

OpenFOAM 内の重合格子ソルバの 妥当性についての検証と魚類遊泳解析への適用

澄川太皓 [1]　三好扶 [2]　福江高志 [3†]

[1] 岩手大学大学院　[2] 岩手大学　[3] 金沢工業大学

Validation of overset grid analysis method in OpenFOAM and application to fish swimming analysis.

Hiroaki SUMIKAWA[*]　Tasuku MIYOSHI[**]　Takashi FUKUE[***†]

[*]Graduate School of Iwate University　[**]Iwate University　[***]Kanazawa Institute of Technology

Abstract

It becomes more important that the explication of mechanism of creature's motion, function, and construction to make more efficient products or understand the process of evolution. Although it is difficult to observe actual creature's behavior, most of previous research focused on movement and propulsive performance of fish using computational fluid dynamics (CFD) analysis. A final goal of this study is to develop a method to simulate the large deformation such as fish locomotion using overPimpleDyMFoam which is one of the solvers in OpenFOAM. In this paper we discussed the validity of the proposed solver as compared to the experimental results of the rotating cylinder. Furthermore, we conducted the CFD analysis of the characteristics of the fish swimming by using the combination of the proposed solver and the overset method. It is found that the proposed analytical method using OpenFOAM is effective to investigate the basic flow phenomena around the swimming fishes.

Keywords: CFD analysis, fish swimming, OpenFOAM, overset mesh, overPimpleDyMFoam.

1. はじめに

水生生物の遊泳動作や形状は，生態学や物理学に基づいた数多くの研究結果（e.g.,[1-8]）から，積年の進化の過程でそれぞれの種の生態や生息環境にあわせて最適化された結果と考えられている．したがって，様々な種の遊泳動作や形状の詳細な分析と比較を行うことができれば，進化の過程の探索や，バイオミメティクスの視点に基づく工学応用などが期待できる．

水生生物の遊泳動作の解析方法は大別して，実挙動の観察とその計測，および数値流体解析（Computational Fluid Dynamics; CFD）に分類できる．前者では，例えば Fish ら[1]は，大型水槽にてトレーニングされたイルカの遊泳動作中の尾鰭周囲に発生する渦の挙動を PIV で計測している．後者では，例えば Payne ら[2]は魚類が移動に費やすエネルギの総量（cost of transport; COT）に着目し，シュモクザメが体をロール方向に 30° から 80° に傾けて泳ぐことで COT を抑える効果がある事を明らかにしている．また，Takagi らは，太平洋クロマグロ[3]やヒラメ[4]が，泳ぎ方を変えることで COT の削減につなげていることを明らかにしている．COT だけでなく魚類の身体形状に関しても，数種類の尾鰭形状と推進効率との関係を述べている研究[5-6]，ハコフグの抗力係数が自動車に比べて小さいことを明らかにした研究[7]や，イトマキエイの遊泳時における姿勢制御能を検証した研究[8]などがある．しかしながら我々の知る限り，魚類をトレーニングし実験や計測に適するような遊泳動作を獲得させた事例はなく，現状では魚類の遊泳動作の解析は CFD に頼らざるを得ない．このため CFD に必要となるパラメータサーベイや解析対象のモデル化は，CFD による魚類の遊泳動作解析の有用性を高める上で極めて重要な要素と言える．

魚類の遊泳動作の CFD では，解析対象となる種の遊泳時の挙動と水の流れを連成させた解析が必要となる．特に異なる種の挙動をそれぞれに再現しながら解析し比較するには，網羅的に CFD を実施することが避けられない．したがって CFD に適用する解析コードは，流体解析の信頼性を担保すること，魚類の挙動を再現するための物体の運動を合わせた連成解析を行えること，様々な種の運動を再現したモデルを自由に組み込め

† *fukue@neptune.kanazawa-it.ac.jp*

(Received May 8, 2019; accepted June 20, 2020; published July 1, 2020)

ること，およびパラメータサーベイの条件が広範になることから，計算コストが最適化されることの 4 点が満たされる必要がある．このため，魚類の泳動作の研究では商用コードやインハウスコードが用いられているケースがほとんどであったが，商用コードは特にソースコードがブラックボックス化されていることが多い．したがって種によって挙動が異なり，かつ実験との詳細比較が難しい魚類の泳動作を解析するには，計算手法の客観的評価が難しく，また，プログラムの改変による自由なモデルの組み込みは一般に難しい．これに対し，インハウスコードはコードの内容を研究者が理解している点，また，条件や計算対象に応じ自由な改造ができる点から最も自由度が高い．一方，連成解析に必要な機能をすべてコードに起こすことは至難の業であり，かつその運用は研究室のノウハウに依存する．したがって複数の研究者によって構成される広範なグループでは，計算技術を共有する観点で不向きと言える．

そこで，魚類の多様な生態や生息環境への適応能を体系的に評価し有用な知見を手に入れるために，商用コードやインハウスコードとは異なる新しい CFD のプラットフォームの構築とその普及が必要である．これを実現するため，我々は OpenFOAM をメインフレームにした魚類の遊泳挙動の解析プラットフォームを構築することを目的とした．本稿では特に解析対象の泳動作に伴うメッシュの変形に関して，メッシュ変形を行っても発散しない解析手法を重合格子ソルバによって構築し，解析の妥当性について検証した．

2. OpenFOAM-v1806 の重合格子ソルバを用いた解析手法

2.1 従来法の課題

OpenFOAM による魚類の遊泳解析については，Bhagat ら[9]が Foam-extend の pimpleDyMFoam を用いて魚類のうねり動作を再現しているが，流速分布の結果から計算が発散した可能性が示唆される．これは魚類のうねり動作時に，身体周囲のメッシュの変形と不均一性が強くなったことに起因する．これに対し，メッシュの切り方を変える[10]，リメッシュする[11]などの手法が提案されているが，計算ステップごとのリメッシュは計算負荷が高く，かつステップ間の計算データの不連続性が顕著になる．

メッシュ変形のない手法として，境界埋め込み法[12]があるが，3D モデルを使った計算メッシュを生成する場合に，その表面形状データの再現性はメッシュ解像度に依存して決まる．したがって，魚類泳動作のような刻々と変化する複雑な形状をより正確に再現しながら解析するには計算コストが高くなる．複数個の構造的な格子を重ね合わせて解析を行う重合格子法[13-15]では，例えば Shen ら[15]のように物体の変形を考慮しない剛体の移動に関する研究が多く，魚類の遊泳動作を再現したものはない．しかしながら，重合格子法は魚類の遊泳動作のような大変形を伴う解析が可能であり，かつ，複雑な表面形状データにメッシュを適合させ構成できる snappyHexMesh[16]を併用することにより，埋め込み境界法よりも計算コストを低く抑えられる．

2.2 本研究の手法

本研究では，大変形解析ができ，snappyHexMesh が併用可能である OpenFOAM-v1806 [17]の重合格子ソルバを用い，商用コードを用いず，計算コストが低く，複雑な形状を有する魚類の遊泳動作を CFD 解析できる手法を構築する．

3. 本提案手法の妥当性検証実験

3.1 実験の目的

重合格子ソルバを用いた本提案手法について，剛体の動的な運動により発生する流れの実験結果と比較することで，妥当性を検証することとした．比較対象として，田中ら[18]が報告している回転円柱回りの流れ場の実験値を取り上げた．レイノルズ数，回転レイノルズ数を実験値と同値となるよう設定して，提案手法による流体解析を実施し，計算精度を比較評価した．

3.2 解析モデルおよび計算手法

解析モデルを図 1 に示す．以下，円柱中心を原点とした座標系を用いて説明する．流れ方向を x 軸，円柱高さ方向を y 軸，流れと直行する方向を z 軸としている．

解析領域は幅 3.6 m，高さ 1.8 m，奥行き 8.0 m とした．解析モデルの流入境界から x 軸方向に 2.0 m 移動した位置および z 軸方向の中心の位置に，直径 0.7 m，長さ 0.8 m の回転円柱を設け回転させた．

次にメッシュ生成について説明する．解析モデル全体にベースとして構築する背景メッシュ（図 1 青色）は blockMesh で作成した．さらに，回転円柱の壁面から 0.5 m の範囲に snappyHexMesh で重合格子領域（図 1 赤色，外径 1.7 m，内径 0.7 m，高さ 1.8 m）を作成した．この重合格子領域に，円柱中心を通る y 軸回りの回転を与えることで，円柱の回転により発生する流れを解析した．

解析ソルバは OpenFOAM-v1806[17]の overPimpleDyMFoam を用い，層流の流れ解析を行った．作動流体は

気温 20 °C，密度 1.293 kg/m³，動粘度 1.50×10^{-5} m²/s の乾燥空気とし，解析モデル入口（x = - 2.0 m の yz 境界面）から流入流速 U_0 = 1.995 m/s の一様流速（円柱の直径(0.7m)を代表寸法とした Reynolds 数 = 9.30×10^4）で流入させた．

解析に用いた支配方程式は下記に示す連続の式と Navier-Stokes 方程式である．ただし，u は速度ベクトル，p は静圧（密度で除した値），μ は粘度，ρ は流体（空気）の密度， I は単位テンソルである．

$$\nabla\cdot u = 0 \tag{1}$$

$$\frac{\partial \mathrm{u}}{\partial \mathrm{t}} + \nabla\cdot(uu) = -\nabla p + \nabla\cdot\left(\frac{\mu}{\rho}\nabla u\right) + \nabla\cdot\left[\frac{\mu}{\rho}\left\{(\nabla u)^T - \frac{1}{3}\nabla\cdot uI\right\}\right] \tag{2}$$

3.3 境界条件

解析モデル入口（x = - 2.0 m の yz 境界面）は一様流速の固定流入条件（1.995 m/s），解析モデル出口（x = 6.0 m の yz 境界面）は自由流出条件とした．円柱表面および，そのほかの計算領域の境界面は全て滑りなしの壁境界条件として設定した．なお，重合格子の外側の境界面（/background_blockMesh/0/zoneID 内の overset）における境界条件は，type overset; としている．

3.4 解析条件

解析条件を表 1 および表 2 に示す．表 1 は円柱の回転角速度が 3.14 rad/s（回転 Reynolds 数 = 5.13×10^4）のものを，表 2 は円柱の回転角速度が 2.093 rad/s（回転 Reynolds 数 = 3.42×10^4）のものを纏めたものである．角速度，重合格子の生成法，解析期間が解析精度に与える影響を評価するため，それぞれの条件を変えた下記の 6 ケースについて解析を行い，結果を比較した．

case 1: 回転角速度 3.14 rad/s， 回転レイノルズ数 5.13×10^4，重合格子サイズ 0.03 m，解析期間 10 秒
case 2: 回転角速度 3.14 rad/s， 回転レイノルズ数 5.13×10^4，重合格子サイズ 0.03 m，解析期間 20 秒
case 3: 回転角速度 2.093 rad/s，回転レイノルズ数 3.42×10^4，重合格子サイズ 0.03 m，解析期間 10 秒
case 4: 回転角速度 2.093 rad/s，回転レイノルズ数 3.42×10^4，重合格子サイズ 0.03 m，解析期間 20 秒
case 5: 回転角速度 2.093 rad/s，回転レイノルズ数 3.42×10^4，重合格子サイズ 0.05 m，解析期間 10 秒
case 6: 回転角速度 2.093 rad/s，回転レイノルズ数 3.42×10^4，重合格子サイズ 0.1 m， 解析期間 10 秒

3.5 妥当性の評価方法

解析結果の妥当性は，解析結果より得られた速度分布，渦度分布，解析結果より算出した揚力係数 C_L，抗力係数 C_D を比較することで評価した．なお，C_L，C_D は次の通り定義した．

$$\begin{pmatrix} C_L \\ C_D \end{pmatrix} = \frac{2}{\rho {U_0}^2 A}\begin{pmatrix} L \\ D \end{pmatrix} \tag{3}$$

なお，L は揚力，D は抗力，A は円柱の投影断面積であり，L および D は回転円柱表面の抗力方向および揚力方向の圧力を面積分して算出した．また，C_L，C_D は各ケースにおける解析開始から解析終了までの平均値，最大値，最小値をそれぞれ算出した．

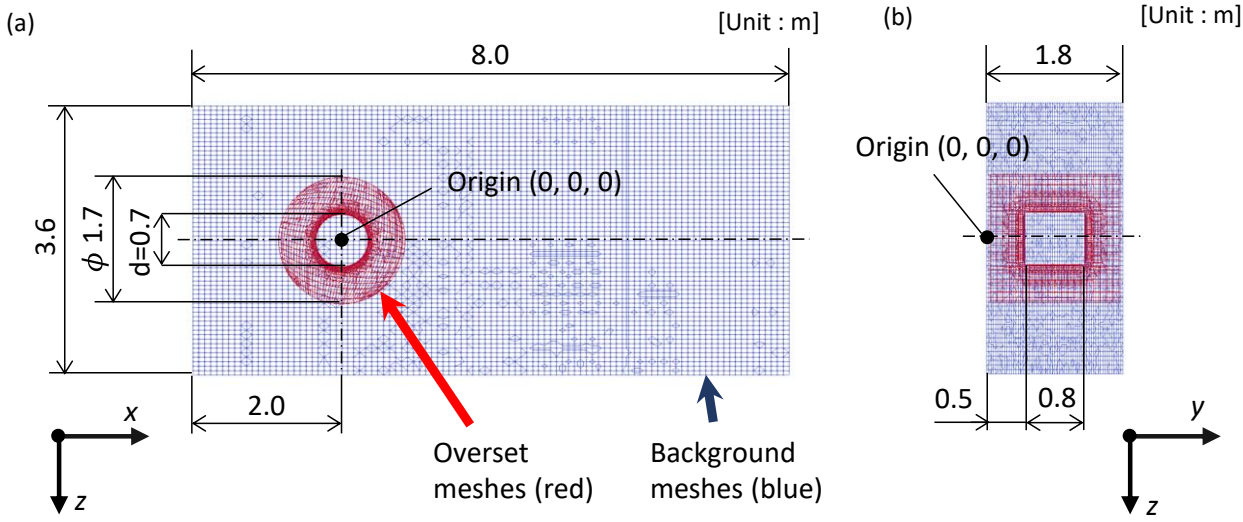

Fig. 1 Overset region and background mesh: (a) is cross section (y = 0.9 m) and (b) is cross section (x = 0 m).

Table 1 Analytical condition for angular velocity ω = 3.14 rad/s.

	case 1	case 2
U_0 : Flow velocity [m/s]	1.995	1.995
N : Revolution [rpm]	30	30
d : Diameter [m]	0.7	0.7
ρ : Density of air [kg/m^3]	1.293	1.293
$\frac{\mu}{\rho}$: Kinematic viscosity of air [m^2/s]	1.5×10^{-5}	1.5×10^{-5}
Reynolds number $\frac{U_0\rho d}{\mu}$	9.30×10^{4}	9.30×10^{4}
Rotational Reynolds number $\frac{\pi\rho d^2 N}{60\mu}$	5.13×10^{4}	5.13×10^{4}
Time [s]	10	20
Grid width [m]	0.03	0.03
Total mesh numbers	2264016	2264016

Table 2 Analytical condition for angular velocity ω = 2.093 rad/s.

	case 3	case 4	case 5	case 6
U_0 : Flow velocity [m/s]	1.995	1.995	1.995	1.995
N : Revolution [rpm]	20	20	20	20
d : Diameter [m]	0.7	0.7	0.7	0.7
ρ : Density of air [kg/m^3]	1.293	1.293	1.293	1.293
$\frac{\mu}{\rho}$: Kinematic viscosity of air [m^2/s]	1.5×10^{-5}	1.5×10^{-5}	1.5×10^{-5}	1.5×10^{-5}
Reynolds number $\frac{U_0\rho d}{\mu}$	9.30×10^{4}	9.30×10^{4}	9.30×10^{4}	9.30×10^{4}
Rotational Reynolds number $\frac{\pi\rho d^2 N}{60\mu}$	3.42×10^{4}	3.42×10^{4}	3.42×10^{4}	3.42×10^{4}
Time [s]	10	20	10	10
Grid width [m]	0.03	0.03	0.05	0.1
Total mesh numbers	2264016	2264016	532082	88408

3.6 解析結果および考察

典型例として case 1 の回転円柱周りにおける速度分布（図 2）および渦度分布（図 3）をそれぞれ示す．図 2 および図 3 から，円柱後方にカルマン渦が形成されることが確認された．また，回転円柱周りおよび重合格子の境界面にて計算結果の不連続性や発散傾向は認められず，安定した解析が達成された．次に，図 4 には case 1 および case 3 の C_L，C_D の時系列変化を示す．C_L について確認すると，解析の初期では揚力が発生せず，一定時間経過後から徐々に増加し，かつ周期的変化が表れることが確認できる．C_D も初期より増加し，ある一定のレベルに到達後，周期的変化が表れることが確認できる．これらの結果は先行研究[19]と傾向が一致しており，特徴を再現することができたと判断できる．

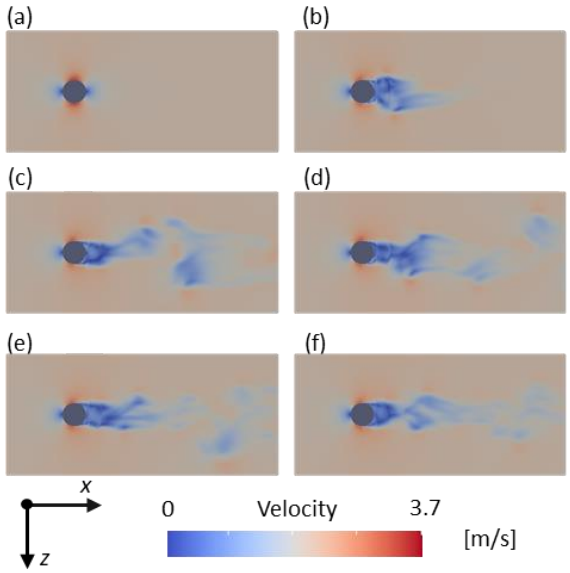

Fig. 2 Velocity distribution (case 1, y = 0.9m): (a) is 0 s, (b) is 2.0 s, (c) is 4.0 s, (d) is 6.0 s, (e) is 8.0 s and (f) is 10 s.

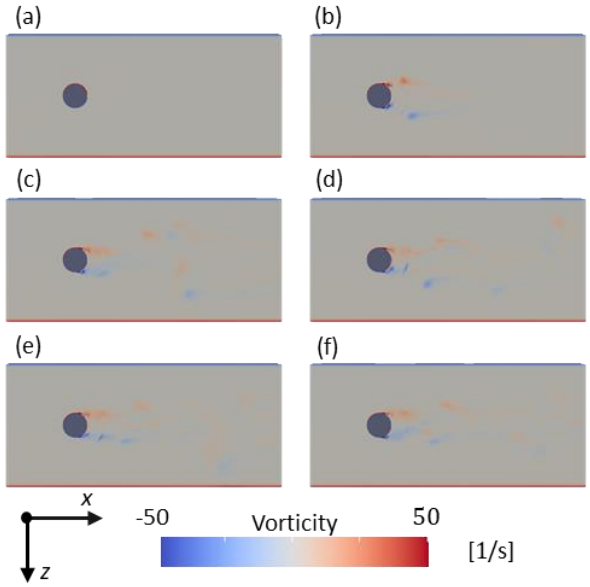

Fig. 3 Vorticity distribution (case 1, y = 0.9 m): (a) is 0 s, (b) is 2.0 s, (c) is 4.0 s, (d) is 6.0 s, (e) is 8.0 s and (f) is 10 s.

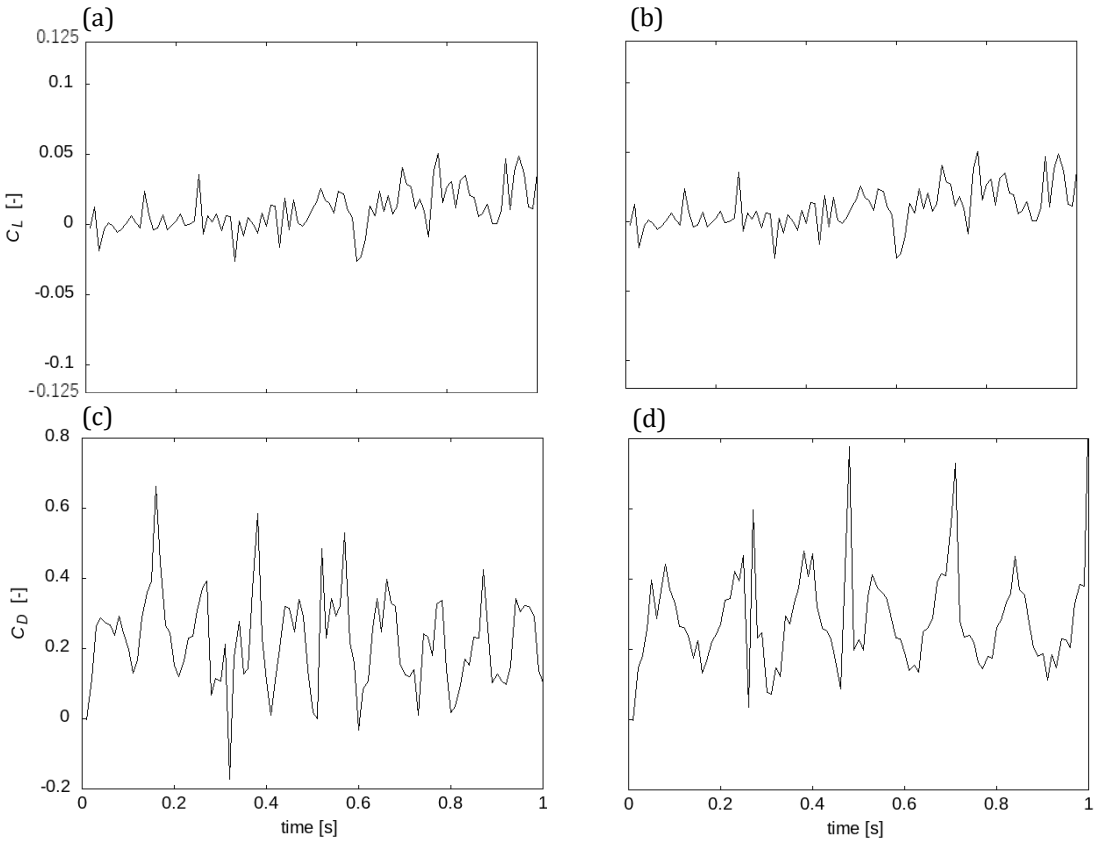

Fig. 4 Time series of Lift and Drag coefficient; (a) is the Lift coefficient of case 1, (b) is the Lift coefficient of case 3, (c) is the Drag coefficient of case 1 and (d) is the Drag coefficient of case 3.

図 5 には角速度 ω = 2.093 rad/s における回転円柱後方（解析領域入口から 2.49 m の位置，z 軸方向の分布）の流速の分布に関する重合格子サイズや解析期間の影響を示す．ここで流速 v は $v=\sqrt{u_x^2+u_y^2+u_z^2}$ (u_x は速度ベクトルuの x 軸方向成分，u_yは同 y 成分，u_zは同 z 成分) として算出している．重合格子サイズが変わると速度分布が変化する傾向を確認できる．重合格子サイズが一番細かい 0.03 mm の結果は，実験結果[18]と同様であった．これと比較すると，重合格子サイズが大きい 0.05 mm あるいは 0.1 mm の結果では，特に回転円柱の中心に近くなると流速分布の傾向まで異なることがわかった．一方，解析期間の長さについては，若干の相違は見られるが，分布のパターンは同様な傾向を示した．以上から，今回の検証範囲では，実験結果[18]との相同性を得られる条件として，重合格子サイズ 0.03 mm は妥当であったと判断できる．一方，解析期間長が与える影響は，重合格子サイズの相違によって生じる影響よりも小さく，解析期間は 10 秒で十分であったと判断できる．

表 3 に，重合格子サイズの差異，解析期間長の差異による C_L の期間平均値，期間最大値，および期間最小値を先行研究[18][19]と比較した結果を示した．また表 4 は C_D について同様に示したものである．

先行研究[18]との比較をすると，C_L，C_D それぞれ，平均値では差異が大きいが，最大値（瞬時値）では平均値ほどの差異はなく，また算出された値の単位も異なる（先行研究[18]では単位長さ当たりの力であり，本稿では無次元数としている）．このため本提案手法での結果との単純な比較は困難であるものの，時系列変動パターンは周期的変化を呈し，解析途中で発散せず安定した解析ができていることから，定性的ではあるものの提案手法の妥当性は確保されたと示唆される．なお，中村ら[19]の 2 次元解析結果も併せて比較検討した．先行研究[19]とは，結果の相違がみられているが，これは先行研究[19]が 2 次元解析の結果であり，また計算領域の寸法やレイノルズ数も異なることにより発生した差であると考えている．

4. 提案手法による魚類遊泳動作解析

4.1 魚類の遊泳動作解析用ライブラリの設定

3 章の内容を踏まえ，OpenFOAM-v1806[17]の重合格子ソルバを用いた魚類の遊泳動作解析プラットフォームを提案する．本提案手法は魚類の遊泳動作解析を目的として開発しており，3 章での回転動作の代わりに魚類泳動作をモーションライブラリとして組み込む必要がある．なお魚類の遊泳動作は，魚類の遊泳動作を表現する式(Cui ら[20])に応じて能動的に魚の形状が大変形するモーションライブラリを作成した．物体名称をマグロとした場合のディレクトリとファイル配置，解析に用いるコマンドなどの詳細は Appendix を参照されたい．

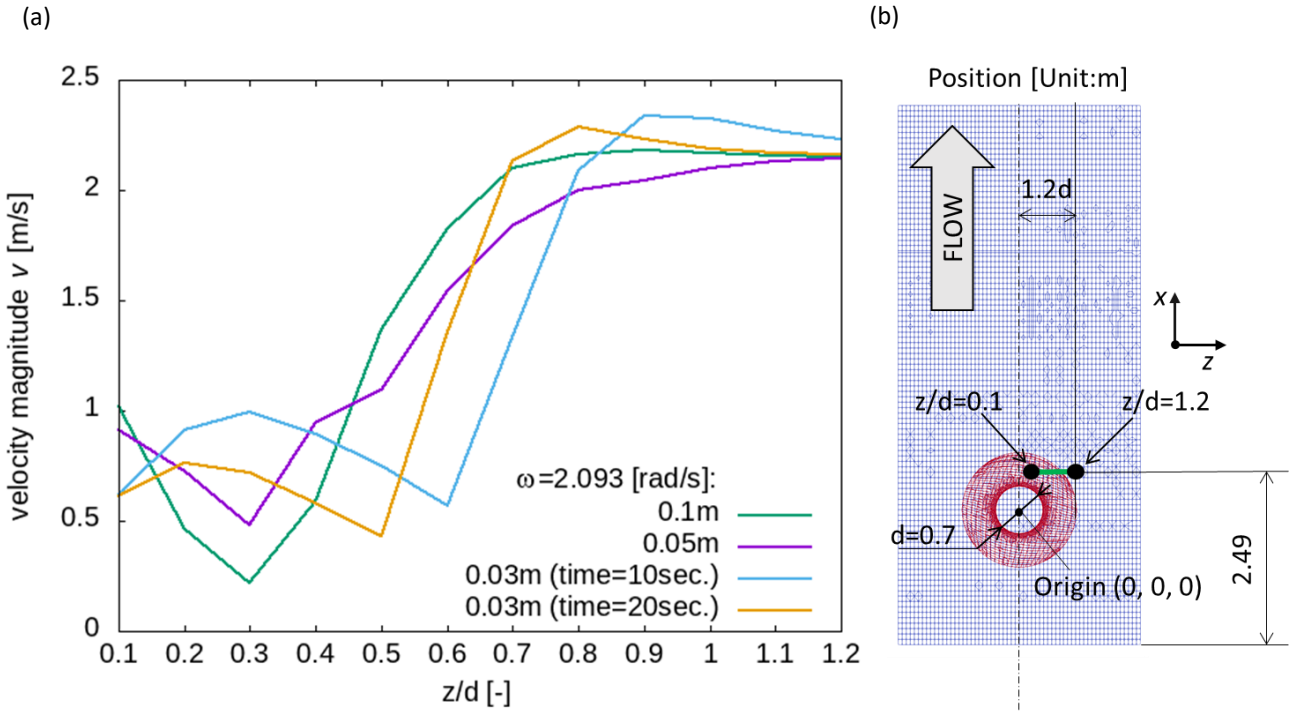

Fig. 5 Distribution of velocity magnitude behind the cylinder:(a) is relationship of velocity magnitude and grid size. (b) is the cross section(y=0.3m) and measurement points. (green line)

Table 3 Comparison of Lift coefficient (C_L) between experimental results and previous results

Grid width [m] (ω = 3.14 rad/s)	**average**	**maximum**	**minimum**
0.03 (10s): case 1	0.00675	0.159675	-0.14042
0.03 (20s): case 2	-0.0097075	0.159675	-0.21209
Reference [18]	0.1		
Reference [19]	-0.436467	-	-
Grid width [m] (ω = 2.093 rad/s)	**average**	**maximum**	**minimum**
0.03 (10s): case 3	0.009693	0.048751	-0.02653
0.03 (20s): case 4	0.015516	0.149626	-0.03222
0.05: case 5	0.00936	0.053318	-0.0454
0.1: case 6	0.00176	0.021401	-0.02161
Reference [18]	0.3		

Table 4 Comparison of Drag coefficient (C_D) between experimental results and previous results

Grid width [m] (ω = 3.14 rad/s)	**average**	**maximum**	**minimum**
0.03 (10s): case 1	0.29658	0.88945	-0.23205
0.03 (20s): case 2	0.253271	1.101015	-0.26175
Reference [18]	0.75		
Reference [19]	1.595024	-	-
Grid width [m] (ω = 2.093 rad/s)	**average**	**maximum**	**minimum**
0.03 (10s): case 3	0.2885	0.77844	-0.00348
0.03 (20s): case 4	0.2702	0.77844	-0.00348
0.05: case 5	0.2001	0.54469	0.00194
0.1: case 6	0.2286	1.09151	0.13758
Reference [18]	0.9		

解析モデルおよび手法を説明する．本稿では魚類泳動作の解析例として *Thunnus*（以下マグロとする）を取り上げた．マグロの 3D モデルは体長 0.69 m，高さ 0.33m，幅 0.17m のもの(図 6)を Blender2.79[22]で作成した．マグロの 3D モデルを幅 1.6 m，高さ 1.2 m，奥行 2.8 m の直方体形状の解析空間内に配置した(図 7)．作動流体は水（動粘度 1.0×10^{-6} m²/s）とし，魚体正面から流速 2.0 m/s で流入させた．流れは層流として解析した．境界条件として，魚体前方の境界面に流速 2.0 m/s の固定流入条件，その他の面を自由流出条件とした．魚類表面は滑りなし壁面とした．

魚類泳動作は *xz* 平面内の運動とし，はじめに 3D モデリングソフトウェアなどを用いて魚類の形状の 3D モデルを作成する．作成した 3D モデルは解析空間内へ配置した．3.2 節と同様に，3D モデル形状周りの重合格子を snappyHexMesh，背景メッシュは blockMesh で作成した．マグロの遊泳動作は，背骨（センターライン）の点の移動を Cui ら[20] の式 (4) により記述し再現した（Sander ら[21], Appendix A.8）．

$$z = (0.178x^2 + 0.0429x + 0.035)\sin\left(12t - \frac{2\pi}{0.7}x\right) \tag{4}$$

ここで x は身体長軸方向の位置座標，z は身体短軸方向の移動量であり泳動作中のうねりを表し，t は時刻である．センターラインの点群の位置座標は，センターラインの移動量に応じて変化させた．計算結果から流速分布，圧力分布，渦度分布を算出した．

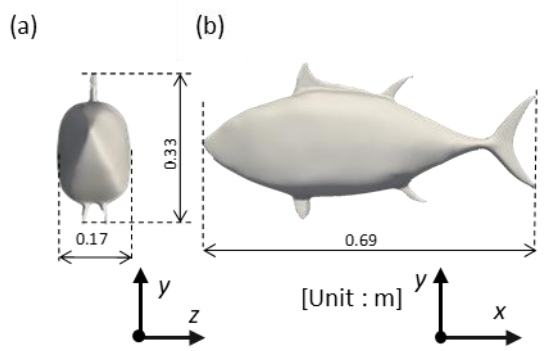

Fig. 6 Analytical model of *Thunnus*: (a) is front view and (b) is side view.

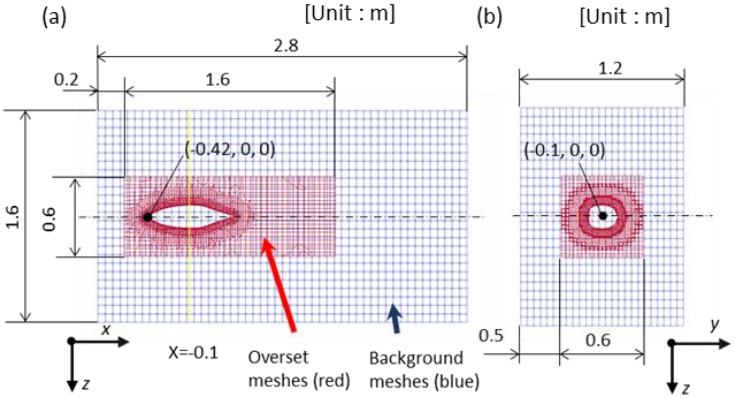

Fig. 7 Overset region and background mesh of *Thunnus*.: (a) is cross section (y = 0.3 m) and (b) is cross section (x = - 0.1 m)

4.2 解析結果と考察

図 8 に本手法を用いた解析で見られたメッシュ変形の様子を示す．また，図 9 に圧力分布を示す．図 8(a), (b)に示すように，マグロの周辺のメッシュに，マグロの泳動差に応じた変形が生じることが確認できた．しかし，図 9(a)，(b)に示す圧力分布の様子から，メッシュが変形したことによる発散は生じず，安定した解析ができた．このことから Bhagat ら[9]の手法では解決できなかったメッシュ変形による発散の問題は，重合格子法を用いた本提案手法によって解決することができた．

図 10 にマグロの遊泳解析の一例として得られた流速分布，圧力分布，渦度分布を示す．Bhagat ら[9]の結果と同様に，マグロの遊泳の時間変化に伴ったうねりの影響を受け，蛇行する後流が確認できた．また，流速分布や圧力分布の不連続性もないことから，安定した解析を行うことができたと判断される．以上のことから，提案した計算手法は定性的ではあるものの魚類の遊泳運動に伴う流れの解析が達成され，他魚種泳動作にも適用できる可能性が示された．

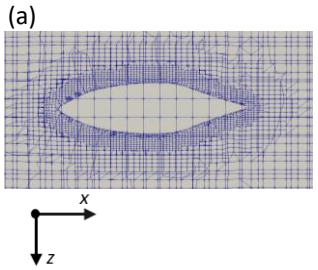

Fig. 8 Mesh details: (a) is the mesh details (t = 0 s, y=0.3m), (b) is the mesh details (t = 1.0 s, y=0.3m).

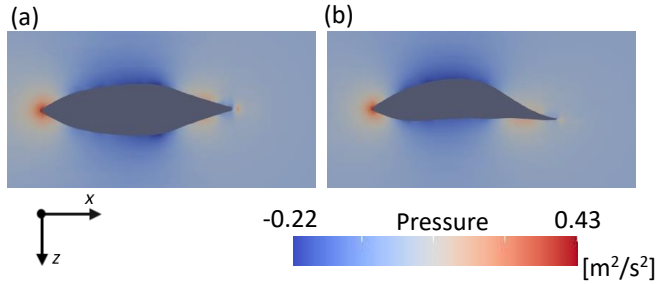

Fig. 9 Pressure distributions: (a) is the pressure distribution (t= 0 s, y=0.3m) and (b) is the pressure distribution (t = 1.0 s, 0.3m)

5. 結論

本稿では，OpenFOAM-v1806[17]に実装される重合格子ソルバを用い，物体の運動を連成させた，魚類の遊泳動作解析プラットフォームを提案した．本手法を回転円柱の実験結果[18-19]と比較することにより提案手法の妥当性を検討した．結果，定性的ではあるもののその特性を呈示できており，提案手法が妥当であったと判断できた．また，マグロを例とした魚類泳動作の CFD として本提案手法を適用し，流速分布，圧力分布，渦度分布が発散せず，提案手法が魚類の遊泳運動に伴う流れ解析にも適用可能であることを示した．

OpenFOAM の重合格子ソルバを用いた本提案手法は，魚類の遊泳運動のみならず，適切なモデリングによって様々な運動を伴う対象に適用可能であり，今後の OpenFOAM の応用範囲の拡大にも貢献する．一方，回転円柱後方でのカルマン渦の生成に伴う C_L, C_D の時系列変化に関する評価や，更なる定量的な信頼性の確保，計算コストの最適化などといった多角的な検証は今後の検討課題である．

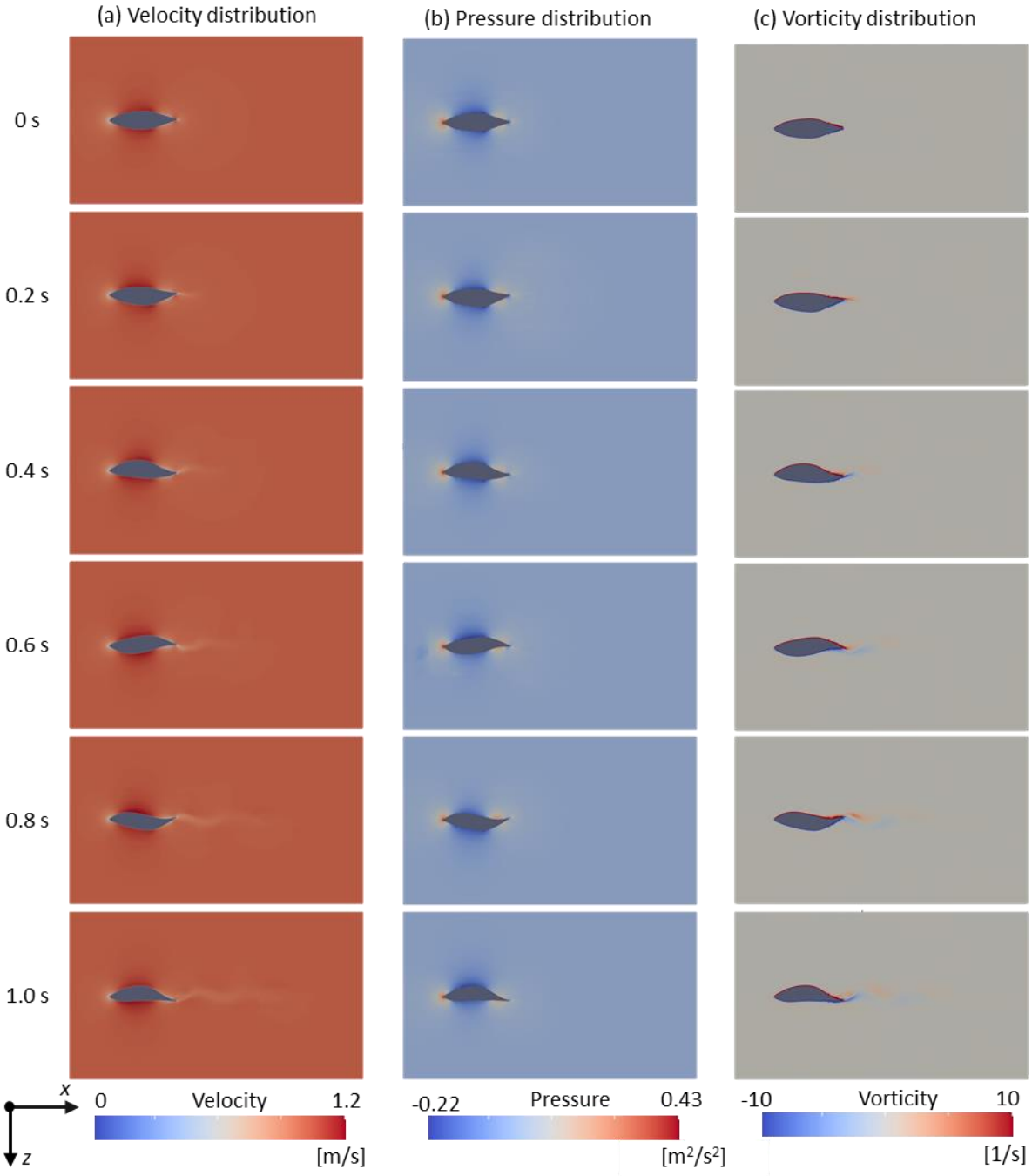

Fig. 10 Typical example in motion of Thunnus: (a) is velocity distribution(y=0.3m), (b) is pressure distribution(y=0.3m), (c) is vorticity distribution(y=0.3m).

参考文献

[1] F. E. Fish, P. Legac, T. M. Williams, T. Wei. Measurement of hydrodynamic force generation by swimming dolphins using bubble DPIV, Journal of Experimental Biology, Vol 217, pp. 252-260, 2014, doi: 10.1242/jeb.087924.

[2] N. L. Payne, G. Iosilevskii, A. Barnett, C. Fischer, R. T. Graham, A. C. Gleiss, Y. Y. Watanabe. Great hammerhead sharks swim on their side to reduce transport costs, Nature communications, Vol 7, Article number 12289, 2016, doi: 10.1038/ncomms12289.

[3] T. Takagi, Y. Tamura, D. Weihs. Hydrodynamics and energy-saving swimming techniques of pacific bluefin tuna, Journal of Theoretical Biology, Vol 336, pp. 158-172, 2013, doi: 10.1016/j.jtbi.2013.07.018.

[4] T. Takagi, R. Kawabe, H. Yoshino, Y. Naito. Functional morphology of the flounder allows stable and efficient gliding: An integrated analysis of swimming behavior, Aquatic Biology, Vol 9, No 2, pp. 149-153, 2010, doi: 10.3354/ab00237.

[5] X. Chang, L. Zhang, X. He. Numerical study of the thunniform mode of fish swimming with different Reynolds number and caudal fin shape, Computers & Fluids, Vol 68, pp. 54-70, 2012, doi:10.1016/j.compfluid.2012.08.004.

[6] H. Sumikawa, T. Fukue, T. Miyoshi. CFD-based visualization of differences in macroscopic flow patterns around several types of caudal fins, The 7th International Symposium on Aero Aqua Bio-Mechanisms, pp. 211-215, Tokyo, 2018.

[7] H. Chowdhury, R. Islam, M. Hussein, M. Zaid, B. Loganathan, F. Alan. Design of an energy efficient car by biomimicry of a boxfish, Energy Procedia, Vol 160, pp. 40-44, 2019, doi: 10.1016/j.egypro.2019.02.116.

[8] T. Shimizu, T. Miyoshi. Postural control in the pitch direction using flexion angles of the root and fin tip of the pectoral fin in *Mobula japanica*, Journal of Aero Aqua Bio-Mechanicsms, Vol 8, No 1, pp. 6-12, 2019, doi: 10.5226/jabmech.8.6.

[9] S. Bhagat. Make a fish swim, http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2017/SahilBhagat/Fish_openfoam.pdf, (accessed 2019-03-25).

[10] Z. Zhao, Q. Xiao, X. Shu, G. Li, H. Liu. A study of zebrafish locomotion using experimental and numerical simulation, The 7th International Symposium on Aero Aqua Bio-Mechanisms, pp. 169-173, Tokyo, 2018.

[11] R. Ramamurti, W. C. Sandberg, R. Löhner, J. A. Walker, M. W. Westneat. Fluid dynamics of flapping aquatic flight in the bird wrasse: three-dimensional unsteady computations with fin, Journal of Experimental Biology Vol 205, pp. 2997-3008, 2002.

[12] Z-Q. Xin, C-J. Wu. Numerical simulations and vorticity dynamics of self-propelled swimming of 3D bionic fish, Science China: Physics, Mechanics and Astronomy, Vol 55, No 2, pp. 272-283, 2012, doi: 10.1007/s11433-011-4603-7.

[13] 加藤由博, 堀之内成明, 科野佐代子. 重合格子法を用いた自動車の床下流れ解析, 豊田中央研究所 R&D レビュー, Vol 32, No 2, pp. 23-33, 1997, https://www.tytlabs.com/japanese/review/rev322pdf/322_023kato.pdf, (accessed 2019-03-25).

[14] I. Borazjani, F. Sotiropoulos. Numerical investigation of the hydrodynamics of carangiform swimming in the transitional and inertial flow, Journal of Experimental Biology, Vol 211, pp. 1541-1558, 2008, doi:10.1242/jeb.015644.

[15] Z. Shen, D. Wan, P. M. Carrica. Dynamic overset grids in OpenFOAM with application to KCS self-propulsion and maneuvering, Ocean Engineering, vol 108, pp. 287-306, 2015, doi:10.1016/j.oceaneng.2015.07.035.

[16] 春日悠,今野雅, OpenFOAM による熱移動と流れの数値解析, 森北出版, 2017, ISBN978-4-627-69101-8.

[17] OpenFOAM. The open source CFD toolbox, 2018, https://www.openfoam.com/releases/openfoam-v1806/, (accessed 2019-03-25).

[18] 田中英穂,永野進. 回転円柱まわりの流れに関する研究, 日本機械学会論文集, Vol 38, No 310, pp. 1343-1352, 1972, https://doi.org/10.1299/kikai1938.38.1343.

[19] 中村佳朗, 賈為, 水野大介. 任意に運動する物体周りの非圧縮性流れの計算法, 日本航空宇宙学会誌, Vol 41, No 469, pp. 96-103, 1993, doi:10.2322/jjsass1969.41.96.

[20] Z. Cui, Z. Yang, L. Shen, H. Z. Jiang, Complex modal analysis of the movements of swimming fish propelled by body and/or caudal fin, journal of Wave Motion, vol 78, pp. 83-97, 2018, doi:10.1016/j.wavemoti.2018.01.001.

[21] A. Sander, E. Stamhuis, A. Baars. Trout swimming: Thrust and efficiency from a numerical perspective, doi:10.13140/RG.2.2.15187.27681.

[22] blender, https://www.blender.org/, (accessed 2019-11-19).

Appendix

本解析で使用したファイルの一部を示す．簡単な説明を各ファイル上部に記載した．各ファイルの詳しい使い方等は ESI のホームページ(https://www.openfoam.com/)やユーザーガイド(https://www.openfoam.com/documentation/)等を参照していただきたい．(取得日 2019 年 11 月 15 日)

A.1 createPachDict (entyu_snappyHexMesh/system)

```
// パッチ・フェイスセットからの境界面の作成
pointSync false;        //Do a synchronization of coupled points after creation of any patches.
Patches
(
    {
        name oversetPatch;
        patchInfo
        {
            type overset;
        }

        constructFrom patches;

        patches (inlet outlet fixedWalls);

        set f0;
    }
    {

        name wing;                                  //name of patch

        patchInfo
        {
            type wall;                              //decide the boundary condition
        }

        constructFrom patches;                      //select cylinder parts patch or set

        patches (wing);                             //decide the name of patch

        set f0;
    }
);
```

A.2 ディレクトリとファイル配置

```
maguro_snappyHexMesh
├── constant
│   ├── polyMesh
│   │   ├── boundary
│   │   ├── cellLevel
│   │   ├── cellZones
│   │   ├── faceZones
│   │   ├── faces
│   │   ├── level0Edge
│   │   ├── neighbor
│   │   ├── owner
│   │   ├── pointLevel
│   │   ├── pointZones
│   │   ├── points
│   │   └── surfaceIndex
│   └── triSurface
│       └── wing_5degrees.obj
└── system
    ├── blockMeshDict
    ├── controlDict
    ├── createPatchDict
    ├── decomposeParDict
    ├── fvSchemes
    ├── fvSolution
    └── snappyHexMeshDict
```

(a)maguro_snappyHexMesh

```
background_blockMesh
├── 0.orig
│   ├── U
│   ├── p
│   ├── pointMotionU
│   └── zoneID
├── constant
│   ├── RASProperties
│   ├── dynamicMeshDict
│   ├── polyMesh
│   │   ├── boundary
│   │   ├── cellZones
│   │   ├── faceZones
│   │   ├── faces
│   │   ├── neighbour
│   │   ├── owner
│   │   ├── pointZones
│   │   ├── points
│   │   └── sets
│   │       ├── c0
│   │       └── c1
│   ├── transportProperties
│   └── turbulenceProperties
└── system
    ├── blockMeshDict
    ├── controlDict
    ├── decomposeParDict
    ├── fvSchemes
    ├── fvSolution
    ├── sampleDict
    ├── setFieldsDict
    └── topoSetDict
```

(b)background_blockMesh.

A.3 本手法におけるコマンドライン

background_blockMesh ディレクトリでは 4 つのコマンドを，maguro_snappyHexMesh では 3 つのコマンドを用いて作業する．背景メッシュは blockMesh を用いて，物体周りのメッシュは snappyHexMesh を用いて作成する．

```
/maguro_snappyHexMesh
    blockMesh
    snappyHexMesh -overwrite
    createPatch -overwrite
```

```
/background_blockMesh
    blockMesh
    mergeMeshes . ../entyu_snappyHexMesh -overwrite
    topoSet
    setFIelds
```

A.4 dynamicMeshDict (background_blockMesh/constant)

本手法では fvMotionSolvers の velocitylapracian を用いて A.8 に示すモーションプログラムを動作させる．

```
// モーションパターンの選択
dynamicOversetFvMeshCoeffs
{
}
```

```
motionSolverLibs ( "libfvMotionSolvers.so" );

solver                velocityLaplacian;

velocityLaplacianCoeffs
{
        diffusivity quadratic inverseDistance 1 (wing);
}
```

A.5 setFieldsDict (background_blockMesh/system)

場合分けした重合格子領域とその他の領域に値を設定する.

```
// セットの作成
defaultFieldValues
(
    volScalarFieldValue zoneID 123
);
regions
(

    cellToCell
    {
        set c0;
        fieldValues
        (
            volScalarFieldValue zoneID 0
        );
    }
    cellToCell
    {
        set c1;
        fieldValues
        (
            volScalarFieldValue zoneID 1
        );
    }
);
```

A.6 zoneID (background_blockMesh/0)

重合格子領域とその他の領域の場合分けを行う.

```
// zoneID の設定
actions
(
    {
        name       c0;
        type       cellSet;
        action     new;
        source     regionToCell;
        sourceInfo
        {
            insidePoints ((0.001 0.001 0.001));
        }
    }
```

```
    {
        name       c1;
        type       cellSet;
        action     new;
        source     cellToCell;
        sourceInfo
        {
            set c0;
        }
    }
    {
        name       c1;
        type       cellSet;
        action     invert;
    }
);
```

A.7 topoSetDict (background_blockmesh/system)

```
//セットの作成
actions
(
    {
        name       c0;                                  //name
        type       cellSet;                             //type
        action     new;                                 //make new one
        source     regionToCell;                        //make cell from region
        sourceInfo
        {
            insidePoints ((0.001 0.001 0.001));         //coodinates in the area
        }
    }
    {
        name       c1;
        type       cellSet;
        action     new;
        source     cellToCell;
        sourceInfo
        {
            set c0;
        }
    }
    {
        name       c1;
        type       cellSet;
        action     invert;
    }
);
```

A.8 fishLocomotionPointPatchVectorField.C

```
// 魚類の遊泳動作式のプログラム
namespace Foam
{
//*********************************************************************
// ①y 軸,z 軸方向のパラメータの設定*******************************
//*********************************************************************
    fishLocomotionPointPatchVectorField::
    fishLocomotionPointPatchVectorField
    (
        const pointPatch& p,
        const DimensionedField<vector, pointMesh>& iF
    )
    :
    fixedValuePointPatchField<vector>(p, iF),
    p0_(p.localPoints()),
    origin_(vector::zero),
    dX_(0.0),
    omega1_(0.0),
    omega2_(0.0),
    length_(0.0),
    waveLength1_(0.0),
    waveLength2_(0.0),
    A_(0.0),
    B_(0.0),
    C_(0.0),
    D_(0.0),
    a_(0.0),
    b_(0.0),
    c_(0.0),
    d_(0.0),
    e_(0.0),
    f_(0.0),
    g_(0.0),
    h_(0.0)
    {}
    fishLocomotionPointPatchVectorField::
    fishLocomotionPointPatchVectorField
    (
        const pointPatch& p,
        const DimensionedField<vector, pointMesh>& iF,
        const dictionary& dict
    )
    :
    fixedValuePointPatchField<vector>(p, iF, dict),
    origin_(dict.lookup("origin")),
    dX_(readScalar(dict.lookup("dX"))),
    omega1_(readScalar(dict.lookup("omega1"))),
    omega2_(readScalar(dict.lookup("omega2"))),
    length_(readScalar(dict.lookup("length"))),
    waveLength1_(readScalar(dict.lookup("waveLength1"))),
    waveLength2_(readScalar(dict.lookup("waveLength2"))),
    A_(readScalar(dict.lookup("A"))),
    B_(readScalar(dict.lookup("B"))),
```

```
    C_(readScalar(dict.lookup("C"))),
    D_(readScalar(dict.lookup("D"))),
    a_(readScalar(dict.lookup("a"))),
    b_(readScalar(dict.lookup("b"))),
    c_(readScalar(dict.lookup("c"))),
    d_(readScalar(dict.lookup("d"))),
    e_(readScalar(dict.lookup("e"))),
    f_(readScalar(dict.lookup("f"))),
    g_(readScalar(dict.lookup("g"))),
    h_(readScalar(dict.lookup("h")))
    {
        if (!dict.found("value"))
        {
            updateCoeffs();
        }
        if (dict.found("p0"))
        {
            p0_ = vectorField("p0", dict , p.size());
        }
        else
        {
            p0_ = p.localPoints();
        }
    }
    fishLocomotionPointPatchVectorField::
    fishLocomotionPointPatchVectorField
    (
        const fishLocomotionPointPatchVectorField& ptf,
        const pointPatch& p,
        const DimensionedField<vector, pointMesh>& iF,
        const pointPatchFieldMapper& mapper
    )
    :
    fixedValuePointPatchField<vector>(ptf, p, iF, mapper),
    p0_(ptf.p0_),
    origin_(ptf.origin_),
    dX_(ptf.dX_),
    omega1_(ptf.omega1_),
    omega2_(ptf.omega2_),
    length_(ptf.length_),
    waveLength1_(ptf.waveLength1_),
    waveLength2_(ptf.waveLength2_),
    A_(ptf.A_),
    B_(ptf.B_),
    C_(ptf.C_),
    D_(ptf.D_),
    a_(ptf.a_),
    b_(ptf.b_),
    c_(ptf.c_),
    d_(ptf.d_),
    e_(ptf.e_),
    f_(ptf.f_),
    g_(ptf.g_),
    h_(ptf.h_)
    {}
```

```
    fishLocomotionPointPatchVectorField::
    fishLocomotionPointPatchVectorField
    (
        const fishLocomotionPointPatchVectorField& ptf,
        const DimensionedField<vector, pointMesh>& iF
    )
    :
    fixedValuePointPatchField<vector>(ptf, iF),
    p0_(ptf.p0_),
    origin_(ptf.origin_),
    dX_(ptf.dX_),
    omega1_(ptf.omega1_),
    omega2_(ptf.omega2_),
    length_(ptf.length_),
    waveLength1_(ptf.waveLength1_),
    waveLength2_(ptf.waveLength2_),
    A_(ptf.A_),
    B_(ptf.B_),
    C_(ptf.C_),
    D_(ptf.D_),
    a_(ptf.a_),
    b_(ptf.b_),
    c_(ptf.c_),
    d_(ptf.d_),
    e_(ptf.e_),
    f_(ptf.f_),
    g_(ptf.g_),
    h_(ptf.h_)
    {}
    void fishLocomotionPointPatchVectorField::updateCoeffs()
    {
        if (this->updated())
        {
            return;
        }
        const polyMesh& mesh = this->internalField().mesh()();
        const Time& t = mesh.time();
        Info << "Using amplitude function A = " << a_ << "*x^2 + " << b_ << "*x + " << c_ << endl;
        vectorField pointPatchOriginal = p0_ - origin_;
        vectorField pointPatchDisplacement(pointPatchOriginal.size(), vector(0,0,0));
        vector pointOriginal;
        vector pointDisplacement = vector(0,0,0);
        scalar deltaXsum = 0;
        scalar phi1 = 2 * 3.1415 /( waveLength1_*length_);
        scalar phi2 = 2 * 3.1415 /( waveLength2_*length_);
        Info << "Ratio body length / wave length1 = " << length_/waveLength1_ << endl;
        Info << "Ratio body length / wave length2 = " << length_/waveLength2_ << endl;
        int nPoints;
//*********************************************************************
// ②センターラインの移動 ******************************
//*********************************************************************
        nPoints = length_ / dX_+1;
        Info << nPoints << endl;
        vectorField centerLineOriginal(nPoints,vector(0,0,0));
        vectorField centerLineDeformed(nPoints,vector(0,0,0));
```

```
        vectorField centerLineDisplacement(nPoints,vector(0,0,0));
        vectorField centerLineRelativeDisplacement(nPoints,vector(0,0,0));
        vectorField centerLineRotation(nPoints,vector(0,0,0));
        forAll (centerLineOriginal,i1)
        {
            centerLineOriginal[i1] = vector(i1*dX_,0,0);
            centerLineDeformed[i1] = vector(i1*dX_,0,0);
            centerLineRelativeDisplacement[i1] = vector(i1*dX_,0,0);
        };
        forAll (centerLineOriginal,i2)
        {
            centerLineDeformed[i2][1] = sin(B_*( omega1_ * t.value() - g_*phi1 * centerLineDeformed[i2][0])) *A_* (a_ *
pow(centerLineDeformed[i2][0], 2) + b_*centerLineDeformed[i2][0] + c_);
            centerLineDeformed[i2][2] = sin(D_*( omega2_ * t.value() - h_*phi2 * centerLineDeformed[i2][0])) *C_* (d_ *
pow(centerLineDeformed[i2][0], 2) + e_*centerLineDeformed[i2][0] + f_) ;
            if ( i2 != 0 )
            {
                centerLineRelativeDisplacement[i2][1] = centerLineDeformed[i2][1] - centerLineDeformed[i2-1][1];
                centerLineRelativeDisplacement[i2][2] = centerLineDeformed[i2][2] - centerLineDeformed[i2-1][2];
                if((centerLineDeformed[i2][1]*centerLineDeformed[i2][1]-centerLineDeformed[i2-1][1]*centerLineDeformed[i2-1][1]) >
0)
                {
                    centerLineRelativeDisplacement[i2][0] = dX_ - sqrt( mag( pow( dX_, 2) - pow( centerLineRelativeDisplacement[i2][1],
2)));
                    deltaXsum = centerLineRelativeDisplacement[i2][0];
                    centerLineDeformed[i2][0] -= deltaXsum;
                }
                if((centerLineDeformed[i2][1]*centerLineDeformed[i2][1]-centerLineDeformed[i2-1][1]*centerLineDeformed[i2-1][1]) <
0)
                {
                    centerLineRelativeDisplacement[i2][0] = -dX_ + sqrt( mag( pow( dX_, 2) + pow( centerLineRelativeDisplacement[i2][1],
2)));
                    deltaXsum = centerLineRelativeDisplacement[i2][0];
                    centerLineDeformed[i2][0] += deltaXsum;
                }
                if((centerLineDeformed[i2][2]*centerLineDeformed[i2][2]-centerLineDeformed[i2-1][2]*centerLineDeformed[i2-1][2]) >
0)
                {
                    centerLineRelativeDisplacement[i2][0] = dX_ - sqrt( mag( pow( dX_, 2) - pow( centerLineRelativeDisplacement[i2][2],
2)));
                    deltaXsum = centerLineRelativeDisplacement[i2][0];
                    centerLineDeformed[i2][0] -= deltaXsum;
                }
                if((centerLineDeformed[i2][2]*centerLineDeformed[i2][2]-centerLineDeformed[i2-1][2]*centerLineDeformed[i2-1][2]) <
0)
                {
                    centerLineRelativeDisplacement[i2][0] = -dX_ + sqrt( mag( pow( dX_, 2) + pow( centerLineRelativeDisplacement[i2][2],
2)));
                    deltaXsum = centerLineRelativeDisplacement[i2][0];
                    centerLineDeformed[i2][0] += deltaXsum;
                }
            }
        }
        centerLineDisplacement = centerLineDeformed - centerLineOriginal;
        forAll(centerLineDeformed, i3)
```

```
        {
            vector currentPoint = centerLineDeformed[i3];
            vector previousPoint = centerLineDeformed[i3-1];
            vector nextPoint = centerLineDeformed[i3+1];
            scalar currentAngle;
            scalar previousAngle;
            if(i3 != centerLineDeformed.size())
            {
                currentAngle = atan( ( nextPoint[1] - currentPoint[1] ) / ( nextPoint[0] - currentPoint[0] ));
            }
            else
            {
                currentAngle = 0;
            }
            if(i3 != 0)
            {
                previousAngle = atan( ( currentPoint[1] - previousPoint[1] ) / ( currentPoint[0] - previousPoint[0] ));
                centerLineRotation[i3][1] = previousAngle - currentAngle;
            }
            else
            {
                previousAngle = 0;
            }
            centerLineRotation[i3][1] = (previousAngle + currentAngle)/2;
            scalar currentAngleZ;
            scalar previousAngleZ;
            if(i3 != centerLineDeformed.size())
            {
                currentAngleZ = atan( ( nextPoint[2] - currentPoint[2] ) / ( nextPoint[0] - currentPoint[0] ));
            }
            else
            {
                currentAngleZ = 0;
            }
            if(i3 != 0)
            {
                previousAngleZ = atan( ( currentPoint[2] - previousPoint[2] ) / ( currentPoint[0] - previousPoint[0] ));
                centerLineRotation[i3][2] = previousAngleZ - currentAngleZ;
            }
            else
            {
                previousAngleZ = 0;
            }
            centerLineRotation[i3][2] = (previousAngleZ + currentAngleZ)/2;
        };
//*****************************************************************************
// ③センターライン周辺の点の移動************************
//*****************************************************************************
        forAll(pointPatchOriginal,i4)
        {
            vector pointOriginal = pointPatchOriginal[i4];
            vector pointOriginalRel;
            vector translation;
            vector rotationAngle;
            vector rotation = vector(0,0,0);
```

```
            vector rotationTranslation = vector(0,0,0);
            forAll(centerLineOriginal, i5)
            {
                vector centerLinePoint0 = centerLineOriginal[i5];
                vector centerLinePoint1;
                if (i5 == centerLineOriginal.size())
                {
                    centerLinePoint1 = centerLineOriginal[i5];
                }
                else
                {
                    centerLinePoint1 = centerLineOriginal[i5+1];
                }
                if( pointOriginal[0] >= centerLinePoint0[0] && pointOriginal[0] < centerLinePoint1[0] && fabs(pointOriginal[0]
- centerLinePoint0[0]) > SMALL && fabs(pointOriginal[0] - centerLinePoint1[0]) > SMALL )
                {
                    scalar position = (pointOriginal[0] - centerLinePoint0[0]) / dX_;
                    vector translationSlope = (centerLineDisplacement[i5+1] - centerLineDisplacement[i5]);
                    translation = centerLineDisplacement[i5] + translationSlope * position;
                    vector rotationAngleSlope = (centerLineRotation[i5+1] - centerLineRotation[i5]);
                    rotationAngle = centerLineRotation[i5] + rotationAngleSlope * position;
                    pointOriginalRel = pointOriginal - vector (pointOriginal[0],0, 0);
                    rotation[0] = ( pointOriginalRel[0] * cos(rotationAngle[1])) - (pointOriginalRel[1] * sin(rotationAngle[1])) -
pointOriginalRel[0];
                    rotation[1] = (pointOriginalRel[0] * sin(rotationAngle[1])) + (pointOriginalRel[1] * cos(rotationAngle[1])) -
pointOriginalRel[1];
                    rotation[2] = (pointOriginalRel[0] * sin(rotationAngle[2])) + (pointOriginalRel[2] * cos(rotationAngle[2]) ) -
pointOriginalRel[2];
                    rotationTranslation = translation + rotation;
                }
                else if (fabs(pointOriginal[0] - centerLinePoint0[0]) <= SMALL && centerLinePoint0 != centerLinePoint1)
                {
                    translation = centerLineDisplacement[i5];
                    rotationAngle[1] = centerLineRotation[i5][1];
                    rotationAngle[2] = centerLineRotation[i5][2];
                    pointOriginalRel = pointOriginal - vector (pointOriginal[0], 0,0);
                    rotation[0] = (pointOriginalRel[0] * cos(rotationAngle[1])) - (pointOriginalRel[1] * sin(rotationAngle[1])) -
pointOriginalRel[0];
                    rotation[1] = (pointOriginalRel[0] * sin(rotationAngle[1])) + (pointOriginalRel[1] * cos(rotationAngle[1])) -
pointOriginalRel[1];
                    rotation[2] = (pointOriginalRel[0] * sin(rotationAngle[2])) + (pointOriginalRel[2] * cos(rotationAngle[2])) -
pointOriginalRel[2];
                    rotationTranslation = translation + rotation;
                }
                else if ( fabs(pointOriginal[0] - centerLinePoint0[0]) <= SMALL && centerLinePoint0 == centerLinePoint1)
                {
                    translation = centerLineDisplacement[i5];
                    Info << "Last CLP met. Trans: " << translation << endl;
                    rotationTranslation = translation;
                 }
            };
            pointPatchDisplacement[i4]=rotationTranslation;
        };
        if (t.value() < 1)              pointPatchDisplacement *= -pow( (t.value()-1), 2) + 1;
        vectorField::operator=
```

```
        (
            4*pointPatchDisplacement
        );
        fixedValuePointPatchField<vector>::updateCoeffs();
    }
    void fishLocomotionPointPatchVectorField::write
    (
        Ostream& os
    ) const
//*********************************************************************
// ④y 軸,z 軸方向のパラメータ読み込みの設定*******************************
//*********************************************************************
    {
        pointPatchField<vector>::write(os);
        os.writeKeyword("origin") << origin_ << token::END_STATEMENT << nl;
        os.writeKeyword("dX")       << dX << token::END_STATEMENT << nl;
        os.writeKeyword("omega1") << omega1_ << token::END_STATEMENT << nl;
        os.writeKeyword("omega2") << omega2_ << token::END_STATEMENT << nl;
        os.writeKeyword("length") << length_ << token::END_STATEMENT << nl;
        os.writeKeyword("waveLength1") << waveLength1_ << token::END_STATEMENT << nl;
        os.writeKeyword("waveLength2") << waveLength2_ << token::END_STATEMENT << nl;
        os.writeKeyword("A")        << A_ << token::END_STATEMENT << nl;
        os.writeKeyword("B")        << B_ << token::END_STATEMENT << nl;
        os.writeKeyword("C")        << C_ << token::END_STATEMENT << nl;
        os.writeKeyword("D")        << D_ << token::END_STATEMENT << nl;
        os.writeKeyword("a")        << a_ << token::END_STATEMENT << nl;
        os.writeKeyword("b")        << b_ << token::END_STATEMENT << nl;
        os.writeKeyword("c")        << c_ << token::END_STATEMENT << nl;
        os.writeKeyword("d")        << d_ << token::END_STATEMENT << nl;
        os.writeKeyword("e")        << e_ << token::END_STATEMENT << nl;
        os.writeKeyword("f")        << f_ << token::END_STATEMENT << nl;
        os.writeKeyword("g")        << g_ << token::END_STATEMENT << nl;
        os.writeKeyword("h")        << h_ << token::END_STATEMENT << nl;
        writeEntry("value", os);
    }
// ***************  ******************* //
    makePointPatchTypeField
    (
        pointPatchVectorField,
        fishLocomotionPointPatchVectorField
    );
// ************************************ //
} // End namespace Foam
// ************************************************************************ //
```