コンピュータアニメーション特論 レポート
第5回 キャラクタアニメーション（2）

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。

1.5　動作変形

1.5.1　姿勢変形の重みを計算

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

```
float  ApplyMotionDeformation( float time, const MotionWarpingParam & deform,
const Posture & input_pose, Posture & output_pose )
{
    // 動作変形の範囲外であれば、入力姿勢を出力姿勢とする
    // 省略

    // ※ レポート課題

    // 動作変形（動作ワーピング）の重みを計算
    float  ratio = 0.0f;
// ratio = ???;

    // 姿勢変形（２つの姿勢の差分（dest − src）に重み ratio をかけたものを元の姿勢 org
        に加える ）
    PostureWarping( input_pose, deform.org_pose, deform.key_pose, ratio, output_pose );

    return  ratio;
}
```

1.5.2 動作ワーピングのための姿勢変形

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

```
void  PostureWarping( const Posture & org, const Posture & src, const Posture & dest,
    float ratio, Posture & p )
{
   // 省略

   // 計算用変数
   Quat4f  q0, q1, q;
   Vector3f  v;
   Matrix3f  rot;

   // ※ レポート課題

   // 各関節の回転を計算
   for ( int i = 0; i < body->num_joints; i++ )
   {
//    p.joint_rotations[ i ] = ???;
   }

   // ルートの向きを計算
// p.root_ori = ???;

   // ルートの位置を計算
// p.root_pos = ???;
}
```

1.6 動作接続・遷移

1.6.1 動作接続のための変換行列の計算

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

```
void  ComputeConnectionTransformation( const Matrix4f & prev_frame, const Matrix4f &
    next_frame, Matrix4f & trans_mat )
{
   // 方法 1
   // trans_mat = ???

   // 方法 2 (水平方向の向き・位置のみを変換)
   Matrix3f  ori;
   float  angle;
   Vector3f  pos;
   Matrix4f  prev_frame2, next_frame2;

   // 変換行列から水平方向の回転のみを抽出して再設定
   // prev_frame2 = ???

   // 変換行列から水平方向の回転のみを抽出して再設定
   // next_frame2 = ???

   // 座標変換を計算
   // trans_mat = ???
}
```

1.6.2 動作遷移のタイミング・姿勢の計算

MotionTransition.cpp の Init 関数と GetPosture 関数の空欄部分を、以下のように作成した。

```
bool  MotionTransition::Init(
   const MotionInfo * prev_motion, const MotionInfo * next_motion, const Matrix4f &
       prev_motion_mat, float prev_begin_time )
{
   // 省略

   // ※ レポート課題

   // 後の動作を開始する時刻（前の動作の開始時刻 prev_begin_time を基準とするローカル時
       刻）
// next_begin_time = ???;

   // 動作遷移のための動作ブレンドを行う開始時刻（前の動作の開始時刻 prev_begin_time を
       基準とするローカル時刻）
// blend_begin_time = ???;

   // 動作遷移のための動作ブレンドを行う終了時刻（前の動作の開始時刻 prev_begin_time を
       基準とするローカル時刻）
// blend_end_time = ???;

   // 省略
}


MotionTransition::MotionTransitionState  MotionTransition::GetPosture(
   float time, Posture * posture )
{
   // 省略

   // 前の動作のローカル時刻（後の動作の開始時刻を基準とするローカル時刻）
   float  local_time = 0.0f;

   // 後の動作のローカル時刻（後の動作の開始時刻を基準とするローカル時刻）
   float  next_motion_local_time = 0.0f;

   // 前の動作の開始時刻を基準とするローカル時刻を計算
   local_time = time - prev_begin_time;

   // 現在の状態を判定
   MotionTransitionState  state = MT_PREV_MOTION;
   if ( local_time > blend_end_time )
      state = MT_NEXT_MOTION;
   else if ( local_time > blend_begin_time )
      state = MT_IN_TRANSITION;

   // 前の動作の姿勢を取得
   if ( state == MT_PREV_MOTION || state == MT_IN_TRANSITION )
   {
      // 前の動作の姿勢を取得
      prev_motion->motion->GetPosture( local_time + prev_motion->begin_time, *
          prev_motion_posture );

      // 前の動作の姿勢の位置・向きに変換行列を適用
      TransformPosture( prev_motion_mat, *prev_motion_posture );
   }

   // 後の動作の姿勢を取得
   if ( state == MT_NEXT_MOTION || state == MT_IN_TRANSITION )
   {
      // ※ レポート課題

      // 後の動作の現在時刻を計算（後の動作の開始時刻を基準とするローカル時刻）
//    next_motion_local_time = ???;
```

```
        // 後の動作から現在時刻の姿勢を取得
        next_motion->motion->GetPosture( next_motion_local_time + next_motion->begin_time
            , *next_motion_posture );

        // 後の動作の姿勢の位置・向きに変換行列を適用
        TransformPosture( next_motion_mat, *next_motion_posture );
    }

    // 動作遷移前であれば、前の動作の姿勢を出力
    if ( state == MT_PREV_MOTION )
    {
        // 省略
    }
    // 動作遷移後であれば、後の動作の姿勢を出力
    else if ( state == MT_NEXT_MOTION )
    {
        // 省略
    }
    // 動作遷移中であれば、前後の動作の姿勢を補間
    else
    {
        // ※ レポート課題

        // ブレンド比率（補間の重み）を計算
//      blend_ratio = ???;

        // 省略

        // 前後の動作の姿勢を補間
        MyPostureInterpolation( *prev_motion_posture, *next_motion_posture, blend_ratio,
            *posture );
    }

    // 省略
}
```

1.7 逆運動学計算（CCD 法）

1.7.1 逆運動学計算（CCD 法）（ルート体節を支点とする場合）

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

```
void  ApplyInverseKinematicsCCD( Posture & posture, int base_joint_no, int ee_joint_no,
    Point3f ee_joint_position )
{
    // 省略

    // CCD法の繰り返し計算（末端関節の位置が収束するか、一定回数繰り返したら終了する）
    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++ )
        {
            // 現在の関節と支点側の体節を取得
            joint = body->joints[ joint_path[ j ] ];
            direction = (float) joint_path_signs[ j ];
            segment = ( direction > 0.0f ) ? joint->segments[ 0 ] : joint->segments[ 1 ];

            // 末端関節の現在位置を取得
            ee_pos = joint_positions[ ee_joint_no ];
```

```
            // ※レポート課題

            // 現在の関節のローカル座標系を取得
//          mat = ???;

            // ワールド座標系から現在の関節のローカル座標系への変換行列を計算
//          inv_mat = ???;

            // 現在の関節から末端関節への方向ベクトル（現在の関節のローカル座標系）を計算
//          ee_vec = ???;

            // 現在の関節から目標位置への方向ベクトル（現在の関節のローカル座標系）を計算
//          goal_vec = ???;

            // 現在の関節の回転軸・回転角度（0～π）を計算
//          rot_axis = ???;
//          rot_angle = ???;

            // 回転を適用（回転の方向を考慮しない）
//          rot.set( AxisAngle4f( rot_axis, rot_angle ) );
//          ???;

            // 回転後の回転を設定
            posture.joint_rotations[ joint->index ].set( rot );

            // 更新された姿勢にもとづいて、各体節・関節の位置・向きを再計算（順運動学計算
                )
            ForwardKinematics( posture, segment_frames, joint_positions );
        }

        // 収束判定、末端関節の目標位置と現在位置の距離が閾値以下になったら終了
        ee_pos = joint_positions[ ee_joint_no ];
        vec.sub( ee_joint_position, ee_pos );
        dist = vec.length();
        if ( dist < distance_threshold )
            break;
    }
}
```

1.7.2 末端関節から支点関節へのパスを探索（任意の関節を支点とする場合）

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

```
void  FindJointPath( const Skeleton * body, int base_joint_no, int ee_joint_no, vector<
    int > & joint_path, vector< int > & joint_path_signs )
{
    // 末端関節からルート体節に向かうパスを探索

    // 省略

    // 支点が常にルート体節、もしくは、末端関節とルート体節の間にあると仮定すれば、ここ
        で終了しても構わない
    // それ以外の場所に支点関節がある場合は、ルートから支点関節までのパスを求めて追加す
        る処理が必要となる

    // ※レポート課題

    // 支点がルート体節 or 支点関節がルート体節から末端関節のパス上にある場合は、終了
    if ( ( base_joint_no == -1 ) || ( joint->index == base_joint_no ) )
        return;

    // 支点関節からルート体節へ向かうパス
```

```
    vector< int >  joint_path2;

    // 探索処理の終了判定用フラグ
    bool  termination = false;

    // 支点関節から探索を開始
    joint = body->joints[ base_joint_no ];

    // 支点関節からルート体節に向かうパスを探索
    while ( true )
    {
        // 末端からルートまでのパスと合流したかどうかを判定し、合流したら終了
        for ( ??? )
        {
            if ( ??? )
            {
                // 末端からルートまでのパスを、合流した体節の前の関節まで縮小
                joint_path.resize( i + 1 );
                termination = true;
                break;
            }
        }
        if ( termination )
            break;

        // 現在の関節をパスに追加
        joint_path2.push_back( joint->index );

        // ルート側の隣の関節を辿り、ルートに到達したら終了
        segment = joint->segments[ 0 ];
        if ( segment->index == 0 )
            break;
        joint = segment->joints[ 0 ];

        // 末端関節に到達した場合（ルートと支点関節の間に末端関節がある場合）は、
        // 末端関節からルートまでのパスはクリアして、支点関節から末端関節までのパスを使用
        if ( joint->index == ee_joint_no )
        {
            joint_path.clear();
            joint_path_signs.clear();
            break;
        }
    }

    // 末端からルートに向かうパスと、支点からルートに向かうパスを結合（後者は逆の順番で
        結合）
    // 各関節における末端関節の方向を表す符号の配列を生成
    joint_path_signs.resize( joint_path.size(), 1 );
    for ( int i = 0; i < joint_path2.size(); i++ )
    {
        joint_path.push_back( joint_path2[ ??? ] );
        joint_path_signs.push_back( -1 );
    }
}
```

1.7.3 逆運動学計算（CCD 法）（任意の関節を支点とする場合）

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

```
void  ApplyInverseKinematicsCCD( Posture & posture, int base_joint_no, int ee_joint_no,
    Point3f ee_joint_position )
{
```

```
    // 省略

    // CCD法の繰り返し計算（末端関節の位置が収束するか、一定回数繰り返したら終了する）
    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++ )
        {
            // 現在の関節と支点側の体節を取得
            joint = body->joints[ joint_path[ j ] ];
            direction = (float) joint_path_signs[ j ];
            segment = ( direction > 0.0f ) ? joint->segments[ 0 ] : joint->segments[ 1 ];

            // 末端関節の現在位置を取得
            ee_pos = joint_positions[ ee_joint_no ];

            // ※レポート課題

            // 現在の関節のローカル座標系を取得
//          ???

            // ワールド座標系から現在の関節のローカル座標系への変換行列を計算
//          ???

            // 現在の関節から末端関節への方向ベクトル（現在の関節のローカル座標系）を計算
//          ???

            // 現在の関節から目標位置への方向ベクトル（現在の関節のローカル座標系）を計算
//          ???

            // 現在の関節の回転軸・回転角度（0～π）を計算
//          ???

            // 回転を適用（回転の方向を考慮）
            if ( direction > 0.0f )
            {
//              ???
            }
            else
            {
//              ???
            }

            // 回転後の回転を設定
            posture.joint_rotations[ joint->index ].set( rot );

            // 末端関節と現在の関節の間にルートがある場合は、ルートに移動・回転を適用
            if ( direction < 0.0f )
            {
                Matrix4f  mat;

                // 現在の支点体節の変換行列を取得
//              ???

                // 順運動学（FK）計算
                ForwardKinematics( posture, segment_frames, joint_positions );

                // 姿勢変更後の支点体節の変換行列を取得
//              ???

                // 支点関節の位置・向きを保つための変換行列を計算
//              ???

                // 腰の位置・向きに座標変換を適用
```

```
//            ???
          }

          // 更新された姿勢にもとづいて、各体節・関節の位置・向きを再計算（順運動学計算
              ）
          ForwardKinematics( posture, segment_frames, joint_positions );
        }

        // 収束判定、末端関節の目標位置と現在位置の距離が閾値以下になったら終了
        ee_pos = joint_positions[ ee_joint_no ];
        vec.sub( ee_joint_position, ee_pos );
        dist = vec.length();
        if ( dist < distance_threshold )
          break;
    }
}
```