独立行政法人 理化学研究所 御中

カットライブラリ整備

ユーザ利用マニュアル

Ver. 2.0.0

2010年11月

(株) 情報数理研究所

目次

初版 version 1.0.0 28 Apr. 2010
version 2.0.0 10 Nov. 2010

1　はじめに

本 Cutlib ライブラリは，等間隔直交格子からなる 3 次元直方体領域に存在するポリゴン群の交点情報を計算する機能を提供します．本ライブラリは，理化学研究所 VCAD 機能情報シミュレーションチームで開発中の物理現象シミュレーションのためのフレームワーク V–Sphere 内で使用することを前提としています．

本ドキュメントの構成は以下のとおりです．この節の残りの部分で，本ライブラリで用いる用語の定義，そして本ライブラリの提供する機能の概略を述べます．続く 2 節で，基本的なデータ構造について説明します．3 節では，交点情報を計算する関数群のインタフェースについて説明します．最後の 4 節で，交点情報を計算する関数群の使用例を示します．

1.1　用語の定義

■計算基準点，計算基準線分　隣接するセル中心を結ぶ線分，または，隣接するノード間を結ぶ線分を計算基準線分，その両端の点を計算基準点と呼ぶことにします．本ライブラリでは，計算基準点毎に，6 方向 $(\pm x, \pm y, \pm z)$ の計算基準線分を横切るポリゴンの存在を調べます．そしてポリゴンが存在したなら，各方向毎に，最も計算基準点に近い交点を持つポリゴンについての情報 (交点情報) を記録します．したがって一般には，計算基準線分の両端の基準点では，違う交点情報が記録されることになります．各計算基準点毎に，交差したポリゴンが存在しなかったならその情報も含めて，6 組の交点情報が記録されます．

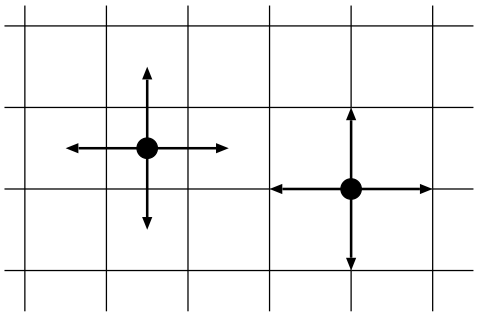

図 1　計算基準点 (黒丸) と計算基準線分 (矢印)，左:セル中心間，右:ノード間

■交点情報，交点座標値，境界 ID　交点情報としては，交点座標と境界 ID を記録します．交点座標値は，本ライブラリでは，計算基準点から最短ポリゴン交点までの距離を計算基準線分長で規格化した値として定義します．特に計算基準線分上にポリゴン交点が存在しなかった場合には，交点座標値を 1.0 と記録することとします．境界 ID は，複数のポリゴン集合 (ポリゴングループ) を識別する ID で，1〜255 の整数をとります．計算基準線分上にポリゴン交点が存在しなかった場合には，境界 ID として 0 を記録することとします．

■Octree　本ライブラリは，Octree による木構造セルの交点情報計算にも対応します．対象となるデータ構造は，直交等間隔分割されたセルをルートとして (ルートセル)，各ルートセルに木構造を持たせたものです．各ルートセルを 8 分割して，それを子セルとします．各子セルに対して，さ

らにそれを 8 分割して子セルの子セルとします．この操作を再帰的に繰り返し，木構造を構築していきます．分割を停止した，すなわち子セルを持たないセルを，リーフセルと呼びます．

1.2 本ライブラリの担当範囲

ポリゴン交点情報を実際に計算するには，以下の手順が必要です．

1. Polylib クラス (並列プログラムの場合は MPIPolylib クラス) のインスタンスを生成
2. 交点計算の対象となるポリゴングループを Polylib のグループ管理ツリーに登録
3. 対象ポリゴンデータの読み込み，kd ツリーの作成
4. ポリゴン交点情報の計算

これらの手順の内，本ライブラリは 4 の部分のみを担当するものとします．本ライブラリが提供するポリゴン交点情報計算関数は，上記 1〜3 までの操作を終了した Polylib クラスのポインタを引数として受け取ります．

Octree データ構造では，SklTree クラスにより既にツリー構造が構築済みであるものとします．

本ライブラリ自身は並列化されていません．ただし，交点情報計算関数の引数として，Polylib クラスではなく，MPIPolylib クラスへのポインタを受け取ることにより，領域分割並列計算に対応できます．その場合には，交点計算の対象として，並列プロセス毎の分割済み空間情報 (セル・ノード情報，Octree 情報) を交点情報計算関数に渡す必要があります．

1.3 本ライブラリの提供機能

■交点情報計算関数 セル中心間版，ノード間版，Octree 版の各関数を提供します．Octree 版では，計算基準点として，リーフセルのみとするか，全ツリー階層の全セルとするかを選択できます．

■交点情報格納用データ構造 メモリ削減のために，交点座標値を 8 ビット圧縮して格納することができます．また，境界 ID の最大値が 31 以下の場合には，それを利用して境界 ID も圧縮格納することができます．セル中心間版およびノード間版では，得られた交点情報は，圧縮の有無にかかわらず，Fortran サブルーチンからも読み出し可能な形式で 1 次元配列に格納されます．

■配列ラッパクラス セル中心間版およびノード間版では，交点情報格納用配列を直に扱う必要のない，配列ラッパクラスによるインタフェースも利用可能です．

■Octree セルデータへのアクセッサクラス Octree 版では，SklCell クラス内のデータ領域へのアクセスを容易にするため，アクセッサ提供クラスを用意しました．

1.4 ヘッダファイルおよび名前空間

本ライブラリの各機能を利用するには，ヘッダファイル Cutlib.h をインクルードする必要があります．

また，本ライブラリが提供する関数，クラス，定数は，全て名前空間「cutlib」内で定義されています．本ドキュメントの以降の記述では，スコープ解決演算子「`cutlib::`」を省略しています．実際の利用時には，using 文で cutlib 名前空間の使用を宣言するか，各キーワードの前に明示的に

スコープ解決演算子をつける必要があります.

2　交点情報格納用データ構造

この節では，本ライブラリの交点情報計算関数を使用するにあたり理解しておく必要がある基本的なデータ構造，**基本データ型**，**配列ラッパクラス**，**セルデータへのアクセッサクラス**，について説明します．

基本データ型　計算基準点における 6 方向分の交点情報 (交点座標，境界 ID) を格納するための型

配列ラッパクラス　セル中心間版およびノード間版交点情報計算関数で使用する，基本データ型配列を内部に持つクラス

セルデータへのアクセッサクラス　Octree 版で使用する，各セルデータ内の交点情報格納領域へのアクセッサを提供するクラス

なお，単独の境界 ID を扱う型として，次のように BidType 型を定義しています．

```
typedef unsigned char BidType;
```

また，交点探査方向には，以下の順に 0〜5 の整数値が割り当てられています．

$-x,\ +x,\ -y,\ +y,\ -z,\ +z$

2.1　基本データ型

基本データ型は，計算基準点における 6 方向分の交点情報を格納するための型です．これらの型は、Fortran サブルーチンからも読み取り可能なように設計されています。交点座標用，境界 ID 用に，それぞれ 2 種類の型があります。

■交点座標基本データ型

CutPos32 **型**　交点座標を float(32 ビット) として格納 (図 2)

```
typedef float CutPos32[6];
```

CutPos8 **型**　交点座標を 8 ビット量子化し (値は 0〜255)，3 つずつ，2 つの 32 ビット整数に格納 (図 3)

```
typedef int32_t CutPos8[2];
```

■境界 ID 基本データ型

CutBid8 **型**　0〜255 の境界 ID(8 ビット) を 3 つずつ，2 つの 32 ビット整数に格納 (図 4)

```
typedef int32_t CutBid8[2];
```

CutBid5 **型**　0〜31 の境界 ID(5 ビット) を 6 つまとめて，1 つの 32 ビット整数に格納 (図 5)

```
typedef int32_t CutBid5;
```

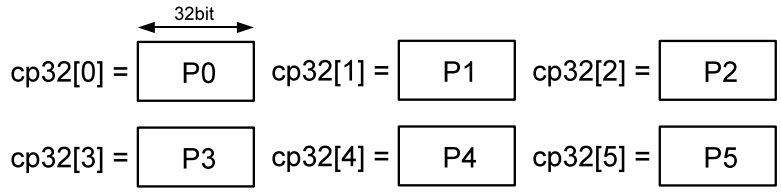

図 2　CutPos32 型変数 `cp32` への交点座標 P0〜P5(32 ビット浮動小数点数) の格納

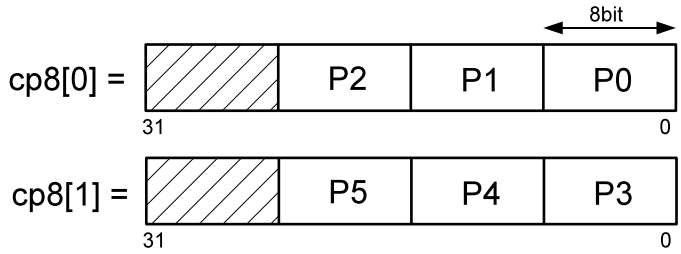

図 3　CutPos8 型変数 `cp8` への交点座標 P0〜P5(8 ビット符号無し整数) の格納

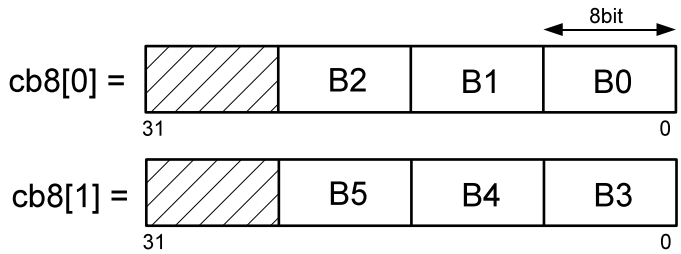

図 4　CutBid8 型変数 `cb8` への境界 ID B0〜B5(8 ビット符号無し整数) の格納

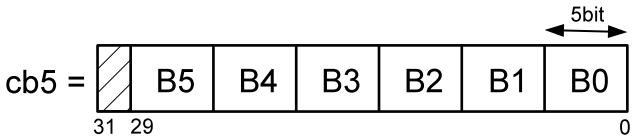

図 5　CutBid5 型変数 `cb5` への境界 ID B0〜B5(5 ビット符号無し整数) の格納

本ドキュメントの以降では，交点座標基本データ型の総称として **CutPos 型**，境界 ID 基本データ型の総称として **CutBid 型**，という表記を用いることにします．

セル中心間版およびノード間版の交点情報計算関数では，得られた交点情報を，交点座標基本データ型と境界 ID 基本データ型の 2 つの 1 次元配列に格納します．計算基準点の個数を $N_x \times N_y \times N_z$ とすると，計算基準点の位置 (i, j, k) と 1 次元配列のインデックス ijk の間に以下の関係があります．

$$ijk = i + j \times N_x + k \times N_x \times N_y$$

それぞれの型およびその 1 次元配列について，以下のようなアクセス用関数を用意しました[*1]．

2.1.1 方向別アクセス関数

```
float GetCutPos(const CutPos& cp, int d)
float GetCutPos(const CutPos cp[], size_t ijk, int d)
BidType GetCutBid(const CutBid& cb, int d)
BidType GetCutBid(const CutBid cb[], size_t ijk, int d)
```

■引数 (IN)

`CutPos& cp`	CutPos32 型または CutPos8 型
`CutBid& cb`	CutBid8 型または CutBid5 型
`CutPos cp[]`	CutPos32 型または CutPos8 型の配列
`CutBid cb[]`	CutBid8 型または CutBid5 型の配列
`int d`	交点探査方向 (0〜5)
`size_t ijk`	1 次元配列インデックス

2.1.2 6 方向同時アクセス関数

```
void GetCutPos(const CutPos& cp, float pos[])
void GetCutPos(const CutPos cp[], size_t ijk, float pos[])
void GetCutBid(const CutBid& cb, BidType bid[])
void GetCutBid(const CutBid cb[], size_t ijk, BidType bid[])
```

■引数 (IN)

`CutPos& cp`	CutPos32 型または CutPos8 型
`CutBid& cb`	CutBid8 型または CutBid5 型
`CutPos cp[]`	CutPos32 型または CutPos8 型の配列
`CutBid cb[]`	CutBid8 型または CutBid5 型の配列
`size_t ijk`	1 次元配列インデックス

■引数 (OUT)

`float pos[6]`	交点座標
`BidType bid[6]`	境界 ID

2.2 配列ラッパクラス

セル中心間版およびノード間版交点情報計算関数では，基本データ型配列によるインタフェースの他に，その配列のラッパクラスによるインタフェースも提供しています．

[*1] これらの関数はインライン関数として定義されています．

CutPos32Array　CutPos32 型配列のラッパクラス
CutPos8Array　CutPos8 型配列のラッパクラス
CutBid8Array　CutBid8 型配列のラッパクラス
CutBid5Array　CutBid5 型配列のラッパクラス

ともに，抽象クラス CutPosArray および CutBidArray を実装したものです[*2](図 6)．それぞれのクラスは，対応する基本データ型の 1 次元配列をメンバに持っています．また，その配列を Fortran ルーチンに渡せるように，配列へのポインタを得るためのメソッドを備えています．

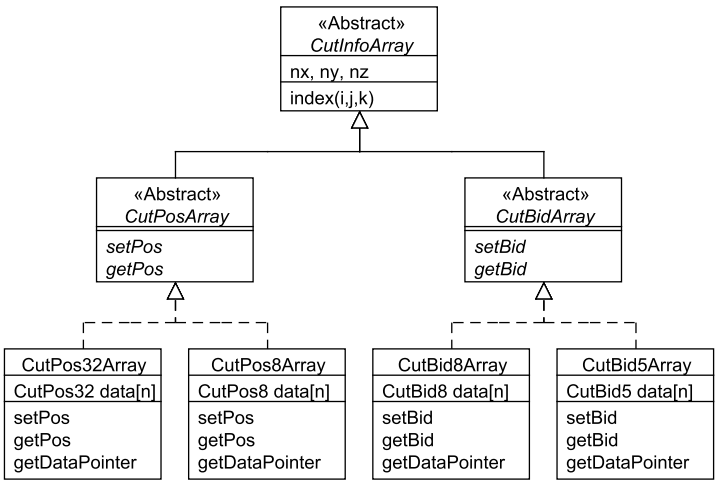

図 6　基本データ型配列のラッパクラス

以下の各メソッドの説明では, 個々のクラス名の代りに, 抽象クラス CutPosArray, CutBidArray を用いています．実際の使用時には，`CutPosArray` → `CutPos32Array` 等，適宜置き換えてください．

2.2.1 コンストラクタ

```
CutPosArray::CutPosArray(size_t nx, size_t ny, size_t nz)
CutPosArray::CutPosArray(const size_t ndim[])
CutBidArray::CutBidArray(size_t nx, size_t ny, size_t nz)
CutBidArray::CutBidArray(const size_t ndim[])
```

■引数 (IN)

`size_t nx, ny, nz`	セル数/ノード数
`size_t ndim[3]`	セル数/ノード数

[*2] 実際には，基本データ型をパラメータに持つテンプレートクラスを通して定義されており，それぞれ，CutPosArrayTemplate<CutPos32>, CutPosArrayTemplate<CutPos8>, CutBidArrayTemplate<CutBid8>, CutBidArrayTemplate<CutBid5> を typedef したものになっています．

2.2.2　データ配列ポインタ取得メソッド

```
CutPos* CutPosArray::getDataPointer()
CutBid* CutBidArray::getDataPointer()
```

■ノート　返り値は，対応する交点情報基本データ型配列へのポインタ．デストラクタが呼ばれると，その配列領域も開放されてしまうので注意すること．

2.2.3　方向別アクセスメソッド

```
float CutPosArray::getPos(size_t i, size_t j, size_t k, int d)
float CutPosArray::getPos(size_t ijk, int d)
BidType CutBidArray::getBid(size_t i, size_t j, size_t k, int d)
BidType CutBidArray::getBid(size_t ijk, int d)
```

■引数 (IN)

`size_t i, j, k`	3 次元配列インデックス
`size_t ijk`	1 次元配列インデックス
`int d`	交点探査方向 (0〜5)

2.2.4　6 方向同時アクセスメソッド

```
void CutPosArray::getPos(size_t i, size_t j, size_t k, float pos[])
void CutPosArray::getPos(size_t ijk, float pos[])
void CutBidArray::getBid(size_t i, size_t j, size_t k, BidType bid[])
void CutBidArray::getBid(size_t ijk, BidType bid[])
```

■引数 (IN)

`size_t i, j, k`	3 次元配列インデックス
`size_t ijk`	1 次元配列インデックス

■引数 (OUT)

`float pos[6]`	交点座標
`BidType bid[6]`	境界 ID

2.2.5　配列サイズ関連メソッド

```
size_t CutPosArray::getSizeX()
size_t CutPosArray::getSizeY()
size_t CutPosArray::getSizeZ()
```

```
size_t CutPosArray::index(size_t i, size_t j, size_t k)

size_t CutBidArray::getSIzeX()
size_t CutBidArray::getSIzeX()
size_t CutBidArray::getSIzeX()
size_t CutBidArray::index(size_t i, size_t j, size_t k)
```

■ノート　getSizeX, getSizeY, getSizeZ の各メソッドは，内部の基本データ型配列の 3 次元でのサイズを返す．index メソッドは，3 次元インデックスを 1 次元インデックスに変換する．

2.3　セルデータへのアクセッサクラス

Octree ソルバでは，セルに付随するデータは，SklCell クラス内のデータ領域に格納します．交点情報の SklCell クラス内のデータ領域への格納，および，格納された交点情報の取得を容易にするために，アクサッサ提供クラスを用意しました．

CutPos32Octree　CutPos32 型データ用アクセッサ提供クラス
CutPos8Octree　CutPos8 型データ用アクセッサ提供クラス
CutBid8Octree　CutBid8 型データ用アクセッサ提供クラス
CutBid5Octree　CutBid5 型データ用アクセッサ提供クラス

ともに，抽象クラス CutPosOctree および CutBidOctree を実装したものです[*3](図 7)．これらのクラスでは，内部に交点情報格納用のデータ領域を持っていません．SklCell クラスのオブジェクトに結合して，そのデータ領域へのアクセスを提供します．

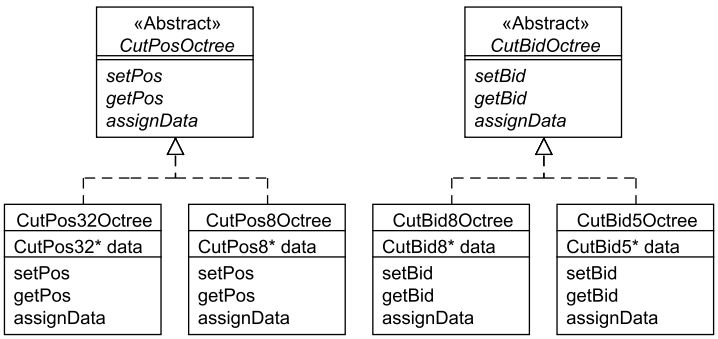

図 7　SklCell データへのアクセッサ提供クラス

以下の各メソッドの説明では，個々のクラス名の代りに，抽象クラス CutPosOctree, CutBidOctree を用いています．実際の使用時には，`CutPosOctree` → `CutPos32Octree` 等，適宜置き換えてください．

[*3] 実際には，基本データ型をパラメータに持つテンプレートクラスを通して定義されており，それぞれ，CutPosOctreeTemplate<CutPos32,6>，CutPosOctreeTemplate<CutPos8,2>，CutBidOctreeTemplate<CutBid8,2>, CutBidOctreeTemplate<CutBid5,1> を typedef したものになっています．

2.3.1 コンストラクタ

```
CutPosOctree::CutPosOctree(int index)
CutBidOctree::CutBidOctree(int index)
```

■引数 (IN)

`int index`	SklCell データ領域内での交点情報格納開始インデックス

2.3.2 SklCell データ領域に結合

```
void CutPosOctree::assignData(float* data)
void CutPosOctree::assignData(float* data)
```

■引数 (IN)

`float* data`	SklCell データ領域へのポインタ

2.3.3 方向別アクセスメソッド

```
float CutPosOctree::getPos(int d)
BidType CutBidOctree::getBid(int d)
```

■引数 (IN)

`int d`	交点探査方向 (0〜5)

2.3.4 6 方向同時アクセスメソッド

```
void CutPosOctree::getPos(float pos[])
void CutBidOctree::getBid(BidType bid[])
```

■引数 (OUT)

`float pos[6]`	交点座標
`BidType bid[6]`	境界 ID

2.3.5 必要領域サイズ取得メソッド

```
unsigned CutPosOctree::getSizeInFloat()
```

```
unsigned CutBidOctree::getSizeInFloat()
```

■ノート　交点情報の格納に必要な領域サイズを float 単位で返す．この値を，Octree 構築時の CreateTree メソッドで指定するセルに格納するデータ数パラメータを決めるのに利用できる．

3　交点情報計算関数インタフェース

この節では，各交点情報計算関数のインタフェースについて述べます．まず、全ての交点情報計算関数に共通するポリゴングループと境界 ID の指定方法，そして，セル中心間版およびノード間版で共通する計算領域の指定方法について説明します．

3.1　計算対象ポリゴングループと境界 ID の指定方法

Polylib 初期化ファイルを通じて，計算対象ポリゴングループとその境界 ID を指定します．Polylib 初期化ファイルでは，各ポリゴングループの定義とポリゴングループ間の階層関係を，XML 形式により記述します．

本ライブラリでは，ポリゴングループの階層関係において，

> 階層最下位のポリゴングループで，
> Param タグにより 1〜255 の ID 番号が指定されたポリゴングループ

を交点情報計算の対象とします．この時，Polylib 初期化ファイルで指定された ID 番号が，そのまま境界 ID となります．ID 番号は，複数のポリゴングループに重複して与えることができます．なお，Polylib では，初期化ファイルにおいて ID 番号の指定を省略したポリゴングループには，ID 番号として 0 が設定されます．

以下の Polylib 初期化ファイル例では，交点情報計算の対象に

> 境界 ID 1 としてポリゴングループ"root/child_A"を
> 境界 ID 2 としてポリゴングループ"root/child_B"を

指定しています．

```
<?xml version="1.0"?>
<Parameter>
  <Elem name="PolygonGroup">
    <Param name="class_name" dtype="STRING" value="PolygonGroup"/>
    <Param name="name"       dtype="STRING" value="root"/>
    <Elem name="PolygonGroup">
      <Param name="class_name" dtype="STRING" value="PolygonGroup"/>
      <Param name="name"       dtype="STRING" value="child_A"/>
      <Param name="filepath"   dtype="STRING" value="child_A.stl"/>
      <Param name="id"         dtype="INT"    value="1"/>
    </Elem>
    <Elem name="PolygonGroup">
      <Param name="class_name" dtype="STRING" value="PolygonGroup"/>
      <Param name="name"       dtype="STRING" value="child_B"/>
      <Param name="filepath"   dtype="STRING" value="child_B.stl"/>
      <Param name="id"         dtype="INT"    value="2"/>
    </Elem>
  </Elem>
</Parameter>
```

Polylib 初期化ファイルの詳細については，Polylib 利用説明書を参照ください．

3.2　計算領域指定方法

セル中心間版およびノード間版では，計算対象となる領域の位置とサイズを次のパラメータにより指定します．

`size_t ncell[3]`	セル数
`size_t nnode[3]`	ノード数 (=`{ncell[0]+1,ncell[1]+1,ncell[2]+1}`)
`float org[3]`	原点座標
`float d[3]`	セルサイズ (=ノード間隔)

図 8 に，計算対象領域を XY 平面に射影した模式図を示します．

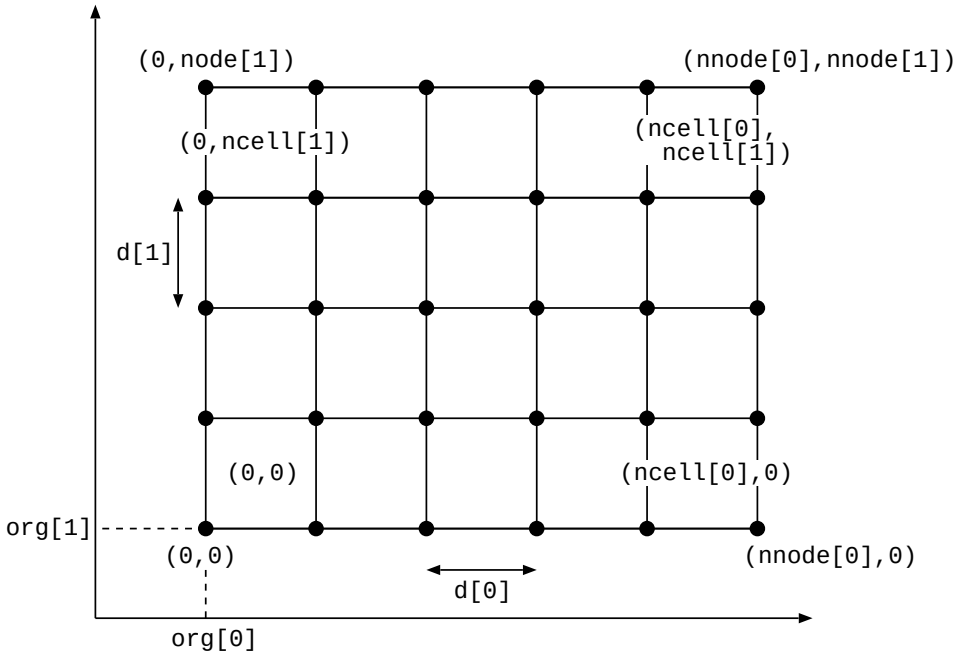

図 8　対象領域指定

計算基準点の総数は，セル中心間版では `ncell[0]`×`ncell[1]`×`ncell[2]`，ノード間版では `node[0]`×`node[1]`×`node[2]` になります．配列ラッパクラスを用いない場合には，このサイズの交点情報基本データ型 1 次元配列を確保しておく必要があります．

また，以下のパラメータを指定することにより，計算基準点の一部についてのみ交点情報を計算することも可能です．

`size_t ista[3]`	計算領域開始セル/ノード位置
`size_t nlen[3]`	計算領域セル/ノード数

図 9，図 10 に，同一の `ista`, `nlen` を指定した場合の，セル中心間版とノード間版での違いを示します．

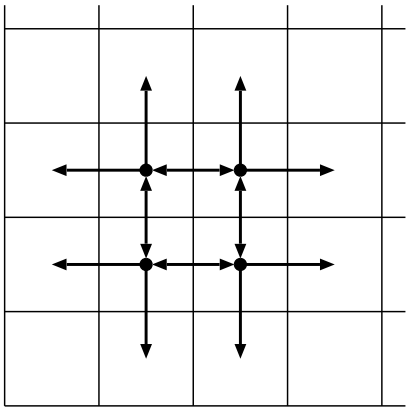

図 9　ista=[1,1],nlen=[2,2] での計算基準点と計算基準線分 (セル中心間版)

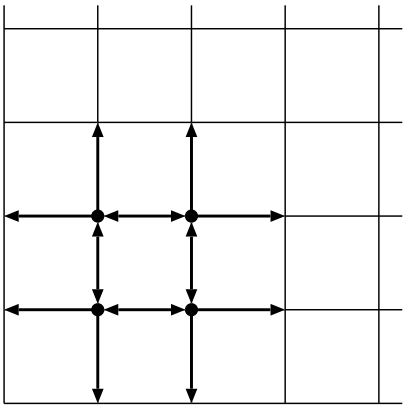

図 10　ista=[1,1],nlen=[2,2] での計算基準点と計算基準線分 (ノード間版)

3.3　リターンコード

交点情報計算関数は全て，整数値のリターンコードを返します．それらは，Cutlib.h で以下のような enum 定数として定義されています．

`SUCCESS`	`= 0`	成功
`BAD_GROUP_LIST`	`= 1`	「境界 ID, ポリゴングループ名」対応リストが不正
`BAD_POLYLIB`	`= 2`	Polylib オブジェクトが不正 (未初期化等)
`BAD_SKLTREE`	`= 3`	SklTree オブジェクトが不正 (未初期化等)
`SIZE_EXCEED`	`= 4`	`ista[]+nlen[]` が配列サイズを越えている
`OTHER_ERROR`	`= 10`	その他のエラー (現在は未使用)

3.4　セル中心間版インタフェース

配列ラッパクラス, 全領域で計算

```
int CutInfoCell(const float org[], const float d[],
                const Polylib* pl,
                CutPosArray* cutPos, CutBidArray* cutBid)
```

配列ラッパクラス, 計算領域指定

```
int CutInfoCell(const size_t ista[], const size_t nlen[],
                const float org[], const float d[],
                const Polylib* pl,
                CutPosArray* cutPos, CutBidArray* cutBid)
```

1 次元配列，全領域で計算

```
int CutInfoCell(const size_t ncell[],
```

```
                const float org[], const float d[],
                const Polylib* pl,
                CutPos cutPos[], CutBid cutBid[])
```

1 次元配列，計算領域指定

```
int CutInfoCell(const size_t ncell[],
                const size_t ista[], const size_t nlen[],
                const float org[], const float d[],
                const Polylib* pl,
                CutPos cutPos[], CutBid cutBid[])
```

■引数 (IN)

`size_t ncell[3]`	セル数
`float org[3]`	領域原点
`float d[3]`	セルサイズ
`size_t ista[3]`	計算対象領域開始セル位置
`size_t nlen[3]`	計算対象領域セル数
`Polylib* pl`	Polylib クラスへのポインタ

■引数 (OUT)

`CutPosArray* cutPos`	CutPos32Array または CutPos8Array へのポインタ
`CutBidArray* cutBid`	CutBid8Array または CutBid5Array へのポインタ
`CutPos cutPos[]`	CutPos32 型または CutPos8 型の 1 次元配列
`CutBid cutBid[]`	CutBid8 型または CutBid5 型の 1 次元配列

■ノート 配列 `cutPos[]`, `cutBid[]` のサイズは `ncell[0]*ncell[1]*ncell[2]`．配列ラッパクラスのインスタンス `cutPos`, `cutBid` は，コンストラクタに `ncell[]` を渡して生成すること．

3.5 ノード間版インタフェース

配列ラッパクラス, 全領域で計算

```
int CutInfoNode(const float org[], const float d[],
                const Polylib* pl,
                CutPosArray* cutPos, CutBidArray* cutBid)
```

配列ラッパクラス, 計算領域指定

```
int CutInfoNode(const size_t ista[], const size_t nlen[],
                const float org[], const float d[],
                const Polylib* pl,
                CutPosArray* cutPos, CutBidArray* cutBid)
```

1 次元配列，全領域で計算

```
int CutInfoNode(const size_t ncell[],
                const float org[], const float d[],
                const Polylib* pl,
                CutPos cutPos[], CutBid cutBid[])
```

1 次元配列，計算領域指定

```
int CutInfoNode(const size_t ncell[],
                const size_t ista[], const size_t nlen[],
                const float org[], const float d[],
                const Polylib* pl,
                CutPos cutPos[], CutBid cutBid[])
```

■引数 (IN)

`size_t ncell[3]`	セル数
`float org[3]`	領域原点
`float d[3]`	セルサイズ (ノード間隔)
`size_t ista[3]`	計算対象領域開始ノード位置
`size_t nlen[3]`	計算対象領域ノード数
`Polylib* pl`	Polylib クラスへのポインタ

■引数 (OUT)

`CutPosArray* cutPos`	CutPos32Array または CutPos8Array へのポインタ
`CutBidArray* cutBid`	CutBid8Array または CutBid5Array へのポインタ
`CutPos cutPos[]`	CutPos32 型または CutPos8 型の 1 次元配列
`CutBid cutBid[]`	CutBid8 型または CutBid5 型の 1 次元配列

■ノート 1 次元配列を用いるインタフェースの第 1 引数が，ノード数 `nnode[]` ではなくセル数 `ncell[]` であることに注意．ただし，`nnode[]={ncell[0]+1,ncell[1]+1,ncell[2]+1}` として，配列 `cutPos[]`, `cutBid[]` のサイズは `nnode[0]*nnode[1]*nnode[2]`．配列ラッパクラスのインスタンス `cutPos`, `cutBid` は，コンストラクタに `nnode[]` を渡して生成すること．

3.6 Octree 版インタフェース

リーフセルのみで計算

```
int CutInfoOctreeLeafCell(SklTree* tree,
                          const Polylib* pl,
                          CutPosOctree* cutPos, CutBidOctree* cutBid)
```

全セルで計算

```
int CutInfoOctreeAllCell(SklTree* tree,
                         const Polylib* pl,
                         CutPosOctree* cutPos, CutBidOctree* cutBid)
```

計算対象セルを指定

```
int CutInfoOctree(SklTree* tree,
                  const Polylib* pl,
                  CutPosOctree* cutPos, CutBidOctree* cutBid,
                  bool leafCellOnly = true)
```

■引数 (IN)

`Polylib* pl`	Polylib クラスへのポインタ
`bool leafCellOnly`	true:リーフセルのみ (ディフォルト)/false:全セル

■引数 (IN/OUT)

SklTree* tree	SklTree へのポインタ
`CutPosOctree* cutPos`	CutPos32Octree または CutPos8Octree へのポインタ
`CutBidOctree* cutBid`	CutBid8Octree または CutBid5Octree へのポインタ

4 使用例

4.1 セル中心間版 (ラッパクラス使用)

```
#include "Cutlib.h"
using namespace cutlib;

   ...

  std::string plConfig;  // Polylib 初期化ファイル名

  size_t ncell[3];     // 全セル数
  float org[3];        // 領域原点座標
  float d[3];          // セル間隔

  size_t ista[3];      // 計算対象領域開始セル位置
  size_t nlen[3];      // 計算対象領域セル数

  /* ここで，各パラメータに値をセット */

  // Polylib 初期化
  Polylib pl = Polylib::get_instance();
  pl->load(plConfig);

  //配列ラッパクラス（およびその内部の配列データ領域）の生成
  CutPos32Array* cutPos = new CutPos32Array(ncell);
//CutPos8Array*  cutPos = new CutPos8Array(ncell);
  CutBid8Array*  cutBid = new CutBid8Array(ncell);
//CutBid5Array*  cutBid = new CutBid5Array(ncell);

  CutInfoCell(ista, nlen, org, d, pl, cutPos, cutBid);

  // 以下は計算した交点情報の読み出し例

  // 6 方向まとめて交点情報を読み出し
  float   pos6[6];
  BidType bid6[6];
  cutPos->getPos(i, j, k, pos6);
  cutBid->getBid(i, j, k, bid6);

  // d(0〜5) 方向のみ読み出し
  float   pos = cutPos->getPos(i, j, k, d);
  BidType bid = cutBid->getBid(i, j, k, d);

  // Fortran 用に配列データをエクスポート
  CutPos32* posData = cutPos->getDataPointer();
//CutPos8*  posData = cutPos->getDataPointer();
  CutBid8*  bidData = cutBid->getDataPointer();
//CutBid5*  bidData = cutBid->getDataPointer();

  // 関数による配列要素からの直接読み出しも可能
  int ijk = i + j*ncell[0] + k*ncell[0]*ncell[1];
  GetCutPos(posData, ijk, pos6);
  bid = GetCutBid(bidData, ijk, d);

   ...
```

4.2 ノード間版 (ラッパクラス使用)

```
#include "Cutlib.h"
using namespace cutlib;

   ...

  std::string plConfig;  // Polylib 初期化ファイル名

  size_t nnode[3];     // 全ノード数
  float org[3];        // 領域原点座標
  float d[3];          // セル間隔

  size_t ista[3];      // 計算対象領域開始ノード位置
  size_t nlen[3];      // 計算対象領域ノード数

  /* ここで，各パラメータに値をセット */

  // Polylib 初期化
  Polylib pl = Polylib::get_instance();
  pl->load(plConfig);

  //配列ラッパクラス（およびその内部の配列データ領域）の生成
  CutPos32Array* cutPos = new CutPos32Array(nnode);
//CutPos8Array*  cutPos = new CutPos8Array(nnode);
  CutBid8Array*  cutBid = new CutBid8Array(nnode);
//CutBid5Array*  cutBid = new CutBid5Array(nnode);

  CutInfoCell(ista, nlen, org, d, pl, cutPos, cutBid);

  // 以下は計算した交点情報の読み出し例

  // 6 方向まとめて交点情報を読み出し
  float   pos6[6];
  BidType bid6[6];
  cutPos->getPos(i, j, k, pos6);
  cutBid->getBid(i, j, k, bid6);

  // d(0〜5) 方向のみ読み出し
  float   pos = cutPos->getPos(i, j, k, d);
  BidType bid = cutBid->getBid(i, j, k, d);

  // Fortran 用に配列データをエクスポート
  CutPos32* posData = cutPos->getDataPointer();
//CutPos8*  posData = cutPos->getDataPointer();
  CutBid8*  bidData = cutBid->getDataPointer();
//CutBid5*  bidData = cutBid->getDataPointer();

  // 関数による配列要素からの直接読み出しも可能
  int ijk = i + j*nnode[0] + k*nnode[0]*nnode[1];
  GetCutPos(posData, ijk, pos6);
  bid = GetCutBid(bidData, ijk, d);

   ...
```

4.3 セル中心間版 (1 次元配列使用)

```
#include "Cutlib.h"
using namespace cutlib;

   ...

  std::string plConfig;  // Polylib 初期化ファイル名

  size_t ncell[3];    // 全セル数
  float org[3];       // 領域原点座標
  float d[3];         // セル間隔

  size_t ista[3];     // 計算対象領域開始セル位置
  size_t nlen[3];     // 計算対象領域セル数

  /* ここで，各パラメータに値をセット */

  // Polylib 初期化
  Polylib pl = Polylib::get_instance();
  pl->load(plConfig);

  //基本データ型の配列
  CutPos32* cutPos = new CutPos32[ncell[0]*ncell[1]*ncell[2]];
//CutPos8*  cutPos = new CutPos8[ncell[0]*ncell[1]*ncell[2]];
  CutBid8*  cutBid = new CutBid8[ncell[0]*ncell[1]*ncell[2]];
//CutBid5*  cutBid = new CutBid5[ncell[0]*ncell[1]*ncell[2]];

  CutInfoCell(ncell, ista, nlen, org, d, pl, cutPos, cutBid);

  // 以下は計算した交点情報の読み出し例

  int ijk = i + j*ncell[0] + k*ncell[0]*ncell[1];

  // 6 方向まとめて交点情報を読み出し
  float   pos6[6];
  BidType bid6[6];
  GetCutPos(cutPos, ijk, pos6);
  GetCutBid(cutBid, ijk, bid6);

  // d(0〜5) 方向のみ読み出し
  float   pos = GetCutPos(cutPos, ijk, d);
  BidType bid = GetCutBid(cutBid, ijk, d);

   ...
```

4.4 ノード間版 (1 次元配列使用)

```
#include "Cutlib.h"
using namespace cutlib;

   ...

  std::string plConfig;  // Polylib 初期化ファイル名

  size_t ncell[3];     // 全セル数
  float org[3];        // 領域原点座標
  float d[3];          // セル間隔

  size_t ista[3];      // 計算対象領域開始セル位置
  size_t nlen[3];      // 計算対象領域セル数

  /* ここで，各パラメータに値をセット */

  // Polylib 初期化
  Polylib pl = Polylib::get_instance();
  pl->load(plConfig);

  // 全ノード数
  size_t nnode[3] = {ncell[0]+1, ncell[1]+1, ncell[2]+1};

  //基本データ型の配列
  CutPos32* cutPos = new CutPos32[nnode[0]*nnode[1]*nnode[2]];
//CutPos8*  cutPos = new CutPos8[nnode[0]*nnode[1]*nnode[2]];
  CutBid8*  cutBid = new CutBid8[nnode[0]*nnode[1]*nnode[2]];
//CutBid5*  cutBid = new CutBid5[nnode[0]*nnode[1]*nnode[2]];

  CutInfoCell(ncell, ista, nlen, org, d, pl, bList, cutPos, cutBid);
            // ↑第 1 引数は nnode ではなく ncell

  // 以下は計算した交点情報の読み出し例

  int ijk = i + j*nnode[0] + k*nnode[0]*nnode[1];

  // 6 方向まとめて交点情報を読み出し
  float   pos6[6];
  BidType bid6[6];
  GetCutPos(cutPos, ijk, pos6);
  GetCutBid(cutBid, ijk, bid6);

  // d(0〜5) 方向のみ読み出し
  float   pos = GetCutPos(cutPos, ijk, d);
  BidType bid = GetCutBid(cutBid, ijk, d);

   ...
```

4.5 Octree 版 (リーフセルのみで計算)

```
#include "Cutlib.h"
using namespace cutlib;

   …

  std::string plConfig;  // Polylib 初期化ファイル名

  SklTree* tree;         // Octree クラス

  unsigned dIndex = ...; // 交点情報格納開始位置

  // アクセッサクラスの生成
  CutPos32Octree* cutPos = new CutPos32Octree(dIndex);
//CutPos8Octree*  cutPos = new CutPos8Octree(dIndex);
  CutBid8Octree*  cutBid
                    = new CutBid8Octree(dIndex + cutPos->getSizeInFloat());
//CutBid5Octree*  cutBid
//                  = new CutBid5Octree(dIndex + cutPos->getSizeInFloat());

  // 各セルに確保させるデータ量
  unsigned dLen = cutPos->getSizeInFloat() + cutBid->getSizeInFloat() + … ;

  /* ここで，Octree 構築，各パラメータに値をセット */

  // Polylib 初期化
  Polylib pl = Polylib::get_instance();
  pl->load(plConfig);

  CutInfoOctreeLeafCell(tree, pl, cutPos, cutBid);

  // 以下は計算した交点情報の読み出し例

  // リーフセルを巡回するループ
  for (SklCell* cell = tree->GetLeafCellFirst(); cell != 0;
       cell = tree->GetLeafCellNext(cell)) {

    // アクセッサクラスにセルのデータ領域を結合
    cutPos->assignData(cell->GetData());
    cutBid->assignData(cell->GetData());

    // 6 方向まとめて交点情報を読み出し
    float   pos6[6];
    BidType bid6[6];
    cutPos->getPos(pos6);
    cutBid->getBid(bid6);

    // d(0〜5) 方向みの読み出し
    float   pos = cutPos->getPos(d);
    BidType bid = cutBid->getBid(d);

    …
  }

  …
```

4.6 Octree 版 (全セルで計算)

```
#include "Cutlib.h"
using namespace cutlib;

// 再帰的にツリーの全セルを巡る関数
void extractDataFromCell(SklCell* cell, CutPosOctree* cp, CutBidOctree* cb);

   …

  std::string plConfig;  // Polylib 初期化ファイル名

  SklTree* tree;         // Octree クラス

  unsigned dIndex = ...; // 交点情報格納開始位置

  // アクセッサクラスの生成
  CutPos32Octree* cutPos = new CutPos32Octree(dIndex);
//CutPos8Octree*  cutPos = new CutPos8Octree(dIndex);
  CutBid8Octree*  cutBid
                    = new CutBid8Octree(dIndex + cutPos->getSizeInFloat());
//CutBid5Octree*  cutBid
//                  = new CutBid5Octree(dIndex + cutPos->getSizeInFloat());

  // 各セルに確保させるデータ量
  unsigned dLen = cutPos->getSizeInFloat() + cutBid->getSizeInFloat() + … ;

  /* ここで，Octree 構築，各パラメータに値をセット */

  // Polylib 初期化
  Polylib pl = Polylib::get_instance();
  pl->load(plConfig);

  CutInfoOctreeAllCell(tree, pl, cutPos, cutBid);

  // 以下は計算した交点情報の読み出し例

  size_t nx, ny, nz;    // ルートセル数
  tree->GetSize(nx, ny, nz);
  for (size_t k = 0; k < nz; k++) {
    for (size_t j = 0; j < ny; j++) {
      for (size_t i = 0; i < nx; i++) {
        SklCell* cell = tree->GetRootCell(i, j, k);
        extractDataFromCell(cell, cutPos, cutBid);
      }
    }
  }

  …

/*
 * 再帰的にツリーの全セルを巡る関数
 */
void extractDataFromCell(SklCell* cell, CutPosOctree* cp, CutBidOctree* cb)
{
  // アクセッサクラスにセルのデータ領域を結合
  cp->assignData(cell->GetData());
  cb->assignData(cell->GetData());

  // 6 方向まとめて交点情報を読み出し
  float   pos6[6];
```

```
  BidType bid6[6];
  cutPos->getPos(pos6);
  cutBid->getBid(bid6);

  // d(0〜5) 方向のみ読み出し
  float   pos = cutPos->getPos(d);
  BidType bid = cutBid->getBid(d);

  // もし子セルがあるなら…
  if (cell->hasChild()) {
    for (TdPos child = 0; child < 8; child++) {
      SklCell* childCell = cell->GetChildCell(child);
      extractDataFromCell(childCell, cp, cb);   // 再帰呼び出し
    }
  }
}

  …
```