MOCAPデータファイル

http://www.tmps.org/index.php?MOCAP%A5%C7%A1%BC%A5%BF%A5%D5%A5%A1%A5%A4%A5%EB

モーションキャプチャを用いた人体動作のデジタル計測技術は，CG アニメーションの世界だけではなく，リハビリテーションやバイオメカニクスの分野においても広く利用されています．本稿では，モーションデータを格納するための代表的なファイルフォーマットである，BVH ファイルと ASF-AMC ファイルの 2 つの形式について説明します．まず，それぞれのファイル形式について簡単に説明した後，それぞれのモーションデータファイルを読み込み，動作の内容をスケルトン表示するプログラムを作成します．

Fig.1 実行画面のスナップショット

参考ページ [†]

BVH，ASF/AMC 形式のファイルフォーマットに関するWeb資料をいくつか挙げておきます．

- BVH モーションデータファイル
  - RIKIYA（商用モーションデータライブラリ．サンプル有）
  - Wisconsin 大 CG コースノート

    - 九工大 CG コースノート
- Acclaimモーションデータファイル
    - CMU Mocap Database(研究用途の大規模モーションデータライブラリ．研究機関からのみアクセス可？)
    - Wisconsin 大 CG コースノート
    - Motion Reality 社資料

BVH ファイル

BVH ファイル形式とは，Biovision 社が提唱したモーションキャプチャデータファイルフォーマットです．現在，Alias Motion Builder をはじめ，3ds MAX の Character studio，Poser などの様々な商用3Dキャラクタアニメーションソフトでもサポートされています．BVH ファイルの特徴を以下にまとめます．

- テキスト形式で記述
- 座標系は右手系．XYZ 各軸の扱い(どの軸が鉛直方向に対応するか等)は任意．
- 関節ノードに関する情報を記述．
- 関節回転はオイラー角形式で記述．
- 回転角度の単位は Degree
- キャラクタのスケルトン階層構造を記述するHIERARCHY部と，動作データを記述するMOTION部の2つから構成

2つめの特徴に挙げたように，BVHファイルの座標系は右手系になります．そのため OpenGL などとの相性は良いのですが，左手系を採用している DirectX Graphics などでは座標系変換が必要になります．

次に，単純な BVH ファイルの記述例を以下に示します．なお，"#"以下のコメントは，実際にファイルに記述することは出来ません(BVH ファイルへのコメントの書き込みは不可)

```
# リスト1
HIERARCHY
ROOT root_name
{
  OFFSET 0 0 0
  CHANNELS 6 Xposition Yposition Zposition Xrotation Yrotation Zrotation
  End Site
  {
    OFFSET 0 10 0
  }
}
MOTION
Frames: 1
Frame Time: 0.033
0 0 0 0 0 0
```

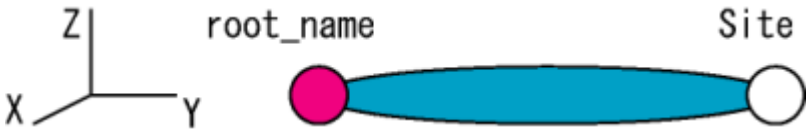

Fig.2 1ボーンスケルトン(右手系)

この BVH ファイルは，Fig.2 に示すような 1 つのボーンで構成されるスケルトンの，1 フレーム分の動作を記述したものです．つまり，1 つのボーンの静止姿勢(3 次元座標と方向)を指定しています．このファイルを例に，まず BVH ファイルの基本要素について説明していきます．

BVH スケルトン階層記述部：HIERARCHY

HIERARCHY 部の基本要素

HIERARCHY キーワードで始まる部分では，スケルトンの構造を定義します．各関節の初期オフセット(つまりボーンの長さと初期方向)，関節の親子接続関係，そして関節自由度の情報を記述します．BVH ファイルにおける関節自由度とは，オイ

ラー角形式で記述された回転情報から，関節回転を表すトランスフォーム行列（もしくはクォータニオン）を合成するための情報です．つまり，回転主軸となる軸の種類と数，回転の合成順序が記述されます．なお，ルートノードには 3 次元位置の自由度も与えることができます．

まず，HIERARCHY 部で利用するキーワードを列挙します．

- HIERARCHY
  - スケルトン階層構造の定義の開始を宣言．
- JOINT（関節ノード）
  - OFFSET と CHANNELS（回転のみ）を要素に持つ
  - 必ず 1 つ以上の JOINT，End を子ノードに持つ．
- ROOT（ルートノード）
  - 階層構造の唯一の始点となる特殊な JOINT
  - OFFSET，CHANNELS (位置と回転)を要素に持つ
  - 必ず 1 つ以上の JOINT もしくは End を子ノードに持つ．
- End（エフェクタ）
  - 階層構造の末端ノードとなる特殊な JOINT．
  - OFFSET のみを要素に持つ．
  - 子ノードは持たない．
- OFFSET （関節オフセットベクトル）
  - 親ノードから関節ノードへの 3 次元オフセット成分．
  - ルートノードの場合は，ワールド座標系における初期位置．
  - X-Y-Z の順序で記述．
- CHANNELS（関節自由度）
  - 関節自由度数に続き，位置の自由度 (Xposition，Yposition，Zposition)，回転の自由度 (Xrotation，Yrotation，Zrotation)を，ルートノードから末端に向かう順序に定義します．
  - リスト 1 の BVH ファイルでは，関節回転によるベクトル v のトランスフォームは，$v' = vR_z(\theta_z)R_y(\theta_y)R_x(\theta_x)T(t_x.t_y.t_z)$ で計算されます．

リスト1 の BVH ファイルでは，"root_name"と命名されたルートノードと，"Site"と命名された末端ノードで構成されるスケルトンが構築されます．ボーンはノード間にのみ生成されますので，このスケルトンのボーン数は 1 となります．また，ルートは XYZ の全てに軸に関する位置の自由度と，X-Y-Z 順で合成されるのオイラー角回転の，計 6 つの自由度を持つことがわかります．

もちろん，さらに少ない自由度を持つスケルトンも定義可能です．リスト 2 の例では Z 軸周りにのみ回転する 1 ボーンのスケルトンを生成し，リスト 3 では X 軸に沿ってのみ平行移動する 1 ボーンスケルトンを生成します．

```
# リスト2
HIERARCHY
ROOT root_name
{
  OFFSET 0 0 0
  CHANNELS 1 Zrotation
  End Site
  {
    OFFSET 0 10 0
  }
}
```

```
# リスト3
HIERARCHY
ROOT root_name
{
  OFFSET 0 0 0
  CHANNELS 1 Xposition
```

```
  End Site
  {
    OFFSET 0 10 0
  }
}
```

階層構造の定義方法

次に，リスト1 を拡張し，2 つのボーンで構成されるスケルトンの HIERARCHY を作成します．

```
# リストA-4
HIERARCHY
ROOT root_name
{
  OFFSET 0 0 0
  CHANNELS 6 Xposition Yposition Zposition Xrotation Yrotation Zrotation
  JOINT joint1
  {
    OFFSET 0 10 0
    CHANNELS 3 Xrotation Yrotation Zrotation
    End Site
    {
      OFFSET 0 10 0
    }
  }
}
```

ルートノードと Site ノードの間に joint1 ノードが追加されています．この例に示すように，ROOT もしくは JOINT の中括弧"{ ～ }"の中に記述された JOINT もしくは　End ノードが，その関節の子ノードとなります．例として，さらに 3 つのボーンでスケルトンを構成する場合を考えます．この場合，(1)全てのボーンを直列に接続する構造と，(2)1 つのボーンから 2 つのボーンに分岐する構造が考えられますが，それぞれについて HIERARCHY を記述すると次のようになります．

```
# リスト5 直列接続
HIERARCHY
ROOT root_name
{
  OFFSET 0 0 0
  CHANNELS 6 Xposition Yposition Zposition Xrotation Yrotation Zrotation
  JOINT joint1
  {
    OFFSET 0 10 0
    CHANNELS 3 Xrotation Yrotation Zrotation
    JOINT joint2
    {
      OFFSET 0 10 0
      CHANNELS 3 Xrotation Yrotation Zrotation
      End Site
      {
        OFFSET 0 10 0
      }
    }
  }
}
```

リスト 5 では，ルートノードから joint1 - joint2 - Site と連結された，4 階層のスケルトンが構築されます．

```
# リスト6 分岐接続
```

```
HIERARCHY
ROOT root_name
{
  OFFSET 0 0 0
  CHANNELS 6 Xposition Yposition Zposition Xrotation Yrotation Zrotation
  JOINT joint1
  {
    OFFSET 0 10 0
    CHANNELS 3 Xrotation Yrotation Zrotation
    End Site1
    {
      OFFSET -5 5 0
    }
    End Site2
    {
      OFFSET 5 5 0
    }
  }
}
```

ルートノードから joint1 ノードへの接続関係はリスト 5 と共通ですが，joint1 ノードが Site1 と Site2 の 2 つのエフェクタを子ノードとして持ちます．そのため，全体で 3 階層のスケルトンが構築されます．

BVH モーションデータ記述部：MOTION

MOTION 部には，HIERARCHY 部で定義された関節自由度に対応した，ルート位置と関節回転角の時系列情報を記述します．まず，MOTION 部で利用するキーワードを列挙します．

- MOTION
  - モーションデータ部の記述の開始を宣言
- Frames: N
  - 総フレーム数（時間長）N のモーションデータを記述すると宣言．
- Frame Time: SPF
  - フレーム当たりの時間長（seconds/sec）を記述．FPS値（Frames Per Ssecond）の逆数．

次に，リスト4 の HIERARCHY に MOTION 部を追加した例を示します．

```
# リスト7
HIERARCHY
ROOT root_name
{
  OFFSET 0 0 0
  CHANNELS 6 Xposition Yposition Zposition Xrotation Yrotation Zrotation
  JOINT joint1
  {
    OFFSET 0 10 0
    CHANNELS 3 Xrotation Yrotation Zrotation
    End Site
    {
      OFFSET 0 10 0
    }
  }
}
MOTION
Frames: 100
Frame Time: 0.033
0 0 0 0 0 0 0 0 0 // フレーム1のスケルトンの姿勢
```

```
0 0 0 0 0 0 0 0 0 // フレーム2のスケルトンの姿勢
…
0 0 0 0 0 0 0 0 0 // フレーム100のスケルトンの姿勢
```

MOTION, Frames, Frames Time の記述に続き，各フレームにおけるスケルトンの姿勢情報を1行づつ記述します．各行の数字の並びは，ファイル冒頭からの CHANNELS の登場順序に対応します（Fig.3参照）．

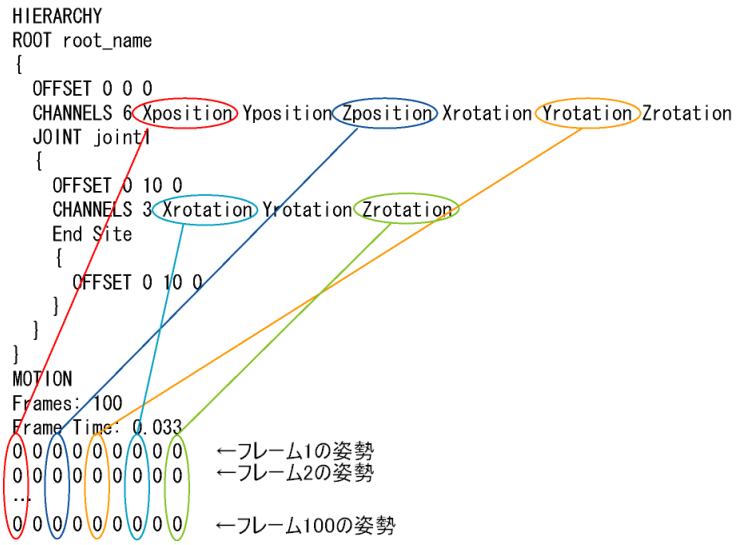

Fig.3 CHANNELS 情報と MOTION 部の対応

BVHファイル作成例

手作業で作成した BVH ファイルをいくつか示します．ファイルの内容と生成アニメーションの対応関係を確認してみてください．

- 1ボーン-静止姿勢（リスト1）
- 1ボーン-Z軸周りの回転（リスト2）
- 1ボーン-X軸方向の平行移動（リスト3）
- 1ボーン-Z,X軸周りの回転
- 1ボーン-X,Y軸方向の平行移動
- 2ボーン-静止姿勢
- 2ボーン-子関節がZ軸周りに回転
- 2ボーン-親関節がZ軸周りに回転
- 2ボーン-両関節がZ軸周りに回転
- 2ボーン-親関節がX軸方向に平行移動，両関節がZ軸周りに回転
- 4ボーン分岐-親関節がY軸方向に平行移動，第1子関節がY軸周りに回転，第3子関節がZ軸周りに回転

ASF-AMC ファイル

ASF-AMC ファイル形式は，Acclaim社（2004年に破産?）が提唱したモーションデータファイルフォーマットです．スケルトン情報を記述するためのASFファイル（Acclaim Skeletal-data Format ?）と，モーションデータを記述するためのAMCファイル（Acclaim Motion-Capture-data ?）で構成されます．ASF-AMC ファイルの特徴を以下にまとめます．

- テキスト形式で記述
- 座標系は右手系．XYZ 各軸の扱い（どの軸が鉛直方向に対応するか等）は任意．
- 関節ノードではなく，ボーンに関する情報を記述
- ボーン回転はオイラー角形式で記述．
- 回転角度の単位は Degree もしくは Radian
- スケルトン階層構造はASFファイルに，動作データはAMCファイルにそれぞれ記述

3つめの特徴に述べたように，ASFファイルはボーン情報の記述に着目したファイルフォーマットです．BVH 形式ではノード間のオフセット情報や接続情報をもとにボーンを構築しますが，ASF 形式ではボーンの長さや接続関係から，各関節ノードの位置や接続関係を構成します．

サンプルとして，リスト1 とほぼ同一内容のスケルトン構造とモーションデータを定義するASF-AMC ファイルを示します．なお，"#"から始まる行はコメント行として扱われます．

- リスト8のASFファイル
- リスト9のAMCファイル

```
# リスト8 サンプルASFファイル
:version 1.10              // ASFファイルバージョン．多くの場合 1.10
:name MinimalSet           // スケルトン名
:units                     // 単位系を変換するための係数
  mass 1.0                 // 重量の単位換算係数(キログラムからグラム，ポンド等)
  length 1.0               // 長さの単位換算(センチメートルからメートル，インチ等)
  angle deg                // 回転角度の単位 (deg もしくは rad)
:documentation             // 付加文書
  Sample asf file
:root                      // ルートノードの自由度，初期オフセット情報．
  order TX TY TZ RX RY RZ  // 自由度．この例では6自由度を持ち，回転順序は X-Y-Z
  axis XYZ                 // 次の orientation データの回転順序
  position 0 0 0           // 初期位置オフセット
  orientation 0 0 0        // 初期回転オフセット
:bonedata                  // 関節情報の記述
  begin
    id 1                   // ボーンインデクス
    name joint1            // ボーン名
    direction 0.0 1.0 0.0  // ボーンの初期方向（単位ベクトル）
    length 10.0            // ボーンの長さ
    axis 0 0 0  XYZ        // ボーン回転の基準座標系
    dof rx ry rz           // ボーン回転自由度．この例では X-Y-Z 順の3自由度
    limits (-180.0 180.0)  // dofで記述した順に各自由度の回転可動域を設定
           (-180.0 180.0)
           (-180.0 180.0)
    bodymass 1.0           // ボーンの質量(option)
    cofmass 1.0 end        // 重心位置(Center OF MASS, option)
  end
:hierarchy                 // ボーンの親子接続情報の設定
  begin
   root joint1             // "親ボーン 子ボーン1 子ボーン2 ..."の形式で記述
  end
```

```
# リスト9 サンプルAMCファイル
:FULLY-SPECIFIED              // 不明
:DEGREES                      // 回転角度の単位
1                             // フレーム1の姿勢の記述開始を宣言
root 0.0 0.0 0.0 0.0 0.0 0.0 // フレーム1におけるルートノードの位置，回転
joint1 0.0 0.0 0.0            // フレーム1における joint1 ボーンの回転
```

BVH形式と比較すると，より多数のデータで構成されています．ASF ファイルでは，まず version，name，units，documentation の各部でファイルもしくはスケルトンについての情報を記述します．次に，root 部でルートノードの関節自由度，初期位置・回転のオフセット情報を定義し，bonedata 部で各ボーンの初期方向，ボーン長，回転自由度，関節可動域を定義します．そして，hierarchy 部でボーンの接続関係を定義するという構成になっています．

ASFファイル

Acclaim 形式のスケルトン情報は ASF ファイルに記述されます．1 つの ASF ファイルに 1 つのスケルトンの情報を定義します．

スケルトン情報: version, name, units, documentation

ファイル全体の内容や，スケルトン全体に関連する情報は，version, name, units, そして documentation 部で定義します．

- version
  - ASF ファイルフォーマットのバージョン情報を定義します．ほとんどの場合，1.10 であると思われます．バージョン 1.2 の策定も検討されていたようですが，結局リリースされることはなかったようです．
- name
  - スケルトンの名前を定義します．
- units
  - mass
    - 質量の単位を換算するための係数を定義します．例えば，ポンド単位で記述された ASF ファイルをキログラムを基本単位とするシステムで利用するような場合，ポンドからキログラムへの換算係数 0.4536 を記述します．
  - length
    - 長さの単位を変換するための係数を定義します．インチ単位で記述された ASF ファイルをセンチ単位を用いるシステムで利用する場合，インチからセンチへの換算係数 2.54 を記述します．
  - angle
    - 回転角度の単位を定義します．deg もしくは rad を記述します．
- documentation
  - ファイルやスケルトンに関する補足文書を記述します．数行にわたって記述することも可能です．

ルート情報: root

root 部では，ルートノードの自由度情報，初期オフセット情報などが定義されます．

- order
  - ルートの自由度の数と種類，回転の合成順序を定義します．TX, TY, TZ がそれぞれX, Y, Z 方向の平行移動，RX, RY, RZ がそれぞれ回転の自由度を示します．リスト 8 の例では，ルートノードによるトランスフォームは $v' = vR_x(\theta_x)R_y(\theta_y)R_z(\theta_z)T(t_x.t_y.t_z)$ で計算されます．
- position
  - ワールド座標系原点からの初期平行移動成分を定義します．
- axis, orientation
  - 初期回転成分を定義します．axis で記述される回転順序と，orientation に記述される回転量を用いて計算されます．リスト 8 の例では，$R = R_x(0)R_y(0)R_z(0)$ となります．

ボーン情報: bonedata

スケルトンを構成するボーンの情報を定義します．:bonedata キーワードに続き，begin ～ end で囲まれた部分にそれぞれ 1 ボーンずつ，インデクス番号や長さ，回転自由度などの情報を記述します．

- id
  - ボーンのインデクス番号を定義します．通常，ファイル先頭からの通し番号です．
- name
  - ボーン名を定義します．
- axis
  - ボーンの基準座標系を定義します．ワールド座標系に対する各ボーン座標系の回転量が与えられます．axis キーワードに続き 3 つの軸周りの回転量を記述し，最後に"XYZ"といった識別子によって，各回転量に対応する回転軸と回転の合成順序を指定します．リスト 8 の例では，ワールド座標系 M に対するローカル座標系 M' は $M' = R_x(0)R_y(0)R_z(0)M$ によって決定します．

- direction
  - ボーンの初期方向を示す単位ベクトルを定義します．このベクトルはボーンローカル座標系における値です．そのため，ワールド座標系におけるベクトル値は axis 部で求まる回転行列の転置 M'^T を用いて，$d' = dM'^T$ と計算されます．
- length
  - ボーン長を定義します．
- dof
  - ボーンの自由度の情報を定義します．rx，ry，rz がそれぞれ X，Y，Z 軸周りの回転の自由度を示し，記述した順序に回転を合成します．リスト 8 の例では，ボーンの回転トランスフォームは $v' = vR_x(\theta_x)R_y(\theta_y)R_z(\theta_z)$ によって計算されます．なお，この回転量もボーンローカル座標系における値です．ワールド座標系における回転量を計算するためには axis 部で求まる回転行列 M' を用いて，$R_x(\theta_x)R_y(\theta_y)R_z(\theta_z)M'$ のように計算されます．
  - 回転成分を持たない固定ボーンの場合，dof 部は省略されます．
- limits
  - dof 部で記述された各自由度の回転可動域を，1 行ずつ定義します．リスト 8 の例では，3 つの回転自由度それぞれに -180～180 度の可動域を与えています（つまり制限無し）．
- bodymass
  - ボーンの質量を定義します．オプション項目なので省略可能です．
- cofmass
  - ボーンの重心位置を定義します．資料が見つからなかったため，記述方法などの詳細は不明です．オプション項目なので省略可能です．

ボーン階層構造情報: hierarchy

hierarchy 部ではボーンの親子接続関係を定義します．各ボーンに接続する子ボーンの情報を，1行ずつ「親ボーン 子ボーン1 子ボーン2 .....」の形式で記述します．ただし，1行目には必ず root ノードに接続する子ボーンの情報を記述します．リスト 8 の例では，root ノードに joint1 ボーンが接続することを定義しています．もし，joint1 ボーンが子ボーン cbone を持つ場合は，次の行に「joint1 cbone」と記述することになります．

AMCファイル

Acclaim 形式のモーションデータ情報は AMC ファイルに記述されます．まず :FULLY-SPECIFIEDキーワード（詳細は不明）を記述し，:DEGREES もしくは :RADIAN キーワードによって回転量の単位を定義します．

続いて，各時間フレームにおける姿勢情報を記述します．最初にフレーム番号を指定し，そのフレームにおけるルートもしくは各ボーンの姿勢情報を1行ずつ記述します．各行の数字の並びは，order もしくは dof における定義に対応します．例として，リスト 9 のモーションデータを 3 フレームに拡張した AMC ファイルを，以下のリスト 10 に示します．

```
# リスト10 3フレーム分のAMCモーションデータ
:FULLY-SPECIFIED
:DEGREES
1
root 0.0 0.0 0.0 0.0 0.0 0.0
joint1 0.0 0.0 0.0
2
root 0.0 1.0 0.0 0.0 0.0 0.0
joint1 0.0 0.0 0.0
3
root 0.0 2.0 0.0 0.0 0.0 0.0
joint1 0.0 0.0 0.0
```

サンプルプログラム

キャラクタのスケルトン構造で作成したスケルトン表示プログラムを拡張し，モーションデータの内容を表示するプログラムを作成します．サンプルプログラムは，VisualC++2005 + DirectX SDK Update October 2005の環境下で，それぞれ MFC と DXUT を用いて作成しています．

DXUT 版(VC++2005, 295kb)
MFC 版(VC++2005, 50kb)

基本的には両者に機能的な差はありませんが，操作インターフェイスは若干異なります．

DXUT 版では，DXUT に用意された CFirstPersonCamera カメラクラスを用いています．いずれかのマウスドラッグで注視点を移動します．また，テンキーの[8][2]で前後方向，[4][6]で左右方向，[9][3]で上下方向にカメラを平行移動します．また，[Load BVH]ボタンでBVHファイルのロード，[Load ASF/AMC]ボタンで ASF と AMC の 2 つのファイルを順にロード，[Save as BVH]ボタンで表示中のデータを BVH ファイルに保存できます．

MFC版では，左ボタンドラッグでカメラの回転移動，右ボタンドラッグでカメラの平行移動，中央ボタンドラッグでズームイン/アウトします．また，[File]メニューの[開く]でモーションデータファイルをロードする際に，"ファイルの種類"プルダウンリストから，ASF/AMC もしくは BVH 形式を選択します．表示中のデータは[File]メニューの[名前を付けて保存]で BVH 形式で保存できます．

サンプルプログラムの概要

BVH ファイルの HIERARCHY 部，もしくは ASF ファイルで定義されたスケルトン構造は，キャラクタのスケルトン構造で作成した CTreeNode，CJoint，CFigure クラスを拡張したクラス群を用いて管理します．また，BVH ファイルの MOTION 部もしくは AMC ファイルに記述されたモーションデータは，今回新たに作成した CMotionData クラスで管理します．なお，モーションデータファイルを解析/保存するための関数群は全て MotionFile.h, MotionFile.cpp 中に作成しており，ファイル形式に対応したいくつかの名前空間を定義しています．

主要なプログラムファイルの doxygen ドキュメントを以下に示します．

サンプルプログラムのdoxygenドキュメント

モーションデータクラス

CMotionData クラスは，ルートの 3 次元位置の時系列変化と，クォータニオン表現された各関節回転量の時系列変化を格納します．特に説明は必要ないと思われますので，クラス宣言の一部だけを以下に示し，実装部は説明を省略します．

```
 1 // モーションデータクラス
 2 class CMotionData
 3 {
 4 private:
 5   size_t m_nJoints;            // 関節数
 6   size_t m_nFrames;            // フレーム数
 7   D3DXVECTOR3 *m_pPosition;    // ルート位置時系列データ
 8   D3DXQUATERNION *m_pRotation; // 関節回転角度時系列データ
 9
10 public:
11   CMotionData(void);                          // デフォルトコンストラクタ
12   CMotionData(size_t frames, size_t joints); // 初期化子付コンストラクタ
13   CMotionData(const CMotionData &src);       // コピーコンストラクタ
14   virtual ~CMotionData(void);                // デストラクタ
15
16 public:
17   bool Initialize(size_t frames, size_t joints);   // 初期化関数
18   CMotionData& operator =(const CMotionData &src); // 代入演算子
19
20 private:
21   bool Allocate(size_t frames, size_t joints); // メモリ領域確保
22   void Destroy();                              // メモリ領域解放
23
24 public:
25   bool IsActive() const;    // データ利用可能状態
26   size_t NumFrames() const; // 総フレーム数の取得
27   size_t NumJoints() const; // 総関節ノード数の取得
```

```
28 |
29 | public:
30 |   bool SetRotation(size_t frm, size_t jid, const D3DXQUATERNION &q); // 回転量の設定
31 |   bool SetRotation(size_t frm, size_t jid, const D3DXMATRIX &m);     // 回転量の設定
32 |   bool SetPosition(size_t frm, const D3DXVECTOR3 &v);                // ルート位置の指定
33 |
34 |   D3DXQUATERNION GetRotation(size_t frm, size_t jid) const;          // 回転量の取得
35 |   D3DXMATRIX GetRotationMatrix(size_t frm, size_t jid) const;        // 回転量の取得
36 |   D3DXVECTOR3 GetPosition(size_t frm) const;                         // ルート位置の取得
37 |   D3DXMATRIX GetPositionMatrix(size_t frm) const;                    // ルート位置の取得
38 ! };
```

ファイル解析処理

ファイル内容を解析し，CFigure，CMotionData クラスインスタンスを生成します．ASF ファイルの解析では，units 部の angle の内容，root 部の axis と orientation の内容，bonedata 部の limits，bodymass，cofmass の内容を全て無視しています．

なお，本稿で説明したファイルフォーマットと照らし合わせれば解読は容易だと思いますのでコード自体の説明は省略します．ただ，ASF ファイルを読み込む際に change_skeletal_structure 関数を用いる理由については簡単に説明します．

```
 1   void change_skeletal_structure(CJoint *node, D3DXVECTOR3 &v, std::map &axes)
 2 + {
 3 |   static int ls_extent = 0;
 4 |   if (node->NumChildren() > 0)
 5 +   {
 6 |     for (size_t i = 0;i < node->NumChildren();++i)
 7 |       change_skeletal_structure(node->GetChild(i), node->GetOffset(), axes);
 8 !   }
 9 |   else
10 +   {
11 |     CJoint *ext = new CJoint(node->GetOffset(), node);
12 |     node->AddChild(ext);
13 |
14 |     char buf[6];
15 |     sprintf(buf, "ext%02d", ls_extent++);
16 |     ext->SetName(buf);
17 |
18 |     D3DXMATRIX m;
19 |     D3DXMatrixIdentity(&m);
20 |     axes[buf] = m;
21 !   }
22 |   node->SetOffset(v);
23 ! }
```

本サンプルプログラムでは，ASF ファイルに記述されたボーン情報を，あたかも関節ノード情報であるかのように解析しています．ただし，CJoint クラスは関節ノードに接続する親ボーンの長さと子ボーンの回転情報を保持する(Fig.4 (a))のに対し，ASF ファイルのボーン情報には自ボーンの長さと回転が記述されます(Fig.4 (b))．そのため，解析の過程ではオフセット情報が 1 階層ずれて対応付けられることになります．また，ASF 形式ではスケルトンの末端を示すノードが欠けています．そのため，change_skeletal_structure 関数では，関節ノードのオフセット情報を親関節ノードのオフセット情報で置き換えることで，対応付けのズレを修正します．さらに，スケルトンの末端を示す新しいノードも生成します．

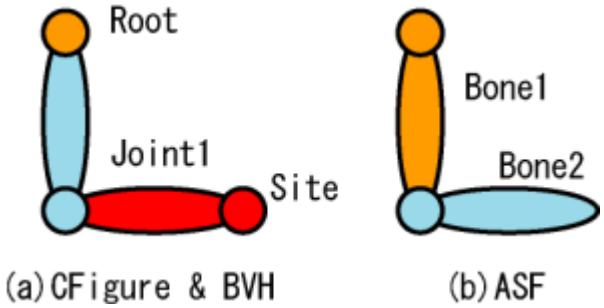

Fig.4 BVH と ASF のノードの扱いの差異

もちろん，CJoint クラスではなく，CBone クラスなどを作成すればよいのでしょうが，そうすると BVH ファイルを扱う場合に結局同じ問題が生じます．また，ASF ファイルの解析過程でオフセットの対応付けを考慮しようとすると，bonedata を解析

する段階で hierarchy の情報が必要となるため，ファイルを 2 パスで解析しなければならなくなります．他にスマートな解決方法を思いつかなかったので，本サンプルではこのような苦肉の策で対応しています．

↑

まとめ †

モーションキャプチャデータを記録するための代表的なファイルフォーマットである，BVH ファイルと ASF/AMC ファイル形式について解説し，簡単なファイル解析/表示プログラムを作成しました．個人的に配布できるモーションデータファイルを所有していませんので，参考ページなどに示した他サイトからファイルを入手して遊んでみてください．

また，本稿で説明した以外にも HTR 形式などのいくつかのファイル形式が存在するようですが，とりあえずこの 2 つを抑えておけば問題ないと思います．また，スケルトンデータを含まず，実際に体表面に付与したマーカー時系列の生データを格納する形式としては，C3D 形式が広く利用されているようです．興味のある方は調べてみるとよいでしょう．

本サンプルプログラムは，私自身もほとんど使い込んでいませんので，バグなど発見されましたら随時お知らせください．
