コンピュータアニメーション特論 プログラミング演習資料
第9~12回 キャラクターアニメーション

九州工業大学 情報工学研究院　尾下 真樹

1　サンプルプログラム

キャラクターアニメーションのサンプルプログラム（SimpleHuman）をもとに、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、動作変形、動作接続・遷移、逆運動学計算（CCD 法）などのキャラクターアニメーションの基本処理を実現するプログラムを作成する。本サンプルプログラムは、複数の基本処理のアプリケーションを含んでおり、マウス中ボタン（または CTRL + 右ボタン）でメニューを表示して、実行するデモを選択できる。もしくは、キーボードの M キーで、アプリケーションを順番に切り替えることができる。各アプリケーションの機能や操作方法は、各基本処理の節で説明する。

本サンプルプログラムは、以下の複数のファイルから構成される。

1. vecmath 補助関数（vecmath_gl.h.h）
2. 骨格・姿勢・動作のデータ構造定義と基本処理関数（SimpleHumn.h/cpp）
3. BVH 動作クラス（BVH.h/cpp）
4. 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）
5. メイン関数（SimpleHumanSampleMain.cpp）
6. 各アプリケーションの定義・実装（???App.h/.cpp）

また、外部ライブラリとして、OpenGL と GLUT に加えて、行列・ベクトルなどを扱うための vecmath C++ ライブラリを使用する。

以降、これらの各ソースコードや主要なデータ構造・処理を順番に解説する。

1.1　骨格・姿勢・動作のデータ構造と基本処理関数

サンプルプログラムの SimpleHumn.h/cpp で、骨格・姿勢・動作のデータ構造と基本処理関数の定義・実装を行っている。

1.1.1　骨格・姿勢・動作のデータ構造

以下のデータ構造（構造体やクラス）が定義されている。

- 人体モデルの体節を表す Segment 構造体
- 人体モデルの関節を表す Joint 構造体
- 人体モデルの骨格を表す Skeleton クラス
- 人体モデルの姿勢を表す Posture クラス

- 人体モデルの動作を表す Motion クラス
- 人体モデルのキーフレーム動作を表す KeyframeMotion クラス

これらのデータ構造は、基本的には構造体の扱い（そのためメンバ変数は全て public としている）だが、コンストラクタやメソッドが定義できた方が便利であるため、骨格・姿勢・動作はクラスとして定義している。

人体モデルの骨格（Skeleton クラス）は、複数の体節（Segment 構造体）と関節（Joint 構造体）の配列によって表現する（ソースコード 1）。体節は、接続される任意の個数の関節と、その接続位置の情報を持つ。関節は、接続される 2 つの体節の情報を持つ。

ソースコード 1: 骨格のデータ構造

```
//
//  人体モデルの体節を表す構造体
//
struct  Segment
{
   // 体節番号・名前
   int  index;
   string  name;

   // 体節の接続関節数
   int  num_joints;

   // 接続関節の配列 [接続関節番号（ルート体節以外は0番目の要素がルート側）]
   Joint **  joints;

   // 各接続関節の接続位置の配列（体節のローカル座標系）[接続関節番号（ルート体節以外
       は0番目の要素がルート側）]
   Point3f *  joint_positions;

   // 体節の末端位置
   bool  has_site;
   Point3f  site_position;
};


//
//  人体モデルの関節を表す構造体
//
struct  Joint
{
   // 関節番号・名前
   int  index;
   string  name;

   // 接続体節（0番目の要素がルート側、1番目の要素が末端側）
   Segment *  segments[ 2 ];
};


//
//  人体モデルの骨格を表すクラス
//
class  Skeleton
{
  public:
   // 関節数
   int  num_segments;

   // 関節の配列 [関節番号]
   Segment **  segments;
```

```
    // 体節数
    int   num_joints;

    // 体節の配列 [体節番号]
    Joint **  joints;


  public:
    // コンストラクタ・デストラクタ
    Skeleton();
    Skeleton( int s, int j );
    ~Skeleton();
};
```

人体モデルの姿勢（Posture クラス）は、骨格情報（Skeleton クラス）にもとづいて、腰の位置と向き、全関節の回転により表す（ソースコード 2)。一般に向きや回転の表現には複数の方法があるが、本プログラムでは、3×3 回転行列（Matrix3f クラス）を用いる。

ソースコード 2: 姿勢のデータ構造

```
//
//  人体モデルの姿勢を表すクラス
//
class  Posture
{
  public:
    // 骨格モデル
    const Skeleton *  body;

    // ルートの位置
    Point3f  root_pos;

    // ルートの向き（回転行列表現）
    Matrix3f  root_ori;

    // 各関節の相対回転（回転行列表現）[関節番号]
    Matrix3f *  joint_rotations;


  public:
    // コンストラクタ・デストラクタ
    Posture();
    Posture( const Skeleton * b );
    Posture( const Posture & p );
    Posture &operator=( const Posture & p );
    ~Posture();

    // 初期化
    void  Init( const Skeleton * b );
};
```

動作を表すデータ構造として、姿勢情報（Posturek クラス）にもとづいて、一定間隔動作データ（Motion クラス）とキーフレーム動作データ（KeyframeMotion クラス）を定義する（ソースコード 3)。一定間隔動作データは、姿勢の配列により表す。一定間隔動作データは、時刻と姿勢の組の配列により表す。

ソースコード 3: 動作のデータ構造

```
//
//  人体モデルの動作を表すクラス
//
class  Motion
```

```
{
  public:
   // 骨格モデル
   const 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;

   // 動作名
   string  name;


  public:
   // コンストラクタ・デストラクタ
   Motion();
   Motion( const Skeleton * b, int n );
   Motion( const Motion & m );
   Motion &operator=( const Motion & m );
   ~Motion();

   // 初期化
   void  Init( const Skeleton * b, int n );

   // 動作の長さを取得
   float  GetDuration() const { return  num_frames * interval; }

   // 姿勢を取得
   Posture *  GetFrame( int no ) const;
   Posture *  GetFrameTime( float time ) const;
   void  GetPosture( float time, Posture & p ) const;
};


//
//  人体モデルのキーフレーム動作を表すクラス
//
class  KeyframeMotion
{
  public:
   // 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;


  public:
   // コンストラクタ・デストラクタ
   KeyframeMotion();
   KeyframeMotion( const Skeleton * b, int num );
   KeyframeMotion( const KeyframeMotion & m );
   KeyframeMotion &operator=( const KeyframeMotion & m );
```

```
    ~KeyframeMotion();

    // 初期化
    void  Init( const Skeleton * b, int num );
    void  Init( const Skeleton * b, int num, const float * times, const Posture * poses
        );

    // 動作の長さを取得
    float  GetDuration() const;

    // 姿勢を取得
    float  GetKeyTime( int no ) const { return  key_times[ no ]; }
    Posture *  GetKeyPosture( int no ) const { return  & key_poses[ no ]; }
    void  GetPosture( float time, Posture & p ) const;
};
```

1.1.2 骨格・姿勢・動作の基本処理関数

主要な基本処理関数として、以下のグローバル関数が定義・実装されている（ソースコード 4)。

- BVH 動作から骨格モデルを生成（ConstructBVHSkeleton 関数）
- BVH 動作から動作データ（+骨格モデル）を生成（ConstructBVHMotion 関数）
- BVH ファイルを読み込んで動作データ（+骨格モデル）を生成（LoadAndConstructBVHMotion 関数）
- 姿勢の描画（DrawPosture 関数）
- 姿勢の影の描画（DrawPostureShadow 関数）
- 回転行列から水平方向の回転角度を計算（ComputeOrientationAngle 関数）
- 水平方向の回転角度から回転行列を計算（ComputeOrientationMatrix 関数）

最初の 3 つ関数は、BVH 形式の動作データから、骨格モデルや動作データの生成を行うための関数である。BVH ファイルの読み込みには、1.2 節で説明する、BVH 動作クラスを使用する。

次の 2 つの関数（DrawPosture 関数、DrawPostureShadow 関数）は、姿勢の描画を行うための関数である。Posture 型で表された姿勢（+骨格）を受け取り、OpenGL の関数を使って、姿勢や姿勢の影をスティックフィギュアとして描画する。

次の 2 つの関数（ComputeOrientationAngle 関数、ComputeOrientationMatrix 関数）は、水平方向の回転行列（Matrix3f 型）と水平方向の回転角度（float 型）の間の変換を行う関数である。

その他、体節や骨格の探索（FindSegment 関数、FindJoint 関数)、順運動学計算（ForwardKinematics 関数)、姿勢補間（PostureInterpolation 関数）などの基本処理を行う関数が定義されている。

ソースコード 4: 骨格・姿勢・動作の基本処理関数

```
//
//  人体モデルの骨格・姿勢・動作の基本処理
//

// 姿勢の初期化（適当な腰の高さを計算・設定）
void  InitPosture( Posture & posture, const Skeleton * body = NULL );

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

// BVH動作から動作データ（+骨格モデル）を生成
Motion *  CoustructBVHMotion( class BVH * bvh, const Skeleton * bvh_body = NULL );
```

```
// BVHファイルを読み込んで動作データ（＋骨格モデル）を生成
Motion *  LoadAndCoustructBVHMotion( const char * bvh_file_name, const Skeleton *
    bvh_body = NULL );

// BVH動作から姿勢を取得
void  GetBVHPosture( const class BVH * bvh, int frame_no, Posture & posture );

// 骨格モデルから体節を名前で探索
int  FindSegment( const Skeleton * body, const char * segment_name );

// 骨格モデルから関節を名前で探索
int  FindJoint( const Skeleton * body, const char * joint_name );

// 順運動学計算
void  ForwardKinematics( const Posture & posture, vector< Matrix4f > & seg_frame_array,
     vector< Point3f > & joi_pos_array );

// 順運動学計算
void  ForwardKinematics( const Posture & posture, vector< Matrix4f > & seg_frame_array
    );

// 姿勢補間（２つの姿勢を補間）
void  PostureInterpolation( const Posture & p0, const Posture & p1, float ratio,
    Posture & p );

// 変換行列の水平向き（方位角）成分を計算（Ｚ軸の正の方向を０とする時計回りの角度を
    −180〜180 の範囲で求める）
float  ComputeOrientationAngle( const Matrix3f & ori );
float  ComputeOrientationAngle( float dx, float dz );

// 水平回転を表す変換行列を計算（Ｚ軸の正の方向を０とする時計回りの角度を −180〜180 の
    範囲で指定する）
void  ComputeOrientationMatrix( float angle, Matrix3f & ori );
Matrix3f  ComputeOrientationMatrix( float angle );

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

// 骨格モデルの１本のリンクを楕円体で描画
void  DrawBone( float x0, float y0, float z0, float x1, float y1, float z1, float
    radius );

// 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 );
```

1.1.3 ソースコード

SimpleHumn.h/cpp の全体をソースコード 5,6 に示す。

ソースコード 5: 骨格・姿勢・動作のデータ構造と基本処理関数の定義（SimpleHuman.h）

```
1 /**
2 ***  Simple Human Library and Samples for Interactive Character Animation
3 ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプ
      ログラム
4 ***  Copyright (c) 2015−, Masaki OSHITA (www.oshita−lab.org)
5 ***  Released under the MIT license http://opensource.org/licenses/mit−license.php
6 **/
```

```
 7
 8
 9 #ifndef  _SIMPLE_HUMAN_H_
10 #define  _SIMPLE_HUMAN_H_
11
12
13 //
14 //  行列・ベクトルの表現には vecmath C++ライブラリ（http://objectclub.jp/download/
       vecmath1）を使用
15 //
16 #include <Vector3.h>
17 #include <Point3.h>
18 #include <Matrix3.h>
19 #include <Matrix4.h>
20 #include <Color3.h>
21 #include <Color4.h>
22
23 // STL（Standard Template Library）を使用
24 #include <vector>
25 #include <string>
26 using namespace  std;
27
28 // プロトタイプ宣言
29 struct  Segment;
30 struct  Joint;
31 class  Skeleton;
32 class  Posture;
33
34
35 //
36 //  人体モデルの体節を表す構造体
37 //
38 struct  Segment
39 {
40    // 体節番号・名前
41    int  index;
42    string  name;
43
44    // 体節の接続関節数
45    int  num_joints;
46
47    // 接続関節の配列 [接続関節番号（ルート体節以外は0番目の要素がルート側）]
48    Joint **  joints;
49
50    // 各接続関節の接続位置の配列（体節のローカル座標系）[接続関節番号（ルート体節以外
          は0番目の要素がルート側）]
51    Point3f *  joint_positions;
52
53    // 体節の末端位置
54    bool  has_site;
55    Point3f  site_position;
56 };
57
58
59 //
60 //  人体モデルの関節を表す構造体
61 //
62 struct  Joint
63 {
64    // 関節番号・名前
65    int  index;
66    string  name;
67
68    // 接続体節（0番目の要素がルート側、1番目の要素が末端側）
```

```
69      Segment *  segments[ 2 ];
70  };
71
72
73  //
74  //  人体モデルの骨格を表すクラス
75  //
76  class  Skeleton
77  {
78    public:
79      // 関節数
80      int  num_segments;
81
82      // 関節の配列 [関節番号]
83      Segment **  segments;
84
85      // 体節数
86      int  num_joints;
87
88      // 体節の配列 [体節番号]
89      Joint **  joints;
90
91
92    public:
93      // コンストラクタ・デストラクタ
94      Skeleton();
95      Skeleton( int s, int j );
96      ~Skeleton();
97  };
98
99
100 //
101 //  人体モデルの姿勢を表すクラス
102 //
103 class  Posture
104 {
105   public:
106     // 骨格モデル
107     const Skeleton *  body;
108
109     // ルートの位置
110     Point3f  root_pos;
111
112     // ルートの向き（回転行列表現）
113     Matrix3f  root_ori;
114
115     // 各関節の相対回転（回転行列表現）[関節番号]
116     Matrix3f *  joint_rotations;
117
118
119   public:
120     // コンストラクタ・デストラクタ
121     Posture();
122     Posture( const Skeleton * b );
123     Posture( const Posture & p );
124     Posture &operator=( const Posture & p );
125     ~Posture();
126
127     // 初期化
128     void  Init( const Skeleton * b );
129 };
130
131
132 //
```

```
133 //  人体モデルの動作を表すクラス
134 //
135 class  Motion
136 {
137   public:
138     // 骨格モデル
139     const Skeleton *  body;
140 
141     // フレーム数
142     int  num_frames;
143 
144     // フレーム間の時間間隔
145     float  interval;
146 
147     // 全フレームの姿勢 [フレーム番号]
148     Posture *  frames;
149 
150     // 動作名
151     string  name;
152 
153 
154   public:
155     // コンストラクタ・デストラクタ
156     Motion();
157     Motion( const Skeleton * b, int n );
158     Motion( const Motion & m );
159     Motion &operator=( const Motion & m );
160     ~Motion();
161 
162     // 初期化
163     void  Init( const Skeleton * b, int n );
164 
165     // 動作の長さを取得
166     float  GetDuration() const { return  num_frames * interval; }
167 
168     // 姿勢を取得
169     Posture *  GetFrame( int no ) const;
170     Posture *  GetFrameTime( float time ) const;
171     void  GetPosture( float time, Posture & p ) const;
172 };
173 
174 
175 //
176 //  人体モデルのキーフレーム動作を表すクラス
177 //
178 class  KeyframeMotion
179 {
180   public:
181     // 骨格モデル
182     const Skeleton *  body;
183 
184     // キーフレーム数
185     int  num_keyframes;
186 
187     // 各キー時刻の配列 [キーフレーム番号]
188     float  *  key_times;
189 
190     // 各キー姿勢の配列 [キーフレーム番号]
191     Posture *  key_poses;
192 
193 
194   public:
195     // コンストラクタ・デストラクタ
196     KeyframeMotion();
```

```
197     KeyframeMotion( const Skeleton * b, int num );
198     KeyframeMotion( const KeyframeMotion & m );
199     KeyframeMotion &operator=( const KeyframeMotion & m );
200     ~KeyframeMotion();
201
202     // 初期化
203     void  Init( const Skeleton * b, int num );
204     void  Init( const Skeleton * b, int num, const float * times, const Posture * poses
            );
205
206     // 動作の長さを取得
207     float  GetDuration() const;
208
209     // 姿勢を取得
210     float  GetKeyTime( int no ) const { return  key_times[ no ]; }
211     Posture *  GetKeyPosture( int no ) const { return  & key_poses[ no ]; }
212     void  GetPosture( float time, Posture & p ) const;
213 };
214
215
216 //
217 //  人体モデルの骨格・姿勢・動作の基本処理
218 //
219
220 // 姿勢の初期化（適当な腰の高さを計算・設定）
221 void  InitPosture( Posture & posture, const Skeleton * body = NULL );
222
223 // BVH動作から骨格モデルを生成
224 Skeleton *  CoustructBVHSkeleton( class BVH * bvh );
225
226 // BVH動作から動作データ（＋骨格モデル）を生成
227 Motion *  CoustructBVHMotion( class BVH * bvh, const Skeleton * bvh_body = NULL );
228
229 // BVHファイルを読み込んで動作データ（＋骨格モデル）を生成
230 Motion *  LoadAndCoustructBVHMotion( const char * bvh_file_name, const Skeleton *
        bvh_body = NULL );
231
232 // BVH動作から姿勢を取得
233 void  GetBVHPosture( const class BVH * bvh, int frame_no, Posture & posture );
234
235 // 骨格モデルから体節を名前で探索
236 int  FindSegment( const Skeleton * body, const char * segment_name );
237
238 // 骨格モデルから関節を名前で探索
239 int  FindJoint( const Skeleton * body, const char * joint_name );
240
241 // 順運動学計算
242 void  ForwardKinematics( const Posture & posture, vector< Matrix4f > & seg_frame_array,
        vector< Point3f > & joi_pos_array );
243
244 // 順運動学計算
245 void  ForwardKinematics( const Posture & posture, vector< Matrix4f > & seg_frame_array
        );
246
247 // 姿勢補間（２つの姿勢を補間）
248 void  PostureInterpolation( const Posture & p0, const Posture & p1, float ratio,
        Posture & p );
249
250 // 変換行列の水平向き（方位角）成分を計算（Ｚ軸の正の方向を０とする時計回りの角度を
        −180〜180 の範囲で求める）
251 float  ComputeOrientationAngle( const Matrix3f & ori );
252 float  ComputeOrientationAngle( float dx, float dz );
253
254 // 水平回転を表す変換行列を計算（Ｚ軸の正の方向を０とする時計回りの角度を −180〜180 の
```

```
    範囲で指定する)
255 void  ComputeOrientationMatrix( float angle, Matrix3f & ori );
256 Matrix3f  ComputeOrientationMatrix( float angle );
257
258 // 姿勢の位置・向きに変換行列を適用
259 void  TransformPosture( const Matrix4f & trans, Posture & posture );
260
261 // 骨格モデルの1本のリンクを楕円体で描画
262 void  DrawBone( float x0, float y0, float z0, float x1, float y1, float z1, float
        radius );
263
264 // 姿勢の描画(スティックフィギュアで描画)
265 void  DrawPosture( const Posture & posture );
266
267 // 姿勢の影の描画(スティックフィギュアで描画)
268 void  DrawPostureShadow( const Posture & posture, const Vector3f & light_dir, const
        Color4f & color );
269
270
271 #endif // _SIMPLE_HUMAN_H_
```

ソースコード 6: 骨格・姿勢・動作のデータ構造と基本処理関数の実装(SimpleHuman.cpp)

```
 1 /**
 2 ***  Simple Human Library and Samples for Interactive Character Animation
 3 ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプ
        ログラム
 4 ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
 5 ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
 6 **/
 7
 8
 9 // ヘッダファイルのインクルード
10 #include "SimpleHuman.h"
11 #include "bvh.h"
12
13 // OpenGL + GLUT を使用
14 #include <gl/glut.h>
15
16 // 標準算術関数・定数の定義
17 #define  _USE_MATH_DEFINES
18 #include <math.h>
19
20
21 // グローバル変数の定義
22
23 // BVHファイルの位置情報に適用するスケーリング比率(デフォルトでは cm→m への変換)
24 const float  bvh_scale = 0.01f;
25
26
27
28 //
29 //  人体モデルの骨格を表すクラス
30 //
31
32 Skeleton::Skeleton()
33 {
34     num_segments = 0;
35     segments = NULL;
36     num_joints = 0;
37     joints = NULL;
38 }
39
40 Skeleton::Skeleton( int s, int j )
```

```
41 {
42    num_segments = s;
43    segments = new Segment*[ num_segments ];
44    for ( int i = 0; i < num_segments; i++ )
45       segments[ i ] = NULL;
46
47    num_joints = j;
48    joints = new Joint*[ num_joints ];
49    for ( int i = 0; i < num_joints; i++ )
50       joints[ i ] = NULL;
51 }
52
53 Skeleton::~Skeleton()
54 {
55    if ( segments )
56    {
57       for ( int i = 0; i < num_segments; i++ )
58       {
59          if ( segments[ i ] )
60          {
61             delete[]  segments[ i ]->joints;
62             delete[]  segments[ i ]->joint_positions;
63             delete  segments[ i ];
64          }
65       }
66       delete[]  segments;
67    }
68    if ( joints )
69    {
70       for ( int i = 0; i < num_joints; i++ )
71       {
72          if ( joints[ i ] )
73          {
74             delete  joints[ i ];
75          }
76       }
77       delete[]  joints;
78    }
79 }
80
81
82 //
83 //  人体モデルの姿勢を表す構造体
84 //
85
86 Posture::Posture()
87 {
88    body = NULL;
89    root_pos.set( 0.0f, 0.0f, 0.0f );
90    root_ori.setIdentity();
91    joint_rotations = NULL;
92 }
93
94 Posture::Posture( const Skeleton * b )
95 {
96    body = b;
97    root_pos.set( 0.0f, 0.0f, 0.0f );
98    root_ori.setIdentity();
99
100   joint_rotations = new Matrix3f[ body->num_joints ];
101   for ( int i = 0; i < body->num_joints; i++ )
102      joint_rotations[ i ].setIdentity();
103 }
104
```

```
105 Posture::Posture( const Posture & p )
106 {
107    body = p.body;
108    root_pos = p.root_pos;
109    root_ori = p.root_ori;
110
111    joint_rotations = new Matrix3f[ body->num_joints ];
112    for ( int i = 0; i < body->num_joints; i++ )
113       joint_rotations[ i ] = p.joint_rotations[ i ];
114 }
115
116 Posture & Posture::operator=( const Posture & p )
117 {
118    if ( !p.body || !p.joint_rotations )
119       return  *this;
120
121    if ( body != p.body )
122    {
123       body = p.body;
124       if ( joint_rotations )
125          delete[]  joint_rotations;
126       joint_rotations = new Matrix3f[ body->num_joints ];
127    }
128
129    root_pos = p.root_pos;
130    root_ori = p.root_ori;
131    for ( int i = 0; i < body->num_joints; i++ )
132       joint_rotations[ i ] = p.joint_rotations[ i ];
133
134    return  *this;
135 }
136
137 void  Posture::Init( const Skeleton * b )
138 {
139    body = b;
140    root_pos.set( 0.0f, 0.0f, 0.0f );
141    root_ori.setIdentity();
142
143    if ( joint_rotations )
144       delete[]  joint_rotations;
145
146    joint_rotations = new Matrix3f[ body->num_joints ];
147    for ( int i = 0; i < body->num_joints; i++ )
148       joint_rotations[ i ].setIdentity();
149 }
150
151 Posture::~Posture()
152 {
153    if ( joint_rotations )
154       delete[]  joint_rotations;
155 }
156
157
158 //
159 //  人体モデルの動作を表すクラス
160 //
161
162 Motion::Motion()
163 {
164    body = NULL;
165    num_frames = 0;
166    interval = 0.033f;
167    frames = NULL;
168 }
```

```
169 |
170 | Motion::Motion( const Skeleton * b, int n ) : Motion()
171 | {
172 |     Init( b, n );
173 | }
174 |
175 | Motion::Motion( const Motion & m )
176 | {
177 |     body = m.body;
178 |     num_frames = m.num_frames;
179 |     interval = m.interval;
180 |
181 |     frames = num_frames ? new Posture[ num_frames ] : NULL;
182 |     for ( int i = 0; i < num_frames; i++ )
183 |         frames[ i ] = m.frames[ i ];
184 | }
185 |
186 | Motion & Motion::operator=( const Motion & m )
187 | {
188 |     body = m.body;
189 |     num_frames = m.num_frames;
190 |     interval = m.interval;
191 |
192 |     if ( frames )
193 |         delete[]  frames;
194 |
195 |     frames = num_frames ? new Posture[ num_frames ] : NULL;
196 |     for ( int i = 0; i < num_frames; i++ )
197 |         frames[ i ] = m.frames[ i ];
198 |
199 |     return  *this;
200 | }
201 |
202 | void  Motion::Init( const Skeleton * b, int n )
203 | {
204 |     body = b;
205 |     num_frames = n;
206 |
207 |     frames = new Posture[ num_frames ];
208 |     for ( int i = 0; i < num_frames; i++ )
209 |         frames[ i ].Init( body );
210 | }
211 |
212 | Motion::~Motion()
213 | {
214 |     if ( frames )
215 |         delete[]  frames;
216 | }
217 |
218 | Posture *  Motion::GetFrame( int no ) const
219 | {
220 |     if ( !frames )
221 |         return  NULL;
222 |
223 |     if ( no <= 0 )
224 |         return & frames[ 0 ];
225 |     else if ( no >= num_frames )
226 |         return & frames[ num_frames - 1 ];
227 |
228 |     return & frames[ no ];
229 | }
230 |
231 | Posture *  Motion::GetFrameTime( float time ) const
232 | {
```

```
233     if ( interval <= 0.0f )
234         return  NULL;
235 
236     return  GetFrame( time / interval );
237 }
238 
239 void  Motion::GetPosture( float time, Posture & p ) const
240 {
241     Posture *  frame = GetFrameTime( time );
242     if ( !frame )
243         return;
244     p = *frame;
245 }
246 
247 
248 //
249 //  人体モデルのキーフレーム動作を表すクラス
250 //
251 
252 KeyframeMotion::KeyframeMotion()
253 {
254     body = NULL;
255     num_keyframes = 0;
256     key_times = NULL;
257     key_poses = NULL;
258 }
259 
260 KeyframeMotion::KeyframeMotion( const Skeleton * b, int num )
261 {
262     Init( b, num );
263 }
264 
265 KeyframeMotion::KeyframeMotion( const KeyframeMotion & m )
266 {
267     body = m.body;
268     num_keyframes = m.num_keyframes;
269 
270     key_times = num_keyframes ? new float[ num_keyframes ] : NULL;
271     key_poses = num_keyframes ? new Posture[ num_keyframes ] : NULL;
272 
273     for ( int i = 0; i < num_keyframes; i++ )
274     {
275         key_times[ i ] = m.key_times[ i ];
276         key_poses[ i ] = m.key_poses[ i ];
277     }
278 }
279 
280 KeyframeMotion & KeyframeMotion::operator=( const KeyframeMotion & m )
281 {
282     if ( key_times )
283         delete[]  key_times;
284     if ( key_poses )
285         delete[]  key_poses;
286 
287     body = m.body;
288     num_keyframes = m.num_keyframes;
289 
290     key_times = num_keyframes ? new float[ num_keyframes ] : NULL;
291     key_poses = num_keyframes ? new Posture[ num_keyframes ] : NULL;
292 
293     for ( int i = 0; i < num_keyframes; i++ )
294     {
295         key_times[ i ] = m.key_times[ i ];
296         key_poses[ i ] = m.key_poses[ i ];
```

```
297     }
298
299     return  *this;
300 }
301
302 KeyframeMotion::~KeyframeMotion()
303 {
304     if ( key_times )
305         delete[]  key_times;
306     if ( key_poses )
307         delete[]  key_poses;
308 }
309
310 void  KeyframeMotion::Init( const Skeleton * b, int num )
311 {
312     if ( key_times )
313         delete[]  key_times;
314     if ( key_poses )
315         delete[]  key_poses;
316
317     body = b;
318     num_keyframes = num;
319
320     key_times = new float[ num_keyframes ];
321     for ( int i = 0; i < num_keyframes; i++ )
322         key_times[ i ] = 0.0f;
323     key_poses = new Posture[ num_keyframes ];
324     for ( int i = 0; i < num_keyframes; i++ )
325         key_poses[ i ].Init( body );
326 }
327
328 void  KeyframeMotion::Init( const Skeleton * b, int num, const float * times, const
      Posture * poses )
329 {
330     if ( key_times )
331         delete[]  key_times;
332     if ( key_poses )
333         delete[]  key_poses;
334
335     body = b;
336     num_keyframes = num;
337
338     key_times = new float[ num_keyframes ];
339     for ( int i = 0; i < num_keyframes; i++ )
340         key_times[ i ] = times[ i ];
341     key_poses = new Posture[ num_keyframes ];
342     for ( int i = 0; i < num_keyframes; i++ )
343         key_poses[ i ] = poses[ i ];
344 }
345
346 // 動作の長さを取得
347 float  KeyframeMotion::GetDuration() const
348 {
349     if ( num_keyframes < 2 )
350         return  0.0f;
351     return  key_times[ num_keyframes - 1 ] - key_times[ 0 ];
352 }
353
354 // 姿勢を取得
355 void  KeyframeMotion::GetPosture( float time, Posture & p ) const
356 {
357     if ( num_keyframes < 1 )
358         return;
359
```

```
360     // 指定時刻がキーフレーム動作の範囲内かを判定
361     if ( time <= key_times[ 0 ] )
362     {
363         p = key_poses[ 0 ];
364         return;
365     }
366     if ( time >= key_times[ num_keyframes - 1 ] )
367     {
368         p = key_poses[ num_keyframes - 1 ];
369         return;
370     }
371
372     // 指定時刻に対応する区間番号を取得
373     int  no = -1;
374     for ( int i = 0; i < num_keyframes - 1; i++ )
375     {
376         if ( ( time >= key_times[ i ] ) && ( time < key_times[ i + 1 ] ) )
377         {
378             no = i;
379             break;
380         }
381     }
382
383     // 補間の割合を計算
384     float  s = ( time - key_times[ no ] ) / ( key_times[ no + 1 ] - key_times[ no ] );
385
386     // 前後のキー姿勢を補間
387     PostureInterpolation( key_poses[ no ], key_poses[ no + 1 ], s, p );
388 }
389
390
391
392 //
393 //  人体モデルの骨格・姿勢・動作の基本処理
394 //
395
396
397 //
398 //  姿勢の初期化（適当な腰の高さを計算・設定）
399 //
400 void  InitPosture( Posture & posture, const Skeleton * body )
401 {
402     if ( !posture.body && !body )
403         return;
404
405     // 骨格情報を設定、姿勢情報を初期化
406     if ( posture.body != body && body )
407         posture = Posture( body );
408
409     // 姿勢情報を初期化
410     else if ( posture.body )
411     {
412         posture.root_pos.set( 0.0f, 0.0f, 0.0f );
413         posture.root_ori.setIdentity();
414         for ( int i=0; i<posture.body->num_joints; i++ )
415             posture.joint_rotations[ i ].setIdentity();
416     }
417
418     // 適当な腰の高さを計算・設定
419     //（最も低い関節の y座標が 0になるように腰の高さを設定）
420     //（本来は、一度計算した高さを記録しておくようにすれば、毎回計算する必要はない）
421     vector< Matrix4f >  seg_frame_array;
422     ForwardKinematics( posture, seg_frame_array );
423     float  root_height = 0.0f;
```

```
424     for ( int i = 0; i < posture.body->num_segments; i++ )
425         if ( root_height > seg_frame_array[ i ].m13 )
426             root_height = seg_frame_array[ i ].m13;
427     posture.root_pos.y = - root_height + 0.05f; // 適当なマージンを加算
428 }
429
430
431 //
432 //  BVH動作から骨格モデルを生成
433 //
434 Skeleton *  CoustructBVHSkeleton( class BVH * bvh )
435 {
436     // 引数チェック
437     if ( !bvh || !bvh->IsLoadSuccess() || ( bvh->GetNumJoint() == 0 ) )
438         return  NULL;
439
440     // 体節・関節数の決定
441     int  num_segments, num_joints;
442     num_segments = bvh->GetNumJoint();
443     num_joints = num_segments - 1;
444
445     // 骨格モデルの初期化
446     Skeleton *  body = new Skeleton( num_segments, num_joints );
447     for ( int i=0; i<num_segments; i++ )
448         body->segments[ i ] = new Segment();
449     for ( int i=0; i<num_joints; i++ )
450         body->joints[ i ] = new Joint();
451
452     // 体節を初期化
453     for ( int i = 0; i < num_segments; i++ )
454     {
455         Segment *  segment = body->segments[ i ];
456
457         // 体節に対応する BVH の関節を取得
458         const BVH::Joint *  parts = bvh->GetJoint( i );
459
460         // 体節番号・名前を設定
461         segment->index = i;
462         segment->name = parts->name;
463
464         // 関節番号・名前を設定
465         if ( i != 0 )
466         {
467             body->joints[ i - 1 ]->index = i - 1;
468             body->joints[ i - 1 ]->name = parts->name;
469         }
470
471         // 体節に接続する関節数（子ノード数 + ルートノード）
472         int  num_connecting_joints;
473         bool  is_root = ( i == 0 );
474         if ( is_root )
475             num_connecting_joints = parts->children.size();
476         else
477             num_connecting_joints = parts->children.size() + 1;
478
479         // 接続関節・接続位置の配列を初期化
480         segment->num_joints = num_connecting_joints;
481         segment->joints = new Joint*[ num_connecting_joints ];
482         segment->joint_positions = new Point3f[ num_connecting_joints ];
483
484         // 各接続関節・接続位置を取得
485         if ( !is_root )
486         {
487             segment->joint_positions[ 0 ].set( 0.0f, 0.0f, 0.0f );
```

```
488             segment->joints[ 0 ] = body->joints[ parts->index - 1 ];
489         }
490         for ( int j = (is_root ? 0 : 1), c = 0; j<num_connecting_joints; j++, c++ )
491         {
492             const BVH::Joint *  child = parts->children[ c ];
493             segment->joints[ j ] = body->joints[ child->index - 1 ];
494             segment->joint_positions[ j ].set( child->offset[ 0 ], child->offset[ 1 ],
                    child->offset[ 2 ] );
495         }
496
497         // 末端位置のオフセットを取得
498         segment->has_site = parts->has_site;
499         if ( parts->has_site )
500             segment->site_position.set( parts->site[ 0 ], parts->site[ 1 ], parts->site[ 2
                    ] );
501
502         // 各関節の接続位置を全接続位置の中心からの相対位置に変換（ルート体節以外）
503         if ( !is_root )
504         {
505             Vector3f  center( 0.0f, 0.0f, 0.0f );
506             for ( int j=0; j<num_connecting_joints; j++ )
507                 center.add( segment->joint_positions[ j ] );
508             if ( parts->has_site )
509                 center.add( segment->site_position );
510             if ( parts->has_site )
511                 center.scale( 1.0f / (float)( num_connecting_joints + 1.0 ) );
512             else
513                 center.scale( 1.0f / (float)num_connecting_joints );
514             for ( int j=0; j<num_connecting_joints; j++ )
515                 segment->joint_positions[ j ].sub( center );
516             if ( parts->has_site )
517                 segment->site_position -= center;
518         }
519         for ( int j = 0; j < num_connecting_joints; j++ )
520             segment->joint_positions[ j ].scale( bvh_scale );
521         if ( parts->has_site )
522             segment->site_position.scale( bvh_scale );
523
524         // 関節の接続体節情報を設定
525         for ( int j = (is_root ? 0 : 1), c = 0; j<num_connecting_joints; j++, c++ )
526         {
527             Joint *  joint = segment->joints[ j ];
528             const BVH::Joint *  child = parts->children[ c ];
529             Segment *  child_segment = body->segments[ child->index ];
530             joint->segments[ 0 ] = segment;
531             joint->segments[ 1 ] = child_segment;
532         }
533     }
534
535     // 生成した骨格モデルを返す
536     return  body;
537 }
538
539
540 //
541 //  BVH動作から動作データ（＋骨格モデル）を生成
542 //
543 Motion *  CoustructBVHMotion( class BVH * bvh, const Skeleton * bvh_body )
544 {
545     // 骨格モデルを生成（生成済みの骨格モデルが入力された場合は省略）
546     const Skeleton *  body = bvh_body;
547     if ( !body )
548     {
549         body = CoustructBVHSkeleton( bvh );
```

```
550         if ( !body )
551             return  NULL;
552     }
553
554     // 動作データの初期化
555     int   num_frames = bvh->GetNumFrame();
556     if ( num_frames == 0 )
557         return  NULL;
558     Motion *  motion = new Motion( body, num_frames );
559     motion->interval = bvh->GetInterval();
560     motion->name = bvh->GetMotionName();
561
562     // 各フレームの姿勢をBVH動作から取得
563     for ( int i = 0; i < num_frames; i++ )
564         GetBVHPosture( bvh, i, motion->frames[ i ] );
565
566     // 生成した動作データを返す
567     return  motion;
568 }
569
570
571 //
572 //  BVHファイルを読み込んで動作データ（＋骨格モデル）を生成
573 //
574 Motion *  LoadAndCoustructBVHMotion( const char * bvh_file_name, const Skeleton *
        bvh_body )
575 {
576     // BVH動作データを読み込み
577     BVH  bvh( bvh_file_name );
578
579     // 読み込みに失敗したら終了
580     if ( !bvh.IsLoadSuccess() )
581         return  NULL;
582
583     // BVH動作から骨格モデルと動作データを生成
584     Motion *  motion = CoustructBVHMotion( &bvh, bvh_body );
585
586     // 生成した動作データを返す
587     return  motion;
588 }
589
590
591 //
592 //  BVH動作の関節回転を計算（オイラー角表現から回転行列表現に変換）
593 //
594 void  ComputeBVHJointRotation( int num_channels, const BVH::Channel * const * channels,
        const float * angles, Matrix3f & rot )
595 {
596     Matrix3f  axis_rot;
597     rot.setIdentity();
598     for ( int i = 0; i < num_channels; i++ )
599     {
600         switch ( channels[ i ]->type )
601         {
602           case BVH::X_ROTATION:
603             axis_rot.rotX( angles[ i ] );
604             break;
605           case BVH::Y_ROTATION:
606             axis_rot.rotY( angles[ i ] );
607             break;
608           case BVH::Z_ROTATION:
609             axis_rot.rotZ( angles[ i ] );
610             break;
611           default:
```

```
612             axis_rot.setIdentity();
613         }
614         rot.mul( rot, axis_rot );
615     }
616 }
617
618
619 //
620 //  BVH動作から姿勢を取得
621 //
622 void  GetBVHPosture( const BVH * bvh, int frame_no, Posture & posture )
623 {
624     if ( !bvh || !bvh->IsLoadSuccess() || !posture.body )
625         return;
626     if ( bvh->GetNumJoint() < posture.body->num_joints )
627         return;
628
629     const Skeleton *  body = posture.body;
630     Vector3f  root_pos;
631     Matrix3f  rot;
632     BVH::Channel *  root_rot_channels[ 3 ];
633     const BVH::Joint *  bvh_joint = NULL;
634     int  num_channels = 0;
635     float  angles[ 6 ];
636
637     // ルート関節の位置・向きを取得
638     const BVH::Joint *  bvh_root = bvh->GetJoint( 0 );
639     int  c = 0;
640     for ( int j = 0; j < bvh_root->channels.size(); j++ )
641     {
642         switch ( bvh_root->channels[ j ]->type )
643         {
644             case BVH::X_POSITION:
645             root_pos.x = bvh->GetMotion( frame_no, bvh_root->channels[ j ]->index );
646             break;
647             case BVH::Y_POSITION:
648             root_pos.y = bvh->GetMotion( frame_no, bvh_root->channels[ j ]->index );
649             break;
650             case BVH::Z_POSITION:
651             root_pos.z = bvh->GetMotion( frame_no, bvh_root->channels[ j ]->index );
652             break;
653             case BVH::X_ROTATION:
654             root_rot_channels[ c++ ] = bvh_root->channels[ j ];
655             break;
656             case BVH::Y_ROTATION:
657             root_rot_channels[ c++ ] = bvh_root->channels[ j ];
658             break;
659             case BVH::Z_ROTATION:
660             root_rot_channels[ c++ ] = bvh_root->channels[ j ];
661             break;
662         }
663     }
664     if ( c == 3 )
665     {
666         for ( int j = 0; j < 3; j++ )
667             angles[ j ] = bvh->GetMotion( frame_no, root_rot_channels[ j ]->index ) * M_PI
                    / 180.0f;
668         ComputeBVHJointRotation( 3, root_rot_channels, angles, rot );
669     }
670     else
671         rot.setIdentity();
672
673     // ルート関節の位置・向きを設定
674     root_pos.scale( bvh_scale );
```

```
675     posture.root_pos = root_pos;
676     posture.root_ori = rot;
677 
678     // 各関節の回転を取得
679     for ( int i = 0; i < body->num_joints; i++ )
680     {
681         bvh_joint = bvh->GetJoint( i + 1 );
682         num_channels = bvh_joint->channels.size();
683 
684         // 関節の回転を取得
685         for ( int j = 0; j  <num_channels; j++ )
686             angles[ j ] = bvh->GetMotion( frame_no, bvh_joint->channels[ j ]->index ) *
                    M_PI / 180.0f;
687         ComputeBVHJointRotation( num_channels, &bvh_joint->channels.front(), angles, rot
                );
688 
689         // 関節の回転を設定
690         posture.joint_rotations[ i ] = rot;
691     }
692 }
693 
694 
695 //
696 //  骨格モデルから体節を名前で探索
697 //
698 int  FindSegment( const Skeleton * body, const char * segment_name )
699 {
700     for ( int i = 0; i < body->num_segments; i ++ )
701     {
702         if ( strcmp( body->segments[ i ]->name.c_str(), segment_name ) == 0 )
703         {
704             return  i;
705         }
706     }
707     return  -1;
708 }
709 
710 
711 //
712 //  骨格モデルから関節を名前で探索
713 //
714 int  FindJoint( const Skeleton * body, const char * joint_name )
715 {
716     for ( int i = 0; i < body->num_joints; i ++ )
717     {
718         if ( strcmp( body->joints[ i ]->name.c_str(), joint_name ) == 0 )
719         {
720             return  i;
721         }
722     }
723     return  -1;
724 }
725 
726 
727 //
728 //  順運動学計算のための反復計算（ルート体節から末端体節に向かって繰り返し再帰呼び出し
        ）
729 //
730 void  ForwardKinematicsIteration(
731     const Segment *  segment, const Segment * prev_segment, const Posture & posture,
732     Matrix4f * seg_frame_array, Point3f * joi_pos_array = NULL )
733 {
734     // 骨格情報
735     const Skeleton *  body = posture.body;
```

```
736
737     // 次の関節・体節
738     Joint *  next_joint;
739     Segment *  next_segment;
740
741     // 次の関節・体節の変換行列
742     Matrix4f  frame;
743
744     // 計算用のベクトル・行列
745     Vector3f  pos;
746     Matrix4f  mat;
747
748     // 現在の体節に接続している各関節に対して繰り返し
749     for ( int j = 0; j < segment->num_joints; j++ )
750     {
751         // 次の関節・次の体節を取得
752         next_joint = segment->joints[ j ];
753         if ( next_joint->segments[ 0 ] != segment )
754             next_segment = next_joint->segments[ 0 ];
755         else
756             next_segment = next_joint->segments[ 1 ];
757
758         // 前の体節側（ルート体節側）の関節はスキップ
759         if ( next_segment == prev_segment )
760             continue;
761
762         // 現在の体節の変換行列を取得
763         frame = seg_frame_array[ segment->index ];
764
765         // 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
766         segment->joint_positions[ j ].get( &pos );
767         frame.transform( &pos );
768         frame.m03 += pos.x;
769         frame.m13 += pos.y;
770         frame.m23 += pos.z;
771
772         // 次の関節の位置を設定
773         if ( joi_pos_array )
774             joi_pos_array[ next_joint->index ].set( frame.m03, frame.m13, frame.m23 );
775
776         // 関節の回転行列をかける
777         mat.set( posture.joint_rotations[ next_joint->index ] );
778         frame.mul( frame, mat );
779
780         // 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
781         next_segment->joint_positions[ 0 ].get( &pos );
782         frame.transform( &pos );
783         frame.m03 -= pos.x;
784         frame.m13 -= pos.y;
785         frame.m23 -= pos.z;
786
787         // 次の体節の変換行列を設定
788         seg_frame_array[ next_segment->index ] = frame;
789
790         // 次の体節に対して繰り返し（再帰呼び出し）
791         ForwardKinematicsIteration( next_segment, segment, posture, seg_frame_array,
                joi_pos_array );
792     }
793 }
794
795
796 //
797 //  順運動学計算
798 //
```

```
799 void  ForwardKinematics( const Posture & posture, vector< Matrix4f > & seg_frame_array,
        vector< Point3f > & joi_pos_array )
800 {
801     // 配列初期化
802     seg_frame_array.resize( posture.body->num_segments );
803     joi_pos_array.resize( posture.body->num_joints );
804 
805     // ルート体節の位置・向きを設定
806     seg_frame_array[ 0 ].set( posture.root_ori, posture.root_pos, 1.0f );
807 
808     // Forward Kinematics 計算のための反復計算（ルート体節から末端体節に向かって繰り返し
            計算）
809     ForwardKinematicsIteration( posture.body->segments[ 0 ], NULL, posture, &
            seg_frame_array.front(), &joi_pos_array.front() );
810 }
811 
812 
813 //
814 //  順運動学計算
815 //
816 void  ForwardKinematics( const Posture & posture, vector< Matrix4f > & seg_frame_array
        )
817 {
818     // 配列初期化
819     seg_frame_array.resize( posture.body->num_segments );
820 
821     // ルート体節の位置・向きを設定
822     seg_frame_array[ 0 ].set( posture.root_ori, posture.root_pos, 1.0f );
823 
824     // Forward Kinematics 計算のための反復計算（ルート体節から末端体節に向かって繰り返し
            計算）
825     ForwardKinematicsIteration( posture.body->segments[ 0 ], NULL, posture, &
            seg_frame_array.front() );
826 }
827 
828 
829 //
830 //  姿勢補間（２つの姿勢を補間）
831 //
832 void  PostureInterpolation( const Posture & p0, const Posture & p1, float ratio,
        Posture & p )
833 {
834     // ２つの姿勢の骨格モデルが異なる場合は終了
835     if ( ( p0.body != p1.body ) || ( p0.body != p.body ) )
836         return;
837 
838     // 骨格モデルを取得
839     const Skeleton *  body = p0.body;
840 
841     // 計算用変数
842     Quat4f  q0, q1, q;
843     Vector3f  v0, v1, v;
844 
845     // ２つの姿勢の各関節の回転を補間
846     for ( int i = 0; i < body->num_joints; i++ )
847     {
848         q0.set( p0.joint_rotations[ i ] );
849         q1.set( p1.joint_rotations[ i ] );
850         if ( q0.x * q1.x + q0.y * q1.y + q0.z * q1.z + q0.w * q1.w < 0 )
851             q1.negate( q1 );
852         q.interpolate( q0, q1, ratio );
853         p.joint_rotations[ i ].set( q );
854     }
855 
```

```
856     // ２つの姿勢のルートの向きを補間
857     q0.set( p0.root_ori );
858     q1.set( p1.root_ori );
859     if ( q0.x * q1.x + q0.y * q1.y + q0.z * q1.z + q0.w * q1.w < 0 )
860        q1.negate( q1 );
861     q.interpolate( q0, q1, ratio );
862     p.root_ori.set( q );
863 
864     // ２つの姿勢のルートの位置を補間
865     v0.set( p0.root_pos );
866     v1.set( p1.root_pos );
867     v.sub( v1, v0 );
868     v.scaleAdd( ratio, v, v0 );
869     p.root_pos.set( v );
870 }
871 
872 
873 //
874 //  変換行列の水平向き（方位角）成分を計算（Ｚ軸の正の方向を０とする時計回りの角度を
        −180〜180 の範囲で求める）
875 //
876 float  ComputeOrientationAngle( const Matrix3f & ori )
877 {
878     return  atan2( ori.m02, ori.m22 ) * 180.0f / M_PI;
879 }
880 
881 float  ComputeOrientationAngle( float dx, float dz )
882 {
883     return  atan2( dx, dz ) * 180.0f / M_PI;
884 }
885 
886 
887 //
888 //  水平回転を表す変換行列を計算（Ｚ軸の正の方向を０とする時計回りの角度を −180〜180 の
        範囲で指定する）
889 //
890 void  ComputeOrientationMatrix( float angle, Matrix3f & ori )
891 {
892     ori.rotY( angle * M_PI / 180.0f );
893 }
894 
895 Matrix3f  ComputeOrientationMatrix( float angle )
896 {
897     Matrix3f  ori;
898     ori.rotY( angle * M_PI / 180.0f );
899     return  ori;
900 }
901 
902 
903 //
904 //  姿勢の位置・向きに変換行列を適用
905 //
906 void  TransformPosture( const Matrix4f & trans, Posture & posture )
907 {
908     // 腰の向きに変換行列を適用
909     Matrix3f  rot;
910     trans.get( &rot );
911     posture.root_ori.mul( rot, posture.root_ori );
912 
913     // 腰の位置に変換行列を適用
914     trans.transform( &posture.root_pos );
915 }
916 
917 
```

```
//
//  骨格モデルの１本のリンクを楕円体で描画
//
void  DrawBone( float x0, float y0, float z0, float x1, float y1, float z1, float
    radius )
{
   // 与えられた２点を結ぶ円柱を描画

   // 円柱の２端点の情報を原点・向き・長さの情報に変換
   GLdouble  dir_x = x1 - x0;
   GLdouble  dir_y = y1 - y0;
   GLdouble  dir_z = z1 - z0;
   GLdouble  bone_length = sqrt( dir_x*dir_x + dir_y*dir_y + dir_z*dir_z );

   // 描画パラメタの設定
   static GLUquadricObj *  quad_obj = NULL;
   if ( quad_obj == NULL )
      quad_obj = gluNewQuadric();
   gluQuadricDrawStyle( quad_obj, GLU_FILL );
   gluQuadricNormals( quad_obj, GLU_SMOOTH );

   glPushMatrix();

   // 平行移動を設定
   glTranslated( ( x0 + x1 ) * 0.5f, ( y0 + y1 ) * 0.5f, ( z0 + z1 ) * 0.5f );

   // 以下、回転を表す行列を計算

   // ｚ軸を単位ベクトルに正規化
   double  length;
   length = sqrt( dir_x*dir_x + dir_y*dir_y + dir_z*dir_z );
   if ( length < 0.0001 ) {
      dir_x = 0.0; dir_y = 0.0; dir_z = 1.0;  length = 1.0;
   }
   dir_x /= length;  dir_y /= length;  dir_z /= length;

   // 基準とするｙ軸の向きを設定
   GLdouble  up_x, up_y, up_z;
   up_x = 0.0;
   up_y = 1.0;
   up_z = 0.0;

   // ｚ軸とｙ軸の外積からｘ軸の向きを計算
   double  side_x, side_y, side_z;
   side_x = up_y * dir_z - up_z * dir_y;
   side_y = up_z * dir_x - up_x * dir_z;
   side_z = up_x * dir_y - up_y * dir_x;

   // ｘ軸を単位ベクトルに正規化
   length = sqrt( side_x*side_x + side_y*side_y + side_z*side_z );
   if ( length < 0.0001 ) {
      side_x = 1.0; side_y = 0.0; side_z = 0.0;  length = 1.0;
   }
   side_x /= length;  side_y /= length;  side_z /= length;

   // ｚ軸とｘ軸の外積からｙ軸の向きを計算
   up_x = dir_y * side_z - dir_z * side_y;
   up_y = dir_z * side_x - dir_x * side_z;
   up_z = dir_x * side_y - dir_y * side_x;

   // 回転行列を設定
   GLdouble  m[16] = { side_x, side_y, side_z, 0.0,
                       up_x,   up_y,   up_z,   0.0,
                       dir_x,  dir_y,  dir_z,  0.0,
```

```
981                         0.0,    0.0,    0.0,    1.0 };
982     glMultMatrixd( m );
983
984     // 円柱の設定
985     GLdouble slices = 16.0; // 円柱の放射状の細分数（デフォルト12）
986     GLdouble stack = 16.0;  // 円柱の輪切りの細分数（デフォルト１）
987
988     // 楕円体を描画
989     glScalef( radius, radius, bone_length * 0.5f );
990     glEnable( GL_NORMALIZE );
991     glutSolidSphere( 1.0f, slices, stack );
992     glDisable( GL_NORMALIZE );
993
994     glPopMatrix();
995 }
996
997
998 //
999 //  姿勢の描画（スティックフィギュアで描画）
1000 //
1001 void  DrawPosture( const Posture & posture )
1002 {
1003     if ( !posture.body )
1004         return;
1005
1006     // 順運動学計算
1007     vector< Matrix4f >  seg_frame_array;
1008     vector< Point3f >  joi_pos_array;
1009     ForwardKinematics( posture, seg_frame_array, joi_pos_array );
1010
1011     float  radius = 0.05f;
1012     Matrix4f  mat;
1013     Vector3f  v1, v2;
1014
1015     // 各体節の描画
1016     for ( int i = 0; i < seg_frame_array.size(); i++ )
1017     {
1018         const Segment *  segment = posture.body->segments[i];
1019         const int  num_joints = segment->num_joints;
1020
1021         // 体節の中心の位置・向きを基準とする変換行列を適用
1022         glPushMatrix();
1023         mat.transpose( seg_frame_array[ i ] );
1024         glMultMatrixf( & mat.m00 );
1025
1026         // １つの関節から末端点へのボーン（楕円体）を描画
1027         if ( ( num_joints == 1 ) && segment->has_site )
1028         {
1029             v1 = segment->joint_positions[ 0 ];
1030             v2 = segment->site_position;
1031             DrawBone( v1.x, v1.y, v1.z, v2.x, v2.y, v2.z, radius );
1032         }
1033         // １つの関節から仮の末端点（重心へのベクトルを２倍した位置）へボーン（楕円体）を
                描画
1034         else if ( ( num_joints == 1 ) && !segment->has_site )
1035         {
1036             v1 = segment->joint_positions[ 0 ];
1037             v2.negate( v1 );
1038             DrawBone( v1.x, v1.y, v1.z, v2.x, v2.y, v2.z, radius );
1039         }
1040         // ２つの関節を接続するボーン（楕円体）を描画
1041         else if ( num_joints == 2 )
1042         {
1043             v1 = segment->joint_positions[ 0 ];
```

```
1044            v2 = segment->joint_positions[ 1 ];
1045            DrawBone( v1.x, v1.y, v1.z, v2.x, v2.y, v2.z, radius );
1046        }
1047        // 重心から各関節へのボーン（楕円体）を描画
1048        else if ( num_joints > 2 )
1049        {
1050            v1.set( 0.0f, 0.0f, 0.0f );
1051            for ( int j=0; j<num_joints; j++ )
1052            {
1053                v2 = segment->joint_positions[ j ];
1054                DrawBone( v1.x, v1.y, v1.z, v2.x, v2.y, v2.z, radius );
1055            }
1056        }
1057
1058        glPopMatrix();
1059    }
1060 }
1061
1062
1063 //
1064 //  姿勢の描画（スティックフィギュアで描画）
1065 //
1066 void  DrawPostureShadow( const Posture & posture, const Vector3f & light_dir, const
        Color4f & color )
1067 {
1068    // 現在の描画設定を取得（描画終了後に元の設定に戻すため）
1069    GLboolean  b_cull_face, b_blend, b_lighting, b_stencil;
1070    glGetBooleanv( GL_CULL_FACE, &b_cull_face );
1071    glGetBooleanv( GL_BLEND, &b_blend );
1072    glGetBooleanv( GL_LIGHTING, &b_lighting );
1073    glGetBooleanv( GL_STENCIL_TEST, &b_stencil );
1074
1075    // 描画設定の変更
1076    if ( b_lighting )
1077        glDisable( GL_LIGHTING );
1078    if ( !b_cull_face )
1079        glEnable( GL_CULL_FACE );
1080    if ( !b_blend )
1081        glEnable( GL_BLEND );
1082
1083    // ブレンディングの設定
1084    glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
1085
1086    // ステンシルバッファの設定
1087    glEnable( GL_STENCIL_TEST );
1088    glStencilFunc( GL_NOTEQUAL, 1, 1 );
1089    glStencilOp( GL_KEEP, GL_KEEP, GL_REPLACE );
1090
1091    // 現在の変換行列を一時保存
1092    glPushMatrix();
1093
1094    // ポリゴンモデルを地面に投影して描画するための変換行列を設定
1095    // 地面への投影行列を計算
1096    float  mat[ 16 ];
1097    mat[ 0 ] = 1.0f;  mat[ 4 ] = - light_dir.x / light_dir.y;  mat[ 8 ] = 0.0f;  mat[ 12
          ] = 0.0f;
1098    mat[ 1 ] = 0.0f;  mat[ 5 ] = 0.0f;                         mat[ 9 ] = 0.0f;  mat[ 13
          ] = 0.01f;
1099    mat[ 2 ] = 0.0f;  mat[ 6 ] = - light_dir.x / light_dir.y;  mat[ 10 ] = 1.0f;  mat[
        14 ] = 0.0f;
1100    mat[ 3 ] = 0.0f;  mat[ 7 ] = 0.0f;                         mat[ 11 ] = 0.0f;  mat[
        15 ] = 1.0f;
1101
1102    // 地面への投影行列をかける
```

```
1103     glMultMatrixf( mat );
1104
1105     // 姿勢の描画（スティックフィギュアで描画）
1106     glColor4f( color.x, color.y, color.z, color.w );
1107     DrawPosture( posture );
1108
1109     // 一時保存しておいた変換行列を復元
1110     glPopMatrix();
1111
1112     // 描画設定を復元
1113     if ( b_lighting )
1114         glEnable( GL_LIGHTING );
1115     if ( b_cull_face )
1116         glEnable( GL_CULL_FACE );
1117     if ( !b_blend )
1118         glDisable( GL_BLEND );
1119     if ( !b_stencil )
1120         glDisable( GL_STENCIL_TEST );
1121 }
```

1.2 BVH動作クラス

BVH形式の動作データを扱うためのBVHクラスが、BVH.h/cppで定義・実装されている。BVHクラスは、本サンプルプログラムとは独立したクラスとしても利用できるようになっており（1.1節で定義されているデータ構造は使用しておらず）、BVH形式に近いデータ構造で骨格・動作情報を管理するようになっている。

BVHクラスは、BVH形式のJOINTやCHANNELに対応する、Joint構造体やChannel構造体のメンバ変数を使って骨格情報を表し、実数（double）型の配列のメンバ変数として動作情報を表す。コンストラクタの引数としてBVHファイルのファイル名を指定することで、BVHファイルの読み込みを行い、メンバ変数に格納する。データアクセスのための関数を利用することで、読み込まれた骨格・動作情報を参照できる。また、指定したフレームの姿勢をOpenGLの関数を使用してスティックフィギュアで描画する、RenderFigure関数も定義・実装されている。

ソースコード 7: BVH動作クラスの定義（bvh.h）

```
1  /**
2  ***  BVH動作ファイルの読み込み・描画クラス
3  ***  Copyright (c) 2004-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7
8  #ifndef  _BVH_H_
9  #define  _BVH_H_
10
11
12 #include <vector>
13 #include <map>
14 #include <string>
15
16 using namespace  std;
17
18
19
20 //
21 //  BVH形式のモーションデータ
22 //
23 class  BVH
24 {
25   public:
```

```
26      /* 内部用構造体 */
27
28      // チャンネルの種類
29      enum ChannelEnum
30      {
31          X_ROTATION, Y_ROTATION, Z_ROTATION,
32          X_POSITION, Y_POSITION, Z_POSITION
33      };
34      struct Joint;
35
36      // チャンネル情報
37      struct Channel
38      {
39          // 対応関節
40          Joint *              joint;
41
42          // チャンネルの種類
43          ChannelEnum          type;
44
45          // チャンネル番号
46          int                  index;
47      };
48
49      // 関節情報
50      struct Joint
51      {
52          // 関節名
53          string               name;
54          // 関節番号
55          int                  index;
56
57          // 関節階層（親関節）
58          Joint *              parent;
59          // 関節階層（子関節）
60          vector< Joint * >    children;
61
62          // 接続位置
63          double               offset[3];
64
65          // 末端位置情報を持つかどうかのフラグ
66          bool                 has_site;
67          // 末端位置
68          double               site[3];
69
70          // 回転軸
71          vector< Channel * >  channels;
72      };
73
74
75      private:
76      // ロードが成功したかどうかのフラグ
77      bool                     is_load_success;
78
79      /* ファイルの情報 */
80      string                   file_name;   // ファイル名
81      string                   motion_name; // 動作名
82
83      /* 階層構造の情報 */
84      int                      num_channel; // チャンネル数
85      vector< Channel * >      channels;    // チャンネル情報 [チャンネル番号]
86      vector< Joint * >        joints;      // 関節情報 [パーツ番号]
87      map< string, Joint * >   joint_index; // 関節名から関節情報へのインデックス
88
89      /* モーションデータの情報 */
```

```
90      int                         num_frame;    // フレーム数
91      double                      interval;     // フレーム間の時間間隔
92      double *                    motion;       // [フレーム番号][チャンネル番号]
93
94
95    public:
96      // コンストラクタ・デストラクタ
97      BVH();
98      BVH( const char * bvh_file_name );
99      ~BVH();
100
101     // 全情報のクリア
102     void  Clear();
103
104     // 全情報を設定（関節の階層構造・動作データの設定）
105     void  Init( const char * name,
106        int n_joi, const Joint ** a_joi, int n_chan, const Channel ** a_chan,
107        int n_frame, double interval, const double * mo );
108
109     // 関節の階層構造の設定
110     void  SetSkeleton( const char * name,
111        int n_joi, const Joint ** a_joi, int n_chan, const Channel ** a_chan );
112
113     // 動作データの設定
114     void  SetMotion( int n_frame, double interval, const double * mo = NULL );
115
116     // BVHファイルのロード
117     void  Load( const char * bvh_file_name );
118
119     // BVHファイルのセーブ
120     void  Save( const char * bvh_file_name );
121
122   public:
123     /*  データアクセス関数  */
124
125     // ロードが成功したかどうかを取得
126     bool  IsLoadSuccess() const { return is_load_success; }
127
128     // ファイルの情報の取得
129     const string &  GetFileName() const { return file_name; }
130     const string &  GetMotionName() const { return motion_name; }
131
132     // 階層構造の情報の取得
133     const int       GetNumJoint() const { return  joints.size(); }
134     const Joint *   GetJoint( int no ) const { return  joints[no]; }
135     const int       GetNumChannel() const { return  channels.size(); }
136     const Channel * GetChannel( int no ) const { return  channels[no]; }
137
138     const Joint *   GetJoint( const string & j ) const  {
139        map< string, Joint * >::const_iterator  i = joint_index.find( j );
140        return  ( i != joint_index.end() ) ? (*i).second : NULL; }
141     const Joint *   GetJoint( const char * j ) const  {
142        map< string, Joint * >::const_iterator  i = joint_index.find( j );
143        return  ( i != joint_index.end() ) ? (*i).second : NULL; }
144
145     // モーションデータの情報の取得
146     int     GetNumFrame() const { return  num_frame; }
147     double  GetInterval() const { return  interval; }
148     double  GetMotion( int f, int c ) const { return  motion[ f*num_channel + c ]; }
149
150     // モーションデータの情報の変更
151     void  SetMotion( int f, int c, double v ) { motion[ f*num_channel + c ] = v; }
152
153   protected:
```

```
154     /*  セーブの補助関数  */
155
156     // 階層構造を再帰的に出力
157     void  OutputHierarchy( ofstream & file, const Joint * joint, int indent_level,
158         vector< int > & channel_list );
159
160   public:
161     /*  姿勢の描画関数  */
162
163     // 指定フレームの姿勢を描画
164     void  RenderFigure( int frame_no, float scale = 1.0f );
165
166     // 指定されたBVH骨格・姿勢を描画（クラス関数）
167     static void  RenderFigure( const Joint * root, const double * data, float scale =
            1.0f );
168
169     // BVH骨格の１本のリンクを描画（クラス関数）
170     static void  RenderBone( float x0, float y0, float z0, float x1, float y1, float z1
            );
171 };
172
173
174
175 #endif // _BVH_H_
```

ソースコード 8: BVH 動作クラスの実装（bvh.cpp）

```
1  /**
2  ***  BVH動作ファイルの読み込み・描画クラス
3  ***  Copyright (c) 2004-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7
8  #include <fstream>
9  #include <string.h>
10
11 #include "BVH.h"
12
13
14 // コントラクタ
15 BVH::BVH()
16 {
17     motion = NULL;
18     Clear();
19 }
20
21 // コントラクタ
22 BVH::BVH( const char * bvh_file_name )
23 {
24     motion = NULL;
25     Clear();
26
27     Load( bvh_file_name );
28 }
29
30 // デストラクタ
31 BVH::~BVH()
32 {
33     Clear();
34 }
35
36
37 // 全情報のクリア
```

```
38 void  BVH::Clear()
39 {
40     int  i;
41     for ( i=0; i<channels.size(); i++ )
42         delete  channels[ i ];
43     for ( i=0; i<joints.size(); i++ )
44         delete  joints[ i ];
45     if ( motion != NULL )
46         delete  motion;
47 
48     is_load_success = false;
49 
50     file_name = "";
51     motion_name = "";
52 
53     num_channel = 0;
54     channels.clear();
55     joints.clear();
56     joint_index.clear();
57 
58     num_frame = 0;
59     interval = 0.0;
60     motion = NULL;
61 }
62 
63 
64 // 全情報を設定
65 void  BVH::Init( const char * name,
66     int n_joi, const Joint ** a_joi, int n_chan, const Channel ** a_chan,
67     int n_frame, double inter, const double * mo )
68 {
69     // 関節の階層構造の設定
70     SetSkeleton( name, n_joi, a_joi, n_chan, a_chan );
71 
72     // 動作データの設定
73     SetMotion( n_frame, inter, mo );
74 }
75 
76 
77 // 関節の階層構造の設定
78 void  BVH::SetSkeleton( const char * name,
79     int n_joi, const Joint ** a_joi, int n_chan, const Channel ** a_chan )
80 {
81     int  i, j;
82 
83     // 初期化
84     Clear();
85 
86     // 基本情報の設定
87     file_name = "";
88     if ( name )
89         motion_name = name;
90     num_channel = n_chan;
91 
92     // チャンネル・関節配列の初期化
93     channels.resize( num_channel );
94     for ( i=0; i<num_channel; i++ )
95         channels[ i ] = new Channel();
96     joints.resize( n_joi );
97     for ( i=0; i<n_joi; i++ )
98         joints[ i ] = new Joint();
99 
100    // チャンネル・関節情報をコピー
101    for ( i=0; i<num_channel; i++ )
```

```
102     {
103        *channels[ i ] = *a_chan[ i ];
104        channels[ i ]->joint = joints[ a_chan[ i ]->joint->index ];
105     }
106     for ( i=0; i<n_joi; i++ )
107     {
108        *joints[ i ] = *a_joi[ i ];
109        for ( j=0; j<joints[ i ]->channels.size(); j++ )
110           joints[ i ]->channels[ j ] = channels[ a_joi[ i ]->channels[ j ]->index ];
111
112        if ( a_joi[ i ]->parent == NULL )
113           joints[ i ]->parent = NULL;
114        else
115           joints[ i ]->parent = joints[ a_joi[ i ]->parent->index ];
116
117        for ( j=0; j<joints[ i ]->children.size(); j++ )
118           joints[ i ]->children[ j ] = joints[ a_joi[ i ]->children[ j ]->index ];
119
120        if ( joints[ i ]->name.size() > 0 )
121           joint_index[ joints[ i ]->name ] = joints[ i ];
122     }
123  }
124
125
126  // 動作データの設定
127  void  BVH::SetMotion( int n_frame, double inter, const double * mo )
128  {
129     num_frame = n_frame;
130     interval = inter;
131     motion = new double[ num_frame * num_channel ];
132     if ( mo != NULL )
133        memcpy( motion, mo, sizeof( double ) * num_frame * num_channel );
134  }
135
136
137  //
138  //  BVHファイルのロード
139  //
140  void  BVH::Load( const char * bvh_file_name )
141  {
142     #define  BUFFER_LENGTH  1024*4
143
144     ifstream  file;
145     char      line[ BUFFER_LENGTH ];
146     char *    token;
147     char      separater[] = " :,\t";
148     vector< Joint * >  joint_stack;
149     Joint *   joint = NULL;
150     Joint *   new_joint = NULL;
151     bool      is_site = false;
152     double    x, y ,z;
153     int       i, j;
154
155     // 初期化
156     Clear();
157
158     // ファイルの情報（ファイル名・動作名）の設定
159     file_name = bvh_file_name;
160     const char *  mn_first = bvh_file_name;
161     const char *  mn_last = bvh_file_name + strlen( bvh_file_name );
162     if ( strrchr( bvh_file_name, '\\' ) != NULL )
163        mn_first = strrchr( bvh_file_name, '\\' ) + 1;
164     else if ( strrchr( bvh_file_name, '/' ) != NULL )
165        mn_first = strrchr( bvh_file_name, '/' ) + 1;
```

```
166     if ( strrchr( bvh_file_name, '.' ) != NULL )
167        mn_last = strrchr( bvh_file_name, '.' );
168     if ( mn_last < mn_first )
169        mn_last = bvh_file_name + strlen( bvh_file_name );
170     motion_name.assign( mn_first, mn_last );
171
172     // ファイルのオープン
173     file.open( bvh_file_name, ios::in );
174     if ( file.is_open() == 0 )  return; // ファイルが開けなかったら終了
175
176     // 階層情報の読み込み
177     while ( ! file.eof() )
178     {
179        // ファイルの最後まできてしまったら異常終了
180        if ( file.eof() )  goto bvh_error;
181
182        // 1行読み込み、先頭の単語を取得
183        file.getline( line, BUFFER_LENGTH );
184        token = strtok( line, separater );
185
186        // 空行の場合は次の行へ
187        if ( token == NULL )  continue;
188
189        // 関節ブロックの開始
190        if ( strcmp( token, "{" ) == 0 )
191        {
192           // 現在の関節をスタックに積む
193           joint_stack.push_back( joint );
194           joint = new_joint;
195           continue;
196        }
197        // 関節ブロックの終了
198        if ( strcmp( token, "}" ) == 0 )
199        {
200           // 現在の関節をスタックから取り出す
201           joint = joint_stack.back();
202           joint_stack.pop_back();
203           is_site = false;
204           continue;
205        }
206
207        // 関節情報の開始
208        if ( ( strcmp( token, "ROOT" ) == 0 ) ||
209             ( strcmp( token, "JOINT" ) == 0 ) )
210        {
211           // 関節データの作成
212           new_joint = new Joint();
213           new_joint->index = joints.size();
214           new_joint->parent = joint;
215           new_joint->has_site = false;
216           new_joint->offset[0] = 0.0;  new_joint->offset[1] = 0.0;  new_joint->offset[2]
                 = 0.0;
217           new_joint->site[0] = 0.0;  new_joint->site[1] = 0.0;  new_joint->site[2] =
                0.0;
218           joints.push_back( new_joint );
219           if ( joint )
220              joint->children.push_back( new_joint );
221
222           // 関節名の読み込み
223           token = strtok( NULL, "" );
224           while ( *token == ' ' )  token ++;
225           new_joint->name = token;
226
227           // インデックスへ追加
```

```
228             joint_index[ new_joint->name ] = new_joint;
229             continue;
230         }
231 
232         // 末端情報の開始
233         if ( ( strcmp( token, "End" ) == 0 ) )
234         {
235             new_joint = joint;
236             is_site = true;
237             continue;
238         }
239 
240         // 関節のオフセット or 末端位置の情報
241         if ( strcmp( token, "OFFSET" ) == 0 )
242         {
243             // 座標値を読み込み
244             token = strtok( NULL, separater );
245             x = token ? atof( token ) : 0.0;
246             token = strtok( NULL, separater );
247             y = token ? atof( token ) : 0.0;
248             token = strtok( NULL, separater );
249             z = token ? atof( token ) : 0.0;
250 
251             // 関節のオフセットに座標値を設定
252             if ( is_site )
253             {
254                 joint->has_site = true;
255                 joint->site[0] = x;
256                 joint->site[1] = y;
257                 joint->site[2] = z;
258             }
259             else
260             // 末端位置に座標値を設定
261             {
262                 joint->offset[0] = x;
263                 joint->offset[1] = y;
264                 joint->offset[2] = z;
265             }
266             continue;
267         }
268 
269         // 関節のチャンネル情報
270         if ( strcmp( token, "CHANNELS" ) == 0 )
271         {
272             // チャンネル数を読み込み
273             token = strtok( NULL, separater );
274             joint->channels.resize( token ? atoi( token ) : 0 );
275 
276             // チャンネル情報を読み込み
277             for ( i=0; i<joint->channels.size(); i++ )
278             {
279                 // チャンネルの作成
280                 Channel *  channel = new Channel();
281                 channel->joint = joint;
282                 channel->index = channels.size();
283                 channels.push_back( channel );
284                 joint->channels[ i ] = channel;
285 
286                 // チャンネルの種類の判定
287                 token = strtok( NULL, separater );
288                 if ( strcmp( token, "Xrotation" ) == 0 )
289                     channel->type = X_ROTATION;
290                 else if ( strcmp( token, "Yrotation" ) == 0 )
291                     channel->type = Y_ROTATION;
```

```
292             else if ( strcmp( token, "Zrotation" ) == 0 )
293                channel->type = Z_ROTATION;
294             else if ( strcmp( token, "Xposition" ) == 0 )
295                channel->type = X_POSITION;
296             else if ( strcmp( token, "Yposition" ) == 0 )
297                channel->type = Y_POSITION;
298             else if ( strcmp( token, "Zposition" ) == 0 )
299                channel->type = Z_POSITION;
300          }
301       }
302
303       // Motionデータのセクションへ移る
304       if ( strcmp( token, "MOTION" ) == 0 )
305          break;
306    }
307
308
309    // モーション情報の読み込み
310    while ( ! file.eof() )
311    {
312       file.getline( line, BUFFER_LENGTH );
313       token = strtok( line, separater );
314       if ( !token )
315          continue;
316       if ( strcmp( token, "Frames" ) == 0 )
317          break;
318    }
319    if ( file.eof() )  goto bvh_error;
320    token = strtok( NULL, separater );
321    if ( token == NULL )  goto bvh_error;
322    num_frame = atoi( token );
323
324    while ( ! file.eof() )
325    {
326       file.getline( line, BUFFER_LENGTH );
327       token = strtok( line, ":" );
328       if ( !token )
329          continue;
330       if ( strcmp( token, "Frame Time" ) == 0 )
331          break;
332    }
333    if ( file.eof() )  goto bvh_error;
334    token = strtok( NULL, separater );
335    if ( token == NULL )  goto bvh_error;
336    interval = atof( token );
337
338    num_channel = channels.size();
339    motion = new double[ num_frame * num_channel ];
340
341    // モーションデータの読み込み
342    for ( i=0; i<num_frame; i++ )
343    {
344       file.getline( line, BUFFER_LENGTH );
345       token = strtok( line, separater );
346       for ( j=0; j<num_channel; j++ )
347       {
348          if ( token == NULL )
349             goto bvh_error;
350          motion[ i*num_channel + j ] = atof( token );
351          token = strtok( NULL, separater );
352       }
353    }
354
355    // ファイルのクローズ
```

```
356     file.close();
357
358     // ロードの成功
359     is_load_success = true;
360
361     return;
362
363 bvh_error:
364     file.close();
365 }
366
367
368 //
369 //  セーブ
370 //
371 void  BVH::Save( const char * bvh_file_name )
372 {
373     int  i, j;
374     ofstream  file;
375
376     // モーションデータを出力するチャンネルの順番
377     vector< int >  channel_order;
378
379     // ファイルのオープン
380     file.open( bvh_file_name, ios::out );
381     if ( file.is_open() == 0 )  return; // ファイルが開けなかったら終了
382
383     // 出力フォーマットの設定
384     file.flags( ios::showpoint | ios::fixed );
385     file.precision( 6 );
386 // int  value_widht = 11;
387
388     // 階層構造の出力
389     file << "HIERARCHY" << endl;
390     OutputHierarchy( file, joints[ 0 ], 0, channel_order );
391
392     // モーションデータの出力
393     file << "MOTION" << endl;
394     file << "Frames: " << num_frame << endl;
395     file << "Frame Time: " << interval << endl;
396     for ( i=0; i<num_frame; i++ )
397     {
398         for ( j=0; j<channel_order.size(); j++ )
399         {
400             // 数値を固定幅で出力する場合は文字幅を設定
401 //          if ( value_widht > 0 )
402 //              file.width( value_widht );
403
404             // モーションデータを出力
405             file << GetMotion( i, channel_order[ j ] );
406
407             // 空白か行末記号を出力
408             if ( j != channel_order.size() - 1 )
409                 file << "  ";
410             else
411                 file << endl;
412         }
413     }
414
415     // ファイルのクローズ
416     file.close();
417 }
418
419
```

```
420 //
421 //  セーブの補助関数
422 //
423
424 // 階層構造を再帰的に出力
425 void  BVH::OutputHierarchy(
426     ofstream & file, const Joint * joint, int indent_level, vector< int > & channel_list
        )
427 {
428     int  i;
429     string  indent, space;
430     Channel *  channel;
431
432     // インデント文字列の初期化
433     indent.assign( indent_level * 4, ' ' );
434     space.assign( "  " );
435
436     // 関節名（ルート名）の出力
437     if ( joint->parent )
438         file << indent << "JOINT" << space << joint->name << endl;
439     else
440         file << indent << "ROOT" << space << joint->name << endl;
441
442     // 関節ブロックの開始
443     file << indent << "{" << endl;
444     indent_level ++;
445     indent.assign( indent_level * 4, ' ' );
446
447     // オフセット位置の出力
448     file << indent << "OFFSET" << space;
449     file << joint->offset[0] << space;
450     file << joint->offset[1] << space;
451     file << joint->offset[2] << endl;
452
453     // チャンネル情報の出力
454     file << indent << "CHANNELS"  << space << joint->channels.size() << space;
455     for ( i=0; i<joint->channels.size(); i++ )
456     {
457         channel = joint->channels[ i ];
458         switch ( channel->type )
459         {
460           case X_ROTATION:
461             file << "Xrotation";  break;
462           case Y_ROTATION:
463             file << "Yrotation";  break;
464           case Z_ROTATION:
465             file << "Zrotation";  break;
466           case X_POSITION:
467             file << "Xposition";  break;
468           case Y_POSITION:
469             file << "Yposition";  break;
470           case Z_POSITION:
471             file << "Zposition";  break;
472         }
473         if ( i != joint->channels.size() - 1 )
474             file << space;
475         else
476             file << endl;
477
478         // 出力チャンネルのリストに追加
479         channel_list.push_back( channel->index );
480     }
481
482     // 末端位置の情報を出力
```

```
483     if ( joint->has_site )
484     {
485         file << indent << "End Site" << endl;
486         file << indent << "{" << endl;
487 
488         indent_level ++;
489         indent.assign( indent_level * 4, ' ' );
490 
491         // オフセット位置の出力
492         file << indent << "OFFSET" << space;
493         file << joint->site[0] << space;
494         file << joint->site[1] << space;
495         file << joint->site[2] << endl;
496 
497         indent_level --;
498         indent.assign( indent_level * 4, ' ' );
499 
500         file << indent << "}" << endl;
501     }
502 
503     // 全ての子関節を再帰的に出力
504     for ( i=0; i<joint->children.size(); i++ )
505     {
506         OutputHierarchy( file, joint->children[ i ], indent_level, channel_list );
507     }
508 
509     // 関節ブロックの終了
510     indent_level --;
511     indent.assign( indent_level * 4, ' ' );
512     file << indent << "}" << endl;
513 }
514 
515 
516 //
517 //  BVH骨格・姿勢の描画関数
518 //
519 
520 #include <math.h>
521 #define  FREEGLUT_STATIC
522 #include <gl/glut.h>
523 
524 
525 // 指定フレームの姿勢を描画
526 void  BVH::RenderFigure( int frame_no, float scale )
527 {
528     // BVH骨格・姿勢を指定して描画
529     RenderFigure( joints[ 0 ], motion + frame_no * num_channel, scale );
530 }
531 
532 
533 // 指定されたBVH骨格・姿勢を描画（クラス関数）
534 void  BVH::RenderFigure( const Joint * joint, const double * data, float scale )
535 {
536     glPushMatrix();
537 
538     // ルート関節の場合は平行移動を適用
539     if ( joint->parent == NULL )
540     {
541         glTranslatef( data[ 0 ] * scale, data[ 1 ] * scale, data[ 2 ] * scale );
542     }
543     // 子関節の場合は親関節からの平行移動を適用
544     else
545     {
546         glTranslatef( joint->offset[ 0 ] * scale, joint->offset[ 1 ] * scale, joint->
```

```
            offset[ 2 ] * scale );
547     }
548
549     // 親関節からの回転を適用（ルート関節の場合はワールド座標からの回転）
550     int  i, j;
551     for ( i=0; i<joint->channels.size(); i++ )
552     {
553        Channel *  channel = joint->channels[ i ];
554        if ( channel->type == X_ROTATION )
555           glRotatef( data[ channel->index ], 1.0f, 0.0f, 0.0f );
556        else if ( channel->type == Y_ROTATION )
557           glRotatef( data[ channel->index ], 0.0f, 1.0f, 0.0f );
558        else if ( channel->type == Z_ROTATION )
559           glRotatef( data[ channel->index ], 0.0f, 0.0f, 1.0f );
560     }
561
562     // リンクを描画
563     // 関節座標系の原点から末端点へのリンクを描画
564     if ( joint->children.size() == 0 )
565     {
566        RenderBone( 0.0f, 0.0f, 0.0f, joint->site[ 0 ] * scale, joint->site[ 1 ] * scale,
               joint->site[ 2 ] * scale );
567     }
568     // 関節座標系の原点から次の関節への接続位置へのリンクを描画
569     if ( joint->children.size() == 1 )
570     {
571        Joint *  child = joint->children[ 0 ];
572        RenderBone( 0.0f, 0.0f, 0.0f, child->offset[ 0 ] * scale, child->offset[ 1 ] *
               scale, child->offset[ 2 ] * scale );
573     }
574     // 全関節への接続位置への中心点から各関節への接続位置へ円柱を描画
575     if ( joint->children.size() > 1 )
576     {
577        // 原点と全関節への接続位置への中心点を計算
578        float  center[ 3 ] = { 0.0f, 0.0f, 0.0f };
579        for ( i=0; i<joint->children.size(); i++ )
580        {
581           Joint *  child = joint->children[ i ];
582           center[ 0 ] += child->offset[ 0 ];
583           center[ 1 ] += child->offset[ 1 ];
584           center[ 2 ] += child->offset[ 2 ];
585        }
586        center[ 0 ] /= joint->children.size() + 1;
587        center[ 1 ] /= joint->children.size() + 1;
588        center[ 2 ] /= joint->children.size() + 1;
589
590        // 原点から中心点へのリンクを描画
591        RenderBone( 0.0f, 0.0f, 0.0f, center[ 0 ] * scale, center[ 1 ] * scale, center[ 2
               ] * scale );
592
593        // 中心点から次の関節への接続位置へのリンクを描画
594        for ( i=0; i<joint->children.size(); i++ )
595        {
596           Joint *  child = joint->children[ i ];
597           RenderBone( center[ 0 ] * scale, center[ 1 ] * scale, center[ 2 ] * scale,
598              child->offset[ 0 ] * scale, child->offset[ 1 ] * scale, child->offset[ 2 ]
                     * scale );
599        }
600     }
601
602     // 子関節に対して再帰呼び出し
603     for ( i=0; i<joint->children.size(); i++ )
604     {
605        RenderFigure( joint->children[ i ], data, scale );
```

```
606     }
607
608     glPopMatrix();
609 }
610
611
612 // BVH骨格の1本のリンクを描画（クラス関数）
613 void  BVH::RenderBone( float x0, float y0, float z0, float x1, float y1, float z1 )
614 {
615     // 与えられた2点を結ぶ円柱を描画
616
617     // 円柱の2端点の情報を原点・向き・長さの情報に変換
618     GLdouble  dir_x = x1 - x0;
619     GLdouble  dir_y = y1 - y0;
620     GLdouble  dir_z = z1 - z0;
621     GLdouble  bone_length = sqrt( dir_x*dir_x + dir_y*dir_y + dir_z*dir_z );
622
623     // 描画パラメタの設定
624     static GLUquadricObj *  quad_obj = NULL;
625     if ( quad_obj == NULL )
626         quad_obj = gluNewQuadric();
627     gluQuadricDrawStyle( quad_obj, GLU_FILL );
628     gluQuadricNormals( quad_obj, GLU_SMOOTH );
629
630     glPushMatrix();
631
632     // 平行移動を設定
633     glTranslated( x0, y0, z0 );
634
635     // 以下、円柱の回転を表す行列を計算
636
637     // z軸を単位ベクトルに正規化
638     double  length;
639     length = sqrt( dir_x*dir_x + dir_y*dir_y + dir_z*dir_z );
640     if ( length < 0.0001 ) {
641         dir_x = 0.0; dir_y = 0.0; dir_z = 1.0;  length = 1.0;
642     }
643     dir_x /= length;  dir_y /= length;  dir_z /= length;
644
645     // 基準とするy軸の向きを設定
646     GLdouble  up_x, up_y, up_z;
647     up_x = 0.0;
648     up_y = 1.0;
649     up_z = 0.0;
650
651     // z軸とy軸の外積からx軸の向きを計算
652     double  side_x, side_y, side_z;
653     side_x = up_y * dir_z - up_z * dir_y;
654     side_y = up_z * dir_x - up_x * dir_z;
655     side_z = up_x * dir_y - up_y * dir_x;
656
657     // x軸を単位ベクトルに正規化
658     length = sqrt( side_x*side_x + side_y*side_y + side_z*side_z );
659     if ( length < 0.0001 ) {
660         side_x = 1.0; side_y = 0.0; side_z = 0.0;  length = 1.0;
661     }
662     side_x /= length;  side_y /= length;  side_z /= length;
663
664     // z軸とx軸の外積からy軸の向きを計算
665     up_x = dir_y * side_z - dir_z * side_y;
666     up_y = dir_z * side_x - dir_x * side_z;
667     up_z = dir_x * side_y - dir_y * side_x;
668
669     // 回転行列を設定
```

```
670     GLdouble  m[16] = { side_x, side_y, side_z, 0.0,
671                         up_x,   up_y,   up_z,   0.0,
672                         dir_x,  dir_y,  dir_z,  0.0,
673                         0.0,    0.0,    0.0,    1.0 };
674     glMultMatrixd( m );
675
676     // 円柱の設定
677     GLdouble radius= 0.01; // 円柱の太さ
678     GLdouble slices = 8.0; // 円柱の放射状の細分数（デフォルト12）
679     GLdouble stack = 3.0;  // 円柱の輪切りの細分数（デフォルト1）
680
681     // 円柱を描画
682     gluCylinder( quad_obj, radius, radius, bone_length, slices, stack );
683
684     glPopMatrix();
685 }
686
687
688
689 // End of BVH.cpp
```

1.2.1 BVH 動作再生サンプルプログラム

BVH クラスを利用して BVH 動作再生を行う独立したサンプルプログラム（bvh_player.cpp）が公開されている。L キーで、再生する BVH 動作の選択・変更を行う。BVH 動作の再生中に、S キーで、動作再生の一時停止と再生を行う。動作再生の一時停止中に、N キーで次のフレーム、P キーで前のフレームに進む。

本サンプルプログラム（bvh_player.cpp）は、OpenGL と GLUT を利用している。キーボードコールバック関数で、BVH 動作の読み込みを行っている。アニメーションコールバック関数で、アニメーションの時間を進める処理を行っている。画面描画コールバック関数で、現在時刻の姿勢の描画を行っている。

ソースコード 9: BVH 動作再生サンプルプログラム（bvh_player.cpp）

```
1  /**
2  ***  BVH Player
3  ***   BVH動作ファイルの読み込み・再生のサンプルプログラム
4  ***   Copyright (c) 2004-2017, Masaki OSHITA (www.oshita-lab.org)
5  ***   Released under the MIT license http://opensource.org/licenses/mit-license.php
6  **/
7
8
9  #ifdef  WIN32
10    #include <windows.h>
11 #endif
12
13 #include <GL/glut.h>
14
15 #include <stdio.h>
16
17 #include "BVH.h"
18
19
20 //
21 //   カメラ・GLUTの入力処理に関するグローバル変数
22 //
23
24 // カメラの回転のための変数
25 static float    camera_yaw = 0.0f;       // Y軸を中心とする回転角度
26 static float    camera_pitch = -20.0f;   // X軸を中心とする回転角度
27 static float    camera_distance = 5.0f; // 中心からカメラの距離
28
```

```
29 // マウスのドラッグのための変数
30 static int     drag_mouse_r = 0; // 右ボタンがドラッグ中かどうかのフラグ（1:ドラッグ中,
       0:非ドラッグ中）
31 static int     drag_mouse_l = 0; // 左ボタンがドラッグ中かどうかのフラグ（1:ドラッグ中,
       0:非ドラッグ中）
32 static int     drag_mouse_m = 0; // 中ボタンがドラッグ中かどうかのフラグ（1:ドラッグ中,
       0:非ドラッグ中）
33 static int     last_mouse_x, last_mouse_y; // 最後に記録されたマウスカーソルの座標
34
35 // ウィンドウのサイズ
36 static int     win_width, win_height;
37
38
39 //
40 //  アニメーション関連のグローバル変数
41 //
42
43 // アニメーション中かどうかを表すフラグ
44 bool   on_animation = true;
45
46 // アニメーションの再生時間
47 float  animation_time = 0.0f;
48
49 // 現在の表示フレーム番号
50 int    frame_no = 0;
51
52 // BVH動作データ
53 BVH *   bvh = NULL;
54
55
56
57 //
58 //  テキストを描画
59 //
60 void  drawMessage( int line_no, const char * message )
61 {
62    int   i;
63    if ( message == NULL )
64       return;
65
66    // 射影行列を初期化（初期化の前に現在の行列を退避）
67    glMatrixMode( GL_PROJECTION );
68    glPushMatrix();
69    glLoadIdentity();
70    gluOrtho2D( 0.0, win_width, win_height, 0.0 );
71
72    // モデルビュー行列を初期化（初期化の前に現在の行列を退避）
73    glMatrixMode( GL_MODELVIEW );
74    glPushMatrix();
75    glLoadIdentity();
76
77    // Zバッファ・ライティングはオフにする
78    glDisable( GL_DEPTH_TEST );
79    glDisable( GL_LIGHTING );
80
81    // メッセージの描画
82    glColor3f( 1.0, 0.0, 0.0 );
83    glRasterPos2i( 8, 24 + 18 * line_no );
84    for ( i=0; message[i]!='\0'; i++ )
85       glutBitmapCharacter( GLUT_BITMAP_HELVETICA_18, message[i] );
86
87    // 設定を全て復元
88    glEnable( GL_DEPTH_TEST );
89    glEnable( GL_LIGHTING );
```

```
90      glMatrixMode( GL_PROJECTION );
91      glPopMatrix();
92      glMatrixMode( GL_MODELVIEW );
93      glPopMatrix();
94  }
95
96
97  //
98  //  ウィンドウ再描画時に呼ばれるコールバック関数
99  //
100 void  display( void )
101 {
102     // 画面をクリア
103     glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT );
104
105     // 変換行列を設定（モデル座標系→カメラ座標系）
106     glMatrixMode( GL_MODELVIEW );
107     glLoadIdentity();
108     glTranslatef( 0.0, 0.0, - camera_distance );
109     glRotatef( - camera_pitch, 1.0, 0.0, 0.0 );
110     glRotatef( - camera_yaw, 0.0, 1.0, 0.0 );
111     glTranslatef( 0.0, -0.5, 0.0 );
112
113     // 光源位置を再設定
114     float  light0_position[] = { 10.0, 10.0, 10.0, 1.0 };
115     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
116
117     // 地面を描画
118     float  size = 1.5f;
119     int  num_x = 10, num_z = 10;
120     double  ox, oz;
121     glBegin( GL_QUADS );
122        glNormal3d( 0.0, 1.0, 0.0 );
123        ox = -(num_x * size) / 2;
124        for ( int x=0; x<num_x; x++, ox+=size )
125        {
126           oz = -(num_z * size) / 2;
127           for ( int z=0; z<num_z; z++, oz+=size )
128           {
129              if ( ((x + z) % 2) == 0 )
130                 glColor3f( 1.0, 1.0, 1.0 );
131              else
132                 glColor3f( 0.8, 0.8, 0.8 );
133              glVertex3d( ox, 0.0, oz );
134              glVertex3d( ox, 0.0, oz+size );
135              glVertex3d( ox+size, 0.0, oz+size );
136              glVertex3d( ox+size, 0.0, oz );
137           }
138        }
139     glEnd();
140
141     // キャラクタを描画
142     glColor3f( 1.0f, 0.0f, 0.0f );
143     if ( bvh )
144        bvh->RenderFigure( frame_no, 0.02f );
145
146     // 時間とフレーム番号を表示
147     char  message[ 64 ];
148     if ( bvh )
149        sprintf( message, "%.2f (%d)", animation_time, frame_no );
150     else
151        sprintf( message, "Press 'L' key to Load a BVH file" );
152     drawMessage( 0, message );
153
```

```
154     // バックバッファに描画した画面をフロントバッファに表示
155     glutSwapBuffers();
156 }
157
158
159 //
160 //  ウィンドウサイズ変更時に呼ばれるコールバック関数
161 //
162 void  reshape( int w, int h )
163 {
164     // ウィンドウ内の描画を行う範囲を設定（ここではウィンドウ全体に描画）
165     glViewport(0, 0, w, h);
166
167     // カメラ座標系→スクリーン座標系への変換行列を設定
168     glMatrixMode( GL_PROJECTION );
169     glLoadIdentity();
170     gluPerspective( 45, (double)w/h, 1, 500 );
171
172     // ウィンドウのサイズを記録（テキスト描画処理のため）
173     win_width = w;
174     win_height = h;
175 }
176
177
178 //
179 // マウスクリック時に呼ばれるコールバック関数
180 //
181 void  mouse( int button, int state, int mx, int my )
182 {
183     // 左ボタンが押されたらドラッグ開始
184     if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
185         drag_mouse_l = 1;
186     // 左ボタンが離されたらドラッグ終了
187     else if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_UP ) )
188         drag_mouse_l = 0;
189
190     // 右ボタンが押されたらドラッグ開始
191     if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_DOWN ) )
192         drag_mouse_r = 1;
193     // 右ボタンが離されたらドラッグ終了
194     else if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_UP ) )
195         drag_mouse_r = 0;
196
197     // 中ボタンが押されたらドラッグ開始
198     if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_DOWN ) )
199         drag_mouse_m = 1;
200     // 中ボタンが離されたらドラッグ終了
201     else if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_UP ) )
202         drag_mouse_m = 0;
203
204     // 再描画
205     glutPostRedisplay();
206
207     // 現在のマウス座標を記録
208     last_mouse_x = mx;
209     last_mouse_y = my;
210 }
211
212
213 //
214 // マウスドラッグ時に呼ばれるコールバック関数
215 //
216 void  motion( int mx, int my )
217 {
```

```
218     // 右ボタンのドラッグ中は視点を回転する
219     if ( drag_mouse_r )
220     {
221         // 前回のマウス座標と今回のマウス座標の差に応じて視点を回転
222 
223         // マウスの横移動に応じてY軸を中心に回転
224         camera_yaw -= ( mx - last_mouse_x ) * 1.0;
225         if ( camera_yaw < 0.0 )
226             camera_yaw += 360.0;
227         else if ( camera_yaw > 360.0 )
228             camera_yaw -= 360.0;
229 
230         // マウスの縦移動に応じてX軸を中心に回転
231         camera_pitch -= ( my - last_mouse_y ) * 1.0;
232         if ( camera_pitch < -90.0 )
233             camera_pitch = -90.0;
234         else if ( camera_pitch > 90.0 )
235             camera_pitch = 90.0;
236     }
237     // 左ボタンのドラッグ中は視点とカメラの距離を変更する
238     if ( drag_mouse_l )
239     {
240         // 前回のマウス座標と今回のマウス座標の差に応じて視点を回転
241 
242         // マウスの縦移動に応じて距離を移動
243         camera_distance += ( my - last_mouse_y ) * 0.2;
244         if ( camera_distance < 2.0 )
245             camera_distance = 2.0;
246     }
247 
248     // 今回のマウス座標を記録
249     last_mouse_x = mx;
250     last_mouse_y = my;
251 
252     // 再描画
253     glutPostRedisplay();
254 }
255 
256 
257 //
258 //  キーボードのキーが押されたときに呼ばれるコールバック関数
259 //
260 void  keyboard( unsigned char key, int mx, int my )
261 {
262     // s キーでアニメーションの停止・再開
263     if ( key == 's' )
264         on_animation = !on_animation;
265 
266     // n キーで次のフレーム
267     if ( ( key == 'n' ) && !on_animation )
268     {
269         animation_time += bvh->GetInterval();
270         frame_no ++;
271         frame_no = frame_no % bvh->GetNumFrame();
272     }
273 
274     // p キーで前のフレーム
275     if ( ( key == 'p' ) && !on_animation && ( frame_no > 0 ) && bvh )
276     {
277         animation_time -= bvh->GetInterval();
278         frame_no --;
279         frame_no = frame_no % bvh->GetNumFrame();
280     }
281 
```

```
282     // r キーでアニメーションのリセット
283     if ( key == 'r' )
284     {
285         animation_time = 0.0f;
286         frame_no = 0;
287     }
288
289     // l キーで再生動作の変更
290     if ( key == 'l' )
291     {
292 #ifdef  WIN32
293         const int  file_name_len = 256;
294         char  file_name[ file_name_len ] = "";
295
296         // ファイルダイアログの設定
297         OPENFILENAME  open_file;
298         memset( &open_file, 0, sizeof(OPENFILENAME) );
299         open_file.lStructSize = sizeof(OPENFILENAME);
300         open_file.hwndOwner = NULL;
301         open_file.lpstrFilter = "BVH Motion Data (*.bvh)\0*.bvh\0All (*.*)\0*.*\0";
302         open_file.nFilterIndex = 1;
303         open_file.lpstrFile = file_name;
304         open_file.nMaxFile = file_name_len;
305         open_file.lpstrTitle = "Select a BVH file";
306         open_file.lpstrDefExt = "bvh";
307         open_file.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;
308
309         // ファイルダイアログを表示
310         BOOL  ret = GetOpenFileName( &open_file );
311
312         // ファイルが指定されたら新しい動作を設定
313         if( ret )
314         {
315             // 動作データを読み込み
316             if ( bvh )
317                 delete  bvh;
318             bvh = new BVH( file_name );
319
320             // 読み込みに失敗したら削除
321             if ( !bvh->IsLoadSuccess() )
322             {
323                 delete  bvh;
324                 bvh = NULL;
325             }
326
327             // アニメーションをリセット
328             animation_time = 0.0f;
329             frame_no = 0;
330         }
331 #endif
332     }
333
334     glutPostRedisplay();
335 }
336
337
338 //
339 //  アイドル時に呼ばれるコールバック関数
340 //
341 void  idle( void )
342 {
343     // アニメーション処理
344     if ( on_animation )
345     {
```

```
346 #ifdef  WIN32
347         // システム時間を取得し、前回からの経過時間に応じてΔｔを決定
348         static DWORD  last_time = 0;
349         DWORD  curr_time = timeGetTime();
350         float  delta = ( curr_time - last_time ) * 0.001f;
351         if ( delta > 0.03f )
352             delta = 0.03f;
353         last_time = curr_time;
354         animation_time += delta;
355 #else
356         // 固定のΔｔを使用
357         animation_time += 0.03f;
358 #endif
359         // 現在のフレーム番号を計算
360         if ( bvh )
361         {
362             frame_no = animation_time / bvh->GetInterval();
363             frame_no = frame_no % bvh->GetNumFrame();
364         }
365         else
366             frame_no = 0;
367 
368         // 再描画の指示を出す（この後で再描画のコールバック関数が呼ばれる）
369         glutPostRedisplay();
370     }
371 }
372 
373 
374 //
375 //  環境初期化関数
376 //
377 void  initEnvironment( void )
378 {
379     // 光源を作成する
380     float  light0_position[] = { 10.0, 10.0, 10.0, 1.0 };
381     float  light0_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
382     float  light0_specular[] = { 1.0, 1.0, 1.0, 1.0 };
383     float  light0_ambient[] = { 0.1, 0.1, 0.1, 1.0 };
384     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
385     glLightfv( GL_LIGHT0, GL_DIFFUSE, light0_diffuse );
386     glLightfv( GL_LIGHT0, GL_SPECULAR, light0_specular );
387     glLightfv( GL_LIGHT0, GL_AMBIENT, light0_ambient );
388     glEnable( GL_LIGHT0 );
389 
390     // 光源計算を有効にする
391     glEnable( GL_LIGHTING );
392 
393     // 物体の色情報を有効にする
394     glEnable( GL_COLOR_MATERIAL );
395 
396     // Ｚテストを有効にする
397     glEnable( GL_DEPTH_TEST );
398 
399     // 背面除去を有効にする
400     glCullFace( GL_BACK );
401     glEnable( GL_CULL_FACE );
402 
403     // 背景色を設定
404     glClearColor( 0.5, 0.5, 0.8, 0.0 );
405 
406     // 初期のＢＶＨ動作データを読み込み
407 // bvh = new BVH( "???.bvh" );
408 }
409 
```

```
410
411 //
412 //  メイン関数（プログラムはここから開始）
413 //
414 int  main( int argc, char ** argv )
415 {
416     // GLUTの初期化
417     glutInit( &argc, argv );
418     glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA | GLUT_STENCIL );
419     glutInitWindowSize( 640, 640 );
420     glutInitWindowPosition( 0, 0 );
421     glutCreateWindow("BVH Player Sample");
422
423     // コールバック関数の登録
424     glutDisplayFunc( display );
425     glutReshapeFunc( reshape );
426     glutMouseFunc( mouse );
427     glutMotionFunc( motion );
428     glutKeyboardFunc( keyboard );
429     glutIdleFunc( idle );
430
431     // 環境初期化
432     initEnvironment();
433
434     // GLUTのメインループに処理を移す
435     glutMainLoop();
436     return 0;
437 }
```

1.3 アプリケーションの基底クラスと GLUT コールバック関数

本サンプルプログラムでは、複数のアプリケーションを切替えて実行できるようになっている。アプリケーションの基底クラス（GLUTBaseApp）と、複数のアプリケーションの切り替えやイベントの呼び出しを行うための GLUT コールバック関数、メイン関数から呼び出されて初期化や GLUT のイベントループの開始を行うフレームワークのメイン関数（SimpleHumanGLUTMain 関数）が、SimpleHumanGLUT.h/cpp で定義・実装されている。

1.3.1 アプリケーションの基底クラス

アプリケーションの基底クラス（GLUTBaseApp）には、以下のような、イベント処理のインターフェースが定義されている。派生クラスでは、これらのインターフェースを継承して、イベント処理を実装する。また、本クラスには、標準的なマウスによる視点操作や描画処理を行うためのメンバ変数・関数も定義・実装されている。

- 初期化（Initialize 関数）
- 開始処理（Start 関数）
- 画面描画（Display 関数）
- アニメーション（Animation 関数）
- ウィンドウサイズ変更（Resize 関数）
- マウスクリック（MouseClick 関数）
- マウスドラッグ（MouseDrag 関数）
- マウス移動（MouseMotion 関数）

- キー入力（Keyboard 関数）
- 特殊キー入力（KeyboardSpecial 関数）

ソースコード 10: アプリケーションの基底クラスの定義（SimpleHumanGLUT.h）

```
1  /**
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプログラム
3  ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7  /**
8  ***  GLUTフレームワーク + アプリケーション基底クラス
9  **/
10
11
12 #ifndef  SIMPLE_HUMAN_GLUT
13 #define  SIMPLE_HUMAN_GLUT
14
15
16 // Windows関数定義の読み込み
17 #include <windows.h>
18
19 // GLUT を使用
20 #include <GL/glut.h>
21
22 //
23 //  行列・ベクトルの表現には vecmath C++ライブラリ（http://objectclub.jp/download/vecmath1）を使用
24 //
25 #include <Vector3.h>
26 #include <Point3.h>
27 #include <Matrix3.h>
28 #include <Matrix4.h>
29 #include <Color3.h>
30 #include <Color4.h>
31
32 // STL を使用
33 #include <vector>
34 #include <string>
35 using namespace std;
36
37
38
39 ////////////////////////////////////////////////////////////////////////////////
40 //
41 //  アプリケーション基底クラス
42 //
43
44
45 //
46 //  アプリケーション基底クラス
47 //
48 class  GLUTBaseApp
49 {
50   protected:
51    // アプリケーション情報
52
53    // アプリケーション名
54    string  app_name;
55
56   protected:
```

```
57      // 視点操作のための変数
58
59      // 視点の方位角
60      float   camera_yaw;
61
62      // 視点の仰角
63      float   camera_pitch;
64
65      // 視点と注視点の距離
66      float   camera_distance;
67
68      // 注視点の位置
69      Point3f   view_center;
70
71    protected:
72      // マウス入力処理のための変数
73
74      // 右・左・中ボタンがドラッグ中かどうかのフラグ
75      bool   drag_mouse_r;
76      bool   drag_mouse_l;
77      bool   drag_mouse_m;
78
79      // 最後に記録されたマウスカーソルの座標
80      int   last_mouse_x;
81      int   last_mouse_y;
82
83    protected:
84      // 画面描画に関する変数
85
86      // 光源位置
87      Point4f   light_pos;
88
89      // 影の方向・色
90      Vector3f   shadow_dir;
91      Color4f   shadow_color;
92
93    protected:
94      // アプリケーション状態の変数
95
96      // ウィンドウのサイズ
97      int   win_width;
98      int   win_height;
99
100     // 初期化フラグ
101     bool   is_initialized;
102
103     // 視点更新フラグ
104     bool   is_view_updated;
105
106
107   public:
108     // コンストラクタ
109     GLUTBaseApp();
110
111     // デストラクタ
112     virtual ~GLUTBaseApp() {}
113
114   public:
115     // アクセサ
116     const string &   GetAppName() { return   app_name; }
117     int   GetWindowWidth() { return   win_width; }
118     int   GetWindowHeight() { return   win_height; }
119     bool   IsInitialized() { return   is_initialized; }
120
```

```
121   public:
122     // イベント処理インターフェース
123
124     //  初期化
125     virtual void  Initialize();
126
127     //  開始・リセット
128     virtual void  Start();
129
130     //  画面描画
131     virtual void  Display();
132
133     // ウィンドウサイズ変更
134     virtual void  Reshape( int w, int h );
135
136     // マウスクリック
137     virtual void  MouseClick( int button, int state, int mx, int my );
138
139     // マウスドラッグ
140     virtual void  MouseDrag( int mx, int my );
141
142     // マウス移動
143     virtual void  MouseMotion( int mx, int my );
144
145     // キーボードのキー押下
146     virtual void  Keyboard( unsigned char key, int mx, int my );
147
148     // キーボードの特殊キー押下
149     virtual void  KeyboardSpecial( unsigned char key, int mx, int my );
150
151     // アニメーション処理
152     virtual void  Animation( float delta );
153
154   protected:
155     // 補助処理
156
157     // 格子模様の床を描画
158     void  DrawFloor( float tile_size, int num_x, int num_z, float r0, float g0, float b0
          , float r1, float g1, float b1 );
159
160     // 文字情報を描画
161     void  DrawTextInformation( int line_no, const char * message, Color3f color =
          Color3f( 1.0f, 0.0f, 0.0f ) );
162 };
163
164
165
166 ////////////////////////////////////////////////////////////////////////////
167 //
168 //  GLUTフレームワークのメイン関数
169 //
170
171
172 //
173 //  GLUTフレームワークのメイン関数（実行するアプリケーションのリストを指定）
174 //
175 int  SimpleHumanGLUTMain( const vector< class GLUTBaseApp * > & app_lists, int argc,
      char ** argv, const char * win_title = NULL, int win_width = 0, int win_height = 0
      );
176
177
178 //
179 //  GLUTフレームワークのメイン関数（実行する一つのアプリケーションを指定）
180 //
```

```
181 int  SimpleHumanGLUTMain( class GLUTBaseApp * app, int argc, char ** argv, const char *
        win_title = NULL, int win_width = 0, int win_height = 0 );
182
183
184
185 #endif // SIMPLE_HUMAN_GLUT
```

ソースコード 11: アプリケーションの基底クラスの実装（SimpleHumanGLUT.cpp）

```
1  /**
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプ
       ログラム
3  ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7  /**
8  ***  GLUTフレームワーク + アプリケーション基底クラス
9  **/
10
11
12 // ヘッダファイルのインクルード
13 #include "SimpleHumanGLUT.h"
14
15
16
17 ////////////////////////////////////////////////////////////////////////////////
18 //
19 //  複数アプリケーションの管理・切替のための変数・関数
20 //
21
22 // 現在実行中のアプリケーション
23 class GLUTBaseApp *    app = NULL;
24
25 // 全アプリケーションのリスト
26 vector< class GLUTBaseApp * >    applications;
27
28 // ソフトウェア説明
29 const char *    software_description = "Human Animation Sample\nCopyright (c) 2015-,
       Masaki OSHITA (www.oshita-lab.org)";
30
31 // 実行アプリケーションの切替関数（プロトタイプ宣言）
32 void  ChangeApp( int app_no );
33
34
35
36 ////////////////////////////////////////////////////////////////////////////////
37 //
38 //  アプリケーション基底クラス
39 //
40
41
42 //
43 //  コンストラクタ
44 //
45 GLUTBaseApp::GLUTBaseApp()
46 {
47     app_name = "Unknown";
48
49     win_width = 0;
50     win_height = 0;
51     is_initialized = false;
52     is_view_updated = true;
53 }
```

```
54
55
56  //
57  //    初期化
58  //
59  void  GLUTBaseApp::Initialize()
60  {
61      is_initialized = true;
62
63      camera_yaw = 0.0f;
64      camera_pitch = -20.0f;
65      camera_distance = 5.0f;
66      view_center.set( 0.0f, 0.0f, 0.0f );
67
68      drag_mouse_r = false;
69      drag_mouse_l = false;
70      drag_mouse_m = false;
71      last_mouse_x = 0;
72      last_mouse_y = 0;
73
74      light_pos.set( 0.0f, 10.0f, 0.0f, 1.0f );
75      shadow_dir.set( 0.0f, 1.0f, 0.0f );
76      shadow_color.set( 0.2f, 0.2f, 0.2f, 0.5f );
77  }
78
79
80  //
81  //    開始・リセット
82  //
83  void  GLUTBaseApp::Start()
84  {
85  }
86
87
88  //
89  //  画面描画
90  //
91  void  GLUTBaseApp::Display()
92  {
93      // 画面をクリア
94      glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT );
95
96      // 変換行列を設定（モデル座標系→カメラ座標系）
97      glMatrixMode( GL_MODELVIEW );
98      glLoadIdentity();
99      glTranslatef( 0.0, 0.0, -camera_distance );
100     glRotatef( -camera_pitch, 1.0, 0.0, 0.0 );
101     glRotatef( -camera_yaw, 0.0, 1.0, 0.0 );
102     glTranslatef( -view_center.x, -0.5, -view_center.z );
103
104     // 光源位置を再設定
105     float  light0_position[] = { light_pos.x, light_pos.y, light_pos.z, light_pos.w };
106     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
107
108     // 格子模様の床を描画
109     DrawFloor( 1.0f, 50, 50, 1.0f, 1.0f, 1.0f, 1.0f, 0.8f, 0.8f );
110 }
111
112
113 //
114 //  ウィンドウサイズ変更
115 //
116 void  GLUTBaseApp::Reshape( int w, int h )
117 {
```

```
118     // ウィンドウのサイズを記録
119     win_width = w;
120     win_height = h;
121
122     // 視点の更新フラグを設定
123     is_view_updated = true;
124 }
125
126
127 //
128 //  マウスクリック
129 //
130 void  GLUTBaseApp::MouseClick( int button, int state, int mx, int my )
131 {
132     // 左ボタンが押されたらドラッグ開始
133     if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
134         drag_mouse_l = true;
135     // 左ボタンが離されたらドラッグ終了
136     else if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_UP ) )
137         drag_mouse_l = false;
138
139     // 右ボタンが押されたらドラッグ開始
140     if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_DOWN ) )
141         drag_mouse_r = true;
142     // 右ボタンが離されたらドラッグ終了
143     else if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_UP ) )
144         drag_mouse_r = false;
145
146     // 中ボタンが押されたらドラッグ開始
147     if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_DOWN ) )
148         drag_mouse_m = true;
149     // 中ボタンが離されたらドラッグ終了
150     else if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_UP ) )
151         drag_mouse_m = false;
152
153     // 現在のマウス座標を記録
154     last_mouse_x = mx;
155     last_mouse_y = my;
156 }
157
158
159 //
160 //   マウスドラッグ
161 //
162 void  GLUTBaseApp::MouseDrag( int mx, int my )
163 {
164     // SHIFTキーの押下状態を取得
165     int  mod = glutGetModifiers();
166
167     // 右ボタンのドラッグ中は視点を回転する
168     if ( drag_mouse_r && !( mod & GLUT_ACTIVE_SHIFT ) )
169     {
170         // 前回のマウス座標と今回のマウス座標の差に応じて視点を回転
171
172         // マウスの横移動に応じてY軸を中心に回転
173         camera_yaw -= ( mx - last_mouse_x ) * 1.0;
174         if ( camera_yaw < 0.0 )
175             camera_yaw += 360.0;
176         else if ( camera_yaw > 360.0 )
177             camera_yaw -= 360.0;
178
179         // マウスの縦移動に応じてX軸を中心に回転
180         camera_pitch -= ( my - last_mouse_y ) * 1.0;
181         if ( camera_pitch < -90.0 )
```

```
182             camera_pitch = -90.0;
183         else if ( camera_pitch > 90.0 )
184             camera_pitch = 90.0;
185 
186         // 視点の更新フラグを設定
187         is_view_updated = true;
188     }
189 
190     // SHIFTキー + 右ボタンのドラッグ中は視点とカメラの距離を変更する
191     if ( drag_mouse_r && ( mod & GLUT_ACTIVE_SHIFT ) )
192     {
193         // 前回のマウス座標と今回のマウス座標の差に応じて視点を回転
194 
195         // マウスの縦移動に応じて距離を移動
196         camera_distance += ( my - last_mouse_y ) * 0.2;
197         if ( camera_distance < 2.0 )
198             camera_distance = 2.0;
199 
200         // 視点の更新フラグを設定
201         is_view_updated = true;
202     }
203 
204     // 今回のマウス座標を記録
205     last_mouse_x = mx;
206     last_mouse_y = my;
207 }
208 
209 
210 //
211 //  マウス移動
212 //
213 void  GLUTBaseApp::MouseMotion( int mx, int my )
214 {
215 }
216 
217 
218 //
219 //  キーボードのキー押下
220 //
221 void  GLUTBaseApp::Keyboard( unsigned char key, int mx, int my )
222 {
223 }
224 
225 
226 //
227 //  キーボードの特殊キー押下
228 //
229 void  GLUTBaseApp::KeyboardSpecial( unsigned char key, int mx, int my )
230 {
231 }
232 
233 
234 //
235 //  アニメーション処理
236 //
237 void  GLUTBaseApp::Animation( float delta )
238 {
239 }
240 
241 
242 //
243 //  以下、補助処理
244 //
245 
```

```
246 |
247 | //
248 | //  格子模様の床を描画
249 | //
250 | void  GLUTBaseApp::DrawFloor( float tile_size, int num_x, int num_z, float r0, float g0
    |     , float b0, float r1, float g1, float b1 )
251 | {
252 |    int  x, z;
253 |    float  ox, oz;
254 |
255 |    glBegin( GL_QUADS );
256 |    glNormal3d( 0.0, 1.0, 0.0 );
257 |
258 |    ox = -( num_x * tile_size ) / 2;
259 |    for ( x = 0; x < num_x; x++ )
260 |    {
261 |       oz = -( num_z * tile_size ) / 2;
262 |       for ( z = 0; z<num_z; z++ )
263 |       {
264 |          if ( ((x + z) % 2) == 0 )
265 |             glColor3f( r0, g0, b0 );
266 |          else
267 |             glColor3f( r1, g1, b1 );
268 |
269 |          glVertex3d( ox, 0.0, oz );
270 |          glVertex3d( ox, 0.0, oz + tile_size );
271 |          glVertex3d( ox + tile_size, 0.0, oz + tile_size );
272 |          glVertex3d( ox + tile_size, 0.0, oz );
273 |
274 |          oz += tile_size;
275 |       }
276 |       ox += tile_size;
277 |    }
278 |    glEnd();
279 | }
280 |
281 |
282 | //
283 | //  文字情報を描画
284 | //
285 | void  GLUTBaseApp::DrawTextInformation( int line_no, const char * message, Color3f
    |     color )
286 | {
287 |    int    i;
288 |    if ( message == NULL )
289 |       return;
290 |
291 |    // 射影行列を初期化（初期化の前に現在の行列を退避）
292 |    glMatrixMode( GL_PROJECTION );
293 |    glPushMatrix();
294 |    glLoadIdentity();
295 |    gluOrtho2D( 0.0, win_width, win_height, 0.0 );
296 |
297 |    // モデルビュー行列を初期化（初期化の前に現在の行列を退避）
298 |    glMatrixMode( GL_MODELVIEW );
299 |    glPushMatrix();
300 |    glLoadIdentity();
301 |
302 |    // Zバッファ・ライティングはオフにする
303 |    glDisable( GL_DEPTH_TEST );
304 |    glDisable( GL_LIGHTING );
305 |
306 |    // メッセージの描画
307 |    glColor3f( color.x, color.y, color.z );
```

```
308     glRasterPos2i( 16, 28 + 24 * line_no );
309     for ( i = 0; message[i] != '\0'; i++ )
310         glutBitmapCharacter( GLUT_BITMAP_HELVETICA_18, message[i] );
311 
312     // 設定を全て復元
313     glEnable( GL_DEPTH_TEST );
314     glEnable( GL_LIGHTING );
315     glMatrixMode( GL_PROJECTION );
316     glPopMatrix();
317     glMatrixMode( GL_MODELVIEW );
318     glPopMatrix();
319 }
320 
321 
322 
323 ////////////////////////////////////////////////////////////////////////////
324 //
325 //  GLUTフレームワーク（イベント処理、初期化・メイン処理）
326 //
327 
328 
329 //
330 //  画面描画時に呼ばれるコールバック関数
331 //
332 void  DisplayCallback( void )
333 {
334     // アプリケーションの描画処理
335     if ( app )
336         app->Display();
337 
338     // バックバッファに描画した画面をフロントバッファに表示
339     glutSwapBuffers();
340 }
341 
342 
343 //
344 //  ウィンドウサイズ変更時に呼ばれるコールバック関数
345 //
346 void  ReshapeCallback( int w, int h )
347 {
348     // ウィンドウ内の描画を行う範囲を設定（ここではウィンドウ全体に描画）
349     glViewport(0, 0, w, h);
350 
351     // カメラ座標系→スクリーン座標系への変換行列を設定
352     glMatrixMode( GL_PROJECTION );
353     glLoadIdentity();
354     gluPerspective( 45, (double)w/h, 1, 500 );
355 
356     // アプリケーションのウィンドウサイズ変更
357     if ( app )
358         app->Reshape( w, h );
359 }
360 
361 
362 //
363 //  マウスクリック時に呼ばれるコールバック関数
364 //
365 void  MouseClickCallback( int button, int state, int mx, int my )
366 {
367     // アプリケーションのマウスクリック
368     if ( app )
369         app->MouseClick( button, state, mx, my );
370 
371     // 再描画
```

```
372     glutPostRedisplay();
373 }
374
375
376 //
377 // マウスドラッグ時に呼ばれるコールバック関数
378 //
379 void  MouseDragCallback( int mx, int my )
380 {
381     // アプリケーションのマウスドラッグ
382     if ( app )
383         app->MouseDrag( mx, my );
384
385     // 再描画
386     glutPostRedisplay();
387 }
388
389
390 //
391 // マウス移動時に呼ばれるコールバック関数
392 //
393 void  MouseMotionCallback( int mx, int my )
394 {
395     // アプリケーションのマウスドラッグ
396     if ( app )
397         app->MouseMotion( mx, my );
398
399     // 再描画
400     glutPostRedisplay();
401 }
402
403
404 //
405 //  キーボードのキーが押されたときに呼ばれるコールバック関数
406 //
407 void  KeyboardCallback( unsigned char key, int mx, int my )
408 {
409     // m キーでモードの切り替え
410     if ( ( key == 'm' ) && app && ( applications.size() > 1 ) )
411     {
412         // 次のアプリケーションを選択
413         int  app_no = 0;
414         for ( int i = 0; i < applications.size(); i++ )
415         {
416             if ( app == applications[i] )
417             {
418                 app_no = i;
419                 break;
420             }
421         }
422         app_no = ( app_no + 1 ) % applications.size();
423
424         // 実行アプリケーションの切替
425         ChangeApp( app_no );
426     }
427
428     // r キーでアプリケーションのリセット
429     if ( key == 'r' )
430     {
431         if ( app )
432             app->Start();
433     }
434
435     // アプリケーションのキー押下
```

```
436     if ( app )
437         app->Keyboard( key, mx, my );
438 
439     // 再描画
440     glutPostRedisplay();
441 }
442 
443 
444 
445 //
446 //  キーボードの特殊キーが押されたときに呼ばれるコールバック関数
447 //
448 void  SpecialKeyboardCallback( int key, int mx, int my )
449 {
450     // アプリケーションの特殊キー押下
451     if ( app )
452         app->KeyboardSpecial( key, mx, my );
453 
454     // 再描画
455     glutPostRedisplay();
456 }
457 
458 
459 //
460 //  アイドル時に呼ばれるコールバック関数
461 //
462 void  IdleCallback( void )
463 {
464     // アニメーション処理
465     if ( app )
466     {
467         // アニメーションの時間変化（Δｔ）を計算
468 #ifdef  WIN32
469         // システム時間を取得し、前回からの経過時間に応じてΔｔを決定
470         static DWORD  last_time = 0;
471         DWORD  curr_time = timeGetTime();
472         float  delta = ( curr_time - last_time ) * 0.001f;
473         if ( delta > 0.03f )
474             delta = 0.03f;
475         last_time = curr_time;
476 #else
477         // 固定のΔｔを使用
478         float  delta = 0.03f;
479 #endif
480 
481         // アプリケーションのアニメーション処理
482         if ( app )
483             app->Animation( delta );
484 
485         // 再描画の指示を出す（この後で再描画のコールバック関数が呼ばれる）
486         glutPostRedisplay();
487     }
488 
489 #ifdef _WIN32
490     // Windows環境では、CTRL+右クリックでもメニューを呼び出せるようにする
491 
492     // 右クリックでのメニュー起動の状態
493     static bool  menu_attached = false;
494 
495     // CTRLキーの押下状態を取得（Win32 API を使用）
496     bool  ctrl = ( GetKeyState( VK_CONTROL ) & 0x80 );
497 
498     // 右クリックでのメニュー起動の登録・解除
499     if ( ctrl && !menu_attached )
```

```
500     {
501         glutAttachMenu( GLUT_RIGHT_BUTTON );
502         menu_attached = true;
503     }
504     else if ( !ctrl && menu_attached )
505     {
506         glutDetachMenu( GLUT_RIGHT_BUTTON );
507         menu_attached = false;
508     }
509 #endif
510 }
511
512
513 //
514 //  実行アプリケーションの切替
515 //
516 void  ChangeApp( int app_no )
517 {
518     if ( ( app_no < 0 ) || ( app_no >= applications.size() ) )
519         return;
520
521     // 現在のウィンドウのサイズを取得
522     int  win_width, win_height;
523     GLUTBaseApp *  curr_app = app;
524     if ( curr_app )
525     {
526         win_width = curr_app->GetWindowWidth();
527         win_height = curr_app->GetWindowHeight();
528     }
529
530     // アプリケーションの初期化・開始
531     app = applications[ app_no ];
532     if ( !app->IsInitialized() )
533         app->Initialize();
534     app->Start();
535     if ( curr_app )
536         app->Reshape( win_width, win_height );
537 }
538
539
540 //
541 //  ポップアップメニュー選択時に呼ばれるコールバック関数
542 //
543 void  MenuCallback( int no )
544 {
545     // 実行アプリケーションの切替
546     if ( ( no >= 0 ) && ( no < applications.size() ) )
547     {
548         ChangeApp( no );
549     }
550
551     // ソフトウェア説明を表示（ダイアログボックスを使用）
552     else if ( no == applications.size() )
553     {
554 #ifdef _WIN32
555         MessageBox( NULL, software_description, "About", MB_OK | MB_ICONINFORMATION );
556 #endif
557     }
558 }
559
560
561 //
562 //  実行アプリケーション切替のためのポップアップメニューの初期化
563 //
```

```
564 void  InitAppMenu()
565 {
566     // メニュー生成
567     int  menu;
568     menu = glutCreateMenu( MenuCallback );
569 
570     // 各アプリケーションのメニュー項目を追加
571     for ( int i = 0; i < applications.size(); i++ )
572     {
573         glutAddMenuEntry( applications[ i ]->GetAppName().c_str(), i );
574     }
575 
576     // ソフトウェア説明のためのメニュー項目を追加
577     glutAddMenuEntry( "About...", applications.size() );
578 
579     // メニュー設定
580     glutSetMenu( menu );
581 
582     // メニュー登録（マウスの中ボタンで表示されるように設定）
583     glutAttachMenu( GLUT_MIDDLE_BUTTON );
584 }
585 
586 
587 //
588 //  レンダリング環境初期化関数
589 //
590 void  initEnvironment( void )
591 {
592     // 光源を作成する
593     float  light0_position[] = { 10.0, 10.0, 10.0, 1.0 };
594     float  light0_diffuse[] = { 0.5, 0.5, 0.5, 1.0 };
595     float  light0_specular[] = { 1.0, 1.0, 1.0, 1.0 };
596     float  light0_ambient[] = { 0.4, 0.4, 0.4, 1.0 };
597     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
598     glLightfv( GL_LIGHT0, GL_DIFFUSE, light0_diffuse );
599     glLightfv( GL_LIGHT0, GL_SPECULAR, light0_specular );
600     glLightfv( GL_LIGHT0, GL_AMBIENT, light0_ambient );
601     glEnable( GL_LIGHT0 );
602 
603     // 光源計算を有効にする
604     glEnable( GL_LIGHTING );
605 
606     // 物体の色情報を有効にする
607     glEnable( GL_COLOR_MATERIAL );
608 
609     // Ｚテストを有効にする
610     glEnable( GL_DEPTH_TEST );
611 
612     // 背面除去を有効にする
613     glCullFace( GL_BACK );
614     glEnable( GL_CULL_FACE );
615 
616     // 背景色を設定
617     glClearColor( 0.5, 0.5, 0.8, 0.0 );
618 }
619 
620 
621 
622 ////////////////////////////////////////////////////////////////////////////
623 //
624 //  ＧＬＵＴフレームワークのメイン関数
625 //
626 
627 
```

```
628 //
629 //  GLUTフレームワークのメイン関数（実行するアプリケーションのリストを指定）
630 //
631 int  SimpleHumanGLUTMain( const vector< class GLUTBaseApp * > & app_lists, int argc,
        char ** argv, const char * win_title, int win_width, int win_height )
632 {
633     // GLUTウィンドウのパラメタの決定（引数で指定されなかった場合はデフォルト値を設定）
634     if ( !win_title || ( strlen( win_title ) == 0 ) )
635         win_title = "Human Animation Sample";
636     if ( win_width <= 0 )
637         win_width = 640;
638     if ( win_height <= 0 )
639         win_height = 640;
640
641     // GLUTの初期化
642     glutInit( &argc, argv );
643     glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA | GLUT_STENCIL );
644     glutInitWindowSize( win_width, win_height );
645     glutInitWindowPosition( 0, 0 );
646     glutCreateWindow( win_title );
647
648     // コールバック関数の登録
649     glutDisplayFunc( DisplayCallback );
650     glutReshapeFunc( ReshapeCallback );
651     glutMouseFunc( MouseClickCallback );
652     glutMotionFunc( MouseDragCallback );
653     glutPassiveMotionFunc( MouseMotionCallback );
654     glutKeyboardFunc( KeyboardCallback );
655     glutSpecialFunc( SpecialKeyboardCallback );
656     glutIdleFunc( IdleCallback );
657
658     // レンダリング環境初期化
659     initEnvironment();
660
661     // 全アプリケーションを登録
662     applications = app_lists;
663
664     // 最初のアプリケーションを実行開始
665     ChangeApp( 0 );
666
667     // 実行アプリケーション切替のためのポップアップメニューの初期化
668     InitAppMenu();
669
670     // GLUTのメインループに処理を移す
671     glutMainLoop();
672     return 0;
673 }
674
675
676 //
677 //  GLUTフレームワークのメイン関数（実行する一つのアプリケーションを指定）
678 //
679 int  SimpleHumanGLUTMain( class GLUTBaseApp * app, int argc, char ** argv, const char *
        win_title, int win_width, int win_height )
680 {
681     vector< class GLUTBaseApp * >     app_lists;
682     app_lists.push_back( app );
683
684     return  SimpleHumanGLUTMain( app_lists, argc, argv, win_title, win_width,
            win_height );
685 }
```

1.3.2 アプリケーションの管理と GLUT コールバック関数

アプリケーションの基底クラス（GLUTBaseApp）に加えて、複数のアプリケーションの管理や切り替えを行うための変数・関数や、GLUT コールバック関数が、SimpleHumanGLUT.cpp で定義・実装されている。

プログラムが含む全てのアプリケーションが、GLUTBaseApp のポインタの可変長配列の変数（applications）により管理されている。また、その中の現在実行中のアプリケーションが、GLUTBaseApp のポインタの変数（app）によりが表されている。ChangeApp 関数で、現在実行中のアプリケーションの変更を行う。以下の GLUT コールバック関数では、現在実行中のアプリケーションのイベント処理を呼び出す。

- 画面描画（DisplayCallback 関数）
- アニメーション（AnimationCallback 関数）
- ウィンドウサイズ変更（ResizeCallback 関数）
- マウスクリック（MouseClickCallback 関数）
- マウスドラッグ（MouseDragCallback 関数）
- マウス移動（MouseMotionCallback 関数）
- キー入力（KeyboardCallback 関数）
- 特殊キー入力（KeyboardSpecialCallback 関数）

1.3.3 フレームワークのメイン関数

プログラムの開始処理を行う SimpleHumanGLUTMain 関数が定義・実装されている。メイン関数から、複数、または、単一のアプリケーションを入力として、SimpleHumanGLUTMain 関数を呼び出すことで、OpenGL・GLUT の初期化、開始時に実行するアプリケーションの選択と初期化、アプリケーションの切替メニューの初期化などを行い、GLUT のメインループに処理を移す。

ソースコード 12: アプリケーションの基底クラスと GLUT コールバック関数の実装（SimpleHumanGLUT.cpp）

```
1  /**
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプログラム
3  ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7  /**
8  ***  GLUTフレームワーク + アプリケーション基底クラス
9  **/
10
11
12 // ヘッダファイルのインクルード
13 #include "SimpleHumanGLUT.h"
14
15
16
17 ////////////////////////////////////////////////////////////////////////////////
18 //
19 //  複数アプリケーションの管理・切替のための変数・関数
20 //
21
22 // 現在実行中のアプリケーション
23 class GLUTBaseApp *     app = NULL;
24
```

```
25 // 全アプリケーションのリスト
26 vector< class GLUTBaseApp * >     applications;
27
28 // ソフトウェア説明
29 const char *     software_description = "Human Animation Sample\nCopyright (c) 2015-,
       Masaki OSHITA (www.oshita-lab.org)";
30
31 // 実行アプリケーションの切替関数（プロトタイプ宣言）
32 void  ChangeApp( int app_no );
33
34
35
36 ////////////////////////////////////////////////////////////////////////////////
37 //
38 //  アプリケーション基底クラス
39 //
40
41
42 //
43 //  コンストラクタ
44 //
45 GLUTBaseApp::GLUTBaseApp()
46 {
47     app_name = "Unknown";
48
49     win_width = 0;
50     win_height = 0;
51     is_initialized = false;
52     is_view_updated = true;
53 }
54
55
56 //
57 //    初期化
58 //
59 void  GLUTBaseApp::Initialize()
60 {
61     is_initialized = true;
62
63     camera_yaw = 0.0f;
64     camera_pitch = -20.0f;
65     camera_distance = 5.0f;
66     view_center.set( 0.0f, 0.0f, 0.0f );
67
68     drag_mouse_r = false;
69     drag_mouse_l = false;
70     drag_mouse_m = false;
71     last_mouse_x = 0;
72     last_mouse_y = 0;
73
74     light_pos.set( 0.0f, 10.0f, 0.0f, 1.0f );
75     shadow_dir.set( 0.0f, 1.0f, 0.0f );
76     shadow_color.set( 0.2f, 0.2f, 0.2f, 0.5f );
77 }
78
79
80 //
81 //    開始・リセット
82 //
83 void  GLUTBaseApp::Start()
84 {
85 }
86
87
```

```
88 //
89 //  画面描画
90 //
91 void  GLUTBaseApp::Display()
92 {
93    // 画面をクリア
94    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT );
95
96    // 変換行列を設定（モデル座標系→カメラ座標系）
97    glMatrixMode( GL_MODELVIEW );
98    glLoadIdentity();
99    glTranslatef( 0.0, 0.0, -camera_distance );
100   glRotatef( -camera_pitch, 1.0, 0.0, 0.0 );
101   glRotatef( -camera_yaw, 0.0, 1.0, 0.0 );
102   glTranslatef( -view_center.x, -0.5, -view_center.z );
103
104   // 光源位置を再設定
105   float  light0_position[] = { light_pos.x, light_pos.y, light_pos.z, light_pos.w };
106   glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
107
108   // 格子模様の床を描画
109   DrawFloor( 1.0f, 50, 50, 1.0f, 1.0f, 1.0f, 1.0f, 0.8f, 0.8f );
110 }
111
112
113 //
114 //  ウィンドウサイズ変更
115 //
116 void  GLUTBaseApp::Reshape( int w, int h )
117 {
118    // ウィンドウのサイズを記録
119    win_width = w;
120    win_height = h;
121
122    // 視点の更新フラグを設定
123    is_view_updated = true;
124 }
125
126
127 //
128 //  マウスクリック
129 //
130 void  GLUTBaseApp::MouseClick( int button, int state, int mx, int my )
131 {
132    // 左ボタンが押されたらドラッグ開始
133    if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
134       drag_mouse_l = true;
135    // 左ボタンが離されたらドラッグ終了
136    else if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_UP ) )
137       drag_mouse_l = false;
138
139    // 右ボタンが押されたらドラッグ開始
140    if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_DOWN ) )
141       drag_mouse_r = true;
142    // 右ボタンが離されたらドラッグ終了
143    else if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_UP ) )
144       drag_mouse_r = false;
145
146    // 中ボタンが押されたらドラッグ開始
147    if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_DOWN ) )
148       drag_mouse_m = true;
149    // 中ボタンが離されたらドラッグ終了
150    else if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_UP ) )
151       drag_mouse_m = false;
```

```
152
153     // 現在のマウス座標を記録
154     last_mouse_x = mx;
155     last_mouse_y = my;
156 }
157
158
159 //
160 //    マウスドラッグ
161 //
162 void  GLUTBaseApp::MouseDrag( int mx, int my )
163 {
164     // SHIFTキーの押下状態を取得
165     int  mod = glutGetModifiers();
166
167     // 右ボタンのドラッグ中は視点を回転する
168     if ( drag_mouse_r && !( mod & GLUT_ACTIVE_SHIFT ) )
169     {
170         // 前回のマウス座標と今回のマウス座標の差に応じて視点を回転
171
172         // マウスの横移動に応じてY軸を中心に回転
173         camera_yaw -= ( mx - last_mouse_x ) * 1.0;
174         if ( camera_yaw < 0.0 )
175             camera_yaw += 360.0;
176         else if ( camera_yaw > 360.0 )
177             camera_yaw -= 360.0;
178
179         // マウスの縦移動に応じてX軸を中心に回転
180         camera_pitch -= ( my - last_mouse_y ) * 1.0;
181         if ( camera_pitch < -90.0 )
182             camera_pitch = -90.0;
183         else if ( camera_pitch > 90.0 )
184             camera_pitch = 90.0;
185
186         // 視点の更新フラグを設定
187         is_view_updated = true;
188     }
189
190     // SHIFTキー + 右ボタンのドラッグ中は視点とカメラの距離を変更する
191     if ( drag_mouse_r && ( mod & GLUT_ACTIVE_SHIFT ) )
192     {
193         // 前回のマウス座標と今回のマウス座標の差に応じて視点を回転
194
195         // マウスの縦移動に応じて距離を移動
196         camera_distance += ( my - last_mouse_y ) * 0.2;
197         if ( camera_distance < 2.0 )
198             camera_distance = 2.0;
199
200         // 視点の更新フラグを設定
201         is_view_updated = true;
202     }
203
204     // 今回のマウス座標を記録
205     last_mouse_x = mx;
206     last_mouse_y = my;
207 }
208
209
210 //
211 //  マウス移動
212 //
213 void  GLUTBaseApp::MouseMotion( int mx, int my )
214 {
215 }
```

```
216 |
217 |
218 | //
219 | //  キーボードのキー押下
220 | //
221 | void  GLUTBaseApp::Keyboard( unsigned char key, int mx, int my )
222 | {
223 | }
224 |
225 |
226 | //
227 | //  キーボードの特殊キー押下
228 | //
229 | void  GLUTBaseApp::KeyboardSpecial( unsigned char key, int mx, int my )
230 | {
231 | }
232 |
233 |
234 | //
235 | //  アニメーション処理
236 | //
237 | void  GLUTBaseApp::Animation( float delta )
238 | {
239 | }
240 |
241 |
242 | //
243 | //  以下、補助処理
244 | //
245 |
246 |
247 | //
248 | //  格子模様の床を描画
249 | //
250 | void  GLUTBaseApp::DrawFloor( float tile_size, int num_x, int num_z, float r0, float g0
    , float b0, float r1, float g1, float b1 )
251 | {
252 |    int  x, z;
253 |    float  ox, oz;
254 |
255 |    glBegin( GL_QUADS );
256 |    glNormal3d( 0.0, 1.0, 0.0 );
257 |
258 |    ox = -( num_x * tile_size ) / 2;
259 |    for ( x = 0; x < num_x; x++ )
260 |    {
261 |       oz = -( num_z * tile_size ) / 2;
262 |       for ( z = 0; z<num_z; z++ )
263 |       {
264 |          if ( ((x + z) % 2) == 0 )
265 |             glColor3f( r0, g0, b0 );
266 |          else
267 |             glColor3f( r1, g1, b1 );
268 |
269 |          glVertex3d( ox, 0.0, oz );
270 |          glVertex3d( ox, 0.0, oz + tile_size );
271 |          glVertex3d( ox + tile_size, 0.0, oz + tile_size );
272 |          glVertex3d( ox + tile_size, 0.0, oz );
273 |
274 |          oz += tile_size;
275 |       }
276 |       ox += tile_size;
277 |    }
278 |    glEnd();
```

```
279 }
280
281
282 //
283 //  文字情報を描画
284 //
285 void  GLUTBaseApp::DrawTextInformation( int line_no, const char * message, Color3f
        color )
286 {
287     int    i;
288     if ( message == NULL )
289         return;
290
291     // 射影行列を初期化（初期化の前に現在の行列を退避）
292     glMatrixMode( GL_PROJECTION );
293     glPushMatrix();
294     glLoadIdentity();
295     gluOrtho2D( 0.0, win_width, win_height, 0.0 );
296
297     // モデルビュー行列を初期化（初期化の前に現在の行列を退避）
298     glMatrixMode( GL_MODELVIEW );
299     glPushMatrix();
300     glLoadIdentity();
301
302     // Zバッファ・ライティングはオフにする
303     glDisable( GL_DEPTH_TEST );
304     glDisable( GL_LIGHTING );
305
306     // メッセージの描画
307     glColor3f( color.x, color.y, color.z );
308     glRasterPos2i( 16, 28 + 24 * line_no );
309     for ( i = 0; message[i] != '\0'; i++ )
310         glutBitmapCharacter( GLUT_BITMAP_HELVETICA_18, message[i] );
311
312     // 設定を全て復元
313     glEnable( GL_DEPTH_TEST );
314     glEnable( GL_LIGHTING );
315     glMatrixMode( GL_PROJECTION );
316     glPopMatrix();
317     glMatrixMode( GL_MODELVIEW );
318     glPopMatrix();
319 }
320
321
322
323 ////////////////////////////////////////////////////////////////////////////
324 //
325 //  GLUTフレームワーク（イベント処理、初期化・メイン処理）
326 //
327
328
329 //
330 //  画面描画時に呼ばれるコールバック関数
331 //
332 void  DisplayCallback( void )
333 {
334     // アプリケーションの描画処理
335     if ( app )
336         app->Display();
337
338     // バックバッファに描画した画面をフロントバッファに表示
339     glutSwapBuffers();
340 }
341
```

```
342
343 //
344 //  ウィンドウサイズ変更時に呼ばれるコールバック関数
345 //
346 void  ReshapeCallback( int w, int h )
347 {
348     // ウィンドウ内の描画を行う範囲を設定（ここではウィンドウ全体に描画）
349     glViewport(0, 0, w, h);
350
351     // カメラ座標系→スクリーン座標系への変換行列を設定
352     glMatrixMode( GL_PROJECTION );
353     glLoadIdentity();
354     gluPerspective( 45, (double)w/h, 1, 500 );
355
356     // アプリケーションのウィンドウサイズ変更
357     if ( app )
358         app->Reshape( w, h );
359 }
360
361
362 //
363 //  マウスクリック時に呼ばれるコールバック関数
364 //
365 void  MouseClickCallback( int button, int state, int mx, int my )
366 {
367     // アプリケーションのマウスクリック
368     if ( app )
369         app->MouseClick( button, state, mx, my );
370
371     // 再描画
372     glutPostRedisplay();
373 }
374
375
376 //
377 // マウスドラッグ時に呼ばれるコールバック関数
378 //
379 void  MouseDragCallback( int mx, int my )
380 {
381     // アプリケーションのマウスドラッグ
382     if ( app )
383         app->MouseDrag( mx, my );
384
385     // 再描画
386     glutPostRedisplay();
387 }
388
389
390 //
391 // マウス移動時に呼ばれるコールバック関数
392 //
393 void  MouseMotionCallback( int mx, int my )
394 {
395     // アプリケーションのマウスドラッグ
396     if ( app )
397         app->MouseMotion( mx, my );
398
399     // 再描画
400     glutPostRedisplay();
401 }
402
403
404 //
405 //  キーボードのキーが押されたときに呼ばれるコールバック関数
```

```
406 //
407 void  KeyboardCallback( unsigned char key, int mx, int my )
408 {
409    // m キーでモードの切り替え
410    if ( ( key == 'm' ) && app && ( applications.size() > 1 ) )
411    {
412       // 次のアプリケーションを選択
413       int  app_no = 0;
414       for ( int i = 0; i < applications.size(); i++ )
415       {
416          if ( app == applications[i] )
417          {
418             app_no = i;
419             break;
420          }
421       }
422       app_no = ( app_no + 1 ) % applications.size();
423
424       // 実行アプリケーションの切替
425       ChangeApp( app_no );
426    }
427
428    // r キーでアプリケーションのリセット
429    if ( key == 'r' )
430    {
431       if ( app )
432          app->Start();
433    }
434
435    // アプリケーションのキー押下
436    if ( app )
437       app->Keyboard( key, mx, my );
438
439    // 再描画
440    glutPostRedisplay();
441 }
442
443
444
445 //
446 //  キーボードの特殊キーが押されたときに呼ばれるコールバック関数
447 //
448 void  SpecialKeyboardCallback( int key, int mx, int my )
449 {
450    // アプリケーションの特殊キー押下
451    if ( app )
452       app->KeyboardSpecial( key, mx, my );
453
454    // 再描画
455    glutPostRedisplay();
456 }
457
458
459 //
460 //  アイドル時に呼ばれるコールバック関数
461 //
462 void  IdleCallback( void )
463 {
464    // アニメーション処理
465    if ( app )
466    {
467       // アニメーションの時間変化（Δｔ）を計算
468 #ifdef  WIN32
469       // システム時間を取得し、前回からの経過時間に応じてΔｔを決定
```

```
470         static DWORD  last_time = 0;
471         DWORD  curr_time = timeGetTime();
472         float  delta = ( curr_time - last_time ) * 0.001f;
473         if ( delta > 0.03f )
474            delta = 0.03f;
475         last_time = curr_time;
476 #else
477         // 固定のΔｔを使用
478         float  delta = 0.03f;
479 #endif
480
481         // アプリケーションのアニメーション処理
482         if ( app )
483            app->Animation( delta );
484
485         // 再描画の指示を出す（この後で再描画のコールバック関数が呼ばれる）
486         glutPostRedisplay();
487      }
488
489 #ifdef _WIN32
490      // Windows環境では、CTRL+右クリックでもメニューを呼び出せるようにする
491
492      // 右クリックでのメニュー起動の状態
493      static bool  menu_attached = false;
494
495      // CTRLキーの押下状態を取得（Win32 API を使用）
496      bool  ctrl = ( GetKeyState( VK_CONTROL ) & 0x80 );
497
498      // 右クリックでのメニュー起動の登録・解除
499      if ( ctrl && !menu_attached )
500      {
501         glutAttachMenu( GLUT_RIGHT_BUTTON );
502         menu_attached = true;
503      }
504      else if ( !ctrl && menu_attached )
505      {
506         glutDetachMenu( GLUT_RIGHT_BUTTON );
507         menu_attached = false;
508      }
509 #endif
510 }
511
512
513 //
514 //  実行アプリケーションの切替
515 //
516 void  ChangeApp( int app_no )
517 {
518      if ( ( app_no < 0 ) || ( app_no >= applications.size() ) )
519         return;
520
521      // 現在のウィンドウのサイズを取得
522      int  win_width, win_height;
523      GLUTBaseApp *  curr_app = app;
524      if ( curr_app )
525      {
526         win_width = curr_app->GetWindowWidth();
527         win_height = curr_app->GetWindowHeight();
528      }
529
530      // アプリケーションの初期化・開始
531      app = applications[ app_no ];
532      if ( !app->IsInitialized() )
533         app->Initialize();
```

```
534     app->Start();
535     if ( curr_app )
536         app->Reshape( win_width, win_height );
537 }
538
539
540 //
541 //  ポップアップメニュー選択時に呼ばれるコールバック関数
542 //
543 void  MenuCallback( int no )
544 {
545     // 実行アプリケーションの切替
546     if ( ( no >= 0 ) && ( no < applications.size() ) )
547     {
548         ChangeApp( no );
549     }
550
551     // ソフトウェア説明を表示（ダイアログボックスを使用）
552     else if ( no == applications.size() )
553     {
554 #ifdef _WIN32
555         MessageBox( NULL, software_description, "About", MB_OK | MB_ICONINFORMATION );
556 #endif
557     }
558 }
559
560
561 //
562 //  実行アプリケーション切替のためのポップアップメニューの初期化
563 //
564 void  InitAppMenu()
565 {
566     // メニュー生成
567     int  menu;
568     menu = glutCreateMenu( MenuCallback );
569
570     // 各アプリケーションのメニュー項目を追加
571     for ( int i = 0; i < applications.size(); i++ )
572     {
573         glutAddMenuEntry( applications[ i ]->GetAppName().c_str(), i );
574     }
575
576     // ソフトウェア説明のためのメニュー項目を追加
577     glutAddMenuEntry( "About...", applications.size() );
578
579     // メニュー設定
580     glutSetMenu( menu );
581
582     // メニュー登録（マウスの中ボタンで表示されるように設定）
583     glutAttachMenu( GLUT_MIDDLE_BUTTON );
584 }
585
586
587 //
588 //  レンダリング環境初期化関数
589 //
590 void  initEnvironment( void )
591 {
592     // 光源を作成する
593     float  light0_position[] = { 10.0, 10.0, 10.0, 1.0 };
594     float  light0_diffuse[] = { 0.5, 0.5, 0.5, 1.0 };
595     float  light0_specular[] = { 1.0, 1.0, 1.0, 1.0 };
596     float  light0_ambient[] = { 0.4, 0.4, 0.4, 1.0 };
597     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
```

```
598     glLightfv( GL_LIGHT0, GL_DIFFUSE, light0_diffuse );
599     glLightfv( GL_LIGHT0, GL_SPECULAR, light0_specular );
600     glLightfv( GL_LIGHT0, GL_AMBIENT, light0_ambient );
601     glEnable( GL_LIGHT0 );
602
603     // 光源計算を有効にする
604     glEnable( GL_LIGHTING );
605
606     // 物体の色情報を有効にする
607     glEnable( GL_COLOR_MATERIAL );
608
609     // Ｚテストを有効にする
610     glEnable( GL_DEPTH_TEST );
611
612     // 背面除去を有効にする
613     glCullFace( GL_BACK );
614     glEnable( GL_CULL_FACE );
615
616     // 背景色を設定
617     glClearColor( 0.5, 0.5, 0.8, 0.0 );
618 }
619
620
621
622 ////////////////////////////////////////////////////////////////////////////
623 //
624 //  GLUTフレームワークのメイン関数
625 //
626
627
628 //
629 //  GLUTフレームワークのメイン関数（実行するアプリケーションのリストを指定）
630 //
631 int  SimpleHumanGLUTMain( const vector< class GLUTBaseApp * > & app_lists, int argc,
        char ** argv, const char * win_title, int win_width, int win_height )
632 {
633     // GLUTウィンドウのパラメタの決定（引数で指定されなかった場合はデフォルト値を設定）
634     if ( !win_title || ( strlen( win_title ) == 0 ) )
635         win_title = "Human Animation Sample";
636     if ( win_width <= 0 )
637         win_width = 640;
638     if ( win_height <= 0 )
639         win_height = 640;
640
641     // GLUTの初期化
642     glutInit( &argc, argv );
643     glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA | GLUT_STENCIL );
644     glutInitWindowSize( win_width, win_height );
645     glutInitWindowPosition( 0, 0 );
646     glutCreateWindow( win_title );
647
648     // コールバック関数の登録
649     glutDisplayFunc( DisplayCallback );
650     glutReshapeFunc( ReshapeCallback );
651     glutMouseFunc( MouseClickCallback );
652     glutMotionFunc( MouseDragCallback );
653     glutPassiveMotionFunc( MouseMotionCallback );
654     glutKeyboardFunc( KeyboardCallback );
655     glutSpecialFunc( SpecialKeyboardCallback );
656     glutIdleFunc( IdleCallback );
657
658     // レンダリング環境初期化
659     initEnvironment();
660
```

```
661     // 全アプリケーションを登録
662     applications = app_lists;
663
664     // 最初のアプリケーションを実行開始
665     ChangeApp( 0 );
666
667     // 実行アプリケーション切替のためのポップアップメニューの初期化
668     InitAppMenu();
669
670     // GLUTのメインループに処理を移す
671     glutMainLoop();
672     return 0;
673 }
674
675
676 //
677 //  GLUTフレームワークのメイン関数（実行する一つのアプリケーションを指定）
678 //
679 int  SimpleHumanGLUTMain( class GLUTBaseApp * app, int argc, char ** argv, const char *
        win_title, int win_width, int win_height )
680 {
681     vector< class GLUTBaseApp * >     app_lists;
682     app_lists.push_back( app );
683
684     return  SimpleHumanGLUTMain( app_lists, argc, argv, win_title, win_width,
            win_height );
685 }
```

1.4 メイン関数

プログラムの開始時に呼び出される main 関数は、SimpleHumanSampleMain.cpp で実装されている。本関数では、プログラムで使用する複数のアプリケーションを生成して、配列に格納する。アプリケーションクラスは、全て GLUTBaseApp クラスを基底クラスとしているため、複数のアプリケーションをまとめて、GLUTBaseApp クラスのポインタの配列として表せる。この配列を引数として渡して、SimpleHumanGLUTMain 関数を呼び出すことで、フレームワークの開始処理を行う。

ソースコード 13: メイン処理（SimpleHumanSampleMain.cpp）

```
1  /**
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理のサンプルプログラム
3  ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4  **/
5
6  /**
7  ***  メイン関数
8  **/
9
10
11 // GLUTフレームワーク＋アプリケーション基底クラスの定義を読み込み
12 #include "SimpleHumanGLUT.h"
13
14 // アプリケーションの定義を読み込み
15 #include "MotionPlaybackApp.h"
16 #include "KeyframeMotionPlaybackApp.h"
17 #include "ForwardKinematicsApp.h"
18 #include "PostureInterpolationApp.h"
19 #include "MotionInterpolationApp.h"
20 #include "MotionTransitionApp.h"
21 #include "MotionDeformationEditApp.h"
22 #include "InverseKinematicsCCDApp.h"
23
```

```
24
25
26 //
27 //  メイン関数（プログラムはここから開始）
28 //
29 int  main( int argc, char ** argv )
30 {
31    // 全アプリケーションのリスト
32    vector< class GLUTBaseApp * >    applications;
33
34    // 全アプリケーションを登録
35    applications.push_back( new MotionPlaybackApp() );
36    applications.push_back( new KeyframeMotionPlaybackApp() );
37    applications.push_back( new ForwardKinematicsApp() );
38    applications.push_back( new PostureInterpolationApp() );
39    applications.push_back( new MotionInterpolationApp() );
40    applications.push_back( new MotionTransitionApp() );
41    applications.push_back( new MotionDeformationEditApp() );
42    applications.push_back( new InverseKinematicsCCDApp() );
43
44    //
        GLUTフレームワークのメイン関数を呼び出し（実行するアプリケーションのリストを指定）
45    SimpleHumanGLUTMain( applications, argc, argv );
46 }
```

1.5 動作再生

ここまでで、サンプルプログラムの基礎となる、骨格・姿勢・動作のデータ構造定義や基本処理関数、GLUTフレームワークの説明を行った。各アプリケーションは、アプリケーションの基底クラス（GLUTBaseApp）の派生クラスとして定義される。本節では、その一つである動作再生アプリケーションについて、説明する。

動作再生アプリケーションは、MotionPlaybackApp クラスにより実現されている。プログラムの開始時には、動作再生アプリケーションが実行されるようになっている。他のアプリケーションの実行中は、マウス中ボタン（または CTRL + 右ボタン）でメニューを表示して、Motion Playback を選択することで、動作再生アプリケーションを実行できる。本アプリケーションの操作方法は、次の通りである。L キーで、再生する BVH 動作の選択・変更を行う。BVH 動作の再生中に、S キーで、動作再生の一時停止と再生を行う。動作再生の一時停止中に、N キーで次のフレーム、P キーで前のフレームに進む。W キーで、再生速度の変更を行う。

MotionPlaybackApp クラスの定義・実装を、ソースコード 14・15 に示す。本クラスについては、全ての処理がサンプルプログラムで実装されているため、各自で処理を追加する必要はない。

メンバ変数として、再生を行う動作データ（Motion * motion）や、動作再生のための変数である、キャラクタの現在姿勢（Posture * curr_posture）やアニメーション再生時間（animation_time）などの変数が定義されている。

メンバ関数として、各イベント処理の関数が定義されている。初期化処理（Initialize 関数）で、動作データの読み込みや、骨格・姿勢・動作の生成を行っている。開始処理（Start 関数）で、動作再生の初期化を行っている。アニメーション処理（Animation 関数）で、アニメーションの時間を進めて、現在時刻の姿勢を取得する処理を行っている。画面描画処理（Display 関数）で、現在時刻の姿勢の描画を行っている。キーボード入力処理（Keyboard 関数）で、キー入力に応じて、アニメーションの停止・再生・コマ送りの操作や、BVH 動作の読み込みを行っている。

その他のメンバ関数として、動作データの読み込みを行う LoadBVH 関数で、BVH ファイル名を引数として受け取り、BVH 形式の動作を読み込んで、骨格・姿勢・動作を生成し、メンバ変数に設定する。本関数は、アプリケーションの初期化処理とキーボード入力イベント処理の両方から呼ばれる。

ソースコード 14: 動作再生アプリケーションの定義（MotionPlaybackApp.h）

```
1 /**
```

```
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプログラム
3  ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7  /**
8  ***  動作再生アプリケーション
9  **/
10
11 #ifndef  _MOTION_PLAYBACK_APP_H_
12 #define  _MOTION_PLAYBACK_APP_H_
13
14
15 // ライブラリ・クラス定義の読み込み
16 #include "SimpleHuman.h"
17 #include "SimpleHumanGLUT.h"
18
19
20 //
21 //  動作再生アプリケーションクラス
22 //
23 class  MotionPlaybackApp : public GLUTBaseApp
24 {
25   protected:
26    // 動作再生の入力情報
27
28    // 動作データ
29    Motion *  motion;
30
31   protected:
32    // 動作再生のための変数
33
34    // 現在の姿勢
35    Posture *  curr_posture;
36
37    // アニメーション再生中かどうかを表すフラグ
38    bool  on_animation;
39
40    // アニメーションの再生時間
41    float  animation_time;
42
43    // アニメーションの再生速度
44    float  animation_speed;
45
46    // 現在の表示フレーム番号
47    int  frame_no;
48
49   public:
50    // コンストラクタ
51    MotionPlaybackApp();
52
53    // デストラクタ
54    virtual ~MotionPlaybackApp();
55
56   public:
57    // イベント処理
58
59    //  初期化
60    virtual void  Initialize();
61
62    //  開始・リセット
63    virtual void  Start();
64
```

```
65      //  画面描画
66      virtual void  Display();
67
68      // キーボードのキー押下
69      virtual void  Keyboard( unsigned char key, int mx, int my );
70
71      // アニメーション処理
72      virtual void  Animation( float delta );
73
74    public:
75      // 補助処理
76
77      // BVH動作ファイルの読み込み、動作・姿勢の初期化
78      void  LoadBVH( const char * file_name );
79
80      // ファイルダイアログを表示してBVH動作ファイルを選択・読み込み
81      void  OpenNewBVH();
82  };
83
84
85  #endif // _MOTION_PLAYBACK_APP_H_
```

ソースコード 15: 動作再生アプリケーションの実装（MotionPlaybackApp.cpp）

```
1   /**
2   ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプ
        ログラム
3   ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4   ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5   **/
6
7   /**
8   ***  動作再生アプリケーション
9   **/
10
11
12  // ライブラリ・クラス定義の読み込み
13  #include "SimpleHuman.h"
14  #include "BVH.h"
15  #include "MotionPlaybackApp.h"
16
17
18
19  //
20  //  コンストラクタ
21  //
22  MotionPlaybackApp::MotionPlaybackApp()
23  {
24      app_name = "Motion Playback";
25
26      motion = NULL;
27      curr_posture = NULL;
28      on_animation = true;
29      animation_time = 0.0f;
30      animation_speed = 1.0f;
31      frame_no = 0;
32  }
33
34
35  //
36  //  デストラクタ
37  //
38  MotionPlaybackApp::~MotionPlaybackApp()
39  {
```

```
40      // 骨格・動作・姿勢の削除
41      if ( motion && motion->body )
42          delete  motion->body;
43      if ( motion )
44          delete  motion;
45      if ( curr_posture )
46          delete  curr_posture;
47  }
48
49
50  //
51  //  初期化
52  //
53  void  MotionPlaybackApp::Initialize()
54  {
55      // 基底クラスの処理を実行
56      GLUTBaseApp::Initialize();
57
58      // サンプルBVH動作データを読み込み
59      LoadBVH( "sample_walking1.bvh" );
60  }
61
62
63  //
64  //  開始・リセット
65  //
66  void  MotionPlaybackApp::Start()
67  {
68      // 基底クラスの処理を実行
69      GLUTBaseApp::Start();
70
71      // アニメーション再生のための変数の初期化
72      on_animation = true;
73      animation_time = 0.0f;
74      frame_no = 0;
75
76      // アニメーション再生処理（姿勢の初期化）
77      Animation( 0.0f );
78  }
79
80
81  //
82  //  画面描画
83  //
84  void  MotionPlaybackApp::Display()
85  {
86      // 基底クラスの処理を実行
87      GLUTBaseApp::Display();
88
89      // キャラクタを描画
90      if ( curr_posture )
91      {
92          glColor3f( 1.0f, 1.0f, 1.0f );
93          DrawPosture( *curr_posture );
94          DrawPostureShadow( *curr_posture, shadow_dir, shadow_color );
95      }
96
97      // 現在のモード、時間・フレーム番号を表示
98      DrawTextInformation( 0, "Motion Playback" );
99      char  message[64];
100     if ( motion )
101         sprintf( message, "%.2f (%d)", animation_time, frame_no );
102     else
103         sprintf( message, "Press 'L' key to Load a BVH file" );
```

```
104     DrawTextInformation( 1, message );
105 }
106
107
108 //
109 //  キーボードのキー押下
110 //
111 void  MotionPlaybackApp::Keyboard( unsigned char key, int mx, int my )
112 {
113     GLUTBaseApp::Keyboard( key, mx, my );
114
115     // s キーでアニメーションの停止・再開
116     if ( key == 's' )
117         on_animation = !on_animation;
118
119     // w キーでアニメーションの再生速度を変更
120     if ( key == 'w' )
121         animation_speed = ( animation_speed == 1.0f ) ? 0.1f : 1.0f;
122
123     // n キーで次のフレーム
124     if ( ( key == 'n' ) && !on_animation && motion )
125     {
126         on_animation = true;
127         Animation( motion->interval );
128         on_animation = false;
129     }
130
131     // p キーで前のフレーム
132     if ( ( key == 'p' ) && !on_animation && motion && ( frame_no > 0 ) )
133     {
134         on_animation = true;
135         Animation( - motion->interval );
136         on_animation = false;
137     }
138
139     // l キーで再生動作の変更
140     if ( key == 'l' )
141     {
142         // ファイルダイアログを表示してBVHファイルを選択・読み込み
143         OpenNewBVH();
144     }
145 }
146
147
148 //
149 //  アニメーション処理
150 //
151 void  MotionPlaybackApp::Animation( float delta )
152 {
153     // アニメーション再生中でなければ終了
154     if ( !on_animation )
155         return;
156
157     // 動作データが読み込まれていなければ終了
158     if ( !motion )
159         return;
160
161     // 時間を進める
162     animation_time += delta * animation_speed;
163     if ( animation_time > motion->GetDuration() )
164         animation_time -= motion->GetDuration();
165
166     // 現在のフレーム番号を計算
167     frame_no = animation_time / motion->interval;
```

```
168 |
169 |     // 動作データから現在時刻の姿勢を取得
170 |     motion->GetPosture( animation_time, *curr_posture );
171 | }
172 |
173 |
174 |
175 | //
176 | //  補助処理
177 | //
178 |
179 |
180 | //
181 | //  BVH動作ファイルの読み込み、動作・姿勢の初期化
182 | //
183 | void  MotionPlaybackApp::LoadBVH( const char * file_name )
184 | {
185 |     // BVHファイルを読み込んで動作データ（＋骨格モデル）を生成
186 |     Motion *  new_motion = LoadAndCoustructBVHMotion( file_name );
187 |
188 |     // BVHファイルの読み込みに失敗したら終了
189 |     if ( !new_motion )
190 |         return;
191 |
192 |     // 現在使用している骨格・動作・姿勢を削除
193 |     if ( motion && motion->body )
194 |         delete  motion->body;
195 |     if ( motion )
196 |         delete  motion;
197 |     if ( curr_posture )
198 |         delete  curr_posture;
199 |
200 |     // 動作再生に使用する動作・姿勢を初期化
201 |     motion = new_motion;
202 |     curr_posture = new Posture( motion->body );
203 |
204 |     // 動作再生開始
205 |     Start();
206 | }
207 |
208 |
209 | //
210 | //  ファイルダイアログを表示してBVH動作ファイルを選択・読み込み
211 | //
212 | void  MotionPlaybackApp::OpenNewBVH()
213 | {
214 | #ifdef  WIN32
215 |     const int  file_name_len = 256;
216 |     char  file_name[file_name_len] = "";
217 |
218 |     // ファイルダイアログの設定
219 |     OPENFILENAME  open_file;
220 |     memset( &open_file, 0, sizeof( OPENFILENAME ) );
221 |     open_file.lStructSize = sizeof( OPENFILENAME );
222 |     open_file.hwndOwner = NULL;
223 |     open_file.lpstrFilter = "BVH Motion Data (*.bvh)\0*.bvh\0All (*.*)\0*.*\0";
224 |     open_file.nFilterIndex = 1;
225 |     open_file.lpstrFile = file_name;
226 |     open_file.nMaxFile = file_name_len;
227 |     open_file.lpstrTitle = "Select a BVH file";
228 |     open_file.lpstrDefExt = "bvh";
229 |     open_file.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;
230 |
231 |     // ファイルダイアログを表示
```

```
232     BOOL  ret = GetOpenFileName( &open_file );
233
234     // ファイルが指定されたら新しい動作を設定
235     if ( ret )
236     {
237         // BVH動作データの読み込み、骨格・姿勢の初期化
238         LoadBVH( file_name );
239
240         // 動作再生の開始
241         Start();
242     }
243 #endif // WIN32
244 }
```

2 レポート課題

レポート課題として、以下のキャラクタアニメーションの基本処理を作成する。

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
7. 逆運動学計算（CCD 法）

各アプリケーションについて、順番に説明する。

2.1 順運動学計算

順運動学計算アプリケーションを実現する、ForwardKinematicsApp クラスの処理を作成する。基本的な機能は動作再生アプリケーションと同様であるが、動作再生中の各姿勢に対して順運動学計算を行い、計算結果（体節の位置・向き、関節の位置）を描画する。

ForwardKinematicsApp クラスの定義・実装を、ソースコード 16・17 に示す。本クラスについては、一部の処理はサンプルプログラムでは空欄となっているため、各自で処理を追加する必要がある。

順運動学計算アプリケーション（ForwardKinematicsApp）クラスは、動作再生アプリケーション（MotionPlaybackApp）クラスの派生クラスとして定義されている。メンバ変数として、順運動学計算結果の結果を格納する、全体節の位置・向きを表す 4×4 行列の配列（Matrix4f 型の可変長配列）や、全関節の位置を表す 3 次元ベクトルの配列（Point3f 型の可変長配列）を持つ。基本となる動作再生処理は、基底クラスのメンバ変数・メンバ関数により実現し、アニメーション再生中の現在姿勢に対する順運動学計算や、順運動学計算結果の描画の処理のみを、ForwardKinematicsApp クラスで追加している。

ソースコード 16: 順運動学計算アプリケーションの定義（ForwardKinematicsApp.h）

```
1 /**
2 ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプログラム
3 ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4 ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5 **/
```

```
6
7  /**
8  ***  順運動学計算アプリケーション
9  **/
10
11 #ifndef  _FORWARD_KINEMATICS_APP_H_
12 #define  _FORWARD_KINEMATICS_APP_H_
13
14
15 // ライブラリ・クラス定義の読み込み
16 #include "SimpleHuman.h"
17 #include "SimpleHumanGLUT.h"
18 #include "MotionPlaybackApp.h"
19
20
21 //
22 //  順運動学計算アプリケーションクラス
23 //  (動作再生アプリケーションに順運動学計算を追加)
24 //
25 class  ForwardKinematicsApp : public MotionPlaybackApp
26 {
27   protected:
28    // 順運動学計算結果の変数
29
30    // 全体節の位置・向き
31    vector< Matrix4f >  segment_frames;
32
33    // 全関節の位置
34    vector< Point3f >  joint_positions;
35
36   public:
37    // コンストラクタ
38    ForwardKinematicsApp();
39
40   public:
41    // イベント処理
42
43    //  開始・リセット
44    virtual void  Start();
45
46    //  画面描画
47    virtual void  Display();
48
49    // アニメーション処理
50    virtual void  Animation( float delta );
51 };
52
53
54 // 補助処理(グローバル関数)のプロトタイプ宣言
55
56 // 順運動学計算(※レポート課題)
57 void  MyForwardKinematics( const Posture & posture, vector< Matrix4f > &
       seg_frame_array, vector< Point3f > & joi_pos_array );
58
59 // 順運動学計算のための反復計算(ルート体節から末端体節に向かって繰り返し再帰呼び出し)
       (※レポート課題)
60 void  MyForwardKinematicsIteration( const Segment *  segment, const Segment *
       prev_segment, const Posture & posture,
61    Matrix4f * seg_frame_array, Point3f * joi_pos_array = NULL );
62
63
64 #endif // _FORWARD_KINEMATICS_APP_H_
```

ソースコード 17: 順運動学計算アプリケーションの実装（ForwardKinematicsApp.cpp）

```
1  /**
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプログラム
3  ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7  /**
8  ***  順運動学計算アプリケーション
9  **/
10
11
12 // ライブラリ・クラス定義の読み込み
13 #include "SimpleHuman.h"
14 #include "ForwardKinematicsApp.h"
15
16
17
18 //
19 //  コンストラクタ
20 //
21 ForwardKinematicsApp::ForwardKinematicsApp()
22 {
23    app_name = "Forward Kinematics";
24 }
25
26
27 //
28 //  開始・リセット
29 //
30 void  ForwardKinematicsApp::Start()
31 {
32    MotionPlaybackApp::Start();
33
34    // 配列初期化
35    segment_frames.resize( motion->body->num_segments );
36    joint_positions.resize( motion->body->num_joints );
37
38    ForwardKinematics( *curr_posture, segment_frames, joint_positions );
39 }
40
41
42 //
43 //  画面描画
44 //
45 void  ForwardKinematicsApp::Display()
46 {
47    GLUTBaseApp::Display();
48
49    // キャラクタを描画
50    if ( curr_posture )
51    {
52       glColor3f( 1.0f, 1.0f, 1.0f );
53       DrawPosture( *curr_posture );
54       DrawPostureShadow( *curr_posture, shadow_dir, shadow_color );
55    }
56
57
58    // 関節・体節の位置・向きを描画
59
60    const float  axis_length = 0.2f;
61    float  line_width;
```

```
62      Matrix4f  frame;
63
64      // デプステストを無効にして、前面に上書きする
65      glDisable( GL_DEPTH_TEST );
66
67      // 関節点を描画（球を描画）
68      for ( int i=0; i<joint_positions.size(); i++ )
69      {
70          // 関節位置に球を描画
71          const Point3f &  pos = joint_positions[ i ];
72          glColor3f( 0.0f, 0.0f, 1.0f );
73          glPushMatrix();
74          glTranslatef( pos.x, pos.y, pos.z );
75          glutSolidSphere( 0.025f, 16, 16 );
76          glPopMatrix();
77      }
78
79      // 体節の座標系を描画（座標軸を描画）
80      glGetFloatv( GL_LINE_WIDTH, &line_width );
81      glLineWidth( 2.0f );
82      for ( int i=0; i<segment_frames.size(); i++ )
83      {
84
85          glPushMatrix();
86          frame.transpose( segment_frames[ i ] );
87          glMultMatrixf( & frame.m00 );
88          glBegin( GL_LINES );
89              glColor3f( 1.0f, 0.0f, 0.0f );
90              glVertex3f( 0.0f, 0.0f, 0.0f );
91              glVertex3f( axis_length, 0.0f, 0.0f );
92              glColor3f( 0.0f, 1.0f, 0.0f );
93              glVertex3f( 0.0f, 0.0f, 0.0f );
94              glVertex3f( 0.0f, axis_length, 0.0f );
95              glColor3f( 0.0f, 0.0f, 1.0f );
96              glVertex3f( 0.0f, 0.0f, 0.0f );
97              glVertex3f( 0.0f, 0.0f, axis_length );
98          glEnd();
99          glPopMatrix();
100     }
101     glLineWidth( line_width );
102
103     glEnable( GL_DEPTH_TEST );
104
105
106     // 現在のモード、時間・フレーム番号を表示
107     DrawTextInformation( 0, "Forward Kinematics" );
108     char  message[64];
109     if ( motion )
110         sprintf( message, "%.2f (%d)", animation_time, frame_no );
111     else
112         sprintf( message, "Press 'L' key to Load a BVH file" );
113     DrawTextInformation( 1, message );
114 }
115
116
117 //
118 //  アニメーション処理
119 //
120 void  ForwardKinematicsApp::Animation( float delta )
121 {
122     MotionPlaybackApp::Animation( delta );
123
124     // アニメーション再生中でなければ終了
125     if ( !on_animation )
```

```
126         return;
127
128     if ( !curr_posture )
129         return;
130
131     // 順運動学計算
132     MyForwardKinematics( *curr_posture, segment_frames, joint_positions );
133 }
134
135
136 //
137 //  順運動学計算
138 //
139 void  MyForwardKinematics( const Posture & posture, vector< Matrix4f > &
        seg_frame_array, vector< Point3f > & joi_pos_array )
140 {
141     // 配列初期化
142     seg_frame_array.resize( posture.body->num_segments );
143     joi_pos_array.resize( posture.body->num_joints );
144
145     // ルート体節の位置・向きを設定
146     seg_frame_array[ 0 ].set( posture.root_ori, posture.root_pos, 1.0f );
147
148     // Forward Kinematics 計算のための反復計算（ルート体節から末端体節に向かって繰り返し
            計算）
149     MyForwardKinematicsIteration( posture.body->segments[ 0 ], NULL, posture, &
            seg_frame_array.front(), &joi_pos_array.front() );
150 }
151
152
153 //
154 //  Forward Kinematics 計算のための反復計算（ルート体節から末端体節に向かって繰り返し再
        帰呼び出し）
155 //
156 void  MyForwardKinematicsIteration(
157     const Segment *  segment, const Segment * prev_segment, const Posture & posture,
158     Matrix4f * seg_frame_array, Point3f * joi_pos_array )
159 {
160     // 骨格情報
161     const Skeleton *  body = posture.body;
162
163     // 次の関節・体節
164     Joint *  next_joint;
165     Segment *  next_segment;
166
167     // 次の関節・体節の変換行列
168     Matrix4f  frame;
169
170     // 計算用のベクトル・行列
171     Vector3f  pos;
172     Matrix4f  mat;
173
174     // 現在の体節に接続している各関節に対して繰り返し
175     for ( int j = 0; j < segment->num_joints; j++ )
176     {
177         // 次の関節・次の体節を取得
178         next_joint = segment->joints[ j ];
179         if ( next_joint->segments[ 0 ] != segment )
180             next_segment = next_joint->segments[ 0 ];
181         else
182             next_segment = next_joint->segments[ 1 ];
183
184         // 前の体節側（ルート体節側）の関節はスキップ
185         if ( next_segment == prev_segment )
```

```
186             continue;
187
188         // 現在の体節の変換行列を取得
189         frame = seg_frame_array[ segment->index ];
190
191         // ※ レポート課題
192
193         // 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
194 //      ???;
195
196         // 次の関節の位置を設定
197         if ( joi_pos_array )
198             joi_pos_array[ next_joint->index ] = pos;
199
200         // 関節の回転行列をかける
201 //      ???;
202
203         // 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
204 //      ???;
205
206         // 次の体節の変換行列を設定
207         if ( seg_frame_array )
208             seg_frame_array[ next_segment->index ] = frame;
209
210         // 次の体節に対して繰り返し（再帰呼び出し）
211         MyForwardKinematicsIteration( next_segment, segment, posture, seg_frame_array,
                joi_pos_array );
212     }
213 }
```

2.1.1 順運動学計算の関数定義

順運動学計算は、入力された人体モデルの骨格+姿勢（各関節の回転+ルートの位置・向き）から、各部位の位置・向きを求めるものである。具体的には、人体モデルの骨格+姿勢（Posture 型の posture）から、全体節の位置・向き（Matrix4f 型の配列 seg_frame_array）と、全関節の位置（Point3f 型の配列 joi_pos_array）を計算する。関節の向きは一意に定義できないため、関節に関しては位置のみを計算する。位置・向きは全て、ワールド座標系で表されたものである。

順運動学計算は、ルート体節から末端の体節に向かって順番に計算を行っていく必要があり、また複数の接続関節を持つ体節では全ての接続関節に対して処理を行う必要があるため、再帰関数を使うと実現しやすい。再帰関数を用いる場合、順運動学計算のメイン関数では、ルート体節に対して再帰関数を呼び出す処理を行うことになる。

ソースコード 18: 順運動学計算の関数定義

```
void  MyForwardKinematics(
    const Posture & posture, Matrix4f * seg_frame_array, Point3f * joi_pos_array )
{
    MyForwardKinematicsIteration( next_segment, segment, posture, seg_frame_array,
        joi_pos_array );
}

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

順運動学計算の各体節に対する繰り返し処理を行う MyForwardKinematicsIteration 関数の引数は、表 1 に示す通りである。

表 1: MyForwardKinematicsIteration 関数の引数

種類	変数定義	表記	説明
入力	const Segment * segment		現在の体節
入力	const Segment * prev_segment		前の体節（ルート側の体節）
入力	const Posture & posture		入力姿勢
出力	Matrix4f seg_frame_array[体節番号]	M_i	全体節の位置・向き
出力	Point3f joi_pos_array[関節番号]		全関節の位置・向き

2.1.2 順運動学計算のための反復処理（レポート課題）

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

順運動学計算の各体節に対する繰り返し処理では、姿勢（pose）に加えて対象の体節（segment）と一つ前の体節（ルート側に隣接する体節）（prev_segment）を入力として受け取り、対象の体節に隣接する全ての（ルート側に隣接するものを除く）関節・体節の位置・向きを計算し、隣接する全ての体節に対して再帰的に繰り返す処理を行う。処理の流れは、以下の通り。

1. 現在の体節 $i-1$ に隣接する全ての関節（次の関節） j に対して、以下の処理を繰り返すことで、隣接する全ての次の体節 i の位置・向き M_i を求める。
   ただし、引数で指定された一つ前の体節の方向へは、繰り返しは行わない。

   1 現在の体節（の中心）の位置・向きを取得 M_{i-1}
   $M = M_{i-1}$

   2 次の関節の位置を変数 joi_pos_array[関節番号] に格納する

   3 現在の体節（の中心）から次の関節への平行移動 $T_{(i-1)\to i}$ をかける （ 空欄 A ～ 空欄 C ）
   $M = MT_{(i-1)\to i}$

   4 次の関節の回転 R_j をかける （ 空欄 D ～ 空欄 E ）
   $M = MR_j$

   5 次の関節から次の体節（の中心）への平行移動 $T_{i\to(i+1)}$ をかける （ 空欄 F ～ 空欄 H ）
   $M_i = MT_{i\to(i+1)}$

   6 次の体節の位置・向きを変数 seg_frame_array[体節番号] に格納する

   以上の一連の処理により、次の体節の位置・向き（体節のローカル座標系からワールド座標系への変換行列）M_i を、現在の体節の位置・向き（体節のローカル座標系からワールド座標系への変換行列） $M_{i-1}\times$ 次の体節のローカル座標系から現在の体節のローカル座標系への座標変換 $T_{(i-1)\to i}R_jT_{i\to(i+1)}$ により計算する。

   $$M_i = M_{i-1}T_{(i-1)\to i}R_jT_{i\to(i+1)} \tag{1}$$

   7 次の体節に対して再帰呼び出しを行う

各体節 i における隣接関節 j への平行移動（体節の中心を基準とするローカル座標系における各隣接関節の位置） $T_{i\to j}$ は、骨格モデル（Segment 構造体の joint_positions）から取得できる。また、関節 j の回転 R_j は、現在姿勢（Posture 構造体の joint_rottions）から取得できる。変換行列に対して平行移動や回転をかけるときには、平行移動や回転を４×４行列に変換してから、４×４行列同士の積を計算する。

ソースコード 19: 順運動学計算の繰り返し計算

```
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
        空欄 A
      空欄 B
      空欄 C

      // 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
        空欄 D
      空欄 E

      // 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
        空欄 F
      空欄 G
      空欄 H

      // 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 );
   }
}
```

2.2 姿勢補間

姿勢補間アプリケーションを実現する、PostureInterpolationApp クラスの処理を作成する。姿勢補間アプリケーションでは、2つのサンプル姿勢を、指定された重みで補間する。左ボタンを左右にドラッグすることで、姿

勢補間の重みを操作できる。2つのサンプル姿勢と、補間結果の姿勢を画面に表示する。姿勢補間の重みに応じてキャラクタの色を変化させる。D キーで、腰の水平位置の固定の有無を切替える。

PostureInterpolationApp クラスの定義・実装を、ソースコード 20・21 に示す。本クラスについては、一部の処理はサンプルプログラムでは空欄となっているため、各自で処理を追加する必要がある。

ソースコード 20: 姿勢補間アプリケーションの定義（PostureInterpolationApp.h）

```
1  /**
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプログラム
3  ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7  /**
8  ***  姿勢補間アプリケーション
9  **/
10
11 #ifndef  _POSTURE_INTERPOLATION_APP_H_
12 #define  _POSTURE_INTERPOLATION_APP_H_
13
14
15 // ライブラリ・クラス定義の読み込み
16 #include "SimpleHuman.h"
17 #include "SimpleHumanGLUT.h"
18
19
20 //
21 //  姿勢補間アプリケーションクラス
22 //
23 class  PostureInterpolationApp : public GLUTBaseApp
24 {
25   protected:
26     // キャラクタ情報
27
28     // キャラクタの骨格
29     Skeleton *  body;
30
31     // キャラクタの姿勢
32     Posture *  curr_posture;
33
34   protected:
35     // 姿勢補間のための変数
36
37     // サンプル姿勢
38     Posture *  posture0;
39     Posture *  posture1;
40
41     // サンプル姿勢の描画色
42     Color3f  posture0_color;
43     Color3f  posture1_color;
44
45     // 姿勢補間の重み
46     float  weight;
47
48     // 現在姿勢の描画色
49     Color3f  figure_color;
50
51   protected:
52     // 描画設定
53
54     // 補間姿勢の腰の位置を固定して描画
55     bool  draw_fixed_position;
```

```
56
57
58    public:
59     // コンストラクタ
60     PostureInterpolationApp();
61
62     // デストラクタ
63     virtual ~PostureInterpolationApp();
64
65    public:
66     // イベント処理
67
68     //  初期化
69     virtual void  Initialize();
70
71     //  開始・リセット
72     virtual void  Start();
73
74     //  画面描画
75     virtual void  Display();
76
77     // マウスドラッグ
78     virtual void  MouseDrag( int mx, int my );
79
80     // キーボードのキー押下
81     virtual void  Keyboard( unsigned char key, int mx, int my );
82
83    public:
84     // 補助処理
85
86     // 姿勢更新
87     void  UpdatePosture();
88 };
89
90
91 // 補助処理（グローバル関数）のプロトタイプ宣言
92
93 // 姿勢補間（2つの姿勢を補間）（※レポート課題）
94 void  MyPostureInterpolation( const Posture & p0, const Posture & p1, float ratio,
       Posture & p );
95
96
97 #endif // _POSTURE_INTERPOLATION_APP_H_
```

ソースコード 21: 姿勢補間アプリケーションの実装（PostureInterpolationApp.cpp）

```
1  /**
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプ
       ログラム
3  ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7  /**
8  ***  姿勢補間アプリケーション
9  **/
10
11
12 // ライブラリ・クラス定義の読み込み
13 #include "SimpleHuman.h"
14 #include "BVH.h"
15 #include "PostureInterpolationApp.h"
16
17 // 標準算術関数・定数の定義
```

```
18 #define _USE_MATH_DEFINES
19 #include <math.h>
20
21
22
23 //
24 //  コンストラクタ
25 //
26 PostureInterpolationApp::PostureInterpolationApp()
27 {
28     app_name = "Posture Interpolation";
29     body = NULL;
30     curr_posture = NULL;
31
32     posture0 = NULL;
33     posture1 = NULL;
34     weight = 0.0f;
35     figure_color.set( 1.0f, 1.0f, 1.0f );
36
37     draw_fixed_position = true;
38 }
39
40
41 //
42 //  デストラクタ
43 //
44 PostureInterpolationApp::~PostureInterpolationApp()
45 {
46 }
47
48
49 //
50 //  初期化
51 //
52 void  PostureInterpolationApp::Initialize()
53 {
54     GLUTBaseApp::Initialize();
55
56     // アプリケーションのテストに使用するサンプル姿勢（動作データ・時刻・描画色）
57     const char *  sample_motion = "sample_walking1.bvh";
58     const float  sample_keytimes[ 2 ] = { 3.00f, 3.74f };
59     const Color3f  sample_colors[] = { Color3f( 0.5f, 1.0f, 0.5f ), Color3f( 0.5f, 0.5f,
           1.0f ) };
60     const float  sample_orientation[ 2 ] = { 180.0f, 180.0f };
61
62     // 動作データの読み込み、骨格・姿勢の初期化
63     BVH  * bvh = new BVH( sample_motion );
64     if ( bvh->IsLoadSuccess() )
65     {
66         // BVH動作から骨格モデルを生成
67         Skeleton *  new_body = CoustructBVHSkeleton( bvh );
68
69         // 姿勢の初期化
70         if ( new_body )
71         {
72             body = new_body;
73             curr_posture = new Posture();
74             InitPosture( *curr_posture, body );
75         }
76     }
77
78     // サンプル姿勢の初期化
79     if ( body && curr_posture && bvh && bvh->IsLoadSuccess() )
80     {
```

```
81          // サンプル姿勢を動作データから取得
82          posture0 = new Posture( body );
83          posture1 = new Posture( body );
84          GetBVHPosture( bvh, sample_keytimes[ 0 ] / bvh->GetInterval(), *posture0 );
85          GetBVHPosture( bvh, sample_keytimes[ 1 ] / bvh->GetInterval(), *posture1 );
86          posture0_color = sample_colors[ 0 ];
87          posture1_color = sample_colors[ 1 ];
88
89          // 水平方向の回転が指定されていれば回転を適用
90          Matrix3f  rot;
91          if ( sample_orientation[ 0 ] != 0.0f )
92          {
93              rot.rotY( sample_orientation[ 0 ] * M_PI / 180.0f );
94              posture0->root_ori.mul( rot, posture0->root_ori );
95          }
96          if ( sample_orientation[ 1 ] != 0.0f )
97          {
98              rot.rotY( sample_orientation[ 1 ] * M_PI / 180.0f );
99              posture1->root_ori.mul( rot, posture1->root_ori );
100         }
101
102         // サンプル姿勢を描画する位置を設定（現在姿勢の左右に配置）
103         posture0->root_pos.x = -1.0f;
104         posture0->root_pos.z = 0.0f;
105         posture1->root_pos.x = 1.0f;
106         posture1->root_pos.z = 0.0f;
107
108         // 現在姿勢を初期化
109         *curr_posture = *posture0;
110     }
111 }
112
113
114 //
115 //  開始・リセット
116 //
117 void  PostureInterpolationApp::Start()
118 {
119     GLUTBaseApp::Start();
120
121     // 重みの初期化
122     weight = 0.5f;
123
124     // 姿勢更新
125     UpdatePosture();
126 }
127
128
129 //
130 //  画面描画
131 //
132 void  PostureInterpolationApp::Display()
133 {
134     GLUTBaseApp::Display();
135
136     // キャラクタを描画
137     if ( curr_posture )
138     {
139         glColor3f( figure_color.x, figure_color.y, figure_color.z );
140         DrawPosture( *curr_posture );
141         DrawPostureShadow( *curr_posture, shadow_dir, shadow_color );
142     }
143
144     // サンプル姿勢を描画
```

```
145     if ( posture0 )
146     {
147         glColor3f( posture0_color.x, posture0_color.y, posture0_color.z );
148         DrawPosture( *posture0 );
149         DrawPostureShadow( *posture0, shadow_dir, shadow_color );
150     }
151     if ( posture1 )
152     {
153         glColor3f( posture1_color.x, posture1_color.y, posture1_color.z );
154         DrawPosture( *posture1 );
155         DrawPostureShadow( *posture1, shadow_dir, shadow_color );
156     }
157
158     // 現在のモード、補間重みを表示
159     DrawTextInformation( 0, "Posture Interpolation" );
160     char  message[ 64 ];
161     sprintf( message, "Weight: %.2f", weight );
162     DrawTextInformation( 1, message );
163 }
164
165
166 //
167 //  マウスドラッグ
168 //
169 void  PostureInterpolationApp::MouseDrag( int mx, int my )
170 {
171     // 左ボタンの左右ドラッグに応じて重みを計算
172     if ( drag_mouse_l )
173     {
174         // 重み計算
175         weight += (float) ( mx - last_mouse_x ) * 2.0f / win_width;
176         if ( weight < 0.0f )
177             weight = 0.0f;
178         if ( weight > 1.0f )
179             weight = 1.0f;
180
181         // 姿勢更新
182         UpdatePosture();
183     }
184
185     GLUTBaseApp::MouseDrag( mx, my );
186 }
187
188
189 //
190 //  キーボードのキー押下
191 //
192 void  PostureInterpolationApp::Keyboard( unsigned char key, int mx, int my )
193 {
194     // 基底クラスの処理を実行
195     GLUTBaseApp::Keyboard( key, mx, my );
196
197     // d キーで描画設定を変更
198     if ( key == 'd' )
199     {
200         draw_fixed_position = !draw_fixed_position;
201         UpdatePosture();
202     }
203 }
204
205
206 //
207 //  姿勢更新
208 //
```

```
209 void  PostureInterpolationApp::UpdatePosture()
210 {
211     if ( !curr_posture || !posture0 || !posture1 )
212         return;
213 
214     // 姿勢補間
215     MyPostureInterpolation( *posture0, *posture1, weight, *curr_posture );
216 
217     // 補間姿勢の腰の位置を固定して描画する場合は、腰の水平位置は原点に固定
218     if ( draw_fixed_position )
219     {
220         curr_posture->root_pos.x = 0.0f;
221         curr_posture->root_pos.z = 0.0f;
222     }
223 
224     // 重みに応じて描画色を設定
225     figure_color.scaleAdd( weight, posture1_color - posture0_color, posture0_color );
226 }
227 
228 
229 //
230 //  以下、補助処理
231 //
232 
233 
234 //
235 //  姿勢補間（２つの姿勢を補間）
236 //
237 void  MyPostureInterpolation( const Posture & p0, const Posture & p1, float ratio,
        Posture & p )
238 {
239     // ２つの姿勢の骨格モデルが異なる場合は終了
240     if ( ( p0.body != p1.body ) || ( p0.body != p.body ) )
241         return;
242 
243     // 骨格モデルを取得
244     const Skeleton *  body = p0.body;
245 
246 
247     // ２つの姿勢の各関節の回転を補間
248     for ( int i = 0; i < body->num_joints; i++ )
249     {
250         // ※ レポート課題
251     }
252 
253     // ２つの姿勢のルートの向きを補間
254     // ※ レポート課題
255 
256     // ２つの姿勢のルートの位置を補間
257     // ※ レポート課題
258 
259 }
```

2.2.1 ２つの姿勢の補間（レポート課題）

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

入力された２つの姿勢（Posture 型の p0, p1）を、指定された重み（ratio）で補間して新しい姿勢（p）を計算する関数を作成する。

処理の流れは、以下の通り。

1. 2つの姿勢の各関節の回転を補間（関節ごとに繰り返し）（ 空欄 A ～ 空欄 B ）
   各姿勢から取得した関節の回転（回転行列による表現）を四元数に変換し、指定された重み（ブレンド比率）にもとづいて球面線形補間（SLERP）を適用することで、回転の補間を計算する。計算結果を姿勢に設定する際に、四元数から回転行列に変換する。

2. 2つの姿勢のルートの向きを補間（ 空欄 C ）
   各関節の回転の補間と同様の計算方法で実現する。

3. 2つの姿勢のルートの位置を補間（ 空欄 D ～ 空欄 E ）
   指定された重み（ブレンド比率）にもとづいて、2つの位置を線形に補間する。

ソースコード 22: 2つの姿勢の補間

```
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++ )
    {
        空欄 A
        if ( q0.x * q1.x + q0.y * q1.y + q0.z * q1.z + q0.w * q1.w < 0 )
            q1.negate( q1 );
        空欄 B
        p.joint_rotations[ i ].set( q );
    }

    // 2つの姿勢のルートの向きを補間
    空欄 C
    if ( q0.x * q1.x + q0.y * q1.y + q0.z * q1.z + q0.w * q1.w < 0 )
        q1.negate( q1 );
    空欄 B
    p.root_ori.set( q );

    // 2つの姿勢のルートの位置を補間
    空欄 D
    空欄 E
    p.root_pos.set( v );
}
```

2.3 キーフレーム動作再生

キーフレーム動作再生アプリケーションを実現する、KeyframeMotionPlaybackApp クラスの処理を作成する。キーフレーム動作再生アプリケーションでは、BVH 動作をキーフレーム動作に変換して、再生する。BVH 動作から、プログラム中で指定された複数の時刻の姿勢を取得して、キーフレーム動作を生成する。元の BVH 動作や取得したキーフレームを並べて、キーフレーム動作を描画する。

KeyframeMotionPlaybackApp クラスの定義・実装を、ソースコード 23・24 に示す。本クラスについては、一部の処理はサンプルプログラムでは空欄となっているため、各自で処理を追加する必要がある。

キーフレーム動作再生アプリケーション（KeyframeMotionPlaybackApp）クラスは、動作再生アプリケーション（MotionPlaybackApp）クラスの派生クラスとして定義されている。

キーフレーム動作からの姿勢取得には、2.2 節で説明した、2 つの姿勢の補間を行う MyPostureInterpolation 関数を使用する。そのため、本アプリケーションを作成する前に、MyPostureInterpolation 関数を作成する必要がある。

ソースコード 23: キーフレーム動作再生アプリケーションの定義（KeyframeMotionPlaybackApp.h）

```
1  /**
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプログラム
3  ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7  /**
8  ***  キーフレーム動作再生アプリケーション
9  **/
10
11 #ifndef  _KEYFRAME_MOTION_PLAYBACK_APP_H_
12 #define  _KEYFRAME_MOTION_PLAYBACK_APP_H_
13
14
15 // ライブラリ・クラス定義の読み込み
16 #include "SimpleHuman.h"
17 #include "SimpleHumanGLUT.h"
18 #include "MotionPlaybackApp.h"
19
20
21 //
22 //  キーフレーム動作再生アプリケーションクラス
23 //
24 class  KeyframeMotionPlaybackApp : public MotionPlaybackApp
25 {
26   protected:
27    // キーフレーム動作再生のための変数
28
29    // キーフレーム動作データ
30    KeyframeMotion *  keyframe_motion;
31
32    // キーフレーム動作からの取得姿勢
33    Posture *  keyframe_posture;
34
35    // キーフレーム動作と元の動作を同期して再生するための時間のオフセット
36    float  motion_time_offset;
37
38   protected:
39    // 描画設定
40
41    // 元の動作を並べて再生表示
42    bool  draw_original_motion;
43
44    // キーフレームの姿勢を並べて表示
45    bool  draw_key_poses;
46
47
48   public:
49    // コンストラクタ
50    KeyframeMotionPlaybackApp();
51
52    // デストラクタ
53    virtual ~KeyframeMotionPlaybackApp();
54
```

```
55    public:
56      // イベント処理
57
58      //  初期化
59      virtual void  Initialize();
60
61      //  画面描画
62      virtual void  Display();
63
64      // キーボードのキー押下
65      virtual void  Keyboard( unsigned char key, int mx, int my );
66
67      // アニメーション処理
68      virtual void  Animation( float delta );
69  };
70
71
72  // 補助処理（グローバル関数）のプロトタイプ宣言
73
74  // キーフレーム動作から姿勢を取得
75  void  GetKeyframeMotionPosture( const KeyframeMotion & motion, float time, Posture & p
        );
76
77
78  #endif // _KEYFRAME_MOTION_PLAYBACK_APP_H_
```

ソースコード 24: キーフレーム動作再生アプリケーションの実装（KeyframeMotionPlaybackApp.cpp）

```
1   /**
2   ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプ
        ログラム
3   ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4   ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5   **/
6
7   /**
8   ***  キーフレーム動作再生アプリケーション
9   **/
10
11
12  // ライブラリ・クラス定義の読み込み
13  #include "SimpleHuman.h"
14  #include "BVH.h"
15  #include "KeyframeMotionPlaybackApp.h"
16
17  // プロトタイプ宣言
18
19  // 姿勢補間（２つの姿勢を補間）（※レポート課題）
20  void  MyPostureInterpolation( const Posture & p0, const Posture & p1, float ratio,
        Posture & p );
21
22
23
24  //
25  //  コンストラクタ
26  //
27  KeyframeMotionPlaybackApp::KeyframeMotionPlaybackApp()
28  {
29      app_name = "Keyframe Motion Playback";
30
31      keyframe_motion = NULL;
32      keyframe_posture = NULL;
33      motion_time_offset = 0.0f;
34
```

```
35      draw_original_motion = true;
36      draw_key_poses = true;
37  }
38
39
40  //
41  //  デストラクタ
42  //
43  KeyframeMotionPlaybackApp::~KeyframeMotionPlaybackApp()
44  {
45      if ( keyframe_motion )
46          delete  keyframe_motion;
47      if ( keyframe_posture )
48          delete  keyframe_posture;
49  }
50
51
52  //
53  //  初期化
54  //
55  void  KeyframeMotionPlaybackApp::Initialize()
56  {
57      GLUTBaseApp::Initialize();
58
59      const char *  sample_motions = "sample_walking1.bvh";
60      const int  num_keytimes = 3;
61      const float  sample_keytimes[ num_keytimes ] = { 2.35f, 3.00f, 3.74f };
62
63      // BVH動作データを読み込み
64      LoadBVH( sample_motions );
65      if ( !motion )
66          return;
67
68      // キーフレームの時刻を設定
69      vector< float >  key_times( num_keytimes );
70      for ( int i = 0; i < num_keytimes; i++ )
71          key_times[ i ] = sample_keytimes[ i ] - sample_keytimes[ 0 ];
72
73      // BVH動作から取得した姿勢をキーフレームの姿勢として設定
74      vector< Posture >  key_poses( num_keytimes );
75      for ( int i = 0; i < num_keytimes; i++ )
76          motion->GetPosture( sample_keytimes[ i ], key_poses[ i ] );
77
78      // キーフレーム動作を初期化
79      keyframe_motion = new KeyframeMotion();
80      keyframe_motion->Init( motion->body, num_keytimes, &key_times.front(), &key_poses.
            front() );
81
82      // キーフレーム動作から取得する姿勢の初期化
83      keyframe_posture = new Posture( motion->body );
84      *keyframe_posture = key_poses[ 0 ];
85
86      // キーフレーム動作と元の動作を同期して再生するための時間のオフセットを設定
87      motion_time_offset = sample_keytimes[ 0 ];
88
89      // サンプルBVH動作に合わせて視点調節
90      camera_yaw += 180.0f;
91  }
92
93
94  //
95  //  画面描画
96  //
97  void  KeyframeMotionPlaybackApp::Display()
```

```
98  {
99      GLUTBaseApp::Display();
100
101     // キーフレーム動作から取得した姿勢を描画
102     if ( keyframe_posture )
103     {
104         glColor3f( 1.0f, 1.0f, 1.0f );
105         DrawPosture( *keyframe_posture );
106         DrawPostureShadow( *keyframe_posture, shadow_dir, shadow_color );
107     }
108
109     // 元のBVH動作から取得した姿勢を描画
110     if ( draw_original_motion && curr_posture )
111     {
112         glPushMatrix();
113         glTranslatef( 1.0f, 0.0f, 0.0f );
114
115         glColor3f( 0.0f, 0.0f, 1.0f );
116         DrawPosture( *curr_posture );
117         DrawPostureShadow( *curr_posture, shadow_dir, shadow_color );
118         glPopMatrix();
119     }
120
121     // キーフレーム動作のキー姿勢を描画
122     if ( draw_key_poses && keyframe_posture )
123     {
124         glPushMatrix();
125         glTranslatef( -1.0f, 0.0f, 0.0f );
126
127         for ( int i=0; i<keyframe_motion->num_keyframes; i++ )
128         {
129             glPushMatrix();
130             glTranslatef( 0.0f, 0.0f, 1.0f * ( i - ( keyframe_motion->num_keyframes - 1 )
                    * 0.5f ) );
131
132             glColor3f( 0.8f, 0.8f, 0.8f );
133             DrawPosture( keyframe_motion->key_poses[ i ] );
134             DrawPostureShadow( keyframe_motion->key_poses[ i ], shadow_dir, shadow_color
                    );
135             glPopMatrix();
136         }
137         glPopMatrix();
138     }
139
140     // 現在のモード、時間・フレーム番号を表示
141     DrawTextInformation( 0, "Keyframe Motion Playback" );
142     char  message[64];
143     if ( motion )
144         sprintf( message, "%.2f (%d)", animation_time, frame_no );
145     else
146         sprintf( message, "Press 'L' key to Load a BVH file" );
147     DrawTextInformation( 1, message );
148 }
149
150
151 //
152 //  キーボードのキー押下
153 //
154 void  KeyframeMotionPlaybackApp::Keyboard( unsigned char key, int mx, int my )
155 {
156     // 基底クラスの処理を実行
157     MotionPlaybackApp::Keyboard( key, mx, my );
158
159     // d キーで描画設定を変更
```

```
160     if ( key == 'd' )
161     {
162         if ( draw_original_motion && draw_key_poses )
163             draw_key_poses = false;
164         else if ( draw_original_motion && !draw_key_poses )
165         {
166             draw_original_motion = false;
167             draw_key_poses = true;
168         }
169         else if ( !draw_original_motion && draw_key_poses )
170             draw_key_poses = false;
171         else
172         {
173             draw_original_motion = true;
174             draw_key_poses = true;
175         }
176     }
177 }
178
179
180 //
181 //  アニメーション処理
182 //
183 void  KeyframeMotionPlaybackApp::Animation( float delta )
184 {
185     // アニメーション再生中でなければ終了
186     if ( !on_animation )
187         return;
188
189     // 動作データが読み込まれていなければ終了
190     if ( !keyframe_motion )
191         return;
192
193     // 時間を進める
194     animation_time += delta * animation_speed;;
195     if ( animation_time > keyframe_motion->GetDuration() )
196         animation_time -= keyframe_motion->GetDuration();
197     frame_no = ( animation_time + motion_time_offset ) / motion->interval;
198
199     // 動作データから現在時刻の姿勢を取得
200     motion->GetPosture( animation_time + motion_time_offset, *curr_posture );
201
202     // キーフレーム動作データから現在時刻の姿勢を取得
203     GetKeyframeMotionPosture( *keyframe_motion, animation_time, *keyframe_posture );
204 }
205
206
207 //
208 //  キーフレーム動作から姿勢を取得
209 //
210 void  GetKeyframeMotionPosture( const KeyframeMotion & motion, float time, Posture & p
    )
211 {
212     // ※レポート課題
213
214     // 指定時刻に対応する区間の番号を取得
215     int  no = -1;
216 // for ( int i = 0; i < ???; i++ )
217     {
218 //      if ( ??? )
219         {
220 //          no = i;
221 //          break;
222         }
```

```
223     }
224
225     // 対応する区間が存在しない場合は終了
226     if ( no == -1 )
227         return;
228
229     // 補間の割合を計算
230     float  s = 0.0f;
231 // s = ??? / ???;
232
233     // 前後のキー姿勢を補間
234     MyPostureInterpolation( motion.key_poses[ no ], motion.key_poses[ no+1 ], s, p );
235 }
```

2.3.1 キーフレーム動作からの姿勢取得（レポート課題）

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

キーフレーム動作（KeyframeMotion 構造体のオブジェクト）から、指定された時刻の姿勢を取得する。指定された時間に対応する区間番号とブレンド比率を計算し、前後のキー姿勢をブレンドすることで、姿勢を計算する。
処理の流れは、以下の通り。

1. 現在の時刻に対応する、区間番号を取得 （ 空欄 A ～ 空欄 B ）
   指定された時間に対応する区間は、全キー時刻の情報（KeyframeMotion 構造体の 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 番目の区間を探索することで取得する。

2. 現在の時刻に対応する、ブレンド比率（0.0～1.0）の計算 （ 空欄 C ～ 空欄 D ）
   ブレンド比率は、区間の開始時に 0.0 になり、区間の終了時に 1.0 になるように計算する。前のキー時刻からの経過時刻を、前後のキー時刻間の長さで割ることで、ブレンド比率を計算できる。

3. 前後のキー姿勢を補間
   現在の区間の前後のキー姿勢を取得（KeyframeMotion 構造体の key_poses 配列より取得）し、上で計算したブレンド比率で補間することで、出力姿勢を計算する。

ソースコード 25: 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の番号を取得
    int  no = -1;
    for ( int i = 0; i <  空欄 A  ; i++ )
    {
        if (  空欄 B  )
        {
            no = i;
            break;
        }
    }

    // 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 = (  空欄 C  ) / (  空欄 D  );
```

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

2.4 動作補間

動作補間アプリケーションを実現する、MotionInterpolationApp クラスの処理を作成する。動作補間アプリケーションでは、2つのサンプル動作を指定された重みで補間しながら再生する。左ボタンを左右にドラッグすることで、動作補間の重みを操作できる。2つのサンプル姿勢と、補間結果の姿勢を画面に表示する。姿勢補間の重みに応じてキャラクタの色を変化させる。

MotionInterpolationApp クラスの定義・実装を、ソースコード 26・27 に示す。本クラスについては、一部の処理はサンプルプログラムでは空欄となっているため、各自で処理を追加する必要がある。

2つのサンプル動作から取得した姿勢の補間には、2.2 節で説明した、2 つの姿勢の補間を行う MyPostureInterpolation 関数を使用する。そのため、本アプリケーションを作成する前に、MyPostureInterpolation 関数を作成する必要がある。

また、動作補間を行いながら動作を繰り返し再生するために、2.6.1 節で説明する、動作接続のための変換行列の計算を行う ComputeConnectionTransformation 関数を使用する。ComputeConnectionTransformation 関数が未作成でも、本アプリケーションを作成することはできるが、1回分の動作再生が終わる度に位置・向きがリセットされる。

ソースコード 26: 動作補間アプリケーションの定義（MotionInterpolationApp.h）

```
1  /**
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプログラム
3  ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7  /**
8  ***  動作補間アプリケーション
9  **/
10
11 #ifndef  _MOTION_INTERPOLATION_APP_H_
12 #define  _MOTION_INTERPOLATION_APP_H_
13
14
15 // ライブラリ・クラス定義の読み込み
16 #include "SimpleHuman.h"
17 #include "SimpleHumanGLUT.h"
18
19 // プロトタイプ宣言
20 struct  MotionInfo;
21
22
23 //
24 //    動作補間アプリケーションクラス
25 //
26 class  MotionInterpolationApp : public GLUTBaseApp
27 {
28   protected:
29    // 動作補間の入力情報
30
31    // 動作データ情報（動作補間に用いる2つの動作）
32    MotionInfo * motions[ 2 ];
33
34    // 動作補間の重み
```

```
35      float   weight;
36
37    protected:
38      // 動作再生のための変数
39
40      // キャラクタの姿勢
41      Posture *   curr_posture;
42
43      // アニメーション中かどうかを表すフラグ
44      bool   on_animation;
45
46      // アニメーションの再生時間
47      float   animation_time;
48
49      // アニメーションの再生速度
50      float   animation_speed;
51
52    protected:
53      // 動作補間のための変数
54
55      // サンプル動作からの取得姿勢
56      Posture *   motion_posture[ 2 ];
57
58      // 現在姿勢の描画色
59      Color3f   figure_color;
60
61    protected:
62      // 動作接続のための変数
63
64      // 繰り返し動作の再生開始時刻
65      float   cycle_start_time;
66
67      // サンプル動作の動作接続（前の動作の終了時の位置・向きに合わせる）ための変換行列
68      Matrix4f   motion_trans_mat[ 2 ];
69
70
71    public:
72      // コンストラクタ
73      MotionInterpolationApp();
74
75      // デストラクタ
76      virtual ~MotionInterpolationApp();
77
78    public:
79      // イベント処理
80
81      //  初期化
82      virtual void  Initialize();
83
84      //  開始・リセット
85      virtual void  Start();
86
87      //  画面描画
88      virtual void  Display();
89
90      // マウスドラッグ
91      virtual void  MouseDrag( int mx, int my );
92
93      // キーボードのキー押下
94      virtual void  Keyboard( unsigned char key, int mx, int my );
95
96      // アニメーション処理
97      virtual void  Animation( float delta );
98
```

```
99    public:
100    // 補助処理
101
102    // 動作再生処理（動作補間＋動作接続）
103    void  AnimationWithInterpolation( float delta );
104 };
105
106
107 // 補助処理（グローバル関数）のプロトタイプ宣言
108
109 // 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
110 //（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）
111 void  ComputeConnectionTransformation( const Matrix4f & prev_frame, const Matrix4f &
        next_frame, Matrix4f & trans_mat );
112
113 // 姿勢の位置・向きに変換行列を適用
114 void  TransformPosture( const Matrix4f & trans, Posture & posture );
115
116
117 #endif // _MOTION_INTERPOLATION_APP_H_
```

ソースコード 27: 動作補間アプリケーションの実装（MotionInterpolationApp.cpp）

```
1  /**
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプ
       ログラム
3  ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7  /**
8  ***  動作補間アプリケーション
9  **/
10
11
12 // ライブラリ・クラス定義の読み込み
13 #include "SimpleHuman.h"
14 #include "MotionInterpolationApp.h"
15 #include "MotionTransition.h"
16 #include "MotionTransitionApp.h"
17 #include "PostureInterpolationApp.h"
18
19 // 標準算術関数・定数の定義
20 #define  _USE_MATH_DEFINES
21 #include <math.h>
22
23
24
25 //
26 //  コンストラクタ
27 //
28 MotionInterpolationApp::MotionInterpolationApp()
29 {
30    app_name = "Motion Interpolation";
31
32    motions[ 0 ] = NULL;
33    motions[ 1 ] = NULL;
34    weight = 0.0f;
35
36    curr_posture = NULL;
37    on_animation = true;
38    animation_time = 0.0f;
39    animation_speed = 1.0f;
```

```
40
41      motion_posture[ 0 ] = NULL;
42      motion_posture[ 1 ] = NULL;
43      figure_color.set( 1.0f, 1.0f, 1.0f );
44
45      cycle_start_time = 0.0f;
46      motion_trans_mat[ 0 ].rotY( M_PI );
47      motion_trans_mat[ 1 ].rotY( M_PI );
48  }
49
50
51  //
52  //  デストラクタ
53  //
54  MotionInterpolationApp::~MotionInterpolationApp()
55  {
56      for ( int i=0; i<2; i++ )
57      {
58          if ( motions[ i ] )
59          {
60              if ( motions[ i ]->motion )
61                  delete  motions[ i ]->motion;
62              delete  motions[ i ];
63          }
64          if ( motion_posture[ i ])
65              delete  motion_posture[ i ];
66      }
67
68      if ( curr_posture && curr_posture->body )
69          delete  curr_posture->body;
70      if ( curr_posture )
71          delete  curr_posture;
72  }
73
74
75  //
76  //  初期化
77  //
78  void  MotionInterpolationApp::Initialize()
79  {
80      GLUTBaseApp::Initialize();
81
82      const Skeleton *  body = NULL;
83
84      // サンプル動作セットの読み込み
85      if ( !motions[ 0 ] )
86      {
87          vector< MotionInfo * >  motion_list;
88          body = LoadSampleMotions( motion_list );
89          motions[ 0 ] = motion_list[ 0 ];
90          motions[ 1 ] = motion_list[ 1 ];
91      }
92
93      // 姿勢の初期化
94      if ( body )
95      {
96          if ( curr_posture )
97              delete  curr_posture;
98          if ( motion_posture[ 0 ] )
99              delete  motion_posture[ 0 ];
100         if ( motion_posture[ 1 ] )
101             delete  motion_posture[ 1 ];
102
103         curr_posture = new Posture( body );
```

```
104         InitPosture( *curr_posture, body );
105         motion_posture[ 0 ] = new Posture( body );
106         motion_posture[ 1 ] = new Posture( body );
107     }
108 }
109 
110 
111 //
112 //  開始・リセット
113 //
114 void  MotionInterpolationApp::Start()
115 {
116     GLUTBaseApp::Start();
117 
118     weight = 0.0f;
119 
120     on_animation = true;
121     animation_time = 0.0f;
122 
123     cycle_start_time = 0.0f;
124 
125     InitPosture( *curr_posture );
126 
127     // 各サンプル動作の接続のための変換行列を計算
128     Matrix4f  init_frame;
129     init_frame.setIdentity();
130     for ( int i=0; i<2; i++ )
131     {
132         // 動作データが読み込まれていなければスキップ
133         if ( !motions[ i ] || !motions[ i ]->motion || !curr_posture )
134             continue;
135 
136         // 動作データから開始姿勢を取得、位置・向きを取得
137         motions[ i ]->motion->GetPosture( motions[ i ]->keytimes[ 0 ], *motion_posture[ i
                ] );
138 
139         // 現在の動作の終了姿勢と次の動作の開始姿勢の姿勢の位置・向きを合わせるための変換
                行列を計算
140         //(curr_posture の位置・向きを、next_motion_posture の位置向きに合わせるための変
                換行列 trans_mat を計算)
141         ComputeConnectionTransformation( *curr_posture, 0.0f, *motion_posture[ i ], 180.0
                f, motions[ i ]->base_segment_no, motion_trans_mat[ i ] );
142     }
143 
144     Animation( 0.0f );
145 }
146 
147 
148 //
149 //  画面描画
150 //
151 void  MotionInterpolationApp::Display()
152 {
153     GLUTBaseApp::Display();
154 
155     // キャラクタを描画
156     if ( curr_posture )
157     {
158         glColor3f( figure_color.x, figure_color.y, figure_color.z );
159         DrawPosture( *curr_posture );
160         DrawPostureShadow( *curr_posture, shadow_dir, shadow_color );
161     }
162 
163     // 現在のモード、補間重みを表示
```

```
164     DrawTextInformation( 0, "Motion Interpolation" );
165     char  message[ 64 ];
166     sprintf( message, "%.2f", animation_time );
167     DrawTextInformation( 1, message );
168     sprintf( message, "Weight: %.2f", weight );
169     DrawTextInformation( 2, message );
170 }
171
172
173 //
174 //  マウスドラッグ
175 //
176 void  MotionInterpolationApp::MouseDrag( int mx, int my )
177 {
178     // 左ボタンの左右ドラッグに応じて重みを計算
179     if ( drag_mouse_l )
180     {
181         // 重み計算
182         weight += (float) ( mx - last_mouse_x ) * 2.0f / win_width;
183         if ( weight < 0.0f )
184             weight = 0.0f;
185         if ( weight > 1.0f )
186             weight = 1.0f;
187
188         // 姿勢更新
189         if ( on_animation )
190             Animation( 0.0f );
191         else
192         {
193             on_animation = true;
194             Animation( 0.0f );
195             on_animation = false;
196         }
197     }
198
199     GLUTBaseApp::MouseDrag( mx, my );
200 }
201
202
203 //
204 //  キーボードのキー押下
205 //
206 void  MotionInterpolationApp::Keyboard( unsigned char key, int mx, int my )
207 {
208     GLUTBaseApp::Keyboard( key, mx, my );
209
210     // s キーでアニメーションの停止・再開
211     if ( key == 's' )
212         on_animation = !on_animation;
213
214     // w キーでアニメーションの再生速度を変更
215     if ( key == 'w' )
216         animation_speed = ( animation_speed == 1.0f ) ? 0.1f : 1.0f;
217
218     // n キーで次のフレーム
219     if ( ( key == 'n' ) && !on_animation && motions[ 0 ] )
220     {
221         on_animation = true;
222         Animation( motions[ 0 ]->motion->interval );
223         on_animation = false;
224     }
225
226     // p キーで前のフレーム
227     if ( ( key == 'p' ) && !on_animation && motions[ 0 ] )
```

```
228     {
229         on_animation = true;
230         Animation( - motions[ 0 ]->motion->interval );
231         on_animation = false;
232     }
233 }
234 
235 
236 //
237 //  アニメーション処理
238 //
239 void  MotionInterpolationApp::Animation( float delta )
240 {
241     // アニメーション再生中でなければ終了
242     if ( !on_animation )
243         return;
244 
245     // 動作再生処理（動作補間＋動作接続）
246     AnimationWithInterpolation( delta );
247 
248     // 注視点を更新
249     view_center.set( curr_posture->root_pos.x, 0.0f, curr_posture->root_pos.z );
250 }
251 
252 
253 //
254 //  動作再生処理（動作補間＋動作接続）
255 //
256 void  MotionInterpolationApp::AnimationWithInterpolation( float delta )
257 {
258     // 補間動作のキー時刻の配列
259     int  num_keyframes = motions[ 0 ]->keytimes.size();
260     vector< float >  keytimes( num_keyframes );
261 
262     // 補間動作の現在時刻（動作開始時を基準とする時間）
263     float  local_time = 0.0f;
264 
265     // 正規化時間（区間番号と区間内の正規化時間）を計算
266     int  seg_no = 0;
267     float  seg_time = 0.0f;
268 
269     // 各サンプル動作の現在時刻（動作データ内の時間）
270     float  motion_time = 0.0f;
271 
272 
273     // 時間を進める
274     animation_time += delta * animation_speed;
275 
276     // サンプル動作が設定されていなければ終了
277     if ( !motions[ 0 ] || !motions[ 1 ] || !motions[ 0 ]->motion || !motions[ 1 ]->
            motion )
278         return;
279 
280     // 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）
281     keytimes[ 0 ] = 0.0f;
282     for ( int i = 1; i < num_keyframes; i++ )
283     {
284         // ※ レポート課題
285 //      keytimes[ i ] = ???;
286     }
287 
288     // 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
289     local_time = animation_time - cycle_start_time;
290 
```

```
291     // 現在の繰り返し動作が終了したら、次の繰り返しを開始
292     if ( local_time > keytimes[ num_keyframes - 1 ] )
293     {
294         cycle_start_time += keytimes[ num_keyframes - 1 ];
295         local_time = animation_time - cycle_start_time;
296 
297         // 各サンプル動作の接続のための変換行列を計算
298         for ( int i = 0; i < 2; i++ )
299         {
300             // 動作データから開始姿勢を取得、位置・向きを取得
301             motions[ i ]->motion->GetPosture( motions[ i ]->keytimes[ 0 ], *motion_posture
                    [ i ] );
302 
303             // 現在の動作の終了姿勢と次の動作の開始姿勢の姿勢の位置・向きを合わせるための
                    変換行列を計算
304             //(curr_posture の位置・向きを、next_motion_posture の位置向きに合わせるため
                    の変換行列 trans_mat を計算)
305             ComputeConnectionTransformation( *curr_posture, 0.0f, *motion_posture[ i ],
                    180.0f, motions[ i ]->base_segment_no, motion_trans_mat[ i ] );
306         }
307     }
308 
309     // 正規化時間(区間番号と区間内の正規化時間)を計算
310     for ( int i = 0; i < num_keyframes-1; i++ )
311     {
312         if ( ( local_time >= keytimes[ i ] ) && ( local_time <= keytimes[ i + 1 ] ) )
313         {
314             seg_no = i;
315 
316             // ※ レポート課題
317 //          seg_time = ???;
318 
319             break;
320         }
321     }
322 
323     // サンプル動作から姿勢を取得
324     for ( int i = 0; i < 2; i++ )
325     {
326         // 各サンプル動作の現在時刻(動作データ内の時間)を計算
327 
328         // ※ レポート課題
329 //      motion_time = ???;
330 
331         // 動作データから姿勢を取得
332         motions[ i ]->motion->GetPosture( motion_time, *motion_posture[ i ] );
333 
334         // 前の動作の終了時の位置・向きに合わせるための変換行列を適用
335         TransformPosture( motion_trans_mat[ i ], *motion_posture[ i ] );
336     }
337 
338     MyPostureInterpolation( *motion_posture[ 0 ], *motion_posture[ 1 ], weight, *
            curr_posture );
339 
340     // 重みに応じて描画色を設定
341     figure_color.scaleAdd( weight, motions[ 1 ]->color - motions[ 0 ]->color, motions[ 0
            ]->color );
342 }
```

2.4.1 動作補間の動作再生処理（レポート課題）

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

２つのサンプル動作から姿勢を取得して、設定された重み（float weight）で姿勢補間を行う。姿勢補間の計算には、ここまでに作成した処理が利用できるので、実質的に新しく追加する必要がある処理は、各サンプル動作からどの時刻の姿勢を取得する必要があるか、現在のアニメーション時刻に対応する各サンプル動作の時刻を求める処理となる。

処理の流れは、以下の通り。

1. 現在の時刻に対応する、区間番号（0〜キー区間数）と正規化時間（0.0〜1.0）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）（ 空欄 A ～ 空欄 B ）
   各サンプル動作における前後のキー時刻間の時間を、重みに応じて補間することで計算。
   2 区間番号（0〜キー区間数）と正規化時間（0.0〜1.0）を計算（ 空欄 C ）
   上で求めた補間動作の各キー時刻にもとづいて、現在時刻（animation_time）に対応する区間番号と区間内での正規化時間を計算する。
2. 各サンプル動作に対して、区間番号と正規化時間に対応するサンプル動作の時刻を計算して姿勢を取得
   1 サンプル動作の各キー時刻と、上で求めた区間番号・正規化時間にもとづいて、サンプル動作での時刻（motion_time[i]）を計算（ 空欄 D ～ 空欄 E ）
   2 各サンプル動作から、求めた時刻の姿勢（motion_posture[i]）を取得
3. 各サンプル動作から取得した姿勢を、指定された重み（weight）に応じて補間

ソースコード 28: 動作補間の動作再生処理

```
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 ] = ( 空欄A ) * ( 1.0f - weight ) + ( 空欄B ) * weight;
   }

   // 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 = 空欄C ;

            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 = 空欄D + seg_time * ( 空欄E );

        // 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
}
```

2.5 動作変形

動作変形アプリケーションを実現する、MotionDeformationApp クラスの処理を作成する。動作中のキー姿勢を変形することで、動作全体を変形して再生する。スペースキーで、変形動作の再生モードと、キー姿勢の変形モードを切替える。キー姿勢の変形モードでは、任意の関節をドラッグすることで、姿勢を変形する。(操作方法の詳細は、2.7 節の逆運動学計算（CCD 法）の操作方法の説明を参照。）タイムライン上でクリックすることで、動作変形を適用する範囲の開始・終了時間を設定する。変形動作の再生中に、D キーで、変形前の動作（白）と変形後の動作（緑）の描画を切替える。

MotionDeformationApp クラスの定義・実装を、ソースコード 29・30 に示す。本クラスについては、一部の処理はサンプルプログラムでは空欄となっているため、各自で処理を追加する必要がある。

キー姿勢の変形機能を含む動作変形アプリケーションは、MotionDeformationEditApp クラスにより実現される。MotionDeformationEditApp は、MotionDeformationApp の派生クラスであり、サンプルプログラムでは、MotionDeformationEditApp クラスが動作変形アプリケーションとして使用されるようになっている。MotionDeformationEditApp クラスについては、全ての処理がサンプルプログラムで実装されているため、各自で処理を追加する必要はない。

キー姿勢の変形を行うために、2.7 節で説明する、逆運動学計算（CCD 法）を呼び出して使用する。逆運動学計算（CCD 法）が未作成でも、本アプリケーションを作成することはできるが、キー姿勢の変形モードでの姿勢変形は行えない。

動作変形のタイミングを可視化・操作するために、画面の下にタイムラインを描画する。タイムラインの描画には、Timeline.h/cpp で定義・実装されている Timeline クラスを使用する。Timeline クラスの定義を、ソースコード 31 に示す。(Timeline クラスの実装は、ここでは省略する。）MotionDeformationApp クラスのメンバ変数として、Timeline のオブジェクトを持ち、そのメンバ関数を呼び出すことで、タイムライン描画の機能を利用する。

ソースコード 29: 動作補間アプリケーションの定義（MotionDeformationApp.h）

```
1 /**
2 ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプ
      ログラム
3 ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
```

```
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7  /**
8  ***  動作変形アプリケーション
9  **/
10
11 #ifndef  _MOTION_DEFORMATION_APP_H_
12 #define  _MOTION_DEFORMATION_APP_H_
13
14
15 // ライブラリ・クラス定義の読み込み
16 #include "SimpleHuman.h"
17 #include "SimpleHumanGLUT.h"
18 #include "InverseKinematicsCCDApp.h"
19
20
21 // プロトタイプ宣言
22 class  Timeline;
23
24
25
26 //
27 //  動作変形（動作ワーピング）の情報（動作中の１つのキー時刻の姿勢を変形）
28 //
29 struct  MotionWarpingParam
30 {
31    // 姿勢変形を適用するキー時刻
32    float         key_time;
33
34    // 変形前のキー時刻の姿勢
35    Posture       org_pose;
36
37    // 変形後のキー時刻の姿勢
38    Posture       key_pose;
39
40    // 姿勢変形の前後のブレンド時間
41    float         blend_in_duration;
42    float         blend_out_duration;
43 };
44
45
46 //
47 //  動作変形アプリケーションクラス
48 //
49 class  MotionDeformationApp : public InverseKinematicsCCDApp
50 {
51   protected:
52    // 動作＋動作変形情報
53
54    // 動作変形を適用する動作データ
55    Motion *           motion;
56
57    // 動作変形情報
58    MotionWarpingParam deformation;
59
60   protected:
61    // 動作再生のための変数
62
63    // 変形前のキャラクタの姿勢
64    Posture *          org_posture;
65
66    // 変形後のキャラクタの姿勢
67    Posture *          deformed_posture;
```

```
68
69      // アニメーション再生中かどうかを表すフラグ
70      bool                on_animation;
71
72      // 動作の再生時間
73      float               animation_time;
74
75      // 動作の再生速度
76      float               animation_speed;
77
78      // 現在の表示フレーム番号
79      int                 frame_no;
80
81    protected:
82      // 画面描画のための変数
83
84      // 動作変形前の姿勢・キー姿勢の描画フラグ
85      bool                draw_original_posture;
86      bool                draw_key_posture;
87      bool                draw_postures_side_by_side;
88
89      // タイムライン描画機能
90      Timeline *          timeline;
91
92
93    public:
94      // コンストラクタ
95      MotionDeformationApp();
96
97      // デストラクタ
98      virtual ~MotionDeformationApp();
99
100   public:
101     // イベント処理
102
103     //  初期化
104     virtual void  Initialize();
105
106     //  開始・リセット
107     virtual void  Start();
108
109     //  画面描画
110     virtual void  Display();
111
112     // ウィンドウサイズ変更
113     virtual void  Reshape( int w, int h );
114
115     // マウスクリック
116     virtual void  MouseClick( int button, int state, int mx, int my );
117
118     //  マウスドラッグ
119     virtual void  MouseDrag( int mx, int my );
120
121     // キーボードのキー押下
122     virtual void  Keyboard( unsigned char key, int mx, int my );
123
124     // アニメーション処理
125     virtual void  Animation( float delta );
126
127   protected:
128     // 内部処理
129
130     // 入力動作・動作変形情報の初期化
131     void  InitMotion( int no );
```

```
132
133     // 動作変形情報にもとづくタイムラインの初期化
134     void  InitTimeline( Timeline * timeline, const Motion & motion, const
            MotionWarpingParam & deform, float curr_time );
135
136     // BVH動作ファイルの読み込み、骨格・姿勢の初期化
137     void  LoadBVH( const char * file_name );
138
139     // 変形後の動作をBVH動作ファイルとして保存
140     void  SaveDeformedMotionAsBVH( const char * file_name );
141 };
142
143
144 //
145 //  動作変形情報にもとづく動作変形処理
146 //
147
148 // 動作変形（動作ワーピング）の情報の初期化
149 void  InitDeformationParameter( const Motion & motion, float key_time, float
        blend_in_duration, float blend_out_duration,
150     MotionWarpingParam & deform );
151
152 // 動作変形（動作ワーピング）の情報の初期化
153 void  InitDeformationParameter( const Motion & motion, float key_time, float
        blend_in_duration, float blend_out_duration,
154     int base_joint_no, int ee_joint_no, Point3f ee_joint_translation,
155     MotionWarpingParam & deform );
156
157 // 動作変形（動作ワーピング）の適用後の動作を生成
158 Motion *  GenerateDeformedMotion( const MotionWarpingParam & deform, const Motion &
        motion );
159
160 // 動作変形（動作ワーピング）の適用後の姿勢の計算
161 float  ApplyMotionDeformation( float time, const MotionWarpingParam & deform, const
        Posture & input_pose, Posture & output_pose );
162
163 // 動作ワーピングの姿勢変形（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 ）
164 void  PostureWarping( const Posture & org, const Posture & src, const Posture & dest,
        float ratio, Posture & p );
165
166
167
168 #endif // _MOTION_DEFORMATION_APP_H_
```

ソースコード 30: 動作補間アプリケーションの実装（MotionDeformationApp.cpp）

```
1  /**
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプ
       ログラム
3  ***  Copyright (c) 2015−, Masaki OSHITA (www.oshita−lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit−license.php
5  **/
6
7  /**
8  ***  動作変形アプリケーション
9  **/
10
11
12 // ライブラリ・クラス定義の読み込み
13 #include "SimpleHuman.h"
14 #include "BVH.h"
15 #include "Timeline.h"
16 #include "MotionDeformationApp.h"
```

```
17
18 // 標準算術関数・定数の定義
19 #define  _USE_MATH_DEFINES
20 #include <math.h>
21
22
23
24 //
25 //  コンストラクタ
26 //
27 MotionDeformationApp::MotionDeformationApp() : InverseKinematicsCCDApp()
28 {
29     app_name = "Motion Deformation Base";
30
31     motion = NULL;
32
33     org_posture = NULL;
34     deformed_posture = NULL;
35     on_animation = true;
36     animation_time = 0.0f;
37     animation_speed = 1.0f;
38     frame_no = 0;
39
40     draw_original_posture = false;
41     draw_postures_side_by_side = false;
42     timeline = NULL;
43 }
44
45
46 //
47 //  デストラクタ
48 //
49 MotionDeformationApp::~MotionDeformationApp()
50 {
51     if ( motion )
52         delete  motion;
53     if ( curr_posture && curr_posture->body )
54         delete  curr_posture->body;
55     if ( curr_posture )
56         delete  curr_posture;
57     if ( org_posture )
58         delete  org_posture;
59     if ( deformed_posture )
60         delete  deformed_posture;
61     if ( timeline )
62         delete  timeline;
63 }
64
65
66 //
67 //  初期化
68 //
69 void  MotionDeformationApp::Initialize()
70 {
71     GLUTBaseApp::Initialize();
72
73     // 入力動作・動作変形情報の初期化
74     InitMotion( 0 );
75
76     // 姿勢描画の設定
77     draw_original_posture = true;
78     draw_key_posture = true;
79     draw_postures_side_by_side = false;
80
```

```
81      // タイムライン描画機能の初期化
82      timeline = new Timeline();
83      if ( motion )
84          InitTimeline( timeline, *motion, deformation, 0.0f );
85
86      // 初期の視点を設定
87      camera_yaw = -90.0f;
88      camera_distance = 4.0f;
89  }
90
91
92  //
93  //  開始・リセット
94  //
95  void  MotionDeformationApp::Start()
96  {
97      GLUTBaseApp::Start();
98
99      on_animation = true;
100     animation_time = 0.0f;
101     frame_no = 0;
102     Animation( 0.0f );
103 }
104
105
106 //
107 //  画面描画
108 //
109 void  MotionDeformationApp::Display()
110 {
111     GLUTBaseApp::Display();
112
113     // 動作変形後の姿勢を描画
114     if ( deformed_posture )
115     {
116         glPushMatrix();
117
118         glColor3f( 0.5f, 1.0f, 0.5f );
119         DrawPosture( *deformed_posture );
120         DrawPostureShadow( *deformed_posture, shadow_dir, shadow_color );
121
122         glPopMatrix();
123     }
124
125     // 動作変形前の姿勢も描画（比較用）
126     if ( draw_original_posture && org_posture )
127     {
128         glPushMatrix();
129
130         if ( draw_postures_side_by_side )
131             glTranslatef( -1.0f, 0.0f, 0.0f );
132
133         glColor3f( 1.0f, 1.0f, 1.0f );
134         DrawPosture( *org_posture );
135         DrawPostureShadow( *org_posture, shadow_dir, shadow_color );
136
137         glPopMatrix();
138     }
139
140     // 動作変形に使用するキー姿勢を描画（比較用）
141     if ( draw_key_posture && draw_postures_side_by_side )
142     {
143         glPushMatrix();
144
```

```
145         glTranslatef( 1.0f, 0.0f, 0.0f );
146
147         glColor3f( 0.5f, 1.0f, 0.5f );
148         DrawPosture( deformation.key_pose );
149         DrawPostureShadow( deformation.key_pose, shadow_dir, shadow_color );
150
151         glPopMatrix();
152     }
153
154     // タイムラインを描画
155     if ( timeline )
156     {
157         timeline->SetLineTime( 1, animation_time );
158         timeline->DrawTimeline();
159     }
160
161     // 現在のモード、時間・フレーム番号を表示
162     DrawTextInformation( 0, "Motion Deformation Base" );
163     char  message[64];
164     if ( motion )
165     {
166         sprintf( message, "%.2f (%d)", animation_time, frame_no );
167         DrawTextInformation( 1, message );
168     }
169 }
170
171
172 //
173 //  ウィンドウサイズ変更
174 //
175 void  MotionDeformationApp::Reshape( int w, int h )
176 {
177     GLUTBaseApp::Reshape( w, h );
178
179     // タイムラインの描画領域の設定（画面の下部に配置）
180     if ( timeline )
181         timeline->SetViewAreaBottom( 0, 0, 0, 2, 32, 2 );
182 }
183
184
185 //
186 //  マウスクリック
187 //
188 void  MotionDeformationApp::MouseClick( int button, int state, int mx, int my )
189 {
190     GLUTBaseApp::MouseClick( button, state, mx, my );
191
192     // マウス座標に対応するタイムラインのトラック番号・時刻を取得
193     int  selected_track_no = timeline->GetTrackByPosition( mx, my );
194     float  selected_time = timeline->GetTimeByPosition( mx );
195
196     // 入力動作トラック上のクリック位置に応じて、変形動作の再生時刻を変更
197     if ( drag_mouse_l && ( selected_track_no == 0 ) )
198     {
199         animation_time = selected_time;
200         Animation( 0.0f );
201     }
202 }
203
204
205 //
206 //  マウスドラッグ
207 //
208 void  MotionDeformationApp::MouseDrag( int mx, int my )
```

```
209 {
210     GLUTBaseApp::MouseDrag( mx, my );
211 
212     // マウス座標に対応するタイムラインのトラック番号・時刻を取得
213     int   selected_track_no = timeline->GetTrackByPosition( mx, my );
214     float  selected_time = timeline->GetTimeByPosition( mx );
215 
216     // 変形動作の再生時刻を変更
217     if ( drag_mouse_l && ( selected_track_no == 0 ) )
218     {
219         animation_time = selected_time;
220         drag_mouse_l = false;
221         Animation( 0.0f );
222         drag_mouse_l = true;
223     }
224 }
225 
226 
227 //
228 //  キーボードのキー押下
229 //
230 void  MotionDeformationApp::Keyboard( unsigned char key, int mx, int my )
231 {
232     GLUTBaseApp::Keyboard( key, mx, my );
233 
234     // 数字キーで入力動作・動作変形情報を変更
235     if ( ( key >= '1' ) && ( key <= '9' ) )
236     {
237         InitMotion( key - '1' );
238     }
239 
240     // d キーで表示姿勢の変更
241     if ( key == 'd' )
242     {
243         if ( !draw_original_posture )
244         {
245             draw_original_posture = true;
246             draw_postures_side_by_side = false;
247         }
248         else if ( draw_original_posture && !draw_postures_side_by_side )
249         {
250             draw_key_posture = true;
251             draw_postures_side_by_side = true;
252         }
253         else if ( draw_original_posture && draw_postures_side_by_side )
254         {
255             draw_key_posture = false;
256             draw_original_posture = false;
257         }
258     }
259 
260     // s キーでアニメーションの停止・再開
261     if ( key == 's' )
262         on_animation = !on_animation;
263 
264     // w キーでアニメーションの再生速度を変更
265     if ( key == 'w' )
266         animation_speed = ( animation_speed == 1.0f ) ? 0.1f : 1.0f;
267 
268     // n キーで次のフレーム
269     if ( ( key == 'n' ) && !on_animation && motion )
270     {
271         on_animation = true;
272         Animation( motion->interval );
```

```
273         on_animation = false;
274     }
275
276     // p キーで前のフレーム
277     if ( ( key == 'p' ) && !on_animation && motion && ( frame_no > 0 ) )
278     {
279         on_animation = true;
280         Animation( - motion->interval );
281         on_animation = false;
282     }
283
284     // r キーでリセット
285     if ( key == 'r' )
286         Start();
287
288     // o キーで変形後の動作を保存
289     if ( key == 'o' )
290     {
291         // 変形後の動作をBVH動作ファイルとして保存
292         SaveDeformedMotionAsBVH( "deformed_motion.bvh" );
293     }
294 }
295
296
297 //
298 //  アニメーション処理
299 //
300 void  MotionDeformationApp::Animation( float delta )
301 {
302     // アニメーション再生中でなければ終了
303     if ( !on_animation )
304         return;
305
306     // マウスドラッグ中はアニメーションを停止
307     if ( drag_mouse_l )
308         return;
309
310     // 動作データが読み込まれていなければ終了
311     if ( !motion )
312         return;
313
314     // 時間を進める
315     animation_time += delta * animation_speed;
316     if ( animation_time > motion->GetDuration() )
317         animation_time -= motion->GetDuration();
318
319     // 現在のフレーム番号を計算
320     frame_no = animation_time / motion->interval;
321
322     // 動作データから現在時刻の姿勢を取得
323     motion->GetPosture( animation_time, *org_posture );
324
325     // 動作変形（動作ワーピング）の適用後の姿勢の計算
326     ApplyMotionDeformation( animation_time, deformation, *org_posture, *deformed_posture
          );
327 }
328
329
330 //
331 //  入力動作・動作変形情報の初期化
332 //
333 void  MotionDeformationApp::InitMotion( int no )
334 {
335     // テストケース1
```

```
336     if ( no == 0 )
337     {
338         // サンプルBVH動作データを読み込み
339         LoadBVH( "fight_punch.bvh" );
340         if ( !motion )
341             return;
342 
343         // キー姿勢のBHV動作データを読み込み
344         BVH *  key_bvh = new BVH( "fight_punch_key.bvh" );
345 
346         // キー姿勢のBVH動作データにもとづいて、変形情報（キー姿勢）を設定
347         if ( key_bvh->IsLoadSuccess() )
348         {
349             // BVH動作から動作データを生成
350             Motion *  key_pose_motion = CoustructBVHMotion( key_bvh );
351 
352             // 動作変形（動作ワーピング）情報の初期化（キー時刻を指定）
353             InitDeformationParameter( *motion, 0.80f, 0.70f, 0.50f, deformation );
354 
355             // 動作変形（動作ワーピング）情報のキー姿勢を設定
356             deformation.key_pose = *key_pose_motion->GetFrame( 0 );
357             deformation.key_pose.body = motion->body;
358 
359             // 使用済みデータの削除
360             delete  key_pose_motion;
361         }
362 
363         // 逆運動学計算を使用して、変形情報（キー姿勢）を設定
364         else
365         {
366             // 動作変形（動作ワーピング）情報の初期化（キー時刻＋キー姿勢の右手の目標位置
                    を指定）
367             InitDeformationParameter( *motion, 0.80f, 0.70f, 0.50f, 0, 15, Vector3f( 0.0f,
                    -0.2f, 0.0f ), deformation );
368         }
369 
370         // 使用済みデータの削除
371         delete  key_bvh;
372     }
373 
374     // 以下、他のテストケースを追加する
375     else if ( no == 1 )
376     {
377     }
378 }
379 
380 
381 //
382 //  動作変形情報にもとづくタイムラインの初期化
383 //
384 void  MotionDeformationApp::InitTimeline( Timeline * timeline, const Motion & motion,
        const MotionWarpingParam & deform, float curr_time )
385 {
386     // タイムラインの時間範囲を設定
387     timeline->SetTimeRange( -0.25f, motion.GetDuration() + 0.25f );
388 
389     // 全要素・縦棒の情報をクリア
390     timeline->DeleteAllElements();
391     timeline->DeleteAllLines();
392 
393     // 変形前の動作を表す要素を設定
394     timeline->AddElement( 0.0f, motion.GetDuration(), Color4f( 1.0f, 1.0f, 1.0f, 1.0f ),
            motion.name.c_str(), 0 );
395 
```

```
396     // 動作変形の範囲を表す要素を設定
397     timeline->AddElement( deform.key_time - deform.blend_in_duration, deform.key_time +
            deform.blend_out_duration,
398         Color4f( 0.5f, 1.0f, 0.5f, 1.0f ), "deformation", 1 );
399
400     // 動作変形のキー時刻を表す縦線を設定
401     timeline->AddLine( deform.key_time, Color4f( 1.0f, 0.0f, 0.0f, 1.0f ) );
402
403     // 動作再生時刻を表す縦線を設定
404     timeline->AddLine( curr_time, Color4f( 0.0f, 0.0f, 0.0f, 1.0f ) );
405 }
406
407
408 //
409 //  BVH動作ファイルの読み込み、骨格・姿勢の初期化
410 //
411 void  MotionDeformationApp::LoadBVH( const char * file_name )
412 {
413     // BVHファイルを読み込んで動作データ（＋骨格モデル）を生成
414     Motion *  new_motion = LoadAndCoustructBVHMotion( file_name );
415
416     // BVHファイルの読み込みに失敗したら終了
417     if ( !new_motion )
418         return;
419
420     // 骨格・動作・姿勢の削除
421     if ( motion && motion->body )
422         delete  motion->body;
423     if ( motion )
424         delete  motion;
425     if ( curr_posture )
426         delete  curr_posture;
427     if ( org_posture )
428         delete  org_posture;
429     if ( deformed_posture )
430         delete  deformed_posture;
431
432     // 動作変形に使用する動作・姿勢の初期化
433     motion = new_motion;
434     curr_posture = new Posture();
435     InitPosture( *curr_posture, motion->body );
436     org_posture = new Posture();
437     InitPosture( *org_posture, motion->body );
438     deformed_posture = new Posture();
439     InitPosture( *deformed_posture, motion->body );
440 }
441
442
443 //
444 //  変形後の動作をBVH動作ファイルとして保存
445 //
446 void  MotionDeformationApp::SaveDeformedMotionAsBVH( const char * file_name )
447 {
448     // 動作変形適用後の動作を生成
449     Motion *  deformed_motion = GenerateDeformedMotion( deformation, *motion );
450
451     // 動作変形適用後の動作を保存
452     // ※省略（各自作成）
453
454
455     // 変形適用後の動作を削除
456     delete  deformed_motion;
457 }
458
```

```
459 |
460 |
461 | //
462 | //  動作変形情報にもとづく動作変形処理
463 | //
464 |
465 |
466 | //
467 | //  動作変形（動作ワーピング）の情報の初期化
468 | //
469 | void  InitDeformationParameter(
470 |    const Motion & motion, float key_time, float blend_in_duration, float
        blend_out_duration,
471 |    MotionWarpingParam & param )
472 | {
473 |    param.key_time = key_time;
474 |    param.blend_in_duration = blend_in_duration;
475 |    param.blend_out_duration = blend_out_duration;
476 |    motion.GetPosture( param.key_time, param.org_pose );
477 |    param.key_pose = param.org_pose;
478 | }
479 |
480 |
481 | //
482 | //  動作変形（動作ワーピング）の情報の初期化
483 | //
484 | void  InitDeformationParameter(
485 |    const Motion & motion, float key_time, float blend_in_duration, float
        blend_out_duration,
486 |    int base_joint_no, int ee_joint_no, Point3f ee_joint_translation,
487 |    MotionWarpingParam & param )
488 | {
489 |    InitDeformationParameter( motion, key_time, blend_in_duration, blend_out_duration,
        param );
490 |
491 |    // 順運動学計算
492 |    vector< Matrix4f >  seg_frame_array;
493 |    vector< Point3f >  joint_position_frame_array;
494 |    ForwardKinematics( param.key_pose, seg_frame_array, joint_position_frame_array );
495 |
496 |    // 指定関節の目標位置
497 |    Point3f  ee_pos;
498 |    ee_pos = joint_position_frame_array[ ee_joint_no ];
499 |    ee_pos.add( ee_joint_translation );
500 |
501 |    // キー姿勢の指定部位の位置を移動
502 |    ApplyInverseKinematicsCCD( param.key_pose, base_joint_no, ee_joint_no, ee_pos );
503 | }
504 |
505 |
506 | //
507 | //  動作変形（動作ワーピング）の適用後の動作を生成
508 | //
509 | Motion *  GenerateDeformedMotion( const MotionWarpingParam & deform, const Motion &
    motion )
510 | {
511 |    Motion *  deformed = NULL;
512 |
513 |    // 動作変形前の動作を生成
514 |    deformed = new Motion( motion );
515 |
516 |    // 各フレームの姿勢を変形
517 |    for ( int i = 0; i < motion.num_frames; i++ )
518 |       ApplyMotionDeformation( motion.interval * i, deform, motion.frames[ i ], deformed
```

```
          ->frames[ i ] );

    // 動作変形後の動作を返す
    return  deformed;
}


//
//  動作変形（動作ワーピング）の適用後の姿勢の計算
//  （変形適用の重み 0.0～1.0 を返す）
//
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
    if ( ( time < deform.key_time - deform.blend_in_duration ) ||
         ( time > deform.key_time + deform.blend_out_duration ) )
    {
        output_pose = input_pose;
        return  0.0f;
    }


    // ※ レポート課題

    // 姿勢変形（動作ワーピング）の重みを計算
    float  ratio = 0.5f;
// 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;
}


//
//  動作ワーピングの姿勢変形（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）
//
void  PostureWarping( const Posture & org, const Posture & src, const Posture & dest, float ratio, Posture & p )
{
    // ３つの姿勢の骨格モデルが異なる場合は終了
    if ( ( org.body != src.body ) || ( src.body != dest.body ) || ( dest.body != p.body ) )
        return;

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


    // ※ 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 = ???;
```

```
577
578     // ルートの位置を計算
579 // p.root_pos = ???;
580 }
```

ソースコード 31: タイムライン描画クラスの定義（Timeline.h）

```
 1 /**
 2 ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプログラム
 3 ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
 4 ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
 5 **/
 6
 7 /**
 8 ***  タイムライン描画機能
 9 **/
10
11 #ifndef  _TIMELINE_H_
12 #define  _TIMELINE_H_
13
14
15 #include <vecmath.h>
16
17 #include <vector>
18 using namespace  std;
19
20
21
22 //
23 //  タイムライン描画機能クラス
24 //
25 class  Timeline
26 {
27   public:
28     /*  データ構造の定義  */
29
30     // 要素情報
31     struct  ElementInfo
32     {
33       /*  設定情報  */
34
35       // 有効フラグ
36       bool              is_enable;
37
38       // 時刻範囲（開始・終了時刻）
39       float             start_time;
40       float             end_time;
41
42       // 描画色
43       Color4f           color;
44
45       // グラデーションで描画するかのフラグと描画色
46       bool              gradation;
47       Color4f           color2;
48
49       // 表示テキスト
50       string            text;
51
52       // 描画するトラック番号
53       int               track_no;
54
55       // 親要素番号（サブ要素のために使用）
56       int               parent_no;
```

```
57
58          // 描画領域の高さの範囲（0.0～1.0）（サブ要素のために使用）
59          float               height_top;
60          float               height_bottom;
61
62          /*  描画用情報  */
63
64          // 要素の描画フラグ
65          bool                draw_flag;
66
67          // スクリーン上の要素の描画位置
68          int                 screen_x0;
69          int                 screen_x1;
70          int                 screen_y0;
71          int                 screen_y1;
72
73          // テキストの描画フラグ
74          bool                draw_text_flag;
75
76          // スクリーン上のテキストの描画位置
77          int                 screen_text_x;
78          int                 screen_text_y;
79      };
80
81      // 縦線情報
82      struct  LineInfo
83      {
84          /*  設定情報  */
85
86          // 有効フラグ
87          bool                is_enable;
88
89          // 時刻
90          float               time;
91
92          // 描画色
93          Color4f             color;
94
95          // 描画するトラック番号の範囲
96          int                 start_track;
97          int                 end_track;
98
99          /*  描画用情報  */
100
101         // 縦棒の描画フラグ
102         bool                draw_flag;
103
104         // スクリーン上の縦棒の描画位置
105         int                 screen_x;
106         int                 screen_y0;
107         int                 screen_y1;
108     };
109
110     // 軌道情報
111     struct  TrajectoryInfo
112     {
113         /*  設定情報  */
114
115         // 有効フラグ
116         bool                is_enable;
117
118         // 時刻範囲（開始・終了時刻）
119         float               start_time;
120         float               end_time;
```

```
121 |
122 |       // 描画色
123 |       Color4f            color;
124 |
125 |       // 軌道の値の配列
126 //     int                 num_values;
127 //     float               values;
128 |       vector< float >    values;
129 |
130 |       // 軌道の値の範囲
131 |       float              value_min;
132 |       float              value_max;
133 |
134 |       // 描画するトラック番号
135 |       int                track_no;
136 |
137 |       // 親要素番号（サブ要素のために使用）
138 //     int                 parent_no;
139 |
140 |       /*  描画用情報  */
141 |
142 |       // 要素の描画フラグ
143 |       bool               draw_flag;
144 |
145 |       // スクリーン上の要素の描画位置
146 |       int                screen_x0;
147 |       int                screen_x1;
148 |       int                screen_y0;
149 |       int                screen_y1;
150 |     };
151 |
152 |   protected:
153 |     /*  描画用の内部情報  */
154 |
155 |     // タイムラインの描画領域
156 |     int                    view_width;
157 |     int                    view_height;
158 |     int                    view_left;
159 |     int                    view_top;
160 |
161 |     // 描画トラック数
162 |     int                    num_tracks;
163 |
164 |     // 各トラックの高さ
165 |     int                    track_height;
166 |
167 |     // 背景描画の設定
168 |     bool                   draw_background;
169 |     Color4f                background_color;
170 |
171 |     // ライン間の間隔の描画の設定
172 |     int                    margin_height;
173 |     bool                   draw_margin;
174 |     Color4f                margin_color;
175 |
176 |     // 描画時刻範囲
177 |     float                  view_start_time;
178 |     float                  view_end_time;
179 |
180 |   protected:
181 |     /*  タイムライン情報  */
182 |
183 |     // 要素情報
184 |     vector< ElementInfo >  all_elements;
```

```
185
186     // サブ要素情報
187     vector< ElementInfo >   all_sub_elements;
188
189     // 縦線情報
190     vector< LineInfo >      all_lines;
191
192     // 軌道情報
193     vector< TrajectoryInfo >  all_trajectories;
194
195     // タイムライン情報の更新フラグ
196     bool                    is_updated;
197
198   public:
199     // コンストラクタ
200     Timeline();
201
202     // デストラクタ
203     virtual ~Timeline();
204
205   public:
206     // タイムラインの設定
207
208     // 描画領域の設定（画面の任意の位置に配置）
209     void  SetViewArea( int left, int right, int top, int num_tracks, int track_height,
          int margin_hight );
210
211     // 描画領域の設定（画面の下部に配置）
212     void  SetViewAreaBottom( int left_margin, int right_margin, int bottom_margin, int
          num_tracks, int track_height, int margin_hight );
213
214     // 描画時刻範囲の設定
215     void  SetTimeRange( float start, float end );
216
217     // 描画色の設定
218     void  SetBackgroundColor( const Color4f & color );
219     void  SetMarginColor( const Color4f & color );
220     void  ClearColor();
221
222   public:
223     // タイムラインに含まれる要素・縦線の設定
224
225     // 要素の追加
226     int  AddElement( float start_time, float end_time, const Color4f & color, const char
           * text = NULL, int track_no = -1 );
227
228     // 要素の削除
229     void  DeleteElement( int no );
230     void  DeleteAllElements();
231
232     // 要素の情報更新
233     void  SetElementEnable( int no, bool is_enable );
234     void  SetElementTime( int no, float start_time, float end_time );
235     void  SetElementColor( int no, const Color4f & color );
236     void  SetElementColor( int no, const Color4f & color, const Color4f & color2 );
237     void  SetElementText( int no, const char * text );
238     void  SetElementTrackNo( int no, int track_no );
239
240     // サブ要素の追加
241     int  AddSubElement( int parent, float start_time, float end_time, float top, float
          bottom, const Color4f & color );
242
243     // サブ要素の削除
244     void  DeleteSubElement( int no );
```

```
245     void  DeleteAllSubElements();
246
247     // サブ要素の情報更新
248     void  SetSubElementEnable( int no, bool is_enable );
249     void  SetSubElementParent( int no, int parent );
250     void  SetSubElementTime( int no, float start_time, float end_time );
251     void  SetSubElementHeight( int no, float top, float bottom );
252     void  SetSubElementColor( int no, const Color4f & color );
253     void  SetSubElementColor( int no, const Color4f & color, const Color4f & color2 );
254
255     // 縦線の追加
256     int  AddLine( float time, const Color4f & color, int start_track = -1, int end_track
            = -1 );
257
258     // 縦線の削除
259     void  DeleteLine( int no );
260     void  DeleteAllLines();
261
262     // 縦線の情報更新
263     void  SetLineEnable( int no, bool is_enable );
264     void  SetLineTime( int no, float time );
265     void  SetLineColor( int no, const Color4f & color );
266
267     // 軌道の追加
268     int  AddTrajectory( float start_time, float end_time, const Color4f & color, int
            num_values, const float * values, float value_min, float value_max, int track_no
            = -1 );
269     int  AddTrajectory( float start_time, float end_time, const Color4f & color, vector<
            float > & values, float value_min, float value_max, int track_no = -1 );
270
271     // 軌道の削除
272     void  DeleteTrajectory( int no );
273     void  DeleteAllTrajectories();
274
275     // 軌道の情報更新
276     void  SetTrajectoryEnable( int no, bool is_enable );
277     void  SetTrajectoryTime( int no, float start_time, float end_time );
278     void  SetTrajectoryColor( int no, const Color4f & color );
279     void  SetTrajectoryValues( int no, int num_values, const float * values );
280     void  SetTrajectoryValues( int no, vector< float > & values );
281     void  SetTrajectoryValueMinMax( int no, float value_min, float value_max );
282     void  SetTrajectoryValueMinMaxAuto( int no );
283     void  SetTrajectoryTrackNo( int no, int track_no );
284
285   public:
286     // 情報取得
287     int  GetViewHeight() { return  view_height; }
288     int  GetNumTracks() { return  num_tracks; }
289     int  GetNumElements();
290     int  GetNumSubElements();
291     int  GetNumLines();
292     int  GetNumTrajectories();
293     ElementInfo *  GetElement( int no );
294     ElementInfo *  GetSubElement( int no );
295     LineInfo *  GetLine( int no );
296     TrajectoryInfo *  GetTrajectory( int no );
297
298   public:
299     // 描画処理
300
301     // タイムライン描画
302     virtual void  DrawTimeline();
303
304     // 画面位置に対応するトラック番号を取得（範囲外の場合は -1 を返す）
```

```
305     int  GetTrackByPosition( int screen_x, int screen_y );
306
307     // 画面位置に対応する時刻を取得
308     float  GetTimeByPosition( int screen_x );
309
310   protected:
311     // 内部処理
312
313     // 全要素の描画位置を更新
314     virtual void  UpdateTimeline();
315
316     // 全体の高さを計算
317     int  ComputeViewHeight();
318
319     // スクリーン描画モードの開始・終了
320     void  BeginScreenMode();
321     void  EndScreenMode();
322
323     // 四角形を描画
324     void  DrawRectangle( int x0, int y0, int x1, int y1, const Color4f & color );
325     void  DrawRectangle( int x0, int y0, int x1, int y1, const Color4f & color, const
            Color4f & color2 );
326
327     // 縦線を描画
328     void  DrawLine( int x, int y0, int y1, const Color4f & color );
329
330     // 軌道を描画
331     void  DrawTrajectory( int x0, int y0, int x1, int y1, int num_values, const float *
            values, float value_min, float value_max, const Color4f & color );
332
333     // テキストを描画
334     void  DrawText( int x, int y, const char * text );
335
336     // テキストの描画幅を取得
337     int  Timeline::GetTextWidth( const char * text );
338
339     // テキストの描画高さを取得
340     int  Timeline::GetTextHeight( const char * text );
341 };
342
343
344 #endif // _TIMELINE_H_
```

2.5.1 動作変形の重みの計算（レポート課題）

動作変形（動作ワーピング）を実現するための、姿勢変形の重みの計算を行い、姿勢変形を適用する。処理の流れは、以下の通り。

1. 入力された動作変形情報と現在時刻にもとづいて、姿勢変形の重みを計算する。（ 空欄 A ～ 空欄 C ）動作変形のキー時刻（deform.key_time）において、重みが最大の 1 になり、キー時刻の前後で、重みが 0 から 1 の間で連続的に変化するように、計算を行う。（条件を満たす重みの関数として、さまざまな関数が考えられるが、今回は、重みを線形に変化させる単純な関数を使用する。）

2. 計算した重みに応じて、姿勢変形を適用する。

ソースコード 32: 動作変形の重みの計算

```
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;
    if ( time <= deform.key_time )
        ratio = ( 空欄A ) / ( 空欄B );
    if ( time > deform.key_time )
        ratio = ( 空欄C ) / ( 空欄D );

    // 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;
}
```

2.5.2 動作変形の姿勢変形（レポート課題）

動作変形（動作ワーピング）を実現するための、姿勢変形を行う。入力姿勢（org）を $\mathbf{p}$、キー時刻における変形前の姿勢（src）を $\mathbf{p}_{src}$、キー時刻における変形後の姿勢（dest）を $\mathbf{p}_{dest}$、出力姿勢（p）を $\mathbf{p}'$、重み（ratio）を w とすると、以下の計算式により、出力姿勢を計算できる。

$$\mathbf{p}' = w(\mathbf{p}_{dest} - \mathbf{p}_{src} + \mathbf{p}) + (1 - w)\mathbf{p} \tag{2}$$

処理の流れは、以下の通り。

1. 各関節の回転とルートの向きについては、式の１項目の回転を計算し（ 空欄A ～ 空欄C ）、その後、姿勢補間と同様の方法で１・２項の回転を重みにもとづいて補間する。
   回転行列の加算や減算は、行列のかけ算や逆行列（転置行列）のかけ算により計算されることに注意する。
2. ルートの位置についても、同様に、１・２項の位置を重みにもとづいて補間する（ 空欄E ～ 空欄G ）。
   位置ベクトルの加算や減算は、ベクトルの和や差により計算できる。

ソースコード 33: 動作変形の姿勢変形

```
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++ )
    {
        rot.空欄A ( 空欄B.joint_rotations[ i ], 空欄C.joint_rotations[ i ]
            );
        rot.mul( rot, 空欄D.joint_rotations[ i ] );

        q0.set( org.joint_rotations[ i ] );
        q1.set( rot );
```

```
        // q0 と q1 の間を重み ratio で補間して q に代入（姿勢補間と同様）

        p.joint_rotations[ i ].set( q );
    }

    // ルートの向きを計算
    rot.__空欄A__( __空欄B__.root_ori, __空欄C__.root_ori );
    rot.mul( rot, __空欄D__.root_ori );

    q0.set( org.root_ori );
    q1.set( rot );

    // q0 と q1 の間を重み ratio で補間して q に代入（姿勢補間と同様）

    p.root_ori.set( q );

    // ルートの位置を計算
    v.sub( __空欄E__.root_pos, __空欄F__.root_pos );
    v.scale( ratio );
    p.root_pos.add( v, __空欄G__.root_pos );
}
```

2.6 動作接続・遷移

動作接続・遷移アプリケーションを実現する、MotionTransitionApp クラスの処理を作成する。動作接続・遷移アプリケーションでは、前後のサンプル動作の接続・遷移を行いながら繰り返し動作再生を行う。左クリックにより、現在の動作の終了時に、別の動作に対して動作接続・遷移を行う。入力がなければ、同じ動作に対して接続・遷移を行い、同じ動作を繰り返し再生する。再生中の動作や、動作遷移中の重みの変化に応じて、キャラクタの色を変化。D キーで、動作接続・遷移を適用するか、動作接続のみを適用するか、を切り替えられる。

動作接続・遷移アプリケーションでは、動作接続・遷移の機能は他のアプリケーションからも利用されることと、動作接続・遷移のための変数や処理はやや複雑になるため、動作接続・遷移を実現するためのクラスと、動作接続・遷移を利用するアプリケーションを、別のクラスやソースファイルに分けて作成する。

動作接続・遷移を実現するための MotionConnection・MotionTransition クラスの定義・実装を、ソースコード 36・37 に示す。これらのクラスについては、一部の処理はサンプルプログラムでは空欄となっているため、各自で処理を追加する必要がある。これらのソースコード（MotionTransition.h/cpp）では、動作のメタ情報を表す MotionInfo 構造体、動作接続を実現するための MotionConnection クラス、動作接続・遷移を実現するための MotionTransition クラス、動作接続のためのグローバル関数が、定義・実装されている。MotionConnection クラスは、MotionTransition クラスを派生させる形で定義・実装されている。通常は、基本的な機能を持った基底クラスを作成して、派生クラスで機能を追加することが多いが、本サンプルプログラムでは、動作接続のみの MotionConnection クラスはあまり重要ではないため、基本となる動作接続・遷移ための MotionTransition クラスを基底クラスとして定義・実装して、その機能を簡略化した派生クラスとして MotionConnection クラスを定義・実装している。

また、動作接続・遷移アプリケーションを実現する MotionTransitionApp クラスの定義・実装を、ソースコード 36・37 に示す。本クラスについては、全ての処理がサンプルプログラムで実装されているため、各自で処理を追加する必要はない。

入力動作や動作接続・遷移のタイミングを可視化するために、画面の下にタイムラインを描画する。タイムラインの描画には、Timeline.h/cpp で定義・実装されている Timeline クラスを使用する（ソースコード 31）。MotionTransitionApp クラスのメンバ変数として、Timeline のオブジェクトを持ち、そのメンバ関数を呼び出すことで、タイムライン描画の機能を利用する。

なお、動作遷移における前後の動作の姿勢のブレンドには、2.2 節で説明した、2 つの姿勢の補間を行う MyPostureInterpolation 関数を使用する。そのため、本アプリケーションを作成する前に、MyPostureInterpolation 関

数を作成する必要がある。

ソースコード 34: 動作接続・遷移クラスの定義（MotionTransition.h）

```
1  /**
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプログラム
3  ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7  /**
8  ***  動作接続・遷移
9  **/
10
11 #ifndef  _MOTION_TRANSITION_H_
12 #define  _MOTION_TRANSITION_H_
13
14
15 // ライブラリ・クラス定義の読み込み
16 #include "SimpleHuman.h"
17
18
19 //
20 //  動作のメタ情報を表す構造体
21 //
22 struct  MotionInfo
23 {
24     // 動作情報
25     Motion *  motion;
26
27     // 動作の開始・終了時刻（動作のローカル時間）
28     float  begin_time;
29     float  end_time;
30
31     // 動作接続・遷移のためのブレンド区間の終了・開始時刻（動作のローカル時間）
32     // ( begin_time <= blend_end_time < blend_begin_time <= end_time )
33     float  blend_end_time;
34     float  blend_begin_time;
35
36     // キー時刻の配列 [キーフレーム番号]
37     vector< float >  keytimes;
38
39     // 次の動作への動作接続のための基準部位
40     int  base_segment_no;
41
42     // 動作開始時・終了時（接続時）の水平向き
43     bool  enable_ori;
44     float  begin_ori;
45     float  end_ori;
46
47     // 描画色
48     Color3f  color;
49 };
50
51
52 // 動作のメタ情報を初期化
53 void  InitMotionInfo( MotionInfo * info, Motion * m = NULL );
54
55
56 //
57 //  動作接続・遷移クラス
58 //
59 class  MotionTransition
```

```
60 {
61   public:
62    // 動作遷移の状態
63    enum  MotionTransitionState
64    {
65       MT_NONE,              // 不明
66       MT_PREV_MOTION,       // 前の動作の再生中
67       MT_IN_TRANSITION,     // 前の動作から後の動作への遷移中
68       MT_NEXT_MOTION        // 後の動作の再生中
69    };
70
71   protected:
72    // 動作遷移の入力情報
73
74    // 前の動作
75    const MotionInfo *  prev_motion;
76
77    // 後の動作
78    const MotionInfo *  next_motion;
79
80    // 前の動作の変換行列
81    Matrix4f  prev_motion_mat;
82
83    // 前の動作の開始時刻（グローバル時刻）
84    float  prev_begin_time;
85
86   protected:
87    // 動作遷移のための情報
88
89    // 動作接続のための基準体節番号
90    int  base_segment_no;
91
92    // 後の動作の位置・向きを前の動作の位置・向きに合わせるための変換行列
93    Matrix4f  next_motion_mat;
94
95    // 後の動作の開始時刻（前の動作の開始時刻を基準とするローカル時刻）
96    float  next_begin_time;
97
98    // 動作のブレンド開始時刻（前の動作の開始時刻を基準とするローカル時刻）
99    float  blend_begin_time;
100
101   // 動作のブレンド終了時刻（前の動作の開始時刻を基準とするローカル時刻）
102   float  blend_end_time;
103
104  protected:
105   // 動作遷移中の姿勢補間のための情報
106
107   // 前の動作の姿勢
108   Posture *  prev_motion_posture;
109
110   // 後の動作の姿勢
111   Posture *  next_motion_posture;
112
113   // 最後の姿勢計算時の動作遷移の状態
114   MotionTransitionState  last_state;
115
116   // 最後の姿勢計算時の前後の動作の姿勢補間の重み
117   float  last_blend_ratio;
118
119  public:
120   // コンストラクタ
121   MotionTransition();
122
123   // デストラクタ
```

```
124     virtual ~MotionTransition();
125
126   public:
127     // 情報取得
128     const MotionInfo *  GetPrevMotion() const { return  prev_motion; }
129     const MotionInfo *  GetNextMotion() const { return  next_motion; }
130     const Matrix4f &  GetPrevMotionMatrix() const { return  prev_motion_mat; }
131     const Matrix4f &  GetNextMotionMatrix() const { return  next_motion_mat; }
132     int  GetBaseSegmentNo() const { return  base_segment_no; }
133     float  GetPrevBeginTime() const { return  prev_begin_time; }
134     float  GetLocalNextBeginTime() const { return  next_begin_time; }
135     float  GetLocalBlendBeginTime() const { return  blend_begin_time; }
136     float  GetLocalBlendEndTime() const { return  blend_end_time; }
137     float  GetNextBeginTime() const { return  next_begin_time + prev_begin_time; }
138     float  GetBlendBeginTime() const { return  blend_begin_time + prev_begin_time; }
139     float  GetBlendEndTime() const { return  blend_end_time + prev_begin_time; }
140     const Posture *  GetLastPrevPose() const { return  prev_motion_posture; }
141     const Posture *  GetLastNextPose() const { return  next_motion_posture; }
142     MotionTransitionState  GetLastState() const { return  last_state; }
143     float  GetLastBlendRatio() const { return  last_blend_ratio; }
144
145   public:
146     // 動作接続・遷移
147
148     // 動作接続・遷移の初期化
149     virtual bool  Init(
150       const MotionInfo * prev_motion, const MotionInfo * next_motion, const Matrix4f &
            prev_motion_mat, float prev_begin_time );
151
152     // 動作接続・遷移の姿勢計算
153     virtual MotionTransitionState  GetPosture( float time, Posture * posture );
154 };
155
156
157 //
158 //  動作接続クラス（動作接続・遷移クラスの簡略版）
159 //
160 class  MotionConnection : public MotionTransition
161 {
162   public:
163     // コンストラクタ
164     MotionConnection() : MotionTransition() {}
165
166     // 動作接続の初期化
167     virtual bool  Init(
168       const MotionInfo * prev_motion, const MotionInfo * next_motion, const Matrix4f &
            prev_motion_mat, float prev_begin_time );
169
170     // 動作接続の姿勢計算
171     virtual MotionTransitionState  GetPosture( float time, Posture * posture );
172 };
173
174
175 // 補助処理（グローバル関数）のプロトタイプ宣言
176
177 // 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
178 //（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）
179 void  ComputeConnectionTransformation( const Matrix4f & prev_frame, const Matrix4f &
        next_frame, Matrix4f & trans_mat );
180
181 // 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
182 // ２つの姿勢の腰の水平位置・水平向きを合わせる
183 //（next_pose の腰の水平位置・水平向きを、prev_pose の腰の水平位置・水平向きに合わせる
```

```
    )
184 void  ComputeConnectionTransformation( const Posture & prev_pose, const Posture &
        next_pose, Matrix4f & trans_mat );
185
186 // 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
187 // ２つの姿勢の基準部位の水平位置と、入力として与えられた水平向きを合わせる
188 //（next_pose の基準部位 base_segment の水平位置を prev_pose の水平位置に合わせて、水平
        向き next_ori を 水平向き prev_ori に合わせる）
189 void  ComputeConnectionTransformation( const Posture & prev_pose, float prev_ori, const
         Posture & next_pose, float next_ori, int base_segment, Matrix4f & trans_mat );
190
191
192
193 #endif // _MOTION_TRANSITION_H_
```

ソースコード 35: 動作接続・遷移クラスの実装（MotionTransition.cpp）

```
 1 /**
 2 ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプ
       ログラム
 3 ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
 4 ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
 5 **/
 6
 7 /**
 8 ***  動作接続・遷移アプリケーション
 9 **/
10
11
12 // ライブラリ・クラス定義の読み込み
13 #include "SimpleHuman.h"
14 #include "MotionTransition.h"
15
16 // 標準算術関数・定数の定義
17 #define  _USE_MATH_DEFINES
18 #include <math.h>
19
20
21 // 補助処理（グローバル関数）のプロトタイプ宣言
22
23 // 姿勢補間（２つの姿勢を補間）（※レポート課題）
24 void  MyPostureInterpolation( const Posture & p0, const Posture & p1, float ratio,
       Posture & p );
25
26
27
28 //
29 //  動作のメタ情報を初期化
30 //
31 void  InitMotionInfo( MotionInfo * info, Motion * m )
32 {
33     info->motion = m;
34     info->begin_time = 0.0f;
35     info->end_time = info->motion ? info->motion->GetDuration() : 0.0f;
36     info->blend_end_time = info->begin_time;
37     info->blend_begin_time = info->end_time;
38     info->base_segment_no = -1;
39     info->enable_ori = false;
40     info->begin_ori = 0.0f;
41     info->end_ori = 0.0f;
42     info->color.set( 1.0f, 1.0f, 1.0f );
43 }
44
45
```

```
46
47 //
48 //  動作接続・遷移クラス
49 //
50
51
52 // コンストラクタ
53 MotionTransition::MotionTransition()
54 {
55     prev_motion = NULL;
56     next_motion = NULL;
57     prev_motion_posture = NULL;
58     next_motion_posture = NULL;
59     prev_motion_mat.setIdentity();
60     next_motion_mat.setIdentity();
61     last_state = MT_NONE;
62 }
63
64
65 // デストラクタ
66 MotionTransition::~MotionTransition()
67 {
68     if ( prev_motion_posture )
69         delete  prev_motion_posture;
70     if ( next_motion_posture )
71         delete  next_motion_posture;
72 }
73
74
75 //
76 //  動作接続・遷移の初期化
77 //
78 bool  MotionTransition::Init(
79     const MotionInfo * prev_motion, const MotionInfo * next_motion, const Matrix4f &
        prev_motion_mat, float prev_begin_time )
80 {
81     // 入力チェック（骨格モデルが異なる動作間の動作接続・遷移には対応しない）
82     if ( !prev_motion || !next_motion || ( prev_motion->motion->body != next_motion->
        motion->body ) )
83         return  false;
84
85     // 動作接続・遷移の入力情報を設定
86     this->prev_motion = prev_motion;
87     this->next_motion = next_motion;
88     this->prev_motion_mat = prev_motion_mat;
89     this->prev_begin_time = prev_begin_time;
90
91     // 動作接続の基準部位を、前の動作の情報にもとづいて設定
92     base_segment_no = prev_motion->base_segment_no;
93
94     // 前後の動作から姿勢を取得するための変数を初期化
95     if ( prev_motion_posture )
96         delete  prev_motion_posture;
97     if ( next_motion_posture )
98         delete  next_motion_posture;
99     prev_motion_posture = new Posture( prev_motion->motion->body );
100    next_motion_posture = new Posture( next_motion->motion->body );
101
102    // 動作遷移のタイミングの計算
103    // 前の動作の blend_begin_time と後の動作の begin_time の時刻を合わせる
104
105    // ※ レポート課題
106
107    // 仮に動作接続の場合の時刻を設定（レポート課題作成前の動作確認用）
```

```
108     next_begin_time = prev_motion->end_time - prev_motion->begin_time;
109     blend_begin_time = next_begin_time;
110     blend_end_time = next_begin_time;
111
112     // 次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）
113 //  next_begin_time = ???;
114
115     // 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）
116 //  blend_begin_time = ???;
117
118     // 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）
119 //  blend_end_time = ???;
120
121     // 前後の動作の動作接続を行う時刻（各動作のローカル時刻）を取得
122     float   prev_local_time = prev_motion->blend_begin_time;
123     float   next_local_time = next_motion->begin_time;
124
125     // 現在の動作の終了姿勢（ワールド座標系）を取得
126     prev_motion->motion->GetPosture( prev_local_time, *prev_motion_posture );
127     TransformPosture( prev_motion_mat, *prev_motion_posture );
128
129     // 次の動作の開始姿勢（次の動作のワールド座標系）を取得
130     next_motion->motion->GetPosture( next_local_time, *next_motion_posture );
131
132     // 現在の動作の終了時の水平向き（ワールド座標系）、次の動作の開始時の水平向き（動作データのワールド座標系）
133     float   prev_ori = 0.0f;
134     float   next_ori = 0.0f;
135
136     // 現在の動作データに向きの情報が設定されていれば、現在の変換行列と動作データの情報から終了時の水平向きを計算
137     if ( prev_motion->enable_ori )
138     {
139         Matrix3f  prev_ori_mat;
140         prev_motion_mat.get( &prev_ori_mat );
141         prev_ori = ComputeOrientationAngle( prev_ori_mat );
142         prev_ori += prev_motion->end_ori;
143         if ( prev_ori > 180.0f )
144             prev_ori -= 360.0f;
145         if ( prev_ori < -180.0f)
146             prev_ori += 360.0f;
147     }
148     // 前の動作の終了姿勢から水平向きを計算
149     else
150         prev_ori = ComputeOrientationAngle( prev_motion_posture->root_ori );
151
152     // 次の動作データに向きの情報が設定されていれば、開始時の水平向きを取得
153     if ( next_motion->enable_ori )
154         next_ori = next_motion->begin_ori;
155     // 次の動作の開始姿勢から水平向きを計算
156     else
157         next_ori = ComputeOrientationAngle( next_motion_posture->root_ori );
158
159     // 現在の動作の終了姿勢と次の動作の開始姿勢の位置・向きを合わせるための変換行列を計算
160     //（next_pose の基準部位 base_segment の水平位置を prev_pose の水平位置に合わせて、水平向き next_ori を 水平向き prev_ori に合わせる）
161     ComputeConnectionTransformation(
162         *prev_motion_posture, prev_ori, *next_motion_posture, next_ori, base_segment_no,
163         next_motion_mat );
164
```

```
165     return  true;
166 }
167
168
169 //
170 //  動作接続・遷移の姿勢計算
171 //
172 MotionTransition::MotionTransitionState  MotionTransition::GetPosture(
173     float time, Posture * posture )
174 {
175     // 動作接続・遷移の初期化のチェック
176     if ( !prev_motion || !next_motion )
177         return  MT_NONE;
178
179     // 前後の動作の姿勢補間の重み（ブレンド比率）
180     float  blend_ratio = 0.0f;
181
182     // 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）
183     float  local_time = 0.0f;
184
185     // 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）
186     float  next_motion_local_time = 0.0f;
187
188     // 前の動作の開始時刻を基準とするローカル時刻を計算
189     local_time = time - prev_begin_time;
190
191     // 現在の状態を判定
192     MotionTransitionState  state = MT_PREV_MOTION;
193     if ( local_time > blend_end_time )
194         state = MT_NEXT_MOTION;
195     else if ( local_time > blend_begin_time )
196         state = MT_IN_TRANSITION;
197
198     // 前の動作の姿勢を取得
199     if ( state == MT_PREV_MOTION || state == MT_IN_TRANSITION )
200     {
201         // 前の動作の姿勢を取得
202         prev_motion->motion->GetPosture( local_time + prev_motion->begin_time, *
                prev_motion_posture );
203
204         // 前の動作の姿勢の位置・向きに変換行列を適用
205         TransformPosture( prev_motion_mat, *prev_motion_posture );
206     }
207
208     // 後の動作の姿勢を取得
209     if ( state == MT_NEXT_MOTION || state == MT_IN_TRANSITION )
210     {
211         // ※ レポート課題
212         //
213         // 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）
214 //      next_motion_local_time = ???;
215
216         // 後の動作から現在時刻の姿勢を取得
217         next_motion->motion->GetPosture( next_motion_local_time + next_motion->begin_time
                , *next_motion_posture );
218
219         // 後の動作の姿勢の位置・向きに変換行列を適用
220         TransformPosture( next_motion_mat, *next_motion_posture );
221     }
222
223     // 動作遷移前であれば、前の動作の姿勢を出力
224     if ( state == MT_PREV_MOTION )
225     {
226         blend_ratio = 0.0f;
```

```
227         *posture = *prev_motion_posture;
228     }
229     // 動作遷移後であれば、後の動作の姿勢を出力
230     else if ( state == MT_NEXT_MOTION )
231     {
232         blend_ratio = 1.0f;
233         *posture = *next_motion_posture;
234     }
235     // 動作遷移中であれば、前後の動作の姿勢を補間
236     else
237     {
238         // ※ レポート課題
239 
240         // ブレンド比率（補間の重み）を計算
241 //      blend_ratio = ???;
242 
243         // ブレンド比率（補間の重み）が範囲外の値になった場合への対応
244         if ( blend_ratio > 1.0f )
245             blend_ratio = 1.0f;
246         if ( blend_ratio < 0.0f )
247             blend_ratio = 0.0f;
248 
249         // 前後の動作の姿勢を補間
250         MyPostureInterpolation( *prev_motion_posture, *next_motion_posture, blend_ratio,
                *posture );
251     }
252 
253     // 最後の姿勢計算時の動作遷移の状態と姿勢補間の重みを記録
254     last_state = state;
255     last_blend_ratio = blend_ratio;
256 
257     // 動作遷移の状態を返す
258     return  state;
259 }
260 
261 
262 
263 //
264 //  動作接続クラス
265 //
266 
267 
268 //
269 //  動作接続の初期化
270 //
271 bool  MotionConnection::Init(
272     const MotionInfo * prev_motion, const MotionInfo * next_motion, const Matrix4f &
            prev_motion_mat, float prev_begin_time )
273 {
274     // 入力チェック（骨格モデルが異なる動作間の動作接続・遷移には対応しない）
275     if ( !prev_motion || !next_motion || ( prev_motion->motion->body != next_motion->
            motion->body ) )
276         return  false;
277 
278     // 動作接続・遷移の入力情報を設定
279     this->prev_motion = prev_motion;
280     this->next_motion = next_motion;
281     this->prev_motion_mat = prev_motion_mat;
282     this->prev_begin_time = prev_begin_time;
283 
284     // 動作接続の基準部位を、前の動作の情報にもとづいて設定
285     base_segment_no = prev_motion->base_segment_no;
286 
287     // 前後の動作から姿勢を取得するための変数を初期化
```

```
288     if ( prev_motion_posture )
289        delete  prev_motion_posture;
290     if ( next_motion_posture )
291        delete  next_motion_posture;
292     prev_motion_posture = new Posture( prev_motion->motion->body );
293     next_motion_posture = new Posture( next_motion->motion->body );
294
295     // 動作遷移のタイミングの計算
296     // 前の動作の end_time と後の動作の begin_time の時刻を合わせる
297     // 動作遷移は行わないため、動作遷移の区間の長さは 0 にする
298
299     // 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）
300     next_begin_time = prev_motion->end_time - prev_motion->begin_time;
301
302     // 動作遷移のための動作ブレンドの開始・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）
303     blend_begin_time = next_begin_time;
304     blend_end_time = next_begin_time;
305
306     // 前後の動作の動作接続を行う時刻（各動作のローカル時刻）を決定
307     float  prev_local_time = prev_motion->end_time;
308     float  next_local_time = next_motion->begin_time;
309
310     // 現在の動作の終了姿勢（ワールド座標系）を取得
311     prev_motion->motion->GetPosture( prev_local_time, *prev_motion_posture );
312     TransformPosture( prev_motion_mat, *prev_motion_posture );
313
314     // 次の動作の開始姿勢（次の動作のワールド座標系）を取得
315     next_motion->motion->GetPosture( next_local_time, *next_motion_posture );
316
317     // 現在の動作の終了姿勢と次の動作の開始姿勢の姿勢の位置・向きを合わせるための変換行
          列を計算
318     //（curr_posture の位置・向きを、next_posture の位置・向きに合わせるための変換行列
          trans_mat を計算）
319     ComputeConnectionTransformation(
320        *prev_motion_posture, 0.0f, *next_motion_posture, 180.0f, base_segment_no,
321        next_motion_mat );
322
323     return  true;}
324
325
326 //
327 //  動作接続の姿勢計算
328 //
329 MotionTransition::MotionTransitionState  MotionConnection::GetPosture( float time,
        Posture * posture )
330 {
331     // 動作接続・遷移の初期化のチェック
332     if ( !prev_motion || !next_motion )
333        return  MT_NONE;
334
335     // 前後の動作の姿勢補間の重み（ブレンド比率）
336     float  blend_ratio = 0.0f;
337
338     // 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）
339     float  local_time = 0.0f;
340
341     // 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）
342     float  next_motion_local_time = 0.0f;
343
344     // 前の動作の開始時刻を基準とするローカル時刻を計算
345     local_time = time - prev_begin_time;
346
```

```
347     // 現在の状態を判定
348     MotionTransitionState  state = MT_PREV_MOTION;
349     if ( local_time > next_begin_time )
350         state = MT_NEXT_MOTION;
351 
352     // 前の動作の姿勢を出力
353     if ( state == MT_PREV_MOTION )
354     {
355         // 前の動作の姿勢を取得
356         prev_motion->motion->GetPosture( local_time + prev_motion->begin_time, *posture
                );
357 
358         // 前の動作の姿勢の位置・向きに変換行列を適用
359         TransformPosture( prev_motion_mat, *posture );
360 
361         // 姿勢補間の重みを設定
362         blend_ratio = 0.0f;
363     }
364 
365     // 後の動作の姿勢を出力
366     else
367     {
368         // 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）
369         next_motion_local_time = local_time - next_begin_time;
370 
371         // 後の動作から現在時刻の姿勢を取得
372         next_motion->motion->GetPosture( next_motion_local_time + next_motion->begin_time
                , *posture );
373 
374         // 後の動作の姿勢の位置・向きに変換行列を適用
375         TransformPosture( next_motion_mat, *posture );
376 
377         // 姿勢補間の重みを設定
378         blend_ratio = 1.0f;
379     }
380 
381     // 最後の姿勢計算時の動作遷移の状態と姿勢補間の重みを記録
382     last_state = state;
383     last_blend_ratio = blend_ratio;
384 
385     // 動作遷移の状態を返す
386     return  state;
387 }
388 
389 
390 
391 //
392 //  補助処理（グローバル関数）
393 //
394 
395 
396 //
397 //  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
398 //  (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)
399 //
400 void  ComputeConnectionTransformation( const Matrix4f & prev_frame, const Matrix4f &
        next_frame, Matrix4f & trans_mat )
401 {
402     // ※ レポート課題
403 
404     // 方法1
405     // trans_mat = ???
406 
```

```
407     // 方法2（水平方向の向き・位置のみを変換）
408     Matrix3f  ori;
409     float  angle;
410     Vector3f  pos;
411     Matrix4f  prev_frame2, next_frame2;
412 
413     // 変換行列から水平方向の回転のみを抽出して再設定
414     // prev_frame2 = ???
415 
416     // 変換行列から水平方向の回転のみを抽出して再設定
417     // next_frame2 = ???
418 
419     // 座標変換を計算
420     // trans_mat = ???
421 }
422 
423 
424 //
425 //  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
426 //  2つの姿勢の腰の水平位置・水平向きを合わせる
427 //（next_pose の腰の水平位置・水平向きを、prev_pose の腰の水平位置・水平向きに合わせる
    ）
428 //
429 void  ComputeConnectionTransformation( const Posture & prev_pose, const Posture &
    next_pose, Matrix4f & trans_mat )
430 {
431     // 前の姿勢の腰の位置・向きを取得
432     Matrix4f  curr_end_frame;
433     curr_end_frame.set( prev_pose.root_ori, prev_pose.root_pos, 1.0f );
434 
435     // 次の姿勢の腰の位置・向きを取得
436     Matrix4f  next_begin_frame;
437     next_begin_frame.set( next_pose.root_ori, next_pose.root_pos, 1.0f );
438 
439     // 次の姿勢の腰の位置・向きを前の姿勢の腰の位置・向きに合わせるための変換行列を計算
440     ComputeConnectionTransformation( curr_end_frame, next_begin_frame, trans_mat );
441 }
442 
443 
444 //
445 //  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
446 //  2つの姿勢の任意の体節の水平位置と、入力として与えられた水平向きを合わせる
447 //（next_pose の基準部位 base_segment の水平位置を prev_pose の水平位置に合わせて、水
    平向き next_ori を 水平向き prev_ori に合わせる）
448 //
449 void  ComputeConnectionTransformation( const Posture & prev_pose, float prev_ori,
450     const Posture & next_pose, float next_ori, int base_segment, Matrix4f & trans_mat )
451 {
452 /*
453     Vector3f  prev_pos, next_pos;
454     Matrix3f  prev_rot, next_rot;
455     Matrix4f  prev_frame, next_frame;
456 
457     // 基準部位としてルート体節が指定された場合は、2つの姿勢のルートの位置を取得
458     if ( base_segment == 0 )
459     {
460         // ※レポート課題
461 //      prev_pos = ???;
462 //      next_pos = ???;
463     }
464     // 基準部位としてルート体節以外が指定された場合は、順運動学計算により、2つの姿勢の
        指定部位の位置を計算
465     else
466     {
```

```
467         // ※レポート課題
468 //      prev_pos = ???;
469 //      next_pos = ???;
470     }
471
472     // 前の姿勢の位置・向きを表す変換行列を計算
473     // ※レポート課題
474 // prev_rot = ???;
475     prev_frame.set( prev_rot, prev_pos, 1.0f );
476
477     // 次の姿勢の位置・向きを表す変換行列を計算
478     // ※レポート課題
479 // next_rot = ???;
480     next_frame.set( next_rot, next_pos, 1.0f );
481
482     // 座標変換を計算
483     // ※レポート課題
484 // trans_mat = ???;
485 */
486
487     // 上記の処理が未実装であれば、動作データの水平向きや基準部位の情報を使用しない、標準的な処理を呼び出す
488     ComputeConnectionTransformation( prev_pose, next_pose, trans_mat );
489 }
```

ソースコード 36: 動作接続・遷移アプリケーションの定義（MotionTransitionApp.h）

```
1  /**
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプログラム
3  ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7  /**
8  ***  動作接続・遷移アプリケーション
9  **/
10
11 #ifndef  _MOTION_TRANSITION_APP_H_
12 #define  _MOTION_TRANSITION_APP_H_
13
14
15 // ライブラリ・クラス定義の読み込み
16 #include "SimpleHuman.h"
17 #include "SimpleHumanGLUT.h"
18
19
20 // プロトタイプ宣言
21 struct  MotionInfo;
22 class  MotionTransition;
23 class  Timeline;
24
25
26 //
27 //  動作接続・遷移アプリケーションクラス
28 //
29 class  MotionTransitionApp : public GLUTBaseApp
30 {
31   protected:
32     // 動作遷移・接続（を含む動作再生）の入力情報
33
34     // 動作データリスト
35     vector< MotionInfo * >  motion_list;
36
```

```
37      // 現在の再生動作番号
38      int  curr_motion_no;
39
40      // 現在の動作の位置・向きに対する変換行列
41      Matrix4f  curr_motion_mat;
42
43      // 現在の動作の再生開始時刻（グローバル時刻）
44      float  curr_start_time;
45
46      // 次の再生動作番号
47      int  next_motion_no;
48
49      // 実行待ちの動作番号
50      int  waiting_motion_no;
51
52    protected:
53      // 動作遷移のための変数
54
55      // 動作接続・遷移
56      MotionTransition *  transition;
57
58      // 動作遷移（前後の動作のブレンディング）を適用するかどうかの設定
59      bool  enable_transition;
60
61    protected:
62      // 動作再生のための変数
63
64      // キャラクタの姿勢
65      Posture *  curr_posture;
66
67      // アニメーション中かどうかを表すフラグ
68      bool  on_animation;
69
70      // アニメーションの再生時間
71      float  animation_time;
72
73      // アニメーションの再生速度
74      float  animation_speed;
75
76    protected:
77      // 情報表示用の変数
78
79      // 現在姿勢の描画色
80      Color3f  figure_color;
81
82      // タイムライン描画機能
83      Timeline *  timeline;
84
85      // 動作遷移回数のカウント（タイムライン表示用）
86      int  transition_count;
87
88
89    public:
90      // コンストラクタ
91      MotionTransitionApp();
92
93      // デストラクタ
94      virtual ~MotionTransitionApp();
95
96    public:
97      // イベント処理
98
99      //  初期化
100     virtual void  Initialize();
```

```
101
102     //  開始・リセット
103     virtual void  Start();
104
105     //  画面描画
106     virtual void  Display();
107
108     // ウィンドウサイズ変更
109     virtual void  Reshape( int w, int h );
110
111     // マウスクリック
112     virtual void  MouseClick( int button, int state, int mx, int my );
113
114     // キーボードのキー押下
115     virtual void  Keyboard( unsigned char key, int mx, int my );
116
117     // キーボードの特殊キー押下
118     virtual void  KeyboardSpecial( unsigned char key, int mx, int my );
119
120     // アニメーション処理
121     virtual void  Animation( float delta );
122
123   public:
124     // 補助処理
125
126     //  次の動作を変更
127     void  SetNextMotion( int no = -1 );
128
129     // 動作再生処理（動作接続・遷移を考慮）
130     void  AnimationWithMotionTransition( float delta );
131
132     // タイムラインへの前後の動作・遷移区間の設定
133     static void  InitTimeline( Timeline & timeline, const MotionTransition & trans, int
            count );
134
135     // タイムラインへの現在時刻・時間範囲の設定
136     static void  UpdateTimeline( Timeline & timeline, float curr_time );
137 };
138
139
140 // 補助処理（グローバル関数）のプロトタイプ宣言
141
142 // サンプル動作セットの読み込み
143 const Skeleton *  LoadSampleMotions( vector< MotionInfo * > &  motion_list, const
        Skeleton * body = NULL );
144
145
146 #endif // _MOTION_TRANSITION_APP_H_
```

ソースコード 37: 動作接続・遷移アプリケーションの実装（MotionTransitionApp.cpp）

```
1  /**
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプ
       ログラム
3  ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7  /**
8  ***  動作接続・遷移アプリケーション
9  **/
10
11
12 // ライブラリ・クラス定義の読み込み
```

```
13 #include "SimpleHuman.h"
14 #include "MotionTransition.h"
15 #include "MotionTransitionApp.h"
16 #include "BVH.h"
17 #include "Timeline.h"
18
19 // 標準算術関数・定数の定義
20 #define _USE_MATH_DEFINES
21 #include <math.h>
22
23
24
25 //
26 //  サンプル動作セットの読み込み
27 //
28 const Skeleton * LoadSampleMotions( vector< MotionInfo * > & motion_list, const
       Skeleton * body )
29 {
30     // アプリケーションのテストに使用する動作データの定義（
          BVHファイル名・キー時刻・描画色）
31     // 各歩行動作の、右足が地面から離れる、右足が着く、左足が離れる、左足が着く、右足が
          離れるの５つのキー時刻を設定
32     // 先頭のキー区間を開始時のブレンド区間とし、末尾のキー区間を終了時のブレンド区間と
          する
33     const int  num_motions = 3;
34     const int  num_keytimes = 5;
35     const char *  sample_motions[ num_motions ] = {
36         "sample_walking1.bvh",
37         "sample_walking2.bvh",
38         "sample_walking3.bvh" };
39     const float  sample_keytimes[ num_motions ][ num_keytimes ] = {
40         { 2.35f, 3.00f, 3.08f, 3.68f, 3.74f },
41         { 1.30f, 2.07f, 2.12f, 2.88f, 2.94f },
42         { 1.20f, 2.00f, 2.08f, 2.80f, 2.86f }
43     };
44     const char *  sample_base_segments[ num_motions ] = {
45         "LeftAnkle",
46         "LeftAnkle",
47         "LeftAnkle"
48     };
49     const float  sample_orientations[ num_motions ][ 2 ] = {
50         { 180.0f, 180.0f },
51         { 180.0f, 180.0f },
52         { 180.0f, 180.0f }
53     };
54     const Color3f  sample_colors[] = {
55         Color3f( 0.5f, 1.0f, 0.5f ),
56         Color3f( 0.5f, 0.5f, 1.0f ),
57         Color3f( 1.0f, 0.5f, 0.5f )
58     };
59
60     // 動作のメタ情報
61     MotionInfo *  info = NULL;
62
63     // 動作データの読み込み、動作情報の設定
64     for ( int i=0; i<num_motions; i++ )
65     {
66         // BVHファイルを読み込んで動作データ（＋骨格モデル）を生成
67         Motion *  new_motion = LoadAndCoustructBVHMotion( sample_motions[ i ], body );
68
69         // 動作データの読み込みに失敗したらスキップ
70         if ( !new_motion )
71             continue;
72
```

```
73          // 最初に読み込んだ動作データの骨格モデルを記録（次以降の動作データの骨格モデルとして使用）
74          if ( !body )
75             body = new_motion->body;
76
77          // 動作のメタ情報の設定
78          info = new MotionInfo();
79          InitMotionInfo( info, new_motion );
80          for ( int j = 0; j < num_keytimes; j++ )
81             info->keytimes.push_back( sample_keytimes[ i ][ j ] );
82          info->begin_time = info->keytimes[ 0 ];
83          info->blend_end_time = info->keytimes[ 1 ];
84          info->blend_begin_time = info->keytimes[ info->keytimes.size() - 2 ];
85          info->end_time = info->keytimes[ info->keytimes.size() - 1 ];
86          info->base_segment_no = FindSegment( new_motion->body, sample_base_segments[ i ] );
87          info->enable_ori = true;
88          info->begin_ori = sample_orientations[ i ][ 0 ];
89          info->end_ori = sample_orientations[ i ][ 1 ];
90          info->color = sample_colors[ i ];
91
92          // 動作リストに追加
93          motion_list.push_back( info );
94       }
95
96       return  body;
97    }
98
99
100
101   //
102   //  コンストラクタ
103   //
104   MotionTransitionApp::MotionTransitionApp()
105   {
106      app_name = "Motion Transition";
107
108      transition = NULL;
109      enable_transition = true;
110
111      curr_posture = NULL;
112      on_animation = true;
113      animation_time = 0.0f;
114      animation_speed = 1.0f;
115
116      figure_color.set( 1.0f, 1.0f, 1.0f );
117      timeline = NULL;
118   }
119
120
121   //
122   //  デストラクタ
123   //
124   MotionTransitionApp::~MotionTransitionApp()
125   {
126      for ( int i=0; motion_list.size(); i++ )
127      {
128         delete  motion_list[ i ]->motion;
129         delete  motion_list[ i ];
130      }
131      motion_list.clear();
132
133      if ( transition )
134         delete  transition;
```

```
135 |
136 |     if ( curr_posture->body )
137 |         delete  curr_posture->body;
138 |     if ( curr_posture )
139 |         delete  curr_posture;
140 |
141 |     if ( timeline )
142 |         delete  timeline;
143 | }
144 |
145 |
146 | //
147 | //  初期化
148 | //
149 | void  MotionTransitionApp::Initialize()
150 | {
151 |     GLUTBaseApp::Initialize();
152 |
153 |     // サンプル動作セットの骨格モデル
154 |     const Skeleton *  body = NULL;
155 |
156 |     // サンプル動作セットの読み込み
157 |     if ( motion_list.size() == 0 )
158 |         body = LoadSampleMotions( motion_list );
159 |
160 |     // 姿勢の初期化
161 |     if ( body )
162 |     {
163 |         if ( curr_posture )
164 |             delete  curr_posture;
165 |         curr_posture = new Posture( body );
166 |         InitPosture( *curr_posture, body );
167 |     }
168 |
169 |     // タイムライン描画機能の初期化
170 |     timeline = new Timeline();
171 | }
172 |
173 |
174 | //
175 | //  開始・リセット
176 | //
177 | void  MotionTransitionApp::Start()
178 | {
179 |     GLUTBaseApp::Start();
180 |
181 |     // 現在の動作を初期化（動作リストの先頭の動作を現在の動作とする）
182 |     curr_motion_no = 0;
183 |     const MotionInfo *  curr_motion_info = NULL;
184 |     if ( motion_list.size() > 0 )
185 |         curr_motion_info = motion_list[ curr_motion_no ];
186 |     else
187 |         curr_motion_no = -1;
188 |
189 |     // 次の動作・待ち動作を初期化
190 |     next_motion_no = -1;
191 |     waiting_motion_no = 0;
192 |
193 |     // 動作接続・遷移機能の生成
194 |     if ( transition )
195 |         delete  transition;
196 |     if ( enable_transition )
197 |         transition = new MotionTransition();
198 |     else
```

```
199         transition = new MotionConnection();
200
201     // アニメーション再生の初期化
202     animation_time = 0.0f;
203     if ( curr_motion_info )
204         figure_color = curr_motion_info->color;
205     transition_count = 0;
206
207     // 現在の動作の開始位置・向きと開始時刻を設定
208     Point3f   init_pos( 0.0f, 0.0f, 0.0f );
209     Matrix3f  init_ori;
210     init_ori.rotY( 180.0f * M_PI / 180.0f );
211     curr_motion_mat.set( init_ori, init_pos, 1.0f );
212     curr_start_time = animation_time;
213
214     // アニメーション処理（開始時の姿勢の取得）
215     Animation( 0.0f );
216 }
217
218
219 //
220 //  画面描画
221 //
222 void  MotionTransitionApp::Display()
223 {
224     GLUTBaseApp::Display();
225
226     // キャラクタを描画
227     if ( curr_posture )
228     {
229         glColor3f( figure_color.x, figure_color.y, figure_color.z );
230         DrawPosture( *curr_posture );
231         DrawPostureShadow( *curr_posture, shadow_dir, shadow_color );
232     }
233
234     // タイムラインを描画
235     if ( timeline )
236         timeline->DrawTimeline();
237
238     // 現在のモード、現在・次の再生動作、アニメーション再生時間、動作接続・遷移の設定を
        表示
239     char  message[ 64 ];
240     DrawTextInformation( 0, "Motion Transition" );
241     if ( curr_motion_no != -1 )
242     {
243         if ( ( next_motion_no != -1 ) && ( next_motion_no != curr_motion_no ) )
244             sprintf( message, "%s -> %s", motion_list[ curr_motion_no ]->motion->name.
                c_str(), motion_list[ next_motion_no ]->motion->name.c_str() );
245         else
246             sprintf( message, "%s", motion_list[ curr_motion_no ]->motion->name.c_str() );
247         DrawTextInformation( 1, message );
248     }
249     sprintf( message, "%.2f", animation_time );
250     DrawTextInformation( 2, message );
251     if ( !enable_transition )
252         DrawTextInformation( 3, "Transition: Off" );
253 }
254
255
256 //
257 //  ウィンドウサイズ変更
258 //
259 void  MotionTransitionApp::Reshape( int w, int h )
260 {
```

```
261     GLUTBaseApp::Reshape( w, h );
262
263     // タイムラインの描画領域の設定（画面の下部に配置）
264     if ( timeline )
265         timeline->SetViewAreaBottom( 0, 0, 0, 3, 32, 2 );
266 }
267
268
269 //
270 //  マウスクリック
271 //
272 void  MotionTransitionApp::MouseClick( int button, int state, int mx, int my )
273 {
274     GLUTBaseApp::MouseClick( button, state, mx, my );
275
276     // 左ボタンが押されたら、次の再生動作を変更
277     if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
278     {
279         int  motion_no = ( curr_motion_no + 1 ) % 3;
280         SetNextMotion( motion_no );
281     }
282 }
283
284
285 //
286 //  キーボードのキー押下
287 //
288 void  MotionTransitionApp::Keyboard( unsigned char key, int mx, int my )
289 {
290     GLUTBaseApp::Keyboard( key, mx, my );
291
292     // 数字キーで次に実行する動作を指定
293     if ( ( key >= '1' ) && ( key <= '9' ) )
294     {
295         int  no = key - '1';
296         SetNextMotion( no );
297     }
298
299     // s キーでアニメーションの停止・再開
300     if ( key == 's' )
301         on_animation = !on_animation;
302
303     // w キーでアニメーションの再生速度を変更
304     if ( key == 'w' )
305         animation_speed = ( animation_speed == 1.0f ) ? 0.1f : 1.0f;
306
307     // n キーで次のフレーム
308     if ( ( key == 'n' ) && !on_animation && transition && transition->GetPrevMotion() )
309     {
310         on_animation = true;
311         Animation( transition->GetPrevMotion()->motion->interval );
312         on_animation = false;
313     }
314
315     // d キーで動作接続・遷移の設定を変更
316     if ( key == 'd' )
317     {
318         enable_transition = !enable_transition;
319         Start();
320     }
321 }
322
323
324 //
```

```
325 //  キーボードの特殊キー押下
326 //
327 void  MotionTransitionApp::KeyboardSpecial( unsigned char key, int mx, int my )
328 {
329     // カーソル上キーが押されたら、次の再生動作を変更
330     if ( key == GLUT_KEY_UP )
331     {
332         SetNextMotion();
333     }
334 }
335
336
337 //
338 //  アニメーション処理
339 //
340 void  MotionTransitionApp::Animation( float delta )
341 {
342     // アニメーション再生中でなければ終了
343     if ( !on_animation )
344         return;
345
346     //  動作再生処理（動作接続・遷移を考慮）
347     AnimationWithMotionTransition( delta );
348
349     // 注視点を更新
350     view_center.set( curr_posture->root_pos.x, 0.0f, curr_posture->root_pos.z );
351 }
352
353
354 //
355 //  次の動作を変更
356 //
357 void  MotionTransitionApp::SetNextMotion( int no )
358 {
359     if ( motion_list.size() == 0 )
360         return;
361
362     // 次の動作が指定されなかった場合は、現在の再生動作の次の番号の動作を次の動作とする
363     if ( no < 0 )
364         no = ( curr_motion_no + 1 ) % motion_list.size();
365     else
366         no = no % motion_list.size();
367
368     // 再生待ちの動作に設定
369     waiting_motion_no = no;
370 }
371
372
373 //
374 //  動作再生処理（動作接続・遷移を考慮）
375 //
376 void  MotionTransitionApp::AnimationWithMotionTransition( float delta )
377 {
378     // 現在の動作の情報を取得
379     MotionInfo *  curr_motion_info = NULL;
380     if ( curr_motion_no!= -1 )
381         curr_motion_info = motion_list[ curr_motion_no ];
382
383     // 次の動作の情報を取得
384     MotionInfo *  next_motion_info = NULL;
385     if ( next_motion_no != -1 )
386         next_motion_info = motion_list[ next_motion_no ];
387
388     // 次の動作が未設定であれば、実行待ち動作を次の動作とする
```

```
389     else if ( waiting_motion_no != -1 )
390     {
391         // 実行待ちの動作の情報を取得
392         next_motion_no = waiting_motion_no;
393         next_motion_info = motion_list[ next_motion_no ];
394 
395         // 実行待ちの動作を初期化
396         waiting_motion_no = -1;
397     }
398 
399     // 動作接続・遷移の初期化
400     //（前後の動作や前の動作の開始時刻が変更されたら、初期化を行う）
401     if ( next_motion_info &&
402          ( transition->GetPrevMotion() != curr_motion_info ) ||
403          ( transition->GetNextMotion() != next_motion_info ) ||
404          ( transition->GetPrevBeginTime() != curr_start_time ) )
405     {
406         // 動作接続・遷移の初期化
407         transition->Init( curr_motion_info, next_motion_info, curr_motion_mat,
                curr_start_time );
408 
409         // タイムラインへの前後の動作・遷移区間の設定
410         InitTimeline( *timeline, *transition, transition_count );
411     }
412 
413 
414     // アニメーションの時間を進める
415     animation_time += delta * animation_speed;
416 
417     // 動作接続・遷移の姿勢計算
418     MotionTransition::MotionTransitionState  state = MotionTransition::MT_NONE;
419     state = transition->GetPosture( animation_time, curr_posture );
420     float  blend_ratio = transition->GetLastBlendRatio();
421 
422     // 姿勢の描画色を設定
423     if ( ( blend_ratio > 0.0f ) && curr_motion_info && next_motion_info )
424         figure_color.scaleAdd( blend_ratio, next_motion_info->color - curr_motion_info->
            color, curr_motion_info->color );
425     else if ( curr_motion_info )
426         figure_color = curr_motion_info->color;
427 
428     // タイムラインへの現在時刻・時間範囲の設定
429     UpdateTimeline( *timeline, animation_time );
430 
431 
432     // 動作接続・遷移の動作ブレンドの完了処理
433     if ( state == MotionTransition::MT_NEXT_MOTION )
434     {
435         // 次の動作の変換行列と開始時刻を取得
436         curr_motion_mat = transition->GetNextMotionMatrix();
437         curr_start_time = transition->GetNextBeginTime();
438 
439         // 現在の動作を次の動作に切り替え
440         curr_motion_no = next_motion_no;
441 
442         // 次の動作を初期化
443         next_motion_no = -1;
444 
445         // 実行待ち動作が未設定であれば、現在の動作を繰り返すように設定する
446         if ( waiting_motion_no == -1 )
447             waiting_motion_no = curr_motion_no;
448 
449         // 動作遷移回数のカウントを加算（タイムライン表示用）
450         transition_count ++;
```

```
451     }
452 }
453
454
455 //
456 //  タイムラインへの現在動作・次の動作・遷移区間の設定
457 //
458 void  MotionTransitionApp::InitTimeline( Timeline & timeline, const MotionTransition &
        trans, int transition_count )
459 {
460     Color4f  prev_color, next_color, trans_color;
461
462     const MotionInfo *  prev_motion = trans.GetPrevMotion();
463     const MotionInfo *  next_motion = trans.GetNextMotion();
464
465     // 動作の開始時刻（グローバル時刻）
466     float  prev_begin_time = trans.GetPrevBeginTime();
467     float  next_begin_time = trans.GetNextBeginTime();
468
469     // タイムラインに各動作を表す要素を追加
470     while( timeline.GetNumElements() < 3 )
471         timeline.AddElement( 0.0f, 1.0f, Color4f( 1.0f, 1.0f, 1.0f, 1.0f ), NULL,
                timeline.GetNumElements() );
472
473     // 前の動作・次の動作を配置するトラック番号を決定
474     int  prev_track_no = 0;
475     int  next_track_no = 2;
476     if ( transition_count % 2 == 1 )
477     {
478         prev_track_no = 2;
479         next_track_no = 0;
480     }
481
482     // 前の動作の要素の情報を設定
483     if ( prev_motion )
484     {
485         timeline.SetElementEnable( 0, true );
486         timeline.SetElementTime( 0, prev_begin_time,
487             prev_begin_time + ( prev_motion->end_time - prev_motion->begin_time ) );
488         prev_color.set( prev_motion->color.x, prev_motion->color.y, prev_motion->color.z,
                1.0f );
489         timeline.SetElementColor( 0, prev_color );
490         timeline.SetElementText( 0, prev_motion->motion->name.c_str() );
491         timeline.SetElementTrackNo( 0, prev_track_no );
492     }
493     else
494     {
495         timeline.SetElementEnable( 0, false );
496     }
497
498     // 遷移後の動作の要素の情報を設定
499     if ( next_motion )
500     {
501         timeline.SetElementEnable( 2, true );
502         timeline.SetElementTime( 2, next_begin_time,
503             next_begin_time + ( next_motion->end_time - next_motion->begin_time ) );
504         next_color.set( next_motion->color.x, next_motion->color.y, next_motion->color.z,
                1.0f );
505         timeline.SetElementColor( 2, next_color );
506         timeline.SetElementText( 2, next_motion->motion->name.c_str() );
507         timeline.SetElementTrackNo( 2, next_track_no );
508     }
509     else
510     {
```

```
511         timeline.SetElementEnable( 2, false );
512     }
513
514     // 遷移区間の要素の情報を設定
515     if ( next_motion )
516     {
517         timeline.SetElementEnable( 1, true );
518         timeline.SetElementTime( 1, trans.GetBlendBeginTime(), trans.GetBlendEndTime() );
519         timeline.SetElementColor( 1, prev_color, next_color );
520         timeline.SetElementText( 1, "transition" );
521         timeline.SetElementTrackNo( 1, 1 );
522     }
523     else
524     {
525         timeline.SetElementEnable( 1, false );
526     }
527
528     // 各動作のブレンド区間の表示に必要な区間を初期化
529     int  size = 4;
530     while( timeline.GetNumSubElements() < size )
531         timeline.AddSubElement( 0, 0.0f, 1.0f, 0.0f, 1.0f, Color4f( 1.0f, 1.0f, 1.0f, 1.0
                f ) );
532
533     // 各動作の開始・終了ブレンド区間の要素の情報を設定
534     float  base_time;
535     int  count = 0;
536     if ( prev_motion )
537     {
538         base_time = prev_begin_time - prev_motion->begin_time;
539         prev_color.set( prev_motion->color.x + 0.1f, prev_motion->color.y + 0.1f,
                prev_motion->color.z + 0.1f, 1.0f );
540         timeline.SetSubElementEnable( count, true );
541         timeline.SetSubElementParent( count, 0 );
542         timeline.SetSubElementTime( count, base_time + prev_motion->begin_time, base_time
                + prev_motion->blend_end_time );
543         timeline.SetSubElementColor( count, prev_color );
544         count ++;
545         timeline.SetSubElementEnable( count, true );
546         timeline.SetSubElementParent( count, 0 );
547         timeline.SetSubElementTime( count, base_time + prev_motion->blend_begin_time,
                base_time + prev_motion->end_time );
548         timeline.SetSubElementColor( count, prev_color );
549         count ++;
550     }
551
552     if ( next_motion )
553     {
554         base_time = next_begin_time - next_motion->begin_time;
555         next_color.set( next_motion->color.x + 0.1f, next_motion->color.y + 0.1f,
                next_motion->color.z + 0.1f, 1.0f );
556         timeline.SetSubElementEnable( count, true );
557         timeline.SetSubElementParent( count, 2 );
558         timeline.SetSubElementTime( count, base_time + next_motion->begin_time, base_time
                + next_motion->blend_end_time );
559         timeline.SetSubElementColor( count, next_color );
560         count ++;
561         timeline.SetSubElementEnable( count, true );
562         timeline.SetSubElementParent( count, 2 );
563         timeline.SetSubElementTime( count, base_time + next_motion->blend_begin_time,
                base_time + next_motion->end_time );
564         timeline.SetSubElementColor( count, next_color );
565         count ++;
566     }
567 }
```

```
568
569
570 //
571 //  タイムラインへの現在時刻の設定
572 //
573 void  MotionTransitionApp::UpdateTimeline( Timeline & timeline, float curr_time )
574 {
575     // 現在時刻のラインの情報を設定
576     while( timeline.GetNumLines() < 1 )
577         timeline.AddLine( curr_time, Color4f( 1.0f, 0.0f, 0.0f, 1.0f ) );
578     timeline.SetLineTime( 0, curr_time );
579
580     // 描画時間範囲の情報を設定
581     timeline.SetTimeRange( curr_time - 2.0f, curr_time + 3.0f );
582 }
```

2.6.1 動作接続のための変換行列の計算（レポート課題）

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

前の動作の終了時の位置・向き $\mathbf{M}_{prev}$ に、次の動作の開始時の位置・向き $\mathbf{M}_{next}$ を合わせるための、変換行列 $\mathbf{M}$ を計算する。

簡単な方法としては、$\mathbf{M} = \mathbf{M}_{prev} \times \mathbf{M}_{next}^{-1}$ の式で計算される変換行列を用いることで、両者の位置・向きを合わせることができる。まずはこの計算方法を使ったプログラムを作成すると良い。（ 空欄 A ～ 空欄 C ）

しかしこの方法では、前の動作の終了時の上下・左右方向の傾きも保存されてしまうため、接続後の次の動作が上下・左右に傾いてしまうことがある。この問題を解決するためには、各行列から水平方向の向きのみを抜き出した行列 $\mathbf{M}'_{prev}, \mathbf{M}'_{next}$ を使って、同様に変換行列を計算する。（ 空欄 C ～ 空欄 D ）

$\mathbf{M}'_{prev}, \mathbf{M}'_{next}$ の計算には、vecmath の関数（Matrix3f クラスの rotY 関数）や C 言語の標準関数（atan2 関数）を使用できる。（各関数のつ使い方は、それぞれのリファレンスマニュアルを参照する。）

ソースコード 38: 動作接続のための変換行列の計算

```
// 方法 1
void  ComputeConnectionTransformation( const Matrix4f & prev_frame, const Matrix4f &
    next_frame, Matrix4f & trans_mat )
{
    // 次の動作の変換行列を計算
    空欄 A
    空欄 B
}

// 方法 2
void  ComputeConnectionTransformation( const Matrix4f & prev_frame, const Matrix4f &
    next_frame, Matrix4f & trans_mat )
{
    Matrix3f  ori;
    int  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_frame.get( &ori );
    prev_frame.get( &pos );
    空欄 C
    空欄 D
    pos.y = 0.0f;
    prev_frame2.set( ori, pos, 1.0f );
```

```
    // 変換行列から水平方向の回転のみを抽出して再設定
    next_frame.get( &ori );
    next_frame.get( &pos );
      空欄C
      空欄D
    pos.y = 0.0f;
    next_frame2.set( ori, pos, 1.0f );

    // 座標変換を計算
    空欄A～Bと同様（prev_frameをprev_frame2、next_frameをnext_frame2に置き換える）
}
```

2.6.2 動作遷移のタイミング・姿勢の計算（レポート課題）

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

動作遷移（前後の動作の接続時のブレンド）を考慮した、動作接続・遷移を実現する。

1. 動作遷移の初期化（MotionTransition クラスの Init メンバ関数）
   前の動作、後の動作、前の動作に対する変換行列、前の動作の開始時刻（グローバル時間）を入力として、動作遷移の初期化を行う。
   動作遷移における前後の動作のブレンドには、さまざま方法があるが、今回は、前の動作の終了時のブレンド開始時刻（blend_begin_time）と後の動作の開始時刻（begin_time）を合わせるように、後の動作の開始時刻、動作ブレンドの開始・終了時刻を計算する。（ 空欄A ～ 空欄C ）
   なお、これらの時刻は全て、前の動作の開始時刻を基準とするローカル時刻で表すことに注意する。

2. 動作遷移の姿勢計算（MotionTransition クラスの GetPosture メンバ関数）
   グローバル時間を入力として、その時間に対応する動作遷移の姿勢を出力する。
   処理の流れは、以下の通り。

   1 初期化処理で計算したタイミングにもとづいて、入力された時刻が、前の動作のみの区間、後の動作のみの区間、前後の動作のブレンド区間のどの区間に対応するかを判定する。

   2 前の動作の姿勢が必要な場合は、前の動作のローカル時刻を計算して、姿勢を取得し、変換行列を適用する。

   3 後の動作の姿勢が必要な場合は、後の動作のローカル時刻を計算して（ 空欄D ）、姿勢を取得し、変換行列を適用する。

   4 前後の動作のブレンド区間であれば、前後の動作の姿勢の補間の比率を計算して（ 空欄E ）、姿勢補間を行う。
   姿勢補間の比率は、区間の開始時で 0 となり、区間の終了時で 1 となるように計算する。（重みを線形に変化させる単純な関数を使用する。）

ソースコード 39: 動作遷移のタイミング・姿勢の計算

```
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 = __空欄A__;

    // 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 = __空欄B__;

    // 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 = __空欄C__;

    // 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 = __空欄D__

        // 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 = 空欄E

        // 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
}
```

2.7 逆運動学計算（CCD 法）

動作変形アプリケーションを実現する、InverseKinematicsCCDApp クラスの処理を作成する。マウス操作により、キャラクタの姿勢を対話的に変更できる。関節点を左ドラッグすることで、末端関節（緑）の目標位置を移動できる。逆運動学計算により、末端関節の目標位置を満たすように、支点関節から末端関節の間の全関節の回転を変化させることで、姿勢を変更する。末端関節は、マウス操作にもとづいて、3 次元空間内の視線に対して垂直な平面上で上下左右に移動する。関節点を SHIFT キー＋左クリックすることで、支点関節（赤）に設定する。V キーで、関節点の描画の有無を切替える。

InverseKinematicsCCDApp クラスの定義・実装を、ソースコード 40・41 に示す。本クラスについては、一部の処理はサンプルプログラムでは空欄となっているため、各自で処理を追加する必要がある。

ソースコード 40: 逆運動学計算（CCD 法）アプリケーションの定義（InverseKinematicsCCDApp.h）

```
1  /**
2  ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプログラム
3  ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4  ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5  **/
6
7  /**
8  ***  逆運動学計算（CCD法）アプリケーション
9  **/
10
11 #ifndef  _INVERSE_KINEMATICS_CCD_APP_H_
12 #define  _INVERSE_KINEMATICS_CCD_APP_H_
13
14
15 // ライブラリ・クラス定義の読み込み
16 #include "SimpleHuman.h"
17 #include "SimpleHumanGLUT.h"
18
19
```

```
20 //
21 //  逆運動学計算（CCD法）アプリケーションクラス
22 //
23 class  InverseKinematicsCCDApp : public GLUTBaseApp
24 {
25   protected:
26    // 姿勢＋逆運動学計算による姿勢変形情報
27
28    // キャラクタの姿勢
29    Posture *  curr_posture;
30
31    // 支点・末端関節
32    int  base_joint_no;
33    int  ee_joint_no;
34
35   protected:
36    // IK計算のための変数
37
38    // 関節点の位置
39    vector< Point3f >  joint_world_positions;
40    vector< Point3f >  joint_screen_positions;
41
42   protected:
43    // 描画設定
44
45    // 関節点の描画設定
46    bool   draw_joints;
47
48
49   public:
50    // コンストラクタ
51    InverseKinematicsCCDApp();
52
53    // デストラクタ
54    virtual ~InverseKinematicsCCDApp();
55
56   public:
57    // イベント処理
58
59    //  初期化
60    virtual void  Initialize();
61
62    //  開始・リセット
63    virtual void  Start();
64
65    //  画面描画
66    virtual void  Display();
67
68    // マウスクリック
69    virtual void  MouseClick( int button, int state, int mx, int my );
70
71    //  マウスドラッグ
72    virtual void  MouseDrag( int mx, int my );
73
74    // キーボードのキー押下
75    virtual void  Keyboard( unsigned char key, int mx, int my );
76
77   public:
78    // Inverse Kinematics 処理
79
80    //  Inverse Kinematics 計算（CCD法）
81    virtual void  ApplyInverseKinematics( Posture & posture, int base_joint_no, int
          ee_joint_no, Point3f ee_joint_position );
82
```

```
83    public:
84      // 関節点の選択・移動のための補助処理
85
86      // 関節点の位置の更新
87      void  UpdateJointPositions( const Posture & posture );
88
89      // 関節点の描画
90      void  DrawJoint();
91
92      // 関節点の選択
93      void  SelectJoint( int mouse_x, int mouse_y, bool ee_or_base );
94
95      // 関節点の移動（視線に垂直な平面上で上下左右に移動する）
96      void  MoveJoint( int mouse_dx, int mouse_dy );
97  };
98
99
100 // 補助処理（グローバル関数）のプロトタイプ宣言
101
102 // 末端関節から支点関節へのパス（関節の配列と各関節における末端関節の方向）を探索
103 void  FindJointPath( const Skeleton * body, int base_joint_no, int ee_joint_no, vector<
        int > & joint_path, vector< int > & joint_path_signs );
104
105 // Inverse Kinematics 計算（CCD法）
106 void  ApplyInverseKinematicsCCD( Posture & posture, int base_joint_no, int ee_joint_no,
        Point3f ee_joint_position );
107
108 // 順運動学計算（※レポート課題）
109 void  MyForwardKinematics( const Posture & posture, vector< Matrix4f > &
      seg_frame_array, vector< Point3f > & joi_pos_array );
110
111
112 #endif // _INVERSE_KINEMATICS_CCD_APP_H_
```

ソースコード 41: 逆運動学計算（CCD 法）アプリケーションの実装（InverseKinematicsCCDApp.cpp）

```
1   /**
2   ***  キャラクタアニメーションのための人体モデルの表現・基本処理 ライブラリ・サンプルプ
        ログラム
3   ***  Copyright (c) 2015-, Masaki OSHITA (www.oshita-lab.org)
4   ***  Released under the MIT license http://opensource.org/licenses/mit-license.php
5   **/
6
7   /**
8   ***  逆運動学計算（CCD法）アプリケーション
9   **/
10
11
12  // ライブラリ・クラス定義の読み込み
13  #include "SimpleHuman.h"
14  #include "InverseKinematicsCCDApp.h"
15  #include "BVH.h"
16
17  // プロトタイプ宣言
18
19  // 順運動学計算（※レポート課題）
20  void  MyForwardKinematics( const Posture & posture, vector< Matrix4f > &
        seg_frame_array, vector< Point3f > & joi_pos_array );
21
22
23
24  //
25  //  コンストラクタ
26  //
```

```
27 InverseKinematicsCCDApp::InverseKinematicsCCDApp()
28 {
29     app_name = "Inverse Kinematics (CCD)";
30
31     curr_posture = NULL;
32     base_joint_no = -1;
33     ee_joint_no = -1;
34
35     draw_joints = true;
36 }
37
38
39 //
40 //  デストラクタ
41 //
42 InverseKinematicsCCDApp::~InverseKinematicsCCDApp()
43 {
44     if ( curr_posture && curr_posture->body )
45         delete  curr_posture->body;
46     if ( curr_posture )
47         delete  curr_posture;
48 }
49
50
51 //
52 //  初期化
53 //
54 void  InverseKinematicsCCDApp::Initialize()
55 {
56     GLUTBaseApp::Initialize();
57
58     // 骨格モデルの初期化に使用する BVHファイル
59     const char *  file_name = "sample_walking1.bvh";
60
61     // 動作データを読み込み
62     BVH *  bvh = new BVH( file_name );
63
64     // BVH動作から骨格モデルを生成
65     if ( bvh->IsLoadSuccess() )
66     {
67         Skeleton *  new_body = CoustructBVHSkeleton( bvh );
68
69         // 姿勢の初期化
70         if ( new_body )
71         {
72             curr_posture = new Posture( new_body );
73             InitPosture( *curr_posture, new_body );
74         }
75     }
76
77     // 動作データを削除
78     delete  bvh;
79 }
80
81
82 //
83 //  開始・リセット
84 //
85 void  InverseKinematicsCCDApp::Start()
86 {
87     GLUTBaseApp::Start();
88
89     if ( !curr_posture )
90         return;
```

```
91
92      // 姿勢初期化
93      InitPosture( *curr_posture );
94
95      // 関節点の更新
96      UpdateJointPositions( *curr_posture );
97
98      // 支点・末端関節の初期化
99      base_joint_no = -1;
100     ee_joint_no = -1;
101 }
102
103
104 //
105 //  画面描画
106 //
107 void  InverseKinematicsCCDApp::Display()
108 {
109     GLUTBaseApp::Display();
110
111     // キャラクタを描画
112     if ( curr_posture )
113     {
114         glColor3f( 1.0f, 1.0f, 1.0f );
115         DrawPosture( *curr_posture );
116         DrawPostureShadow( *curr_posture, shadow_dir, shadow_color );
117     }
118
119     // 視点が更新されたら関節点の位置を更新
120     if ( curr_posture && is_view_updated )
121     {
122         UpdateJointPositions( *curr_posture );
123
124         // 視点の更新フラグをクリア
125         is_view_updated = false;
126     }
127
128     // 関節点を描画
129     if ( curr_posture && draw_joints )
130     {
131         DrawJoint();
132     }
133
134     // 現在のモードを表示
135     DrawTextInformation( 0, "Inverse Kinematics (CCD-IK)" );
136 }
137
138
139 //
140 //  マウスクリック
141 //
142 void  InverseKinematicsCCDApp::MouseClick( int button, int state, int mx, int my )
143 {
144     GLUTBaseApp::MouseClick( button, state, mx, my );
145
146     // 左ボタンが押されたら、IKの支点・末端関節を選択
147     if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
148     {
149         // Shiftキーが押されていれば、支点関節を選択
150         if ( glutGetModifiers() & GLUT_ACTIVE_SHIFT )
151             SelectJoint( mx, my, false );
152         // Shiftキーが押されていなければ、末端関節を選択
153         else
154             SelectJoint( mx, my, true );
```

```
155     }
156 }
157
158
159 //
160 //  マウスドラッグ
161 //
162 void  InverseKinematicsCCDApp::MouseDrag( int mx, int my )
163 {
164     // 左ボタンのドラッグ中は、IKの末端関節の目標位置を操作
165     if ( drag_mouse_l )
166     {
167         MoveJoint( mx - last_mouse_x, my - last_mouse_y );
168     }
169
170     GLUTBaseApp::MouseDrag( mx, my );
171 }
172
173
174 //
175 //  キーボードのキー押下
176 //
177 void  InverseKinematicsCCDApp::Keyboard( unsigned char key, int mx, int my )
178 {
179     GLUTBaseApp::Keyboard( key, mx, my );
180
181     // v キーで関節点の描画の有無を変更
182     if ( key == 'v' )
183         draw_joints = !draw_joints;
184
185     // r キーで姿勢をリセット
186     if ( key == 'r' )
187         Start();
188 }
189
190
191 //
192 //  末端関節から支点関節へのパス（関節の配列と各関節における末端関節の方向）を探索
193 //  （支点関節の番号が -1 の場合は、ルート体節を支点とする）
194 //  （joint_path_signs は、各関節の子側に末端関節がある場合は 1、親側に末端関節がある場
        合は -1 を出力）
195 //
196 void  FindJointPath( const Skeleton * body, int base_joint_no, int ee_joint_no, vector<
        int > & joint_path, vector< int > & joint_path_signs )
197 {
198     // 出力の配列をクリア
199     joint_path.clear();
200     joint_path_signs.clear();
201
202     // 探索時の現在の関節・体節
203     const Joint *  joint = NULL;
204     const Segment *  segment = NULL;
205
206     // 末端関節から探索を開始
207     joint = body->joints[ ee_joint_no ];
208
209     // 末端関節からルート体節に向かうパスを探索
210     while ( true )
211     {
212         // ルート側の隣の関節を辿り、ルートに到達したら終了
213         segment = joint->segments[ 0 ];
214         if ( segment->index == 0 )
215             break;
216         joint = segment->joints[ 0 ];
```

```
217
218         // 現在の関節をパスに追加
219         joint_path.push_back( joint->index );
220
221         // 途中で支点関節に到達したら終了
222         if ( joint->index == base_joint_no )
223             break;
224     }
225
226     // 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（全て子側に末端関節がある）
227     joint_path_signs.resize( joint_path.size(), 1 );
228
229
230     // 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
231     // 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
232     return;
233
234 /*
235     // ※レポート課題
236
237     // 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
238     if ( ( base_joint_no == -1 ) || ( joint->index == base_joint_no ) )
239         return;
240
241     // 支点関節からルート体節へ向かうパス
242     vector< int >  joint_path2;
243
244     // 探索処理の終了判定用フラグ
245     bool  termination = false;
246
247     // 支点関節から探索を開始
248     joint = body->joints[ base_joint_no ];
249
250     // 支点関節からルート体節に向かうパスを探索
251     while ( true )
252     {
253         // 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
254         for ( ??? )
255         {
256             if ( ??? )
257             {
258                 // 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
259                 joint_path.resize( i + 1 );
260                 termination = true;
261                 break;
262             }
263         }
264         if ( termination )
265             break;
266
267         // 現在の関節をパスに追加
268         joint_path2.push_back( joint->index );
269
270         // ルート側の隣の関節を辿り、ルートに到達したら終了
271         segment = joint->segments[ 0 ];
272         if ( segment->index == 0 )
273             break;
274         joint = segment->joints[ 0 ];
275
276         // 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、
277         // 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
278         if ( joint->index == ee_joint_no )
```

```
279         {
280             joint_path.clear();
281             joint_path_signs.clear();
282             break;
283         }
284     }
285 
286     // 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）
287     // 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
288     joint_path_signs.resize( joint_path.size(), 1 );
289     for ( int i = 0; i < joint_path2.size(); i++ )
290     {
291         joint_path.push_back( joint_path2[ ??? ] );
292         joint_path_signs.push_back( -1 );
293     }
294 */
295 
296 }
297 
298 
299 //
300 //  Inverse Kinematics 計算（CCD法）
301 //  入出力姿勢、支点関節番号（-1の場合はルートを支点とする）、末端関節番号、末端関節の目標位置を指定
302 //
303 void  InverseKinematicsCCDApp::ApplyInverseKinematics( Posture & posture, int base_joint_no, int ee_joint_no, Point3f ee_joint_position )
304 {
305     ApplyInverseKinematicsCCD( posture, base_joint_no, ee_joint_no, ee_joint_position );
306 }
307 
308 
309 //
310 //  Inverse Kinematics 計算（CCD法）
311 //  入出力姿勢、支点関節番号（-1の場合はルートを支点とする）、末端関節番号、末端関節の目標位置を指定
312 //
313 void  ApplyInverseKinematicsCCD( Posture & posture, int base_joint_no, int ee_joint_no, Point3f ee_joint_position )
314 {
315     // 最大繰り返し数の設定
316     const int  max_iteration = 10;
317 
318     // 位置が収束したと判断するための閾値の設定
319     const float  distance_threshold = 0.01f;
320 
321 
322     // 順運動学計算結果の格納用変数
323     vector< Matrix4f >  segment_frames;
324     vector< Point3f >  joint_positions;
325 
326     // 骨格情報
327     const Skeleton *  body = posture.body;
328 
329     // 現在の関節
330     const Joint *  joint = NULL;
331 
332     // 現在の関節の支点側の体節
333     const Segment *  segment = NULL;
334 
335     // ルートから見た現在の関節の方向（末端側の場合は 1、支点側の場合は -1）
336     float  direction;
337 
```

```
338     // 末端関節の現在位置（ワールド座標系）
339     Point3f  ee_pos;
340
341     // 現在の関節の局所座標系（ワールド座標系における関節の位置＋親側の体節の向き）
342     Matrix4f  local_frame;
343
344     // ワールド座標系から現在の関節の局所座標系への変換行列
345     Matrix4f  trans_mat;
346
347     // 末端関節の現在位置（局所座標系）
348     Point3f  local_pos;
349
350     // 支点関節から末端関節へのベクトル（局所座標系）
351     Vector3f  ee_vec;
352
353     // 末端関節の現在位置から目標位置へのベクトル（局所座標系）
354     Vector3f  goal_vec;
355
356     // 関節の回転軸と回転角度
357     Vector3f  rot_axis;
358     float  rot_angle = 0.0f;
359
360     // 現在の関節の回転
361     Matrix3f  rot;
362
363     // 末端関節の目標位置と現在位置の距離
364     Vector3f  vec;
365     float  dist = -1.0f;
366
367
368     // 引数チェック
369     if ( !posture.body || ( ee_joint_no == -1 ) || ( base_joint_no == ee_joint_no ) )
370        return;
371
372     // 末端関節から支点関節へのパス（関節の配列と各関節における末端関節の方向）を探索
373     vector< int >  joint_path, joint_path_signs;
374     FindJointPath( body, base_joint_no, ee_joint_no, joint_path, joint_path_signs );
375
376     // 現在の姿勢での各体節・関節の位置・向きを計算（順運動学計算）
377     ForwardKinematics( posture, segment_frames, joint_positions );
378
379     // 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）
380     for ( int i = 0; i < max_iteration; i++ )
381     {
382        // 末端関節から支点関節に向かって順番に繰り返し
383        for ( int j = 0; j < joint_path.size(); j++ )
384        {
385           // 現在の関節と支点側の体節を取得
386           joint = body->joints[ joint_path[ j ] ];
387           direction = (float) joint_path_signs[ j ];
388           segment = ( direction > 0.0f ) ? joint->segments[ 0 ] : joint->segments[ 1 ];
389
390           // 末端関節の現在位置を取得
391           ee_pos = joint_positions[ ee_joint_no ];
392
393           // ※ レポート課題
394
395           // 現在の関節のローカル座標系を取得
396 //        mat = ???;
397
398           // 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
399 //        inv_mat = ???;
400
401           // 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
```

```
402 //          ee_vec = ???;
403
404             // 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
405 //          goal_vec = ???;
406
407             // 現在の関節の回転軸・回転角度（0～π）を計算
408 //          rot_axis = ???;
409 //          rot_angle = ???;
410
411             // 回転角度が微少であれば、回転は適用せずにスキップする
412             if ( rot_angle < 0.001f )
413                continue;
414
415             // 回転を適用（回転の方向を考慮しない）
416 //          rot.set( AxisAngle4f( rot_axis, rot_angle ) );
417 //          ???;
418
419             // 回転後の回転を設定
420             posture.joint_rotations[ joint->index ].set( rot );
421
422             // 末端関節と現在の関節の間にルート体節がある場合は、ルート体節に移動・回転を
                    適用
423             if ( direction < 0.0f )
424             {
425             }
426
427             // 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
                    ）
428             MyForwardKinematics( posture, segment_frames, joint_positions );
429          }
430
431          // 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
432          ee_pos = joint_positions[ ee_joint_no ];
433          vec.sub( ee_joint_position, ee_pos );
434          dist = vec.lengthSquared();
435          if ( dist < distance_threshold * distance_threshold )
436             break;
437       }
438 }
439
440
441 //
442 //  以下、補助処理
443 //
444
445
446 //
447 //  関節点の位置の更新
448 //
449 void  InverseKinematicsCCDApp::UpdateJointPositions( const Posture & posture )
450 {
451    if ( !curr_posture )
452       return;
453
454    // 順運動学計算
455    vector< Matrix4f >  seg_frame_array;
456    ForwardKinematics( posture, seg_frame_array, joint_world_positions );
457
458    // OpenGL の変換行列を取得
459    double  model_view_matrix[ 16 ];
460    double  projection_matrix[ 16 ];
461    int  viewport_param[ 4 ];
462    glGetDoublev( GL_MODELVIEW_MATRIX, model_view_matrix );
463    glGetDoublev( GL_PROJECTION_MATRIX, projection_matrix );
```

```
464     glGetIntegerv( GL_VIEWPORT, viewport_param );
465
466     // 画面上の各関節点の位置を計算
467     int  num_joints = joint_world_positions.size();
468     GLdouble  spx, spy, spz;
469     joint_screen_positions.resize( num_joints );
470     for ( int i=0; i<num_joints; i++ )
471     {
472         const Point3f &  wp = joint_world_positions[ i ];
473         Point3f &  sp = joint_screen_positions[ i ];
474
475         gluProject( wp.x, wp.y, wp.z,
476             model_view_matrix, projection_matrix, viewport_param,
477             &spx, &spy, &spz );
478         sp.x = spx;
479         sp.y = viewport_param[ 3 ] - spy;
480     }
481 }
482
483
484 //
485 //  関節点の描画
486 //
487 void  InverseKinematicsCCDApp::DrawJoint()
488 {
489     if ( !curr_posture )
490         return;
491
492     // デプステストを無効にして、前面に上書きする
493     glDisable( GL_DEPTH_TEST );
494
495     // 関節点を描画（球を描画）
496     for ( int i = 0; i < joint_world_positions.size(); i++ )
497     {
498         // 支点関節は赤で描画
499         if ( i == base_joint_no )
500             glColor3f( 1.0f, 0.0f, 0.0f );
501         // 末端関節は緑で描画
502         else if ( i == ee_joint_no )
503             glColor3f( 0.0f, 1.0f, 0.0f );
504         // 他の関節は青で描画
505         else
506             glColor3f( 0.0f, 0.0f, 1.0f );
507
508         // 関節位置に球を描画
509         const Point3f &  pos = joint_world_positions[ i ];
510         glPushMatrix();
511             glTranslatef( pos.x, pos.y, pos.z );
512             glutSolidSphere( 0.025f, 16, 16 );
513         glPopMatrix();
514     }
515
516     // 支点関節が指定されていない場合は、ルート体節を支点とする（ルート体節の位置に球を
        描画）
517     if ( base_joint_no == -1 )
518     {
519         // ルート体節位置に球を描画
520         glColor3f( 1.0f, 0.0f, 0.0f );
521         const Point3f &  pos = curr_posture->root_pos;
522         glPushMatrix();
523             glTranslatef( pos.x, pos.y, pos.z );
524             glutSolidSphere( 0.025f, 16, 16 );
525         glPopMatrix();
526     }
```

```
527 |
528 |     glEnable( GL_DEPTH_TEST );
529 | }
530 |
531 |
532 | //
533 | //  関節点の選択
534 | //
535 | void  InverseKinematicsCCDApp::SelectJoint( int mouse_x, int mouse_y, bool ee_or_base )
536 | {
537 |     if ( !curr_posture )
538 |         return;
539 |
540 |     const float  distance_threthold = 20.0f;
541 |     float  distance, min_distance = -1.0f;
542 |     int  closesed_joint_no = -1;
543 |     float  dx, dy;
544 |
545 |     // 入力座標と最も近い位置にある関節を探索
546 |     for ( int i = 0; i < joint_screen_positions.size(); i++ )
547 |     {
548 |         dx = joint_screen_positions[ i ].x - mouse_x;
549 |         dy = joint_screen_positions[ i ].y - mouse_y;
550 |         distance = sqrt( dx * dx + dy * dy );
551 |         if ( ( i == 0 ) || ( distance <= min_distance ) )
552 |         {
553 |             min_distance = distance;
554 |             closesed_joint_no = i;
555 |         }
556 |     }
557 |
558 |     // 距離が閾値以下であれば選択
559 |     if ( ee_or_base )
560 |     {
561 |         if ( min_distance < distance_threthold )
562 |             ee_joint_no = closesed_joint_no;
563 |         else
564 |             ee_joint_no = -1;
565 |     }
566 |     else
567 |     {
568 |         if ( min_distance < distance_threthold )
569 |             base_joint_no = closesed_joint_no;
570 |         else
571 |             base_joint_no = -1;
572 |     }
573 | }
574 |
575 |
576 | //
577 | //  関節点の移動（視線に垂直な平面上で上下左右に移動する）
578 | //
579 | void  InverseKinematicsCCDApp::MoveJoint( int mouse_dx, int mouse_dy )
580 | {
581 |     if ( !curr_posture )
582 |         return;
583 |
584 |     // 末端関節が選択されていなければ終了
585 |     if ( ee_joint_no == -1 )
586 |         return;
587 |
588 |     // 画面上の移動量と3次元空間での移動量の比率
589 |     const float  mouse_pos_scale = 0.01f;
590 |
```

```
591     // OpenGL の変換行列を取得
592     double  model_view_matrix[ 16 ];
593     glGetDoublev( GL_MODELVIEW_MATRIX, model_view_matrix );
594
595     Vector3f  vec;
596     Point3f &  pos = joint_world_positions[ ee_joint_no ];
597
598     // カメラ座標系のX軸方向に移動
599     vec.set( model_view_matrix[ 0 ], model_view_matrix[ 4 ], model_view_matrix[ 8 ] );
600     pos.scaleAdd( mouse_dx * mouse_pos_scale, vec, pos );
601
602     // カメラ座標系のX軸方向に移動
603     vec.set( model_view_matrix[ 1 ], model_view_matrix[ 5 ], model_view_matrix[ 9 ] );
604     pos.scaleAdd( - mouse_dy * mouse_pos_scale, vec, pos );
605
606     // Inverse Kinematics 計算を適用
607     ApplyInverseKinematics( *curr_posture, base_joint_no, ee_joint_no, pos );
608
609     // 関節点の更新
610     UpdateJointPositions( *curr_posture );
611 }
```

2.7.1 逆運動学計算（CCD 法）（ルート体節を支点とする場合）（レポート課題）

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

CCD 法による逆運動学計算の処理を作成する。最初は、ルート体節が支点であると仮定して、その場合に正しく動作するプログラムを作成する。この場合、末端関節から見て支点は常に親側にあるため、回転軸の方向や関節・体節を辿る方向を変化させる必要がなく、プログラムが簡単になる。

処理の流れは、以下の通り。

1. 末端関節の現在位置が目標位置に収束する（または繰り返し回数が一定回数を超える）まで、以下の処理を繰り返し計算
2. 末端関節から支点関節に向かって順番に繰り返し
   1 現在の関節（joint）を取得、現在の関節に隣接する支点側の体節（segment）も取得
   2 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
      1 末端関節の現在位置を取得（ワールド座標系） （ 空欄 A ）
      順運動学計算結果の関節位置の配列（joint_positions）から取得する。
      2 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 （ 空欄 B ）
      順運動学計算結果の体節の変換行列の配列（segment_frames）と関節位置から計算する。
      3 現在の関節から末端関節の現在位置までのベクトル $\mathbf{l}$ を計算（関節のローカル座標系） （ 空欄 C ）
      4 現在の関節から末端関節の目標位置までのベクトル $\Delta\mathbf{p}$ を計算（関節のローカル座標系） （ 空欄 D ）
      $\Delta\mathbf{p}$ が長すぎると計算誤差が大きくなるため、長過ぎる場合には縮小する処理を入れる。（目標移動量が小さければ、省略可。）
      5 末端関節の位置を現在位置から目標位置に近づけるための、現在の関節の回転を計算 （ 空欄 E ）

$$\mathbf{w} = \mathbf{l} \times \Delta\mathbf{p} \tag{3}$$

$\mathbf{w}$ の方向から回転軸、長さから回転角度を計算

6 計算した回転を、現在の関節の回転に適用して、姿勢を更新 （ 空欄 F ）
AxisAngle4f 型の変数を回転軸・角度を指定して初期化し、Matrix3f 型に変換して、現在の関節回転にかける。
任意の関節が支点になる場合は、回転の方向に注意する必要があるが、ルート体節が支点の場合は、考慮する必要はない。

7 計算結果を、現在姿勢の関節回転（posture.joint_rotations）に設定

8 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）

3. 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であれば、繰り返し処理を終了）

ソースコード 42: 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++ )
      {
         // 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
         空欄 A

         // 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
         空欄 B

         // 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
         空欄 C

         // 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
         空欄 D

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

         // 回転を適用（回転の方向を考慮しない）
         空欄 F

         // 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;
    }
}
```

2.7.2 末端関節から支点関節へのパスの探索（任意の関節を支点とする場合）（レポート課題）

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

任意の関節を支点とする場合は、末端関節から支点関節へのパス（順番に辿る関節のリスト）を探索する処理の拡張が必要となる。もともとのサンプルプログラムで、ルート体節から末端関節へのパスを探索する処理は作成されているので、そのパスに、支点関節からルート体節までのパスを探索して結合する処理を追加する。
すなわち、支点関節→末端関節のパス = 支点関節→ルート体節のパス + ルート体節→末端関節のパス となる。
基本的には、支点関節からルート関節に向かうパスを探索して、これまでのパスに結合する。ただし、以下のように、末端関節・支点関節の組み合わせによっては、例外への対応が必要となる。

1. ルート体節（腰）から支点関節へのパス上に末端関節がある場合（例：右肩が支点、右手が末端）
   最初に求めたルートから末端関節までのパスは消去して、支点関節から末端関節までのパスを使用する。
2. 上の場合以外で、ルート体節（腰）から見て末端関節と支点関節が同じ方向にある場合（例：左手が支点、右手が末端）
   最初に求めたルートから末端関節までのパスから、ルートから２つのパスが合流する関節までのパスを消去して、支点関節から合流関節までのパスを追加する。

処理の流れは、以下の通り。

1. 支点関節からルート体節に向かうパスを探索
   1 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、合流したら探索を終了 （ 空欄 A ～ 空欄 B ）
   2 現在の関節をパスに追加
   3 ルート側の隣の関節を辿り、ルートに到達したら終了
   4 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
2. 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
3. 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 （ 空欄 C ）

ソースコード 43: 末端関節から支点関節へのパスの探索（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 )
{
    // 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 ( __空欄A__ )
        {
            if ( __空欄B__ )
            {
                // 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[ __空欄C__ ] );
        joint_path_signs.push_back( -1 );
    }
}
```

2.7.3 逆運動学計算（CCD 法）（任意の関節を支点とする場合）（レポート課題）

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

任意の関節を支点とする場合、基本的な処理は 2.7.1 節で作成した処理と同じである（ 空欄 A ～ 空欄 E ）が、ルート体節よりも支点側にある関節を回転する際に、以下のような処理を加える必要がある。

1. 適用する回転の方向の考慮
   姿勢表現では親側（ルート側）から子側（ルートと反対側）の回転を表すが、現在の関節から末端関節までのパス上にルート体節がある場合は、CCD 計算では子側から親側の回転を計算することになるため、回転の方向によって計算方法を変更する。（ 空欄 G ～ 空欄 H ）

2. ルートの移動・回転
   現在の関節から末端関節までのパス上にルート体節がある場合は、現在の関節を回転させると、ルート体節も移動・回転することになる。計算した現在の関節の回転に応じたルート体節の位置・向きの変化を計算して、現在姿勢のルートの位置・向きの変数（posture.root_pos, posture.root_ori）を更新する。
   ルート体節の移動・回転は、例えば、以下のような方法で計算できる。

   1 姿勢変更前の支点体節（支点関節の支点側に隣接する体節）の位置・向き $\mathbf{M}_0$ を取得 （ 空欄 I ）

   2 変更した姿勢にもとづいて順運動学計算

   3 姿勢変更後の支点体節位置・向き $\mathbf{M}_1$ を取得 （ 空欄 J ）

   4 支点体節の位置・向きを保つための変換行列 $\mathbf{M}$ を計算 （ 空欄 K ）

$$\mathbf{M} = \mathbf{M}_0 \, \mathbf{M}_1^{-1} \tag{4}$$

   5 現在姿勢の腰の位置・向きに、変換行列 $\mathbf{M}$ を適用 （ 空欄 L ）

```
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
      空欄A

      // 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
      空欄B

      // 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
      空欄C
```

```
            // 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
            空欄D

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

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

            // 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
                空欄I

                // 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
                空欄J

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

                // 腰の位置・向きに座標変換を適用
                空欄L
            }

            // 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;
    }
}
```