# Generating mesh with periodic boundary condition

# Adding noise to mesh

## Import Mesh

In[ ]:=
```
DumpGet[
  "C:\\Users\\aliha\\Desktop\\wolfram-vertex-3D\\create geometry\\infinitesheet.mx"];
yLim[[1]] =
  0.;
```

In[ ]:=
```
edges = SetPrecision[edges, 10];
indToPtsAssoc = SetPrecision[indToPtsAssoc, 10];
ptsToIndAssoc = KeyMap[SetPrecision[#, 10] &, ptsToIndAssoc];
xLim = SetPrecision[xLim, 10];
yLim = SetPrecision[yLim, 10];
faceListCoords = Map[Lookup[indToPtsAssoc, #] &, cellVertexGrouping, {2}];
```

# Initialization/F[x]'s

In[ ]:=
```
With[{xlim1 = xLim[[1]], xlim2 = xLim[[2]], ylim1 = yLim[[1]], ylim2 = yLim[[2]]},
  periodicRules = Dispatch[{
     {x_ /; x ≥ xlim2, y_ /; y ≤ ylim1, z_} :> SetPrecision[{x - xlim2, y + ylim2, z}, 10],
     {x_ /; x ≥ xlim2, y_ /; ylim1 < y < ylim2, z_} :> SetPrecision[{x - xlim2, y, z}, 10],
     {x_ /; xlim1 < x < xlim2, y_ /; y ≤ ylim1, z_} :> SetPrecision[{x, y + ylim2, z}, 10],
     {x_ /; x < 0., y_ /; y ≤ ylim1, z_} :> SetPrecision[{x + xlim2, y + ylim2, z}, 10],
     {x_ /; x < 0., y_ /; ylim1 < y < ylim2, z_} :> SetPrecision[{x + xlim2, y, z}, 10],
     {x_ /; x < 0., y_ /; y > ylim2, z_} :> SetPrecision[{x + xlim2, y - ylim2, z}, 10],
     {x_ /; 0. < x < xlim2, y_ /; y > ylim2, z_} :> SetPrecision[{x, y - ylim2, z}, 10],
     {x_ /; x > xlim2, y_ /; y ≥ ylim2, z_} :> SetPrecision[{x - xlim2, y - ylim2, z}, 10]
    }];

  transformRules = Dispatch[{
     {x_ /; x ≥ xlim2, y_ /; y ≤ ylim1, _} :> SetPrecision[{-xlim2, ylim2, 0}, 10],
     {x_ /; x ≥ xlim2, y_ /; ylim1 < y < ylim2, _} :> SetPrecision[{-xlim2, 0, 0}, 10],
     {x_ /; xlim1 < x < xlim2, y_ /; y ≤ ylim1, _} :> SetPrecision[{0, ylim2, 0}, 10],
     {x_ /; x < 0, y_ /; y ≤ ylim1, _} :> SetPrecision[{xlim2, ylim2, 0}, 10],
     {x_ /; x < 0, y_ /; ylim1 < y < ylim2, _} :> SetPrecision[{xlim2, 0, 0}, 10],
     {x_ /; x < 0, y_ /; y > ylim2, _} :> SetPrecision[{xlim2, -ylim2, 0}, 10],
     {x_ /; 0 < x < xlim2, y_ /; y > ylim2, _} :> SetPrecision[{0, -ylim2, 0}, 10],
     {x_ /; x > xlim2, y_ /; y ≥ ylim2, _} :> SetPrecision[{-xlim2, -ylim2, 0}, 10],
     {___Real} :> SetPrecision[{0, 0, 0}, 10]}];
 ];
```

In[ ]:=
```
wrappedMat = AssociationThread[
   Keys[cellVertexGrouping] → Map[Lookup[indToPtsAssoc, #] /. periodicRules &,
     Lookup[cellVertexGrouping, Keys[cellVertexGrouping]], {2}]];
```

In[ ]:=
```
triangulateFaces[faces_] := Block[{edgelen, ls, mean},
   (ls = Partition[#, 2, 1, 1];
      edgelen = Norm[SetPrecision[First[#] - Last[#], 10]] & /@ ls;
      mean = Total[edgelen * (Midpoint /@ ls)] / Total[edgelen];
      mean = mean ~SetPrecision~ 10;
      Map[Append[#, mean] &, ls]) & /@ faces
  ];
```

In[ ]:=

```
wrappedBoundaryPts[indptsassoc_, ptstoindassoc_] :=
  With[{xlim1 = xLim[[1]], xlim2 = xLim[[-1]], ylim1 = yLim[[1]], ylim2 = yLim[[-1]]},
   Block[{pts, ptsxy, ptsx, ptsy, zmin, zmax, posx, negx,
     posy, negy, outsidePts, outsidePtscoords, shiftedpts},
    pts = Values@indptsassoc;
    ptsxy = pts[[All, 1 ;; 2]];
    ptsx = ptsxy[[All, 1]];
    ptsy = ptsxy[[All, 2]];
    {zmin, zmax} = MinMax@pts[[All, -1]];
    posx = Position[UnitStep[ptsx - xlim2], 1];
    negx = Position[ptsx, x_ /; x < xlim1];
    posy = Position[UnitStep[ptsy - ylim2], 1];
    negy = Position[ptsy, y_ /; y < ylim1];
    outsidePts = Union@Flatten[posx ~Append~ negx ~Append~ posy ~Append~ negy];
    outsidePtscoords = Lookup[indptsassoc, outsidePts];

    shiftedpts = Map[
      Block[{tempvec = #},
       tempvec = Which[tempvec[[1]] ≥ xlim2,
         {tempvec[[1]] - xlim2, tempvec[[2]], tempvec[[-1]]} ~SetPrecision~ 10,
         tempvec[[1]] < xlim1,
         {tempvec[[1]] + xlim2, tempvec[[2]], tempvec[[-1]]} ~
          SetPrecision~ 10, True, tempvec];
       tempvec = Which[tempvec[[2]] ≥ ylim2,
         {tempvec[[1]], tempvec[[2]] - ylim2, tempvec[[-1]]} ~SetPrecision~ 10,
         tempvec[[2]] < ylim1,
         {tempvec[[1]], tempvec[[2]] + ylim2, tempvec[[-1]]} ~
          SetPrecision~ 10, True, tempvec]
       ] &, outsidePtscoords];
    Thread[
     {outsidePtscoords, Lookup[indptsassoc, Lookup[ptstoindassoc, shiftedpts]]}]
   ]
  ];
```

In[ ]:=
```
displaceVertices[indToPtsAssoc_, stitchedPtsInds_, cellVertexGrouping_,
   stdDev_ : 0.01] := Block[{noiseFunc, lenstitchedPtsInds, unstitchedptsInds,
    newunstitchedpts, newunstitchedindtopts, newstitchedindtopts,
    $indToPtsAssoc, $ptsToIndAssoc, $edges, $faceListCoords, $wrappedMat},
   noiseFunc[μ_, σ_, n_] := RandomVariate[NormalDistribution[μ, σ], {n, 3}];
   lenstitchedPtsInds = Length@stitchedPtsInds;
   unstitchedptsInds = Complement[Keys@indToPtsAssoc, Flatten@stitchedPtsInds];
   newunstitchedpts = SetPrecision[Lookup[indToPtsAssoc, unstitchedptsInds] +
      noiseFunc[0., stdDev, Length@unstitchedptsInds], 10];
   newunstitchedindtopts = Thread[unstitchedptsInds → newunstitchedpts];
   newstitchedindtopts =
    MapThread[(x ↦ x → SetPrecision[Lookup[indToPtsAssoc, x] + #2, 10]) /@ #1 &,
     {stitchedPtsInds, noiseFunc[0., stdDev, lenstitchedPtsInds]}];
   $indToPtsAssoc = KeySort@<|newunstitchedindtopts ~ Join ~ newstitchedindtopts|>;
   $ptsToIndAssoc = AssociationMap[Reverse, $indToPtsAssoc];
   $faceListCoords = Map[Lookup[$indToPtsAssoc, #] &, cellVertexGrouping, {2}];
   $wrappedMat = AssociationThread[
     Keys[cellVertexGrouping] → Map[Lookup[$indToPtsAssoc, #] /. periodicRules &,
       Lookup[cellVertexGrouping, Keys[cellVertexGrouping]], {2}]];
   $edges = Flatten[Map[Partition[#, 2, 1, 1] &, Map[Lookup[$indToPtsAssoc, #] &,
        Values[cellVertexGrouping], {2}], {2}], 2] // DeleteDuplicatesBy[Sort];
   {$indToPtsAssoc, $ptsToIndAssoc, $faceListCoords, $wrappedMat, $edges}
  ];
```

In[ ]:=
```
𝒟 = Rectangle[{First@xLim, First@yLim}, {Last@xLim, Last@yLim}];
```

In[ ]:=
```
getLocalTopology[ptsToIndAssoc_, indToPtsAssoc_, vertexToCell_,
    cellVertexGrouping_, wrappedMat_, faceListCoords_][vertices_] :=
  Module[{localtopology = <||>, wrappedcellList = {}, vertcellconns,
    localcellunion, vertInBounds, v, wrappedcellpos, vertcs = vertices,
    transVector, wrappedcellCoords, wrappedcells, vertOutofBounds,
    shiftedPt, transvecList = {}, $faceListCoords = Values@faceListCoords,
    vertexQ},
   vertexQ = MatchQ[vertices, {__?NumberQ}];
   If[vertexQ,
    vertcellconns =
     AssociationThread[{#}, {vertexToCell[ptsToIndAssoc[#]]}] &@vertices;
    vertcs = {vertices};
    localcellunion = Flatten[Values@vertcellconns],
    (* this will yield vertex → cell indices connected in the local mesh *)
    vertcellconns =
     AssociationThread[#, Lookup[vertexToCell, Lookup[ptsToIndAssoc, #]]] &@vertices;
    localcellunion = Union@Flatten[Values@vertcellconns];
   ];
   (* condition to be an internal
    edge: both vertices should have 3 or more neighbours *)
   (*Print["All topology known"];*)
   (* the cells in the local mesh define the entire network topology →
    no wrapping required *)
```

```
(* else cells need to be wrapped because other cells are
  connected to the vertices → periodic boundary conditions *)
With[{vert = #},
   If[((𝒟[#, RegionMember[Most[vert]]] &) &&
       ! (vert[[1]] == xLim[[2]] || vert[[2]] == yLim[[2]])),
     (* the vertex has less than 3 neighbouring cells but
      the vertex is within bounds *)
     (*Print["vertex inside bounds with fewer than 3 cells"];*)
     v = vertInBounds = vert;
     (* find cell indices that are attached to the vertex in wrappedMat *)
     wrappedcellpos = DeleteDuplicatesBy[
       Cases[Position[wrappedMat, x_ /; SameQ[x, v], {3}],
        {Key[p : Except[Alternatives @@
              Join[localcellunion, Flatten@wrappedcellList]]], y__} :> {p, y}],
       First];
     (*wrappedcellpos = wrappedcellpos/.
        {Alternatives@@Flatten[wrappedcellList],__} :> Sequence[];*)
     (* if a wrapped cell has not been considered earlier (i.e. is new)
      then we translate it to the position of the vertex *)
     If[wrappedcellpos ≠ {},
      If[vertexQ,
       transVector = SetPrecision[(v - Extract[$faceListCoords,
              Replace[#, {p_, q__} :> {Key[p], q}, {1}]]) & /@ wrappedcellpos, 10],
       (*the main function is enquiring an edge and not a vertex*)
       transVector =
        SetPrecision[(v - Extract[$faceListCoords, #]) & /@ wrappedcellpos, 10]
      ];
      wrappedcellCoords = MapThread[#1 →
          Map[Function[x, SetPrecision[x + #2, 10]], $faceListCoords[[#1]], {2}] &,
        {First /@ wrappedcellpos, transVector}];
      wrappedcells = Keys@wrappedcellCoords;
      AppendTo[wrappedcellList, Flatten@wrappedcells];
      AppendTo[transvecList, transVector];
      AppendTo[localtopology, wrappedcellCoords];
       (*local topology here only has wrapped cell *)
     ],
     (*Print["vertex out of bounds"];*)
     (* else vertex is out of bounds *)
     vertOutofBounds = vert;
     (* translate the vertex back into mesh *)
     transVector = vertOutofBounds /. transformRules;
     shiftedPt = SetPrecision[vertOutofBounds + transVector, 10];
     (* find which cells the vertex is a part of in the wrapped matrix *)
     wrappedcells = Complement[
       Union@Cases[Position[wrappedMat, x_ /; SameQ[x, shiftedPt], {3}],
          x_Key :> Sequence @@ x, {2}] /. Alternatives @@ localcellunion → Sequence[],
       Flatten@wrappedcellList];
     (*forming local topology now that we know the wrapped cells *)
     If[wrappedcells ≠ {},
      AppendTo[wrappedcellList, Flatten@wrappedcells];
```

```
        wrappedcellCoords = AssociationThread[wrappedcells,
          Map[Lookup[indToPtsAssoc, #] &, cellVertexGrouping[#] & /@ wrappedcells, {2}]
         ];
        With[{opt = (vertOutofBounds /. periodicRules)},
         Block[{pos, vertref, transvec},
           Do[
            With[{cellcoords = wrappedcellCoords[cell]},
             pos = FirstPosition[cellcoords /. periodicRules, opt];
             vertref = Extract[cellcoords, pos];
             transvec = SetPrecision[vertOutofBounds - vertref, 10];
             AppendTo[transvecList, transvec];
             AppendTo[localtopology, cell →
               Map[SetPrecision[# + transvec, 10] &, cellcoords, {2}]];
            ], {cell, wrappedcells}]
           ];
         ];
        ];
       ] & /@ vertcs;
  If[localcellunion ≠ {},
   AppendTo[localtopology,
    Thread[localcellunion →
      Map[Lookup[indToPtsAssoc, #] &, cellVertexGrouping /@ localcellunion, {2}]]
   ]
  ];
  (*Print[Values@localtopology//Min/@Map[Precision,#,{3}]&];*)
  transvecList = Which[
    MatchQ[transvecList, {{{__?NumberQ}}}], First[transvecList],
    MatchQ[transvecList, {{__?NumberQ} ..}], transvecList,
    True, transvecList //. {x___, {p : {__?NumberQ} ..}, y___} :> {x, p, y}
   ];
  {localtopology, Flatten@wrappedcellList, transvecList}
 ];
```

# Addition of noise

In[ ]:=

```
outsideptspairs = wrappedBoundaryPts[indToPtsAssoc, ptsToIndAssoc];
mappedpts = GroupBy[# ~ Join ~ Reverse[#, 2] &@
    (Rule @@ Lookup[ptsToIndAssoc, #] & /@ outsideptspairs), First → Last];
```

In[ ]:=
```
stitchedPtsInds = Block[{wrapI, bpconn, val, temp, keys = Keys@mappedpts},
    ParallelTable[
     wrapI = {i};
     bpconn = First@FixedPoint[
         (val = #[[-1]];
           temp = Flatten[Lookup[mappedpts, #] &@val];
           wrapI = If[Complement[temp, #1[[1]]] ≠ {},
             Union@Flatten@Append[wrapI, temp], wrapI];
           {wrapI, temp}) &, {wrapI, wrapI}, SameTest → (#[[1]] === #2[[1]] &)],
     {i, keys}]
   ] // DeleteDuplicatesBy[Sort];
```

In[ ]:=
```
bpts = Union@Flatten@stitchedPtsInds;
innerptsInds = Complement[Keys@indToPtsAssoc, bpts];
```

In[ ]:=
```
SeedRandom[3];
{$indToPtsAssoc, $ptsToIndAssoc, $faceListCoords, $wrappedMat, $edges} =
  displaceVertices[indToPtsAssoc, stitchedPtsInds, cellVertexGrouping, 0.05];
```

In[ ]:=
```
$wrappedMatTrim = KeyTake[$wrappedMat, Union@Flatten[vertexToCell /@ bpts, 1]];
```

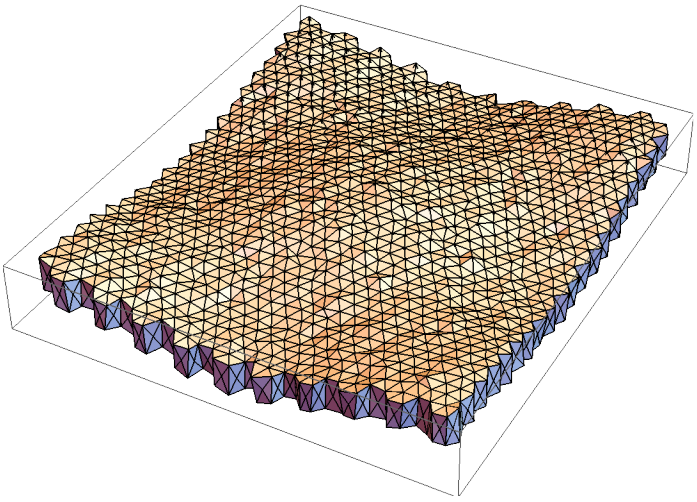
In[ ]:=
```
(problematicidx =
   Flatten@Position[Keys[getLocalTopology[$ptsToIndAssoc, $indToPtsAssoc, vertexToCell,
             cellVertexGrouping, $wrappedMatTrim, $faceListCoords][$indToPtsAssoc[#]] //
           First] & /@ bpts // Map[Length], _?(# < 3 &)]) == {}
```

Out[ ]=
```
True
```

In[ ]:=
```
mesh = Map[Lookup[$indToPtsAssoc, #] &, cellVertexGrouping, {2}];
```

In[ ]:=
```
Graphics3D[Polyhedron@Flatten[triangulateFaces@#, 1] & /@ Values[mesh]]
```

Out[ ]=



In[ ]:=
```
(*Manipulate[Graphics3D[{Polyhedron@Flatten[triangulateFaces@#,1]&/@Values[mesh],
   PointSize[0.02],Red,Point@$indToPtsAssoc[#]&/@bpts[[problemidx]],
   PointSize[0.01],Green,Point@$indToPtsAssoc[#]&/@Complement[bpts,bpts[[problemidx]]],
   PointSize[0.02],Black,Dynamic[Point@$indToPtsAssoc[i]]},
  ImageSize→Large],{i,1,Length[Keys@$indToPtsAssoc],1}]*)
```

*all boundary vertices should have 3 neighbours*

In[ ]:=
```
Keys[getLocalTopology[$ptsToIndAssoc, $indToPtsAssoc,
        vertexToCell, cellVertexGrouping, $wrappedMatTrim, $faceListCoords][
      $indToPtsAssoc[#]] // First] & /@ bpts // Counts@*Map[Length]
```

Out[ ]= ⟨| 3 → 316 |⟩

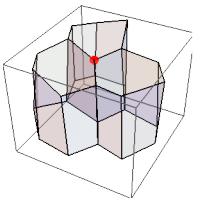
*all vertices should have 3 neighbours*

In[ ]:=
```
Keys[getLocalTopology[$ptsToIndAssoc, $indToPtsAssoc, vertexToCell, cellVertexGrouping,
       $wrappedMatTrim, $faceListCoords][$indToPtsAssoc[#]] // First] & /@
  Range[Max@$ptsToIndAssoc] // Counts@*Map[Length]
```

Out[ ]= ⟨| 3 → 1760 |⟩

In[ ]:=
```
Graphics3D[{{Opacity[0.2], Polyhedron /@ Values[
       getLocalTopology[$ptsToIndAssoc, $indToPtsAssoc, vertexToCell, cellVertexGrouping,
          $wrappedMatTrim, $faceListCoords][$indToPtsAssoc[#]] // First]},
    {Red, PointSize[0.05], Point@$indToPtsAssoc[#]}}, ImageSize → Tiny] &[
 RandomChoice@*Range@*Max@$ptsToIndAssoc]
```

Out[ ]=



```
(*Manipulate[Graphics3D[{{Opacity[0.2],
    Polyhedron/@Values[getLocalTopology[$ptsToIndAssoc,$indToPtsAssoc,vertexToCell,
         cellVertexGrouping,$wrappedMat,$faceListCoords][$indToPtsAssoc[i]]//First]},
   {Red,PointSize[0.03],Point@$indToPtsAssoc[i]}}],{i,1,Length@indToPtsAssoc,1}]*)
```

# Exporting Mesh

Export the new mesh and all of the associated data-structures

```
(*{edges,indToPtsAssoc,ptsToIndAssoc,wrappedMat,faceListCoords}=
 {$edges,$indToPtsAssoc,$ptsToIndAssoc,$wrappedMat,$faceListCoords};
DumpSave["C:\\Users\\aliha\\Desktop\\wolfram-vertex-3D\\add noise
   to mesh\\infinitesheet-noise.mx",{edges,indToPtsAssoc,ptsToIndAssoc,
  vertexToCell,cellVertexGrouping,wrappedMat,faceListCoords,xLim,yLim}];*)
```