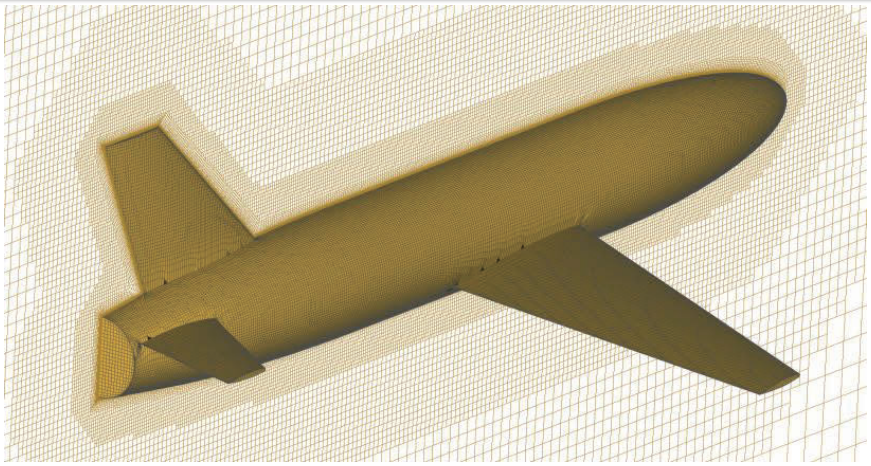
(FDC/ANSS合同企画6)「航空教育支援フォーラム：JAXA提供研究成果の最新動向」

最新動向紹介：BOXFUN



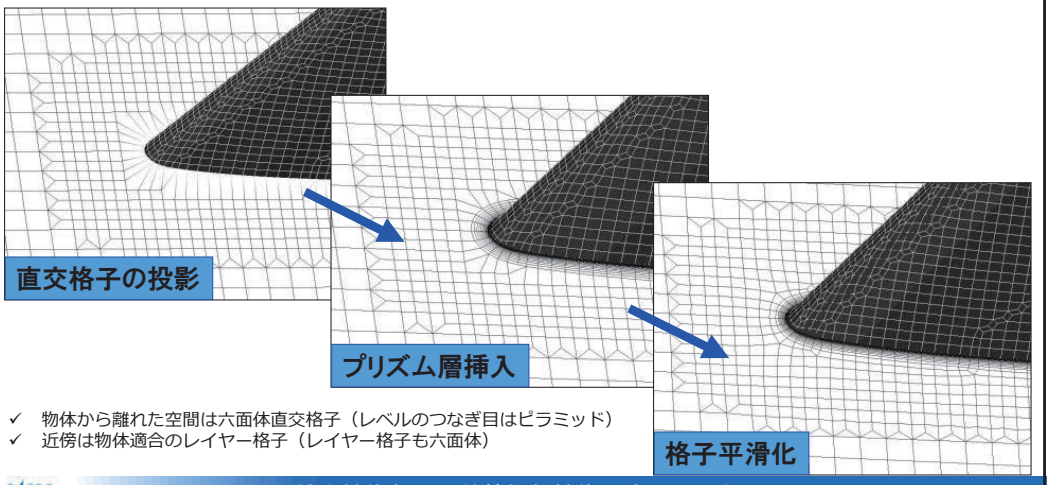
宇宙航空研究開発機構　航空技術部門
数値解析技術研究ユニット
石田　崇

発表内容

- 研究背景
  - HexaGridからBOXFUNへ
- BOXFUN紹介
  - BOXFUN SURF
  - BOXFUN VOX
  - BOXFUN UNS
- 現状の課題
- まとめと今後

HexaGridの開発

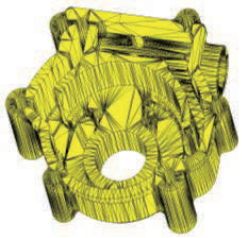
- 六面体ベースの自動格子生成ソフト
  - 手動で作成すると~1カ月⇒HexaGridで1~2時間
  - 直交格子に基づく非構造格子⇒高速に生成可能，複雑形状に対応
  - 通常のPCまたはJSS（JAXAスパコン）で動作

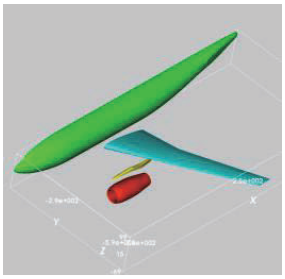


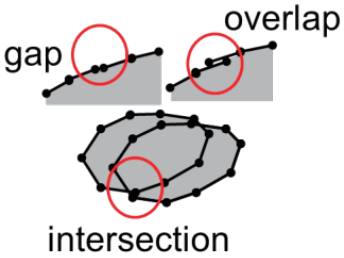
✓ 物体から離れた空間は六面体直交格子（レベルのつなぎ目はピラミッド）
✓ 近傍は物体適合のレイヤー格子（レイヤー格子も六面体）

HexaGridの特徴

- 質の悪いSTLデータにも対応 ⇒ CADデータ修正作業の削減
- 複数のSTLデータに対応可能 ⇒ パーツの入れ替えが容易



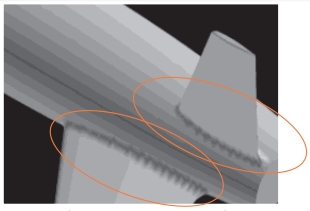




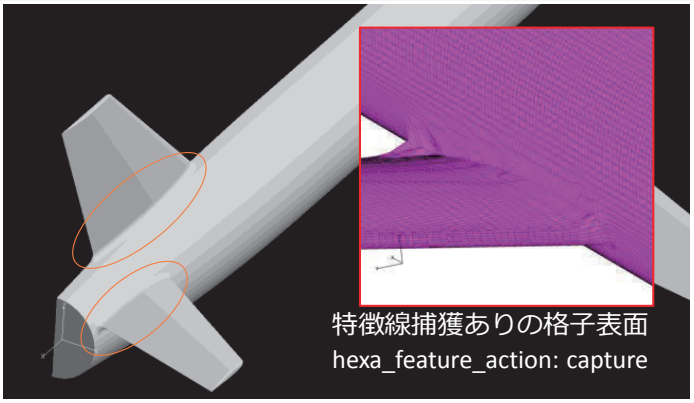
※STLデータに小さなギャップ、オーバーラップ、交差があっても対応可能

- 少ないパラメータで自動格子生成 ⇒ ユーザフレンドリー
  - 領域サイズ (x, y, zの最大・最小)
  - 物体表面のセルサイズ（最大・最小）
  - レイヤー格子の最小格子幅、拡大率

HexaGridの課題



特徴線捕獲なしの格子表面
hexa_feature_action: smooth



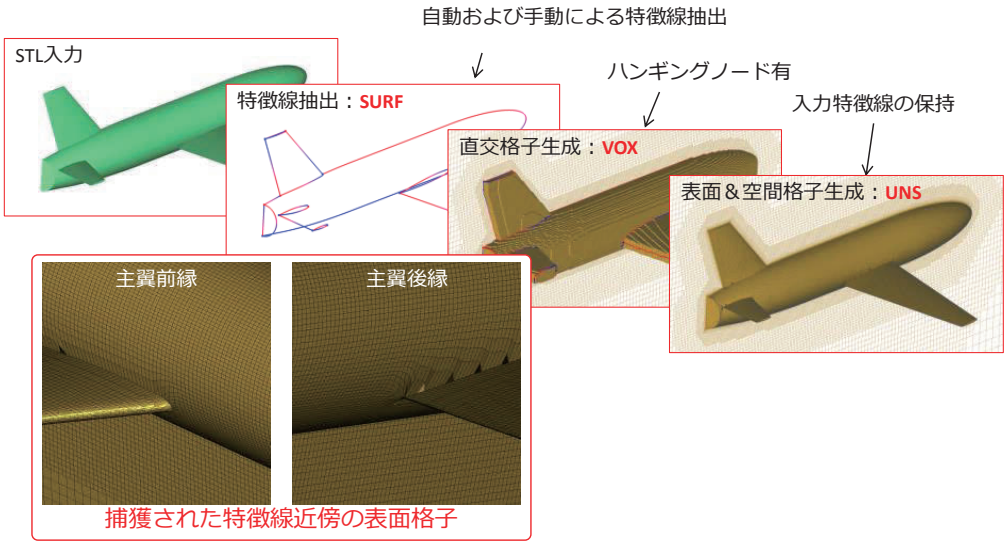
特徴線捕獲ありの格子表面
hexa_feature_action: capture

- 特徴線捕獲のオプションはあるが，うまく捕獲できずにバリが出来る．
  ⇒凹形状に対する特徴線捕獲が苦手
- シングルプロセス用のプログラムなので，格子生成に限界がある．
  ⇒大規模格子生成に向いていない

BOXFUNの開発

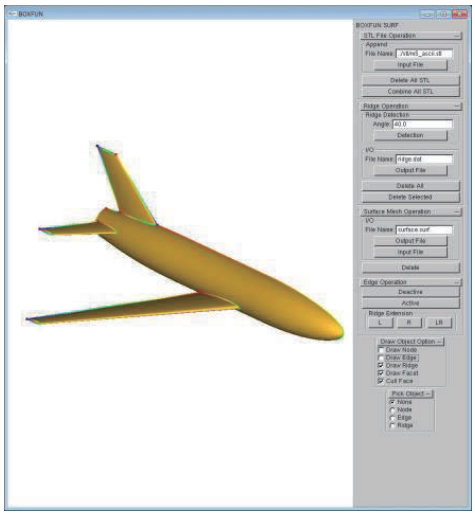
- HexaGridの課題を解決するべく，新たな格子生成コードの開発に着手
  - 特徴線捕獲　⇒マニュアル操作機能を組み込む
  - 大規模格子生成　⇒Building-Cube法のフレームワーク活用
- 名称：BOXFUN
  Block-based vOXel for Fine UNstructured grid
- 開発環境
  - 開発言語：　C++, Visual Studio 2010
  - 可視化ライブラリ：　OpenGL, GLUT
  - GUIフレーム：　GLUI（研究開発用），Qt（配布用）
- 動作環境
  - Windows
  - Linux（JSS2のリモートデスクトップ含む）
- プログラム構成
  - BOXFUN SURF：　特徴線抽出モジュール
  - BOXFUN VOX：　直交格子生成モジュール
  - BOXFUN UNS：　非構造格子変換モジュール

格子生成の流れ



BOXFUN SURF

■ 入力STLファイルから非構造データおよび稜線情報を構築するモジュール



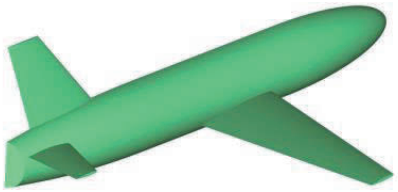
■ STLファイル入力
- ✓ バイナリ
- ✓ アスキー

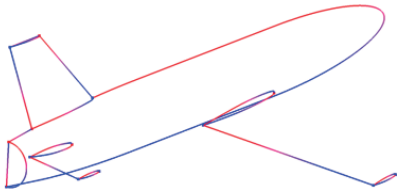
■ 非構造データ作成
- ✓ node
- ✓ edge
- ✓ triangle

■ 稜線情報抽出
- ✓ 自動抽出
- ✓ 手動抽出（マウスピック）

■ 表面格子データ作成
- ✓ facet情報
- ✓ 稜線情報

特徴線抽出

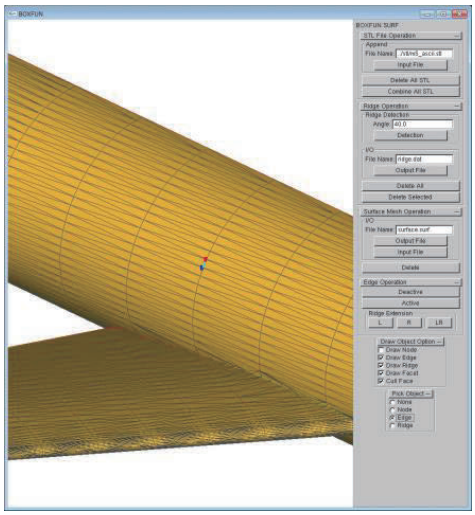


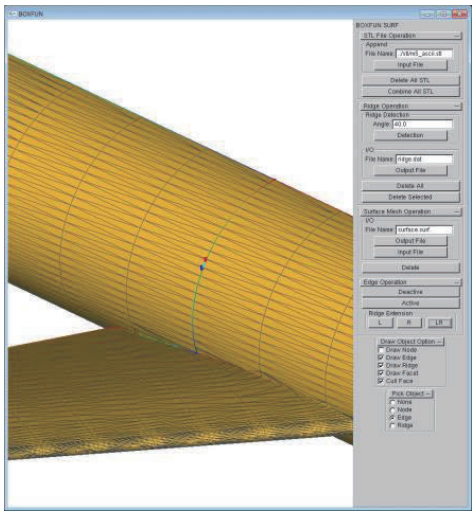


- 入力はSTLファイル形式を想定
- 閾値を入力し，形状特徴を抽出
- マウスによる手動抽出も可能

BOXFUN SURF

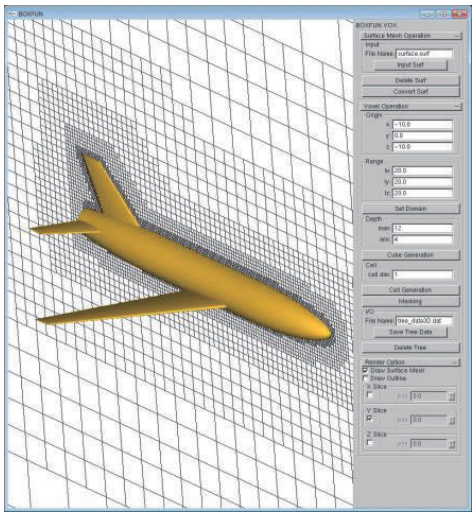
- 手動による稜線情報作成





BOXFUN VOX

■ 入力表面格子データを基に階層型直交格子を生成するモジュール



■表面格子データ入力
- ✓バイナリ

■直交格子生成
- ✓階層型
- ✓セルベース/ブロックベース
- ✓IB法用モジュール
- ✓LBM用モジュール

■直交格子データI/O
- ✓Tree情報
- ✓バイナリ

Voxelデータ構造の詳細

理研フレームワークより

```
template<size_t D>
OctreeNode{
    OctreeIPos<D>          m_pos;
    union{
        OctreeNode<D>      *m_child;
        OctreeLeaf<D>      *m_leaf;
    };
};

template<size_t D>
OctreeInfo{
    unsigned short         depth;
    unsigned short         pdg[D];
};
```

m_pos : 2+2D=2(D+1)[byte]
m_child : 4[byte]
合計 : 2(D+4)[byte]

BOXFUN

```
template<size_t D>
OmnitreeNode{
    OmnitreeIPos<D>        m_pos;
    union{
        OmnitreeNode<D>    *m_child;
        OmnitreeLeaf<D>    *m_leaf;
    };
};

template<size_t D>
OmnitreeInfo{
    unsigned short         flags;
    unsigned short         depth;
    unsigned int           pdg[D];
};
```

m_pos : 2+2+4D=4(D+1)[byte]
m_child : 4[byte]
合計 : 4(D+2)[byte]

情報のビット管理

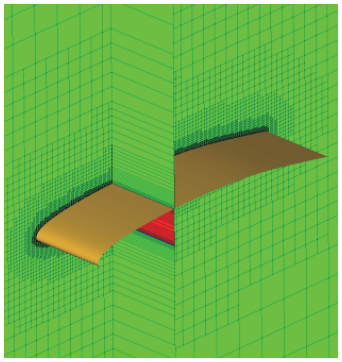
OmnitreeInfo::depth															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
flag	Z depth					Y depth					X depth				

OmnitreeInfo::flags															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID								flag					dimension		
domain number for MPI								leaf	wall	fluid	agg.	div.	Z	Y	X

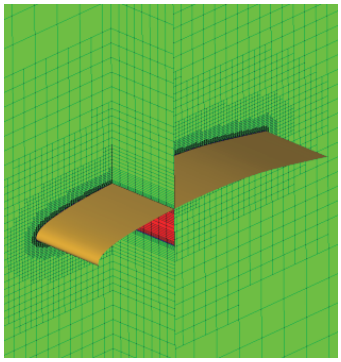
unsigned short型変数　：2[byte]＝16[bit]
unsigned int型変数　：4[byte]＝32[bit]
各軸方向のdepth　：5[bit]　＝0~31
解像度　：2^{31}　＝2147483648
(代表長さL_∞，外部領域$100L_\infty$の時，$4.7 \times 10^{-8}L_\infty$)

擬似2次元格子対応

- 従来のHexaGridには無い機能
- 3次元格子生成モジュールを用いて擬似2次元格子を作成出来る
  - 奥行き方向に座標が一致するよう修正機能を追加
  - Omnitreeの制約上，分割数は2のべき乗
  - 奥行きの長さは自由に設定可能



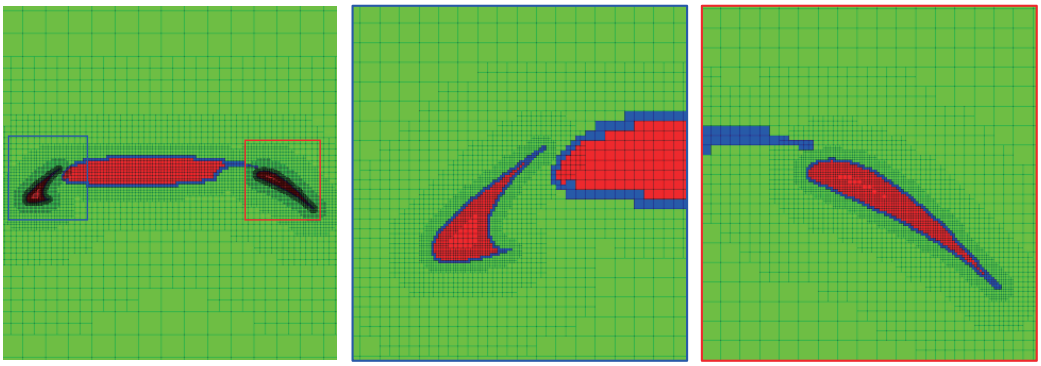
奥行き1セル



奥行き8セル

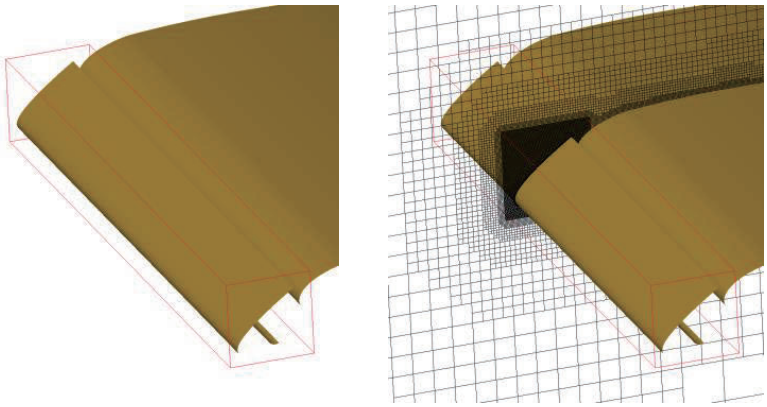
パーツ毎の細分化設定

- パーツ毎に最大・最小階層を設定
- 多要素翼型の格子生成例
  - スラット・フラップ　：15階層
  - メイン　：13階層



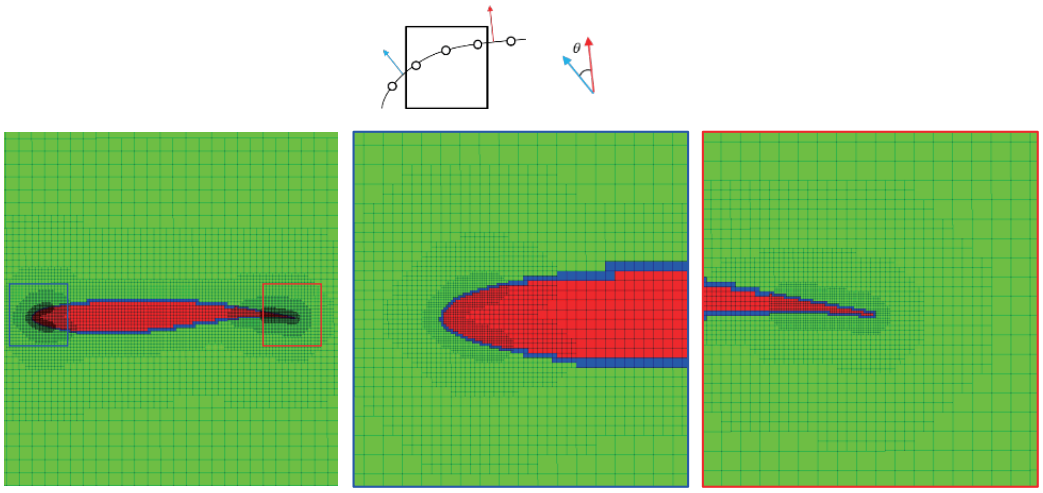
Refinement Boxによる細分化

- GUI上で細分化領域を指定して細分化する
- 現状では領域指定はboxのみ（始点座標，領域サイズ，階層）



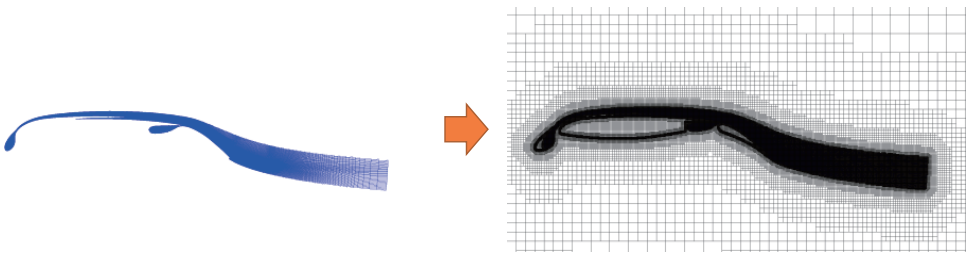
Adaptive Refinement

■ 曲率の大きい所を簡易的※に細分化する
※着目しているvoxelを中心に検査体積を設定，内部に含まれる表面格子の法線ベクトル同士の内積を計算し，最小値が閾値以下なら分割



Adaptive Refinement

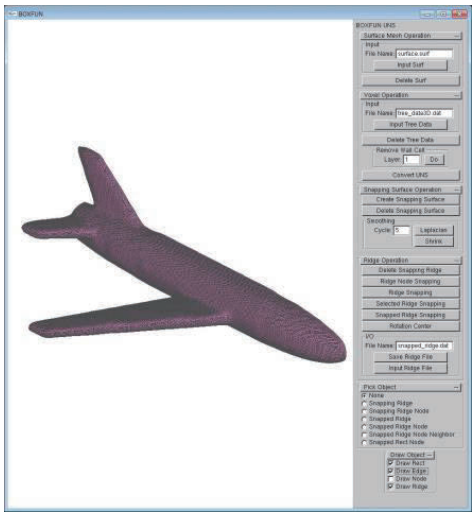
■計算結果から細分化領域を特定・細分化



高揚力装置の後流細分化の例

BOXFUN UNS

■ 6面体非構造格子を生成するモジュール



■ 直交格子データ入力
- ✓ バイナリ

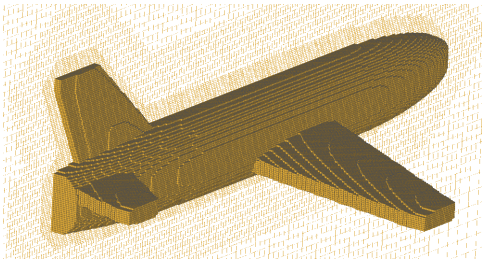
■ 表面格子データ入力
- ✓ バイナリ

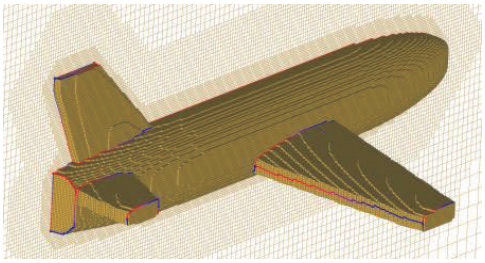
■ 直交格子投影
- ✓ 稜線投影
- ✓ 手動修正（マウスピック）

■ プリズム層挿入
- ✓ 成長率
- ✓ 最小格子幅

■ 非構造格子出力
- ✓ fsgrid
- ✓ 表面格子生成
- ✓ 空間格子生成

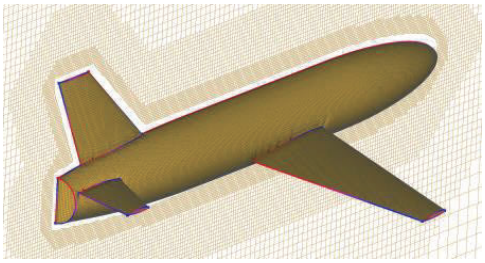
直交投影面作成

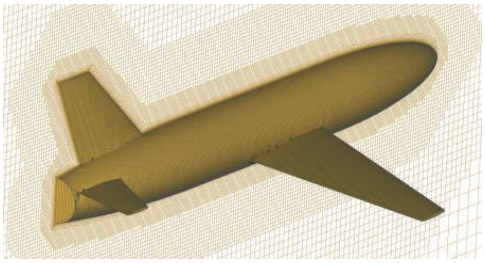




- STLに対して階層型直交格子を生成
- 物体壁面に投影する直交格子面を作成
- 特徴線情報を直交格子面に対応付ける（手動による修正が必要）

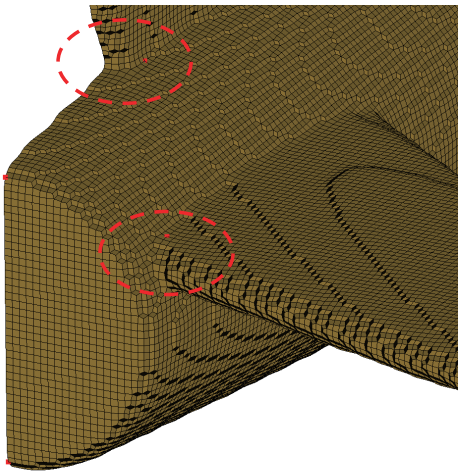
形状への投影＆プリズム層挿入

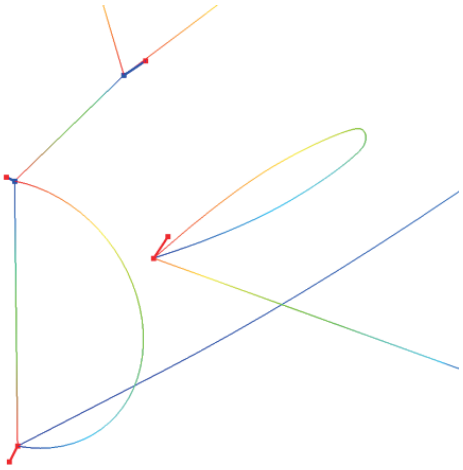




- 直交格子面を物体に投影させ表面格子作成
- 隙間にプリズム層を挿入

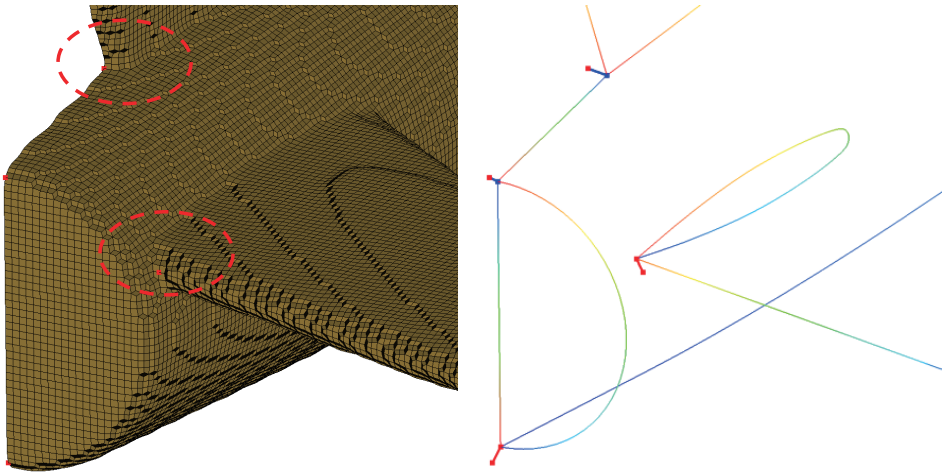
BOXFUN UNS
特徴線の始点・終点の接続修正　Before





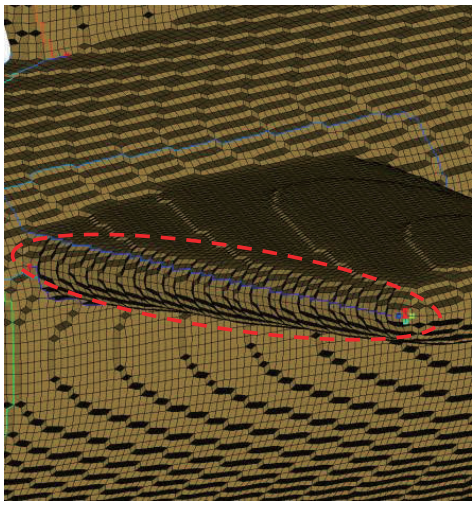
※法線は面積による重み付で方向が決まるため，必ずしも望ましい接続関係にはならない.

BOXFUN UNS
特徴線の始点・終点の接続修正 After

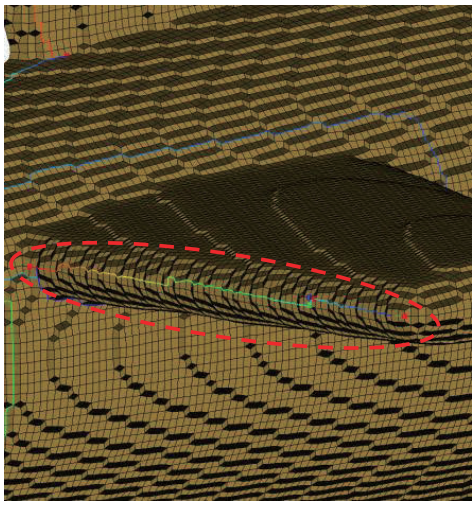


BOXFUN UNS
特徴線の接続関係修正 Before/After

Before



After



※最近点の進行方向ベクトルを用いるため，望ましい接続関係にならないことの方が多い.

特徴線捕獲の改良

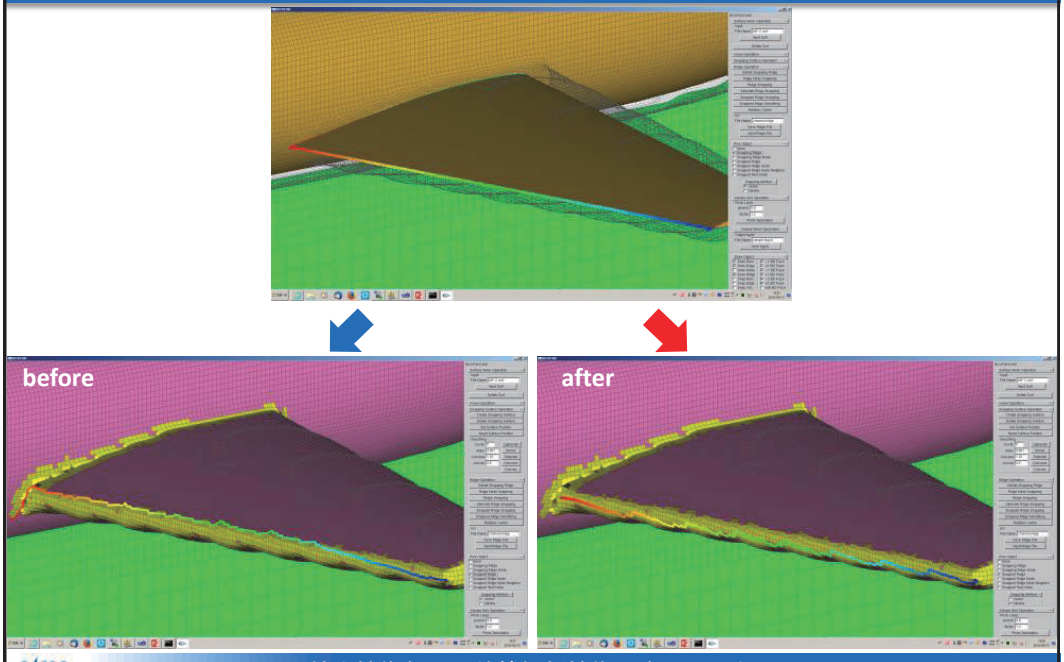
■ ダイクストラ法（Dijkstra's Algorithm）を適用し、自動化を図る

■ ノード毎のコスト（評価関数）には，着目している特徴線との距離を用い、距離の総和が最小になる経路を探索する

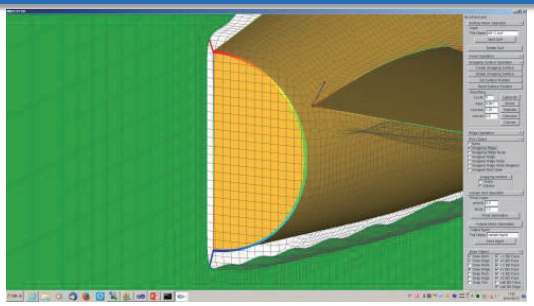
■ 手順

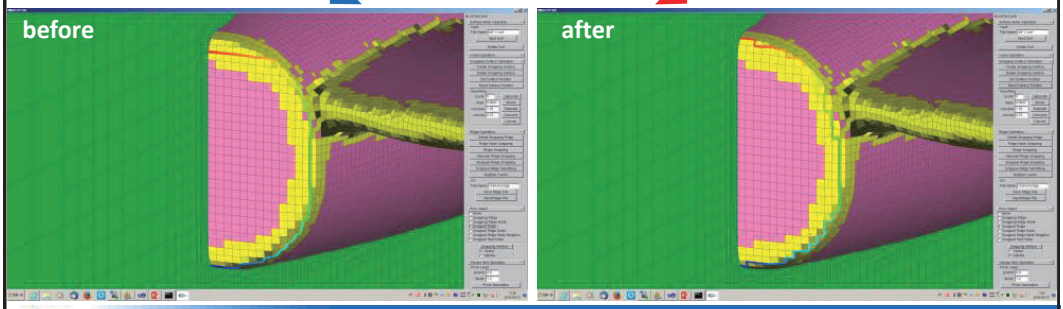
① ある特徴線に着目
② 全ての表面格子点と特徴線との距離計算を行う
③ 距離を評価関数にしてダイクストラ法を適用
④ 始点からコスト計算開始
⑤ 対象の表面格子点が無くなるまでコスト計算実施
⑥ 終点から始点に向かう最短経路（格子点群）をリストアップ

直線形状に対する適用例



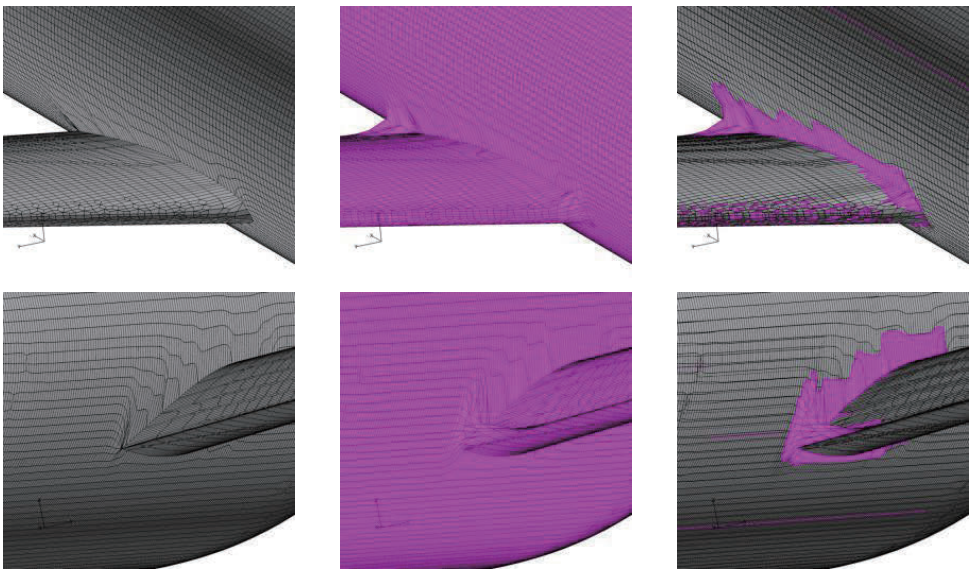
曲線形状に対する適用例





HexaGridとの比較

BOXFUN HexaGrid 比較図



マニュアル操作により形状特徴の再現性が向上

データ構造の整理(64bit環境下)

Shape		
item	number	memory[byte]
bbx ptr	1	8
virtual	1	8
index	1	4
attribute	1	4
total		24

CUNSNode		
item	number	memory[byte]
Shape	1	24
position	3	24
bool	2	8
vector	1	32
ptr	2	16
total		104

CUNSEdge		
item	number	memory[byte]
Shape	1	24
node ptr	2	16
edge ptr	1	8
shape ptr	2	16
total		64

CUNSRect		
item	number	memory[byte]
Shape	1	24
normal	3	24
node ptr	4	32
edge ptr	4	32
shape ptr	4	32
total		144

CUNSHexa		
item	number	memory[byte]
shape	1	24
node	8	64
edge	12	96
face	6	48
total		232

- Shapeクラスを継承させて各種要素のデータ構造を定義
- Hanging node/edge/rectをまとめて取り扱おうとすると、データが肥大化
  ⇒現状ではhanging nodeありの状態

メモリ使用量(64bit環境下)

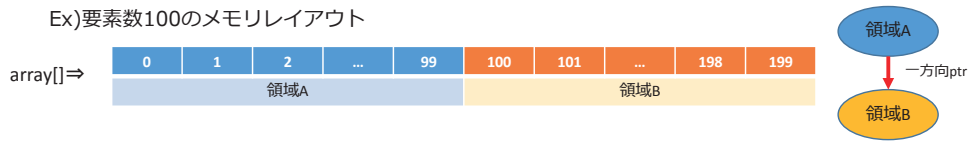
- 非構造データの要素数見積もり（All Hexaの場合）
  - Hexa　nhexa
  - Node　~nhexa
  - Edge　~3*nhexa
  - Rectangle　~3*nhexa
- 各要素のメモリ使用量
  - Hexa　232　[bytes]
  - Node　104　[bytes]
  - Edge　64　[bytes]
  - Rectangle　144　[bytes]
- 全体のメモリ使用量
  - 10M hexa　9.6+α[GB]
  - 100M hexa　96+α [GB]

低メモリ化対応：メモリ確保の変更

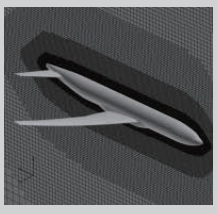
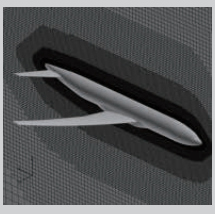
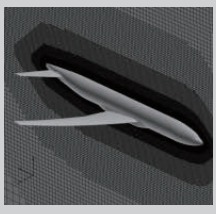
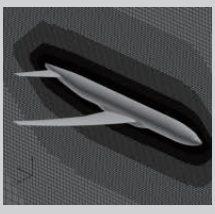
- 従来のメモリ確保方法
  - Voxel要素数（Hexa要素数）から空間格子の要素数を余分に確保
    - Hexa: nhexa
    - Node: 8*nhexa ・・・Hexa要素は頂点数8
    - edge: 12*nhexa ・・・Hexa要素は辺数12
    - Rect: 6*nhexa ・・・Hexa要素は面数6
  - 四角形要素数：nbdrectとプリズム層数：nprismからプリズム層の要素数を同様に確保

- 今回のメモリ確保方法
  - 配列クラスを新たに定義
  - 事前に使用する要素数を決める
  - 格子生成作業中に配列が溢れたら、次のメモリ空間に新たにメモリ領域を確保する
⇒配列の連結リスト
  - 新たなメモリ領域でもarray[]でアクセスできる仕組みを組み込む⇒演算子オーバーロード
  - メモリアクセスのオーバーヘッドは増えるが、使用量は抑えられる

Ex)要素数100のメモリレイアウト

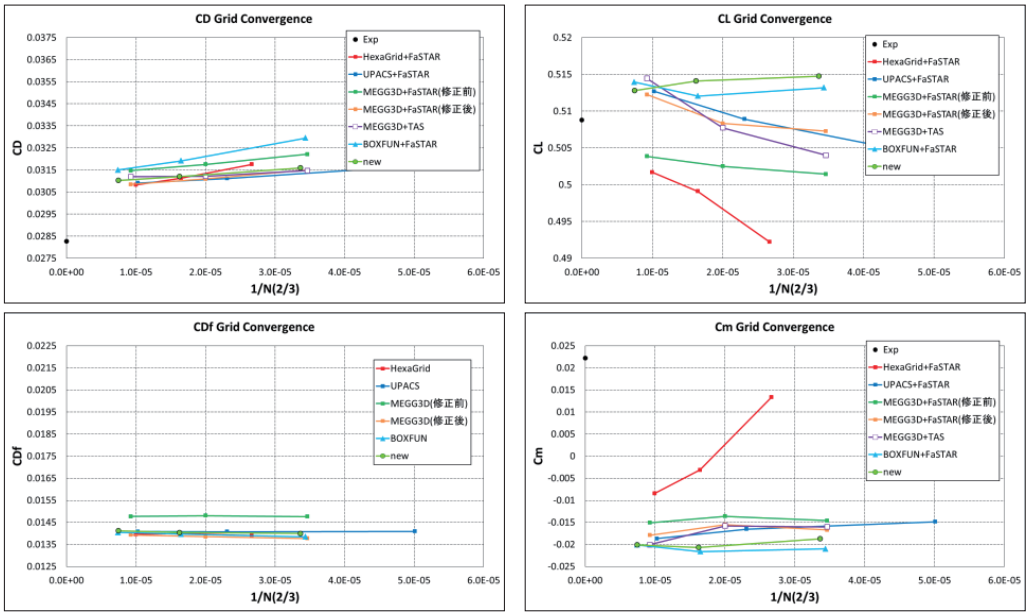


大規模格子生成：NASA-CRM

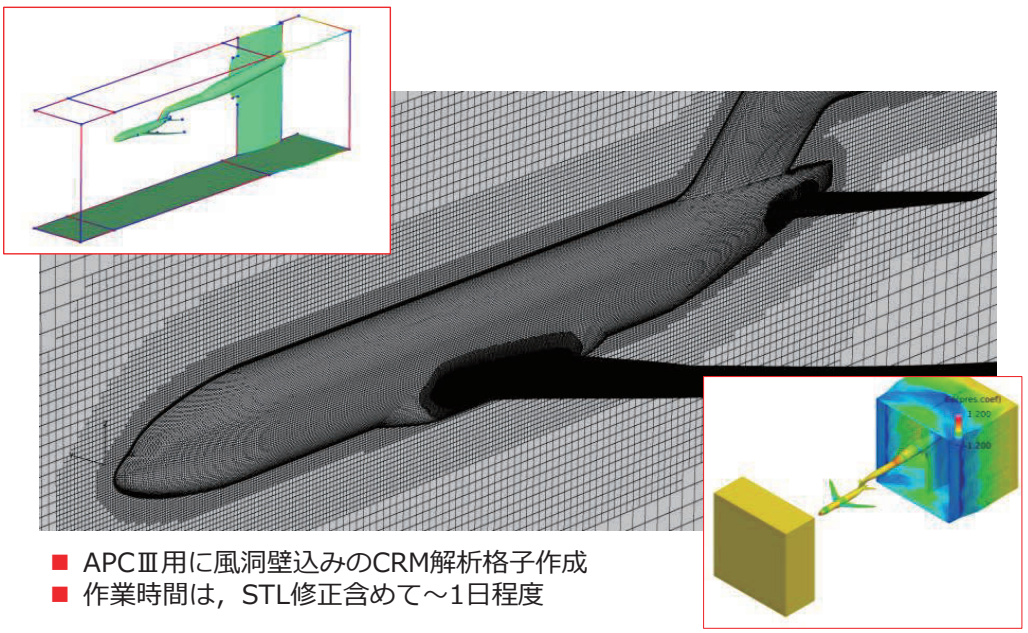
	coarse	medium	fine	Extra
最大階層	13	14	15	16
Smoothing range	20	20	20	20
格子点数	5332807	15787099	50451793	174936195
総セル数	5126829	15310414	49152135	170838235
格子図				

- 壁面でのadaptive refinementは無し⇒表面でのhanging-nodeの影響調査は未実施

APCの条件での格子収束

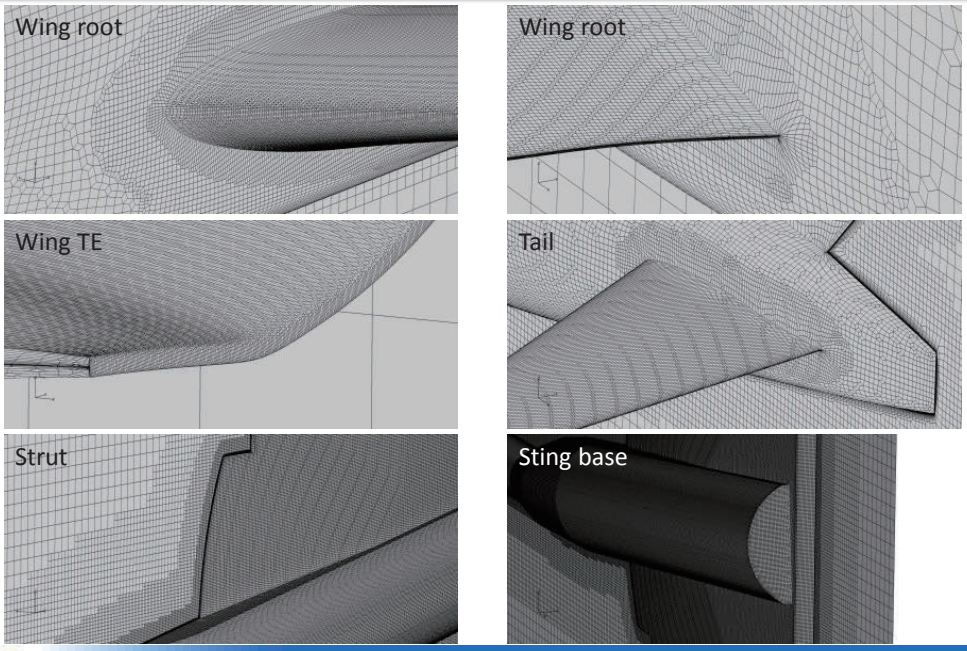


風洞壁込みのCRM解析



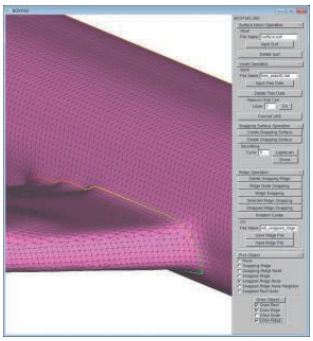
- APCⅢ用に風洞壁込みのCRM解析格子作成
- 作業時間は，STL修正含めて～1日程度

風洞壁込みのCRM解析



現状の課題

- 表面格子が四角形のため，表現できる形状に制約がある．
  - 凹部ではnegative volumeが発生するが，FaSTARの前処理でcell center位置を修正して回避
  - 四角形を三角形に分割して対応．
  - 格子フォーマットの改良



- HexaGridのように完全自動では無い．
  - 形状再現性と作業時間はトレードオフ
- メモリ使用量が大きい．
  - 直交格子自体はLaptopで数千万～数億ボクセル程度生成可能．
  - 直交格子から非構造格子データに落とし込むときにメモリを要する．
  - 現状では表面格子生成と空間格子生成を分けることで対応．

まとめと今後

- BOXFUNの開発状況について報告した.
  - GUIを用いたマニュアル操作による特徴線捕獲の改善
  - BCMフレームワークを用いた並列化

- 今後の予定
  - 自動化アルゴリズムの改良
    - 表面格子を四角形⇒三角形へと分割し，形状表現の自由度を高める
    - 特徴線捕獲の自動化を促進し，ユーザの負担軽減
    - HexaGridライクな使い方への対応
  - リファクタリング
    - 低メモリ化，メモリリーク対応，GUI上でのrefinement boxによる柔軟な細分化，分散並列対応，　etc
  - GUIの整備
    - 操作性向上
    - マニュアルの整備
  - B版配布

ご清聴ありがとうございました.