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. 0.4

# チュートリアル



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

# 目次

# 1　チュートリアル概要

ここではプログラム説明書に従って REVOCAP_Refiner のインストールが済んでいるとして、REVOCAP_Refiner をソルバーに組み込む方法を説明する。

# 2　関数仕様

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

Fortran 言語では次のようになる。

```fortran
program RefinerSample
  implicit none
  integer, parameter :: RCAP_SIZE_T = 8
  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、節点番号および要素番号の開始番号(nodeOffset,elementOffset)とする。

以下ではすべてサンプルは Fortran 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生成を行った CAD 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。 現在は未対応である。
現在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４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
  integer, parameter :: RCAP_SIZE_T = 8
  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
```

Fortran 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（4 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、個数を返す。
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細分のトポロジーを判定するときの材料として使われる。

最初に４面体からなる要素を細分し、次に その４面体の境界の３角形からなる要素について rcapRefineElement を呼び出した場合には、 ３角形の細分による中間節点の節点番号は、４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。
最初に４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４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 を実行してください。詳細は次章に記述してある。

### 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

後者の rcapRefineElementMulti を使う場合は節点配列は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。それは eTypeArray に 格納して呼び出してください。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。
rcapRefineElement で配列を NULL または最初の節点番号を -1 として与えたときには、細分後の要素の個数を返したが、この関数では細分後の要素の個数は refinedNum で 与えられるため、戻り値は細分した結果を格納するのに必要な節点配列の大きさを返す。
一度 resultNodeArray を NULL または最初の成分を -1 として呼び出せば、 細分に必要な節点配列の大きさと、要素の個数が分かるので、 戻り値を格納する配列 resultEtypeArray と resultNodeArray の大きさを確認してから再度呼び出すことができる。

### 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ません （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、節点上の分布を与えるものである。 FrontFlow/blue の境界条件を更新するために、整数値で境界条件を更新するときに優先順位を 記述する。中間節点に与える整数値についてのルールは以下の通りである。

- もとの節点なか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。

rcapRefineElement を呼び出す前に rcapAppendNodeGroup rcapAppendElementGroup rcapAppendFaceGroup rcapAppendBNodeGroup rcapAppendBNodeVarInt で 細分時に更新したい条件を登録する。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もできる。 登録後に rcapRefineElement を実行すると、内部で登録した節点グループ、要素グループ、面グループは更新される。 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 では細分対象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 を実行後、rcapCommit を実行した後は、それまで細分後の境界条件

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 呼び出してください。

### 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 4 面体要素の場合、面を 3 角形要

素、辺を線分要素とみなし、 rcapRefineElement を呼び出す。4 面体要素を細分した後に３角形要素を細分した場合、 ３角形要素の中間節点の節点番号は、４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３角形要素を細分し、その後で４面体要素を細分した場合は、すでに面が３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、３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。

### 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。
入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
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
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４面体と 50 個の３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４面体が 1 から 100 まで、３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４面体の要素番号は 1 から 800 まで、３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によっても可能である。 後から問い合わせる方法は、次の 2 通りある。
問い合わせ関数 rcapGetMiddle を呼び出す
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（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を知ることができるが、 その対応は REVOCAP_Refiner 側で覚えているので､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
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された要素ではないことに注意してください。
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、細分前の節点番号ではないことに注意してください。

## 3 サンプルプログラム

### 3.1 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 4 面体 1 つのモデルを細分するプログラムを挙げる。

```
/*
 * gcc -D_CONSOLE mainTetra.c -L../lib/i486-linux/ -lstdc++ -lRcapRefiner
 *
 * 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 <stdio.h>
#include <stdlib.h>  /* for calloc, free */

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;
```

```
/* 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 ----------¥n");
/* 細分前の要素数 */
printf("Original Element Count = %d¥n", elementCount );
/* 節点グループの登録 */
rcapAppendNodeGroup("ng0",ng0Count,ng0);
ng0Count = rcapGetNodeGroupCount("ng0");
printf("Original Node Group Count %u¥n", ng0Count );
/* 細分前の節点数 */
nodeCount = rcapGetNodeCount();
printf("Original Node Count = %u¥n", nodeCount );

printf("---------- REFINE ----------¥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);
printf("Refined Element Count = %u¥n", elementCount );
rcapCommit();

/* 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 %u¥n", ng0Count );
rcapGetNodeGroup("ng0",ng0Count,result_ng0);
free( result_ng0 );

/* 細分後の節点 */
refineNodeCount = rcapGetNodeCount();
printf("Refined Node Count = %u¥n", refineNodeCount );
resultCoords = (float64_t*)calloc( 3*refineNodeCount, sizeof(float64_t) );
```

```
        rcapGetNodeSeq64( refineNodeCount, nodeOffset, resultCoords );
        free( resultCoords );

        free( refineTetras );
        rcapTermRefiner();
        return 0;
}
#endif
```

### 3.2　六面体を細分する

```
/*
 * gcc -D_CONSOLE mainHexa.c -L../lib/i486-linux/ -lstdc++ -lRcapRefiner
 *
 * 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 <stdio.h>
#include <stdlib.h>   /* for calloc, free */

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;
/* 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 ----------¥n");
/* 細分前の要素数 */
printf("Original Element Count = %d¥n", elementCount );
/* 節点グループの登録 */
rcapAppendNodeGroup("ng0",ng0Count,ng0);
ng0Count = rcapGetNodeGroupCount("ng0");
printf("Original Node Group Count %u¥n", ng0Count );
rcapAppendBNodeGroup("bng0",bng0Count,bng0);
bng0Count = rcapGetBNodeGroupCount("bng0");
printf("Original Boundary Node Group Count %u¥n", bng0Count );
/* 細分前の節点数 */
nodeCount = rcapGetNodeCount();
printf("Original Node Count = %u¥n", nodeCount );

printf("---------- REFINE ----------¥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 );
printf("Refined Element Count = %u¥n", elementCount );
rcapCommit();

/* 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 %u¥n", ng0Count );
rcapGetNodeGroup("ng0",ng0Count,result_ng0);
free( result_ng0 );

bng0Count = rcapGetBNodeGroupCount("bng0");
result_bng0 = (int32_t*)calloc( bng0Count, sizeof(int32_t) );
printf("Refined Boundary Node Group Count %u¥n", bng0Count );
rcapGetBNodeGroup("bng0",bng0Count,result_bng0);
```

```
        free( result_bng0 );

        /* 細分後の節点 */
        refineNodeCount = rcapGetNodeCount();
        printf("Refined Node Count = %u¥n", refineNodeCount );
        resultCoords = (float64_t*)calloc( 3*refineNodeCount, sizeof(float64_t) );
        rcapGetNodeSeq64( refineNodeCount, nodeOffset, resultCoords );
        free( resultCoords );

        free( refineHexas );

        rcapTermRefiner();
        return 0;
}

#endif
```

### 3.3 多段細分する

```
/*
 * gcc -D_CONSOLE mainMultiStage.c -L../lib/i486-linux/ -lstdc++ -lRcapRefiner
 *
 * 2段階実行サンプルプログラム
 *
 */

#ifdef _CONSOLE

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

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;

printf("---------- ORIGINAL DATA ----------¥n");
/* 節点番号のオフセット値を与える */
rcapInitRefiner( nodeOffset, elementOffset );
/*
 * globalId と座標値を Refiner に教える
 * localIds は NULL を与えると 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 );
/* 細分前の要素数 */
printf("Original Element Count = 2¥n" );
/* 節点グループの登録 */
rcapAppendNodeGroup("innovate",ngCount,ng0);
ngCount = rcapGetNodeGroupCount("innovate");
printf("Original Node Group Count %u¥n", ngCount );
/* 面グループの登録 */
rcapAppendFaceGroup("revolute",fgCount,fg0);
fgCount = rcapGetFaceGroupCount("revolute");
printf("Original Face Group Count %u¥n", fgCount );
/* 細分前の節点数 */
nodeCount = rcapGetNodeCount();
printf("Original Node Count = %u¥n", nodeCount );

printf("---------- REFINE STEP 1 ----------¥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 );
printf("Refined Element Count = %u¥n", elementCount );
```

```
rcapCommit();
/* 細分後の節点グループの更新 */
ngCount = rcapGetNodeGroupCount("innovate");
printf("Refined Node Group Count %u¥n", ngCount );
result_ng0 = (int32_t*)calloc( ngCount, sizeof(int32_t) );
rcapGetNodeGroup("innovate",ngCount,result_ng0);
/* ここでチェック */
free( result_ng0 );
result_ng0 = NULL;
/* 細分後の面グループの更新 */
fgCount = rcapGetFaceGroupCount("revolute");
printf("Refined Face Group Count %u¥n", fgCount );
result_fg0 = (int32_t*)calloc( 2*fgCount, sizeof(int32_t) );
rcapGetFaceGroup("revolute",fgCount,result_fg0);
/* ここでチェック */
free( result_fg0 );
result_fg0 = NULL;
/* 細分後の節点の個数 */
refineNodeCount = rcapGetNodeCount();
printf("Refined Node Count = %u¥n", refineNodeCount );
resultCoords = (float64_t*)calloc( 3*refineNodeCount, sizeof(float64_t) );
rcapGetNodeSeq64( refineNodeCount, 0, resultCoords );
free( resultCoords );
resultCoords = NULL;

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

printf("---------- REFINE STEP 2 ----------¥n");

/* 要素の細分 */
refineElementCount = rcapRefineElement( elementCount, etype, refineTetras, NULL );
refine2Tetras = (int32_t*)calloc( 4*refineElementCount, sizeof(int32_t) );
elementCount = rcapRefineElement( elementCount, etype, refineTetras, refine2Tetras );
printf("Refined Element Count = %u¥n", elementCount );
rcapCommit();
```

```
        /* 細分後の節点グループの更新 */
        ngCount = rcapGetNodeGroupCount("innovate");
        printf("Refined Node Group Count %u¥n", ngCount );
        result_ng0 = (int32_t*)calloc( ngCount, sizeof(int32_t) );
        rcapGetNodeGroup("innovate",ngCount,result_ng0);
        free( result_ng0 );
        result_ng0 = NULL;

        /* 細分後の面グループの更新 */
        fgCount = rcapGetFaceGroupCount("revolute");
        printf("Refined Face Group Count %u¥n", fgCount );
        result_fg0 = (int32_t*)calloc( 2*fgCount, sizeof(int32_t) );
        rcapGetFaceGroup("revolute",fgCount,result_fg0);
        free( result_fg0 );
        result_fg0 = NULL;

        /* 細分後の節点の個数 */
        refineNodeCount = rcapGetNodeCount();
        printf("Refined Node Count = %u¥n", refineNodeCount );

        resultCoords = (float64_t*)calloc( 3*refineNodeCount, sizeof(float64_t) );
        rcapGetNodeSeq64( refineNodeCount, 0, resultCoords );
        free( resultCoords );
        resultCoords = NULL;

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

        rcapTermRefiner();
        return 0;
}

#endif
```

## 3.4　混合要素を細分する

```
/*
 * gcc -D_CONSOLE mainMultiType.c -L../lib/i486-linux/ -lstdc++ -lRcapRefiner
 *
 * 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（要素タイプ別に）
 *
 */

#ifdef _CONSOLE

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

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 nodes[14] = {
                1, 2, 3, 4, 5, 6, 7, 8,
```

```
            5,  9,  6,  8,  10,  7,
};
/* 細分後の節点配列 : 出力は 8*8 + 8*6 = 112 個 */
int32_t* refineNodes = NULL;
/* 細分する要素の型(定数値) */
int8_t etypes[2] = {
            RCAP_HEXAHEDRON,
            RCAP_WEDGE,
};
int8_t* resultEtypes = NULL;
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 refineElementCount = 0;
/* 細分後の節点配列の大きさ */
size_t refineNodesArraySize = 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;

/* 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 ----------¥n");
/* 細分前の要素数 */
```

```
	printf("Original Element Count = %d¥n", elementCount );
	/* 節点グループの登録 */
	rcapAppendNodeGroup("ng0",ng0Count,ng0);
	ng0Count = rcapGetNodeGroupCount("ng0");
	printf("Original Node Group Count %u¥n", ng0Count );
	/* 細分前の節点数 */
	nodeCount = rcapGetNodeCount();
	printf("Original Node Count = %u¥n", nodeCount );

	printf("---------- REFINE ----------¥n");
	/* 要素の細分 */
	refineNodesArraySize = rcapRefineElementMulti( elementCount, etypes, nodes, &refine
ElementCount, NULL, NULL );
	if( refineNodesArraySize == 0 ){
		rcapTermRefiner();
		return 0;
	}
	printf("Refined Element Size = %u¥n", refineElementCount);
	printf("Refined Node Array Size = %u¥n", refineNodesArraySize);
	refineNodes = (int32_t*)calloc( refineNodesArraySize, sizeof(int32_t) );
	/* 要素の型も受け取る場合 */
	resultEtypes = (int8_t*)calloc( refineElementCount, sizeof(int8_t) );
	refineElementCount = rcapRefineElementMulti( elementCount, etypes, nodes, &refineEl
ementCount, resultEtypes, refineNodes );
	printf("Refined Element Count = %u¥n", refineElementCount );
	printf("Refined types¥n");
	for(i=0;i<refineElementCount;++i){
		printf("Refined Element type %d = %d¥n", i, resultEtypes[i] );
	}
	free( resultEtypes );
/*
	refineElementCount = rcapRefineElementMulti( elementCount, etypes, nodes, &refineEl
ementCount, NULL, refineNodes );
	printf("Refined Element Count = %u¥n", refineElementCount );
*/
	rcapCommit();
```

```
/* 細分後の要素 */
printf("HEXA¥n");
for(i=0;i<8;++i){
        printf("%d %d %d %d %d %d %d %d¥n",
                refineNodes[8*i],
                refineNodes[8*i+1],
                refineNodes[8*i+2],
                refineNodes[8*i+3],
                refineNodes[8*i+4],
                refineNodes[8*i+5],
                refineNodes[8*i+6],
                refineNodes[8*i+7]);
}
printf("WEDGE¥n");
for(i=0;i<8;++i){
        printf("%d %d %d %d %d %d¥n",
                refineNodes[64+6*i],
                refineNodes[64+6*i+1],
                refineNodes[64+6*i+2],
                refineNodes[64+6*i+3],
                refineNodes[64+6*i+4],
                refineNodes[64+6*i+5]);
}

/* 細分後の節点グループ */
ng0Count = rcapGetNodeGroupCount("ng0");
result_ng0 = (int32_t*)calloc( ng0Count, sizeof(int32_t) );
printf("Refined Node Group Count %u¥n", ng0Count );
rcapGetNodeGroup("ng0",ng0Count,result_ng0);
free( result_ng0 );

/* 細分後の節点 */
refineNodeCount = rcapGetNodeCount();
printf("Refined Node Count = %u¥n", refineNodeCount );
resultCoords = (float64_t*)calloc( 3*refineNodeCount, sizeof(float64_t) );
```

```
        rcapGetNodeSeq64( refineNodeCount, nodeOffset, resultCoords );
        free( resultCoords );

        free( refineNodes );

        rcapTermRefiner();
        return 0;
}

#endif
```