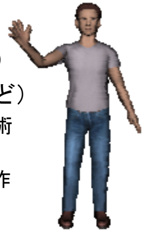
コンピュータグラフィックス特論 II

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実現の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

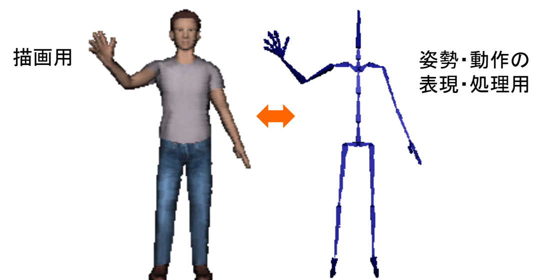
- キャラクタ・アニメーションの基礎
- 骨格モデル・姿勢・動作の表現
- 動作データの作成
- 運動学
- 姿勢・動作ブレンディング
- 応用技術

今日の内容

- 前回の復習
- 姿勢・動作ブレンディング
  - 姿勢補間
  - キーフレームアニメーション
  - 動作接続・遷移
  - 動作補間
- レポート課題

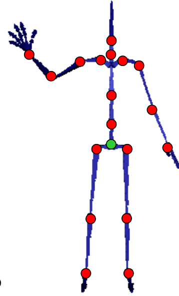
前回の復習

人体モデルの表現



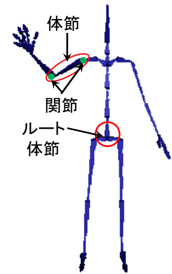
骨格モデルの表現

- 多関節体モデルによる表現
  - 複数の体節（リンク）が関節で接続されたモデル
  - 体節
    - 複数の関節が接続されており、体節の長さや体節内での関節の接続位置は固定
  - 関節
    - 2つの体節の間を接続
    - 関節の回転により姿勢が変化する



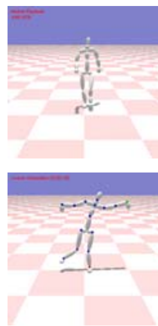
骨格モデルの表現方法

- 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



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

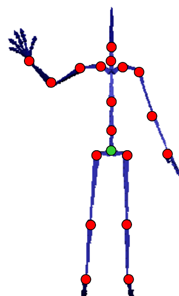
- デモプログラムの一部（human_sample.cpp）
  - 骨格・姿勢のデータ構造定義（SimpleHumn.h/cpp）
  - BVH動作クラス（BVH.h/cpp）
  - アプリケーションの基底クラス（GLUTBaseApp.h/cpp）
    - 各イベント処理のためのメソッドの定義を含む
    - 本クラスを派生させて、各アプリケーションクラス定義
  - メイン処理、コールバック関数（GLUTBaseApp.cpp）
    - 全アプリケーションを管理、切替
    - 現在のアプリケーションのイベント処理を呼び出し
  - 各デモの主要な処理は各自で実装

骨格・姿勢・動作のデータ構造

- 骨格・姿勢の構造体定義（SimpleHumn.h/cpp）

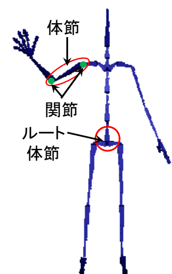
```
// 多関節体の体節を表す構造体
struct Segment

// 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
```

- BVH動作クラス（BVH.h/cpp）



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

```
// 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
   int                   num_segments;
   Segment **            segments;
   int                   num_joints;
   Joint **              joints;
};
```

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

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

※ IK計算などで、体節・関節を順番に辿りながら処理をするときのために、ルートがどちら側を判断するためのルールや情報があった方が良い

姿勢の表現方法

```
// 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 *   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
};
```

動作の表現方法

```
// 人体モデルの動作を表す構造体
struct  Motion
{
   // 骨格モデル
   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;
};
```

骨格・動作の読み込み

- 骨格モデルや動作データが必要
- BVH形式の動作データを読み込み、骨格モデル・動作データに変換する
  - BVH動作のクラス（BVH）や読み込み処理は、前回説明した通り
  - BHVクラスのままでは扱いづらいため、変換

```
// BVH動作から骨格モデルを生成
Skeleton *  CoustructBVHSkeleton( class BVH * bvh );

// BVH動作からデータ（＋骨格モデル）を生成
Motion *  CoustructBVHMotion( class BVH * bvh, Skeleton * b );
```

描画処理

- 姿勢描画

```
// 姿勢の描画（スティックフィギュアで描画）
void  DrawPosture( const Posture & posture );

// 姿勢の影の描画（スティックフィギュアで描画）
void  DrawPostureShadow( const Posture & posture,
        const Vector3f & light_dir, const Color4f & color );
```

  - 内部で順運動学計算を呼び出し

アプリケーションの基底クラス

- GLUTBaseApp

```
class  GLUTBaseApp
{
  protected:
    // 視点操作のための変数
    // マウス入力処理のための変数
    // アプリケーション状態の変数
  public:
    // イベント処理インターフェース
    virtual void  Initilize();
    virtual void  Start();
    virtual void  Display();
    ・・・
    virtual void  Animation( float delta );
};
```

GLUTメイン処理

- 複数のアプリケーションを管理・切り替え

```
// 全アプリケーションのリスト
vector< GLUTBaseApp * >    applications;

// 現在実行中のアプリケーション
GLUTBaseApp *    app = NULL;
```

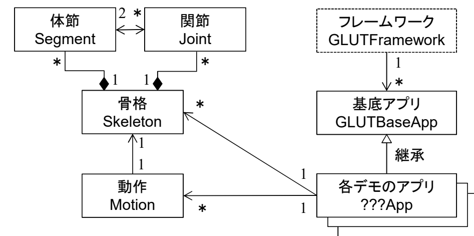
上のリストの中の、現在実行中のアプリケーションを表す

- GLUTコールバック関数から、現在のアプリケーションのイベント処理を呼び出し

```
//void  DisplayCallback( void )
{
    app->Display();
    ・・・
}
```

クラス図

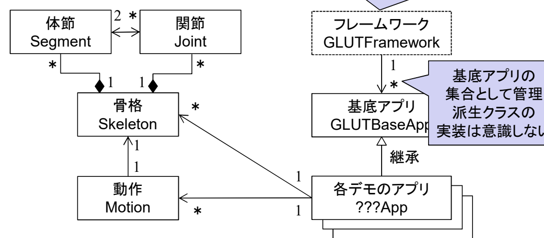
- クラス・構造体間の関係



クラス図

- クラス・構造体間の関係

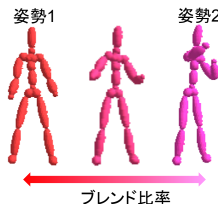
グローバル関数の集まりで構成されるので、クラスではないが、ここでは一つのクラスと同様に記述



姿勢・動作ブレンディング

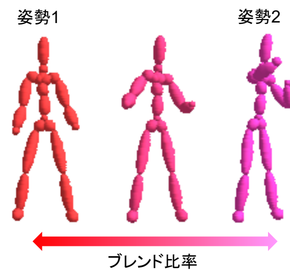
姿勢・動作ブレンディング

- 基礎技術
  - 2つの姿勢の補間（ブレンディング）
    - 混合、補間、合成など 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（ブレンド）して、新しい姿勢を生成

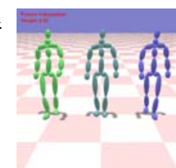


2つの姿勢の補間の計算方法

- 2つの入力姿勢を指定された重み(比率)で補間(ブレンド)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

姿勢補間アプリケーション

- PostureInterpolationApp
  - 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



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 = ???;
}
```

姿勢補間のプログラミング

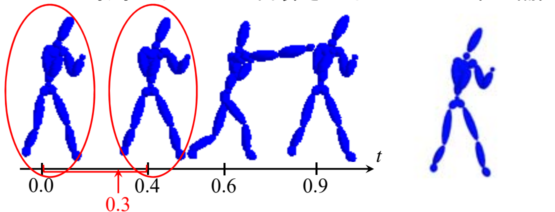
- 位置の補間(腰の位置)
  - 線形補間
- 向き・回転の補間(腰の向き・関節の回転)
  - 四元数を使った球面線形補間(SLERP)
  - vecmathのクラス・メソッドを使って計算可能
    - キーフレームアニメーションの回の説明を参照

姿勢補間の応用例

- キーフレームアニメーション
- 動作接続・遷移
- 動作補間

キーフレームアニメーション

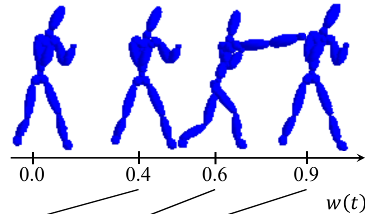
- キー姿勢の間を補間して動作を生成
  - (時刻、姿勢データ)の組の配列から動作を生成
  - 各区間で、前後の2つのキーフレームの姿勢を、時刻にもとづいて計算されるブレンド比率で補間



0.0 0.4 0.6 0.9 t
0.3

キーフレームアニメーション

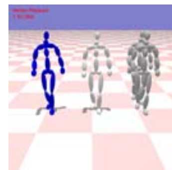
- ブレンド比率(姿勢補間の重み)の変化
  - 時間の関数として設定
  - 関節ごとに異なる関数を設定することもできる



0.0 0.4 0.6 0.9 t
$w(t)$

キーフレーム動作再生アプリケーション

- KeyframeMotionPlaybackApp
  - 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

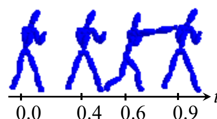


キーフレーム動作再生アプリケーション

- 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;
};
```



0.0 0.4 0.6 0.9 t

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

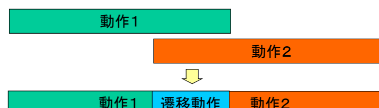
作成済みの姿勢補間関数を利用

姿勢補間の応用例

- キーフレームアニメーション
- 動作接続・遷移
- 動作補間

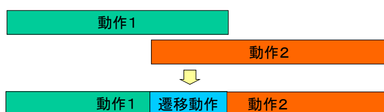
動作接続・遷移

- 動作接続・遷移・合成（トランジション、ブレンド）
  - 短い動作の間をつなげて長い動作を生成
  - 前後の動作の間を滑らかにつなぐ手法
    - 時間に応じた比率で前後の動作の姿勢を混合
    - 前の動作の終了部分と、次の動作の開始部分が、同じような動作である場合に適用可能



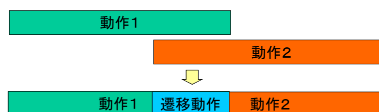
動作接続・遷移

- 動作接続・遷移・合成（トランジション、ブレンド）
  - 接続・遷移・合成などの呼び方がある
  - 厳密に定義すると、以下のように定義できる
    - 接続は、動作間の位置やタイミングを合わせて接続
    - 遷移は、動作接続時に滑らかな遷移を実現
    - 接続と遷移の組み合わせが必要になる



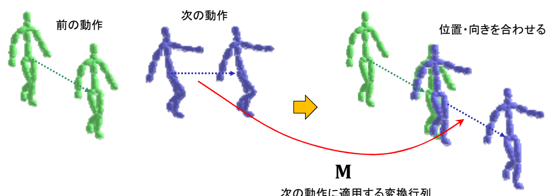
動作接続・遷移の方法

- 動作接続
  - 前の動作の終わりの位置・向きと、次の動作の始まりの位置・向きやタイミングを合わせる
- 動作遷移（動作ブレンド）
  - 時刻 t に対応する、前後の動作の時刻 t_i とウェイト w_i を決定し、両動作の姿勢を補間



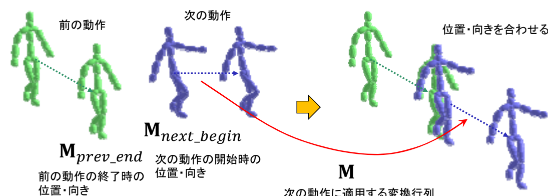
動作接続での位置・向き合わせ

- 前の動作の終了時の姿勢の位置・向きに、次の動作の始まりの位置・向きがつながるように、次の動作に座標変換を適用
  - 腰の位置・向きを基準に座標変換を計算



動作接続での位置・向き合わせ

- 前の動作の終了時の姿勢の位置・向きに、次の動作の始まりの位置・向きがつながるように、次の動作に座標変換を適用
  - 腰の位置・向きを基準に座標変換を計算



動作接続の処理方法

- 2つの動作の位置・向きを合わせる
  - 前の動作の終了時の位置・向きと、次の動作の開始時の位置・向きが一致するように、次の動作の各姿勢に変換行列を適用

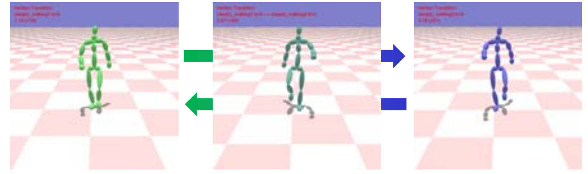
$$\mathbf{M} = \mathbf{M}_{prev_end} \left(\mathbf{M}_{next_begin} \right)^{-1}$$

次の動作に適用する変換行列　前の動作の終了時の位置・向き　次の動作の開始時の位置・向き

  - 実際には上の計算方法では、上下回転・左右回転・上下位置のずれを含む場合、接続した動作の方向・位置がおかしくなる
    - 各行列から水平回転・水平移動の成分のみを取り出して計算すると、より適切な変換行列が計算できる

動作接続・遷移アプリケーション

- MotionTransitionApp
  - 動作再生 ＋ 動作接続・遷移
  - マウス操作（左クリック）に応じて、次に再生する動作を変更
    - 変更なしの場合は、同じ動作に接続・遷移（繰り返し）



動作接続・遷移アプリケーション

- MotionTransitionApp（一部未実装）
  - 再生する複数の動作を設定
    - 2つのBVH動作を設定
      - 各動作の開始・終了時刻、ブレンド開始・終了時刻を設定
  - マウス操作（左クリック）に応じて、次の動作を変更
  - アニメーション処理（アニメーション関数）
    - 動作再生（現在時刻の姿勢取得）処理を呼び出し
  - 動作接続・遷移を含む動作再生（各自作成）
    - AnimationWithConnection関数＋ComputeConnectionTransformation関数（動作接続のみ）
    - AnimationWithConnectionTransition関数（動作接続・遷移）

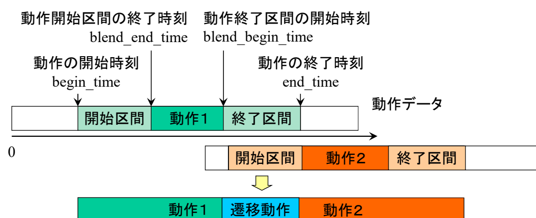
動作接続・遷移のための情報

- 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）
    float           blend_end_time;
    float           blend_begin_time;
}
```

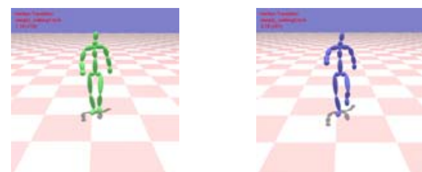
動作接続・遷移のための情報

- 動作のメタ情報
  - 動作の開始・終了時刻（動作のローカル時間）
  - ブレンド終了・開始時刻（動作のローカル時間）



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

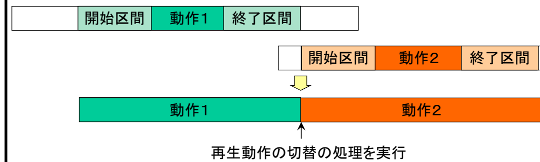


サンプル動作

- 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
    - 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)
  - 最初と最後の区間を、ブレンド区間とする
    - 本来は、動作遷移時は、前後の動作で同じ動き行う区間同士をブレンドすることが望ましいが、動作接続を分かりやすくするために、このブレンド区間を使用

動作接続の処理方法

- 動作接続のタイミング
  - 前の動作の終了時刻と次の動作の開始時刻を合わせる
    - 開始・終了ブレンド区間は無視
  - 切替時、次の動作に適用する変換行列を計算



動作接続のプログラミング(1)

- 動作再生 ＋ 動作接続

```
// アニメーション処理
void  AnimationWithConnection( float delta )
{
   // アニメーションの時間を進める

   // 現在の動作の再生が終了したら、次の動作に遷移・接続を実行
   if ( ・・・ )
      // 次の動作に適用する、接続のための変換行列を計算
      ComputeConnectionTransformation( ・・・ );

   // 現在の動作から姿勢を取得、接続のための変換行列を適用
   TransformPosture ( ・・・ );

}
```

動作接続のプログラミング(2)

- 動作接続のための変換行列の計算

```
// 2つの姿勢の位置・向きを合わせるための変換行列を計算
//(next_frame の位置・向きを、prev_frame の位置向きに合わせる
   ための変換行列 trans_mat を計算)
void  ComputeConnectionTransformation(
   const Matrix4f & prev_frame, const Matrix4f & next_frame,
   Matrix4f & trans_mat )
{
```

$$\mathbf{M} = \mathbf{M}_{prev_end}\left(\mathbf{M}_{next_begin}\right)^{-1}$$

trans_mat　prev_frame　next_frame

各自作成

```
}

// 姿勢の位置・向きに変換行列を適用
void  TransformPosture( const Matrix4f & trans, Posture & posture );
```

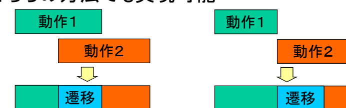
動作接続のプログラミング(3)

- 動作接続のための変換行列の計算
  - 実際には上の計算方法では、上下回転・左右回転・上下位置のずれを含む場合、接続した動作の方向・位置がおかしくなる
  - 水平方向の向きの成分のみを取り出して計算すると、より適切な変換行列を計算できる

```
// 変換行列の水平向き(方位角)成分を計算
float  ComputeOrientation( const Matrix3f & ori );
```

動作遷移の処理方法

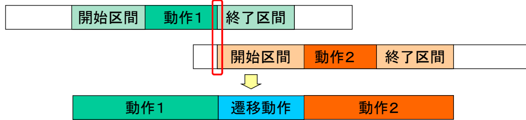
- 動作遷移区間で姿勢ブレンドを適用
  - 2つの動作の時間を重ねる場合は、動作中の姿勢同士のブレンド(左図)
  - 時間を重ねない場合は、前の動作の終了姿勢と、次の動作の姿勢をブレンド(右図)
    - どちらの方法でも実現可能



- 適切な区間の情報をあらかじめ設定しておく

動作遷移のタイミング

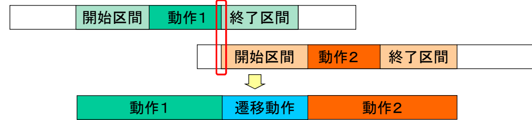
- 動作遷移の区間設定や姿勢ブレンディングの方法にはさまざまなやり方がある
  - 今回は、以下の2つのタイミングを合わせる
    - 前の動作の終了時のブレンド区間の開始時刻
    - 次の動作の開始時のブレンド区間の開始時刻



前後の動作の位置・向きを合わせる＋ブレンド開始　ブレンド終了

動作遷移の再生処理

- ブレンド開始時の処理
- ブレンド終了時の処理（次の動作への切替）
- ブレンド開始前は、前の動作から姿勢取得
- ブレンド中は、両動作から姿勢取得・補間

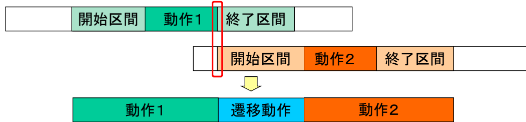


前後の動作の位置・向きを合わせる＋ブレンド開始　ブレンド終了

動作遷移の再生処理

- 動作遷移（ブレンド中）は、両動作から姿勢取得して補間
  - 補間の重みは時間にもとづいて計算
    - 0.0 → 1.0 に単調増加するような関数 $w(t)$ を使用

$$\mathbf{p} = w(t)\mathbf{P}_{next}(t) + (1 - w(t))\mathbf{P}_{prev}(t)$$



前後の動作の位置・向きを合わせる＋ブレンド開始　ブレンド終了

動作接続・遷移のプログラミング

- 動作再生 ＋ 動作接続・遷移

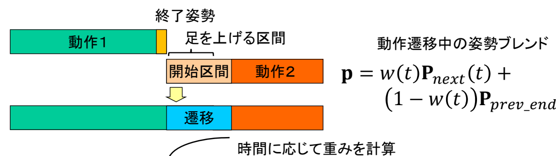
```
// アニメーション処理
void  AnimationWithConnectionTransition( float delta )
{
    // アニメーションの時間を進める

    // 動作遷移のブレンド開始
    if ( ・・・ )
        // 次の動作に適用する、接続のための変換行列を計算

    // 動作遷移のブレンド終了
    if ( ・・・ )
        // 再生動作の切替

    // 現在の動作から姿勢を取得
    // 動作遷移のブレンド中であれば、次の動作の姿勢とブレンド
}
```

現在の動作と次の動作から、適切なタイミングの姿勢を取得して変換行列を計算

現在時刻から ブレンド比率を計算

別の動作遷移のタイミングの例

- 動作遷移の区間設定や姿勢ブレンディングの方法にはさまざまなやり方がある
  - 前の動作の終了姿勢を次の動作の開始部分とブレンドする方法もある
  - 動作開始時のブレンド区間のみを使用



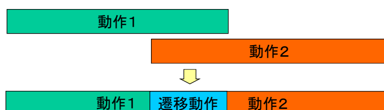
$$\mathbf{p} = w(t)\mathbf{P}_{next}(t) + (1 - w(t))\mathbf{P}_{prev_end}$$

動作遷移での姿勢補正

- 前後の動作の姿勢を補間するだけでは、動作遷移中の動作が不自然になる場合がある
  - 前後の動作で地面に着いている足の位置が異なると、動作遷移中に足が地面の上を滑るような動作になってしまう
- IK計算により足の位置を補正
  - 足が接地している間の足の位置を決定
  - 足が接地している間は、足の位置をIKで固定
  - 足が接地する前・後の一定時間は、足の位置が動作遷移中の位置になるように、IKで修正

動作遷移での姿勢補正

- IK計算により足の位置を補正
  - 足が接地している間の足の位置を決定
  - 足が接地している間は、足の位置をIKで固定
  - 足が接地する前・後の一定時間は、足の位置が動作遷移中の位置になるように、IKで修正



動作接続・遷移の利用時の注意

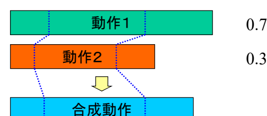
- 自然な動作接続・遷移の実現のためには、適切な遷移区間や姿勢修正方法を設定しておく必要がある
  - 前後の動作の組み合わせごとに適切な設定が必要
- 姿勢が大きく異なる動作間に適用することは難しい

姿勢補間の応用例

- キーフレームアニメーション
- 動作接続・遷移
- 動作補間

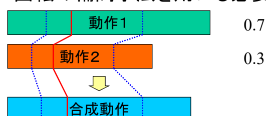
動作補間

- 動作補間（モーション・インターポレーション）
  - 複数の動作をブレンド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
    - 任意の比率で混ぜ合わせることができる
  - 事前に複数の動作を同期しておく必要がある
    - 同じタイミングにキー時刻を設定しておく



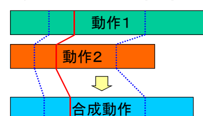
動作補間の計算方法

- 任意の時刻 t の姿勢 $\mathbf{p}_i$ を計算
  - 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
  - 各動作の時刻 t_i 姿勢 $\mathbf{p}_i$ を取得
  - 各姿勢 $\mathbf{p}_i$ をウェイト 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



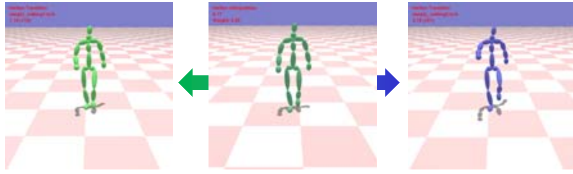
動作補間でのタイミングの計算

- 時刻 t に対応する、各動作の時刻 t_i を決定
  - 補間動作（合成動作）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
  - 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
  - 各動作のキー時刻にもとづき、時刻 t_i を計算



動作補間アプリケーション

- MotionInterpolationApp
  - 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
    - 動作再生中でも、補間の重みを変更できる



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
      - 5つのキー時刻を設定(右足を上げる、右足をつく、左足を上げる、左足をつく、右足を上げる)
  - 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

動作補間のプログラミング

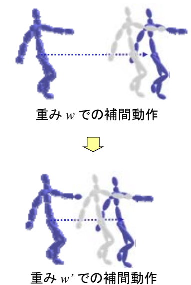
- 動作補間・再生

```
// アニメーション処理
void  MotionTransitionApp::Animation( 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

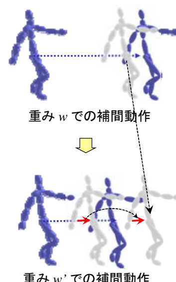
動作補間中の重みの変更(改良)

- 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



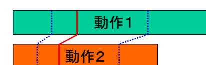
動作補間中の重みの変更(改良)

- 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(改良)

- 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

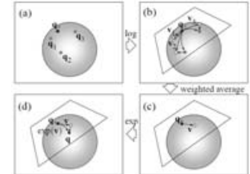


複数の動作の補間

- 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
- 複数の回転の補間を実現する方法として、回転を四元数の対数表現に変換する方法がある

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、何らかの特徴パラメタ（速さ、距離、方向、高さなど）から決定できる
  - あらかじめ、n 個の動作データのそれぞれに、k 次元のパラメタ $\mathbf{f}_i$ 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、何らかの特徴パラメタ（速さ、距離、方向、高さなど）から決定できる
  - あらかじめ、n 個の動作データのそれぞれに、k 次元のパラメタ $\mathbf{f}_i$ 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

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

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（疑似逆行列）により計算
  - 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

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

$A_j(\mathbf{p})$: 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.

レポート課題

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
     1. 動作接続のみ
     2. 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

今回説明（他の課題は前回説明）

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

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

動作接続・遷移アプリケーション

- MotionTransitionApp（一部未実装）
  - 再生する複数の動作を設定
    - 2つのBVH動作を設定
      - 各動作の開始・終了時刻、ブレンド開始・終了時刻を設定
  - マウス操作（左クリック）に応じて、次の動作を変更
  - アニメーション処理（アニメーション関数）
    - 動作再生（現在時刻の姿勢取得）処理を呼び出し
  - 動作接続・遷移を含む動作再生（各自作成）
    - AnimationWithConnection関数＋ComputeConnectionTransformation関数（動作接続のみ）
    - AnimationWithConnectionTransition関数（動作接続・遷移）

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
      - 5つのキー時刻を設定（右足を上げる、右足をつく、左足を上げる、左足をつく、右足を上げる）
  - 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

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. 動作接続・遷移
     1. 動作接続のみ
     2. 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

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
     1. 動作接続のみ
     2. 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

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）をもとに作成したプログラムを提出
    - 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
  - Moodleの本講義のコースから提出
  - 締切：6月1日（木） 18:00（厳守）

まとめ

- 前回の復習
- 姿勢・動作ブレンディング
  - 姿勢補間
  - キーフレームアニメーション
  - 動作接続・遷移
  - 動作補間
- レポート課題

次回予告

- キャラクタ・アニメーションの基礎
- 骨格モデル・姿勢・動作の表現
- 動作データの作成
- 運動学
- 姿勢・動作ブレンディング
- 応用技術