```
/**
***  BVHファイルの読み込・描画クラス
***  Copyright (c) 2004, Masaki OSHITA
**/


#include <fstream>
#include <string.h>

#ifdef WIN32
    #include <windows.h>
#endif

#include <gl/gl.h>
#include "mogl.h"

#include "BVH.h"



// コントラクタ
BVH::BVH()
{
    motion = NULL;
    Clear();
}

// コントラクタ
BVH::BVH( const char * bvh_file_name )
{
    motion = NULL;
    Clear();

    Load( bvh_file_name );
}

// デストラクタ
BVH::~BVH()
{
    Clear();
}


// 全情報のクリア
void  BVH::Clear()
{
    int  i;
    for ( i=0; i<channels.size(); i++ )
        delete  channels[ i ];
    for ( i=0; i<joints.size(); i++ )
        delete  joints[ i ];
    if ( motion != NULL )
        delete  motion;

    is_load_success = false;

    file_name = "";
    motion_name = "";

    num_channel = 0;
    channels.clear();
    joints.clear();
    joint_index.clear();

    num_frame = 0;
    interval = 0.0;
    motion = NULL;
}



//
//  BVHファイルのロード
//
void  BVH::Load( const char * bvh_file_name )
{
    #define  BUFFER_LENGTH  1024*4

    ifstream  file;
    char      line[ BUFFER_LENGTH ];
    char *    token;
    char      separater[] = " :,¥t";
    vector< Joint * >   joint_stack;
    Joint *   joint = NULL;
    Joint *   new_joint = NULL;
    bool      is_site = false;
    double    x, y ,z;
    int       i, j;

    // 初期化
    Clear();

    // ファイルの情報（ファイル名・動作名）の設定
    file_name = bvh_file_name;
    const char *  mn_first = bvh_file_name;
    const char *  mn_last = bvh_file_name + strlen( bvh_file_name );
    if ( strrchr( bvh_file_name, '¥¥' ) != NULL )
        mn_first = strrchr( bvh_file_name, '¥¥' ) + 1;
    else if ( strrchr( bvh_file_name, '/' ) != NULL )
        mn_first = strrchr( bvh_file_name, '/' ) + 1;
    if ( strrchr( bvh_file_name, '.' ) != NULL )
        mn_last = strrchr( bvh_file_name, '.' );
    if ( mn_last < mn_first )
        mn_last = bvh_file_name + strlen( bvh_file_name );
    motion_name.assign( mn_first, mn_last );

    // ファイルのオープン
    file.open( bvh_file_name, ios::in );
    if ( file.is_open() == 0 )  return; // ファイルが開けなかったら終了

    // 階層情報の読み込み
    while ( ! file.eof() )
    {
        // ファイルの最後まできてしまったら異常終了
        if ( file.eof() )  goto bvh_error;

        // １行読み込み、先頭の単語を取得
        file.getline( line, BUFFER_LENGTH );
        token = strtok( line, separater );

        // 空行の場合は次の行へ
        if ( token == NULL )  continue;

```

```
124         // 関節ブロックの開始
125         if ( strcmp( token, "{" ) == 0 )
126         {
127             // 現在の関節をスタックに積む
128             joint_stack.push_back( joint );
129             joint = new_joint;
130             continue;
131         }
132         // 関節ブロックの終了
133         if ( strcmp( token, "}" ) == 0 )
134         {
135             // 現在の関節をスタックから取り出す
136             joint = joint_stack.back();
137             joint_stack.pop_back();
138             is_site = false;
139             continue;
140         }
141 
142         // 関節情報の開始
143         if ( ( strcmp( token, "ROOT" ) == 0 ) ||
144              ( strcmp( token, "JOINT" ) == 0 ) )
145         {
146             // 関節データの作成
147             new_joint = new Joint();
148             new_joint->index = joints.size();
149             new_joint->parent = joint;
150             new_joint->has_site = false;
151             new_joint->offset[0] = 0.0;  new_joint->offset[1] = 0.0;  new_joint->offset[2] = 0.0;
152             new_joint->site[0] = 0.0;  new_joint->site[1] = 0.0;  new_joint->site[2] = 0.0;
153             joints.push_back( new_joint );
154             if ( joint )
155                 joint->children.push_back( new_joint );
156 
157             // 関節名の読み込み
158             token = strtok( NULL, "" );
159             while ( *token == ' ' )  token ++;
160             new_joint->name = token;
161 
162             // インデックスへ追加
163             joint_index[ new_joint->name ] = new_joint;
164             continue;
165         }
166 
167         // 末端情報の開始
168         if ( ( strcmp( token, "End" ) == 0 ) )
169         {
170             new_joint = joint;
171             is_site = true;
172             continue;
173         }
174 
175         // 関節のオフセット or 末端位置の情報
176         if ( strcmp( token, "OFFSET" ) == 0 )
177         {
178             // 座標値を読み込み
179             token = strtok( NULL, separater );
180             x = token ? atof( token ) : 0.0;
181             token = strtok( NULL, separater );
182             y = token ? atof( token ) : 0.0;
183             token = strtok( NULL, separater );
184             z = token ? atof( token ) : 0.0;
185 
186             // 関節のオフセットに座標値を設定
187             if ( is_site )
188             {
189                 joint->has_site = true;
190                 joint->site[0] = x;
191                 joint->site[1] = y;
192                 joint->site[2] = z;
193             }
194             else
195             // 末端位置に座標値を設定
196             {
197                 joint->offset[0] = x;
198                 joint->offset[1] = y;
199                 joint->offset[2] = z;
200             }
201             continue;
202         }
203 
204         // 関節のチャンネル情報
205         if ( strcmp( token, "CHANNELS" ) == 0 )
206         {
207             // チャンネル数を読み込み
208             token = strtok( NULL, separater );
209             joint->channels.resize( token ? atoi( token ) : 0 );
210 
211             // チャンネル情報を読み込み
212             for ( i=0; i<joint->channels.size(); i++ )
213             {
214                 // チャンネルの作成
215                 Channel *  channel = new Channel();
216                 channel->joint = joint;
217                 channel->index = channels.size();
218                 channels.push_back( channel );
219                 joint->channels[ i ] = channel;
220 
221                 // チャンネルの種類の判定
222                 token = strtok( NULL, separater );
223                 if ( strcmp( token, "Xrotation" ) == 0 )
224                     channel->type = X_ROTATION;
225                 else if ( strcmp( token, "Yrotation" ) == 0 )
226                     channel->type = Y_ROTATION;
227                 else if ( strcmp( token, "Zrotation" ) == 0 )
228                     channel->type = Z_ROTATION;
229                 else if ( strcmp( token, "Xposition" ) == 0 )
230                     channel->type = X_POSITION;
231                 else if ( strcmp( token, "Yposition" ) == 0 )
232                     channel->type = Y_POSITION;
233                 else if ( strcmp( token, "Zposition" ) == 0 )
234                     channel->type = Z_POSITION;
235             }
236         }
237 
238         // Motionデータのセクションへ移る
239         if ( strcmp( token, "MOTION" ) == 0 )
240             break;
241     }
242 
243 
244     // モーション情報の読み込み
245     file.getline( line, BUFFER_LENGTH );
246     token = strtok( line, separater );
```

```
247     if ( strcmp( token, "Frames" ) != 0 )  goto bvh_error;
248     token = strtok( NULL, separater );
249     if ( token == NULL )  goto bvh_error;
250     num_frame = atoi( token );
251 
252     file.getline( line, BUFFER_LENGTH );
253     token = strtok( line, ":" );
254     if ( strcmp( token, "Frame Time" ) != 0 )  goto bvh_error;
255     token = strtok( NULL, separater );
256     if ( token == NULL )  goto bvh_error;
257     interval = atof( token );
258 
259     num_channel = channels.size();
260     motion = new double[ num_frame * num_channel ];
261 
262     // モーションデータの読み込み
263     for ( i=0; i<num_frame; i++ )
264     {
265         file.getline( line, BUFFER_LENGTH );
266         token = strtok( line, separater );
267         for ( j=0; j<num_channel; j++ )
268         {
269             if ( token == NULL )
270                 goto bvh_error;
271             motion[ i*num_channel + j ] = atof( token );
272             token = strtok( NULL, separater );
273         }
274     }
275 
276     // ファイルのクローズ
277     file.close();
278 
279     // ロードの成功
280     is_load_success = true;
281 
282     return;
283 
284 bvh_error:
285     file.close();
286 }
287 
288 
289 
290 //
291 //  BVH骨格・姿勢の描画関数
292 //
293 
294 
295 // 指定フレームの姿勢を描画
 H3 void  BVH::RenderFigure( int frame_no, float scale )
297 {
298     // BVH骨格・姿勢を指定して描画
299     RenderFigure( joints[ 0 ], motion + frame_no * num_channel, scale );
300 }
301 
302 
303 // 指定されたBVH骨格・姿勢を描画（クラス関数）
 H3 void  BVH::RenderFigure( const Joint * joint, const double * data, float scale )
305 {
306     glPushMatrix();
307 
308     // ルート関節の場合は平行移動を適用
309     if ( joint->parent == NULL )
310     {
311         glTranslatef( data[ 0 ] * scale, data[ 1 ] * scale, data[ 2 ] * scale );
312     }
313     // 子関節の場合は親関節からの平行移動を適用
314     else
315     {
316         glTranslatef( joint->offset[ 0 ] * scale, joint->offset[ 1 ] * scale, joint->offset[ 2 ] * scale );
317     }
318 
319     // 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）
320     int  i, j;
321     for ( i=0; i<joint->channels.size(); i++ )
322     {
323         Channel *  channel = joint->channels[ i ];
324         if ( channel->type == X_ROTATION )
325             glRotatef( data[ channel->index ], 1.0f, 0.0f, 0.0f );
326         else if ( channel->type == Y_ROTATION )
327             glRotatef( data[ channel->index ], 0.0f, 1.0f, 0.0f );
328         else if ( channel->type == Z_ROTATION )
329             glRotatef( data[ channel->index ], 0.0f, 0.0f, 1.0f );
330     }
331 
332     // リンクを描画
333     // 関節座標系の原点から末端点へのリンクを描画
334     if ( joint->children.size() == 0 )
335     {
336         RenderBone( 0.0f, 0.0f, 0.0f, joint->site[ 0 ] * scale, joint->site[ 1 ] * scale, joint->site[ 2 ] * scale );
337     }
338     // 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
339     if ( joint->children.size() == 1 )
340     {
341         Joint *  child = joint->children[ 0 ];
342         RenderBone( 0.0f, 0.0f, 0.0f, child->offset[ 0 ] * scale, child->offset[ 1 ] * scale, child->offset[ 2 ] * scale );
343     }
344     // 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
345     if ( joint->children.size() > 1 )
346     {
347         // 原点と全関節への接続位置への中心点を計算
348         float  center[ 3 ] = { 0.0f, 0.0f, 0.0f };
349         for ( i=0; i<joint->children.size(); i++ )
350         {
351             Joint *  child = joint->children[ i ];
352             center[ 0 ] += child->offset[ 0 ];
353             center[ 1 ] += child->offset[ 1 ];
354             center[ 2 ] += child->offset[ 2 ];
355         }
356         center[ 0 ] /= joint->children.size() + 1;
357         center[ 1 ] /= joint->children.size() + 1;
358         center[ 2 ] /= joint->children.size() + 1;
359 
360         // 原点から中心点へのリンクを描画
361         RenderBone( 0.0f, 0.0f, 0.0f, center[ 0 ] * scale, center[ 1 ] * scale, center[ 2 ] * scale );
362 
363         // 中心点から次の関節への接続位置へのリンクを描画
364         for ( i=0; i<joint->children.size(); i++ )
365         {
366             Joint *  child = joint->children[ i ];
367             RenderBone( center[ 0 ] * scale, center[ 1 ] * scale, center[ 2 ] * scale,
368                 child->offset[ 0 ] * scale, child->offset[ 1 ] * scale, child->offset[ 2 ] * scale );
369         }
```

```
370     }
371
372     // 子関節に対して再帰呼び出し
373     for ( i=0; i<joint->children.size(); i++ )
374     {
375         RenderFigure( joint->children[ i ], data, scale );
376     }
377
378     glPopMatrix();
379 }
380
381
382 // BVH骨格の１本のリンクを描画（クラス関数）
 H3 void  BVH::RenderBone( float x0, float y0, float z0, float x1, float y1, float z1 )
384 {
385     // ２点を結ぶ円柱を描画
386     modgSolidPipe( x0, y0, z0, x1, y1, z1, 0.01, 8, 3 );
387 }
388
389
390
391 // End of BVH.cpp
392
```