文部科学省次世代IT基盤構築のための研究開発
「イノベーション基盤シミュレーションソフトウェアの研究開発」

CISS フリーソフトウェア

HEC-MW

Ver. 3.0

ユーザーマニュアル

お問い合わせ先
（契約窓口）　（財）生産技術研究奨励会
〒153-8505　東京都目黒区駒場４－６－１
（ソフトウェア管理元）　東京大学生産技術研究所　革新的シミュレーション研究センター
〒153-8505　東京都目黒区駒場４－６－１
Fax：03-5452-6662

目次

1. はじめに

大規模並列有限要素基盤 HEC-MW の公開版ユーザズマニュアルである。

2. インストール方法

2.1 インストールできるソフトウェア

本ソフトウェアは、有限要素法（FEM）による大規模シミュレーションコードを開発する際、共通に利用される機能を提供することにより、解析コード開発者がアプリケーションソフトの開発に専念できるようにするためのソフトウェアである。本ソフトウェアをインストールすると、本ソフトウェアの機能が集約されたライブラリーが作成される。また、サンプルプログラムも作成される。このライブラリーをユーザの作成するプログラムにリンクすることで本ソフトウェアの機能を利用することができる。

2.2 インストールに必要なソフトウェアおよび動作確認を行っている環境

本ソフトウェアを正しく動作させるために、計算機上には、OS の他に、次のソフトウェアが必要である。

- C++コンパイラ
- boost
- MPI

これらのソフトウェアについては、次のバージョンで動作確認を行っている。

表 2.1 インストールに必要なソフトウェアおよび動作確認を行っている環境

項目	windows	linux
OS	Windows XP (32 bit)	Cent OS, ubuntu
C++	Visual C++ 2008 Express 以上	g++ 4.3 以上
boost	boost 1.43.0	boost 1.34
MPI	MPICH2	MPICH2

2.3 ファイルの解凍

サンプルプログラムを含めたプログラム一式が tar + gzip の形式でひとつのファイルになっている(ファイル名 ver20100512.tar.gz)．これを gunzip + tar で解凍する。

```
% gunzip ver20100512.tar.gz
% tar xvf - < ver20100512.tar
```

```
% cd ver20100512
% ls
 src lib test Makefile Makefile.in
```

図 2.3.1　ファイルの解凍方法

解凍されたファイル群は、ソースプログラム（src)、ライブラリー（lib)、サンプル（test）の３つのディレクトリから構成される。ソースプログラムはサブルーチンソースの集まりであり、インストールを実行するとライブラリーのディレクトリにライブラリーが作成される。テストプログラムは２つの並列化レベルに対応したサンプルプログラムが作成されている。Makefile.in に機種依存情報が格納されている。

表 2.2　インストールディレクトリ内容

ディレクトリ名	内容
lib	インストールディレクトリで make を実行するとライブラリー(libmw3.a)がこのディレクトリ内に作成される。ファイル解凍時は何もない。
src	本ソフトウェアのライブラリーとなるサブルーチン群のソースである。
test	線型弾性の構造解析プログラムおよびそのテストデータが格納されている。
Makefile	ライブラリーおよびテストプログラムの実行体の作成
Makefile.in	計算機環境および並列計算環境の違いによるインストール方法の違いを記述したファイルである。

2.4　コンパイルおよび実行の準備

以下、Linux の例で、記述する。次のような流れでコンパイルおよび実行準備をする。

```
% vi Makefile.in ← 利用者の環境にしたがって Makefile.in を編集
% make clean
% make　←　./src をコンパイルし./lib に*.a を作成し ./test のメインの実行体作成
```

図 2.4.1　コンパイルおよび実行の準備方法

使用する計算機の MPI 環境について調べ、include ファイル、l ライブラリー、および MPI 実行コマンドの所在を記述にし、インストールディレクトリのファイル「Makefile.in」の MPI 関連を編集する。

```
#============================================================
#
# Software Name :HEC middleware Ver. 3.0
#
#     Makelfile for MW3
#
#                              Written by T.Takeda,       2010/03/18
#                                         K.Goto,         2010/01/12
#                                         K.Matsubara, 2010/03/18
#
#     Contact address : IIS, The University of Tokyo CISS
#
#============================================================


#
# Where is my home?
#
MW3_HOME=  （tar で展開した内容のホームディレクトリ）
MPICH_HOME=  （mpi のホームディレクトリ）


#
# Which compiler to use
#
CPC= $(MPICH_HOME)/bin/mpicxx


#
# What options to be used by the compiler
#
COPT= -DHAVE_MPI
INC_DIR= $(MW3_HOME)/src


#
# What options to be used by the loader
#
LOPT=
LIB= $(MW3_HOME)/lib/libmw3.a
LLIB= -L$(MW3_HOME)/lib -lmw3


#
# What archiving to use
#
AR = ar rv


#
# What to use for indexing the archive
#
RANLIB = ar -ts
#RANLIB = ranlib
#RANLIB =


#
# Which mpirun to use
#
RUN= $(MPICH_HOME)/bin/mpirun
EXE= $(MW3_HOME)/test/a.out
```

```
#
# How to compile c++ codes
#
.SUFFIXES: .cxx .o
.cxx.o:
	$(CPC) -c $(COPT) -I$(INC_DIR) $< -o $@
```

図 2.4.2　Makefile.in の内容

2.5　ライブラリーおよび実行体の作成

インストールディレクトリで make を実行すると、各ディレクトリでコンパイルが実行され、プラットフォームのライブラリー、およびテストプログラムの実行体が作成される。

```
% make　←　./src をコンパイルし./lib に*.a を作成し ./test のメインの実行体作成
```

図 2.5.1　ライブラリーおよび実行体の作成方法

2.6　サンプルプログラムの実行

2.6.1　インストールディレクトリからの実行する場合

以下では、代表的なプログラムのチェック方法を示す。いずれの例題も数十から数千自由度の問題であり、テスト例題の実行時間は、数秒から数分程度の例題である。本ソフトウェアに含まれる全テスト例題の実行が行われる。

動作方法はそれぞれの Makefile の中の check の項目に記述されている。正常に起動しない場合には、 MPI コマンドのコマンドパスおよび Makefile.in で指定したライブラリーの位置を確認すること。

```
% make　check
```

図 2.6.1　サンプルプログラム実行方法（その 1）

2.6.2　ディレクトリ test から実行する場合

次のような方法で、本資料に記載された例題を実行することができる。

```
% cd test
% make test1
```

図 2.6.2　サンプルプログラム実行方法（その 2）

表 2.3　テストデータの内容

コマンド	内容	データ名
% make test1	片持ち梁（４要素）１プロセスでの実行	beam_0.dat
% make test2	MPC を用いた非連続メッシュの１プロセスでの実行	beamMPC_0.dat
% make test3	片持ち梁（４要素）のリファイナを利用した 2 プロセスでの実行	beamPara2_*.dat
% make test4	片持ち梁（４要素）のリファイナを利用した ４プロセスでの実行	beamPara4_*.dat

2.7　再コンパイルをする場合

環境設定値等が変化して実行体を再び作成したい場合や、異常な動作をするためにコンパイルし直したい場合には、次の操作を行う。必要に応じて現状の Makefile.in のバックアップをとっておくことを推奨する。

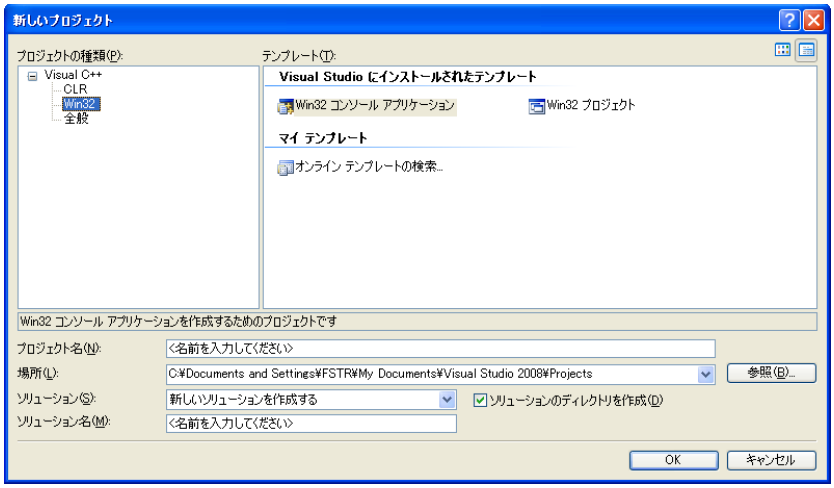
```
% vi Makefile.in ← 利用者の環境にしたがって Makefile.in を編集
% make clean
% make   ←   ./src をコンパイルし./lib に*.a を作成し  ./test のメインの実行体作成
```

図 2.7.1　再コンパイルの方法

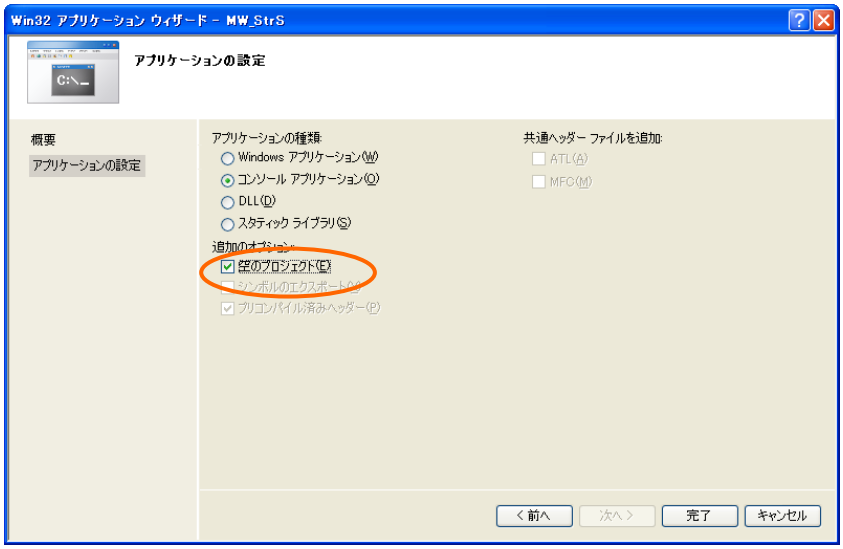
2.8　Windows で解析コードをビルドする

Visual C++ Express で HEC_MW ライブラリーをリンクさせて解析コードをビルドする。

ファイル→新規作成→プロジェクトを開き win32 コンソールアプリケーションをクリックして OK をクリック。

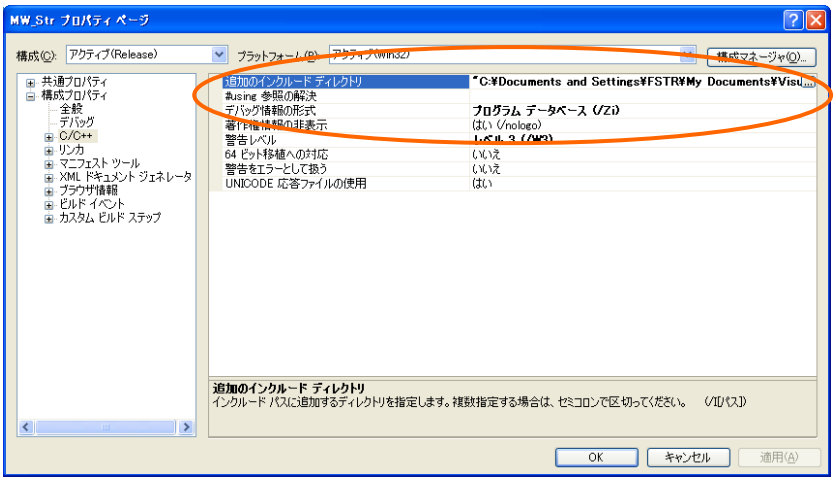


プロジェクトの設定で、"空のプロジェクト”を選ぶ。

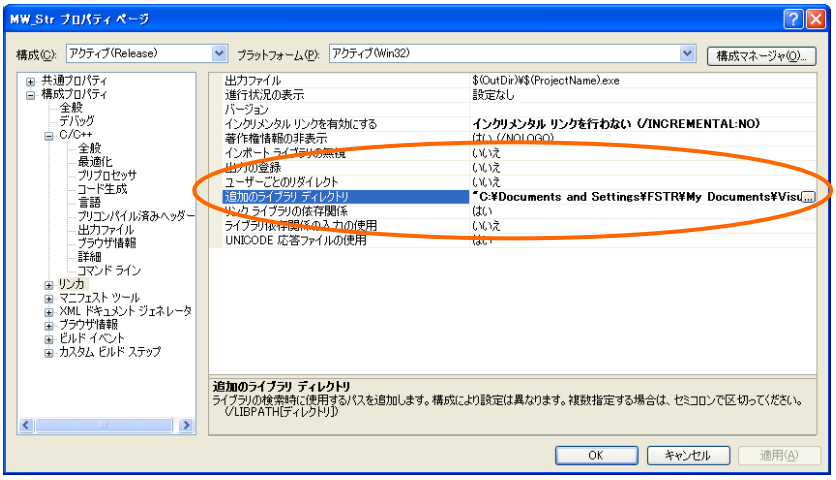


プロジェクトが生成されたら、ファイル→追加→既存項目と進み、ダウンロードした MW_Str.cxx を読み込み、次のプロパティ設定を行ってビルド環境を設定する。

Visual C++ Express のプロジェクト・プロパティダイアログボックスを開きインクルードディレクトリの指定に、ダウンロードした "MW_Str の include ディレクトリ" を指定。

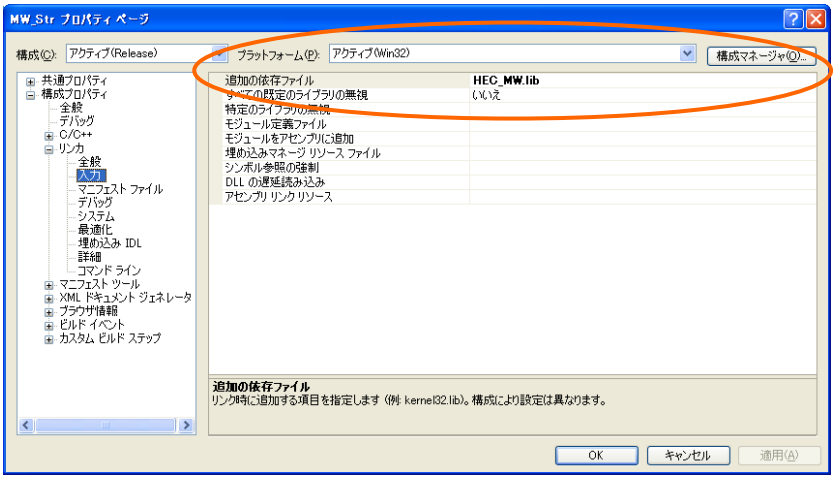


ダウンロードした **HEC_MW.lib**、**HEC_MWS.lib** をインストールしたディレクトリを指定。



並列版の場合は、追加の依存ファイルに **HEC_MW.lib** を指定。※並列版を使用する場合は、MPICH2 がインストールされている必要がある。

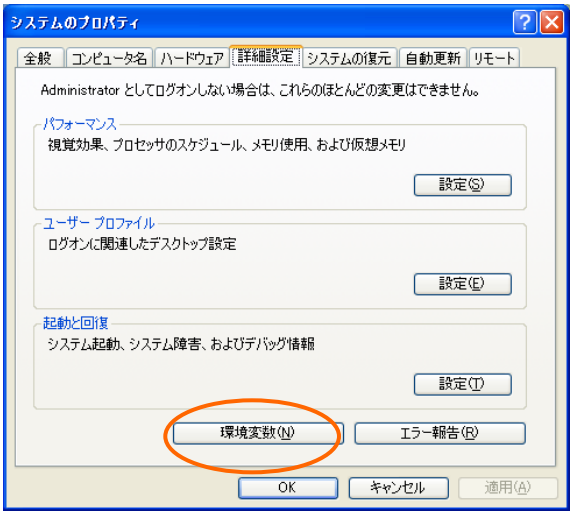
シングル版の場合は、**HEC_MWS.lib** を指定。

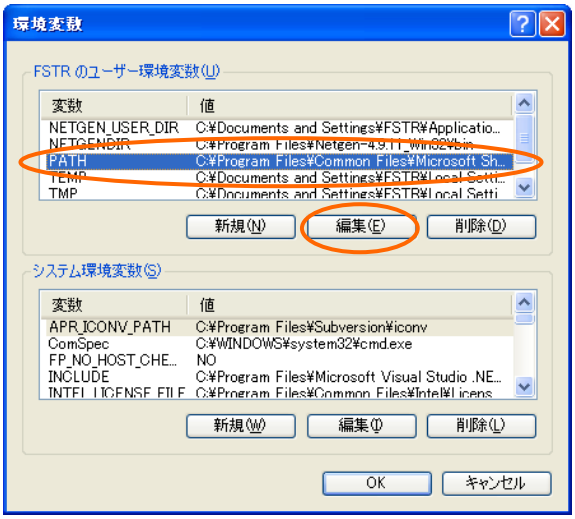


ビルド→ソリューションのビルドをクリックしてビルドする。

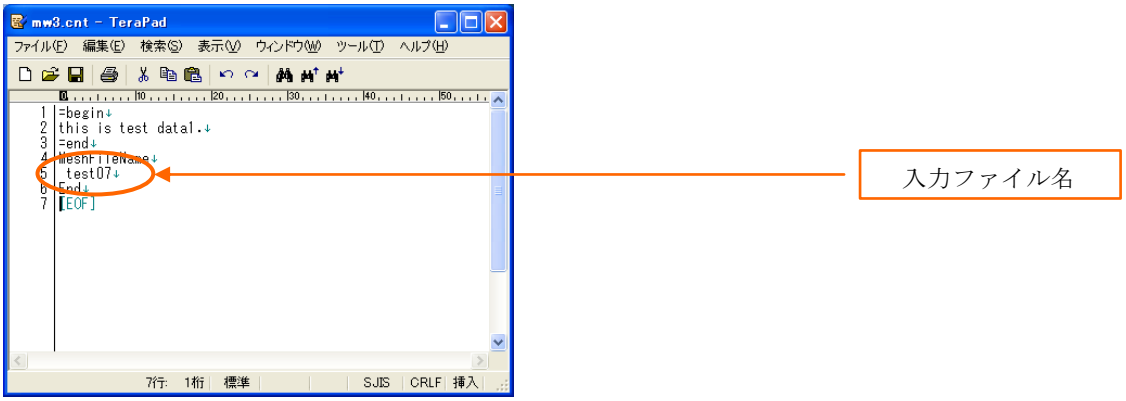
・ビルドした解析コードを実行する

Windows の環境変数の PATH に **MPICH2¥bin** と **HEC_MW.dll**、**HEC_MWS.dll** のディレクトリを記述して、パスを通しておく。　＊シングル版を使う場合は MPICH2 へのパスを通す必要はない。





テキストエディターを使って **mw3.cnt** の **MeshFileName** ブロックに使用するファイル名を記述する。　＊テキストエディターはノートパッドでもワードパッドでも何でも良い。



・コマンドプロンプトを開き、実行コマンドを入力する。

a.並列版の場合のコマンド入力

C:¥..... >***mpiexec -n 2 mw_str***

User credentials needed to launch processes:

account (domain¥user) [SS141¥FSTR]: △△△△

password: ******

PC のユーザー・アカウント(ユーザー名)

パスワード

b.シングル版の場合のコマンド入力

C:¥ ..... >***mw_strs***

3. システム概要

3.1 概要

有限要素法（FEM）による大規模シミュレーションコードを開発する際、共通に利用される機能を提供することにより、解析コード開発者がアプリケーションソフトの開発に専念できるようにする。

本ライブラリーの API（Application Programming Interface）を通じて、並列線形代数ソルバー、ファイル入出力、階層型アセンブリー構造データ、行列・ベクトル、要素ライブラリーなどの機能が提供される。

本ライブラリーを使用して開発された FEM プログラムは自動的に階層化、並列化され、不連続なメッシュデータを接合(アセンブル構造)することができる。

3.2 システムの構成

本システムは、以下の各部分から構成される。

a. 「HEC_MW3 ライブラリー」
   階層型アセンブリーデータ構造の管理
   行列およびベクトル データの管理
   要素ライブラリー
   並列線形ソルバー
   ファイル入出力
   Fortran プログラミングインターフェイス*
b.「並列メッシュ領域分割用ユーティリティーソフトウェア」
c.「可視化ライブラリー」
d. 検証用テストコード
*印 次期バージョンで対応予定

図 3.2.1 に本システムの全体構成を示す。

「並列メッシュ領域分割用ユーティリティーソフトウェア」は、並列有限要素法の計算に使用する、大規模非構造メッシュの領域分割を効率的に実施するためのユーティリティーである。

「検証用シミュレーションコード」はこれらの「HEC-MW3」の性能、有用性を検証するために、HEC-MW3 を使用して開発される並列有限要素法によるプログラムである。

図 3.2.1　本システムの全体構成

*印 次期バージョンで対応予定

4. HEC-MW3 の機能

4.1 データ構造

HEC_MW3 は、階層型アセンブリーデータ構造を構築できる。
アセンブリーデータ構造とは、個別にメッシュ生成したデータを接合していくデータ構造であり、階層型データ構造とは、入力されたメッシュデータを再分割することで幾何学的マルチグリッドソルバーに対応させたデータ構造のことである。

アセンブリーデータ構造を生成するには、各アセンブリーパーツの接合面を、それぞれ要素面グループとして定義しておき、それらのペアの一方をマスター面、もう一方をスレーブ面とし、スレーブ面上の各節点の自由度をマスター面上の対応する要素面に固定することで実現している。
上記の関係式は、HEC_MW3 内部にて自動生成され、MPC 条件として線形代数ソルバーに組み込まれることによりアセンブリー部品を結合している。

階層構造は、アセンブリーパーツを表すメッシュデータの各要素を再分割することで実現している。
再分割のために新たに節点を生成追加し、その節点データをもとに要素を分割し、上位階層の要素として新たに要素生成を行っている。

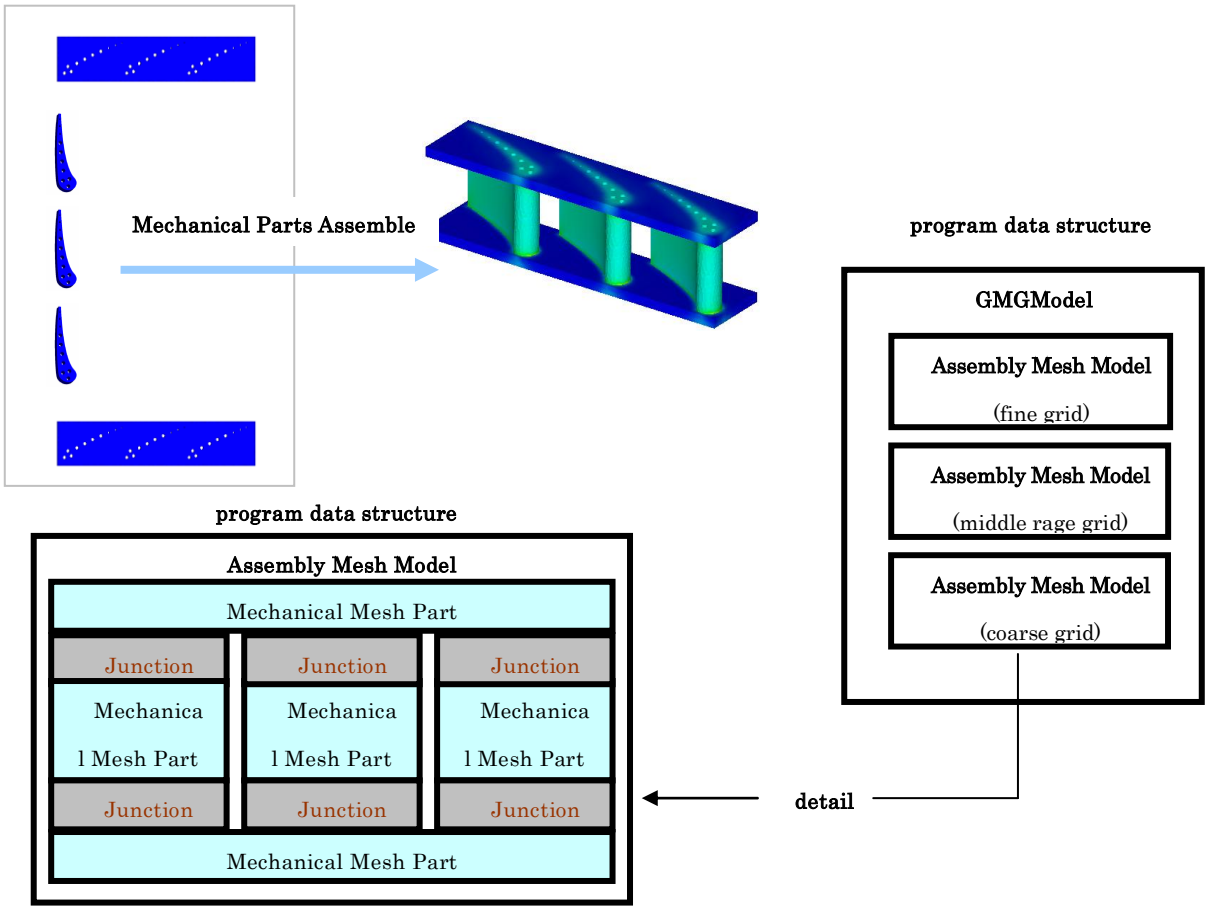


図 4.1.1　データ構造

階層型データ構造について更に記すと、ソリッド要素については全ての要素を Hexa 要素になるように再分割していく、これは FEM の解析において Hexa 要素の研究が進んでおり、現象によっては Hexa 要素でしか計算できない分野も存在することから、解析モジュール開発の適用範囲を広げることを目的とした処理である。

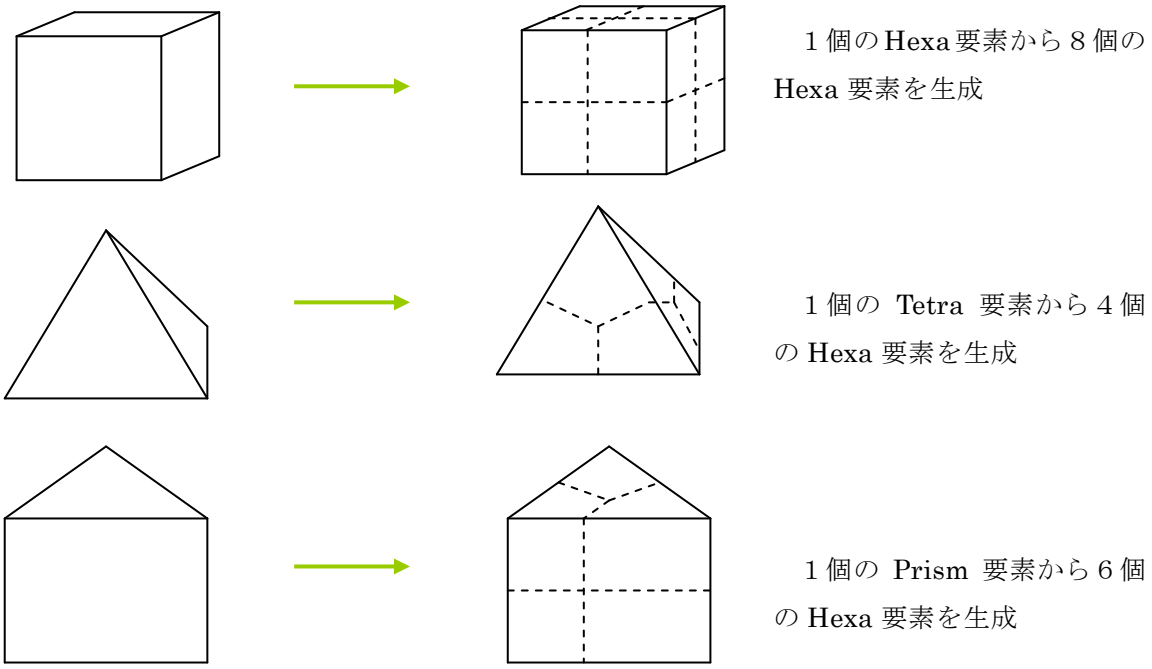


図 4.1.2　階層型データ構造

アセンブリーデータ構造についても、更に記しておく。

アセンブリーデータを構築するためには、接合面の"検索”と、マスター・スレーブの"関係式"を生成する必要があり、検索について説明する。

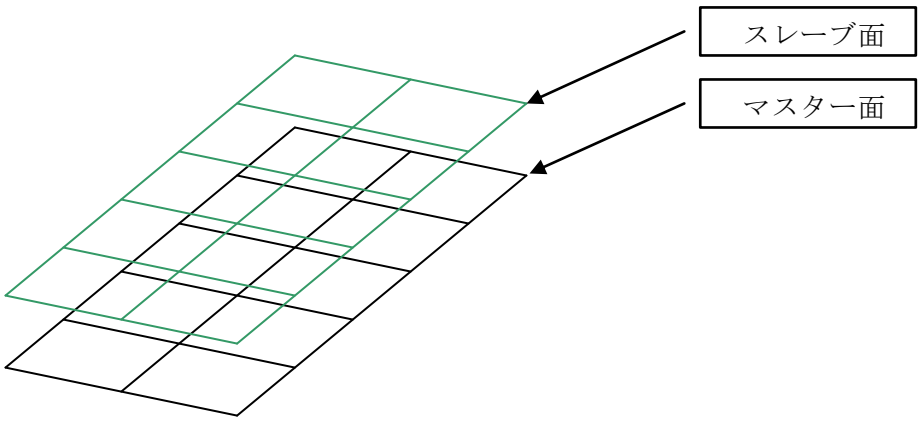


図 4.1.3　アセンブリーデータ構造

上図のようにマスター・スレーブ面が３次元空間に存在するとして、任意のマスター面に拘束されるスレーブ点をスレーブ面全体から検索する必要がある。この検索を高速に行うために、八分木を用いて対象スレーブ点の絞り込みを行う。

八分木は、マスター・スレーブ面全体(接合メッシュ)を覆うように定義し、接合メッシュの節点数の数に応じて八分木分割数レベルを一段ずつあげていき、マスター面と同一空間に存在するスレーブ点を検索対象としてリストアップする。

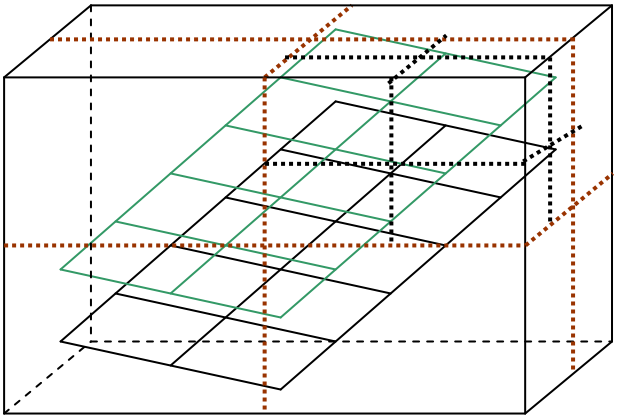


図 4.1.4　八分木とマススレーブ面

八分木により絞り込まれたスレーブ点を対象として、マスター面に平行な局所座標を持つ厚さ方向が薄い Bounding Box を定義し、更に絞り込みを行う。

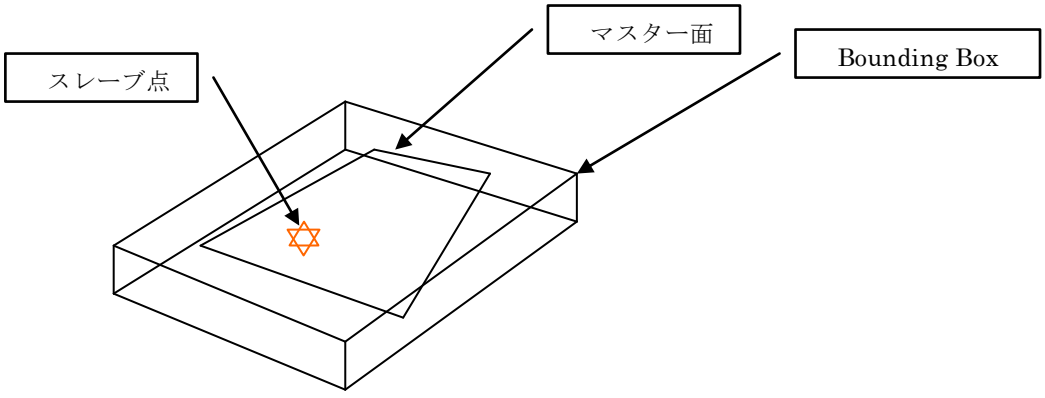


図 4.1.5　Bounding Box による絞り込み

Bounding Box 内に存在するスレーブ点をリストアップしておき、マスター面の平面内に存在するとして、面の内外判定を行い、面内に存在するスレーブ点をマスター面に所属するスレーブ点として

登録して検索を終了する。

検索の結果登録されたスレーブ点は、マスター面の各頂点への線形補間を幾何学的関係を用いて計算し、MPC ソルバーに用いる拘束式の係数を決定する。

4.2　領域分割

並列計算のための領域分割機能を提供する。

領域分割は、並列プロセス数に合わせてメッシュを分割してプロセス単位にメッシュを提供する機能である。領域分割プログラムは、FrontISTR の書式のファイルを読み込み、分割された MW3 中間ファイル書式ファイルを出力する。

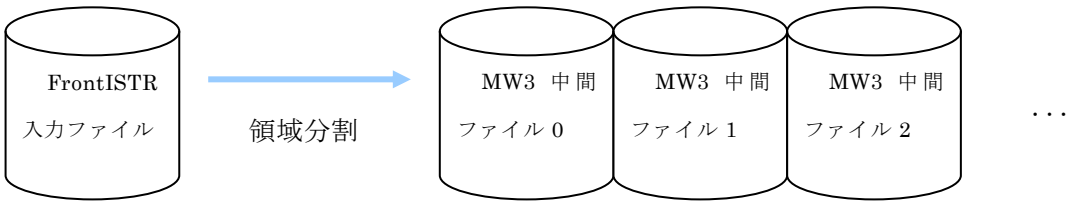


図 4.2.1　領域分割ファイル

節点分割型の領域分割は、オーバーラップ領域に要素 1 層分の厚みがあり、階層化を行うと要素 2 層分の厚みとなるが、実際に必要なのは 1 層だけである。また、輸出節点と輸入節点の並べ方が各領域で一致するようにしなければならない。

もともと、粗いメッシュにおける輸出節点と輸入節点の並びは領域間で一致している。そこで、新たに追加された節点の所属領域の決め方を統一しておけば、各領域での新しい輸出節点と輸入節点の並びを一致させることができる。

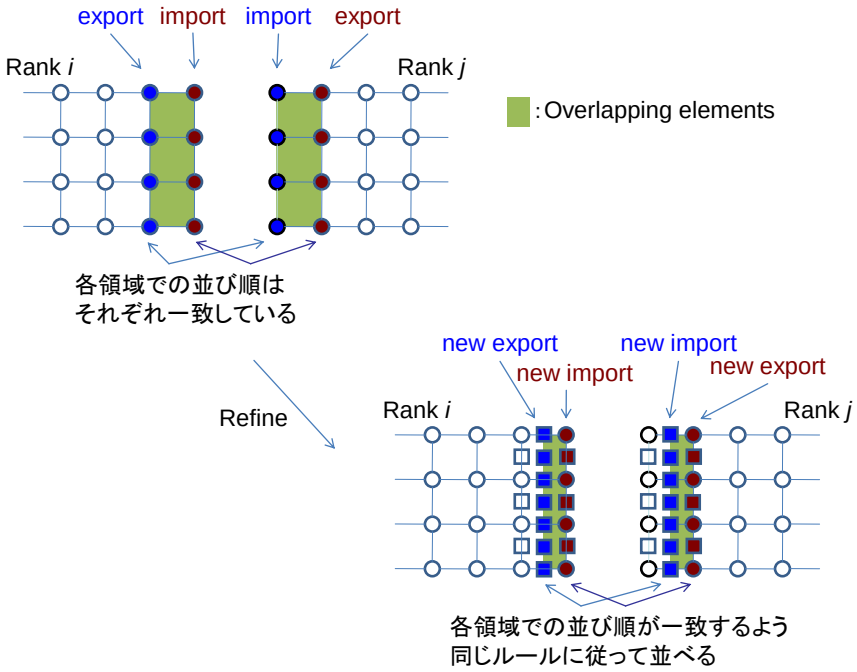


図 4.2.2 Re-construction of communication tables.

要素分割型の領域分割は、オーバーラップ領域は存在せず輸出節点と輸入節点は同一であり、階層化を行うと通信界面が分割されていくことになる。その際に新たに追加された節点の並びが同一になるように階層化を行う。マルチグリッドソルバーはこの通信方式を使用する。

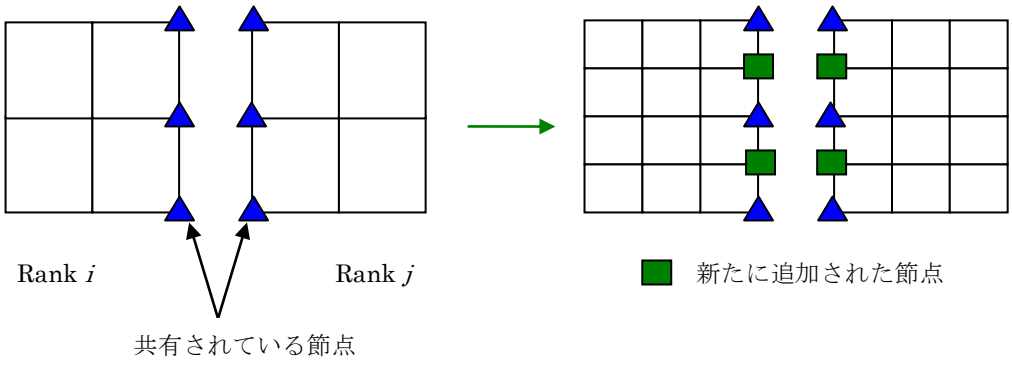


図 4.2.3 要素分割型、領域分割

4.3 MPI

本ライブラリーでは、並列処理ライブラリーとして MPI を用いている。

MPI ライブラリーは並列処理を前提としているライブラリーであるため、シングルプロセスの場合は、通常のプロセス起動となり MPI ライブラリーは不要で、並列プロセスとは異なる。

しかし本ライブラリーにおいては、シングル- 並列プロセスどちらの場合でも、同一実装にするために、MPI をラップしたモジュールを提供する。

表 4.1　ラップする MPI の機能は下記の関数群

MPI 関数	HEC_MW 関数	備考
Isend	CHecMPI::Isend	MPI 使用の有無に関わりなく CHecMPI 関数は起動可能
Irecv	CHecMPI:: Irecv	〃
Wait	CHecMPI::Wait	〃
Allreduce	CHecMPI::Allreduce	〃
Barrier	CHecMPI::Barrier	〃
Abort	CHecMPI::Abort	〃
Init	CHecMPI::Initialize	〃
Finalize	CHecMPI::Finalize	〃
Send	CHecMPI::Send	〃
Recv	CHecMPI::Recv	〃
Gather	CHecMPI::Gather	〃
Scatter	CHecMPI::Scatter	〃

MPI を有効にするためのプリプロセッサフラグは"HAVE_MPI"　(" " は不要)である。

4.4　線形代数ソルバー

並列線形ソルバーを本ライブラリーは提供する。

並列計算では、解析に先立って、全体領域のメッシュデータを領域分割し、同じフォーマットの部分領域ごとのメッシュデータを作成する。

この部分領域ごとのデータは分散領域メッシュと呼ばれ、これを並列計算に使う各プロセッサに持たせる。各プロセッサにおいては独立に剛性行列の作成がなされ、線形代数ソルバーの中で MPI を使用した領域間通信を行うことによって領域全体の整合性をとり、並列計算を実現する。

本ライブラリで利用可能な線形代数ソルバーの種類は以下である。

・ 前処理付き反復法;CG、GMRES、BiCGSTAB、GPBiCG
・ 反復法に用いる前処理 ; Jacobi、ILU、SSOR、幾何学的マルチグリッド

この中で、基本となる CG 法についてのアルゴリズムを次に示す。
アルゴリズムの詳細は理論説明書を参照されたい。

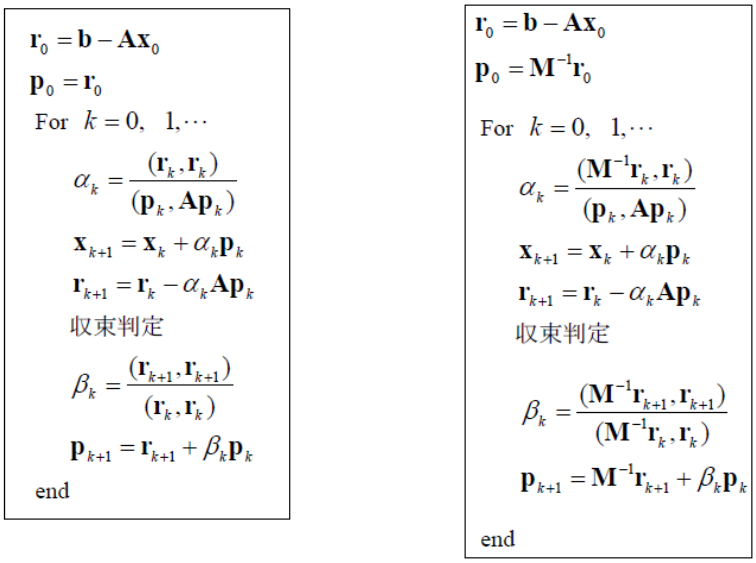


図 4.4.1　CG 法アルゴリズムおよび前処理つき CG 法アルゴリズム

4.5　ベクトル管理

本ライブラリーで解くべき線形代数方程式 A x = b の、x ,b ベクトル配列の管理を行う。

4.5.1　自由度混在ベクトルの管理

ベクトル管理機能において、任意自由度のメッシュデータのベクトル列を管理する機能を有する。

4.5.2　基礎演算機能

本ライブラリーは、ベクトルのノルム計算、ベクトル同士の内積計算、ベクトルの代入機能を提供する。

表 4.2　ベクトルに関する基礎演算機能

関数名	機能
CAssyVector::add	ベクトル同士の和
CAssyVector::subst	ベクトルの代入
CAssyVector::norm2	ベクトルのノルムの２乗
CAssyVector::innerProd	ベクトルのペアの内積

4.6　マトリックス管理

本ライブラリーで解くべき線形代数方程式 A x = b の、行列 A 疎行列の管理を行う。

4.6.1　自由度混在型マトリックスの管理

マトリックス管理機能において、任意自由度のメッシュデータのマトリックスを管理する機能を有する。ただし、アセンブリ構造に存在する複数メッシュブロックのうち、ひとつのメッシュブロックでは自由度数は一致していなければならない。

4.6.2　非ゼロプロファイルの生成

FEM のアルゴリズムにより生成される行列は疎行列と呼ばれ、そのほとんどを 0 が占めているが、実際の計算に必要な値は、非ゼロの行列成分であり、非ゼロプロファイルを生成することで計算の実行時間の短縮が図れる。そのため、非ゼロプロファイルの生成と管理を行う機能を本ライブラリーは有する。

	1	2	3	4	5	6	7	8
1	1		1		3	4		
2		2		2	3	5		
3	1		4		4		5	
4		9		3		10		
5			32		4		2	
6		1		5		6		7
7	1		1				1	8
8		3		5	6			2

```
NP, NPL, NPU   8, 10, 12
D              {1, 2, 4, 3, 4, 6, 1, 2}
AL             {1, 9, 32, 1, 5, 1, 1, 3, 5, 6}
indexL         {0: 0, 0, 1, 2, 3, 5, 7, 10}
itemL          {1, 2, 3, 2, 4, 1, 3, 2, 4, 5}
AU             {1, 3, 4, 2, 3, 5, 4, 5, 10, 2, 7, 8}
indexU         {0: 3, 6, 8, 9, 10, 11, 12, 12}
itemU          {3, 5, 6, 4, 5, 6, 5, 7, 6, 7, 8, 8}
```

図 4.6.1　CRS Format の例

要素剛性行列の足し込み

FEM アルゴリズムにおいて、最終的な線形代数方程式 A x = b の A を求めるためには、要素剛性行列を全体行列に足し込む機能が必要となる、この機能を提供する。

4.6.3 基礎演算機能

本ライブラリーは、マトリックスとベクトルの積を求める機能を提供する。また、不完全 LU 分解を行う機能を有する。

表 4.3 マトリクスに関する基礎演算機能

関数名	機能
CAssyMatrix::multVector	マトリクスとベクトルの積
CAssyMatrix::multMPC	MPC 行列とベクトルの積
CAssyMatrix::residual	マトリクスとベクトルの積とベクトルを比較する(主に Ax-b を計算する場合に利用する)
CAssyMatrix::setupPreconditioner	不完全 LU 分解を実行する
CAssyMatrix::precond	不完全 LU 分解に基づく前進消去後退代入を行う

4.7 MW3 要素ライブラリー

HEC_MW3 は、Hexa(1 次、2 次)、Tetra(1 次、2 次)、Prism(1 次、2 次)、Quad(1 次、2 次)、Triangle(1 次、2 次)、Line(1 次、2 次)の 12 種類の形状関数を所有している。

以下に、形状関数を列挙する。

4.7.1 Hexa 要素

1 次要素

節点数	8 節点
積分点数	1 点、8 点

形状関数(積分点座標：r, s, t)

$N_0 = 0.125\ (1 - r)\ (1 - s)\ (1 - t)$

$N_1 = 0.125\ (1 + r)\ (1 - s)\ (1 - t)$

$N_2 = 0.125\ (1 + r)\ (1 + s)\ (1 - t)$

$N_3 = 0.125\ (1 - r)\ (1 + s)\ (1 - t)$

$N_4 = 0.125\ (1 - r)\ (1 - s)\ (1 + t)$

$N_5 = 0.125\ (1 + r)\ (1 - s)\ (1 + t)$

$N_6 = 0.125\ (1 + r)\ (1 + s)\ (1 + t)$

$N_7 = 0.125\ (1 - r)\ (1 + s)\ (1 + t)$

2 次要素

節点数	20 節点

積分点数	1 点、8 点、27 点

形状関数(積分点座標：r, s, t)

$N_0 = -0.125(1 - r)(1 - s)(1 - t)(2+r+s+t)$

$N_1 = -0.125(1 + r)(1 - s)(1 - t)(2-r+s+t)$

$N_2 = -0.125(1 + r)(1 + s)(1 - t)(2-r-s+t)$

$N_3 = -0.125(1 - r)(1 + s)(1 - t)(2+r-s+t)$

$N_4 = -0.125(1 - r)(1 - s)(1 + t)(2+r+s-t)$

$N_5 = -0.125(1 + r)(1 - s)(1 + t)(2-r+s-t)$

$N_6 = -0.125(1 + r)(1 + s)(1 + t)(2-r-s-t)$

$N_7 = -0.125(1 - r)(1 + s)(1 + t)(2+r-s-t)$

$N_8 = 0.25(1 - r^2)(1 - s)(1 - t)$

$N_9 = 0.25(1 + r)(1-s^2)(1 - t)$

$N_{10} = 0.25(1 - r^2)(1 + s)(1 - t)$

$N_{11} = 0.25(1 - r)(1 - s^2)(1 - t)$

$N_{12} = 0.25(1-r^2)(1 - s)(1 + t)$

$N_{13} = 0.25(1 + r)(1-s^2)(1 + t)$

$N_{14} = 0.25(1-r^2)(1 + s)(1 + t)$

$N_{15} = 0.25(1 - r)(1-s^2)(1 + t)$

$N_{16} = 0.25(1 - r)(1 - s)(1-t^2)$

$N_{17} = 0.25(1 + r)(1 - s)(1-t^2)$

$N_{18} = 0.25(1 + r)(1 + s)(1-t^2)$

$N_{19} = 0.25(1 - r)(1 + s)(1-t^2)$

4.7.2 Tetra 要素

1 次要素

節点数	4 節点
積分点数	1 点

形状関数(積分点座標：L_1, L_2, L_3)

$N_0 = 1 - L_1 - L_2 - L_3$

$N_1 = L_1$

$N_2 = L_2$

$N_3 = L_3$

2 次要素

節点数	10 節点
積分点数	1 点、4 点、15 点

形状関数(積分点座標：L_1, L_2, L_3)

$a = 1.0 - L_1 - L_2 - L_3$

$N_0 = (2a - 1.0)a$

$N_1 = L_1(2L_1 - 1.0)$

$N_2 = L_2(2L_2 - 1.0)$

$N_3 = L_3(2L_3 - 1.0)$

$N_4 = 4L_1 a$

$N_5 = 4L_1L_2$

$N_6 = 4L_2 a$

$N_7 = 4L_3 a$

$N_8 = 4L_1L_3$

$N_9 = 4L_2L_3$

4.7.3 Prism 要素

1 次要素

節点数	6 節点
積分点数	2 点

形状関数(積分点座標：L_1, L_2, ξ)

$a = 1.0 - L_1 - L_2$

$N_0 = 0.5a(1.0 - \xi)$

$N_1 = 0.5L_1(1.0 - \xi)$

$N_2 = 0.5L_2(1.0 - \xi)$

$N_3 = 0.5a(1.0 + \xi)$

$N_4 = 0.5L_1(1.0 + \xi)$

$N_5 = 0.5L_2(1.0 + \xi)$

2 次要素

節点数	15 節点
積分点数	6 点、9 点、18 点

形状関数(積分点座標：L_1, L_2, ξ)

$a = 1.0 - L_1 - L_2$

$N_0 = 0.5a(1-\xi)(2a - 2-\xi)$

$N_1 = 0.5L_1(1-\xi)(2L_1 - 2-\xi)$

$N_2 = 0.5L_2(1-\xi)(2L_2 - 2-\xi)$

$N_3 = 0.5\ a(1+\xi)(2a - 2+\xi)$

$N_4 = 0.5L_1(1+\xi)(2L_1 - 2+\xi)$

$N_5 = 0.5L_2(1+\xi)(2L_2 - 2+\xi)$

$N_6 = 2L_1a(1-\xi)$

$N_7 = 2L_1L_2(1-\xi)$

$N_8 = 2L_2\ a(1-\xi)$

$N_9 = 2L_1\ a(1+\xi)$

$N_{10} = 2L_1L_2(1+\xi)$

$N_{11} = 2L_2\ a(1+\xi)$

$N_{12} = a(1-\xi^2)$

$N_{13} = L_1(1-\xi^2)$

$N_{14} = L_2(1-\xi^2)$

4.7.4 Quad 要素 エンティティ番号

1 次要素

節点数	4 節点
積分点数	1 点

形状関数(積分点座標：r, s)

$N_0 = 0.25(1-r)(1-s)$

$N_1 = 0.25(1+r)(1-s)$

$N_2 = 0.25(1+r)(1+s)$

$N_3 = 0.25(1-r)(1+s)$

2 次要素

節点数	8 節点
積分点数	4 点、9 点

形状関数(積分点座標：r, s)

$N_0 = 0.25(1 - r)(1 - s)(-1-r-s)$

$N_1 = 0.25(1 + r)(1 - s)(-1+r-s)$

$N_2 = 0.25(1 + r)(1 + s)(-1+r+s)$

$N_3 = 0.25(1 - r)(1 + s)(-1-r+s)$

$N_4 = 0.5(1-r^2)(1-s)$

$N_5 = 0.5(1-s^2)(1+r)$

$N_6 = 0.5(1-r^2)(1+s)$

$N_7 = 0.5(1-s^2)(1-r)$

4.7.5 Triangle 要素エンティティ番号

1 次要素

節点数	3 節点
積分点数	1 点

形状関数(積分点座標：r, s)

$N_0 = r$

$N_1 = s$

$N_2 = 1 - r - s$

2 次要素

節点数	6 節点
積分点数	3 点

形状関数(積分点座標：r, s)

$t = 1-r-s;$

$N_0 = t(2t-1)$

$N_1 = r(2r-1)$

$N_2 = s(2s-1)$

$N_3 = 4rt$

$N_4 = 4rs$

$N_5 = 4st$

4.7.6 Beam 要素エンティティ番号

1 次要素

節点数	2 節点
積分点数	1 点

形状関数(積分点座標：r)

$N_0 = 0.5(1-r)$

$N_1 = 0.5(1+r)$

2 次要素

節点数	3 節点
積分点数	2 点

形状関数(積分点座標：r)

$N_0 = -0.5(1-r)r$

$N_1 = 0.5(1+r)r$

$N_2 = 1-r^2$

4.8　計算結果の可視化

本ライブラリーによる計算結果の可視化は、HEC_MW2 において開発された VISUALIZER を継承する。VISUALIZER は、本ライブラリーを用いて解析された結果を、エンドユーザの入力したデータ書式に基づき任意視点からのコンター図、等値面図をビットマップデータとして出力する。また、本リリース版のサンプルプログラムでは、UCD ファイルも出力される。

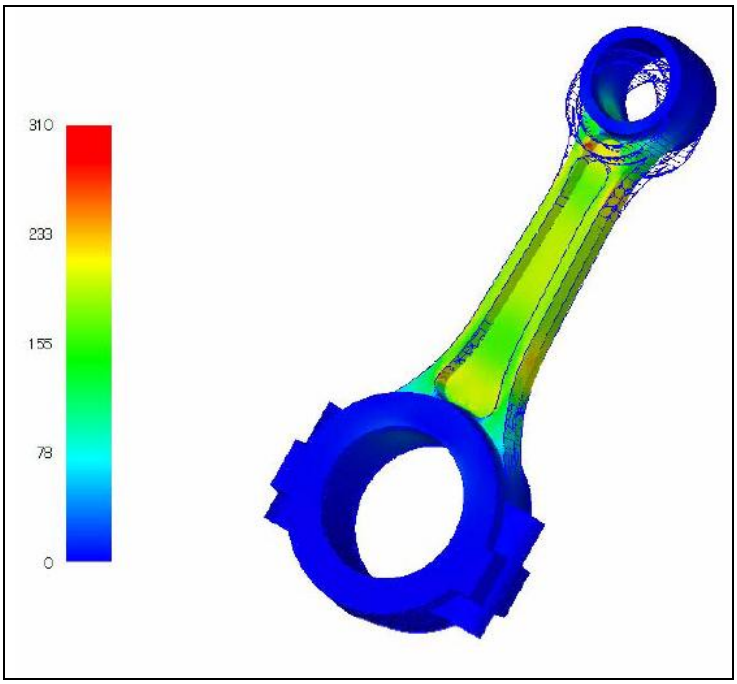


図 4.8.1　コンター図の表示例

図 4.8.2　等値面の表示例

4.8.1　入力ファイル

MW システム全体の入力ファイルは FrontISTR 形式のファイルが入力ファイルとなる。
MW3 の入力ファイルは、領域分割プログラムによって生成される中間ファイルである。

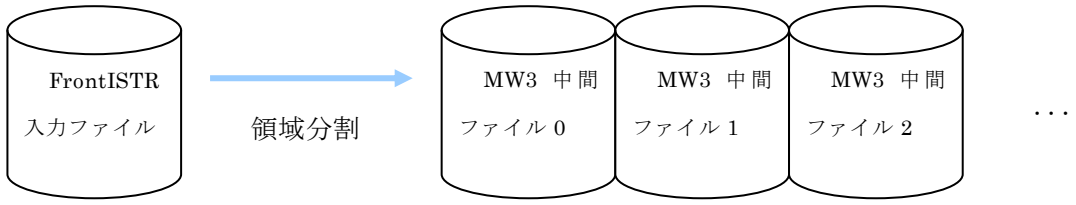


図 4.8.3　領域分割ファイル

表 4.4　FrontISTR タグの内容（メッシュファイル）

タグ	内容
!HEADER	メッシュファイルのタイトル
!ZERO	絶対零度
!NODE	節点情報
!ELEMENT	要素情報
!NGROUP	節点グループ
!EGROUP	要素グループ
!SGROUP	面グループ
!EQUATION	拘束点情報
!AMPLITUDE	非定常荷重
!SECTION	セクション情報
!MATERIAL	材料情報
!INITIAL CONDITION	初期条件
!INCLUDE	外部ファイル入力
!CONNECTIVITY	コネクティビティ
!END	メッシュデータの終端

リスタートファイルは、本バージョンでは未対応であり、次期バージョンにて対応する予定である。

4.8.2 出力ファイル

4.8.2.1. 計算結果ファイル

未対応

4.8.2.2. リスタートファイル

未対応

4.8.2.3. 可視化ビットマップ

VISUALIZER により計算結果をビットマップとして出力。

5. MW3 を利用したプログラムの記述方法

HEC_MW3 の使い方をサンプルコードを用いて説明する。

5.1 MW3 API を利用した線形静解析のサンプルプログラム

MW3-API を利用して 3 次元の線型弾性プログラムを記述した例を示す。このサンプルプログラムは、本ソフトウェアのライブラリの使用例ですので、本サンプルプログラムにはつぎのような制約条件があります。

- 六面体 1 次要素に限定したプログラムとなっている。
- 境界条件については、ハードコーディングしてある。

5.1.1 メインプログラム

```
// MW3 用インクルードファイル
#include "./HEC_MW3.h"

int main(int argc, char** argv)
{

// MW3 の生成(コンストラクタ)
CMW *pMW = CMW::Instance();

//  物性の指定
double nyu = 0.3, E = 1.0E9;
double coef = ((1.0-nyu)*E)/((1.0+nyu)*(1.0-2*nyu));

// D 行列の設定
double D[6][6] = { {coef, coef*nyu/(1.0-nyu), coef*nyu/(1.0-nyu), 0, 0, 0},
                    {coef*nyu/(1.0-nyu), coef, coef*nyu/(1.0-nyu), 0, 0, 0},
                   {coef*nyu/(1.0-nyu), coef*nyu/(1.0-nyu), coef, 0, 0, 0},
                   {0, 0, 0, coef*(1-2*nyu)/(2*(1.0-nyu)), 0, 0},
                   {0, 0, 0, 0, coef*(1-2*nyu)/(2*(1.0-nyu)), 0},
                   {0, 0, 0, 0, 0, coef*(1-2*nyu)/(2*(1.0-nyu))} };
```

```
// メインプログラムでの作業用変数の宣言
vvvdouble dNdx;
double weight ,detJ;
double ElemMatrix[24][24];
double B[6][24], DB[6][24], BDB[24*24];
int iElem, iElemMax, iNode, iNodeMax, i, j, k;
int iLocalNode, numOfLocalNode, igauss, numOfInteg;

// 要素タイプの指定；ここでは六面体一次要素の設定
const uint ishape = ElementType::Hexa;

// 入力ファイルのディレクトリ指定 mw3.cnt のファイルパスを指定
const char* cpath= "./";

// MW3 用データ構造の設定
pMW->Initialize(argc, argv, cpath);
pMW->FileRead();
pMW->Refine();
pMW->FileWrite();

// FEM 用のデータ設定およびデータ構造の設定
//  ①行列解法のための BCRS 行列の領域確保と初期化、およびインデクス作成
//  ②右辺ベクトルの初期化
//  ③操作する対象の指定（本プログラムではハードコードされている）
pMW->Initialize_Matrix();
pMW->Initialize_Vector();
pMW->SelectAssembleModel( 0 );
pMW->SelectMeshPart_ID( 0 );

// MW3 用データ構造から要素数と節点数を設定
pMW->NumOfIntegPoint(ishape, numOfInteg);
iElemMax = pMW->getElementSize();
iNodeMax = pMW->getNodeSize();

// 要素のループの始まり
for( iElem =0; iElem < iElemMax; iElem++){
```

アセンブルモデルの選択
引数:マルチグリッドレベル

メッシュパーツの選択
引数:パーツ番号

```
// 要素剛性の初期化
        for(i=0; i<24; i++) for(j=0; j<24; j++) ElemMatrix[i][j] = 0.0;

// 要素毎に必要となる dN/dx を計算する。すべての積分点に対する値を計算する。
        pMW->dNdx(ishape, numOfInteg, iElem, dNdx);

// 積分点のループの始まり
        for( igauss=0; igauss<numOfInteg; igauss++){

// B 行列の設定
                for(iLocalNode=0; iLocalNode < numOfLocalNode; iLocalNode++){
                        B[0][iLocalNode*3     ] = dNdx[igauss][iLocalNode][0];
                        B[0][iLocalNode*3 +1] = 0.0;
                        B[0][iLocalNode*3 +2] = 0.0;
                        B[1][iLocalNode*3     ] = 0.0;
                        B[1][iLocalNode*3 +1] = dNdx[igauss][iLocalNode][1];//dNdY
                        B[1][iLocalNode*3 +2] = 0.0;
                        B[2][iLocalNode*3     ] = 0.0;
                        B[2][iLocalNode*3 +1] = 0.0;
                        B[2][iLocalNode*3 +2] = dNdx[igauss][iLocalNode][2];//dNdZ
                        B[3][iLocalNode*3     ] = dNdx[igauss][iLocalNode][1];//dNdY
                        B[3][iLocalNode*3 +1] = dNdx[igauss][iLocalNode][0];//dNdX
                        B[3][iLocalNode*3 +2] = 0.0;
                        B[4][iLocalNode*3     ] = 0.0;
                        B[4][iLocalNode*3 +1] = dNdx[igauss][iLocalNode][2];//dNdZ
                        B[4][iLocalNode*3 +2] = dNdx[igauss][iLocalNode][1];//dNdY
                        B[5][iLocalNode*3     ] = dNdx[igauss][iLocalNode][2];//dNdZ
                        B[5][iLocalNode*3 +1] = 0.0;
                        B[5][iLocalNode*3 +2] = dNdx[igauss][iLocalNode][0];//dNdX
                }

// DB の計算
                for(i=0; i<6; i++) {
                        for(j=0; j<24; j++) {
                                DB[i][j] = 0.0;
                                for(k=0; k<6; k++) {
                                        DB[i][j] += D[i][k]*B[k][j];
                                }
```

iElem 番-要素の導関数を参照変数に代入

MW3 から取得した **dNdx** を使用して B 行列を計算

```
                    }
                }

// BtDB の計算
                for(i=0; i<24; i++) {
                    for(j=0; j<24; j++) {
                        BDB[i][j] = 0.0;
                        for(k=0; k<6; k++) {
                            BDB[i][j] += B[k][i]*DB[k][j];
                        }
                    }
                }

// 積分点毎に BtDB の値を加算し、要素剛性を作成する
                pMW->detJacobian(ishape, numOfInteg, igauss, detJ);
                pMW->Weight(ishape, numOfInteg, 0, weight);
                for(i=0;i<24;i++)
                    for(j=0;j<24;j++) ElemMatrix[i][j] += BDB[i][j] * weight * detJ;
        }

// 各要素で作成できた要素剛性を全体剛性に足し込む
        pMW->Matrix_Add_Elem(0,  0,  iElem,  ElemMatrix);
}

// 境界条件の設定
pMW->Sample_Set_BC();

// 反復法による求解
uint max_iteration = 1000;
double tolerance = 1.0e-8;
uint method = 1;          // 1:CG, 2:BiCBSTAB, 3:GPBiCG, 4:GMRES
uint precondition = 1; // 1:Jacobi, 2:MG, 3:SSOR, 4:ILU
pMW->Solve( max_iteration, tolerance, method, precondition);

// MW3 の終了処理
pMW->Finalize();
}
```

一時的な関数、次ページで説明

図 5.1.1　サンプルプログラム

5.1.2　境界条件の設定

境界条件の設定を Sample_Set_BC()で行っている。

pMW->Sample_Set_BC()；について

境界条件の指定方法は未実装(バージョンアップ時に対応予定)のため、基本的な関数を用意しておき、それを呼び出す。上記のテストメインで利用している例題の関数を下記に示す。次期バージョンでは、下記のサンプルで利用されている Set_BC()を利用して、境界条件を設定する。

```
void CMW::Sample_Set_BC()
{
// この関数の作業用変数
        double X, Y, Z, value0, value1, value2;

        int iNodeMax = getNodeSize();
        for( int iNode = 0; iNode < iNodeMax; iNode++){
                X = mpMesh->getNodeIX(iNode)->getX();
                Y = mpMesh->getNodeIX(iNode)->getY();
                Z = mpMesh->getNodeIX(iNode)->getZ();

// 片持ち梁の他の 1 端(Z=4)の X 上端(X=1)に 0 番目(X)の自由度に荷重を与える。
// 全体剛性は操作せず、右辺の荷重項に value0 を設定する関数
// メインで定義した 0 番目の行列（全体剛性）に対する操作
                if(abs( Z - 4.0 ) < 1.0e-5 && abs( X - 1.0 ) < 1.0e-5 ) {
                        value0 = 1.0e6;
                        Set_BC( 0, iNode, 0, value0);
                };

// 片持ち梁の他の 1 端(Z=0)の 3 自由度を固定する。
// 全体剛性の対角に value1 を設定し、右辺の荷重項に value2 を設定する関数
// メインで定義した 0 番目の行列（全体剛性）と 0 番目のベクトルに対する操作
                if(abs( Z ) < 1.0e-5) {
                        value1 = 1.0e15;
                        value2 = 0.0;
                        Set_BC( 0, 0, iNode, 0, value1, value2);
                        Set_BC( 0, 0, iNode, 1, value1, value2);
                        Set_BC( 0, 0, iNode, 2, value1, value2);
                }
        };
}
```

図 5.1.2　境界条件のサンプルプログラム

5.2　使用例 1；片持ち梁の基本例題

5.2.1　計算条件

次のような片持ち梁の形状でテストを実施した。

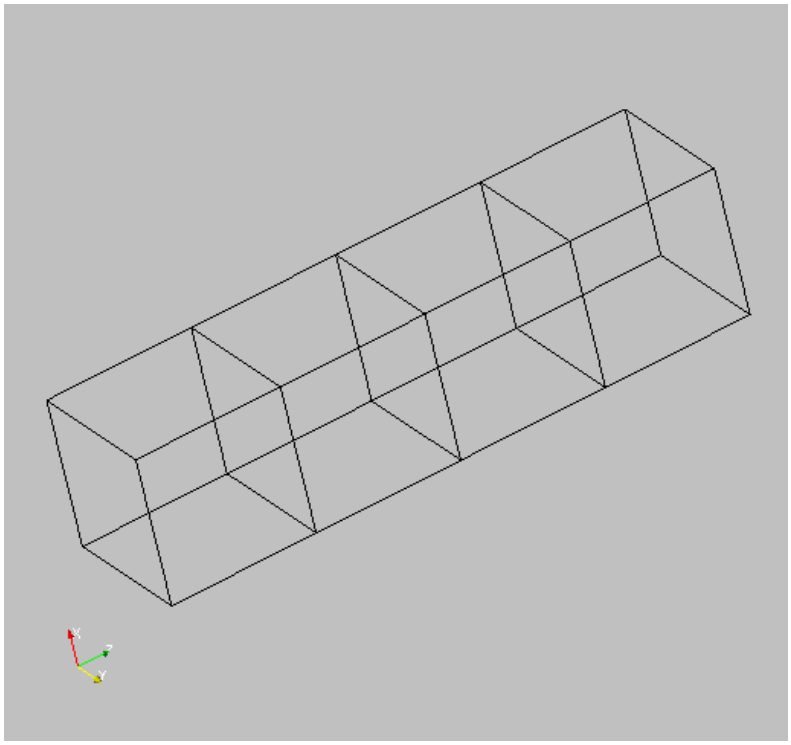


図 5.2.1　片持ち梁形状

5.2.2　入力データ

```
Refine
   0
End

AssyModel
   1   0   0
   0
End

Node
         20          0         19          0
V 0 3       0       0.000       0.000       0.000
V 0 3       1       1.000       0.000       0.000
V 0 3       2       1.000       1.000       0.000
V 0 3       3       0.000       1.000       0.000
V 0 3       4       0.000       0.000       1.000
V 0 3       5       1.000       0.000       1.000
V 0 3       6       1.000       1.000       1.000
V 0 3       7       0.000       1.000       1.000
V 0 3       8       0.000       0.000       2.000
V 0 3       9       1.000       0.000       2.000
V 0 3      10       1.000       1.000       2.000
V 0 3      11       0.000       1.000       2.000
V 0 3      12       0.000       0.000       3.000
V 0 3      13       1.000       0.000       3.000
```

```
V 0 3    14       1.000       1.000       3.000
V 0 3    15       0.000       1.000       3.000
V 0 3    16       0.000       0.000       4.000
V 0 3    17       1.000       0.000       4.000
V 0 3    18       1.000       1.000       4.000
V 0 3    19       0.000       1.000       4.000
End

Element
        4         0         3         0
Hexa      0      0      1      2      3      4      5      6      7
Hexa      1      4      5      6      7      8      9     10     11
Hexa      2      8      9     10     11     12     13     14     15
Hexa      3     12     13     14     15     16     17     18     19
End
```

図 5.2.2　片持ち梁入力データ

5.2.3　計算結果

(1) 標準出力

標準出力に次のような結果が出力される。

```
myRank => 0,number of process => 1
Logger                  _____________________________________________Remove old LogFile
 ---------------------------------------------------------------------------
|                                                                           |
|                                  HEC_MW3                                    |
|                                                                           |
|                                                                           |
|                              FEM Static Library                          |
|                                                 Version 0.0               |
|                                                                           |
|                                                                           |
|                                                                2009.05.12 |
|                                                                2009.05.12 |
 ---------------------------------------------------------------------------
Info                    ____________________________________________ HEC_MW3 Initialized
Debug                     ___________________________________________ファイル名=> beam
Info                    ____________________________________________ HEC_MW3 FileRead
Info                    ____________________________________________ HEC_MW3 FileWrite
FileWriterChunk,    ilevel= 0
- Mesh-Level loop ( 0 / 1 )
initialize in CMatrixBCRS 3 20
mMGLevel 0
numOfContact 0
- Mesh loop ( 0 / 1 )
 --- start of CG solver --- 1
 [TYPE:2] Preconditioner
iteration:       1, residue:       1.690050e+00
iteration:       2, residue:       1.891452e+00
iteration:       3, residue:       2.283102e+00
iteration:       4, residue:       4.318988e+00
iteration:       5, residue:       1.444613e+00
iteration:       6, residue:       3.021775e-01
```

```
iteration:       7, residue:       1.611362e-01
iteration:       8, residue:       1.189049e-01
iteration:       9, residue:       1.068138e-01
iteration:      10, residue:       1.001430e-01
iteration:      11, residue:       6.058775e-02
iteration:      12, residue:       6.483495e-02
・・・途中省略
iteration:      24, residue:       1.037083e-06
iteration:      25, residue:       1.956918e-07
iteration:      26, residue:       3.628898e-08
iteration:      27, residue:       2.098408e-09
iteration:      28, residue:       1.595392e-10
iteration:      29, residue:       1.333973e-11
iteration:      30, residue:       6.283148e-13
 --- end of CG solver ---
 --- end of output to a UCD file. (outUCD0000_00000.inp) ---
Info                    _____________________________________________ HEC_MW3 Finalized
~CMeshFactory
~CMesh start,    mMGLevel==0
~CMesh     end,    mMGLevel==0
~CIndexBucket
~CAssyModel
~CIndexBucketMesh
~CGMGModel
```

図 5.2.3　片持ち梁標準出力

(2)　可視化結果

本サンプルプログラムから、先に示した片持ち梁の例題について、出力された可視化ファイルを可視化したサンプルである。

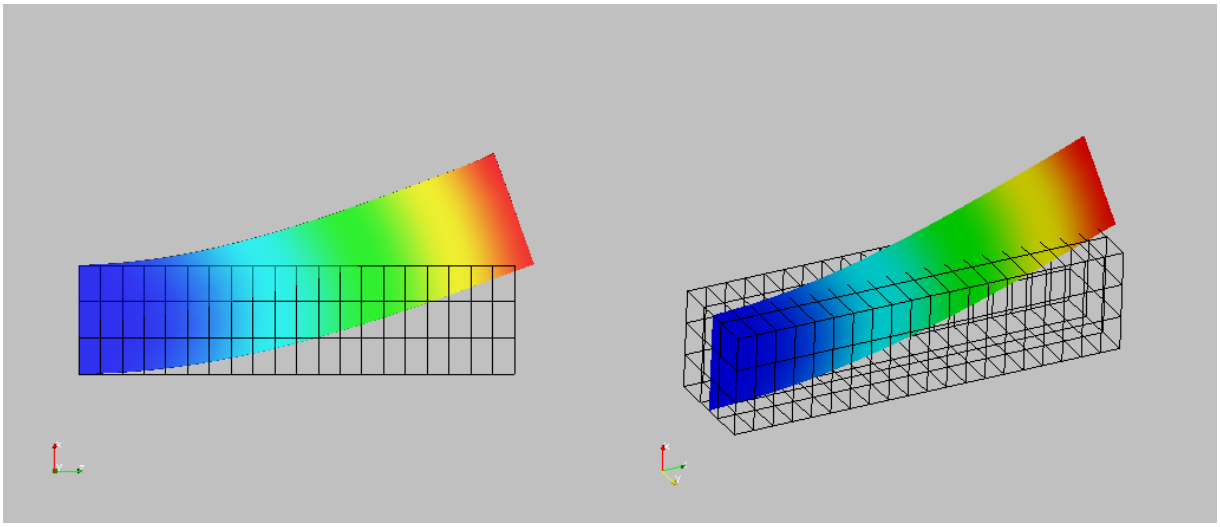


図 5.2.4　片持ち梁解析結果の可視化（その 1）

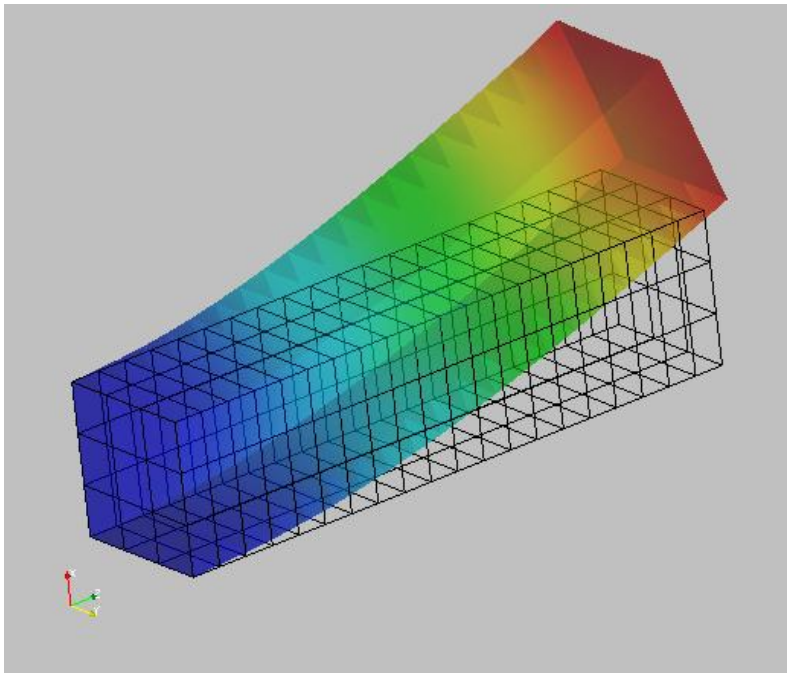


図 5.2.5　片持ち梁解析結果の可視化（その 2）

5.3　使用例 2；リファイナを利用した計算

5.3.1　計算条件

次のような片持ち梁の形状でテストを実施した。

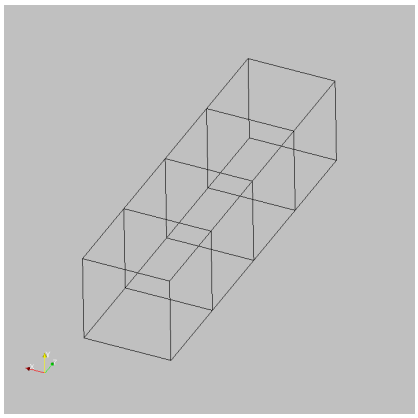
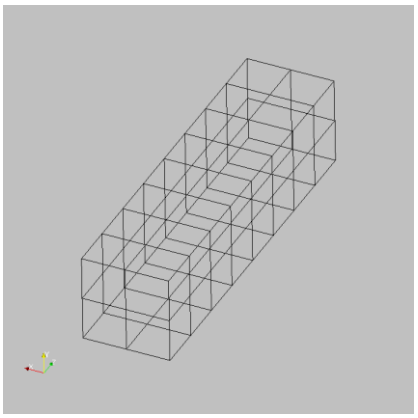
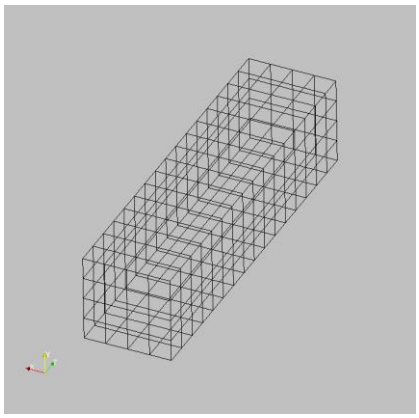
		
レベル 0	レベル 1	レベル 2

図 5.3.1　リファイナを用いたデータ形状

5.3.2　入力データ

```
Refine
  3    ← リファインのレベルを指定する（このデータを変更するのみ）
End

AssyModel
  1  0  0
  0
End

Node
        20          0         19          0
V 0 3      0       0.000       0.000       0.000
V 0 3      1       1.000       0.000       0.000
V 0 3      2       1.000       1.000       0.000
V 0 3      3       0.000       1.000       0.000
V 0 3      4       0.000       0.000       1.000
V 0 3      5       1.000       0.000       1.000
V 0 3      6       1.000       1.000       1.000
V 0 3      7       0.000       1.000       1.000
V 0 3      8       0.000       0.000       2.000
V 0 3      9       1.000       0.000       2.000
V 0 3     10       1.000       1.000       2.000
V 0 3     11       0.000       1.000       2.000
V 0 3     12       0.000       0.000       3.000
V 0 3     13       1.000       0.000       3.000
V 0 3     14       1.000       1.000       3.000
V 0 3     15       0.000       1.000       3.000
V 0 3     16       0.000       0.000       4.000
V 0 3     17       1.000       0.000       4.000
V 0 3     18       1.000       1.000       4.000
V 0 3     19       0.000       1.000       4.000
End

Element
        4          0          3          0
Hexa       0       0       1       2       3       4       5       6       7
Hexa       1       4       5       6       7       8       9      10      11
Hexa       2       8       9      10      11      12      13      14      15
Hexa       3      12      13      14      15      16      17      18      19
End
```

図 5.3.2　リファイナを利用する入力データ

5.3.3　計算結果

各レベルの結果を示す。(図において、変位は倍率を乗じているが、この図では、その倍率はレベル毎に異なる値を利用している。)

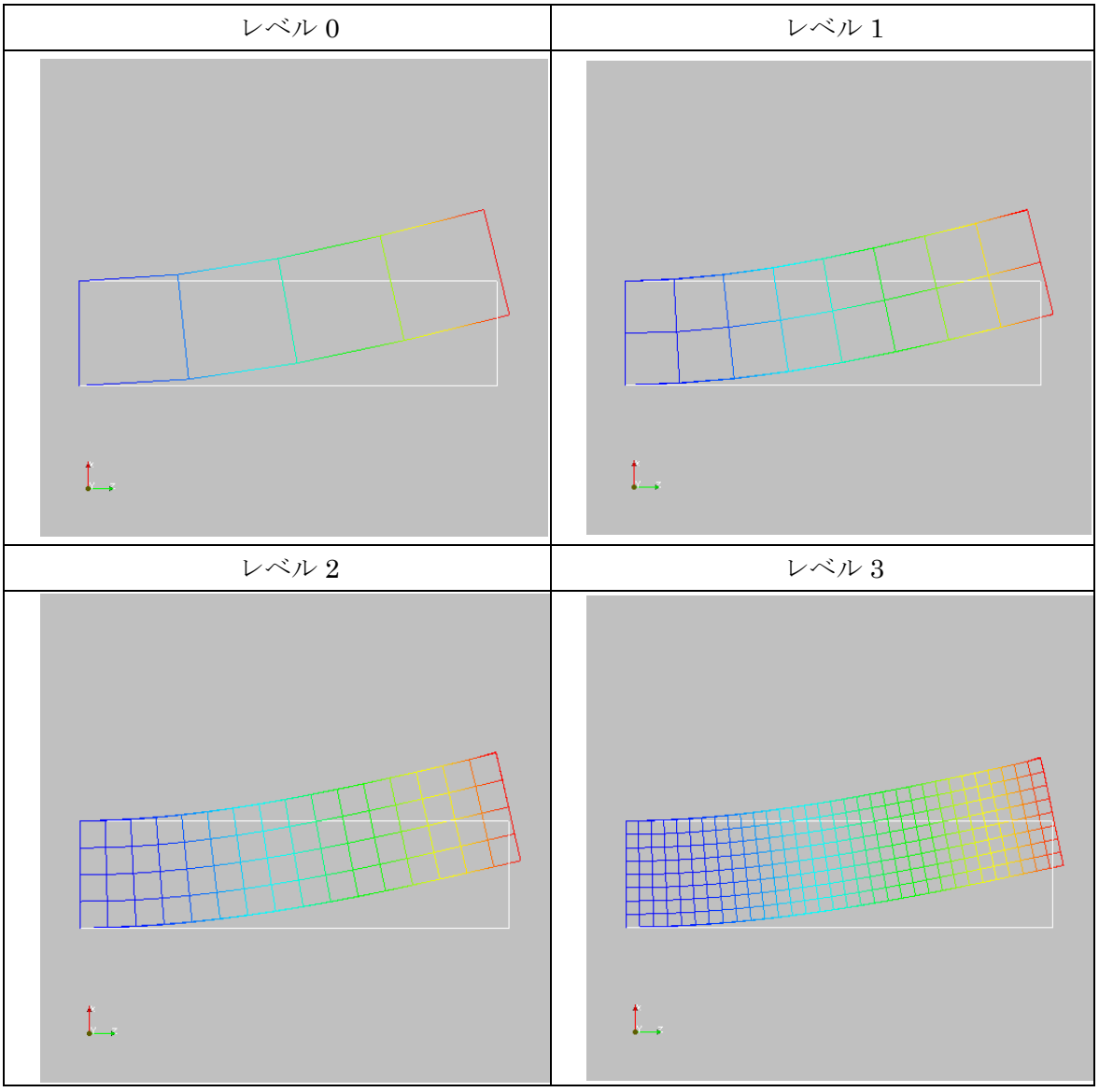


図 5.3.3　リファイナを利用した解析結果

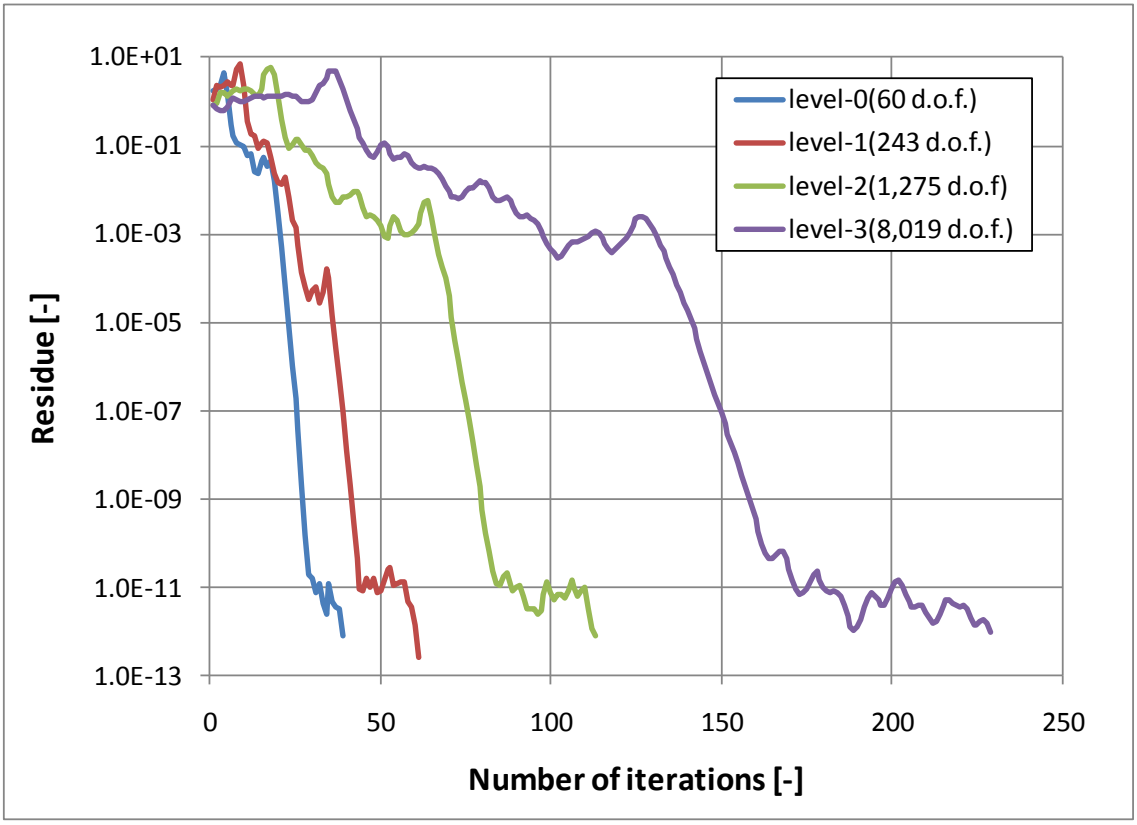


図 5.3.4　リファイナを利用したケースの収束状況

また、参考までにレベル 5 の結果を示す。

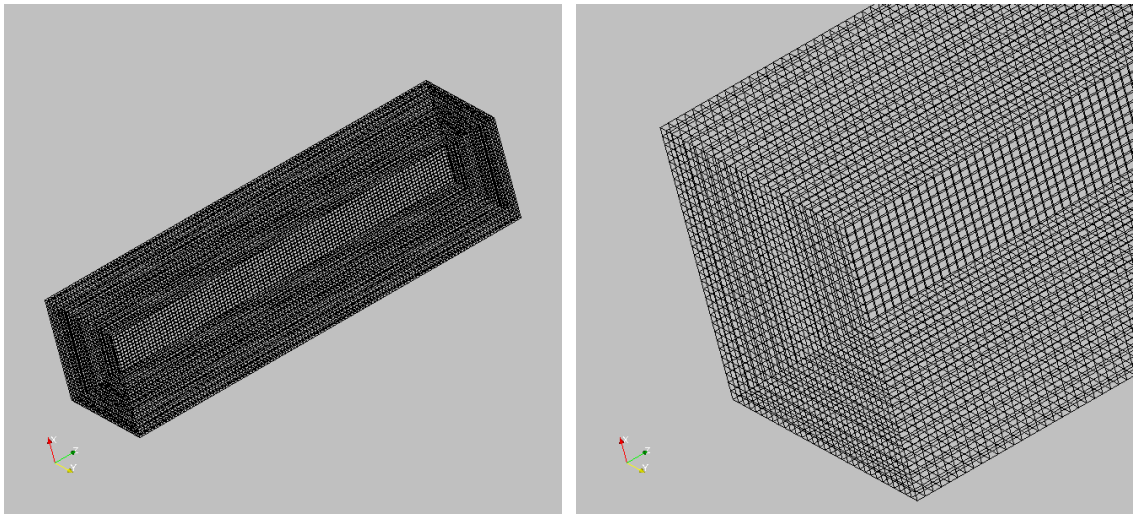


図 5.3.5　レベル 5 の格子

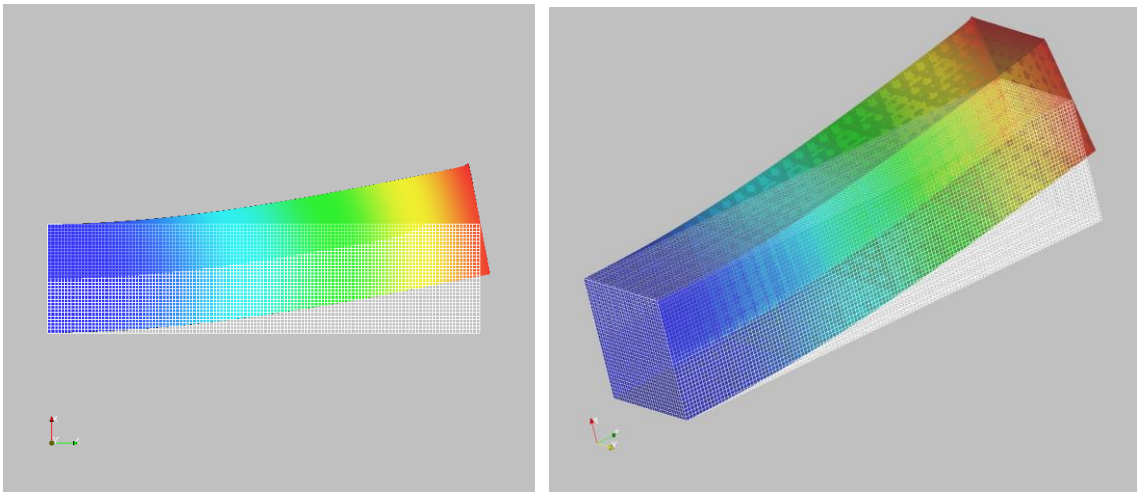


図 5.3.6　レベル 5 の計算結果

5.4　使用例 3；MPC を利用した計算（連続メッシュ）

5.4.1　MPC 前処理付き反復法について

MPC 条件の組み込みに際しては、組み込みの容易さから、ペナルティ法が採用されることが多い。しかし、この場合、方程式が悪条件となり、反復法による求解が難しく、直接法の利用がより適しています。一方、大規模問題を並列環境で解く場合、直接法の利用は不可能であり、反復法の利用が必須となります。

このため、ペナルティ法に変わる MPC 条件の組み込み方法として、自由度消去法を導入しました。自由度消去法を用いた場合、MPC により接合される各パーツが一体のものとしてモデル化された場合と等価な方程式となるため、ペナルティ法のように方程式が悪条件となることがありません。

通常、自由度消去法による MPC 条件の組み込みは、マトリックスの非ゼロのプロファイルが変更されます。HEC-MW のマトリックスは非ゼロ成分のみを保持している（CRS：Compressed Row Storage）ため、非ゼロのプロファイルを変更するためには煩雑な処理が必要となり、望ましくありません。そこで、以下に示す方法により、マトリックス自体には MPC 条件を組み込まず、ソルバーの各反復の中で MPC による拘束自由度を消去する方法を採用いたしました。

5.4.2　MPC のプログラム内部での処理について

解くべき方程式は以下のように表されます。

$$\mathbf{Ku} = \mathbf{f} \tag{5.1}$$

$$\mathbf{Bu}=\mathbf{g} \tag{5.2}$$

ここで、$\mathbf{K}$ は係数マトリックス（$N\times N$）、$\mathbf{u}$ は未知数ベクトル（$1\times N$）、$\mathbf{f}$ は右辺ベクトル（$1\times N$）、$\mathbf{B}$ および $\mathbf{g}$ はそれぞれ多点拘束条件を表す係数マトリックスおよび定数ベクトルであり、多点拘束条件の数を M とすると、大きさはそれぞれ $M\times N$ および $1\times M$ です。

(5.2)から M 個の従属自由度を決め、それらを消去し、それ以外のの独立自由度についての方程式を解くことを考えます。

全自由度のうち、独立自由度のみが意味を持つ未知数ベクトルを $\mathbf{u}'$（$1\times N$） 、$\mathbf{u}$ と $\mathbf{u}'$ との間の変換行列を $\mathbf{T}$ とすると、(5.2)の拘束条件は

$$\mathbf{u}=\mathbf{T}\mathbf{u}'+\mathbf{u}_g \tag{5.3}$$

の形で表すことができます。ただし、$\mathbf{u}_g$ は $\mathbf{B}$ および $\mathbf{g}$ から決まる定数ベクトルです。

たとえば、$N=5$、$u_5-u_4=1$ の場合、多点拘束条件は、(5.2)の形式では、

$$\begin{bmatrix}0 & 0 & 0 & 1 & -1\end{bmatrix}\begin{Bmatrix}u_4\\u_5\end{Bmatrix}=\begin{bmatrix}1\end{bmatrix}$$

(1.8-3)の形式では、

$$\begin{Bmatrix}u_1\\u_2\\u_3\\u_4\\u_5\end{Bmatrix}=\begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&1&0\end{bmatrix}\begin{Bmatrix}u_1'\\u_2'\\u_3'\\u_4'\\u_5'\end{Bmatrix}+\begin{Bmatrix}0\\0\\0\\0\\1\end{Bmatrix}$$

と表わされます。

(5.3)を(5.1)に代入し、定数項を右辺に移項すると、

$$\mathbf{KT}\mathbf{u}'=\mathbf{f}-\mathbf{K}\mathbf{u}_g$$

さらに、係数行列を対称にするため、両辺の左から $\mathbf{T}^T$ をかけると

$$\mathbf{T}^T\mathbf{K}\mathbf{T}\mathbf{u}'=\mathbf{T}^T\left(\mathbf{f}-\mathbf{K}\mathbf{u}_g\right) \tag{5.4}$$

が得られる。(5.4)が、最終的に解くべき、多点拘束を組み込んだ方程式です。

(5.4)に対して反復解法を適用する。その際、$\mathbf{T}^T\mathbf{K}\mathbf{T}$ を陽には保持せず、$\mathbf{T}$ や $\mathbf{T}^T$ との掛け算を必要に応じてその都度行う。通常の反復法におけるマトリックス・ベクトル積１回について、本手法では $\mathbf{T}$ および $\mathbf{T}^T$ との積の計算が増えることになるが、$\mathbf{T}$ はほとんど単位行列であるため、その計算コストは低く抑えることが可能です。

なお、(5.4)において、$\mathbf{T}^T\mathbf{K}\mathbf{T}$ の従属自由度に関する行および列にはすべて０が入ることになり、解が唯一ではありません。しかし、反復解法を適用した場合、未知ベクトルの初期値として従属自由度成分を０とすることで、修正ベクトル・残差ベクトルともに従属自由度成分は常に０となり、従属自

由度成分を無視した形での求解が可能です。

したがって、これらの処理をプログラム内部で実行する場合には、行列ベクトル積の

$$\mathbf{K}x$$

の計算をする替わりに

$$\mathbf{T}^T\mathbf{K}\mathbf{T}x$$

とすればいいことがわかります。

5.4.3　計算条件

次のような片持ち梁の形状でテストを実施した。

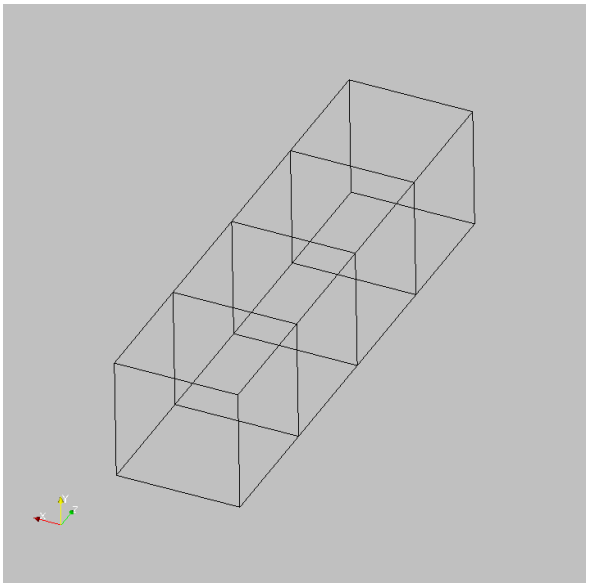


図 5.4.1　MPC を利用した連続メッシュ形状

5.4.4　入力データ

```
Refine
   0
End

AssyModel
   2   1   0
   0
   1
End

Node
          12          0         11          0
V 0 3       0       0.000       0.000       0.000
V 0 3       1       1.000       0.000       0.000
V 0 3       2       1.000       1.000       0.000
V 0 3       3       0.000       1.000       0.000
```

```
  V 0 3      4        0.000        0.000        1.000
  V 0 3      5        1.000        0.000        1.000
  V 0 3      6        1.000        1.000        1.000
  V 0 3      7        0.000        1.000        1.000
  V 0 3      8        0.000        0.000        2.000
  V 0 3      9        1.000        0.000        2.000
  V 0 3     10        1.000        1.000        2.000
  V 0 3     11        0.000        1.000        2.000
  End

  Element
           2          0          1          0
  Hexa       0       0       1       2       3       4       5       6       7
  Hexa       1       4       5       6       7       8       9      10      11
  End

  Node
          12          1         11          0
  V 0 3      0        0.000        0.000        2.000
  V 0 3      1        1.000        0.000        2.000
  V 0 3      2        1.000        1.000        2.000
  V 0 3      3        0.000        1.000        2.000
  V 0 3      4        0.000        0.000        3.000
  V 0 3      5        1.000        0.000        3.000
  V 0 3      6        1.000        1.000        3.000
  V 0 3      7        0.000        1.000        3.000
  V 0 3      8        0.000        0.000        4.000
  V 0 3      9        1.000        0.000        4.000
  V 0 3     10        1.000        1.000        4.000
  V 0 3     11        0.000        1.000        4.000
  End

  Element
           2          1          1          0
  Hexa       0       0       1       2       3       4       5       6       7
  Hexa       1       4       5       6       7       8       9      10      11
  End

  ContactMesh
    1 0 0
    0 0 0
    8 7 0
  0     0.0     0.0     2.0     0   8   0 0 v   3   0
  1     1.0     0.0     2.0     0   9   0 0 v   3   0
  2     1.0     1.0     2.0     0 10   0 0 v   3   0
  3     0.0     1.0     2.0     0 11   0 0 v   3   0
  4     0.0     0.0     2.0     1   0   0 1 v   3   0
  5     1.0     0.0     2.0     1   1   0 1 v   3   0
  6     1.0     1.0     2.0     1   2   0 1 v   3   0
  7     0.0     1.0     2.0     1   3   0 1 v   3   0
    1 0 0
    0   0 1 1   Quad 0 1 2 3 0
    1 0 0
    0   1 0 0   Quad 4 5 6 7 0
End
```

図 5.4.2　MPC を利用した連続メッシュ形状の入力データ

5.4.5　計算結果

表 5.1　MPC ありと MPC なしの計算結果の比較

MPC あり	MPC なし
0 : 4.992890e-10 8.266443e-10 4.000001e-09 1 : 5.007112e-10 -8.263335e-10 -4.000001e-09 2 : 5.007112e-10 8.263335e-10 -4.000001e-09 3 : 4.992890e-10 -8.266443e-10 4.000001e-09 4 : 3.110572e-02 4.895740e-03 2.590097e-02 5 : 3.109334e-02 -4.901381e-03 -2.589808e-02 6 : 3.109334e-02 4.901381e-03 -2.589808e-02 7 : 3.110572e-02 -4.895740e-03 2.590097e-02 8 : 1.078342e-01 2.438936e-03 4.566668e-02 9 : 1.079108e-01 -2.407338e-03 -4.568017e-02 10 : 1.079108e-01 2.407338e-03 -4.568017e-02 11 : 1.078342e-01 -2.438936e-03 4.566668e-02 12 : 1.078342e-01 2.438936e-03 4.566668e-02 13 : 1.079108e-01 -2.407338e-03 -4.568017e-02 14 : 1.079108e-01 2.407338e-03 -4.568017e-02 15 : 1.078342e-01 -2.438936e-03 4.566668e-02 16 : 2.162759e-01 1.169299e-03 5.734546e-02 17 : 2.158140e-01 -1.338262e-03 -5.725267e-02 18 : 2.158140e-01 1.338262e-03 -5.725267e-02 19 : 2.162759e-01 -1.169299e-03 5.734546e-02 20 : 3.384734e-01 8.680069e-04 6.102590e-02 21 : 3.412658e-01 1.652938e-05 -6.162524e-02 22 : 3.412658e-01 -1.652935e-05 -6.162524e-02 23 : 3.384734e-01 -8.680069e-04 6.102590e-02	0 : 4.992890e-10 8.266443e-10 4.000001e-09 1 : 5.007112e-10 -8.263335e-10 -4.000001e-09 2 : 5.007112e-10 8.263335e-10 -4.000001e-09 3 : 4.992890e-10 -8.266443e-10 4.000001e-09 4 : 3.110572e-02 4.895740e-03 2.590097e-02 5 : 3.109334e-02 -4.901381e-03 -2.589808e-02 6 : 3.109334e-02 4.901381e-03 -2.589808e-02 7 : 3.110572e-02 -4.895740e-03 2.590097e-02 8 : 1.078342e-01 2.438936e-03 4.566668e-02 9 : 1.079108e-01 -2.407338e-03 -4.568017e-02 10 : 1.079108e-01 2.407338e-03 -4.568017e-02 11 : 1.078342e-01 -2.438936e-03 4.566668e-02 12 : 2.162759e-01 1.169299e-03 5.734546e-02 13 : 2.158140e-01 -1.338262e-03 -5.725267e-02 14 : 2.158140e-01 1.338262e-03 -5.725267e-02 15 : 2.162759e-01 -1.169299e-03 5.734546e-02 16 : 3.384734e-01 8.680069e-04 6.102590e-02 17 : 3.412658e-01 1.652938e-05 -6.162524e-02 18 : 3.412658e-01 -1.652935e-05 -6.162524e-02 19 : 3.384734e-01 -8.680069e-04 6.102590e-02

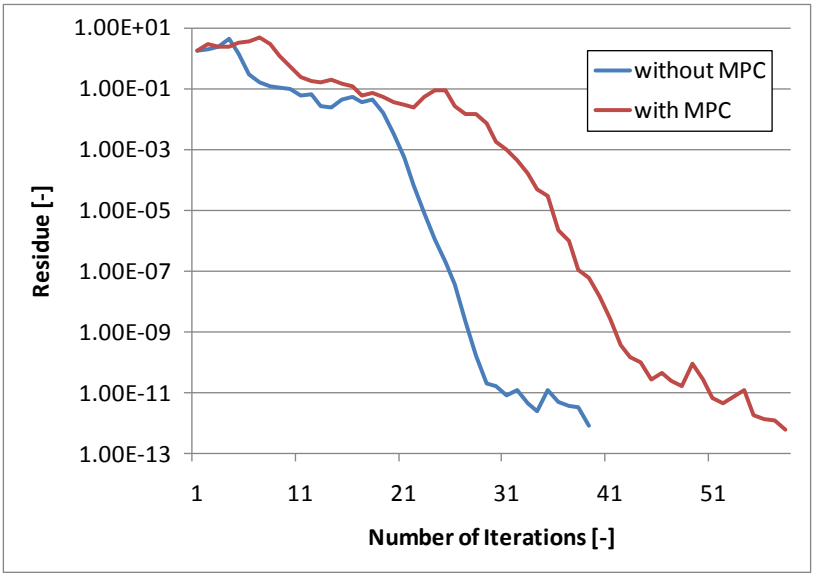


図 5.4.3　MPC の収束状況

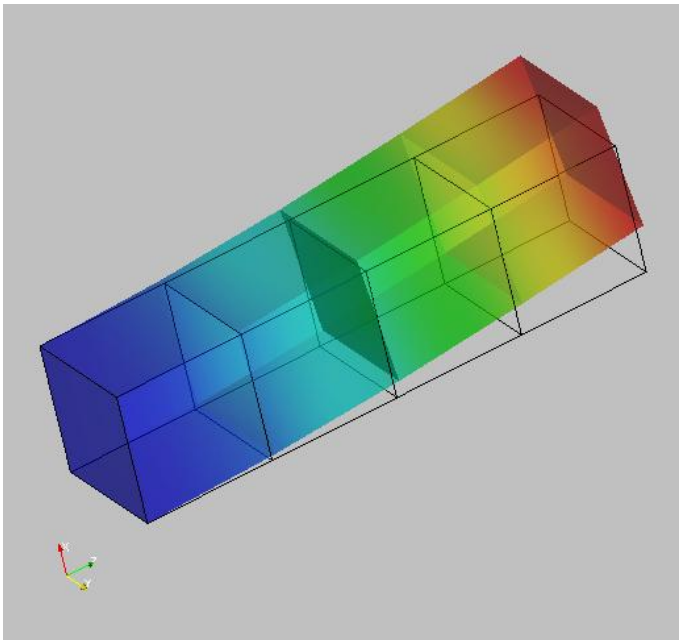


図 5.4.4　MPC の計算結果

5.5　使用例 4；MPC を利用した計算（不連続メッシュ）

5.5.1　計算条件

不連続メッシュで作成した片持ち梁の形状でテストを実施した。つぎにその不連続メッシュの形状を示す。

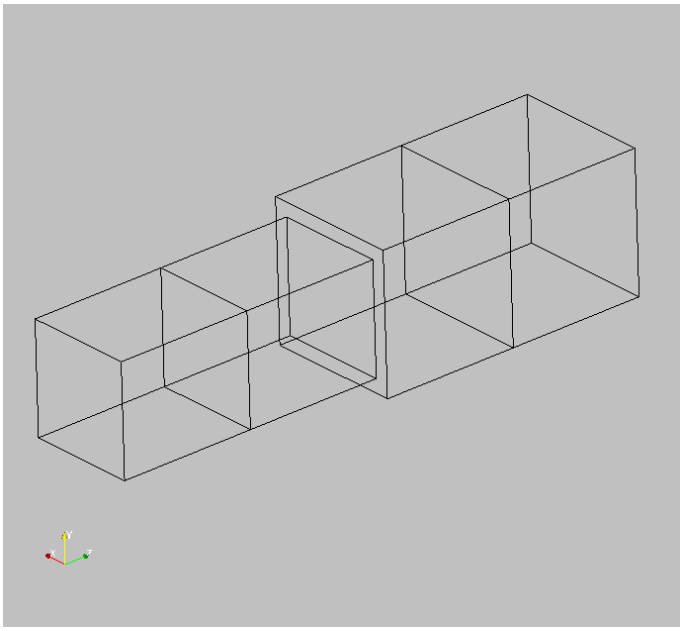


図 5.5.1　MPC を利用した不連続メッシュ形状

5.5.2　入力データ

```
Refine
   0
End
AssyModel
   2   1   0
   0
   1
End
Node
         12          0         11          0
V 0 3      0       0.100       0.100       0.000
V 0 3      1       0.900       0.100       0.000
V 0 3      2       0.900       0.900       0.000
V 0 3      3       0.100       0.900       0.000
V 0 3      4       0.100       0.100       1.000
V 0 3      5       0.900       0.100       1.000
V 0 3      6       0.900       0.900       1.000
V 0 3      7       0.100       0.900       1.000
V 0 3      8       0.100       0.100       2.000
V 0 3      9       0.900       0.100       2.000
V 0 3     10       0.900       0.900       2.000
V 0 3     11       0.100       0.900       2.000
End
Element
          2          0          1          0
Hexa       0       0       1       2       3       4       5       6       7
Hexa       1       4       5       6       7       8       9      10      11
End
Node
         12          1         11          0
V 0 3      0       0.000       0.000       2.000
V 0 3      1       1.000       0.000       2.000
V 0 3      2       1.000       1.000       2.000
V 0 3      3       0.000       1.000       2.000
V 0 3      4       0.000       0.000       3.000
V 0 3      5       1.000       0.000       3.000
V 0 3      6       1.000       1.000       3.000
V 0 3      7       0.000       1.000       3.000
V 0 3      8       0.000       0.000       4.000
V 0 3      9       1.000       0.000       4.000
V 0 3     10       1.000       1.000       4.000
V 0 3     11       0.000       1.000       4.000
End
Element
          2          1          1          0
Hexa       0       0       1       2       3       4       5       6       7
Hexa       1       4       5       6       7       8       9      10      11
End
ContactMesh
 1 0 0
 0 0 0
 8 7 0
```

```
  0     0.0     0.0     2.0     0   8   0 0 v   3   0
  1     1.0     0.0     2.0     0   9   0 0 v   3   0
  2     1.0     1.0     2.0     0 10   0 0 v   3   0
  3     0.0     1.0     2.0     0 11   0 0 v   3   0
  4     0.0     0.0     2.0     1   0   0 1 v   3   0
  5     1.0     0.0     2.0     1   1   0 1 v   3   0
  6     1.0     1.0     2.0     1   2   0 1 v   3   0
  7     0.0     1.0     2.0     1   3   0 1 v   3   0
   1 0 0
   0   0 1 1   Quad 0 1 2 3 0
   1 0 0
   0   1 0 0   Quad 4 5 6 7 0
End
```

図 5.5.2　MPC を利用した不連続メッシュ形状入力データ

5.5.3　実行結果

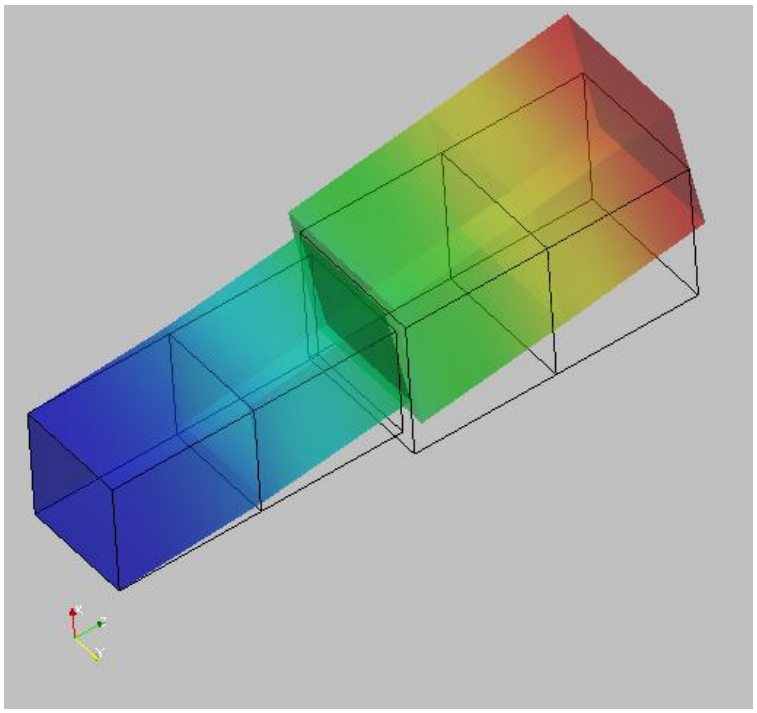


図 5.5.3　MPC を利用した不連続メッシュ形状計算結果（その 1）

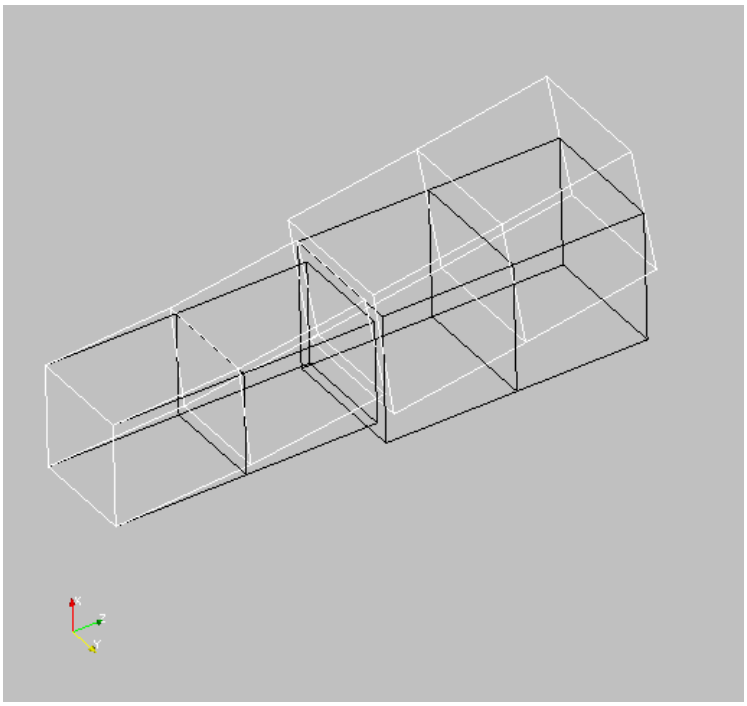


図 5.5.4　MPC を利用した不連続メッシュ形状計算結果（その 2）

5.6　使用例 5；MGCG を利用した計算

5.6.1　MGCG について

通常の反復法はメッシュサイズに相当する波長を持った誤差成分の減衰には適しているが、誤差の成分のうち、長い波長の成分は反復を繰り返してもなかなか収束しない。粗い格子を使用することで、長波長成分を効率的に減衰させることを目的とした手法がマルチグリッド法である。マルチグリッド法の各反復における手順の概要を以下に示す。

① 細メッシュで残差を計算
② 残差を粗メッシュに制限近似(restriction)
③ 粗メッシュで補正式の反復計算
④ 細メッシュへ補正量を延長補間(prolongation)
⑤ 細メッシュでの解を更新

MGCG 法は、CG 法に上記の MG 法を前処理に利用した方法である。本ソフトウェアでは、マルチグリッド法を CG 法などの反復解法の前処理として利用した手法を利用した。

5.6.2 MGCG のプログラム内部での処理について

MGCG 法の概要を下図に示す。図中で左枠内はマルチグリッド前処理を含む MGCG 法のアルゴリズムであり、右枠内はマルチグリッド前処理のアルゴリズムである。

$\mathbf{r}_0 = \mathbf{f} - \mathbf{K}\mathbf{u}_0$
for $k = 0..maxiter$
$\mathbf{z}^k = \mathrm{MG}(\mathbf{K}, \mathbf{r}^k, \mathrm{initial\_}\mathbf{z}^k, \gamma, \mu_1, \mu_2)$
$\rho_k = (\mathbf{r}_k, \mathbf{z}_k)$
if $k \equiv 0$
$\mathbf{p}_0 = \mathbf{r}_0$
else
$\beta_k = \dfrac{\rho_k}{\rho_{k-1}}$
$\mathbf{p}_k = \mathbf{z}_k + \beta_k \mathbf{p}_{k-1}$
endif
$\mathbf{q}_k = \mathbf{K}\mathbf{p}_k$
$\alpha_k = \dfrac{\rho_k}{(\mathbf{p}_k, \mathbf{q}_k)}$
$\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha_k \mathbf{p}_k$
$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{q}_k$
check convergence; continue if necessary
end
	$\mathrm{MG}(\mathbf{K}_{level}, \mathbf{f}, \mathbf{u}, \gamma, \mu_1, \mu_2)$
	{
	if $level \equiv$ coarsest_level
	solve $\mathbf{K}_{level}\mathbf{u} = \mathbf{f}$
	else
	$\mathbf{u} = \mathrm{pre\_smoothing}(\mathbf{K}_{level}, \mathbf{f}, \mathbf{u}, \mu_1)$
	$\mathbf{r} = \mathrm{restric}(\mathbf{f} - \mathbf{K}_{level}\mathbf{u})$
	$\Delta\mathbf{u}_C = \mathrm{initial\_}\Delta\mathbf{u}$
	for $n = 1..\gamma$
	$\Delta\mathbf{u}_C = \mathrm{MG}(\mathbf{K}_{level-1}, \mathbf{r}, \Delta\mathbf{u}_C, \gamma, \mu_1, \mu_2)$
	end
	$\mathbf{u} = \mathbf{u} + \mathrm{prolongate}(\Delta\mathbf{u}_C)$
	$\mathbf{u} = \mathrm{post\_smoothing}(\mathbf{K}_{level}, \mathbf{f}, \mathbf{u}, \mu_2)$
	endif
	return u
	}

MGCG 法	MG 前処理

図 5.6.1 MGCG 法の概要

プログラムの内部では、前処理のひとつの機能として呼ばれているため、ユーザはマルチグリッドであることを意識してプログラミングする必要はない。実際のプログラム内部の処理は、つぎの 3 つの処理をことを行っている。

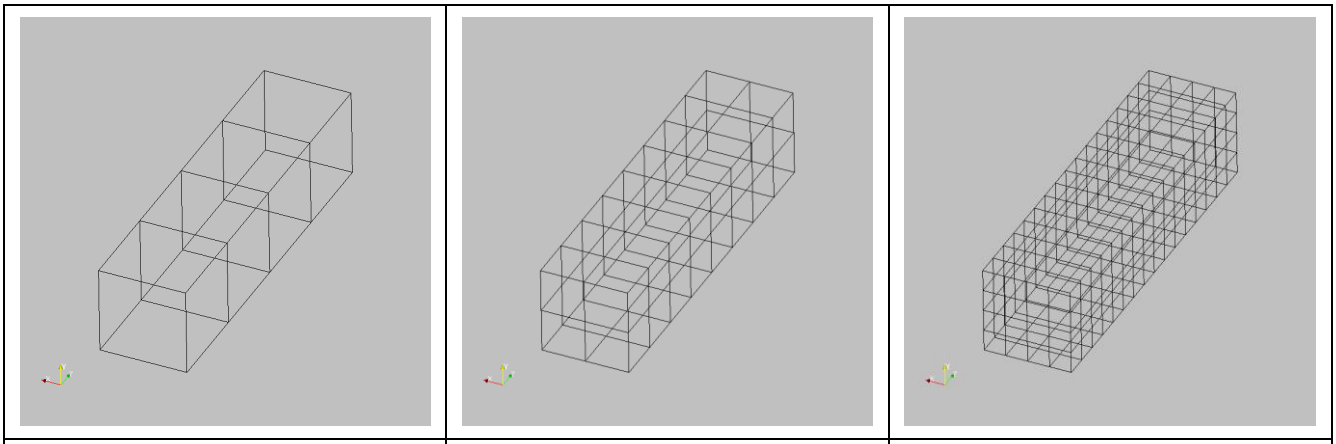
① 粗格子から密格子のすべてのレベルの格子情報の生成。レベル間のインデクスも作成する必要がある(restriction and prolongation)。

② 上記インデクスを利用した全レベル（粗格子から密格子までのレベル）の剛性行列の作成をユーザが行う(サンプルプログラムに示す)。

③ ソルバの前処理では、①の上下関係を表すインデクスを利用して、②で作成した剛性行列を呼び出しながら機能する。

5.6.3　計算条件

片持ち梁で、3 レベルでのマルチグリッドの基本的なテストを行った。各レベルのメッシュ形状と入力データを示す。



レベル 0	レベル 1	レベル 2

図 5.6.2　MGCG のテストで利用した各レベルの形状

5.6.4　入力データ

```
Refine
   2
End

AssyModel
   2   1   0
   0
   1
End

Node
         12          0         11          0
V 0 3       0       0.000       0.000       0.000
V 0 3       1       1.000       0.000       0.000
V 0 3       2       1.000       1.000       0.000
V 0 3       3       0.000       1.000       0.000
V 0 3       4       0.000       0.000       1.000
V 0 3       5       1.000       0.000       1.000
V 0 3       6       1.000       1.000       1.000
V 0 3       7       0.000       1.000       1.000
V 0 3       8       0.000       0.000       2.000
V 0 3       9       1.000       0.000       2.000
V 0 3      10       1.000       1.000       2.000
```

```
V 0 3     11        0.000        1.000        2.000
End

Element
          2          0          1          0
Hexa        0        0        1        2        3        4        5        6        7
Hexa        1        4        5        6        7        8        9       10       11
End

Node
         12          1         11          0
V 0 3       0        0.000        0.000        2.000
V 0 3       1        1.000        0.000        2.000
V 0 3       2        1.000        1.000        2.000
V 0 3       3        0.000        1.000        2.000
V 0 3       4        0.000        0.000        3.000
V 0 3       5        1.000        0.000        3.000
V 0 3       6        1.000        1.000        3.000
V 0 3       7        0.000        1.000        3.000
V 0 3       8        0.000        0.000        4.000
V 0 3       9        1.000        0.000        4.000
V 0 3      10        1.000        1.000        4.000
V 0 3      11        0.000        1.000        4.000
End

Element
          2          1          1          0
Hexa        0        0        1        2        3        4        5        6        7
Hexa        1        4        5        6        7        8        9       10       11
End

ContactMesh
 1 0 0
 0 0 0
 8 7 0
0     0.0     0.0     2.0     0   8    0 0 v    3    0
1     1.0     0.0     2.0     0   9    0 0 v    3    0
2     1.0     1.0     2.0     0 10    0 0 v    3    0
3     0.0     1.0     2.0     0 11    0 0 v    3    0
4     0.0     0.0     2.0     1    0    0 1 v    3    0
5     1.0     0.0     2.0     1    1    0 1 v    3    0
6     1.0     1.0     2.0     1    2    0 1 v    3    0
7     0.0     1.0     2.0     1    3    0 1 v    3    0
 1 0 0
 0    0 1 1    Quad 0 1 2 3 0
 1 0 0
 0    1 0 0    Quad 4 5 6 7 0
End
```

図 5.6.3　MGCG 入力データ

5.6.5 計算結果

マルチグリッドと従来のソルバーの場合の結果を比較し、完全に一致していることを確認した。

表 5.2 MGCG とブロックヤコビ CG 法

マルチグリッド前処理付き CG 法	ブロックヤコビ CG 法
0 5.141359e-05 6.772549e-05 1.880514e-04	0 5.141359e-05 6.772549e-05 1.880514e-04
1 5.141359e-05 -6.772549e-05 -1.880514e-04	1 5.141359e-05 -6.772549e-05 -1.880514e-04
2 5.141359e-05 6.772549e-05 -1.880514e-04	2 5.141359e-05 6.772549e-05 -1.880514e-04
3 5.141359e-05 -6.772549e-05 1.880514e-04	3 5.141359e-05 -6.772549e-05 1.880514e-04
4 1.146248e-01 1.328394e-02 9.850462e-02	4 1.146248e-01 1.328394e-02 9.850462e-02
5 1.146246e-01 -1.328407e-02 -9.850464e-02	5 1.146246e-01 -1.328407e-02 -9.850464e-02
6 1.146246e-01 1.328407e-02 -9.850464e-02	6 1.146246e-01 1.328407e-02 -9.850464e-02
7 1.146248e-01 -1.328394e-02 9.850462e-02	7 1.146248e-01 -1.328394e-02 9.850462e-02
8 4.023766e-01 8.837073e-03 1.709948e-01	8 4.023766e-01 8.837073e-03 1.709948e-01
9 4.023853e-01 -8.829493e-03 -1.709936e-01	9 4.023853e-01 -8.829493e-03 -1.709936e-01
10 4.023853e-01 8.829493e-03 -1.709936e-01	10 4.023853e-01 8.829493e-03 -1.709936e-01
11 4.023766e-01 -8.837073e-03 1.709948e-01	11 4.023766e-01 -8.837073e-03 1.709948e-01
12 8.064219e-01 4.281656e-03 2.145121e-01	12 8.064219e-01 4.281656e-03 2.145121e-01
13 8.060102e-01 -4.572574e-03 -2.145266e-01	13 8.060102e-01 -4.572574e-03 -2.145266e-01
14 8.060102e-01 4.572574e-03 -2.145266e-01	14 8.060102e-01 4.572574e-03 -2.145266e-01
15 8.064219e-01 -4.281656e-03 2.145121e-01	15 8.064219e-01 -4.281656e-03 2.145121e-01
16 1.263918e+00 4.899008e-04 2.268296e-01	16 1.263918e+00 4.899008e-04 2.268296e-01
17 1.292596e+00 6.738315e-03 -2.360223e-01	17 1.292596e+00 6.738315e-03 -2.360223e-01
18 1.292596e+00 -6.738315e-03 -2.360223e-01	18 1.292596e+00 -6.738315e-03 -2.360223e-01
19 1.263918e+00 -4.899008e-04 2.268296e-01	19 1.263918e+00 -4.899008e-04 2.268296e-01
410 4.947894e-01 -1.915704e-03 -9.151597e-02	410 4.947894e-01 -1.915704e-03 -9.151597e-02
411 6.976287e-01 1.352303e-03 1.023959e-01	411 6.976287e-01 1.352303e-03 1.023959e-01
412 6.975599e-01 -1.389865e-03 -1.024006e-01	412 6.975599e-01 -1.389865e-03 -1.024006e-01
413 4.947849e-01 -1.917822e-03 9.151473e-02	413 4.947849e-01 -1.917822e-03 9.151473e-02
414 4.947894e-01 1.915704e-03 -9.151597e-02	414 4.947894e-01 1.915704e-03 -9.151597e-02
415 6.976287e-01 -1.352303e-03 1.023959e-01	415 6.976287e-01 -1.352303e-03 1.023959e-01
416 6.975599e-01 1.389865e-03 -1.024006e-01	416 6.975599e-01 1.389865e-03 -1.024006e-01
417 9.187899e-01 7.567198e-04 1.096917e-01	417 9.187899e-01 7.567198e-04 1.096917e-01
418 9.182937e-01 -8.980550e-04 -1.096176e-01	418 9.182937e-01 -8.980550e-04 -1.096176e-01
419 1.149676e+00 4.236754e-04 1.131685e-01	419 1.149676e+00 4.236754e-04 1.131685e-01
420 1.151399e+00 1.931069e-04 -1.123071e-01	420 1.151399e+00 1.931069e-04 -1.123071e-01
421 9.187899e-01 -7.567198e-04 1.096917e-01	421 9.187899e-01 -7.567198e-04 1.096917e-01
422 9.182937e-01 8.980551e-04 -1.096176e-01	422 9.182937e-01 8.980551e-04 -1.096176e-01
423 1.149676e+00 -4.236754e-04 1.131685e-01	423 1.149676e+00 -4.236754e-04 1.131685e-01
424 1.151399e+00 -1.931069e-04 -1.123071e-01	424 1.151399e+00 -1.931069e-04 -1.123071e-01

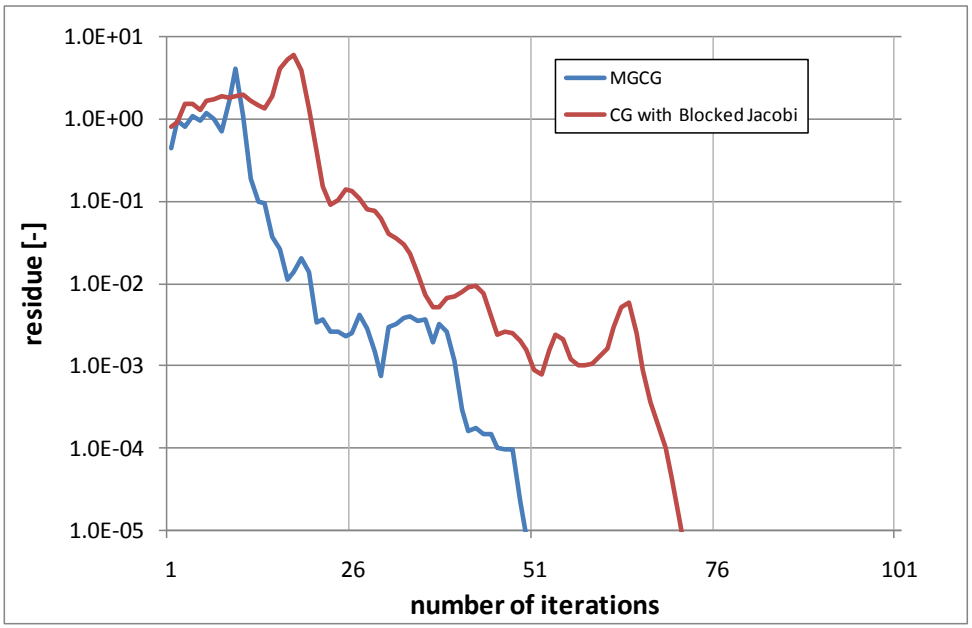


図 5.6.4　MGCG の収束状況

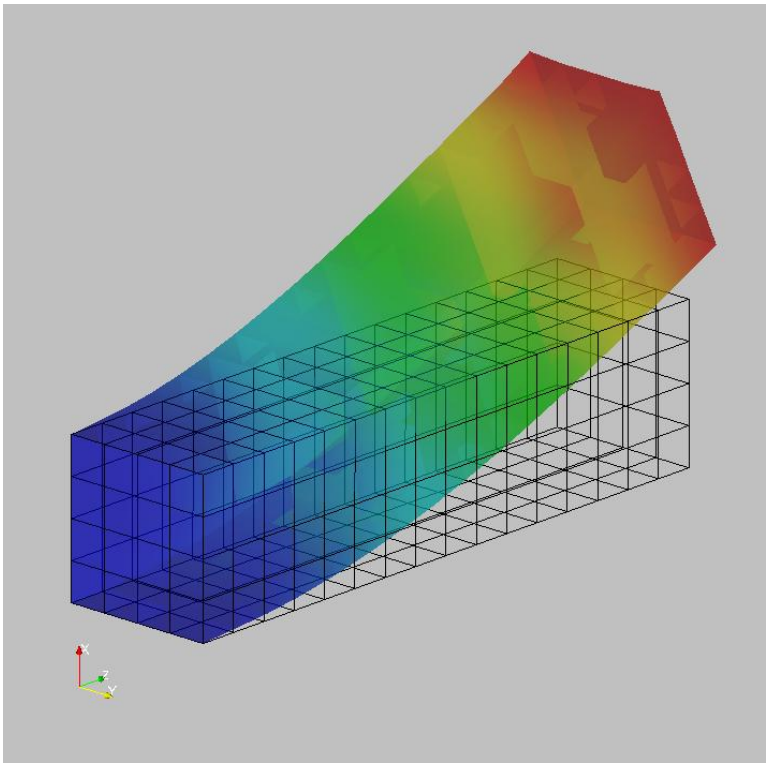


図 5.6.5　MGCG の計算結果

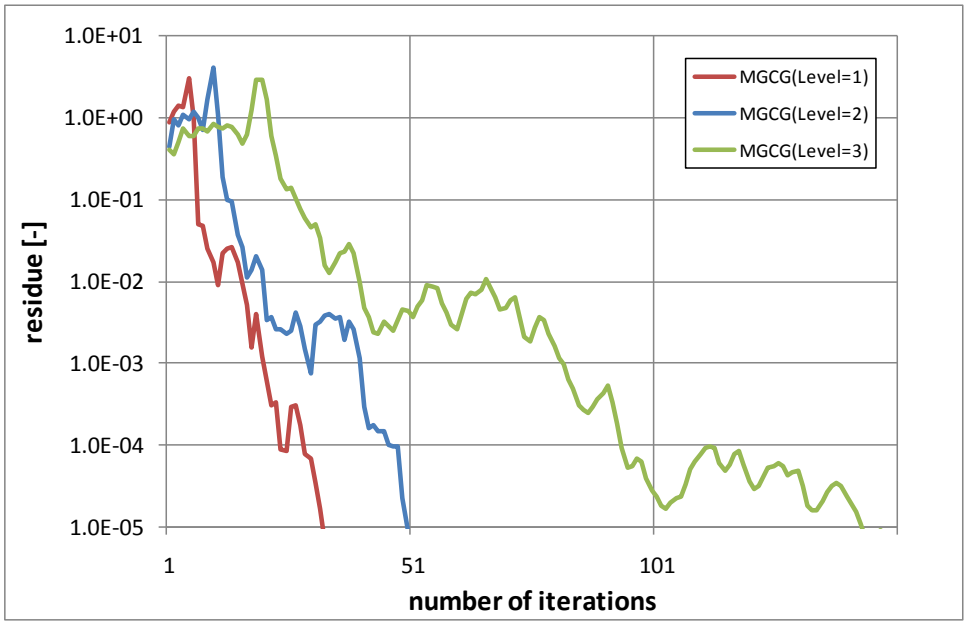


図 5.6.6　MGCG 収束状況

5.7　使用例 6；並列化を利用した例題

5.7.1　計算条件

片持ち梁で、並列化の基本的なテストを行った。メッシュ形状と入力データを示す。このデータでは、refine 時に利用される face の情報は含めていない（face 情報は refine する場合にのみ利用される）。次の節で face のデータも含めたテストケースを示す。

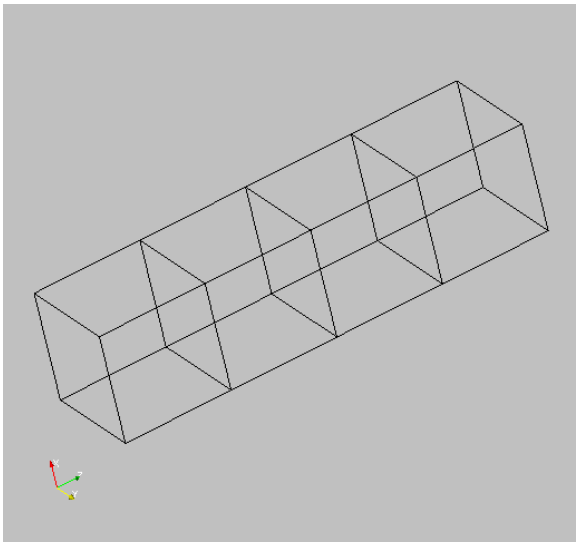


図 5.7.1　並列化テスト形状

5.7.2　入力データ

```
Refine
   0
End

AssyModel
   1   0   0
   0
End

Node
         12          0         11          0
V 0 3      0       0.000       0.000       0.000
V 0 3      1       1.000       0.000       0.000
V 0 3      2       1.000       1.000       0.000
V 0 3      3       0.000       1.000       0.000
V 0 3      4       0.000       0.000       1.000
V 0 3      5       1.000       0.000       1.000
V 0 3      6       1.000       1.000       1.000
V 0 3      7       0.000       1.000       1.000
V 0 3      8       0.000       0.000       2.000
V 0 3      9       1.000       0.000       2.000
V 0 3     10       1.000       1.000       2.000
V 0 3     11       0.000       1.000       2.000
End

Element
          2          0          1          0
Hexa       0       0       1       2       3       4       5       6       7
Hexa       1       4       5       6       7       8       9      10      11
End

CommMesh2
 0 1
 0 1 4 0 1
End

CommNodeCM2
 0   0 4
 0   8   0.000       0.000       2.000
 1   9   1.000       0.000       2.000
 2 10   1.000       1.000       2.000
 3 11   0.000       1.000       2.000
End
```

図 5.7.2　並列化テスト形状入力データ（ランク 0）

```
Refine
   0
End
```

```
AssyModel
   1   0   0
   0
End

Node
         12          0         11          0
V 0 3      0       0.000       0.000       2.000
V 0 3      1       1.000       0.000       2.000
V 0 3      2       1.000       1.000       2.000
V 0 3      3       0.000       1.000       2.000
V 0 3      4       0.000       0.000       3.000
V 0 3      5       1.000       0.000       3.000
V 0 3      6       1.000       1.000       3.000
V 0 3      7       0.000       1.000       3.000
V 0 3      8       0.000       0.000       4.000
V 0 3      9       1.000       0.000       4.000
V 0 3     10       1.000       1.000       4.000
V 0 3     11       0.000       1.000       4.000
End

Element
          2          0          1          0
Hexa       0       0       1       2       3       4       5       6       7
Hexa       1       4       5       6       7       8       9      10      11
End

CommMesh2
 0 1
 0 1 4 1 0
End

CommNodeCM2
 0   0 4
 0   0   0.000       0.000       2.000
 1   1   1.000       0.000       2.000
 2   2   1.000       1.000       2.000
 3   3   0.000       1.000       2.000
End
```

図 5.7.3　並列化テスト形状入力データ（ランク 1）

5.7.3　計算結果

並列化した場合とシングルプロセッサの場合の結果を比較し、完全に一致していることを確認した。

表 5.3　2 プロセッサとシングルプロセッサでの計算結果の比較

並列化	シングルプロセッサ
ランク 0	

0 : 4.992890e-10 8.266443e-10 4.000001e-09 1 : 5.007112e-10 -8.263335e-10 -4.000001e-09 2 : 5.007112e-10 8.263335e-10 -4.000001e-09 3 : 4.992890e-10 -8.266443e-10 4.000001e-09 4 : 3.110572e-02 4.895740e-03 2.590097e-02 5 : 3.109334e-02 -4.901381e-03 -2.589808e-02 6 : 3.109334e-02 4.901381e-03 -2.589808e-02 7 : 3.110572e-02 -4.895740e-03 2.590097e-02 8 : 1.078342e-01 2.438936e-03 4.566668e-02 9 : 1.079108e-01 -2.407338e-03 -4.568017e-02 10 : 1.079108e-01 2.407338e-03 -4.568017e-02 11 : 1.078342e-01 -2.438936e-03 4.566668e-02	0 : 4.992890e-10 8.266443e-10 4.000001e-09 1 : 5.007112e-10 -8.263335e-10 -4.000001e-09 2 : 5.007112e-10 8.263335e-10 -4.000001e-09 3 : 4.992890e-10 -8.266443e-10 4.000001e-09 4 : 3.110572e-02 4.895740e-03 2.590097e-02 5 : 3.109334e-02 -4.901381e-03 -2.589808e-02 6 : 3.109334e-02 4.901381e-03 -2.589808e-02 7 : 3.110572e-02 -4.895740e-03 2.590097e-02 8 : 1.078342e-01 2.438936e-03 4.566668e-02 9 : 1.079108e-01 -2.407338e-03 -4.568017e-02 10 : 1.079108e-01 2.407338e-03 -4.568017e-02 11 : 1.078342e-01 -2.438936e-03 4.566668e-02 12 : 2.162759e-01 1.169299e-03 5.734546e-02 13 : 2.158140e-01 -1.338262e-03 -5.725267e-02 14 : 2.158140e-01 1.338262e-03 -5.725267e-02 15 : 2.162759e-01 -1.169299e-03 5.734546e-02 16 : 3.384734e-01 8.680069e-04 6.102590e-02 17 : 3.412658e-01 1.652938e-05 -6.162524e-02 18 : 3.412658e-01 -1.652935e-05 -6.162524e-02 19 : 3.384734e-01 -8.680069e-04 6.102590e-02
ランク 1	
0 : 1.078342e-01 2.438936e-03 4.566668e-02 1 : 1.079108e-01 -2.407338e-03 -4.568017e-02 2 : 1.079108e-01 2.407338e-03 -4.568017e-02 3 : 1.078342e-01 -2.438936e-03 4.566668e-02 4 : 2.162759e-01 1.169299e-03 5.734546e-02 5 : 2.158140e-01 -1.338262e-03 -5.725267e-02 6 : 2.158140e-01 1.338262e-03 -5.725267e-02 7 : 2.162759e-01 -1.169299e-03 5.734546e-02 8 : 3.384734e-01 8.680069e-04 6.102590e-02 9 : 3.412658e-01 1.652935e-05 -6.162524e-02 10 : 3.412658e-01 -1.652938e-05 -6.162524e-02 11 : 3.384734e-01 -8.680069e-04 6.102590e-02	

※青い字の部分が、空間内で節点が重なる部分である

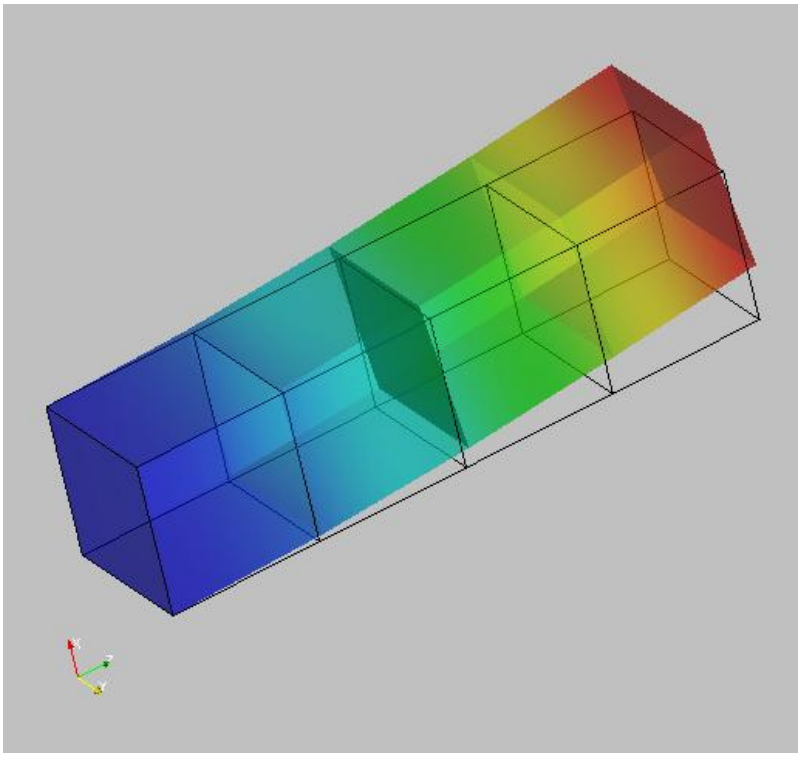


図 5.7.4　並列時の計算結果

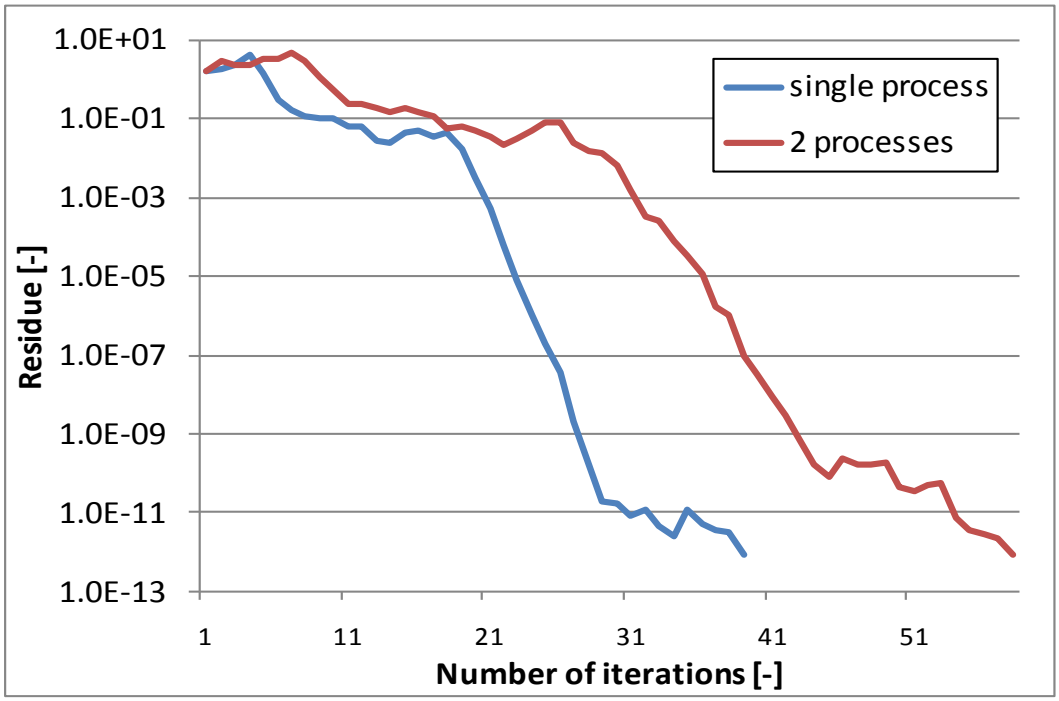


図 5.7.5　並列時の収束状況

5.7.4　Refine を含む並列例題

先に述べたケースにおいて、refine のレベルを 1,2 で正しく動作することを確認した。収束状況を示す。また、前節で示したデータにつぎの Face に関するデータを追加している。

```
CommFace
 0 0 1
 Quad 0   1 1      0 1 2 3
End
```

図 5.7.6　face に関する入力データ（ランク 0）

```
CommFace
 0 0 1
 Quad 0   0 0      0 1 2 3
End
```

図 5.7.7　face に関する入力データ（ランク 1）

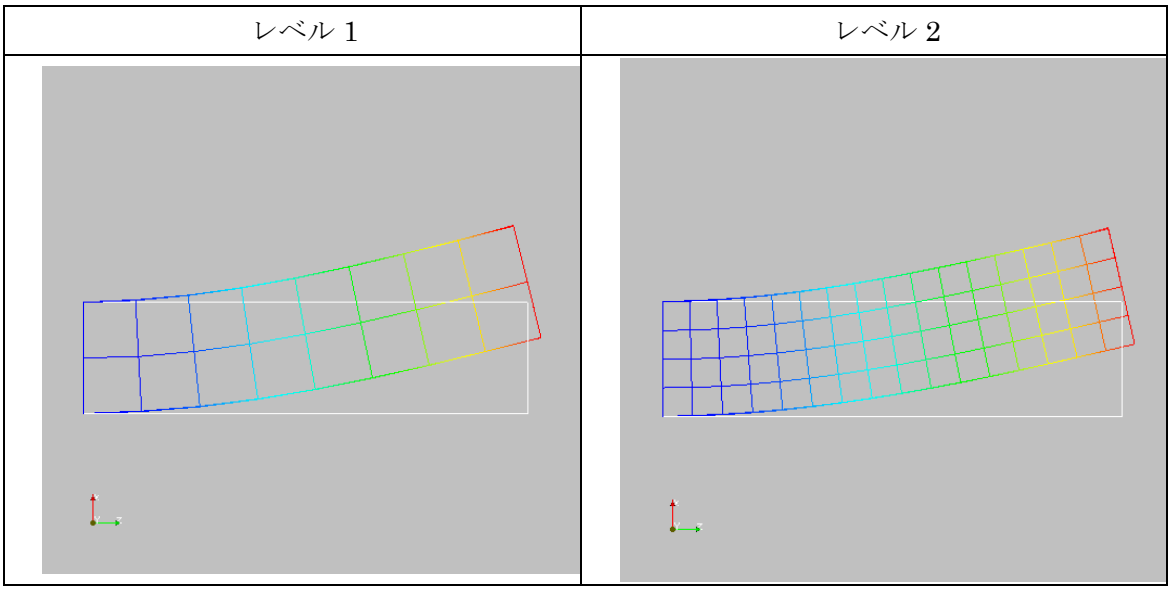


図 5.7.8　並列の refine 時の結果

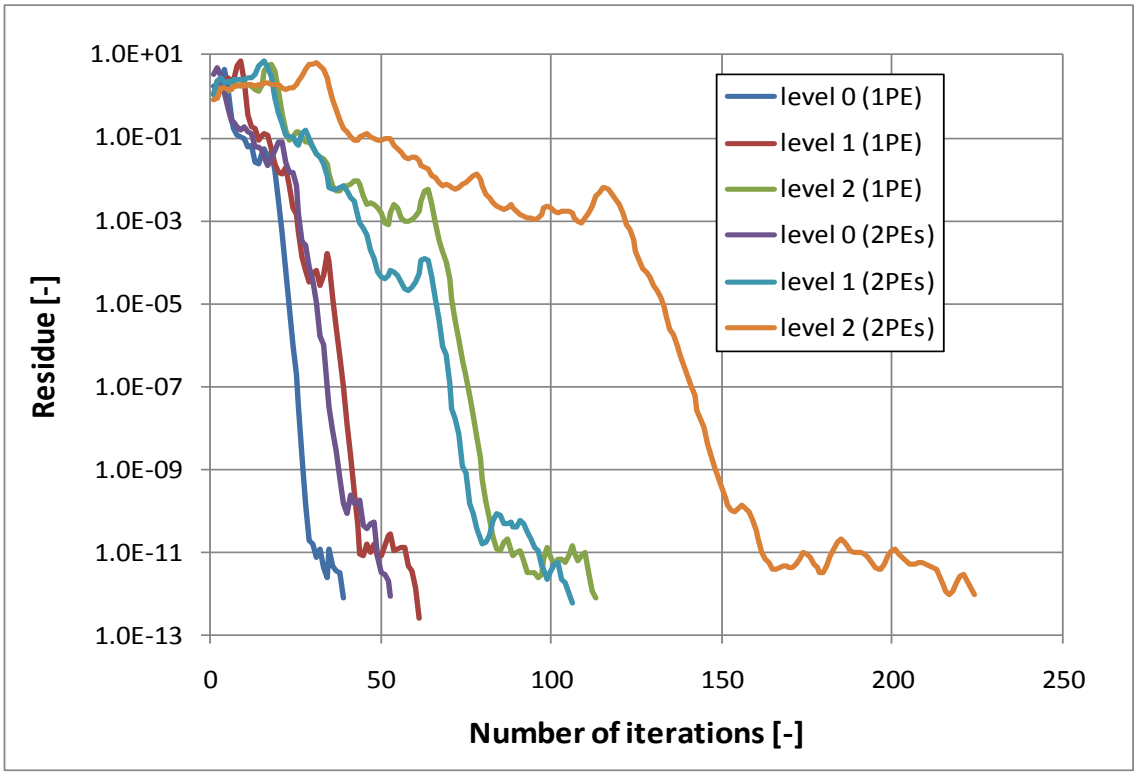


図 5.7.9　並列時の収束状況

6. MW3API

6.1 初期化・終了処理

```
Initialize(int argc, char** argv, const char* path);
Finalize();
```

MW3 ライブラリーの初期化宣言である。Initialize は、mw3.cnt ファイルを読み込み、領域分割されたメッシュファイルのファイル名を取得し、MPI の Init を呼び出し、ロガーの初期化を行う。
Finalize は、MPI の Finalize を呼び出し終了する。

引数 path は、MW3.cnt ファイルへのパスである。
MW3.cnt は、MW3 中間ファイル*.msh のファイルベース名を記述したファイルである。

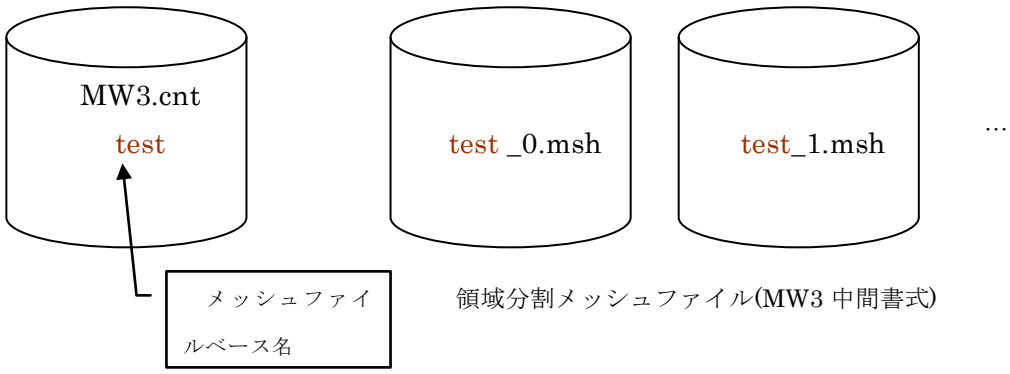


図 6.1.1 領域分割メッシュファイル(MW3 中間書式)

MW3 が読みこむメッシュデータファイルは、領域分割されたファイルであり、ファイル名から拡張子を除いたベースファイル名の後半部分に領域番号が"_"とともに付加されている。このベースファイル名を記述するための固定ファイルが"mw3.cnt"ファイルである。

6.2 File 関連

```
FileRead();
FileWrite(); ※デバッグ出力
```

FileRead は、MW3 中間ファイルを入力する。

FileWrite は、入力データのデバッグ用ファイルを出力する。

6.3 線形代数ソルバー

```
Initialize_Matrix();
```

線形代数ソルバーで利用する行列の初期化宣言である。

```
Initialize_Vector();
```

線形代数ソルバーで利用するベクトルの初期化宣言である。

```
Matrix_Add_Elem(int iAssy, int iMatrix, int iElem, double ElemMatrix[24][24]);
```

引数で指定した要素剛性行列(ElemMatrix)を全体剛性行列へ BCRS フォーマットで組み立てる関数である。

```
Set_BC(int matrix_id, int vector_id, int iNode, int iDof, double value1, double value2); ①
Set_BC(int vector_id, int iNode, int iDof, double value);②
```

境界条件を全体剛性行列と右辺ベクトルに設定する関数であり、
①全体剛性行列の対角に value1 を設定し、右辺の荷重項に value2 を設定する関数である。
②全体剛性行列は操作せず、右辺の荷重項に value0 を設定する関数である。

```
Solve(int iter_max, double tolerance, bool flag_iter, bool flag_time_log);
```

線形ソルバーの実行関数である。

6.4 階層メッシュ構造の構築

階層構造を含めた、メッシュデータ内部構造を構築する関数である。

```
Refine()
```

MW3 中間ファイルの"Refine～End"ブロック内に記述された階層数のマルチグリッドモデルを構築する関数である。

階層数がゼロであっても、メッシュ内部関係を構築するために呼び出す必要がある。
※必須関数

6.5　メッシュデータ・アクセス

MW3 のメッシュデータにアクセスするための関数について説明する。

メッシュデータ API を利用して MW3 内部のメッシュデータにアクセスする手順を大まかに記すと、選択順序は；

1. Assemble Model(階層構造から必要な階層(Level)のアセンブル・モデルを取得)
2. Mesh (Assemble Model から、必要な Mesh パーツを取得)
3. Element (Mesh から、必要な要素を取得)
4. Node (要素の構成 NodeID から Node を取得)

のように呼び出すことで、メッシュのデータにアクセスすることができる。

以下に個別の関数について記述する。

```
GetNumOfAssembleModel(uint& numOfAssembleModel);  ①
SelectAssembleModel(const uint& mgLevel);②
```

①Assemble Model の個数つまり、階層数(mMGLevel+1)の取得する関数である。

②Assemble Model の選択をする関数である。引数は階層レベル番号である。

```
GetNumOfMeshPart(uint& numOfMeshPart);①
SelectMeshPart_ID(const uint& mesh_id);②
SelectMeshPart_IX(const uint& index);③
```

①選択されている Assemble Model の Mesh パーツの個数を取得する関数である。

②選択されている Assemble Model の Mesh パーツの選択をする関数である。
　引数はメッシュパーツ ID 番号である。

③選択されている Assemble Model の Mesh パーツの選択をする関数である。
　引数は Assemble Mode に存在する Mesh パーツの配列インデックス番号である。

※for 文のようなループ処理内で Mesh パーツを選択する場合には③を用いる。

```
getElementSize();①
getNodeSize();②
```

①選択されている Mesh パーツの要素数を取得する関数である。

②選択されている Mesh パーツの節点数を取得する関数である。

```
SelectElement_ID(const uint& elem_id);①
SelectElement_IX(const uint& index);②
GetElementType(uint& elemType);③
```

①選択されている Mesh パーツの要素の選択をする関数である。引数は要素 ID 番号である。
②選択されている Mesh パーツの要素の選択をする関数である。
　引数は Mesh パーツに存在する要素の配列インデックス番号である。
※for 文のようなループ処理内で要素を選択する場合には②を用いる。
③要素タイプの取得をする関数である。

```
GetNumOfElementVert(uint& numOfVert);①
GetElementVertNodeID(vuint& vNodeID);②
GetNumOfElementEdge(uint& numOfEdge);③
GetElementEdgeNodeID(vuint& vNodeID);④
```

要素の構成節点情報の取得する関数群である。
①選択されている要素の頂点数を取得する関数である。引数 numOfVert に代入される。
②選択されている要素の頂点のノード ID を取得する関数である。引数 vNodeID に代入される。
③選択されている要素の辺数を取得する関数である。引数 numOfEdge に代入される。
④選択されている要素の辺のノード ID を取得する関数である。引数 vNodeID に代入される。

```
GetNodeCoord(const uint& node_id, double& x, double& y, double& z);
```

選択された Mesh 内の節点座標を取得する関数である。引数 x,　y,　z に代入される。

6.6　形状関数 API

MW3 が所有する形状関数ライブラリーについて説明する。

```
NumOfIntegPoint(const uint& shapeType, uint& numOfInteg);
```

形状関数種類別の積分点数の取得関数である。
ShapeType を指定することで積分点数が、引数 numOfInteg に代入される。

ShapeType の種類は、下記になる。
Hexa 要素　：Hexa81, Hexa82, Hexa201, Hexa202, Hexa203,
Tetra 要素：Tetra41, Tetra101, Tetra104, Tetra1015,
Prism 要素：Prism62, Prism156, Prism159, Prism1518,
Quad 要素　：Quad41, Quad84, Quad89,
Triangle 要素：Triangle31, Triangle63,
Line 要素：　Line21, Line32,

使用方法は；

例えば六面体 1 次関数-1 点積分であれば ShapeType::Hexa81 とすることで、形状関数の種類を表す整数を取得できる。

```
ShapeFunc_on_pt(const uint& shapeType, const uint& igauss, vdouble& N);①
ShapeFunc(const uint& shapeType, vvdouble& N);②
```

形状関数を取得する関数である。引数 N に代入される。

①積分点での形状関数を取得する。積分点における全ての形状関数が配列で返される。

②全積分点の形状関数を一括で取得する、[積分点]-[形状関数番号]の順序の 2 重配列である。

```
dNdr_on_pt(const uint& shapeType, const uint& igauss, vvdouble& dNdr);①
dNdr(const uint& shapeType, vvvdouble& dNdr);②
```

自然座標での導関数を取得する関数である。引数 dNdr に代入される。

①積分点ごと dN/dr を取得する。dNdr は[形状関数]-[座標方向]の 2 重配列である。

②dN/dr を一括で取得する。dNdr は[積分点]- [形状関数]-[座標方向]の 3 重配列である。

```
Calculate_dNdx(const uint& elemType, const uint& numOfInteg, const uint&
elem_index);①
dNdx_on_pt(const uint& igauss, vvdouble& dNdx);②
dNdx(const uint& elemType, const uint& numOfInteg, const uint& elem_index,
                                                              vvvdouble&
dNdx);③
```

空間座標での導関数を計算-取得する関数である。

①指定した要素 Index 番号の要素について空間座標での導関数を計算する。

※要素タイプと積分点数を指定することで形状関数を指定する。

②直前に計算された積分点の導関数を取得する。値は引数 dNdx に代入される。[形状関数]-[座標方向]の 2 重配列である。

※①と②はペアで使用する関数である。つまり dNdx_on_pt()は、Calculate_dNdx(...)を実行後に使用する必要がある。

③指定した要素 Index 番号の要素について空間座標での導関数を計算し、導関数を一括取得する。値は引数 dNdx に代入される。dNdx は[積分点]- [形状関数]-[座標方向]の 3 重配列である。

※要素タイプと積分点数を指定することで形状関数を指定する。

要素タイプ

Hexa, Hexa2,

Prism, Prism2,
Tetra, Tetra2,
Quad, Quad2,
Triangle, Triangle2,
Beam, Beam2,

要素タイプの使用方法は；

例えば Hexa2 次要素であれば ElementType::Hexa2 とすることで、要素の種類を表す整数を取得できる。

```
detJacobian(const uint& elemType, const uint& numOfInteg, const uint& igauss, double& detJ);
```

Jacobian 行列式を取得する関数である。引数 detJ に代入される。

この関数を呼び出す直前に使用した dNdx 計算の detJ の値が代入される。

※要素タイプと積分点数を指定することで形状関数を指定する。

```
Weight(const uint& elemType, const uint& numOfInteg, const uint& igauss, double& w);
```

Gauss 積分点の重みを取得する関数である。引数 w に代入される。

※要素タイプと積分点数を指定することで形状関数を指定する。

7. MW3 ファイル形式

パーティショナーから出力される MW3 中間ファイルの書式を説明する。

7.1 ファイル形式の定義

ブロックデータ構造をとり、各ブロックの内容は以下である。デリミタはスペースを用いる。
各ブロックデータは、"ブロック名～End"の中に記述する。

7.1.1 階層レベル

ブロック名：Refine～End

階層レベル数

※ここで指定した階層レベルに従ってマルチグリッドモデルが構築される。

7.1.2 アセンブルモデル

ブロック名：AssyModel～End

メッシュパーツ数　最大メッシュパーツ ID 番号　最小メッシュパーツ ID 番号 メッシュパーツ ID … ※メッシュパーツ ID はメッシュパーツ数ぶんを繰り返し記述

7.1.3 ノード

各メッシュパーツ内の節点情報を記述
ブロック名：Node～End

節点数 メッシュパーツ ID　最大節点 ID　最小節点 ID 節点タイプ スカラー変数の数　ベクトル変数の数 節点 ID　X　Y　Z … ※節点は、節点数ぶんを繰り返し記述

7.1.4 エレメント

各メッシュパーツ内の要素情報を記述
ブロック名：Element～End

要素数 メッシュパーツ ID 最大要素 ID 最小要素 ID 要素タイプ 要素 ID 要素を構成する節点 ID…要素を構成する節点 ID … ※要素数ぶんを繰り返し記述

7.1.5 通信 Mesh(節点分割型)

節点分割型の通信領域のメッシュデータを記述
ブロック名：CommMesh～End

メッシュパーツ数 最大メッシュパーツ ID 最小メッシュパーツ ID メッシュパーツ ID メッシュパーツ毎の通信メッシュ数 通信メッシュ ID 自身のランク番号 通信相手のランク番号 … ※通信メッシュ ID は、通信メッシュ数ぶんを繰り返し記述

7.1.6 通信ノード

節点分割型の通信領域の節点データを記述
ブロック名：CommNode～End

通信節点数 メッシュ ID 通信メッシュ ID 最大通信節点 ID 最小通信節点 ID 通信節点 ID 節点 ID 所属ランク …

7.1.7 通信要素

節点分割型の通信領域の要素データを記述
ブロック名：CommElement～End

通信要素数 メッシュパーツ ID 通信メッシュ ID 最大通信要素 ID 最小通信要素 ID

要素タイプ　通信要素 ID　要素を構成する通信節点 ID…
…

7.1.8　通信 Mesh(要素分割型)

要素分割型の通信領域のメッシュデータを記述

ブロック名：CommMesh2～End

メッシュパーツ ID　メッシュパーツ毎の通信メッシュ数
通信メッシュ ID　通信面の数 通信節点の数　自身のランク番号　通信先ランク番号
…

7.1.9　通信ノード(CommNodeCM2)

要素分割型の通信領域の節点データを記述

ブロック名：CommNodeCM2～End

要素分割型通信メッシュ ID メッシュパーツ ID 通信節点数
通信節点 ID　節点 ID　X　Y　Z
…

7.1.10 通信面(CommFace)

要素分割型の通信領域の界面データを記述

ブロック名：CommFace～End

要素分割型通信メッシュ ID　メッシュパーツ ID　通信界面数
面タイプ　通信面 ID　エンティティ番号　構成通信節点 ID... 構成通信節点 ID
…

面タイプ:"Quad", "Triangle","Beam"

要素エンティティ番号:ソリッド要素=面番号、シェル要素=辺番号、ビーム要素=頂点番号

7.1.11 接合メッシュ(ContactMesh)

メッシュパーツを接合する面を記述

ブロック名：ContactMesh～End

接合メッシュ数 最大接合メッシュ ID　最小接合メッシュ ID 接合メッシュ ID　自身のランク番号　通信先ランク番号 … 接合節点数　最大接合節点 ID　最小接合節点 ID 接合節点 ID　X　Y Z　メッシュパーツ ID　節点 ID 所属ランク番号 maslave 番号　接合節点のタイプ 自由度数　ダミー … … マスター接合面数 最大接合面 ID　最小接合面 ID マスター接合面 ID　メッシュパーツ ID　要素 ID　要素面番号　面形状タイプ　面を構成する接合節点 ID …面を構成する接合節点 ID　所属ランク … … スレーブ接合面数 最大接合面 ID　最小接合面 ID スレーブ接合面 ID　メッシュパーツ ID　要素 ID　要素面番号　面形状タイプ　面を構成する接合節点 ID …面を構成する接合節点 ID　所属ランク … …

maslave 番号 : 0:マスター、 1:スレーブ
面形状の値は、文字列: “Quad” もしくは、”Triangle”
＊対応するデータが存在しない場合は“-”を使用.
接合節点のタイプ: “s”,”S”,”v”,”V”,”sv”,”SV”

7.2　要素エンティティ番号

7.2.1　Hexa 要素 エンティティ番号

(1) 形状定義

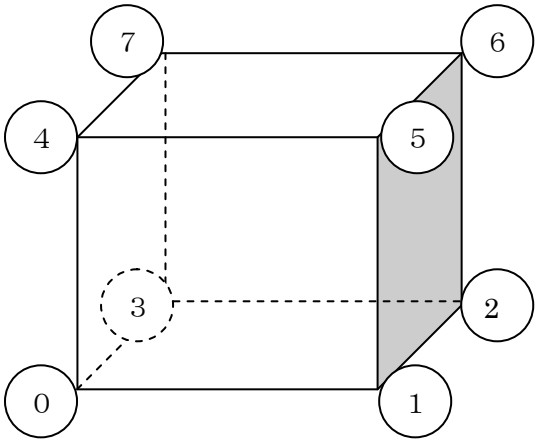


図 7.2.1　Hexa 要素

(2) Hexa 面番号

面 0：　0－1－2－3(図の底面)
面 1：　4－5－6－7(図の上面)
面 2：　1－5－6－2(図の右面)
面 3：　0－3－7－4(図の左面)
面 4：　0－4－5－1(図の手前)
面 5：　2－6－7－3(図の奥)

(3) Hexa 辺番号

辺０：０－１	辺４：４－５	辺８：０－４
辺１：１－２	辺５：５－６	辺９；１－５
辺２：２－３	辺６：６－７	辺１０：２－６
辺３：３－０	辺７：７－４	辺１１：３－７

7.2.2　Tetra 要素 エンティティ番号

(1) 形状定義

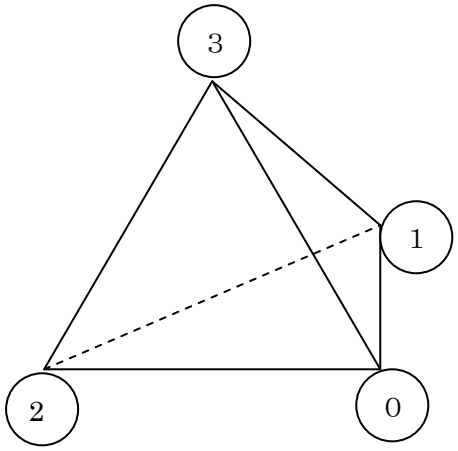


図 7.2.2　Tetra 要素

(2) Tetra 面番号

面 0：　0－1－2(図の底面)
面 1：　0－3－1(図の右面)
面 2：　1－3－2(図の奥)
面 3：　0－2－3(図の手前)

(3) Tetra 辺番号

辺０：０－１	辺３：０－３
辺１：１－２	辺４：１－３
辺２：２－０	辺５：２－３

7.2.3　Prism 要素 エンティティ番号

(1) 形状定義

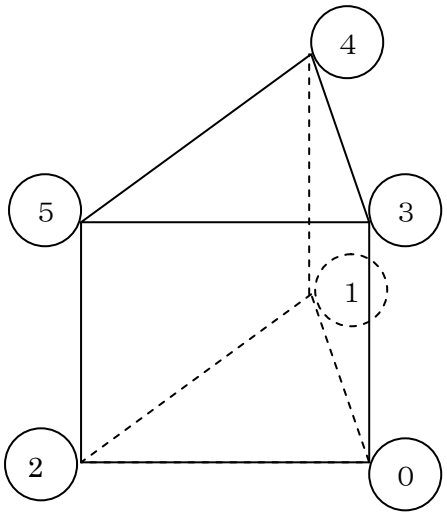


図 7.2.3　Prism 要素

(2) Prism 面番号

面 0：　0－1－2(図の底面)
面 1：　3－4－5(図の上面)
面 2：　0－3－4－1(図の奥右)
面 3：　1－4－5－2(図の奥左)
面 4：　0－2－5－3(図の手前)

(3) Prism 辺番号

辺0：0－1	辺3：0－3	辺6：3－4
辺1：2－0	辺4：1－4	辺7：4－5
辺2：1－2	辺5：2－5	辺8：5－3

7.2.4　Quad 要素 エンティティ番号

(1) 形状定義

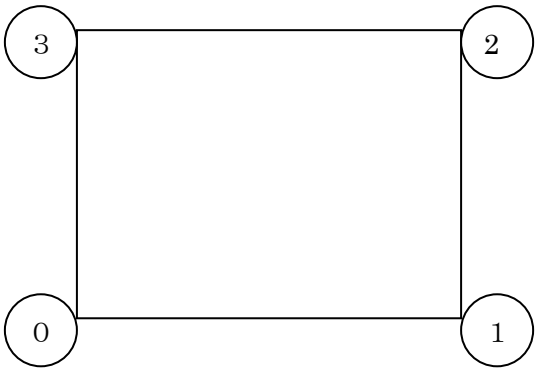


図 7.2.4　Quad 要素

(2) Quad 面番号

面 0：　0－1－2－3

(3) Quad 辺番号

辺０：０－１
辺１：１－２
辺２：２－３
辺３：３－０

7.2.5　Triangle 要素エンティティ番号

(1) 形状定義

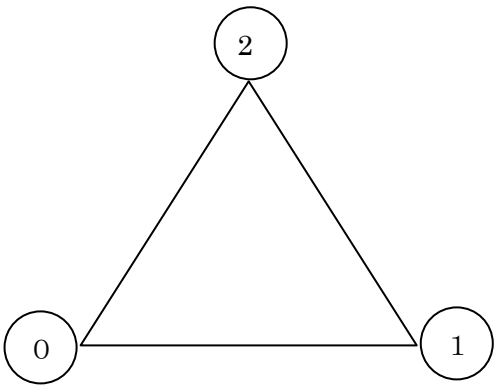


図 7.2.5　Triangle 要素

(2) Triangle 面番号

面０： ０－１－２

(3) Triangle 辺番号

辺０：０－１
辺１：１－２
辺２：２－０

7.2.6 Beam 要素エンティティ番号

7.2.6.1. Beam 面番号

(1) 形状定義

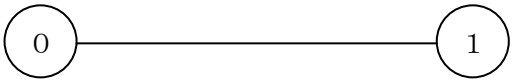


図 7.2.6 Beam 要素

(2) Beam 面番号

なし

(3) Beam 辺番号

辺０：０－１

7.3 ユーティリティー

7.3.1 ロガー機能

プログラム開発をする上で必要なログ機能を提供する。
ロガーは、プログラム中にロガー呼び出しルーチンを記述することで使用できる。
ロガーは、4 つのステート(Debug,Error,Warn,Info)により動作が異なり、FEM 解析コードの開発者が、これを指定する。ステートによる動作の差異は、下表に示す。
ロガーは、出力先をコンソール端末とファイルの 2 種類を選択することがでる。
ロガーのモードによる動作を次の表にまとめておく。

表 7.1　ロガーのモードによる動作

Logger_Mode で指定する State	Logger_Monitor が動作する State	備考
Debug	Debug, Error, Warn, Info	HEC_MW 3 を用いたプログラム開発に必要な情報は Debug 指定しておく。リリース時には Mode を Error 以下に変更すること。
Error	Error, Warn, Info	論理的なエラーなどに利用。
Warn	Warn, Info	ユーザへの警告。
Info	Info	著作権等の情報表示に利用。

8. 制限事項

・ 本プログラムは、つぎのような位置付けで公開しております。
  - プログラム開発者にライブラリーを提供します。
  - そのライブラリーを利用したサンプルプログラムも合わせて提供しています。
・ インストール方法の章に記述した計算機環境で動作確認を行っています。