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_Refiner Ver. 1.0

チュートリアルガイド

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 チュートリアル概要

ここではインストールガイドに従って REVOCAP_Refiner のインストールが済んでいるとして、REVOCAP_Refiner をソルバに組み込む方法を説明する。

2 細分の手順

2.1 起動と終了

REVOCAP_Refiner の起動は rcapInitRefiner 関数で行い、終了は rcapTermRefiner 関数で行う。 REVOCAP_Refiner のすべての処理は rcapInitRefiner を読んだ後、rcapTermRefiner を呼ばれる前で有効である。 それ以外のところで REVOCAP_Refiner の処理を呼んでも何も処理を行わないか、エラーコードを返す。C 言語では次のようになる。

```
#include "rcapRefiner.h"
int main(void){
   rcapInitRefiner(0,0);
   /* ここで Refiner の処理を記述する */
   rcapTermRefiner();
   return 0;
}
```

Fortran90 言語では次のようになる。

```
program RefinerSample
   implicit none
   include "rcapRefiner.inc"
   call rcapInitRefiner(1,1)
   ! ここで Refiner の処理を記述する
   call rcapTermRefiner()
end program RefinerSample
```

rcapInitRefiner の引数は、呼び出し側のソルバの節点番号および要素番号の開始番号(nodeOffset,elementOffset)とする。一般に節点番号を配列の添え字にしている場合は C 言語の場合は 0 であり、Fortran90 言語の場合は 1 である。

以下のサンプルはすべて Fortran90 言語とする。

2.2 モデルの登録

REVOCAP_Refiner へは、節点データ、要素データ、境界条件データを登録することができる。 登録してあるデータは要素の細分の時に自動的に更新される。
CAD データから取り出した曲面情報を用いて形状補正を行う場合は、 細分する前のモデルのメッシュに対応した形状データと、 局所節点番号と大域節点番号の対応を与える必要がある。
対応しているモデルのデータは以下の通りである。

- 節点座標
- 四面体１次要素、２次要素
- 六面体１次要素、２次要素
- 三角柱１次要素、２次要素
- 四角錐１次要素、２次要素
- 三角形１次要素、２次要素
- 四角形１次要素、２次要素
- 線分１次要素、２次要素
- 節点グループ
- 要素グループ
- 面（要素番号と要素内面番号の組）グループ
- 境界節点グループ
- 境界節点整数値

節点と四面体１次要素を登録して細分するには次のようにする。

```
program RefinerSample
  implicit none
  include "rcapRefiner.inc"
  double precision :: coords(30)
  integer(kind=4), dimension(4) :: tetras
  integer(kind=4), dimension(32) :: refineTetras
  integer(kind=4) :: nodeCount
  integer(kind=4) :: elementCount
  coords = (/&
    & 0.0, 0.0, 0.0, &
    & 1.0, 0.0, 0.0, &
    & 0.0, 1.0, 0.0, &
    & 0.0, 0.0, 1.0, &
```

```
      & (0.0, I=13, 30) /)
    tetras = (/&
      & 1, 2, 3, 4, /)
    nodeCount = 4
    elementCount = 1
    call rcapInitRefiner(1,1)
    call rcapSetNode64( nodeCount, coords, 0, 0 )
    call rcapRefineElement( elementCount, RCAP_TETRAHEDRON, tetras,
refineTetras )
    call rcapCommit()
    call rcapTermRefiner()
 end program RefinerSample
```

Fortran90 では NULL アドレスの代入の代わりに -1 を代入することに注意する。

2.3 要素の細分（単一要素）

REVOCAP_Refiner で FEM モデルの要素の細分を行うには、rcapRefineElement 関数を使う。 この関数は細分したい要素の型（四面体１次要素など）と、要素の個数、要素の節点配列を与えて、 細分した要素の節点配列を取得する。 細分した要素の節点配列はあらかじめ呼び出し側でメモリ上に確保しておく必要がある。

細分後の節点配列の個数を取得するには､データ取得用の節点配列を NULL とするか、最初の節点番号に -1 を入れて呼び出せばよい。

rcapRefineElement は rcapTermRefiner を実行するまで何度も呼び出すことができる。

rcapRefineElement を呼び出す場合は、そこで細分をする要素の節点の座標を rcapSetNode32 または rcapSetNode64 であらかじめ登録しておく必要がある。 節点の座標は要素を細分するときに、中間節点の決定および細分のトポロジーを決定するために使われる。

最初に四面体からなる要素を細分し、次に その四面体の境界の三角形からなる要素について rcapRefineElement を呼び出した場合には、 三角形の細分による中間節点の節点番号は、四面体の細分の時に付与された中間節点の節点番号を使う（三角形の細分で新たに中間節点を生成しない）。

最初に四面体からなる要素について rcapRefineElement を呼び出して細分し、 さらにその結果である細分された四面体に対し、rcapRefineElement を呼び出した場合には、 2 段階の細分がなされる。ただし、境界条件を同時に更新する場合に 2 段階の細分を行う場合は、 １段階目の細分が終わったときに rcapCommit を実行する必要がある。詳細は次章を参照のこと。

2.4 要素の細分（混合要素）

rcapRefineElement 関数は要素の型、個数、節点配列を与えるため、要素の種類はすべて同じである必要がある。 混合要素の FEM モデルを細分するためには、次の 2 通りの方法がある。

- 呼び出し側で要素の型ごとにまとめて、rcapRefineElement を複数回呼ぶ
- rcapRefineElementMulti を使う

前者の場合は、既に説明してあるので、ここでは後者の rcaoRefineElementMulti 関数を使う場合について説明する。関数 rcapRefineElementMulti の引数と戻り値は、C 言語の場合は、

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

であり、Fortran90 言語の場合は、

```
function rcaprefineelementmulti(num, etypeArray, nodeArray, refinedNum,
resultEtypeArray, resultNodeArray )
  integer :: rcaprefineelementmulti
  integer, intent(in) :: num
  integer, intent(in), dimension(:) :: etypeArray
  integer, intent(in), dimension(:) :: nodeArray
  integer, intent(inout) :: refinedNum
  integer, intent(inout), dimension(:) :: resultEtypeArray
  integer, intent(inout), dimension(:) :: resultNodeArray
```

である。rcapRefineElementMulti を使う場合には、節点配列が複数の種類の要素を含むため、 何番目にどの種類の要素かを記述した配列を別に用意する必要がある。それを eTypeArray に格納して rcapRefineElementMulti を呼び出す。戻り値も複数の種類の要素が混在した節点配列となるため、 戻り値の要素の種類を格納する配列を与える必要がある。resultEtypeArray および resultNodeArray は呼び出し側が確保してから関数を呼び出さなければならない。

resultEtypeArray および resultNodeArray の配列の大きさを調べるには、resultNodeArray に C 言語なら NULL を、Fortran90 言語なら-1 を与えて呼び出すと、実際には細分を行わずに、refinedNum に細分結果の要素の個数、戻り値に細分結果を格納するための節点配列の大きさを返す（rcapRefineElement 関数と戻り値の意味が違うことに注意する）。実際に細分を行う時には、refinedNum の大きさの配列を resultEtypeArray に確保し、戻り値の大きさの配列を resultNodeArray に確保してから再度呼び出す。

REVOCAP_Refiner はこの関数の引数で与えられた配列を内部で参照しているので、rcapClearRefiner を呼ぶまでは、etypeArray、 nodeArray、resultEtypeArray、resultNodeArray、のメモリを解放してはならない。

2.5 境界条件を要素の細分時に同時に更新する

REVOCAP_Refiner では FEM モデルの要素の細分時に、境界条件を同時に更新することができる。 現在対応している境界条件は

- 節点グループ
- 要素グループ
- 面（要素番号と要素内面番号のペア）グループ
- 境界節点グループ
- 境界節点整数値グループ

である。

境界節点グループは、境界面にのみ定義されている節点グループで、更新後も 境界面にのみ存在することが保証される。 ただし、領域分割後の領域境界面に存在する場合を排除していない（領域境界面は与えないことになっている）ので、ソルバの呼び出し側で 領域境界面にある境界条件を削除する必要がある。

境界節点整数値グループは、境界面にのみ定義されている、節点上の整数値の分布を与えるものである。 FrontFlow/blue の境界条件を更新する場合に用いる。境界条件を整数値に割り当てて、境界条件を細分時に更新する。中間節点に割り当てられる整数値についてのルールは以下の通りである。

- もとの節点の中に変数が与えられていない節点があれば、中間節点には変数は与えない。
- もとの節点に変数が与えられていて、変数の値がすべて等しいときは、中間節点にその等しい値を与える。
- もとの節点に変数が与えられていて、変数の値が異なるときは、中間節点に最も小さい値を与える。

すなわち、小さい値が優先となっている。従って境界条件を整数値に割り当てるときには、中間節点に割り当てられる境界条件の優先順位の高いものを小さい値にする。

細分時に更新したい条件（節点グループなど）を登録するには、rcapRefineElement を呼び出す前に rcapAppendNodeGroup、 rcapAppendElementGroup、 rcapAppendFaceGroup、 rcapAppendBNodeGroup、 rcapAppendBNodeVarInt を呼び出す。登録する場合の識別子を変えることで複数の条件を登録することができる。 登録後に rcapRefineElement を実行すると、登録した節点グループ、要素グループ、面グループは内部で更新される。 rcapCommit を実行すると、rcapAppendBNodeGroup および rcapAppendBNodeVarInt で登録した境界条件が内部で更新される。 さらに

rcapCommit を実行後は細分後のデータの取り出しが可能になる。 細分後のデータを取り出すには、 rcapGetNodeGroup 、 rcapGetElementGroup 、 rcapGetFaceGroup 、 rcapGetBNodeGroup、rcapBNodeVarInt を使う。 データの個数（節点グループならば節点数）を取得するには rcapGetNodeGroupCount 、 rcapGetElementGroupCount 、 rcapGetFaceGroupCount、 rcapBnodeGroupCount、rcapGetBNodeVarIntCount で調べることができる。

REVOCAP_Refiner では細分対象の境界条件と、細分後の境界条件の両方を内部で保持している。 rcapGetNodeGroup などで取得できる境界条件は、細分対象の境界条件である。 rcapCommit では細分後の境界条件を細分対象の境界条件に置き換える。 すなわち rcapRefineElement を呼び出した後で、rcapCommit を呼び出すと、それまでの細分後の境界条件が、 細分対象の境界条件になるため、rcapGetNodeGroup などで細分後の境界条件を取得することができるようになる。 多段階の細分を行う場合は、それぞれの段階の細分が完了した時点で rcapCommit を実行することが必要である。

以下がその例である。

```
 program RefinerSample
！途中略
    integer(kind=4), dimension(3) :: ng0
    integer(kind=4), dimension(:), allocatable :: result_ng0
    character(80) :: str
    integer(kind=4) :: res
！途中略
    str = "const"//CHAR(0)
    ng0 = (/ 1, 2, 3 /)
    nodeCount = 3
    call rcapAppendNodeGroup(str,nodeCount,ng0)
    call rcapRefineElement( elementCount, RCAP_TETRAHEDRON, tetras,
refineTetras )
    call rcapCommit()
    res = rcapGetNodeGroupCount(str)
    allocate( result_ng0(res) )
    call rcapGetNodeGroup(str,res,result_ng0)
！途中略
    deallocate( result_ng0 )
    call rcapTermRefiner()
 end program RefinerSample
```

rcapRefineElement を呼び出したときに、その前に rcapAppendNodeGroup で登録された 節点グループを同時に更新する。 登録するときにつける識別子は C 言語との互換性のため CHAR(0) を最後につけて呼び出す必要がある。

2.6 要素の面や辺を細分する方法

要素の面や辺を細分する場合は次のようにする。 例えば四面体要素の場合、面を三角形要素、辺を線分要素とみなし、 rcapRefineElement を呼び出す。四面体要素を細分した後に三角形要素を細分した場合、 三角形要素の中間節点の節点番号は、四面体要素を細分したときに与えられた中間節点の節点番号が用いられる。 逆に面の三角形要素を細分し、その後で四面体要素を細分した場合は、すでに面が三角形要素で細分されている場合には その面上の中間節点の節点番号は、三角形要素を細分したときに与えられた中間節点の節点番号が用いられる。すなわち、どちらの場合も重複して中間節点が与えられることはない。

2.7 細分後の要素番号について

面グループや要素グループは要素番号を用いて表現されている。rcapRefineElement で要素を細分してこれらの境界条件を更新すると、それに付随する要素番号 も更新されるが、要素番号の付け方は以下の通りとする。

(1) 入力される要素番号は elementOffset から順に並んでいるものとする
(2) rcapRefineElement を 2 回呼ぶ場合は 1 回目の最後の要素番号の次から入力する要素番号が順に並んでいるものとする
(3) 出力される要素番号は rcapRefineElement を呼ぶ順に elementOffset から順に並んでいるものとする
(4) rcapRefineElement を 2 回呼ぶ場合は 1 回目の最後に出力した要素番号の次から出力する要素番号が順に並んでいるものとする
(5) 要素番号は rcapClearRefiner を呼ぶと elementOffset にリセットされる

例えば、elementOffset = 1 で rcapRefineElement を 100 個の四面体と 50 個の三角形について呼んだとする。

```
call rcapRefineElement( tetraCount, RCAP_TETRAHEDRON, tetras,
refineTetras )
call rcapRefineElement( triCount, RCAP_TRIANGLE, tris, refineTris )
```

この場合、細分前の要素番号は四面体が 1 から 100 まで、三角形が 101 から 150 まで与えられていると解釈し、 細分された四面体の要素番号は 1 から 800 まで、三角形の要素番号が 801 から 1000 まで与えられていると解釈する。 この要素番号に従って、要素グループと面グループの境界条件を更新する。

2.8　細分後に中間節点の節点番号を取得する

細分後の中間節点の節点番号は rcapRefineElement の結果から判断できるが、それ以外に Refiner に問い合わせることも可能である。 後から問い合わせる方法は、次の 2 通りある。

(1) 問い合わせ関数 rcapGetMiddle を呼び出す
(2) 要素細分関数 rcapRefineElement を両端の節点番号からなる線分要素について呼び出す

すでに細分されて中間節点が存在する場合は、両者の振る舞いは同じである。 細分されていなくて中間節点が存在しない場合は、前者は無効な節点番号 -1 を返すが、 後者は改めて細分を行い、中間節点を生成してその節点番号を返す。

2.9　細分後に中間節点から親節点（親要素）を取得する

細分後に rcapRefineElement の結果から中間節点から親節点（親要素）を知ることができるが、 その対応は REVOCAP_Refiner が覚えているので、呼び出し側で検索用のテーブルなどを用意しておく必要はない。rcapGetOriginal で中間節点を与えると、親節点（親要素）を取得することができる。 ここで親節点（親要素）と呼んでいるものは、以下のルールで与えられる。

- もとの要素の辺の中点に生成された中間節点の場合：両端の 2 点を親節点として返す
- 四角形の中心、または六面体の面の中心に生成された中間節点の場合：四角形を構成する 4 点（2 次要素なら 8 点）を親要素として返す。六面体の面の中心に生成された中間節点の場合は、ここで与えられる要素は細分で得られる要素（六面体）ではないことに注意する。
- 六面体の中心に生成された中間節点の場合：六面体を構成する 8 点（2 次要素の場合は 20 点）を親要素として返す。
- もとの要素の辺ではなく、細分によって生成された中間節点同士をつないでできて辺の中間節点の場合：両端の 2 点を親節点として返す。この場合は親節点として返す値は、細分前に存在する節点番号ではないことに注意する。

3　形状補正機能について

3.1　概要

REVOCAP_Refiner の形状補正機能では、CAD データからの形状データを与えることで、細分時に生成される中間節点を曲面上に自動的に適合させることができる。入力データとして形状データ、および細分前のメッシュ表面上の節点と CAD 曲面との対応のデータを与える必要がある。これらは同一の独自形式のファイルに記述される。入力データの詳細に

ついて以下で述べる。

3.2 曲面データ

形状補正機能が対応している曲面の種類は以下のとおりである。

- Bezier 曲面
- B-Spline 曲面
- NURBS 曲面

現在のところ、形状補正機能を有効にするためには、以下の制約がある。
円筒など周期的なパラメータ表現される曲面については、1 つの曲面で表現されていてはならない。例えば、円筒などは円方向に 2 分割して 2 つの曲面として表現されていなければならない。一般に曲面座標(u,v)の u または v の両端が同一の空間の点を表すような場合は対応していない。

3.3 入力データ

形状データ（曲面）と細分前のメッシュ表面上の節点との対応は、所定のフォーマットで 1 つのファイルにまとめて記述し、rcapSetCADFilename 関数で与える。フォーマットの詳細についてはマニュアルに記述されている。この入力データは REVOCAP_PrePost で作成することができる。このファイルで記述されている節点番号は領域分割前の大域的な節点番号であり、かつソルバの節点開始番号の如何に関わらず、0 から開始されている。
並列計算機環境で分割された領域ごとにモデルの細分を行う場合、分割領域ごとに細分を行うときの節点番号は局所節点番号で、曲面との対応は大域節点番号であるので、REVOCAP_Refiner に大域節点番号と、局所節点番号の対応を与える必要がある。この方法は 2 通りある。

- rcapSetNode32、rcapSetNode64 で座標を与えるときに、局所節点番号と大域節点番号の対応を同時に与える。
- rcapSetPartitionFilename でソルバが REVOCAP_Coupler に領域分割したデータを与えるためのファイルを指定する。そのファイルから REVOCAP_Refiner は大域節点番号と、局所節点番号の対応を取得する。

3.4 呼び出し時の注意

形状データと細分前の表面節点と曲面との対応を与える（すなわち rcapSetCADFilename を呼び出す）のは、REVOCAP_Refiner を初期化 rcapInitRefiner の後、大域節点番号と局所節点番号の対応を与える前に呼び出さなければならない。

3.5 形状関数による形状補正機能

2 次要素を細分する場合に、中間節点の位置をその要素の形状関数を使って 2 次曲線上に生

成することができる。この機能を有効にするには、rcapInitRefiner を実行した後で rcapSetSecondFitting(1)を呼び出す。無効にするには rcapSetSecondFitting(0)を呼び出す。初期状態は形状関数による補正は無効である。

4 サンプルプログラム

4.1 四面体を細分する

ソースコードにいくつかのサンプルプログラムが添付されている。ここでは、最も単純な場合である四面体 1 つのモデルを細分する C 言語のプログラムを挙げる。

```
/*
 *
 * サンプル実行例&テスト用プログラム
 * 四面体の細分チェック用
 *
 */

#ifdef _CONSOLE

#include "rcapRefiner.h"
#include "rcapRefinerMacros.h"
#include <stdio.h>
#include <stdlib.h>   /* for calloc, free */
#include <assert.h>

int main(void)
{
        /* 四面体を 1 つ並べる */
        float64_t coords[12] = {
                0.0, 0.0, 0.0,
                1.0, 0.0, 0.0,
                0.0, 1.0, 0.0,
                0.0, 0.0, 1.0,
        };
        size_t refineNodeCount = 0;
```

```
float64_t* resultCoords = NULL;
int32_t tetras[4] = {
        1, 2, 3, 4,
};
/* 細分後の四面体の節点配列：出力は 8 個 */
int32_t* refineTetras = NULL;
/* 細分する要素の型(定数値) */
int8_t etype = RCAP_TETRAHEDRON;
int32_t nodeOffset = 1;
int32_t elementOffset = 1;
/* 初期節点の個数 */
size_t nodeCount = 4;
/* 初期要素の個数 */
size_t elementCount = 1;
/* 細分後の要素の個数 */
size_t refineElementCount = 0;

/* 境界条件（節点グループ） */
int32_t ng0[3] = {1,2,3};
int32_t* result_ng0 = NULL;
size_t ng0Count = 3;

/* カウンタ */
int32_t i,j;

/* 節点番号のオフセット値を与える */
rcapInitRefiner( nodeOffset, elementOffset );
printf("----- Original Model -----¥n");
/* 座標値を Refiner に与える */
rcapSetNode64( nodeCount, coords, NULL, NULL );
/* 細分前の節点数 */
nodeCount = rcapGetNodeCount();
assert( nodeCount == 4 );
printf("Node：Count = %"PRIsz"¥n", nodeCount );
for(i=0;(size_t)i<nodeCount;++i){
        printf("%d：%f, %f, %f¥n", i+nodeOffset, coords[3*i], coords[3*i+1],
```

```
coords[3*i+2] );
	}
	/* 細分前の要素数 */
	assert( elementCount == 1 );
	printf("Element : Count = %"PRIsz"¥n", elementCount );
	for(i=0;(size_t)i<elementCount;++i){
		printf("%d : (%d) %d, %d, %d, %d¥n", i+elementOffset, etype,
			tetras[4*i], tetras[4*i+1], tetras[4*i+2], tetras[4*i+3]);
	}
	/* 節点グループの登録 */
	rcapAppendNodeGroup("ng0",ng0Count,ng0);
	ng0Count = rcapGetNodeGroupCount("ng0");
	assert( ng0Count == 3 );
	printf("Node Group : Count = %"PRIsz"¥n", ng0Count );
	for(i=0;(size_t)i<ng0Count;++i){
		printf("%d¥n", ng0[i]);
	}

	printf("----- Refined Model -----¥n");

	/* 要素の細分 */
	refineElementCount = rcapRefineElement( elementCount, etype, tetras,
NULL );
	refineTetras = (int32_t*)calloc( 4*refineElementCount, sizeof(int32_t) );
	elementCount = rcapRefineElement( elementCount, etype, tetras,
refineTetras);
	rcapCommit();

	/* 細分後の節点 */
	refineNodeCount = rcapGetNodeCount();
	printf("Node : Count = %"PRIsz"¥n", refineNodeCount );
	resultCoords = (float64_t*)calloc( 3*refineNodeCount, sizeof(float64_t) );
	rcapGetNodeSeq64( refineNodeCount, nodeOffset, resultCoords );
	for(j=0;(size_t)j<refineNodeCount;++j){
		printf("%d : %f, %f, %f¥n", j+nodeOffset, resultCoords[3*j],
resultCoords[3*j+1], resultCoords[3*j+2] );
```

```
        }
        free( resultCoords );

        /* 細分後の要素 */
        printf("Element : Count = %"PRIsz"¥n", refineElementCount );
        for(i=0;(size_t)i<refineElementCount;++i){
                printf("%d : (%d) %d, %d, %d, %d¥n", i+elementOffset, etype,
                        refineTetras[4*i],  refineTetras[4*i+1],  refineTetras[4*i+2],
refineTetras[4*i+3] );
        }

        /* 細分後の節点グループの更新 */
        ng0Count = rcapGetNodeGroupCount("ng0");
        result_ng0 = (int32_t*)calloc( ng0Count, sizeof(int32_t) );
        printf("Refined Node Group : Count = %"PRIsz"¥n", ng0Count );
        rcapGetNodeGroup("ng0",ng0Count,result_ng0);
        for(i=0;(size_t)i<ng0Count;++i){
                printf("%d¥n", result_ng0[i]);
        }
        free( result_ng0 );

        free( refineTetras );
        rcapTermRefiner();
        return 0;
}

#endif
```

4.2　六面体を細分する

```
/*
 *
 * サンプル実行例＆テスト用プログラム
 * 六面体の細分チェック用
 *
 */

#ifdef _CONSOLE

#include "rcapRefiner.h"
#include "rcapRefinerMacros.h"
#include <stdio.h>
#include <stdlib.h>    /* for calloc, free */
#include <assert.h>

int main(void)
{
        /* 六面体を 2 つ並べる  */
        float64_t coords[36] = {
                0.0, 0.0, -1.0,
                1.0, 0.0, -1.0,
                1.0, 1.0, -1.0,
                0.0, 1.0, -1.0,
                0.0, 0.0,  0.0,
                1.0, 0.0,  0.0,
                1.0, 1.0,  0.0,
                0.0, 1.0,  0.0,
                0.0, 0.0,  1.0,
                1.0, 0.0,  1.0,
                1.0, 1.0,  1.0,
                0.0, 1.0,  1.0,
        };
        size_t refineNodeCount = 0;
        float64_t* resultCoords = NULL;
        int32_t hexas[16] = {
```

```
            1, 2, 3, 4, 5, 6, 7, 8,
            5, 6, 7, 8, 9,10,11,12
};
/* 細分後の六面体の節点配列：出力は 2*8=16 個 */
int32_t* refineHexas = NULL;
/* 細分する要素の型(定数値) */
int8_t etype = RCAP_HEXAHEDRON;
int32_t nodeOffset = 1;
int32_t elementOffset = 1;
/* 初期節点の個数 */
size_t nodeCount = 12;
/* 初期要素の個数 */
size_t elementCount = 2;
/* 細分後の要素の個数 */
size_t refineElementCount = 0;

/* 境界条件（節点グループ） */
int32_t ng0[4] = {1,2,5,6};
int32_t* result_ng0 = NULL;
size_t ng0Count = 4;
/* 境界条件（境界節点グループ） */
int32_t bng0[3] = {5,6,7};
int32_t* result_bng0 = NULL;
size_t bng0Count = 3;

/* カウンタ */
int32_t i,j;

/* 節点番号のオフセット値を与える */
rcapInitRefiner( nodeOffset, elementOffset );
printf("----- Original Model -----¥n");
/* 座標値を Refiner に与える */
rcapSetNode64( nodeCount, coords, NULL, NULL );
/* 細分前の節点数 */
nodeCount = rcapGetNodeCount();
assert( nodeCount == 12 );
```

```
	printf("Node：Count = %"PRIsz"¥n", nodeCount );
	for(i=0;(size_t)i<nodeCount;++i){
		printf("%d：%f, %f, %f¥n", i+nodeOffset, coords[3*i], coords[3*i+1],
coords[3*i+2] );
	}
	/* 細分前の要素数 */
	assert( elementCount == 2 );
	printf("Element：Count = %"PRIsz"¥n", elementCount );
	for(i=0;(size_t)i<elementCount;++i){
		printf("%d ： (%d) %d, %d, %d, %d, %d, %d, %d, %d¥n",
i+elementOffset, etype,
			hexas[8*i], hexas[8*i+1], hexas[8*i+2], hexas[8*i+3],
			hexas[8*i+4], hexas[8*i+5], hexas[8*i+6], hexas[8*i+7] );
	}
	/* 節点グループの登録 */
	rcapAppendNodeGroup("ng0",ng0Count,ng0);
	ng0Count = rcapGetNodeGroupCount("ng0");
	assert( ng0Count == 4 );
	printf("Node Group：Count = %"PRIsz"¥n", ng0Count );
	for(i=0;(size_t)i<ng0Count;++i){
		printf("%d¥n", ng0[i]);
	}
	rcapAppendBNodeGroup("bng0",bng0Count,bng0);
	bng0Count = rcapGetBNodeGroupCount("bng0");
	assert( bng0Count == 3 );
	printf("Boundary Node Group：Count = %"PRIsz"¥n", bng0Count );
	for(i=0;(size_t)i<bng0Count;++i){
		printf("%d¥n", bng0[i]);
	}

	printf("----- Refined Model -----¥n");

	/* 要素の細分 */
	refineElementCount = rcapRefineElement( elementCount, etype, hexas,
NULL );
	refineHexas = (int32_t*)calloc( 8*refineElementCount, sizeof(int32_t) );
```

```
	elementCount = rcapRefineElement( elementCount, etype, hexas, refineHexas );
	rcapCommit();

	/* 細分後の節点 */
	refineNodeCount = rcapGetNodeCount();
	printf("Node：Count = %"PRIsz"¥n", refineNodeCount );
	resultCoords = (float64_t*)calloc( 3*refineNodeCount, sizeof(float64_t) );
	rcapGetNodeSeq64( refineNodeCount, nodeOffset, resultCoords );
	for(j=0;(size_t)j<refineNodeCount;++j){
		printf("%d : %f, %f, %f¥n", j+nodeOffset, resultCoords[3*j], resultCoords[3*j+1], resultCoords[3*j+2] );
	}
	free( resultCoords );

	/* 細分後の要素 */
	printf("Element：Count = %"PRIsz"¥n", refineElementCount );
	for(i=0;(size_t)i<refineElementCount;++i){
		printf("%d : (%d) %d, %d, %d, %d, %d, %d, %d, %d¥n", i+elementOffset, etype,
			refineHexas[8*i], refineHexas[8*i+1], refineHexas[8*i+2], refineHexas[8*i+3],
			refineHexas[8*i+4], refineHexas[8*i+5], refineHexas[8*i+6], refineHexas[8*i+7] );
	}

	/* 細分後の節点グループの更新 */
	ng0Count = rcapGetNodeGroupCount("ng0");
	result_ng0 = (int32_t*)calloc( ng0Count, sizeof(int32_t) );
	printf("Refined Node Group：Count = %"PRIsz"¥n", ng0Count );
	rcapGetNodeGroup("ng0",ng0Count,result_ng0);
	for(i=0;(size_t)i<ng0Count;++i){
		printf("%d¥n", result_ng0[i]);
	}
	free( result_ng0 );
```

```
	bng0Count = rcapGetBNodeGroupCount("bng0");
	result_bng0 = (int32_t*)calloc( bng0Count, sizeof(int32_t) );
	printf("Refined Boundary Node Group : Count = %"PRIsz"¥n", bng0Count );
	rcapGetBNodeGroup("bng0",bng0Count,result_bng0);
	for(i=0;(size_t)i<bng0Count;++i){
		printf("%d¥n", result_bng0[i]);
	}
	free( result_bng0 );

	free( refineHexas );

	rcapTermRefiner();
	return 0;
}

#endif
```

4.3　多段細分する

```
/*
 *
 *  ２段階実行サンプルプログラム
 *
 */

#ifdef _CONSOLE

#include "rcapRefiner.h"
#include "rcapRefinerMacros.h"
#include <stdio.h>
#include <stdlib.h>    /* for calloc, free */
#include <assert.h>

int main(void)
{
        /*
         * 使い方の例
         * 初めの 5 つは細分する前の節点座標
         */
        float64_t coords[15] = {
                0.0, 0.0, 0.0,
                1.0, 0.0, 0.0,
                0.0, 1.0, 0.0,
                0.0, 0.0, 1.0,
                1.0, 1.0, 1.0
        };
        /* 細分後の座標：必要に応じて calloc する */
        float64_t* resultCoords = NULL;
        /* 四面体の節点配列：入力は 2 個 */
        int32_t tetras[8] = {
                0, 1, 2, 3,
                1, 2, 3, 4
        };
        /* 細分後の四面体の節点配列：出力は 2*8=16 個*/
```

```
int32_t* refineTetras = NULL;
/*  ２段階細分後の四面体の節点配列：出力は 2*8*8=128 個*/
int32_t* refine2Tetras = NULL;
/*  細分する要素の型(定数値) */
int8_t etype = RCAP_TETRAHEDRON;
/*  メッシュの節点配列に現れる節点番号の最初の値  */
/* C  から呼ぶときは  0 fortran  から呼ぶ場合は  1 */
int32_t nodeOffset = 0;
int32_t elementOffset = 0;
/*  初期節点の個数  */
size_t nodeCount = 5;
/*  初期要素の個数  */
size_t elementCount = 2;
/*  細分後の要素の個数  */
size_t refineElementCount = 0;
/*  細分後の節点の個数  */
size_t refineNodeCount = 0;
/*  要素の細分と同時に更新する節点グループ  */
int32_t ng0[3] = {0,1,4};
size_t ngCount = 3;
int32_t* result_ng0 = NULL;
/*  要素の細分と同時に更新する面グループ  */
/*  要素番号と要素内面番号の順に交互に並べる  */
int32_t fg0[4] = {0,0,1,1};   /* [1,2,3] [1,4,3] */
size_t fgCount = 2;
int32_t* result_fg0 = NULL;

/*  ループのカウンタ  */
int32_t i = 0;
int32_t j = 0;

/*  節点番号のオフセット値を与える  */
rcapInitRefiner( nodeOffset, elementOffset );

printf("----- Original Model -----¥n");
/*
```

```
        * globalId と座標値を Refiner に教える
        * localIds は NULL をあたえると coords は nodeOffset から順番に並んでいるものと解釈する
        */
        rcapSetNode64( nodeCount, coords, NULL, NULL );
        /* 細分前の節点数 */
        nodeCount = rcapGetNodeCount();
        assert( nodeCount == 5 );
        printf("Node：Count = %"PRIsz"¥n", nodeCount );
        for(i=0;(size_t)i<nodeCount;++i){
                printf("%d：%f, %f, %f¥n", i+nodeOffset, coords[3*i], coords[3*i+1], coords[3*i+2] );
        }
        /* 細分前の要素数 */
        assert( elementCount == 2 );
        printf("Element：Count = %"PRIsz"¥n", elementCount );
        for(i=0;i<2;++i){
                printf("%d ：(%d) %d, %d, %d, %d¥n", i+elementOffset, etype, tetras[4*i], tetras[4*i+1], tetras[4*i+2], tetras[4*i+3] );
        }
        /* 節点グループの登録 */
        rcapAppendNodeGroup("innovate",ngCount,ng0);
        ngCount = rcapGetNodeGroupCount("innovate");
        assert( ngCount == 3 );
        printf("Node Group：Count = %"PRIsz"¥n", ngCount );
        for(i=0;i<3;++i){
                printf("%d¥n", ng0[i]);
        }
        /* 面グループの登録 */
        rcapAppendFaceGroup("revolute",fgCount,fg0);
        fgCount = rcapGetFaceGroupCount("revolute");
        assert( fgCount == 2 );
        printf("Face Group：Count = %"PRIsz"¥n", fgCount );
        for(i=0;i<2;++i){
                printf("%d, %d¥n", fg0[2*i], fg0[2*i+1]);
        }
```

```
	/*---------------------- REFINE STEP 1 ----------------------------------------*/

	/* 要素の細分 */
	refineElementCount = rcapRefineElement( elementCount, etype, tetras, NULL );
	assert( refineElementCount == 16 );
	refineTetras = (int32_t*)calloc( 4*refineElementCount, sizeof(int32_t) );
	refineElementCount = rcapRefineElement( elementCount, etype, tetras, refineTetras );
	assert( refineElementCount == 16 );
	rcapCommit();

	printf("----- Refined Model 1 -----¥n");

	/* 細分後の節点 */
	refineNodeCount = rcapGetNodeCount();
	printf("Node：Count = %"PRIsz"¥n", refineNodeCount );

	resultCoords = (float64_t*)calloc( 3*refineNodeCount, sizeof(float64_t) );
	rcapGetNodeSeq64( refineNodeCount, nodeOffset, resultCoords );
	for(j=0;(size_t)j<refineNodeCount;++j){
		printf("%d : %f, %f, %f¥n", j+nodeOffset, resultCoords[3*j], resultCoords[3*j+1], resultCoords[3*j+2] );
	}

	/* 細分後の要素 */
	printf("Element：Count = %"PRIsz"¥n", refineElementCount );
	for(i=0;(size_t)i<refineElementCount;++i){
		printf("%d : (%d) %d, %d, %d, %d¥n", i+elementOffset, etype, refineTetras[4*i], refineTetras[4*i+1], refineTetras[4*i+2], refineTetras[4*i+3] );
	}

	/* 細分後の節点グループの更新 */
	ngCount = rcapGetNodeGroupCount("innovate");
	printf("Node Group：Count = %"PRIsz"¥n", ngCount );
```

```
	assert( ngCount > 0 );
	result_ng0 = (int32_t*)calloc( ngCount, sizeof(int32_t) );
	rcapGetNodeGroup("innovate",ngCount,result_ng0);
	for(i=0;(size_t)i<ngCount;++i){
		printf("%d¥n", result_ng0[i]);
	}
	free( result_ng0 );
	result_ng0 = NULL;

	/* 細分後の面グループの更新 */
	fgCount = rcapGetFaceGroupCount("revolute");
	printf("Face Group：Count = %"PRIsz"¥n", fgCount );
	assert( fgCount > 0 );
	result_fg0 = (int32_t*)calloc( fgCount*2, sizeof(int32_t) );
	rcapGetFaceGroup("revolute",fgCount,result_fg0);
	assert( fgCount == 8 );
	for(i=0;(size_t)i<fgCount;++i){
		printf("%d, %d¥n", result_fg0[2*i], result_fg0[2*i+1]);
	}
	free( result_fg0 );
	result_fg0 = NULL;

	free( resultCoords );
	resultCoords = NULL;

	/* 第２段の細分の前にキャッシュをクリア */
	rcapClearRefiner();

	/*---------------------- REFINE STEP 2 ----------------------------------------*/

	/* 要素の細分 */
	elementCount = refineElementCount;
	refineElementCount = rcapRefineElement( elementCount, etype,
refineTetras, NULL );
	refine2Tetras = (int32_t*)calloc( 4*refineElementCount, sizeof(int32_t) );
	refineElementCount = rcapRefineElement( elementCount, etype,
```

```
refineTetras, refine2Tetras );
        rcapCommit();

        printf("----- Refined Model 2 -----¥n");

        /* 細分後の節点 */
        refineNodeCount = rcapGetNodeCount();
        printf("Node：Count = %"PRIsz"¥n", refineNodeCount );

        resultCoords = (float64_t*)calloc( 3*refineNodeCount, sizeof(float64_t) );
        rcapGetNodeSeq64( refineNodeCount, nodeOffset, resultCoords );
        for(j=0;(size_t)j<refineNodeCount;++j){
                printf("%d : %f, %f, %f¥n", j+nodeOffset, resultCoords[3*j],
resultCoords[3*j+1], resultCoords[3*j+2] );
        }

        /* 細分後の要素 */
        printf("Element：Count = %"PRIsz"¥n", refineElementCount );
        for(i=0;(size_t)i<refineElementCount;++i){
                printf("%d：(%d) %d, %d, %d, %d¥n", i+elementOffset, etype,
                        refine2Tetras[4*i], refine2Tetras[4*i+1],
refine2Tetras[4*i+2], refine2Tetras[4*i+3] );
        }

        /* 細分後の節点グループの更新 */
        ngCount = rcapGetNodeGroupCount("innovate");
        printf("Node Group：Count = %"PRIsz"¥n", ngCount );
        assert( ngCount > 0 );
        result_ng0 = (int32_t*)calloc( ngCount, sizeof(int32_t) );
        rcapGetNodeGroup("innovate",ngCount,result_ng0);
        for(i=0;(size_t)i<ngCount;++i){
                printf("%d¥n", result_ng0[i]);
        }
        free( result_ng0 );
        result_ng0 = NULL;
```

```
        /* 細分後の面グループの更新 */
        fgCount = rcapGetFaceGroupCount("revolute");
        printf("Face Group：Count = %"PRIsz"¥n", fgCount );
        assert( fgCount > 0 );
        result_fg0 = (int32_t*)calloc( fgCount*2, sizeof(int32_t) );
        rcapGetFaceGroup("revolute",fgCount,result_fg0);
        assert( fgCount == 8 );
        for(i=0;(size_t)i<fgCount;++i){
                printf("%d, %d¥n", result_fg0[2*i], result_fg0[2*i+1]);
        }
        free( result_fg0 );
        result_fg0 = NULL;

        free( resultCoords );
        resultCoords = NULL;

        free( refineTetras );
        refineTetras = NULL;
        free( refine2Tetras );
        refine2Tetras = NULL;

        rcapTermRefiner();
        return 0;
}

#endif
```

4.4　混合要素を細分する

```
/*
 *
 * サンプル実行例＆テスト用プログラム
 * 複数種類の要素細分チェック用
 *
 */

#ifdef _CONSOLE

#include "rcapRefiner.h"
#include "rcapRefinerMacros.h"
#include <stdio.h>
#include <stdlib.h>    /* for calloc, free */
#include <assert.h>

int main(void)
{
        /* 六面体の上に三角柱を乗せる */
        float64_t coords[30] = {
                0.0, 0.0, -1.0,
                1.0, 0.0, -1.0,
                1.0, 1.0, -1.0,
                0.0, 1.0, -1.0,
                0.0, 0.0,  0.0,
                1.0, 0.0,  0.0,
                1.0, 1.0,  0.0,
                0.0, 1.0,  0.0,
                0.5, 0.0,  1.0,
                0.5, 1.0,  1.0,
        };
        size_t refineNodeCount = 0;
        float64_t* resultCoords = NULL;
        int32_t hexas[8] = {
                1, 2, 3, 4, 5, 6, 7, 8,
        };
```

```
int32_t wedges[6] = {
        5, 9, 6, 8, 10, 7,
};
/* 細分後の六面体の節点配列：出力は 1*8=8 個 */
int32_t* refineHexas = NULL;
/* 細分後の三角柱の節点配列：出力は 1*8=8 個 */
int32_t* refineWedges = NULL;
/* 細分する要素の型(定数値) */
int8_t etype = RCAP_HEXAHEDRON;
int32_t nodeOffset = 1;
int32_t elementOffset = 1;
/* 初期節点の個数 */
size_t nodeCount = 10;
/* 初期要素の個数 */
size_t elementCount = 2;
/* 細分後の要素の個数 */
size_t refineHexaCount = 0;
size_t refineWedgeCount = 0;
size_t refineElementCount = 0;

/* 境界条件（節点グループ） */
int32_t ng0[5] = {1,2,5,6,9};
int32_t* result_ng0 = NULL;
size_t ng0Count = 5;

/* カウンタ */
int32_t i,j;

/* 節点番号のオフセット値を与える */
rcapInitRefiner( nodeOffset, elementOffset );
/* 座標値を Refiner に与える */
rcapSetNode64( nodeCount, coords, NULL, NULL );

printf("----- Original Model -----¥n");
/* 細分前の節点数 */
nodeCount = rcapGetNodeCount();
```

```
	assert( nodeCount == 10 );
	printf("Node：Count = %"PRIsz"¥n", nodeCount );
	for(i=0;(size_t)i<nodeCount;++i){
		printf("%d：%f, %f, %f¥n", i+nodeOffset, coords[3*i], coords[3*i+1],
coords[3*i+2] );
	}
	/* 細分前の要素数 */
	assert( elementCount == 2 );
	printf("Element：Count = %"PRIsz"¥n", elementCount );
	j = 0;
	etype = RCAP_HEXAHEDRON;
	elementCount = 1;
	for(i=0;(size_t)i<elementCount;++i){
		printf("%d ： (%d) %d, %d, %d, %d, %d, %d, %d, %d¥n",
j+elementOffset, etype,
			hexas[8*i], hexas[8*i+1], hexas[8*i+2], hexas[8*i+3],
			hexas[8*i+4], hexas[8*i+5], hexas[8*i+6], hexas[8*i+7] );
		j++;
	}
	etype = RCAP_WEDGE;
	elementCount = 1;
	for(i=0;(size_t)i<elementCount;++i){
		printf("%d ： (%d) %d, %d, %d, %d, %d, %d¥n", j+elementOffset,
etype,
			wedges[6*i], wedges[6*i+1], wedges[6*i+2],
			wedges[6*i+3], wedges[6*i+4], wedges[6*i+5] );
		j++;
	}

	/* 節点グループの登録 */
	rcapAppendNodeGroup("ng0",ng0Count,ng0);
	ng0Count = rcapGetNodeGroupCount("ng0");
	printf("Node Group：Count = %"PRIsz"¥n", ng0Count );
	assert( ng0Count == 5 );
	for(i=0;(size_t)i<ng0Count;++i){
		printf("%d¥n", ng0[i]);
```

```
	}

	printf("----- Refined Model -----¥n");
	/* 要素の細分 */
	etype = RCAP_HEXAHEDRON;
	elementCount = 1;
	refineHexaCount = rcapRefineElement( elementCount, etype, hexas, NULL );
	refineHexas = (int32_t*)calloc( 8*refineHexaCount, sizeof(int32_t) );
	etype = RCAP_WEDGE;
	elementCount = 1;
	refineWedgeCount = rcapRefineElement( elementCount, etype, wedges, NULL );
	refineWedges = (int32_t*)calloc( 6*refineWedgeCount, sizeof(int32_t) );
	refineElementCount = 0;
	etype = RCAP_HEXAHEDRON;
	elementCount = 1;
	refineHexaCount = rcapRefineElement( elementCount, etype, hexas, refineHexas );
	refineElementCount += refineHexaCount;
	etype = RCAP_WEDGE;
	elementCount = 1;
	refineWedgeCount = rcapRefineElement( elementCount, etype, wedges, refineWedges );
	refineElementCount += refineWedgeCount;
	rcapCommit();

	/* 細分後の節点 */
	refineNodeCount = rcapGetNodeCount();
	printf("Node：Count = %"PRIsz"¥n", refineNodeCount );
	resultCoords = (float64_t*)calloc( 3*refineNodeCount, sizeof(float64_t) );
	rcapGetNodeSeq64( refineNodeCount, nodeOffset, resultCoords );
	for(j=0;(size_t)j<refineNodeCount;++j){
		printf("%d : %f, %f, %f¥n", j+nodeOffset, resultCoords[3*j], resultCoords[3*j+1], resultCoords[3*j+2] );
	}
```

```
	free( resultCoords );

	/* 細分後の要素 */
	printf("Element : Count = %"PRIsz"¥n", refineElementCount );
	j = 0;
	etype = RCAP_HEXAHEDRON;
	for(i=0;(size_t)i<refineHexaCount;++i){
		printf("%d : (%d) %d, %d, %d, %d, %d, %d, %d, %d¥n",
j+elementOffset, etype,
			refineHexas[8*i], refineHexas[8*i+1], refineHexas[8*i+2],
refineHexas[8*i+3],
			refineHexas[8*i+4], refineHexas[8*i+5],
refineHexas[8*i+6], refineHexas[8*i+7] );
		j++;
	}
	etype = RCAP_WEDGE;
	for(i=0;(size_t)i<refineWedgeCount;++i){
		printf("%d : (%d) %d, %d, %d, %d, %d, %d¥n", j+elementOffset,
etype,
			refineWedges[6*i], refineWedges[6*i+1],
refineWedges[6*i+2],
			refineWedges[6*i+3], refineWedges[6*i+4],
refineWedges[6*i+5] );
		j++;
	}

	/* 細分後の節点グループ */
	ng0Count = rcapGetNodeGroupCount("ng0");
	result_ng0 = (int32_t*)calloc( ng0Count, sizeof(int32_t) );
	printf("Node Group : Count = %"PRIsz"¥n", ng0Count );
	rcapGetNodeGroup("ng0",ng0Count,result_ng0);
	for(j=0;(size_t)j<ng0Count;++j){
		printf("%d¥n", result_ng0[j]);
	}
	free( result_ng0 );
```

```
        free( refineHexas );
        free( refineWedges );

        rcapTermRefiner();
        return 0;
}

#endif
```

4.5　細分時に形状補正を行う

実行するディレクトリから相対パスでdata/column2/column2.rnfに形状データと曲面と表面節点の対応を記述したファイルを置くこと。開発者の便宜のため、Refiner/data/column2以下に columns2.rnf とこのサンプルと同じ六面体 2 つからメッシュを HECMW 形式（hec_column.msh）と FrontFlow/blue GF 形式（ffb_column.mesh）で同梱してあるので、適宜利用されたい。

```
/*
 *
 * 形状補正機能テスト用プログラムサンプル
 * 六面体２つからなる格子を円柱で形状補正する
 *
 * Sample Program for refinement with fitting to CAD surfaces.
 * This model consists two hexahedra with a column.
 *
 */

#ifdef _CONSOLE

#include "rcapRefiner.h"
#include "rcapRefinerMacros.h"
#include <stdio.h>
#include <stdlib.h>    /* for calloc, free */
#include <assert.h>

int main(void)
{
        int32_t nodeOffset = 1;
        int32_t elementOffset = 1;
        float64_t coords[36] = {
                0.0, 1.0, 0.0,
                0.0, 0.0, 1.0,
                0.0,-1.0, 0.0,
                0.0, 0.0,-1.0,
                1.0, 1.0, 0.0,
                1.0, 0.0, 1.0,
```

```
            1.0,-1.0, 0.0,
            1.0, 0.0,-1.0,
            2.0, 1.0, 0.0,
            2.0, 0.0, 1.0,
            2.0,-1.0, 0.0,
            2.0, 0.0,-1.0,
};
// CAD ファイルに記述してあるグローバル節点番号との対応
// nodeOffset = 1 を引いた値が CAD ファイルに記述してあることに注意
int32_t globalIds[12] = {
            101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112
};
int32_t localIds[12] = {
            1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
};
size_t refineNodeCount = 0;
float64_t* resultCoords = NULL;
int32_t hexas[16] = {
            1, 2, 3, 4, 5, 6, 7, 8,
            5, 6, 7, 8, 9, 10, 11, 12,
};
/* 細分後の要素の節点配列 */
int32_t* refineHexas = NULL;
int8_t etype = RCAP_HEXAHEDRON;
/* 初期節点の個数 */
size_t nodeCount = 12;
/* 初期要素の個数 */
size_t elementCount = 2;
/* 細分後の要素の個数 */
size_t refineElementCount = 0;

/* 境界条件（節点グループ） */
int32_t ng0[4] = {1,4,5,8};
int32_t* result_ng0 = NULL;
size_t ng0Count = 4;
```

```
	/* カウンタ */
	int32_t i,j;

	/* 節点番号のオフセット値を与える */
	rcapInitRefiner( nodeOffset, elementOffset );
	rcapSetCADFilename( "data/column2/column2.rnf" );

	printf("----- Original Model -----¥n");
	/* 座標値を Refiner に与える */
	/* 節点配列で与える局所節点番号と、CAD ファイルに記述してある大域節点番号との対応も与える */
	rcapSetNode64( nodeCount, coords, globalIds, localIds );
	/* 細分前の節点数 */
	nodeCount = rcapGetNodeCount();
	assert( nodeCount == 12 );
	printf("Node：Count = %"PRIsz"¥n", nodeCount );
	for(i=0;(size_t)i<nodeCount;++i){
		printf("%d：%f, %f, %f¥n", i+nodeOffset, coords[3*i], coords[3*i+1], coords[3*i+2] );
	}
	/* 細分前の要素数 */
	assert( elementCount == 2 );
	printf("Element：Count = %"PRIsz"¥n", elementCount );
	for(i=0;(size_t)i<elementCount;++i){
		printf("%d ： (%d) %d, %d, %d, %d, %d, %d, %d, %d¥n", i+elementOffset, etype,
			hexas[8*i], hexas[8*i+1], hexas[8*i+2], hexas[8*i+3],
			hexas[8*i+4], hexas[8*i+5], hexas[8*i+6], hexas[8*i+7]);
	}
	/* 節点グループの登録 */
	rcapAppendNodeGroup("ng0",ng0Count,ng0);
	ng0Count = rcapGetNodeGroupCount("ng0");
	assert( ng0Count == 4 );
	printf("Node Group：Count = %"PRIsz"¥n", ng0Count );
	for(i=0;(size_t)i<ng0Count;++i){
		printf("%d¥n", ng0[i]);
```

```
	}

	printf("----- Refined Model -----¥n");

	/* 要素の細分 */
	refineElementCount = rcapRefineElement( elementCount, etype, hexas, NULL );
	refineHexas = (int32_t*)calloc( 8*refineElementCount, sizeof(int32_t) );
	elementCount = rcapRefineElement( elementCount, etype, hexas, refineHexas);
	rcapCommit();

	/* 細分後の節点 */
	refineNodeCount = rcapGetNodeCount();
	printf("Node：Count = %"PRIsz"¥n", refineNodeCount );
	resultCoords = (float64_t*)calloc( 3*refineNodeCount, sizeof(float64_t) );
	rcapGetNodeSeq64( refineNodeCount, nodeOffset, resultCoords );
	for(j=0;(size_t)j<refineNodeCount;++j){
		printf("%d : %f, %f, %f¥n", j+nodeOffset, resultCoords[3*j], resultCoords[3*j+1], resultCoords[3*j+2] );
	}
	free( resultCoords );

	/* 細分後の要素 */
	printf("Element：Count = %"PRIsz"¥n", refineElementCount );
	for(i=0;(size_t)i<refineElementCount;++i){
		printf("%d : (%d) %d, %d, %d, %d, %d, %d, %d, %d¥n", i+elementOffset, etype,
			refineHexas[8*i], refineHexas[8*i+1], refineHexas[8*i+2], refineHexas[8*i+3],
			refineHexas[8*i+4], refineHexas[8*i+5], refineHexas[8*i+6], refineHexas[8*i+7] );
	}

	/* 細分後の節点グループの更新 */
	ng0Count = rcapGetNodeGroupCount("ng0");
```

```
        result_ng0 = (int32_t*)calloc( ng0Count, sizeof(int32_t) );
        printf("Refined Node Group : Count = %"PRIsz"¥n", ng0Count );
        rcapGetNodeGroup("ng0",ng0Count,result_ng0);
        for(i=0;(size_t)i<ng0Count;++i){
                printf("%d¥n", result_ng0[i]);
        }
        free( result_ng0 );

        free( refineHexas );
        rcapTermRefiner();
        return 0;
}

#endif
```

4.6　形状関数による形状補正を行う

```
/*
 *
 * 形状関数による四面体 2 次要素の細分テスト用プログラム
 *
 * Sample Program for refinement by shape functions.
 * This model consists of one element of second degree tetrahedron.
 *
 */

#ifdef _CONSOLE

#include "rcapRefiner.h"
#include "rcapRefinerMacros.h"
#include <stdio.h>
#include <stdlib.h>    /* for calloc, free */
#include <assert.h>

int main(void)
{
        int32_t nodeOffset = 0;
```

```
int32_t elementOffset = 0;
/* 初期節点 */
size_t nodeCount = 10;
float64_t coords[30] = {
        0.0, 0.0, 0.0, // 0
        1.0, 0.0, 0.0, // 1
        0.0, 1.0, 0.0, // 2
        0.0, 0.0, 1.0, // 3
        0.5, 0.5,-0.1, // 4
        0.0, 0.5,-0.1, // 5
        0.5, 0.0,-0.1, // 6
        0.0, 0.0, 0.5, // 7
        0.5, 0.0, 0.5, // 8
        0.0, 0.5, 0.5   // 9
};
/* 細分後の節点 */
size_t refineNodeCount = 0;
float64_t* resultCoords = NULL;
/* 細分前の要素 */
int8_t etype = RCAP_TETRAHEDRON2;
size_t elementCount = 1;
int32_t tetras[10] = {
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9
};
/* 細分後の要素 */
size_t refineElementCount = 0;
int32_t* refineTetras = NULL;

/* カウンタ */
int32_t i,j;

/* 初期化：節点番号、要素番号のオフセット値を与える */
rcapInitRefiner( nodeOffset, elementOffset );

/* 形状関数による中間節点を有効にする */
rcapSetSecondFitting(1);
```

```
	printf("----- Original Model -----¥n");
	/* 座標値を Refiner に与える */
	rcapSetNode64( nodeCount, coords, NULL, NULL );
	/* 細分前の節点数 */
	nodeCount = rcapGetNodeCount();
	assert( nodeCount == 10 );
	printf("Node：Count = %"PRIsz"¥n", nodeCount );
	for(i=0;(size_t)i<nodeCount;++i){
		printf("%d：%f, %f, %f¥n", i+nodeOffset, coords[3*i], coords[3*i+1],
coords[3*i+2] );
	}
	/* 細分前の要素数 */
	assert( elementCount == 1 );
	printf("Element：Count = %"PRIsz"¥n", elementCount );
	for(i=0;(size_t)i<elementCount;++i){
		printf("%d：(%d) %d, %d, %d, %d, %d, %d, %d, %d, %d, %d¥n",
i+elementOffset, etype,
			tetras[10*i], tetras[10*i+1], tetras[10*i+2],
tetras[10*i+3], tetras[10*i+4],
			tetras[10*i+5], tetras[10*i+6], tetras[10*i+7],
tetras[10*i+8], tetras[10*i+9]
			);
	}

	printf("----- Refined Model -----¥n");

	/* 要素の細分 */
	refineElementCount = rcapRefineElement( elementCount, etype, tetras,
NULL );
	refineTetras = (int32_t*)calloc( 10*refineElementCount, sizeof(int32_t) );
	elementCount = rcapRefineElement( elementCount, etype, tetras,
refineTetras);
	rcapCommit();

	/* 細分後の節点 */
```

```
        refineNodeCount = rcapGetNodeCount();
        printf("Node：Count = %"PRIsz"¥n", refineNodeCount );
        resultCoords = (float64_t*)calloc( 3*refineNodeCount, sizeof(float64_t) );
        rcapGetNodeSeq64( refineNodeCount, nodeOffset, resultCoords );
        for(j=0;(size_t)j<refineNodeCount;++j){
                printf("%d : %f, %f, %f¥n", j+nodeOffset, resultCoords[3*j],
resultCoords[3*j+1], resultCoords[3*j+2] );
        }
        free( resultCoords );

        /* 細分後の要素 */
        printf("Element：Count = %"PRIsz"¥n", refineElementCount );
        for(i=0;(size_t)i<refineElementCount;++i){
                printf("%d：(%d) %d, %d, %d, %d, %d, %d, %d, %d, %d, %d¥n",
i+elementOffset, etype,
                        refineTetras[10*i], refineTetras[10*i+1],
refineTetras[10*i+2], refineTetras[10*i+3], refineTetras[10*i+4],
                        refineTetras[10*i+5], refineTetras[10*i+6],
refineTetras[10*i+7], refineTetras[10*i+8], refineTetras[10*i+9]
                        );
        }
        free( refineTetras );

        rcapTermRefiner();
        return 0;
}

#endif
```

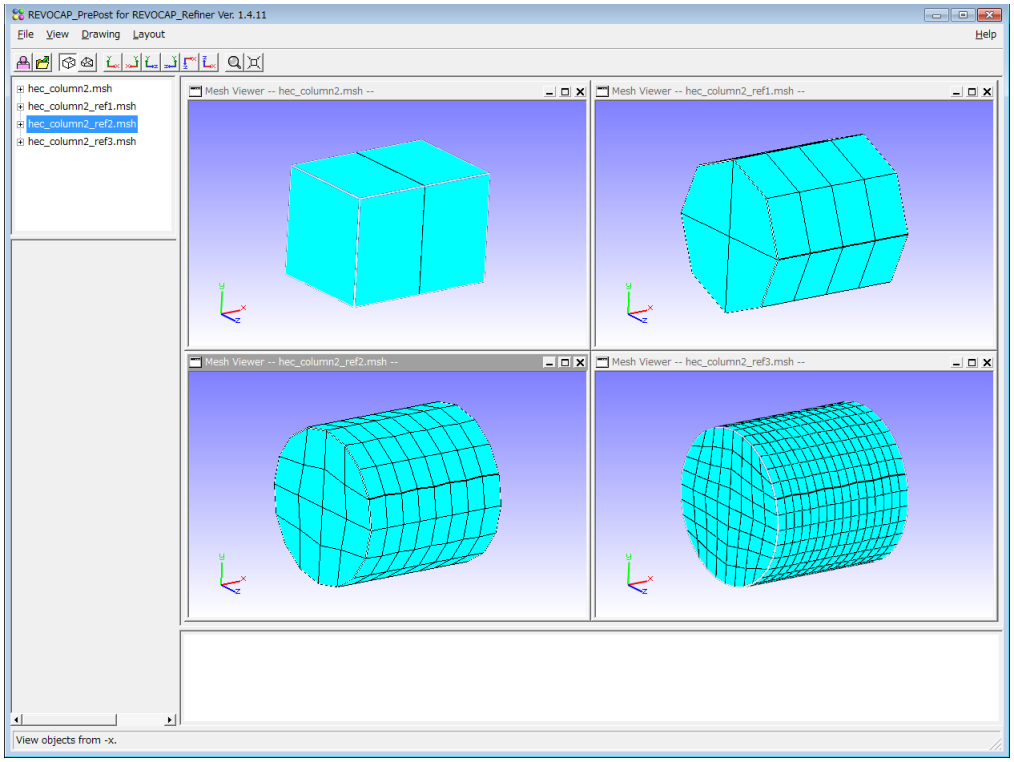
5 サンプルデータ

5.1 円柱

データは Refiner/data/column にある。

- hec_column.msh HEC_MW 形式の初期メッシュ

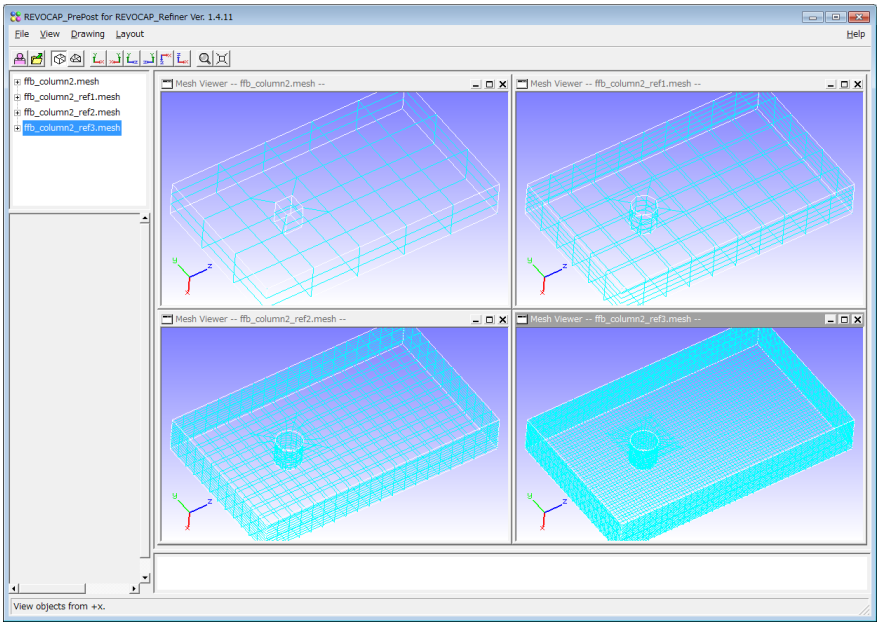
- ffb_column.mesh FrontFlow/blue GF 形式の初期メッシュ
- column.rnf 形状と初期適合データ
  領域分割後の局所節点番号のテストを兼ねて、初期適合データの節点番号は初期メッシュの節点番号から 100 ずれていることに注意する。
  形状は HEC_MW 形式、FrontFlow/blue GF 形式とも同一のものである。
  形状適合細分例は以下の通り。



5.2 円柱２

データは Refiner/data/column2 にある。

- hec_column2.msh HEC_MW 形式の初期メッシュ
- ffb_column2.mesh FrontFlow/blue GF 形式の初期メッシュ
- column2.rnf 形状と初期適合データ
  構造用（HEC_MW 形式）は円柱と同じで、流体用（FrontFlow/blue GF 形式）の初期メッシュを円柱の外部に取ったものである。
  形状適合細分例は以下の通り。



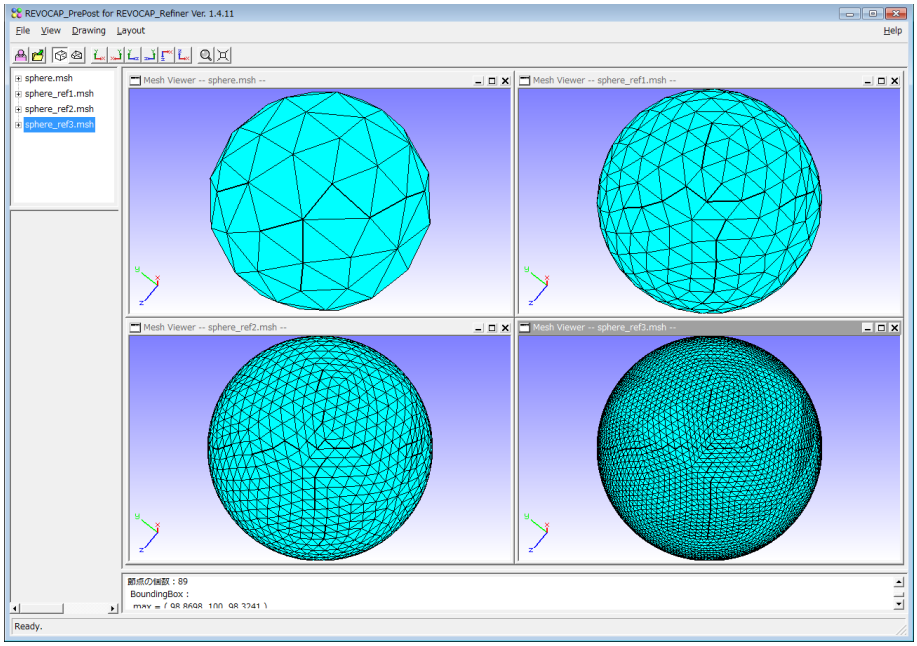
5.3 球面

データは Refiner/data/sphere にある。

- sphere.igs CAD 形状データ
- sphere.msh ADVENTURE_TetMesh 形式初期メッシュ
- sphere_fitting.rnf 形状と初期適合データ

  sphere.msh は ADVENTURE_TetMesh で生成されたメッシュであり、sphere_fitting.rnf は REVOCAP_PrePost で生成された初期適合データである。初期メッシュは形状適合の効果をみるためにあえて粗めに生成している（基準長さ=半径/2=50.0）。

  形状適合細分例は以下の通り。



5.4　ボトル形状

データは Refiner/data/bottle にある。

- bottle.mesh FrontFlow/blue GF 形式の初期メッシュ
- bottle_fitting.rnf 形状と初期適合データ

bottle.mesh は混合要素のテストのためのメッシュであり、四面体、六面体、三角柱、四角錐の各要素が混在している。

形状適合細分例は以下の通り。