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

マルチ力学シミュレータ REVOCAP

モデル細分化ツール

REVOCAP_PrePost Ver. 1.0

ユーザマニュアル

本ソフトウェアは文部科学省次世代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」プロジェクトによる成果物です。本ソフトウェアを無償でご使用になる場合「CISS フリーソフトウェア使用許諾条件」をご了承頂くことが前提となります。営利目的の場合には別途契約の締結が必要です。これらの契約で明示されていない事項に関して、或いは、これらの契約が存在しない状況においては、本ソフトウェアは著作権法など、関係法令により、保護されています。

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
E-mail：software@ciss.iis.u-tokyo.ac.jp

目次

1 概要

REVOCAP_Refiner は、マルチ力学シミュレーションソフトウェアの一部として、マルチ力学シミュレーションソフトウェアで開発されるソルバ、カップラに組み込まれて使われるオンメモリ上で有限要素法、有限体積法の非構造格子の要素を細分するライブラリである。

メッシュ生成、境界条件設定、領域分割等の通常のプレ処理が行われた解析モデルについて、ソルバまたはカップラはオンメモリで解析モデルを細分することができる。

単純に細分するだけでは、曲面形状の幾何的な解像度は細分前のメッシュの解像度を越えることができない。REVOCAP_Refiner では、この点にも留意し要素細分のときに、メッシュ生成に用いた CAD モデルを再度用いて、要素細分と同時に形状の補正を行い、細分された解析モデルの CAD モデルに対する形状の解像度を改善する。

2 動作環境

REVOCAP_Refiner の動作環境は、組み込まれて利用されるソルバおよびカップラの動作環境に準ずる。超並列計算機および次世代スパコンの上での動作を想定している。各種スパコン、PC クラスタ、次世代スパコン上 Unix 互換の OS のもとで動作する。動作が確認されている環境は以下の通り。

- SGI Altix (Linux)
- FUJITSU PRIMERGY (Linux)
- HA8000（Linux）
- PC-Cluster (Linux CentOS)

3 開発言語、コンパイラに関する情報

REVOCAP_Refiner は REVOCAP_PrePost とメッシュ処理部を共通化している。その部分の開発は C++ 言語でなされている。ソルバとのインターフェイスを取る部分は C 言語で開発されている。言語標準の以外の外部ライブラリについては特に利用していない。
動作が確認されているコンパイラは以下の通り。

- g++
- Intel C
- PGI C

また、REVOCAP_Refiner を呼び出すソルバ側の Fortran 言語について、動作が確認されているコンパイラは以下の通り。

- gfortran
- Intel Fortan
- PGI Fortran

日立コンパイラは namespace の対応でエラーが発生していることを確認している。

4 インストール方法

ここでは REVOCAP_Refiner のインストール方法、すなわちソルバに組み込むためのライブラリの作成手順を説明する。ソルバへ組み込むための API については、5 ライブラリ API 規約を参照のこと。ソルバとのリンクの方法は、それぞれのソルバのマニュアルの参照のこと。

4.1 ソースの展開

提供されている圧縮されたソースコード REVOCAP_Refiner-X.X.X.tgz を展開する。X.X.X はバージョン番号である。GNU 系の tar コマンドで展開する場合は以下のようになる。

```
tar xvfz REVOCAP_Refiner-X.X.X.tgz
```

4.2 OPTIONS ファイルの編集

REVOCAP_Refiner を展開したディレクトリには OPTIONS ファイルが存在する。そこに環境に応じたコンパイラの設定などが記述されている。主要な環境（PC-Cluster、HA8000 など）については設定例が記述されているので、参考にする。

4.3 make の実行

展開したディレクトリで、以下のコマンドを実行します。

```
make Refiner
```

REVOCAP_Refiner に関する make ターゲットでは、以下の処理を実行します。

- Refiner：REVOCAP_Refiner のライブラリをビルドします
- RefinerSampleC：REVOCAP_Refiner の C 言語サンプルをビルドします
- RefinerSampleF：REVOCAP_Refiner の Fortran 言語サンプルをビルドします
- RefinerDoc：REVOCAP_Refiner のドキュメントを生成します。Doxygen がインストールされている必要があります。
- RefinerClean：REVOCAP_Refiner に関連する生成したファイルを削除します。

4.4 生成したファイルの確認とテスト

OPTIONS ファイルに記述した LIB 変数と ARCH 変数の下に生成されたライブラリがある。ソルバがこのライブラリを利用するには、展開したディレクトリの Refiner/rcapRefiner.h をインクルードすればよい（他のヘッダーファイルはインクルードする必要はない）。

4.5　サンプルプログラム概説

詳細はチュートリアルを参照のこと。

main.c

もっとも単純な細分プログラムとして、2 つの四面体からなるモデルを細分する C 言語のプログラム。

mainHexa.c

２つの六面体からなるモデルを細分する C 言語のプログラム。

mainHexa2.c

２つの六面体 2 次要素からなるモデルを細分する C 言語のプログラム。

mainMultiStage.c

細分を 2 回繰り返す C 言語のプログラム。

mainMultiType.c

要素タイプが混在しているモデルを細分する C 言語のプログラム。

mainMultiByType.c

要素タイプが混在しているモデルを要素タイプごとにまとめて細分する C 言語のプログラム。

mainNV.c

境界面上の点を整数値で識別したモデルを細分する C 言語のプログラム。

mainFittingRefine.c

CAD 曲面による形状補正を行って細分する C 言語のプログラム。

mainShapeFuncFitting.c

２次要素の形状関数で形状補正をおこなって細分する C 言語のプログラム。

main.F90

main.c の Fortran90 言語版。

mainMulti.F90

mainMultiType.c の Fortran90 言語版。

test.rcap.f

FrontFlow/blue の中から呼び出す場合の Fortran 言語のサンプルプログラム。

5 ライブラリ API 規約

5.1 呼び出し方

REVOCAP_Refiner はソルバまたはカップラに組み込まれてそのプロセスの中で実行される（フォークしない)。新たなプロセスを起動してプロセス間通信でデータのやり取りをするのではなく、ソルバやカップラのプロセスの中で、関数を通じてデータのやり取りをする。関数への入力や出力となる変数のメモリ管理は呼び出し側が責任を持つ。ソルバやカップラへ受け渡さないREVOCAP_Refinerの内部データはREVOCAP_Refinerがメモリ管理について責任を持つ。

REVOCAP_Refiner のメモリ空間は呼び出されるプロセスの中で一意である。同じプロセスの中で複数の REVOCAP_Refiner の処理を行うことは、現バージョンでは未対応である。初期化と終了を正しく使わない場合はデータが破壊される場合があるので注意が必要である。

5.2 名前

名前は rcapXXX(...)とする

fortran77 言語、Fortran90 言語の呼び出しのための関数名の変換はマクロではなく、ラッパ関数を経由することで行う。ビルドする場合には、呼び出す Fortran のコンパイラの種類によってライブラリのコンパイル時に異なるマクロ定数を与えなければならない。

マクロ変数	シンボルの変換	対応しているコンパイラ
FORTRAN_CALL_C_DOWNCASE_（デフォルト）	関数名すべて小文字＋アンダースコア	gfortran Intel Fortran PGI Fortran
FORTRAN_CALL_C_DOWNCASE__	関数名すべて小文字＋アンダースコア２文字	
FORTRAN_CALL_C_UPCASE	関数名すべて大文字	
FORTRAN_CALL_C_UPCASE_	関数名すべて大文字＋アンダースコア	

なお、マクロ定数 FORTRAN90 は Fortran 言語から呼び出す場合は必ず指定すること。

5.3 配列のメモリ確保の方法

配列の受け渡しは呼び出し側で allocate して、このモジュールでは allocate も free もせず、

要素に代入するだけとする。
配列があふれた場合はエラーコードを返す。

5.4 定数

要素の型を表す RCAP_TETRAHEDRON, RCAP_TETRAHEDRON2, RCAP_HEXAHEDRON など
C 言語で使う場合は、この rcapRefiner.h を include する。
Fortran 言語で使う場合は rcapRefiner.inc を include する。

5.5 変数の型

UNIX 標準 C の stdint.h および、浮動小数点についても同様の定義をする。unsigned 型は Fortran で定義できない場合があるため使わない。C 言語の size_t 型はコンテナの大きさをあらわす数として使うが、Fortran ではラッパで符号付きに変換して使う。実行環境に依存する可能性のある型はプリプロセッサで吸収する。

5.6 入出力引数

- 引数は入力系を前に、出力系を後ろにする
- 配列の入力を与えるときは、配列の大きさと先頭アドレスを与える
- 要素の型など、他の情報で配列の大きさが決まる場合は、配列の大きさは省略する
- 出力で配列を用いる場合は、あらかじめ allocate された配列に値を代入することで行う
- 戻り値はエラー処理、出力で配列を用いた場合の値を代入した個数のために使う

5.7 要素の節点配列の順番

細分する要素の節点配列の順番は 8 要素ライブラリで挙げるとおりとする。ソルバ、カップラごとに異なる可能性があることに注意する。この順番は革新プロジェクトにおける REVOCAP_PrePost の内部で用いられているものである。
また、節点配列の順番や、面番号の付け方については定数配列としてあたえているので、ソルバ、カップラ、から適宜参照できるものとする。

6 プログラム構成

この章は REVOCAP_Refiner の内部の情報について述べている。ソルバ、カップラが利用する場合は特に API のみ意識すればよく、この章の内容は意識しなくて構わない。
REVOCAP_Refiner における要素ライブラリ、幾何処理、要素コンテナおよび節点コンテナについては REVOCAP_PrePost のメッシュ処理部と共通である。メッシュ処理部は次の 7 つのサブモジュールからなる。

(1) Geometry：幾何処理モジュール

(2) Matrix: 行列・ベクトル計算モジュール

(3) MeshDB：メッシュデータ管理モジュール

(4) MeshGL：メッシュ描画モジュール

(5) RevocapIO：入出力モジュール

(6) MeshGen：メッシュ生成モジュール

(7) Shape：CAD 形状処理モジュール

REVOCAP_Refiner ではメッシュ描画モジュール MeshGL は利用していない。
REVOCAP_PrePost のメッシュ処理部における要素コンテナ、節点コンテナはアクセス方法と、イテレータのインターフェイスが規定されているが、実装方法は特に決まっていない。したかって、目的に応じて最適な実装を選択することができる。
REVOCAP_Refiner では、ソルバまたはカップラに組み込まれて用いること、要素配列のためのメモリはソルバまたはカップラが確保するという規約になっているため、ここではソルバまたはカップラが確保したメモリを REVOCAP_PrePost の要素コンテナとしてアクセス可能にするためのラッパを実装している。
節点配列についても同様にメモリはソルバまたはカップラが確保するという規約になっているが、要素細分の場合に生成される中間節点のメモリは REVOCAP_Refiner が一時的に記憶しておく必要があるため、要素細分に適した節点コンテナを REVOCAP_Refiner の内部で実装している。
REVOCAP_Refiner のデータ構造は上記のとおり、REVOCAP_PrePost メッシュ処理部の節点コンテナ、要素コンテナを拡張している。独自のデータ構造は、細分時の中間節点マネージャ、細分の前後の境界条件を記憶するためのテーブルがある。従って、データ構造による階層は以下の表のようになる。

表 5.7.1 データ構造による階層

ソルバ・カップラ	節点配列、要素配列、境界条件配列
REVOCAP_Refiner	拡張節点コンテナ、拡張要素コンテナ、中間節点マネージャ、境界条件変換テーブル
REVOCAP_PrePost（メッシュ処理部）	基底節点コンテナ、基底要素コンテナ、データコンテナ、要素ライブラリ、幾何データ

ソルバ･カップラからの節点､要素をそれぞれ格納した配列は REVOCAP_Refiner により、拡張節点コンテナ、拡張要素コンテナにラップされて、REVOCAP_PrePost 基底節点コンテナ、基底要素コンテナの機能を用いて処理される。ソルバ・カップラからの境界条件を格納した配列は､REVOCAP_Refiner によって REVOCAP_PrePost のメッシュ処理部のデータコンテナに変換して格納される。

7 データ構造

この章は REVOCAP_Refiner の内部の情報について述べている。ソルバ、カップラが利用する場合は特に API のみ意識すればよく、この章の内容は意識しなくて構わない。
REVOCAP_Refiner における内部のデータ構造は、REVOCAP_PrePost におけるメッシュ処理部が受け持ち、次の 5 つに分類される。

(1) 節点情報

(2) 要素情報

(3) 領域情報

(4) 境界情報

(5) 解析情報

節点情報は 3 次元座標値から成る。3 次元座標値に対して唯一の識別子が与えられる。要素情報は節点の識別子によってコネクティビティを記述したものである。領域情報は、要素がどの材料領域に属するかを記述したものである。それぞれの要素の次元により、立体要素グループ（3 次元）、表面要素グループ（2 次元）、エッジグループ（1 次元）がある。
境界情報は、領域間の幾何学的相互関係を記述したものである。例えば、立体要素グループである領域情報の境界面を抽出して、表面要素グループとして部分領域情報としたときに、もとの領域情報と部分領域情報の間のマッピングを記述することができる。解析モデルのある平面を選択して境界条件を課すなどの通常の構造解析で必要となる設定をすることができる。
解析情報は、境界条件および材料条件を合わせた概念である。境界条件は境界情報（領域の境界であるような表面要素グループ）と物理値とのマッピングであり、材料条件は領域情報と物理値とのマッピングである。

8 要素ライブラリ

以下は REVOCAP_Refiner が内部データとして保持することのできる要素の一覧である。これは REVOCAP_PrePost と共通である。
接続行列は i 番目の節点と j 番目の節点の接続情報を表している。ただし、ライブラリの利用者が直接この行列を利用することはない。
1 次要素とみなしてつながっているときは±1 とし、
2 次要素とみなしてつながっているときは±2 とする。
符号は向きを表し、線要素、平面要素のみ有効である。
辺、面情報では局所面番号および面の中の節点の順番を記述してある。各行は立体要素の場合はその面、平面要素の場合はその辺の節点の接続情報が記述されている。（四面体要素なら、それぞれの行が三角形要素の節点 ID の配列）面は外から見て反時計回りが正となるように向き付けられている。

8.1 線分 1 次要素(SEGMENT)

接続行列

0	1
-1	0

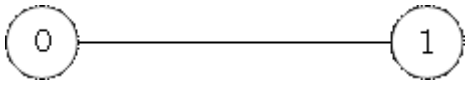

図 8-1 線分 1 次要素（SEGMENT）

8.2 線分 2 次要素 (SEGMENT2)

接続行列

0	1	2
-1	0	-2
-2	2	0

図 8-2 線分 2 次要素（SEGMENT2）

8.3　三角形 1 次要素 (TRIANGLE)

接続行列

0	1	-1
-1	0	1
1	-1	0

辺

1	2
2	0
0	1

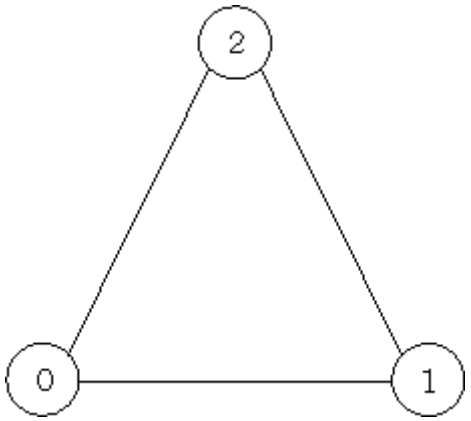

図 8-3　三角形 1 次要素（TRIANGLE）

8.4　三角形 2 次要素 (TRIANGLE2)

接続行列

0	1	-1	0	-2	2
-1	0	1	2	0	-2
1	-1	0	-2	2	0
0	-2	2	0	0	0
2	0	-2	0	0	0
-2	2	0	0	0	0

辺

1	2	3
2	0	4
0	1	5

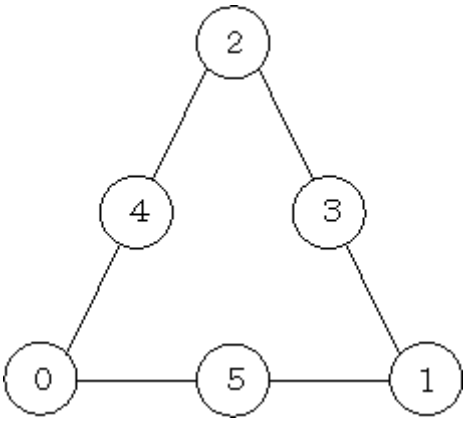

図 8-4　三角形 2 次要素（TRIANGLE2）

8.5　四角形 1 次要素 (QUAD)

接続行列

0	1	0	-1
-1	0	1	0
0	-1	0	1
1	0	-1	0

辺

0	1
1	2
2	3
3	0

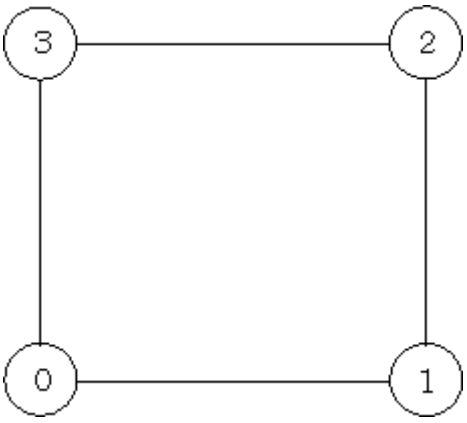

図 8-5　四角形 1 次要素（QUAD）

8.6　四角形 2 次要素 (QUAD2)

接続行列

0	1	0	-1	2	0	0	-2
-1	0	1	0	-2	2	0	0
0	-1	0	1	0	-2	2	0
1	0	-1	0	0	0	-2	2
-2	2	0	0	0	0	0	0
0	-2	2	0	0	0	0	0
0	0	-2	2	0	0	0	0
2	0	0	-2	0	0	0	0

辺

0	1	4
1	2	5
2	3	6
3	0	7

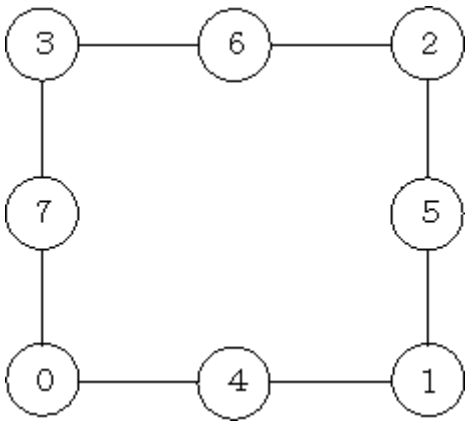

図 8-6　四角形 2 次要素（QUAD2）

8.7　四面体 1 次要素 (TETRAHEDRON)

接続行列

0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

面

1	2	3
0	3	2
0	1	3
0	2	1

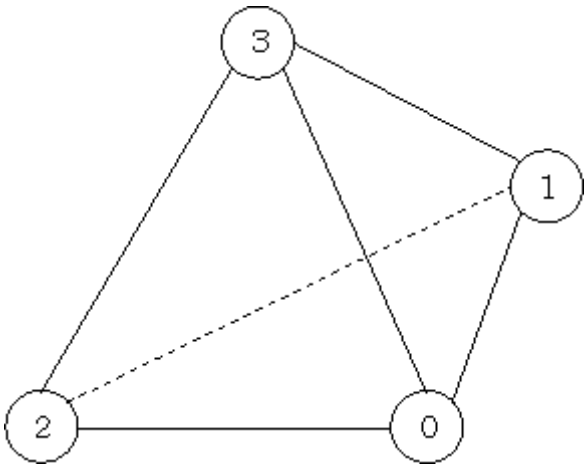

図 8-7　四面体 1 次要素（TETRAHEDRON）

8.8　四面体 2 次要素 (TETRAHEDRON2)

接続行列

0	1	1	1	0	2	2	2	0	0
1	0	1	1	2	0	2	0	2	0
1	1	0	1	2	2	0	0	0	2
1	1	1	0	0	0	0	2	2	2
0	2	2	0	0	0	0	0	0	0
2	0	2	0	0	0	0	0	0	0
2	2	0	0	0	0	0	0	0	0
2	0	0	2	0	0	0	0	0	0
0	2	0	2	0	0	0	0	0	0
0	0	2	2	0	0	0	0	0	0

面

1	2	3	9	8	4
0	3	2	9	5	7
0	1	3	8	7	6
0	2	1	4	6	5

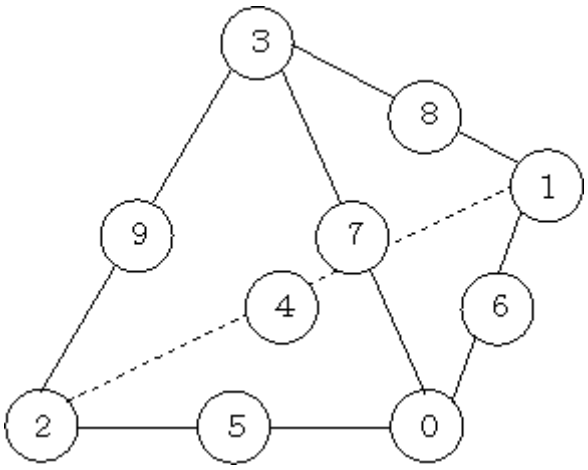

図 8-8　四面体 2 次要素（TETRAHEDRON2）

8.9　六面体 1 次要素 (HEXAHEDRON)

接続行列

0	1	0	1	1	0	0	0
1	0	1	0	0	1	0	0
0	1	0	1	0	0	1	0
1	0	1	0	0	0	0	1
1	0	0	0	0	1	0	1
0	1	0	0	1	0	1	0
0	0	1	0	0	1	0	1
0	0	0	1	1	0	1	0

面

3	2	1	0
4	5	6	7
1	5	4	0
1	2	6	5
3	7	6	2
4	7	3	0

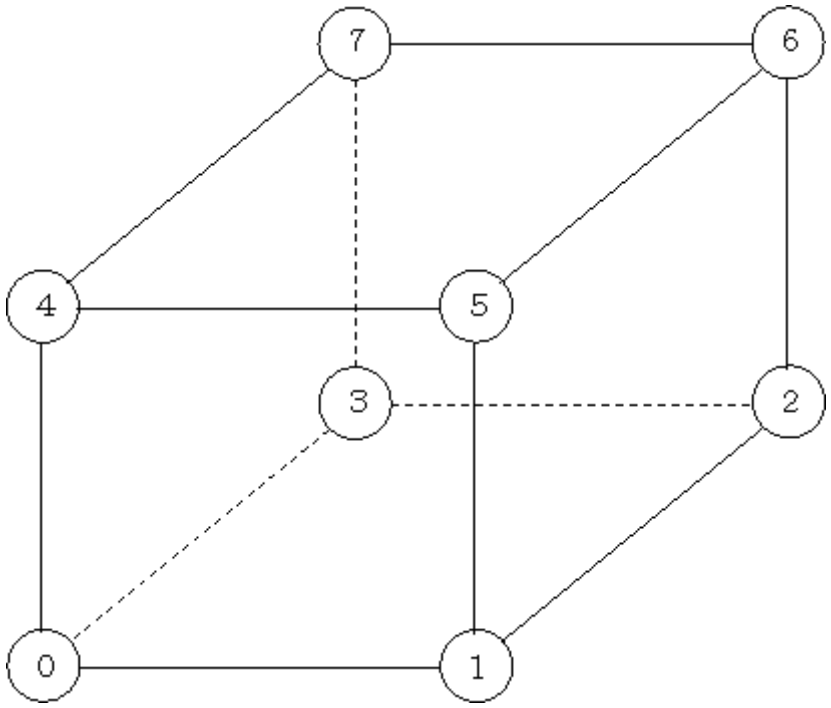

図 8-9　六面体 1 次要素（HEXAHEDRON）

8.10 六面体 2 次要素 (HEXAHEDRON2)

接続行列

0	1	0	1	1	0	0	0	2	0	0	2	0	0	0	0	2	0	0	0
1	0	1	0	0	1	0	0	2	2	0	0	0	0	0	0	0	2	0	0
0	1	0	1	0	0	1	0	0	2	2	0	0	0	0	0	0	0	2	0
1	0	1	0	0	0	0	1	0	0	2	2	0	0	0	0	0	0	0	2
1	0	0	0	0	1	0	1	0	0	0	0	2	0	0	2	2	0	0	0
0	1	0	0	1	0	1	0	0	0	0	0	2	2	0	0	0	2	0	0
0	0	1	0	0	1	0	1	0	0	0	0	0	2	2	0	0	0	2	0
0	0	0	1	1	0	1	0	0	0	0	0	0	0	2	2	0	0	0	2
2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	2	2	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0

2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0

面

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

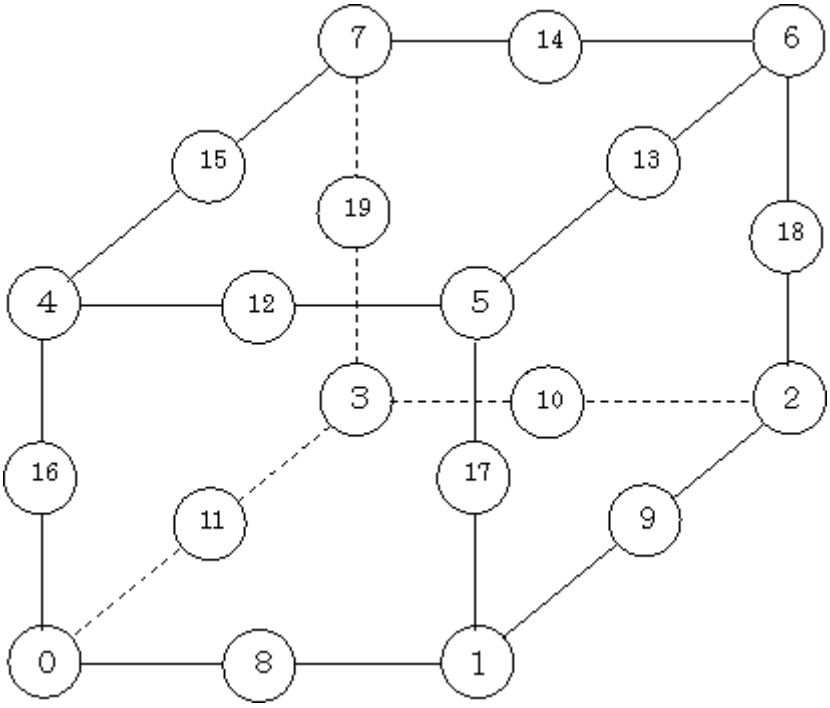

図 8-10　六面体 2 次要素（HEXAHEDRON2）

8.11 三角柱 1 次要素 (WEDGE)

接続行列

0	1	1	1	0	0
1	0	1	0	1	0
1	1	0	0	0	1
1	0	0	0	1	1

0	1	0	1	0	1
0	0	1	1	1	0

面

0	2	1	
3	4	5	
0	1	4	3
1	2	5	4
2	0	3	5

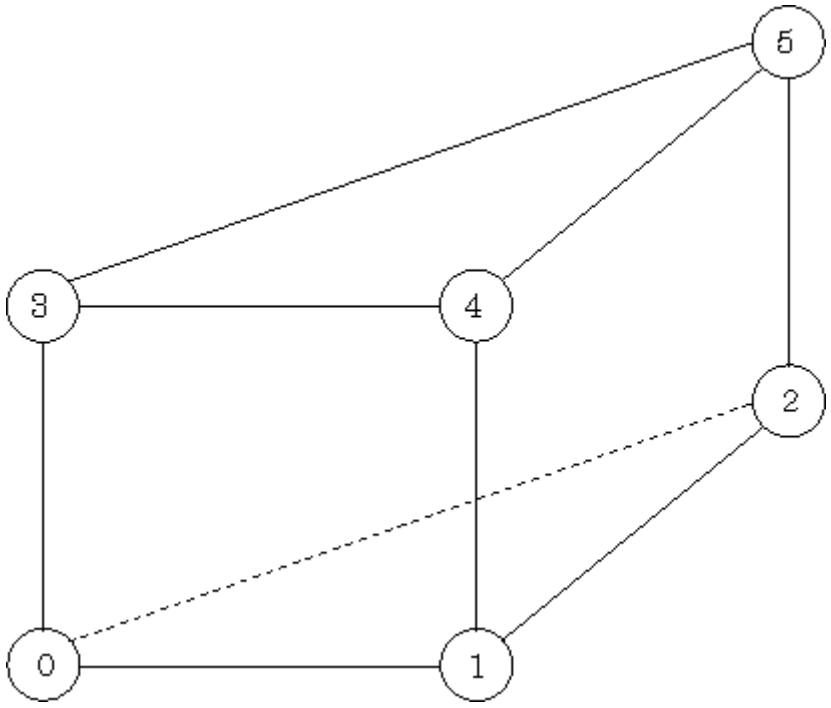

図 8-11　三角柱 1 次要素（WEDGE）

8.12 三角柱 2 次要素 (WEDGE2)

接続行列

0	1	1	1	0	0	0	2	2	0	0	0	2	0	0
1	0	1	0	1	0	2	0	2	0	0	0	0	2	0
1	1	0	0	0	1	2	2	0	0	0	0	0	0	2
1	0	0	0	1	1	0	0	0	0	2	2	2	0	0
0	1	0	1	0	1	0	0	0	2	0	2	0	2	0
0	0	1	1	1	0	0	0	0	2	2	0	0	0	2
0	2	2	0	0	0	0	0	0	0	0	0	0	0	0

2	0	2	0	0	0	0	0	0	0	0	0	0	0	0
2	2	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2	2	0	0	0	0	0	0	0	0	0
0	0	0	2	0	2	0	0	0	0	0	0	0	0	0
0	0	0	2	2	0	0	0	0	0	0	0	0	0	0
2	0	0	2	0	0	0	0	0	0	0	0	0	0	0
0	2	0	0	2	0	0	0	0	0	0	0	0	0	0
0	0	2	0	0	2	0	0	0	0	0	0	0	0	0

面

0	2	1	6	8	7		
3	4	5	9	10	11		
0	1	4	3	8	13	11	12
1	2	5	4	6	14	9	13
2	0	3	5	7	12	10	14

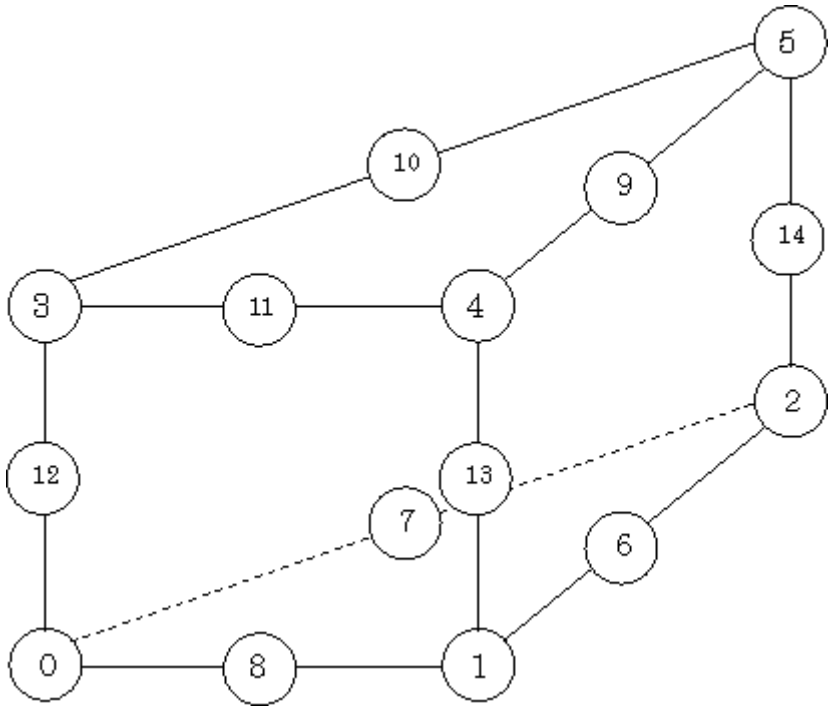

図 8-12　三角柱 2 次要素（WEDGE2）

8.13 四角錐 1 次要素 (PYRAMID)

接続行列

0	1	1	1	1

1	0	1	0	1
1	1	0	1	0
1	0	1	0	1
1	1	0	1	0

面

0	1	2	
0	2	3	
0	3	4	
0	4	1	
4	3	2	1

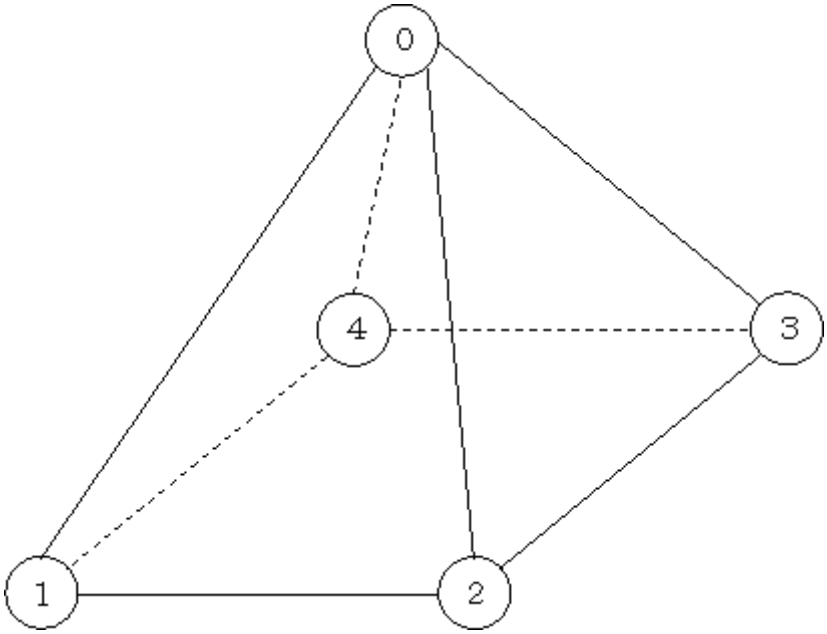

図 8-13　四角錐 1 次要素（PYRAMID）

8.14 四角錐 2 次要素 (PYRAMID2)

接続行列

0	1	1	1	1	2	2	2	2	0	0	0	0
1	0	1	0	1	2	0	0	0	2	0	0	2
1	1	0	1	0	0	2	0	0	2	2	0	0
1	0	1	0	1	0	0	2	0	0	2	2	0
1	1	0	1	0	0	0	0	2	0	0	2	2

2	2	0	0	0	0	0	0	0	0	0	0	0
2	0	2	0	0	0	0	0	0	0	0	0	0
2	0	0	2	0	0	0	0	0	0	0	0	0
2	0	0	0	2	0	0	0	0	0	0	0	0
0	2	2	0	0	0	0	0	0	0	0	0	0
0	0	2	2	0	0	0	0	0	0	0	0	0
0	0	0	2	2	0	0	0	0	0	0	0	0
0	2	0	0	2	0	0	0	0	0	0	0	0

面

0	1	2	9	6	5		
0	2	3	10	7	6		
0	3	4	11	8	7		
0	4	1	12	5	8		
4	3	2	1	11	10	9	12

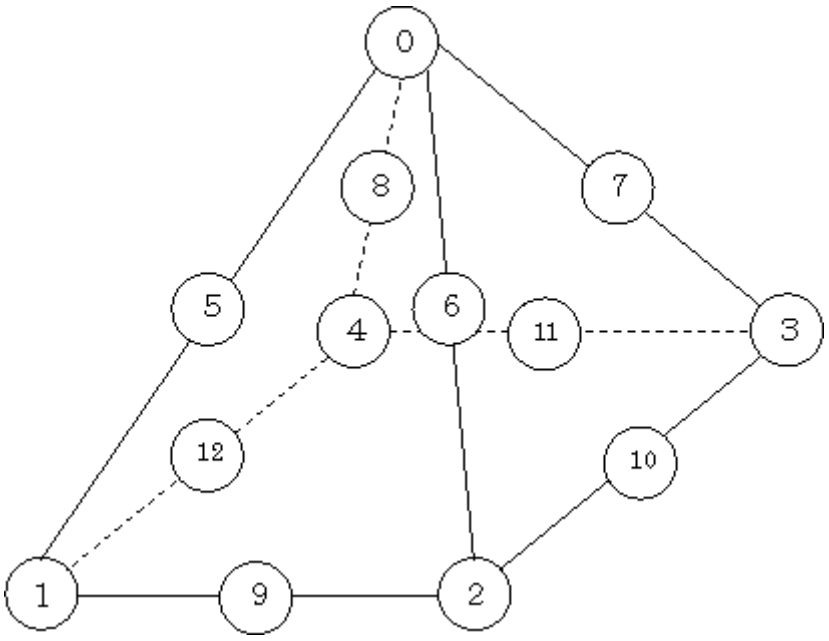

図 8-14 四角錐２次要素(PYRAMID2)

9　解析モデルの細分

9.1　四面体１次要素の細分

以下の手順で行われる。

(1)　それぞれの辺に中間節点を追加する

(2)　元の四面体の頂点を含むような４つの四面体と中央の八面体に分割する

(3)　中央の八面体について最も長さの短い対角線を追加して、４つの四面体に分割する

結果として８個の四面体に分割される。

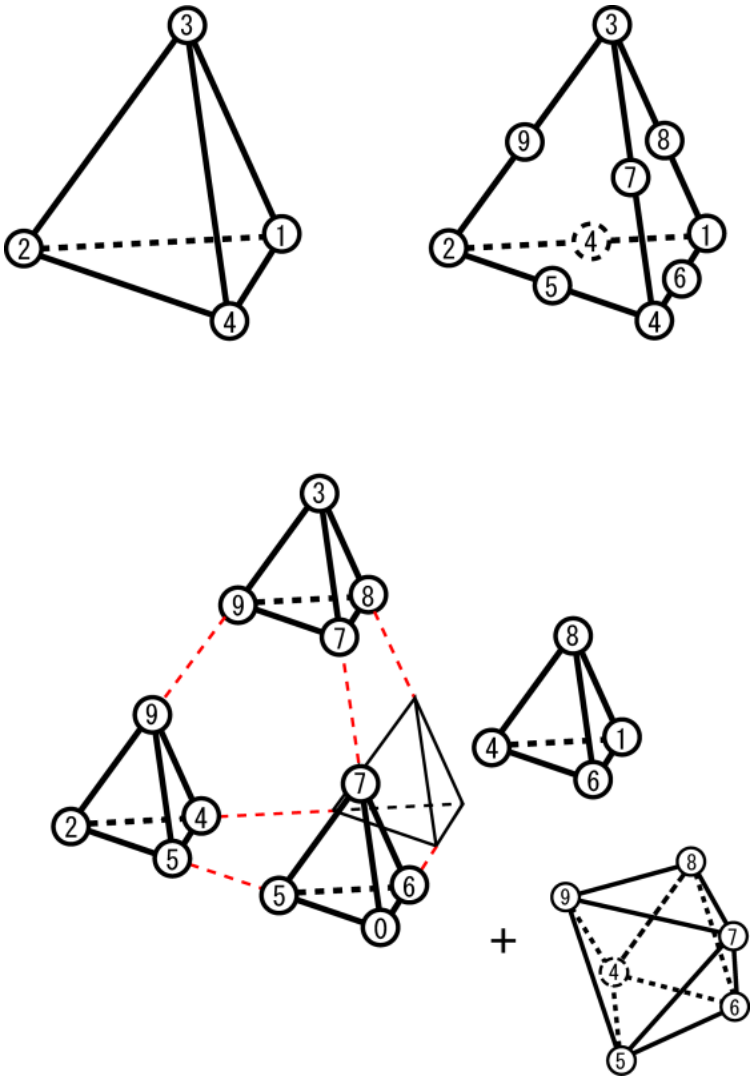

図 9-1　四面体細分（１）

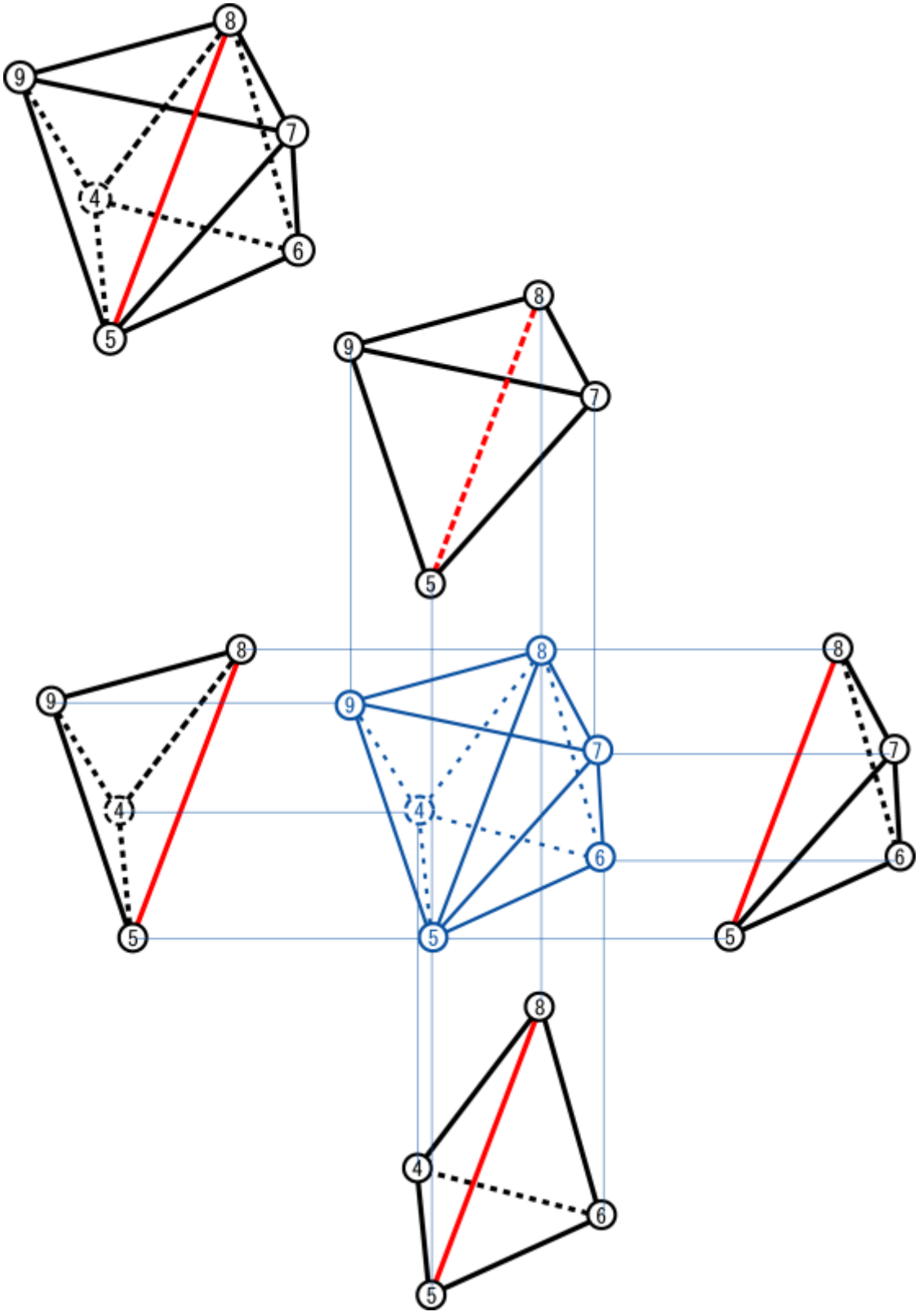

図 9-2　四面体細分（２）

9.2　六面体１次要素の細分

六面体については以下のようにして細分される。

(1) それぞれの辺の中点、面の中心、要素全体の中心に節点を追加する

(2) 元の六面体の頂点を含むような六面体８個に分割する。

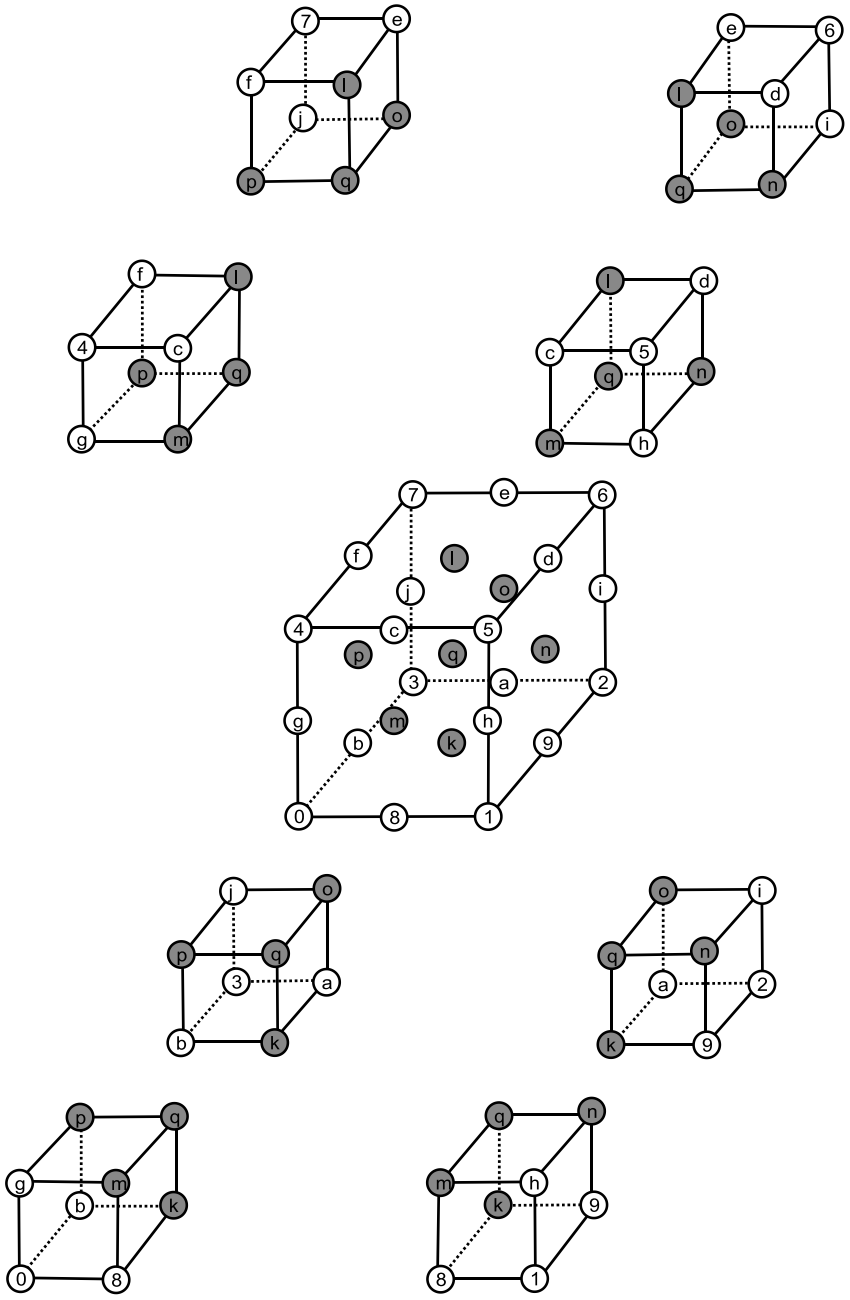

図 9-3 六面体細分

9.3　三角柱１次要素の細分

三角柱については以下のように細分される

(1) それぞれの辺の中点と四角形の面の中心に節点を追加する

(2) 四角形の面の中心の３つの節点を通るような面で元の三角柱を２つに分割する

(3) 分割した三角柱を、三角形の面の分割を上面、底面に適用して４つの三角柱に分割する

結果として８つの三角柱が得られる。

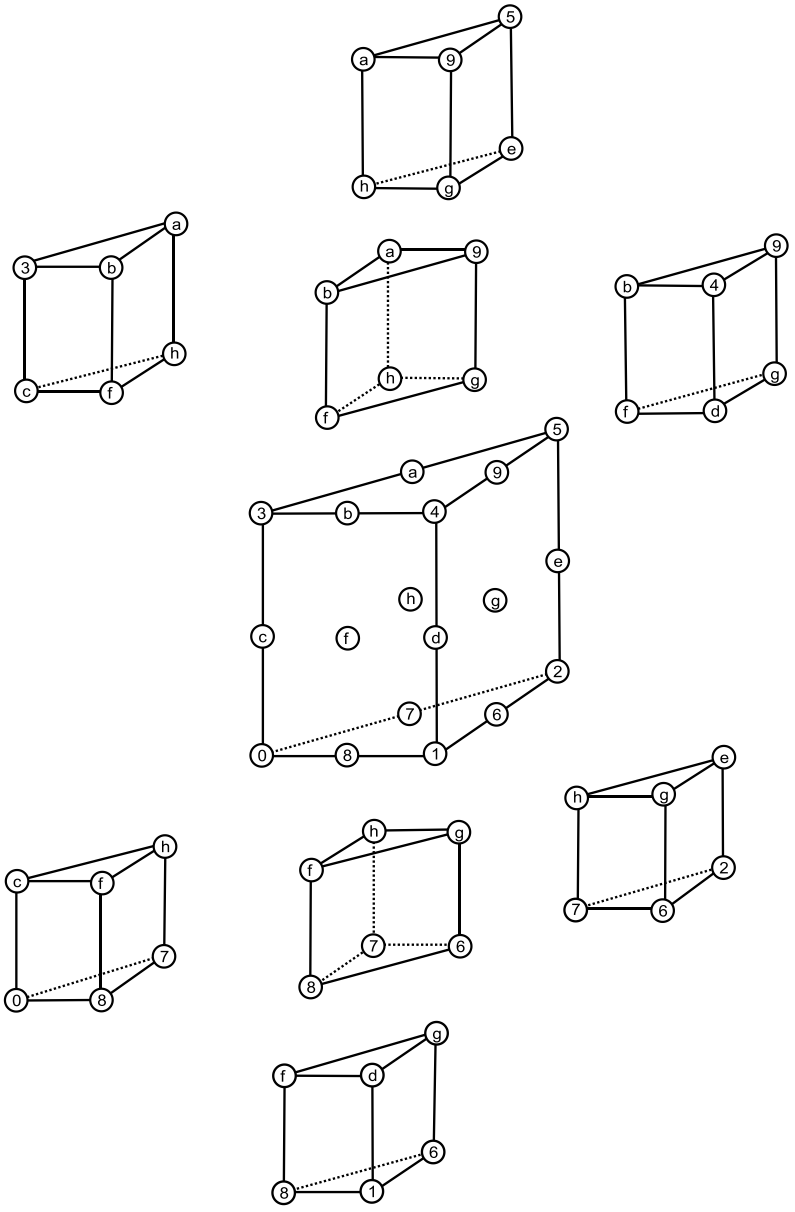

図 9-4 三角柱細分

9.4　四角錐１次要素の細分

四角錐の細分は以下のように行う

(4) それぞれの辺の中点と、底面の四角形の中心に節点を追加する。

(5) 元の四角錐の頂点を含むような５つの四角錐、底面の四角形の中心と側面の三角形の３つの中間節点による４つの四面体、底面の四角形の中心を天頂とするような四角錐に分割する。

結果として６つの四角錐と４つの四面体の合計 10 個の要素が得られる。

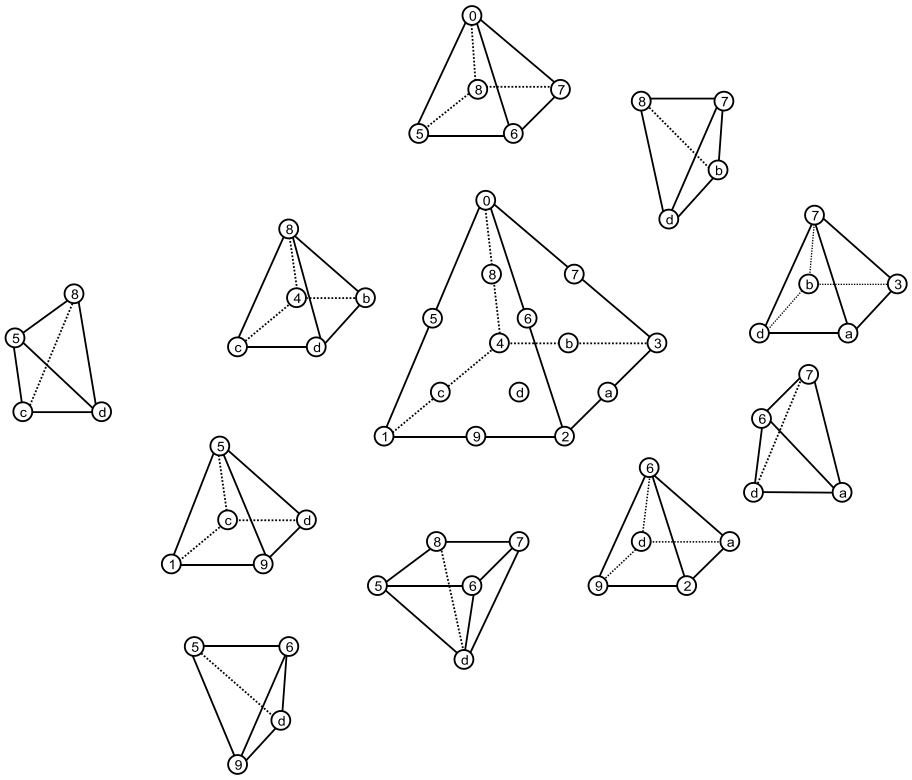

図 9-5 四角錐細分

9.5　三角形１次要素の細分

それぞれの辺の中点に新たな節点を追加し、新たに 4 個の要素を生成する。

9.6　線分１次要素の細分

それぞれの辺の中点に新たな節点を追加し、新たに 2 個の要素を生成する。

9.7　境界条件

REVOCAP_Refiner は要素の細分の前に境界条件を登録しておくと、細分するときに同時

に更新することができる。更新できる境界条件の種類は次の通りである。

- 節点グループ
- 要素グループ
- 面グループ（要素番号と要素内面番号の組）

分布については対応していない。FrontSTR の節点拘束、面荷重（一定値）、FrontFlow/blue の境界条件については上記の境界条件で対応可能である。

境界条件の更新のための関数があるのではなく、細分時の更新は要素の細分と同時に行われる。従って、境界条件の細分は以下の手順で行う。

(1) 境界条件を Refiner に登録する(rcapAppendNodeGroup、rcapAppendElementGroup、rcapAppendFaceGroup)

(2) 細分前の節点を Refiner に登録する(rcapSetNodeXX)

(3) 要素を細分する(rcapRefineElement, rcapRefineElementMulti)

(4) 境界条件をコミットする(rcapCommit)

(5) 更新された境界条件を Refiner から取得する(rcapGetNodeGroup、rcapGetElementGroup、rcapGetFaceGroup)

(1)と(2)は順序が逆でもよい。

複数の境界条件を同時に更新するために境界条件には名前をつけて登録する。更新後に取得するにはその名前を Refiner に与える。

9.8 節点グループの更新規則

節点グループの要素細分時の更新規則は以下の通りとする。

- 要素の辺の中点に中間節点が与えられた場合は、その辺の両端の節点がともに節点グループに含まれる場合に限り、中点を節点グループに追加する。
- 要素の面の中心に節点が与えられた場合は、その面の頂点の節点がすべて節点グループに含まれる場合に限り、中心を節点グループに追加する。
- 要素の中心に節点が与えられた場合は、その要素の頂点の節点がすべて節点グループに含まれる場合に限り、中心を節点グループに追加する。

すなわち周辺の節点についての論理積を取る。

10 形状補正機能

10.1 CAD 形状データによる形状補正のための前処理

CAD の形状データを用いて、解析モデルの要素細分を行ったときに形状の補正を行うには、細分前に CAD の形状データと、解析モデルの節点の間の対応付けを行っておく必要がある。

10.2 CAD 形状データによる細分

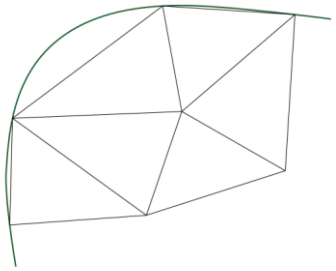

図 10-1　細分前

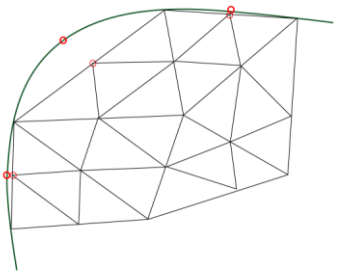

図 10-2　細分後

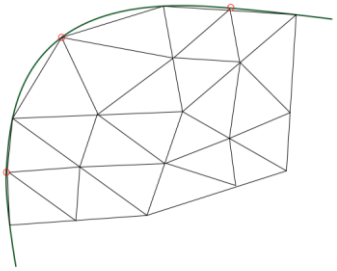

図 10-3 形状補正後

10.3 領域分割と形状補正の関係

REVOCAP_Refiner はソルバが並列計算を行う場合には領域分割後のモデルに対して要素の細分を行うが、領域分割後は計算ノードごとの局所節点番号でモデルが記述されているため、節点と CAD の形状データとの関係が与えられていない。そこで REVOCAP_Refiner があらかじめ計算ノードでの局所節点番号と大域的な節点番号を管理することで、局所節点番号と CAD の形状データとの関係を再構築する。それによって、領域分割後も形状補正が可能になる。
ソルバまたはカップラが大域節点番号と局所節点番号の対応を管理している場合は、ソルバまたはカップラが REVOCAP_Refiner を呼び出す場合にその対応を与えることで領域分割後の形状補正を行えるようにする。
実際に並列処理時に形状補正を行うためには以下の手順で行う。

(1) CAD データから疎なメッシュを生成するときに、曲面上にある節点と CAD 曲面の情報との対応をファイルに記録する

(2) 疎なメッシュを領域分割し、節点の大域番号と局所番号の対応をファイルに記録する

(3) 領域分割後のそれぞれの計算ノードでソルバおよびカップラが REVOCAP_Refiner を呼び出す

(4) 各領域ごと REVOCAP_Refiner に領域分割されたモデルを読み込む

(5) 節点の大域番号と局所番号の対応を記録したファイルを読み込む

(6) 曲面上の節点の大域番号と CAD の曲面の対応を記録したファイルを読み込む

(7) 曲面上にある領域分割後の節点の局所番号と CAD の曲面の対応関係を得る

(8) 要素の細分時にその要素の面が曲面上にあるときは、CAD の形状情報を使って、細分した面の節点を曲面上に乗るように補正をする

(9) 必要な場合は要素の細分の仕方を変えて、メッシュの品質の悪化を防ぐ

(10) 細分後に生成された中間節点を含め、曲面上にある節点の局所番号と CAD の曲面の対応情報をファイルに記録する

細分時に形状補正を行った場合に出力されるファイルは、節点の局所番号と CAD の曲面の対応が与えられている。従ってもう一度細分する場合に形状補正する場合には、大域節点番号を与えずに行えばよい。
すなわち２回目以降の細分時の形状補正、並列処理しない場合の細分時の形状補正は以下

の手順で行う。

(1) 曲面上にある節点番号と CAD 曲面の情報との対応をファイルに記録する

(2) 領域分割後のそれぞれの計算ノードでソルバおよびカップラが REVOCAP_Refiner を呼び出す

(3) REVOCAP_Refiner にモデルを読み込む

(4) 曲面上の節点と CAD の曲面の対応を記録したファイルを読み込む

(5) 要素の細分時にその要素の面が曲面上にあるときは、CAD の形状情報を使って、細分した面の節点を曲面上に乗るように補正をする

(6) 必要な場合は要素の細分の仕方を変えて、メッシュの品質の悪化を防ぐ

(7) 細分後に生成された中間節点を含め、曲面上にある節点の局所番号と CAD の曲面の対応情報を出力する

すなわち上記の(2) (5) (7)の工程を省略できる。
節点と CAD 曲面の対応関係を記録したファイル、節点の大域番号と局所番号を記録したファイル、についてのファイルフォーマットは ADVENTURE 形式および IGES 形式などを適宜利用するものとする。

10.4 形状補正機能の制限事項

形状補正機能については以下の制限がある。

曲面データ構造について

REVOCAP_Refiner が扱うことのできる曲面のデータは 11CAD ファイルフォーマットで与えられる Nurbs 形式に限ることに注意する。特に Primitive で表現されることの多い、単純な平面、円柱、円錐、球面なども同値な Nurbs 表現に変換して与えなければならない。また、単一の曲面で曲面座標(u,v)が周期的になっている場合、すなわち u の最小値と最大値で空間の同一の点を与えるような場合は REVOCAP_Refiner では扱うことができない。このような場合は、(多くの場合形式的に) いくつかの曲面に分ける必要がある。

曲面データをまたがる要素について

REVOCAP_Refiner に与えられるデータは曲面データをまたがる要素ご存在してはならない。特に円柱や球面を複数の Nurbs 面で表現したときに、それらの境界面をまたぐような要素が存在した場合は、その要素の中間節点をどの曲面データに適合させるかが決まらないため、形状適合が働かない場合がある。

11 CAD ファイルフォーマット

ここでは rcapSetCADFilename 関数で呼ばれる CAD ファイルのフォーマットついて記述する。
はじめに概要をまとめる。

- YAML フォーマットで記述する
- 記述可能な曲面の種類は、NURBS、B-Spline、Bezier の３種類である
- 同一のファイルにグローバル節点番号と曲面の間の対応関係も記述する
- このファイルは REVOCAP_PrePost が生成することができる

一般的な YAML フォーマットの仕様として、インデントを使い階層構造を表現する。ただしインデントにはタブが使えずに、スペースのみが使えることに注意する。ここではスペース 2 個でインデントする。その他の YAML についての仕様は http://yaml.org/ 等を参照していただきたい。

11.1 トップレベルノード

曲面のコレクションを記述する surface、および節点番号と曲面の対応関係を記述する data がある。

11.2 surface ノード

surface ノードの下位は nurbs、bspline、 bezier のいずれかからなる曲面のノードのコレクションである。

11.3 nurbs ノード

nurbs ノードは id、uknots、vknots、ctrlpts をキーとするマッピングである。

id		整数値	曲面を識別するための 0 以上の整数を与える。
uknots	order	整数値	nurbs 曲面の u 方向の位数を与える。
	vector	浮動小数点値のリスト	nurbs 曲面の u 方向のノットベクトルをあたえる。
vknots	order	整数値	nurbs 曲面の v 方向の位数を与える。

	vector	浮動小数点値のリスト	nurbs 曲面の v 方向のノットベクトルを与える。
ctrlPts		浮動小数点の 4 次元ベクトルのリスト	nurbs 曲面の制御点の (x,y,z) 座標とウェイトを合わせた 4 次元ベクトルのリストを与える。

11.4 data ノード

data ノードは name、mode、vtype、stype、size、value からなるマッピングのコレクションである。
これは REVOCAP_PrePost のニュートラルメッシュファイルのフォーマットと共通である。

name	文字列	他のデータと区別するための識別子。ここでは'fitting'という文字列を与える。
mode	文字列	データの種類を規定するための識別子。ここでは'NODEVARIABLE'という文字列を与える。
vtype	文字列	データの値を規定するための識別子。ここでは'VECTOR2WITHINT'という文字列を与える。
stype	文字列	このデータの付加的な意味を与えるための識別子。プログラム内部での処理のために使う。ここでは 'shape' という文字列を与える。
size	整数値	データの個数を与える。
value	整数値と（浮動小数点 2 個と整数値のベクトル）のペア	最初の整数値でグローバル節点番号、浮動小数点 2 個(u,v)と整数値は、整数値で曲面の id を与え、(u,v) で節点をその曲面上の最も近い

		点で与えたときの（u,v）座標を与える。 節点番号に対して、複数のベクトルを与えてもよく、ベクトルが与えられていない節点が存在してもよい。 このファイルでは、ソルバーの節点番号の付け方にかかわらず、0 から始まる。

11.5 例

例として、2 つ並んだ 6 面体の節点が円柱の表面で近似される場合を挙げる。

```
# 0.70710678 = sqrt(0.5)
# center (0.0, 0.0, 0.0)
# radius 1.0
# axis (1.0, 0.0, 0.0)
# height 2.0
---
surface:
  - nurbs:
      id: 0
      uknots:
        order: 3
        vector: [0.0, 0.0, 0.0, 0.5, 0.5, 1.0, 1.0, 1.0]
      vknots:
        order: 2
        vector: [0.0, 0.0, 1.0, 1.0]
      ctrlpts:
        - [0.0,   1.0,   0.0,   1.0]
        - [0.0,   1.0,   1.0,   0.70710678]
        - [0.0,   0.0,   1.0,   1.0]
        - [0.0, -1.0,   1.0,   0.70710678]
        - [0.0, -1.0,   0.0,   1.0]
        - [2.0,   1.0,   0.0,   1.0]
        - [2.0,   1.0,   1.0,   0.70710678]
        - [2.0,   0.0,   1.0,   1.0]
```

```
        - [2.0, -1.0,    1.0,    0.70710678]
        - [2.0, -1.0,    0.0,    1.0]
  - nurbs:
      id: 1
      uknots:
        order: 3
        vector: [0.0, 0.0, 0.0, 0.5, 0.5, 1.0, 1.0, 1.0]
      vknots:
        order: 2
        vector: [0.0, 0.0, 1.0, 1.0]
      ctrlpts:
        - [0.0, -1.0,    0.0,    1.0]
        - [0.0, -1.0, -1.0,    0.70710678]
        - [0.0,    0.0, -1.0,    1.0]
        - [0.0,    1.0, -1.0,    0.70710678]
        - [0.0,    1.0,    0.0,    1.0]
        - [2.0, -1.0,    0.0,    1.0]
        - [2.0, -1.0, -1.0,    0.70710678]
        - [2.0,    0.0, -1.0,    1.0]
        - [2.0,    1.0, -1.0,    0.70710678]
        - [2.0,    1.0,    0.0,    1.0]
data:
  - name: fitting
    mode: NODEVARIABLE
    vtype: VECTOR2WITHINT
    stype: shape
    size: 18
    value:
    - [0, [0.0, 0.0, 0]]
    - [1, [0.5, 0.0, 0]]
    - [2, [1.0, 0.0, 0]]
    - [4, [0.0, 0.5, 0]]
    - [5, [0.5, 0.5, 0]]
    - [6, [1.0, 0.5, 0]]
    - [8, [0.0, 1.0, 0]]
    - [9, [0.5, 1.0, 0]]
```

- [10, [1.0, 1.0, 0]]
- [2, [0.0, 0.0, 1]]
- [3, [0.5, 0.0, 1]]
- [0, [1.0, 0.0, 1]]
- [6, [0.0, 0.5, 1]]
- [7, [0.5, 0.5, 1]]
- [4, [1.0, 0.5, 1]]
- [10, [0.0, 1.0, 1]]
- [11, [0.5, 1.0, 1]]
- [8, [1.0, 1.0, 1]]

12 形状関数による補正機能

2 次要素については、形状関数で与えられる要素内の座標が線形ではないので、要素内座標での中点と、幾何的な中点が異なる場合がある。ここでは前者の形状関数で中点を求めることで、境界を形状関数が表す曲面として補正する。
通常、中間節点が幾何的な中点と異なるものは境界面にだけ現れると考えられるため、ここでは要素の境界面上にある中間節点を生成するときに、境界面の要素の形状関数を使って補正を行う。有限要素法では、四面体の形状関数を境界面に制限したものが三角形の形状関数となるため、境界面に制限して計算することは妥当である。三角形 2 次要素の形状関数として、以下のものを採用した。

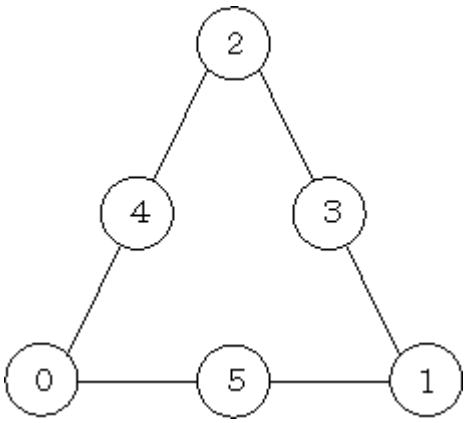

図 12-1 三角形 2 次要素

0	(1-s-t)*(1-2*s-2*t)
1	s*(2*s-1)
2	t*(2*t-1)
3	4*s*t
4	4*t*(1-s-t)
5	4*s*(1-s-t)

この 2 次要素を細分するときの要素内座標とそれぞれの形状関数の値をまとめると、次のようになる。

頂点番号	s	t	0	1	2	3	4	5
0と5	0.25	0	0.375	−0.125	0	0	0	0.75
5と1	0.75	0	−0.125	0.375	0	0	0	0.75
1と3	0.75	0.25	0	0.375	−0.125	0.75	0	0
3と2	0.25	0.75	0	−0.125	0.375	0.75	0	0
2と4	0	0.75	−0.125	0	0.375	0	0.75	0
4と0	0	0.25	0.375	0	−0.125	0	0.75	0
5と3	0.5	0.25	−0.125	0	−0.125	0.5	0.25	0.5
3と4	0.25	0.5	−0.125	−0.125	0	0.5	0.5	0.25
4と5	0.25	0.25	0	−0.125	−0.125	0.25	0.5	0.5

そこで次の3点関数と5点関数を内部で実装した。

createMiddleNode3(n0,n1,n2) = 0.375*n0 + 0.75*n1 -0.125*n2

createMiddleNode5(n0,n1,n2,n3,n4) = 0.5*n0 + 0.5*n1 -0.125*n2 – 0.125*n3 + 0.25*n4

三角形2次要素の辺上の中間節点の生成にはcreateMiddleNode3を用い、内部にある中間節点の生成にはcreateMiddleNode5を用いた。

13 メッシュ品質レポート機能

一般にメッシュの品質の指標とされているものとして

- アスペクト比
- 辺の長さの比
- 最大角度
- 最小角度

などが挙げられる。REVOCAP_Refiner では、rcapQualityReport 関数で、第一引数でメッシュの品質の指標名を文字列で与えて、レポートをファイルに出力する機能を実装している。レポート機能としては、それぞれの指標の最大値、最小値、平均値、簡易度数分布（ヒストグラム）を出力する。

付録: REVOCAP_Refiner 関数 API 一覧

- void **rcapGetVersion** (void)

Version 文字列を標準出力に出力する。

- void **rcapInitRefiner** (int32_t nodeOffset, int32_t elementOffset)

Refiner を初期化し、節点番号と要素番号のオフセット値を与える。

- void **rcapClearRefiner** (void)

Refiner が内部で保持している中間節点データのキャッシュをクリアする。

- void **rcapTermRefiner** (void)

Refiner の終了処理を行う。これを呼び出した後はすべての処理が無効になる。

- void **rcapSetCADFilename** (const char *filename)

形状補正に用いる CAD ファイルを指定する。

細分前のメッシュはこの CAD ファイルから生成されているものとする。 この関数を呼んで CAD ファイルを指定しなかった場合は、形状補正は行わない。 メッシュ生成時の節点の globalID と CAD の形状データの対応が与えられているとする。領域分割後のメッシュに対して細分を行う場合は、setPartitionFilename などで globalID と localID の対応付けを与える必要がある。

- void **rcapSetSecondFitting** (int32_t flag)

中間節点の生成に2次要素の形状関数を使うかどうかを設定する。

- void **rcapSetPartitionFilename** (const char *filename)

節点の globalID と localID の対応関係を記述したファイルを指定する。

指定しない場合は、globalID と localID は区別しない。 ファイルではなく、節点座標を登録するときに globalID と localID の関係を与えることもできる。

- void **rcapSetNode64** (size_t num, **float64_t** *coords, int32_t *globalIds, int32_t *localIds)

節点座標を Refiner に与える

- void **rcapSetNode32** (size_t num, **float32_t** *coords, int32_t *globalIds, int32_t *localIds)

節点座標を Refiner に与える

- size_t **rcapGetNodeCount** (void)

現在 Refiner が保持している節点の個数を返す。細分したら増える。

- void **rcapGetNode64** (size_t num, int32_t *localIds, **float64_t** *coords)

Refiner が管理している節点座標を取得する

- void **rcapGetNode32** (size_t num, int32_t *localIds, **float32_t** *coords)

Refiner が管理している節点座標を取得する

- void **rcapGetNodeSeq64** (size_t num, size_t initId, **float64_t** *coords)

initId から連続して num 個の節点座標を取得する

- void **rcapGetNodeSeq32** (size_t num, size_t initId, **float32_t** *coords)

rcapGetNodeSeq64 の 32bit 版

- size_t **rcapRefineElement** (size_t num, int8_t etype, int32_t *nodeArray, int32_t *resultNodeArray)

要素をそれぞれ辺の２分割して細分する

- size_t **rcapRefineElementMulti** (size_t num, int8_t *etypeArray, int32_t *nodeArray, size_t *refinedNum, int8_t *resultEtypeArray, int32_t *resultNodeArray)

複数の種類の型が混在しているモデルを一度に細分する

- void **rcapCommit** (void)

rcapRefineElement により細分されたデータ（節点グループ、要素グループ、面グループ）をコミットする。 すなわち、以下の rcapGet[Node|Element|Face]Group[Count] メソ'83bドの対象を細分前のデータから 細分後のデータに変える。細分前のデータは削除される。 この関数を実行後に rcapRefineElement を再度実行した場合、更新されるデータは 細分後のデータになる。 また、rcapRefineElement で細分後の要素に付与される要素番号も elementOffset 値にリセットされる。

- void **rcapAppendNodeGroup** (const char dataname[80], size_t num, int32_t *nodeArray)

細分と同時に更新する節点グループを登録

- size_t **rcapGetNodeGroupCount** (const char dataname[80])

Refiner に登録されている節点グループの節点の個数を返す

- void **rcapGetNodeGroup** (const char dataname[80], size_t num, int32_t *nodeArray)

Refiner に登録されている節点グループを返す

- void **rcapAppendBNodeGroup** (const char dataname[80], size_t num, int32_t *nodeArray)

BoundaryNodeGroup とは、境界面上にのみある節点グループのこと。この関数で登録する。

- size_t **rcapGetBNodeGroupCount** (const char dataname[80])

Refiner に登録されている境界節点グループの節点の個数を返す

- void **rcapGetBNodeGroup** (const char dataname[80], size_t num, int32_t *nodeArray)

Refiner に登録されている境界節点グループを返す

- void **rcapAppendBNodeVarInt** (const char dataname[80], size_t num, int32_t *nodeArray, int32_t *nodeVars)

BoundaryNodeVariableInt とは、境界面上にのみある節点上の整数値変数のこと

- size_t **rcapGetBNodeVarIntCount** (const char dataname[80])

Refiner に登録されている整数値境界節点変数の節点の個数を返す

- void **rcapGetBNodeVarInt** (const char dataname[80], size_t num, int32_t *nodeArray, int32_t *nodeVars)

Refiner に登録されている整数値境界節点変数を返す

- void **rcapAppendElementGroup** (const char dataname[80], size_t num, int32_t *elementArray)

ElementGroup とは、要素番号の集合のこと。

- size_t **rcapGetElementGroupCount** (const char dataname[80])

Refiner に登録されている要素グループの要素の個数を返す

- void **rcapGetElementGroup** (const char dataname[80], size_t num, int32_t *elementArray)

Refiner に登録されている要素グループを返す

- void **rcapAppendFaceGroup** (const char dataname[80], size_t num, int32_t *faceArray)

FaceGroup とは、要素番号、要素内面番号の組のこと。連成面を細分する場合などに用いる。

- size_t **rcapGetFaceGroupCount** (const char dataname[80])

Refiner に登録されている面グループの個数を返す

- void **rcapGetFaceGroup** (const char dataname[80], size_t num, int32_t *faceArray)

Refiner に登録されている面グループを返す

- void **rcapSetInterpolateMode** (const char mode[32])

NodeVariable を登録したときに、中間節点に与える値の決め方を選択します。 現在は "MIN" "MAX" "MIDDLE" の3種類に対応しています。 MIN は中間節点を生成するのに用いた節点上の値の最小値を与えます。 MAX は中間節点を生成するのに用いた節点上の値の最大値を与えます。 MIDDLE は中間節点を生成するのに用いた節点上の値の平均値を与えます。

- void **rcapGetInterpolateMode** (char mode[32])

NodeVariable を登録したときに、中間節点に与える値の決め方を返します。 戻り値は "MIN" "MAX" "MIDDLE" という文字列のいずれかです。

- int8_t **rcapGetOriginal** (int32_t localNodeId, int32_t *originalNodeArray)

中間節点から、それを生成するのに使った辺、面、要素の節点配列を返す

- int32_t **rcapGetMiddle** (int8_t etype, int32_t *originalNodeArray)

辺、面、要素を与えて、それから作られた中間節点を戻り値で返す

- void **rcapQualityReport** (const char name[80], const char *filename)

要素の品質に関する情報を出力する。