2D Vertex Model

geometrical f(x)

```
In[∘]:= (*get vertex in the list by ccw order*)
        Clear[getCounterClockwise];
        getCounterClockwise[vertex_, vertices_] := Block[{pos, v},
            pos = First@@Position[vertices, vertex];
            If[pos == Length[vertices], pos = 1, pos += 1];
            vertices[[pos]]
          ];
```

```
In[∘]:= (*get vertex in the list by cw order*)
        Clear[getClockwise];
        getClockwise[vertex_, vertices_] := Block[{ls, pos},
            pos = First@@Position[vertices, vertex];
            If[pos == 1, pos = Length[vertices], pos -= 1];
            vertices[[pos]]
          ];
```

```
In[∘]:= getCounterClockwise[{x_i, y_i}, {{x_i-1, y_i-1}, {x_i, y_i}, {x_i+1, y_i+1}, d, e}]
Out[∘]= {x_1+i, y_1+i}
```

```
In[∘]:= getClockwise[{x_i, y_i}, {{x_i-1, y_i-1}, {x_i, y_i}, {x_i+1, y_i+1}, d, e}]
Out[∘]= {x_-1+i, y_-1+i}
```

```
In[∘]:= Clear[areaOfPolygon];
        areaOfPolygon[cells_ /; Head[cells] === Association] := Map[Area@*Polygon, cells];
```

```
In[∘]:= Clear[areaPolygon];
        areaPolygon[vertices_] := Block[{edges},
          edges = Partition[vertices, 2, 1, 1];
          0.5 Abs@Total[(#[[1, 1]] * #[[2, 2]]) - (#[[2, 1]] * #[[1, 2]]) & /@ edges]
         ]
```

```
In[∘]:= randpoly = Polygon[ ⊟   Number of points: 5
                                Embedding dimension: 2
                                Type: simple polygon
                                Bounds: {{0.05, 0.98}, {0.27, 0.92}}
                                Area: 0.29                          ];
```

```
In[∘]:= Area[randpoly]
Out[∘]= 0.287052
```

```
In[∘]:= areaPolygon[MeshPrimitives[randpoly, 0] /. Point → Sequence]
Out[∘]= 0.287052
```

In[•]:=

```
Clear[perimeterOfPolygon];
perimeterOfPolygon[cells_ /; Head[cells] === Association] :=
   (Perimeter@*Polygon) /@ cells;
```

In[•]:=

```
Clear[perimeterPolygon];
perimeterPolygon[vertices_] := Block[{edges},
   edges = Partition[vertices, 2, 1, 1];
   Total[Apply[EuclideanDistance] /@ edges]
  ]
```

In[•]:=

```
Perimeter@randpoly
```

Out[•]=

```
2.4033
```

In[•]:=

```
perimeterPolygon[MeshPrimitives[randpoly, 0] /. Point → Sequence]
```

Out[•]=

```
2.4033
```

In[•]:=

```
Clear[centroidPolygon];
centroidPolygon[vertices_] := Mean[vertices]
```

In[•]:=

```
(*counterclockwise polygonQ*)
Block[{signedarea = 0, j, vertlen = 5},
  Do[
   j = Mod[i, vertlen] + 1;
   signedarea += (x_i y_j - x_j y_i),
   {i, vertlen}];
  Echo[1/2 (signedarea)]
 ];
```

$$» \frac{1}{2}(-x_2 y_1 + x_5 y_1 + x_1 y_2 - x_3 y_2 + x_2 y_3 - x_4 y_3 + x_3 y_4 - x_5 y_4 - x_1 y_5 + x_4 y_5)$$

In[•]:=

```
Clear[polyCounterClockwiseQ];
polyCounterClockwiseQ[poly_] := Block[{area = 0, j, vertLength = Length[poly]},
   Do[
    j = Mod[i, vertLength] + 1;
    area += poly[[i, 1]] * poly[[j, 2]];
    area -= poly[[j, 1]] * poly[[i, 2]],
    {i, vertLength}
   ];
   (area / 2.) > 0
  ]
```

In[∘]:=

```
(*Clear[sortCC];
sortCC[polyinds_,indTopts_,ptsToInds_]:=Block[{cent,poly},
  poly=Lookup[indTopts,polyinds];
  Lookup[ptsToInds,
   DeleteDuplicates@
    Flatten[MeshPrimitives[ConvexHullMesh[poly],1]/.Line→Sequence,1]
  ]
 ];*)
```

In[∘]:=

```
(*sort points for a convex polygon in counter-clockwise direction*)
```

In[∘]:=

```
Clear[sortPointsCC];
sortPointsCC[polyinds_, indTopts_, ptsToInds_] :=
 Block[{cent, ordering, polyPoints},
  polyPoints = Lookup[indTopts, polyinds];
  cent = Mean[polyPoints];
  ordering = Ordering[ArcTan[#[[1]], #[[2]]] &@ (# - cent) & /@ polyPoints];
  Lookup[ptsToInds, Part[polyPoints, ordering]]
 ]
```

mesh restructuring operations

In[∘]:=

```
bagged = CreateDataStructure["DynamicArray"]
```

Out[∘]=

```
DataStructure[ Type:DynamicArray
               Length:0 ]
```

T1 transition

In[∘]:=

```
(* T1 transition: neighbour switching *)
```

In[∘]:=

```
Clear@edgesforT1;
edgesforT1[edgeLs_, indToPts_, threshLength_ : 0.002] := Block[{edges, dist, bool},
   edges = Lookup[indToPts, #] & /@ edgeLs;
   dist = EuclideanDistance @@ # & /@ edges;
   bool = Thread[dist ≤ threshLength];
   {Pick[edges, bool, True], Pick[edgeLs, bool, True]}
  ];
```

In[∘]:=

```
Clear@T1transitionFn;
T1transitionFn[edges_, indToPtsAssoc_,
   vertexToCellG_, cellToVertexG_, dSep_ : 0.0075] :=
  Block[{findEdges, edgeind, connectedcellKeys, edge, newpts, cellvertIndices,
    cellvertices, pos, cellpolys, memF, keyscellP, selcellKeys, ptToCell,
    newptsindices, indToPts = indToPtsAssoc, ptsToInds, PtIndToCell, keysToMap,
    cellindicesAssoc, f1, otherkeys, f2, polysharingEdge, bag = bagged,
    vertToCellG = vertexToCellG, cellToVertG = cellToVertexG, testpts, edgechanged,
    localtopology, ls, cs, cc, ccw, trimmedtopo, ordering, edgeordered},
   {edgechanged, findEdges} = edgesforT1[edges, indToPts];
```

```
(* finding all possible edges for T1 transition *)

If[findEdges ≠ {},
 Scan[
  (edgeind = #;
    If[ContainsAll[Keys[indToPts], edgeind],
     (* should be an edge not
      connected to an edge that has already undergone a T1 *)
     connectedcellKeys = DeleteDuplicates[Flatten@
        Lookup[vertToCellG, edgeind]];
     cellvertIndices = Lookup[cellToVertG, connectedcellKeys];
     edge = Lookup[indToPts, edgeind];
     If[Length[connectedcellKeys] == 1,
      (*edge that is exposed to the void to be merged as a single vertex*)
      newpts = Mean[edge];
      newptsindices = Max[Keys@indToPts] + 1;
      KeyDropFrom[indToPts, edgeind];
      AppendTo[indToPts, newptsindices → newpts];
      bag["Append", edgeind];
      ptsToInds = AssociationMap[Reverse, indToPts];
      cellToVertG = MapAt[
        DeleteDuplicates[# /. (Alternatives @@ edgeind) → newptsindices] &,
        cellToVertG, Key[connectedcellKeys /. {z_Integer} ⧴ z]
       ],
      (*else proceed with T1 transition*)
      newpts = With[{midPt = Mean[edge]},
        midPt + dSep Normalize[(# - midPt)] & /@
         Flatten[RotationTransform[-π/2, midPt] /@ {edge}, 1]
       ];
      testpts = With[{midPt = Mean[edge]},
        midPt + 0.000001 Normalize[(# - midPt)] & /@ newpts
       ];
      pos = Position[cellvertIndices, {OrderlessPatternSequence[
           ___, First@edgeind, ___, Last@edgeind, ___]}, {1}];
      polysharingEdge = Extract[cellvertIndices, pos];
      (* the edge should not be part of any Δ *)
      If[(AllTrue[polysharingEdge, Length[#] ≠ 3 &]) &&
        ContainsNone[edgeind, Union@*Flatten@*Normal@bag],
       cellvertices = Map[Lookup[indToPts, #] &, cellvertIndices];
       cellpolys = Polygon /@ cellvertices;
       memF = Function[x, RegionMember@*DiscretizeRegion@x, Listable][
         Extract[cellpolys, pos]];
       keyscellP = Extract[connectedcellKeys, pos];
       selcellKeys = Thread[keyscellP → memF];
       ptToCell = Quiet[# → First @@ Select[selcellKeys, Function[x,
                Last[x][#]]] & /@ testpts /. HoldPattern[_ → First[]] → Nothing] ;
       (* pt to cell *)
       ptToCell = ptToCell /. Thread[testpts → newpts];
       newptsindices = Range[# + 1, # + 2] &[Max[Keys@indToPts]];
```

```
        AppendTo[indToPts, Thread[newptsindices → newpts]];
        ptsToInds = AssociationMap[Reverse, indToPts];
        bag["Append", edgeind];
        PtIndToCell = MapAt[ptsToInds, ptToCell, {All, 1}] /. Rule → List ;
        (*index to cell*)
        keysToMap = MapAt[Key, PtIndToCell, {All, 2}];
        cellindicesAssoc =
         AssociationThread[connectedcellKeys, cellvertIndices];

        f1 = Fold[MapAt[Function[x, DeleteDuplicates[x /. Thread[ edgeind →
                 #2[[1]] ]]], #1, #2[[2]]] &, cellindicesAssoc, keysToMap];
        f1 = Lookup[indToPts, #] & /@ f1;

        KeyDropFrom[indToPts, edgeind];

        otherkeys = List@*Key /@ Complement[connectedcellKeys, keyscellP];
        (*Print@Graphics[{EdgeForm[Black],
            FaceForm[Red],Polygon/@cellvertices},ImageSize→Tiny];*)

        localtopology = AssociationThread[connectedcellKeys → cellvertices];
        ls = Lookup[localtopology, First @@ otherkeys];
        cs = FirstCase[ls, Alternatives @@ edge];
        cc = getClockwise[cs, ls];
        ccw = getCounterClockwise[cs, ls];
        trimmedtopo =
         KeyDrop[localtopology, connectedcellKeys ~ Complement ~ keyscellP];
        ordering = {FirstCase[Position[trimmedtopo, cc],
           x : {Key[First@keyscellP], _} | {Key[Last@keyscellP], _} ⧴ First @@ x],
          FirstCase[Position[trimmedtopo, ccw], x : {Key[First@keyscellP], _} |
              {Key[Last@keyscellP], _} ⧴ First @@ x ]};

        If[Length[otherkeys] ⩵ 1,
         edgeordered = ordering /. Reverse[ptToCell, 2];
         f2 = MapAt[Replace[#, Alternatives @@ edge ⧴ Splice[edgeordered], {1}] &,
           f1, First@otherkeys];
         (*Print@Graphics[{EdgeForm[Black],FaceForm[Red],
             Polygon/@Values@f2},ImageSize→Tiny];*)
         ,
         If[MatchQ[ordering, {_Missing, _} | {_, _Missing}],
          ordering =
           Flatten[ordering /. _Missing ⧴ Complement[newpts, Keys@ptToCell], 1];
          edgeordered = ordering,
          edgeordered = ordering /. Reverse[ptToCell, 2];
         ];
         f2 = MapAt[Replace[#, Alternatives @@ edge ⧴ Splice[edgeordered], {1}] &,
           f1, First@otherkeys];
         f2 = MapAt[Replace[#, Alternatives @@ edge ⧴ Splice[
               Reverse[edgeordered]], {1}] &, f2, Last@otherkeys];
         (*Print@Graphics[{EdgeForm[Black],FaceForm[Red],Polygon/@
              Values@f2},ImageSize→Tiny];*)
        ];
        AppendTo[cellToVertG, Lookup[ptsToInds, #] & /@ f2];
       ];
      ];
      vertToCellG = GroupBy[
           Flatten[(Reverse[#, 2] &) @* Thread /@ Normal@cellToVertG], First → Last];
        ]) &, findEdges]
    ];
    bagged["DropAll"];
    {edgechanged, indToPts, cellToVertG, vertToCellG}
   ];
```

T2 transition

```
In[•]:= (* T2 transition: removal of cell *)
```

```
In[•]:=
Clear@cellsforT2;
cellsforT2[areaAssoc_, cellVertexG_, thresh_ : 10^-4] := Block[{keys, ls, inds},
   keys = Keys@Select[areaAssoc, # < thresh &];
   ls = Lookup[cellVertexG, keys];
   (*inds=Flatten@Position[ls,x_/;(3≤Length[x]≤10),{1}];
   (* 3 ≤ cell edges ≤ 6 *)  *)
   inds = Flatten@Position[ls, x_ /; (Length[x] == 3), {1}];
   (* cell edges == 3 *)
   If[inds ≠ {}, {keys[[inds]], ls[[inds]]}, {{}, {}}] (*cell inds, vertices*)
  ];
```

```
In[•]:=
Clear@T2TransitionFn;
T2TransitionFn[{cellsToRemove_, vertindsRemove_}, indTopts_, cellToVertexG_,
   areaPolygonAssoc_, periPolygonAssoc_] := Block[{newVertices, maxkey, newindices,
     newentries, indToPts = indTopts, ruleDisp, removeentries,
     CVG = cellToVertexG, notaCell, VertCellGrouping},
    newVertices = Mean@Lookup[indTopts, #] & /@ vertindsRemove;
    maxkey = Max@* Keys@indTopts;
    newindices = Range[maxkey + 1, maxkey + Length[newVertices]];
    newentries = Thread[newindices → newVertices];
    KeyDropFrom[indToPts, Union@Flatten[vertindsRemove]];
    AppendTo[indToPts, newentries];
    ruleDisp =
     Dispatch@Flatten[MapThread[Thread[#1 → #2] &, {vertindsRemove, newindices}]];
    removeentries = Union@Flatten@cellsToRemove;
    KeyDropFrom[CVG, removeentries];
    CVG = DeleteDuplicates /@ Replace[CVG, ruleDisp, {2}];
    notaCell = Keys@Select[Length /@ CVG, # < 3 &];
    KeyDropFrom[CVG, notaCell];
    VertCellGrouping =
     GroupBy[Flatten[(Reverse[#, 2] &) @* Thread /@ Normal@CVG], First → Last];
    {indToPts, CVG, VertCellGrouping, KeyDrop[areaPolygonAssoc,
      removeentries ~ Join ~ notaCell],
     KeyDrop[periPolygonAssoc, removeentries ~ Join ~ notaCell]}
   ] /; vertindsRemove ≠ {};
```

T3 transition

cell division

Difference from Farahdifar's paper: in his article he dilates the cells before dividing them. Here I randomly select a cell and divide it into two.

```
In[∘]:= (* probability of division based on the cell area *)
```

```
In[∘]:=
Clear[selectDivCells];
selectDivCells[areaPolygon_, areathresh_ : 2.2, thresh_ : 0.0025] :=
  Block[{candidates, pos},
   candidates = Normal@Select[areaPolygon / Mean[areaPolygon], # > areathresh &];
   pos = Position[0.1 RandomReal[1, Length@candidates], x_ /; x < thresh];
   Keys@Extract[candidates, pos]
  ];
```

```
In[∘]:= (* division events more random *)
```

```
In[∘]:=
Clear[pickcellsDiv];
pickcellsDiv[cellToVertG_, areaAssoc_] := Block[{pickcells, selcells, pos},
   pickcells = Keys@Select[Pick[areaAssoc,
       Thread[RandomReal[{0, 1}, Length[areaAssoc]] < 0.001], True], # > 0.005 &];
   pos = Position[Lookup[cellToVertG, pickcells], x_ /; Length[x] > 3, {1}];
   Extract[pickcells, pos]
  ];
```

```
In[∘]:=
Clear[cellDivision];
cellDivision[polygonind_, indToPoints_, areaAssoc_, perimAssoc_, cellToVertG_] :=
  Block[{x, y, num, matrix, xx, xy, yy, eigvals, eigVecs, maxeigpos, cent, edges, edgesL,
     intersects, intersectionPts, posIntersections, repPart, α, β, polygonPts,
     newkeys = Range[# + 1, # + 2] &[Max@Keys[indToPoints]], newPtToInds, indtoPtAssoc =
      indToPoints, ptToIndAssoc, edgeinds, contour, poly1, poly2, res, seq,
     newcells = Range[# + 1, # + 2] &[Max@Keys[areaAssoc]], CVG = cellToVertG,
     addcellsRule, polygonPtsInds, VCG, polygonptsTrans},
   VCG = GroupBy[Flatten[(Reverse[#, 2] &) @* Thread /@ Normal@CVG], First → Last];
   polygonPtsInds = CVG[polygonind];
   num = Length@polygonPtsInds;
   ptToIndAssoc = AssociationMap[Reverse, indToPoints];
   polygonPts = Lookup[indToPoints, polygonPtsInds];
   polygonptsTrans = TranslationTransform[-Mean[polygonPts]]@polygonPts;
   Evaluate[Table[{x_i, y_i}, {i, num + 1}]] =
    Append[polygonptsTrans, First@polygonptsTrans];
```

$$I_{xx} = \left(\frac{1}{12}\right) \sum_{i=1}^{num} (x_i\, y_{i+1} - x_{i+1}\, y_i)\ (y_i{}^2 + y_i\, y_{i+1} + y_{i+1}{}^2);$$

$$I_{yy} = \left(\frac{1}{12}\right) \sum_{i=1}^{num} (x_i\, y_{i+1} - x_{i+1}\, y_i)\ (x_i{}^2 + x_i\, x_{i+1} + x_{i+1}{}^2);$$

$$I_{xy} = \left(\frac{1}{24}\right) \sum_{i=1}^{num} (x_i\, y_{i+1} - x_{i+1}\, y_i)\ (x_i\, y_{i+1} + 2\, x_i\, y_i + 2\, x_{i+1}\, y_{i+1} + x_{i+1}\, y_i);$$

```
   Table[{Unevaluated[Subscript[x, j]] =.,
    Unevaluated[Subscript[y, j]] =.}, {j, num + 1}];
  matrix = ( Ixx    - Ixy
             - Ixy   Iyy );
  {eigvals, eigVecs} = Eigensystem@matrix;
  maxeigpos = Position[eigvals, Max@eigvals];
  {edges, edgeinds} = Partition[#, 2, 1, 1] & /@ {polygonPts, polygonPtsInds};
  edgesL = Line /@ edges;
  cent = centroidPolygon[polygonPts];
  intersects = RegionIntersection[
      InfiniteLine[{cent, cent + Extract[eigVecs, maxeigpos][[1]]}], #] & /@ edgesL;
  intersectionPts = Cases[intersects, {(_Real | _Integer) ..}, {3}];
  newPtToInds = Thread[intersectionPts → newkeys];
  posIntersections = Flatten@Position[intersects, _Point, {1}];
  MapThread[
    (res = Complement[Intersection @@ Lookup[VCG, #2], {polygonind}];
      If[res ≠ {},
       seq = Partition[CVG[First@res], 2, 1, 1];
       AppendTo[CVG,
        First@res → DeleteDuplicates@
          Flatten@SequenceSplit[seq, {x___, p : {OrderlessPatternSequence[
                 #2[[1]], #2[[-1]]]}, y___} ⧴ {x, Insert[p, #1, 2], y}]
       ];
      ]) &, {newkeys, edgeinds[[posIntersections]]}];

  repPart =
   Thread[{Thread[{ReverseSort@posIntersections, 2}], Reverse[intersectionPts]}];
  {α, β} = intersectionPts;
  AppendTo[ptToIndAssoc, newPtToInds];
  AppendTo[indtoPtAssoc, Reverse[newPtToInds, 2]];
  contour = DeleteDuplicates@
    Flatten[Fold[Insert[#1, #2[[2]], #2[[1]]] &, edges, repPart], 1];
  poly1 = Join @@ SequenceCases[contour, {___, α} | {β, ___}];
  poly2 = Join @@ SequenceCases[contour, {α, __, β}];
  KeyDropFrom[CVG, polygonind];
  addcellsRule = Thread[newcells → {poly1, poly2}];
  AppendTo[CVG, addcellsRule /. ptToIndAssoc];
  {indtoPtAssoc, CVG, Append[KeyDrop[areaAssoc, polygonind],
    MapAt[Area@*Polygon, addcellsRule, {All, 2}]],
   Append[KeyDrop[perimAssoc, polygonind],
    MapAt[Perimeter@*Polygon, addcellsRule, {All, 2}]]}
 ];
```

force computation

```
In[•]:=  F_AreaElasticity[indTopts_, vertexToCellG_, cellToVertexG_, areaPolygonAssoc_] :=
          Block[{cellinds, force, vertKeys = Keys[indTopts],
            vertLs, vertex, gc, gcc, diffVec, grad, coeff},
           First@*Last@Reap@Do[
              cellinds = Lookup[vertexToCellG, i];
              force = {0, 0};
              vertex = indTopts[i];
              Do[
               vertLs = Lookup[indTopts, Lookup[cellToVertexG, j]];
               gcc = getCounterClockwise[vertex, vertLs];
               gc = getClockwise[vertex, vertLs];
               diffVec = gcc - gc;
                      1  (  0  1 )
               grad = ─  (       ).diffVec;
                      2  ( -1  0 )
               coeff = 2 ka (areaPolygonAssoc[j] - A0);
               force += coeff * grad, {j, cellinds}
              ];
              Sow@force, {i, vertKeys}]
          ]
```

```
          (  0  1 )
In[•]:=   (       ).(getCounterClockwise[{x_i, y_i}, {{x_{i-1}, y_{i-1}}, {x_i, y_i}, {x_{i+1}, y_{i+1}}}] -
          ( -1  0 )
            getClockwise[{x_i, y_i}, {{x_{i-1}, y_{i-1}}, {x_i, y_i}, {x_{i+1}, y_{i+1}}}]) // MatrixForm
Out[•]//MatrixForm=
```

$$\begin{pmatrix}-y_{-1+i}+y_{1+i}\\ x_{-1+i}-x_{1+i}\end{pmatrix}$$

```
                     (  0  1 )
In[•]:=  MatrixForm[ (       ).({x_{i+1}, y_{i+1}} - {x_{i-1}, y_{i-1}})]
                     ( -1  0 )
Out[•]//MatrixForm=
```

$$\begin{pmatrix}-y_{-1+i}+y_{1+i}\\ x_{-1+i}-x_{1+i}\end{pmatrix}$$

In[∘]:=

```
F_{PerimeterElasticity}[indTopts_, vertexToCellG_, cellToVertexG_, periPolygonAssoc_] :=
 Block[{cellinds, force, vertKeys = Keys[indTopts], vertLs,
   vertex, gc, gcc, v1, v2, coeff, grad},
  First@*Last@Reap@Do[
     cellinds = Lookup[vertexToCellG, i];
     force = {0, 0};
     vertex = indTopts[i];
     Do[
      vertLs = Lookup[indTopts, Lookup[cellToVertexG, j]];
      gc = getClockwise[vertex, vertLs];
      v1 = Normalize[vertex - gc];
      gcc = getCounterClockwise[vertex, vertLs];
      v2 = Normalize[gcc - vertex];
      grad = v1 - v2;
      coeff = 2 γ (periPolygonAssoc[j] - P0);
      force += coeff * grad, {j, cellinds}
     ];
     Sow@force, {i, vertKeys}]
 ]
```

In[∘]:=

```
MatrixForm@Normalize[{x_i, y_i} - {x_j, y_j}]
```

Out[∘]//MatrixForm=

$$\begin{pmatrix} \frac{x_i-x_j}{\sqrt{\text{Abs}[x_i-x_j]^2+\text{Abs}[y_i-y_j]^2}} \\ \frac{y_i-y_j}{\sqrt{\text{Abs}[x_i-x_j]^2+\text{Abs}[y_i-y_j]^2}} \end{pmatrix}$$

In[∘]:=

```
F_{LineTension}[indTopts_, ptsToInd_, edges_] :=
 Block[{vertKeys = Keys@indTopts, $v1, $v2, v1, force, uv},
  force = AssociationThread[vertKeys → 0.];
  Do[
   {$v1, $v2} = Lookup[indTopts, i];
   uv = Normalize[$v1 - $v2];
   v1 = ptsToInd[$v1];
   force[v1] += κ * uv, {i, edges}];
  Values[force]
 ]
```

In[∘]:=

```
F_T[indTopts_, ptsToInds_, vertexToCellG_,
   cellToVertexG_, areaPolygonAssoc_, periPolygonAssoc_, edges_] := - (
    F_{AreaElasticity}[indTopts, vertexToCellG, cellToVertexG, areaPolygonAssoc] +
     F_{PerimeterElasticity}[indTopts, vertexToCellG, cellToVertexG, periPolygonAssoc] +
     F_{LineTension}[indTopts, ptsToInds, edges]);
```

create mesh

In[∘]:=

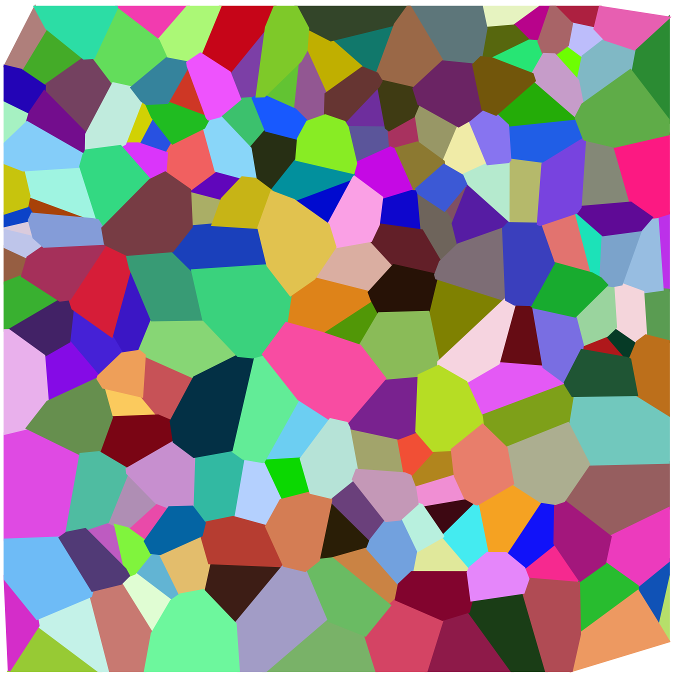
```
SeedRandom[3];
mesh = VoronoiMesh[RandomReal[1, {200, 2}], {{0, 1}, {0, 1}}, ImageSize → Medium];
```

In[∘]:=

```
pts = MeshPrimitives[mesh, 0] /. Point → Sequence;
```

```
In[∘]:= cornerpts = pts[[-4 ;;]];
        pts = pts[[1 ;; -5]];

In[∘]:= $ptsToInd = ptsToInd = AssociationThread[pts → Range@Length@pts];
        $indTopts = indTopts = AssociationMap[Reverse][ptsToInd];

In[∘]:= cellmeshprim = MeshPrimitives[mesh, 2];
        cells = (MeshPrimitives[#, 0] & /@ cellmeshprim) /. Point → Sequence /.
            Thread[cornerpts → Nothing];

In[∘]:= $cellToVertexG =
          cellToVertexG = AssociationThread[Range[Length@cells] → Map[ptsToInd, cells, {2}]];
        $vertexToCell = vertexToCell =
            GroupBy[Flatten[(Reverse[#, 2] &) @* Thread /@ Normal@cellToVertexG], First → Last];

In[∘]:= Graphics[Map[{RandomColor[], Polygon@Lookup[indTopts, #]} &, Values@cellToVertexG],
         ImageSize → Medium]
```

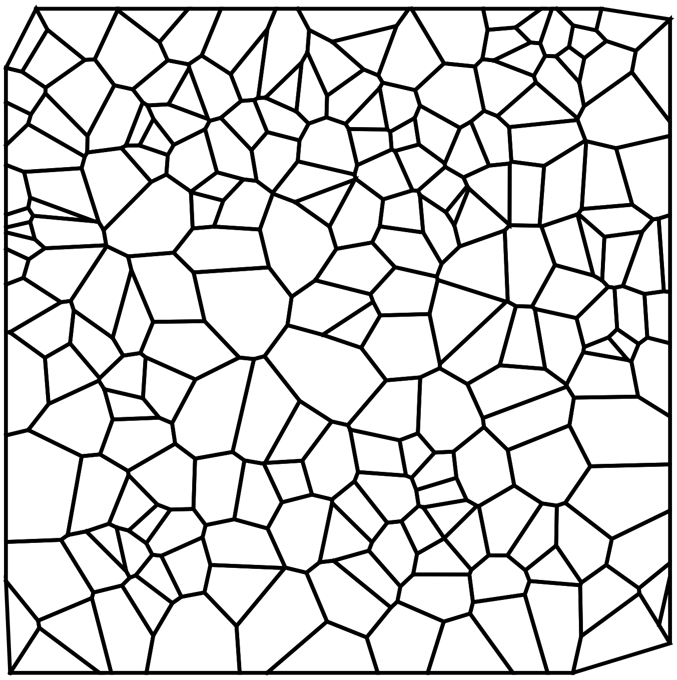
Out[∘]=



```
In[∘]:= (*edges=Flatten[Map[Partition[#,2,1,1]&,Values[cellToVertexG]],1];*)
        $cellToPts = cellToPts = Lookup[indTopts, #] & /@ cellToVertexG;
        $periPolygonAssoc = periPolygonAssoc = perimeterPolygon /@ cellToPts;
        $areaPolygonAssoc = areaPolygonAssoc = areaPolygon /@ cellToPts;

In[∘]:= Clear[plt, indTopts, ptsToInd, vertexToCell,
          cellToVertexG, periPolygonAssoc, areaPolygonAssoc, cellToPts, edges];
```

```
In[]:= pltOriginal = Graphics[{Black, Thick,
          Values@Map[Line[Join[##, {First@#}]] &@Lookup[$indTopts, #] &, $cellToVertexG]}]
```

Out[]=



run simulation 1

```
In[]:= (* PARAMETERS*)
```

```
In[]:= ka = 1; A0 = 1; γ = 0.04 * ka * A0; δt = 0.0013; P0 = 0; κ = 0.025;
```

```
In[]:= ka = 1000; A0 = 0.01; γ = 0.04 * ka * A0; δt = 0.0008; P0 = 0; κ = 0.025;
```

In[]:= $\{\gamma / (\text{ka A0}), \kappa / (\text{ka A0}^{3/2})\}$

```
Out[]= {0.04, 0.025}
```

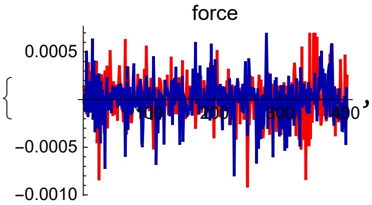
```
In[]:= Tmax = 4000;
       t = δt;
       indTopts = $indTopts;
       ptsToInd = $ptsToInd;
       vertexToCell = $vertexToCell;
       cellToVertexG = $cellToVertexG;
       periPolygonAssoc = $periPolygonAssoc;
       areaPolygonAssoc = $areaPolygonAssoc;
       cellToPts = $cellToPts;
       edges = DeleteDuplicatesBy[
           Flatten[Map[Partition[#, 2, 1, 1] &, Values@$cellToVertexG], 1], Sort];
```

```
In[]:= tt = {};
       SeedRandom[1];
       Module[{cellsToRemove, vertsToRemove, edgechanged, polydiv},
         saveres = First@Last@Reap@Monitor[
While[t ≤ Tmax δt,
 (* T2 transitions *)
 cj = Round[t / δt];
 {cellsToRemove, vertsToRemove} =
  cellsforT2[areaPolygonAssoc, cellToVertexG];
 If[cellsToRemove ≠ {},
  {indTopts, cellToVertexG, vertexToCell, areaPolygonAssoc,
    periPolygonAssoc} = T2TransitionFn[{cellsToRemove, vertsToRemove},
    indTopts, cellToVertexG, areaPolygonAssoc, periPolygonAssoc]
 ];

 (* T1 transitions *)
 edges = DeleteDuplicatesBy[
   Flatten[Map[Partition[#, 2, 1, 1] &, Values[cellToVertexG]], 1], Sort];
 {edgechanged, indTopts, cellToVertexG, vertexToCell} =
  T1transitionFn[edges, indTopts, vertexToCell, cellToVertexG];
 cellToPts = Lookup[indTopts, #] & /@ cellToVertexG;
 areaPolygonAssoc = areaPolygon /@ cellToPts;
 periPolygonAssoc = perimeterPolygon /@ cellToPts;

 (* Divisions *)
 polydiv = selectDivCells[areaPolygonAssoc];
 (*polydiv=pickcellsDiv[cellToVertexG,areaPolygonAssoc];*)
 If[polydiv ≠ {},
  Scan[
   ({indTopts, cellToVertexG, areaPolygonAssoc, periPolygonAssoc} =
      cellDivision[#, indTopts, areaPolygonAssoc,
       periPolygonAssoc, cellToVertexG]) &,
   polydiv];
  vertexToCell = GroupBy[Flatten[
     (Reverse[#, 2] &) @* Thread /@ Normal@cellToVertexG], First → Last];
 ];
 ptsToInd = AssociationMap[Reverse, indTopts];
 edges = DeleteDuplicatesBy[
   Flatten[Map[Partition[#, 2, 1, 1] &, Values[cellToVertexG]], 1], Sort];
 (* update positions *)
 If[cj == 5 || (cj + 1) == Tmax,
  AppendTo[tt, F_T[indTopts, ptsToInd, vertexToCell, cellToVertexG,
     areaPolygonAssoc, periPolygonAssoc, edges] δt]];

 indTopts = AssociationThread[
   Keys[indTopts] → (Values[indTopts] + F_T[indTopts, ptsToInd, vertexToCell,
        cellToVertexG, areaPolygonAssoc, periPolygonAssoc, edges] δt)];
 cellToPts = Lookup[indTopts, #] & /@ cellToVertexG;
 areaPolygonAssoc = areaPolygon /@ cellToPts;
 periPolygonAssoc = perimeterPolygon /@ cellToPts;
 (*plt=
   Graphics[{ColorData[1][1],Thick,Values@Map[Line[Join[##,{First@#}]]&@
         Lookup[indTopts,#]&,cellToVertexG]},ImageSize→Medium];*)
 (*plt=Graphics[{FaceForm[LightBlue],EdgeForm[{Black}],Values[
      Polygon@Lookup[indTopts,#]&/@cellToVertexG]},ImageSize→Large];*)
 plt = Graphics[{■, Thick, Values@Map[Line[Join[##, {First@#}]] &@
        Lookup[indTopts, #] &, cellToVertexG]}, ImageSize → Medium];
 If[Mod[cj, 2] == 0, Sow[plt]];
 t += δt;
```

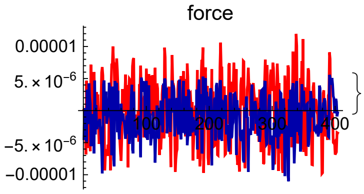
```
        ], {cj, plt}
       ]
  ];
```

In[∘]:=
```
(ListLinePlot[#, PlotStyle → {{Thickness[0.01], Red}, {Thickness[0.01], Darker@Blue}},
     PlotLabel → "force"] &) @* Transpose /@ tt
```

Out[∘]= {



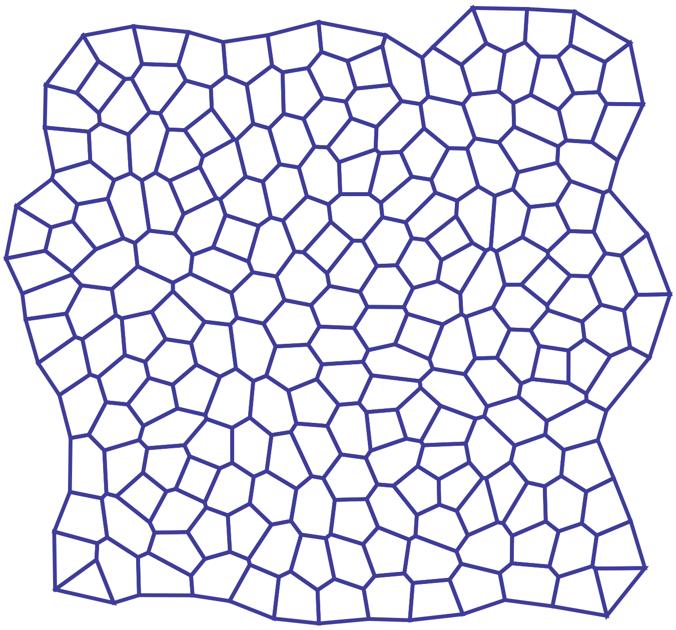
,



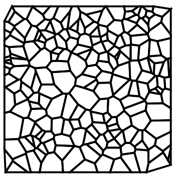
}

In[∘]:=
```
Graphics[{🟦, Thick,
   Values@Map[Line[Join[##, {First@#}]] &@Lookup[indTopts, #] &, cellToVertexG]},
  ImageSize → Medium]
```

Out[∘]=

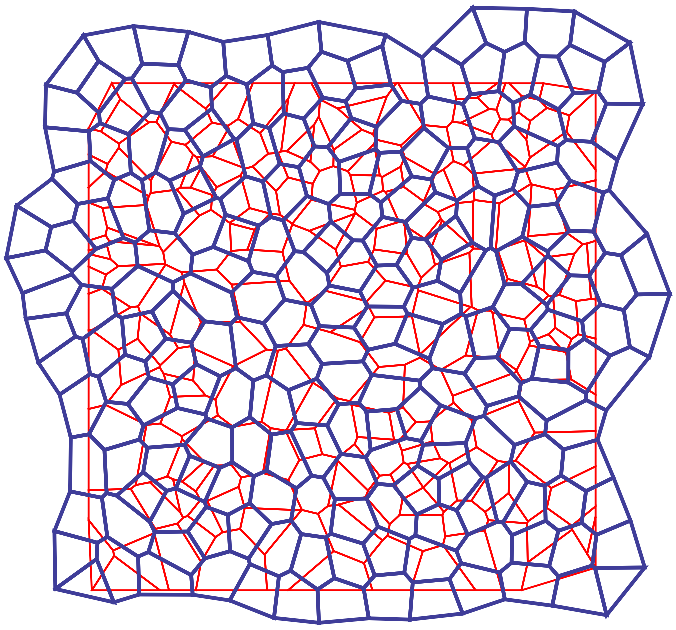


In[∘]:=



```
Show[  /. Black → Red, Graphics[{🟦, Thick,
      Values@Map[Line[Join[##, {First@#}]] &@Lookup[indTopts, #] &, cellToVertexG]}]]
```

Out[∘]=



run simulation 2

In[∘]:=

```
ka = 1000; A0 = 0.01; γ = 0.015 * ka * A0; P0 = 0; ϰ = -0.025; δt = 0.0001;
```

In[∘]:=

```
Tmax = 4000;
t = δt;
indTopts = $indTopts;
ptsToInd = $ptsToInd;
vertexToCell = $vertexToCell;
cellToVertexG = $cellToVertexG;
periPolygonAssoc = $periPolygonAssoc;
areaPolygonAssoc = $areaPolygonAssoc;
cellToPts = $cellToPts;
edges = DeleteDuplicatesBy[
   Flatten[Map[Partition[#, 2, 1, 1] &, Values@$cellToVertexG], 1], Sort];
```

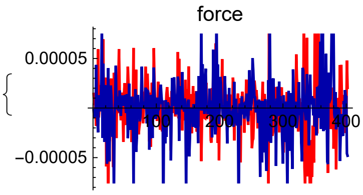
In[∘]:=

```
tt = {};
SeedRandom[1];
Module[{cellsToRemove, vertsToRemove, edgechanged, polydiv},
  saveres = First@Last@Reap@Monitor[
       While[t ≤ Tmax δt,
        (* T2 transitions *)
        cj = Round[t / δt];
        {cellsToRemove, vertsToRemove} =
          cellsforT2[areaPolygonAssoc, cellToVertexG];
         If[cellsToRemove ≠ {},
          {indTopts, cellToVertexG, vertexToCell, areaPolygonAssoc,
            periPolygonAssoc} = T2TransitionFn[{cellsToRemove, vertsToRemove},
            indTopts, cellToVertexG, areaPolygonAssoc, periPolygonAssoc]
         ];

         (* T1 transitions *)
         edges = DeleteDuplicatesBy[
           Flatten[Map[Partition[#, 2, 1, 1] &, Values[cellToVertexG]], 1], Sort];
         {edgechanged, indTopts, cellToVertexG, vertexToCell} =
          T1transitionFn[edges, indTopts, vertexToCell, cellToVertexG];
         cellToPts = Lookup[indTopts, #] & /@ cellToVertexG;
         areaPolygonAssoc = areaPolygon /@ cellToPts;
         periPolygonAssoc = perimeterPolygon /@ cellToPts;

         (* Divisions *)
         polydiv = selectDivCells[areaPolygonAssoc];
         (*polydiv=pickcellsDiv[cellToVertexG,areaPolygonAssoc];*)
         If[polydiv ≠ {},
          Scan[
           ({indTopts, cellToVertexG, areaPolygonAssoc, periPolygonAssoc} =
              cellDivision[#, indTopts, areaPolygonAssoc,
               periPolygonAssoc, cellToVertexG]) &,
           polydiv];
          vertexToCell = GroupBy[Flatten[
             (Reverse[#, 2] &) @* Thread /@ Normal@cellToVertexG], First → Last];
         ];
         ptsToInd = AssociationMap[Reverse, indTopts];
         edges = DeleteDuplicatesBy[
           Flatten[Map[Partition[#, 2, 1, 1] &, Values[cellToVertexG]], 1], Sort];
         (* update positions *)
         If[cj == 5 || (cj + 1) == Tmax,
          AppendTo[tt, F_T[indTopts, ptsToInd, vertexToCell, cellToVertexG,
             areaPolygonAssoc, periPolygonAssoc, edges] δt]];

         indTopts = AssociationThread[
           Keys[indTopts] → (Values[indTopts] + F_T[indTopts, ptsToInd, vertexToCell,
                cellToVertexG, areaPolygonAssoc, periPolygonAssoc, edges] δt)];
         cellToPts = Lookup[indTopts, #] & /@ cellToVertexG;
         areaPolygonAssoc = areaPolygon /@ cellToPts;
         periPolygonAssoc = perimeterPolygon /@ cellToPts;
         (*plt=
           Graphics[{ColorData[1][1],Thick,Values@Map[Line[Join[##,{First@#}]]&@
                 Lookup[indTopts,#]&,cellToVertexG]},ImageSize→Medium];*)
         (*plt=Graphics[{FaceForm[LightBlue],EdgeForm[{Black}],Values[
              Polygon@Lookup[indTopts,#]&/@cellToVertexG]},ImageSize→Large];*)
         plt = Graphics[{■, Thick, Values@Map[Line[Join[##, {First@#}]] &@
                Lookup[indTopts, #] &, cellToVertexG]}, ImageSize → Medium];
         If[Mod[cj, 2] == 0, Sow[plt]];
         t += δt;
        ], {cj, plt}
       ]
  ];
```

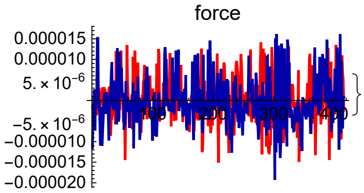
```
(*  with soft network parameter *)
```

In[∘]:=
```
(ListLinePlot[#, PlotStyle → {{Thickness[0.01], Red}, {Thickness[0.01], Darker@Blue}},
    PlotLabel → "force"] &) @* Transpose /@ tt
```

Out[∘]= {



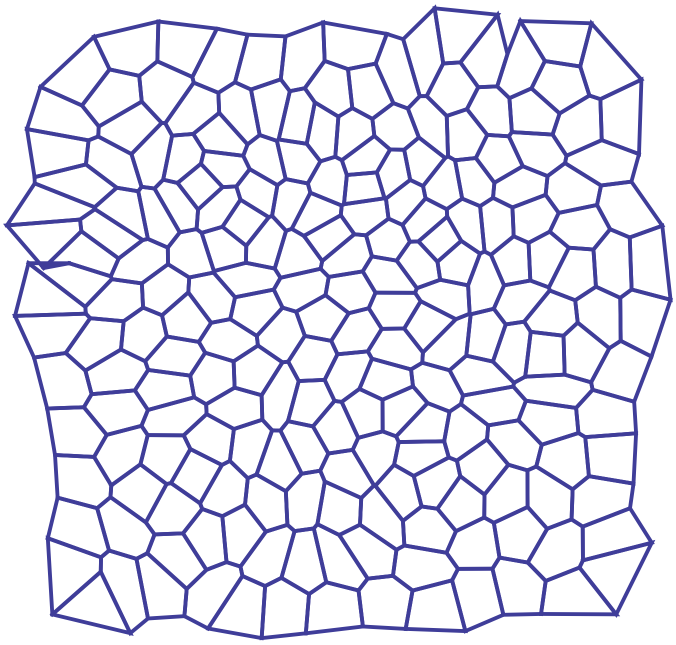
,

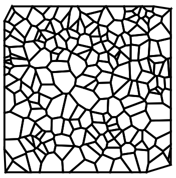


}

In[∘]:=
```
Graphics[{🟦, Thick,
  Values@Map[Line[Join[##, {First@#}]] &@Lookup[indTopts, #] &, cellToVertexG]},
 ImageSize → Medium]
```

Out[∘]=



In[∘]:=


```
Show[ /. Black → Red, Graphics[{■, Thick,
    Values@Map[Line[Join[##, {First@#}]] &@Lookup[indTopts, #] &, cellToVertexG]}]]
```

Out[∘]=