module - computing Volume ∇

In[2]:=
```
DumpGet["D:\\LocalData\\hashmial\\Research\\Vertex
    Model 3D\\sanity checks for functions\\meshGen_noise.mx"];
```

In[3]:=
```
yLim[[1]] = 0.;
edges = SetPrecision[edges, 10];
faceListCoords = SetPrecision[faceListCoords, 10];
(*convert faceListCoords into an association*)
indToPtsAssoc = SetPrecision[indToPtsAssoc, 10];
ptsToIndAssoc = KeyMap[SetPrecision[#, 10] &, ptsToIndAssoc];
xLim = SetPrecision[xLim, 10];
yLim = SetPrecision[yLim, 10];
faceListCoords = Map[Lookup[indToPtsAssoc, #] &, cellVertexGrouping, {2}];
```

In[11]:=
```
Clear@periodicRules;
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 ≤ xlim1, y_ /; y ≤ ylim1, z_} :→
       SetPrecision[{x + xlim2, y + ylim2, z}, 10],
      {x_ /; x ≤ xlim1, y_ /; ylim1 < y < ylim2, z_} :→
       SetPrecision[{x + xlim2, y, z}, 10],
      {x_ /; x ≤ xlim1, y_ /; y ≥ ylim2, z_} :→
       SetPrecision[{x + xlim2, y - ylim2, z}, 10],
      {x_ /; xlim1 < 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 ≤ xlim1, y_ /; y ≤ ylim1, _} :→ SetPrecision[{xlim2, ylim2, 0}, 10],
      {x_ /; x ≤ xlim1, y_ /; ylim1 < y < ylim2, _} :→ SetPrecision[{xlim2, 0, 0}, 10],
      {x_ /; x ≤ xlim1, y_ /; y ≥ ylim2, _} :→ SetPrecision[{xlim2, -ylim2, 0}, 10],
      {x_ /; xlim1 < 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[13]:=
```
origcellOrient = <|MapIndexed[First[#2] → #1 &, faceListCoords]|>;
boundaryCells =
  With[{xlim1 = xLim[[1]], xlim2 = xLim[[2]], ylim1 = yLim[[1]], ylim2 = yLim[[2]]},
   Union[First /@ Position[origcellOrient,
       {x_ /; x ≥ xlim2, __} | {x_ /; x ≤ xlim1, __} |
        {_, y_ /; y ≥ ylim2, _} | {_, y_ /; y ≤ ylim1, _}] /. Key[x_] :→ x]
  ];
wrappedMat = AssociationThread[
   Keys[cellVertexGrouping] → Map[Lookup[indToPtsAssoc, #] /. periodicRules &,
     Lookup[cellVertexGrouping, Keys[cellVertexGrouping]], {2}]];
```

In[16]:=
```
meanTri = Compile[{{faces, _Real, 2}},
  Mean@faces,
  CompilationTarget → "C", RuntimeAttributes → {Listable},
  Parallelization → True
 ]
```

Out[16]= CompiledFunction[Argument count: 1 Argument types: {{_Real, 2}}]

In[17]:=
```
Clear[triNormal];
triNormal = Compile[{{ls, _Real, 2}},
  Block[{res},
   res = Partition[ls, 2, 1];
   Cross[res[[1, 1]] - res[[1, 2]], res[[2, 1]] - res[[2, 2]]]
  ], CompilationTarget → "C", RuntimeAttributes → {Listable}
 ]
```

Out[18]= CompiledFunction[Argument count: 1 Argument types: {{_Real, 2}}]

In[19]:=
```
Clear[meanFaces, triangulateToMesh];
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}
 ]

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[20]=
```
CompiledFunction[ Argument count: 1
                  Argument types: {{_Real, 2}} ]
```

In[22]:=
```
Clear@cellCentroids;
cellCentroids[polyhedCentAssoc_, keystopo_, shiftvec_] :=
  Block[{assoc = <||>, regcent, counter},
   AssociationThread[Keys@keystopo →
     KeyValueMap[
      Function[{key, cellassoc},
       If[KeyFreeQ[shiftvec, key],
        Lookup[polyhedCentAssoc, cellassoc],
        If[KeyFreeQ[shiftvec[key], #],
           regcent = polyhedCentAssoc[#],
           regcent = polyhedCentAssoc[#] + shiftvec[key][#];
           regcent
          ] & /@ cellassoc
        ]
      ], keystopo]
    ]
  ];
```

In[24]:=
```
𝒟 = Rectangle[{First@xLim, First@yLim}, {Last@xLim, Last@yLim}];
```

In[28]:=
```
ClearAll@getLocalTopology;
getLocalTopology[ptsToIndAssoc_, indToPtsAssoc_, vertexToCell_,
    cellVertexGrouping_, wrappedMat_, faceListCoords_][vertices_] :=
  Block[{localtopology = <||>, wrappedcellList = {}, vertcellconns,
    localcellunion, v, wrappedcellpos, vertcs = vertices, rl1, rl2,
    transVector, wrappedcellCoords, wrappedcells, vertOutofBounds,
    shiftedPt, transvecList = {}, $faceListCoords = Values@faceListCoords,
```

```
    vertexQ, boundsCheck, rules, extractcellkeys, vertind,
    cellsconnected, wrappedcellsrem},
   vertexQ = MatchQ[vertices, {__?NumberQ}];
   If[vertexQ,
    (vertcellconns =
      AssociationThread[{#} , {vertexToCell[ptsToIndAssoc[#]]}] &@vertices;
     vertcs = {vertices};
     localcellunion = Flatten[Values@vertcellconns]),
    (vertcellconns = AssociationThread[# ,
         Lookup[vertexToCell, Lookup[ptsToIndAssoc, #]]] &@vertices;
     localcellunion = Union@Flatten[Values@vertcellconns])
   ];

   If[localcellunion ≠ {},
    AppendTo[localtopology,
     Thread[localcellunion →
       Map[Lookup[indToPtsAssoc, #] &, cellVertexGrouping /@ localcellunion, {2}]]
    ]
   ];
   (* condition to be an internal edge: both vertices should have 3 neighbours *)
   (* if a vertex has 3 cells in its local neighbourhood then the entire
     network topology about the vertex is known → no wrapping required *)
   (* else we need to wrap around the vertex because other cells
     are connected to it → periodic boundary conditions *)
   With[{vert = #},
      vertind = ptsToIndAssoc[vert];
      cellsconnected = vertexToCell[vertind];
      If[Length[cellsconnected] ≠ 3,
       If[(𝒟~RegionMember~Most[vert]),
         (*Print["vertex inside bounds"];*)
         v = vert;
         With[{x = v[[1]], y = v[[2]]}, boundsCheck =
            (x == xLim[[1]] || x == xLim[[2]] || y == yLim[[1]] || y == yLim[[2]])];

         extractcellkeys = If[boundsCheck,
           {rl1, rl2} = {v, v /. periodicRules};
           rules = Block[{x$},
             With[{r = rl1, s = rl2},
              DeleteDuplicates[
               HoldPattern[SameQ[x$, r]] || HoldPattern[SameQ[x$, s]]]
             ]
            ];
           Position @@ With[{rule = rules},
             Hold[wrappedMat, x_ /; ReleaseHold@rule, {3}]
            ],
           Position[wrappedMat, x_ /; SameQ[x, v], {3}]
          ];
         (* find cell indices that are attached to the vertex in wrappedMat *)
         wrappedcellpos = DeleteDuplicatesBy[
           Cases[extractcellkeys,
            {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[$faceListCoords, Replace[#,
          {p_, q__} :→ {Key[p], q}, {1}]]) & /@ wrappedcellpos, 10],
  (* call to function is enquiring an edge and not a vertex*)
  transVector =
   SetPrecision[(v - Extract[$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];
],
(* the else clause: vertex is out of bounds *)
(*Print["vertex out of bounds"];*)
vertOutofBounds = vert;
(* translate the vertex back into mesh *)
transVector = vertOutofBounds /. transformRules;
shiftedPt = SetPrecision[vertOutofBounds + transVector, 10];
(* ------------- CORE B ------------- *)
(* find which cells the
 shifted vertex is a part of in the wrapped matrix *)
wrappedcells = Complement[
  Union@Cases[Position[wrappedMat, x_ /;
       SameQ[x, shiftedPt] || SameQ[x, vertOutofBounds], {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) | vertOutofBounds},
  Block[{pos, vertref, transvec},
    Do[
     With[{cellcoords = wrappedcellCoords[cell]},
      pos = FirstPosition[cellcoords /. periodicRules, opt];
      If[Head[pos] === Missing,
       pos = FirstPosition[
          Chop[cellcoords /. periodicRules, 10^-6], Chop[opt, 10^-6]];
      ];
      vertref = Extract[cellcoords, pos];
      transvec = SetPrecision[vertOutofBounds - vertref, 10];
      AppendTo[transvecList, transvec];
      AppendTo[localtopology,
       cell → Map[SetPrecision[# + transvec, 10] &, cellcoords, {2}]];
     ], {cell, wrappedcells}]
```

```
              ];
            ];
          ];
          (* to detect wrapped cells not detected by CORE B*)
          (* ------------- CORE C ------------- *)
          Block[{pos, celllocs, ls, transvec, assoc, tvecLs = {}, ckey},
           ls = Union@Flatten@Join[cellsconnected, wrappedcells];
           If[Length[ls] ≠ 3,
            pos = Position[faceListCoords, x_ /; SameQ[x, shiftedPt], {3}];
            celllocs = DeleteDuplicatesBy[Cases[pos, Except[{Key[Alternatives @@ ls],
                  __}]], First] /. {Key[x_], z__} ↦ {Key[x], {z}};
            If[celllocs ≠ {},
             celllocs = Transpose@celllocs;
             assoc = <|
               MapThread[
                (transvec = SetPrecision[vertOutofBounds -
                     Extract[faceListCoords[Sequence @@ #1], #2], 10];
                  ckey = Identity @@ #1;
                  AppendTo[tvecLs, transvec];
                  ckey → Map[SetPrecision[Lookup[indToPtsAssoc, #] + transvec,
                      10] &, cellVertexGrouping[Sequence @@ #1], {2}]
                 ) &, celllocs]
               |>;
             AppendTo[localtopology, assoc];
             AppendTo[wrappedcellList, Keys@assoc];
             AppendTo[transvecList, tvecLs];
            ];
           ];
          ];
        ];
      ];
    ] & /@ vertcs;

  transvecList = Which[
    MatchQ[transvecList, {{{__?NumberQ}}}], First[transvecList],
    MatchQ[transvecList, {{__?NumberQ} ..}], transvecList,
    True, transvecList //. {x___, {p : {__?NumberQ} ..}, y___} ↦ {x, p, y}
   ];
  {localtopology, Flatten@wrappedcellList, transvecList}
 ];
```

Launch Kernels

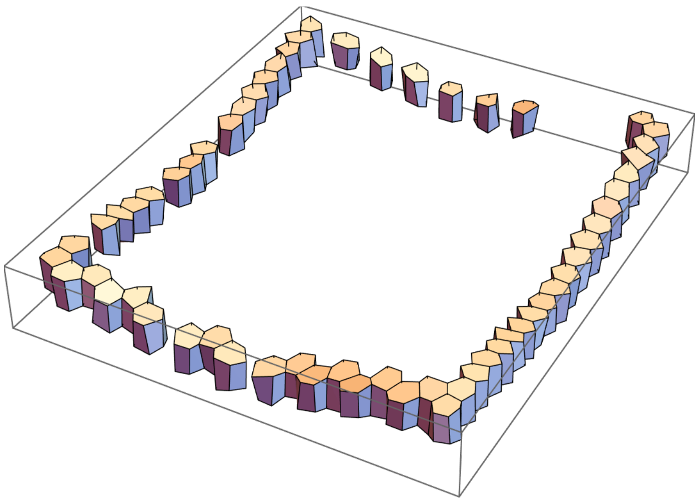
In[30]:=
```
LaunchKernels[]
```

Out[30]=
```
{KernelObject[ Name: local KernelID: 1 ], KernelObject[ Name: local KernelID: 2 ],
 KernelObject[ Name: local KernelID: 3 ], KernelObject[ Name: local KernelID: 4 ],
 KernelObject[ Name: local KernelID: 5 ], KernelObject[ Name: local KernelID: 6 ]}
```

prerequisite run

In[31]:=
```
Graphics3D[Polygon /@ (faceListCoords /@ boundaryCells)]
```

Out[31]=



In[•]:=
```
(*missing boundary cells need to be found *)
```

In[32]:=
```
bcells = KeyTake[faceListCoords, boundaryCells];
```

In[33]:=
```
Length@boundaryCells
```

Out[33]=
```
60
```

In[34]:=
```
keyLs = Union@(Flatten@Lookup[vertexToCell,
        Lookup[ptsToIndAssoc,
         With[{xlim1 = xLim[[1]],
            ylim1 = yLim[[1]], ylim2 = yLim[[2]], xlim2 = xLim[[2]]},
           DeleteDuplicates@Cases[bcells,
             {x_ /; x ≥ xlim2, __} | {x_ /; x ≤ xlim1, __} |
              {_, y_ /; y ≥ ylim2, _} | {_, y_ /; y ≤ ylim1, _}, {3}]
          ] /. periodicRules
        ]
       ]~Join~boundaryCells);
```

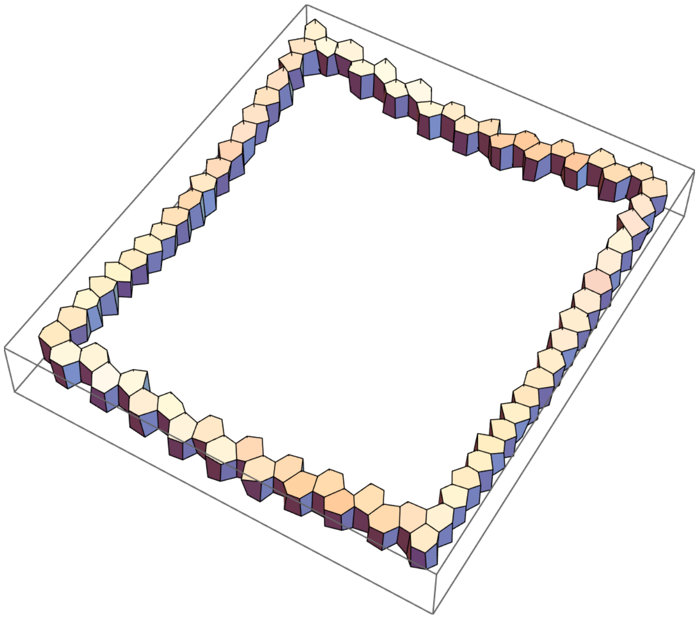
In[35]:=
```
Length[keyLs] - Length[boundaryCells]
```

Out[35]=
```
16
```

In[36]:=
```
border = faceListCoords /@ keyLs;
```

In[37]:=
```
Graphics3D[{Polygon /@ border}, ImageSize → Medium]
```

Out[37]=



In[38]:=
```
wrappedMatC = KeyTake[wrappedMat, keyLs];
```

In[39]:=
```
vertKeys = Keys@indToPtsAssoc;
```

In[40]:=
```
(
  topo = <|# → (getLocalTopology[ptsToIndAssoc, indToPtsAssoc,
           vertexToCell, cellVertexGrouping, wrappedMatC, faceListCoords][
          indToPtsAssoc[#]] // First) & /@ vertKeys
     |>;
 ) // AbsoluteTiming
```

Out[40]=
```
{1.16552, Null}
```

Growing/Static cells

randomly select cells in the mesh to grow

In[41]:=
```
cellIds = Keys@cellVertexGrouping;
```

In[42]:=
```
fractionPopulation = 0.07;
growingcellIndices = RandomSample[cellIds, Round[fractionPopulation Length@cellIds]]
```

Out[43]=
```
{113, 323, 303, 204, 66, 126, 282, 297, 310, 184, 78, 361, 192,
 212, 328, 343, 263, 333, 114, 250, 69, 47, 216, 16, 103, 259, 129, 8}
```

In[44]:=
```
nongrowingCellIndices = cellIds ~ Complement ~ growingcellIndices;
```

finding triangles connected to a vertex

In[45]:=
```
pointind = 1167;
```

In[114]:=
```
point = indToPtsAssoc@pointind;
```

In[46]:=
```
triangulatedCells = triangulateToMesh /@ faceListCoords;
polyhedraAssoc = Polyhedron@Flatten[#, 1] & /@ triangulatedCells;
```

In[48]:=
```
(trimesh = Map[triangulateToMesh, topo, {2}]); // AbsoluteTiming
```

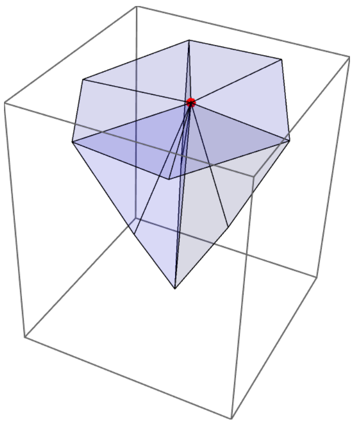
Out[48]=
```
{2.06562, Null}
```

In[49]:=
```
examplevertToTri =
   GroupBy[Flatten[Values@trimesh[#], 2], MemberQ[indToPtsAssoc[#]]][True] &[
    1]; // AbsoluteTiming
```

Out[49]=
```
{0.0003805, Null}
```

In[50]:=
```
(examplevertToTri =
    GroupBy[Flatten[Values@trimesh[#], 2], MemberQ[indToPtsAssoc[#]]][True];
   Graphics3D[{{Opacity[0.15], Blue, Triangle /@ examplevertToTri},
     Red, PointSize[0.03], Point@indToPtsAssoc[#]},
    ImageSize → Small]
  ) &[RandomInteger[Max@Keys@indToPtsAssoc]]
```

Out[50]=

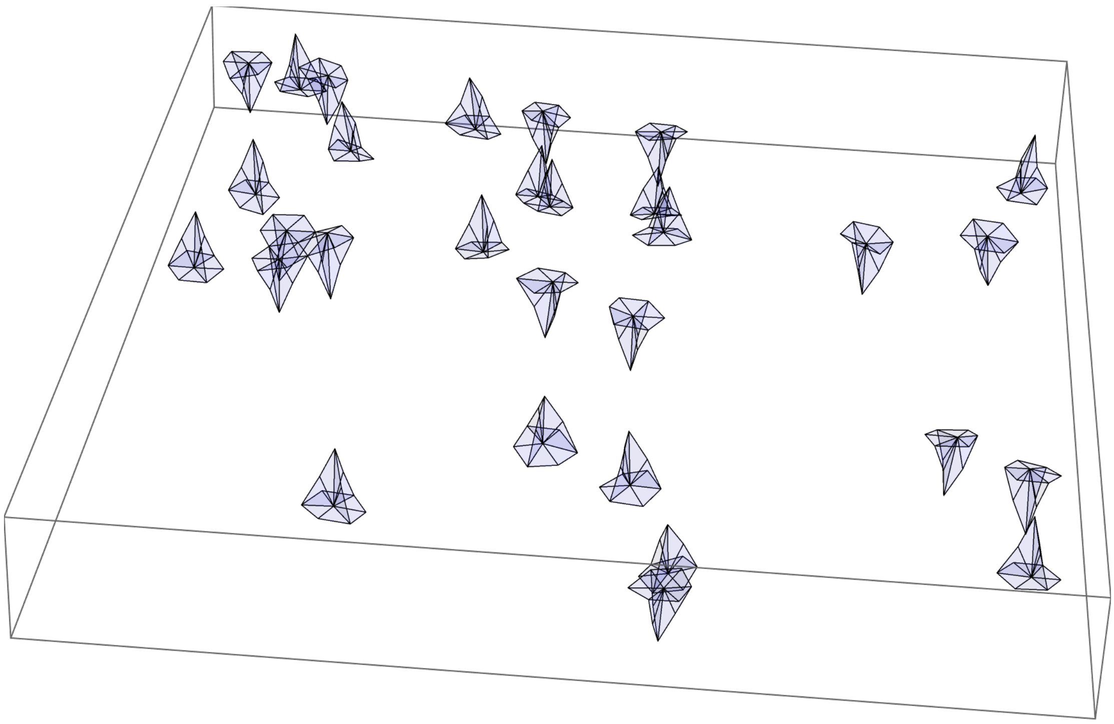


In[51]:=
```
(associatedtri = With[{ItoPA = indToPtsAssoc, tmesh = trimesh},
     AssociationThread[vertKeys, Function[vert, <|GroupBy[
             Flatten[#, 1], MemberQ[ItoPA[vert]]
            ][True] & /@ tmesh[vert]|>] /@ vertKeys]
    ];
 ) // AbsoluteTiming
```

Out[51]=
```
{0.623909, Null}
```

In[52]:=
```
SeedRandom[3];
Graphics3D[{Opacity[0.1], Blue, Triangle /@
   Flatten[Values@Values@RandomSample[associatedtri, 30], 2]}, ImageSize → Large]
```

Out[53]=



In[54]:=
```
(centTri = <|# → meanTri[Values[associatedtri@#]] & /@ Keys@indToPtsAssoc|>;) //
 AbsoluteTiming
```

Out[54]= {0.367853, Null}

In[55]:=
```
centTri = SetPrecision[#, 10] & /@ centTri;
```

In[56]:=
```
(normals = Map[SetPrecision[#, 10] &, triNormal@Values@# & /@ associatedtri]); //
 AbsoluteTiming
```

Out[56]= {0.478333, Null}

In[57]:=
```
(normNormals = Map[Normalize, normals, {3}];) // AbsoluteTiming
```

Out[57]= {0.116964, Null}

In[58]:=
```
(triangulatedmesh = triangulateToMesh /@ faceListCoords); // AbsoluteTiming
(polyhedra = Polyhedron@* (Flatten[#, 1] &) /@ triangulatedmesh;) // AbsoluteTiming
```

Out[58]= {0.170019, Null}

Out[59]= {0.0011387, Null}

In[60]:=
```
(polyhedcent = RegionCentroid /@ polyhedra); // AbsoluteTiming
```

Out[60]= {4.68856, Null}

In[61]:=
```
(
  topoF = <|# → (getLocalTopology[
           ptsToIndAssoc, indToPtsAssoc, vertexToCell, cellVertexGrouping,
           wrappedMatC, faceListCoords][indToPtsAssoc[#]]) & /@ vertKeys
     |>;
 ) // AbsoluteTiming
```
Out[61]= `{1.25695, Null}`

In[62]:=
```
(keyslocaltopoF = Keys@*First /@ topoF); // AbsoluteTiming
```
Out[62]= `{0.0040015, Null}`

In[63]:=
```
(shiftVecAssoc = Association /@ Map[Apply[Rule],
      Thread /@ Select[(#[[2 ;; 3]]) & /@ topoF, # ≠ {{}, {}} &], {2}]); // AbsoluteTiming
```
Out[63]= `{0.0049915, Null}`

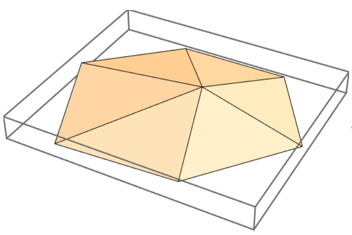
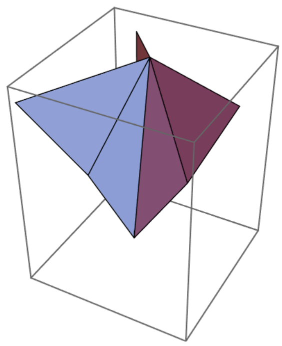
In[64]:=
```
(cellcentroids = cellCentroids[polyhedcent, keyslocaltopoF, shiftVecAssoc]);
```

In[65]:=
```
(signednormals = AssociationThread[Keys@indToPtsAssoc,
     Map[
      MapThread[
        #2 Sign@MapThread[Function[{x, y}, (y - #1).x], {#2, #3}] &,
        {cellcentroids[#], normNormals[#], centTri[#]}] &, Keys@indToPtsAssoc]
     ]
  ); // AbsoluteTiming
```
Out[65]= `{0.209725, Null}`

make sets of open/closed triangles

In[66]:=
```
opencloseTri = Flatten[Values@#, 1] & /@ associatedtri;
```

In[67]:=
```
Graphics3D /@ Map[Triangle,
  GroupBy[GatherBy[opencloseTri[1], Intersection], Length, Flatten[#, 1] &], {2}]
```

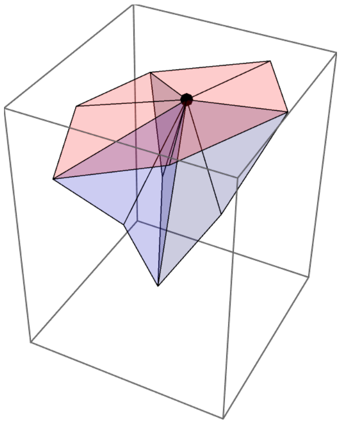
Out[67]= ⟨| 1 → , 2 →  |⟩

In[68]:=
```
triDistAssoc = Block[{trianglemembers},
   Map[
    (trianglemembers = #;
      GroupBy[GatherBy[trianglemembers, Intersection], Length, Flatten[#, 1] &]) &,
    opencloseTri]
  ];
```

In[69]:=
```
{opentri, closedtri} = {triDistAssoc[pointind][1], triDistAssoc[pointind][2]};
```

In[144]:=
```
Graphics3D[
 {{Opacity[0.2], Red, Map[Triangle][opentri], Blue, Map[Triangle][closedtri]},
  {Black, PointSize[0.04], Point@indToPtsAssoc[pointind]}}, ImageSize → Small]
```

Out[ ]=



associate normals with triangles

In[145]:=
```
vertTriNormalpairings = <|# → <|Thread[Flatten[Values@associatedtri[#], 1] →
        Flatten[signednormals@#, 1]]|> & /@ vertKeys|>;
```

To associate the open/closed triangles with their respective normals we simply need to perform a lookup in the association for (vertex1,vertex2,vertex3) - a triangle face - and its normal.

In[146]:=
```
normalsO = Lookup[vertTriNormalpairings[pointind], opentri];
```

In[147]:=
```
normalsC = Lookup[vertTriNormalpairings[pointind], closedtri];
```

In[148]:=
```
normalLs = normalsO~Join~normalsC;
```

volume gradient F[x]

gradient of volume is computed as: $1/3\ \Sigma\ A_{\Delta}\vec{N}$

```
volumeGradient[point_, opentri_, closedtri_, normalLs_, cellids_,
   localtopology_, polyhedraAssoc_, growingCellIds_] :=
  Reap@Block[{topo, topology, normalassoc, gradV, gradVCont,
      triangulatedCellsSel, polyhedraSel, volume, growingIndkeys},
    triangulatedCellsSel = triangulateToMesh /@ localtopology;
    polyhedraSel = Lookup[polyhedraAssoc, cellids];
    topo = (Cases[#, x_ /; MemberQ[x, point]] &) @* (Flatten[#, 1] &) /@
      triangulatedCellsSel;
    Sow[topo];
    normalassoc = AssociationThread[opentri ~ Join ~ closedtri, normalLs];
    gradV = Table[topology = topo[cell];
      (1.0 / 3.0) Total[Map[Area[Triangle[#]] * normalassoc[#] &, topology]],
      {cell, cellids}];
    volume = AssociationThread[cellids → ConstantArray[V_o, Length@cellids]];
    growingIndkeys =
     Replace[Intersection[cellids, growingCellIds], k_Integer :> {Key[k]}, {1}];
    volume = If[growingIndkeys ≠ {},
      Values@MapAt[(1 + V_growth time) # &, volume, growingIndkeys],
      Values@volume
     ];
    gradVCont = k_cv Total[((Volume[polyhedraSel] - volume) / (volume^2)) gradV]
   ];
```

In[150]:=
```
{cellids, localtopology} = Through[{Keys, Identity}[#]] &[First@topoF[pointind]];
```

In[151]:=
```
{res, pyramids} = volumeGradient[point, opentri, closedtri,
   normalLs, cellids, localtopology, polyhedraAssoc, growingcellIndices];
```

In[84]:=
```
Column@Through[{Length, Identity}[#]] &[res]
```

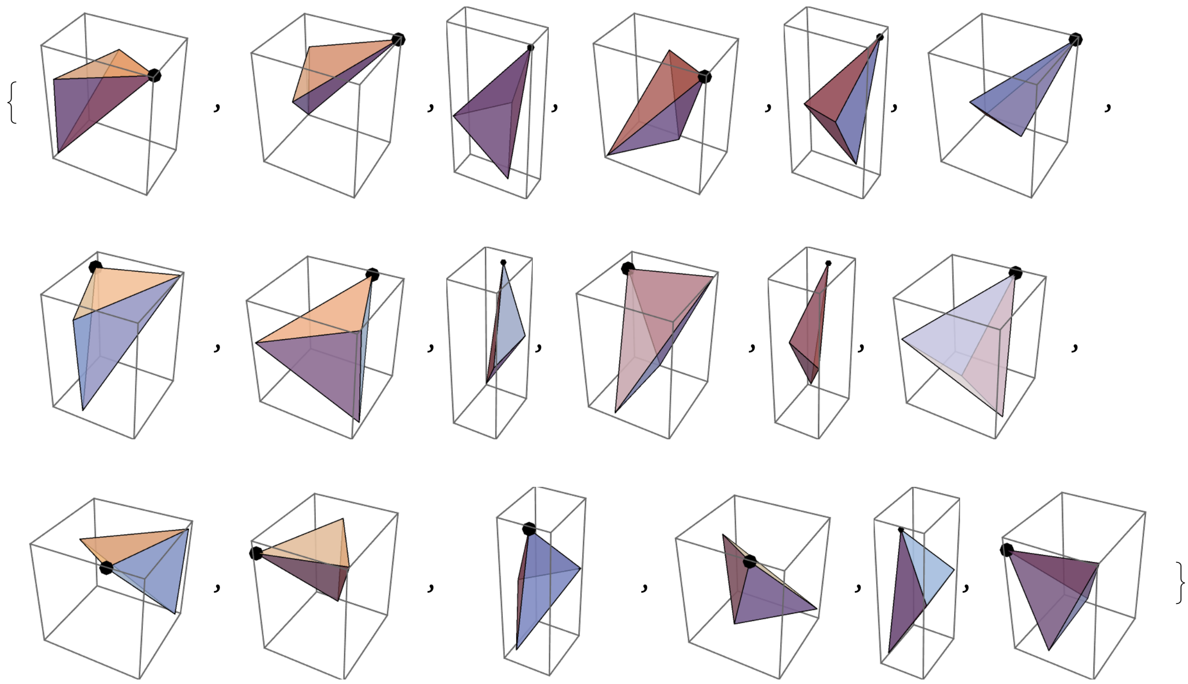
Out[84]=

3

$$\left\{k_{cv}\left(\frac{0.11384\,(0.506575-V_o)}{V_o^2}-\frac{0.0956144\,(0.588289-V_o)}{V_o^2}-\frac{0.0668358\,(0.624122-V_o)}{V_o^2}\right),\right.$$
$$k_{cv}\left(-\frac{0.000830056\,(0.506575-V_o)}{V_o^2}+\frac{0.11885\,(0.588289-V_o)}{V_o^2}-\frac{0.123506\,(0.624122-V_o)}{V_o^2}\right),$$
$$\left.k_{cv}\left(\frac{0.0482789\,(0.506575-V_o)}{V_o^2}+\frac{0.0784126\,(0.588289-V_o)}{V_o^2}+\frac{0.0646893\,(0.624122-V_o)}{V_o^2}\right)\right\}$$

In[85]:=
```
plt = Graphics3D[
    {{Opacity[0.7], #}, {Black, PointSize[0.075], Point@point}}, ImageSize → Tiny] & /@
  Flatten@MapThread[Function[x, Tetrahedron@Join[{#}, x]] /@ #2 &,
    {cellcentroids[pointind], Values@pyramids[[1, 1]]}]
```

Out[85]=



In[86]:=
```
Show[plt, Graphics3D[{Blue, PointSize[0.04], Point@cellcentroids[pointind]}],
 ImageSize → Small]
```

Out[86]=