1　定数書き換え

vismo_const.F90に設置できるカメラ数の最大値など、いろいろな定数が定義されているので、必要に応じて書き換える。下記に一例を記す。

1.1　integer, parameter :: vsmImgKind = 2 ! 0:BMP, 1:PPM, 2:PNG：出力画像形式。0,1の時は-lpng も png4vismo.c のコンパイルも必要ない

1.2　integer, parameter :: vsmMaxCamera = 256：カメラの最大数

1.3　integer, parameter :: vsmMaxVisKind = 15：設定できる可視化の最大数

2　コンパイル

Fortran コンパイラおよび MPI ライブライが必要です。

2.1　コンパイルオプション

- 使用している Fortran コンパイラで isnan が使用できない場合は、コンパイルオプションに、-DNO_ISNAN を加える。
- real のデータを可視化したい場合、-DFLOAT (後述の vismo__AddScalar や vismo__AddVector の scalar, vectx などが、real の配列を受け入れるようになる)
- PPM や BMP がうまく出力されない場合、visimo_image.F90内の vsmWriteImageBMP や PPM の open 文を環境に合わせて書き換える。あるいはコンパイル時に-DCBMPPPM をつけて、画像出力部分を C 言語で記述した関数(bmpppm4vismo.c)に置き換える。
- PNG 出力するときは、-DCPNG オプションと vismo_const.F90の vsmImgKind を2にする。
- 矢印、流線の描画がどうも遅い場合、Fortran の transfer 関数が足を引っ張っている可能性がある(Fujitus FX100など)。-DCTRANS で、C 言語の関数(transfer4vismo.c)に置き換える。
- MPI 並列していないコードとの結合の場合は、-DSERIAL が必要です。(カーテシアン版のみ)
- C 言語とのコードとの結合の場合は、-DCLANG が必要です。cvismo.c のコンパイルも必要。また、gfortran の場合-DUSE_GFORT、intel の場合-DUSE_IFORT が必要。そのほかのコンパイラでも cvismo.c を書き換えることで対応可能。

2.2　Makefile

通常の場合は Makefile と test8.f90、C 言語のコードとの結合は MakefileC と test8c.c、MPI 並列しない場合は MakefileS と test1.f90, test1c.c、を参考にしてください。

3　シミュレーションコードとの結合

3.1　Fortran のコードとの結合(MPI 並列の場合)

test8.f90や test8rectl.f90、Makefile を参考にしてください。

まず、プログラムの最初のところで

```
use vismo
```

と記述する。

下記のサブルーチンをシミュレーションコードの適切な場所からコールする。

```
call vismo__init(mpi_comm_world, myrank, psize, "configtest.vsm")
```

vismo のイニシャライズ。引数は、MPI のコミュニケータ、プロセスのランク、プロセスサイズ（数)、vismo のビジュアリゼーションのコンフィグファイル名。

call vismo__addScalar(scalar, LXSIZE, LYSIZE, LZSIZE)
call vismo__addVector(vectx, vecty, vectz, LXSIZE, LYSIZE, LZSIZE)

scalar, vectx, vecty,vectz は可視化するデータの配列。LXSIZE, LYSIZE, LZSIZE はデータの x,y,z 方向の大きさ。
あるいは

call vismo__addScalar(scalar, SIZE)
call vismo__addVector(vectx, vecty, vectz, SIZE)
この場合 SIZE は integer, dimension(3)で SIZE=[LXSIZE, LYSIZE, LZSIZE]

call vismo__addPtcl(p_x, p_y, p_z, q, n)
or
call vismo__addPtclCol(p_x, p_y, p_z, q, scal, n) :: カラーマップを使う場合
可視化する変数の登録。変数の配列は、target 属性が必要。
q は0だと描画、それ以外だと描画しない 。n は粒子の数
scal は particle の値の属性で、その値と球用のカラーマップを使って色を決める。

n = 5として、 配列 q(integer 型)の中身が、
0 0 0 1 0
ならば、４番目の球は描かれません。
配列 scal(倍精度実数)の中身は、0.0 0.1 0.2 0.3 0.4な感じで、 この値から各球の色を決めます。 p_x, p_y, p_z は倍精度実数の配列で、各球の座標です(x,y,z)

call vismo__initCoords(xxx) ! xxx = vismo__uniform or vismo__rect

YinYang 版では
call vismo__initCoords(xx, rad) ! xx = vismo__Yin or vismo_Yang
! rad(1):min radius, rad(2):max radius, rad(3): dr

call vismo__setUniCoord(n, corner, dx)
call vismo__setLocalUniCoord(ln, lx0, rln, loccorner, dx)
座標の登録。詳しくは、下の方で。

call vismo__preparevis
vismo の準備

call vismo__visualization(time) ! arg is timestep (integer)
画像を生成したいところで、コールしてください。引数は時間スタンプに使います。整数です。

call vismo__finalize
最後に呼ぶ。Vismo 内部で動的に確保したメモリなどを解放する。

coord 関係は makeCrdData を見てもらえるといいです。

○vismo__setUniCoord
call vismo__setUniCoord(n, corner, dx)
n: x,y,z 方向のグリッドサイズ（データ全体、のりしろ重複カウントせず）
corner: 原点の位置
dx: x,y,z 方向の刻み幅です。（等間隔なので）

○vismo__setLocalUniCoord
call vismo__setLocalUniCoord(ln, lx0, rln, loccorner, dx)
これは自プロセスが担当するローカルな領域の座標情報です。
ln: x,y,z 方向のグリッドサイズ(のりしろ含む)
lx0: 「可視化に使う領域」の端のグリッド番号
rln: ln からのりしろを除いたグリッドサイズ
loccorner: MPI 受け持ちの領域の端の座標。のり代含む。
dx: vismo__setUniCoord の dx と同じ

下記のようなのを想定しています。（□は○と●の位置を整えるためのもので意味なし）

○●●●●●○
□□□□□□●●●●●●○
□□□□□□□□□□□□●●●●●●○
□□□□□□□□□□□□□□□□□□●●●●●●○

○はのりしろ、●は可視化に使う領域を決めるグリッド。○にもデータが入っている前提です。（データが入っていないと、等値面に穴が開く、シミができるなど、ちょっとした不都合があります）

○と●の合計が ln
●の数が rln
左端の黒丸の位置が lx0
のり代を含む左端の座標が loccorner。（上の場合は lx0の位置と一致している）
注意事項は、シミュレーションではのり代でも、上記のように
●が重なるように VISMO の座標を設定しないと、うまく可視化できません。
lx0に代入するグリッド番号は、各プロセスで1から始まるとした番号です。データ全体の番号ではありません。

RectiLinear の場合
vismo__setUniCoord の代わりに vismo__setRectCoord(n, coordx, coordy, coordz)
を使う
integer, dimension(3), intent(in) :: n
real(kind=vdp), dimension(nx(1)), intent(in) :: coordx //倍精度
real(kind=vdp), dimension(nx(2)), intent(in) :: coordy //倍精度
real(kind=vdp), dimension(nx(3)), intent(in) :: coordz //倍精度

vismo__setLocalUniCoord の代わりに vismo__setLocalRectCoord(ln, lx0, rln, coordx, coordy, coordz)
integer, dimension(3), intent(in) :: ln, lx0, rln
real(kind=vdp), dimension(nx(1)), intent(in) :: coordx //倍精度

real(kind=vdp), dimension(nx(2)), intent(in) :: coordy //倍精度
real(kind=vdp), dimension(nx(3)), intent(in) :: coordz //倍精度

3.2 Fortran のコードとの結合(MPI 並列なし)

test1.f90、MakefileS を参考にしてください。-DSERIAL が必要です。
Vismo のイニシャライズのサブルーチンに、mpi コミュニケータ、プロセス番号、プロセスサイズの引数はありません。
call vismo__init("configtest.vsm")

座標の設定では、n=lx=rln, lx0=1, corner=loccorner とすれば良い。

3.3 C のコードとの結合

test8c.c, test1c.c, MakefileC, MakefileS を参考にしてください。-DCLANG が必要です。
まず、
#incude "vismo.h"
の記述が必要です。また、cvismo.c のコンパイルも必要です。
関数名は、上記の Fortran のサブルーチンと同じです。データの配列の例は、1次元ですが、3次元の配列でも、キャストすれば可能だと思います。また、データ領域は連続していることを想定しています。MPI 並列していない場合は、-DSERIAL が必要です。
gcc + gfortran (-DUSE_GFORT)と Intel C+ Intel Fotran(-DUSE_IFORT)でコンパイルできるようにしてありますが、他のコンパイラでも nm vismo.o で fortran のサブルーチンがどのような名前で入っているか調べた上で、cvismo.c を書き換えてください。例えば、gfortran では、vismo__visualization は、___vismo_MOD_vismo__visualization という名で入っています。

4 VISMO のパラメータファイルのルール

4.1 全体

小文字は数値を記述する場所。区切りはスペース。TAB は使用不可。configtest.vsm, configtest2.vsm を参考にしてください。
球表示、半透明表示は、カーテシアン版のみ対応です。

4.2 パラメータファイルの書き方

別ファイルの読み込み
INCLUDE ファイル名

画像サイズ(ピクセルサイズ)
IMAGESIZE width height //横(width)×縦(height)

フレームの太さ
FRAMEWIDTH width //width=太さ

背景の色
BGCOLOR r g b //r,g,b 0.0-1.0

カメラ（視点）
CAMERA

[PERSPECTIVE or ORTHOGONAL] // 省略した場合、Perspective (透視射影)
POSITION x y z //カメラの位置
FRONT x y z //カメラの前方向
UP x y z //カメラの上方向
FOVY f // perspective の場合必要。単位は degree
NEARFAR near far //near より近い、far より遠い部分は描画しない
WIDTH w //Ortho の場合必要。w はスクリーンの横の長さ。縦の長さは IMGESIZE から計算される
END_CAMERA

複数のカメラを自動的に作りたい場合。軸、半径、中心とカメラの数をセットすると、カメラを複数自動的に生成する。
MULTIPLECAMERAS
[PERSPECTIVE or ORTHOGONAL] // 省略した場合、Perspective (透視射影)
AXIS x y z //回転軸の方向
RADIUS r // 回転半径
CENTER x y z // 回転中心
NVAMERA n //カメラの数
FOVY f // perspective の場合必要。単位は degree
NEARFAR near far //near より近い、far より遠い部分は描画しない
WIDTH w //Ortho の場合必要。w はスクリーンの横の長さ。縦の長さは IMGESIZE から計算される
END_MULTIPLECAMERAS

照明
LIGHT
[PARALLEL or POINT] // 省略した場合、parallel
POSITION x y z //光源の位置
AMB r g b //環境光
DIF r g b //拡散光
SPEC r g b //反射光
SHIN s //反射光のパラメータ
END_LIGHT
POSITION や AMB などの順番はこの通りでなくてもよい。RGB は0.0-1.0

デプスキュー
DEPTHCUE kind near far
kind: 1:linear, 2 exp(-x), 3:exp(-x*x)
near と far の間を徐々に暗くする

カラーマップ
SCAL_COLORMAP
SCALNUM n //n=スカラ番号
EQUALLY_SPACED // value が等間隔の場合このトークンを入れる
value1 r g b a // R,G,B,A は0.0～1.0
value2 r g b a
・・・・
END_SCAL_COLORMAP

VECT_COLORMAP
VECTNUM n //n=ベクトル番号
EQUALLY_SPACED// value が等間隔の場合このトークンを入れる
value1 r g b // スカラーと違って a はない
value2 r g b
・・・・
END_VECT_COLORMAP

PTCL_COLORMAP
PTCLNUM n //n=粒子番号
EQUALLY_SPACED // value が等間隔の場合このトークンを入れる
value1 r g b rad // R,G,B は0.0～1.0, RAD は半径
value2 r g b rad
・・・・
END_PTCL_COLORMAP

可視化パラメータ
VISUALIZATION

END_VISUALIZATION
で囲む

4.3　VISUALIZATION～END_VISUALIZATION 内の記述

等値面
ISOSURFACE
SCALNUM n //スカラの番号
LEVEL level //等値面レベル
AMB r g b //AMB, DIF, SPEC, SHIN は光源と同様の意味
DIF r g b //ユーザーが r,g,b を指定して色(材質)を決める
SPES r g b
SHIN shin
[ALPHA alpha] // 不透明度。省略した場合、alpha=1.0
END_ISOSURFACE

スライス
SLICE
SCALNUM n
EQ a b c d // f(x,y,z) = ax+by+cz+d=0
AMB amb //等値面と異なりユーザーが r,g,b は指定しない。
DIF dif //反射率のみを決める
SPEC spec
SHIN shin
[ALPHA alpha] // 不透明度。省略した場合、alpha=1.0
END_SLICE

ボリュームレンダリング
VOLUME_RENDERING
SCALNUM n
AMB amb
DIF dif

SPEC spec
SHIN shin
END_VOLUME_RENDERING

流線
STREAM_LINES
VECTNUM n
AMB [r g b or amb] // r,g,b を指定して MONO とすると、その色で描く。
DIF [r g b or dif]　　// amb や dif の値を入れるとベクトルの絶対値
SPEC [r g b or spec] //で色付けする
SHIN shin
RADIUS rad //チューブ状に描くので半径が必要
[MONO or COLOR] //省略すると COLOR
SEED // 流線の出発点
x y z
x y z
…
END_SEED
END_STREAM_LINES

矢印
ARROWS
VECTNUM n
EQ a b c d // f(x,y,z) = ax+by+cz+d=0
DENSITY d // 矢印の密度
RADIUS r //矢印の半径(筒の部分)
LENGTH l //矢印の長さ
END_ARROWS

粒子表示
PARTICLE
PTCLNUM n
COLOR //カラーマップを使用するときに入れる
RADIUS r // カラーマップを使用する場合は不要
AMB [r g b or amb] // カラーマップを使用する場合は amb、それ以外の場合は r,g,b を指定する（その色で描く）。
DIF [r g b or dif]// AMB, DIF, SPEC は、流線と同様
SPEC [r g b or spec]
SHIN s
END_PARTICLE

[DEPTHCUE]
デプスキューを **ON** にする。省略可

LIGHT n
光源(ライト)番号 **n** を **ON** にする

INNERSPHERE
内核描画(**YinYang** 版）

OUTERSPHEREFRAME
外核フレーム描画(YinYang 版）