コンピュータアニメーション特論

第11回　キャラクタアニメーション（4）

九州工業大学 情報工学研究院　尾下真樹

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
- 動作補間の改良・拡張
- 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、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

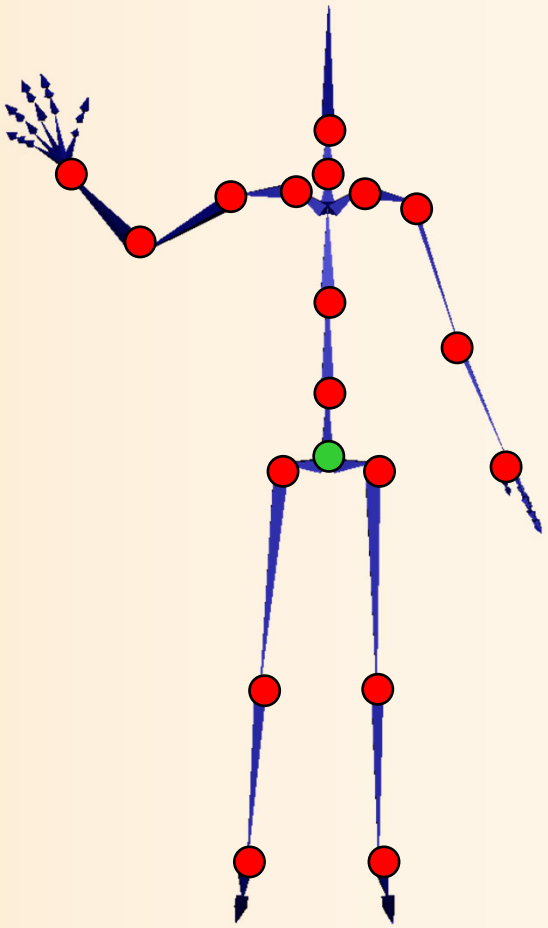
今日の内容

- 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
- 動作補間の改良・拡張
- レポート課題（1）

前回までの復習

骨格・姿勢・動作の表現

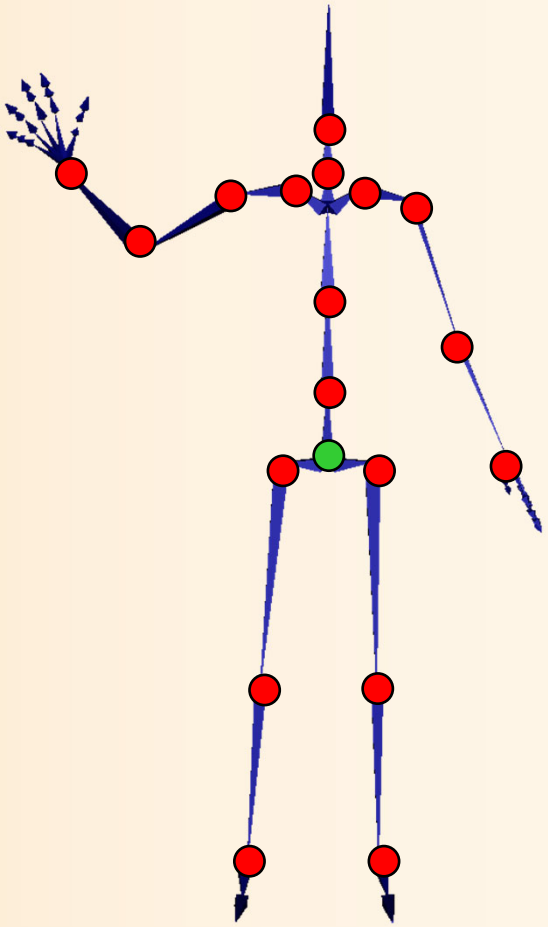
- 人体の骨格の表現
  - 多関節体モデルによる表現
  - 複数の体節と関節
    - 関節は２つの体節の間を接続
- 姿勢の表現
  - 全関節の回転 ＋ 腰の位置・向き
- 動作の表現
  - 姿勢の時間変化
    - 一定間隔 or キーフレーム動作データ



骨格・姿勢・動作のデータ構造

- 骨格・姿勢・動作の
  構造体・クラスの定義
  （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

// 人体モデルの動作を表すクラス
class  Motion
```

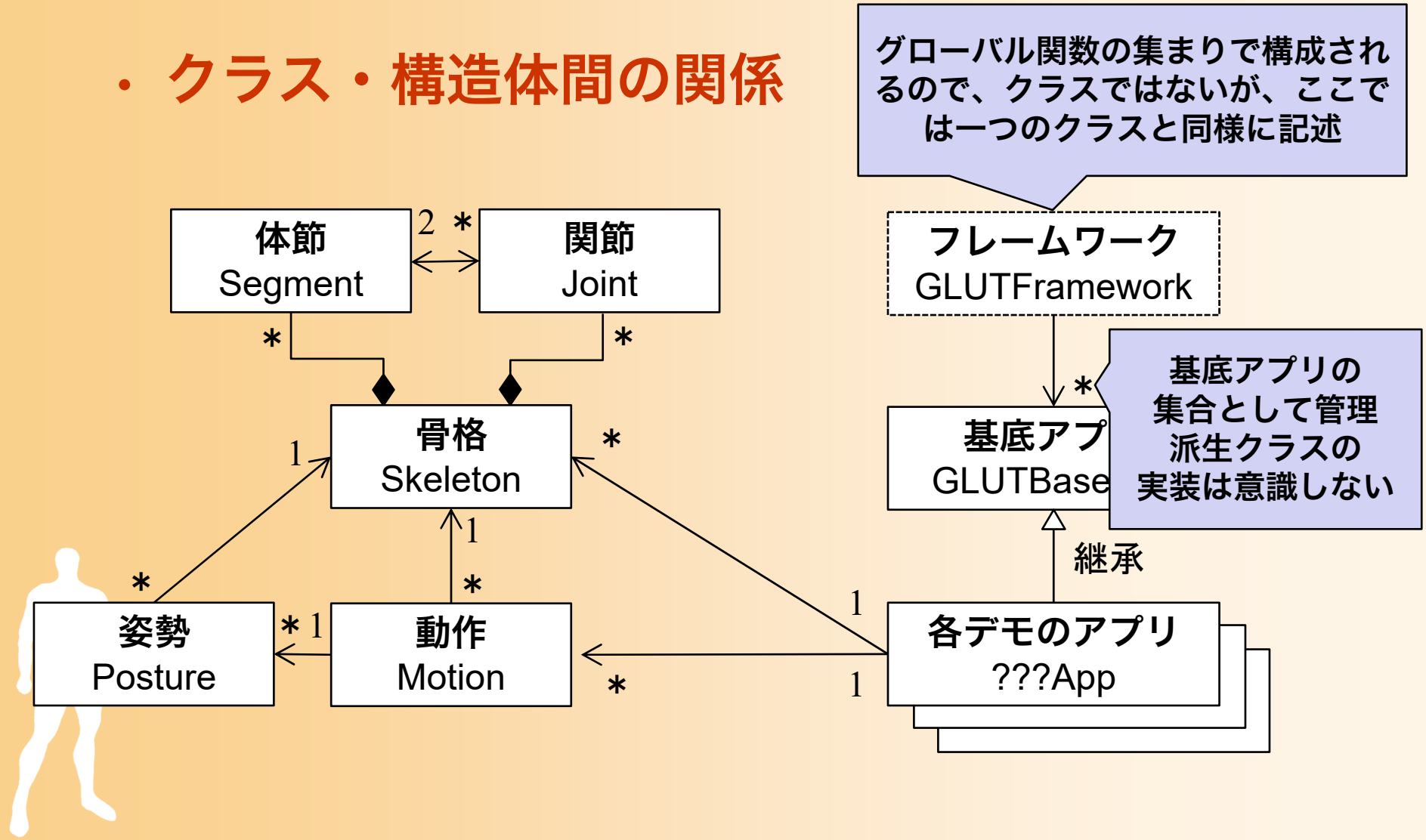


サンプルプログラム

- デモプログラムの一部のサンプルプログラム
  - 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とGLUTコールバック関数（SimpleHumanGLUT.h/cpp）
    - アプリケーションの基底クラス GLUTBaseAppの定義・実装
      - 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
    - 複数のアプリケーションの管理と、現在の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）

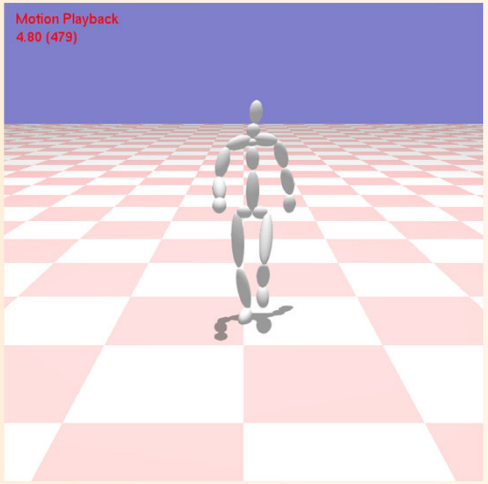
クラス図

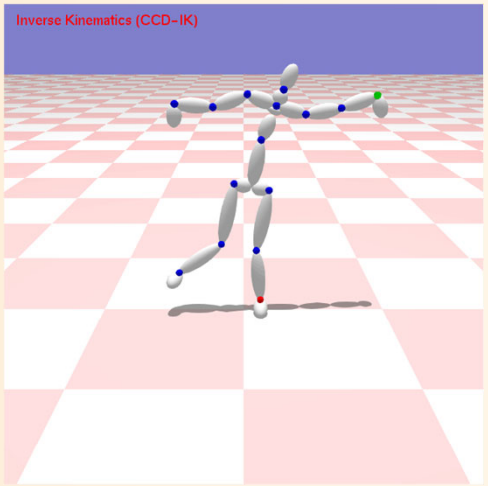
・クラス・構造体間の関係



サンプルプログラム

- 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）





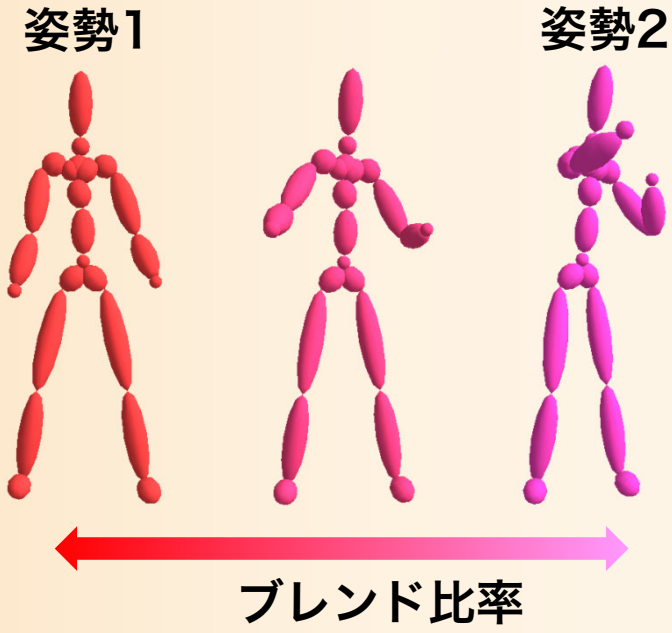
前回までの内容

- 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

姿勢補間

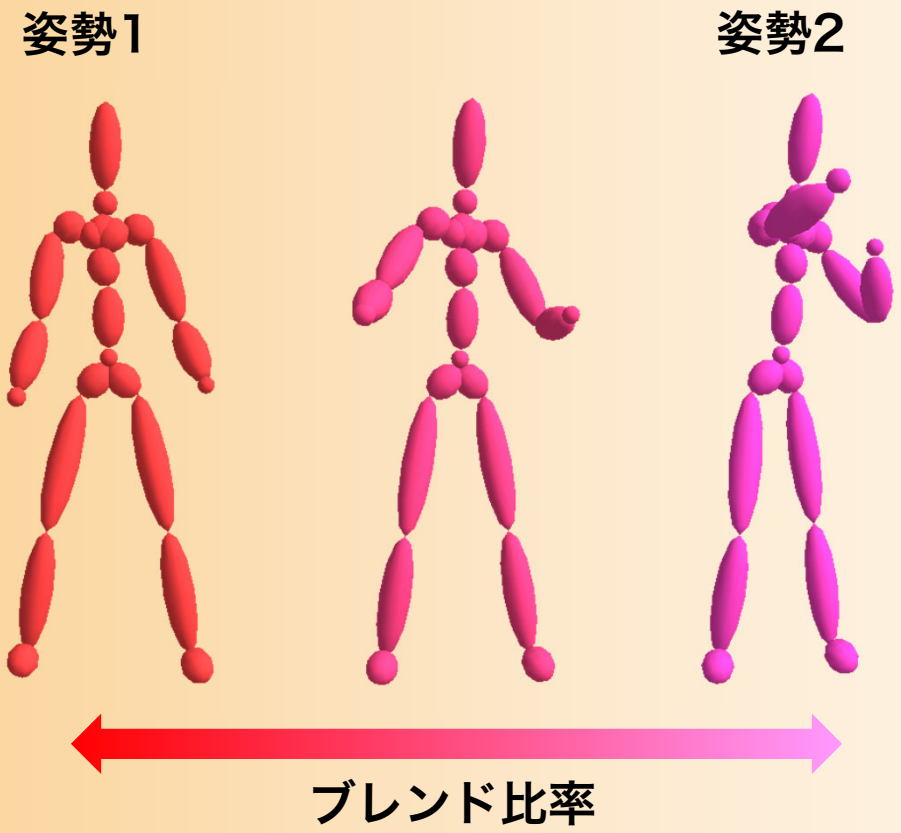
姿勢補間

- 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

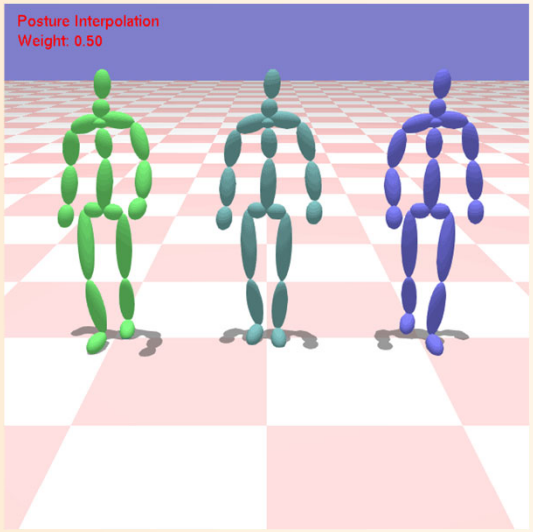
- ２つの入力姿勢を指定された重み（比率）で補間（混合）して、新しい姿勢を生成

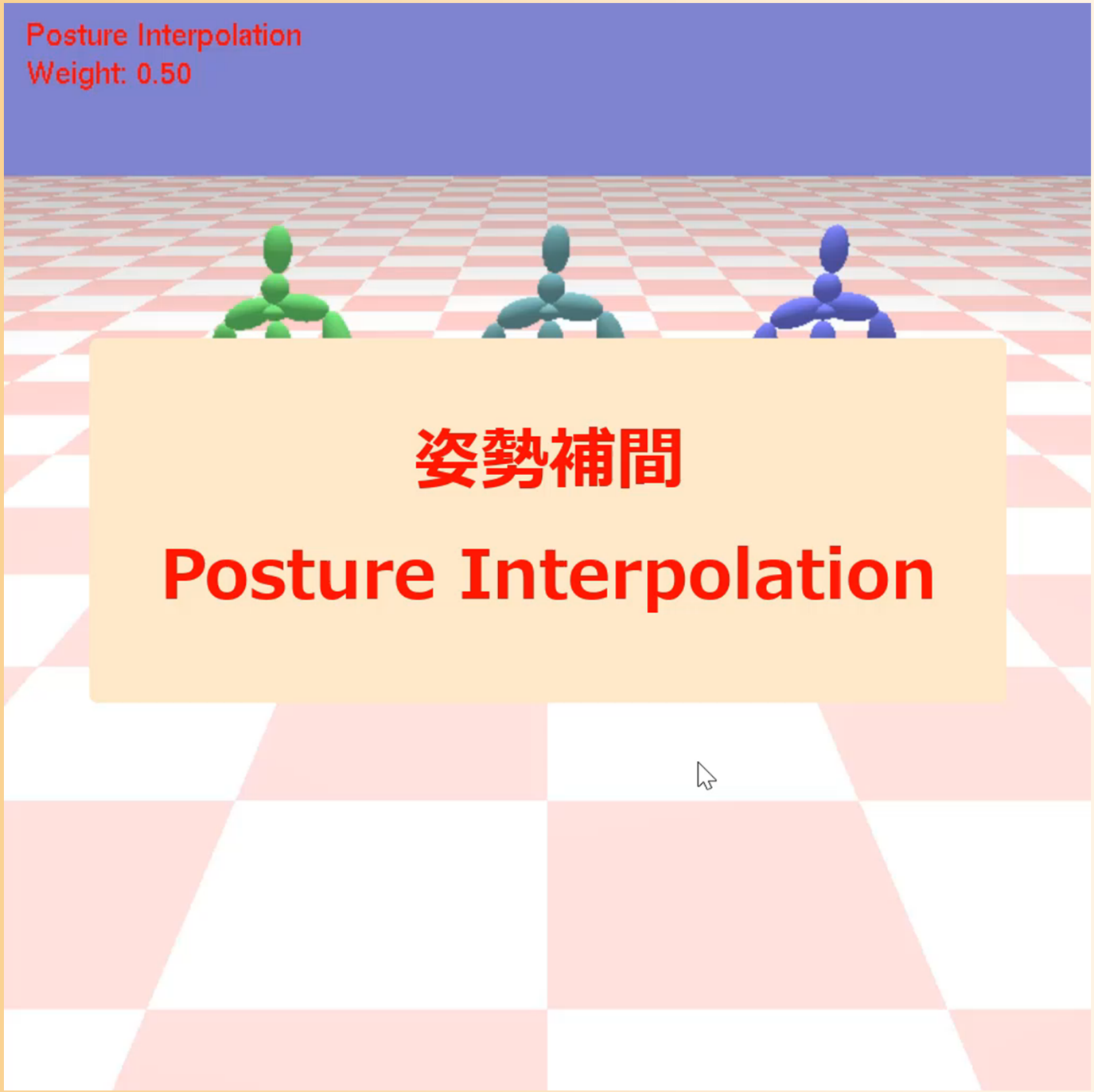
$$\mathbf{p} = w\,\mathbf{p}_1 + (1-w)\,\mathbf{p}_2$$

- 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

姿勢補間アプリケーション（2）

- クラス定義（PostureInterpolationApp）
  - ２つのサンプル姿勢と姿勢補間の重み
  - 補間結果の姿勢

```
class  PostureInterpolationApp : public GLUTBaseApp
{
  protected:
    // サンプル姿勢
    Posture *  posture0;
    Posture *  posture1;
    // 姿勢補間の重み
    float  weight;

    // キャラクタの姿勢
    Posture *  curr_posture;
```

０ ～ １ の範囲で表す
０ のときに補間結果の姿勢は posture0
１ のときに補間結果の姿勢は posture1
になる

姿勢補間アプリケーション（3）

- メンバ関数（PostureInterpolationApp）
  - 初期化処理で、サンプル姿勢の読み込み
  - 開始処理で、姿勢補間の重みの初期化
  - マウスドラッグ処理で、姿勢補間の重みの変更と補間結果の姿勢の計算（MyPostureInterpolation関数）
  - 描画処理で、補間姿勢（+サンプル姿勢）の描画

```
class  PostureInterpolationApp : public GLUTBaseApp
{
    virtual void  Initialize();
    virtual void  Start();
    virtual void  Display();
    virtual void  MouseDrag( int mx, int my );
    virtual void  Keyboard( unsigned char key, int mx, int my );
```

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

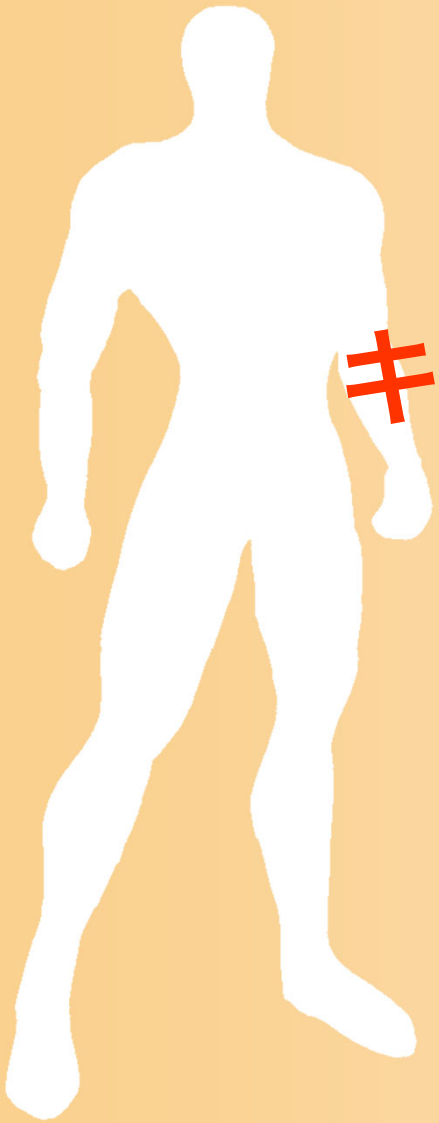
- 向き・回転の補間（腰の向き・関節の回転）
  - 四元数を使った球面線形補間（SLERP）
    - vecmathのクラス・メソッドを使って計算可能
    - キーフレームアニメーションの講義の説明を参照
- 位置の補間（腰の位置）
  - 線形補間

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

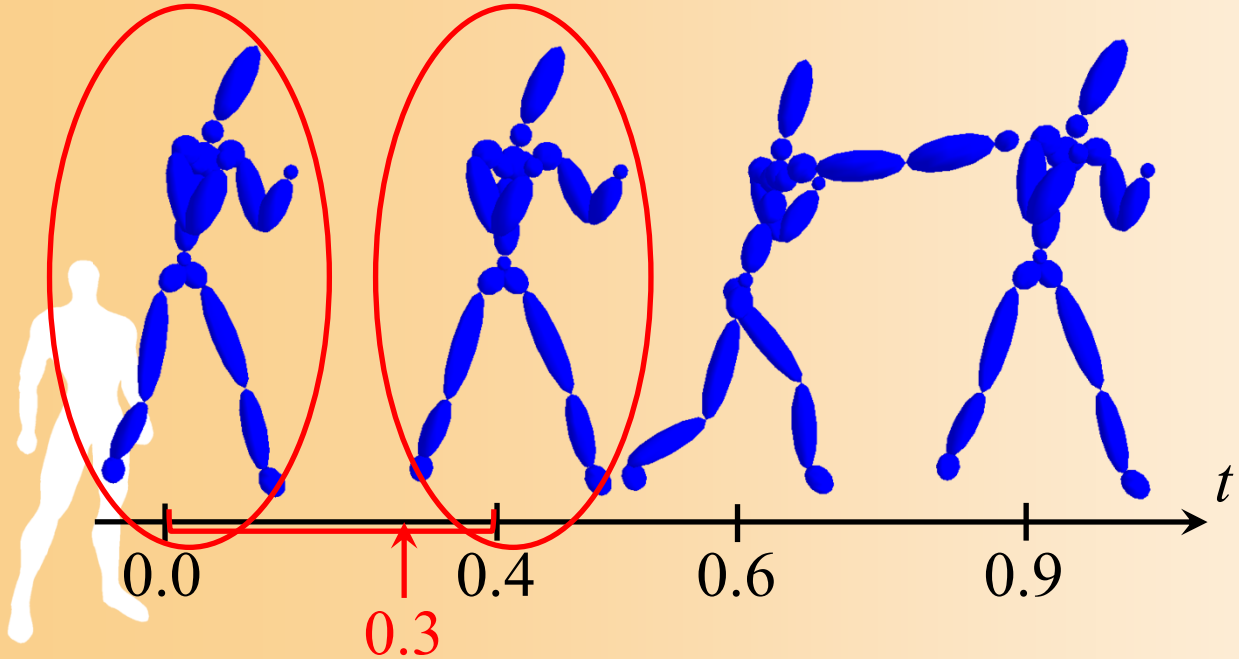
- 前回までの復習
- 姿勢補間
- キーフレーム動作再生
- 動作補間
- 動作補間の改良・拡張
- レポート課題（1）



キーフレーム動作再生

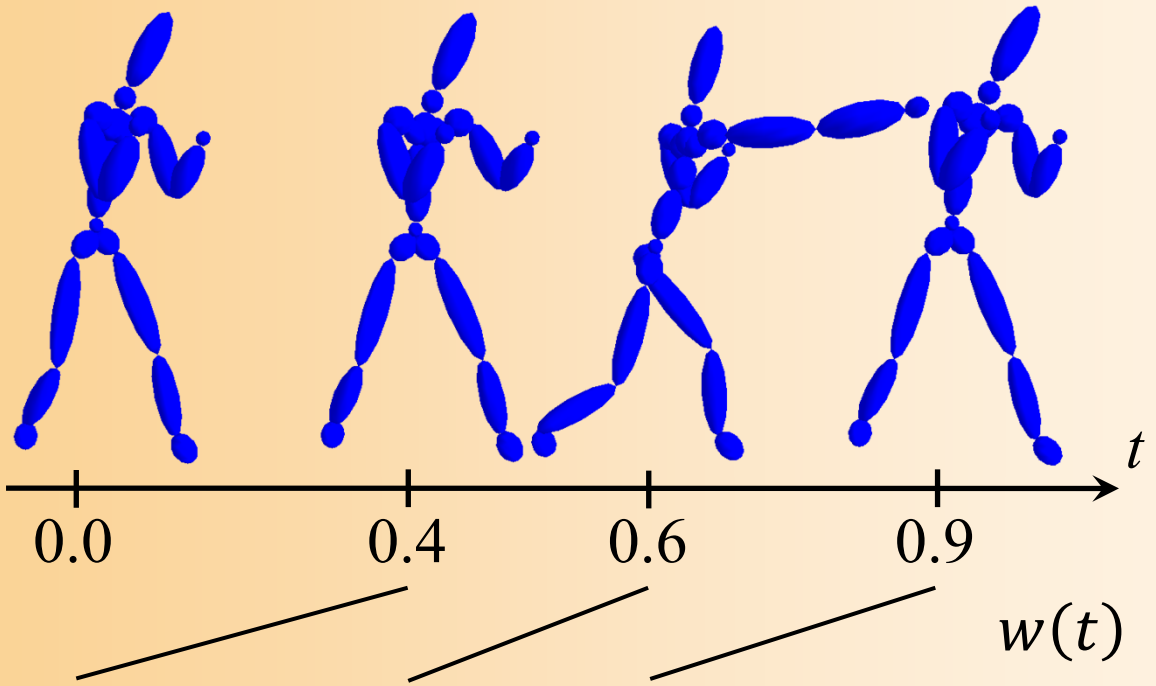
キーフレーム動作再生

- 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



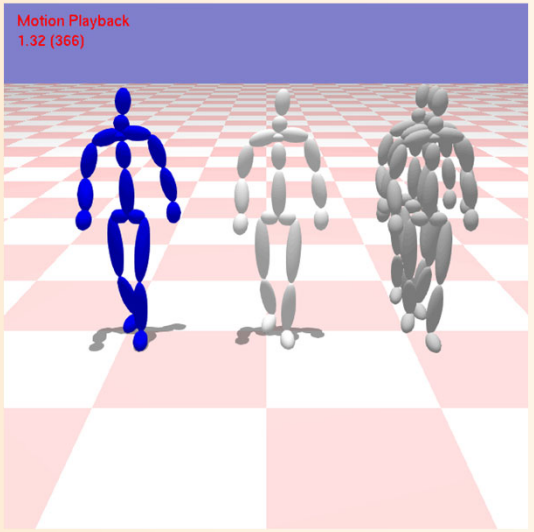
前後のキーフレーム姿勢の補間

- 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

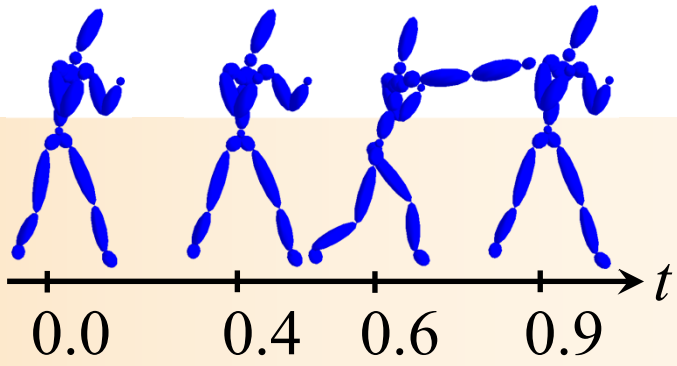
キーフレーム動作再生アプリケーション（2）

- クラス定義（KeyframeMotionPlaybackApp）
  - MotionPlaybackApp から派生
    - 比較用に元の一定間隔動作データも再生するため
  - メンバ変数として、キーフレーム動作データや、キーフレーム動作からの取得姿勢を定義

```
class  KeyframeMotionPlaybackApp : public MotionPlaybackApp
{
  protected:
    // キーフレーム動作データ
    KeyframeMotion *  keyframe_motion;

    // キーフレーム動作からの取得姿勢
    Posture *  keyframe_posture;
```

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;
};
```



キーフレーム動作再生アプリケーション（3）

- メンバ関数（KeyframeMotionPlaybackApp）
  - 基本的な処理は動作再生アプリケーションと同様
    - 一定間隔動作データの再生は基底クラスの処理を呼び出し
  - 初期化処理で、キーフレーム動作の初期化
  - アニメーション処理で、現在時刻の姿勢を取得（GetKeyframeMotionPosture関数）
  - 描画処理で、現在時刻の姿勢を描画


```
class  KeyframeMotionPlaybackApp : public MotionPlaybackApp
{
    virtual void  Initialize();
    virtual void  Display();
    virtual void  Keyboard( unsigned char key, int mx, int my );
    virtual void  Animation( float delta );
```

キーフレーム動作再生アプリケーション（4）

- 他のアプリケーションとの関係
  - キーフレーム動作からの姿勢の計算に、姿勢補間関数を使用
    - MyPostureInterpolation関数
    - 先に姿勢補間アプリケーションを作成する必要がある

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

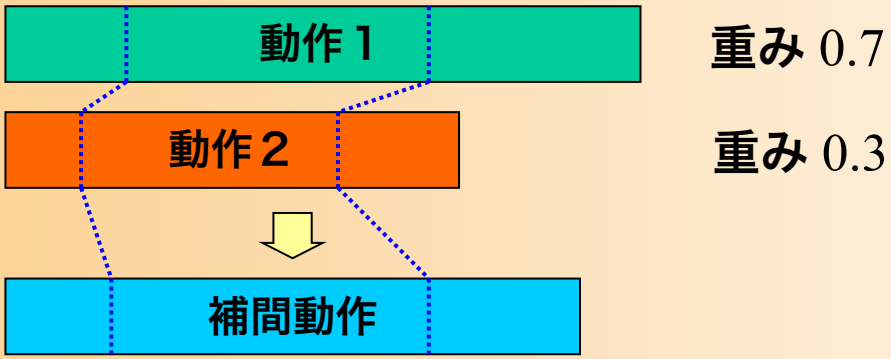
今日の内容

- 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
- 動作補間の改良・拡張
- レポート課題（1）

動作補間

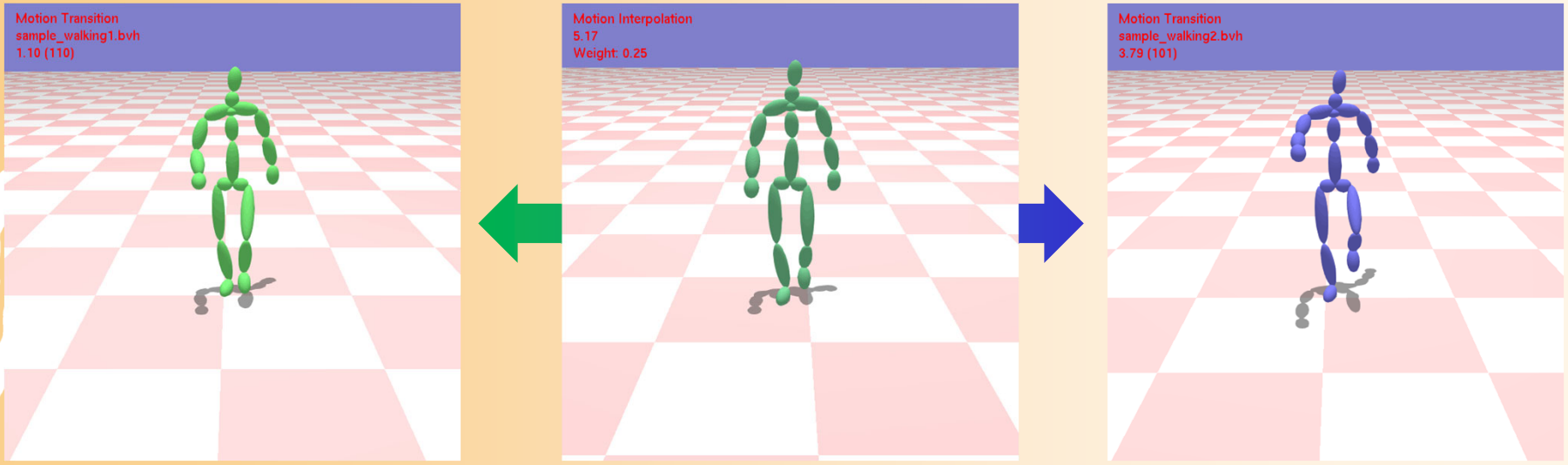
動作補間

- 動作補間（Motion Interpolation）
  - 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

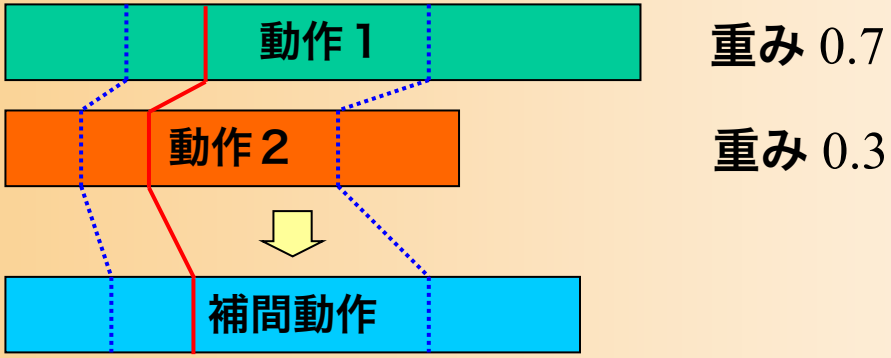
- 動作補間アプリケーション
  - ２つのサンプル動作を補間して再生
  - マウス操作（左右方向の左ドラッグ）に応じて動作補間の重みを変更（動作再生中も変更可能）
  - 動作補間処理




Motion Interpolation
2.79
Weight: 0.00
動作補間
Motion Interpolation

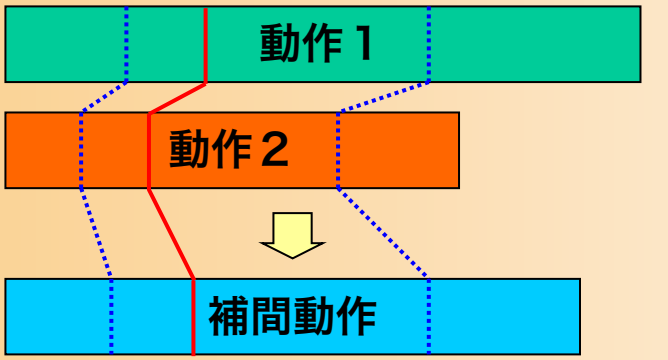
動作補間の計算方法

- 任意の時刻 t の姿勢 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 p_i を取得
  3. 各姿勢 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



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



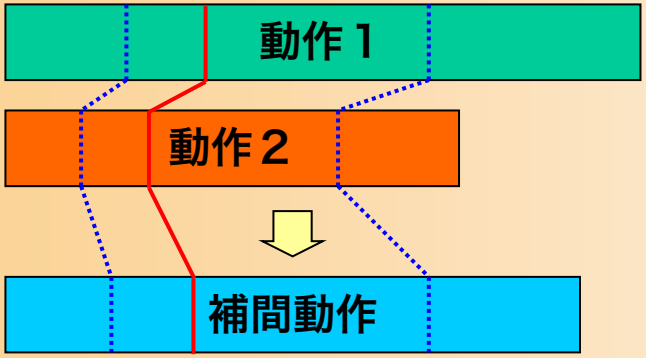
動作補間での姿勢の計算

- 各動作の姿勢 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

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

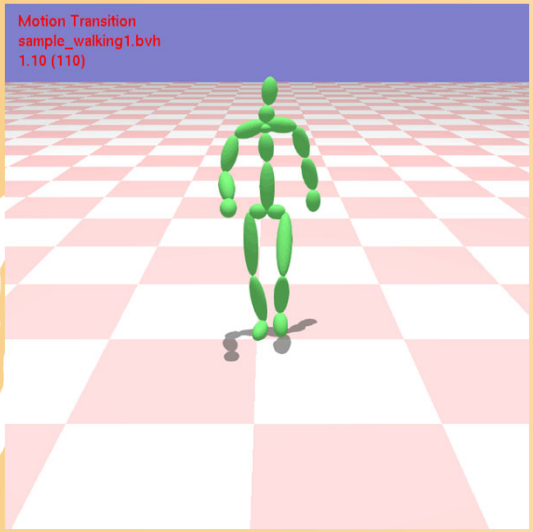


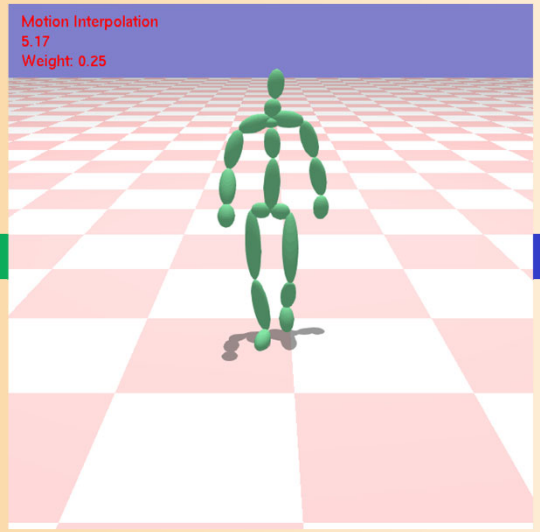
重み 0.7

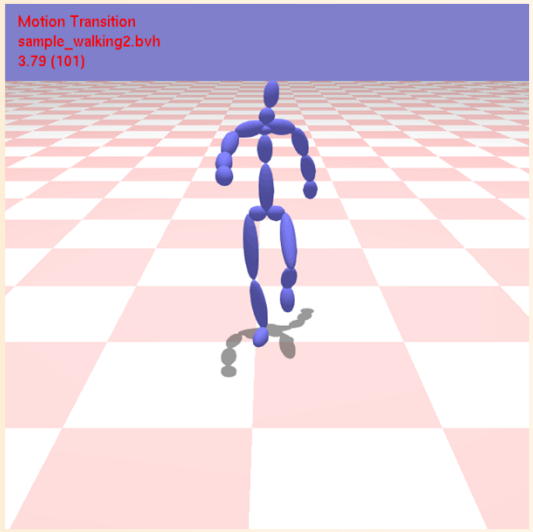
重み 0.3

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

動作補間アプリケーション（2）

- クラス定義（MotionInterpolationApp）
  - ２つのサンプル動作と動作補間の重み
  - 動作補間の結果の現在時刻の姿勢

```
class  MotionInterpolationApp : public GLUTBaseApp
{
  protected:
    // 動作データ情報（動作補間に用いる2つの動作）
    MotionInfo * motions[ 2 ];

    // 動作補間の重み
    float  weight;

    // キャラクタの姿勢
    Posture *  curr_posture;
```


０ ～ １ の範囲で表す
0 のときに補間結果の動作は motions[0]
1 のときに補間結果の動作は motions[1]
になる

動作補間アプリケーション（3）

- メンバ関数（MotionInterpolationApp）
  - 初期化処理で、サンプル動作の読み込み
  - マウスドラッグ処理で、動作補間の重みの変更
  - アニメーション処理で、動作補間を含む動作再生
    - AnimationWithInterpolationメンバ関数
  - 描画処理で、動作補間の結果の現在姿勢の描画

```
class  MotionInterpolationApp : public GLUTBaseApp
{
    virtual void  Initialize();
    virtual void  Start();
    virtual void  Display();
    virtual void  MouseDrag( int mx, int my );
    virtual void  Keyboard( unsigned char key, int mx, int my );
    virtual void  Animation( float delta );
```

動作補間アプリケーション（4）

- 他のアプリケーションとの関係
  - 動作補間の各時刻の姿勢の計算に、姿勢補間関数を使用
    - MyPostureInterpolation関数
    - 先に姿勢補間アプリケーションを作成する必要がある
  - 動作の繰り返し再生に、動作接続の関数を使用
    - ComputeConnectionTransformation関数
    - 動作接続が未作成の場合、動作の繰り返し時に位置・向きが不連続になるが、動作補間の確認には支障はないため、先に動作補間を作成する

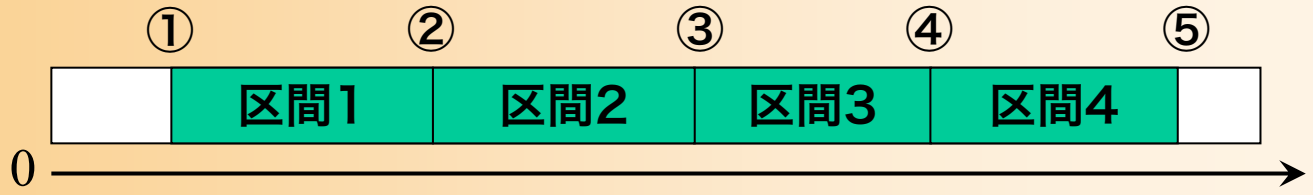
動作補間に用いる動作情報

- 動作のメタ情報の構造体（MotionTransition.h）

```
// 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;
    //キー時刻の配列 [キーフレーム番号]
    vector< float >  keytimes;
    ....
}
// 動作データのリスト（メンバ変数）
vector< MotionInfo * >  motion_list;
```

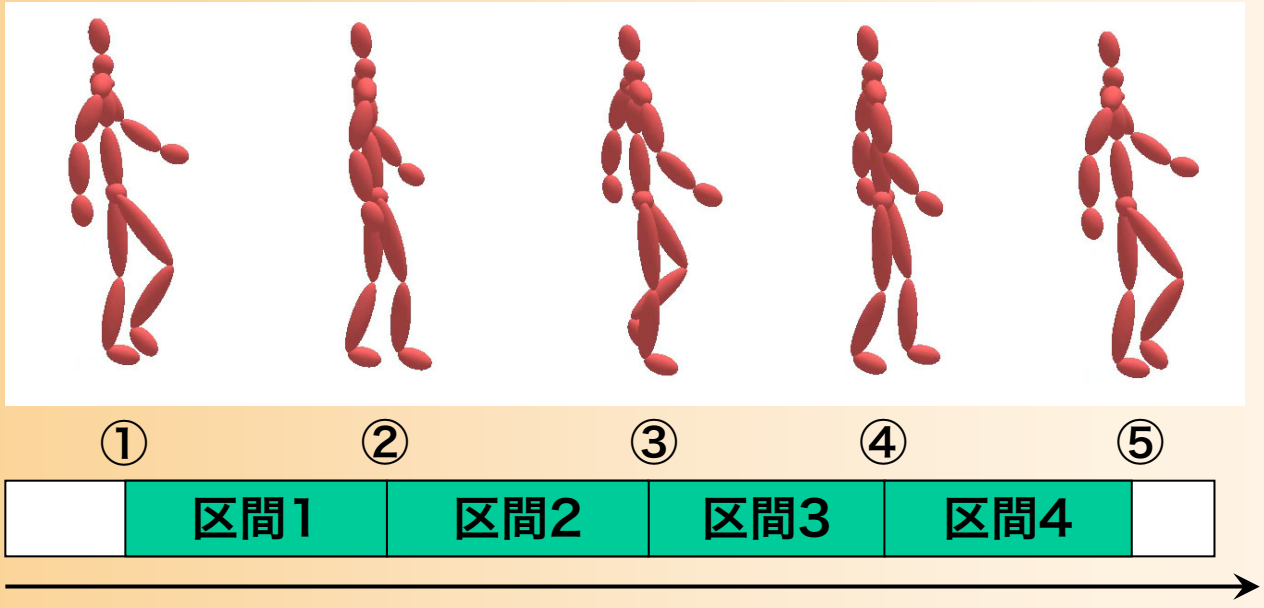
サンプル動作

- 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
  - 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
    1. 右足を上げ始める（動作開始）
    2. 右足を着く（ブレンド区間終了）
    3. 左足を上げ始める
    4. 左足を着く（ブレンド区間開始）
    5. 右足を上げ始める（動作終了）



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
  - 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



サンプル動作の読み込み

- LoadSampleMotions関数 (MotionTransitionApp.h)
  - 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

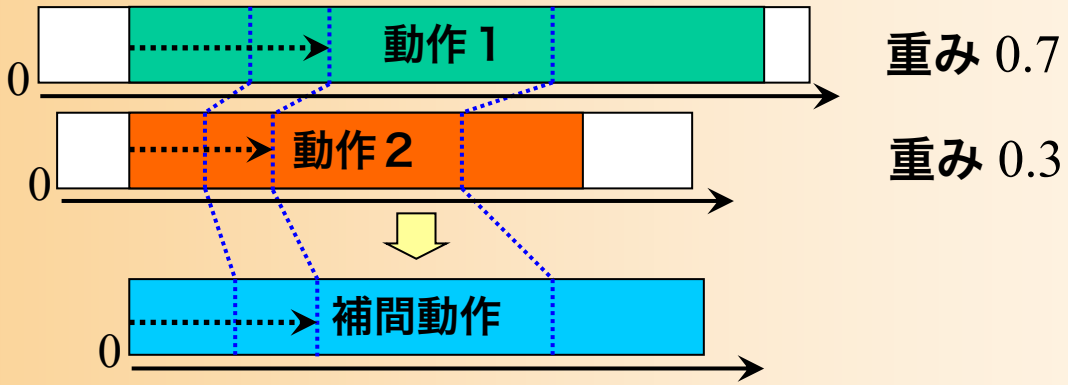
- 動作補間・再生

```
// 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

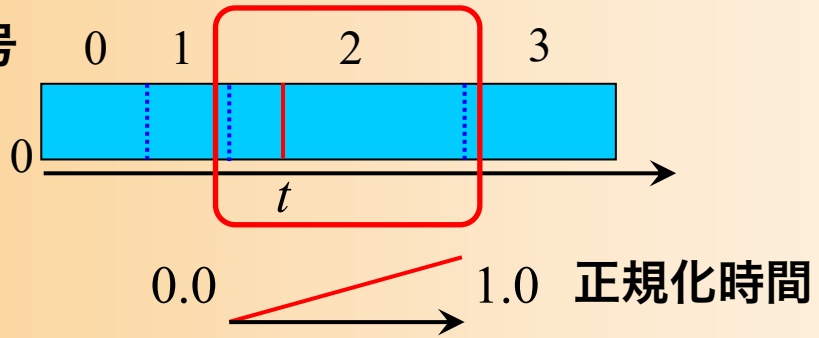
動作補間のプログラミング（1）

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

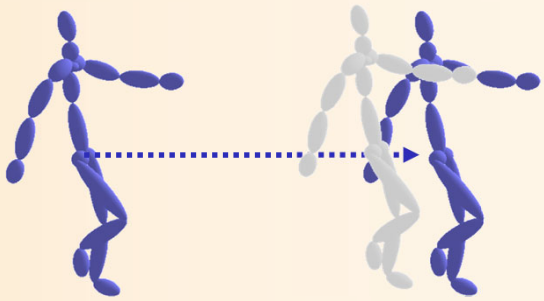
動作補間の改良・拡張

- 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

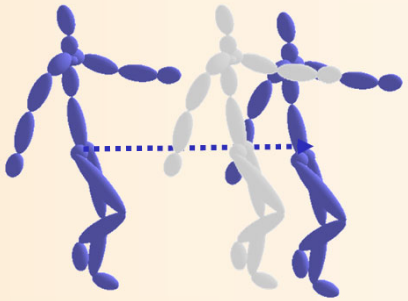
- 動作補間中の重みの変更への対応
- 動作補間のタイミングの計算（どの時刻の姿勢同士を補間するか）
- 複数（３つ以上）の動作の補間
- 特徴パラメタを使った動作補間の重みの計算

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



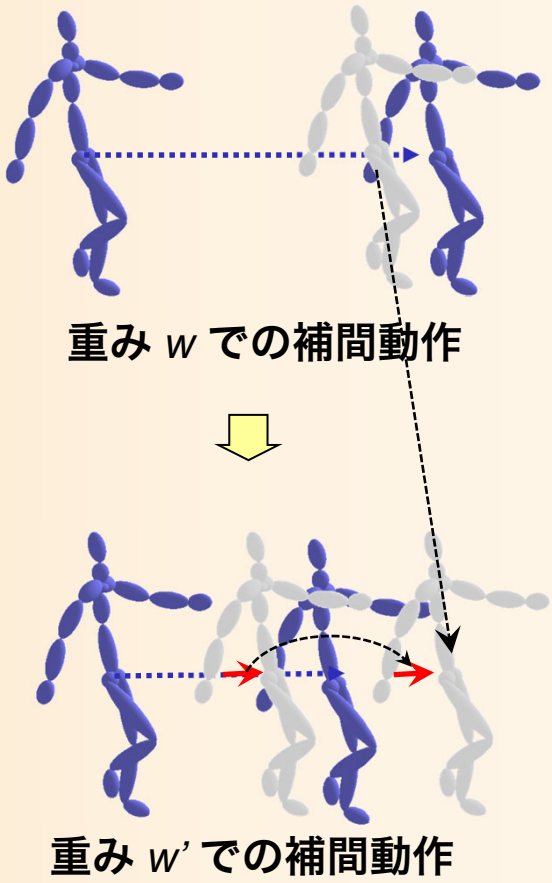
重み w での補間動作



重み w' での補間動作

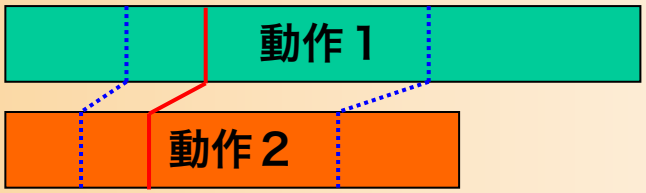
動作補間中の重みの変更（2）

- 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

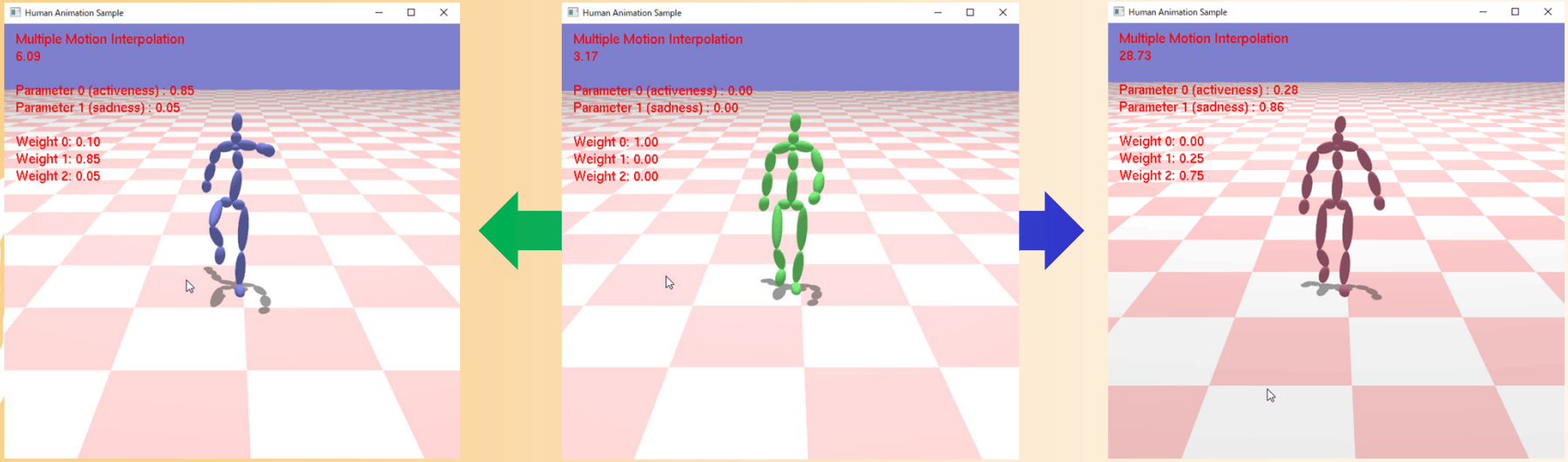


複数の動作の補間

- 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



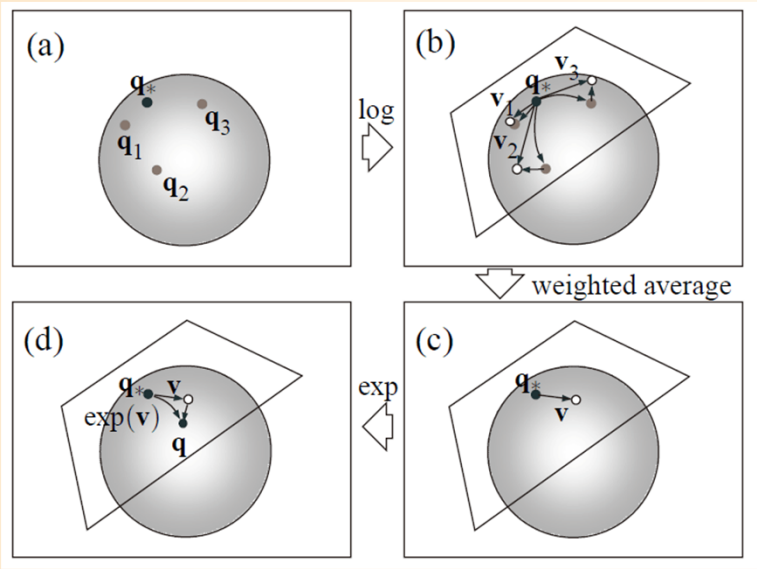
デモプログラム

- 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 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

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

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
- 動作補間の改良・拡張
- 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. 動作変形
    6. 動作接続・遷移
    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. 動作変形
6. 動作接続・遷移
    1. 動作接続
    2. 動作接続・遷移
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（1）
  - 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
  - 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あれば、演習問題の説明・選択肢を参考にして考えても良い
  - 締切後に解答が表示されるので確認する
    - レポート課題では、正しく動作するプログラムが提出されていれば、演習問題の正答の通りのプログラムが作成されていなくとも構わない

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
- 動作補間の改良・拡張
- 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、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