2D VERTEX MODEL

geometrical f(x)

In[•]:=
```
Clear[getCounterClockwise];
getCounterClockwise[vertex_, vertices_] := Block[{pos, v},
   pos = First @@ Position[vertices, vertex];
   If[pos == Length[vertices], pos = 1, pos += 1];
   vertices[[pos]]
  ];
```

In[•]:=
```
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[•]:=
```
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]
 ]
```

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

```
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
 ]
```

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

```
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

T1 transition (neighbour swapping)

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

```
Clear@edgesforT1;
edgesforT1[edgeLs_, indToPts_, threshLength_ : 0.0015] := Block[{edges, dist},
   edges = Lookup[indToPts, #] & /@ edgeLs;
   dist = EuclideanDistance @@ # & /@ edges;
   {Pick[edges, Thread[dist ≤ threshLength], True],
    Pick[edgeLs, Thread[dist ≤ threshLength], True]}
  ];
```

```
Clear@T1transitionFn;
T1transitionFn[edges_, indToPtsAssoc_, vertexToCellG_, cellToVertexG_, dSep_ : 0.01] :=
```

```
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 = CreateDataStructure["DynamicArray"], vertToCellG = vertexToCellG,
  cellToVertG = cellToVertexG, testpts, edgechanged},
 {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@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]];
            KeyDropFrom[indToPts, edgeind];
            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];
            otherkeys = List@*Key /@ Complement[connectedcellKeys, keyscellP];
            f2 = MapAt[(# /. (Alternatives @@ edgeind) → Splice[newptsindices] //
                  sortPointsCC[#, indToPts, ptsToInds] &) &, f1, otherkeys];
            AppendTo[cellToVertG, f2];
           ];
          ];
          vertToCellG = GroupBy[
            Flatten[(Reverse[#, 2] &) @* Thread /@ Normal@cellToVertG], First → Last];
         ]) &, findEdges]
    ];
    {edgechanged, indToPts, cellToVertG, vertToCellG}
   ];
```

T2 transition

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

In[ ]:=
```
Clear@cellsforT2;
cellsforT2[areaAssoc_, cellVertexG_, thresh_ : 10^-5] := Block[{keys, ls, inds},
   keys = Keys@Select[areaAssoc, # < thresh &];
   ls = Lookup[cellVertexG, keys];
   inds = Flatten@Position[ls, x_ /; (3 ≤ Length[x] ≤ 6), {1}];
   (* 3 ≤ cell edges ≤ 6 *)
   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

In[ ]:=
```
T3candidates[vertexToCell_, indTopts_, cellToVertexG_] :=
  Block[{outervertindices, outercellsinds, outercells, regmem, outerverticespts},
   {outervertindices, outercellsinds} = Through[{Keys, Union@*Flatten@*Values}[#]] &@
     Select[vertexToCell, Length[#] < 3 &];
   outercells = Lookup[indTopts, cellToVertexG@#] & /@ outercellsinds;
   regmem = SignedRegionDistance@*Polygon /@ outercells;
   outerverticespts = Lookup[indTopts, outervertindices];
   {Position[(Thread[(#[outerverticespts] < 0)] & /@ regmem), True],
    outercellsinds, outercells, outervertindices}
  ];
```

In[ ]:=
```
T3Transition[markers_, outercellsinds_,
   outercells_, outervertindices_, vertToCell_, pToI_, ItoP_, CVG_] :=
  Block[{ci, vi, minorcellind, vert, vertexToCell = vertToCell, numcells,
    majorcellind, intersectcell, ptsToInd = pToI, majorcell, cellToVertexG = CVG,
    commonvertexQ, edgespartof, indTopts = ItoP, edgecoords, edgesminorcell,
    lines, intersects, fpos, intersectpts, newptsindices, ls},
   If[markers ≠ {},
    Do[
     (*take the marker and handle cases *)
     {ci, vi} = marker;
```

```
minorcellind = outercellsinds[[ci]];
vert = outervertindices[[vi]];
majorcellind = If[Head[#] === Integer, numcells = 1;
    #, numcells = Length[#];
    #] &@Replace[Lookup[vertexToCell, vert], {z_Integer} :> z];
intersectcell = Lookup[ptsToInd, outercells[[ci]]];
majorcell = Lookup[cellToVertexG, majorcellind];
Print[Flatten@{majorcellind, minorcellind}];
commonvertexQ = If[numcells == 1,
   (Union[Flatten@Cases[Partition[majorcell, 2, 1, 1],
        {OrderlessPatternSequence[vert, _]}]] ⋂ intersectcell),
   Function[ (Union[Flatten@Cases[Partition[#, 2, 1, 1],
       {OrderlessPatternSequence[vert, _]}]] ⋂ intersectcell) ] /@ majorcell
  ] // (If[# ≠ {}, First@#, {}] &) @* Flatten;
Which[
 (*Case A*)
 numcells == 1 && commonvertexQ === {},
 edgespartof = Cases[Partition[cellToVertexG[majorcellind], 2, 1, 1],
   {OrderlessPatternSequence[vert, _]}];
 edgecoords = Lookup[indTopts, #] & /@ edgespartof;
 edgesminorcell = Partition[cellToVertexG[minorcellind], 2, 1, 1];
 lines = Map[Line@Lookup[indTopts, #] &, edgesminorcell];
 intersects =
  Map[Function[x, Map[RegionIntersection[Line[x], #] &, lines]], edgecoords];
 fpos = Last@FirstPosition[intersects, _Point, {2}];
 intersectpts = Cases[intersects, {__?NumberQ}, {-2}];
 newptsindices = Range[Max[ptsToInd] + 1, Max[ptsToInd] + 2];
 AppendTo[indTopts, Thread[newptsindices → intersectpts]];
 KeyDropFrom[indTopts, vert];
 ptsToInd = AssociationMap[Reverse, indTopts];
 cellToVertexG = MapAt[sortPointsCC[
     Flatten[# /. Thread[vert → {newptsindices}]], indTopts, ptsToInd] &,
   cellToVertexG, Key[majorcellind]];
 cellToVertexG = MapAt[
   Block[{y},
     y = Partition[#, 2, 1, 1];
     sortPointsCC[DeleteDuplicates@
       Flatten@Replace[y, {x : OrderlessPatternSequence @@ y[[fpos]]} :>
          Flatten@Insert[{x}, newptsindices, 2], {1}], indTopts, ptsToInd]
    ] &, cellToVertexG, Key[minorcellind]];
 ,
 (*Case B*)
 numcells == 2 && commonvertexQ === {},
 edgesminorcell = Partition[cellToVertexG[minorcellind], 2, 1, 1];
 lines = Map[Line@Lookup[indTopts, #] &, edgesminorcell];
 Do[
  edgespartof = Cases[Partition[cellToVertexG[majcelliter], 2, 1, 1],
    {OrderlessPatternSequence[vert, _]}];
  edgecoords = Lookup[indTopts, #] & /@ edgespartof;
  intersects =
   Map[Function[x, Map[RegionIntersection[Line[x], #] &, lines]], edgecoords];
```

```
 fpos = Last@FirstPosition[intersects, _Point, {2}];
 intersectpts = Cases[intersects, {__?NumberQ}, {-2}];
 Scan[(If[KeyFreeQ[ptsToInd, #],
     newptsindices = Max[ptsToInd] + 1;
     ptsToInd[#] = newptsindices;
     indTopts[newptsindices] = #]) &, intersectpts];
 newptsindices = Lookup[ptsToInd, intersectpts];
 cellToVertexG = MapAt[sortPointsCC[
     Flatten[# /. Thread[vert → {newptsindices}]], indTopts, ptsToInd] &,
   cellToVertexG, Key[majcelliter]]
 , {majcelliter, majorcellind}
];
KeyDropFrom[indTopts, vert];
ptsToInd = AssociationMap[Reverse, indTopts];
cellToVertexG = MapAt[
  Block[{y},
    y = Partition[#, 2, 1, 1];
    sortPointsCC[DeleteDuplicates@
      Flatten@Replace[y, {x : OrderlessPatternSequence @@ y[[fpos]]} :>
         Flatten@Insert[{x}, newptsindices, 2], {1}], indTopts, ptsToInd]
   ] &, cellToVertexG, Key[minorcellind]],
(*Case C*)
numcells == 1 && commonvertexQ =!= {},
edgespartof = Cases[Partition[cellToVertexG[majorcellind], 2, 1, 1],
  {OrderlessPatternSequence[vert, _]}];
edgecoords = Lookup[indTopts, #] & /@ edgespartof;
edgesminorcell = Partition[cellToVertexG[minorcellind], 2, 1, 1];
lines = Map[Line@Lookup[indTopts, #] &, edgesminorcell];
intersects =
 Map[Function[x, Map[RegionIntersection[Line[x], #] &, lines]], edgecoords];
fpos = Last@FirstPosition[intersects, _Point, {2}];
intersectpts = First@
  DeleteCases[Cases[intersects, {__?NumberQ}, {-2}], indTopts[commonvertexQ]];
newptsindices = Max[ptsToInd] + 1;
AppendTo[indTopts, newptsindices → intersectpts];
KeyDropFrom[indTopts, {vert, commonvertexQ}];
ptsToInd = AssociationMap[Reverse, indTopts];
cellToVertexG =
 MapAt[DeleteDuplicates[# /. (commonvertexQ | vert) → newptsindices] &,
  cellToVertexG, Key[majorcellind]];
cellToVertexG = MapAt[# /. commonvertexQ → newptsindices &,
  cellToVertexG, Key[minorcellind]],
(*Case D*)
numcells == 2 && commonvertexQ =!= {},
ls = {};
edgesminorcell = Partition[cellToVertexG[minorcellind], 2, 1, 1];
lines = Map[Line@Lookup[indTopts, #] &, edgesminorcell];
Do[
 edgespartof = Cases[Partition[cellToVertexG[majcelliter], 2, 1, 1],
   {OrderlessPatternSequence[vert, _]}];
 edgecoords = Lookup[indTopts, #] & /@ edgespartof;
```

```
      intersects =
       Map[Function[x, Map[RegionIntersection[Line[x], #] &, lines]], edgecoords];
      fpos = Last@FirstPosition[intersects, _Point, {2}];
      intersectpts =
       DeleteCases[Cases[intersects, {__?NumberQ}, {-2}], indTopts[commonvertexQ]];
      If[Length[intersectpts] == 2, AppendTo[ls, intersectpts]];
      Scan[(If[KeyFreeQ[ptsToInd, #],
          newptsindices = Max[ptsToInd] + 1;
          ptsToInd[#] = newptsindices;
          indTopts[newptsindices] = #]) &, intersectpts];
      newptsindices = Lookup[ptsToInd, intersectpts];
      cellToVertexG = MapAt[
        sortPointsCC[DeleteDuplicates@
           Flatten[# /. (commonvertexQ | vert) → newptsindices], indTopts, ptsToInd] &,
        cellToVertexG, Key[majcelliter]];
      ,
      {majcelliter, majorcellind}
     ];
     cellToVertexG = MapAt[
       sortPointsCC[DeleteDuplicates@Flatten[# /.
            commonvertexQ → Lookup[ptsToInd, intersectpts]], indTopts, ptsToInd] &,
       cellToVertexG, Key[minorcellind]];
    ];
    KeyDropFrom[indTopts, {vert, commonvertexQ}];
    ptsToInd = AssociationMap[Reverse, indTopts];
    vertexToCell = GroupBy[
      Flatten[(Reverse[#, 2] &) @* Thread /@ Normal@cellToVertexG], First → Last];
    , {marker, markers}]
   ];
   {indTopts, ptsToInd, vertexToCell, cellToVertexG}
  ];
```

cell division

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 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},
   VCG = GroupBy[Flatten[(Reverse[#, 2] &) @* Thread /@ Normal@CVG], First → Last];
   polygonPtsInds = CVG[polygonind];
   num = Length@polygonPtsInds;
   ptToIndAssoc = AssociationMap[Reverse, indToPoints];
   polygonPts = Lookup[indToPoints, polygonPtsInds];
   Evaluate[Table[{xi, yi}, {i, num + 1}]] = Append[polygonPts, First@polygonPts];
   Ixx = (1/12) Sum_{i=1}^{num} (xi yi+1 - xi+1 yi) (yi^2 + yi yi+1 + yi+1^2);
   Iyy = (1/12) Sum_{i=1}^{num} (xi yi+1 - xi+1 yi) (xi^2 + xi xi+1 + xi+1^2);
   Ixy = (1/24) Sum_{i=1}^{num} (xi yi+1 - xi+1 yi) (xi yi+1 + 2 xi yi + 2 xi+1 yi+1 + xi+1 yi);
   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[ ]:=
```
ka = 1; A0 = 1; γ = 0.04 * ka * A0; δt = 0.02; P0 = 0; κ = 0.025;
```

In[ ]:=
```
F_AreaElasticity[indTopts_, vertexToCellG_, cellToVertexG_, areaPolygonAssoc_] :=
 Block[{cellinds, temp, vertKeys = Keys[indTopts],
   vertLs, vertex, gc, gcc, diffVec, grad, coeff},
  First@*Last@Reap@Do[
    cellinds = Lookup[vertexToCellG, i];
    temp = {0, 0};
    vertex = indTopts[i];
    Do[
     vertLs = Lookup[indTopts, Lookup[cellToVertexG, j]];
     gcc = getCounterClockwise[vertex, vertLs];
     gc = getClockwise[vertex, vertLs];
     diffVec = gcc - gc;
     grad = 0.5 * {{0, 1}, {-1, 0}}.diffVec;
     coeff = ka (areaPolygonAssoc[j] - A0);
     temp += grad * coeff, {j, cellinds}
    ];
    Sow@temp, {i, vertKeys}]
 ]
```

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

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

In[ ]:=
```
F_ActiveContraction[indTopts_, vertexToCellG_, cellToVertexG_, areaPolygonAssoc_] :=
 Block[{cellinds, temp, vertKeys = Keys@indTopts, vertLs,
   vertex, gc, gcc, diffVec, grad, coeff},
  First@*Last@Reap@Do[
     cellinds = Lookup[vertexToCellG, i];
     temp = {0, 0};
     vertex = indTopts[i];
     Do[
      vertLs = Lookup[indTopts, Lookup[cellToVertexG, j]];
      gcc = getCounterClockwise[vertex, vertLs];
      gc = getClockwise[vertex, vertLs];
      diffVec = gcc - gc;
      grad = 0.5 * {{0, 1}, {-1, 0}}.diffVec;
      coeff = 0.1 ka * (areaPolygonAssoc[j]);
      temp += grad * coeff, {j, cellinds}];
     Sow@temp, {i, vertKeys}]
 ]
```

In[ ]:=
```
F_T[indTopts_, vertexToCellG_,
   cellToVertexG_, areaPolygonAssoc_, periPolygonAssoc_] := -(
    F_AreaElasticity[indTopts, vertexToCellG, cellToVertexG, areaPolygonAssoc] +
     F_PerimeterElasticity[indTopts, vertexToCellG, cellToVertexG, periPolygonAssoc] +
     F_LineTension[indTopts, vertexToCellG, cellToVertexG] +
     F_ActiveContraction[indTopts, vertexToCellG, cellToVertexG, areaPolygonAssoc]
   );
```

create mesh and run simulation

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]];
```

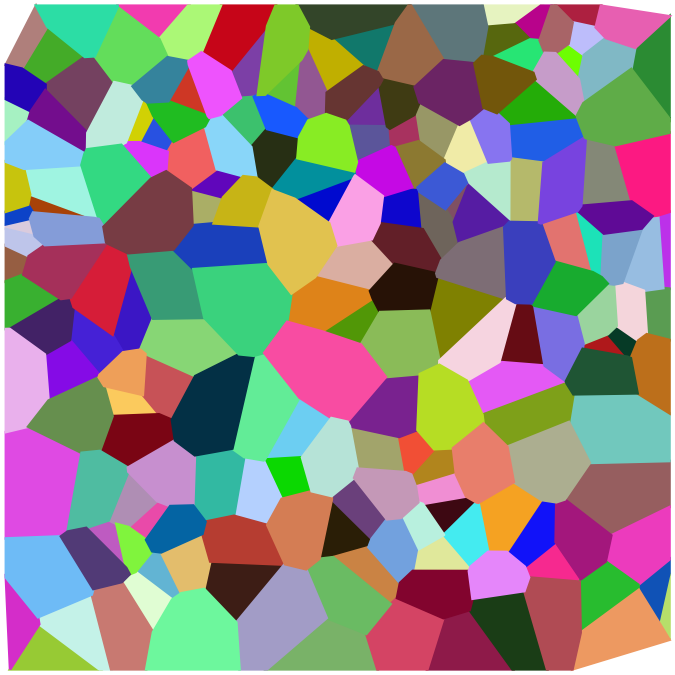
In[ ]:=
```
$ptsToInd = ptsToInd = AssociationThread[pts → Range@Length@pts];
$indTopts = indTopts = AssociationMap[Reverse][ptsToInd];
```

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[ ]=

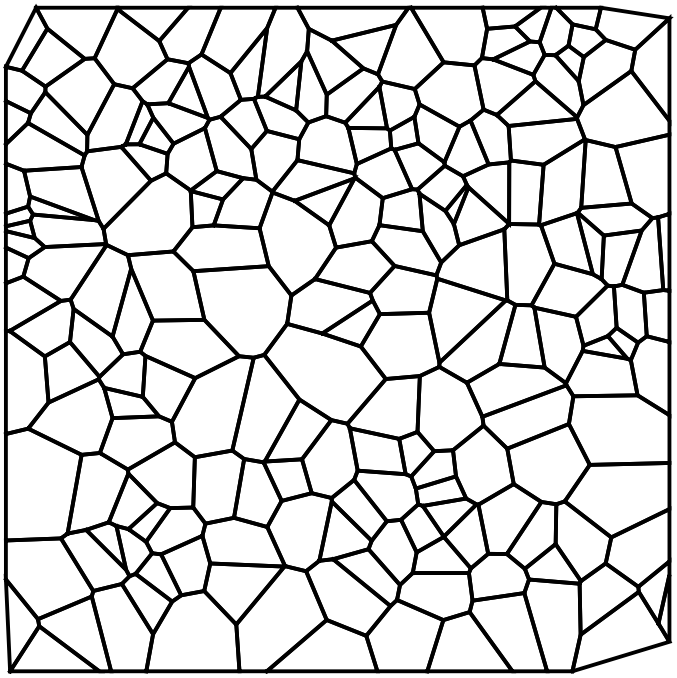


In[ ]:=
```
(*edges=Flatten[Map[Partition[#,2,1,1]&,Values[cellToVertexG]],1];*)
$cellToPts = cellToPts = Lookup[indTopts, #] & /@ cellToVertexG;
$periPolygonAssoc = periPolygonAssoc = perimeterPolygon /@ cellToPts;
$areaPolygonAssoc = areaPolygonAssoc = areaPolygon /@ cellToPts;
```

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[ ]=



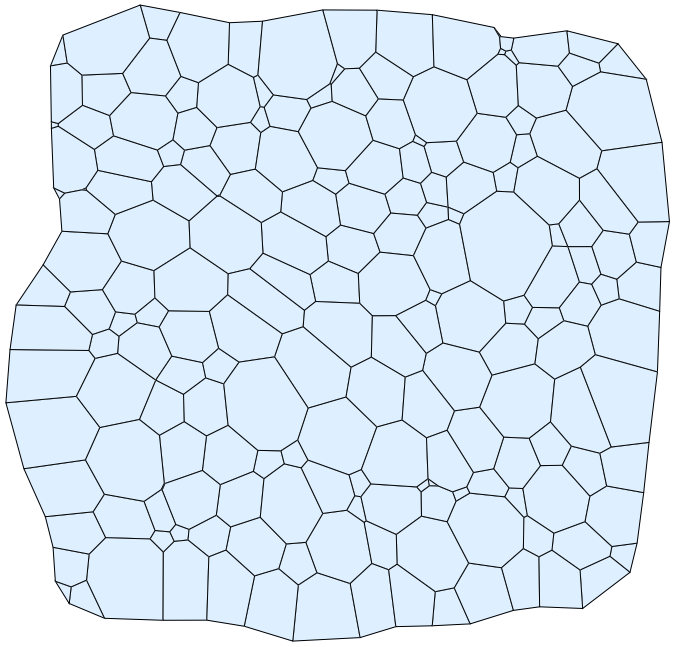
In[ ]:=
```
t = δt;
indTopts = $indTopts;
ptsToInd = $ptsToInd;
vertexToCell = $vertexToCell;
cellToVertexG = $cellToVertexG;
periPolygonAssoc = $periPolygonAssoc;
areaPolygonAssoc = $areaPolygonAssoc;
cellToPts = $cellToPts;
edges = DeleteDuplicatesBy[
   Flatten[Map[Partition[#, 2, 1, 1] &, Values@$cellToVertexG],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

In[ ]:=

```
Module[{cellsToRemove, vertsToRemove, edgechanged, polydiv},
  saveres = First@Last@Reap@Monitor[
       While[t ≤ 100 δt,
        (* T2 transitions *)
        {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];
        ];
        (* update positions *)
        indTopts = AssociationThread[Keys[indTopts] → (Values[indTopts] + F_T[indTopts,
              vertexToCell, cellToVertexG, areaPolygonAssoc, periPolygonAssoc] 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 → Medium];
        Sow[plt];
        t += δt;
       ], plt
      ]
 ];
```

In[ ]:= plt

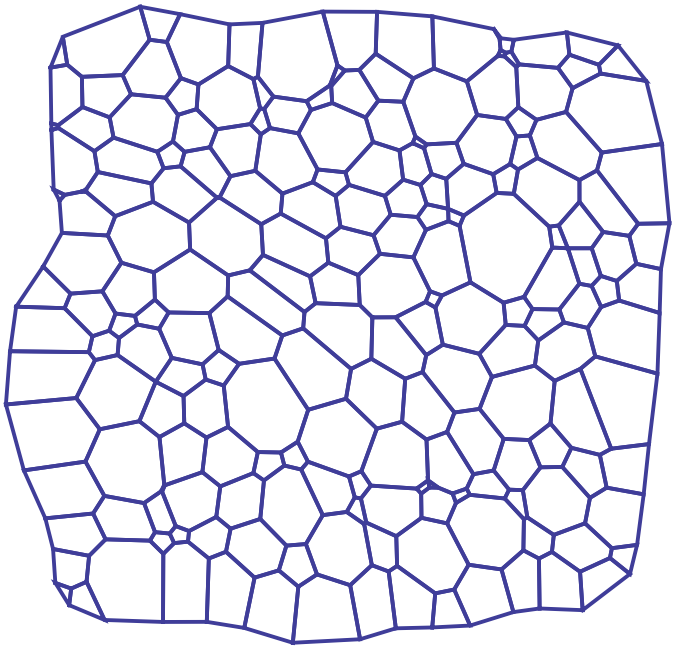
Out[ ]=

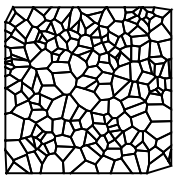


In[ ]:= (*Export["C:\\Users\\aliha\\Desktop\\result.gif",saveres,AnimationRepetitions→ ∞]*)

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[ ]=