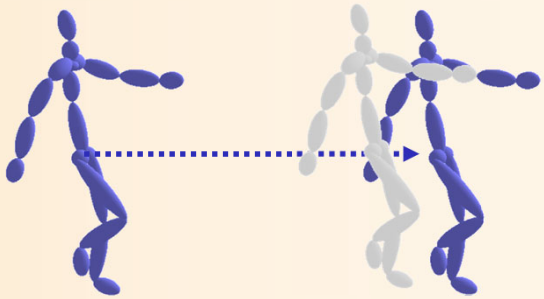
動作補間の改良・拡張

- ここまでは、2つの動作の単純な補間のみ
- 実際には、様々な改良・拡張が必要となる

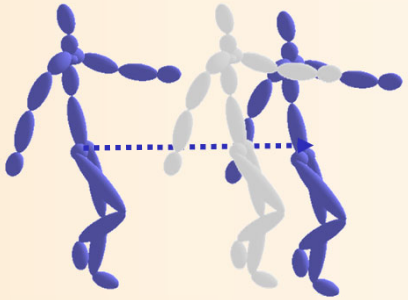
- 動作補間中の重みの変更への対応
- 動作補間のタイミングの計算（どの時刻の姿勢同士を補間するか）
- 複数（3つ以上）の動作の補間
- 特徴パラメタを使った動作補間の重みの計算

動作補間中の重みの変更(1)

- 動作補間中に重みを変更すると、腰の位置・向きが不連続になる可能性がある
  - 重みが変わると、動作開始時からの移動・回転量が大きく変わる可能性がある
    - 特に、歩行などの移動を含む動作で、サンプル動作によって移動距離が大きく異なる場合



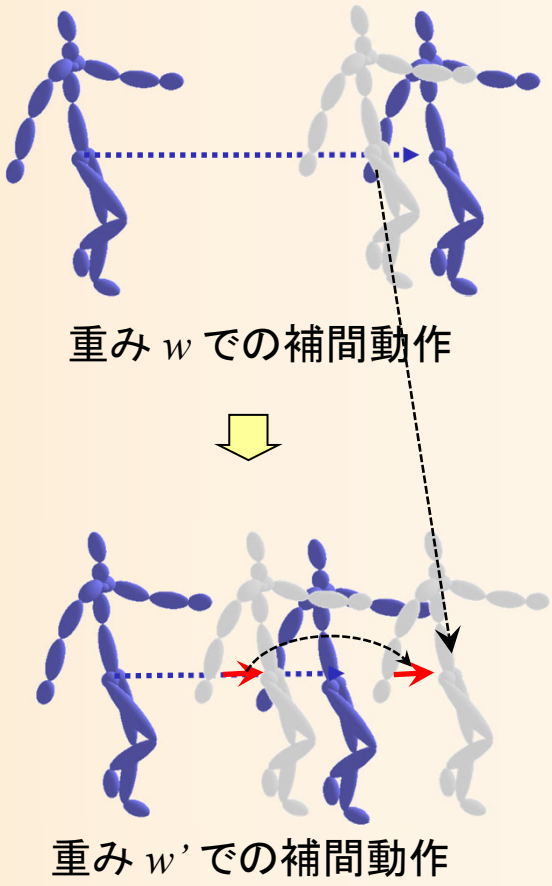
重み w での補間動作



重み w' での補間動作

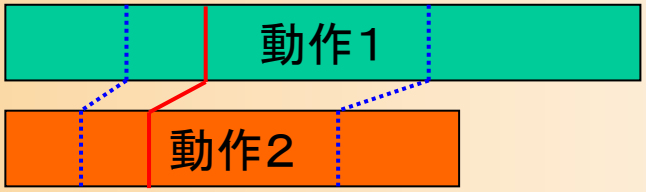
動作補間中の重みの変更(2)

- 動作補間中に重みを変更すると、腰の位置・向きが不連続になる可能性がある
- 現在の重みでの補間動作における前フレームからの腰の移動・回転量を計算して、前の姿勢の位置・向きに適用すれば、対応可能



動作補間でのタイミングの計算

- どの時刻の姿勢同士を補間するか
- 時刻 t に対応する、各動作の時刻 t_i を決定
  - 自然な合成動作を生成するためには、正規化時間から各動作の時刻を決めるのではなく、姿勢が近くなるタイミングを選択した方が良い
  - Dynamic Time Warping により、動作間の姿勢が近くなるような時間対応を求めることができる

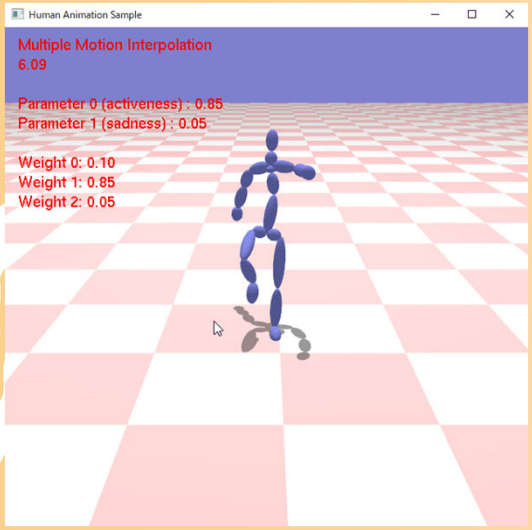


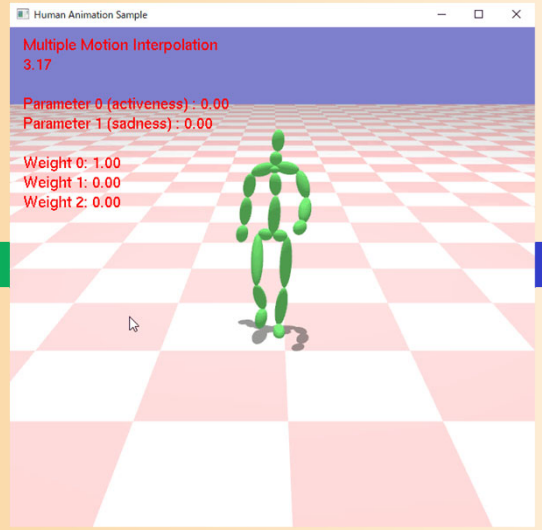
複数の動作の補間

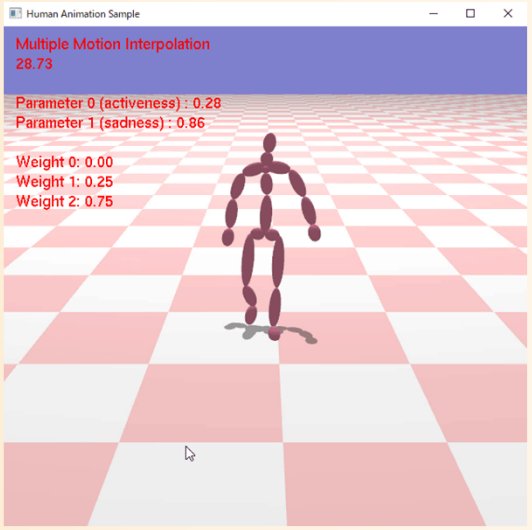
- 基本的な動作補間の処理は、補間する動作の数に関わらず共通
- 最終的に各サンプル動作から取得した姿勢を補間する際に、複数の姿勢（回転）の補間が必要となる
  - 四元数を使った球面線形補間（SLERP）は、2つの回転の補間にしか適用できない
- 複数の回転の補間を実現する方法
  - 回転を対数ベクトルに変換して補間

デモプログラム

- 複数動作補間アプリケーション
  - 3つのサンプル動作を補間して再生
  - マウス操作（左ドラッグ）に応じて動作補間の重みを変更（動作再生中も変更可能）
    - 2次元のパラメタ（activeness, sadness）を操作







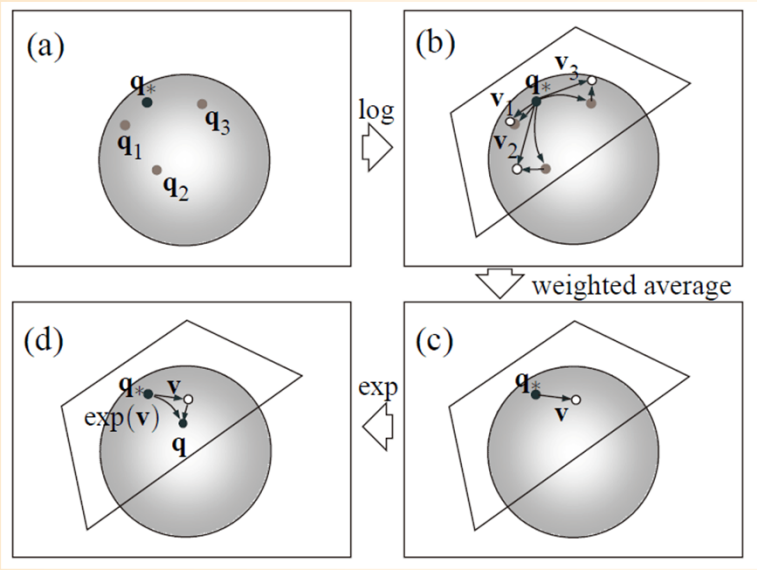
デモプログラム

- 複数動作補間アプリケーション
  - 3つのサンプル動作を補間して再生
    - 各サンプル動作に、キー時刻や2次元の特徴量（activeness, sadness）の情報を設定しておく
  - マウス操作（左ドラッグ）に応じて動作補間の重みを変更（動作再生中も変更可能）
    - 2次元のパラメタ（activeness, sadness）を操作
    - 2次元のパラメタを満たすための、動作補間の重みを計算（回帰モデル）
  - 複数姿勢補間


Multiple Motion Interpolation
0.10
Parameter 0 (activeness) : 0.00
Parameter 1 (sadness) : 0.00
Weight 0: 1.00
Weight 1: 0.00
複数動作補間
Multiple Motion Interpolation

対数表現による補間（復習）

- 四元数を対数ベクトル表現に変換
- 対数ベクトルの線形補間により、複数の回転を補間できる
  - 単純に補間すると誤差が大きくなる
  - 平均回転 q* を求めて、それと各回転の差分を表すベクトルを補間



Sang Il Park, Hyun Joon Shin, Sung Yong Shin, "On-line locomotion generation based on motion blending", ACM SIGGRAPH Symposium on Computer Animation 2002, pp. 105-111, 2002.

動作補間の重みの決定(1)

- 特にサンプル動作が多数ある場合、希望する補間動作を生成するための適切な重みを設定することは難しい
- 各動作の重み w_i を、何らかの特徴量(速さ、距離、方向、高さなど)から決定できる
  - 回帰モデルの利用
  - あらかじめ、n 個の動作データのそれぞれに、k 次元の特徴パラメタ $\mathbf{f}_i$ を設定しておく($k \leqq n$)

動作補間の重みの決定(2)

- 各動作の重み w_i を、何らかの特徴量(速さ、距離、方向、高さなど)から決定できる
  - 回帰モデルの利用
  - あらかじめ、n 個の動作データのそれぞれに、k 次元の特徴パラメタ $\mathbf{f}_i$ を設定しておく($k \leqq n$)
  - 動作補間時に k 次元の特徴パラメタ $\mathbf{f}$ が指定されると、それを満たすような重み $\mathbf{w}$ を計算
    - 全動作の重み $\mathbf{w}$ の和が 1.0 になるようにする
    - 各動作の重みは 0.0〜1.0 の範囲とする

$$\mathbf{f} = \begin{pmatrix} \mathbf{f}_1 \\ \cdots \\ \mathbf{f}_n \end{pmatrix} \mathbf{w} \quad \xrightarrow[\text{逆変換を求める}]{} \quad \mathbf{w} = \begin{pmatrix} \mathbf{f}_1 \\ \cdots \\ \mathbf{f}_n \end{pmatrix}^{+} \mathbf{f}$$

動作補間の重みの計算(3)

- 線形・非線形変換の組み合わせによる方法
  - 線形変換を最小二乗法により計算
    - 単純な方法としては、擬似逆行列を使って計算可能
  - 各サンプルの近くでは、そのサンプルの重みが大きくなるように、非線形の補正を適用
    - Radial Basis Function（放射基底関数）が一般的に用いられる

$$\mathbf{w}_i = \sum_{j=0}^{M} a_{ij} A_j(\mathbf{p}) + \sum_{j=0}^{N} r_{ik} R_k(\mathbf{p})$$

$A_j(\mathbf{p})$: 線形基底　　a_{ij}: 線形係数

$R_k(\mathbf{p})$: 非線形基底　r_{ik}: 非線形係数

C. Rose, M. F. Cohen, B. Bodenheimer, **"Verbs and Adverbs: Multidimensional Motion Interpolation"**, *IEEE Computer Graphics and Applications*, Vol.18, No.5, pp.32-40, 1998.

今日の内容

- 前回までの復習
- 姿勢補間
- キーフレーム動作再生
- 動作補間
- レポート課題(1)

レポート課題（1）

レポート課題

- キャラクタ・アニメーション（1）
  - サンプルプログラムの未実装部分（前半）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. 動作補間
  - サンプルプログラムの未実装部分（後半）は次の課題
    5. 動作接続・遷移
    6. 動作変形
    7. 逆運動学計算（CCD法）

レポート課題間の関連

- 前の課題で作成した処理を、次の課題でも使用

1. 順運動学計算
2. 姿勢補間
3. キーフレーム動作再生
4. 動作補間
5. 動作接続・遷移
    1. 動作接続のみ
    2. 動作接続・遷移
6. 動作変形
7. 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

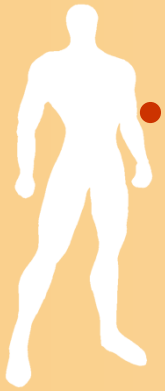
レポート課題

- 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をもとに作成したプログラム（???App.cpp）を提出
    - 他の変更なしのソースファイルやデータは、提出する必要はない
  - 全ての課題が終わらない場合は、できた部分だけでも提出する（ある程度できていれば合格点とする）
    - 作成しやすい課題から、任意の順番で作成して良い
    - できなかった課題の項目は、レポートのテンプレートから削除すること
  - Moodleの本講義のコースから提出
  - 締切：Moodleの提出ページを参照

レポート課題 提出方法

Moodleから、以下のファイルを提出

- 作成したプログラム（テキスト形式）
  - ForwardKinematicsApp.cpp
  - PostureInterpolationApp.cpp
  - KeyframeMotionPlaybackApp.cpp
  - MotionInterpolationApp.cpp
- 変更箇所のみを抜き出したレポート（PDF）
  - Moodle に公開している LaTeX のテンプレートをもとに、作成する



レポート課題 演習問題

- レポート課題の提出に加えて、レポート課題の理解度を確認するための Moodle 演習問題にも解答する
  - 解答締切は、レポート提出と同じ
    - 締切までは解答を変更可、締切後に正解が表示
  - レポート課題のヒントにもなっているので、レポート課題で分からない箇所がえれば、演習問題の説明・選択肢を参考にして考えても良い
  - 本演習問題の点数は、演習課題の成績の一部として考慮する

まとめ

- 前回までの復習
- 姿勢補間
- キーフレーム動作再生
- 動作補間
- レポート課題(1)

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
- 姿勢補間、キーフレームアニメーション、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