コンピュータグラフィックス特論II レポート
第5回 キャラクタ・アニメーション

学生番号: 12345678　　氏名: 九工大 太郎

2017 年 5 月 19 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

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

   // 各接続関節ごとに反復
   for ( int j=0; j<segment->joints.size(); j++ )
   {
      // 省略

      // 次の関節・体節の変換行列を計算

      // ※レポート課題（ここに自分が作成したプログラムを記述する）

      // 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

```
void PostureInterpolation( 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;

   // ※レポート課題（ここに自分が作成したプログラムを記述する）

}
```

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

```
void GetKeyframeMotionPosture( const KeyframeMotion & motion, float time, Posture & p
    )
{
   // 指定時刻に対応する区間番号を取得
   // 省略

   // ※レポート課題（ここに自分が作成したプログラムを記述する）

}
```

1.4 動作接続・遷移

1.4.1 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（動作接続）

```
void ComputeConnectionTransformation( const Matrix4f & prev_frame, const Matrix4f &
    next_frame, Matrix4f & trans_mat )
{
   // ※レポート課題（ここに自分が作成したプログラムを記述する）

}
```

1.4.2 動作再生処理（動作接続・補間）

```
void MotionTransitionApp::AnimationWithConnectionTransition( float delta )
{
   // 省略

   // 動作接続・遷移の動作ブレンドの開始
   if ( !on_motion_transition && ( local_time > curr_motion_blend_begin_time -
       curr_motion_begin_time ) )
   {
      // 現在の動作・次の動作の位置・向きを合わせる時刻を決定（それぞれの動作のローカル
          時刻）

      // ※レポート課題（ここに自分が作成したプログラムを記述する）
```

```
        // 省略
    }

    // 省略

    // 動作接続・遷移のためのブレンド中は、次の動作の姿勢とブレンド
    if ( on_motion_transition && enable_transition )
    {
        // 省略

        // ブレンド比率を計算

        // ※レポート課題（ここに自分が作成したプログラムを記述する）

        // 省略
    }

    // 省略
}
```

1.5 動作補間

1.5.1 動作再生処理

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

    // 正規化時間（区間番号と区間内の正規化時間）を計算

    // ※レポート課題（ここに自分が作成したプログラムを記述する）

    // サンプル動作から姿勢を取得
    float  motion_time = 0.0f;
    for ( int i = 0; i < 2; i++ )
    {
        // 動作データの時間を計算

        // ※レポート課題（ここに自分が作成したプログラムを記述する）

        // 省略
    }

    // 省略
}
```

1.6 逆運動学計算（CCD 法）

1.6.1 逆運動学計算（CCD 法）（ルート体節を支点とする場合）

```
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++ )
        {
            // 省略

            // 末端関節の位置を目標位置に近づけるための現在の関節の回転を計算・適用

            // ※レポート課題（ここに自分が作成したプログラムを記述する）

            // 省略
        }

        // 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

        // ※レポート課題（ここに自分が作成したプログラムを記述する）

    }
}
```

1.6.2 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）

```
void  FindJointPath( const Skeleton * body, int base_joint_no, int ee_joint_no, vector<
    int > & joint_path, vector< int > & joint_path_signs )
{
    // 省略

    // ルートから支点関節までのパスを求めて追加

    // ※レポート課題（ここに自分が作成したプログラムを記述する）

}
```

1.6.3 逆運動学計算（CCD 法）（任意の関節を支点とする場合）

```
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++ )
        {
            // 省略

            // 末端関節と現在の関節の間にルート体節がある場合は、ルート体節を移動・回転
            if ( direction < 0.0f )
            {
                // ※ レポート課題（ここに自分が作成したプログラムを記述する）

            }

            // 省略

}
```