コンピュータアニメーション特論 レポート
第4回 キャラクタアニメーション（1）

学生番号: 12345678　　氏名: 九工大 太郎

20xx 年 x 月 x 日

レポートの書き方の注意：（この部分は、提出レポートからは削除すること）

- 以下の様式中の「※ レポート課題」の部分を、自分が作成したプログラムに置き換える。
- 変数定義やインデントを適切に行うこと。動作しないプログラムや見にくいプログラムは、減点となる。
- 様式で指定されている箇所以外に変更を加えた場合は、どの関数を追加変更したのかが分かるように、関数定義を含めて変更内容を枠内に記述する。

1　キャラクタアニメーションの実現

キャラクタアニメーションの基本処理を実現するように、以下の通り、元のプログラムの処理の一部に変更を加えた。

1.1　順運動学計算

1.1.1　順運動学計算のための反復計算

ForwardKinematicsApp.cpp の MyForwardKinematicsIteration 関数の空欄部分を、以下のように作成した。

```
void  MyForwardKinematicsIteration(
   const Segment *  segment, const Segment * prev_segment, const Posture & posture,
   Matrix4f * seg_frame_array, Point3f * joi_pos_array )
{
   // 省略

   Joint *  next_joint;
   Segment *  next_segment;
   Matrix4f  mat;

   // 各接続関節ごとに反復
   for ( int j=0; j<segment->joints.size(); j++ )
   {
      // 次の関節・次の体節を取得
      next_joint = segment->joints[ j ];
      if ( next_joint->segments[ 0 ] != segment )
         next_segment = next_joint->segments[ 0 ];
      else
         next_segment = next_joint->segments[ 1 ];

      // 前の体節側（ルート体節側）の関節はスキップ
      if ( next_segment == prev_segment )
         continue;

      // 現在の体節の変換行列を取得
```

```
        mat = seg_frame_array[ segment->index ];

        // 現在の体節の座標系から、接続関節への座標系への平行移動をかける
//      ???;

        // 次の関節の位置を設定
        if ( joi_pos_array )
            joi_pos_array[ next_joint->index ] = pos;

        // 次の関節の回転行列をかける
//      ???;

        // 関節の座標系から、次の体節の座標系への平行移動をかける
//      ???;

        // 次の体節の変換行列を設定
        if ( seg_frame_array )
            seg_frame_array[ next_segment->index ] = frame;

        // 次の体節を呼び出す
        MyForwardKinematicsIteration( next_segment, segment, posture, seg_frame_array,
            joi_pos_array );
    }
}
```

1.2 姿勢補間

1.2.1 2つの姿勢を補間

PostureInterpolationApp.cpp の MyPostureInterpolation 関数の空欄部分を、以下のように作成した。

```
void  MyPostureInterpolation( const Posture & p0, const Posture & p1, float ratio,
    Posture & p )
{
    // 省略

    // 骨格モデルを取得
    const Skeleton *  body = p0.body;

    // 計算用変数
    Quat4f  q0, q1, q;
    Vector3f  v0, v1, v;

    // 2つの姿勢の各関節の回転を補間
    for ( int i = 0; i < body->num_joints; i++ )
    {
        // ???
    }

    // 2つの姿勢のルートの向きを補間
    // ???

    // 2つの姿勢のルートの位置を補間
    // ???
}
```

1.3 キーフレーム動作再生

1.3.1 キーフレーム動作からの姿勢取得

KeyframeMotionPlaybackApp.cpp の GetKeyframeMotionPosture 関数の空欄部分を、以下のように作成した。

```
void  GetKeyframeMotionPosture( const KeyframeMotion & motion, float time, Posture & p
    )
{
   // 指定時刻に対応する区間の番号を取得
   int  no = -1;
// no = ???;

   // 対応する区間が存在しない場合は終了
   if ( no == -1 )
      return;

   // 補間の割合を計算
   float  s =
// s = ???;

   // 前後のキー姿勢を補間
   MyPostureInterpolation( motion.key_poses[ no ], motion.key_poses[ no+1 ], s, p );
}
```

1.4 動作補間

1.4.1 動作再生処理（動作補間）

MotionInterpolationApp.cpp の AnimationWithInterpolation 関数の空欄部分を、以下のように作成した。

```
void  MotionInterpolationApp::AnimationWithInterpolation( float delta )
{
   // 省略

   // 時間を進める
   animation_time += delta * animation_speed;

   // 補間動作のキー時刻を計算（サンプル動作のキー時刻を重みで平均）
   keytimes[ 0 ] = 0.0f;
   for ( int i = 1; i < num_keyframes; i++ )
   {
      // ※ レポート課題
//    keytimes[ i ] = ???;
   }

   // 補間動作の現在時刻（動作開始時を基準とする時間）を計算
   local_time = animation_time - cycle_start_time;

   // 現在の繰り返し動作が終了したら、次の繰り返しを開始
   if ( local_time > keytimes[ num_keyframes - 1 ] )
   {
      // 省略
   }

   // 正規化時間（区間番号と区間内の正規化時間）を計算
   for ( int i = 0; i < num_keyframes-1; i++ )
   {
      if ( ( local_time >= keytimes[ i ] ) && ( local_time <= keytimes[ i + 1 ] ) )
      {
         seg_no = i;
```

```
            // ※ レポート課題
//          seg_time = ???;

            break;
        }
    }

    // サンプル動作から姿勢を取得
    for ( int i = 0; i < 2; i++ )
    {
        // ※ レポート課題
//      motion_time = ???;

        // 動作データから姿勢を取得、変換行列を適用
        // 省略
    }

    // 姿勢補間
    MyPostureInterpolation( *motion_posture[ 0 ], *motion_posture[ 1 ], weight, *
        curr_posture );

    // 省略
}
```