DEMO_0004_Hybrid_Lattices

Table of Contents

This is a demo for:

- Building geometry for multi-morphology lattices of in cubic domain using hybrid formulation:

1. Example-1: Utilizes hybrid formulation for a 4-morph TPMS lattice in cubic domain

2. Example-2: Utilizes hybrid formulation for a 3-morph Spinodoid lattices in cubic domain

Name

Author: *Mahtab Vafaee*, mahtab.vafaee@gmail.com

```
Change log:
2023/11/15 MV Created
2024/01/31 MV Sorted for publishing
--------------------------------------------------------------------------

clear; close all; clc;
```

Plot settings

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

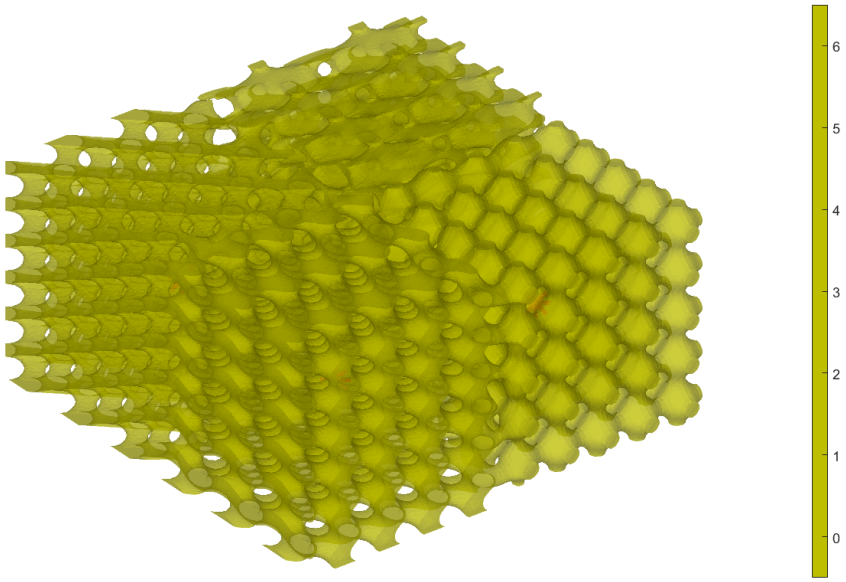
Example 1: Cubic Hybrid TPMS (Figure-4(a-c))

```
% Control parameters
sampleSize=[2 2 1]; %Heigh of the sample
pointSpacing=sampleSize/100; % Resolution

overSampleRatio=1;
numStepsLevelset=ceil(overSampleRatio.*(sampleSize./pointSpacing)); %Number of
 voxel steps across period for image data (roughly number of points on mesh
 period)
```

```
inputStruct_A.L=sampleSize; % characteristic length
inputStruct_A.Ns=numStepsLevelset; % number of sampling points
inputStruct_A.isocap=1; %Option to cap the isosurface
inputStruct_A.surfaceCase='g'; %Surface type

inputStruct_B = inputStruct_A;
inputStruct_C = inputStruct_A;
inputStruct_D = inputStruct_A;

% Set parameters for individual lattices
inputStruct_A.numPeriods=[9 6 6]; %Number of periods in each direction
inputStruct_A.levelset=-0.1 ; %Isosurface level
inputStruct_A.gradiantF=0 ; %Gradiant Factor
levelset_A=inputStruct_A.levelset;

inputStruct_B.numPeriods=[6 6 6];
inputStruct_B.levelset=-0.4;
inputStruct_B.gradiantF=0 ; %Gradiant Factor
levelset_B=inputStruct_B.levelset;
inputStruct_B.surfaceCase='d';

inputStruct_C.numPeriods=[10 10 5];
inputStruct_C.levelset=-0.4;
inputStruct_C.gradiantF=0; %Gradiant Factor
levelset_C=inputStruct_C.levelset;
inputStruct_C.surfaceCase='p';

inputStruct_D.numPeriods=[4 8 8];
inputStruct_D.levelset=-0.2;
inputStruct_D.gradiantF=0 ; %Gradiant Factor
levelset_D=inputStruct_D.levelset;
inputStruct_D.surfaceCase='n';

% Compute individual TPMS
% No need to store faces and vertices, only require underlying S,
% grid coordinates, and levelset values
[~,~,~,S_A,X,Y,Z]=triplyPeriodicMinimalSurface(inputStruct_A);
[~,~,~,S_B,~,~,~]=triplyPeriodicMinimalSurface(inputStruct_B);
[~,~,~,S_C,~,~,~]=triplyPeriodicMinimalSurface(inputStruct_C);
[~,~,~,S_D,~,~,~]=triplyPeriodicMinimalSurface(inputStruct_D);

% Define the central location of each individual lattices in space
% E.g., At center_A, the structure will definitely correspond to input_A.
% As we move away from center_A, it will slowly transition into other
% lattices with input_B and input_C.
center_A = [0.5, 0.5, 0.5];
center_B = [0.5, 1.5, 0.5];
center_C = [1.5, 0.5, 0.5];
center_D = [1.5, 1.5, 0.5];

% Transition lengthscal and shape
% kappa controls the lengthscale of transition between TPMS
% Higher kappa => faster transition
% Lower kappa => slower transition
```

```
kappa = 8;

% Using Gaussian (a.k.a. radial basis functions) interpolation.
% One can use any interpolation scheme of choice as long as weights at
% every grid point sum up to 1.
% Computing the weights for each spinodoid evaluated on all grid points.
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));
weights_D = exp(-kappa * Squared_distance_from_point(X,Y,Z,center_D));

% Compute weight functions
% Weights must sum up to 1.
sum_weights = weights_A + weights_B ...
+ weights_C + weights_D;

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

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

% 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
% Hybrid_vizualize(f,v,c,map, center_V);
map= [0.75 0.75 0];
center_V= [center_A; center_B ; center_C; center_D];
Hybrid_vizualize(f,v,c,map,center_V)
```



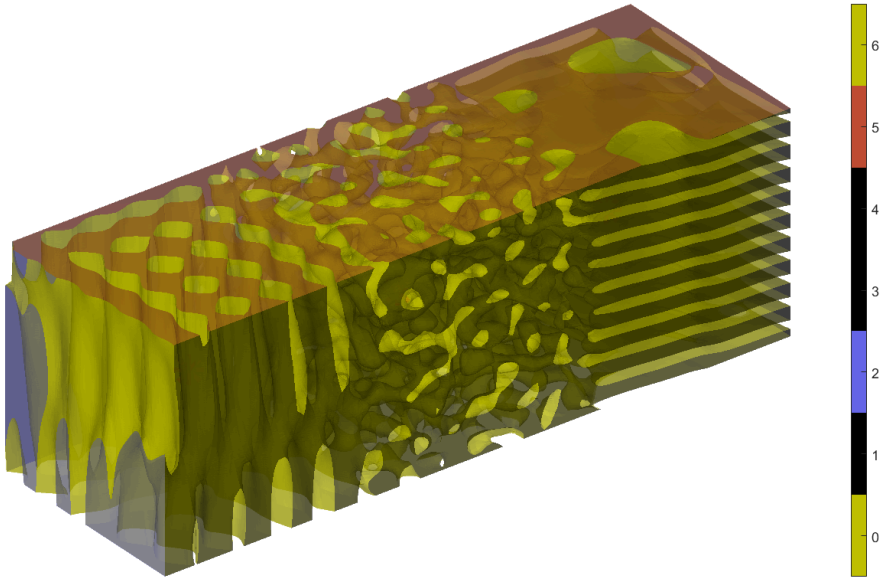
Example 2: Cubic Hybrid Spinodoid (Figure-4(d-f))

```
% Control parameters
sampleSize=[3 1 1]; %Heigh of the sample
res=[100 100 100]; % set the resolution in 3D

inputStruct.isocap=true;
inputStruct_A.domainSize=sampleSize;
inputStruct_A.resolution=res;
```

```
inputStruct_A.numWaves=1000;

inputStruct_B = inputStruct_A;
inputStruct_C = inputStruct_A;

% Set parameters for individual spinodoid
inputStruct_A.waveNumber=10*pi;
inputStruct_A.relativeDensity=0.3;
inputStruct_A.thetas=[15 15 0];
levelset_A=inputStruct_A.relativeDensity;

inputStruct_B.waveNumber=15*pi;
inputStruct_B.relativeDensity=0.7;
inputStruct_B.thetas=[90 90 0];
levelset_B=inputStruct_B.relativeDensity;

inputStruct_C.waveNumber=20*pi;
inputStruct_C.relativeDensity=0.4;
inputStruct_C.thetas=[0 0 15];
levelset_C=inputStruct_C.relativeDensity;

% Compute individual spinodoids
% No need to store faces and vertices, only require underlying S,
% grid coordinates, and levelset values
[~,~,~,S_A,X,Y,Z]=spinodoid(inputStruct_A);
[~,~,~,S_B,~,~,~]=spinodoid(inputStruct_B);
[~,~,~,S_C,~,~,~]=spinodoid(inputStruct_C);

% Define the central location of each individual spinodoid in space
% E.g., At center_A, the spinodoid will definitely correspond to input_A.
% As we move away from center_A, it will slowly transition into other
% spinodoids with input_B and input_C.
center_A = [0.5, 0.5, 0.5];
center_B = [1.5, 0.5, 0.5];
center_C = [2.5, 0.5, 0.5];

% kappa controls the lengthscale of transition between spinodoids
% Higher kappa => faster transition
% Lower kappa => slower transition
kappa = 8;

% Using Gaussian (a.k.a. radial basis functions) interpolation.
% One can use any interpolation scheme of choice as long as weights at
% every grid point sum up to 1.
% Computing the weights for each spinodoid evaluated on all grid points.
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));

% Weights must sum up to 1.
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_B .* (S_B - levelset_B)...
            + weights_C .* (S_C - levelset_C);

% 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
% Hybrid_vizualize(f,v,c,map, center_V);
center_V=[center_A; center_B; center_C];
map=[0.75 0.75 0
    0 0 0
    0.4 0.4 0.9
    0 0 0
```

```
    0 0 0
    0.75 0.3 0.2
    0.75 0.75 0];
Hybrid_vizualize(f,v,c, map, center_V)
```



Published with MATLAB® R2021b