User's Guide of Polylib

Polygon Management Library

Ver. 4.0.0

Advanced Visualization Research Team
Advanced Institute for Computational Science
RIKEN

7-1-26, Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo, 650-0047, Japan

http://www.aics.riken.jp/

March 2016

Release

Edition	4.0.0	2016-03-25
	3.1.0	2013-11-14
	3.0.0	2013-09-16
	2.6.8	2013-07-20
	2.6	2013-06-24
	2.2	2012-11-27
	2.1	2012-04-31
	2.0.0	2010-06-30
	1.0.0	2010-02-26

COPYRIGHT

目次

第 1 章

Polylib の概要

本ユーザーガイドでは，ポリゴン要素を管理するライブラリについて，その機能と利用方法を説明します．

1.1　概要

Polygon Management Library (以下，Polylib) は，ポリゴンデータを保持・管理するためのクラスライブラリです．クラスライブラリの詳細については，「リファレンスマニュアル」を参照してください．

1.2　Polylib の機能

Polylib の主な機能を以下に列挙します．

- 初期化ファイルを利用した STL ファイルの読み込み (Ver.2.0.0 追加機能)
  - 初期化ファイルに記述されたポリゴングループ階層構造，および STL ファイルを読み込み，オンメモリに管理します．
- ポリゴンデータのグルーピング
  - 読み込んだポリゴンデータを STL ファイル単位にグルーピングして管理します．複数のポリゴングループをまとめたグループを作成するなどの，階層的なグループ管理が可能です．グルーピングの設定は初期化ファイルに記述します．
- ポリゴンデータの検索
  - 読み込み済のポリゴンデータについて，指定された領域内に含まれるポリゴンを検索します．検索対象のポリゴンデータは，Polylib 管理下のポリゴン全体や，任意のポリゴングループなどの指定が可能です．
- 並列計算環境下でのポリゴンデータの分散
  - マスターランクで読み込んだポリゴンデータを，領域分割情報に基づき各ランクに配信します．
- ポリゴンデータの移動 (Ver.2.0.0 追加機能)
  - 時間発展計算実行中に，ユーザプログラム側で定義されたポリゴン頂点座標移動関数に基づきポリゴンデータの移動を行います．
  - 並列計算環境下では，隣接ランク領域へ移動したポリゴン情報をランク間でやりとりします．
- 移動関数登録 (Ver.4.0.0 追加機能)
  - ポリゴングループのインスタンスに対して任意の移動関数を登録可能です．
- ポリゴンデータの再読み込み (Ver.2.0.0 追加機能)
  - 一時保存処理により保存されたファイルを再読み込みします．
  - 並列計算環境下での再読み込みは，マスターノードでの集約読み込みと，各ランクでの分割読み込みが選択できます．
- 実数型の選択 (Ver.4.0.0 追加機能)
  - コンパイルオプションにより単精度／倍精度が, 選択できます．
- 逐次処理／並列処理の選択 (Ver.4.0.0 追加機能)
  - コンパイルオプションにより逐次処理／並列処理が, 選択できます．
- autoconf & automake 対応　(Ver.3.0.0 追加機能)
  - configure スクリプト作成に autotools (autoconf automake) を用いました．
- 長田パッチフォーマット対応 (Ver.4.0.0 追加機能)
  - 対応フォーマットに長田パッチを追加しました．
- 1 プロセス複数領域対応 (Ver.4.0.0 追加機能)

  - 並列実行時に 1 プロセスあたり複数の担当領域を管理出来るようにしました.

- ポリゴングループ属性、ポリゴン属性の追加 (Ver.4.0.0 追加機能)
  - 任意の属性が設定可能です.

- Fortran 対応 (Ver.4.0.0 追加機能)
  - Fortran インターフェースを追加しました.

- 自動テスト (Ver.4.0.0 追加機能)
  - 環境の構築が正常に行われたかどうか確認するため、および、レベルダウンを防止するため自動テストの機能を追加しました.
  - configure，make 後に make_check を行うことによりテストを実行できます.（バッチ処理、stagein/stageout は未対応です.）

1.3　動作環境

以下の環境下で動作確認済です.

- 開発 OS：Ubuntu14.0.4 (64bit)
- 開発言語：C++(ライブラリ本体)，C/Fortran(インターフェーステスト用)
- 開発コンパイラ：g++，gcc，gfortran 4.8.4, Intel C++ Compiler XE 2013 SP1
- 並列ライブラリ：OpenMPI 1.6.5
- TextParser ライブラリ：TextParser Version1.6.5
- 長田パッチライブラリ：Npatch Version 1.0.0

1.4　ライセンス

Polylib は，バージョンにより以下の 2 つのライセンスの適用となります.

- Version 1.0 ~ 2.x
  理化学研究所 VCAD ライセンス `http://vcad-hpsv.riken.jp/`

- Version 3.0 以降
  修正 BSD ライセンス（２条項）

1.5　リポジトリ

公開リポジトリは以下になります.

```
https://github.com/avr-aics-riken/Polylib
```

1.6　インストール

Polylib の環境構築には，TextParser ライブラリが必要です． また，長田パッチを利用する場合は，Npatch ライブラリが必要です．各ライブラリの「利用者マニュアル」および「INSTALL ファイル」の説明に従い事前にインストールしてください．Polylib のインストールは「INSTALL ファイル」の説明に従いインストールを行って下さい.

第 2 章

プログラム構造

本章では，Polylib プログラムのクラス構成，データ構造，入出力ファイルなどについて説明します．

2.1　クラス構成

以下に Polylib のクラス図概要を示します.

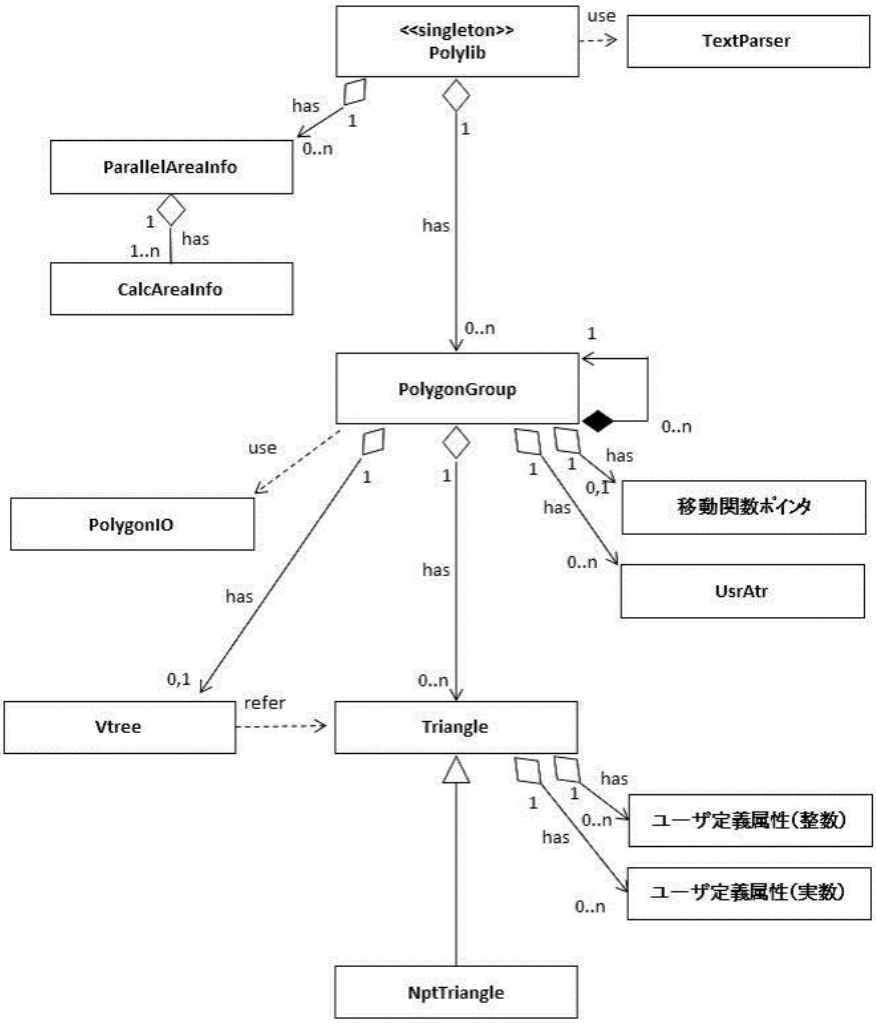

図 2.1　主要クラス図

各クラスの説明は表 2.1 の通りです.

表 2.1　主要クラス一覧

クラス名	概要
Polylib	Polylib 本体です．本クラス内部にポリゴングループ階層構造，ポリゴン情報を保持します．本クラスは singleton クラスであり，1 プロセス内に 1 インスタンスのみ存在します．
TextParser	Polylib 初期化ファイルを load/save するためのユーティリティクラスです．
ParallelAreaInfo	並列実行時の各ランクの計算領域情報（複数可）を管理します．
CalcAreaInfo	1 計算領域情報のクラスです．
PolygonGroup	三角形ポリゴン集合をグルーピングして管理するためのクラスです．ポリゴングループ同士の階層的な包含関係も表現します．
PolygonIO	ジオメトリデータファイルを load/save するためのユーティリティクラスです．
UsrAtr	ポリゴングループの任意属性データのクラスです．key(文字列) と value(文字列) のペアで管理します．
Vtree	KD 木データ構造クラスです．
Triangle	三角形ポリゴンクラスです．3 頂点座標へのポインタ，法線ベクトル，面積を保持します．
NptTriangle	長田パッチのクラスです．Triangle の情報に加えて長田パッチのパラメータ（制御点）情報を保持します．

2.2　データ構造

2.2.1　ポリゴングループの管理構造

ポリゴングループの保持・管理は，Polylib クラスのメンバ変数である，std::vector<PolygonGroup> m_pg_list で行います．この vector コンテナはポリゴングループ階層構造の最上位の PolygonGroup インスタンスを保持します．PolygonGroup クラスは，メンバ変数 std::vector<PolygonGroup*> m_children により，PolygonGroup インスタンス同士の階層構造を保持します．

PolygonGroup は複数の子要素を持つことができますが，親要素は最大で 1 つです．（親要素数がゼロならば最上位の PolygonGroup です）また，階層構造最下位の PolygonGroup のみ，STL/NPT ファイルから読み込んだ三角形ポリゴン情報を保持します．

これらの階層構造については，ユーザが作成する Polylib 初期化ファイルに記述されており，Polylib は初期化処理時にこのファイルを読み込むことで，グループ階層構造，およびポリゴンデータをオンメモリに構築し，管理します．

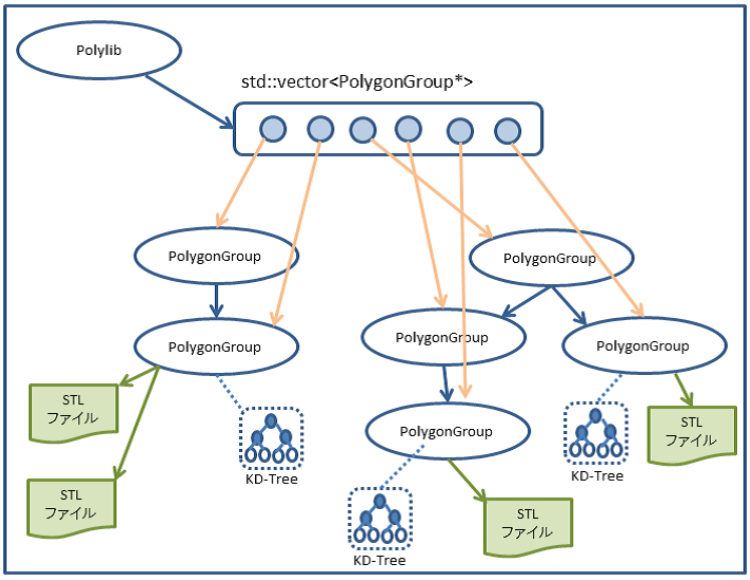

図 2.2　ポリゴングループの管理構造

2.2.2　ポリゴンデータの管理構造

三角形ポリゴンの保持･管理は，PolygonGroup クラスのメンバ変数である，`std::vector<Triangle*> *m_tri_list` で行います．
この vector コンテナには，STL/NPT ファイルから読み込んだ三角形ポリゴン情報を Triangle クラスのインスタンスとして生成して登録します．

また，三角形ポリゴンを高速に検索するために，三角形ポリゴンのバウンディングボックスベースで包含判定を行う KD-Tree 構造を PolygonGroup に実装しています．KD 木のリーフ要素である三角形ポリゴン情報は，`PolygonGroup::m_tri_list` コンテナに格納された各インスタンスへのポインタです．

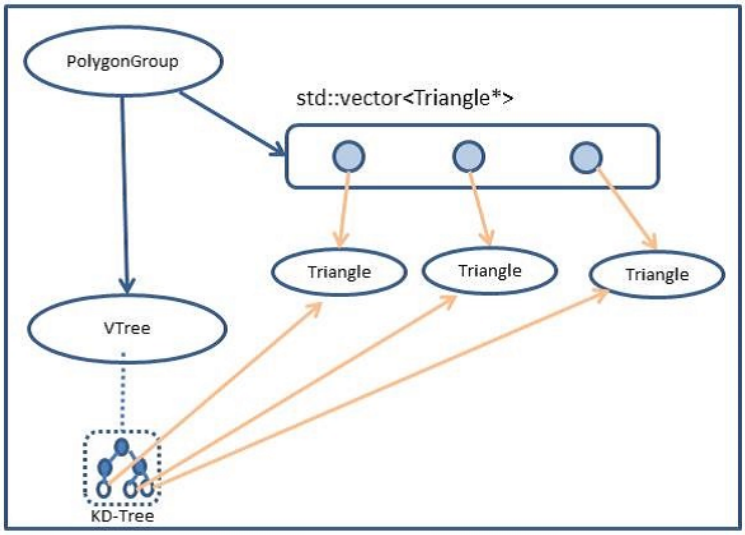

図 2.3　ポリゴンデータの管理構造

2.3　入力ファイル

2.3.1　Polylib 初期化ファイル

Polylib 初期化ファイルは，TextParser ライブラリでパース可能な形式のテキストファイルです．
TextPaser 記述方式のについては TextParser ライブラリのマニュアルを参照してください．
Polylib 初期化ファイルは，デフォルトではカレントディレクトリに存在する polylib_config.tp を読み込みますが，任意のファイル名をデータロード API 引数で指定可能です．

記述例を図 2.4 に示します．この例では，STL ファイルをロードしていますが，Npt ファイルも同様に記述します．

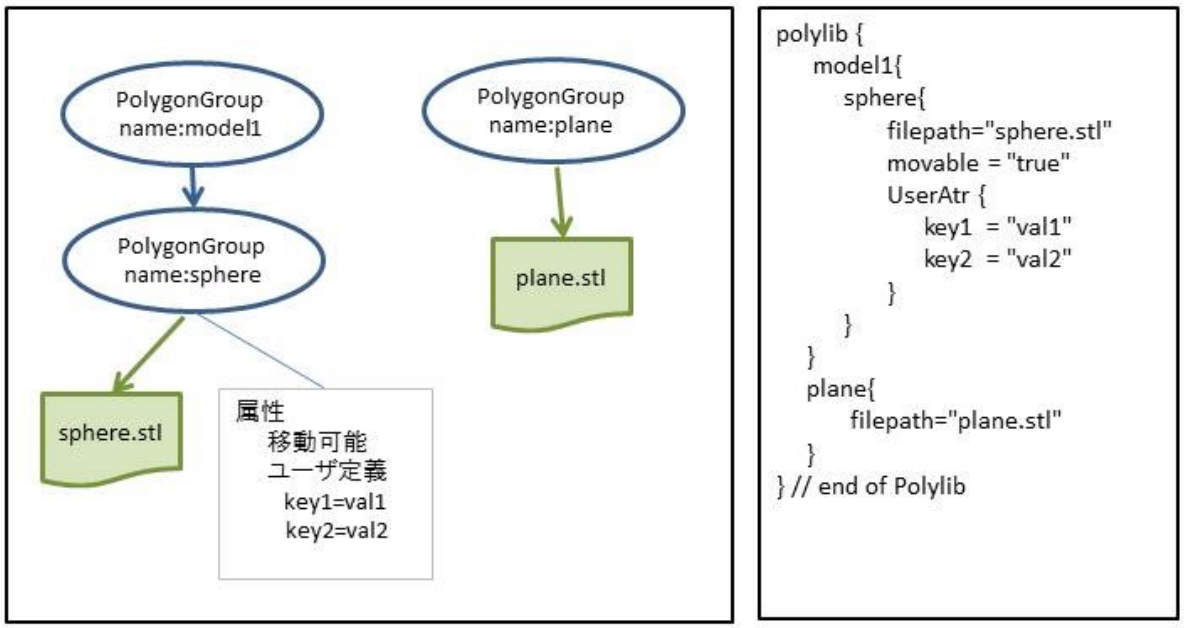

図 2.4　初期化ファイルの例

要素を {} で囲むグループラベルについて表 2.2 に説明します．

表 2.2　グループラベルの説明

ラベル名	説明
Polylib	Polylib 初期化ファイルにおけるルートラベルです．
UserAtr	PolygonGroup のユーザ定義属性の開始ラベルです．UserAtr は Polylib 内の予約語ですので，グループの名前としては使えません．UserAtr{ } 内に キー = 値で属性を設定します．キー，値ともに任意の文字列です．
polygon_group_name	ポリゴングループ名称を記述します．子要素としてポリゴングループを階層的に持つことができます．名称は同じ階層内の同一レベルにおいて一意でなければなりません．

ラベル=値形式のラベルについて表 2.3 に説明します．

表 2.3　ラベルの説明

ラベル名	説明
filepath	当該ポリゴングループにひもづく STL/NPT ファイルのパス．カレントディレクトリからの相対パスまたは絶対パスで指定する．階層最下位のポリゴングループでのみ指定が可能．
movable	当該ポリゴングループが move() メソッドにより移動するかどうかを指定する．value には“ture” もしくは“false” を指定する．ポリゴン形状が存在するポリゴングループのみ意味を持つ．指定しない場合、“false”とみなされます．

2.3.2　STL ファイル

初期化ファイルで指定されたファイル名の STL ファイルを読み込みます．入力となる STL ファイルはアスキー形式，バイナリ形式いずれでもかまいません．拡張子は”stl”であれば形式を自動判別して読み込みます．

バイナリ形式 STL ファイルの場合，各三角形毎に存在する 2 バイト未使用領域を使ってユーザ定義値を設定することが可能です．値は int Triangle::m_exid に設定されます．

2.3.3　NPT ファイル

初期化ファイルで指定されたファイル名の NPT ファイルを読み込みます．入力となる NPT ファイルはアスキー形式，バイナリ形式いずれでもかまいません．拡張子は“npt”であれば形式を自動判別して読み込みます．

2.4　出力ファイル

本節では，Polylib::save() などのデータセーブ系 API で出力されたデータファイルについて説明します．

2.4.1　Polylib 初期化ファイル

データセーブ時のポリゴングループの階層構造と，保存した STL/NPT ファイル名を polylib 初期化ファイル形式で保存します．保存時のファイル命名規則は以下の通りです．

```
polylib_config_{ユーザ指定文字列}.tp
```

ユーザ指定文字列はデータセーブ系 API 引数で指定可能です．無指定の場合，保存時のタイムスタンプを yyyymmddHHmmss 形式で設定します．

2.4.2　ポリゴンファイル

データセーブ時のポリゴン情報を保存します．保存時のファイル命名規則は以下の通りです．

```
{ポリゴングループ名称フルパス}_{ユーザ指定文字列}.{stl_a|stl_b|npt_a|npt_b}
```

ポリゴングループ名称フルパスとは，階層最上位のポリゴングループ名称から，当該グループ名称までを'_' でつなげたものです．ユーザ指定文字列はデータセーブ系 API 引数で指定可能です．無指定の場合，保存時のタイムスタンプを yyyymmddHHmmss 形式で設定します．

ファイル拡張子はデータセーブ系 API で指定した保存ファイル形式に基づき設定されます．

なお，データロード時に指定した polylib 初期化ファイルにおいて，複数の STL/NPT ファイルを指定したポリゴングループについてデータセーブを行うと，ポリゴン情報は 1 つの STL/NPT ファイルに纏めて出力されます．

第 3 章

API 利用方法

本章では，Polylib の主な API の利用方法を説明します.

3.1　単一プロセス版の主な API の利用方法

本節では単一プロセス版 Polylib の主な API 利用方法を，API 呼び出し順に沿って説明します．
単一プロセス版 Polylib の API を利用する手順は図 3.1 の通りです．

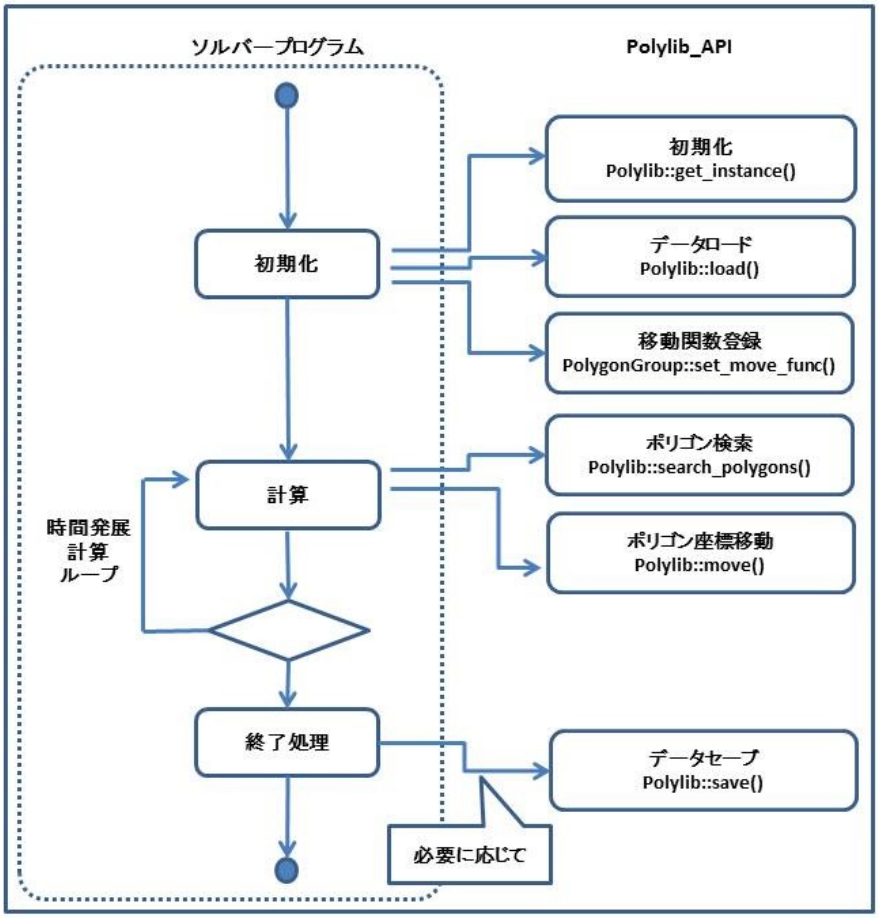

図 3.1　API 利用手順 (単一プロセス版)

3.1.1　初期化 API

Polylib インスタンスの生成

```
static Polylib* Polylib::get_instance();
```

Polylib は singleton クラスであるため，ユーザプログラム中から Polylib インスタンスを明示的に生成する必要はありません．Static メソッドである Polylib::get_instance() を呼び出すことで，プロセス内唯一の Polylib インスタンスが返却されます．

また，Polylib::get_instance() により返却されたインスタンスをユーザプログラムで明示的に消去する必要はありません．インスタンスはプロセス終了時に自動的に消去されます．

3.1.2　データロード API

```
POLYLIB_STAT Polylib::load(
        const std::string   config_name = "polylib_config.tp",
        PL_REAL         scale = 1.0
);
```

引数 config_name で指定された Polylib 初期化ファイルを読み込み，そこに記述された内容に基づきポリゴングループ階層構造をオンメモリに生成します．そして最下層ポリゴングループに指定された STL/NPT ファイルを読み込み，三角形ポリゴン情報のインスタンスの生成と，ポリゴン検索用 KD 木の生成を行います．
引数 config_name が指定されなかった場合，デフォルト初期化ファイル名である”polylib_config.tp”をカレントディレクトリから読み込みます．
引数 scale が指定されなかった場合，scale は 1.0 となり縮尺なしで読み込みます．

3.1.3　移動関数登録

```
POLYLIB_STAT PolygonGroup::set_move_func(
    void (*func)(PolygonGroup*,PolylibMoveParams*)
);
```

時間発展に伴い，ポリゴンを移動させる関数を登録します．移動が必要なすべてのポリゴングループのインスタンスに対して設定を行います．

3.1.4　検索 API

```
POLYLIB_STAT Polylib::search_polygons(
        std::vector<Triangle*>&  tri_list,
        const std::string&       group_name,
        const Vec3<PL_REAL>&  min_pos,
        const Vec3<PL_REAL>&  max_pos,
        const bool               every
) const;
```

ポリゴングループ名 group_name で指定されたグループ階層構造下から，位置ベクトル min_pos と max_pos により指定される矩形領域に含まれる三角形ポリゴンを検索します．引数 group_name はポリゴングループ名称フルパスで指定します．
ポリゴングループ名称フルパスとは，階層最上位のポリゴングループ名称から，当該グループ名称までを‘/’ でつなげたものです．
たとえば，階層最上位のポリゴングループ名が”group_A”でその直下にある”group_B”内のポリゴンを検索する場合，引数 group_name に指定する文字列は以下の通りです．

```
"group_A/group_B"
```

引数 every の指定方法は以下の通りです．

- ture: 3 頂点が全て指定領域内に含まれる三角形を検索
- false: 一部でも指定領域と交差する三角形を検索

std::vector で返却されたポリゴンリストの要素である Triangle 型ポインタの指し示す Triangle インスタンスを消去してはいけません．

指定点に最も近い三角形ポリゴンの検索

```
POLYLIB_STAT Polylib::search_nearest_polygon(
        Triangle*&              tri,
        const std::string&        group_name,
        const Vec3<PL_REAL>&  pos
) const;
```

ポリゴングループ名 group_name で指定されたグループ階層構造下から，位置ベクトル pos により指定される点に最も近い三角形ポリゴンを検索します．

3.1.5　ポリゴン座標移動 API

```
POLYLIB_STAT Polylib::move(PolylibMoveParams& param);
```

ポリゴングループ毎に登録された移動関数に基づき，ポリゴンの頂点座標を変更します．PolylibMoveParams クラス定義は以下の通りです．

```
class PolylibMoveParams {
public:
        int m_current_step;           /* 現在の計算ステップ番号     */
        int m_next_step;              /* 移動後の計算ステップ番号    */
        PL_REAL m_delta_t;            /* 計算ステップあたりの時間変異*/
        PL_REAL m_params[10];         /* ユーザ定義パラメータ（任意）*/
};
```

move メソッドの実装方法の実際については，後述のチュートリアルを参照してください．

3.1.6　データセーブ API

```
POLYLIB_STAT Polylib::save(
        std::string&            config_name_out,
        const std::string&      file_format,
        std::string             extend = ""
);
```

本 API 呼び出し時点でのグループ階層構造を Polylib 初期化ファイル形式に，ポリゴン情報を指定した形式のファイ

ルに出力します．引数 config_name_out は出力引数で，保存された Polylib 初期化ファイル名が設定されます．
引数 format は，次のように保存する STL/NPT ファイルの形式を指定します．

- ”stl_a”の場合，STL のアスキー形式で保存します．
- ”stl_b”の場合，STL のバイナリ形式で保存します．
- ”npt_a”の場合，NPT のアスキー形式で保存します．
- ”npt_b”の場合，NPT のバイナリ形式で保存します．

引数 extend は，保存するファイル名に任意の文字列を付加します．ファイル名の書式については，2.4 章を参照してください．

3.2　MPI 版の主な API の利用方法

本節では MPI 版 Polylib の主な API 利用方法を，API 呼び出し順に沿って説明します．
MPI 版 Polylib の API を利用する手順は下図の通りです．

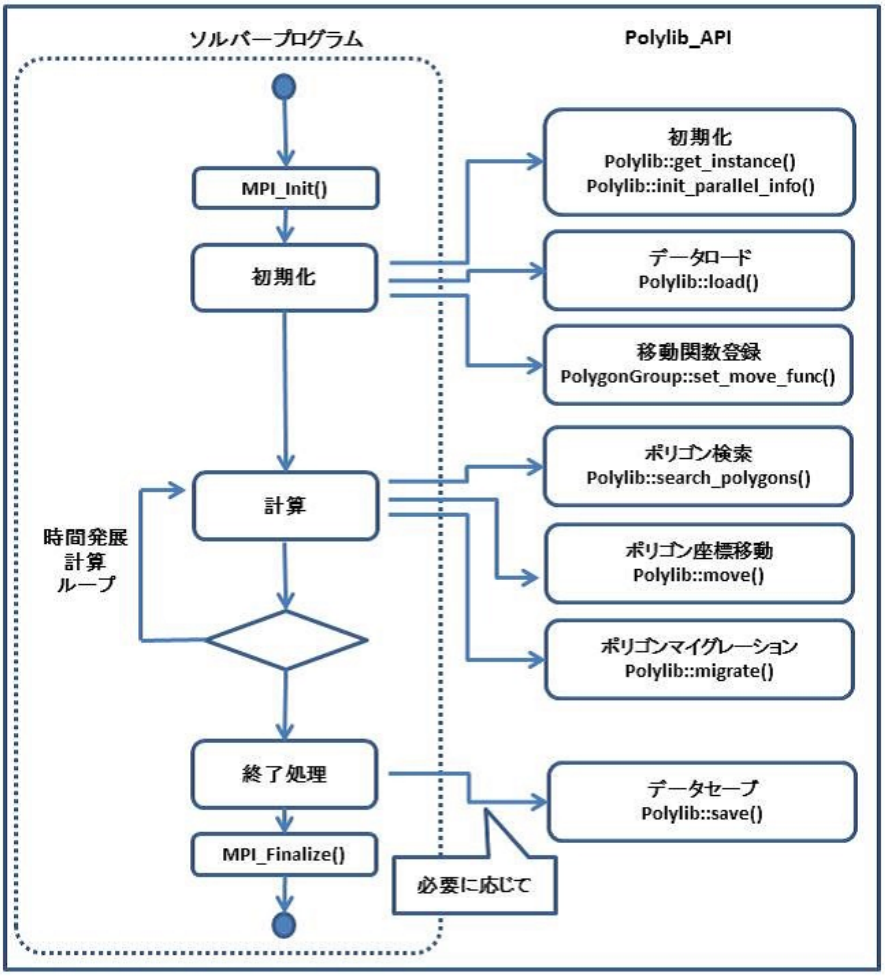

図 3.2　API 利用手順 (MPI 版)

追加で呼びだすメソッド以外，呼びだすメソッドのインターフェースは単一プロセス版を同じです．単一プロセス版から追加されたメソッドのみ以下に説明します．

3.2.1　初期化 API

並列計算情報の設定

```
POLYLIB_STAT MPIPolylib<PL_REAL>::init_parallel_info(
        MPI_Comm     comm,
        PL_REAL        bpos[3],
        unsigned int bbsize[3],
        unsigned int gcsize[3],
        PL_REAL        dx[3]
);
```

並列計算特有の値を MPIPolylib に設定します．各引数の意味は以下の通りです．

- comm MPI コミュニケータ
- bpos[3] 自ランクが受け持つ計算領域の基準座標
- bbsize[3] 自ランクが受け持つ計算領域のボクセル数
- gcsize[3] 自ランクが受け持つガイドセルのボクセル数
- dx[3] ボクセル 1 辺の長さ

引数で設定された自ランク計算領域情報は，本 API 内部処理により MPI 通信により全ランクへ配信されます．
計算領域に関する各値の意味を下図に示します．

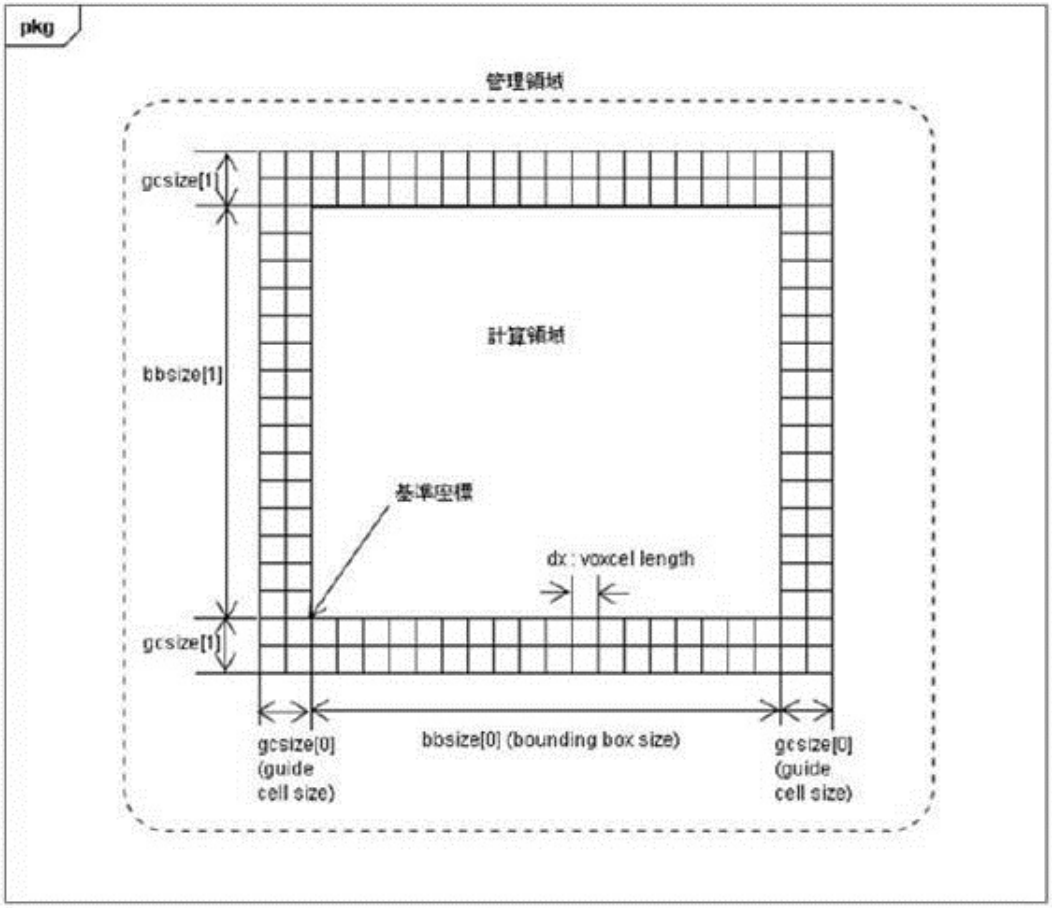

図 3.3　MPIPolylib::init_parallel_info() 各引数の意味

3.2.2　ポリゴンマイグレーション API

```
POLYLIB_STAT Polylib::migrate();
```

前述のメソッド Polylib::move により隣接ランク計算領域に移動した三角形ポリゴン情報を，隣接ランク同士で MPI 通信により送受信します．

MPI 版 PolylibAPI 利用の注意点

- Polylib の内部では，MPI の初期化・終了処理は行いません．ソルバー側で MPI_Init() および MPI_Finalize() を呼び出す必要があります．
- Polylib の API のうち，MPI 通信を伴う API は全ランクで同時実行される必要があります．MPI 通信を伴う利用者様側で使われる主要 API は以下の通りです．
  - Polylib::init_parallel_info();
  - Polylib::load();
  - Polylib::save();
  - Polylib::migrate();
  - PolygonGroup::get_group_num_global_tria();
  - PolygonGroup::get_group_num_global_area();
  - PolygonGroup::get_polygons_reduce_atrI();
  - PolygonGroup::get_polygons_reduce_atrR();
- serarch 系のメソッドが返却するポリゴン情報は，自ランクの担当領域内のみのポリゴンとなります．

3.3　C 言語用 Polylib 主な API 利用方法（MPI 版）

本節では，C 言語用の MPI 版 Polylib の主な API 利用方法を，API 呼び出し順に沿って説明します．C 言語用 MPI 版 Polylib の API を利用する手順は図 3.4 の通りです．

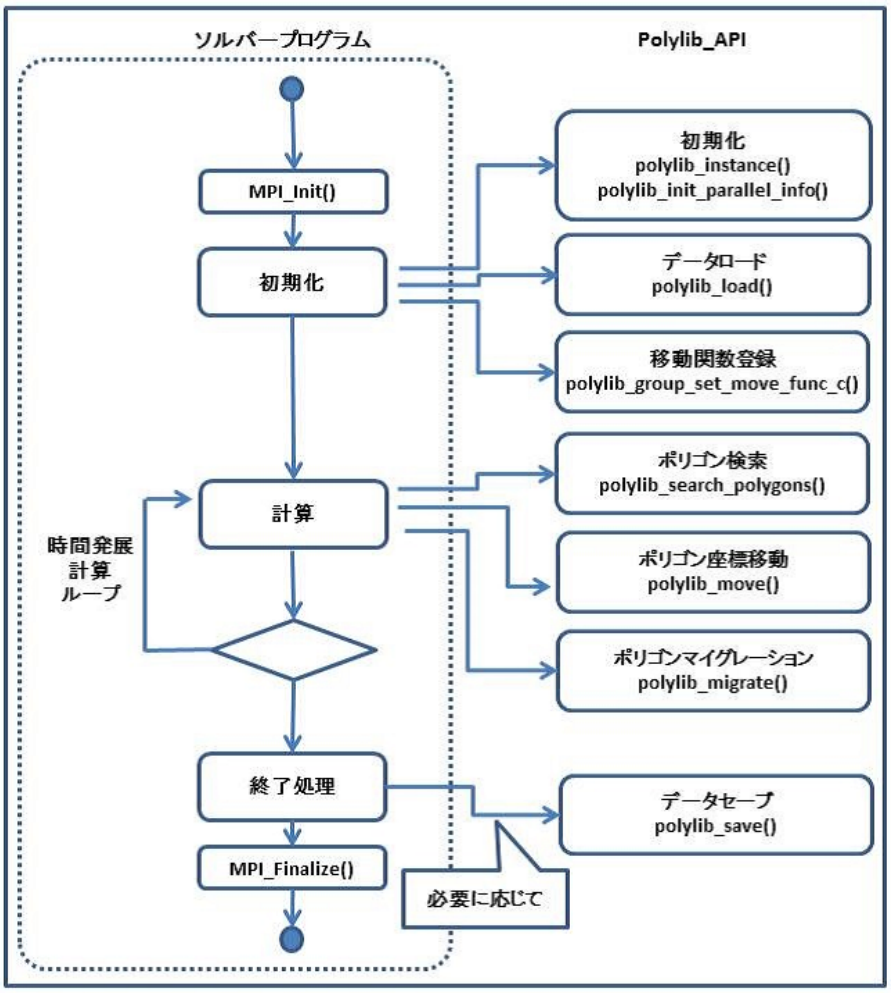

図 3.4　API 利用手順 (C 言語 MPI 版)

C 言語用 Polylib の各 API は，基本的に C++ 版 Polylib の各メソッドのラッピング関数として実現されています．各 API は機能上は同等です．C では C++ のクラスオブジェクトは使用出来ませのでタグを使って操作します．ポリゴン検索を例として説明します．

ポリゴン検索

```
POLYLIB_STAT polylib_search_polygons(
        int             *num,
        PL_ELM_TAG  **tags,
        char*           group_name,
        PL_REAL       min_pos[3],
        PL_REAL       max_pos[3],
        int             every
);
```

C 言語では stl の vector が使用出来ませんので内部で領域を確保して、検索ポリゴン数および検索ポリゴンのタグを返

しています．tags は使用後、free して下さい．返されたポリゴンのタグよりポリゴンの情報を取得します．

（ポリゴン情報取得 例）

```
        // 頂点座標取得
        void polylib_triangle_get_vertexes(
                PL_ELM_TAG tag,  // Triangle/NptTriangle を操作するためのタグ
                PL_REAL vertex[9]  // 3 頂点の座標
         );
        // 法線ベクトル取得
        void polylib_triangle_get_normal(
                PL_ELM_TAG tag,  // Triangle/NptTriangle を操作するためのタグ
                PL_REAL norm[3]  // 法線ベクトル
         );
```

3.4　Fortran 言語用 Polylib 主な API 利用方法（MPI 版）

本節では Fortran 言語用の MPI 版 Polylib の主な API 利用方法を，API 呼び出し順に沿って説明します．Fortran 言語用 MPI 版 Polylib の API を利用する手順は 3.5 の通りです．

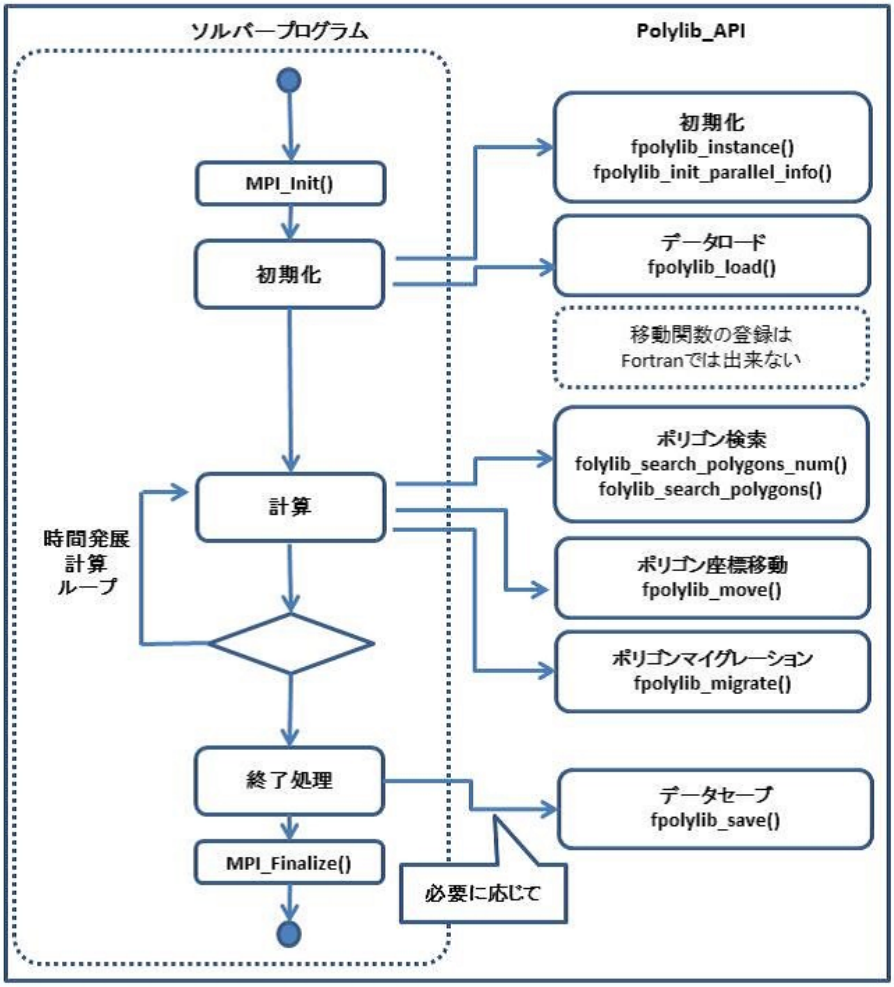

図 3.5　API 利用手順 (Fortran 言語 MPI 版)

Fortran 言語用 Polylib の各 API は基本的に C++ 版 Polylib の各メソッドのラッピング関数として実現されています．各 API は機能上は同等です．Fortran では C++ のクラスオブジェクトは使用出来ませのでタグを使って操作します．相違点を以下に示します．

並列計算情報の設定

```
      fpolylib_init_parallel_info ( bpos, bbsize, gcsize, ret )
              real(PL_REAL_PN)  bpos(3)
              integer              bbsize(3)
              integer              gcsize(3)
              real(PL_REAL_PN)  dx(3)
              integer              ret
```

C++/C と違いコミュニケーターの指定は出来ません．固定で MPI_COMM_WORLD となります．Fortran と C++ でコミュニケーターの表現が違い，情報を引き渡せないため固定のコミュニケーターとなっています．

移動関数登録

Fortran からは移動関数の設定は出来ません．必要な場合は C/C++ にて登録して下さい．

ポリゴン検索

```
! ポリゴン検索数取得
fpolylib_search_polygons_num ( num,        group_name, min_pos, max_pos, every, ret)
        integer                          num
        character(PL_GRP_PATH_LEN)  group_name
real(PL_REAL_PN)              min_pos(3)
real(PL_REAL_PN)              max_pos(3)
integer                          every
integer                          ret
```

```
! ユーザ側 Fortran プログラム
! ポリゴン検索結果を格納するための領域確保）例
integer(PL_TAG_PN), allocatable :: tags(:)
allocate  ( tags(num)  )
```

```
! ポリゴン検索
fpolylib_search_polygons ( num, tags, group_name, min_pos, max_pos, every, ret)
        integer                          num
        integer(PL_TAG_PN)             tags(:)
        character(PL_GRP_PATH_LEN)  group_name
        real(PL_REAL_PN)               min_pos(3)
        real(PL_REAL_PN)               max_pos(3)
        integer                           every
        integer                           ret
```

Fortran 用 API では C/C++ と違い，API 内部で領域を確保し，その領域に検索結果のタグを設定して返すことが出来ません．(API 内部は C++ で作成しており，C++ で確保した領域を Fortran に返しても Fortran 側での操作（領域解放等）が行えないためです)

そのため，検索結果を格納するのに十分な領域をユーザ側で確保して渡すか，先に検索結果数のみを求めて，その検索結果数分確保した領域をポリゴン検索ルーチンに渡す必要があります．ポリゴンのタグが求まれば，タグよりポリゴンの情報を取得します．

(ポリゴン情報取得 例)

```
        ! 頂点座標取得
        fpolylib_triangle_get_vertexes( tag, vertex )
                integer(PL_TAG_PN) tag      ! Triangle/NptTriangle を操作するためのタグ
                real(PL_REAL_PN)  vertex(9) ! 3 頂点の座標
        ! 法線ベクトル取得
        fpolylib_triangle_get_normal( tag, norm )
                integer(PL_TAG_PN)  tag     ! Triangle/NptTriangle を操作するためのタグ
                real(PL_REAL_PN)    norm(3) ! 法線ベクトル
```

Fortran 版 Polylib API 使用時の注意事項

Fortran 版 API では，任意のコミュニケータが設定できないこと，および，移動関数を登録できないことを考えると，初期化, 終了化は C++/C のインターフェースを使用し，計算箇所のみ Fortran インターフェースを使用することを推奨します.

3.5　ロード時のメモリ削減用 API

ポリゴンデータを STL/NPT ファイルから読み出す際にロード時に使用するメモリ量を削減する API を提供しています．設定されたメモリ量に応じてファイルからデータを分割して読み出し，分割されたデータ毎に各ランク（プロセス）にポリゴンデータを分散して配信します．
並列実行時のみ有効です．単一プロセスでは全データを 1 プロセスで担当するしかないためです．
Polylib::load() を呼びだす前に以下のメソッドを呼出します．

```
void Polylib::set_max_memory_size_mb( int max_size_mb )
```

Polylib が利用する最大メモリサイズ (MB) を設定します．
ロード時のみ有効です．セーブ時は上記の設定は効きません．

3.6　動作確認用 API

本節では Polylib の動作確認用 API について説明します．これらの API はソルバー開発時に利用するものであり，動作速度やメモリ消費量の点において非効率的ですので，通常のソルバー実行時には利用しないでください．

3.6.1　ポリゴングループ階層構造確認用 API

```
        void Polylib::show_group_hierarchy( FILE *fp = NULL  );
```

Polylib 管理下の全てのポリゴングループについて，その名称を階層レベルに従ったインデントをつけて引数 fp で指定されたファイルへ出力します．fp が未指定の場合は標準出力に出力します．

3.6.2　ポリゴングループ情報確認用 API

```
        POLYLIB_STAT Polylib::show_group_info(
                const std::string&  group_name,
                bool detail = false  // ポリゴンの座標値・法線ベクトルを出力するか否か
        );
```

指定された名称のポリゴングループについて，グループの情報と配下の三角形ポリゴン情報を標準出力に出力します．出力内容は以下の通りです．

- 親グループ名称
- 自身の名称
- STL ファイル名
- 登録三角形数
- 各三角形の 3 頂点ベクトルの座標
- 法線ベクトルの座標
- 面積

3.6.3　ポリゴン座標移動距離確認用 API

```
POLYLIB_STAT PolygonGroup::init_check_leaped();
POLYLIB_STAT PolygonGroup::check_leaped(
        std::vector< Vec3<PL_REAL> >& origin,
        std::vector< Vec3<PL_REAL> >& cell_size
);
```

PolygonGroup に登録した移動関数内において，三角形ポリゴンの頂点座標が隣接ボクセルより遠方へ移動したか否かをチェックするために利用する API です．
move メソッドでのポリゴンが隣接ボクセルより遠方へ移動した場合，結果が異常になります．（ポリゴン数が少なくなったりします）
PolygonGroup::init check leaped() は，チェック処理の初期化関数です．登録した移動関数内で，実際に頂点移動処理を行う前に呼び出します．
移動前の頂点座標を一時的に保存しますので，当該ポリゴングループの三角形ポリゴン数に応じてメモリを消費します．PolygonGroup::check leaped() は，移動前後の頂点座標の距離を確認する関数です．登録した移動関数内で，頂点移動処理実行後に呼び出します．
隣接ボクセルより遠方に移動した頂点については，標準エラー出力にそのポリゴン ID，移動前後の頂点座標情報を出力します．
並列環境下で本 API を利用する場合，各ランクにおける check_leaped() の引数 origin, cell_size は，Polylib::get_myproc() で取得できる ParallelInfo 構造体のメンバ変数 m_area から取得することが可能です．
なお，PolygonGroup::init_check_leaped() で確保された一時的メモリ領域は，PolygonGroup::check_leaped() を呼び出すと解放されます．

3.6.4　メモリ消費量確認用 API

```
size_t used_memory_size();      // byte 単位
size_t used_memory_size_mb();   // Mbyte 単位
```

Polylib が確保しているメモリ量を返却します．
並列実行時は，本メソッドを呼び出したランクにおけるメモリ量が返されます．
報告されるメモリ消費量は概算です．PolygonGroup クラスのユーザ属性等の Polylib フレームワーク外の消費メモリ利用については含まれません．

3.7　エラーコード

Polylib 内部でエラーが発生した場合に返却されるエラーコード POLYLIB_STAT 型は，include/common/PolylibStat.h に定義されています．エラーコードの一覧を下表に示します．

表 3.1　エラーコード一覧

エラーコード	意味
PLSTAT_OK	処理が成功
PLSTAT_NG	一般的なエラー
PLSTAT_INSTANCE_EXISTED	Polylib インスタンスがすでに存在している
PLSTAT_INSTANCE_NOT_EXIST	Polylib インスタンスが存在しない
PLSTAT_MPI_ERROR	MPI 関数がエラーを戻した
PLSTAT_ARGUMENT_NULL	引数のメモリ確保が行われていない
PLSTAT_MEMORY_NOT_ALLOC	メモリ確保に失敗した
PLSTAT_LACK_OF_MEMORY	メモリ不足
PLSTAT_CONFIG_ERROR	定義ファイルでエラー発生
PLSTAT_STL_IO_ERROR	STL ファイル IO エラー
PLSTAT_NPT_IO_ERROR	長田パッチファイル IO エラー
PLSTAT_UNKNOWN_FILE_FORMAT	ファイルが.stla，.stlb，.stl，.npta，.nptb，.npt 以外
PLSTAT_LACK_OF_LOAD_MEMORY	ロード処理時のメモリ不足
PLSTAT_FILE_NOT_SET	リーフグループにファイル名が未設定
PLSTAT_GROUP_NOT_FOUND	グループ名が Polylib に未登録
PLSTAT_GROUP_NAME_EMPTY	グループ名が空である
PLSTAT_GROUP_NAME_DUP	グループ名が重複している
PLSTAT_POLYGON_NOT_EXIST	ポリゴンが存在しない
PLSTAT_NODE_NOT_FIND	KD 木生成時に検索点が見つからなかった
PLSTAT_ROOT_NODE_NOT_EXIST	KD 木のルートノードが存在しない
PLSTAT_NOT_NPT	長田パッチではない
PLSTAT_ATR_NOT_EXIST	属性が未設定

第 4 章

ツール

Polylib で用意しているツールの機能と利用方法を説明します．
なお，Npatch ライブラリなしでインストールした場合，以下のツールは存在しません．

4.1　stl_to_npt

STL ファイルを NPT(長田パッチ) ファイルに変換するツールです．利用方法を以下に示します．

```
（並列環境の場合）
 mpirun -np  nproc  stl_to_npt   stl_file

（逐次環境の場合）
 stl_to_npt   stl_file
```

mpirun -np :　MPI 実行のための記述（必要に応じて mpiexec -n 等に変更して下さい）
nproc :　並列プロセス数
stl_to_npt :　プログラム名
stl_file :　STL ファイルパス

出力ファイル名は入力した STL ファイル名の拡張子を “npt” に変更したものとなります．長田パッチに変換するためには，ポリゴンの各頂点の法線ベクトルが必要です．
各頂点の法線ベクトルを求めるためには，頂点の同一点判定処理を行います．
同一点判定は，頂点数が多い場合に非常に時間がかかる処理です．
当ツールは，同一点判定を行うために Polylib の機能を使用しており高速，かつ，並列に同一点判定を行っています．

4.2　npt_to_stl

NPT(長田パッチ) ファイルを STL ファイルに変換するツールです．利用方法を以下に示します．

```
      npt_to_stl   npt_file
```

npt_to_stl :　プログラム名
npt_file :　NPT ファイルパス

出力ファイル名は入力した NPT ファイル名の拡張子を “stl” に変更したものとなります．

4.3　npt_to_stl4

NPT(長田パッチ) ファイルの各辺の中点の曲面補間点を追加してポリゴン数を 4 倍とし、STL ファイルに変換するツールです．利用方法を以下に示します．

```
      npt_to_stl4   npt_file
```

npt_to_stl4 :　プログラム名
npt_file :　NPT ファイルパス

出力ファイル名は入力した NPT ファイル名の拡張子”.npt” を “_4.stl” に変更したものとなります．長田パッチの形状を簡易的に確認するためのツールです．

第 5 章

テストコード

環境を正しく構築出来たか確認するために自動テスト機能を実装しています.

5.1　テストプログラム

表 5.1　テストプログラム一覧

プログラム名	内容
file_io_stl	STL ファイル IO テスト
file_io_npt	NPT ファイル IO テスト
search_polygon	ポリゴン検索テスト
attribute	ポリゴングループ属性，ポリゴン属性テスト
move_polygon	ポリゴン移動テスト
multi_bbox	1 プロセス複数領域テスト
load_reduce_mem	ロード時メモリ削減テスト
c_interface	C 言語インターフェーステスト
f_interface	Fortran インターフェーステスト

autotools による環境構築時に make check を実行することにより動作確認を行うことが出来ます.

第 6 章

チュートリアル

本章では，例題を用いた Polylib の使用法について説明します．

6.1　サンプルモデルによるチュートリアル

Polylib の利用方法を説明するために，図 6.1 のサンプルモデルを考えます．

各物体形状にはそれぞれ名前が付いており，時間発展計算の時刻ステップに応じて物体形状が移動する物体についてはその移動計算式が分かっているものとします．

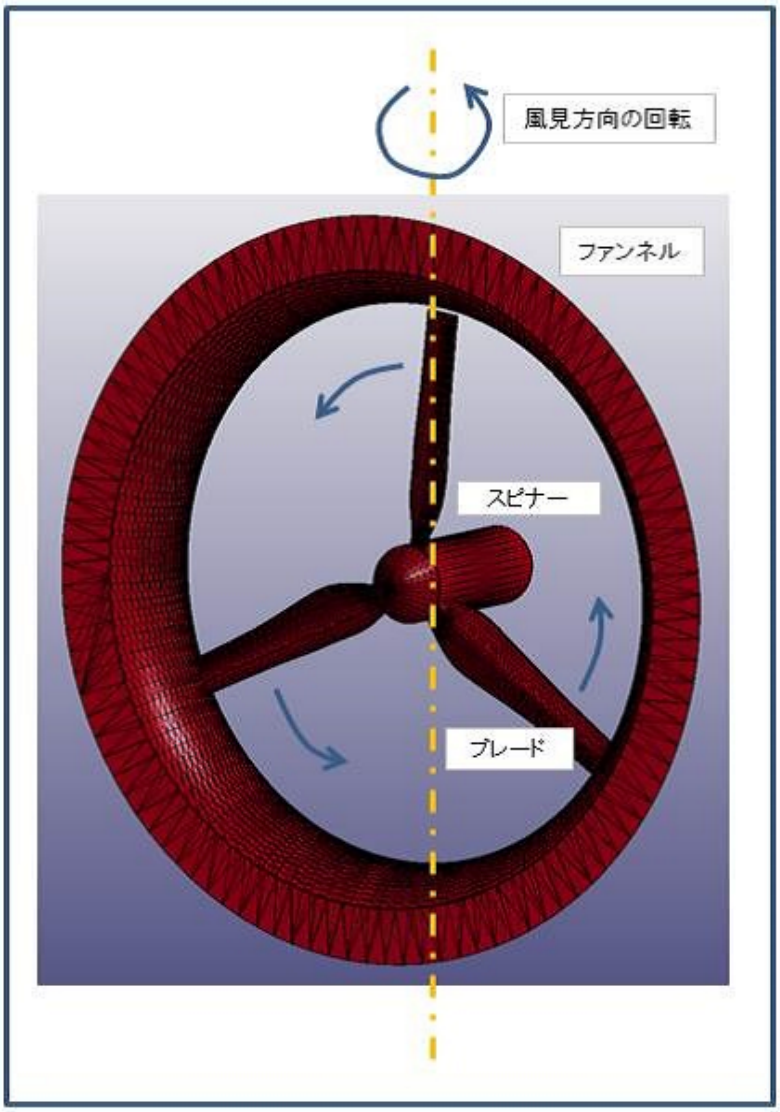

図 6.1　サンプルモデル

6.2　初期化ファイル

サンプルモデルの初期化ファイルの内容を以下に示します.

```
polylib {
    windmill{
        // 風車のユーザ定義属性
        //    ここでは回転軸情報を与える
        //    center_*          : 回転軸の原点
        //    yaw_axis_vec_*    : 風見方向の回転軸中心のベクトル
        //    roll_axis_vec_*    : ロータの回転軸中心のベクトル（初期値）
        UserAtr{
            center_x = "0.0"
            center_y = "0.0"
            center_z = "0.0"
            yaw_axis_vec_x    = "0.0"
            yaw_axis_vec_y    = "0.0"
            yaw_axis_vec_z    = "1.0"
            roll_axis_vec_x   = "1.0"
            roll_axis_vec_y   = "0.0"
            roll_axis_vec_z   = "0.0"
        }

        // 形状
        spinner{
            filepath="WL3000/WL3000_WLsolid-spinner.stl"
            movable = "true"
        }
        blades{
            filepath="WL3000/WL3000_WLsolid-blades.stl"
            movable = "true"
        }
        funnel{
            filepath="WL3000/WL3000_WLsolid-funnel.stl"
            movable = "true"
        }
    }
} // end of Polylib
```

6.3　プログラムソース

サンプルモデルを前提とした Polylib を利用するメインプログラム例を以下に示します.

```
#include <stdlib.h>
#include "Polylib.h"
#include "CalcGeo.h"

using namespace PolylibNS;
using namespace std;

// 領域分割情報
// ランク数 4 : Y 方向 2 分割, Z 方向 2 分割　均等分割
static ParallelBbox myParaBbox[4] = {
    { // rank0
        {-1800.0, -1800.0, -1800.0,},   // 基点座標
        {     36,       18,       18,},   // 計算領域のボクセル数
        {      1,        1,        1,},   // ガイドセルのボクセル数
        {  100.0,   100.0,   100.0 }    // ボクセル 1 辺の長さ
    },
    { // rank1
        {-1800.0,     0.0, -1800.0,},   // 基点座標
        {     36,       18,       18,},   // 計算領域のボクセル数
        {      1,        1,        1,},   // ガイドセルのボクセル数
        {  100.0,   100.0,   100.0 }    // ボクセル 1 辺の長さ
    },
    { // rank2
        {-1800.0, -1800.0,     0.0,},   // 基点座標
        {     36,       18,       18,},   // 計算領域のボクセル数
        {      1,        1,        1,},   // ガイドセルのボクセル数
        {  100.0,   100.0,   100.0 }    // ボクセル 1 辺の長さ
    },
    { // rank3
        {-1800.0,     0.0,     0.0,},   // 基点座標
        {     36,       18,       18,},   // 計算領域のボクセル数
        {      1,        1,        1,},   // ガイドセルのボクセル数
        {  100.0,   100.0,   100.0 }    // ボクセル 1 辺の長さ
    }
};

//-----------------------------------------------------
//  風車座標系の保存領域
//      風見方向の回転のみ
//-----------------------------------------------------
static PL_REAL windmill_origin[3];    // 風車原点
static PL_REAL windmill_x_axis[3];    // 風車 X 軸方向ベクトル

                                      //      ロータの回転軸
static PL_REAL windmill_z_axis[3];    // 風車 Z 軸方向ベクトル
                                      //      風見方向の回転軸

//-----------------------------------------------------
//  移動関数（funnel ）
//      風見方向の回転のみ
//-----------------------------------------------------
void move_func_funnel(
          PolygonGroup*      pg,
          PolylibMoveParams* params
       )
{
    // 風車の回転角度 (rad) を求める
    PL_REAL delta_rad_yaw = params->m_params[0];  // 風車の回転角差分 (rad)/step
    PL_REAL rad_yaw = delta_rad_yaw * (params->m_next_step - params->m_current_step);

    // 風車の回転マトリックスを求める
    PL_REAL mat_yaw[4][4];
    Calc_3dMat4Rot2(
```

```
            windmill_origin,   // [in]  回転軸上の 1 点の座標
            windmill_z_axis,   // [in]  回転軸のベクトル
            rad_yaw,           // [in]  回転角 (rad)
            mat_yaw          // [out] 回転用４×４変換マトリクス（行ベクトル系）
         );

    // ポリゴン情報取得
    std::vector<Triangle* > *tri_list =  pg->get_triangles();

    // 三角リスト内の全ての三角形について頂点座標を更新する
    //   回転軸を中心として回転させる
    for(int i=0; i<tri_list->size(); i++ ) {
        Vec3<PL_REAL>* vertex = (*tri_list)[i]->get_vertexes();

        for( int j=0; j<3; j++ ) {
            PL_REAL  pos[4],pos_o[4];
            pos[0]=vertex[j].x; pos[1]=vertex[j].y; pos[2]=vertex[j].z; pos[3]=1.0;
            // 座標を回転させる
            Calc_3dMat4Multi14( pos, mat_yaw, pos_o );
            // 座標更新
            vertex[j].x=pos_o[0]; vertex[j].y=pos_o[1]; vertex[j].z=pos_o[2];
            // 法線ベクトル更新・面積非更新
            (*tri_list)[i]->update( true, false );
        }
    }

    // 頂点座標が移動したことにより、KD 木の再構築が必要
    // 再構築フラグを立てる
    pg->set_need_rebuild();
}

//-----------------------------------------------------
//  移動関数 (blades/spinner )
//      風見方向に回転した後、ブレードの回転
//-----------------------------------------------------

void move_func_blades(
          PolygonGroup*      pg,
          PolylibMoveParams* params
       )
{
    // 風車の回転角度 (rad) を求める
    PL_REAL delta_rad_yaw = params->m_params[0];  // 風車の回転角差分 (rad)/step
    PL_REAL rad_yaw = delta_rad_yaw * (params->m_next_step - params->m_current_step);

    // ブレードの回転角度 (rad) を求める
    PL_REAL rpm = params->m_params[1];  // 回転数/分
    PL_REAL rad_roll = 2.0*PAI*(rpm/60.0)
       * (params->m_next_step - params->m_current_step)*params->m_delta_t;

    // 風車の回転マトリックスを求める
    PL_REAL mat_yaw[4][4];
    Calc_3dMat4Rot2(
            windmill_origin,   // [in]  回転軸上の 1 点の座標
            windmill_z_axis,   // [in]  回転軸のベクトル
            rad_yaw,           // [in]  回転角 (rad)
            mat_yaw          // [out] 回転用４×４変換マトリクス（行ベクトル系）
         );

    // ブレードのローカルの回転マトリックスを求める
    PL_REAL mat_roll[4][4];
    Calc_3dMat4Rot2(
            windmill_origin,  // [in]  回転軸上の 1 点の座標
            windmill_x_axis,  // [in]  回転軸のベクトル (global)
            rad_roll,         // [in]  回転角 (rad)
            mat_roll         // [out] 回転用４×４変換マトリクス（行ベクトル系）
         );

    // 最終的な回転マトリックスを求める
    PL_REAL mat[4][4];
    Calc_3dMat4Multi44( mat_roll, mat_yaw, mat );
```

```
    // ポリゴン情報取得
    std::vector<Triangle* > *tri_list =  pg->get_triangles();

#ifdef DEBUG
    // 頂点が隣接セルよりも遠くへ移動した三角形情報チェック（前処理）
    //  デバッグ用
    pg->init_check_leaped();
#endif

    // 三角リスト内の全ての三角形について頂点座標を更新する
    //   回転軸を中心として回転させる
    for(int i=0; i<tri_list->size(); i++ ) {
        Vec3<PL_REAL>* vertex = (*tri_list)[i]->get_vertexes();

        for( int j=0; j<3; j++ ) {
            PL_REAL  pos[4],pos_o[4];
            pos[0]=vertex[j].x; pos[1]=vertex[j].y; pos[2]=vertex[j].z; pos[3]=1.0;
            // 座標を回転させる
            Calc_3dMat4Multi14( pos, mat, pos_o );  // 行ベクトル系
            // 座標更新
            vertex[j].x=pos_o[0]; vertex[j].y=pos_o[1]; vertex[j].z=pos_o[2];
            // 法線ベクトル更新・面積非更新
            (*tri_list)[i]->update( true, false );
        }
    }

    // 頂点座標が移動したことにより、KD 木の再構築が必要
    // 再構築フラグを立てる
    pg->set_need_rebuild();

#ifdef DEBUG
    // 頂点が隣接セルよりも遠くへ移動した三角形情報チェック（後処理）
    //  デバッグ用
    Polylib* p_polylib = Polylib::get_instance();
    ParallelAreaInfo* area_info = p_polylib->get_myproc_area();
    std::vector< Vec3<PL_REAL> > origin;
    std::vector< Vec3<PL_REAL> > cell_size;
    for(int i=0; i<area_info->m_areas.size(); i++ ) {
        origin.push_back( area_info->m_areas[i].m_bpos );
        cell_size.push_back( area_info->m_areas[i].m_dx );
    }

    pg->check_leaped( origin,cell_size );
#endif

}


//------------------------------------------------------------------------
//  メインルーチン
//------------------------------------------------------------------------

int main(int argc, char** argv )
{
    POLYLIB_STAT ret;
    int     iret;
    int     num_rank;
    int     myrank;

    //-----------------------------------------
    //  ファイルパス
    //-----------------------------------------

    std::string config_file_name = "polylib_config.tp";   // 入力：初期化ファイル名

    //-----------------------------------------
    //  初期化
    //-----------------------------------------

    // MPI 初期化
```

```
MPI_Init( &argc, &argv );
MPI_Comm_size( MPI_COMM_WORLD, &num_rank );
MPI_Comm_rank( MPI_COMM_WORLD, &myrank );

// Polylib 初期化
Polylib* p_polylib = Polylib::get_instance();

// 並列計算関連情報の設定と初期化
ret = p_polylib->init_parallel_info(
            MPI_COMM_WORLD,
            myParaBbox[myrank].bpos,
            myParaBbox[myrank].bbsize,
            myParaBbox[myrank].gcsize,
            myParaBbox[myrank].dx
        );

//-----------------------------------------
//  ロード
//-----------------------------------------

// 初期化ファイルを指定してデータロード
ret = p_polylib->load( config_file_name );

// ポリゴングループポインタ取得
std::string   pg_windmill_path = "windmill";
PolygonGroup* pg_windmill = p_polylib->get_group( pg_windmill_path );
if( pg_windmill == NULL )  {
    exit(1);
}

std::string   pg_blades_path = "windmill/blades";
PolygonGroup* pg_blades = p_polylib->get_group( pg_blades_path );
if( pg_blades == NULL )  {
    exit(1);
}

std::string   pg_spinner_path = "windmill/spinner";
PolygonGroup* pg_spinner = p_polylib->get_group( pg_spinner_path );
if( pg_spinner == NULL )  {
    exit(1);
}

std::string   pg_funnel_path = "windmill/funnel";
PolygonGroup* pg_funnel = p_polylib->get_group( pg_funnel_path );
if( pg_funnel == NULL )  {
    exit(1);
}

// 風車座標系 初期値設定
std::string   key, val;

key = "center_x";
pg_windmill->get_atr( key, val );
windmill_origin[0] = atof( val.c_str() );
key = "center_y";
pg_windmill->get_atr( key, val );
windmill_origin[1] = atof( val.c_str() );
key = "center_z";
pg_windmill->get_atr( key, val );
windmill_origin[2] = atof( val.c_str() );
key = "yaw_axis_vec_x";
pg_windmill->get_atr( key, val );
windmill_z_axis[0] = atof( val.c_str() );
key = "yaw_axis_vec_y";
pg_windmill->get_atr( key, val );
windmill_z_axis[1] = atof( val.c_str() );
key = "yaw_axis_vec_z";
pg_windmill->get_atr( key, val );
windmill_z_axis[2] = atof( val.c_str() );
key = "roll_axis_vec_x";
pg_windmill->get_atr( key, val );
```

```
windmill_x_axis[0] = atof( val.c_str() );
key = "roll_axis_vec_y";
pg_windmill->get_atr( key, val );
windmill_x_axis[1] = atof( val.c_str() );
key = "roll_axis_vec_z";
pg_windmill->get_atr( key, val );
windmill_x_axis[2] = atof( val.c_str() );

// 移動関数登録
//   blades と spinner は同一中心軸に対して同じ回転をさせれば良いので
//   同じ移動関数を登録する

pg_blades->set_move_func ( move_func_blades );
pg_spinner->set_move_func( move_func_blades );
pg_funnel->set_move_func ( move_func_funnel );

// move parameter 初期化
PolylibMoveParams params;
memset( params.m_params, 0x00, 10*sizeof(PL_REAL) );

//---------------------------------
// タイムステップループ
//---------------------------------

int nstep = 100;

for(int istep=0; istep<nstep ; istep++ )  {

    //-------------------------------------------
    // 現在のステップで計算実行
    //-------------------------------------------

    /*
    vector<Triangle*> tri_list;
    p_polylib->search_polygons( tri_list, / * 検索条件を設定 * / );

    // 解析処理実行

    */

            // 風向きの偏向による風車の角度変更を設定する
    PL_REAL delta_rad_yaw = (0.5*PAI/180.0);  // 1step で 0.5 度偏向
                                    // テスト用:実際の変化量としては大きすぎる
            // ブレードの回転速度を求めたものとする
    PL_REAL rpm = 20;                          // 1 分間に 20 回転 （1 回転/3 秒）

    //-------------------------------------------
    // 次計算ステップに進むためにポリゴン情報更新
    //-------------------------------------------

    // move パラメタ設定
    params.m_current_step = istep;
    params.m_next_step    = istep + 1;
    params.m_delta_t      = 0.01;            // delta 秒
    params.m_params[0]    = delta_rad_yaw;   // 風車の角度偏向
    params.m_params[1]    = rpm;             // 回転速度

    //--------------------
    // move 実行
    //--------------------
    ret = p_polylib->move( params );


    // 風車座標系の更新
    //      原点と Z 軸は変わらないので X 軸のみ更新する
        // 回転角度 (rad)
    PL_REAL rad_yaw = delta_rad_yaw * (params.m_next_step - params.m_current_step);
        // 風車の回転マトリックスを求める
    PL_REAL mat_yaw[4][4];
    Calc_3dMat4Rot2(
        windmill_origin,  // [in]  回転軸上の 1 点の座標
```

```
            windmill_z_axis,  // [in]  回転軸のベクトル
            rad_yaw,          // [in]  回転角 (rad)
            mat_yaw          // [out] 回転用４×４変換マトリクス（行ベクトル系）
        );
             // X 軸の更新
        PL_REAL  vec_in[4],vec_out[4];
        vec_in[0]=windmill_x_axis[0];
        vec_in[1]=windmill_x_axis[1];
        vec_in[2]=windmill_x_axis[2];
        vec_in[3]=1.0;
        Calc_3dMat4Multi14( vec_in, mat_yaw, vec_out );
        windmill_x_axis[0]=vec_out[0];
        windmill_x_axis[1]=vec_out[1];
        windmill_x_axis[2]=vec_out[2];

        //--------------------
        // migrate 実行
        //--------------------
        ret = p_polylib->migrate();

        //---------------------------------------------
        // テスト用の検証
        //---------------------------------------------

        // テスト用にファイル出力
        if( istep!=0 && (istep%10)==0 ) {
            std::string  config_name_out;
            std::string  fmt_out = PolygonIO::FMT_STL_A;
            char buff[16];
            sprintf( buff,"%d",istep );
            std::string  sstep = buff;
            std::string  extend = "istep" + sstep;
            // 各ランク毎に途中経過を出力
            ret = p_polylib->save_parallel( config_name_out, fmt_out, extend );
            // 途中経過を出力
            ret = p_polylib->save( config_name_out, fmt_out, extend );
        }
    }

    //-----------------------------------------
    //  セーブ（処理後）
    //-----------------------------------------
    {
        std::string  config_name_out;
        std::string  fmt_out = PolygonIO::FMT_STL_A;
        ret = p_polylib->save( config_name_out, fmt_out );

    //-----------------------------------------
    //  終了化
    //-----------------------------------------
    MPI_Finalize();

    return 0;
}
```

第 7 章

Appendix

その他，補足資料です．

7.1　NPT ファイルフォーマット

7.1.1　アスキー形式

表 7.1　NPT テキストファイル形式

No.	項目	レコードフォーマット
1	ファセット数	ＮＮＮＮ ＮＮＮＮ：ポリゴン数を整数で記述する
2	ファセット開始ラベル	facet ファセットの開始宣言レコード
3	頂点 1 座標	vertex xxx yyy zzz xxx,yyy,zzz：xyz 座標の数値
4	頂点 2 座標	vertex xxx yyy zzz xxx,yyy,zzz：xyz 座標の数値
5	頂点 3 座標	vertex xxx yyy zzz xxx,yyy,zzz：xyz 座標の数値
6	長田パッチ parameter1	coef1 xxx yyy zzz xxx,yyy,zzz：xyz の数値 辺 1 の 3 次ベジェ制御点 1
7	長田パッチ parameter2	coef2 xxx yyy zzz xxx,yyy,zzz：xyz の数値 辺 1 の 3 次ベジェ制御点 2
8	長田パッチ parameter3	coef3 xxx yyy zzz xxx,yyy,zzz：xyz の数値 辺 2 の 3 次ベジェ制御点 1
9	長田パッチ parameter4	coef4 xxx yyy zzz xxx,yyy,zzz：xyz の数値 辺 2 の 3 次ベジェ制御点 2
10	長田パッチ parameter5	coef5 xxx yyy zzz xxx,yyy,zzz：xyz の数値 辺 3 の 3 次ベジェ制御点 1
11	長田パッチ parameter6	coef6 xxx yyy zzz xxx,yyy,zzz：xyz の数値 辺 3 の 3 次ベジェ制御点 2
12	長田パッチ parameter7	coef7 xxx yyy zzz xxx,yyy,zzz：xyz の数値 3 角形中央の 3 次ベジェ制御点
No.2～No.12 をファセット数 繰り返す		

長田パッチパラメータの制御点は頂点座標からの相対座標ではありません.

7.1.2　バイナリ形式

NPT ファイルのバイナリ形式は，STL に合わせて単精度、かつ、バイトオーダーはリトルエンディアンとしています．

表 7.2　NPT バイナリファイル形式

No.	項目	レコードフォーマット
1	ファセット数	整数 (4byte)
2	頂点 1 座標	単精度実数 (4byte) × 3
3	頂点 2 座標	単精度実数 (4byte) × 3
4	頂点 3 座標	単精度実数 (4byte) × 3
5	長田パッチ parameter1	単精度実数 (4byte) × 3 辺 1 の 3 次ベジェ制御点 1
6	長田パッチ parameter2	単精度実数 (4byte) × 3 辺 1 の 3 次ベジェ制御点 2
7	長田パッチ parameter3	単精度実数 (4byte) × 3 辺 2 の 3 次ベジェ制御点 1
8	長田パッチ parameter4	単精度実数 (4byte) × 3 辺 2 の 3 次ベジェ制御点 2
9	長田パッチ parameter5	単精度実数 (4byte) × 3 辺 3 の 3 次ベジェ制御点 1
10	長田パッチ parameter6	単精度実数 (4byte) × 3 辺 3 の 3 次ベジェ制御点 2
11	長田パッチ parameter7	単精度実数 (4byte) × 3 3 角形中央の 3 次ベジェ制御点
No.2〜No.11 をファセット数 繰り返す		

長田パッチパラメータの制御点は頂点座標からの相対座標ではありません．

第 8 章

アップデート情報

本ライブラリのアップデート情報について記します.

8.1　アップデート履歴

- Version 4.0.0 2016-3-25
  - 長田パッチフォーマット対応
  - 利用者様から見たオブジェクト指向要素の削減および構成のシンプル化
    クラスの整理・削減、クラス継承削減、移動関数登録、ユーザ定義属性追加
    コンパイルオプションによる実数型および逐次／並列処理の切り替え
  - Fortran インターフェース追加
  - 1 プロセス複数領域対応
  - 自動テスト対応

- Version 3.1.0 2013-11-14
  - 長田パッチフォーマット対応
  - ポリゴン頂点へのユーザ定義データ（スカラー／ベクター）設定機能の追加
  - ポリゴン頂点のユーザ定義データを含んだ VTK 形式ファイル出力機能の追加

- Version 3.0.0 2013-09-16
  - ポリゴンデータの省メモリ化
    重複する頂点座標データを持たない効率的なデータ構造の導入.
  - ポリゴンデータの実数型の選択機能
  - autoconf & automake 対応
  - 入出力ファイルフォーマットの追加
    obj ファイルの読み込み/書き出しを追加.
    stl ファイルのアスキーバイナリーの自動判別.
    これに伴い，ファイル拡張子は*.stl で統一.
  - サンプルプログラムの作成
  - 不要なコードの整理

- Version 2.6.8 2013-07-20
  - PolygonGroup::set all exid of trias() の修正
    m id, m id defined の両方に値をセット.

- Version 2.6.7 2013-07-20
  - Version 情報取得メソッドの追加
    Version.h.in と getVersionInfo() を追加.

- Version 2.6.6 2013-07-17
  - type ラベルの追加
    PolygonGroup::m type.
  - ポリゴンの exid に値をセットするメソッドを追加
    PolygonGroup::set all exid of trias() を追加.

- Version Version 2.6.5 2013-07-15
- Version 2.6.4 2013-06-27
- Version 2.6.3 2013-06-27
- Version 2.6.2 2013-06-26
- Version 2.6.1 2013-06-25

- Version 2.6 2013-06-24
  - autotools 導入
    並列版のみ autotools でビルド可能.

- Version 2.5 2013-06-17
- Version 2.4 2013-05-08
- Version 2.3 2013-03-25

- Version 2.2 2012-11-27
  - 直近ポリゴン検索機能の追加
    指定された点にもっとも近い三角形ポリゴンを検索する機能を追加.

- Version 2.1 2012-04-31
  - TextParser 書式の初期化ファイルに対応
    初期化ファイル書式を，XML 形式から TextParser 書式に変更.
  - id 付きバイナリ STL ファイル読み込みに対応
    FXgen が出力する id 付きバイナリ STL ファイルの読み込みに対応.
  - STL ファイル読み込み時の縮尺変換
    Polylib::load メソッドに引数 float scale=1.0 を追加し，
    ジオメトリデータを縮尺変換して読み込むオプションを追加.

- Version 2.0.3 2012-04-22
- Version 2.0.2 2010-11-17
- Version 2.0.1 2010-11-05

- Version 2.0.0 2010-06-30
  - ポリゴン移動機能の追加
    ユーザ定義によるポリゴン座標移動関数を用いた，時間発展計算実行中のポリゴン群の移動機能を追加.
    並列計算環境化においては，隣接 PE 計算領域間を移動したポリゴン情報を自動的に PE 間で融通.
  - 計算中断・再開への対応
    時間発展計算途中の計算中断時にポリゴン情報をファイル保存する機能を追加.
    ファイルの保存は，各ランク毎保存，もしくはマスターランクでの集約保存が選択可能.
    また，時間発展計算の再開時に利用することを想定し途中保存したファイルを指定して
    ポリゴン情報の読み込みを行う機能も追加.
  - データ登録系 API の整理
    ポリゴングループの登録や STL ファイルの読み込みなどのポリゴンデータ登録処理を XML 形式の設定
    ファイルを利用.
    データ登録系 API を大幅に刷新.

- Version 1.0.0 2010-02-26
  - 初版リリース