module - topological operations

here we are multiplying some factors in the associated functions (for topological operations) so that we can check if the operations are being done correctly

init

In[∘]:=
```
DumpGet[
  "C:\\Users\\aliha\\Desktop\\wolfram-vertex-3D\\create geometry\\infinitesheet.mx"];
Names["Global`*"]
```

Out[∘]=
```
{args, cellVertexGrouping, dims, edges,
 indToPtsAssoc, ptsToIndAssoc, vertexToCell, xLim, yLim}
```

```
(*DumpGet["C:\\Users\\aliha\\Desktop\\VERTEX MODEL\\meshGen_noise.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}];
```

Initialize

In[∘]:=
```
Needs["IGraphM`"];
```

```
IGraph/M 0.3.108 (December 17, 2018)
Evaluate IGDocumentation[] to get started.
```

In[∘]:=
```
δ = 1.0;
𝒟 = Rectangle[{First@xLim, First@yLim}, {Last@xLim, Last@yLim}];
```

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[ ]:=
```
SetAttributes[orderlessHead, {Orderless}];
```

triangulate faces

In[ ]:=
```
triangulateFaces[faces_] := Block[{edgelen, ls, mean},
   (If[Length[#] ≠ 3,
       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[•]:=
```
ClearAll[meanFaces];
meanFaces = Compile[{{faces, _Real, 2}},
  Block[{facepart, edgelen, mean},
   facepart = Partition[faces, 2, 1];
   AppendTo[facepart, {facepart[[-1, -1]], faces[[1]]}];
   edgelen = Table[Norm[SetPrecision[First@i - Last@i, 10]], {i, facepart}];
   mean = Total[edgelen * (Mean /@ facepart)] / Total[edgelen];
   mean],
  RuntimeAttributes → {Listable}, CompilationTarget → "C",
  CompilationOptions → {"InlineExternalDefinitions" → True}
 ]
(*Needs["CompiledFunctionTools`"]*)
(*CompilePrint[meanFaces];*)
triangulateToMesh[faces_] := Block[{mf, partfaces},
   mf = SetPrecision[meanFaces@faces, 10];
   partfaces = Partition[#, 2, 1, 1] & /@ faces;
   MapThread[
    If[Length[#] ≠ 3,
      Function[x, Join[x, {#2}]] /@ #1,
      {#[[All, 1]]}
     ] &, {partfaces, mf}]
  ];
```

Out[•]= CompiledFunction[Argument count: 1 Argument types: {{_Real, 2}}]

Get Local Topology

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}
 ];
```

Topological/Network operations

In[ ]:=
```
(* tests to check whether 'α','β' or an invalid pattern is present *)
Clear[a, b];
$invalidPatternsEdge = Graph[{a ⟷ b, b ⟷ c, a ⟷ c}];
edgeinTrianglePatternQ[graph_] := IGSubisomorphicQ[$invalidPatternsEdge, graph];
(*checks to determine if any invalid pattern is present in the graph*)
$invalidPatterns = {Graph[{a ⟷ b, b ⟷ a}], Graph[{a ⟷ b, b ⟷ c, a ⟷ c, a ⟷ d, c ⟷ d}]};
InvalidEdgePatternQ[graph_] := AnyTrue[$invalidPatterns, IGSubisomorphicQ[#, graph] &];
InvalidTrigonalPatternQ[graph_] :=
  AnyTrue[$invalidPatterns, IGSubisomorphicQ[#, graph] &];
```

In[ ]:=
```
faceIntersections[polyhed_] := AnyTrue[
 Length /@ (Intersection @@@ Replace[Subsets[Partition[#, 2, 1, 1] & /@ polyhed, {2}],
       List → orderlessHead, {4}, Heads → True]), # ≥ 2 &];

gammaPatternFreeQ[polyhedList_] := Not[Or @@ (faceIntersections /@ polyhedList)];
```

I → Δ operator

In[ ]:=
```
ItoΔpreprocess1::description =
  "the module finds the vertices of the edge (to be converted)
    and all the points attached to it";
ItoΔpreprocess1[candidate_, currentTopology_, localTopology_] :=
  Block[{r10, r11, ptsPartitioned, vertAttached,
    cellsPartOf, cellsElim, ptsAttached},
   {r10, r11} = candidate; (* edge unpacked into vertices: r10,r11 *)
   (* r10 → {vertices attached with r10}, r11 → {vertices attached with r11} *)
   ptsPartitioned = If[Keys[#],
       r10 → Flatten[Last@#, 1], r11 → Flatten[Last@#, 1]] & /@ (
      Normal@KeySortBy[
         GroupBy[
          (currentTopology /. {OrderlessPatternSequence[r11, r10]} → Sequence[]),
          MemberQ[#, r10] &], MatchQ[False]] /. {r10 | r11 → Sequence[]});
   (* the code below creates pairings between vertices
    such that r1 is packed with r4, r2 with r5 & r3 with r6 *)
   vertAttached = Flatten[Values@ptsPartitioned, 1];
   cellsPartOf =
    Union[Position[localTopology, #, {3}] /. {Key[x_], __} ⧴ x] & /@ vertAttached;
   cellsElim = Complement[Union@Flatten[cellsPartOf],
       Union@Flatten@#[[1]] ⋂ Union@Flatten@#[[2]]] &@TakeDrop[cellsPartOf, 3];
   If[cellsElim ≠ {},
    cellsPartOf = cellsPartOf /. Alternatives @@ cellsElim → Sequence[]
   ];
   ptsAttached = Values@GroupBy[Thread[vertAttached → cellsPartOf], Last → First];
   {r10, r11, ptsAttached}
];
```

artificial factor of 0.15 in the function below

In[ ]:=
```
ItoΔpreprocess2::description =
  "the module finds the position of new vertices r7,r8 and r9";
ItoΔpreprocess2[ptsAttached_, {r10_, r11_}] :=
  Block[{r01, u1T, r1, r4, r2, r5, r3, r6, w07, w08, w09,
    v07, v08, v09, lmax, r7, r8, r9},
   r01 = Mean[{r10, r11}];
   u1T = (r10 - r11) / Norm[r10 - r11];
   {{r1, r4}, {r2, r5}, {r3, r6}} = ptsAttached;
   w07 = 0.5 ((r1 - r01) / Norm[r1 - r01] + (r4 - r01) / Norm[r4 - r01]);
   w08 = 0.5 ((r2 - r01) / Norm[r2 - r01] + (r5 - r01) / Norm[r5 - r01]);
   w09 = 0.5 ((r3 - r01) / Norm[r3 - r01] + (r6 - r01) / Norm[r6 - r01]);
   v07 = w07 - (w07.u1T) u1T;
   v08 = w08 - (w08.u1T) u1T;
   v09 = w09 - (w09.u1T) u1T;
   lmax = Max[Norm[v08 - v07], Norm[v09 - v08], Norm[v07 - v09]];
   r7 = SetPrecision[r01 + 0.15 (δ / lmax) v07, 10];
   r8 = SetPrecision[r01 + 0.15 (δ / lmax) v08, 10];
   r9 = SetPrecision[r01 + 0.15 (δ / lmax) v09, 10];
   {r1, r2, r3, r4, r5, r6, r7, r8, r9}
];
```

In[ ]:=
```
insertTrigonalFace::description = "the module inserts the trigonal face into the cell";
insertTrigonalFace[topology_, r7_, r8_, r9_, r10_, r11_] := Block[{posInserts},
   posInserts = Position[
     FreeQ[#, {___, OrderlessPatternSequence[r10, r11], ___}] & /@ topology, True];
   If[posInserts ≠ {},
    Insert[topology, {r7, r8, r9}, Flatten[{#, -1}] & /@ posInserts],
    topology]
  ];
```

In[ ]:=
```
Clear@corrTriOrientationHelper;
corrTriOrientationHelper[topology_, trigonalface_] := Block[{allTri,
    selectTriAttached, selectTriSharedEdge, selectTri, partTri, partAttachedTri},
   partTri = Partition[trigonalface, 2, 1, 1];
   allTri = Flatten[triangulateFaces@topology, 1];
   selectTriAttached =
    Cases[allTri, {OrderlessPatternSequence[__, Alternatives @@ trigonalface]}];
   selectTriSharedEdge = Select[selectTriAttached,
     Length[Intersection[#, trigonalface]] == 2 &];
   selectTri = RandomChoice@selectTriSharedEdge;
   partAttachedTri = Partition[selectTri, 2, 1, 1];
   If[Intersection[partAttachedTri, partTri] ≠ {},
    topology /. trigonalface :→ Reverse@trigonalface,
    topology
   ]
  ];
```

In[∘]:=
```
Clear@corrTriOrientation;
corrTriOrientation[localtopology_, trigonalface_] :=
  Block[{cells, affectedIDs, topo},
   cells = Map[DeleteDuplicates@* (Flatten[#, 1] &), localtopology, {2}];
   affectedIDs = Partition[First /@ Position[cells, trigonalface], 1];
   topo = MapAt[corrTriOrientationHelper[#1, trigonalface] &, cells, affectedIDs];
   Map[Partition[#, 2, 1, 1] &, topo, {2}]
  ];
```

In[∘]:=
```
ItoΔoperation::description =
  "the module removes vertices r10,r11 and connects the points
    r1-r6 with the new points r7-r9";
ItoΔoperation[graphnewLocalTopology_, cellCoords_, r1_, r2_, r3_, r4_,
   r5_, r6_, r7_, r8_, r9_, r10_, r11_] := Block[{mat},
    mat = insertTrigonalFace[cellCoords, r7, r8, r9, r10, r11];
    Map[Partition[#, 2, 1, 1] &, mat, {2}] /. {
      {OrderlessPatternSequence[r11, r10]} :> Sequence[],
      {PatternSequence[r11, q : r4 | r5 | r6]} :>
       Switch[q, r4, {r7, r4}, r5, {r8, r5}, r6, {r9, r6}],
      {PatternSequence[q : r4 | r5 | r6], r11} :>
       Switch[q, r4, {r4, r7}, r5, {r5, r8}, r6, {r6, r9}],
      {PatternSequence[r10, q : r1 | r2 | r3]} :>
       Switch[q, r1, {r7, r1}, r2, {r8, r2}, r3, {r9, r3}],
      {PatternSequence[q : r1 | r2 | r3, r10]} :>
       Switch[q, r1, {r1, r7}, r2, {r2, r8}, r3, {r3, r9}]}
   ] /; (! InvalidEdgePatternQ[graphnewLocalTopology]);
```

In[∘]:=
```
bindCellsToNewTopology[adjoiningCells_, network_, func_ : Identity] /;
   gammaPatternFreeQ[network] := Thread[adjoiningCells → func[network]];
```

```
In[ ]:= modifier::description = "the module makes
            modifications to the datastructures after topological transitions";
        modifier[candidate_, adjoiningCells_, indToPtsAssoc_,
           ptsToIndAssoc_, cellVertexGrouping_,
        vertexToCell_, celltopologicalChanges_, updatedLocalNetwork_, newAdditions_] :=
          Block[{dropVertInds, $ptsToIndAssoc = ptsToIndAssoc,
            $indToPtsAssoc = indToPtsAssoc, $cellVertexGrouping = cellVertexGrouping,
            $vertexToCell = vertexToCell},
           dropVertInds = Lookup[$ptsToIndAssoc, candidate];
           KeyDropFrom[$ptsToIndAssoc, candidate];
           KeyDropFrom[$indToPtsAssoc, dropVertInds];
           {AssociateTo[$ptsToIndAssoc, #~Reverse~2], AssociateTo[$indToPtsAssoc, #]} &@
            newAdditions;
           AssociateTo[$cellVertexGrouping, MapAt[$ptsToIndAssoc,
             celltopologicalChanges, {All, 2, All, All}]];
           KeyDropFrom[$vertexToCell, Sort@dropVertInds];
           AssociateTo[$vertexToCell,
            (First[#] → Part[adjoiningCells, Union[
                  First /@ Position[updatedLocalNetwork, Last@#, {3}]]]) & /@ newAdditions];
           {$indToPtsAssoc, $ptsToIndAssoc, $cellVertexGrouping, $vertexToCell}
          ];
```

δ being multiplied by 1.1

```
In[ ]:= Ito∆[ind_, edges_, faceListCoords_, indToPtsAssoc_,
          ptsToIndAssoc_, cellVertexGrouping_, vertexToCell_, wrappedMat_] :=
         Block[{edgelen, edgesel, candidate, graphCurrentTopology, currentTopology, z, ž,
           localtopology = {}, adjoiningcells, cellCoords, r10, r11, ptsAttached, r1, r2, r3,
           r4, r5, r6, r7, r8, r9, newLocalTopology, graphnewLocalTopology, modifiednetwork,
           cellTopologicalChanges, maxVnum, wrappedcells, celltransvecAssoc, newAdditions,
           transvec, ls, vpt, cellTopologicalChangesBeforeShift, positions, cellspartof,
           vertices, $indToPtsAssoc = indToPtsAssoc, $ptsToIndAssoc = ptsToIndAssoc,
           $cellVertexGrouping = cellVertexGrouping, $vertexToCell = vertexToCell,
           $edges = edges, $wrappedMat = wrappedMat, $faceListCoords = faceListCoords},

          edgelen = EuclideanDistance @@@ $edges;
          (*here we check the length of all the edges*)
          edgesel = Pick[$edges, 1 - UnitStep[edgelen - δ * 1.1], 1];
          (*select edges that have length less than critical value δ*)
          Scan[
           (candidate = #; (*candidate edge*)
             vertices = DeleteDuplicates@Flatten[$edges, 1];
             If[AllTrue[candidate, MemberQ[vertices, #] &],
              (*this means that the edge exists in the network.
                 If there are two adjacent edges
               that need to be transformed and one gets transformed first
               then the second one will not exist*)
              (* get all edges that are connected to our edge of interest *)
              currentTopology = Cases[$edges,
```

```
      {OrderlessPatternSequence[x_, p : Alternatives @@ candidate]} :→ {p, x}];
    (* this part of code takes care of border cells *)
    If[Length[currentTopology] < 7,
     (*Print[" # of edges is < than 7 "];*)
     (* here we get the local topology of our network *)
     {localtopology, wrappedcells, transvec} =
      getLocalTopology[$ptsToIndAssoc, $indToPtsAssoc, $vertexToCell,
        $cellVertexGrouping, $wrappedMat, $faceListCoords][candidate];
     Print[Keys@localtopology];
     (* this yields all the unique edges
      in the localtopology and extract vertex pairs, such that
      {candidate_vertex, vertex attached to candidate} *)
     With[{edg = DeleteDuplicatesBy[
          Flatten[Map[Partition[#, 2, 1, 1] &, Values@localtopology, {2}], 2], Sort]},
      currentTopology = Cases[edg,
          {OrderlessPatternSequence[x_, p : Alternatives @@ candidate]} :→ {p, x}];
     ];
    ];
    (*creating a graph from the current topology*)
    graphCurrentTopology =
     Graph@Replace[currentTopology, List → UndirectedEdge, {2}, Heads → True];
    If[edgeinTrianglePatternQ@graphCurrentTopology,
     (*edge is part of a trigonal face and
      hence nothing is to be done. this prevents `α` pattern *)
     None,
     {z, ž} = candidate; (* edge vertices unpacked *)
     If[localtopology == {},
       (* here we get the local topology of our network *)
      {localtopology, wrappedcells, transvec} =
        getLocalTopology[$ptsToIndAssoc, $indToPtsAssoc, $vertexToCell,
          $cellVertexGrouping, $wrappedMat, $faceListCoords][candidate];
     ];
     {adjoiningcells, cellCoords} = {Keys@#, Values@#} &@localtopology;
     (* adjoining cells and their vertices *)
     Print[adjoiningcells];
     (* label vertices joining the candidate edge *)
     {r10, r11, ptsAttached} =
      Ito∆preprocess1[candidate, currentTopology, localtopology];
     (* getting all vertices for transformation including (r7,r8,r9) *)
     {r1, r2, r3, r4, r5, r6, r7, r8, r9} = Ito∆preprocess2[ptsAttached, {r10, r11}];
     (*print old and predicted topology *)
     Print[
      Graphics3D[{PointSize[0.025], Red, Point@{r1, r4, r7},
        Green, Point@{r2, r5, r8}, Blue, Point@{r3, r6, r9}, Purple,
        Point@r10, Pink, Point@r11, Black, Line@currentTopology, Dashed,
        Line[{r1, r7}], Line[{r4, r7}], Line[{r2, r8}], Line[{r5, r8}],
        Line[{r3, r9}], Line[{r6, r9}], Purple, Line@ptsAttached}, ImageSize → Small]
     ];

     If[! IGSubisomorphicQ[$invalidPatternsEdge, graphCurrentTopology],
      (* atleast no α pattern will be generated. I think
```

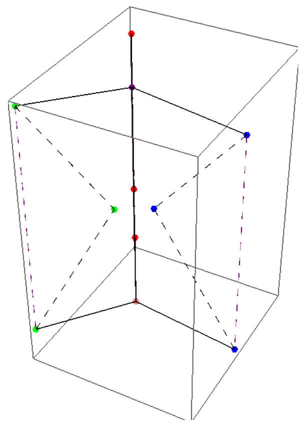
```
  this has been checked in the If statement prior to this *)
 (* Scheme: apply [I]→[H]; check if the new topology is valid (i.e. no α,β);
 check if the new topology is free of γ pattern;
 replace network architecture *)
 (*forming new topology and graph*)
 newLocalTopology = {r1 ⟷ r7, r4 ⟷ r7,
   r2 ⟷ r8, r5 ⟷ r8, r3 ⟷ r9, r6 ⟷ r9, r7 ⟷ r8, r8 ⟷ r9, r9 ⟷ r7};
 graphnewLocalTopology = Graph@newLocalTopology;
 (* apply ItoΔ operation *)
 modifiednetwork = ItoΔoperation[graphnewLocalTopology,
   cellCoords, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11];
 modifiednetwork = corrTriOrientation[modifiednetwork, {r7, r8, r9}];
 (*bind cells with their new topology if γ pattern is absent*)
 cellTopologicalChanges = bindCellsToNewTopology[adjoiningcells,
   modifiednetwork, Map[Map[DeleteDuplicates@Flatten[#, 1] &]]];

 (*print topology post operation *)
 If[(cellTopologicalChanges ≠ {}) ||
   (Head[cellTopologicalChanges] =!= bindCellsToNewTopology),
  Print[ind → Graphics3D[{{Opacity[0.1], Blue,
       Polyhedron /@ Values[cellTopologicalChanges]},
      {Red, Line@candidate}}, Axes → True]]
 ];

 If[(cellTopologicalChanges ≠ {}) ||
   (Head[cellTopologicalChanges] =!= bindCellsToNewTopology),
  (*if you are here then it means that cell topology was altered *)
  modifiednetwork = Values@cellTopologicalChanges;
  (*vertex coordinates of the modified topology*)
  maxVnum = Max[Keys@$indToPtsAssoc]; (*maximum number of vertices so far*)
  If[wrappedcells ≠ {},
   (* if there are wrapped
    cells send them back to their respective positions *)
   (* wrapped cells with their respective vectors for translation *)
   celltransvecAssoc = AssociationThread[wrappedcells, transvec];
   cellTopologicalChangesBeforeShift = cellTopologicalChanges;
   (* here we send the cells
     back to their original positions → unwrapped state *)
   cellTopologicalChanges = (x ↦ With[{p = First[x]},
        If[MemberQ[wrappedcells, p],
         p → Map[SetPrecision[# - celltransvecAssoc[p], 10] &, Last[x], {2}], x]
       ]) /@ cellTopologicalChanges;

   ls = {};
   Scan[
    vpt ↦
     (positions = Position[cellTopologicalChangesBeforeShift, vpt];
      positions = DeleteDuplicates[{First[#]} & /@ positions];
      cellspartof = Extract[adjoiningcells, positions];
      Fold[
```

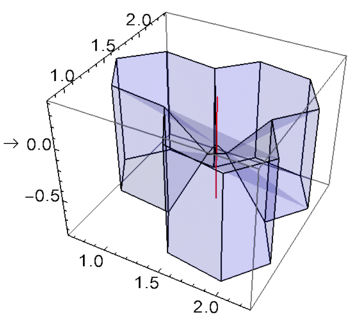
```
            Which[MemberQ[wrappedcells, #2],
              AppendTo[ls, SetPrecision[vpt - celltransvecAssoc[#2], 10] ],
              True, If[! MemberQ[ls, vpt], AppendTo[ls, vpt]]] &, ls, cellspartof]),
          {r7, r8, r9}];

         newAdditions = Thread[(Range[Length[ls]] + maxVnum) → ls],
         newAdditions = Thread[(Range[3] + maxVnum) → {r7, r8, r9}]
          (* labels for new vertices *)
        ];

        (* appropriate changes are made to the datastruct *)
        {$indToPtsAssoc, $ptsToIndAssoc, $cellVertexGrouping, $vertexToCell} =
         modifier[candidate, adjoiningcells, $indToPtsAssoc,
          $ptsToIndAssoc, $cellVertexGrouping,
          $vertexToCell, cellTopologicalChanges, modifiednetwork, newAdditions];

        $faceListCoords = Map[Lookup[$indToPtsAssoc, #] &, $cellVertexGrouping, {2}];

        $edges =
         Flatten[Map[Partition[#, 2, 1, 1] &, Map[Lookup[$indToPtsAssoc, #] &, Values[
              $cellVertexGrouping], {2}], {2}], 2] // DeleteDuplicatesBy[Sort];

        $wrappedMat = AssociationThread[Keys[$cellVertexGrouping] →
           Map[Lookup[$indToPtsAssoc, #] /. periodicRules &,
            Lookup[$cellVertexGrouping, Keys[$cellVertexGrouping]], {2}]];
       ];
      ];
     ];
    ]) &, {edgesel[[ind]]}];
  {$edges, $indToPtsAssoc, $ptsToIndAssoc, $cellVertexGrouping,
   $vertexToCell, $wrappedMat, $faceListCoords, edgesel[[ind]], adjoiningcells}
 ]
```

example edge 1

```
In[■]:= {$edges, $indToPtsAssoc, $ptsToIndAssoc, $cellVertexGrouping, $vertexToCell, $wrappedMat,
    $faceListCoords, seledge, adjoiningCells} = Ito∆[313, edges, faceListCoords,
    indToPtsAssoc, ptsToIndAssoc, cellVertexGrouping, vertexToCell, wrappedMat];
```

{23, 42, 43}



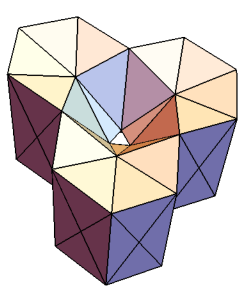
313 → 

In[∘]:=
```
adjoiningCells
```

Out[∘]= {23, 42, 43}

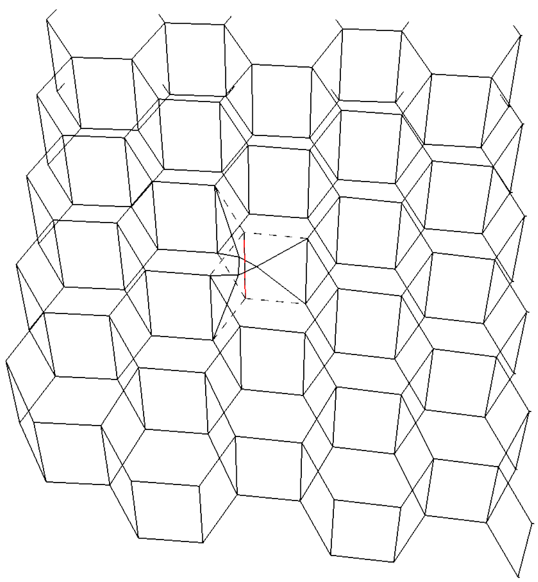
In[∘]:=
```
(triangulateFaces /@
    Map[Lookup[$indToPtsAssoc, #] &, ($cellVertexGrouping /@ adjoiningCells), {2}]) //
  Map[Polyhedron[Flatten[#, 1]] &, #] & // Graphics3D[#, ImageSize → Small, Boxed → False] &
```

Out[∘]=

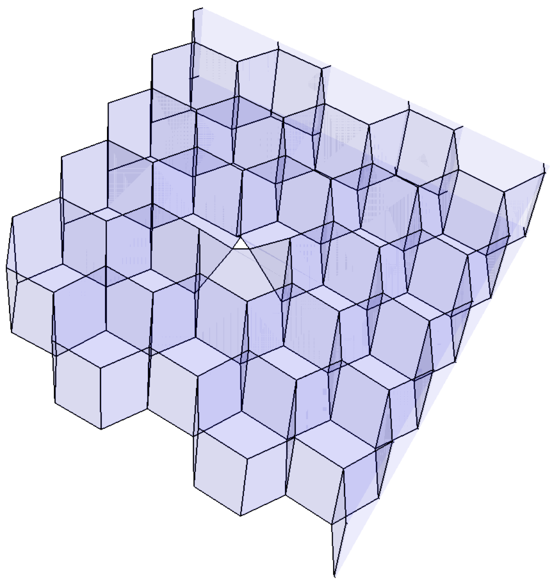


In[ ]:=
```
With[{winsize = 2},
 Graphics3D[{Black, {Dashed, Line@edges}, {Black, Line@$edges},
   Red, Line[#~Append~First@#] &@seledge}, ImageSize → Medium,
  PlotRange → ((MinMax[#] + {-winsize, winsize}) &/@Transpose@seledge), Boxed → False]
]
```

Out[ ]=



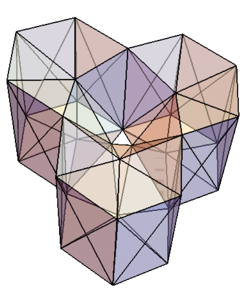
In[∘]:=
```
Block[{winsize = 2},
 Print@Graphics3D[{Opacity[0.075], Blue, Polyhedron /@ Values@$faceListCoords},
   PlotRange → ((MinMax[#] + {-winsize, winsize}) & /@ Transpose@seledge), Boxed → False]
]
```



In[∘]:=
```
Through[{Max@*Keys, Length}[#]] &@$indToPtsAssoc
```

Out[∘]= {1763, 1761}

In[∘]:=
```
Flatten[(triangulateFaces /@
    Map[Lookup[$indToPtsAssoc, #] &, ($cellVertexGrouping /@ adjoiningCells), {2}]), 1] //
 Graphics3D[{Opacity[0.5], #}, ImageSize → Small, Boxed → False] &@*Map[Polyhedron]
```

Out[∘]=


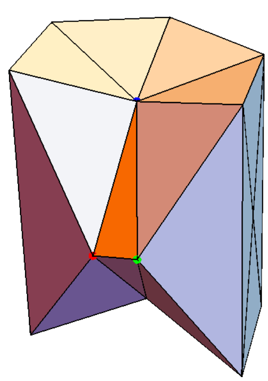
In[•]:=
```
adjoiningCells
```
Out[•]=
```
{23, 42, 43}
```

In[•]:=
```
Lookup[$cellVertexGrouping, adjoiningCells]
```
Out[•]=
```
{{{1761, 178, 21, 14, 13, 174, 1762}, {1762, 176, 17, 20, 25, 180, 1761}, {1761, 180, 178},
  {178, 180, 25, 21}, {21, 25, 20, 14}, {14, 20, 17, 13}, {13, 17, 176, 174}, {174, 176, 1762}},
 {{255, 256, 1763, 1762, 174, 173, 250}, {257, 254, 175, 176, 1762, 1763, 258},
  {255, 257, 258, 256}, {256, 258, 1763}, {1762, 176, 174},
  {174, 176, 175, 173}, {173, 175, 254, 250}, {250, 254, 257, 255}},
 {{259, 260, 181, 178, 1761, 1763, 256}, {261, 258, 1763, 1761, 180, 183, 262},
  {259, 261, 262, 260}, {260, 262, 183, 181}, {181, 183, 180, 178},
  {178, 180, 1761}, {1763, 258, 256}, {256, 258, 261, 259}}}
```

In[•]:=
```
Sort /@ Map[Length, Lookup[$cellVertexGrouping, adjoiningCells], {2}] // Counts
```
Out[•]=
```
⟨|{3, 3, 4, 4, 4, 4, 7, 7} → 3|⟩
```

In[•]:=
```
{pol1, pol2, pol3} = (triangulateToMesh /@
    Map[Lookup[$indToPtsAssoc, #] &, ($cellVertexGrouping /@ adjoiningCells), {2}]);
```

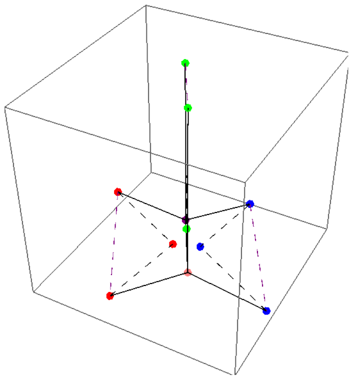
In[•]:=
```
(*orientation Red,Green,Blue → anti-clockwise *)
```

In[•]:=
```
Function[{pol, face, tri},
  Polyhedron@Flatten[pol, 1] // Graphics3D[{#, Orange, Triangle@pol[[face]][[tri]],
      PointSize[0.03], MapAt[Point, Thread[{{Red, Green, Blue}, pol[[face]][[tri]]}],
       {All, 2}]}, ImageSize → Small, Boxed → False] &
 ][
 pol1,
 1,
 7]
```
Out[•]=


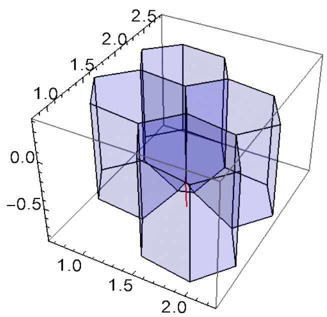
example edge 2

In[•]:=
```
{$edges, $indToPtsAssoc, $ptsToIndAssoc, $cellVertexGrouping, $vertexToCell, $wrappedMat,
   $faceListCoords, seledge, adjoiningCells} = ItoΔ[312, edges, faceListCoords,
   indToPtsAssoc, ptsToIndAssoc, cellVertexGrouping, vertexToCell, wrappedMat];
```

{23, 24, 42, 43}

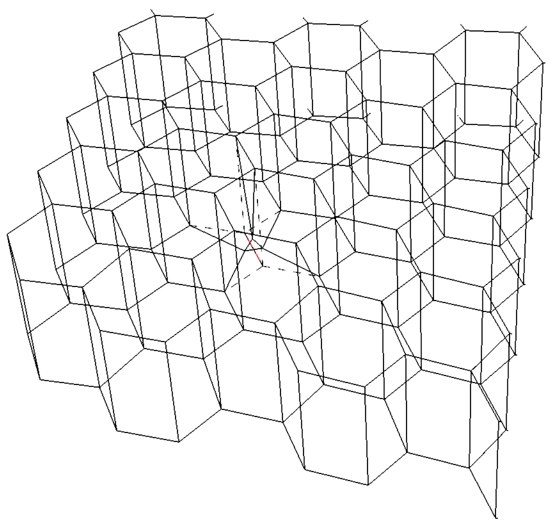


312 →

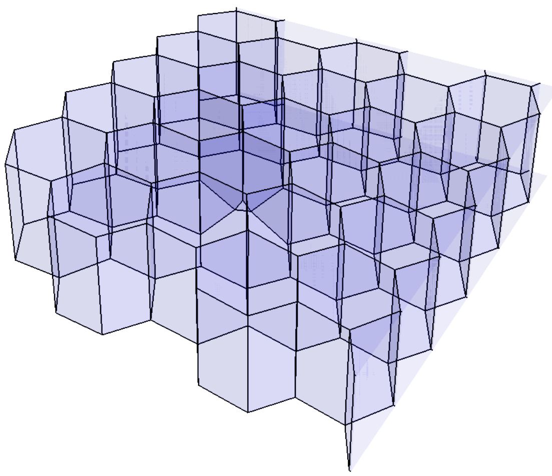


In[ ]:=
```
With[{winsize = 2},
 Graphics3D[{Black, {Dashed, Line@edges}, {Black, Line@$edges},
   Red, Line[#~Append~First@#] &@seledge}, ImageSize → Medium,
  PlotRange → ((MinMax[#] + {-winsize, winsize}) & /@ Transpose@seledge), Boxed → False]
]
```

Out[ ]=

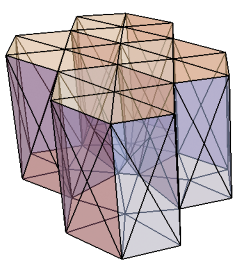


In[∘]:=
```
Block[{winsize = 2},
 Print@Graphics3D[{Opacity[0.075], Blue, Polyhedron /@ Values@$faceListCoords},
   PlotRange → ((MinMax[#] + {-winsize, winsize}) & /@ Transpose@seledge), Boxed → False]
]
```



In[∘]:=
```
Flatten[(triangulateFaces /@
     Map[Lookup[$indToPtsAssoc, #] &, ($cellVertexGrouping /@ adjoiningCells), {2}]), 1] //
  Graphics3D[{Opacity[0.5], #}, ImageSize → Small, Boxed → False] &@*Map[Polyhedron]
```

Out[∘]=



In[∘]:=
```
adjoiningCells
```

Out[∘]=
```
{23, 24, 42, 43}
```

In[●]:=
```
Lookup[$cellVertexGrouping, adjoiningCells]
```

Out[●]=
```
{{{177, 178, 21, 14, 13, 174}, {1761, 176, 17, 20, 25},
   {177, 1762, 178}, {178, 1762, 1761, 25, 21}, {21, 25, 20, 14},
   {14, 20, 17, 13}, {13, 17, 176, 174}, {174, 176, 1761, 1762, 177}},
  {{181, 182, 29, 22, 21, 178}, {183, 1763, 1761, 25, 28, 33, 184},
   {181, 183, 184, 182}, {182, 184, 33, 29}, {29, 33, 28, 22}, {22, 28, 25, 21},
   {21, 25, 1761, 1762, 178}, {178, 1762, 1763, 183, 181}, {1763, 1762, 1761}},
  {{255, 256, 177, 174, 173, 250}, {257, 254, 175, 176, 1761, 1763, 258},
   {255, 257, 258, 256}, {256, 258, 1763, 1762, 177}, {177, 1762, 1761, 176, 174},
   {174, 176, 175, 173}, {173, 175, 254, 250}, {250, 254, 257, 255}, {1761, 1762, 1763}},
  {{259, 260, 181, 178, 177, 256}, {261, 258, 1763, 183, 262},
   {259, 261, 262, 260}, {260, 262, 183, 181}, {181, 183, 1763, 1762, 178},
   {178, 1762, 177}, {177, 1762, 1763, 258, 256}, {256, 258, 261, 259}}}
```

In[●]:=
```
Sort /@ Map[Length, Lookup[$cellVertexGrouping, adjoiningCells], {2}] // Counts
```

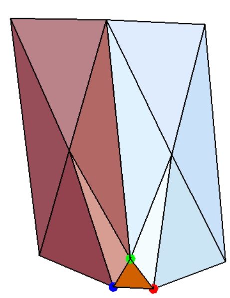
Out[●]=
```
<|{3, 4, 4, 4, 5, 5, 5, 6} → 2, {3, 4, 4, 4, 4, 5, 5, 6, 7} → 2|>
```

In[●]:=
```
{pol1, pol2, pol3, pol4} = (triangulateToMesh /@
    Map[Lookup[$indToPtsAssoc, #] &, ($cellVertexGrouping /@ adjoiningCells), {2}]);
```

In[●]:=
```
(*orientation Red,Green,Blue → anti-clockwise *)
```

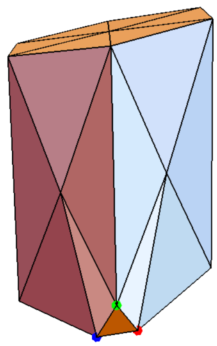
In[●]:=
```
Function[{pol, face, tri},
  Polyhedron@Flatten[pol, 1] // Graphics3D[{#, Orange, Triangle@pol[[face]][[tri]],
      PointSize[0.03], MapAt[Point,
       Thread[{{Red, Green, Blue}, pol[[face]][[tri]]}], {All, 2}]}, ImageSize → Small,
     Boxed → False] &
 ][pol2, 9, 1]
```

Out[●]=



In[∘]:=
```
Function[{pol, face, tri},
  Polyhedron@Flatten[pol, 1] //
   Graphics3D[{#, Orange, Triangle@pol[[face]][[tri]], PointSize[0.03],
      MapAt[Point, Thread[{{Red, Green, Blue}, pol[[face]][[tri]]}], {All, 2}]},
     ImageSize → Small, Boxed → False] &
 ][
 pol3,
 9,
 1]
```

Out[∘]=



In[∘]:=
```
Through[{Max@*Keys, Length}[#]] &@$indToPtsAssoc
```

Out[∘]= {1763, 1761}

In[∘]:=
```
(Map[Lookup[$indToPtsAssoc, #] &, ($cellVertexGrouping /@ adjoiningCells), {2}][[4]] //
   triangulateFaces) ===
 (Map[Lookup[$indToPtsAssoc, #] &, ($cellVertexGrouping /@ adjoiningCells), {2}][[4]] //
   triangulateToMesh)
```

Out[∘]= True

In[∘]:=
```
(* 2 points are removed because an edge *)
Union@Flatten[Lookup[cellVertexGrouping, adjoiningCells]] ~
 Complement ~ Union@Flatten[Lookup[$cellVertexGrouping, adjoiningCells]]
```

Out[∘]= {179, 180}

In[∘]:=
```
(* 3 points are added because a trigonal face *)
```

In[∘]:=
```
Union@Flatten[Lookup[$cellVertexGrouping, adjoiningCells]] ~
 Complement ~ Union@Flatten[Lookup[cellVertexGrouping, adjoiningCells]]
```

Out[∘]= {1761, 1762, 1763}

Δ → I operator

Here instead of picking triangles i am selecting face with 4 edges

In[∘]:=
```
(* pick candidate △ faces to transform *)
pickTriangulatedFaces[faceListCoords_] :=
  Block[{triangleCandidates, triangleCandidatesSel},
   triangleCandidates = Cases[faceListCoords, x_ /; Length[x] == 4, {2}];
   (* yield all △ faces from the mesh & retain
    those that pass Satoru's 2nd condition *)triangleCandidatesSel =
    AllTrue[EuclideanDistance @@@ Partition[#, 2, 1, 1], # ≤ δ &] & /@ triangleCandidates;
   Pick[triangleCandidates, triangleCandidatesSel, True]
  ];
```

In[∘]:=
```
△toIoperation[network_, rules_] := Block[{ruleapply},
   ruleapply =
    ((network /. rules) /. Line[] → Sequence[]) /. {Line → Sequence, {} → Sequence[]};
   Map[DeleteDuplicates@Flatten[#, 1] &, ruleapply, {2}]
  ];
```

In[∘]:=
```
rules△toI[currentTopology_, ptsTri_, ptPartition_] :=
  Block[{attachedEdges, triedges, reconnectRules, rules},
    (*edges connected with face*)
    attachedEdges = DeleteCases[currentTopology,
      Alternatives @@ ({OrderlessPatternSequence @@ #} & /@ Partition[ptsTri, 2, 1, 1])];
    triedges = Complement[currentTopology, attachedEdges];
    (* only edges that form the trigonal face *)
    reconnectRules = Flatten[Cases[attachedEdges,
         q : {y_, p : Alternatives @@ Last[#]} :→ q → {First@#, p}] & /@ ptPartition];
    rules = Dispatch[reconnectRules ~ Join ~ Reverse[reconnectRules, {3}] ~ Join ~ (
        {OrderlessPatternSequence @@ #} → Sequence[] & /@ triedges)];
    rules
   ] /; (! InvalidTrigonalPatternQ[
      Graph@Replace[currentTopology, List → UndirectedEdge, {2}, Heads → True]]);
```

artificial factor of 0.1 multiplied below

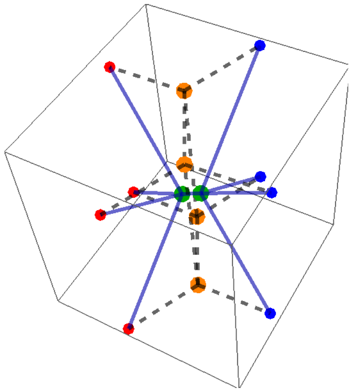
In[ ]:=
```
ΔtoIpreprocess[ptsTri_, currentTopology_] :=
  Block[{sortptsTri, uTH, r1, r2, PtsAcrossFaces,
    ptsAttached, newPts, ptPartition, newLocalTopology, vec},
   With[{r0H = Mean@ptsTri},
    sortptsTri = SortBy[ptsTri, ArcTan[# - r0H] &];
    (* arrange the points in an clockwise || anti-clockwise manner *)uTH = Function[
       Cross[#2 - #1, #3 - #1] / (Norm[#2 - #1] Norm[#3 - #1])][Sequence @@ sortptsTri];
    r1 = SetPrecision[r0H + 0.5 δ * 0.1 uTH, 10];
    r2 = SetPrecision[r0H - 0.5 δ * 0.1 uTH, 10];
    vec = Normalize[r1 - r2];
    ptsAttached = DeleteCases[currentTopology~Flatten~1, Alternatives @@ ptsTri];
    (* are points above or below the Δ *)
    PtsAcrossFaces = GroupBy[ptsAttached, Sign[vec.(r0H - #)] &];
    (* compute the 2 new points from the 3 old points *)
    newPts = <|Sign[vec.(r0H - #)] → # & /@ {r1, r2}|>;
    ptPartition = Values@Merge[{newPts, PtsAcrossFaces}, Identity];
    (* which points belong with r1 and which points with r2 *)newLocalTopology =
     Flatten[Map[x ↦ First[x] ⟷ # & /@ Last[x], ptPartition], 1]~Join~{r1 ⟷ r2};
    {ptPartition, newLocalTopology, r1, r2}]
  ];
```

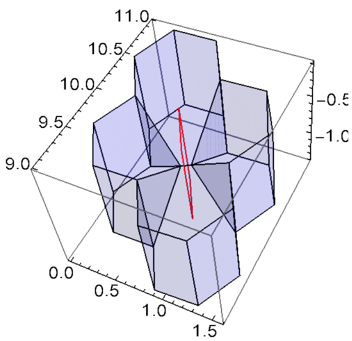
In[ ]:=
```
ΔtoI[ind_, edges_, faceListCoords_, indToPtsAssoc_,
   ptsToIndAssoc_, cellVertexGrouping_, vertexToCell_, wrappedMat_] :=
  Block[{selectTriangle, candidate, currentTopology, ptsAttached, graphCurrentTopology,
    ptsTri, PtPartition, newLocalTopology, adjoiningCells, prevNetwork,
    updatedLocalNetwork, rules, celltopologicalChanges, r1, r2, maxVnum, newAdditions,
    localtopology, wrappedcells, transvec, cellCoords, ls, positions, celltransvecAssoc,
    cellTopologicalChangesBeforeShift, cellspartof, $faceListCoords = faceListCoords,
    $ptsToIndAssoc = ptsToIndAssoc, $indToPtsAssoc = indToPtsAssoc,
    $vertexToCell = vertexToCell, $cellVertexGrouping = cellVertexGrouping,
    $wrappedMat = wrappedMat, vpt, $edges = edges, selectTriangles},
   selectTriangles = pickTriangulatedFaces@*Values@$faceListCoords;
   Scan[
    (candidate = #;
      If[And @@ (KeyMemberQ[$ptsToIndAssoc, #] & /@ candidate),
       (* get local network topology from the Δ face: basically which coordinates
          the face is linked to *){localtopology, wrappedcells, transvec} =
        getLocalTopology[$ptsToIndAssoc, $indToPtsAssoc, $vertexToCell,
          $cellVertexGrouping, $wrappedMat, $faceListCoords][candidate];
       {adjoiningCells, cellCoords} = {Keys@#, Values@#} &@localtopology;
       (* adjoining cells and their vertices *)
       (*ok till here*)
       prevNetwork = Map[Partition[#, 2, 1, 1] &, cellCoords, {2}];
       (* this yields all the unique edges
        in the current topology and extract vertex pairs, such that
        {candidate_vertex, vertex_attached with candidate} *)
       currentTopology = Cases[DeleteDuplicatesBy[Flatten[prevNetwork, 2], Sort],
         {OrderlessPatternSequence[x_, p : Alternatives @@ candidate]} :→ {p, x}];
       (*creating a graph from the current topology*)
```

```
graphCurrentTopology =
 Graph@Replace[currentTopology, List → UndirectedEdge, {2}, Heads → True];
If[! InvalidTrigonalPatternQ[graphCurrentTopology],
  (* transform the network topology by applying [H]→[I] operation *)
  ptsTri = candidate; (* vertices of the faces *)
  {PtPartition, newLocalTopology, r1, r2} =
   ΔtoIpreprocess[ptsTri, currentTopology];
  Print@Graphics3D[{{Dashed, Thick, Opacity[0.6], Black, Line@currentTopology},
     {Thick, Opacity[0.6], Darker@Blue,
      Line[newLocalTopology /. UndirectedEdge → List]},
     {Red, PointSize[0.035], Point@PtPartition[[2, -1]]},
     {Blue, PointSize[0.035], Point@PtPartition[[1, -1]]},
     {Orange, PointSize[0.05], Point@candidate}, {Darker@Green,
      PointSize[0.05], Point@{r1, r2}}}, ImageSize → Small];
  rules = rulesΔtoI[currentTopology, ptsTri, PtPartition];
  Switch[
   rules, _rulesΔtoI, None,
   _,
   (updatedLocalNetwork = prevNetwork~ΔtoIoperation~rules;
    celltopologicalChanges = bindCellsToNewTopology[adjoiningCells,
       updatedLocalNetwork] /. _bindCellsToNewTopology → {};
    If[celltopologicalChanges ≠ {},
     Print[Graphics3D[
       {{Opacity[0.1], Blue, Polyhedron /@ Values[celltopologicalChanges]},
        {Red, Line[candidate~Append~First[candidate]]}}, Axes → True]];

     maxVnum = Max@*Keys@$indToPtsAssoc;

     If[wrappedcells ≠ {},
      (* if there are wrapped
       cells send them back to their respective positions *)
      (* wrapped cells with their respective vectors for translation *)
      celltransvecAssoc = AssociationThread[wrappedcells, transvec];
      cellTopologicalChangesBeforeShift = celltopologicalChanges;
      (* here we send the cells
        back to their original positions → unwrapped state *)
      celltopologicalChanges = (x ↦ With[{p = First[x]},
           If[MemberQ[wrappedcells, p], p →
             Map[SetPrecision[# - celltransvecAssoc[p], 10] &, Last[x], {2}], x]
          ]) /@ celltopologicalChanges;

      ls = {};
      Scan[
       vpt ↦
        (positions = Position[cellTopologicalChangesBeforeShift, vpt];
         positions = DeleteDuplicates[{First[#]} & /@ positions];
         cellspartof = Extract[adjoiningCells, positions];
         Fold[
          Which[MemberQ[wrappedcells, #2],
            AppendTo[ls, SetPrecision[vpt - celltransvecAssoc[#2], 10] ],
```

```
                   True, If[! MemberQ[ls, vpt], AppendTo[ls, vpt]]] &,
                 ls, cellspartof]), {r1, r2}];

             newAdditions = Thread[(Range[Length[ls]] + maxVnum) → ls],
             newAdditions = Thread[(Range[2] + maxVnum) → {r1, r2}]
             (* labels for new vertices *)
            ];
            updatedLocalNetwork =
             Map[Partition[#, 2, 1, 1] &, Values[celltopologicalChanges], {2}];

            {$indToPtsAssoc, $ptsToIndAssoc, $cellVertexGrouping, $vertexToCell} =
             modifier[candidate, adjoiningCells, $indToPtsAssoc,
              $ptsToIndAssoc, $cellVertexGrouping, $vertexToCell,
              celltopologicalChanges, updatedLocalNetwork, newAdditions];

            $faceListCoords =
             Map[Lookup[$indToPtsAssoc, #] &, $cellVertexGrouping, {2}];

            $edges = Flatten[Map[Partition[#, 2, 1, 1] &, Map[
                 Lookup[$indToPtsAssoc, #] &, Values[$cellVertexGrouping],
                 {2}], {2}], 2] // DeleteDuplicatesBy[Sort];

            $wrappedMat = AssociationThread[Keys[$cellVertexGrouping] →
               Map[Lookup[$indToPtsAssoc, #] /. periodicRules &,
                Lookup[$cellVertexGrouping, Keys[$cellVertexGrouping]], {2}]];
           ])
         ]
        ]
      ]) &, {selectTriangles[[ind]]}];
   {$edges, $indToPtsAssoc, $ptsToIndAssoc, $cellVertexGrouping, $vertexToCell,
    $wrappedMat, $faceListCoords, selectTriangles[[ind]], adjoiningCells}
  ];
```

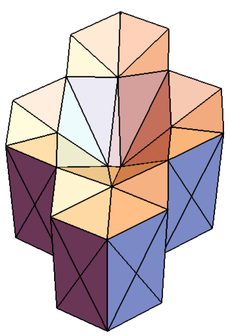
In[ ]:=
```
{$edges, $indToPtsAssoc, $ptsToIndAssoc, $cellVertexGrouping, $vertexToCell, $wrappedMat,
   $faceListCoords, selface, adjoiningCells} = ΔtoI[202, edges, faceListCoords,
   indToPtsAssoc, ptsToIndAssoc, cellVertexGrouping, vertexToCell, wrappedMat];
```



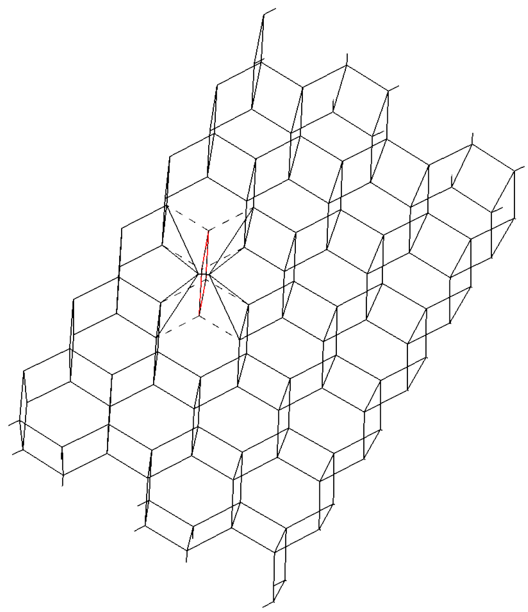


In[ ]:=
```
(triangulateFaces /@
    Map[Lookup[$indToPtsAssoc, #] &, ($cellVertexGrouping /@ adjoiningCells), {2}]) //
  Map[Polyhedron[Flatten[#, 1]] &, #] & // Graphics3D[#, ImageSize → Small, Boxed → False] &
```

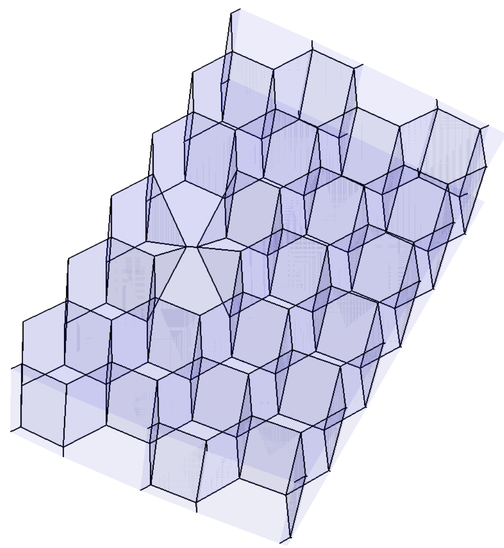
Out[ ]=



In[ ]:=
```
Block[{winsize = 2}, Print@Graphics3D[{Black, {Dashed, Line@edges},
    {Black, Line@$edges}, Red, Line[#~Append~First@#] &@selface},
   PlotRange → ((MinMax[#] + {-winsize, winsize}) & /@ Transpose@selface), Boxed → False]
 ]
```



In[•]:=
```
Block[{winsize = 2},
 Print@Graphics3D[{Opacity[0.075], Blue, Polyhedron /@ Values@$faceListCoords},
   PlotRange → ((MinMax[#] + {-winsize, winsize}) & /@ Transpose@selface), Boxed → False]
 ]
```



In[•]:=
```
Through[{Max@*Keys, Length}[#]] &@$indToPtsAssoc
```

Out[•]=
```
{1762, 1758}
```

In[•]:=
```
adjoiningCells
```

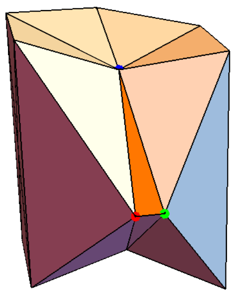
Out[•]=
```
{13, 14, 33, 34}
```

In[•]:=
```
{pol1, pol2, pol3, pol4} = (triangulateToMesh /@
    Map[Lookup[$indToPtsAssoc, #] &, ($cellVertexGrouping /@ adjoiningCells), {2}]);
```

In[•]:=
```
(*orientation Red,Green,Blue → anti-clockwise *)
```

In[∘]:=
```
Function[{pol, face, tri},
  Polyhedron@Flatten[pol, 1] // Graphics3D[{#, Orange, Triangle@pol[[face]][[tri]],
      PointSize[0.03], MapAt[Point,
       Thread[{{Red, Green, Blue}, pol[[face]][[tri]]}], {All, 2}]}, ImageSize → Small,
     Boxed → False] &
 ][pol3, 1, 3]
```

Out[∘]=



In[∘]:=
```
Function[{pol, face, tri},
  Polyhedron@Flatten[pol, 1] // Graphics3D[{#, Orange, Triangle@pol[[face]][[tri]],
      PointSize[0.03], MapAt[Point,
       Thread[{{Red, Green, Blue}, pol[[face]][[tri]]}], {All, 2}]}, ImageSize → Small,
     Boxed → False] &
 ][pol4, 6, 1]
```

Out[∘]=