DEMO_0009_Multi_Morph_Cylindrical_Arrangment

Table of Contents

This is a demo for:

- Building geometry for multi-morphology TPMS structures (gyroid and diamond) in cylindrical arrangement, with transition in different directions.

This demo contains:

1. Case-1: TPMS in cylindrical arrangment, radial transition.
2. Case-2: TPMS in cylindrical arrangment, circumferential transition.
3. Case-3: TPMS in cylindrical arrangment, axial transition.

Name


```
%aee@gmail.com>
%
%  Change log:
%  2023/11/15 MV Created
%  2024/02/2 MV
% ----------------------------------------------------------------------
```

```
clear; close all; clc;
```

Plot settings

```
fontSize=20;
faceAlpha1=0.8;
markerSize=10;
lineWidth1=3;
lineWidth2=4;
markerSize1=25;
```

Control parameters

```
res=100; %Resolution
```

Setting-up input parameters for individual lattices

```
inputStruct_A.size=[0, 3, 2*pi, 3]; % [r_in, r_out, theta, length], for
 section view :[0, 3, pi, 3]
inputStruct_A.Ns=res; % number of sampling points
inputStruct_A.isocap=1; %Option to cap the isosurface
inputStruct_A.surfaceCase='g'; %Surface type

r1=inputStruct_A.size(1,1); % inner radius
r2=inputStruct_A.size(1,2); % outter radius
L=inputStruct_A.size(1,4); % height

inputStruct_B = inputStruct_A;
inputStruct_C = inputStruct_A;

% Set parameters for individual gyroid
inputStruct_A.numPeriods=[3 10 2]; %Number of periods in each direction
inputStruct_A.levelset=-0.8 ; %Isosurface level
levelset_A=inputStruct_A.levelset;

inputStruct_B.numPeriods=[2 8 2];
inputStruct_B.levelset=-0.9;
levelset_B=inputStruct_B.levelset;
inputStruct_B.surfaceCase='d';

inputStruct_C.numPeriods=[2 20 2];
inputStruct_C.levelset=-0.5;
levelset_C=inputStruct_C.levelset;
inputStruct_C.surfaceCase='g';
```

Compute individual TPMS

No need to store faces and vertices, only require underlying S, grid coordinates, and levelset values

```
[F,V,C,S_A,X,Y,Z,r,theta]=TPMS_LCS(inputStruct_A);
[~,~,~,S_B,~,~,~,~,~]=TPMS_LCS(inputStruct_B);
[~,~,~,S_C,~,~,~,~,~]=TPMS_LCS(inputStruct_C);
```

Transition lengthScale and shape

kappa controls the lengthscale of transition between lattices Higher kappa => faster transition Lower kappa => slower transition

```
kappa =15;

% Transition shape type
transitionType = 1;
switch transitionType
    case 1 % radial
        center_A = [1, 1, 1];
        center_B = [2, 2, 1];
```

```
        center_C = [4, 4, 1];

        weights_A = exp(-kappa * ((r.^2-(center_A(1,1)).^2 +
 center_A(1,2)).^2));
        weights_B = exp(-kappa * ((r.^2-(center_B(1,1)).^2 +
 center_B(1,2)).^2));
        weights_C = exp(-kappa * ((r.^2-(center_C(1,1)).^2 +
 center_C(1,2)).^2));

    case 2 % circumferential
        theta_A=0;
        theta_B=2/3*pi;
        theta_C=4/3*pi;

        center_A = [r2*cos(theta_A), r2*sin(theta_A), L];
        center_B = [r2*cos(theta_B), r2*sin(theta_B), L];
        center_C = [r2*cos(theta_C), r2*sin(theta_C), L];

        weights_A = exp(-kappa *
 (Squared_distance_from_point(X,Y,Z,center_A)));
        weights_B = exp(-kappa *
 (Squared_distance_from_point(X,Y,Z,center_B)));
        weights_C = exp(-kappa *
 (Squared_distance_from_point(X,Y,Z,center_C)));

%  Alternative approach
%         theta_A=0;
%         theta_B=2/3*pi;
%         theta_C=4/3*pi;
%
%         weights_A = exp(-kappa * (theta-theta_A).^2);
%         weights_B = exp(-kappa * (theta-theta_B).^2);
%         weights_C = exp(-kappa * (theta-theta_C).^2);

    case 3 % axial
        center_A = [0, 0, 0.5];
        center_B = [0, 0, 1.5];
        center_C = [0, 0, 2.5];

        weights_A = exp(-kappa *
 (Squared_distance_from_point(X,Y,Z,center_A)));
        weights_B = exp(-kappa *
 (Squared_distance_from_point(X,Y,Z,center_B)));
        weights_C = exp(-kappa *
 (Squared_distance_from_point(X,Y,Z,center_C)));

end
```

Weights must sum up to 1.

```
% Computing the weights for each TPMS evaluated on all grid points.
sum_weights = weights_A + weights_B + weights_C;
```

```
weights_A = weights_A ./ sum_weights;
weights_B = weights_B ./ sum_weights;
weights_C = weights_C ./ sum_weights;

% Interpolating using the above weights
graded_S =  weights_A .* (S_A - levelset_A) ...
            + weights_C .* (S_C - levelset_C) ...
            + weights_B .* (S_B - levelset_B);
```

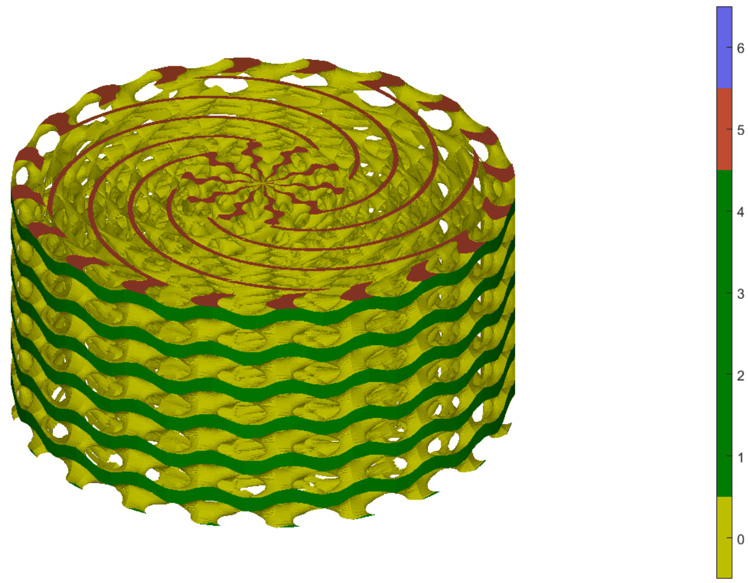
Compue isosurface

```
graded_levelset = 0;

[f,v] = isosurface(X,Y,Z,graded_S,graded_levelset);
c=zeros(size(f,1),1);

% Compute isocaps
[fc,vc] = isocaps(X,Y,Z,graded_S,graded_levelset,'enclose','below');

% Boilerplate code for preparing output for exporting/visualization
nc=patchNormal(fc,vc);
cc=zeros(size(fc,1),1);
cc(nc(:,1)<-0.5)=1;
cc(nc(:,1)>0.5)=2;
cc(nc(:,2)<-0.5)=3;
cc(nc(:,2)>0.5)=4;
cc(nc(:,3)<-0.5)=5;
cc(nc(:,3)>0.5)=6;

% Join sets
[f,v,c]=joinElementSets({f,fc},{v,vc},{c,cc});

% Merge nodes
[f,v]=mergeVertices(f,v);

% Check for unique faces
[~,indUni,~]=unique(sort(f,2),'rows');
f=f(indUni,:); %Keep unique faces
c=c(indUni);

% Remove collapsed faces
[f,logicKeep]=patchRemoveCollapsed(f);
c=c(logicKeep);

% Remove unused points
[f,v]=patchCleanUnused(f,v);

% Invert faces
f=fliplr(f);
```

Visualize

```
map=[0.75 0.75 0
```

```
    0 0.5 0
    0   0.5   0
    0 0.5 0
    0 0.5 0
    0.75 0.3 0.2
    0.4 0.4 0.9];

Hybrid_vizualize(f,v,c,map,[]);
```



Published with MATLAB® R2021b